The KEV Illusion

Separating True Threats from Pretend-Critical Risks

How Should Enterprise Security Teams Actually Respond to CISA's KEV Alerts?

Presented By OX

May 2025 By OX Research | Moshe Siman Tov Bustan

Table of Contents

Executive Summary: Not All Critical Vulnerabilities Are Critical for You

The very summarized summary: Context determines criticality. Our analysis of 10 KEV vulnerabilities found none applicable (or very low probability) to typical cloud container environments, proving that environmental assessment should drive patching priorities over universal "critical" labels.

Since its establishment in 2021, CISA's Known Exploited Vulnerabilities (KEV) catalog has become a trusted resource for defenders. While KEV is an excellent tool for focusing attention, it encompasses attacks across diverse platforms- from personal phones and webcams to cloud containers- without differentiating their contextual relevance.

Treating all KEV vulnerabilities with equal urgency, as is sometimes demanded by compliance regulations, and regardless of environmental context, creates unnecessary workload for already overwhelmed security teams and diverts resources from genuinely critical issues. Our analysis of 10 recent KEV CVEs demonstrates that many are completely irrelevant or unexploitable in cloud containerized environments, challenging the "patch everything" approach.

To be clear: This analysis does not suggest that KEV vulnerabilities should be ignored. Many KEV-listed CVEs do represent significant threats to cloud environments and should remain high-priority for remediation. Rather, our findings demonstrate that context-driven assessment is essential. The same vulnerability that poses a critical risk in one environment may be completely irrelevant in another.

While KEV is an excellent tool for focusing attention, it encompasses attacks across diverse platforms- from personal phones and webcams to cloud containers- without differentiating their contextual relevance

Key Findings

- **6 CVEs** were originally reported on Android and require Android-specific environments to reproduce, physical access for USB connections, or terminal access. While 2 of these 6 Android-related CVEs do apply to most OSs built on the Linux kernel, successful exploitation would require chaining them with additional vulnerabilities.

- **1 CVE** was initially reported in Apple's ecosystem browsers, where the environment's cookie-management logic was flawed- an issue that doesn't apply to cloud containerized environments.

- **3 CVEs** were initially reported in libraries used by the Google Chrome browser. This attack path is irrelevant for cloud containers, as most don't use these libraries for content processing and rendering.

Summary: Vulnerability Analysis

CVE	Description	Cloud Container Relevance	Reported On	OX Detection Count*	What to do when it pops up?
CVE-2024-53104	Linux webcam driver vulnerability	Unexploitable	Android	29,037	Fix only when local access is available
CVE-2024-50302	Linux USB driver vulnerability	Unexploitable	Android	30,298	Fix only when local access is available
CVE-2019-2215	Android Binder Vulnerability	Unexploitable	Android	25,352	Can be ignored on non-Android platforms
CVE-2021-0920	Linux Network Socket Vulnerability	Exploitable only under very specific conditions	Android	61,532	Fix only when local access is available, or when directly exposed to the internet
CVE-2021-1048	Linux System Call Monitor Vulnerability	Exploitable only under very specific conditions	Android	61,532	Fix only when local access is available, or when directly exposed to the internet
CVE-2023-0266	Linux Sound Driver Vulnerability	Unexploitable	Android	76,583	Fix only when local access is available
CVE-2024-44309	Apple Cookie Vulnerability	Unexploitable	Apple Browsers	325	Can be ignored on non-browser platforms
CVE-2023-4863	Chrome Image Processing Library Vulnerability	Exploitable only under very specific conditions	Chrome	275,328	Fix only if your service is used for image processing or uses image processing logic
CVE-2020-15999	Chrome Font Processing Library Vulnerability	Exploitable only under very specific conditions	Chrome	67,732	Fix only if your service is used for font processing or uses font processing logic
CVE-2023-5217	Chrome Video Processing Library Vulnerability	Exploitable only under very specific conditions	Chrome	26,027	Fix only if your service is used for video processing or uses image processing video

**The number of times the CVE was detected in open source containers by OX scanners*

Key Takeaways

Context matters significantly when reporting and prioritizing CVEs, even those labeled as Critical KEVs. With security teams facing 180+ new KEVs annually, contextual prioritization is essential for effective vulnerability management.

1

Recommendations for Security Teams

Before treating a KEV as critical:

- **Determine the original context** in which the CVE was reported and compare it to your environment
- **Search for POCs and exploit examples.** If none exist, there is a reduced likelihood (though not zero) that an attacker would develop the exploit themselves
- **Assess the vulnerability's relationship to sensitive information.** Critical services managing sensitive information should be prioritized over other services in your environment

With security teams facing 180+ new KEVs annually, contextual prioritization is essential for effective vulnerability management

2

Recommendations for CISA and Vulnerability Monitoring Organizations

Enhance KEV catalog with:

- **Platform-specific relevance indicators** (Android, iOS, Containers, Servers, Services, Web, Browsers)
- **CVE origin information** (discovered in Chrome browsers, Linux kernel, Docker environments, etc.)
- **Attack chain and attack path context**

This additional contextual information would enable security teams to implement a more precise and efficient workflow when handling critical vulnerabilities in their environments, reducing alert fatigue and focusing resources where they matter most.

How These 10 CVEs Made It Onto Our List

200
separate environments examined

10,000
most common CVEs identified

25
listed as KEV CVEs

10
chosen to undergo comprehensive technical analysis

Introduction: Fantastic Resource, Poor Implementation

Since its establishment in 2021, the CISA Known Exploited Vulnerabilities (KEV) catalog has become a trusted resource for security engineers, application security practitioners, and defenders. The catalog methodically maintains an updated list of software and hardware vulnerabilities actively exploited in the wild. Government backing gives KEV listings credibility as verified exploits, naturally elevating their remediation priority. Indeed, KEV is a fantastic resource for focusing attention on what adversaries care about broadly; however, it covers all kinds of attacks, on all kinds of devices and environments, spanning from a personal phone, through webcams, and onto cloud containers.

Treating KEV vulnerabilities without proper context is a blunt oversimplification that undermines the catalog's purpose, creating excessive work for already overwhelmed teams. Should all KEV vulnerabilities raise red flags? Absolutely not.

In this piece, we will demonstrate through analysis of 10 KEV CVEs how, in many cases, issues cataloged as critical can be completely irrelevant and unexploitable depending on your specific use case. All examples mentioned suffer from the same core problem: they were found exploitable on one platform, but are being detected as critical KEV CVE on different, unrelated platforms, regardless of the attack path a real threat actor would take to exploit them. KEV is like hearing the fire alarm ringing in your neighborhood. It doesn't mean it is in your building and even if it does it does mean there is a fire.

This analysis aims to help security teams better prioritize their remediation efforts, avoid unnecessary endless patching cycles, and focus resources where they matter most.

Methodology

We examined over 200 separate environments and gathered the 10,000 most common CVEs in cloud native applications. From this data set, we extracted only CVEs that were also listed in CISA's Known Exploited Vulnerabilities (KEV) catalog. This filtering process identified a total of 25 KEVs.

Out of these 25 KEVs, we identified 10 that initially appeared to be technically unexploitable or highly impractical to exploit in cloud containerized environments. These were selected based on preliminary assessment of their attack vectors, required exploitation conditions, and the platforms they were originally discovered on.

Each CVE was subjected to a comprehensive technical analysis. The following report presents our detailed analysis demonstrating why these vulnerabilities cannot be exploited in typical cloud containerized environments, despite their classification as critical KEVs.

In this piece, we will demonstrate through analysis of 10 KEV CVEs how, in many cases, issues cataloged as critical can be completely irrelevant and unexploitable

Shared Code, Different Environments: The Case of Sugar

Every operating system runs on code in the digital realm. And while these systems may look and behave differently, many share common code fragments. Think of these shared code elements as ingredients: sugar, for instance, can be used to make cookies, a dark chocolate cake, a loaf of bread, tomato sauce, or simply sweeten a cup of tea. Depending on how it's used, the same ingredient can lead to vastly different outcomes. Just like that, a single piece of code can play a very different role depending on the operating system it's part of.

Common End User Operating Systems	Common Cloud Operating Systems
Android	Windows Server
PC	Debian
iOS	Alpine
Mac	Redhat
Linux Desktop	Ubuntu Server
Debian Desktop	CentOS (still used despite EOL)
Chrome OS	Amazon Linux
Ubuntu Desktop	Oracle Linux
Fedora Workstation	SUSE Linux Enterprise Server (SLES)
Windows	

Depending on how it's used, the same ingredient can lead to vastly different outcomes. Just like sugar: a single piece of code can play a very different role depending on the operating system it's part of.

For example, a vulnerability found in a Linux kernel is highly likely to also exist in an Android kernel. But just like an ingredient used in multiple recipes, the same code can behave very differently depending on the system's context. A vulnerability that may pose a critical risk in Android might have a negligible impact in a cloud container environment where the execution path and exposure differ substantially.

Take, for example, a CVE flagged as critical in the Linux kernel, originally discovered as part of an Android attack chain. This vulnerability is irrelevant in a cloud-based container environment, as containers do not share the same attack surface as Android devices or follow the same execution paths. Yet, when CVEs are stripped from their original context, they are often flagged as urgent in environments where they are entirely harmless.

This is where the industry gets caught in a detrimental cycle: enterprise AppSec teams are bombarded with open issues and patching tasks for vulnerabilities that, while critical in other contexts, have no bearing on their environments. The result is wasted time - updating, triaging, and validating irrelevant fixes instead of addressing actual threats.

Defenders must become aware of these nuances. Identifying which CVE applies to which platform, and understanding how a real-world attacker might exploit a vulnerability, is essential for determining whether and how a remediation should be implemented.

Exploiting End-User vs Cloud: Understanding Threat Actors

Understanding threat actors' strategic objectives and corresponding methodologies is important.

Vulnerabilities affecting end-user devices can be serious and require attention, but they typically have containment boundaries that limit their spread to the enterprise cloud environment. Meanwhile, vulnerabilities directly targeting cloud infrastructure often present a more expansive attack surface with potentially greater business impact.

This context helps security teams and CISOs appropriately prioritize their defensive efforts, allocating more resources to business-critical applications and infrastructure. The following table charts explain:

Threat Actors' Goals	Description
Financial Gain	Selling stolen information, credit card data, ransomware, crypto-mining, and network hacks
Data Theft	Stealing personal, corporate data, trade secrets, or user info
Disruption	Sabotaging systems or infrastructure
Ideology	Whistleblowing, protests, and hacktivism

Table 1: Threat Actor Objectives

Techniques	Description
Phishing	Tricking users into sharing personal data
Exploits	Using known vulnerabilities for unauthorized access
Brute Force	Guessing passwords or using stolen ones
Malicious Media	Dropping infected USBs or devices to gain access

Table 2: Common Techniques Used by Threat Actors

In cloud security, the primary risks are **data theft** and **ransomware**, typically through **zero-day** and **n-day exploits.**

Zero-Day Exploits: Personal Device vs Cloud Containers

- **Personal Device:** An attacker exploits a device vulnerability (e.g., Android remote code execution) via a payload sent through an image or video, which executes malicious code.
- **Cloud Containers:** Attacks on cloud servers are less direct due to multiple barriers (e.g., OS, CPU architecture). More common methods include exploiting open ports or accessing services through protocols like SSH, often requiring local privilege escalation through further exploits.

The attack path in cloud environments is more complex, often requiring breaking through additional layers like remote shells or local privilege escalation.

Type of Attack	Non-cloud Environment	Cloud Environment
Social Engineering / Phishing	End users, targeted attacks, require initial info (email, phone, social media)	Irrelevant, unless specifically targeted
XSS (Cross-Site Scripting)	Web servers with user input	Containers with internet access, external web services, and user input
Zero-day Kernel Exploits	Devices with direct internet access or physical access	Potentially VMs/containers with terminal access or direct internet access
Zero-day Remote Code Execution Exploits	Devices with direct internet access or physical access	Services with exposed APIs, user input, and data processing mechanisms
Browser Exploits	End users, giving access to device and user info	Irrelevant
Command Injection	Vulnerable apps connected to databases	Cloud-hosted web apps or databases lacking input sanitization

Table 3: Exploitability Across Non-Cloud and Cloud Environments

CVE Analysis

In the following Section, we will review 10 Critical KEV CVEs, and see why none of them is actually relevant for cloud-based enterprises and their security teams:

1

CVE-2024-53104 - Linux Kernel UVC Driver Vulnerability

Show me a cloud container connected to a webcam

Reported on: Android

Relevant for: Android users, Linux end-users

Not relevant for: Windows Users & servers, Cloud Containerized Environments

Found by OX scanners in open-source containers: 29,037 occurrences

Bottom line: Highly critical local exploitation issue, mostly on devices using an external camera or that have video drivers installed. **Not critical for Cloud Containerized Environments**

This issue was reported by Google's internal security teams and external security researchers on December 2, 2024.

The Linux kernel UVC driver is in charge of communicating with video devices over USB, such as webcams and digital cameras, without the need to install additional USB drivers.

To exploit this vulnerability, the vulnerable machine must have the UVC driver installed and a physically connected webcam device sending maliciously crafted frames. Alternatively, a local attacker could create an emulated webcam device on the machine to send such frames to the device.

When analyzing this CVE from a containerized environment point of view, usually the UVC drivers will be missing from the kernel, and if exists, there's little to no use in them, making this issue practically non-exploitable.

The vulnerability was found in the driver which connects to webcams, and specifically in the function which is responsible for reading and parsing each frame's metadata, every frame has a "frame descriptor" which is that specific's frame data. We can see its code inside `/drivers/media/usb/uvc/uvc_driver.c`, where the vulnerable parser's function name is `uvc_parse_format`.

In the vulnerable version of the code, when frame descriptors with an "undefined" type are passed, they cause the code to allocate memory without using the allocated memory sections, meaning that an attacker can send maliciously crafted frames with undefined type over and over again, and potentially lead to memory leakage and arbitrary code execution.

```
diff --git a/drivers/media/usb/uvc/uvc_driver.c b/drivers/media/usb/uvc/uvc_driver.c
index 004511d..9b87a13 100644
--- a/drivers/media/usb/uvc/uvc_driver.c
+++ b/drivers/media/usb/uvc/uvc_driver.c

@@ -368,7 +368,7 @@ static int uvc_parse_format(struct uvc_device *dev,
   * Parse the frame descriptors. Only uncompressed, MJPEG and frame
   * based formats have frame descriptors.
   */
- while (buflen > 2 && buffer[1] == USB_DT_CS_INTERFACE &&
+ while (ftype && buflen > 2 && buffer[1] == USB_DT_CS_INTERFACE &&
         buffer[2] == ftype) {
   frame = &format->frame[format->nframes];
   if (ftype != UVC_VS_FRAME_FRAME_BASED)
```

Source: Google Android Source Code

References:
Google Git
Android Security Bulletin February 2025

2

CVE-2024-50302 - Linux Kernel HID Core Non-Zeroed Memory Vulnerability

Show me a cloud container connected to a USB device

Reported on: Android

Relevant for: Android users, Linux end-users

Not relevant for: Windows Users & servers, Cloud Containerized Environments

Found by OX scanners in open-source containers: 30,298 occurrences

Bottom line: Highly critical local exploitation issue, mostly for devices with USB drivers and USB inputs. Not critical for Cloud Containerized Environments.

This issue was reported on November 19, 2024.

This CVE targets the HID module in the Linux kernel, it means that the attacker needs local or physical access to the device, by emulating a USB peripheral or connecting a real device with malformed hardware, a threat actor can manipulate the kernel to allocate and expose important information through the USB driver.

If we analyze Docker environments, for example, we can find that most USB and HID-related drivers are not accessible or do not even exist, and those that do exist are not loaded into memory.

From a cloud security perspective, this CVE is highly unlikely to be exploited. While it is a good practice to update your environment to the latest security patch, this issue does not pose a real threat to your environment.

The vulnerability itself lies on a non-zeroed memory allocation inside the Linux kernel's HID module, when analyzing the patch we can see that just a single line was changed, the

memory allocation function “kmalloc” is now “kzalloc”, this is important because “kmalloc” allocates a buffer from memory without initializing it, this means that whatever information that was written to that area of memory before allocation is now inside the current buffer, where “kzalloc” write zeros all over the allocated buffer before returning it.

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 15f4a804779745..07c2e5e38fcbaa 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1664,7 +1664,7 @@ u8 *hid_alloc_report_buf(struct hid_report *report, gfp_t flags)
  u32 len = hid_report_len(report) + 7;
- return kmalloc(len, flags);
+ return kzalloc(len, flags);
 }
 EXPORT_SYMBOL_GPL(hid_alloc_report_buf);
```

Source: git.kernel.org

By repeatedly triggering the function that eventually uses kmalloc to allocate memory, an attacker can gain read access to discarded memory sections in the kernel. This information can contain sensitive keys, strings, and memory pointers, which can help the attacker defeat ASLR defenses, bypass security measures, and gain elevated privileges.

References:
Google Git
Linux Kernel USB HID Memory Leak Vulnerability
Android Zero-Days Used by Authorities to Unlock Confiscated Devices

3

CVE-2019-2215 - Android's IPC Binder Use-After-Free

'Cause we are living in an Android world, and I am an Android vuln

Reported on: Android

Relevant for: Android users

Not relevant for: Windows Users & servers, Linux end-users, Cloud Containerized Environments

Found by OX scanners in open-source containers: 25,352 occurrences

Bottom line: Critical local exploitation issue, exploitable mostly on Android devices. **Not critical for Cloud Containerized Environments.**

This CVE was found in Android's binder module, which is responsible for IPC (inter-process communication) and enables applications and frameworks to communicate within the Android ecosystem. In general, Android applications should run separately from one another, in separate environments. This means that no application has the ability to read another application's files and metadata. One of the exceptions to this behavior is the user of IPC, where Android exposes sets of APIs (Binder, AIDL) in order to give applications a limited ability to communicate with each other for specific purposes.

The Android binder driver had a race condition where a thread's waitqueue could remain registered in epoll structures even after the thread was freed. This happened because the kernel didn't call `wake_up_poll`(..., POLLFREE) during cleanup, which is the mechanism epoll uses to remove references to freed waitqueues.

If an attacker reclaims the freed waitqueue memory with controlled data, the kernel may later call function pointers or access structures that are now compromised, leading to code execution or privilege escalation.

Inside the patch, we can see that logic was added to ensure that the internal reference to waitqueue from the epoll is removed correctly using the `wake_up_poll` function with the POLLFREE flag. Removing this reference ensures that the waitqueue's memory cannot be referenced, used, or read by an external process.

Cloud containerized environments are not affected by this issue. It relates only to Android's ecosystem since the fix is in a very specific Android code.

```
diff --git a/drivers/staging/android/binder.c b/drivers/staging/android/binder.c
index f95b559..b2e4f493b 100644
--- a/drivers/staging/android/binder.c
+++ b/drivers/staging/android/binder.c
@@ -4532,6 +4532,18 @@
   if (t)
    spin_lock(&t->lock);
  }
+
+ /*
+  * If this thread used poll, make sure we remove the waitqueue
+  * from any epoll data structures holding it with POLLFREE.
+  * 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)
```

From Android Source Code

References:
Android use-after-free in Binder
Android: Use-After-Free in Binder driver
Bound services overview - Binder
Android Interface Definition Language (AIDL)

4

CVE-2023-0266 - ALSA Driver (Advanced Linux Sound Architecture) race condition

How good is the sound in your container?

Reported on: Android

Relevant for: Android users, Linux end-users, and Linux devices with Audio Drivers

Not relevant for: Windows Users & servers, Cloud Containerized Environments

Found by OX scanners in open-source containers: 76,583 occurrences

Bottom line: Critical local exploitation issue, found as part of an Android exploitation chain. Exploitation is highly unlikely in Cloud Containerized Environments.

This CVE is a use-after-free vulnerability in the Linux sound driver - ALSA (Advanced Linux Sound Architecture), it was reported by Google's Threat Analysis Group (TAG) in December 2022.

This vulnerability was detected in the wild as part of an Android exploit chain. In order to get full remote access, the threat actor needed to chain together a zero-day in the Samsung browser, a zero-day in the Mali GPU, and finally, the ALSA 32-bit to 64-bit compatibility layer.

The vulnerability is a use-after-free vulnerability in the Linux sound device, a threat actor gaining local access to the device can interact with the device's sound driver under /dev/snd/, usually /dev/snd/controlC0, the threat actor opens the sound device, then calls the ioctl function, specifically reading 32-bit values which might look like this.

```
C/C+
fd = open("/dev/snd/control1C0", O_RDWR)
ioctl(fd, SNDRV_CTL_IOCTL_ELEM_READ32,
...)
```

Later, the threat actor will create multiple threads, removing the fd and reading or writing at the same time, which gives the ability to access the freed memory due to the missing locks on the **`read/write`** functions in the **`ioctl`** call.

The 32-bit and 64-bit ioctls differ until they both call **`snd_ctl_elem_{write|read}`** so when the lock was moved up the 64-bit call chain, it was entirely removed from the 32-bit ioctls. Here's the code path for **`SNDRV_CTL_IOCTL_ELEM_WRITE`** in 64-bit mode on kernel 5.10.107 post-refactor:

```
snd_ctl_ioctl
  snd_ctl_elem_write_user
    [takes controls_rwsem]
    snd_ctl_elem_write [lock properly held, all good]
    [drops controls_rwsem]

And here is the code path for that same ioctl called in 32-bit mode:

snd_ctl_ioctl_compat
  snd_ctl_elem_write_user_compat
    ctl_elem_write_user
      snd_ctl_elem_write [missing lock, not good]
```

Source: Project Zero

Analyzing the fix

Inside multiple functions under **`sound/code/control.c`**, an unlock logic was added to properly handle locking around element read and write in 32-bit **`ioctl`** functions, without these locks the threat actor could potentially gain read and write access to arbitrary kernel addresses.

```
ALSA: pcm: Move rwsem lock inside snd_ctl_elem_read to prevent UAF
Takes rwsem lock inside snd_ctl_elem_read instead of snd_ctl_elem_read_user like it was done
for write in commit 1fa4445 ("ALSA: control - introduce snd_ctl_notify_one() helper"). Doing
this way we are also fixing the following locking issue happening in the compat path which
can be easily triggered and turned into an use-after-free.

64-bits:
snd_ctl_ioctl
  snd_ctl_elem_read_user
    [takes controls_rwsem]
    snd_ctl_elem_read [lock properly held, all good]
    [drops controls_rwsem]

32-bits:
snd_ctl_ioctl_compat
  snd_ctl_elem_write_read_compat
    ctl_elem_write_read
      snd_ctl_elem_read [missing lock, not good]

CVE-2023-0266 was assigned for this issue.
```

```
1231  1236      snd_power_unref(card);
1232  1237       if (ret < 0)
1233              return ret;
      1238       goto unlock;
1234  1239       if (!snd_ctl_skip_validation(&info) &&
1235  1240           sanity_check_elem_value(card, control, &info, pattern) < 0) {
1236  1241        dev_err(card->dev,
1237  1242         "control %i:%i:%i:%s:%i: access overflow\n",
1238  1243         control->id.iface, control->id.device,
1239  1244         control->id.subdevice, control->id.name,
1240  1245         control->id.index);
1241              return -EINVAL;
      1246       ret = -EINVAL;
      1247       goto unlock;
1242  1248       }
      1249      unlock:
      1250       up_read(&card->controls_rwsem);
1243  1251       return ret;
```

Source: Github

The ALSA driver doesn't come preinstalled on regular cloud container environments, its existence can be found under **`/var/lib/alsa`** and **`/usr/share/alsa`**. For installation, a user might need to invoke apt install alsa-utils. Also, the use of an existing **`/dev/snd/controlC0`** i/o soundcard device is needed, which might go by another name depending on the OS and different implementations.

This sound driver vulnerability might be applicable for a cloud environment where the sound driver is present on the machine and the threat actor has already gained some kind of local access in order to execute code that interacts with the device's audio drivers under **`/dev/snd/`**.

References:
Analyzing a Modern In-the-wild Android Exploit
Github - Commit 56b88b5

5

CVE-2024-44309 - Apple Eco-System Cookie Management Issue

No cookies in my cloud

Reported on: Apple

Relevant for: Apple end users, browsers using WebKit

Not relevant for: Cloud Containerized Environments, Android users and Linux Servers

Found by OX scanners in open-source containers: 325 occurrences

Bottom line: Cookie privilege escalation in browsers.

The CVE was found by Clément Lecigne and Benoît Sevens from Google's Threat Analysis Group (TAG), they found that by exploiting a cross-site scripting attack inside a maliciously crafted web content, the web content is processed by WebKit in a way that lets the attacker steal user sessions, cookies and other personal information.

This CVE targets mostly Apple browsers that are using the WebKit engine to render web pages and manage cookies. Non-Apple devices are usually not affected by this, but browsers and other Linux applications using WebKit to handle cookies might be affected.

This CVE is only relevant for browser-based applications for browser developers, and for users using browsers with a WebKit engine, which exist mostly on Apple devices, since the core issue is inside the cookie management core logic.

Usually, there is not much information about patches in Apple's ecosystem due to Apple's closed-source policy. The CVE doesn't directly link to the patch, and the Bugzilla issue is restricted, but since the WebKit project is open source, we were able to find the actual patch in the WebKit's source code on Github.

When analyzing the fix, we can see that multiple checks were added to grant or deny cookie access, enforcing a 'disallow and terminate' policy to more effectively prevent access by external processes.

```
461  462          auto iterator = m_allowedFirstPartiesForCookies.find(processIdentifier);
462  463          if (iterator == m_allowedFirstPartiesForCookies.end()) {
463  464              ASSERT_NOT_REACHED();
464      -            return false;
     465  +            return terminateOrDisallow;
465  466          }
466  467
467  468          if (iterator->value.first == LoadedWebArchive::Yes)
468      -            return true;
     469  +            return AllowCookieAccess::Allow;
469  470
470  471          auto& set = iterator->value.second;
471  472          if (!
                 std::remove_reference_t<decltype(set)>::isValidValue(firstPartyDomain)) {
472  473              ASSERT_NOT_REACHED();
473      -            return false;
     474  +            return terminateOrDisallow;
474  475          }
```

```
542  542      {
543      -      bool hasCookieAccess = protectedNetworkProcess()-
                >allowsFirstPartyForCookies(m_webProcessIdentifier,
                loadParameters.request.firstPartyForCookies());
544      -      if (UNLIKELY(!hasCookieAccess))
     543  +      auto allowCookieAccess = protectedNetworkProcess()-
                >allowsFirstPartyForCookies(m_webProcessIdentifier,
                loadParameters.request.firstPartyForCookies());
     544  +      if (UNLIKELY(allowCookieAccess !=
                NetworkProcess::AllowCookieAccess::Allow))
545  545              RELEASE_LOG_ERROR(Loading, "scheduleResourceLoad: Web process does not
                have cookie access to url %" SENSITIVE_LOG_STRING " for request %"
                SENSITIVE_LOG_STRING,
                loadParameters.request.firstPartyForCookies().string().utf8().data(),
                loadParameters.request.url().string().utf8().data());
546  546
547      -      MESSAGE_CHECK(hasCookieAccess);
     547  +      MESSAGE_CHECK(allowCookieAccess !=
                NetworkProcess::AllowCookieAccess::Terminate);
```

Source: Github

Mitigation involves updating your devices to the latest security patch by Apple, including the following versions and devices

- Safari 18.1.1
- iOS 17.7.2 (or iOS 18.1.1, if applicable)
- iPadOS 17.7.2 (or iPadOS 18.1.1, if applicable)
- macOS Sequoia 15.1.1
- visionOS 2.1.1

The Stars' Alignment Exploits

The following two examples, CVE-2021-0920 and CVE-2021-1048, tell a slightly different story. While it's true that they can be exploited in cloud containerized environments, they were discovered as part of a complex, multi-step exploitation chain. This means that successful real-world exploitation would typically require multiple working exploits or full unprivileged system access. Since they would only be viable under rare and tightly controlled conditions, we named them "The Stars' Alignment Exploits".

6

CVE-2021-0920 - Linux Socket Use-After-Free

The thieves are already in, would you buy an alarm system now?

Reported on: Android

Relevant for: Android users, Linux end-users, and Linux Servers

Not relevant for: Windows Users & servers, Cloud Containerized Environments

Found by OX scanners in open-source containers: 61,532 occurrences

Bottom line: Critical local exploitation issue, found as part of a Samsung Android device's exploitation chain. **Exploitation is highly unlikely in Cloud Containerized Environments**

This vulnerability was disclosed on December 15, 2021. It is a use-after-free vulnerability that was discovered as part of an in-the-wild Samsung full-chain exploit, while the other CVEs in the reported chain reference the Samsung internal browser and the Chrome browser.

This use-after-free vulnerability originates from a logical error related to “garbage collection” and socket descriptors. Garbage collection is when a function in the code exits and the operating system clears out the allocated memory. Socket descriptors are unique identifiers that the operating system uses to manage and access network connections.

For an attacker to successfully exploit the vulnerability, local device access is required. In the original findings, this vulnerability was able to run on the local machine because it was chained with other vulnerabilities that reached the vulnerable device in the first place. Originally, the actor gained full device access after exploiting a remote code execution originating in the device’s browser.

After gaining local device access, the attacker’s code creates multiple network socket connections from the device to itself, allowing it to send and receive connections with itself. Then a call is made to the “`recvmsg`” function, which reads the socket descriptors created. By using the “`recvmsg`” function call with the “`MSG_PEEK`” flag, it increases the reference count of two variables - “`total_refs`” but not “`inflight_refs`”, which the operating system uses to know if a descriptor is used or closed. The issue was that “`total_refs`” was updated this way but “`inflight_refs`” was not update, then, the garbage collector runs ans sees that `total_refs` value is higher than the `inflight_refs`, causing the garbage collector to think it should free the pointer pointing to the network socket, while it is still being used.

In order to use this exploit, the attacker needs local access to the machine, the ability to create and listen to sockets, and spray the

heap successfully with the correct timing and values to make this exploit work.

In cloud environments, such an attack is technically possible but requires a combination with other zero-day exploits to gain local access to the machine.

Additionally, it was only found in the context of Android devices, where the attacker knows the targeted device's kernel version and other publicly available device information, as opposed to cloud environments, where the attacker can gain access to this information only after gaining initial access.

Analyzing the fix

A lock function was added to protect from the garbage collector freeing the socket stream incorrectly. It wraps the "**`scm_fp_dup`**" function and adds a lock and unlock function. Using this logic, the garbage collector can only free the file descriptor after the unlock is done, removing the possibility of a race condition.

```
+static void unix_peek_fds(struct scm_cookie *scm, struct sk_buff *skb)
+{
+       scm->fp = scm_fp_dup(UNIXCB(skb).fp);
+       /*
+       */
+ spin_lock(&unix_gc_lock);
+ spin_unlock(&unix_gc_lock);
+}
static int unix_scm_to_skb(struct scm_cookie *scm, struct sk_buff *skb, bool send_fds)
 {
  int err = 0;
@@ -2175,7 +2222,7 @@ static int unix_dgram_recvmsg(struct socket *sock, struct msghdr *msg,
   sk_peek_offset_fwd(sk, size);

   if (UNIXCB(skb).fp)
-   scm.fp = scm_fp_dup(UNIXCB(skb).fp);
+   unix_peek_fds(&scm, skb);
  }
  err = (flags & MSG_TRUNC) ? skb->len - skip : size;

@@ -2418,7 +2465,7 @@ static int unix_stream_read_generic(struct unix_stream_read_state
*state,
    /* It is questionable, see note in unix_dgram_recvmsg.
     */
    if (UNIXCB(skb).fp)
-    scm.fp = scm_fp_dup(UNIXCB(skb).fp);
+    unix_peek_fds(&scm, skb);

    sk_peek_offset_fwd(sk, chunk);
```

Source: Android Source Code

References:
Android Security Bulletin—November 2021
The quantum state of Linux kernel garbage collection - Part 1
Android sk_buff use-after-free in Linux

7

CVE-2021-1048 - Local privilege escalation vulnerability in the Linux kernel's epoll

The devil is in the kernel

Reported on: Android

Relevant for: Android users, Linux end-users and servers, some Cloud Containerized Environments

Not relevant for: Windows Servers, Containers with no direct internet access

Found by OX scanners in open-source containers: 61,532 occurrences

Bottom line: A critical local exploitation issue, exploits mainly targeting Android devices, **in rare cases might work on some Linux servers.**

CVE-2021-1048 is a **local privilege escalation vulnerability** in the Linux kernel's `epoll` subsystem, reported by syzbot, an automated bug-finding system that uses the syzkaller fuzzer. The vulnerability was patched on September 9, 2020.

This issue was actively exploited in the wild on Android devices. While potentially relevant in containerized cloud environments and Linux servers, real-world exploitation outside Android remains unlikely due to the complexity of the required conditions.

`epoll` is a Linux system call used to efficiently monitor multiple file descriptors for readiness in input/output operations. A vulnerability was found in the `ep_loop_check_proc`() function within `fs/eventpoll.c`, an internal helper responsible for detecting and preventing infinite loops and potential kernel deadlocks within the `epoll` mechanism.

A **race condition** was discovered in the `ep_loop_check_proc()` function within `fs/eventpoll.c`, which is responsible for preventing infinite loops and potential kernel deadlocks. The problem stemmed from **missing locking around the `get_file()` function** - a mechanism used to increment a file descriptor's reference count.

This oversight allowed a **use-after-free** scenario:

- One thread could release a file descriptor.
- Simultaneously, another thread could call `get_file()` on the same descriptor, leading to memory corruption or unauthorized access.

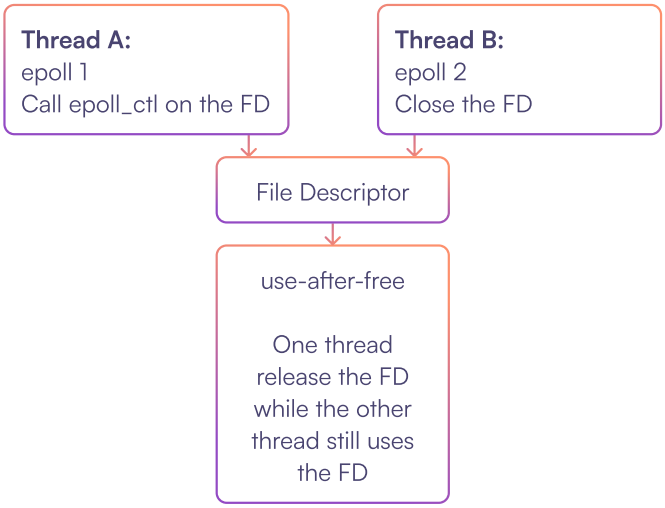
In attacks observed on Android, non-privileged users were able to create circular references between multiple `epoll` instances before the loop detection logic could safely validate them-bypassing protections and gaining elevated privileges.

A flaw was discovered in the `ep_loop_check_proc` function within `fs/eventpoll.c`; this internal helper is responsible for detecting and preventing infinite loops and potential kernel deadlocks in the `epoll` system.

This issue was exploited in the wild on Android devices, where non-privileged users could abuse a race condition in epoll's file descriptor

handling to gain elevated privileges. This was done by creating circular references between epoll instances before the loop detection logic could safely validate them.

The logic inside **ep_loop_check_proc** was missing a locking mechanism around the "**get_file**" function, which enabled a use-after-free condition when one thread released a file descriptor while another thread was simultaneously attempting to increment its reference count.



An attacker could potentially create multiple **epoll** instances and file descriptors, running on two different threads, one thread adds descriptors and calls **epoll_ctl**. At the same time, the other closes or duplicates them, potentially causing the race condition to trigger a use-after-free in one of the file descriptors.

One of the supposed implementations of this CVE in GitHub, directly calls the **epoll_ctl** function on a file descriptor of "**/dev/binder**", which is in charge of Android inter-process communication, emphasizing the relationship between this CVE and Android-specific exploitation.

Analyzing the fix

The kernel patch replaced **get_file** with **get_file_rcu**. The **get_file_rcu** function is a safer variant that uses the RCU (Read-Copy-Update) locking mechanism to ensure the reference count is handled correctly. This change ensures that file descriptor structures are not freed while still in use, reducing the possibility of a race condition and preventing attackers from triggering the use-after-free scenario.

```
@@ -1995,9 +1995,9 @@ static int ep_loop_check_proc(void *priv, void *cookie, int
call_nests)
     * during ep_insert().
     */
    if (list_empty(&epi->ffd.file->f_tfile_llink)) {
-    get_file(epi->ffd.file);
-    list_add(&epi->ffd.file->f_tfile_llink,
-      &tfile_check_list);
+    if (get_file_rcu(epi->ffd.file))
+     list_add(&epi->ffd.file->f_tfile_llink,
+       &tfile_check_list);
    }
   }
  }
```

Source: Google Source Code

References:
Memory Corruption Vulnerability in Linux Kernel
Android kernel refcount increment on mid-destruction file
Github - CVE-2021-1048.c

The Picasso Exploits

The following three examples (CVE-2023-4863, CVE-2020-15999, and CVE-2023-5217) were discovered in libraries used by the Chrome browser. These vulnerabilities were exploited to execute arbitrary code, gain control of the user's browser, and, in some cases, potentially escape the browser sandbox to compromise the underlying system.

However, successful exploitation of these vulnerabilities in a cloud containerized environment is highly unlikely. For such an attack to succeed, the container would need to actively use one of the vulnerable libraries to process user-supplied files (videos, images, or fonts). Furthermore, the malicious file must be carefully crafted based on specifics like the container's CPU architecture (e.g., x86, ARM) and kernel version - details that are typically unknown or irrelevant in ephemeral, sandboxed cloud environments.

Given these constraints and the **high cost of crafting such an exploit**, attackers are far more likely to target client-side Chrome users than invest resources into targeting cloud-based containers. We gave these attack paths the nickname **Picasso Exploits**, as they only look like a threat from afar.

8

CVE-2023-4863 - Heap buffer overflow in libwebp in Google Chrome

Portrait of exploit Stein

Reported on: Chrome

Relevant for: Chromium-based browsers end-users

Not relevant for: Linux Servers, Cloud Containerized Environments

Found by OX scanners in open-source containers: 275,328 occurrences

Bottom line: Remote exploitation issue in an image parsing library inside Chromium browsers. Exploitation is highly unlikely in Cloud Containerized Environments

This CVE was reported by Apple's Security Engineering and Architecture team (SEAR) on September 6, 2023.

This CVE lets the attacker perform an out-of-bounds memory write using a crafted HTML page containing a maliciously crafted WebP file. This means that when a user opens the attacker's web page, the attacker could write arbitrary code to memory, potentially giving the attacker full access to read memory of the current browser's process and files, and execute code on the user's current device.

Exploiting this CVE could be used as part of remote code execution, for example, if an application is designed to immediately open and show a picture, such as an instant messaging preview - the attacker could send the image and instantly trigger the vulnerability without any user interaction.

This CVE's vulnerability lies in the image processing library libwebp, in order to perform code execution it needs to rely on the library actually receiving and parsing a WebP image file, and also having prerequisite knowledge on the victim's machine, since each memory layout of each program and operating system is different, the payload delivered inside the WebP image file needs to be crafted per system.

Inside WebP, there's a use of Huffman Tables, which is a lossless compression algorithm,

which makes the file size smaller while maintaining the same pixel-information that constructs the final image. Inside the Huffman Tables implementation, parts of the compression logic allocated more memory than expected in some cases, which caused the out-of-bounds memory write to happen.

Analyzing the fix

When going over the fix, we can see the memory allocation function used for loading the image to memory, originally it allocated memory for Huffman coded data only for 8-bits long codes, but WebP supports up to 15-bits, meaning the crafted image with Huffman encoded 15-bits codes was allocating more memory than originally planned, this was changed to dynamically allocate according to the real table size, which can be seen in the **`ReadHuffmanCodeLengths`**, inside **`src/dec/vp8l_dec.c`**.

```
265       -   HuffmanCode table[1 << LENGTHS_TABLE_BITS];
      265 +   HuffmanTables tables;
266   266
267       -   if (!VP8LBuildHuffmanTable(table, LENGTHS_TABLE_BITS,
268       -                               code_length_code_lengths,
269       -                               NUM_CODE_LENGTH_CODES)) {
      267 +   if (!VP8LHuffmanTablesAllocate(1 << LENGTHS_TABLE_BITS, &tables) ||
      268 +       !VP8LBuildHuffmanTable(&tables, LENGTHS_TABLE_BITS,
      269 +                              code_length_code_lengths, NUM_CODE_LENGTH_CODES))
          {
270   270       goto End;
271   271     }
272   272
          @@ -286,7 +286,7 @@ static int ReadHuffmanCodeLengths(
286   286       int code_len;
287   287       if (max_symbol-- == 0) break;
288   288       VP8LFillBitWindow(br);
289       -     p = &table[VP8LPrefetchBits(br) & LENGTHS_TABLE_MASK];
      289 +     p = &tables.curr_segment->start[VP8LPrefetchBits(br) &
          LENGTHS_TABLE_MASK];
```

Source: Github

The libwebp library runs usually as part of a client-side application such as Chrome web browser, GIMP image editing application, and other email and messaging applications, but it can also be used for image editing services and format converters.

For cloud based platform this issue is relevant only for image processing services, where there's an API receiving and processing a potential WebP file, and even in those scenarios the possibility of a successful exploit is far fetched, since the attacker needs to do a large amount of blind trial and error sending WebP images with different shell codes, assuming the service contains the relevant vulnerable libwebp library.

References:
Github - Commit 902bc91

9

CVE-2020-15999 - Heap buffer overflow in Freetype in Google Chrome

Boy with a font

Reported on: Chrome

Relevant for: Chromium-based browsers end-users

Not relevant for: Linux Servers, Cloud Containerized Environments

Found by OX scanners in open-source containers: 67,732 occurrences

Bottom line: Remote exploitation issue in a font parsing library inside Chromium browsers. Exploitation is highly unlikely in Cloud Containerized Environments

This CVE was reported on October 19, 2020, by Sergei Glazunov of Google Project Zero.

This CVE relies on a vulnerability in the "`Load_SBit_Png`" function inside the `src/sfnt/pngshim.c`. The code inside pngshim.c is in charge of processing PNG files inside font files, applications such as image editing software, browsers and mostly any application that loads custom fonts inside the UI or for font and image editing purposes might be vulnerable to this attack.

By crafting a malicious font file containing a PNG, the attacker can gain an out-of-bounds memory write. Using this technique, the attacker can manipulate the application's memory, and run arbitrary code.

Analyzing the fix

When going through the fix inside `pngshim.c`, we see that the `Load_SBit_Png` has logic that rejects loading to memory images that are too large to process, but the logic that rejects the image is now before the function that allocates memory. This means that if, before the fix, the image was too large, it would allocate memory, then exit, whereas now it sees that it's too large and exits the function immediately.

```
@@ -332,6 +332,13 @@

     if ( populate_map_and_metrics )
     {
+     /* reject too large bitmaps similarly to the rasterizer */
+      if ( imgWidth > 0x7FFF || imgHeight > 0x7FFF )
+      {
+        error = FT_THROW( Array_Too_Large );
+        goto DestroyExit;
+      }
+
       metrics->width  = (FT_UShort)imgWidth;
       metrics->height = (FT_UShort)imgHeight;

@@ -340,14 +347,7 @@
       map->pixel_mode = FT_PIXEL_MODE_BGRA;
       map->pitch      = (int)( map->width * 4 );
       map->num_grays  = 256;
-
-     /* reject too large bitmaps similarly to the rasterizer */
-      if ( map->rows > 0x7FFF || map->width > 0x7FFF )
-      {
-        error = FT_THROW( Array_Too_Large );
-        goto DestroyExit;
-      }
-    }
+   }
```

Source: Chromium Source Code

This CVE's vulnerability is in the font processing library freetype2. To exploit it, the attacker must have prerequisite knowledge on the victim's machine, since the arbitrary code needed for the attack to work might be different on different OS versions and machines' CPUs.

The application must also have a processing logic inside that loads a font from user input. This attack vector is more common on personal computers where a user is tricked to install a specific font file, and on browsers where the server might send a specific font to be rendered in the user's browser.

For cloud security purposes, this CVE is relevant to applications that rely on providing image editing and font loading logics inside the code, specifically those that include user input to provide those fonts. These scenarios are less likely to happen on cloud platforms where most processing happens on the server side and user input is limited.

Since almost any operating system has some kind of font handling library for general purposes, it is very common to see these kinds of CVE flagged on your system, but when taking into account the exploitability path for this kind of attacks, we can safely assume that they are unlikely to be a real issue.

References:
FreeType Heap Buffer Overflow in Load_SBit_Png

10

CVE-2023-5217 - Heap buffer overflow in vp8 encoding in libvpx in Google Chrome

Self portrait uploading video

Reported on: Chrome

Relevant for: Chromium-based browsers end-users

Not relevant for: Linux Servers, Cloud Containerized Environments

Found by OX scanners in open-source containers: 26,027 occurrences

Bottom line: Remote exploitation issue in a video parsing library inside Chromium browsers. Exploitation is highly unlikely in Cloud Containerized Environments

This CVE was reported on September 27, 2023, by Clément Lecigne from Google's Threat Analysis Group (TAG).

CVE-2023-5217 is an out-of-bounds write vulnerability in the video codec library libvpx. When a malformed or maliciously crafted video file is processed, the vulnerability can lead to memory corruption, potentially allowing an attacker to execute arbitrary code.

libvpx is a video encoding library that is used in media players, web browsers, and video processing applications. It is most widely used in Google Chrome, which is this CVE's primary attack vector.

Analyzing the fix

By crafting a malicious video file running inside an HTML page, it is possible to trigger the VP8 encoding routine, specifically in the "`encode_mb_row`" function which is responsible for row processing, by triggering certain conditions the attacker can write beyond the allocated buffer and perform out-of-bounds memory write, which then could be used to run privileged code.

More precisely, the exploit involves crafted frames that issue a "reconfiguration" command, which modifies parameters such as video resolution, bitrate, or the number of threads used during encoding. By analyzing the patch, we can see the affected function `vp8_change_config` inside `vp8/encoder/onyx_if.c` has a thread updating safety check, to ensure that multiple reconfigurations do not abuse the changes in the number of encoding threads in order to gain an out-of-bounds write.

```
vp8/encoder/onyx_if.c
@@ -1447,6 +1447,11 @@ void vp8_change_config(VP8_COMP *cpi, VP8_CONFIG *oxcf)
{
  last_h = cpi->oxcf.Height;
  prev_number_of_layers = cpi->oxcf.number_of_layers;

+   if (cpi->initial_width) {
+     // TODO(https://crbug.com/1486441): Allow changing thread counts; the
+     // allocation is done once in vp8_create_compressor().
+     oxcf->multi_threaded = cpi->oxcf.multi_threaded;
+   }
  cpi->oxcf = *oxcf;

  switch (cpi->oxcf.Mode) {
```

Patch diff from Github

For cloud security purposes, this CVE is relevant for situations where the containerized environment actively uses libvpx to handle user-input videos and renders them, or for browser and video encoding software developers who ship their programs as a product to end clients. For other containerized environments and other intents and purposes, this CVE is highly unlikely to be exploitable or relevant for your cloud infrastructure.

References:
A PoC to trigger CVE-2023-5217 from the Browser WebCodecs or MediaRecorder interface.

Conclusions

After analyzing the above 10 KEV CVEs, we found that none apply to real-life exploitation scenarios in cloud containerized environments. Our analysis revealed:

6 CVEs

6 CVEs originally reported on Android, where most require Android-specific environments to reproduce, physical access for USB connections, and terminal access. While 2 of these 6 Android-related CVEs do apply to most OSs built on the Linux kernel, successful exploitation would require chaining them with additional vulnerabilities.

1 CVE

1 CVE initially reported in Apple's ecosystem browsers, where the environment's cookie-management logic was flawed - an issue that doesn't apply to cloud containerized environments.

3 CVEs

3 CVEs initially reported in libraries used by the Google Chrome browser, where exploited libraries can execute code on the client's device. This attack path is irrelevant for cloud containers, as most don't use these libraries for content processing and rendering.

These examples demonstrate that context matters significantly when reporting and prioritizing CVEs, even those labeled as Critical KEVs.

Recommendations for Security Teams

We suggest that before prioritizing remediation for a CVE, especially those from KEV or with high CVSS scores, security teams should analyze the vulnerability in the following way:

1

Determine the Environmental Context

Determine the context in which the CVE was reported and compare it to your environment:

- Was it found and reported for mobile devices while you operate cloud containers?
- Was it reported for client-side issues while your component operates server-side?
- Is the CVE related to network security, while your component isn't connected to the internet?

2

Evaluate Attack Path Probability in Your Environment

Check if the CVE details in NVD contain information about the patch and origin. Many CVEs imply directly or indirectly that they are part of an Android attack chain, making them non-reproducible in other environments like cloud containers.

3

Check for Published Exploits

Search for POC and exploit examples for the CVE. If none exist, there is a reduced likelihood (though not zero) that an attacker would develop the exploit themselves.

4

Evaluate Business Impact in Your Environment

Assess whether the vulnerable component is directly tied to sensitive information. Critical services managing sensitive information should be prioritized over other services in your environment.

Recommendations for CISA & Vulnerability Monitoring Orgs

When adding new CVEs to vulnerability catalogs, we recommend including contextual information to help security teams quickly assess the relevance of each vulnerability to their specific environments.

1

Include Platform-Specific Indicators

Examples: Android, iOS, Containers, Servers, Services, Web, Browsers.

2

Include CVE Origin Information

Specify if the vulnerability was discovered in environments such as Chrome browsers, the Linux kernel, Docker containers, etc., and whether it was part of an attack chain.

3

Include Attack Type Classification

Examples: Remote code execution, one-click attack, or attacks requiring local access.

This information would enable security teams to implement a more precise and efficient workflow when handling critical vulnerabilities in their environments.