Software Security 1

Administrative

Kevin Borgolte kevin.borgolte@rub.de

Tentative Lecture Schedule / Deadlines

Wednesday 10-12

1. Oct 9 – Lecture
2. Oct 16 – Recitation
3. Oct 23 – Lecture ← First assignment due
4. Oct 30 – Recitation
5. Nov 6 – Lecture ← Second assignment due
6. Nov 13 – Recitation
7. Nov 20 – Lecture ← Third assignment due
8. Nov 27 – Recitation
9. Dec 4 – Lecture ← Fourth assignment due
10. Dec 11 – Recitation
11. Dec 18 – Lecture ← Fifth assignment due
12. **Jan 8 – Lecture**
13. Jan 15 – Recitation ← Sixth assignment due
14. Jan 22 – Exam Q&A
15. Jan 29 – Exam Setup Test ← Seventh assignment due

Assignments

- Assignment 6 (due Jan 16th, 0:00 Bochum time)
  - 4 tasks, focusing on C++ and <u>race conditions</u>
  - Due: January 16th, 0:00 Bochum time
- Assignment 7 (due Jan 30th, 0:00 Bochum time)
  - N tasks that will (try to) scope to be exam-sized
    - Please keep track of the time for them, we will be polling for it
- Exam
  - Please do not forget to register on Flexnow, <u>the deadline is 12.01.2025</u>
  - Exam room test set up: Wed Jan 29th, 10–12 (recitation slot)

Software Security 1

Race Conditions

Kevin Borgolte kevin.borgolte@rub.de

Program Execution

```
initialize()
check_input()
do_action()
check_input()
do_action()
check_input()
do_action()
terminate()
```

Program Execution

Process 1	Process 2	Process 3
`initialize()`	`initialize()`	`initialize()`
`check_input()`	`check_input()`	`check_input()`
`do_action()`	`do_action()`	`do_action()`
`check_input()`	`check_input()`	`check_input()`
`do_action()`	`do_action()`	`do_action()`
`check_input()`	`check_input()`	`check_input()`
`do_action()`	`do_action()`	`do_action()`
`terminate()`	`terminate()`	`terminate()`

Program Execution

```
P1 initialize()
P2 initialize()
P1 check_input()
P1 do_action()
P2 check_input()
P1 check_input()
P1 do_action()
P3 initialize()
P3 check_input()
P3 do_action()
P2 do_action()
P1 terminate()
```

- No true parallelization
  - Some partial serialization of parallel/concurrent events necessary
- Fundamental reason: Bottlenecks
  - More threads/processes than execution units (physical CPU cores)
  - Limited channel to memory/disk/network (must wait for input/output, kernel pauses one process and schedules another)

Order of Execution

- Without dependencies (implicit or explicit), the order of execution is only guaranteed within a single thread (and not even within a single process)

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
```

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P2 initialize()
P1 initialize()
P1 check_input()
P2 check_input()
P2 do_action()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

are valid orders of executions for P1 and P2

TOCTOU: Time of Check - Time of Use

- Relying on an order of execution that is not actually guaranteed is **dangerous:** Time of Check - Time of Use

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
```

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P2 initialize()
P1 initialize()
P1 check_input()
P2 check_input()
P2 do_action()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

TOCTOU: Time of Check - Time of Use

- Relying on an order of execution that is not actually guaranteed is **dangerous:** Time of Check - Time of Use

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
```

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P2 initialize()
P1 initialize()
P1 check_input()
P2 check_input()
P2 do_action()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

P1 might modify the input that P2 has checked

TOCTOU: Time of Check - Time of Use

- Relying on an order of execution that is not actually guaranteed is **dangerous:** Time of Check - Time of Use

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P1 initialize()
P1 check_input()
P2 initialize()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
```

```
P1 initialize()
P2 initialize()
P1 check_input()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P1 terminate()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 check_input()
P2 do_action()
P2 terminate()
```

```
P2 initialize()
P1 initialize()
P1 check_input()
P2 check_input()
P2 do_action()
P2 check_input()
P1 do_action()
P1 check_input()
P1 do_action()
P2 do_action()
P1 check_input()
P2 check_input()
P1 do_action()
P2 do_action()
P1 terminate()
P2 terminate()
```

P1 might modify the input that P2 has checked

Misusing such concurrency errors means <u>racing</u> to affect the program's state at a <u>weak point</u>, thus <u>race condition.</u>

Misusing Race Conditions

- Misusing a race condition means changing the state of a program at a point in the program where the program assumes the state has not changed
- This includes especially
  - Filesystem
  - Memory

```
initialize()
check_input()
WEAK POINT
do_action()
terminate()
```

Filesystem Races

```
1.  int main(int argc, char *argv[]) {
2.    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0755);
3.    write(fd, "#!/bin/sh\necho SAFE\n", 20);
4.    close(fd);
5.    execl("/bin/sh", "/bin/sh", argv[1], NULL);
6.  }
```

Any idea where the issue lies?

Filesystem Races

```
1.  int main(int argc, char *argv[]) {
2.    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0755);
3.    write(fd, "#!/bin/sh\necho SAFE\n", 20);
4.    close(fd);
5.    execl("/bin/sh", "/bin/sh", argv[1], NULL);
6.  }
```

Any idea where the issue lies?

The file we write to is not guaranteed to be the same file that we execute

Filesystem Races

```
1.  int main(int argc, char *argv[]) {
2.    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0755);
3.    write(fd, "#!/bin/sh\necho SAFE\n", 20);
4.    close(fd);
5.    execl("/bin/sh", "/bin/sh", argv[1], NULL);
6.  }
```

- Potential problem: The time window we have for misusing the race condition might be very small and make it impractical
- General solution: Slow the victim process down (a lot)

Slowing Down Processes

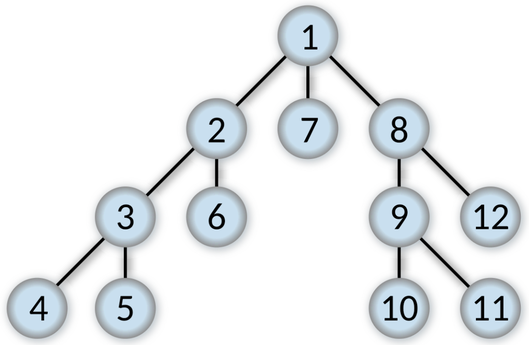
- Using kernel scheduler functionality
  - nice to adjust priority for CPU time of a process
  - ionice to adjust input/output priority for a process
  - Might not always be possible (no control over process etc.)

Slowing Down Processes

- Using kernel scheduler functionality
  - nice to adjust priority for CPU time of a process
  - ionice to adjust input/output priority for a process
  - Might not always be possible (no control over process etc.)
- Triggering slow behavior indirectly
  - Operations are usually not constant time
    - Filesystem access for ./foobar is much faster than /a/b/.../zzz/foobar
    - Values close to maximum allowed limits is often sufficient, e.g., Linux path length is 4096 bytes, and can additionally contain up to 40 symbolic links
  - Algorithmic complexity can help, but is rarely necessary

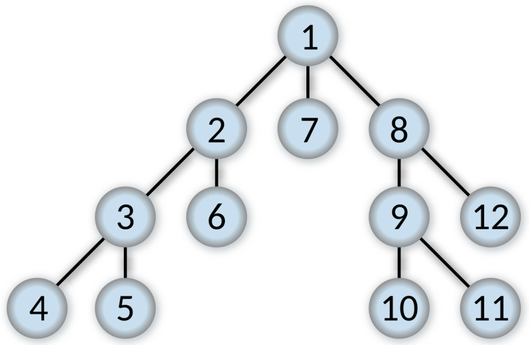
Algorithmic Complexity

- Abusing algorithmic complexity means leveraging the worst case complexity of an algorithm for one's purpose
- Algorithmic cost depends on input and algorithm
  - For example, complexity of finding node 8 is lower with breadth-first (4) search than with depth-first search (8) if we always expand the left most graph node



Algorithmic Complexity

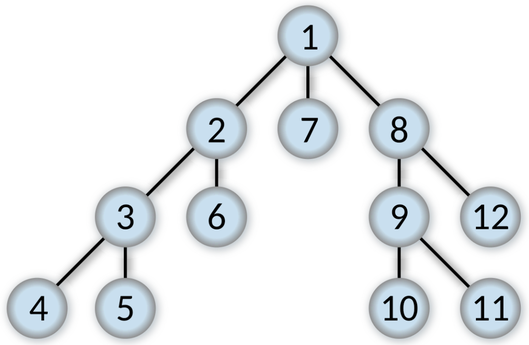
- Abusing algorithmic complexity means leveraging the worst case complexity of an algorithm for one's purpose
- Algorithmic cost depends on input and algorithm
  - For example, complexity of finding node 8 is lower with breadth-first (4) search than with depth-first search (8) if we always expand the left most graph node



- Degenerate inputs can take a much longer time than the average case, carefully crafting them is the idea
  - For example, quick sort: average case $O(n \log n)$, but worst case $O(n^2)$

Algorithmic Complexity

- Abusing algorithmic complexity means leveraging the worst case complexity of an algorithm for one's purpose
- Algorithmic cost depends on input and algorithm
  - For example, complexity of finding node 8 is lower with breadth-first (4) search than with depth-first search (8) if we always expand the left most graph node



- Degenerate inputs can take a much longer time than the average case, carefully crafting them is the idea
  - For example, quick sort: average case $O(n \log n)$, but worst case $O(n^2)$
- AC affects time and space/memory, and they can cause DoS

Filesystem TOCTOU in Practice: CVE-2019-7307

```diff
=== modified file 'apport/report.py'
--- apport/report.py    2019-05-16 18:48:29 +0000
+++ apport/report.py    2019-07-09 02:13:39 +0000
@@ -978,15 +978,25 @@
         ifpath = os.path.expanduser(_ignore_file)
         if orig_home is not None:
             os.environ['HOME'] = orig_home
-        if not os.access(ifpath, os.R_OK) or os.path.getsize(ifpath) == 0:
+        euid = os.geteuid()
+        try:
+            # drop permissions temporarily to try open users ignore file
+            os.seteuid(os.getuid())
+            fp = open(ifpath, 'r')
+        except (IOError, OSError):
+            fp = None
+        finally:
+            os.seteuid(euid)
+        if fp is None or os.fstat(fp.fileno()).st_size == 0:
             # create a document from scratch
             dom = xml.dom.getDOMImplementation().createDocument(None, 'apport', None)
         else:
             try:
-                dom = xml.dom.minidom.parse(ifpath)
+                dom = xml.dom.minidom.parse(fp)
             except ExpatError as e:
                 raise ValueError('%s has invalid format: %s' % (_ignore_file, str(e)))
-
+        if fp is not None:
+            fp.close()
         # remove whitespace so that writing back the XML does not accumulate
         # whitespace
         dom.documentElement.normalize()
```

Apport before versions 2.14.1-0ubuntu3.29+esm1, 2.20.1-0ubuntu2.19, 2.20.9-0ubuntu7.7, 2.20.10-0ubuntu27.1, 2.20.11-0ubuntu5 contained a TOCTTOU vulnerability when reading the users ~/.apport-ignore.xml file, which allows a local attacker to replace this file with a symlink to any other file on the system and so cause Apport to include the contents of this other file in the resulting crash report. The crash report could then be read by that user either by causing it to be uploaded and reported to Launchpad, or by leveraging some other vulnerability to read the resulting crash report, **and so allow the user to read arbitrary files on the system.**

Processes and Threads

- We mostly looked at processes so far, which are isolated
  - Own virtual memory
  - Own registers
  - Own file descriptors
  - Own security properties (uid/gid/seccomp, etc.)
- Processes can also have multiple threads
  - Always one thread, the main thread of a process
  - Shared virtual memory (but has its own stack)
  - Shared file descriptors
- On Linux, usually pthreads
  - What is shared and separate depends on OS and thread implementation (just like for DMAs)

Threads and Memory

- Threads share virtual memory
  - Opportunity for race conditions!

```c
unsigned int size = 42;

void read_data() {
  char buffer[16];
  if(size < 16) {
    printf("Valid size! Enter payload up to %d bytes.\n", size);
    printf("Read %d bytes!\n", read(0, buffer, size));
  } else {
    printf("Invalid size %d!\n", size);
  }
}

void *thread_allocator(int arg) {
  while(1) {
    read_data();
  }
}

main() {
  pthread_t allocator;
  pthread_create(&allocator, NULL, thread_allocator, 0);
  while(size != 0) {
    read(0, &size, 1);
  }
  exit(0);
}
```

Threads and Memory

- Threads share virtual memory
  - Opportunity for race conditions!

TOCTOU: main() might change size between if and read()

```
1.  unsigned int size = 42;

3.  void read_data() {
4.    char buffer[16];
5.    if(size < 16) {
6.      printf("Valid size! Enter payload up to %d bytes.\n", size);
7.      printf("Read %d bytes!\n", read(0, buffer, size));
8.    } else {
9.      printf("Invalid size %d!\n", size);
10.   }
11. }
12.
13. void *thread_allocator(int arg) {
14.   while(1) {
15.     read_data();
16.   }
17. }
18.
19. main() {
20.   pthread_t allocator;
21.   pthread_create(&allocator, NULL, thread_allocator, 0);
22.   while(size != 0) {
23.     read(0, &size, 1);
24.   }
25.   exit(0);
26. }
```

Double Fetch

```
1.  int check_safety(char *user_buffer, int maximum_size) {
2.    int size;
3.    copy_from_user(&size, user_buffer, sizeof(size));
4.    return size <= maximum_size;
5.  }
6.
7.  static long device_ioctl(struct file *file, unsigned int cmd, unsigned long user_buffer) {
8.    int size;
9.    char buffer[16];
10.   if (!check_safety(user_buffer, 16)) {
11.     return;
12.   }
13.   copy_from_user(&size, user_buffer, sizeof(size));
14.   copy_from_user(buffer, user_buffer + sizeof(size), size);
15. }
```

Double Fetch

```
1.  int check_safety(char *user_buffer, int maximum_size) {
2.    int size;
3.    copy_from_user(&size, user_buffer, sizeof(size));
4.    return size <= maximum_size;
5.  }
6.
7.  static long device_ioctl(struct file *file, unsigned int cmd, unsigned long user_buffer) {
8.    int size;
9.    char buffer[16];
10.   if (!check_safety(user_buffer, 16)) {
11.     return;
12.   }
13.   copy_from_user(&size, user_buffer, sizeof(size));
14.   copy_from_user(buffer, user_buffer + sizeof(size), size);
15. }
```

Sneaky Double Fetch

```
1.  NTSTATUS
2.  PspBuildCreateProcessContext(
3.      _In_ PS_ATTRIBUTE_LIST *attr_list,
4.      _In_ KPROCESSOR_MODE previous_mode,
5.      _In_ UINT64 unknown,
6.      _Out_ VOID *process_context)
7.  {
8.    PS_ATTRIBUTE *attr = &attr_list->Attributes;
9.    PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
10.   PS_MITIGATION_AUDIT_OPTIONS_MAP options;
11.   // ...
12.
13.   try {
14.     // ...
15.   switch (attr->Attribute) {
16.     case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
17.       if (attr->Size > sizeof(options)) {
18.         goto ERROR;
19.       }
20.       memset(&options, 0, sizeof(options));
21.       options_ptr = attr->ValuePtr;
22.       if (previous_mode != KernelMode) {
23.         ProbeForRead(options_ptr, sizeof(options), 0);
24.         memmove(&options, options_ptr, attr->Size);
25.       }
26.       // ...
27.     }
28.   }
29.   // ...
30. }
```

Windows 11 23H2 (no double fetch)

```
1.  NTSTATUS
2.  PspBuildCreateProcessContext(
3.      _In_ PS_ATTRIBUTE_LIST *attr_list,
4.      _In_ KPROCESSOR_MODE previous_mode,
5.      _In_ UINT64 unknown,
6.      _Out_ VOID *process_context)
7.  {
8.    PS_ATTRIBUTE *attr = &attr_list->Attributes;
9.    PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
10.   PS_MITIGATION_AUDIT_OPTIONS_MAP options;
11.   // ...
12.
13.   try {
14.     // ...
15.   switch (attr->Attribute) {
16.     case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
17.       if (attr->Size > sizeof(options)) {
18.         goto ERROR;
19.       }
20.       memset(&options, 0, sizeof(options));
21.       options_ptr = attr->ValuePtr;
22.       if (previous_mode != KernelMode) {
23.         ProbeForRead(options_ptr, sizeof(options), 0);
24.         RtlCopyVolatileMemory(&options, options_ptr, attr->Size);
25.       }
26.       // ...
27.     }
28.   }
29.   // ...
30. }
```

Windows 11 24H2 (double fetch, CVE-2024-26218)

Sneaky Double Fetch

```
1.  NTSTATUS
2.  PspBuildCreateProcessContext(
3.      _In_ PS_ATTRIBUTE_LIST *attr_list,
4.      _In_ KPROCESSOR_MODE previous_mode,
5.      _In_ UINT64 unknown,
6.      _Out_ VOID *process_context)
7.  {
8.    PS_ATTRIBUTE *attr = &attr_list->Attributes;
9.    PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
10.   PS_MITIGATION_AUDIT_OPTIONS_MAP options;
11.   // ...
12.
13.   try {
14.     // ...
15.   switch (attr->Attribute) {
16.     case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
17.       if (attr->Size > sizeof(options)) {
18.         goto ERROR;
19.       }
20.       memset(&options, 0, sizeof(options));
21.       options_ptr = attr->ValuePtr;
22.       if (previous_mode != KernelMode) {
23.         ProbeForRead(options_ptr, sizeof(options), 0);
24.         memmove(&options, options_ptr, attr->Size);
25.       }
26.       // ...
27.     }
28.   }
29.   // ...
30. }
```

Windows 11 23H2 (no double fetch)

```
1.  NTSTATUS
2.  PspBuildCreateProcessContext(
3.      _In_ PS_ATTRIBUTE_LIST *attr_list,
4.      _In_ KPROCESSOR_MODE previous_mode,
5.      _In_ UINT64 unknown,
6.      _Out_ VOID *process_context)
7.  {
8.    PS_ATTRIBUTE *attr = &attr_list->Attributes;
9.    PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
10.   PS_MITIGATION_AUDIT_OPTIONS_MAP options;
11.   // ...
12.
13.   try {
14.     // ...
15.   switch (attr->Attribute) {
16.     case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
17.       if (attr->Size > sizeof(options)) {
18.         goto ERROR;
19.       }
20.       memset(&options, 0, sizeof(options));
21.       options_ptr = attr->ValuePtr;
22.       if (previous_mode != KernelMode) {
23.         ProbeForRead(options_ptr, sizeof(options), 0);
24.         RtlCopyVolatileMemory(&options, options_ptr, attr->Size);
25.       }
26.       // ...
27.     }
28.   }
29.   // ...
30. }
```

Windows 11 24H2 (double fetch, CVE-2024-26218)

Sneaky Double Fetch

```
NTSTATUS
PspBuildCreateProcessContext(
    _In_ PS_ATTRIBUTE_LIST *attr_list,
    _In_ KPROCESSOR_MODE previous_mode,
    _In_ UINT64 unknown,
    _Out_ VOID *process_context)
{
  PS_ATTRIBUTE *attr = &attr_list->Attributes;
  PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
  PS_MITIGATION_AUDIT_OPTIONS_MAP options;
  // ...

  try {
    // ...
  switch (attr->Attribute) {
    case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
      if (attr->Size > sizeof(options)) {
        goto ERROR;
      }
      memset(&options, 0, sizeof(options));
      options_ptr = attr->ValuePtr;
      if (previous_mode != KernelMode) {
        ProbeForRead(options_ptr, sizeof(options), 0);
        memmove(&options, options_ptr, attr->Size);
      }
      // ...
    }
  }
  // ...
}
```

- Hidden behind `attr`!

```
mov     r8, [rdi+8]                 ; Size
cmp     r8, 18h
ja      ERROR
xorps   xmm0, xmm0
xor     eax, eax
movups  [rsp+278h+var_70], xmm0
mov     [rsp+278h+var_60], rax0
mov     rdx, [rdi+10h]              ; Src
lea     rcx, [rsp+278h+var_70]      ; void *
test    r10b, r10b
jz      short prev_mode_kernel
cmp     rdx, r9
cmovnb  rdx, r9
call    memmove
```

Windows 11 23H2 (no double fetch)

Sneaky Double Fetch

```
NTSTATUS
PspBuildCreateProcessContext(
    _In_ PS_ATTRIBUTE_LIST *attr_list,
    _In_ KPROCESSOR_MODE previous_mode,
    _In_ UINT64 unknown,
    _Out_ VOID *process_context)
{
  PS_ATTRIBUTE *attr = &attr_list->Attributes;
  PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
  PS_MITIGATION_AUDIT_OPTIONS_MAP options;
  // ...

  try {
    // ...
  switch (attr->Attribute) {
    case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
      if (attr->Size < sizeof(options)) {
        goto ERROR;
      }
      memset(&options, 0, sizeof(options));
      options_ptr = attr->ValuePtr;
      if (previous_mode != KernelMode) {
        ProbeForRead(options_ptr, sizeof(options), 0);
        memmove(&options, options_ptr, attr->Size);
      }
      // ...
    }
  }
  // ...
}
```

- Hidden behind `attr`!

```
mov     r8, [rdi+8]                 ; Size
cmp     r8, 18h
ja      ERROR
xorps   xmm0, xmm0
xor     eax, eax
movups  [rsp+278h+var_70], xmm0
mov     [rsp+278h+var_60], rax0
mov     rdx, [rdi+10h]              ; Src
lea     rcx, [rsp+278h+var_70]      ; void *
test    r10b, r10b
jz      short prev_mode_kernel
cmp     rdx, r9
cmovnb  rdx, r9
call    memmove
```

Windows 11 23H2 (no double fetch)

Sneaky Double Fetch

```c
NTSTATUS
PspBuildCreateProcessContext(
    _In_ PS_ATTRIBUTE_LIST *attr_list,
    _In_ KPROCESSOR_MODE previous_mode,
    _In_ UINT64 unknown,
    _Out_ VOID *process_context)
{
  PS_ATTRIBUTE *attr = &attr_list->Attributes;
  PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
  PS_MITIGATION_AUDIT_OPTIONS_MAP options;
  // ...

  try {
    // ...
  switch (attr->Attribute) {
    case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
      if (attr->Size < sizeof(options)) {
        goto ERROR;
      }
      memset(&options, 0, sizeof(options));
      options_ptr = attr->ValuePtr;
      if (previous_mode != KernelMode) {
        ProbeForRead(options_ptr, sizeof(options), 0);
        memmove(&options, options_ptr, attr->Size);
      }
      // ...
    }
  }
  // ...
}
```

- Hidden behind `attr`!

```asm
mov     r8, [rdi+8]                 ; Size
cmp     r8, 18h
ja      ERROR
xorps   xmm0, xmm0
xor     eax, eax
movups  [rsp+278h+var_70], xmm0
mov     [rsp+278h+var_60], rax0
mov     rdx, [rdi+10h]              ; Src
lea     rcx, [rsp+278h+var_70]      ; void *
test    r10b, r10b
jz      short prev_mode_kernel
cmp     rdx, r9
cmovnb  rdx, r9
call    memmove
```

Windows 11 23H2 (no double fetch)

Sneaky Double Fetch

```
NTSTATUS
PspBuildCreateProcessContext(
    _In_ PS_ATTRIBUTE_LIST *attr_list,
    _In_ KPROCESSOR_MODE previous_mode,
    _In_ UINT64 unknown,
    _Out_ VOID *process_context)
{
  PS_ATTRIBUTE *attr = &attr_list->Attributes;
  PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
  PS_MITIGATION_AUDIT_OPTIONS_MAP options;
  // ...

  try {
    // ...
  switch (attr->Attribute) {
    case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
      if (attr->Size > sizeof(options)) {
        goto ERROR;
      }
      memset(&options, 0, sizeof(options));
      options_ptr = attr->ValuePtr;
      if (previous_mode != KernelMode) {
        ProbeForRead(options_ptr, sizeof(options), 0);
        RtlCopyVolatileMemory(&options, options_ptr, attr->Size);
      }
      // ...
    }
  }
  // ...
}
```

- Hidden behind `attr`!

```
cmp     qword ptr [rdi+8], 18h
ja      ERROR
xorps   xmm0, xmm0
xor     eax, eax
movups  [rsp+278h+var_70], xmm0
mov     [rsp+278h+var_60], rax0
mov     rdx, [rdi+10h]            ; Src
lea     rcx, [rsp+278h+var_70]    ; void *
test    r10b, r10b
mov     r8, [rdi+8]               ; Size
jz      short prev_mode_kernel
cmp     rdx, r13
cmovnb  rdx, r13
call    RtlCopyVolatileMemory
```

Windows 11 24H2 (double fetch, CVE-2024-26218)

Sneaky Double Fetch

```c
NTSTATUS
PspBuildCreateProcessContext(
    _In_ PS_ATTRIBUTE_LIST *attr_list,
    _In_ KPROCESSOR_MODE previous_mode,
    _In_ UINT64 unknown,
    _Out_ VOID *process_context)
{
  PS_ATTRIBUTE *attr = &attr_list->Attributes;
  PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
  PS_MITIGATION_AUDIT_OPTIONS_MAP options;
  // ...

  try {
    // ...
  switch (attr->Attribute) {
    case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
      if (attr->Size > sizeof(options)) {
        goto ERROR;
      }
      memset(&options, 0, sizeof(options));
      options_ptr = attr->ValuePtr;
      if (previous_mode != KernelMode) {
        ProbeForRead(options_ptr, sizeof(options), 0);
        RtlCopyVolatileMemory(&options, options_ptr, attr->Size);
      }
      // ...
    }
  }
  // ...
}
```

- Hidden behind `attr`!

```asm
cmp     qword ptr [rdi+8], 18h
ja      ERROR
xorps   xmm0, xmm0
xor     eax, eax
movups  [rsp+278h+var_70], xmm0
mov     [rsp+278h+var_60], rax0
mov     rdx, [rdi+10h]              ; Src
lea     rcx, [rsp+278h+var_70]      ; void *
test    r10b, r10b
mov     r8, [rdi+8]                 ; Size
jz      short prev_mode_kernel
cmp     rdx, r13
cmovnb  rdx, r13
call    RtlCopyVolatileMemory
```

Windows 11 24H2 (double fetch, CVE-2024-26218)

Sneaky Double Fetch

```
NTSTATUS
PspBuildCreateProcessContext(
    _In_ PS_ATTRIBUTE_LIST *attr_list,
    _In_ KPROCESSOR_MODE previous_mode,
    _In_ UINT64 unknown,
    _Out_ VOID *process_context)
{
  PS_ATTRIBUTE *attr = &attr_list->Attributes;
  PS_MITIGATION_AUDIT_OPTIONS_MAP *options_ptr;
  PS_MITIGATION_AUDIT_OPTIONS_MAP options;
  // ...

  try {
    // ...
  switch (attr->Attribute) {
    case PS_ATTRIBUTE_MITIGATION_AUDIT_OPTIONS:
      if (attr->Size > sizeof(options)) {
        goto ERROR;
      }
      memset(&options, 0, sizeof(options));
      options_ptr = attr->ValuePtr;
      if (previous_mode != KernelMode) {
        ProbeForRead(options_ptr, sizeof(options), 0);
        RtlCopyVolatileMemory(&options, options_ptr, attr->Size);
      }
      // ...
    }
  }
  // ...
}
```

- Hidden behind `attr`!

```
cmp     qword ptr [rdi+8], 18h
ja      ERROR
xorps   xmm0, xmm0
xor     eax, eax
movups  [rsp+278h+var_70], xmm0
mov     [rsp+278h+var_60], rax0
mov     rdx, [rdi+10h]              ; Src
lea     rcx, [rsp+278h+var_70]      ; void *
test    r10b, r10b
mov     r8, [rdi+8]                 ; Size
jz      short prev_mode_kernel
cmp     rdx, r13
cmovnb  rdx, r13
call    RtlCopyVolatileMemory
```

Windows 11 24H2 (double fetch, CVE-2024-26218)

Sneaky Double Fetch

- Why does the compiler changes the access to `attr->Size`?
  - Broad changes in Windows 11 24H2 in treating user memory volatile
    - See also https://github.com/microsoft/xdp-for-windows/pull/188
  - `Size` was marked volatile
    - Thus, the compiler needs to fetch it again on acces
    - Before, the compiler optimized the double fetch away!
  - This change introduced new vulnerabilities into the Windows 11 kernel
    - CVE-2024-26218: Double Fetch to Stack Buffer Overflow
    - CVE-2024-21345: Double Fetch to Arbitrary Write

via https://exploits.forsale/24h2-nt-exploit/

Other Data Races

```
1.  unsigned int num = 0;

3.  void *thread_main(int arg) {
4.    while (1) {
5.      num++;
6.      num--;
7.      if (num != 0) {
8.        printf("NUM: %d\n", num);
9.      }
10.   }
11. }
12.
13. main() {
14.   pthread_t t1, t2;
15.   pthread_create(&t1, NULL, thread_main, 0);
16.   pthread_create(&t2, NULL, thread_main, 0);
17.   getchar();
18.   exit(0);
19. }
```

Other Data Races

```
1.  unsigned int num = 0;

3.  void *thread_main(int arg) {
4.    while (1) {
5.      num++;
6.      num--;
7.      if (num != 0) {
8.        printf("NUM: %d\n", num);
9.      }
10.   }
11. }
12.
13. main() {
14.   pthread_t t1, t2;
15.   pthread_create(&t1, NULL, thread_main, 0);
16.   pthread_create(&t2, NULL, thread_main, 0);
17.   getchar();
18.   exit(0);
19. }
```

```
num == 0
```

Other Data Races

```
1.  unsigned int num = 0;

3.  void *thread_main(int arg) {
4.    while (1) {
5.      num++;
6.      num--;
7.      if (num != 0) {
8.        printf("NUM: %d\n", num);
9.      }
10.   }
11. }
12.
13. main() {
14.   pthread_t t1, t2;
15.   pthread_create(&t1, NULL, thread_main, 0);
16.   pthread_create(&t2, NULL, thread_main, 0);
17.   getchar();
18.   exit(0);
19. }
```

```
num == 0

T1: mov rdi, [num]  ; 0
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
T2: mov rdi, [num]  ; 2
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
T2: mov rdi, [num]  ; 2
T1: dec rdi         ; 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
T2: mov rdi, [num]  ; 2
T1: dec rdi         ; 1
T2: dec rdi         ; 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
T2: mov rdi, [num]  ; 2
T1: dec rdi         ; 1
T2: dec rdi         ; 1
T1: mov [num], rdi  ; 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
T2: mov rdi, [num]  ; 2
T1: dec rdi         ; 1
T2: dec rdi         ; 1
T1: mov [num], rdi  ; 1
T2: mov [num], rdi  ; 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
T2: mov rdi, [num]  ; 2
T1: dec rdi         ; 1
T2: dec rdi         ; 1
T1: mov [num], rdi  ; 1
T2: mov [num], rdi  ; 1
num == 1
```

Other Data Races

```c
unsigned int num = 0;

void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

```
num == 0
T1: mov rdi, [num]  ; 0
T1: inc rdi         ; 1
T1: mov [num], rdi  ; 1
num == 1
T2: mov rdi, [num]  ; 1
T2: inc rdi         ; 2
T2: mov [num], rdi  ; 2
num == 2
T1: mov rdi, [num]  ; 2
T2: mov rdi, [num]  ; 2
T1: dec rdi         ; 1
T2: dec rdi         ; 1
T1: mov [num], rdi  ; 1
T2: mov [num], rdi  ; 1
num == 1
```

Data races can have odd effects!

Preventing Data Races and Race Conditions

- mutexes (mutual exclusion) (e.g., ptmalloc uses them)
  - Only one thread can access the section that they are guarding (critical section)
  - Other threads block and wait
    = Slows down program
  - Explicit serialization
- Immutability etc. do not usually help if shared state is needed

```
unsigned int num = 0;
void *thread_main(int arg) {
  while (1) {
    num++;
    num--;
    if (num != 0) {
      printf("NUM: %d\n", num);
    }
  }
}

main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, thread_main, 0);
  pthread_create(&t2, NULL, thread_main, 0);
  getchar();
  exit(0);
}
```

Detecting Data Races

- No silver bullet to detect data races, open research questions
  - We discuss some techniques to detect them in "Program Analysis"
- Enumerating all execution sequences practically infeasible
- Some tools to detect data races exist
  - helgrind, drd
  - Relevant code must be triggered = test coverage problem

Signals and Reentrancy

- Recall: Sigreturn-oriented programming
- Handling signals: What happens?
  - Signal pauses process and invoke the signal handler
    - This also works during critical sections defined by mutexes!
  - Side note: You can send any signal to any process that has the same real user id, even if the effective user id is root (setuid binaries)
  - Effectively, you can interrupt and stop any process and divert execution to the signal handler (which will return to the interrupted location)

Unexpected Order

```c
int x = 0;

void handler(int signum) {
  x = 0;
}

int main(void) {
  signal(SIGUSR1, handler);

  while (true) {
    if (x == 0) {
      x++;
    }
    x--;
    if (x != 0) {
      printf("%d\n", x);
    }
  }
}
```

- What can happen?

Unexpected Order

```c
int x = 0;

void handler(int signum) {
  x = 0;
}

int main(void) {
  signal(SIGUSR1, handler);

  while (true) {
    if (x == 0) {
      x++;
    }
    x--;
    if (x != 0) {
      printf("%d\n", x);
    }
  }
}
```

- What can happen?
  - If x is 0, then x++ sets it to 1, and x-- sets it back to 0

Unexpected Order

```c
int x = 0;

void handler(int signum) {
  x = 0;
}

int main(void) {
  signal(SIGUSR1, handler);

  while (true) {
    if (x == 0) {
      x++;
    }
    x--;
    if (x != 0) {
      printf("%d\n", x);
    }
  }
}
```

- What can happen?
  - If `x` is 0, then `x++` sets it to 1, and `x--` sets it back to 0
    - Effectively, this makes the condition `x != 0` unsatisfiable

Unexpected Order

```c
int x = 0;

void handler(int signum) {
  x = 0;
}

int main(void) {
  signal(SIGUSR1, handler);

  while (true) {
    if (x == 0) {
      x++;
    }
    x--;
    if (x != 0) {
      printf("%d\n", x);
    }
  }
}
```

- What can happen?
  - If x is 0, then x++ sets it to 1, and x-- sets it back to 0
    - Effectively, this makes the condition `x != 0` unsatisfiable
  - However, if we signal USR1 after the first if, but before the x--, then x will be 0 and decremented to -1, which leads to x being print and us entering a loop of x repeatedly being decremented

Reentrancy

```c
int x, y, tmp;

void swap(int *x, int *y)
{
  tmp = *x;
  *x = *y;
  *y = tmp;
}

void call_swap(int signum)
{
  swap(&x, &y);
}

int main(void)
{
  signal(SIGUSR1, call_swap);
  while (true) {
    x = 1, y = 2;
    call_swap(0);
    if (!(x == 2 && y == 1)) {
      printf("x=%d y=%d\n", x, y);
      exit(EXIT_FAILURE);
    }
  }
}
```

- A function is <u>reentrant</u> if it continues to operate properly even when interrupted with itself
- What happens if the program receives SIGUSR1 before `*y = tmp`?

Safe Signal Handling

- Do not call non-reentrant functions from signal handlers
  - Your handler might have interrupted them!
  - Another signal might interrupt your signal handler
  - Depending on how the signal handler is set up, the same signal might also interrupt the current handler

Safe Signal Handling

- Do not call non-reentrant functions from signal handlers
  - Your handler might have interrupted them!
  - Another signal might interrupt your signal handler
  - Depending on how the signal handler is set up, the same signal might also interrupt the current handler
- Many POSIX functions are not guaranteed to be signal safe
  - In particular, nonreentrant functions are not signal safe
  - malloc() is not async-signal-safe (often deadlocks)
  - See also man 7 signal-safety, https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html, https://www.gnu.org/software/libc/manual/html_node/Nonreentrancy.html