Where's Waldo...'s Brothers and Sisters?

Variant Analysis on Recent 0-days

Maddie Stone
@maddiestone
BlueHat IL 2020

Who am I? - Maddie Stone

- Security Researcher on Google Project Zero
  - Focusing on 0-days used in the wild
- Previously, Google's Android Sec team
- Reverse all the things
- Speaker at REcon, OffensiveCon, BlackHat, & more!
- BS in Computer Science, Russian, & Applied Math, MS in Computer Science

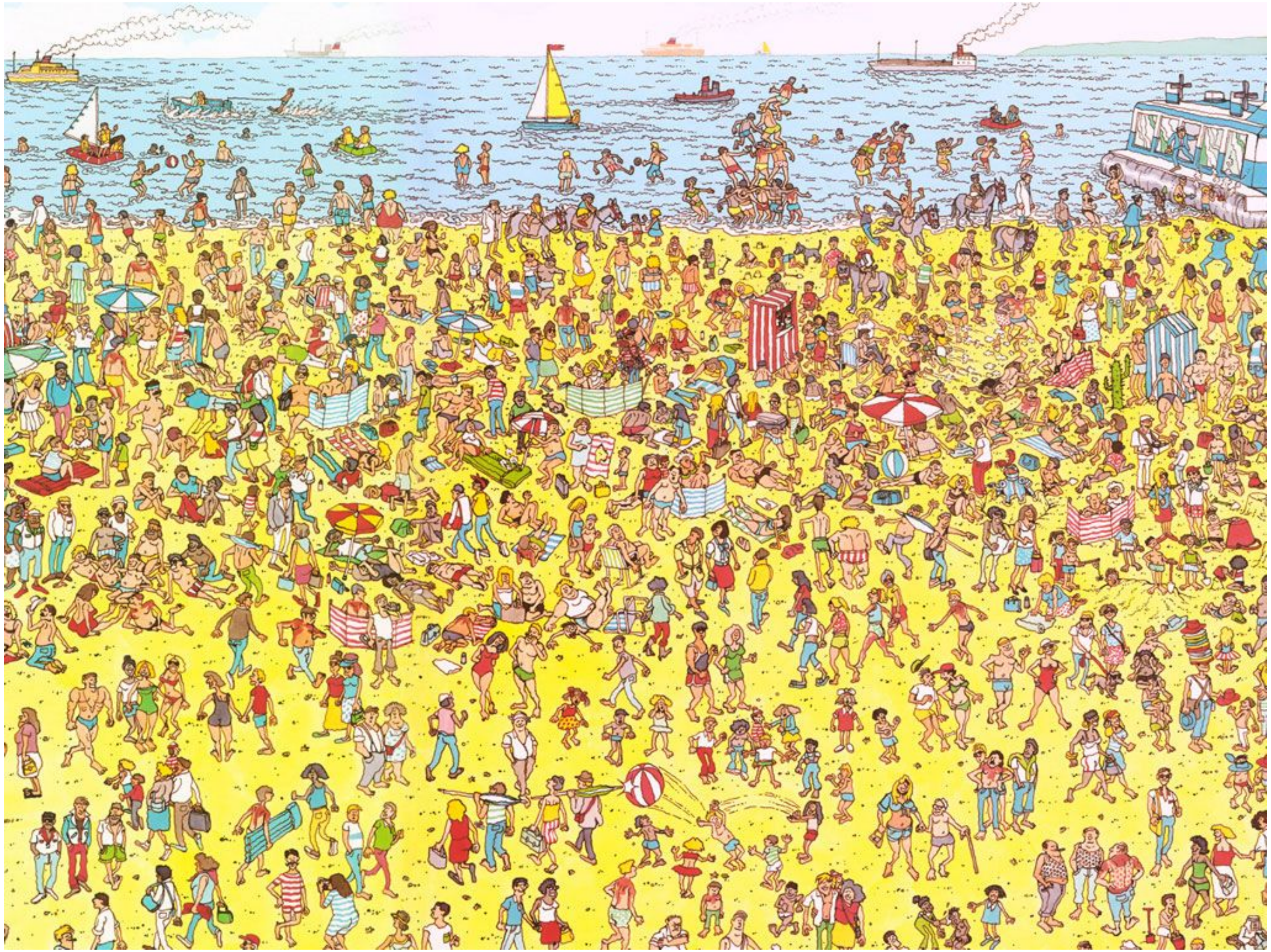
@maddiestone

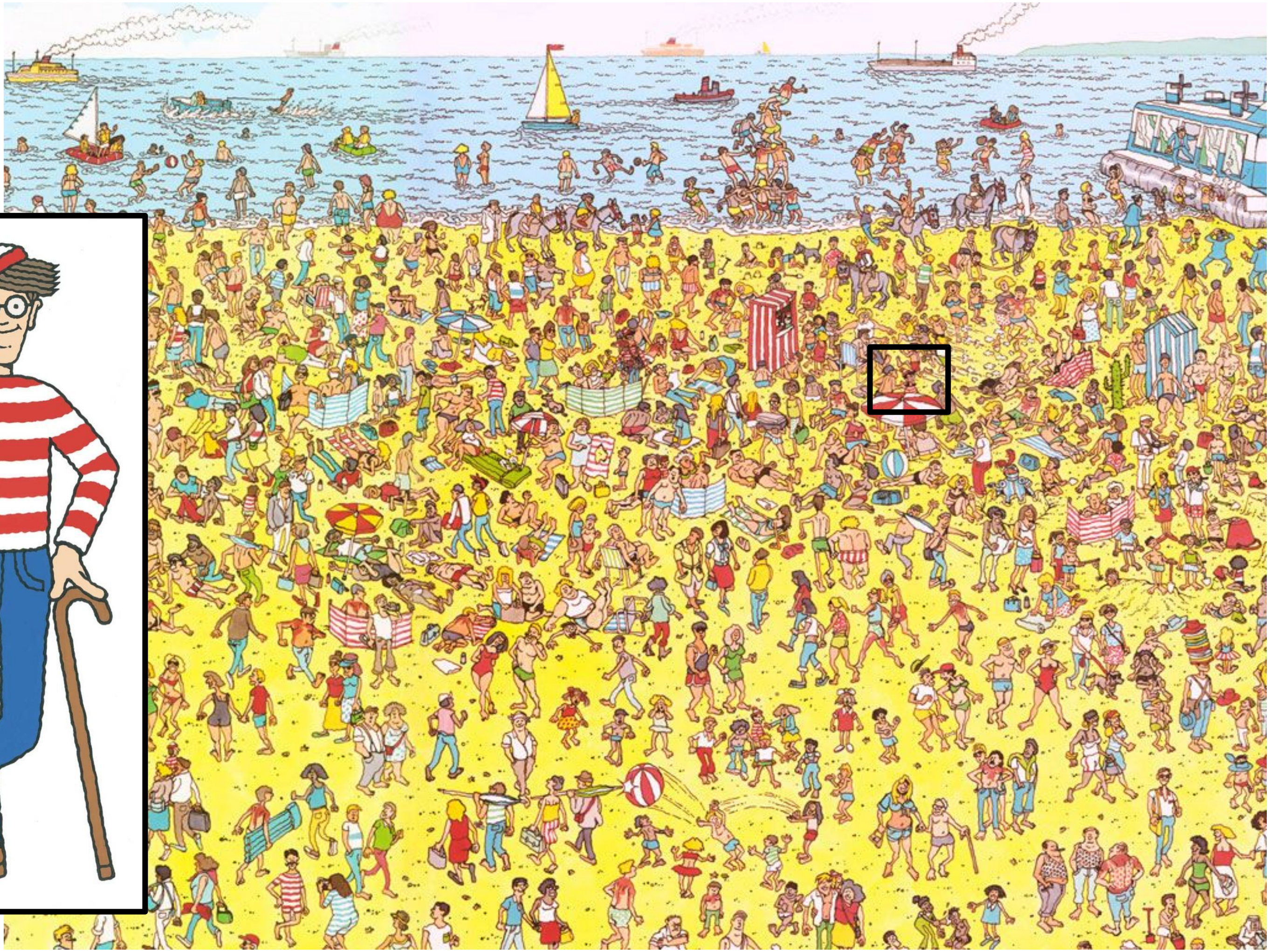
Google

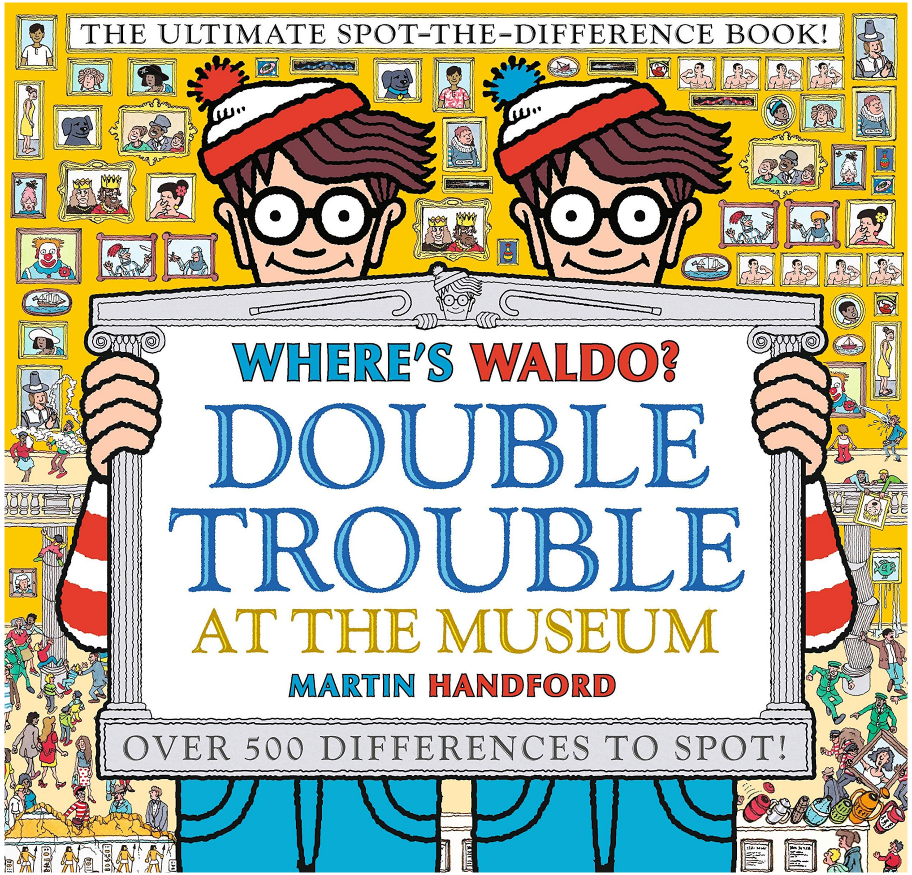
Agenda

- What is variant analysis?
- Why do we care about variant analysis?
- Case studies
    - Android binder UAF
    - Chrome webaudio UAF
    - Internet Explorer jscript
- Conclusion

What is variant analysis?




Google


THE ULTIMATE SPOT-THE-DIFFERENCE BOOK!
WHERE'S WALDO?
DOUBLE TROUBLE
AT THE MUSEUM
MARTIN HANDFORD
OVER 500 DIFFERENCES TO SPOT!

Google

The process of looking for additional vulnerabilities based on a single known vulnerability.

Why do we care about vulnerability analysis?

Why do we care about variant analysis?

- Playing “whack-a-mole” for vulnerabilities one-at-a-time is not an efficient nor sustainable strategy.

Why do we care about variant analysis?

- Playing “whack-a-mole” for vulnerabilities one-at-a-time is not an efficient nor sustainable strategy.
- Researchers often find more than one vulnerability at a time. Thus we can assume, attackers are too.

Make 0-day hard.

Case Study: Android Binder UAF

CVE-2019-2215

Android Binder UAF: About the Bug

- Use-after-free in the Android Binder driver due to poll handler using a wait queue that is not tied to the lifetime of the file.

Android Binder UAF: About the Bug

- Use-after-free in the Android Binder driver due to poll handler using a wait queue that is not tied to the lifetime of the file.
- Patched in upstream ~18 months prior.
  - Discovered by syzcaller in Nov 2017

Variant Analysis Approach

1) Bugs patched in upstream, but not in already launched Android devices.
    a) Diffed the Pixel device patch history with the upstream Linux kernel for `/drivers/android/binder.c`
    b) Discussions with Android about syncing with upstream kernel

Google

Variant Analysis Approach

1) Bugs patched in upstream, but not in already launched Android devices.
    a) Diffed the Pixel device patch history with the upstream Linux kernel for `/drivers/android/binder.c`
    b) Discussions with Android about syncing with upstream kernel
2) Drivers whose poll handler uses a wait queue that is not tied to the lifetime of the file.

Variant Analysis Approach

1) Bugs patched in upstream, but not i
   a) Diffed the Pixel device patch hi
      for /drivers/android/binder.c
   b) Discussions with Android abou
2) Drivers whose poll handler uses a w
   lifetime of the file.

```
static unsigned int binder_poll(struct file *filp,
struct poll_table_struct *wait)
{
  struct binder_proc *proc = filp->private_data;
  struct binder_thread *thread = NULL;
  bool wait_for_proc_work;
  thread = binder_get_thread(proc);
  if (!thread)
    return POLLERR;
  binder_inner_proc_lock(thread->proc);
  thread->looper |= BINDER_LOOPER_STATE_POLL;
  wait_for_proc_work =
   binder_available_for_proc_work_ilocked(thread);
  binder_inner_proc_unlock(thread->proc);
  poll_wait(filp, &thread->wait, wait);
  if (binder_has_work(thread, wait_for_proc_work))
    return POLLIN;
  return 0;
}
```

Variant Analysis Approach

1) Bugs patched in upstream, but not i
   a) Diffed the Pixel device patch hi
      for /drivers/android/binder.c
   b) Discussions with Android abou

a w

The file operation is on the binder_proc, but we are passing the wait queue that is in binder_thread.

binder_thread can be freed prior to binder_proc.

```
static unsigned int binder_poll(struct file *filp,
struct poll_table_struct *wait)
{
  struct binder_proc *proc = filp->private_data;
  struct binder_thread *thread = NULL;
  bool wait_for_proc_work;
  thread = binder_get_thread(proc);
  if (!thread)
    return POLLERR;
  binder_inner_proc_lock(thread->proc);
  thread->looper |= BINDER_LOOPER_STATE_POLL;
  wait_for_proc_work =
   binder_available_for_proc_work_ilocked(thread);
  binder_inner_proc_unlock(thread->proc);
  poll_wait(filp, &thread->wait, wait);
  if (binder_has_work(thread, wait_for_proc_work))
    return POLLIN;
  return 0;
}
```

Variant Analysis Approach

1) Bugs patched in upstream, but not in already launched Android devices.
    a) Diffed the Pixel device patch history with the upstream Linux kernel for /drivers/android/binder.c
    b) Discussions with Android about syncing with upstream kernel
2) Drivers whose poll handler uses a wait queue that is not tied to the lifetime of the file.
    a) Manually searched 214/236 files in the Linux 4.4 kernel where there is a call to `poll_wait`.

Google

Variant Analysis Results

Approach #1 (Bugs patched in upstream, but not in ASB):

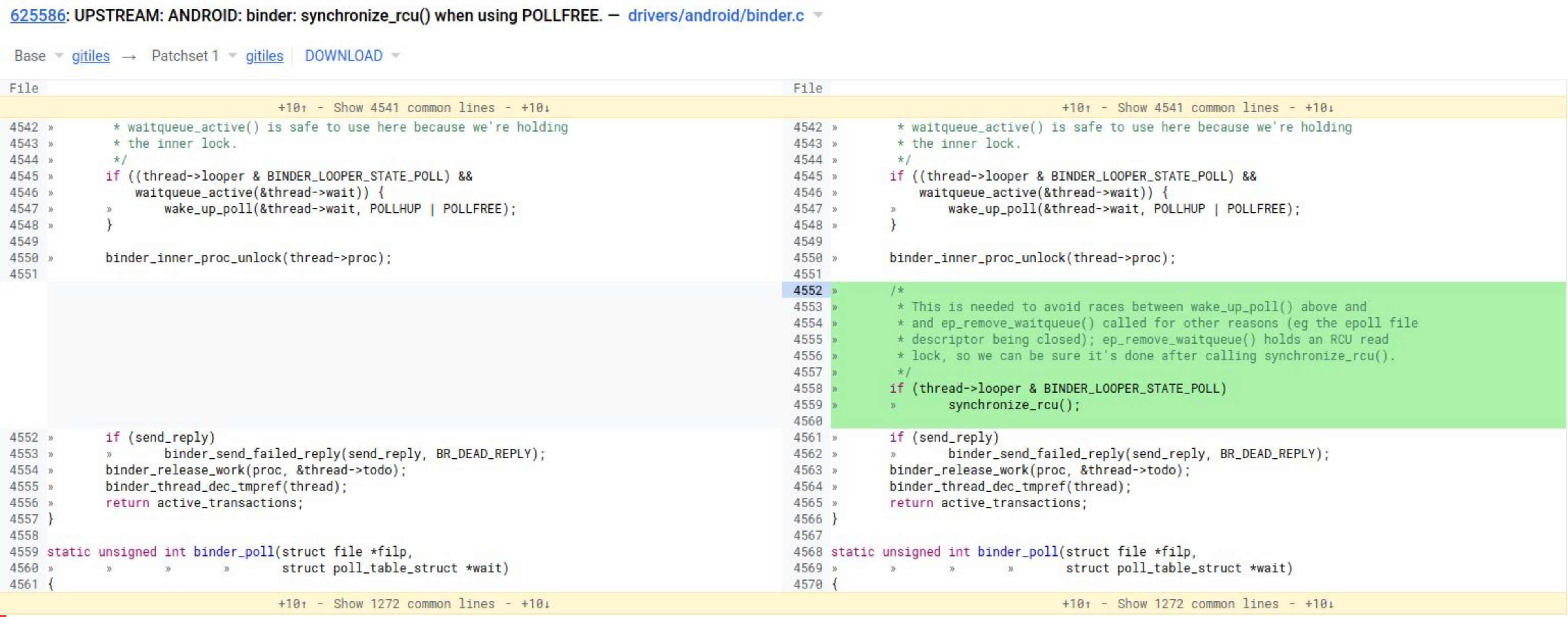
- CVE-2020-0030: Potential UAF due to race condition in binder_thread_release

Variant Analysis Results

Approach #1 (Bugs patched in upstream, but not in ASB):

- CVE-2020-0030: Potential UAF due to race condition in binder_thread_release
- Reported by syzcaller in Feb 2018.
- Patched upstream in Feb 2018.

Variant Analysis Results (CVE-2020-0030)



625586: UPSTREAM: ANDROID: binder: synchronize_rcu() when using POLLFREE. — drivers/android/binder.c

Base gitiles → Patchset 1 gitiles DOWNLOAD

File

```
+10↑ - Show 4541 common lines - +10↓
»	 * waitqueue_active() is safe to use here because we're holding
»	 * the inner lock.
»	 */
»	if ((thread->looper & BINDER_LOOPER_STATE_POLL) &&
»	    waitqueue_active(&thread->wait)) {
»	»	wake_up_poll(&thread->wait, POLLHUP | POLLFREE);
»	}

»	binder_inner_proc_unlock(thread->proc);

»	if (send_reply)
»	»	binder_send_failed_reply(send_reply, BR_DEAD_REPLY);
»	binder_release_work(proc, &thread->todo);
»	binder_thread_dec_tmpref(thread);
»	return active_transactions;
}

static unsigned int binder_poll(struct file *filp,
»	»	»	»	struct poll_table_struct *wait)
{
+10↑ - Show 1272 common lines - +10↓
```

File

```
+10↑ - Show 4541 common lines - +10↓
»	 * waitqueue_active() is safe to use here because we're holding
»	 * the inner lock.
»	 */
»	if ((thread->looper & BINDER_LOOPER_STATE_POLL) &&
»	    waitqueue_active(&thread->wait)) {
»	»	wake_up_poll(&thread->wait, POLLHUP | POLLFREE);
»	}

»	binder_inner_proc_unlock(thread->proc);

»	/*
»	 * This is needed to avoid races between wake_up_poll() above and
»	 * and ep_remove_waitqueue() called for other reasons (eg the epoll file
»	 * descriptor being closed); ep_remove_waitqueue() holds an RCU read
»	 * lock, so we can be sure it's done after calling synchronize_rcu().
»	 */
»	if (thread->looper & BINDER_LOOPER_STATE_POLL)
»	»	synchronize_rcu();

»	if (send_reply)
»	»	binder_send_failed_reply(send_reply, BR_DEAD_REPLY);
»	binder_release_work(proc, &thread->todo);
»	binder_thread_dec_tmpref(thread);
»	return active_transactions;
}

static unsigned int binder_poll(struct file *filp,
»	»	»	»	struct poll_table_struct *wait)
{
+10↑ - Show 1272 common lines - +10↓
```

Google

Variant Analysis Results (CVE-2020-0030)

`ioctl(BINDER_THREAD_EXIT)`	**`epoll_ctl(EPOLL_CTL_DEL)`**
`POLLFREE`	`ep_remove_wait_queue`
`  list_del_init(&wait->task_list)`	`  rcu_read_lock`
	`  whead = smp_load_acquire(&pwq->whead)`
`  smp_store_release(whead, NULL)`	`  if(whead)`
`...`	
`binder_dec_tmpref`	
`  ...`	
`  binder_free_thread`	
`    ...`	
`    kfree(thread)`	
	`    remove_wait_queue(whead, &pwq->wait)`
	`      spin_lock_irqsave(&whead->lock,flags)`

Variant Analysis Results (CVE-2020-0030)

ioctl(BINDER_THREAD_EXIT)	**epoll_ctl(EPOLL_CTL_DEL)**
`POLLFREE`	`ep_remove_wait_queue`
`list_del_init(&wait->task_list)`	`rcu_read_lock`
	`whead = smp_load_acquire(&pwq->whead)`
`smp_store_release(whead, NULL)`	`if(whead)`
`...`	
`binder_dec_tmpref`	
`...`	
`binder_free_thread`	
`...`	
$\boxed{\texttt{kfree(thread)}}$	
Wait queue is freed.	`remove_wait_queue(whead, &pwq->wait)`
	`spin_lock_irqsave(&whead->lock,flags)`

Google

Variant Analysis Results (CVE-2020-0030)

`ioctl(BINDER_THREAD_EXIT)`	**`epoll_ctl(EPOLL_CTL_DEL)`**
`POLLFREE`	`ep_remove_wait_queue`
`  list_del_init(&wait->task_list)`	`  rcu_read_lock`
	`  whead = smp_load_acquire(&pwq->whead)`
`  smp_store_release(whead, NULL)`	`  if(whead)`
`...`	
`binder_dec_tmpref`	
`  ...`	
`  binder_free_thread`	
`    ...`	
`    kfree(thread)`	UAF!
	`    remove_wait_queue(whead, &pwq->wait)`
	`      spin_lock_irqsave(&whead->lock,flags)`

Variant Analysis Results (CVE-2020-0030)

`ioctl(BINDER_THREAD_EXIT)`	**`epoll_ctl(EPOLL_CTL_DEL)`**
`POLLFREE`	`ep_remove_wait_queue`

`binder_dec_tmpref`
`  ...`
`  binder_free_thread`
`    ...`
`    kfree(thread)`
	UAF!
	`    remove_wait_queue(whead, &pwq->wait)`
	`      spin_lock_irqsave(&whead->lock,flags)`

Android patched in [February 2020 Android Security Bulletin]
(CVE-2020-0030).

Variant Analysis Results

Approach #1 (Bugs patched in upstream, but not in ASB):

- CVE-2020-0030: Potential UAF due to race condition in binder_thread_release

Approach #2 (Looking at other uses of `poll_wait`):

- Identified one potential bug, but the driver appeared to only be used in a single device a few years ago and then the driver/chip was replaced.

Lessons Learned

- Think about variant analysis more broadly than just the technical details of the vulnerability. Such as:
    - How it was found
    - What area of code it's in
    - Etc.
- When the product you're securing is downstream, a focus needs to be on taking the patches from upstream as well.
- Probably should have tried a static analysis tool.

Case Study: Chrome Webaudio UAF

CVE-2019-13720

Chrome webaudio UAF: About the Vuln

- Use-after-free in the **`webaudio`** component (**`convolver_node.cc`**) of the Blink renderer.
  - Enables renderer remote code execution.
  - Caused by only using a mutex on one thread that accesses the members, not both.
- Discovered by Kaspersky [blog post]

Chrome webaudio UAF: About the Vuln

- The UAF is caused by two different threads, the main thread and the audio rendering thread, operating on `ConvolverNode` members at the same time.
    - If you call `ConvolverHandler::SetBuffer` in the main thread, the `reverb_` member is freed.

Google

Main thread

```
void ConvolverHandler::SetBuffer(AudioBuffer* buffer,
                                 ExceptionState& exception_state) {
  DCHECK(IsMainThread());

  if (!buffer) {
    reverb_.reset();
    shared_buffer_ = nullptr;
    return;
  }

  if (buffer->sampleRate() != Context()->sampleRate()) {
    exception_state.ThrowDOMException(
        DOMExceptionCode::kNotSupportedError,
        "The buffer sample rate of " + String::Number(buffer->sampleRate()) +
            " does not match the context rate of " +
            String::Number(Context()->sampleRate()) + " Hz.");
    return;
  }

[...]
```

Main thread

```
void ConvolverHandler::SetBuffer(AudioBuffer* buffer,
                                 ExceptionState& exception_state) {
  DCHECK(IsMainThread());

  if (!buffer) {
    reverb_.reset();
    shared_buffer_ = nullptr;
    return;
  }

  if (buffer->sampleRate() != Context()->sampleRate()) {
    exception_state.ThrowDOMException(
        DOMExceptionCode::kNotSupportedError,
        "The buffer sample rate of " + String::Number(buffer->sampleRate()) +
            " does not match the context rate of " +
            String::Number(Context()->sampleRate()) + " Hz.");
    return;
  }

[...]
```

Freed without taking a mutex.

Chrome webaudio UAF: About the Vuln

- The UAF is caused by two different threads, the main thread and the audio rendering thread, operating on `ConvolverNode` members at the same time.
  - If you call `ConvolverHandler::SetBuffer` in the main thread, the `reverb_` member is freed.
  - The audio rendering thread, is triggered by the call `startRendering` and calls the `Process` method. It uses `reverb_`.

Audio rendering thread

```
void ConvolverHandler::Process(uint32_t frames_to_process) {
  AudioBus* output_bus = Output(0).Bus();
  DCHECK(output_bus);

  // Synchronize with possible dynamic changes to the impulse response.
  MutexTryLocker try_locker(process_lock_);
  if (try_locker.Locked()) {
    if (!IsInitialized() || !reverb_) {
      output_bus->Zero();
    } else {
      // Process using the convolution engine.
      // Note that we can handle the case where nothing is connected to the
      // input, in which case we'll just feed silence into the convolver.
      // FIXME:  If we wanted to get fancy we could try to factor in the 'tail
      // time' and stop processing once the tail dies down if
      // we keep getting fed silence.
      reverb_->Process(Input(0).Bus(), output_bus, frames_to_process);
    }
  } else {
    // Too bad - the tryLock() failed.  We must be in the middle of setting a
    // new impulse response.
    output_bus->Zero();
  }
}
```

Google

```
void ConvolverHandler::Process(uint32_t frames_to_process) {
  AudioBus* output_bus = Output(0).Bus();
  DCHECK(output_bus);

  // Synchronize with possible dynamic changes to the impulse response.
  MutexTryLocker try_locker(process_lock_);
  if (try_locker.Locked()) {
    if (!IsInitialized() || !reverb_) {
      output_bus->Zero();
    } else {
      // Process using the c
      // Note that we can ha
      // input, in which cas
      // FIXME:  If we wante
      // time' and stop proc
      // we keep getting fed
      reverb_->Process(Input
    }
  } else {
    // Too bad - the tryLock() failed.  We must be in the middle of setting a
    // new impulse response.
    output_bus->Zero();
  }
}
```

Audio rendering thread

```
MutexTryLocker try_locker(process_lock_);
```

Takes the mutex here, in the audio rendering thread, but there was no mutex taken in the main thread.

Google

```
void ConvolverHandler::Process(uint32_t frames_to_process) {
  AudioBus* output_bus = Output(0).Bus();
  DCHECK(output_bus);

  // Synchronize with possible dynamic changes to the impulse response.
  MutexTryLoc
  if (try_loc
    if (!IsIn
      output_
    } else {
      // Proc
      // Note
      // input, in which case we'll just feed silence into the convolver.
      // FIXME:  If we wanted to get fancy we could try to factor in the 'tail
      // time' and stop processing once the tail dies down if
      // we keep getting fed silence.
      reverb_->Process(Input(0).Bus(), output_bus, frames_to_process);
    }
  } else {
    // Too bad - the tryLock() failed.  We must be in the middle of setting a
    // new impulse response.
    output_bus->Zero();
  }
}
```

Audio rendering thread

```
reverb_->Process(Input(0).Bus(), output_bus, frames_to_process);
```

reverb_ member is used here after it has been freed by the main thread.

Google

Variant Analysis Approach

If the vulnerability can be described in one simple sentence, it’s a good candidate for Semmle.

- Sergei Glazunov

Variant Analysis Approach

- Semmle/CodeQL - a static analysis platform where you can write QL queries to search for certain patterns in source-code.

Variant Analysis Approach

- Semmle/CodeQL - a static analysis platform where you can write QL queries to search for certain patterns in source-code.
- Write a query to search for instances where there is a member variable and it’s used in two threads, but a mutex is only taken in one.

Variant Analysis Results

- Sergei Glazunov's CodeQL query found a variant (which has a few variants). [P0 Issue 1963, CVE-2019-13732]
  - 3 true positives out of 23 total results when run over just the `webaudio` module.
  - Variant found in `PannerHandler`.

Google

Variant Analysis Results

- Sergei Glazunov's CodeQL query found a variant (which has a few variants). [P0 Issue 1963]

"Unlike in the original issue, the main thread function acquires a lock before changing the **`panner_`** pointer, but **`TailTime`** (as well as **`LatencyTime`** and **`RequiresTailProcessing`**), which is called on the audio thread, [doesn't acquire a lock].

Usually functions in the audio thread rely on the lock being taken in **`Process`**. In this case, however, calls to **`TailTime`** doesn't go through **`Process`**. Thus, an attacker can free the object pointed by **`panner_`** while another thread is executing one of its methods."

Variant Analysis Results

- Sergei Glazunov’s CodeQL query found a variant (which has a few variants). [P0 Issue 1963]

“Unlike in the original issue, the main thread function acquires a lock before changing the **`panner_`** pointer, but **`TailTime`** (as well as **`LatencyTime`** and **`RequiresTailProcessing`**), which is called on the audio thread, [doesn't acquire a lock].

Usually functions in the audio thread rely on the lock being taken in **`Process`**. In this case, however, calls to **`TailTime`** doesn't go through **`Process`**. Thus, an attacker can free the object pointed by **`panner_`** while another thread is executing one of its methods.”

Variants of Variant

Variant Analysis Results

- Sergei Glazunov's CodeQL query found a variant (which has a few variants). [P0 Issue 1963, CVE-2019-13732]
  - 3 true positives out of 23 total results when run over just the

CVE-2019-13732: Patched in Chrome 79.0.3945.79, released 10 Dec 2019.

Lessons Learned

- From comment#2 on P0 Issue 1963:

“Unfortunately, the patch for this bug is incomplete. The call path triggered by the original test case is now safe since the fix has made `ProcessIfNecessary` acquire a lock. However, there's another call path to both `LatencyTime` and `TailTime` from `DeferredTaskHandler::UpdateTailProcessingHandlers`, which is still unprotected.”

Lessons Learned

- From [comment#2 on P0 Issue 1963:]()

"Unfortunately, the patch for this bug is incomplete. The call path triggered by the original test case is now safe since the fix has made **`ProcessIfNecessary`** acquire a lock. However, there's another call path to both **`LatencyTime`** and **`TailTime`** from **`DeferredTaskHandler::UpdateTailProcessingHandlers`**, which is still unprotected."

The bug report mentioned three functions. Instead of patching in these functions, the implemented patch was implemented in one of the 2 code paths that call these functions.

Lessons Learned

- From comment#2 on P0 Issue 1963:

“Unfortunately, the patch for this bug is incomplete. The call path triggered

CVE-2020-6406: Patched in Chrome 80.0.3987.87, released 4 Feb 2020

both `LatencyTime` and `TailTime` from `DeferredTaskHandler::UpdateTailProcessingHandlers`, which is still unprotected.”

Lessons Learned

- Vendors are under a lot of pressure to get patches out quickly, but this can lead to poor patch quality (not complete for all variants).

Lessons Learned

- Vendors are under a lot of pressure to get patches out quickly, but this can lead to poor patch quality (not complete for all variants).
- Vendors can also focus on just ensuring that a POC no longer works, even when its mentioned that other variants exist.

Lessons Learned

- Vendors are under a lot of pressure to get patches out quickly, but this can lead to poor patch quality (not complete for all variants).
- Vendors can also focus on just ensuring that a POC no longer works, even when its mentioned that other variants exist.
- Static analysis has its place and can work for variant analysis!

Case Study: Internet Explorer jscript

CVE-2019-1367/ CVE-2019-1429

About the Vuln

- Bug class: JScript variable (represented as VAR structure) isn't properly tracked by garbage collector

About the Vuln

- Bug class: JScript variable (represented as VAR structure) isn’t properly tracked by garbage collector
- January 2018: Multiple bugs reported for bug class. [1504, 1505, 1506, 1587]

About the Vuln

- Bug class: JScript variable (represented as VAR structure) isn't properly tracked by garbage collector
- January 2018: Multiple bugs reported for bug class. [1504, 1505, 1506, 1587]
- December 2018: CVE-2018-8653 -- First detected instance of this bug class used in the wild
    - Couldn't be caught by fuzzer used in January issues because relied on JScript features not implemented in fuzzer.
    - Discovered by Google's Threat Analysis Group

About the Vuln

- Bug class: JScript variable (represented as VAR structure) isn't properly tracked by garbage collector
- January 2018: Multiple bugs reported for bug class. [1504, 1505, 1506, 1587]
- December 2018: CVE-2018-8653 -- First detected instance of this bug class used in the wild
- September 2019: CVE-2019-1367 -- Function arguments not being tracked by garbage collector during `Array.sort` callback

About the Vuln

- Bug class: JScript variable (represented as VAR structure) isn't properly tracked by garbage collector
- January 2018: Multiple bugs reported for bug class. [1504, 1505, 1506, 1587]
- December 2018: CVE-2018-8653 -- First detected instance of this bug class used in the wild
- September 2019: CVE-2019-1367 -- Function arguments not being tracked by garbage collector during `Array.sort` callback

About the Vuln

- Bug class: JScript variable (represented as VAR structure) isn't properly tracked by garbage collector
- January 2018: Multiple bugs reported for bug class. [1504, 1505, 1506, 1587]
- December 2018: CVE-2018-8653 -- First detected instance of this bug class used in the wild
- September 2019: CVE-2019-1367 -- Function arguments not being tracked by garbage collector during `Array.sort` callback
  - Discovered by Google's Threat Analysis Group
  - Fix was incomplete. Fixed again under CVE-2019-1429

Variant Analysis Approach

- Variant analysis done by Ivan Fratric

- Manual analysis - Manually attempt to free function arguments in every JScript callback that Ivan knows
- Fuzzer - Run another fuzzing session with modified JScript.dll to more easily check this bug class:
    - Freed VARs are modified so that accessing them would crash immediately
    - Freed VARs never allocated again

Variant Analysis Results

- During the manual review, found 1 trivial variant. [P0 Issue 1947]
    - Members of the `arguments` object aren't tracked by the garbage collector during the `toJSON` callback

Variant Analysis Results

- During the manual review, found 1 trivial variant. [P0 Issue 1947]
  - Members of the `arguments` object aren't tracked by the garbage collector during the `toJSON` callback

Everything is exactly the same as the original vulnerability except it’s the `toJSON` callback instead of the `Array.sort` callback.

Variant Analysis Results

- During the manual review, found 1 trivial variant. [P0 Issue 1947]
    - Members of the `arguments` object aren't tracked by the garbage collector during the `toJSON` callback
- The modified fuzzer returned two findings: the original vulnerability and this one.

Variant Analysis Results

- During the manual review, found 1 trivial variant. [P0 Issue 1947]
  - Members of the `arguments` object aren't tracked by the garbage

Microsoft patched in November 2019 update, grouping it with CVE-2019-1429

Lessons Learned

- More variant analysis on the bug class when it’s initially reported.

Lessons Learned

- More variant analysis on the bug class when it’s initially reported.
    - Address bug classes comprehensively, not each vulnerability individually.

Lessons Learned

- More variant analysis on the bug class when it’s initially reported.
    - Address bug classes comprehensively, not each vulnerability individually.
- Quality and complete patches.

Lessons Learned

- More variant analysis on the bug class when it’s initially reported.
  - Address bug classes comprehensively, not each vulnerability individually.
- Quality and complete patches.
  - The bug wasn’t fixed initially
  - AND the trivial variant wasn’t patched :(

Lessons Learned

- More variant analysis on the bug class when it’s initially reported.
  - Address bug classes comprehensively, not each vulnerability individually.
- Quality and complete patches.
  - The bug wasn’t fixed initially
  - AND the trivial variant wasn’t patched :(
  - Sharing proposed patches with the reporter can help find these issues earlier to ensure a complete patch.

Lessons Learned

- More variant analysis on the bug class when it's initially reported.
    - Address bug classes comprehensively, not each vulnerability individually.
- Quality and complete patches.
    - The bug wasn't fixed initially
    - AND the trivial variant wasn't patched :(
    - Sharing proposed patches with reporter can help find these issues earlier to ensure a complete patch.

Yes, this is harder when the vulnerability is being exploited in-the-wild and thus is under a 7-day deadline. But that can be even more reason to engage researchers to ensure quality and completeness.

Conclusion

Doing Variant Analysis

1. There are lots of different techniques that can be used when doing variant analysis:
   a. Manual Analysis
   b. Static Analysis
   c. Fuzzing

Doing Variant Analysis

1. There are lots of different techniques that can be used when doing variant analysis:
   a. Manual Analysis
   b. Static Analysis
   c. Fuzzing
2. Think both about:
   a. Variants of the bug class (technical details)
   b. Variants in how the attacker may have found the bug
      i. Patches applied upstream but not downstream
      ii. Hard to reach code path

Takeaways

- Patching to a proof-of-concept is not sufficient is often not fixing the vulnerability, much less patching variants.
- Many times, there are trivial variants identified when doing variant analysis on 0-days used in the wild.
    - The attackers are almost certainly finding these because they want to put in the least amount of investment and already know the details of the original vulnerabilities that they used.

What researchers can do

- Check the patches for the vulnerabilities you report
- Hold vendors accountable for fixing variants of the vulnerabilities you report too
    - Walk the vendor through how you found the variants so they understand the “how” in addition to the “what”
    - Propose ideas for how to address the bug class or class of exploiting the vulnerability comprehensively

What vendors can do

- Don’t just focus on the single vulnerability, think about variants in your code base and how to fix the whole class.
- Engage with researchers on your patches, most of us want to help!
- Even if you don’t have time to do a full variant analysis before a patch date/disclosure, continue it after that date.

What Project Zero is doing

- Working to engage vendors to review fix methodologies and implementations.
  - Fixing earlier is easier and cheaper, and better for users. Win Win!
- Published a new disclosure policy focusing on complete and quality patches.

2019	2020 Trial
1. 90 days or when the bug is fixed (decided by researcher discretion), whichever is the earliest.	1. Full 90 days, regardless of when the bug is fixed. Earlier disclosure with mutual agreement.
2. Policy goal: ◦ Faster patch development	2. Policy goals: ◦ Faster patch development ◦ Thorough patch development ◦ Improved patch adoption
3. Inconsistent handling of incomplete fixes. Such issues are either filed as separate vulnerabilities or added to existing reports at researcher discretion.	3. Details of incomplete fixes will be reported to the vendor and added to the existing report (which may already be public) and will not receive a new deadline.
4. Bugs fixed in the grace period* would be opened to the public sometime after a patch was released.	4. Project Zero tracker reports are immediately opened when patched during the grace period*.
5. Project Zero tracker reports are opened at researcher discretion after the deadline expires.	5. Project Zero tracker reports are opened automatically on Day 90 (or earlier under mutual agreement).

Google

2019	2020 Trial
1. 90 days or when the bug is fixed (decided by researcher discretion), whichever is the earliest.	1. Full 90 days, regardless of when the bug is fixed. Earlier disclosure with mutual agreement.
2. Policy goal: ◦ Faster patch development	2. Policy goals: ◦ Faster patch development ◦ Thorough patch development ◦ Improved patch adoption
3. Inconsistent handling of incomplete fixes. Such issues are either filed as separate vulnerabilities or added to existing reports at researcher discretion.	3. Details of incomplete fixes will be reported to the vendor and added to the existing report (which may already be public) and will not receive a new deadline.
4. Bugs fixed in the grace period* would be opened to the public sometime after a patch was released.	4. Project Zero tracker reports are immediately opened when patched during the grace period*.
5. Project Zero tracker reports are opened at researcher discretion after the deadline expires.	5. Project Zero tracker reports are opened automatically on Day 90 (or earlier under mutual agreement).

Google

2019	2020 Trial
1. 90 days or when the bug is fixed (decided by researcher discretion), whichever is the earliest.	1. Full 90 days, regardless of when the bug is fixed. Earlier disclosure with mutual agreement.
2. Policy goal: ◦ Faster patch development	2. Policy goals: ◦ Faster patch development ◦ Thorough patch development ◦ Improved patch adoption
3. Inconsistent handling of incomplete fixes. Such issues are either filed as separate vulnerabilities or added to existing reports at researcher discretion.	3. Details of incomplete fixes will be reported to the vendor and added to the existing report (which may already be public) and will not receive a new deadline.
4. Bugs fixed in the grace period* would be opened to the public sometime after a patch was released.	4. Project Zero tracker reports are immediately opened when patched during the grace period*.
5. Project Zero tracker reports are opened at researcher discretion after the deadline expires.	5. Project Zero tracker reports are opened automatically on Day 90 (or earlier under mutual agreement).

Google

Thank you!

Maddie Stone
@maddiestone