A 0-click exploit chain for the Pixel 9 Part 3: Where do we go from here?

(/2026-JAN-14) Natalie Silvanovich

While our previous two blog posts provided technical recommendations for increasing the effort required by attackers to develop 0-click exploit chains, our experience finding, reporting and exploiting these vulnerabilities highlighted some broader issues in the Android ecosystem. This post describes the problems we encountered and recommendations for improvement.

Audio Attack Surface

The Dolby UDC is part of the 0-click attack surface of most Android devices because of audio transcription in the Google Messages application. Incoming audio messages are transcribed before a user interacts with the message. On Pixel 9, a second process `com.google.android.tts` also decodes incoming audio. Its purpose is not completely clear, but it seems to be related to making incoming messages searchable.

Both processes decode audio using all decoders available on the device, including the UDC, which is integrated by the OEMs of most devices, though the bulk of incoming messages use a small number of audio formats. In particular, it is very unlikely that an incoming message will contain audio in formats supported by the Dolby UDC, as Android devices do not provide encoders for these formats, and they are mostly used by commercial media, such as movies and TV shows. Removing the UDC and other uncommonly-used decoders from the 0-click attack surface of Android would protect users from the worst consequences of vulnerabilities in these codecs.

The explosion of AI-powered features on mobile phones has the potential to greatly increase their 0-click attack surface. While this trade-off can sometimes benefit users, it is important for mobile vendors to be aware of the impact on security. It is not uncommon for software changes to unintentionally increase the amount of code that can be exercised by attackers remotely. Ongoing review of how new features affect 0 and 1-click attack surfaces coupled with deliberate decisions are necessary to protect users.

Bug Discovery Time Frames

One surprising aspect of this research was how quickly we found both vulnerabilities used in the exploit chain. Project Zero reviewed the Dolby UDC as a part of a one-week team hackathon, and it took less than two days for Ivan to find CVE-2025-54957. Likewise, Seth found CVE-2025-36934 after less than one day of reviewing the BigWave driver.

Of course, it's easy to forget the effort that went into finding these attack surfaces– the Dolby hackathon required roughly three weeks of preparation to study the entry points of the codec and set-up tooling to debug it, and likewise, reviewing the BigWave driver involved a driver analysis tool that took roughly 4 weeks to develop. We also reviewed other audio codecs with mixed results before reviewing the Dolby UDC.

Still, the time investment required to find the necessary vulnerabilities was small compared to the impact of this exploit, especially for the privilege escalation stage. Moreover, a lot of the time we spent finding the UDC bug was a one-time cost that we expect to enable future research. The time needed to find the bugs for a 0-click exploit chain on Android can almost certainly be measured in person-weeks for a well-resourced attacker.

Android has invested a fair amount in the security of media codecs through vulnerability rewards programs and by fuzzing them with tools like OSS-Fuzz. While it is unlikely that fuzzing would have uncovered this particular UDC bug, as far as we know, Pixel's's fuzzing efforts do not cover the UDC. Gaps in vendors' understanding of their attack surface is a common source of 0-click vulnerabilities. While bugs occur in heavily-secured components, it can be easier for attackers to focus on areas that are overlooked. Android and OEMs could benefit from a rigorous analysis of its 0-click attack surface, and comprehensive efforts to fuzz and review them.

Drivers, on the other hand, continue (https://googleprojectzero.blogspot.com/2024/06/driving-forward-in-android-drivers.html) to (https://googleprojectzero.blogspot.com/2024/12/qualcomm-dsp-driver-unexpectedly-excavating-exploit.html) be (https://googleprojectzero.blogspot.com/2023/09/analyzing-modern-in-wild-android-exploit.html) a 'soft target' on Android. While Android, and its upstream driver vendors such as Samsung, Qualcomm, ARM and Imagination have made some efforts to improve driver security, they have been outpaced by attackers' ability to find and exploit these bugs. Google's Threat Intelligence Group (GTIG) has detected and reported 16 Android driver vulnerabilities being used by attackers in the wild since 2023. Driver security remains an urgent problem affecting Android's users that will likely require multiple approaches to improve. Rewriting the most vulnerable drivers in managed languages such as Rust, performing consistent security reviews on new drivers, reducing driver access from unprivileged contexts and making driver code more easily updatable on Android devices are likely all necessary to counter attacker's extensive capabilities in this area.

Ease of Exploitability

We estimate that exploiting the Dolby UDC vulnerability in the exploit chain took eight person-weeks and exploiting the BigWave driver vulnerability took 3 weeks for a basic proof-of-concept. This is not a lot of time considering the vast capabilities this type of exploit chain gives attackers. While many Android security features increased the challenge we faced in exploiting these issues, we were also surprised by two mitigations that did not provide their documented protection.

The Dolby UDC decoder process on the Pixel 9 lacked a seccomp policy, though this policy is implemented (https://android.googlesource.com/platform/frameworks/av/+/22c571b/services/mediacodec/minijail/seccomp_policy/mediacodec-seccomp-arm.policy) in AOSP and several other Android 16 devices we tested. If the policy in AOSP had been enforced on the Pixel 9, it likely would have added at least one person-month to the time spent

developing this exploit. For security features to be effective, it is important that they are verified on a regular basis, ideally for every release, otherwise it is possible that regressions go unnoticed.

We also discovered that kASLR is not effective on Pixel devices, due to a problem that has been known since 2016, detailed in this blog post (https://googleprojectzero.blogspot.com/2025/11/defeating-kaslr-by-doing-nothing-at-all.html) . Both Android and Linux made a decision to deprioritize development work that would have restored its effectiveness. This decision made exploiting the BigWave vulnerability easier, we estimate it would have taken roughly six weeks longer to exploit this vulnerability with effective kASLR, though with the additional time required, we may not have pursued it.

It is also notable that we have not been able to successfully exploit the Dolby UDC vulnerability on Mac or iPhone so far, as it was compiled with the -fbounds-safety (https://clang.llvm.org/docs/BoundsSafety.html) compiler flag, which added a memory bounds check that prevents the bug from writing out of bounds. Dolby should consider providing such compiler based protections across all platforms. Apple also recently implemented MIE, a hardware-based memory-protection technology similar to Memory Tagging (MTE), on new devices. While MIE would not prevent the Dolby UDC vulnerability from being exploited in the absence of -fbounds-safety due to UDC using a custom allocator, it would probabilistically hinder an iOS kernel vulnerability similar to the BigWave driver bug from being exploitable.

Pixel 8 onwards shipped (https://googleprojectzero.blogspot.com/2023/11/first-handset-with-mte-on-market.html) with MTE, but unfortunately, the feature has not been enabled except for users who opt into Advanced Protection mode, to the detriment of Pixel's other users. Apple's inclusion of memory protection features, despite their financial and performance cost, clearly paid off with regards to protecting its users from the UDC exploit as well as possible kernel privilege escalation. There is the potential to protect Android users similarly.

Another remarkable aspect of this exploit chain is how few bugs it contains. Gaining kernel privileges from a 0-click context required only two software defects. Longer exploit chains are typically required on certain platforms because of effective sandboxing and other privilege limitation features. To bypass these, attackers need to find multiple bugs to escalate privileges through multiple contexts. This suggests potential sandboxing opportunities on Android, especially with regards to reducing the privileges of the frequently-targeted media decoding processes.

Patch Timeframe

Both vulnerabilities in this exploit chain were public and unfixed on Pixel for some time. The UDC vulnerability was reported to Dolby on June 26, 2025, and the first binary fixes were pushed to ChromeOS on September 18, 2025. Pixel shared with us that they did not receive binary patches from Dolby until October 8, 2025. We disclosed the bug publicly on October 15, 2025, after 30 days patch adoption time, as per Project Zero's disclosure policy (https://googleprojectzero.blogspot.com/p/vulnerability-disclosure-policy.html) . Samsung was the first mobile vendor to patch the vulnerability, on November 12, 2025. Pixel did not ship a patch for the vulnerability until January 5, 2026.

It is alarming that it took 139 days for a vulnerability exploitable in a 0-click context to get patched on any Android device, and it took Pixel 54 days longer. The vulnerability was public for 82 days before it was patched by Pixel.

One cause of the slow fix time was likely Dolby's advisory (https://professional.dolby.com/siteassets/pdfs/dolby-security-advisory-CVE-2025-54957-Oct-14-25.pdf) . We informed Dolby that this issue was highly exploitable when we filed the bug, and provided status updates, including technical details of our exploit, as the work progressed. Despite this, the advisory describes the vulnerability's impact as follows:

We are aware of a report found with Google Pixel devices indicating that there is a possible increased risk of vulnerability if this bug is used alongside other known Pixel vulnerabilities. Other Android mobile devices could be at risk of similar vulnerabilities.

This is not an accurate assessment of the risk this vulnerability poses. As shown in Part 1 of this blog post, the vulnerability is exploitable on its own, with no additional bugs. Dolby is likely referring to the fact that additional vulnerabilities are required to escalate privileges from the mediacodec context on Android, but almost all modern vulnerabilities require this, and we informed them that there is strong evidence that exploit vendors have access to kernel privilege escalation vulnerabilities on most Android devices. No other vendor we've encountered has described a vulnerability allowing code execution in a sandboxed context as requiring the bug to be "used alongside other known [...] vulnerabilities."

Dolby's advisory also says:

For other device classes, we believe the risk of using this bug maliciously is low and the most commonly observed outcome is a media player crash or restart.

We believe this understates the risk of the vulnerability to other platforms. It's difficult to determine the "risk of [attackers] using this bug maliciously," even well-resourced threat analysis teams like GTIG have difficulty determining this for a particular bug with any accuracy. Moreover, "most commonly observed outcome is a media player crash or restart" is true of even the most severe memory corruption vulnerabilities. This is why most security teams classify vulnerabilities based on the maximum access an attacker could achieve with them. Except for on Apple devices, where the UDC is compiled with -fbounds-safety, this bug enables code execution in the context that the UDC runs. The impact of this bug on users is also platform-dependent, for example, it presents a higher risk on Android, where untrusted audio files are processed without user interaction than on a smart TV which only plays audio from a small number of trusted streaming sources, but this doesn't change that an attacker can generally achieve code execution by exploiting this bug. Ideally, Dolby would have provided its integrators with this information, and allowed them to make risk decisions depending on how they use and sandbox the UDC.

It's not clear what information Dolby provided Android and Pixel, but Android publishes its priority matrix here (https://source.android.com/docs/security/overview/updates-resources#rating_modifiers) . Since mediacodec is considered (https://android-developers.googleblog.com/2019/05/queue-hardening-enhancements.html) a constrained context, when we reported it, the UDC bug fell into the category of "remote arbitrary code execution in a constrained context", and it was rated *Moderate*. Conversely, Samsung rated this bug as *Critical*. Android shared with us they recently updated their priority matrix, and future vulnerabilities of this type will be classified as *Critical*.

We reported the BigWave vulnerability to Pixel on June 20, 2025 and it was also rated *Moderate*. As per the matrix above, "Local arbitrary code execution in a privileged context, the bootloader chain, THB, or the OS kernel" makes this bug *High* base severity, but the severity modifier "Requires running as a privileged context to

execute the attack" was applied. While the modifier text states a "privileged context", our experience is that the modifier is frequently applied to vulnerabilities that are not directly accessible from an unprivileged context, including those accessible from constrained contexts like mediacodec. The severity was changed to *High* on September 18, 2025 and a fix was shipped to devices on January 6, 2026. We shared the bug publicly after 90 days, on September 19, 2025, in accordance with our disclosure policy.

While different software vendors and projects have different philosophies with regards to vulnerability prioritization, deprioritizing both of these bugs left users vulnerable to a 0-click exploit chain. Some vendors make bugs in 0-click entrypoints high priority, while others choose to prioritize bugs in the sandboxes that isolate these entrypoints. There are benefits and downsides to each approach, but vendors need to prioritize at least one bug in the chain in order to provide users with a basic level of protection against 0-click exploits.

This type of diffusion of responsibility isn't uncommon in vulnerability management. Series of bugs that can be combined to cause severe user harm are often individually deprioritized, and codec vendors like Dolby often consider it largely the platform's responsibility to mitigate the impact of memory corruption vulnerabilities, while platforms like Android rely too heavily on their supply chain being bug-free. Developers of software with the best security posture tend to take the stance that all external software should be considered compromised, and invest in protecting against this eventuality. This and other defense-in-depth approaches is what makes exploit chains difficult for attackers, and has the best chance of protecting users.

Patch Propagation

Even though the Dolby UDC vulnerability was eventually patched by Pixel, it will take some time for all other Android users to receive an update. This is because mobile updates are gated on a variety of factors, including carrier approval, and not every OEM provides security updates in a timely manner, if at all.

Android has a mechanism to update specific system libraries that circumvents this process called APEX (https://source.android.com/docs/core/ota/apex) . Libraries packaged with APEX can be updated by Google directly through the Google Play Store, leading to a much faster update cycle. Since the UDC does not ship as part of Android, it does not have this capability, though this could be changed with significant licensing and shipping ownership changes.

Conclusion

It's easy to look at a 0-click exploit chain like the one we developed and see a unique technical feat, when what it really reveals is capabilities currently available to many attackers. While developing the exploit was time-consuming, and required certain technical knowledge, it involved nothing that isn't achievable with sufficient investment. All considered, we were surprised by how small that investment turned out to be.

It can also be tempting to see this exploit as a series of esoteric, difficult-to-detect errors, but there are actions that can reduce the risk of such exploits, including analysis and reduction of 0-click attack surface, consistent testing of security mitigations, rapid patching and investment in memory mitigations.

Most humans alive today trust their privacy, financial well-being and sometimes personal safety to a mobile device. Many measures are available that could protect them against the most dangerous adversaries. Vendors

should take action to reduce the risk of memory-corruption vulnerabilities to the platform and deliver security patches to users in a reasonable timeframe.■

make zeroday hard.