Google

0-day In-the-Wild Exploitation in 2022...so far

Maddie Stone (@maddiestone)
FIRST 2022

0-day exploit:
an exploit targeting a vulnerability that defenders don’t yet know about

Code execution 0-day in Windows has been under active exploit for 7 weeks

Protecting Android users from 0-Day attacks

Bahrain: Devices of three activists hacked with Pegasus spyware

Apple Rushes Out Patches for 0-Days in MacOS, iOS

North Korean hackers unleashed Chrome 0-day exploit on hundreds of US targets

2 New Mozilla Firefox 0-Day Bugs Under Active Attack — Patch Your Browser ASAP!

Jordanian Human Rights Defenders and Journalists Hacked with Pegasus Spyware

Make 0-day hard.

Learn from 0-days exploited in the wild to **make 0-day hard.**

0day "In the Wild"

Last updated: 2021-05-03

This spreadsheet is used to track cases of zero-day exploits that were detected "in the wild". This means the vulnerability was detected in real attacks against users as a zero-day vulnerability (i.e. not known to the public or the vendor at the time of detection). This data is collected from a range of public sources. We include relevant links to third-party analysis and attribution, but we do this only for your information; their inclusion does not mean we endorse or validate the content there.

An introduction to this spreadsheet is available on the Project Zero blog:

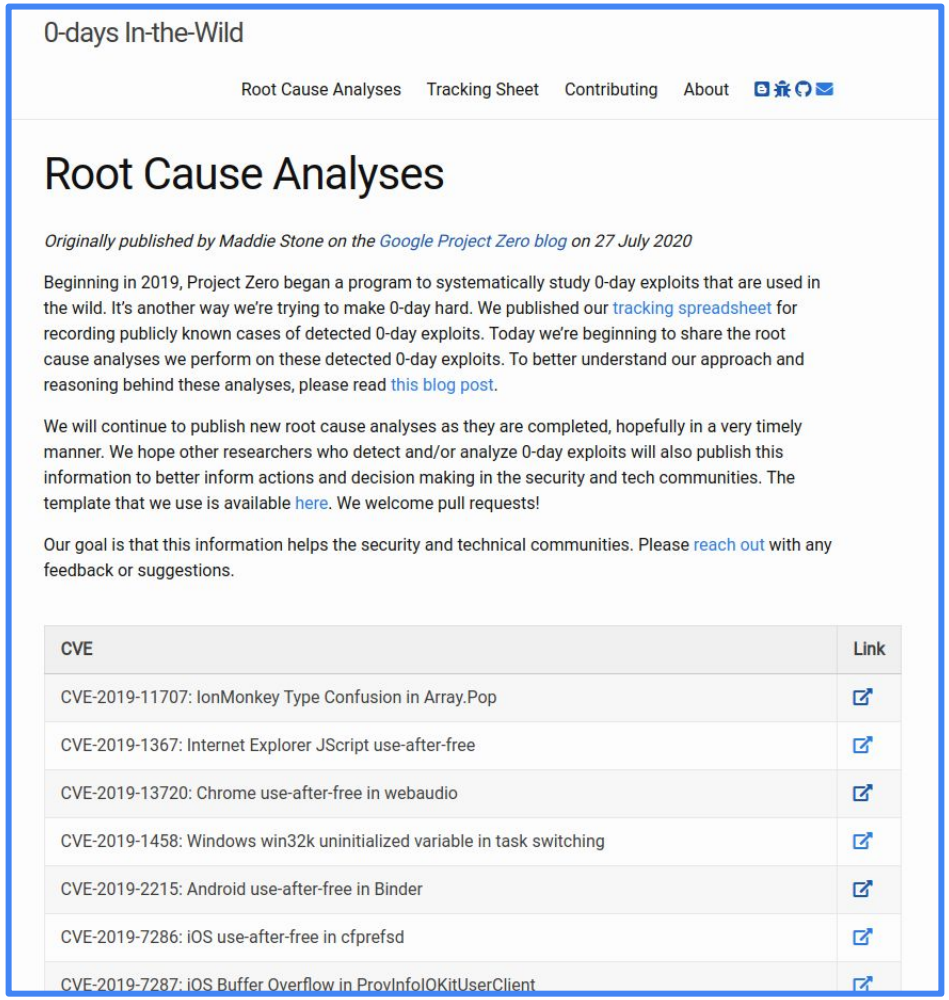
https://googleprojectzero.blogspot.com/p/0day.html

Some additional notes on how the data is processed:

- **Scope for inclusion:** there are some 0day exploits (such as CVE-2017-12824) in areas that aren't active research targets for Project Zero. Generally this list includes targets that Project Zero has previously investigated (i.e. there are bug reports in our issue tracker) or will investigate in the near future.
- **Security supported:** this list does not include exploits for software that is explicitly EOL at the time of discovery (such as the ExplodingCan exploit for IIS on Windows Server 2003, surfaced in 2017).
- **Post-disclosure:** this list does not include CVEs that were opportunistically exploited by attackers in the gap between public disclosure (or "full disclosure") and a patch becoming available to users (such as CVE-2015-0072, CVE-2018-8414 or CVE-2018-8440).
- **Reasonable inference:** this list includes exploits that were not discovered in an active breach, but were leaked or discovered in a form that suggests with high confidence that they were probably used "in the wild" at some point (e.g. Equation Group and Hacking Team leaks).
- **Date resolution:** we only set the date of discovery when the reporter specifies one. If a discovery is indicated as being made in "late April" or "early March", we record that as if no date was provided.
- **Attribution:** generally the “claimed attribution” column refers to the attack team that is reportedly using the exploit, but in some cases it can refer to the supplier of the exploit (c.f. HackingTeam, NSO Group, Exodus Intel) if no other information is available.
- **Time range:** data collection starts from the day we announced Project Zero -- July 15, 2014.

For additions, corrections, questions, or comments, please contact 0day-in-the-wild@google.com

https://docs.google.com/spreadsheets/d/1lkNJ0uQwbeC1ZTRrxdtuPLCIl7mlUreoKfSIgajnSyY/



0-days In-the-Wild

Root Cause Analyses Tracking Sheet Contributing About

Root Cause Analyses

Originally published by Maddie Stone on the Google Project Zero blog on 27 July 2020

Beginning in 2019, Project Zero began a program to systematically study 0-day exploits that are used in the wild. It's another way we're trying to make 0-day hard. We published our tracking spreadsheet for recording publicly known cases of detected 0-day exploits. Today we're beginning to share the root cause analyses we perform on these detected 0-day exploits. To better understand our approach and reasoning behind these analyses, please read this blog post.

We will continue to publish new root cause analyses as they are completed, hopefully in a very timely manner. We hope other researchers who detect and/or analyze 0-day exploits will also publish this information to better inform actions and decision making in the security and tech communities. The template that we use is available here. We welcome pull requests!

Our goal is that this information helps the security and technical communities. Please reach out with any feedback or suggestions.

CVE	Link
CVE-2019-11707: IonMonkey Type Confusion in Array.Pop	
CVE-2019-1367: Internet Explorer JScript use-after-free	
CVE-2019-13720: Chrome use-after-free in webaudio	
CVE-2019-1458: Windows win32k uninitialized variable in task switching	
CVE-2019-2215: Android use-after-free in Binder	
CVE-2019-7286: iOS use-after-free in cfprefsd	
CVE-2019-7287: iOS Buffer Overflow in ProvInfoIOKitUserClient	

https://googleprojectzero.github.io/0days-in-the-wild/rca.html

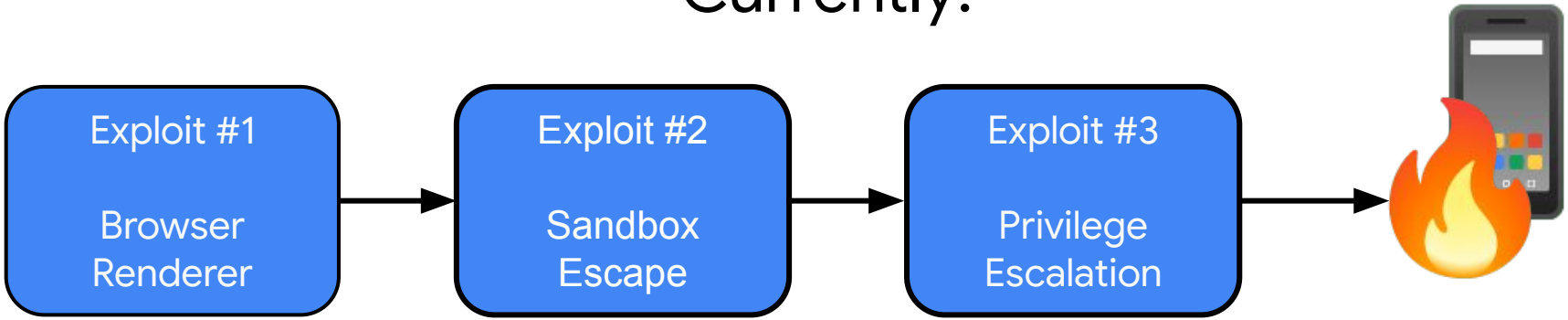
Make 0-day hard.

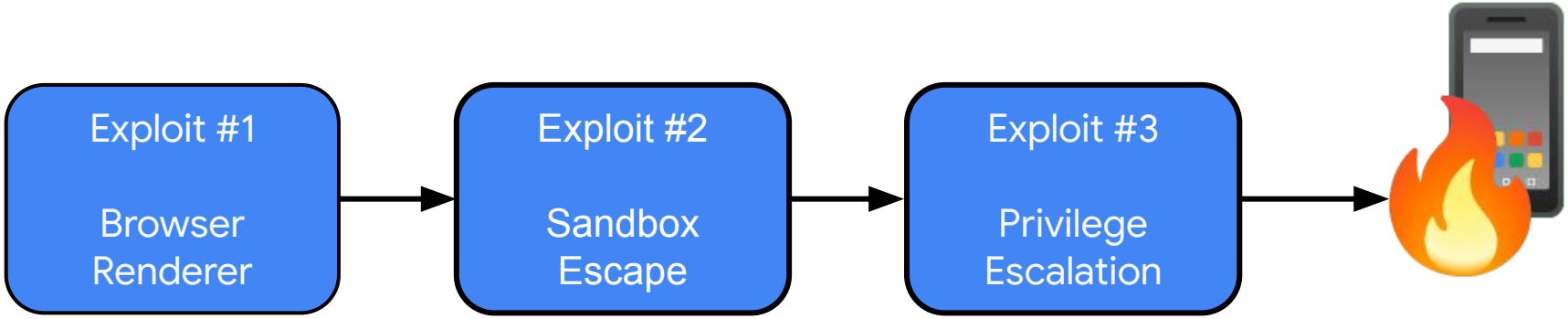
1. Increase cost* per exploit.

2. Increase number of exploits required.

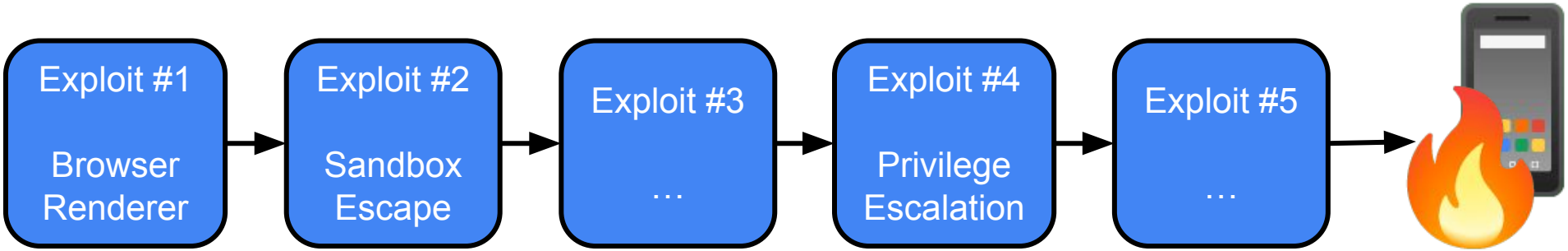
*Cost is not the same thing as price. Cost can also mean time, expertise, etc.

Currently:


Exploit #1
Browser Renderer
Exploit #2
Sandbox Escape
Exploit #3
Privilege Escalation



Harder...



When 0-day is hard, to have a successful 0-day exploit you'll have to:

- Discover new bug classes
- Research novel attack surfaces
- Develop brand new exploitation techniques
- Use exploits selectively & only in highly targeted attacks

Currently 0-day is **not** hard.

50% of in-the-wild 0-days from 2022 are variants of previously patched vulnerabilities

as of 15 June 2022

22% are variants of in-the-wild 0-days from 2021

as of 15 June 2022

2020 Year in Review Report:

> “25% of the 0-days detected in 2020 are closely related to previously publicly disclosed vulnerabilities”

https://googleprojectzero.blogspot.com/2021/02/deja-vu-lnerability.html

Google

2022 Case Studies

Win32k CVE-2022-21882

Windows Win32k: CVE-2022-21882 & CVE-2021-1732

- CVE-2022-21882 in-the-wild 0-day from 2022
  - Patched in January 2022
- CVE-2021-1732 in-the-wild 0-day from 2021
  - Patched in February 2021

- Exact same user-mode callback bug just triggered in different ways

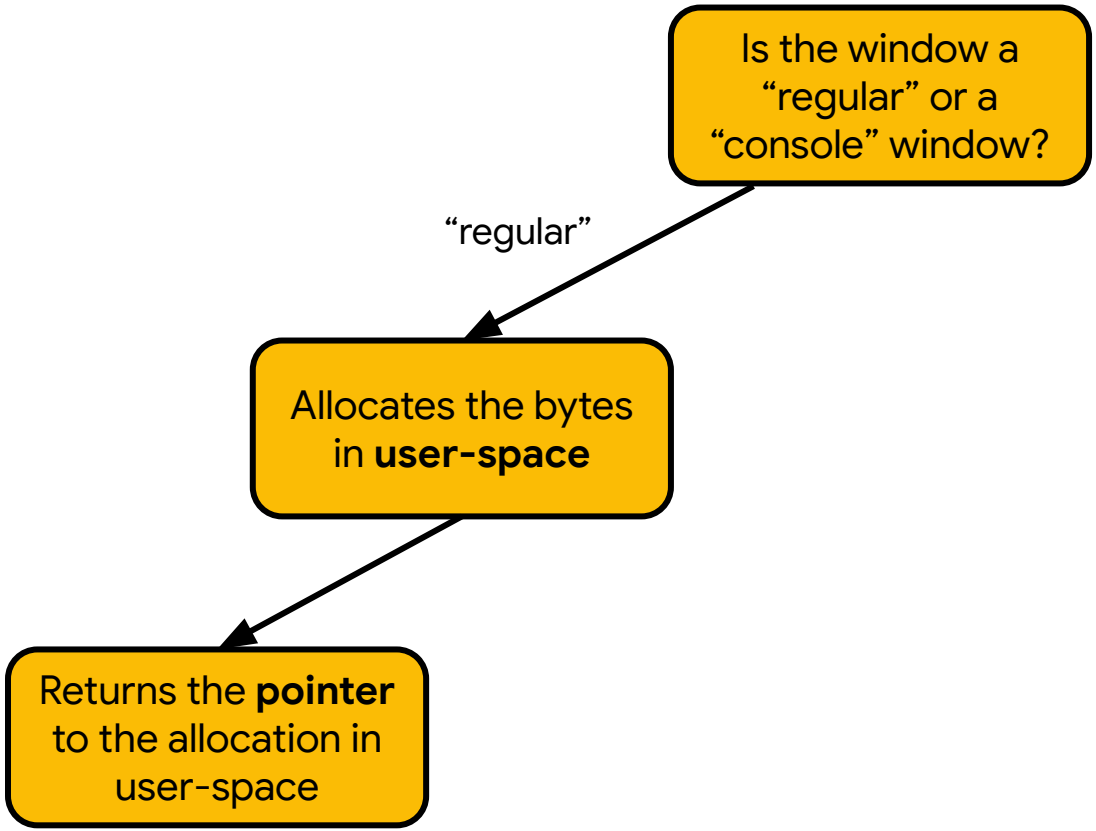
```
+0x040 hMod16           : Uint2B
+0x042 fnid             : Uint2B
+0x048 spwndNext        : Ptr64 tag
+0x050 spwndPrev        : Ptr64 tag
+0x058 spwndParent      : Ptr64 tag
+0x060 spwndChild       : Ptr64 tag
+0x068 spwndOwner       : Ptr64 tag
+0x070 rcWindow         : tagRECT
+0x080 rcClient         : tagRECT
+0x090 lpfnWndProc      : Ptr64
+0x098 pcls             : Ptr64 tag
+0x0a0 hrgnUpdate       : Ptr64 HRG
+0x0a8 ppropList        : Ptr64 tagPROPLIST
+0x0b0 pSBInfo          : Ptr64 tagSBINFO
+0x0b8 spmenuSys        : Ptr64 tagMENU
+0x0c0 spmenu           : Ptr64 tagMENU
+0x0c8 hrgnClip         : Ptr64 HRGN__
+0x0d0 hrgnNewFrame     : Ptr64 HRGN__
+0x0d8 strName          : _LARGE_UNICODE_STRING
+0x0e8 cbwndExtra       : Int4B
+0x0f0 spwndLastActive  : Ptr64 tagWND
+0x0f8 hImc             : Ptr64 HIMC__
+0x100 dwUserData       : Uint8B
kd> dt win32k!tagWND
```

`cbwndExtra` field in `tagWND` struct -

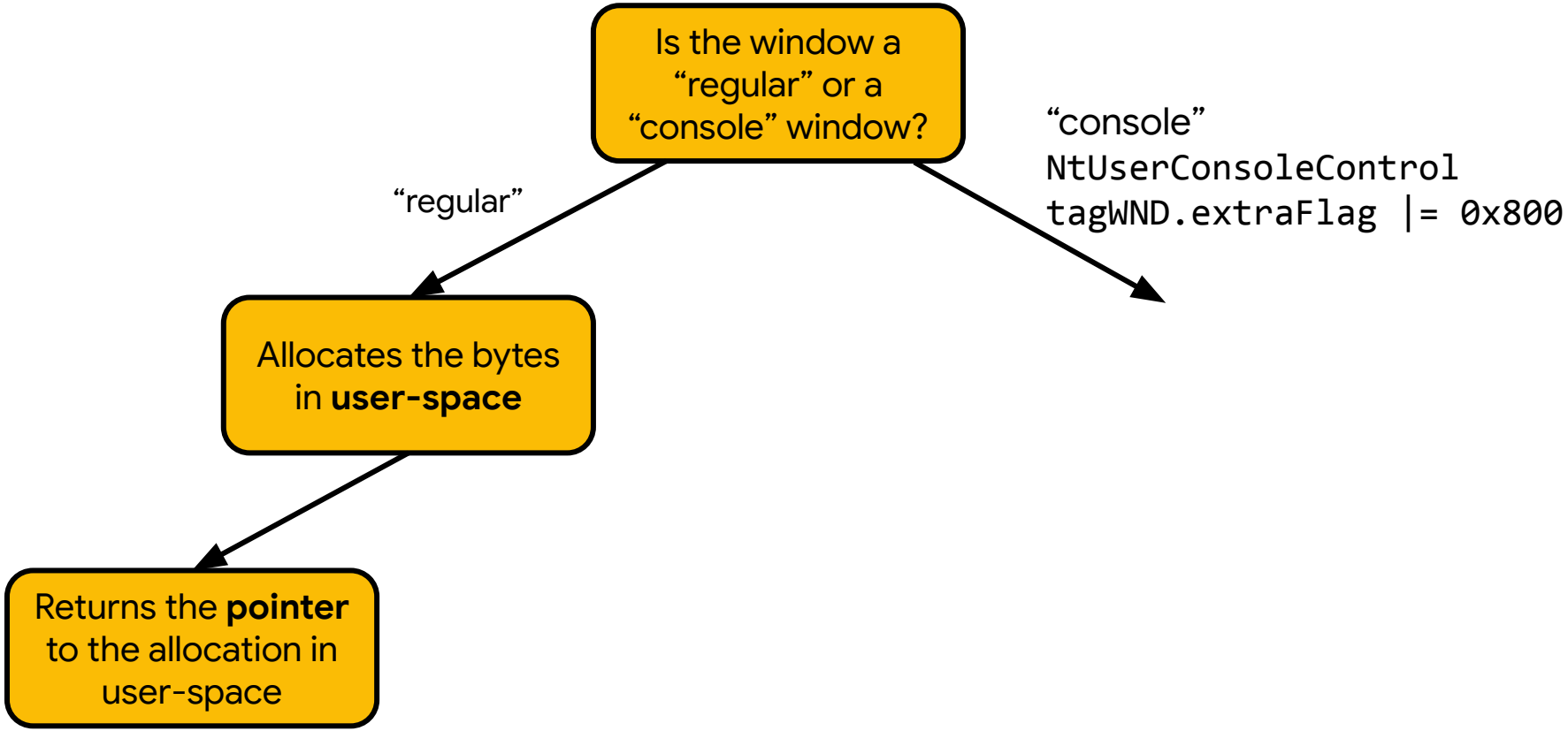
Stores number of extra bytes to add to tagWND structure

Image from byteraptors!
https://byteraptors.github.io/windows/exploitation/2020/06/03/exploitingcve2019-1458.html

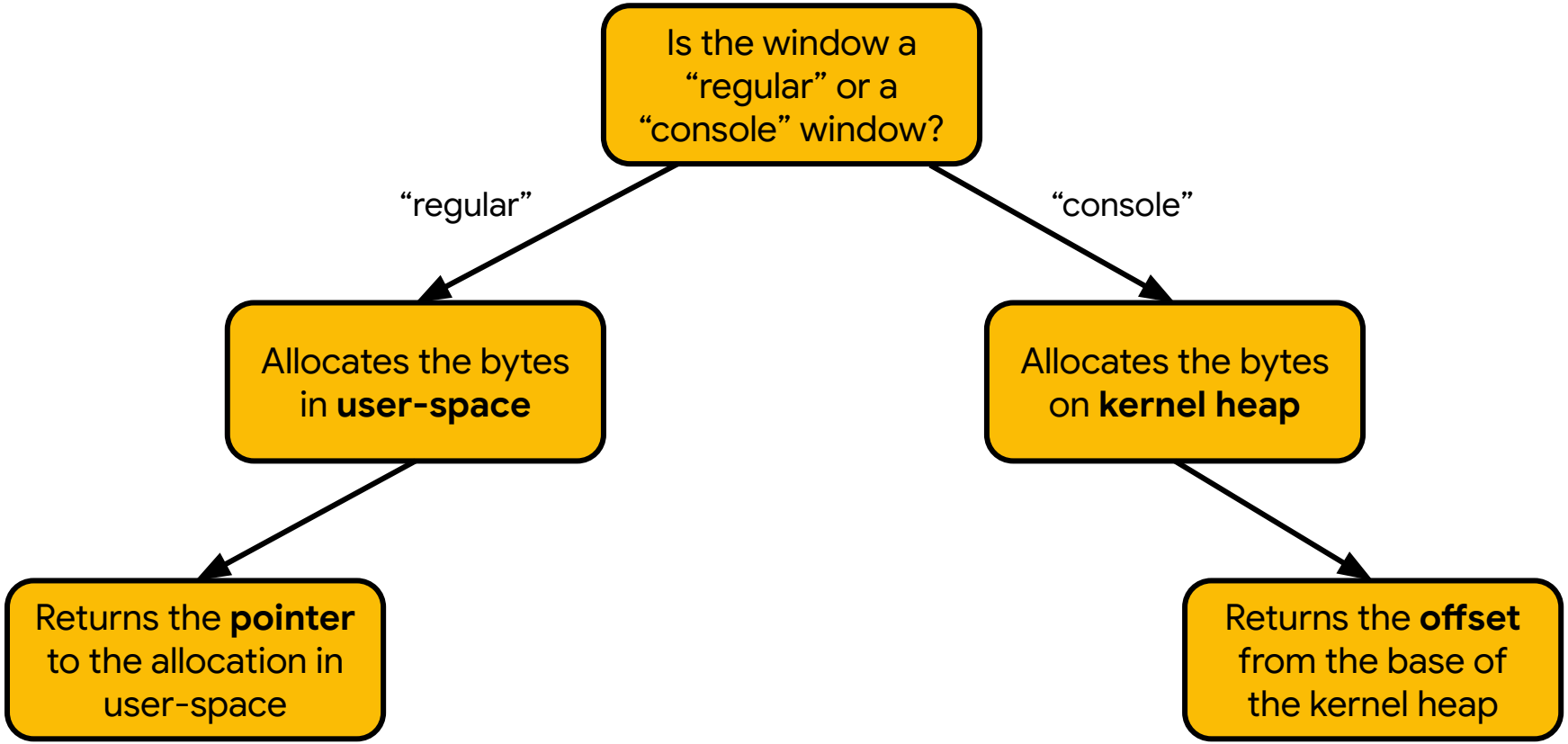
`xxxClientAllocWindowClassExtraBytes`: allocates the extra bytes



`xxxClientAllocWindowClassExtraBytes`: allocates the extra bytes



`xxxClientAllocWindowClassExtraBytes`: allocates the extra bytes



Triggering the Bug

1. Set `cbWndExtra` to custom value for a “regular” window
2. `xxxClientAllocWindowClassExtraBytes` allocates cbwndExtra bytes on user-mode heap
3. Change window to a console window during the user-mode callback
4. Kernel interprets return value as offset from kernel heap ⟶

   kernel out-of-bounds read & write

What happened?

- The original exploit for CVE-2021-1732 triggered the call to through `xxxCreateWindowEx`
- The patch only modified `xxxCreateWindowEx` to check if the window type had been changed, but...
- There are many more ways to call `xxxClientAllocWindowClassExtraBytes`, which caused the same result
- The new patch seems to prevent triggering this bug in `xxxClientAllocWindowClassExtraBytes`

IOMobileFrameBuffer CVE-2022-22587

iOS IOMobileFrameBuffer: CVE-2022-22587 & CVE-2021-30983

- CVE-2022-22587 in-the-wild 0-day from 2022
  - Patched in January 2022 (15.3)
- CVE-2021-30983 in-the-wild 0-day from 2021 & used at Tianfu Cup 2021
  - Patched in December 2021 (15.2)

IOMobileFrameBuffer Bugs

- Manages the screen framebuffer
- Accessible from the app sandbox
  - Apple has been adding hardening measures as a response to all these bugs
- Kernel extension on Intel macs and iPhone 11's and earlier
- Runs on the display coprocessor (DCP) on M1 macs and iPhones 12 & 13

IOMFB::UniformityCompensator::set

```
uint8_t* pages = compensator->inline_buffer; // +0x24
for (int pg_cnt = 0; pg_cnt < 3; pg_cnt++) {
  uint8_t* this_page = pages;
  for (int i = 0; i < controlled_size; i++) {
    memcpy(this_page, indirect_buffer_ptr, 4 * controlled_size);
    indirect_buffer_ptr += 4 * controlled_size;
    this_page += 0x100;
  }
  pages += 0x4000;
}
```

```
uint8_t* pages = compensator->inline_buffer; // +0x24
for (int pg_cnt = 0; pg_cnt < 3; pg_cnt++) {
  uint8_t* this_page = pages;
  for (int i = 0; i < controlled_size; i++) {
    memcpy(this_page, indirect_buffer_ptr, 4 * controlled_size);
    indirect_buffer_ptr += 4 * controlled_size;
    this_page += 0x100;
  }
  pages += 0x4000;
}
```

CVE-2021-30983: Buffer overflow

```
if (controlled_size > 0x41) {
  [...]
  return 0x80000001;
}
[...]
uint8_t* pages = compensator->inline_buffer; // +0x24
for (int pg_cnt = 0; pg_cnt < 3; pg_cnt++) {
  uint8_t* this_page = pages;
  for (int i = 0; i < controlled_size; i++) {
    memcpy(this_page, indirect_buffer_ptr, 4 * controlled_size);
    indirect_buffer_ptr += 4 * controlled_size;
    this_page += 0x100;
  }
  pages += 0x4000;
}
```

CVE-2021-30983 Fix

```
if (controlled_size > 0x41) {
  [...]
  return 0x80000001;
}
[...]
uint8_t* pages = compensator-
for (int pg_cnt = 0; pg_cnt < 3; pg_cnt++) {
  uint8_t* this_page = pages;
  for (int i = 0; i < controlled_size; i++) {
    memcpy(this_page, indirect_buffer_ptr, 4 * controlled_size);
    indirect_buffer_ptr += 4 * controlled_size;
    this_page += 0x100;
  }
  pages += 0x4000;
}
[...lots of subtractions down here then used for writing to buffer...]
```

CVE-2021-30983 Fix:
Added upper bound, but no lower bound

```
if (controlled_size > 0x
  [...]
  return 0x80000001;
}
[...]
uint8_t* pages = compens
for (int pg_cnt = 0; pg_
  uint8_t* this_page = pages;
  for (int i = 0; i < controlled_size; i++) {
    memcpy(this_page, indirect_buffer_ptr, 4 * controlled_size);
    indirect_buffer_ptr += 4 * controlled_size;
    this_page += 0x100;
  }
  pages += 0x4000;
}
```

CVE-2022-22587:
When `controlled_size = 0`,
controlled_size underflows during subtraction again leading to out-of-bounds read/write

```
[...lots of subtractions down here then used for writing to buffer...]
```

```
if ((controlled_size - 2) > 0x3F) {
  [...]
  return 0x80000001;
}
[...]
uint8_t* pages = compensator->inline_buffer; // +0x24
for (int pg_cnt = 0; pg_cnt < 3; pg_cnt++) {
  uint8_t* this_page = pages;
  for (int i = 0; i < controlled_size; i++) {
    memcpy(this_page, indirect_buffer_ptr, 4 * controlled_size);
    indirect_buffer_ptr += 4 * controlled_size;
    this_page += 0x100;
  }
  pages += 0x4000;
}
[...lots of subtractions down here then used for writing to buffer...]
```

CVE-2022-22587 Fix:
Added lower bound

Chromium CVE-2022-1096

Chromium: CVE-2022-1096

- CVE-2016-5128 researcher reported bug
  - Patched in July 2016
- CVE-2021-30551 in-the-wild 0-day from 2021
  - Patched in June 2021
- CVE-2022-1096 in-the-wild 0-day from 2022
  - Patched in March 2022
- CVE-2022-1232 - fixed the CVE-2022-1096 incomplete fix
  - Patched in April 2022

Chromium Property Access Interceptor Bugs

- A special method that runs every time a user tries to access a property of the object
- Can lead to user JavaScript execution ***during*** the property assignment process
- Don’t check the state of the object after running the user JavaScript execution
- Leaves objects in corrupted states which can be used for UXSS or a type confusion for remote code execution

CVE-2016-5128 - March 2016

- Property access interceptor for `HTMLEmbedElement`
- Allowed access to a property belonging to a different website, which was used for UXSS

CVE-2021-30551 - June 2021

- Also the property access interceptor for `HTMLEmbedElement`
- If the object doesn't have a property with the specified name it can be added during the user JavaScript execution in the interceptor
- Doesn't re-check the status of the property and still returns that the property doesn't exist
- The property is then added to the object again such that the object now has two properties with the same name.

CVE-2022-1096 - March 2022

- Property access interceptor for `CSSStyleDeclaration`
- CVE-2021-30551 POC accessed a property in the object's prototype chain
- CVE-2022-1096 was the same bug, but the property accessed was directly the object's

```
Maybe<bool> Object::SetPropertyInternal(LookupIterator* it,
                                        Handle<Object> value,
                                        Maybe<ShouldThrow> should_throw,
                                        StoreOrigin store_origin,
                                        bool* found) {
```


https://source.chromium.org/chromium/chromium/src/+/0981e91a4f8692af337e2588562ad1504f4bffdc:v8/src/objects/objects.cc;l=2474

```
case LookupIterator::INTERCEPTOR: {
  if (it->HolderIsReceiverOrHiddenPrototype()) {
    Maybe<bool> result =
        JSObject::SetPropertyWithInterceptor(it, should_throw, value);
    if (result.IsNothing() || result.FromJust()) return result;
  } else {
    Maybe<PropertyAttributes> maybe_attributes =
        JSObject::GetPropertyAttributesWithInterceptor(it);
    if (maybe_attributes.IsNothing()) return Nothing<bool>();
    if ((maybe_attributes.FromJust() & READ_ONLY) != 0) {
      return WriteToReadOnlyProperty(it, value, should_throw);
    }
    if (maybe_attributes.FromJust() == ABSENT) {
      it->Next();
    } else {
      it->NotFound();
    }
    return Object::SetSuperProperty(it, value, store_origin, should_throw);
  }
  break;
}
```


https://chromium.googlesource.com/v8/v8/+/f9857fdf743eeb263aec3944259ad811f564291b/src/objects/objects.cc#2557

```
case LookupIterator::INTERCEPTOR: {
  if (it->HolderIsReceiverOrHiddenPrototype()) {
    Maybe<bool> result =
        JSObject::SetPropertyWithInterceptor(it, should_throw, value);
    if (result.IsNothing() || result.FromJust()) return result;
  } else {
    Maybe<PropertyAttributes> maybe_attributes =
        JSObject::GetPropertyAttributesWithInterceptor(it);
    if (maybe_attributes.IsNothing()) return Nothing<bool>();
    if ((maybe_attributes.FromJust() & READ_ONLY) != 0) {
      return WriteToReadOnlyProperty(it, value, should_throw);
    }
    if (maybe_attributes.FromJust() == ABSENT) {
      it->Next();
    } else {
      it->NotFound();
    }
    return Object::SetSuperProperty(it, value, store_origin, should_throw);
  }
  break;
}
```

CVE-2021-30551 Fix

https://chromium.googlesource.com/v8/v8/+/f9857fdf743eeb263aec3944259ad811f564291b/src/objects/objects.cc#2557

```
case LookupIterator::INTERCEPTOR: {
  if (it->HolderIsReceiverOrHiddenPrototype()) {
    Maybe<bool> result =
        JSObject::SetPropertyWithInterceptor(it, should_throw, value);
    if (result.IsNothing() || result.FromJust()) return result;
  } else {
    Maybe<PropertyAttributes> maybe_attributes =
        JSObject::GetPropertyAttributesWithInterceptor(it);
    if (maybe_attributes.IsNothing()) return Nothing<bool>();
    if ((maybe_attributes.FromJust() & READ_ONLY) != 0) {
      return WriteToReadOnlyProperty(it, value, should_throw);
    }
    if (maybe_attributes.FromJust() == ABSENT) {
      it->Next();
    } else {
      it->NotFound();
    }
    return Object::SetSuperProperty(it, value, store_origin, should_throw);
  }
  break;
}
```


https://chromium.googlesource.com/v8/v8/+/f9857fdf743eeb263aec3944259ad811f564291b/src/objects/objects.cc#2557

```
case LookupIterator::INTERCEPTOR: {
  if (it->HolderIsReceiverOrHiddenPrototype()) {
    Maybe<bool> result =
        JSObject::SetPropertyWithInterceptor(it, should_throw, value);
    if (result.IsNothing() || result.FromJust()) return result;
    it->Next();
  } else {
    Maybe<PropertyAttributes> maybe_attributes =
        JSObject::GetPropertyAttributesWithInterceptor(it);
    if (maybe_attributes.IsNothing()) return Nothing<bool>();
    if ((maybe_attributes.FromJust() & READ_ONLY) != 0) {
      return WriteToReadOnlyProperty(it, value, should_throw);
    }
    if (maybe_attributes.FromJust() == ABSENT) {
      it->Next();
    } else {
      it->NotFound();
    }
    return Object::SetSuperProperty(it, value, store_origin, should_throw);
  }
  return Object::SetSuperProperty(it, value, store_origin, should_throw);
}
```

CVE-2022-1096 Fix

https://chromium.googlesource.com/v8/v8/+/bacf506f6f7a187c7b08468fb559e4d6007d2a96/src/objects/objects.cc#2509

CVE-2022-1232 - April 2022

- Also in the property access interceptor for `CSSStyleDeclaration`
- Same bug as CVE-2022-1096 is reachable by `Object.defineProperty()`
- Needed to patch `JSObject::DefineOwnPropertyIgnoreAttributes`, not just `Object::SetPropertyInternal`

CVE-2022-1096

```
style = document.createElement('p').style;
style.prop = { toString: () => {
  style.prop = 1;
}};
```

CVE-2022-1232

```
style = document.createElement('p').style;
Object.defineProperty(style, 'prop', {
  value: { toString() { style.prop = 1 } }
});
```

And more...

Product	2022 ITW CVE	Variant
Chromium	CVE-2022-1364	CVE-2021-21195
WebKit “Zombie”	CVE-2022-22620	Bug was originally fixed in 2013, patch was regressed in 2016
Windows “PetitPotam”	CVE-2022-26925	CVE-2021-36942 - Patch was regressed
Google Pixel	CVE-2021-39793	Linux bug
Atlassian Confluence	CVE-2022-26134	CVE-2021-26084
Windows “Follina”	CVE-2022-30190	CVE-2021-40444 (2021 itw)

What can we do?

Correct and comprehensive patches

1. Root cause analysis
2. Variant analysis
3. Patch analysis
4. Detection techniques
5. Exploit technique mitigations
6. Other hardening/systemic improvements

Transparency

THANK YOU!

@maddiestone
0day-in-the-wild <at> google.com

Google

References

- 2021 Year in Review blog post
- 2020 Year in Review blog post
- 2019 Year in Review blog post
- Project Zero blog post on 2021 iOS in-the-wild exploit
- Project Zero blog on 2022 WebKit 0-day's history
- Project Zero 0-day tracking sheet
- 0-day in-the-wild root cause analyses
- Talk about performing root cause analysis
- Talk about performing variant analysis

Headlines for Slide #3

- https://arstechnica.com/information-technology/2022/05/code-execution-0day-in-windows-has-been-under-active-exploit-for-7-weeks/
- https://arstechnica.com/information-technology/2022/03/north-korean-hackers-unleashed-chrome-0-day-exploit-on-hundreds-of-us-targets/
- https://threatpost.com/apple-rushes-out-patches-0-days-macos-ios/179222/
- https://www.amnesty.org/en/latest/news/2022/02/bahrain-devices-of-three-activists-hacked-with-pegasus-spyware/
- https://blog.google/threat-analysis-group/protecting-android-users-from-0-day-attacks/
- https://citizenlab.ca/2022/04/peace-through-pegasus-jordanian-human-rights-defenders-and-journalists-hacked-with-pegasus-spyware/
- https://thehackernews.com/2022/03/2-new-mozilla-firefox-0-day-bugs-under.html