信息安全导论 hw2

小组成员：

- PB20111704 张宇昂
- PB20111647 鲍润晖
- PB20000103 王炳勋
- PB20111651 何泽昊

Problem 1

1. 由论文Section 3中的描述："When a process is created by fork, it inherits the tree user IDs from its parent process"，所以新进程的user id仍为x
2. 我们分这两种情况讨论：
   - 调用前euid = y > 0, saved user id suid=m and real user id ruid = m；调用后：进程可以为任何非特权uid
   - 调用前 euid = y = 0；调用后：进程只能为euid = 0
3. 从安全角度有如下好处：
   - 每个程序有单独的内存空间，防止内存互相访问导致的干扰与数据泄露
   - 分配独立的用户ID可以限制应用程序的访问权限，防止恶意程序通过攻击其他程序来获得权限
4. 注意到第二问，zygote的uid是root id，而由zygote创建的进程具有和zygote相同的uid即root id，调用setuid从安全性的角度考虑是降低进程的权限来避免其访问不该访问的资源
5. 可以用 `sudo chmod u+s /usr/bin/passwd` 来设置setuid让passwd以root权限运行，所以非root用户就无需root权限去修改密码；需要仔细编写passwd源代码的原因是该程序需要root权限运行，若该程序存在安全漏洞，则攻击者可以通过攻击该程序去获取root权限并对密码文件进行篡改，或对系统进行攻击等操作，所以仔细编写passwd是非常重要的

Problem 2

10位密码的所有可能性为95^{10}，所以完全穷举测试密码需要的时间为：$95^{10} \div 6400000 = 9355264675599.67s$，这等价于2165566年

Problem 3

a. 当程序被当做setuid root 程序运行时，操作系统会赋予文件所有者的权限。因此，如果攻击者攻击者创建了一个名为“file.dat”的符号链接指向另一个文件，运行此代码会使该文件中的内容被修改为“Hello world”。
b. 会发生。在检查文件存在与打开文件的操作之间，由于各行代码执行之间仍存在时间间隔，攻击者仍有时间创建指向另一个文件的链接。
c. 若要避免a中的问题，应在sleep后再次stat()文件，以确保文件路径未更改。更改后代码如下：

```
if (!stat("./file.dat", buf)) return; // abort if file exists
sleep(10); // sleep for 10 seconds
if (!stat("./file.dat", buf)) return; // abort if file exists
fp = fopen("./file.dat", "w" ); // open file for write
fprintf(fp, "Hello world" );
close(fp);
```

这会避免在第一个stat()后创建符号链接。

Problem 4

a. 攻击者可以通过收集常用的密码，然后对他们执行MD5或SHA-1，并与泄露的数据库中的密码进行对比。使用salt后，攻击者需要用相应的salt计算这些常用密码的散列，这大大增加了计算量,因此增加了安全性。
b. 增加salt的size大小会增加攻击者破解密码的难度，然而也存在副作用：会占用更多的存储空间，也可能导致不兼容问题。

Problem 5

a

优点：
更灵活地控制内存访问权限

缺点：
更复杂，开销更大

如果在用户层想要访问一些内核层才有的基础功能时，需要更多层的切换

b

MULTITICS OS使用七个操作系统层
如今，许多处理器都允许虚拟机管理程序模式，该模式称为第 0 层，并授予直接硬件访问权限。它与VMware等虚拟化软件结合使用

Problem 6

a

- 创建用户test，用 `sudo useradd test`
- 设置该用户密码，用 `sudo passwd test`
  - 密码设置为abcd

b

- 读取 `/etc/passwd` ，可以看到与test用户有关的信息
  - `test:x:1001:1001::/home/test:/bin/sh`
- 读取 `/etc/shadow` ，找到test用户所在行
  - `test:$6$iYqlOej8EcVFHrDJ$FJg26ZeXZlMz6hdJ3iOdt9r/vgMOLXBTJvNyfsYSgKcHT0f6tkrH1kOeum/HjjiBEJKBQC50RwwgamO6D6jfA.:19485:0:99999:7:::`

c

- shadow entry format
  如下为示意图
  REF

```
mark:$6$.n.:17736:0:99999:7:::
[--] [----] [---] - [---] ----
|      |      |   |   |   |||+-----------> 9. Unused
|      |      |   |   |   ||+------------> 8. Expiration date
|      |      |   |   |   |+-------------> 7. Inactivity period
|      |      |   |   |   +--------------> 6. Warning period
|      |      |   |   +------------------> 5. Maximum password age
|      |      |   +----------------------> 4. Minimum password age
|      |      +--------------------------> 3. Last password change
|      +---------------------------------> 2. Encrypted Password
+----------------------------------------> 1. Username
```

  - 对于2，使用 `$type$salt$hashed` 格式， `$type` 代表的哈希算法如下
    `$1$` – MD5
    `$2a$` – Blowfish
    `$2y$` – Eksblowfish
    `$5$` – SHA-256
    `$6$` – SHA-512
- parse out the salt
  使用随机字符码混合密码加密算法产生新的密码
- password hash
  将密码加盐，计算出hash值，以防止字典攻击
- hash algorithm
  - SHA-512
    - 填充
      在初始消息中增加填充位，使初始消息长度等于1024的倍数少128位
    - 添加长度
      计算不包括填充位的消息长度，将其作为一个128位的块附加到填充位的后面
    - 分块
      将得到的内容分成512位的块
    - 初始化链接变量
      在SHA-512中，生成一个长度为512位的消息摘要，需要8个64位链接变量
    - 处理块
      （1）将链接变量A~ H复制到变量a~ h中，看作单个寄存器，用于存储临时的中间结果和最终结果
      （2）将当前的1024位块分解成16子块
      （3）SHA-512有80轮，每轮以当前的1024位块、寄存器abcdefgh和常量K[t]（其中t=0~ 79）作为三个输入，用SHA-512算法步骤更新寄存器abcdrfgh的内容

d

- 操作如下

```
naxiphos@ubuntu:/etc$ openssl passwd -6 -salt iYqlOej8EcVFHrDJ abcd
$6$iYqlOej8EcVFHrDJ$FJg26ZeXZlMz6hdJ3iOdt9r/vgMOLXBTJvNyfsYSgKcHT0f6tkrH1kOeum/
HjjiBEJKBQC50RwwqamO6D6jfA.
```

  可以看出，我们用openssl计算出的结果与储存在shadow文件中的结果一致

e

- 更改密码

  更改至 `efgh`

  - shadow文件

```
test:$6$atIHUOfspBjQyhNM$/LYn65eJSLtGNgQkgeMyhlJiN4j5Pclac1eThI8CJSixxg0M.KKDQk
ToKZRjXpDpIq1rkg13nY2Oz7I95DyKZ.:19485:0:99999:7:::
```

  - 计算结果

```
naxiphos@ubuntu:/etc$ openssl passwd -6 -salt atIHUOfspBjQyhNM efgh
$6$atIHUOfspBjQyhNM$/LYn65eJSLtGNgQkgeMyhlJiN4j5Pclac1eThI8CJSixxg0M.KKDQkToKZR
jXpDpIq1rkg13nY2Oz7I95DyKZ.
```

- 不难看出，重复实验得到相同的实验结果

Problem 7

选择论文**The security architecture of the chromium browser**

summary

本文介绍了Chromium浏览器的安全架构：它将浏览器分为浏览器内核和渲染引擎两个保护域。后者被置于沙盒之中，对于浏览器内核可被视为黑盒。

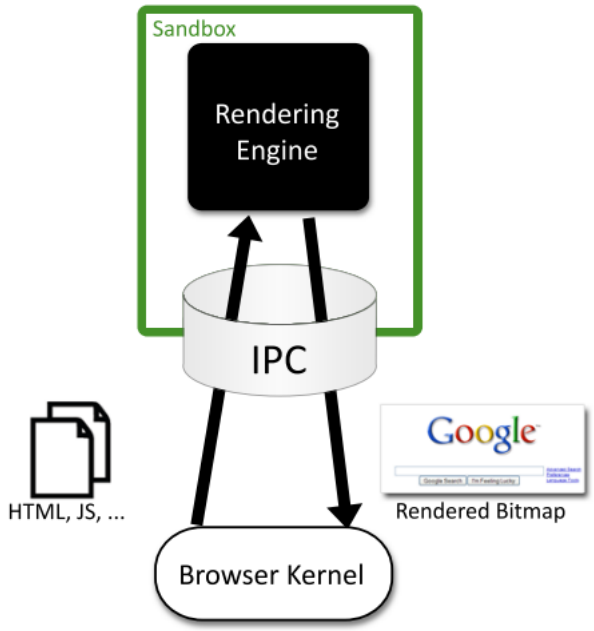

Figure 1: The browser kernel treats the rendering engine as a black box that parses web content and emits bitmaps of the rendered document.

同时也为浏览器漏洞定义了一个威胁模型

2. THREAT MODEL

本篇论文使用如下威胁模型来评估 Chromium 的架构如何有效地保护用户免受攻击

其中假设攻击者具有如下能力：

- 攻击者拥有一个尚未添加到浏览器黑名单的域名、域名的有效 HTTPS 证书，并控制网络上的至少一台主机
- 攻击者能够说服用户访问他或她的网站

- 攻击者知道并能够利用用户浏览器中未修补的任意漏洞

而Chromium架构可以防止攻击者达到以下的目标：

- 安装持久性的恶意软件（即使关闭浏览器也可以运行的软件）
- 在用户与别的程序交互时监视用户的击键
- 读取用户硬盘上的敏感文件

3. CHROMIUM'S ARCHITECTURE

Chromium的架构有两个模块：渲染引擎和浏览器内核。在高层次上，渲染引擎负责将 HTTP 响应和用户输入事件转换为渲染的位图，而浏览器内核负责与操作系统交互。
浏览器内核被信任充当用户，而渲染引擎仅被信任充当网络。

浏览器内核公开了一个 API，渲染引擎使用该 API 发出网络请求、访问持久性存储以及在用户屏幕上显示位图。

Rendering Engine	Browser Kernel
HTML parsing	Cookie database
CSS parsing	History database
Image decoding	Password database
JavaScript interpreter	Window management
Regular expressions	Location bar
Layout	Safe Browsing blacklist
Document Object Model	Network stack
Rendering	SSL/TLS
SVG	Disk cache
XML parsing	Download manager
XSLT	Clipboard

Both
URL parsing
Unicode parsing

4. THE SANDBOX

渲染引擎负责大部分解析和解码任务，从历史上看，这些任务一直是
大量浏览器漏洞的来源。所以为了抵御利用这些漏洞功能的攻击者，Chromium 在沙盒中运行每个渲染引擎。这个沙盒限制渲染引擎的进程发出一些关键的系统调用

理想情况下，沙箱会强制渲染引擎使用浏览器内核 API 与外部世界交互

目前，Chromium 依赖于 Windows 的特定功能来对渲染引擎进行沙箱处理。每当渲染引擎尝试访问“安全对象”时，Windows 安全管理器都会检查渲染引擎的安全令牌是否具有足够的权限来访问该对象。

5. THE BROWSER KERNEL INTERFACE

以下是一些浏览器操作通过渲染引擎和浏览器交互的实现方式：

- 渲染
  渲染引擎并不访问窗口句柄。为了向用户显示位图，渲染引擎将位图发送给浏览器内核，浏览器内核将位图复制到屏幕
- 用户输入
  操作系统不是将用户输入直接传递给渲染引擎，而是将这些事件传递给浏览器内核，浏览器内核根据当前聚焦的用户界面元素调度这些事件
- 联网
  渲染引擎不是直接访问网络，而是通过浏览器内核从网络中检索 URL。在为 URL 请求提供服务之前，浏览器内核会检查渲染引擎是否已获得请求 URL 的授权

Advantage

论文作者调查了最近的浏览器漏洞，发现 Chromium 漏洞中67.4%会出现在渲染引擎。他们还发现该架构将具有缓解了 70.4%（54 个中的 38 个）最严重的漏洞。

论文中还对比了该架构较其它浏览器架构的优点：

- 与现有站点兼容，架构支持 Web 平台的所有功能。

- 将渲染引擎视为黑盒，降低了浏览器内核的复杂性。
- 最大限度地减少用户安全决策，避免不断出现安全提示。

Disadvantage

但是它仍有一些缺点：

- 沙盒能力在一些场景下不适用：如 FAT32 文件系统不支持访问控制列表。因此 Windows 安全管理器在授予对 FAT32 文件的访问权限时会忽略进程的安全令牌
- 该架构也有无法解决的安全问题，比如：
  - 网络钓鱼：攻击者诱使用户将不诚实的网站与诚实的网站混淆
  - 源隔离：Chromium 的架构将渲染引擎视为代表整个 Web 主体，这意味着破坏渲染引擎的攻击者可以代表任何网站进行操作

这些问题需要配合上别的机制才能解决。比如为了解决前者，浏览器将已知的钓鱼网站列入黑名单；为了解决后者，要在于渲染引擎中执行同源策略