CTF 从入门到放弃

Firmianay | Phantom0301 | Skyel1u

https://github.com/firmianay/CTF-All-In-One

2018–01–29

声明

目录

缩略短语表

1 前言

“与其相信谣言，不如一起学习”。

1.1 常见问题

- 什么是 CTF，我为什么要学?

  CTF 是网络安全技术人员之间进行技术竞技的一种比赛形式，通过学习，可以在法律允许的范围内，快速地了解和掌握相关安全技术。

- 阅读本书的预备知识是什么?

  本书是为初学者准备的，不要求有预备知识，但如果对 Linux 操作系统，对编程有一定了解，肯定会有帮助。

- 我可以买到纸质版吗?

  抱歉，目前没有。除非对纸质书有偏好，你可以自行打印。否则由于内容整体尚未完成，且更新很快，作者更推荐使用电子版。

- 本书有 PDF/epub/mobi 格式的吗?

  目前没有 epub/mobi 版本。暂时有 pdf，可在 GitBook 页面下载，这群人正在努力学习 LaTeX 的使用，以提供更优雅的排版和 PDF 文件。

- 我能打印本书或者作为教材教课吗?

  太棒了！必须的！本书使用 Creative Commons license (CC BY-SA 4.0)，大可随意使用。作为一个开源项目，我们当然希望更多的人了解并参与进来。

- 本书为何免费，有何目的?

  技术类书籍大多赚不到钱，只是作者的兴趣使然，顺便给自己打个广告。让更多的人了解并推广，才是我们的目的所在，也是开源精神之所在。

- 我还有其他问题。

  你也可以选择提交 issue 到本仓库，也可以邮件联系我们。

1.2 合作与贡献

随着信息安全的迅速发展，CTF 竞赛也在如火如荼的开展，有人说“今天的 ACM 就是明天的 CTF”，颇有几分道理。
目前市场上已经充斥着大量的 ACM 书籍，而 CTF 以其知识内容之分散、考察面之广泛、题目类型之多变，让许多新手不知所措，同时也加大了该方面书籍的编写难度。

此书本着开源之精神，以分享他人提高自己为目的，将是一本大而全的 CTF 领域指南。因本人能力和时间有限，不可能精通各个类别的知识，欢迎任何人提出任何建议，和我一起完成此书。千万不要觉得自己是初学者就不敢提交 PR（issue），千万不要担心自己提交的 PR（issue）会有问题，毕竟最后合并的人是我，背锅的也是我:）

如果你还有关于本书的其他想法，请直接给我发邮件 firmianay@gmail.com。

1.3 致谢

感谢内容贡献者：skyel1u、phantom0301
感谢 XDSEC，把我引上了安全这条路，认识了很多志同道合的小伙伴。
感谢 GitHub 上的朋友，是你们的 star 给我写作的动力。

2 CTF 基础知识

2.1 CTF 简介

2.1.1 概述

CTF（Capture The Flag）中文一般译作夺旗赛，在网络安全领域中指的是网络安全技术人员之间进行技术竞技的一种比赛形式。CTF 起源于 1996 年 DEFCON 全球黑客大会，以代替之前黑客们通过互相发起真实攻击进行技术比拼的方式。发展至今，已经成为全球范围网络安全圈流行的竞赛形式，2013 年全球举办了超过五十场国际性 CTF 赛事。而 DEFCON 作为 CTF 赛制的发源地，DEFCON CTF 也成为了目前全球最高技术水平和影响力的 CTF 竞赛，类似于 CTF 赛场中的“世界杯”。

CTF 为团队赛，通常以三人为限，要想在比赛中取得胜利，就要求团队中每个人在各种类别的题目中至少精通一类，三人优势互补，取得团队的胜利。同时，准备和参与 CTF 比赛是一种有效将计算机科学的离散面、聚焦于计算机安全领域的方法。

2.1.2 赛事介绍

CTF 是一种流行的信息安全竞赛形式，其英文名可直译为“夺得 Flag”，也可意译为“夺旗赛”。其大致流程是，参赛团队之间通过进行攻防对抗、程序分析等形式，率先从主办方给出的比赛环境中得到一串具有一定格式的字符串或其他内容，并将其提交给主办方，从而夺得分数。为了方便称呼，我们把这样的内容称之为“Flag”。

CTF 竞赛模式具体分为以下三类：

- **解题模式（Jeopardy）** 在解题模式 CTF 赛制中，参赛队伍可以通过互联网或者现场网络参与，这种模式的 CTF 竞赛与 ACM 编程竞赛、信息学奥赛比较类似，以解决网络安全技术挑战题目的分值和时间来排名，通常用于在线选拔赛。题目主要包含逆向、漏洞挖掘与利用、Web 渗透、密码、取证、隐写、安全编程等类别。

- **攻防模式（Attack-Defense）** 在攻防模式 CTF 赛制中，参赛队伍在网络空间互相进行攻击和防守，挖掘网络服务漏洞并攻击对手服务来得分，修补自身服务漏洞进行防御来避免丢分。攻防模式 CTF 赛制可以实时通过得分反映出比赛情况，最终也以得分直接分出胜负，是一种竞争激烈，具有很强观赏性和高度透明性的网络安全赛制。在这种赛制中，不仅仅是比参赛队员的智力和技术，也比体力（因为比赛一般都会持续 48 小时及以上），同时也比团队之间的分工配合与合作。
- **混合模式（Mix）** 结合了解题模式与攻防模式的 CTF 赛制，比如参赛队伍通过解题可以获取一些初始分数，然后通过攻防对抗进行得分增减的零和游戏，最终以得分高低分出胜负。采用混合模式 CTF 赛制的典型代表如 iCTF 国际 CTF 竞赛。

2.1.3 题目类别

- Reverse
  - 题目涉及到软件逆向、破解技术等，要求有较强的反汇编、反编译功底。主要考查参赛选手的逆向分析能力。
  - 所需知识：汇编语言、加密与解密、常见反汇编工具
- Pwn
  - Pwn 在黑客俚语中代表着攻破，获取权限，在 CTF 比赛中它代表着溢出类的题目，其中常见类型溢出漏洞有整数溢出、栈溢出、堆溢出等。主要考查参赛选手对漏洞的利用能力。
  - 所需知识：C，OD+IDA，数据结构，操作系统
- Web
  - Web 是 CTF 的主要题型，题目涉及到许多常见的 Web 漏洞，如 XSS、文件包含、代码执行、上传漏洞、SQL 注入等。也有一些简单的关于网络基础知识的考察，如返回包、TCP/IP、数据包内容和构造。有些题目环境比

较接近真实环境，因此通过各种高质量 Web 题目可以迅速提高选手的安全技能。

  - 所需知识：基础的 web 开发能力，一门脚本语言（PHP、Python）、JavaScript、数据库、网络协议等

- Crypto
  - 题目考察各种加解密技术，包括古典加密技术、现代加密技术甚至出题者自创加密技术，以及一些常见编码解码，主要考查参赛选手密码学相关知识点。通常也会和其他题目相结合。
  - 所需知识：数论、现代密码学和一门编程语言（C 或 Python）

- Misc
  - Misc 即安全杂项，题目涉及隐写术、流量分析、电子取证、人肉搜索、数据分析、大数据统计等，覆盖面比较广，主要考查参赛选手的各种基础综合知识。
  - 所需知识：常见隐写术工具、Wireshark 等流量审查工具、编码知识；此类题目知识繁杂，有时候更需要一个灵活的思路

- Mobile(Android)
  - 主要分为 Android 和 iOS 两个平台，以 Android 逆向为主，破解 APK 并提交正确答案。
  - 所需知识：Java，Android 开发，常见工具

2.1.4 高质量的 CTF 比赛

详见：

- 国外比赛平台：http://www.ctftime.org
- 国内 CTF 评分网站：https://www.ctfrank.org

2.1.5 竞赛小贴士

- 寻找团队
  - 彼此激励 24 小时以上的连续作战
  - 彼此分享交流技术与心得是最快的成长途径
  - 强有力的团队可以让你安心专注于某一领域
  - 在探索技术的道路上不会孤独
- 有效训练
  - 坚持不懈地训练是成为强者的必经途径
    - wargame
    - 经典赛题配合 writeup 加以总结
    - https://github.com/ctfs 以赛代练, 总结与分享
- wargame 推荐
  - 漏洞挖掘与利用 (pwn)
    - pwnable.kr
    - https://exploit-exercises.com
    - https://io.netgarage.org
  - 逆向工程与软件破解 (re)
    - reversing.kr
    - http://crackmes.de
  - web 渗透 (web)
    - webhacking.kr
    - https://xss-game.appspot.com
  - 综合类
    - http://overthewire.org/wargames
    - https://w3challs.com

* https://chall.stypr.com/?chall
* https://pentesterlab.com

2.1.6 线下赛 AWD 模式概述

Attack With Defence，简而言之就是你既是一个 hacker，又是一个 manager。

比赛形式：一般就是一个 ssh 对应一个服务，可能是 web 也可能是 pwn，然后 flag 五分钟一轮，各队一般都有自己的初始分数，flag 被拿会被拿走 flag 的队伍均分，主办方会对每个队伍的服务进行 check，check 不过就扣分，扣除的分值由服务 check 正常的队伍均分。

怎样拿到 Flag

web 主要是向目标服务器发送 http 请求，返回 flag，bin 主要是通过 exploit 脚本读取 `/home/username` 下某个文件夹下的 `flag` 文件。

Web 题目类型

- 出题人自己写的 CMS 或者魔改后的 CMS(注意最新漏洞、1day 漏洞等)
- 常见 (比如 Wordpress、Discuz!) 或者不常见 CMS 等
- 框架型漏洞 (CI 等)
- 如何在 CTF 中当搅屎棍
- AWD 模式生存技巧
- 能力：
  - 漏洞反应能力
  - 快速编写脚本
  - web 代码审计
  - 心态放好，因为 web 比较容易抓取流量，所以即使我们被打，我们也可以及时通过分析流量去查看别的队伍的 payload，从而进行反打。

- 脚本准备：一句话，文件包含，不死马、禁止文件上传等
- **警惕 web 弱口令，用最快的速度去补**

Bin 题目类型

大部分是 PWN，题目类型包括栈、堆、格式化字符串等等。
能力：

- 迅速找到二进制文件的漏洞，迅速打 patch 的能力
- 全场打 pwn 的 exp 脚本编写
- 熟悉服务器运维，如果二进制分析遇到障碍难以进行，那就去帮帮 web 选手运维
- 尽快摸清楚比赛的 check 机制
- 看看现场环境是否可以提权，这样可以方便我们搞操作（如魔改 libc 等等）

技巧

- 如果自己拿到 FB，先用 NPC 服务器或者自己服务器测试，格外小心自己的 payload 不要被别的队伍抓取到, 写打全场的 exp 时，一定要加入混淆流量。
- 提前准备好 PHP 一句话木马等等脚本。
- 小心其他队伍恶意攻击使我们队伍机器的服务不能正常运行，因此一定要备份服务器的配置。
- 尽可能在不搞崩服务和绕过 check 的情况下，上 WAF，注意分析别人打过来的流量，如果没有混淆，可以大大加快我们的漏洞分析速度。
- 工具准备: 中国菜刀、Nmap、Xshell、合适的扫描器等。
- 如果自己丢分，心态不要崩。
- 不要忽视 Github 等平台，可能会有写好的 exp 可以用。
- 将 flag 的提交自动化。

2.1.7 搭建 CTF 平台

已经有现成开源平台供我们使用：

- FBCTF - The Facebook CTF is a platform to host Jeopardy and “King of the Hill” style Capture the Flag competitions.
- CTFd - CTFd is a Capture The Flag in a can. It’s easy to customize with plugins and themes and has everything you need to run a jeopardy style CTF.
- SecGen - SecGen creates vulnerable virtual machines so students can learn security penetration testing techniques.

2.1.8 参考

https://baike.baidu.com/item/ctf/9548546

2.2 学习方法

提问的智慧

2.3 Linux 基础

2.3.1 常用 Linux 命令

常用基础命令

```
ls                  # 用来显示目标列表
cd [path]           # 用来切换工作目录
pwd                 # 以绝对路径的方式显示用户当前工作目录
man [command]       # 查看Linux中的指令帮助等信息
echo [string]       # 打印一行文本，参数“-e”可激活转义字符
cat [file]          # 连接文件并打印到标准输出设备上
less [file]         # 允许用户向前或向后浏览文字档案的内容
mv [file1] [file2]  # 对文件或目录重新命名，或者将文件从一个目录移到
    ↪ 另一个目录
```

```
cp [file1] [file2]   # 复制文件到指定的目的文件或目录
rm [file]            # 可以删除一个目录中的一个或多个文件或目录
ps                   # 用于报告当前系统的进程状态
top                  # 实时查看系统的整体运行情况
kill                 # 杀死一个进程
ifconfig             # 查看或设置网络设备
ping                 # 查看网络上的主机是否工作
netstat              # 显示网络连接、路由表和网络接口信息
nc(netcat)           # 建立 TCP 和 UDP 连接并监听
su                   # 切换当前用户身份到其他用户身份
touch [file]         # 创建新的空文件
mkdir [dir]          # 创建目录
chmod                # 变更文件或目录的权限
chown                # 变更某个文件或目录的所有者和所属组
nano / vim / emacs   # 字符终端的文本编辑器
exit                 # 退出 shell
|                    # 将一个命令的标准输出作为另一个命令的标准输入
```

Listing 2.1: 基础命令

Bash 快捷键

```
Up(Down)           # 上（下）一条指令
Ctrl + c           # 终止当前进程
Ctrl + z           # 挂起当前进程，使用“fg”可唤醒
Ctrl + d           # 删除光标处的字符
Ctrl + l           # 清屏
Ctrl + a           # 移动到命令行首
Ctrl + e           # 移动到命令行尾
Ctrl + b           # 按单词后移（向左）
Ctrl + f           # 按单词前移（向右）
Ctrl + Shift + c   # 复制
Ctrl + Shift + v   # 粘贴
```

Listing 2.2: Bash 快捷键

更多细节请查看：bash keyboard shortcuts

根目录结构

```
$ uname -a
Linux manjaro 4.11.5-1-ARCH #1 SMP PREEMPT Wed Jun 14 16:19:27 CEST
    ↪ 2017 x86_64 GNU/Linux

$ ls -al /
drwxr-xr-x  17 root root  4096 Jun 28 20:17 .
drwxr-xr-x  17 root root  4096 Jun 28 20:17 ..
lrwxrwxrwx   1 root root     7 Jun 21 22:44 bin -> usr/bin
drwxr-xr-x   4 root root  4096 Aug 10 22:50 boot
drwxr-xr-x  20 root root  3140 Aug 11 11:43 dev
drwxr-xr-x 101 root root  4096 Aug 14 13:54 etc
drwxr-xr-x   3 root root  4096 Apr  8 19:59 home
lrwxrwxrwx   1 root root     7 Jun 21 22:44 lib -> usr/lib
lrwxrwxrwx   1 root root     7 Jun 21 22:44 lib64 -> usr/lib
drwx------   2 root root 16384 Apr  8 19:55 lost+found
drwxr-xr-x   2 root root  4096 Oct  1  2015 mnt
drwxr-xr-x  15 root root  4096 Jul 15 20:10 opt
dr-xr-xr-x 267 root root     0 Aug  3 09:41 proc
drwxr-x---   9 root root  4096 Jul 22 22:59 root
drwxr-xr-x  26 root root   660 Aug 14 21:08 run
lrwxrwxrwx   1 root root     7 Jun 21 22:44 sbin -> usr/bin
drwxr-xr-x   4 root root  4096 May 28 22:07 srv
dr-xr-xr-x  13 root root     0 Aug  3 09:41 sys
drwxrwxrwt  36 root root  1060 Aug 14 21:27 tmp
drwxr-xr-x  11 root root  4096 Aug 14 13:54 usr
drwxr-xr-x  12 root root  4096 Jun 28 20:17 var
```

Listing 2.3: 目录结构

由于不同的发行版会有略微的不同，我们这里使用的是基于 Arch 的发行版 Manjaro，以上就是根目录下的内容，我们介绍几个重要的目录：

- `/bin`、`/sbin`：链接到 `/usr/bin`，存放 Linux 一些核心的二进制文件，其包含的命令可在 shell 上运行。
- `/boot`：操作系统启动时要用到的程序。
- `/dev`：包含了所有 Linux 系统中使用的外部设备。需要注意的是这里并不是存放外部设备的驱动程序，而是一个访问这些设备的端口。

- `/etc`：存放系统管理时要用到的各种配置文件和子目录。
- `/etc/rc.d`：存放 Linux 启动和关闭时要用到的脚本。
- `/home`：普通用户的主目录。
- `/lib`、`/lib64`：链接到 `/usr/lib`，存放系统及软件需要的动态链接共享库。
- `/mnt`：这个目录让用户可以临时挂载其他的文件系统。
- `/proc`：虚拟的目录，是系统内存的映射。可直接访问这个目录来获取系统信息。
- `/root`：系统管理员的主目录。
- `/srv`：存放一些服务启动之后需要提取的数据。
- `/sys`：该目录下安装了一个文件系统 sysfs。该文件系统是内核设备树的一个直观反映。当一个内核对象被创建时，对应的文件和目录也在内核对象子系统中被创建。
- `/tmp`：公用的临时文件存放目录。
- `/usr`：应用程序和文件几乎都在这个目录下。
- `/usr/src`：内核源代码的存放目录。
- `/var`：存放了很多服务的日志信息。

进程管理

- `top` 可以实时动态地查看系统的整体运行情况。
- `ps` 用于报告当前系统的进程状态。可以搭配 `kill` 指令随时中断、删除不必要的程序。
- 查看某进程的状态：`$ ps -aux | grep [file]`，其中返回内容最左边的数字为进程号（PID）。
- `kill` 用来删除执行中的程序或工作。
- 删除进程某 PID 指定的进程：`$ kill [PID]`

UID 和 GID

Linux 是一个支持多用户的操作系统，每个用户都有 User ID(UID) 和 Group ID(GID)，UID 是对一个用户的单一身份标识，而 GID 则对应多个 UID。知道某个用户的 UID 和 GID 是非常有用的，一些程序可能就需要 UID/GID 来运行。可以使用 id 命令来查看：

```
$ id root
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm)
    ↪ ,6(disk),10(wheel),19(log)

$ id firmy
uid=1000(firmy) gid=1000(firmy) groups=1000(firmy),3(sys),7(lp),10(
    ↪ wheel),90(network),91(video),93(optical),95(storage),96(scanner
    ↪ ),98(power),56(bumblebee)
```

Listing 2.4: 查看 uid

UID 为 0 的root 用户类似于系统管理员，它具有系统的完全访问权。我自己新建的用户 firmy，其 UID 为 1000，是一个普通用户。GID 的关系存储在 /etc/group 文件中：

```
$ cat /etc/group
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,firmy
......
```

Listing 2.5: 查看 gid

所有用户的信息（除了密码）都保存在 /etc/passwd 文件中，而为了安全起见，加密过的用户密码保存在 /etc/shadow 文件中，此文件只有root 权限可以访问。

```
$ sudo cat /etc/shadow
root:$6$root$wvK.pRXFEH8OGYkpiu1tEWYMOueo4tZtq7mYnldiyJBZDMe.mKwt.
    ↪ WIJnehb4bhZchL/93Oe1ok9UwxYf79yR1:17264::::::
firmy:$6$firmy$dhGT.WP91lnpG5/10GfGdj5L1fFVSoYlxwYHQn.
    ↪ llc5eKOvr7J8nqqGdVFKykMUSDNxix5Vh8zbXIapt0oPd8
    ↪ .:17264:0:99999:7:::
```

Listing 2.6: 查看 shadow+ 文件

由于普通用户的权限比较低，这里使用sudo 命令可以让普通用户以 root 用户的身份运行某一命令。使用su 命令则可以切换到一个不同的用户：

```
$ whoami
firmy
$ su root
# whoami
root
```

Listing 2.7: 切换用户

whoami 用于打印当前有效的用户名称，shell 中普通用户以$ 开头，root 用户以# 开头。在输入密码后，我们已经从firmy 用户转换到 root 用户了。

权限设置

在 Linux 中，文件或目录权限的控制分别以读取、写入、执行 3 种一般权限来区分，另有 3 种特殊权限可供运用。

使用ls -l [file] 来查看某文件或目录的信息：

```
$ ls -l /
lrwxrwxrwx   1 root root     7 Jun 21 22:44 bin -> usr/bin
drwxr-xr-x   4 root root  4096 Jul 28 08:48 boot
-rw-r--r--   1 root root 18561 Apr  2 22:48 desktopfs-pkgs.txt
```

Listing 2.8: 查看文件目录信息

第一栏从第二个字母开始就是权限字符串，权限表示三个为一组，依次是所有者权限、组权限、其他人权限。每组的顺序均为 rwx，如果有相应权限，则表示成相应字母，如果不具有相应权限，则用-表示。

- r：读取权限，数字代号为“4”
- w：写入权限，数字代号为“2”

- x：执行或切换权限，数字代号为“1”

通过第一栏的第一个字母可知，第一行是一个链接文件（l），第二行是个目录(d)，第三行是个普通文件（-）。　　用户可以使用chmod 指令去变更文件与目录的权限。权限范围被指定为所有者（u)、所属组（g)、其他人（o）和所有人（a)。

- -R：递归处理，将指令目录下的所有文件及子目录一并处理；
- < 权限范围 >+< 权限设置 >：开启权限范围的文件或目录的该选项权限设置
- $ chmod a+r [file]：赋予所有用户读取权限
- < 权限范围 >-< 权限设置 >：关闭权限范围的文件或目录的该选项权限设置
- $ chmod u-w [file]：取消所有者写入权限
- < 权限范围 >=< 权限设置 >：指定权限范围的文件或目录的该选项权限设置；
- $ chmod g=x [file]：指定组权限为可执行
- $ chmod o=rwx [file]：制定其他人权限为可读、可写和可执行

1	2	3	4	5	6	7	8	9	10
File Type	User Permission			Group Permission			Other Permission		
	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
d/l/s/p/-/c/b	r	w	e	r	w	e	r	w	e

Figure 2.1: 文件描述符

字节序

目前计算机中采用两种字节存储机制：大端 (Big-endian) 和小端 (Little-endian)。

Big-endian 规定 MSB 在存储时放在低地址，在传输时放在流的开始；LSB 存储时放在高地址，在传输时放在流的末尾。Little-endian 则相反。

常见的 Intel 处理器使用 Little-endian，而 PowerPC 系列处理器则使用 Big-endian，另外 TCP/IP 协议和 Java 虚拟机的字节序也是 Big-endian。

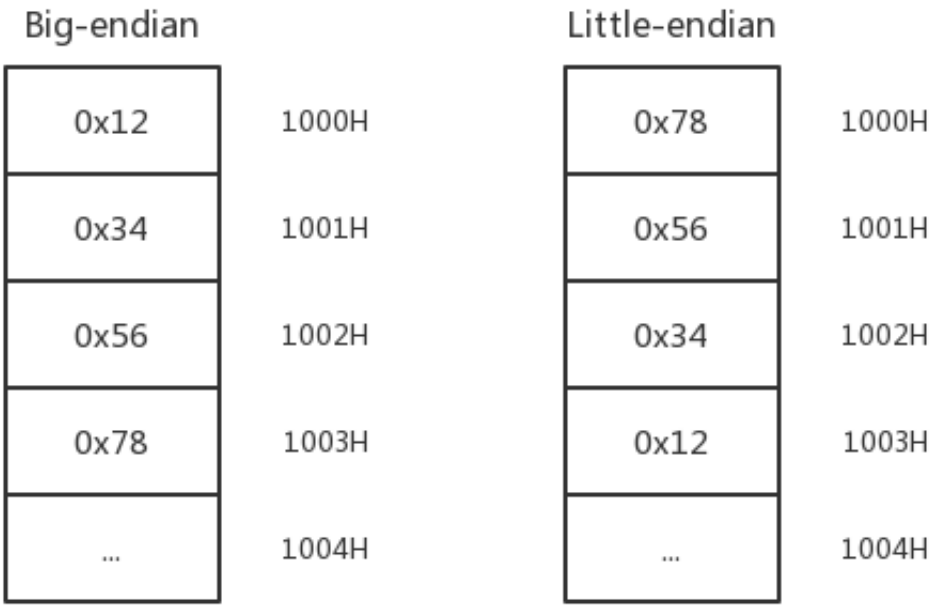

Figure 2.2: 字节序

我们在内存中实际地看一下，在地址 0xffffd584 处有字符 1234，在地址 0xffffd588 处有字符 5678。

```
gdb-peda$ x/w 0xffffd584
0xffffd584:     0x34333231
gdb-peda$ x/4wb 0xffffd584
0xffffd584:     0x31    0x32    0x33    0x34
gdb-peda$ python print('\x31\x32\x33\x34')
1234

gdb-peda$ x/w 0xffffd588
0xffffd588:     0x38373635
gdb-peda$ x/4wb 0xffffd588
0xffffd588:     0x35    0x36    0x37    0x38
gdb-peda$ python print('\x35\x36\x37\x38')
5678

gdb-peda$ x/2w 0xffffd584
0xffffd584:     0x34333231      0x38373635
gdb-peda$ x/8wb 0xffffd584
0xffffd584:     0x31    0x32    0x33    0x34    0x35    0x36    0x37
    ↪     0x38
gdb-peda$ python print('\x31\x32\x33\x34\x35\x35\x36\x37\x38')
123455678
db-peda$ x/s 0xffffd584
```

```
0xffffd584:     "12345678"
```

Listing 2.9: 内存中的字节

输入输出

- 使用命令的输出作为可执行文件的输入参数
- `$ ./vulnerable 'your_command_here'`
- `$ ./vulnerable $(your_command_here)`
- 使用命令作为输入
- `$ your_command_here | ./vulnerable`
- 将命令行输出写入文件
- `$ your_command_here > filename`
- 使用文件作为输入
- `$ ./vulnerable < filename`

文件描述符

在 Linux 系统中一切皆可以看成是文件，文件又分为：普通文件、目录文件、链接文件和设备文件。文件描述符（file descriptor）是内核管理已被打开的文件所创建的索引，使用一个非负整数来指代被打开的文件。

标准文件描述符如下：

文件描述符	用途	stdio 流
0	标准输入	stdin
1	标准输出	stdout
2	标准错误	stderr

当一个程序使用 fork 生成一个子进程后，子进程会继承父进程所打开的文件表，此时，父子进程使用同一个文件表，这可能导致一些安全问题。如果使用 fork()，子进程虽然运行于父进程的空间，但拥有自己的进程表项。

核心转储

当程序运行的过程中异常终止或崩溃，操作系统会将程序当时的内存、寄存器状态、堆栈指针、内存管理信息等记录下来，保存在一个文件中，这种行为就叫做核心转储（Core Dump）。

会产生核心转储的信号:

Signal	Action	Comment
SIGQUIT	Core	Quit from keyboard
SIGILL	Core	Illegal Instruction
SIGABRT	Core	Abort signal from abort
SIGSEGV	Core	Invalid memory reference
SIGTRAP	Core	Trace/breakpoint trap

开启核心转储:

- 输入命令 ulimit -c，输出结果为 0，说明默认是关闭的。
- 输入命令 ulimit -c unlimited 即可在当前终端开启核心转储功能。
- 如果想让核心转储功能永久开启，可以修改文件 /etc/security/limits.conf，增加一行：
- #<domain>　　<type>　<item>　　<value>
- *　　soft　　core　　unlimited

修改转储文件保存路径:

- 通过修改 /proc/sys/kernel/core_uses_pid，可以使生成的核心转储文件名变为 core.[pid] 的模式:
- # echo 1 > /proc/sys/kernel/core_uses_pid
- 还可以修改 /proc/sys/kernel/core_pattern 来控制生成核心转储文件的保存位置和文件名格式:

- `# echo /tmp/core-%e-%p-%t > /proc/sys/kernel/core_pattern`

此时生成的文件保存在 /tmp/ 目录下，文件名格式为 core-[filename]-[pid]-[time]。例子：

```
$ cat core.c
#include <stdio.h>
void main(int argc, char **argv) {
    char buf[5];
    scanf("%s", buf);
}
$ gcc -m32 -fno-stack-protector core.c
$ ./a.out
AAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
$ file /tmp/core-a.out-12444-1503198911
/tmp/core-a.out-12444-1503198911: ELF 32-bit LSB core file Intel
    ↪ 80386, version 1 (SYSV), SVR4-style, from './a.out', real uid:
    ↪ 1000, effective uid: 1000, real gid: 1000, effective gid: 1000,
    ↪  execfn: './a.out', platform: 'i686'
$ gdb a.out /tmp/core-a.out-12444-1503198911 -q
Reading symbols from a.out...(no debugging symbols found)...done.
[New LWP 12444]
Core was generated by `./a.out'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x5655559b in main ()
gdb-peda$ info frame
Stack level 0, frame at 0x41414141:
 eip = 0x5655559b in main; saved eip = <not saved>
 Outermost frame: Cannot access memory at address 0x4141413d
 Arglist at 0x41414141, args:
 Locals at 0x41414141, Previous frame's sp is 0x41414141
Cannot access memory at address 0x4141413d
```

Listing 2.10: 内存转储示例

调用约定

函数调用约定是对函数调用时如何传递参数的一种约定。关于它的约定有许多种，下面我们分别从内核接口和用户接口介绍 32 位和 64 位 Linux 的调用约定。

内核接口

x86-32 系统调用约定：Linux 系统调用使用寄存器传递参数。

eax 为 syscall_number，ebx、ecx、edx、esi、ebp 用于将 6 个参数传递给系统调用。返回值保存在 eax 中。所有其他寄存器（包括 EFLAGS）都保留在 int 0x80 中。

x86-64 系统调用约定：内核接口使用的寄存器有：rdi、rsi、rdx、r10、r8、r9。系统调用通过 syscall 指令完成。除了 rcx、r11 和 rax，其他的寄存器都被保留。系统调用的编号必须在寄存器 rax 中传递。系统调用的参数限制为 6 个，不直接从堆栈上传递任何参数。返回时，rax 中包含了系统调用的结果。而且只有 INTEGER 或者 MEMORY 类型的值才会被传递给内核。

用户接口

x86-32 函数调用约定：

参数通过栈进行传递。最后一个参数第一个被放入栈中，直到所有的参数都放置完毕，然后执行 call 指令。这也是 Linux 上 C 语言函数的方式。

x86-64 函数调用约定：

x86-64 下通过寄存器传递参数，这样做比通过栈有更高的效率。它避免了内存中参数的存取和额外的指令。根据参数类型的不同，会使用寄存器或传参方式。如果参数的类型是 MEMORY，则在栈上传递参数。如果类型是 INTEGER，则顺序使用 rdi、rsi、rdx、rcx、r8 和 r9。所以如果有多于 6 个的 INTEGER 参数，则后面的参数在栈上传递。

环境变量

/proc/[pid]

2.4 Web 安全基础

2.5 逆向工程基础

2.6 现代密码学基础

2.7 Android 安全基础

附录

Index