哈希长度扩展攻击

1 绪论

当客户端和服务器通信时，它们可能会受到中间人（MITM）攻击。攻击者可以拦截来自客户端的请求，他可能选择修改数据，然后将修改后的请求发送到服务器。在这种情况下，服务器需要验证收到的请求的完整性。验证完整性的标准方法是在请求上附加一个称为消息认证码（MAC）的标签。计算 MAC 的方法有很多，但其中一些方法并不安全。

MAC 是用密钥和消息计算出来的。一种简单的计算 MAC 的方法是将密钥与消息连接起来组成一个字符串，然后计算其单向哈希。这种方法看起来不错，但是会遭受长度扩展攻击。这种攻击使攻击者可以修改消息，并可以为修改后的消息生成有效的 MAC，攻击者无需知道密钥。

本实验的目的是帮助学生了解长度扩展攻击的工作原理。学生将对服务器程序发起攻击，他们将伪造一个有效的命令让服务器执行该命令。

阅读材料 对单向哈希函数的详细介绍可以参见：

- SEED 书, *Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.

实验环境 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。既然我们使用容器来建立实验环境，本实验不太依赖 SEED VM。您可以在其他 VM、物理机器以及云端 VM 上进行此实验。

2 实验环境

我们为此实验设立了一个 Web 服务器。客户端可以向该服务器发送一系列的命令。每个请求必须附加一个有效的 MAC。服务器只有成功验证了 MAC 后才会执行请求中的命令。我们使用主机 VM 作为客户端，并使用一个容器作为 Web 服务器。

容器安装及其命令 请从实验的网站下载 `Labsetup.zip` 文件到你的 VM 中，解压它，进入 `Labsetup` 文件夹，然后用 `docker-compose.yml` 文件安装实验环境。对这个文件及其包含的所有 `Dockerfile` 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 .bashrc 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 docker ps 命令找出容器的 ID ，然后使用 docker exec 在该容器上启动 Shell 。我们已经在 .bashrc 文件中为这两个命令创建了别名。

```
$ dockps        // docker ps --format "{{.ID}}  {{.Names}}" 的别名
$ docksh <id>   // docker exec -it <id> /bin/bash 的别名

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
//     输入整个 ID 字符串。只要它们在所有容器当中
//     是独一无二的, 那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

关于 Web 服务器。 我们将服务器程序部署在 www.seedlab-hashlen.com 域名上。在我们的 VM 中，我们通过将以下条目添加到 /etc/hosts 文件中来实现将此主机名映射到 Web 服务器容器 10.9.0.80（如果该条目不在 VM 中，请添加该条目）。

```
10.9.0.80  www.seedlab-hashlen.com
```

服务器代码位于 Labsetup/image_flask/app 文件夹中。它有两个目录：www 目录包含服务器代码，LabHome 目录包含秘密文件和用于计算 MAC 的密钥。

发送请求。 服务器程序接受以下命令:

- `lstcmd` 命令: 服务器将列出 `LabHome` 文件夹中的所有文件。
- `download` 命令: 服务器将从 `LabHome` 目录返回指定文件的内容。

客户端发送到服务器的正常请求如下所示。客户端需要传递 `uid` 参数给服务器。服务器根据 `uid` 从 `LabHome/key.txt` 获取 MAC 密钥。下例中的命令为 `lstcmd` ，其值设置为 `1`。它请求服务器列出所有文件。最后一个参数是基于密钥（由客户端和服务器共享）和命令参数计算出的 MAC 。在执行命令之前，服务器将验证 MAC 以确保命令的完整性。

```
http://www.seedlab-hashlen.com/?myname=JohnDoe&uid=1001&lstcmd=1
&mac=dc8788905dbcbceffcdd5578887717c12691b3cf1dac6b2f2bcfabc14a6a7f11
```

学生应将 `myname` 字段中的 `JohnDoe` 值替换为真实的名字（不允许使用空格）。此参数是为了确保不同学生的结果不同。服务器不使用此参数，但会检查该参数是否存在。如果不包含此字段，则请求会被拒绝。教师可以使用此参数来检查学生是否自己完成了作业。如果学生在此任务中不使用真实姓名，则没有任何分数。

下面展示了另一个示例。该请求包括两个命令：列出所有文件并下载文件 `secret.txt`。同样需要附加有效的 MAC ，否则服务器将不会执行这些命令。

```
http://www.seedlab-hashlen.com/?myname=JohnDoe&uid=1001&lstcmd=1
&download=secret.txt
&mac=dc8788905dbcbceffcdd5578887717c12691b3cf1dac6b2f2bcfabc14a6a7f11
```

3 实验任务

3.1 任务 1: 发送列出文件的请求

在此任务中，我们将向服务器发送一个正常请求，这样我们可以看到服务器如何响应该请求。我们要发送的请求如下：

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<need-to-fill>
&lstcmd=1&mac=<need-to-calculate>
```

要发送这样的请求，除了使用我们的真实姓名外，我们需要填写两个缺少的参数。学生需要从 `LabHome` 目录中的 `key.txt` 中选择一个 uid 号。该文件列出了每个 uid 的密钥。学生可以使用任何 uid 及其关联的密钥。例如，学生可以使用 uid `1001` 及其密钥 `123456`。

缺少的第二个参数是 MAC ，先将密钥与请求 `R` 的内容（仅参数部分）连接起来，其之间用冒号分开。请参见以下示例：

```
Key:R = 123456:myname=JohnDoe&uid=1001&lstcmd=1
```

然后使用以下命令计算 MAC ：

```
$ echo -n "123456:myname=JohnDoe&uid=1001&lstcmd=1" | sha256sum
7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3  -
```

这样我们就可以构造完整的请求，然后使用浏览器将其发送到服务器程序：

```
http://www.seedlab-hashlen.com/?myname=JohnDoe&uid=1001&lstcmd=1
&mac=7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3
```

任务 请向服务器发送下载命令，并显示可以获得结果。

3.2 任务 2: 构建填充

要进行哈希长度扩展攻击，我们需要了解如何为单向哈希计算填充。SHA-256 的块大小为 64 字节，因此在哈希计算过程中，消息 M 将被填充为 64 字节的倍数。根据 RFC 6234 ，SHA256 的填充包括一个字节的 \x80 ，后跟多个 0 ，然后是一个 64 位（8 字节）的长度字段（长度是 M 的长度，即二进制数的位数）。

假定原始消息是 M = "This is a test message"。M 的长度为 22 字节，因此填充为 64-22 = 42 字节，包括长度字段的 8 字节。M 的二进制数的长度为 22 * 8 = 176 = 0xB0。所以，加了填充的数据如下：

```
"This is a test message"
"\x80"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\xB0"
```

需要注意的是，长度字段使用大端序，即如果消息的长度为 0x012345 ，则填充中的长度字段应为：

```
"\x00\x00\x00\x00\x00\x01\x23\x45"
```

任务。 学生需要为以下消息构造填充数据（<key> 和 <uid> 的实际值应从 LabHome/key.txt 文件中获取。

```
<key>:myname=<name>&uid=<uid>&lstcmd=1
```

3.3 任务 3: 使用密钥计算 MAC

在本实验中，我们添加一个额外的消息 N = "Extra message" 到填充后的原始消息上：M = "This is a test message" ，并计算它的哈希值。该程序如下所示。

```
/* calculate_mac.c */
#include <stdio.h>
#include <openssl/sha.h>

int main(int argc, const char *argv[])
```

```
{
   SHA256_CTX c;
   unsigned char buffer[SHA256_DIGEST_LENGTH];
   int i;

   SHA256_Init(&c);
   SHA256_Update(&c,
      "This is a test message"
      "\x80"
      "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
      "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
      "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
      "\x00\x00\x00"
      "\x00\x00\x00\x00\x00\x00\x00\xB0"
      "Extra message",
      64+13);
   SHA256_Final(buffer, &c);

   for(i = 0; i < 32; i++) {
     printf("%02x", buffer[i]);
   }
   printf("\n");
   return 0;
}
```

学生可以编译运行上面的程序：

```
$ gcc calculate_mac.c -o calculate_mac -lcrypto
$ ./calculate_mac
```

任务。 学生应该更改上面清单中的代码，为以下请求计算 MAC（假设我们知道 MAC 密钥）：

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>
&lstcmd=1<padding>&download=secret.txt
&mac=<hash-value>
```

就像上一个任务一样，`<name>` 的值应该是你的真实姓名。`<uid>` 和 MAC 密钥的值应从 `LabHome/key.txt` 文件中获取。把请求发送到服务器，查看能否成功下载 `secret.txt` 文件。

应该注意的是，在 URL 中，需要通过将 `\x` 更改为 `%` 来对填充中的所有十六进制数字进行编码。例如，在上面的 URL 中，填充中的 `\x80` 应该替换为 `%80`。在服务器端，URL 中的编码数据将被改回十六进制数。

3.4 任务 4: 长度扩展攻击

在上一个任务中，我们展示了合法用户如何计算 MAC （了解 MAC 密钥）。在此任务中，我们将以攻击者的身份进行，即我们在不知道 MAC 密钥的情况下计算 MAC。假设我们知道了一个有效的请求 `R`，我们也知道 MAC 秘钥的长度，我们的工作是基于 `R` 生成新的请求，同时仍然能够计算有效的 MAC 。

给定原始消息 `M = "This is a test message"` 及其 MAC 值，我们将说明如何在填充的 `M` 的末尾添加消息 `"Extra message"` ，然后在不知道秘密 MAC 密钥的情况下计算其 MAC 。

```
$ echo -n "This is a test message" | sha256sum
6f3438001129a90c5b1637928bf38bf26e39e57c6e9511005682048bedbef906
```

以下程序可用于计算新消息的 MAC ：

```
/* length_ext.c */
#include <stdio.h>
#include <arpa/inet.h>
#include <openssl/sha.h>

int main(int argc, const char *argv[])
{
  int i;
  unsigned char buffer[SHA256_DIGEST_LENGTH];
  SHA256_CTX c;

  SHA256_Init(&c);
  for(i=0; i<64; i++)
     SHA256_Update(&c, "*", 1);

  // MAC of the original message M (padded)
  c.h[0] = htole32(0x6f343800);
  c.h[1] = htole32(0x1129a90c);
  c.h[2] = htole32(0x5b163792);
  c.h[3] = htole32(0x8bf38bf2);
  c.h[4] = htole32(0x6e39e57c);
  c.h[5] = htole32(0x6e951100);
  c.h[6] = htole32(0x5682048b);
  c.h[7] = htole32(0xedbef906);

  // 添加额外信息
  SHA256_Update(&c, "Extra message", 13);
  SHA256_Final(buffer, &c);

  for(i = 0; i < 32; i++) {
     printf("%02x", buffer[i]);
  }
```

```
  printf("\n");
  return 0;
}
```

如下编译上面的程序:

```
$ gcc length_ext.c -o length_ext -lcrypto
```

任务。 学生应首先为以下请求生成有效的 MAC （其中 <uid> 和 MAC 密钥应从 LabHome/key.txt 文件获得）:

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>
&lstcmd=1&mac=<mac>
```

请基于上面计算出的 <mac> 构造一个新的、包含 download 命令的请求。这一次，你不允许使用密钥。URL 形如:

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>
&lstcmd=1<padding>&download=secret.txt&mac=<new-mac>
```

把构造出的请求发送给服务器，展示你能成功获得 secret.txt 文件的内容。

3.5 任务 5: 使用 HMAC 缓解攻击

到目前为止,我们已经看到通过将密钥和消息连接在一起计算 MAC 所造成的危害。在此任务中,我们将改正这一错误。计算 MAC 的标准方法是使用 HMAC 。学生应该修改服务器程序的 verify_mac() 函数，并使用 Python 的 hmac 模块来计算 MAC。该函数位于 lab.py 中。给定密钥和消息（均为字符串类型），计算 HMAC 如下所示:

```
mac = hmac.new(bytearray(key.encode('utf-8')),
               msg=message.encode('utf-8', 'surrogateescape'),
               digestmod=hashlib.sha256).hexdigest()
```

在使用 HMAC 进行 MAC 计算时，学生应重复任务 1 以发送列出文件的请求。假设选择的密钥是 123456 ，则可以通过以下方式计算 HMAC :

```
$ python3
Python 3.5.2
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> import hmac
>>> import hashlib
>>> key='123456'
>>> message='lstcmd=1'
>>> mac = hmac.new(bytearray(key.encode('utf-8')),
                   msg=message.encode('utf-8', 'surrogateescape'),
                   digestmod=hashlib.sha256).hexdigest()
```

学生应说明为什么当客户端和服务器使用 HMAC 时，使用长度扩展和附加命令的恶意请求将使 MAC 验证失败。

4 递交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。