|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 武汉大学国家网络安全学院教学实验报告 | | | | | |
| 课程名称 | 网络对抗演练 | | 实验日期 | | 2024/6/5 |
| 实验名称 | 漏洞挖掘 | | 实验周次 | | 第4周 |
| 姓名 | 学号 | | 专业 | | 班级 |
| **赵伯俣** | **2021302181156** | | **信安** | | **6** |
| **刘竞优** | **2021302181057** | | **信安** | | **6** |
| **张竣尧** | **2021302181004** | | **信安** | | **7** |
| 实验目的及要求  [一、漏洞挖掘环境搭建 3](#_Toc169210964)  [二、漏洞挖掘实战 3](#_Toc169210965)  [三、CVE-2021-1682 3](#_Toc169210966)  [四、固件漏洞挖掘环境搭建 3](#_Toc169210967)  [五、嵌入式交叉编译 3](#_Toc169210968)  [六、路由器漏洞实战 4](#_Toc169210969)  实验过程及分析  [一、漏洞挖掘环境搭建 4](#_Toc169210970)  [1、搭建windows 内核调试环境，下载虚拟机镜像并安装虚拟机 4](#_Toc169210971)  [2、宿主机上搭建windbg调试环境，并基于第一个实验中的windows虚拟机搭建内核调试环境 5](#_Toc169210972)  [3、安装Bindiff 8](#_Toc169210973)  [4、安装Ubuntu fuzz和编译环境 9](#_Toc169210974)  [5、在第四个搭建的环境中安装afl-fuzz工具 10](#_Toc169210975)  [6、编译upx fuzz目标程序 10](#_Toc169210976)  [7、安装VS 2022和Windows 10 SDK(10.0.19041.0) 14](#_Toc169210977)  [二、AFL模糊测试upx 15](#_Toc169210978)  [三、CVE-2021-1682 22](#_Toc169210979)  [四、固件漏洞挖掘环境搭建 30](#_Toc169210980)  [1、Binwalk分析 30](#_Toc169210981)  [2、Qemu模拟ARM 33](#_Toc169210982)  [五、嵌入式交叉编译 34](#_Toc169210983)  [1、完成交叉编译工具链生成 34](#_Toc169210984)  [2、完成静态编译BusyBox大小端 47](#_Toc169210985)  [3、LD\_PRELOAD劫持随机数生成 51](#_Toc169210986)  [六、路由器漏洞实战 56](#_Toc169210987)  [1、D-Link Dir-645溢出漏洞 56](#_Toc169210988)  [2、TOTOLink逻辑漏洞 73](#_Toc169210989)  实验感想与体会  [赵伯俣 78](#_Toc169210990)  [刘竞优 78](#_Toc169210991)  [张竣尧 78](#_Toc169210992) | | | | | |
| 实验目的及要求 一、漏洞挖掘环境搭建 1. 搭建windows 内核调试环境  2. 宿主机上搭建windbg调试环境，并基于第一个实验中的windows虚拟机搭建内核调试环境  3. 安装Bindiff  4. 安装Ubuntu Fuzz和编译环境  5. 在Ubuntu中安装afl-fuzz工具  6. 编译upx fuzz目标程序  7. 宿主机安装VisualStudio2022，并安装Win10 SDK开发环境 二、漏洞挖掘实战 使用AFL-FUZZ 挖掘最新版本UPX漏洞，学会安装使用该漏洞挖掘工具并进行漏洞挖掘，记录全流程操作和细节。 三、CVE-2021-1682 使用第一天搭建的windows 虚拟机镜像分析CVE-2021-1682,写出poc，漏洞的形成原因，poc执行中的约束条件及漏洞调试的报告。 固件漏洞挖掘环境搭建  1. 完成Binwalk工具安装，通过分析固件获取隐藏在固件中的密钥(截图固件中的登陆密钥)。 2. 完成静态编译的ARM架构的Hello World程序，使用qemu-arm-static、模拟运行该程序。(截图打印个人学号- Hello \*学号\*)  嵌入式交叉编译  1. 完成完成交叉编译工具链生成(生成MIPS大小端工具链) - 截图运行结果 2. 完成静态编译Busybox工具(MIPS大小端编译-LSB、MSB) 3. LD\_PRELOAD劫持随机数生成(改成自定义生成函数-格式学号+hook+ok) - 截图运行效果  路由器漏洞实战  1. 完成DIR-645溢出漏洞，通过栈溢出漏洞利用，获取shell - 截图运行结果 2. 完成TOTOLINK逻辑漏洞，通过逻辑漏洞，获取后台管理界面 - 截图后台界面 | | | | | |
|  | | | | | |
| 实验过程及分析 一、漏洞挖掘环境搭建1、搭建windows 内核调试环境，下载虚拟机镜像并安装虚拟机 因为hyper-v只有在专业版下才有，我们还是使用vmware进行安装     2、宿主机上搭建windbg调试环境，并基于第一个实验中的windows虚拟机搭建内核调试环境 内核调试环境参考[Windbg双击调试（真机WIN10+虚拟机WIN10）\_windbg10.怎么 调试-CSDN博客](https://blog.csdn.net/weixin_41489908/article/details/130045492)  windbg安装较为简单，这里不再说明，下面我们先设置前面我们安装的win10（注意要把打印机给删除，否则串行端口会被占用，后面连接不上）    之后重启进入win10，我们需要修改配置文件    或者直接来到msconfig下进行配置（最好提前备份）    之后回到宿主机上，给以管理员模式打开windbg并设置管道调试    之后重启win客户机（开机时记得重选配置） windbg已经连上，而虚拟机中的win则会卡住不动     3、安装Bindiff 我们还是到github上查看，因为这个google买下原公司后直接开源了，这里我们下载最新的版本，但是支持的ida是8.0-8.4的我们还需要再下载新的ida    安装好后，我们先编写两个实例程序    然后都分别使用ida打开后，并打包数据库    下来使用bindiff分别打开    得到如下界面   4、安装Ubuntu fuzz和编译环境 这里因为ubuntu18.04之前实验安装过了，我们省略过程   5、在第四个搭建的环境中安装afl-fuzz工具 首先来到官网，进行下载    拖到我们的虚拟机中进行解压，然后配置编译安装     6、编译upx fuzz目标程序 结果一样，我们直接下载压缩包然后解压缩就行    编写一个简单的程序    然后用alf-fuzz进行插桩处理，其实就是编译    用upx对程序进行加壳处理    然后建立相关文件    然后开始运行fuzz测试    放大我们的终端用以显示信息    测试一段时间后我们需要自己手动结束，因为语料库是非常大的。结束之后我们来到fuzz\_out/crashes中进行查看，结果如下图所示     7、安装VS 2022和Windows 10 SDK (10.0.19041.0) 过程非常简单，在“修改”中点击勾选我们需要安装的内容，之后更新即可       二、AFL模糊测试upx 创建fuzz-upx目录用来漏洞发现，从github上拉取upx最新版源码    修改环境变量的gcc和g++为afl-fuzz工具的编译器，使得编译时对upx程序进行插桩    关闭核心转储    下载项目需要的依赖包    创建build/release目录用于编译项目    cmake编译项目      make构建项目    创建in、out目录，分别存储fuzz的样本及fuzz出的crash还有覆盖率等信息，编写一个简单程序，gcc编译并strip掉符号作为初始样本，放入afl\_upx\_in目录      启动fuzz，等待出crash样本      使用htop查看查看系统资源的使用情况。    运行20分钟后，可以得到8个crash    先将得到的crash备份    查看id0的程序内容    使用upx对其中一个程序加壳，出现Segmentation fault错误  id:000000,sig:11,src:000126,op:havoc,rep:4    使用gdb调试upx程序，设置输入，同样是Segmentation fault      可以看到错误发生在 PackLinuxElf64::invert\_pt\_dynamic 函数中，具体位置在源代码的第8028行。问题出现在尝试访问字符串表 (strtab) 的终结位置，确切地说是这一行代码:  || '\0' != ((char \*)file\_image.getVoidPtr())[-1 + strtab\_max + strtab\_beg]  这行代码在检查字符串表的最后一个字符是否为 null 字符 (\0)，这是为了验证字符串表的格式正确性。如果字符串表的最后一个字符不是 null 字符，这通常表明有问题。  分析错误原因  （1）内存访问越界: 这行代码中的 [-1 + strtab\_max + strtab\_beg] 可能导致越界访问。如果 strtab\_max + strtab\_beg 超出了实际映射到内存中的文件大小，那么访问这个地址就会导致段错误。strtab\_beg 是字符串表的起始位置，strtab\_max 是字符串表的大小。  （2）指针计算错误: 这行代码使用 -1 索引来访问字符串，这有可能是因为数组索引错误或计算不正确导致的。需要检查 strtab\_max 和 strtab\_beg 的计算是否正确。    接着将其他七个crash作为upx的输入程序，可以看到出现错误的地方都相同，这几个crash都是相同类型的  id:000001,sig:11,src:000257,op:havoc,rep:16    id:000002,sig:11,src:000337,op:havoc,rep:64    id:000003,sig:11,src:000339,op:havoc,rep:8    id:000004,sig:11,src:000349,op:havoc,rep:16    id:000005,sig:11,src:000407,op:havoc,rep:32    id:000006,sig:11,src:000544,op:havoc,rep:16    id:000007,sig:11,src:000666,op:havoc,rep:8    id:000008,sig:11,src:000240,op:havoc,rep:4   三、CVE-2021-1682 下载746和685，下来之后是blob格式的文件，只能限定于idapro打开，而且需要注意ida与bindiff的版本兼容问题。  我们分别将下载的两个东西拖入到ida，直接ida打开是不识别的，因为下载下来是blob文件，需要后面拖到里面。我们先拖入一个来生成idb，之后打开另一个选择bindiff进行对比    之后筛选选择此次cve的主要漏洞函数    打开bindiff之后如下图所示    放大之后如下所示，然后我们根据这段代码来到ida里面进行查看    下图左边是没有修复的源伪代码图      可以看到就是多加了一个判断条件语句块，当a2+4小于248时进入跳转，从前面我们看看这个a2是该函数“EtwpNotify”的形参，在下面有个检查然后对a2+40处进行赋值，那么我们就能够猜出来a2应该是一个指针。对函数名进行交叉搜寻，能够得到“NtTraceControl“调用了该函数，我们追踪这个a2    我们来到这个函数下面看看    而这个NtTraceControl函数则必定是由TraceControlA/W调用的内核态函数，我们可以据此查找到相应的用户态函数，看看内容  [ControlTraceW 函数 (evntrace.h) - Win32 apps | Microsoft Learn](https://learn.microsoft.com/zh-cn/windows/win32/api/evntrace/nf-evntrace-controltracew)，该函数用于刷新、查询、更新或停止指定的事件跟踪会话。  参考[[原创] 剖析 InfinityHook 原理 掀起一场更激烈的攻与防恶战-编程技术-看雪-安全社区|安全招聘|kanxue.com](https://bbs.kanxue.com/thread-253450.htm)  这里是涉及到了windows的事件跟踪，简单理解就是用来写程序日志的    该函数原型如下：   1. EXTERN\_C 2. NTSYSCALLAPI 3. NTSTATUS 4. NTAPI 5. NtTraceControl( 6. \_In\_ **ULONG** FunctionCode, 7. \_In\_reads\_bytes\_opt\_(InBufferLen) **PVOID** InBuffer, 8. \_In\_ **ULONG** InBufferLen, 9. \_Out\_writes\_bytes\_opt\_(OutBufferLen) **PVOID** OutBuffer, 10. \_In\_ **ULONG** OutBufferLen, 11. \_Out\_ **PULONG** ReturnLength 12. );   第一个参数是常量，1-5依次代表启动跟踪、停止跟踪、查询跟踪、更新跟踪、刷新跟踪。    我们再来看看这几个case情况，当case 17时能够进入到我们上面的语句块中    我们定位到了v8参数，它是由a1赋值得到的，由于现在微软没有这个函数的相关信息，我们根据讲解内容，得知这里应该是一个“OutBuffer”形参    向上跟踪得知，这里就是用的rbx寄存器    此处我们得到一些约束条件“FunctionCode”=17；  “EtwpNotifyGuid”函数中对InBuffer+40位置需要有一个合法的GUID对象，表示存在的“ETW\_GUID\_ENTRY”对象；  此外InBuffer+76位置不能=1，我们发现这里并没有什么能够利用的地方，我们拐回去，对参数进行跟踪，发现有一个利用a2+4处的值进行利用的    猜测应该就是借此来分配数据块的，我们点击去看看都有什么操作。    进去看到主要的内容就是上面，参考[exAllocatePoolWithTag 函数 (wdm.h) - Windows drivers | Microsoft Learn](https://learn.microsoft.com/zh-cn/windows-hardware/drivers/ddi/wdm/nf-wdm-exallocatepoolwithtag)，主要适用于给例程分配指定类型的池内存，并返回已分配的指针。  这里我们就很明显可以发现了，再打补丁之前，并没有对这个分配大小进行检查，此时我们就可以给“a2+4”处放一个较大的内存。该函数返回指针a3，我们再次跟踪    我们来到这里，可以看到最终对内核态空间有一个访问读取，即越界访问读取。  前面我们知道该ETW系统需要获取GUID来进行，也就是“ETW\_GUID\_ENTRY”对象。根据[利用“简单”漏洞 – 只需 35 个简单步骤或更短的步骤！– Winsider Seminars & Solutions Inc.（Winsider Seminars & Solutions Inc.） --- Exploiting a “Simple” Vulnerability – In 35 Easy Steps or Less! – Winsider Seminars & Solutions Inc. (windows-internals.com)](https://windows-internals.com/exploiting-a-simple-vulnerability-in-35-easy-steps-or-less/#5-a-custom-guid)的骚操作，我们可以自己设置厂商，当然比较麻烦。我们也可以使用其他程序的GUID，因为有些是具有可修改权限的，借此可以达到利用目的。编写poc如下所示    最后拖到win10虚拟机中运行一下，得到蓝瓶    最后能够看到终止代码“PAGE\_FAULT IN NONPAGED AREA” 它通常与硬件故障、驱动程序问题或错误的内存访问有关。当计算机发生此类错误时，系统会自动停止运行以防止损坏数据。也就正好照应我们之前的越界访问 四、固件漏洞挖掘环境搭建1、Binwalk分析 首先查看该固件包的基本信息如下  （1）LZMA 压缩数据:  偏移量 0x0 - 这个段以 LZMA 压缩数据开始，其中 properties: 0x5d 指明了 LZMA 数据流的属性。字典大小为 33554432 字节，未压缩大小为 4840000 字节。  描述 - 从描述来看，这可能是固件的主要部分，常见于引导或主操作系统的映像。  （2）SquashFS 文件系统:  偏移量 0x1C000 - 这个段是一个 SquashFS 文件系统，该文件系统是为嵌入式系统设计的高度压缩的只读文件系统。  字典大小 131072 字节和其他参数如 block size: 131072 bytes, 2306 inodes 表示了文件系统的结构和大小。这通常用于存储应用程序和配置文件。  （3）时间戳:  文件系统的创建时间为 2015-09-17 16:56:59，这可能是固件最后一次更新或编译的时间。  基于以上信息，这个固件包含一个引导段（可能是操作系统或固件的一部分），后跟一个包含各种系统文件和应用程序的 SquashFS 文件系统。    接下来检测 ARM 架构  三个 ARM 指令集的函数序幕（function prologue），这通常是函数的开始部分。  （1）偏移量 0xB6AAE:  指示在该偏移处发现了 ARM 指令，具体是函数的开头（函数序幕）。这表明该部分可能是一个独立的函数或一段执行代码的起始。  （2）偏移量 0x1F6E24:  同样，在这个地址也检测到了函数序幕，表明另一个可能的代码段或函数的起始。  （3）偏移量 0x75DEF:  这是第三处函数序幕的位置，进一步确认了固件中包含多段可执行代码。  从这些信息可以看出，这个固件文件包含 ARM 处理器架构的可执行代码。这些代码可能是固件的主要逻辑部分，用于控制硬件或执行特定的操作任务。    接下来根据magic签名的扫描结果对固件进行递归提取时出现报错，推测可能是windows系统的原因    在kali虚拟机上重新执行该指令    可以成功获取到固件的根目录\_firmware.bin.extracted  进入到squashfs-root目录查看文件目录树结构如下所示  在目录结构中查找login字段，找到logininfo.xml文件    将找到的文件打开后成功获取登陆密钥   2、Qemu模拟ARM 编写hello程序    安装交叉编译工具链    使用交叉编译工具链进行静态编译    安装适用于ARM架构的QEMU模拟器    使用QEMU的ARM模拟器运行hello程序，成功输出小组三人的学号，并且可以看到只能在arm架构上运行，在x86上无法运行。   五、嵌入式交叉编译1、完成交叉编译工具链生成 小端：首先在终端中执行指令make menuconfig打开操作界面    选择Target options    选择更改目标架构    接下来选择MIPS32的小端架构    之后退出到根目录界面中选择工具链    之后选择C语言链接库    将C语言链接库修改为uClibc-ng    然后选择内核    将内核版本选择为Linux 4.19    然后选择gcc编译器版本    将编译器版本设置为gcc 11.x    然后回退到主界面选择Build options    进入libraries选项    选择支持静态和动态    然后回退到主界面进入到系统配置中    选择Init system选项    将系统设置为None    进入到Target packages中去掉BusyBox    全部设置完成之后返回到主界面，将配置进行保存    之后运行make BR\_JLEVEL=4 V=s进行编译    编译完成后进入到output/host/bin/，查看生成的工具如下所示    接下来新建一个项目目录，在目录中新建test.c文件如下    使用编译生成的gcc工具mipsel-linux-gcc对test.c文件进行编译    接下来查看编译生成的可执行文件的基本信息可以发现该可执行程序为mips32位小端（LSB），链接方式为静态链接    对可执行文件使用mipsel-linux-strip工具strip掉ELF可执行文件或者目标文件中不必要的调试信息，通过对比发现文件的大小减少了将近一倍    运行该可执行程序如下    大端：将之前的小段文件项目目录更改名称之后重新解压项目包    将小端操作时的dl目录拷贝到当前项目包中    之后的配置部分与小端相同，只是在选择架构时选择MIPS32大端，将目标架构更改为MIPS32大端    之后将配置保存    重新执行make BR\_JLEVEL=8 V=s进行编译    查看output/host/bin目录下的工具可以发现大端工具mips-linux-gcc    对test.c进行编译，生成可执行文件test\_big    查看该文件的基本信息如下    同样对可执行文件使用工具strip掉ELF可执行文件或者目标文件中不必要的调试信息，通过对比发现文件的大小减少了将近一倍    运行该可执行程序如下   2、完成静态编译BusyBox大小端 完成静态编译Busybox工具(MIPS大小端编译-LSB、MSB) - 截图运行结果  按照以下指令进行编译  curl http://busybox.net/downloads/busybox-1.36.1.tar.bz2 | tar xjf -  mkdir -p obj/busybox  cd busybox-1.36.1  make O=../obj/busybox defconfig #独立在新文件中进行相关配置  cd ../obj/busybox  make menuconfig  下载源码文件，这里下载太慢直接在网站上下载    解压文件    开始配置      修改配置，使用静态编译  -> Settings  -> Build Options  [\*] Build BusyBox as a static binary (no shared libs)    设置交叉编译前缀  -> Settings  -> Build Options  Cross compoler prefix    保存并退出配置  编译  make CROSS\_COMPILE=/home/liu1057/week4/buildroot/output/host/bin/mipsel-linux-    查看编译的结果程序    在qemu上运行该程序    要编译为大端程序，只需要修改Cross compoler prefix为mips大端的编译器，其他配置相同      编译  make CROSS\_COMPILE=/home/liu1057/week4/buildroot/output/host/bin/mipsel-linux-    查看编译的结果程序    在qemu上运行该程序   3、LD\_PRELOAD劫持随机数生成 我们除了buildroot之外也可以直接通过apt安装交叉编译工具链，需要的只有两条命令（debian下有相关说明[Manpages of binutils-mips-linux-gnu in Debian stretch — Debian Manpages](https://manpages.debian.org/stretch/binutils-mips-linux-gnu/index.html)）  “sudo apt-get -y install gcc-mips-linux-gnu”  “sudo apt-get -y binutils-mips-linux-gnu”  这里下载的默认都是大端的，小端的只需要改为mipsel即可[Debian -- 在 sid 中的 binutils-mipsel-linux-gnu 软件包详细信息](https://packages.debian.org/zh-cn/sid/binutils-mipsel-linux-gnu)    Mips交叉编译工具链大端编译busybox，首先进入到busybox对应目录下进行配置，有三种模式，我们只用缺省配置      首先需要我们把上面提到的工具都换一个地方，因为下载之后默认是和gcc一样在bin文件夹下，我们还是将其mips有关的单独拿出来    然后我们就需要设置一下我们的工具链配置（仅在当前终端下有效）    在些许warning之后，我们使用file命令查看，编译成功，得到mips大端下的busybox文件    LD\_PRELOAD就是linux下的环境变量，允许用户定义在程序运行之前优先加载的动态链接库，说白了就是在调用标准的printf函数之前，会首先调用我们编写的so库中的printf函数，也就是实现hook劫持  我们先编写一个随机数生成打印函数    编译运行后如下所示    接下来我们自己编写一个随机数生成函数，让其返回固定内容      然后指定静态编译，并配置环境变量    之后再次运行如下，实现劫持    其中我们可以ldd查看文件加载的动态库的顺序    很明显我们自己编写的so库已经成为了首先加载的对象，那么我们的rand函数自然就完成了替换。当然我们这里只是临时替换一下环境变量，如果重新打开终端再执行的话，就会恢复原状    然后我们来到mips下进行一番，还是前面的rand.c代码    接下来我们配置so库   六、路由器漏洞实战1、D-Link Dir-645溢出漏洞 首先在kali虚拟机中使用binwalk提取得到固件根目录    发现并没有提取到目标文件系统    观察binwalk运行的信息可以发现是因为缺少sasquatch库导致，所以先下载sasquatch库    重新进行提取可以发现成功获取到目标文件系统    查看位于htdocs目录下的cgibin文件的基本信息，可以发现该文件是32位小端序可执行文件，指令架构为MIPS    将该可执行文件放入到IDA中进行分析    在汇编代码中查找可以发现service.cgi字段    在函数名称列表中可以找到servicecgi\_main函数如下    查看该函数的交叉引用情况如下可以发现servicecgi\_main函数会通过lxmldbc\_system调用到system函数    查看servicecgi\_main函数的汇编代码如下，可以看到程序会调用REQUEST METHOD请求方法来处理get和post请求，但是会同时进入到loc\_40CE6C函数中。    查看loc\_40CE6C函数的内容如下，可以发现两个请求运行路径基本相同    因为两个处理函数均会调用cgibin\_parse\_request函数，在IDA中查看该函数的内容，可以发现该函数会对CONTENT\_TYPE和CONTENT\_LENGTH进行解析    在该函数中对参数进行解析完毕之后就会回到servicecgi\_main函数中，可以发现EVENT和SERVICE都会调用lxmldbc\_system函数    查看lxmldbc\_system函数可以发现该函数会调用system函数，同时该函数在调用system执行command时并没有进行命令过滤，我们可以在这里引发命令注入漏洞。    综上所述，我们可以在访问的时候加上EVENT或者利用已经加入ACTION字段的SERVICE，在阶段之后加上恶意指令即可，注入命令之后因为&在http请求中为请求域分隔符，所以需要将&转码为urlencode编码的%26。  因为在本机中下载了firmware-analysis-toolkit的所有版本以及firmware-analysis-plus均不能运行该硬件，所以接下来采用firmware-analysis-plus的作者在github上发出的ubuntu20.04的镜像进行接下来的步骤  在下载好的镜像虚拟机中执行如下指令启动软件  python3 fap.py -q /home/jerry/firmware-analysis-plus/qemu-builds/2.5.0/ DIR645A1\_FW103B11.bin    运行一段时间后发现工具仍然会在一个位置死循环    于是尝试使用affityOS,将固件放置到/home/oit/tools/firmadyne目录下    运行fat.py程序    在文件名称选项中输入我们的固件路径/home/oit/tools/firmadyne/DIR645A1\_FW103B11.bin  固件品牌名随意输入a  接下来在firmadyne的密码处输入firmadyne，在oit的密码处输入attify123    最终生成目标ip地址    查看该虚拟机网卡    打开burp    修改火狐浏览器代理    之后访问终端给出的ip：192.168.0.1成功实现固件仿真    随即不用输入密码直接点击login，登陆成功之后成功劫持到cookie    接下来使用火狐浏览器访问192.168.0.1/service.cgi?EVENT=test,抓包结果如下    将抓取的流量包发送给重放器    修改EVENT的内容为test | pwd%26可以发现response返回了pwd命令的执行结果，这里我们使用的截断方式为|    接下来尝试使用 ; 进行截断构造流量包如下所示，执行ls命令后成功返回命令执行结果    接下来尝试使用 %26 进行截断，发现依然能够成功执行ls命令    在Decoder中查看换行符的URL编码为%0a    使用%0a进行截断也能够成功执行ls命令    由此我们成功实现利用EVENT获取目标shell  因为在分析过程中我们知道可以使用EVENT也可以使用SERVICE所以我们继续尝试service，访问http://192.168.0.1/service.cgi?ACTION=START&SERVICE=test之后济进行抓包    将抓取的流量包发送给重放器    修改流量包内容，使用|截断来执行ls命令，成功返回根目录文件    因为在之前IDA的分析中我们可以知道GET与POST两种请求命令所执行的函数相同，所以在这里尝试使用POST请求触发漏洞  在bp中将GET请求更改为POST    然后与之前的步骤一样将流量包发送给重放器    之后同样使用|截断执行ls命令，同样能够执行成功    综上所述，成功实现获取目标路由器固件的shell 2、TOTOLink逻辑漏洞 首先安装FirmAE，git克隆一下就行，下面是目录结构    下载下来之后我们还需要把固件都给放置到FirmAE目录下，首先进行“.download.sh”用于从github拔取一些依赖，因为全是github上的仓库，所以速度有些感人。之后运行“./install.sh”下载一些其它的别的东西    中间可能会出现如下情况，只要一路回车运行即可    安装成功之后会发现电脑中多了chrome和openjdk，属于正常情况。此时我们会发现多出了一个binwalk的文件夹，我们需要安装一下，如果直接init然后run的话会发生如下错误（有这个错误需要多安装几次）      之后运行“./init.sh”来初始化pg数据库    最后来到根目录下，模拟运行totolink    Start后面还需要等待非常长一段时间，看待如下IP，证明成功    然后我们下载burpsuite，并用其内置的浏览器登录这个网址，这样一来我们就不用再安装拉取证书了，我们来到登陆页面随输入一下，抓包结果如下所示    放包之后我们有收到一个这个    好的，密码错误。    但是我们注意到了其中的authCode，我们再用正确账密输入试试    中间的包报过来就是如下结果，显示要我们要登录如下网址，并且还带了一个authCode=1    成功登录后页面如下    所以我们能够理解一下这个登录逻辑，应该是在login.asp中请求认证，认证成功之后拿到authCode作为身份认证，之后拿着这个登录到发送给真正的后台进行验证，我们直接在浏览器中输入    然后成功登录 | | | | | |
|  | | | | | |
| 实验感想与体会 赵伯俣 在本周的实验中，我负责binwalk固件分析，交叉编译工具链生成和D-Link DIR-645几个部分的任务，本周的实验在试验任务上的难度相比于前几周的任务并不算大，但是在配置实验环境时需要消耗掉不少的时间，在机型binwalk固件分析时由于事先并不知道在Kali虚拟机中自带有binwalk工具，所以在Windows和Ubuntu系统中都配置了相应的环境，在完成交叉编译工具链生成任务时小端工具链部分的复现比较顺利，但是在大端工具链部分出现了很多问题，最后在老师后面的演示指导下成功编译出了大端工具链，在编译途中出现了虚拟机死机与编译时间过长等问题，最后借助网络资料克服。  在完成D-Link DIR-645溢出漏洞分析时，大部分的时间都在配置固件仿真环境，我们尝试了在Ubuntu22.04环境下安装firmware-analysis-toolkit工具，并且尝试了该工具的v0.2，v0.3以及plus等各个版本，但是都会出现死循环或者程序报错问题，在Kali环境下使用该工具也会出现相同报错，并且在虚拟机的共享文件夹中执行该工具会出现网络连接失败的错误。尝试使用firmware-analysis-plus作者提供的Ubuntu镜像文件也会出现报错。最后在查阅资料时发现可以使用AffityOS虚拟机成功实现硬件仿真。最后的漏洞部分比较简单，成功获取到了目标路由器的shell。 刘竞优张竣尧 在此次实验中我完成了CVE的分析与其他环境搭建等任务。这周的任务量相对来说并不大，因为大部分时间都在搭建环境，涉及到的范围较广，从fuzz到二进制分析。但最难的还是win内核漏洞的分析，因为对windows编程和内核知识的极度缺乏，导致后续分析时十分困难，而且因为时间原因，并不能够完整系统的学习底层有关数据结构，最多只是囫囵吞枣将结果展示出来，如果想要真正理解，恐怕还是任重而道远。这次最大的收获就是windbg的内核调试模式与bindiff的使用，非常震撼，想必会为以后的进一步学习打下一定基础。  接着上一周的实验，这次的实验可算是真正的碰到了IOT设备上，此前从来没有接触过该领域，属于全新的体验。通过下载设备上相应的二进制文件，然后进行模拟，借助binwalk等工具分离出文件系统之后进行分析，有点类似于电子取证的过程，此外因为这次涉及到的都是mips架构上的内容，所以需要学习一下。总的来说，较好的丰富了我的安全见识，也让我进一步了解到了IOT设备漏洞的有关内容，并可能会使我在以后的工作中重视这方面，而非单纯的前后端安全。 | | | | | |
|  | | | | | |
| 1. 教师评语 | | | | | |
|  | | | | | |
| **教师评分（请填写好姓名、学号）** | | | | | |
| 姓名 | | 学号 | | 分数 | |
| 赵伯俣 | | 2021302181156 | |  | |
| 刘竞优 | | 2021302181057 | |  | |
| 张竣尧 | | 2021302181004 | |  | |
| 教师签名：  2024年 6月 13日 | | | | | |