Google

2022 Year in Review

0-days Detected In-the-Wild in 2022

Maddie Stone
@maddiestone
Zer0Con 2023

Make 0day hard.

0-day exploits exploited in-the-wild.

0-day exploits **detected** & **disclosed** as in-the-wild.

40

0-days detected in-the-wild in 2022*

0day "In the Wild"

Last updated: 2023-04-11

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:

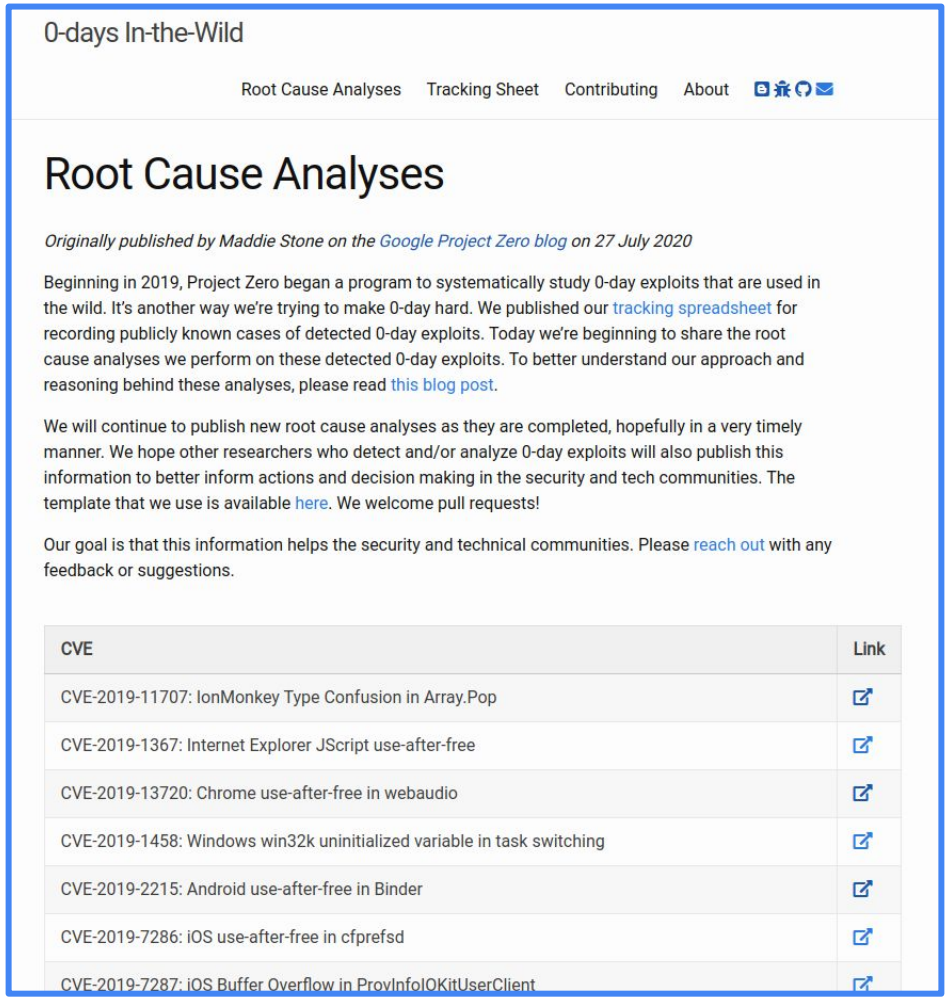
https://googleprojectzero.blogspot.com/p/0day.html

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.
- **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/edit#gid=0



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.

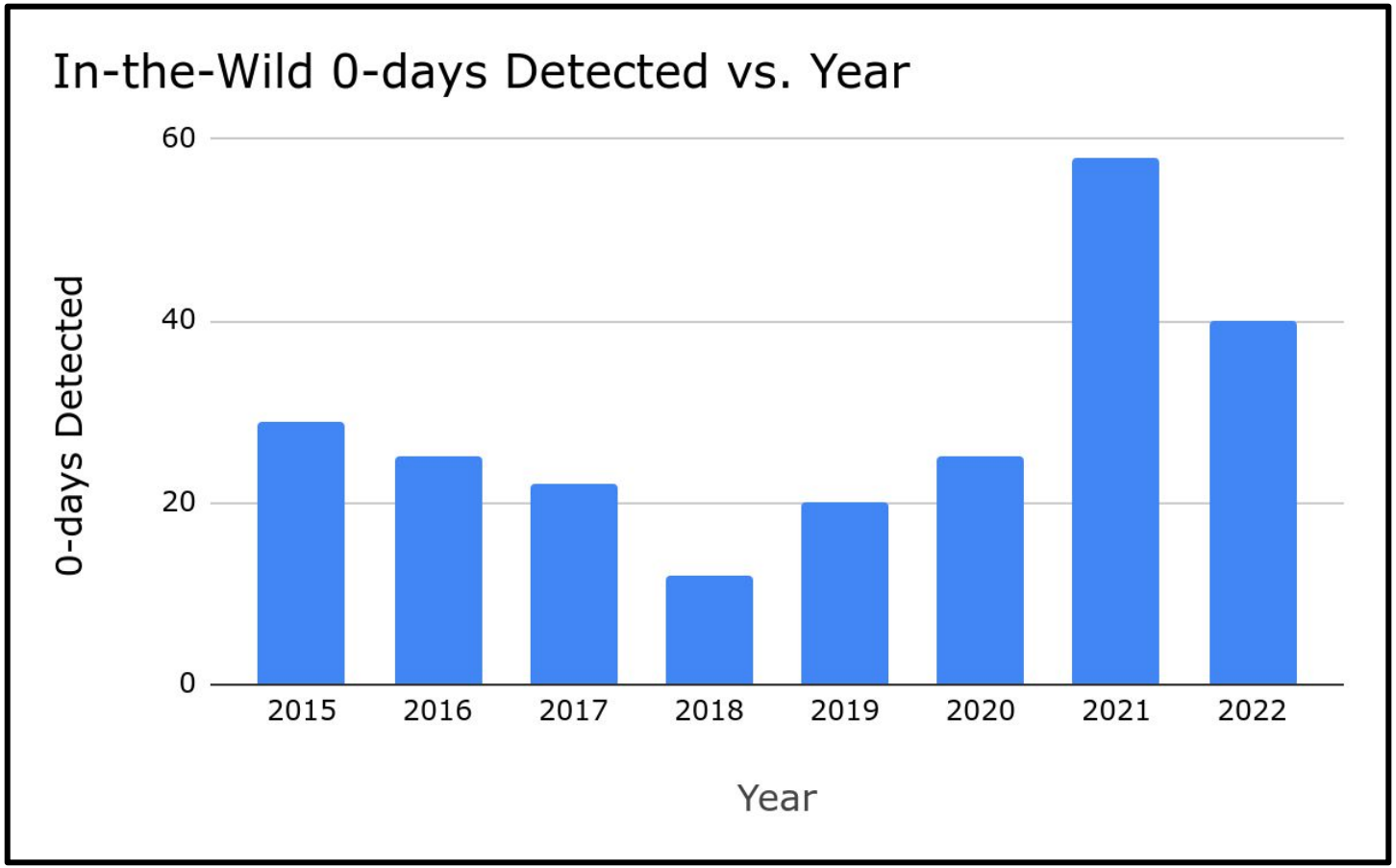
We will continue to publish new root cause analyses as they are completed, hopefully in a very timely manner. We hope other researchers who detect and/or analyze 0-day exploits will also publish this information to better inform actions and decision making in the security and tech communities. The template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

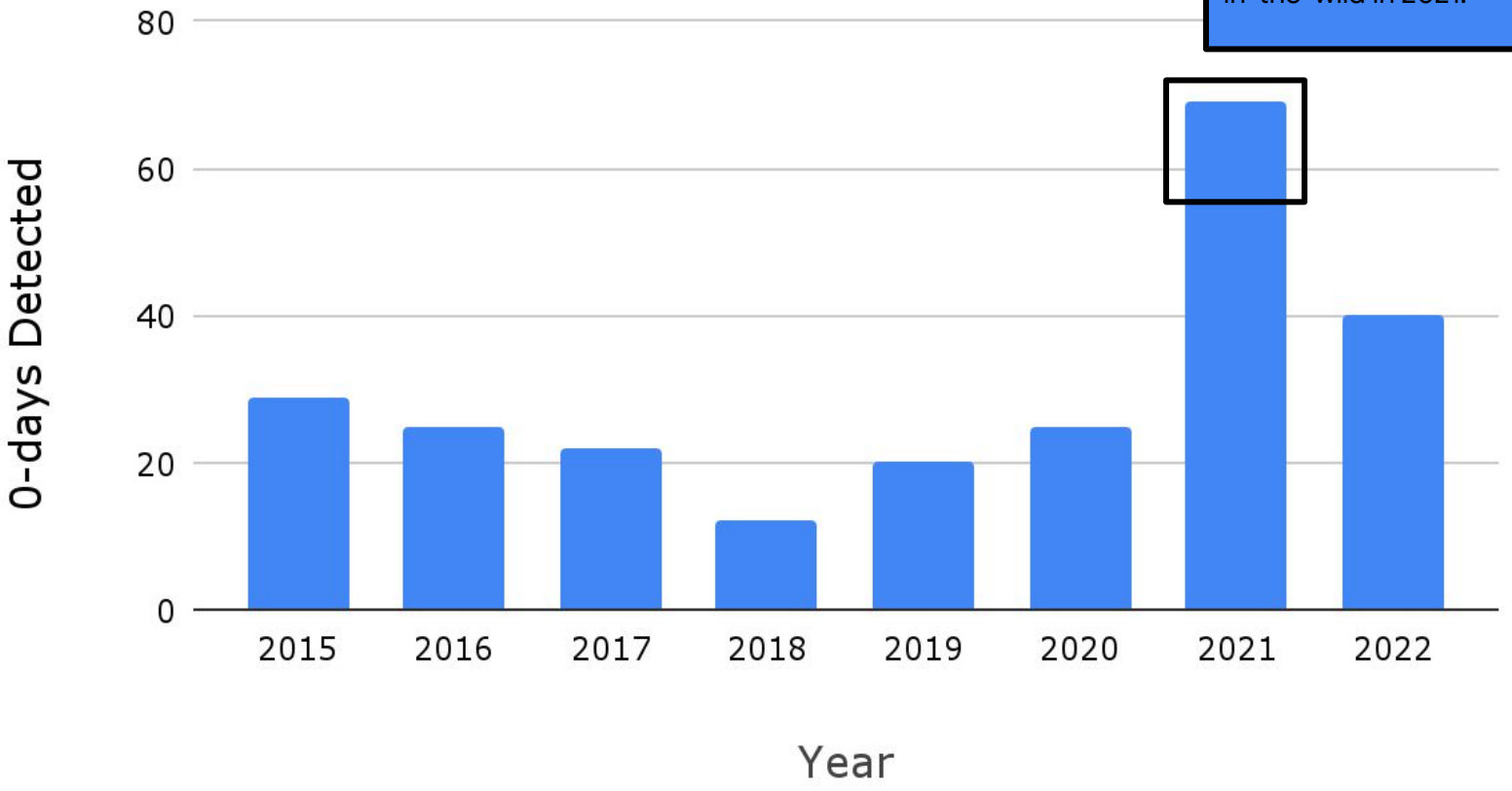
CVE	Link
CVE-2019-11707: IonMonkey Type Confusion in Array.Pop	
CVE-2019-1367: Internet Explorer JScript use-after-free	
CVE-2019-13720: Chrome use-after-free in webaudio	
CVE-2019-1458: Windows win32k uninitialized variable in task switching	
CVE-2019-2215: Android use-after-free in Binder	
CVE-2019-7286: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

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

Caveat: These are my takes & thoughts. I'd love to hear yours.


In-the-Wild 0-days Detected vs. Year
60
40
20
0
0-days Detected
2015
2016
2017
2018
2019
2020
2021
2022
Year

In-the-Wild 0-days Detected vs. Year



Since April 2022, we've learned about 11 more 0-days exploited in-the-wild in 2021.

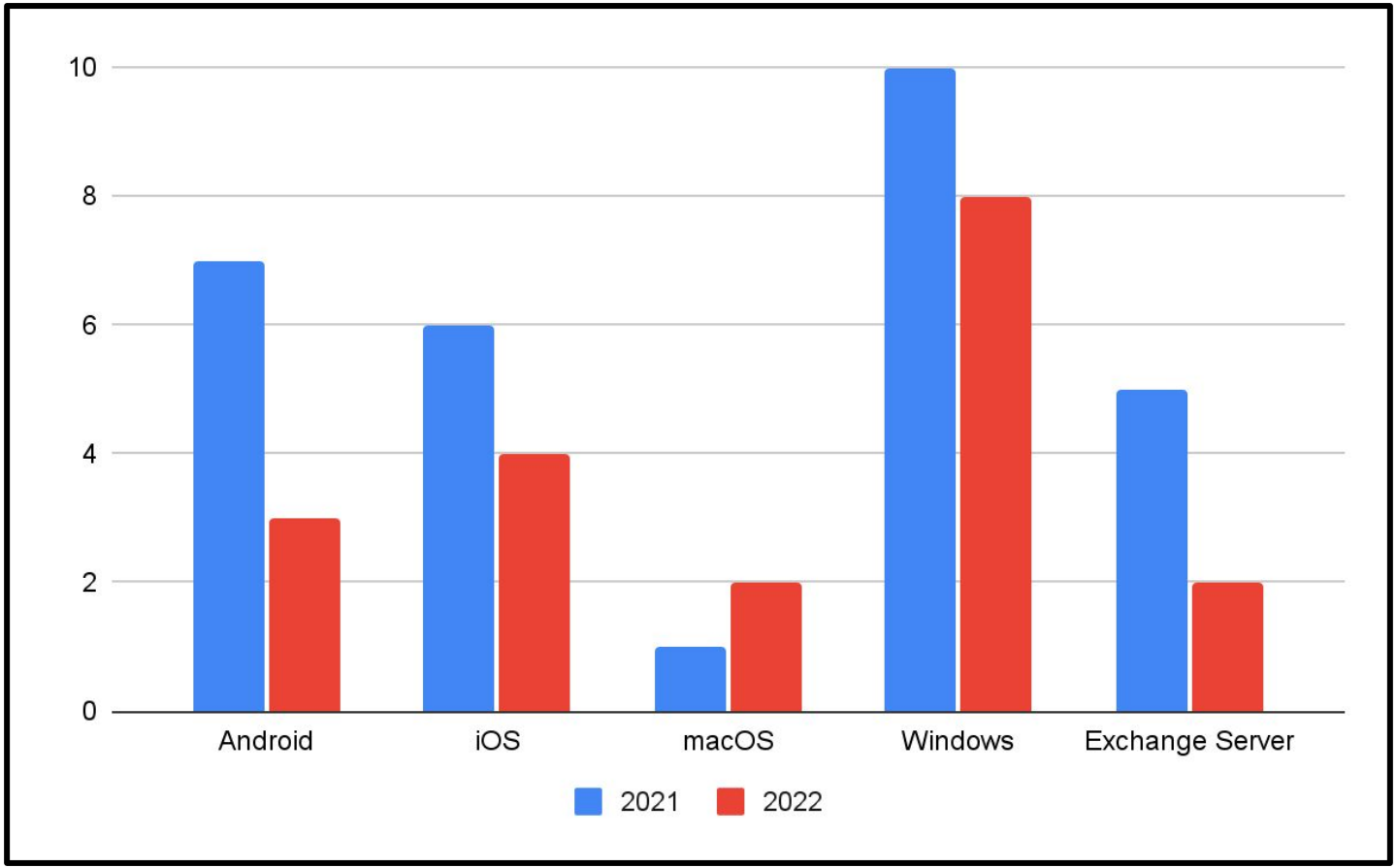
oogle, Mandiant say zero-day numbers reached all-time highs in 2021

Google: Record Year for Zero Days in 2021

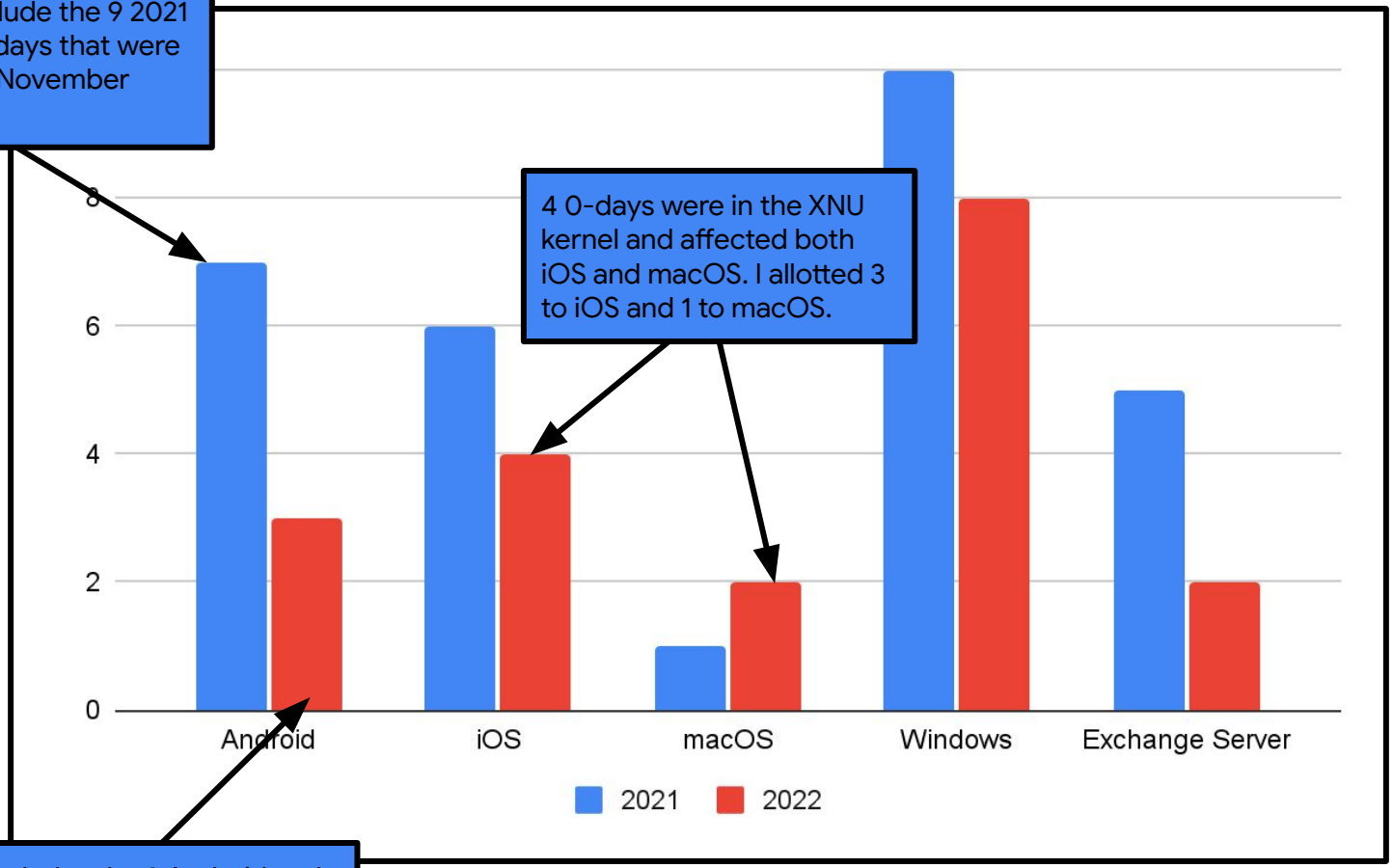
2021 has broken the record for zero-day hacking attacks

Hackers are exploiting 0-days more than ever

Record High Number of Zero-Day Vulnerabilities, Numbers May Be Owed to Improved Detection Says Mandiant and Google

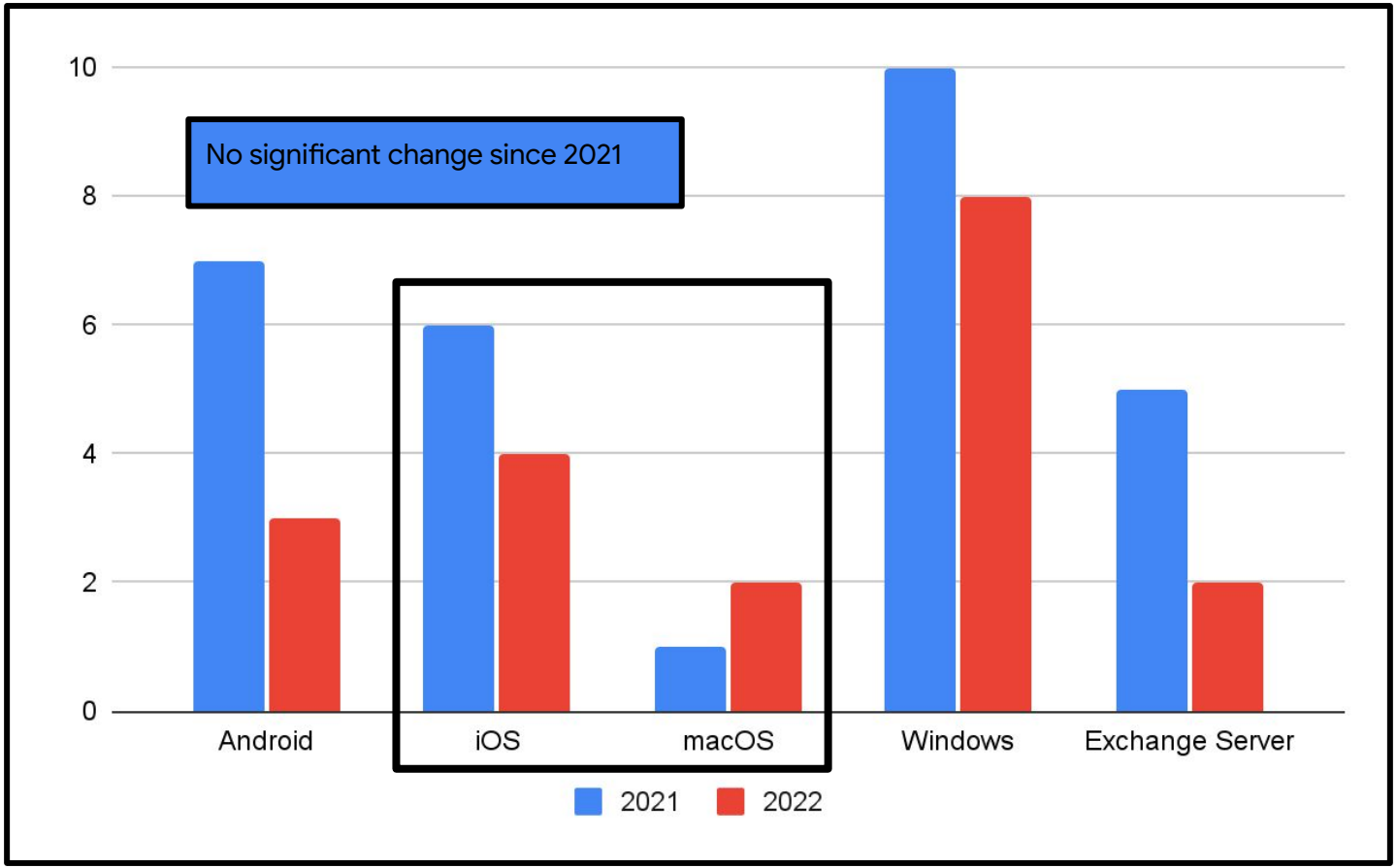

10
8
6
4
2
0
Android
iOS
macOS
Windows
Exchange Server
2021
2022

Does not include the 9 2021 Samsung 0-days that were disclosed in November 2022.

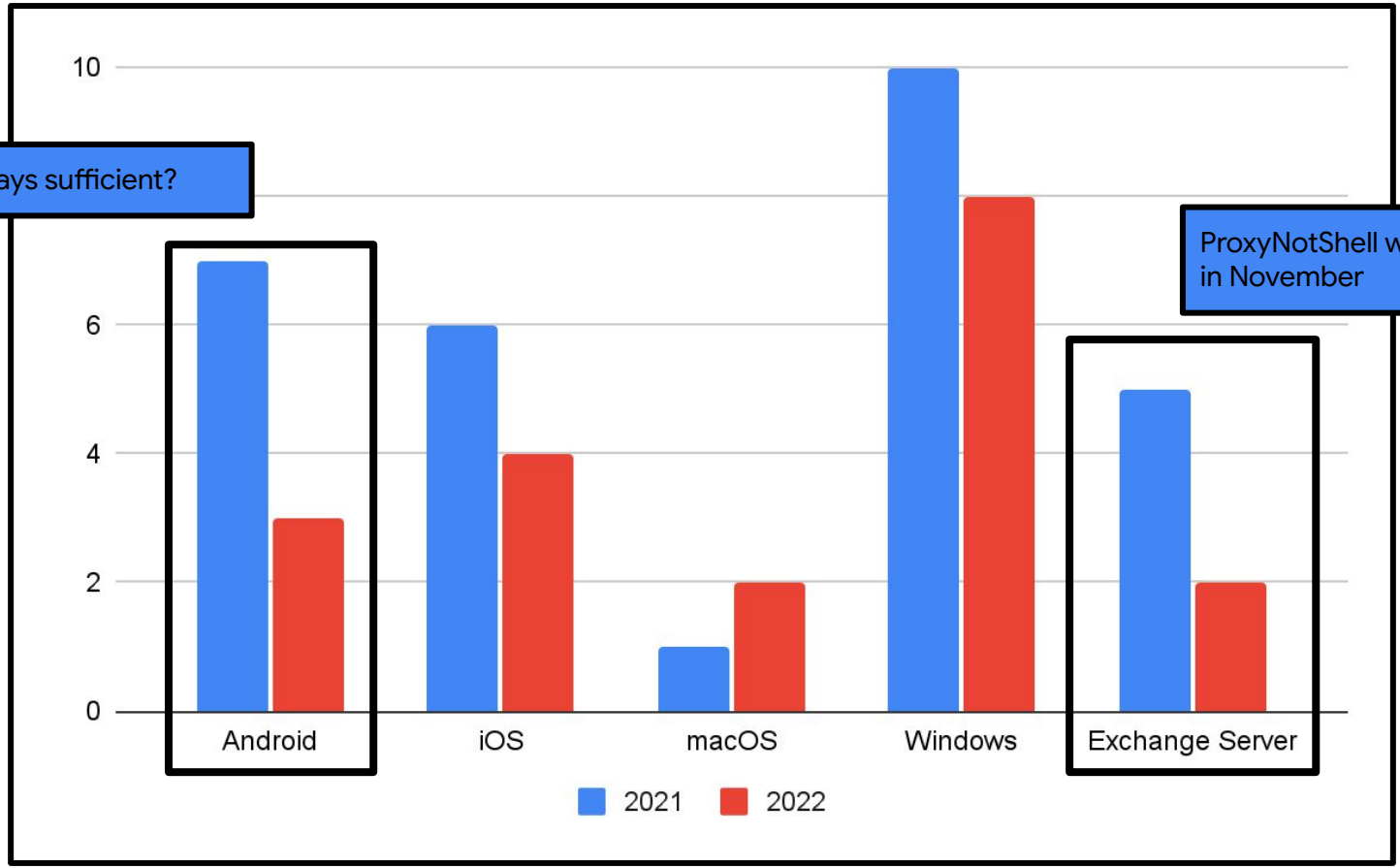


4 0-days were in the XNU kernel and affected both iOS and macOS. I allotted 3 to iOS and 1 to macOS.

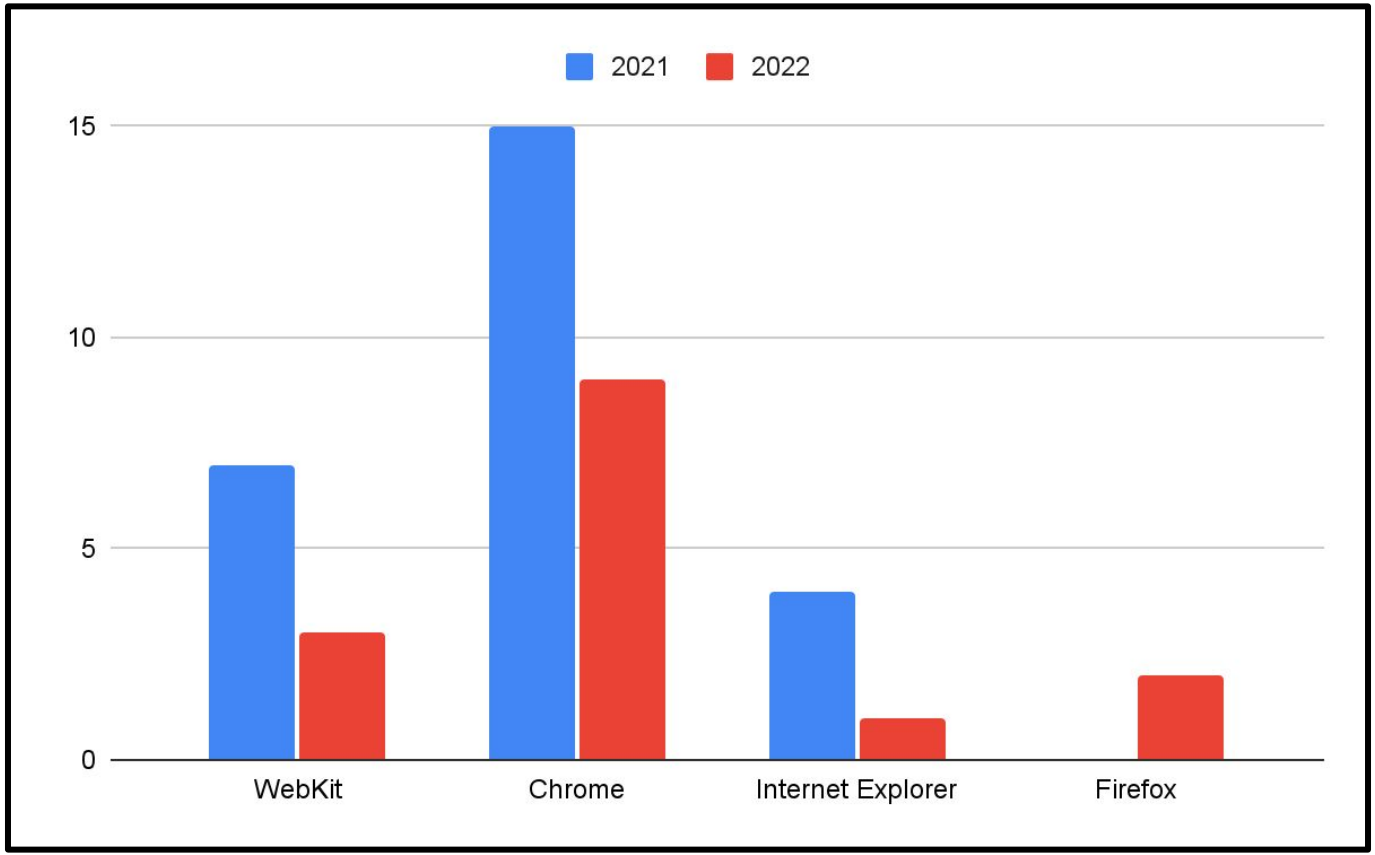
Includes the 2 Android and the Samsung bug


No significant change since 2021
10
8
6
4
2
0
Android
iOS
macOS
Windows
Exchange Server
2021
2022

Were n-days sufficient?



ProxyNotShell was patched in November



2020: Watering hole attacks

2021: One-time links

2022: 0-clicks or messaging app bugs?

New Surveys Show Burnout Is An International Crisis

2022 TRENDS-REPORT

Burnout and stress are everywhere

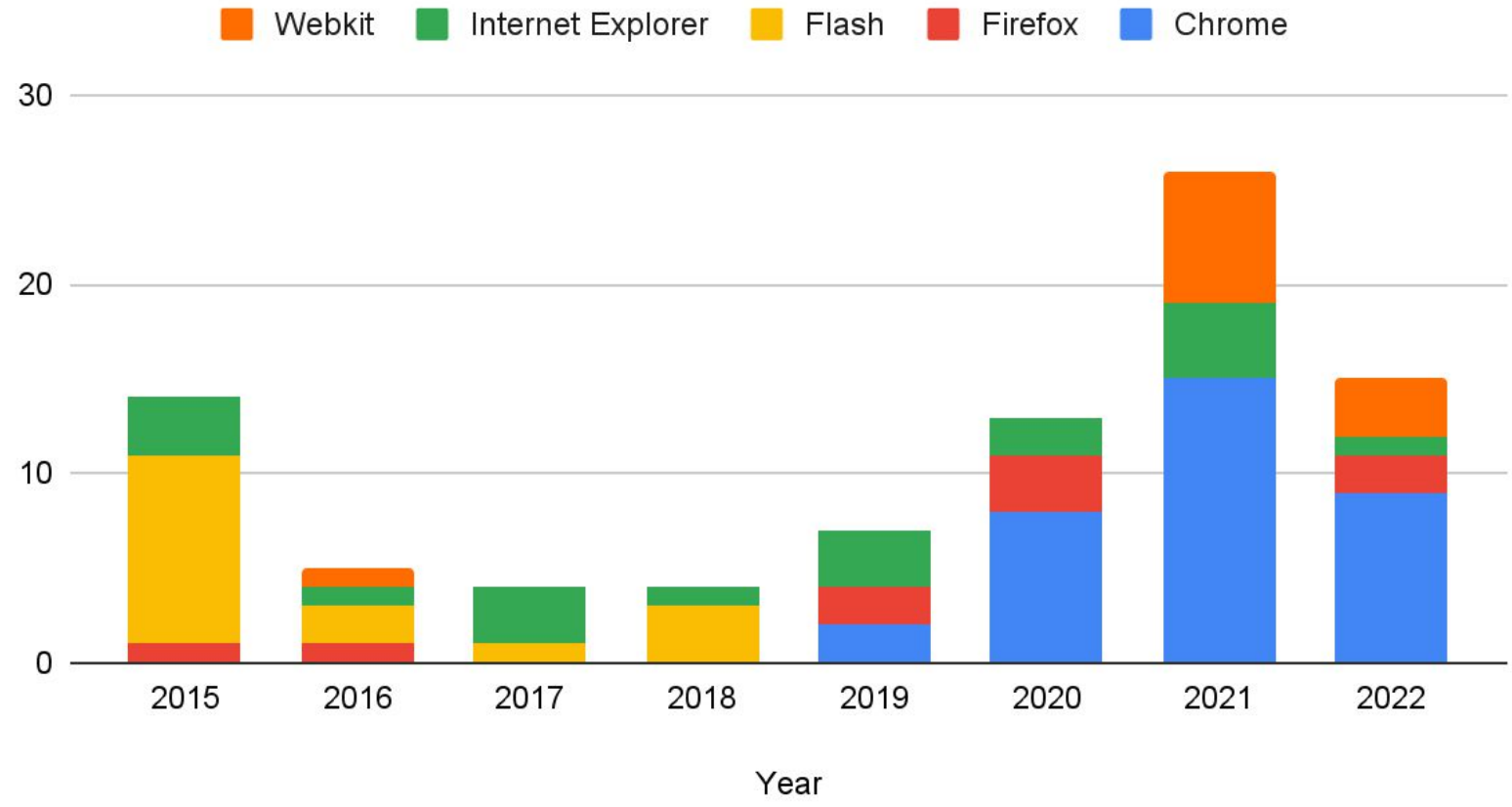
Burnout Levels Are Higher Than Ever — So Why Is No One Listening?

Burnout was supposed to get better. It hasn't.

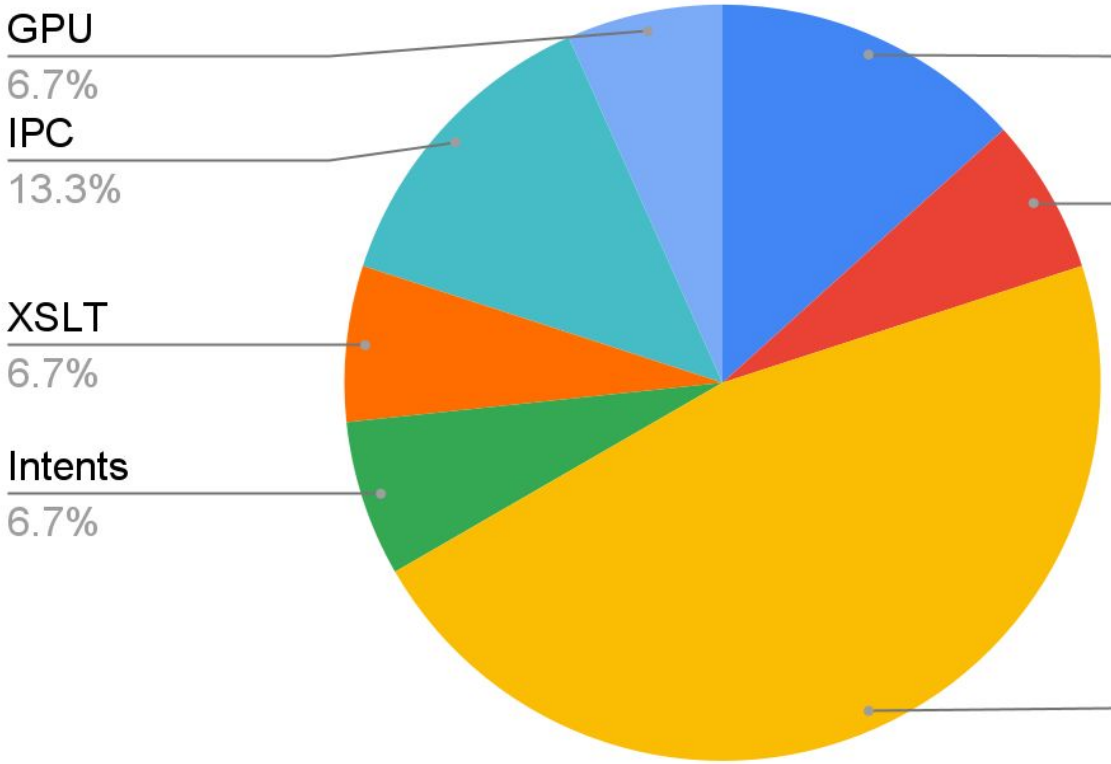
It's not just you: Almost half of American office workers feel burned out at work.

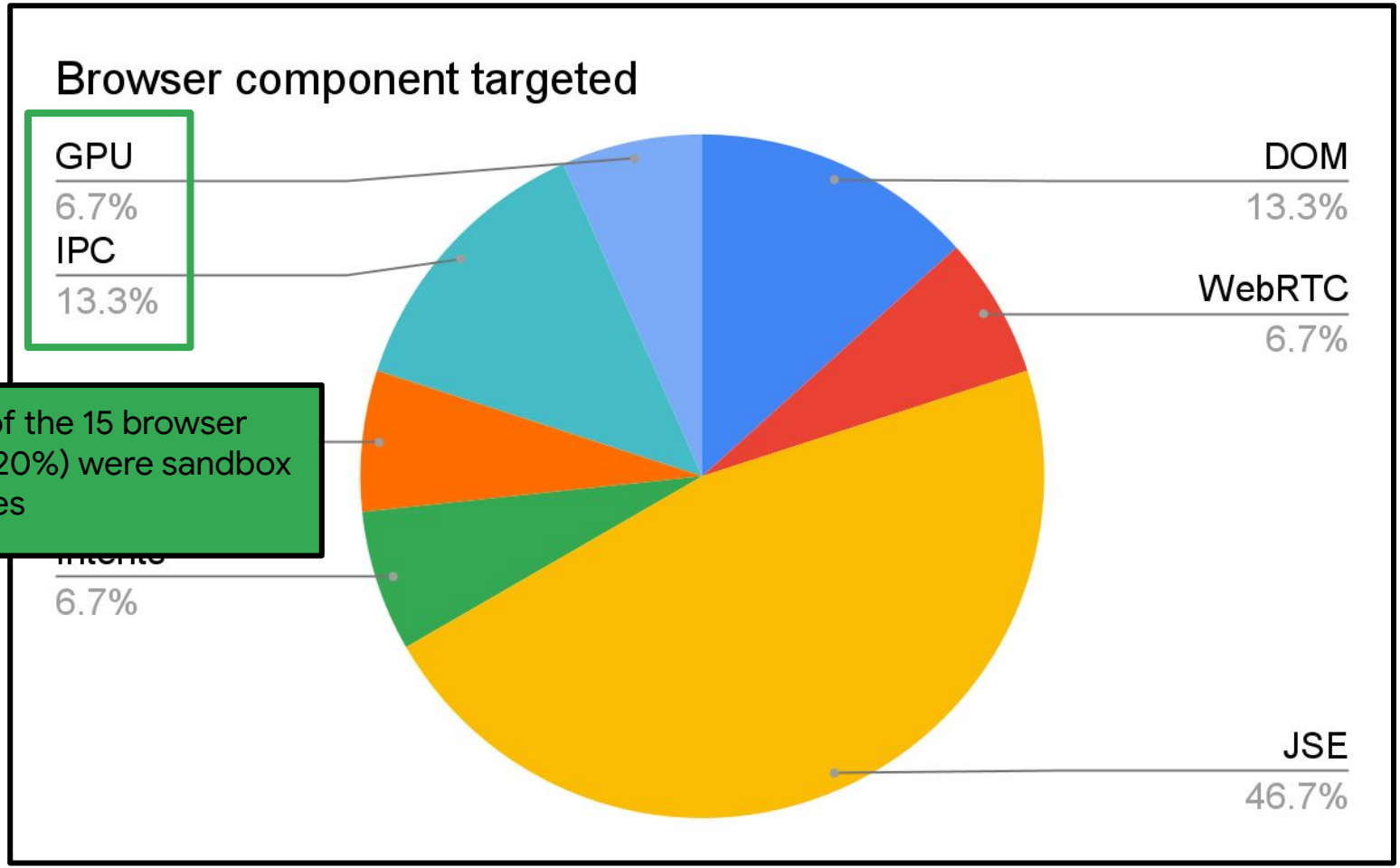
Browsers

Detected itw 0-days targeting browsers

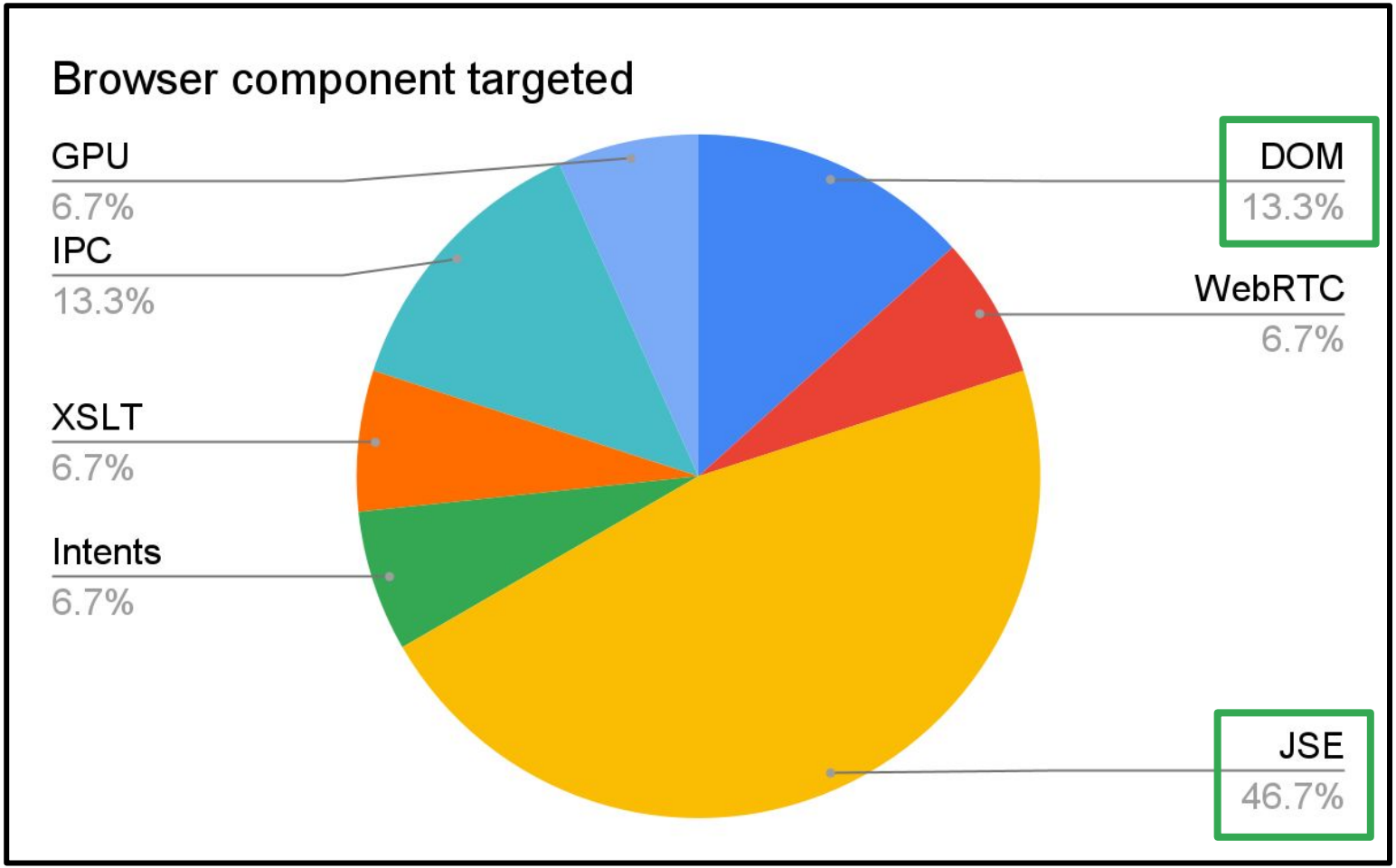


Browser component targeted





3 out of the 15 browser bugs (20%) were sandbox escapes


Browser component targeted
GPU
6.7%
IPC
13.3%
XSLT
6.7%
Intents
6.7%
DOM
13.3%
WebRTC
6.7%
JSE
46.7%

Use-after-freedom: MiraclePtr

V8 Sandbox

Aka. “Ubercage”

[libc++] Enable assertions on all builds and add a handler for release builds

Enabling assertions improves security in libc++ by guarding against undefined behaviors (e.g. accessing an element out of bounds in std::vector).

How Lockdown Mode protects your device

When Lockdown Mode is enabled, some apps and features will function differently, including:

- Messages - Most message attachment types are blocked, other than certain images, video, and audio. Some features, such as links and link previews, are unavailable.
- Web browsing - Certain complex web technologies are blocked, which might cause some websites to load more slowly or not operate correctly. In addition, web fonts might not be displayed, and images might be replaced with a missing image icon.

WebAssembly and Back Again: Fine-Grained Sandboxing in Firefox 95

Chrome: CVE-2022-2294

Buffer overflow in WebRTC

https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2022/CVE-2022-2294.html

Google

SDP Munging

```
function sdp_munge(offer) {
    let sdp = offer.sdp;
    // remove one of the send_codecs_ from the offer
    sdp = sdp.replace(/\r?\na=rid:(.+)\s+send\r?\na=simulcast:send\s+.+;\1/, '');
    offer.sdp = sdp;
    return offer;
}
async function trigger(pc) {
    var pc = createConnection();
    var offer = await pc.createOffer();
    var munged_offer = sdp_munge(offer);
    await pc.setLocalDescription(munged_offer);
}
```

SDP Munging

```
function sdp_munge(offer) {
    let sdp = offer.sdp;
    // remove one of the send_codecs_ from the offer
    sdp = sdp.replace(/\r?\na=rid:(.+)\s+send\r?\na=simulcast:send\s+.+;\1/, '');
    offer.sdp = sdp;
    return offer;
}
async function trigger(pc) {
    var pc = createConnection();
    var offer = await pc.createOffer();
    var munged_offer = sdp_munge(offer);
    await pc.setLocalDescription(munged_offer);
}
```

Using DCHECK to ensure that the size of
current_parameters.encodings >= init_parameters.encodings

```
RtpParameters current_parameters =
    media_channel_->GetRtpSendParameters(ssrc_);
RTC_DCHECK_GE(current_parameters.encodings.size(),
              init_parameters_.encodings.size());
for (size_t i = 0; i < init_parameters_.encodings.size(); ++i) {
  init_parameters_.encodings[i].ssrc = current_parameters.encodings[i].ssrc;
  init_parameters_.encodings[i].rid = current_parameters.encodings[i].rid;
  current_parameters.encodings[i] = init_parameters_.encodings[i];
}
```

```
RtpParameters current_parameters =
    media_channel_->GetRtpSendParameters(ssrc_);
RTC_DCHECK_GE(current_parameters.encodings.size(),
              init_parameters_.encodings.size());
for (size_t i = 0; i < init_parameters_.encodings.size(); ++i) {
  init_parameters_.encodings[i].ssrc = current_parameters.encodings[i].ssrc;
  init_parameters_.encodings[i].rid = current_parameters.encodings[i].rid;
  current_parameters.encodings[i] = init_parameters_.encodings[i];
}
```

Out of bounds reads on
current_parameters.encodings

```
RtpParameters current_parameters =
    media_channel_->GetRtpSendParameters(ssrc_);
RTC_DCHECK_GE(current_parameters.encodings.size(),
              init_parameters_.encodings.size());
for (size_t i = 0; i < init_parameters_.encodings.size(); ++i) {
  init_parameters_.encodings[i].ssrc = current_parameters.encodings[i].ssrc;
  init_parameters_.encodings[i].rid  = current_parameters.encodings[i].rid;
  current_parameters.encodings[i] = init_parameters_.encodings[i];
}
```

Out of bounds write on
current_parameters.encodings

```
RtpParameters current_parameters =
    media_channel_->GetRtpSendParameters(ssrc_);
RTC_DCHECK_GE(current_parameters.encodings.size(),
              init_parameters_.encodings.size());
for (size_t i = 0; i < init_parameters_.encodings.size(); ++i) {
  init_parameters_.encodings[i].ssrc = current_parameters.encodings[i].ssrc;
  init_parameters_.encodings[i].rid  = current_parameters.encodings[i].rid;
  current_parameters.encodings[i] = init_parameters_.encodings[i];
}
```

Changed DCHECK to CHECK

```
RtpParameters current_parameters =
    media_channel_->GetRtpSendParameters(ssrc_);
RTC_CHECK_GE(current_parameters.encodings.size(),
             init_parameters_.encodings.size());
for (size_t i = 0; i < init_parameters_.encodings.size(); ++i) {
  init_parameters_.encodings[i].ssrc = current_parameters.encodings[i].ssrc;
  init_parameters_.encodings[i].rid  = current_parameters.encodings[i].rid;
  current_parameters.encodings[i] = init_parameters_.encodings[i];
}
```

Chrome: CVE-2022-3075

Insufficient data validation in Mojo

Google

```
template <typename T>
bool ValidateResponseGenericT(Message* message,const char* class_name, base::span<const T> info) {

  if (!message->is_serialized() || ControlMessageHandler::IsControlMessage(message)) {
    return true;
  }
  ValidationContext validation_context(message, class_name,ValidationContext::kResponseValidator);

  if (!ValidateMessageIsResponse(message, &validation_context))
    return false;

  auto entry = FindGenericValidationInfo(message->header()->name, info);

  if (!entry.response_validator) {
    ReportValidationError(&validation_context, VALIDATION_ERROR_MESSAGE_HEADER_UNKNOWN_METHOD);
    return false;
  }
  return entry.response_validator(message->payload(), &validation_context);
}
```

https://source.chromium.org/chromium/chromium/src/+/456a79d6c2bd7fa8f7d63334a581a749a2f560f4:mojo/public/cpp/bindings/lib/generated_code_util.cc;l=67

```
template <typename T>
bool ValidateResponseGenericT(Message* message,const char* class_name, base::span<const T> info) {

  if (!message->is_serialized() || ControlMessageHandler::IsControlMessage(message)) {
    return true;
  }
  ValidationContext validation_context(message, class_name,ValidationContext::kResponseValidator);

  if (!ValidateMessageIsResponse(
    return false;

  auto entry = FindGenericValidationInfo(message->header()->name, info);

  if (!entry.response_validator) {
    ReportValidationError(&validation_context, VALIDATION_ERROR_MESSAGE_HEADER_UNKNOWN_METHOD);
    return false;
  }
  return entry.response_validator(message->payload(), &validation_context);
}
```

Uses the name in the message header to find the correct validation function.

https://source.chromium.org/chromium/chromium/src/+/456a79d6c2bd7fa8f7d63334a581a749a2f560f4:mojo/public/cpp/bindings/lib/generated_code_util.cc;l=67

```
bool InterfaceEndpointClient::HandleValidatedMessage(Message* message) {
[...]
  } else if (message->has_flag(Message::kFlagIsResponse)) {
    uint64_t request_id = message->request_id();
[...]
    std::unique_ptr<MessageReceiver> responder;
    {
      base::AutoLock lock(async_responders_lock_);
      auto it = async_responders_.find(request_id);
      if (it == async_responders_.end())
    return false;
      responder = std::move(it->second);
      async_responders_.erase(it);
    }

    internal::MessageDispatchContext dispatch_context(message);
    return responder->Accept(message);
[...]
}
```

https://source.chromium.org/chromium/chromium/src/+/456a79d6c2bd7fa8f7d63334a581a749a2f560f4:mojo/public/cpp/bindings/lib/interface_endpoint_client.cc;l=851?q=mojo%2Fpublic%2Fcpp%2Fbindings%2Flib%2Finterface_endpoint_client.cc&ss=chromium

```
bool InterfaceEndpointClient::HandleValidatedMessage(Message* message) {
[...]
  } else if (message->has_flag(Message::kFlagIsResponse)) {
    uint64_t request_id = message->request_id();
[...]
    std::unique_ptr<MessageReceiver> responder;
    {
      base::AutoLock lock(async_responders_lock_);
      auto it = async_responders_.find(request_id);
      if (it == async_responders_.end())
    return false;
      responder = std::move(it->second);
      async_responders_.erase(it);
    }

    internal::MessageDispatchCont
    return responder->Accept(message);
[...]
}
```

Uses the message request_id to look up the correct responder

https://source.chromium.org/chromium/chromium/src/+/456a79d6c2bd7fa8f7d63334a581a749a2f560f4:mojo/public/cpp/bindings/lib/interface_endpoint_client.cc;l=851?q=mojo%2Fpublic%2Fcpp%2Fbindings%2Flib%2Finterface_endpoint_client.cc&ss=chromium

If you don’t want mojo to check the contents of your message...

If you don’t want mojo to check the contents of your message...

```
message.header()->name = 11;
```

```
message.header()->name = 11;
```

```
bool WidgetInputHandler_WaitForInputProcessed_ResponseParams_Data::Validate(
        const void* data, mojo::internal::ValidationContext* validation_context) {
[...]
 if (!ValidateUnversionedStructHeaderAndSizeAndClaimMemory(data, 8, validation_context)) {
   return false;
 }
[...]
}

bool ValidateUnversionedStructHeaderAndSizeAndClaimMemory(const void* data, size_t v0_size,
        ValidationContext* validation_context) {
  [...]
  const auto& header = *static_cast<const StructHeader*>(data);
  if ((header.version == 0 &&
       header.num_bytes != v0_size) ||
       header.num_bytes < v0_size) {
    ReportValidationError(validation_context, VALIDATION_ERROR_UNEXPECTED_STRUCT_HEADER);
    return false;
  }
  return true;
}
```

```
bool WidgetInputHandler_WaitForInputProcessed_ResponseParams_Data::Validate(
        const void* data, mojo::internal::ValidationContext* validation_context) {
[...]
 if (!ValidateUnversionedStructHeaderAndSizeAndClaimMemory(data, 8, validation_context)) {
```

```
auto *struct_header =
message.payload_buffer()->Get<StructHeader>(message.header()->num_bytes);
struct_header->version = 1;
```

```
bool ValidateUnversionedStructHeaderAndSizeAndClaimMemory(const void* data, size_t v0_size,
        ValidationContext* validation_context) {
  [...]
  const auto& header = *static_cast<const StructHeader*>(data);
  if ((header.version == 0 &&
       header.num_bytes != v0_size) ||
       header.num_bytes < v0_size) {
    ReportValidationError(validation_context, VALIDATION_ERROR_UNEXPECTED_STRUCT_HEADER);
    return false;
  }
  return true;
}
```

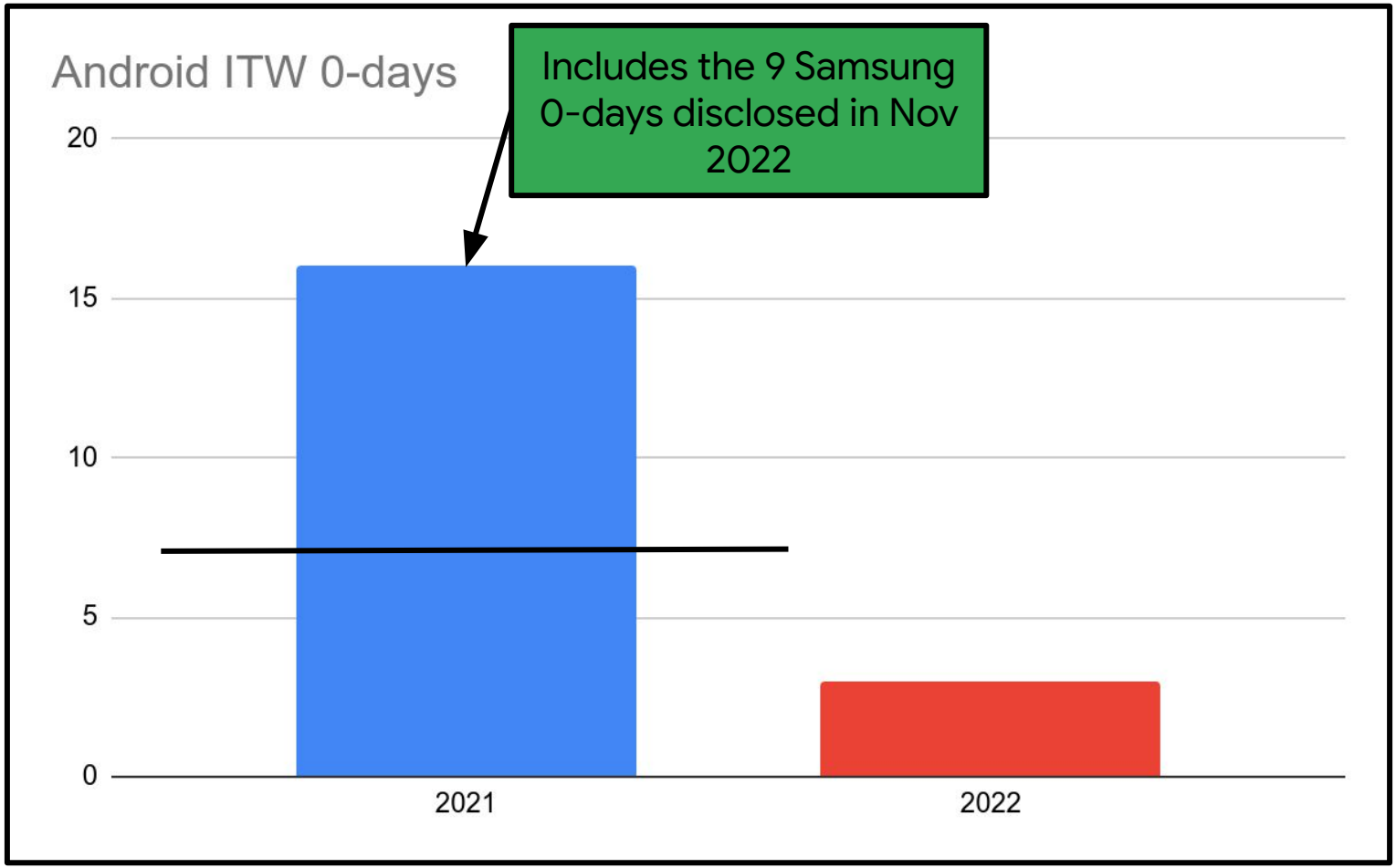
```
bool WidgetInputHandler_DispatchEvent_ResponseParams_Data::Validate(
    const void* data, mojo::internal::ValidationContext* validation_context) {
  if (!data)
    return true;
  if (!ValidateUnversionedStructHeaderAndSizeAndClaimMemory(data, 48, validation_context))
    return false;
[...]
  if (!::blink::mojom::internal::InputEventResultSource_Data::Validate(object->source, validation_context))
    return false;
  if (!mojo::internal::ValidatePointerNonNullable(object->updated_latency, 2, validation_context))
    return false;
  if (!mojo::internal::ValidateStruct(object->updated_latency, validation_context))
    return false;
  if (!::blink::mojom::internal::InputEventResultState_Data::Validate(object->state, validation_context))
    return false;
  if (!mojo::internal::ValidateStruct(object->overscroll, validation_context))
    return false;
  if (!mojo::internal::ValidateStruct(object->touch_action, validation_context))
    return false;
  if (!mojo::internal::ValidateStruct(object->scroll_result_data, validation_context))
    return false;
  return true;
}
```

“The biggest challenge with this bug is the exploitation options are endless.
How do you choose?”

- Mark Brand

Consumer Platforms

Android


Android ITW 0-days
Includes the 9 Samsung 0-days disclosed in Nov 2022
20
15
10
5
0
2021
2022

CVE-2022-22265: Double free in Samsung NPU driver

CVE-2021-22600: Double free in Linux kernel

CVE-2021-39793: Out of bounds write in Mali GPU

Mind the Gap

By Ian Beer, Project Zero

Note: The vulnerabilities discussed in this blog post (CVE-2022-33917) are fixed by the upstream vendor, but at the time of publication, these fixes have not yet made it downstream to affected Android devices (including Pixel, Samsung, Xiaomi, Oppo and others). Devices with a Mali GPU are currently vulnerable.

Android Exploit Chain

The Android exploit chain targeted users on phones with an ARM GPU running Chrome versions prior to 106. It consisted of three exploits, including one 0-day:

- CVE-2022-3723, a type confusion vulnerability in Chrome, found by Avast in the wild and fixed in October 2022 in version 107.0.5304.87.
- CVE-2022-4135, a Chrome GPU sandbox bypass only affecting Android (0-day at time of exploitation), fixed in November 2022. Sergei Glazunov from Project Zero helped analyze the exploit and wrote a root cause analysis for this bug.
- CVE-2022-38181, a privilege escalation bug fixed by ARM in August 2022. It is unclear if attackers had an exploit for this vulnerability before it was reported to ARM.

https://googleprojectzero.blogspot.com/2023/03/multiple-internet-to-baseband-remote-rce.html
https://blog.google/threat-analysis-group/spyware-vendors-use-0-days-and-n-days-against-popular-platforms/

GHSL-2022-054: Use-after-free (UAF) in the Arm Mali Kernel driver - CVE-2022-38181

Man Yue Mo

Coordinated Disclosure Timeline

- 2022-07-12: Issue reported to Android security team as ticket 238770628 with proof of concepts to trigger the bug.
- 2022-07-13: Issue assigned internal Android ID 238863570
- 2022-07-14: Exploit for arbitrary kernel code execution and rooting of Pixel 6 from untrusted app shared with Android Security Team.
- 2022-07-15: Android security team replied and said they were unable to disable SELinux with the exploit.
- 2022-07-16: From the replied, it is unclear to me what the problem is, other than a firmware mismatch that causes the exploit to exit at an early stage, so I clarified the exact version of the firmware image used in the exploit and also included an updated file with extra debugging information.
- 2022-07-20: Android security team rated the vulnerability with "High" severity.
- 2022-08-04: Android security team decided that the issue is "device-specific" and labelled it as "Won't Fix"[1]
- 2022-08-04: I asked Android security team: 1. To confirm if the issue is out-of-scope even if it affects Pixel 6 and if they believe it is a security issue and 2. Whether they can pass my contact details to Arm and give me a reference number so I can track the progress with Arm
- 2022-08-05: Reach out to Arm security to enquire about the bug report.
- 2022-08-09: Android security team confirms that the issue is out-of-scope without giving any explanations, and ignored my request to be included in the communications with Arm[2]
- 2022-08-15: Received reply from Arm, saying that they agree the issue is a security vulnerability and that a CVE ID will be assigned to the issue.
- 2022-09-05: Shared our disclosure policy with Arm with a reminder of the disclosure deadline of 2022-10-10.
- 2022-09-08: Arm asked for a 30 day extension to the deadline, stating that a patch should be release near the deadline. The requested extension was granted, with an agreed disclosure date of around mid November.
- 2022-10-03: CVE-2022-38181 was assigned to the issue.
- 2022-10-07: The Arm driver version r40p0 was released to address the issue.
- 2022-10-31: Notify both Arm and Android of our planned disclosure date of 2022-11-23.
- 2023-01-05: The issue appeared to have been fixed in Pixel phones in the January update as bug: 259695958. However, the is no mentioning of this bug ID, the original bug ID (238770628) nor the CVE ID (CVE-2022-38181) associated with this issue in the bulletin.

- July 2022: Reported to Android Security team
- Aug 2022: Bug fixed by ARM
- Nov 2022: In-the-wild exploit discovered
- Jan 2023: Silently patched in Pixel devices
- April 2023: Included in Android Security Bulletin

https://securitylab.github.com/advisories/GHSL-2022-054_Arm_Mali/

Samsung: CVE-2022-22265

Double free in NPU

https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2022/CVE-2022-22265.html

Initialization

```
format_list = (vs4l_format_list *)malloc(0x10uLL);
  if ( !format_list )
    return -10;
  format_structs = (vs4l_format *)calloc(format_cnt,
0x24uLL);
  if ( !format_structs )
    return -10;
  for ( i = 0; i < format_cnt; ++i )
  {
    format_structs[i].target = 0;
    format_structs[i].format = '2BGR';
    format_structs[i].plane = 0;
    format_structs[i].width = 22;
    format_structs[i].height = 1;
    format_structs[i].stride = 0;
    format_structs[i].cstride = 0;
    format_structs[i].channels = 1;
    format_structs[i].pixel_format = 8;
  }
  format_list->direction = 1;
  format_list->count = format_cnt;
  format_list->formats = format_structs;
```

```
allocate_and_init_ION(npu_fd, format_cnt);

// Call IOCTL with VS4L_DIRECTION_IN to initialize in
queue
ioctl(npu_fd, VS4L_VERTEXIOC_S_FORMAT, format_list);

// Call IOCTL with VS4L_DIRECTION_OUT to initialize out
queue
format_list->direction = 2
ioctl(npu_fd, VS4L_VERTEXIOC_S_FORMAT, format_list);

// Begin process in and out queues
ioctl(npu_fd, VS4l_VERTEXIOC_STREAMON);
```

Initialization

```
format_list = (vs4l_format_list *)malloc(0x10uLL);
  if ( !format_list )
    return -10;
  format_structs = (vs4l_format *)calloc(format_cnt,
0x24uLL);
  if ( !format_structs )
    return -10;
  for ( i = 0; i < format_cnt; ++i )
  {
    format_structs[i].target = 0;
    format_structs[i].format = '2BGR';
    format_structs[i].plane = 0;
    format_structs[i].width = 22;
    format_structs[i].height = 1;
    format_structs[i].stride = 0;
    format_structs[i].cstride = 0;
    format_structs[i].channels = 1;
    format_structs[i].pixel_format = 8;
  }
  format_list->direction = 1;
  format_list->count = format_cnt;
  format_list->formats = format_structs;
```

```
struct vs4l_format {
      __u32             target;
      __u32             format;
      __u32             plane;
      __u32             width;
      __u32             height;
      __u32             stride;
      __u32             cstride;
      __u32             channels;
      __u32             pixel_format;
};

struct vs4l_format_list {
      __u32                     direction;
      __u32                     count;
      struct vs4l_format        *formats;
};
```

Trigger first free

```
format_structs->format = 0;
ret = ioctl(npu_fd, VS4L_VERTEXIOC_S_FORMAT, format_list);
```

Trigger first free

```
format_structs->format = 0;
ret = ioctl(npu_fd, VS4L_VERTEXIOC_S_FORMAT, format_list);
```

Set format member in first format entry to NULL (instead of “2BGR”)

Trigger first free

```
format_structs->format = 0;
ret = ioctl(npu_fd, VS4L_VERTEXIOC_S_FORMAT, format_list);
```

Calling `VS4L_VERTEXIOC_S_FORMAT` again, but this time with an invalid `format_list`

```
int vb_queue_s_format(struct vb_queue *q, struct vs4l_format_list *flist)
{
     int ret = 0;
     u32 i;
     struct vs4l_format *f;
     struct vb_fmt *fmt;
     q->format.count = flist->count;
     q->format.formats = kcalloc(flist->count, sizeof(struct vb_format), GFP_KERNEL);
[...]
    for (i = 0; i < flist->count; ++i) {
        f = &flist->formats[i];
        fmt = __vb_find_format(f->format);
        if (!fmt) {
           vision_err("__vb_find_format is fail\n");
           kfree(q->format.formats);
           ret = -EINVAL;
           goto p_err;
        }
     [...]
}
```

```
int vb_queue_s_format(struct vb_queue *q, struct vs4l_format_list *flist)
{
     int ret = 0;
     u32 i;
     struct vs4l_format *f;
     struct vb_fmt *fmt;
     q->format.count = flist->count;
     q->format.formats = kcalloc(flist->count, sizeof(struct vb_format), GFP_KERNEL);
[...]
    for (i = 0; i < flist->count; ++i) {
        f = &flist->formats[i];
        fmt = __vb_find_format(f->format);
        if (!fmt) {
           vision_err("__vb_find_format is fail\n");
           kfree(q->format.formats);
           ret = -EINVAL;
           goto p_err;
        }
     [...]
}
```

```
int vb_queue_s_format(struct vb_queue *q, struct vs4l_format_list *flist)
{
     int ret = 0;
     u32 i;
     struct vs4l_format *f;
     struct vb_fmt *fmt;
     q->format.count = flist->cou
     q->format.formats = kcalloc(f
[...]
    for (i = 0; i < flist->count; ++i) {
        f = &flist->formats[i];
        fmt = __vb_find_format(f->format);
        if (!fmt) {
           vision_err("__vb_find_format is fail\n");
           kfree(q->format.formats);
           ret = -EINVAL;
           goto p_err;
        }
     [...]
}
```

f->format is the value that we set to NULL, an invalid value

```
int vb_queue_s_format(struct vb_queue *q, struct vs4l_format_list *flist)
{
     int ret = 0;
     u32 i;
     struct vs4l_format *f;
     struct vb_fmt *fmt;
     q->format.count = flist->cou
     q->format.formats = kcalloc(flist->count, sizeof(struct vb_format), GFP_KERNEL);
[...]
    for (i = 0; i < flist->count; ++i) {
        f = &flist->formats[i];
        fmt = __vb_find_format(f->format);
        if (!fmt) {
            vision_err("__vb_find_format is fail\n");
            kfree(q->format.formats);
            ret = -EINVAL;
            goto p_err;
        }
     [...]
}
```

__vb_find_format(f->format) will fail

```
int vb_queue_s_format(struct vb_queue *q, struct vs4l_format_list *flist)
{
     int ret = 0;
     u32 i;
     struct vs4l_format *f;
     struct vb_fmt *fmt;
     q->format.count = flist->cou
     q->format.formats = kcalloc(flist->count, sizeof(struct vb_format), GFP_KERNEL);
[...]
    for (i = 0; i < flist->count; ++i) {
        f = &flist->formats[i];
        fmt = __vb_find_format(f->format);
        if (!fmt) {
           vision_err("  vb find format is fail\n");
           kfree(q->format.formats);
           ret = -EINVAL;
           goto p_err;
        }
     [...]
}
```

Free #1: kfree(q->format.formats)

Trigger second free

```
ret = ioctl(npu_fd, VS4L_VERTEXIOC_STREAM_OFF);
```

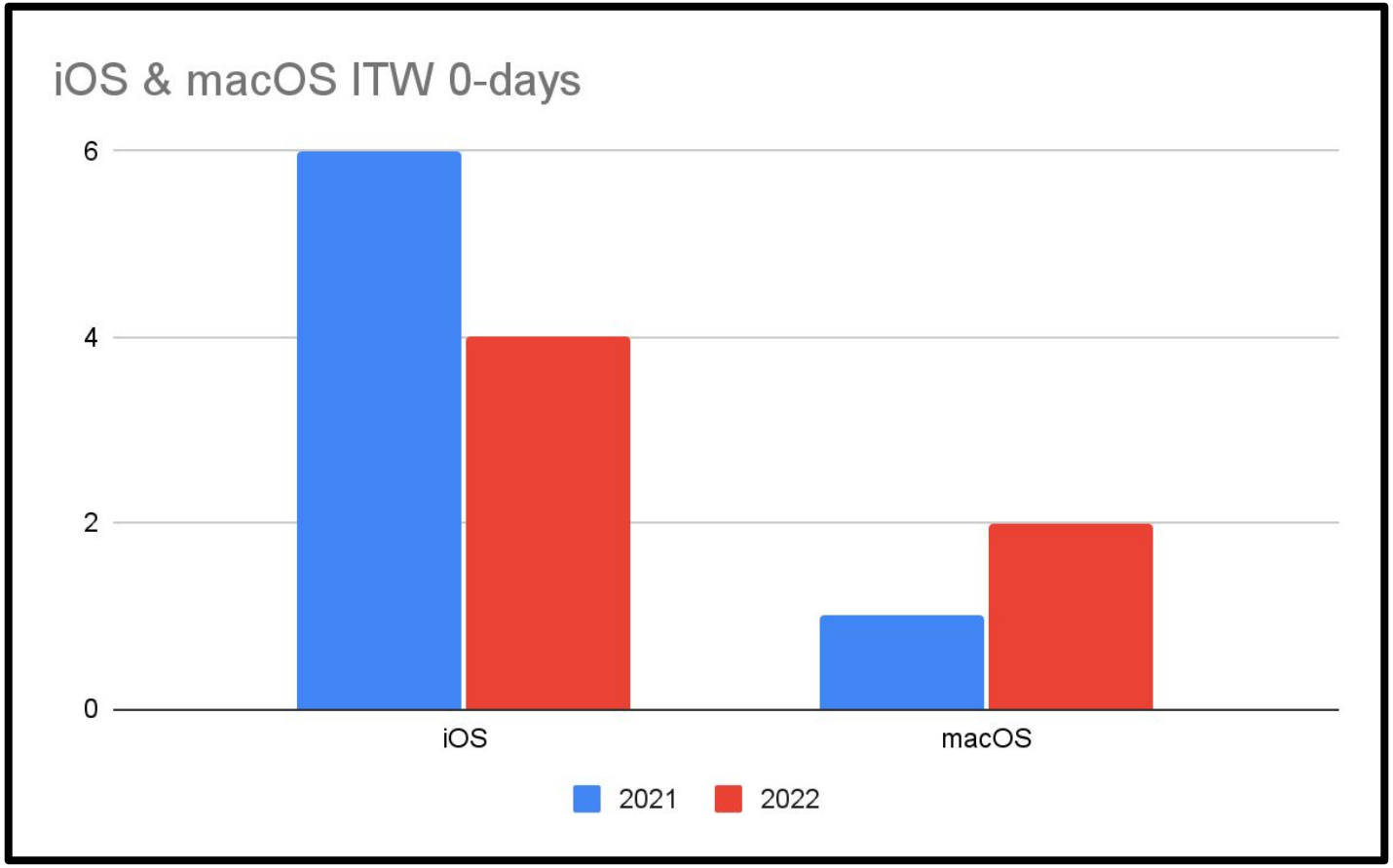
```
static int npu_vertex_streamoff(struct file *file)
{
    int ret = 0;
    struct npu_vertex_ctx *vctx = file->private_data;
    struct npu_vertex *vertex = vctx->vertex;
    struct npu_queue *queue = &vctx->queue;
    struct mutex *lock = &vctx->lock;
    struct npu_session *session = container_of(vctx, struct npu_session, vctx);

[...]

    ret = npu_queue_stop(queue, 0);

    [...]

}
```

```
int npu_queue_stop(struct npu_queue *queue, int is_forced)
{
        int ret = 0;
        struct vb_queue *inq, *otq;
        inq = &queue->inqueue;
        otq = &queue->otqueue;
[...]
        if (!is_forced)
                ret = vb_queue_stop(inq);
        else
                ret = vb_queue_stop_forced(inq);

        if (ret) {
                npu_err("fail(%d) in vb_queue_stop%s(inq)\n", ret,
(is_forced)?"_forced":"");
                goto p_err;
        }

        if (!is_forced)
                ret = vb_queue_stop(otq);
        [...]
```

```
static int __vb_queue_stop(struct vb_queue *q, int is_forced)
{
        int ret = 0;
        u32 i;
        struct vb_bundle

        __vb_queue_clear(q);

        q->streaming = 0;
        wake_up_all(&q->done_wq);
        [...]

        kfree(q->format.formats);

        [...]
}
```

Free #2: kfree(q->format.formats)

iOS & macOS



- CVE-2022-22587 - iOS & macOS, IOMobileFrameBuffer
- CVE-2022-22674 - macOS only, Intel Graphics Driver
- CVE-2022-22675 - macOS & iOS, AppleAVD
- CVE-2022-32894 - macOS & iOS, Apple Firmware Kit
- CVE-2022-32917 - macOS & iOS, AppleSPU
- CVE-2022-42827 - iOS only

- CVE-2022-22587 - iOS & macOS, IOMobileFrameBuffer
- CVE-2022-22674 - macOS only, Intel Graphics Driver
- CVE-2022-22675 - macOS & iOS, AppleAVD
- CVE-2022-32894 - macOS & iOS, Apple Firmware Kit
- CVE-2022-32917 - macOS & iOS, AppleSPU
- CVE-2022-42827 - iOS only

2021

3 IOMobileFrameBuffer

2 XNU Kernel

1 Core Graphics

1 CommCenter

2022

1 IOMobileFrameBuffer

1 AppleAVD

1 AppleFirmwareKit

1 AppleSPU

1 Intel Graphics Driver

1 ???

iOS/macOS: CVE-2022-22675

Out of bounds write in AppleAVD

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

CVE-2022-22675 Detail

Description

An out-of-bounds write issue was addressed with improved bounds checking. This issue is fixed in tvOS 15.5, watchOS 8.6, macOS Big Sur 11.6.6, macOS Monterey 12.3.1, iOS 15.4.1 and iPadOS 15.4.1. An application may be able to execute arbitrary code with kernel privileges. Apple is aware of a report that this issue may have been actively exploited..

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 7.8 HIGH **Vector:** CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

CVE-2022-22675 Detail

Description

An out-of-bounds write issue was addressed with improved bounds checking. This issue is fixed in tvOS 15.5, watchOS 8.6, macOS Big Sur 11.6.6, macOS Monterey 12.3.1, iOS 15.4.1 and iPadOS 15.4.1. An application may be able to execute arbitrary code with kernel privileges. Apple is aware of a report that this issue may have been actively exploited..

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD

Base Score: 7.8 HIGH

Vector: CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

CVSS:3.1/**AV:L**/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

Vulnerability details:

There is a buffer overflow when processing the Hardware Reference Device (HRD) of an H.264 stream in the function *AVC_RBSP::parseHRD*. The AppleAVD.kext kernel module reads values describing the bitrates of the HRD from the stream in a loop and copies them into a buffer. This buffer has a fixed size of 32 elements, meanwhile the number of elements copied is determined by the *cpb_cnt_minus1* value read from the stream, which can have a maximum value of 255, allowing the buffer to be overflowed.

Note that while the advisories describe the impact of this issue as a local privilege escalation, it is theoretically possible to exploit it to achieve fully-remote code execution in MacOS 12.3/iOS 15.4. These versions use AppleAVD to perform thumbnailing of incoming images in iMessage, so this code path is available to a fully-remote attacker.

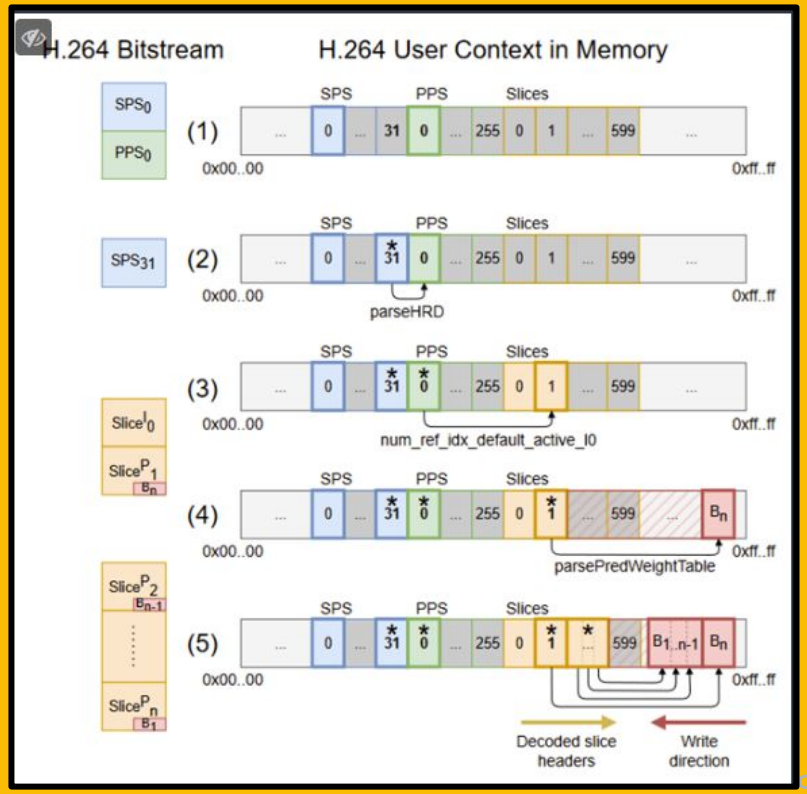
https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

wrv
@wrv@infosec.exchange

Returning to the CVE-2022-22675 RCA, our starting point was that an out-of-bounds cpb_cnt_minus1 can overwrite other members in the decoder struct.

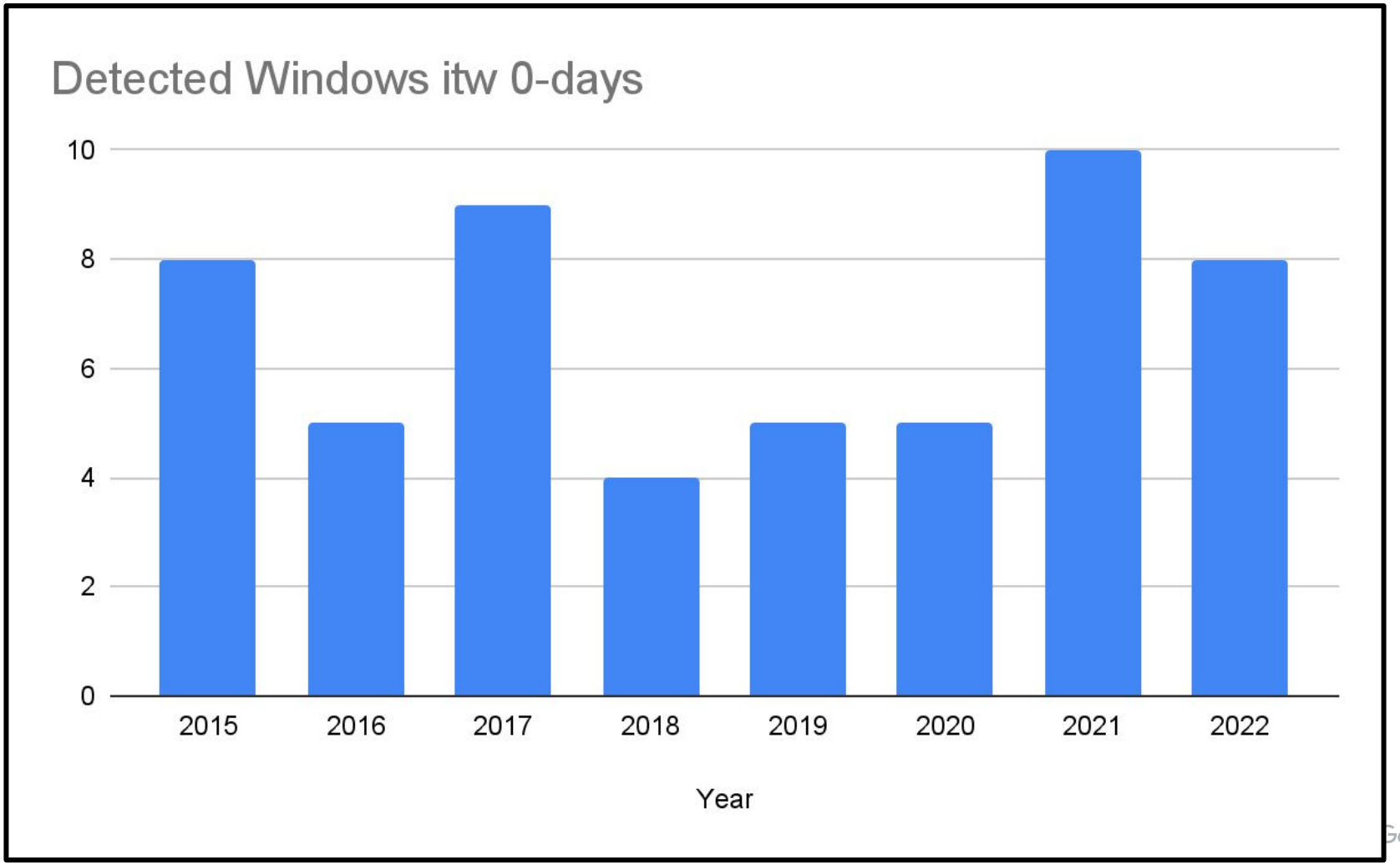
Given our familiarity with H.264, we found that overwriting the syntax element that controls the number of reference pictures could likely lead to a second overflow.

At a high level, we achieve a heap overflow by:
1. Ordering the bitstream so that structs we want to overflow are in memory.
2. Triggering CVE-2022-22675 to overwrite the number of reference pictures.
3. Using the overwritten number of reference pictures in a Slice that calls parsePredWeightTable
4. Injecting an intentional decoding failure at the point we want to stop writing.
5. Using multiple slices to write arbitrary length values at an attacker chosen offset.



https://infosec.exchange/@wrv/110081609346902517
https://wrv.github.io/h26forge.pdf

Windows


Detected Windows itw 0-days
10
8
6
4
2
0
2015
2016
2017
2018
2019
2020
2021
2022
Year

- 1 win32k
- 1 Windows Common Log File System (CLFS)
- 1 LSA Spoofing
- 1 Microsoft Windows Support Diagnostic Tool (MSDT)
- 1 CSRSS
- 1 COM+
- 1 Print Spooler
- 1 CNG Key Isolation Service

- 1 win32k
- 1 CLFS
- 1 LSA Spoofing
- 1 MSDT
- 2 CSRSS
- 1 COM+
- ~~1 Print spooler~~
- 1 CNG Key Isolation Service

excluding “Mark of the Web”

- 1 win32k
- 1 CLFS
- 1 LSA Spoofing
- 1 MSDT
- 2 CSRSS
- 1 COM+
- ~~1 Print spooler~~
- 1 CNG Key Isolation Service

Windows: CVE-2022-22047

```
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity name="SomePrivilegedExecutable" processorArchitecture="amd64" version="1.0.0.0" type="win32"/>
<description>Windows Shell</description>
  <file loadFrom="c:\repro\payload.dll" name="advapi32.dll"/>
</assembly>
```

https://www.zerodayinitiative.com/blog/2023/1/23/activation-context-cache-poisoning-exploiting-csrss-for-privilege-escalation

```
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity name="SomePrivilegedExecutable" processorArchitecture="amd64" version="1.0.0.0" type="win32"/>
<description>Windows Shell</description>
 <file loadFrom="c:\repro\payload.dll" name="advapi32.dll"/>
</assembly>
```

https://www.zerodayinitiative.com/blog/2023/1/23/activation-context-cache-poisoning-exploiting-csrss-for-privilege-escalation

Enterprise

- 1 Atlassian Confluence
- 2 Sophos Firewall
- 1 Trend Micro Apex Central
- 1 Fortinet FortiOS
- 1 Citrix ADC/Gateway
- 2 Exchange Server

17/40 in-the-wild 0-days from 2022 are variants of previously known bugs.

Déjà vu-lnerability

A Year in Review of 0-days Exploited In-The-Wild in 2020

Posted by Maddie Stone, Project Zero

2020 was a year full of 0-day exploits. Many of the Internet's most popular browsers had their moment in the spotlight. Memory corruption is *still* the name of the game and how the vast majority of detected 0-days are getting in. While we tried new methods of 0-day detection with modest success, 2020 showed us that there is still a long way to go in detecting these 0-day exploits in-the-wild. But what may be the most notable fact is that 25% of the 0-days detected in 2020 are closely related to previously publicly disclosed vulnerabilities. In other words, **1 out of every 4 detected 0-day exploits could potentially have been avoided if a more thorough investigation and patching effort were explored.** Across the industry, incomplete patches — patches that don't correctly and comprehensively fix the root cause of a vulnerability — allow attackers to use 0-days against users with less effort.

https://googleprojectzero.blogspot.com/2021/02/deja-vu-lnerability.html
https://googleprojectzero.blogspot.com/2022/06/2022-0-day-in-wild-exploitationso-far.html

Déjà vu-lnerability

A Year in Review of 0-days Exploited In-The-Wild in 2020

Posted by Maddie Stone, Project Zero

2020 was a year full of 0-day exploits. Many of the Internet's most popular browsers had their moment in the spotlight. Memory corruption is *still* the name of the
getting in. While we tried new methods of 0-day dete
is still a long way to go in detecting these 0-day expl
that 25% of the 0-days detected in 2020 are closely
other words, **1 out of every 4 detected 0-day explo
thorough investigation and patching effort were**
patches that don't correctly and comprehensively fix
use 0-days against users with less effort.

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

Posted by Maddie Stone, Google Project Zero

This blog post is an overview of a talk, " 0-day In-the-Wild Exploitation in 2022…so far", that I gave at the FIRST conference in June 2022. The slides are available here.

For the last three years, we've published annual year-in-review reports of 0-days found exploited in the wild. The most recent of these reports is the 2021 Year in Review report, which we published just a few months ago in April. While we plan to stick with that annual cadence, we're publishing a little bonus report today looking at the in-the-wild 0-days detected and disclosed in the first half of 2022.

As of June 15, 2022, there have been 18 0-days detected and disclosed as exploited in-the-wild in 2022. When we analyzed those 0-days, we found that at least nine of the 0-days are variants of previously patched vulnerabilities. At least half of the 0-days we've seen in the first six months of 2022 could have been prevented with more comprehensive patching and regression tests. On top of that, four of the 2022 0-days are variants of 2021 in-the-wild 0-days. Just 12 months from the original in-the-wild 0-day being patched, attackers came back with a variant of the original bug.

https://googleprojectzero.blogspot.com/2021/02/deja-vu-lnerability.html
https://googleprojectzero.blogspot.com/2022/06/2022-0-day-in-wild-exploitationso-far.html

Product	2022 ITW CVE	Variant
Windows win32k	CVE-2022-21882	CVE-2021-1732 (2021 itw)
iOS IOMobileFrameBuffer	CVE-2022-22587	CVE-2021-30983 (2021 itw)
WebKit “Zombie”	CVE-2022-22620	Bug was originally fixed in 2013, patch was regressed in 2016
Firefox WebGPU IPC	CVE-2022-26485	Fuzzing crash fixed in 2021
Android in ARM Mali GPU	CVE-2021-39793 CVE-2022-22706	CVE-2021-28664 (2021 itw)
Sophos Firewall	CVE-2022-1040	CVE-2020-12271 (2020 itw)
Chromium v8	CVE-2022-1096	CVE-2021-30551 (2021 itw)
Chromium	CVE-2022-1364	CVE-2021-21195

Product	2022 ITW CVE	Variant
Windows “PetitPotam”	CVE-2022-26925	CVE-2021-36942 - Patch regressed
Windows “Follina”	CVE-2022-30190	CVE-2021-40444 (2021 itw)
Atlassian Confluence	CVE-2022-26134	CVE-2021-26084 (2021 itw)
Chromium Intents	CVE-2022-2856	CVE-2021-38000 (2021 itw)
Exchange SSRF “ProxyNotShell”	CVE-2022-41040	CVE-2021-34473 “ProxyShell”
Exchange RCE “ProxyNotShell”	CVE-2022-41082	CVE-2023-21529 “ProxyShell”
Internet Explorer JScript9	CVE-2022-41128	CVE-2021-34480
Windows “Print Spooler”	CVE-2022-41073	CVE-2022-37987
WebKit JSC	CVE-2022-42856	2016 bug

20% of the in-the-wild 0-days from 2022 are variants of previous ITW 0-days.

ARM Mali: CVE-2021-39793 / CVE-2022-22706

https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2021/CVE-2021-39793.html

- Mali GPU allows userspace to create a GPU memory object from host-virtual memory areas and access those object from userspace
- Mali flags to track permissions to GPU memory objects:
  - **`KBASE_REG_GPU_RD`** and **`KBASE_REG_GPU_WR`** for read/write access from jobs running on the GPU through GPU-virtual addresses
  - **`KBASE_REG_CPU_RD`** and **`KBASE_REG_CPU_WR`** for read/write access from host kernel code (on behalf of userspace) and host userspace

- Mali GPU allows userspace to create a GPU memory object from host-virtual memory areas and access those object from userspace
- Mali flags to track permissions to GPU memory objects:
  - **KBASE_REG_GPU_RD** and **KBASE_REG_GPU_WR** for read/write access from jobs running on the GPU through GPU-virtual addresses
  - **KBASE_REG_CPU_RD** and **KBASE_REG_CPU_WR** for read/write access from host kernel code (on behalf of userspace) and host userspace

CVE-2021-28664

```
static struct kbase_va_region *kbase_mem_from_user_buffer(
          struct kbase_context *kctx, unsigned long address,
          unsigned long size, u64 *va_pages, u64 *flags)
 {
[...]

 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
                        reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
                        pages, NULL);
 #else
                        reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
 #endif
 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(address, *va_pages,
                        reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
 #else
        faulted_pages = get_user_pages(address, *va_pages,
                        reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
                        pages, NULL);
 #endif
```

CVE-2021-28664

```
static struct kbase_va_region *kbase_mem_from_user_buffer(
         struct kbase_context *kctx, unsigned long address,
         unsigned long size, u64 *va_pages, u64 *flags)
 {
[...]

 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
       faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
                        reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
                        pages, NULL);
 #else
                        reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
 #endif
 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
       faulted_pages = get_user_pages(address, *va_pages,
                        reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
 #else
      faulted_pages = get_user_pages(address, *va_pages,
                       reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
                       pages, NULL);
 #endif
```

CVE-2021-28664

```
static struct kbase_va_region *kbase_mem_from_user_buffer(
          struct kbase_context *kctx, unsigned long address,
          unsigned long size, u64 *va_pages, u64 *flags)
 {
[...]

 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
                        reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
                        pages, NULL);
 #else
                        reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
 #endif
 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(address, *va_pages,
                        reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
 #else
      faulted_pages = get_user_pages(address, *va_pages,
                        reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
                        pages, NULL);
 #endif
```

CVE-2021-28664

Patched March 2021

```
static struct kbase_va_region *kbase_mem_from_user_buffer(
                struct kbase_context *kctx, unsigned long address,
                unsigned long size, u64 *va_pages, u64 *flags)
 {
[...]
+       int write;
[...]
+       write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);
+
 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
-                       reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
-                       pages, NULL);
+                       write ? FOLL_WRITE : 0, pages, NULL);
 #else
-                       reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
+                       write, 0, pages, NULL);
 #endif
 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(address, *va_pages,
-                       reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
+                       write, 0, pages, NULL);
 #else
        faulted_pages = get_user_pages(address, *va_pages,
-                       reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
-                       pages, NULL);
+                       write ? FOLL_WRITE : 0, pages, NULL);
   #endif
```

CVE-2021-28664

Patched March 2021

```
static struct kbase_va_region *kbase_mem_from_user_buffer(
                struct kbase_context *kctx, unsigned long address,
                unsigned long size, u64 *va_pages, u64 *flags)
 {
[...]
+       int write;
[...]
+       write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);
+
 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
-                       reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
-                       pages, NULL);
+                       write ? FOLL_WRITE : 0, pages, NULL);
 #else
-                       reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
+                       write, 0, pages, NULL);
 #endif
 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(address, *va_pages,
-                       reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
+                       write, 0, pages, NULL);
 #else
        faulted_pages = get_user_pages(address, *va_pages,
-                       reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
-                       pages, NULL);
+                       write ? FOLL_WRITE : 0, pages, NULL);
   #endif
```

CVE-2021-28664

Patched March 2021

```
static struct kbase_va_region *kbase_mem_from_user_buffer(
                struct kbase_context *kctx, unsigned long address,
                unsigned long size, u64 *va_pages, u64 *flags)
 {
[...]
+       int write;
[...]
+       write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);
+
 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
-                       reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
-                       pages, NULL);
+                       write ? FOLL_WRITE : 0, pages, NULL);
 #else
-                       reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
+                       write, 0, pages, NULL);
 #endif
 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
        faulted_pages = get_user_pages(address, *va_pages,
-                       reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
+                       write, 0, pages, NULL);
 #else
        faulted_pages = get_user_pages(address, *va_pages,
-                       reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
-                       pages, NULL);
+                       write ? FOLL_WRITE : 0, pages, NULL);
   #endif
```

CVE-2021-39793/CVE-2022-22706

```
int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
                struct kbase_va_region *reg)
 {
        struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
        struct page **pages = alloc->imported.user_buf.pages;
        unsigned long address = alloc->imported.user_buf.address;
        struct mm_struct *mm = alloc->imported.user_buf.mm;
        long pinned_pages;
        long i;


[...]
        pinned_pages = pin_user_pages_remote(
                mm, address, alloc->imported.user_buf.nr_pages,
                reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0, pages, NULL,
                NULL);
[...]
```

CVE-2021-39793/CVE-2022-22706

```
int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
                struct kbase_va_region *reg)
 {
        struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
        struct page **pages = alloc->imported.user_buf.pages;
        unsigned long address = alloc->imported.user_buf.address;
        struct mm_struct *mm = alloc->imported.user_buf.mm;
        long pinned_pages;
        long i;


[...]
         pinned_pages = pin_user_pages_remote(
                 mm, address, alloc->imported.user_buf.nr_pages,
                 reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0, pages, NULL,
                 NULL);
[...]
```

CVE-2021-39793/CVE-2022-22706

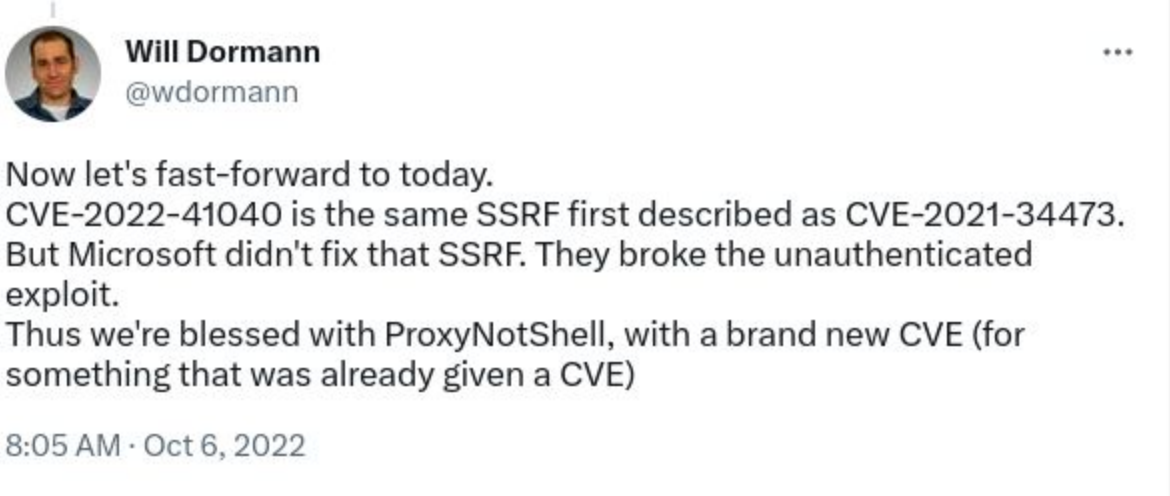
```
int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
                struct kbase_va_region *reg)
 {
        struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
        struct page **pages = alloc->imported.user_buf.pages;
        unsigned long address = alloc->imported.user_buf.address;
        struct mm_struct *mm = alloc->imported.user_buf.mm;
        long pinned_pages;
        long i;


[...]
         pinned_pages = pin_user_pages_remote(
                 mm, address, alloc->imported.user_buf.nr_pages,
                 reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0, pages, NULL,
                 NULL);
[...]
```

CVE-2021-39793/CVE-2022-22706

Patched March 2022

```
int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
                struct kbase_va_region *reg)
 {
        struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
        struct page **pages = alloc->imported.user_buf.pages;
        unsigned long address = alloc->imported.user_buf.address;
        struct mm_struct *mm = alloc->imported.user_buf.mm;
        long pinned_pages;
        long i;
+       int write;
[...]

+       write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);

[...]
        pinned_pages = pin_user_pages_remote(
                mm, address, alloc->imported.user_buf.nr_pages,
-               reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0, pages, NULL,
-               NULL);
+               write ? FOLL_WRITE : 0, pages, NULL, NULL);
[...]
```

ProxyNotShell: CVE-2022-41040 & CVE-2022-41082



https://twitter.com/wdormann/status/1578038749996666880

Piotr Bazydło
@chudyPB

It seems that my Exchange RCE was fixed - CVE-2023-21529. It was a bypass for the CVE-2022-41082 patch. Fun fact: I found this bypass before the patch 😆

11:11 AM · Feb 14, 2023 · **21.2K** Views

https://twitter.com/chudyPB/status/1625573475690635270

Safari: CVE-2022-22620

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22620.html
https://googleprojectzero.blogspot.com/2022/06/an-autopsy-on-zombie-in-wild-0-day.html

February 2022

```
1094  1094  // This does the same kind of work that didOpenURL does, except it relies on the fact
1095  1095  // that a higher level already checked that the URLs match and the scrolling is the right thing to do.
1096        -void FrameLoader::loadInSameDocument(const URL& url, SerializedScriptValue* stateObject, bool isNewNavigation)
      1096  +void FrameLoader::loadInSameDocument(URL url, RefPtr<SerializedScriptValue> stateObject, bool isNewNavigation)
```

https://trac.webkit.org/changeset/288539/webkit/trunk/Source/WebCore/loader/FrameLoader.cpp#file0

January 2013

```
360 360  void closeAndRemoveChild(Frame*);
361 361
362      void loadInSameDocument(const KURL&,SerializedScriptValue* stateObject, bool isNewNavigation);
    362  void loadInSameDocument(const KURL&,PassRefPtr<SerializedScriptValue> stateObject, bool isNewNavigation);
```

https://trac.webkit.org/changeset/140748/webkit#file7

- December 2009 - state object History API added.
  - `HistoryItem.m_stateObject` is type `RefPtr<SerializedScriptValue>`
  - `HistoryItem::stateObject()` returns `SerializedScriptValue*`
  - `FrameLoader::loadInSameDocument` takes `stateObject` argument as `SerializedScriptValue*`
- January 2013 - Patching Sergei's bug
  - `HistoryItem::stateObject` returns a `PassRefPtr<SerializedScriptValue>`
  - `FrameLoader::loadInSameDocument` takes `stateObject` argument as `PassRefPtr<SerializedScriptValue>`
- September 2015- Deprecating use of PassRefPtr in history directory
  - `HistoryItem::stateObject` returns `RefPtr` instead of `PassRefPtr`
- October 2016 - (Potentially) ad-hoc refactoring
  - `HistoryItem::stateObject()` is changed to return raw pointer instead of `RefPtr`
- December 2016 - CVE-2022-22600 introduced
  - `FrameLoader::loadInSameDocument` changed to take `stateObject` as a raw pointer instead of `PassRefPtr<SerializedScriptValue>`
- January 2022 - CVE-2022-22600 patched
  - `FrameLoader::loadInSameDocument` changed to take `stateObject` as a `RefPtr<SerializedScriptValue>`

What do we do?

Correct and comprehensive patches

Transparency

👏 KUDOS 👏

for disclosing when a vulnerability is in-the-wild

Adobe
Apple
Apache
ARM
Atlassian
Citrix
Fortinet
Google
Microsoft
Mozilla
Sophos
Trend Micro

Continue investing in detection

Continue investing in detection

+ collaboration!

THANK YOU!

@maddiestone

Google