登录

Exim off-by-one RCE：CVE2018-6789利用及保护机制的绕过

行之 / 2018-03-09 12:52:40 / 浏览数 10075 安全技术 漏洞分析 顶(0) 踩(0)

前言

原文：https://devco.re/blog/2018/03/06/exim-off-by-one-RCE-exploiting-CVE-2018-6789-en/

CVE2018-6789是一个off-by-one的漏洞，文章对该漏洞的利用流程进行了详细的表述。
译者在文章开始前总结一些简单的预备知识。

off-by-one漏洞

off-by-one意为一个字节溢出。

栈：

这里从网上引用一个demo便于理解

```
#include <stdio.h>
#include <string.h>
void foo(char* arg);
void bar(char* arg);
void foo(char* arg) {
bar(arg); /* [1] */
}
void bar(char* arg) {
char buf[256];
strcpy(buf, arg); /* [2] */
}
int main(int argc, char *argv[]) {
if(strlen(argv[1])>256) { /* [3] */
 printf("Attempted Buffer Overflow\n");
 fflush(stdout);
 return -1;
}
foo(argv[1]); /* [4] */
return 0;
}
```

结合代码和图片来看，从代码可以看到当用户输入256字节的数据，foo函数调用strcpy(buf, arg);
执行时,foo的EBP的LSB会被覆盖。从图中可以看出当EBP被一个NULL字节所覆盖时，ebp从0xbffff2d8变为0xbffff200，由于用户输入被复制到该目标缓冲区，攻击者可以

堆：

由于ptmalloc的堆块验证机制的不完善，使得即使只有一个字节的溢出也使堆的off-by-one漏洞变得可利用。简单举个例子。
假设有这样3个块：

之后A发生了off-by-one于是堆结构变成了这个样子

图中的红色区域我们可以改掉Bblock的大小，使其增加到C，之后我们free掉B,再分配B+C大小的块，这样可以间接实现对CBlock的读写。

ACL访问控制列表

ACL使Access Control
List的缩写，主要的目的是在提供传统的owner,group,others的read,write,execute权限之外的细部权限设定。ACL可以针对单一的使用者，单一的档案或目录来进行r,w,x的

传统的Linux下，上面的权限分配正常但是当下面的情况出现时，就出现了问题:

上图情况出现时，就出现了问题，而这也是ACL所解决的。

OverView

我们在2018年2月5日报告了Exim的base64解码函数中的溢出漏洞，标识为CVE-2018-6789。
自从exim第一次发布以来就存在这个错误，因此所有版本都受到影响。
根据我们的研究，可以利用它来获得预授权远程代码执行，并且至少有400,000台服务器处于风险之中。 补丁版本4.90.1已经发布，我们建议立即升级exim。

Affected

所有低于4.90.1版本的Exim

One byte overflow in base64 decoding

Vulnerability Analysis
漏洞的成因在b64decode函数中解码缓冲区长度的计算错误:

```
base64.c:153b64decode

b64decode(const uschar* code, uschar **ptr)
{
Int x, y;
Uschar* result = store_get(3*(Ustrlen(code)/4)+1);
*ptr = result;
//perform decoding
}
```

如上所示，exim分配一个3 *（len / 4）+1字节的缓冲区来存储解码后的base64数据。 但是，当输入不是有效的base64字符串且长度为4n + 3时，exim分配3n + 1，但在解码时会占用3n + 2个字节。 这会导致单字节堆溢出（aka逐个）。

一般来说，这个错误是无害的，因为被覆盖的通常是未使用的内存。 但是，当字符串适合某些特定长度时，该字节会覆盖一些关键数据。
值得注意的是，由于这个字节是可控的，使得对其利用更加可行。另外，Base64解码是一个基本功能，因此这个错误可以很容易地触发，导致远程代码执行。

Exploitation

为了评估这个错误的严重程度，我们开发了一个针对exim的SMTP守护进程的攻击。 以下段落描述了用于实现pre-auth远程代码执行的开发机制。
为了利用这一个字节的溢出，我们有必要诱骗内存管理机制。此外在阅读本节之前，强烈建议您具有堆漏洞利用的基本知识。

我们的EXP需要一下几样东西：
Debain(stretch) and Ubuntu(zesty)
SMTP daemon of Exim4 package installed with apt-get(4.89/4.88)
Config enabled(uncommented in default config)CRAM-MD5 authenticator(any other authenticator using base64 alse works)
Basic SMTP sommands(EHLO,MAIL FROM/RCPT TO)and AUTH

Memory allocation

首先，我们回顾一下源代码并搜索有用的内存分配。 正如我们在前一篇文章中提到的，exim使用自定义函数进行动态分配：

```
extern BOOL    store_extend_3(void *, int, int, const char *, int);  /* The */
extern void    store_free_3(void *, const char *, int);     /* value of the */
extern void   *store_get_3(int, const char *, int);         /* 2nd arg is   */
extern void   *store_get_perm_3(int, const char *, int);    /* __FILE__ in  */
extern void   *store_malloc_3(int, const char *, int);      /* every call,  */
extern void    store_release_3(void *, const char *, int);  /* so give its  */
extern void    store_reset_3(void *, const char *, int);    /* correct type */
```

函数store_free（）和store_malloc（）直接调用glibc的malloc（）和free（）。
Glibc需要一个稍大的（0x10字节）块，并将其元数据存储在每个分配的第一个0x10字节（x86-64）中，然后返回数据的位置。 下面的插图描述了块的结构：

元数据包括前一个块的大小（正好在内存中的那个），当前块的大小和一些标志。 大小的前三位用于存储标志。
在这个例子中，0x81的大小意味着当前块是0x80字节，并且前一个块正在使用中。
在exim中,大部分被释放的块被放入一个双向链表中，称为unsorted bin。
Glibc根据标志位维护它为了避免碎片化，Glibc会将相邻的已被释放块合并到一个更大的块。
对于每个分配请求，glibc都会以FIFO（先进先出）顺序检查这些块，并重新使用这些块。

针对一些性能问题，exim使用store_get（），store_release（），store_extend（）和store_reset（）维护自己的链表结构。

storeblocks的主要特点是每块至少有0x2000字节，这使我们的漏洞利用受到限制。 请注意，storeblock也是数据块。
因此，如果我们查看内存，其内存结构看起来就像这个样子：

这里我们列举出用来部署堆数据的函数：

EHLO主机名
对于每个EHLO（或HELO）命令，exim将主机名的指针存储在sender_host_name中。
store_free（）旧名称
store_malloc（）新名称

```
smtp_in.c: 1833 check_helo
/* Discard any previous helo name */

if (sender_helo_name != NULL)
{
store_free(sender_helo_name);
sender_helo_name = NULL;
}
...
if (yield) sender_helo_name = string_copy_malloc(start);
return yield;
```

无法识别的命令
对于每个无法识别的带有不可打印字符的命令，exim都会分配一个缓冲区来将其转换为可打印的
store_get（）存储错误消息

```
smtp_in.c: 5725 smtp_setup_msg
done = synprot_error(L_smtp_syntax_error, 500, NULL,
US"unrecognized command");
```

AUTH
在大多数身份验证过程中，exim使用base64编码与客户端进行通信。 编码和解码字符串存储在由store_get（）分配的缓冲区中。
store_get（）用于字符串
可以包含不可打印的字符，NULL字节
不一定是null终止

重置EHLO / HELO，MAIL，RCPT
每当有命令正确完成时，exim就会调用smtp_reset（）。
此函数调用store_reset（）将块链重置为重置点，这意味着在last命令后所有通过store_get（）分配的storeblocks都会被释放。
store_reset（）重置点（在函数的开始处设置）
在释放块的时候添加

```
smtp_in.c: 3771 smtp_setup_msg

int
smtp_setup_msg(void)
{
int done = 0;
BOOL toomany = FALSE;
BOOL discarded = FALSE;
BOOL last_was_rcpt = FALSE;
void *reset_point = store_get(0);

DEBUG(D_receive) debug_printf("smtp_setup_msg entered\n");

/* Reset for start of new message. We allow one RSET not to be counted as a
nonmail command, for those MTAs that insist on sending it between every
message. Ditto for EHLO/HELO and for STARTTLS, to allow for going in and out of
TLS between messages (an Exim client may do this if it has messages queued up
for the host). Note: we do NOT reset AUTH at this point. */

smtp_reset(reset_point);
```

Exploit steps

为了充分利用off-by-one，解码后的base64数据下的块应该易于释放和控制。 经过多次尝试，我们发现sender_host_name是一个不错的选择。
我们安排堆布局，为base64数据留下一个空闲的块，高于sender_host_name。我们在sender_host_name之前留下一个空闲快给base64数据

Put a huge chunk into unsorted bin
首先，我们发送一个包含巨大主机名的EHLO消息，以使其在堆中分配和释放,留下一个0x6060长度的unsorted bin。

Cut the first storeblock
然后我们发送一个无法识别的字符串来触发store_get（ ）并在释放的块内分配storeblock。

Cut the second storeblock and release the first one
我们再次发送EHLO消息以获得第二个存储区。 由于EHLO完成后调用了smtp_reset，所以第一个块被顺序释放。
堆布局准备好后，我们可以使用off-by-one覆盖原始块大小。 我们将0x2021修改为0x20f1，这稍微扩展了块。

Send base64 data and trigger off-by-one
要触发off-by-one，我们启动一个AUTH命令来发送base64数据。 溢出字节正好覆盖下一个块的第一个字节并扩展下一个块。

Forge a reasonable chunk size
由于块已扩展，下一块块的开始被更改为原始块的内部。 因此，我们需要让它看起来像一个正常的块来通过glibc的理智检查。
我们在这里发送另一个base64字符串，因为它需要空字节和不可打印字符来伪造块大小。

Release the extended chunk
要控制扩展块的内容，我们需要首先释放块，因为我们无法直接编辑块。 也就是说，我们应该发送一个新的EHLO消息来释放旧的主机名。
但是，正常的EHLO消息在成功之后会调用smtp_reset，这可能会导致程序中止或崩溃。 为了避免这种情况，我们发送一个无效的主机名称，如a+。

Overwrite the next pointer of overlapped storeblock

块释放后后，我们可以使用AUTH检索它并覆盖部分重叠的存储块。 这里我们使用一种称为partial write的技巧。
有了这个，我们可以在不破坏ASLR（地址空间布局随机化）的情况下修改指针。
我们部分地改变了包含ACL（访问控制列表）字符串的storeblock的下一个指针。 ACL字符串是由一组全局指针指向的，例如：

```
uschar *acl_smtp_auth;
uschar *acl_smtp_data;
uschar *acl_smtp_etrn;
uschar *acl_smtp_expn;
uschar *acl_smtp_helo;
uschar *acl_smtp_mail;
uschar *acl_smtp_quit;
uschar *acl_smtp_rcpt
```

这些指针在exim进程开始时根据配置进行初始化设置。 例如，如果configure中有一行acl_smtp_mail =
acl_check_mail，则指针acl_smtp_mail指向字符串acl_check_mail。 无论何时使用MAIL FROM，exim都会先扩展acl_check_mail来执行ACL检查。
在扩展时，如果遇到$ {run {cmd}}，exim会尝试执行命令，所以只要我们控制ACL字符串，就可以实现代码执行。
另外，我们不需要直接劫持程序控制流程，因此我们可以轻松地绕过诸如PIE（位置独立可执行文件），NX等保护机制。

Reset storeblocks and retrieve the ACL storeblock
现在，ACL存储块位于链接列表链中。 一旦smtp_reset（ ）被触发，它将被释放，然后我们可以通过分配多个块来再次检索它。

Overwrite ACL strings and trigger ACL check
最后，我们覆盖包含ACL字符串的整个块。 现在我们发送诸如EHLO，MAIL，RCPT等命令来触发ACL检查。
一旦我们触及配置中定义的acl，我们就可以实现远程代码执行。

参考链接：
https://googleprojectzero.blogspot.com/
https://sploitfun.wordpress.com/2015/06/09/off-by-one-vulnerability-heap-based/
https://sploitfun.wordpress.com/2015/02/26/heap-overflow-using-unlink/
https://bbs.pediy.com/thread-217390.htm
https://www.contextis.com/resources/white-papers/glibc-adventures-the-forgotten-chunks
http://linux.vbird.org/linux_basic/0410accountmanager.php#acl_talk_what
https://sploitfun.wordpress.com/2015/06/07/off-by-one-vulnerability-stack-based-2/

点击收藏 | 0 关注 | 1
上一篇：Hack With Rewrite 下一篇：【老文】渗透Facebook 的思...
1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

目录