Wissen gevoelige data en compiler optimalisering

Tim Kuijsten - ARPA2 All Hands

25 juni 2019 - Utrecht

Wissen gevoelige data

- Hygiene, o.a. voor mitigation van eventuele toekomstige lekkages
  - bijv. Heartbleed (memory management)
  - Rowhammer (architectuur)
  - Spectre (architectuur)
- Bijv. een key na gebruik wissen op alle plaatsen waar deze voorkomt
  - Heap
  - Stack
  - Registers
- O.a. in dermem.c van Kip wordt geheugen met gevoelige data gegarandeerd gewist na gebruik

memset, dead store

- Functie voor het genereren van nieuwe random keys
- Wissen van tijdelijke lokale stack variabele na gebruik:

```
#define KEYSIZE 32

void
genkey(unsigned char *key)
{
        unsigned char k[KEYSIZE];

        arc4random_buf(k, KEYSIZE);
        memcpy(key, k, KEYSIZE);
        memset(k, 0, KEYSIZE);
}
```

Memset met O0

- Assembly behorend bij *genkey* **cc -O0**

```c
#define KEYSIZE 32

void
genkey(unsigned char *key)
{
        unsigned char k[KEYSIZE];

        arc4random_buf(k, KEYSIZE);
        memcpy(key, k, KEYSIZE);
        memset(k, 0, KEYSIZE);
}
```

```
push   %rbp
mov    %rsp,%rbp
push   %r11
sub    $0x28,%rsp
lea    0xffffffffffffffd0(%rbp),%rax
mov    %rdi,0xfffffffffffffff0(%rbp)
mov    $0x20,%ecx
mov    %ecx,%esi
mov    %rax,%rdi
callq  1530 <arc4random_buf@plt>
lea    0xffffffffffffffd0(%rbp),%rax
mov    0xfffffffffffffff0(%rbp),%rsi
mov    0xffffffffffffffd0(%rbp),%rdi
mov    %rdi,(%rsi)
mov    0xffffffffffffffd8(%rbp),%rdi
mov    %rdi,0x8(%rsi)
mov    0xffffffffffffffe0(%rbp),%rdi
mov    %rdi,0x10(%rsi)
mov    0xffffffffffffffe8(%rbp),%rdi
mov    %rdi,0x18(%rsi)
xorps  %xmm0,%xmm0
movaps %xmm0,0x10(%rax)
movaps %xmm0,(%rax)
add    $0x28,%rsp
pop    %r11
pop    %rbp
```

Memset met O1

- Assembly behorend bij *genkey* **cc -O1**

```c
#define KEYSIZE 32

void
genkey(unsigned char *key)
{
        unsigned char k[KEYSIZE];

        arc4random_buf(k, KEYSIZE);
        memcpy(key, k, KEYSIZE);
        memset(k, 0, KEYSIZE);
}
```

```
push   %rbp
mov    %rsp,%rbp
push   %r11
push   %r14
sub    $0x20,%rsp
mov    %rdi,%r14
lea    0xffffffffffffffd0(%rbp),%rdi
mov    $0x20,%esi
callq  14d0 <arc4random_buf@plt>
movaps 0xffffffffffffffd0(%rbp),%xmm0
movaps 0xffffffffffffffe0(%rbp),%xmm1
movups %xmm1,0x10(%r14)
movups %xmm0,(%r14)
add    $0x20,%rsp
pop    %r14
pop    %r11
pop    %rbp
```

Oplossingen

- Verschillende oplossingen mogelijk, maar geen een die portable is:
  - Gebruik van explicit_bzero in glibc en op *BSD maar niet macOS of Windows
  - Gebruik van -fno-builtin-memset, werkt niet met oudere llvm
  - memset_s sinds C11, maar optioneel en nog door weinig compilers ondersteund (wel macOS)
  - Volatile pointers, maar werkt schijnbaar niet altijd

Mitigations

- Libsodium combineert van alles
- SecureZeroMemory op Windows
- memset_s uit C11 Annex K
- explicit_bzero
  - FreeBSD 11
  - OpenBSD 5.5
  - glibc 2.25
- explicit_memset in NetBSD
- Weak symbols
- Volatile pointer

```c
void
sodium_memzero(void * const pnt, const size_t len)
{
#ifdef _WIN32
    SecureZeroMemory(pnt, len);
#elif defined(HAVE_MEMSET_S)
    if (len > 0U && memset_s(pnt, (rsize_t) len, 0, (rsize_t) len) != 0) {
        sodium_misuse(); /* LCOV_EXCL_LINE */
    }
#elif defined(HAVE_EXPLICIT_BZERO)
    explicit_bzero(pnt, len);
#elif defined(HAVE_EXPLICIT_MEMSET)
    explicit_memset(pnt, 0, len);
#elif HAVE_WEAK_SYMBOLS
    if (len > 0U) {
        memset(pnt, 0, len);
        _sodium_dummy_symbol_to_prevent_memzero_lto(pnt, len);
    }
# ifdef HAVE_INLINE_ASM
    __asm__ __volatile__ ("" : : "r"(pnt) : "memory");
# endif
#else
    volatile unsigned char *volatile pnt_ =
        (volatile unsigned char *volatile) pnt;
    size_t i = (size_t) 0U;

    while (i < len) {
        pnt_[i++] = 0U;
    }
#endif
}
```

Referenties

- Memsad - why clearing memory is hard
  https://media.ccc.de/v/35c3-9788-memsad
- void sodium_memzero(void * const pnt, const size_t len):
  https://github.com/jedisct1/libsodium/blob/master/src/libsodium/sodium/utils.c#L112-L142
- Zero'ing memory, compiler optimizations and memset_s
  https://www.cryptologie.net/article/419/zeroing-memory-compiler-optimizations-and-memset_s/
- Zeroing buffers is insufficient
  http://www.daemonology.net/blog/2014-09-06-zeroing-buffers-is-insufficient.html

EOF