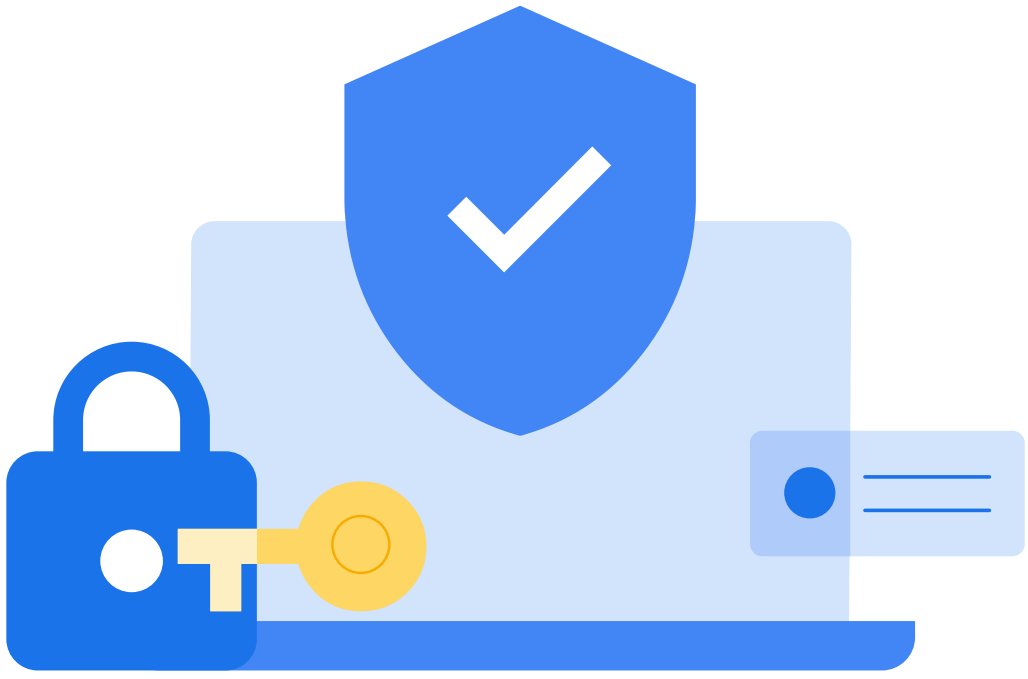
Google

An Overview of Google's Commitment to Secure by Design

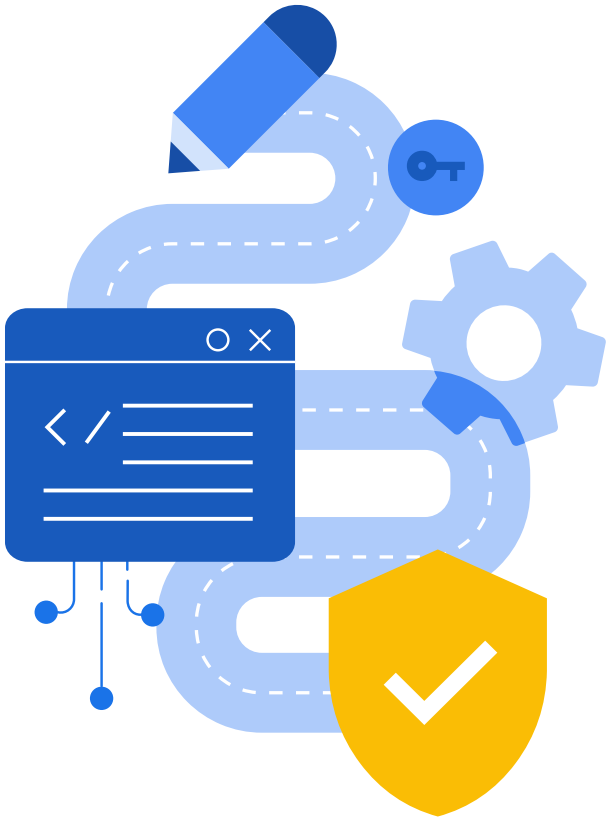


Safer with Google

Contents

Introduction

In an increasingly interconnected world, the concept of Secure by Design has emerged as the critical path towards designing, implementing, and maintaining resilient systems. For over two decades, Google has been following Secure by Design principles, embedding security into our products and development processes, and sharing our journey with the world.



In May 2024, Google signed the Cybersecurity and Infrastructure Security Agency's (CISA) Secure by Design pledge. As part of this commitment, we are sharing our approach to Secure by Design in this paper, focusing on what we have already accomplished over the past two decades towards fulfilling the seven goals of the CISA pledge. We will continue publishing as our journey progresses.

"Secure by Design" versus "Secure by Default"

"Secure by Default" and "Secure by Design" are often used interchangeably, but they actually represent distinct approaches to building secure systems. While both aim to minimize vulnerabilities and enhance security, they differ in scope and implementation.

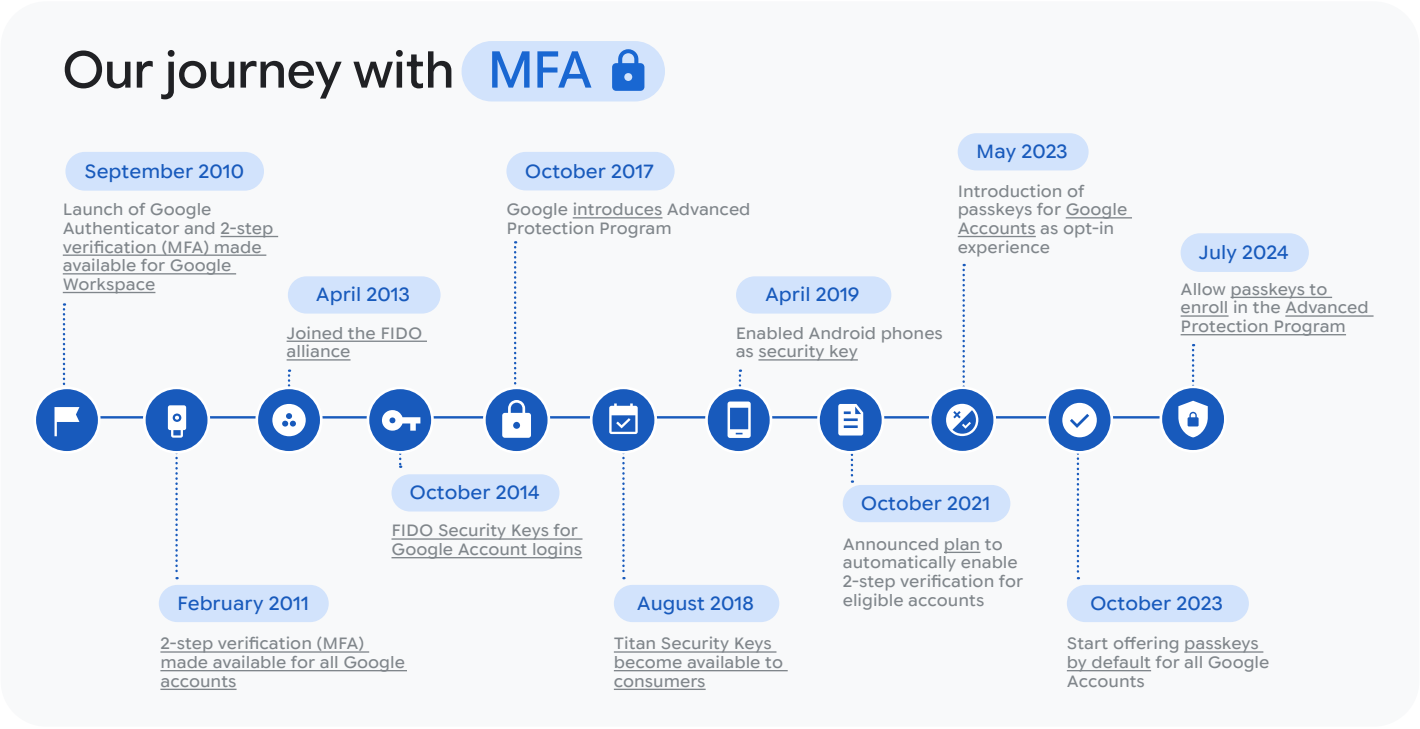
Secure by Default focuses on ensuring that the system's out-of-the-box default settings are set to a secure mode, minimizing the need for users or administrators to take actions to secure the system. This approach aims to provide a baseline level of security for all users.

Secure by Design is a proactive approach that emphasizes incorporating security considerations throughout the entire software development lifecycle. It's about anticipating potential threats and vulnerabilities early on and making design choices that mitigate those risks. This approach involves using secure coding practices, conducting security reviews, and embedding security throughout the design process. Secure by Design is an overarching philosophy that guides the development process, ensuring that security is not an afterthought but an integral part of the system's DNA.[1] This paper is primarily focused on Google's approach to Secure by Design.

[1] For more details, see: Kern, Christoph. 2024. "Secure by Design at Google". Google Security Engineering Whitepaper. https://storage.googleapis.com/gweb-research2023-media/pubtools/7661.pdf

Pledge Goal 1: Multi-Factor Authentication (MFA)

Password theft, facilitated through tactics like phishing, poses a serious risk to the online ecosystem[2] and Google supports CISA's goal of increasing use of Multi-Factor Authentication (MFA). MFA is a major milestone on the path towards ubiquitous, phishing-resistant authentication.



In February 2011, Google launched SMS-based MFA as a free option for all Google users. Research shows that SMS-based MFA can help block 100% of automated bots, 99% of bulk phishing attacks, and 66% of targeted attacks. While SMS-based MFA has had a significant impact on user security, we have seen evidence that it is still vulnerable to phishing.[3]

To address fundamental weaknesses in authentication mechanisms, in 2013, Google joined the Fast IDentity Online (FIDO) Alliance, an industry coalition including companies like Mastercard and Apple, focused on open standards and protocols to deliver strong MFA solutions. Since then, Google has contributed to FIDO's initiatives for security keys and passkeys. Security keys enable hardware-based MFA and are effective at reducing the likelihood of phishing attacks. In 2019, Google teamed up with researchers from New York University and the University of California, San Diego for a year-long study on wide-scale and targeted attacks. Zero users that exclusively used security keys fell victim to targeted phishing during our investigation.

Google has also worked with the FIDO Alliance on the development of passkeys. Passkeys are a safer and easier alternative to passwords, allowing users to sign in to apps and websites with a fingerprint or facial recognition. They do not require special hardware, improving overall usability, but are still based on the same FIDO protocols as security keys. Since launching passkeys, they have been used to authenticate users more than 2.5 billion times across over 750 million Google Accounts. Passkeys are available free of charge to all of our users.

[2] In 2013, we published on the epidemic of poor password security. Grosse, E. and Upadhyay, M. "Authentication at Scale." In IEEE Security & Privacy, vol. 11, no. 1, pp. 15-22, Jan.-Feb. 2013, doi: 10.1109/MSP.2012.162.
[3] CISA's Cyber Safety Review Board has called for broad adoption of FIDO(2) solutions and the deprecation of SMS for delivery of MFA codes. See "REVIEW OF THE ATTACKS ASSOCIATED WITH LAPSUS$ AND RELATED THREAT GROUPS", July 24, 2023 pg 32-33.

Use of MFA in our Products

At Google, we have seen first-hand how automatically enabling MFA for users decreases the rates of account hijacking due to password theft.

Google auto-enrolls eligible consumer users[4] into account-level MFA (also called 2-Step Verification or "2SV"). As a result, MFA is required when signing into a Google Account from a new device. Since 2021, Google has automatically enrolled over 400 million consumer accounts into MFA. Additionally, Google also requires MFA for any sign-in session that appears out of the ordinary to our risk engine, irrespective of whether the user is specifically enrolled in MFA. In practice, this means MFA is available, and in use, free of charge to all users who have a phone number or other means of verification on file. More than 70% of Google Accounts, owned by people regularly using our products, automatically benefit from this feature.

> More than 70% of Google Accounts, owned by people regularly using our products, automatically benefit from this feature.

Google provides defaults which we believe are in the best interest of our customers, as well as options for customers to adjust these defaults if they wish. Google Cloud Administrators can directly enforce the use of MFA for some or all of their users, and also restrict the types of MFA methods that may be used. For example, some users may only be allowed to use phishing-resistant security keys or passkeys, while others may be allowed to use any method except SMS-based MFA. Administrators also have the option of enforcing MFA after a SAML sign-in, offering protection against the scenario where an Identity Provider has been compromised.

In 2023, Google enforced MFA for all Workspace reseller administrator accounts, and started enforcing MFA for customer administrator accounts as well. For Google Cloud, additional efforts to increase MFA adoption are underway, moving to a model requiring MFA for all users. Near the end of 2024, Google Cloud will be encouraging all customers to enroll and enable MFA via in-console messaging. Starting in 2025, Google will roll out mandatory MFA enforcement for all Google Cloud users that log in with a password. Finally, later in 2025, Google will roll out MFA enforcement for all users who federate authentication to Google Cloud.

All our MFA features, Enterprise Single Sign-on, and ongoing work on default policies **are available free of charge** to all Workspace and Google Cloud customers.

MFA in the Google Enterprise

Google adopted FIDO-backed security keys in 2013 to protect our internal accounts and systems.

Every member of our workforce, including third-parties with access to our systems, is required to use a security key managed and issued by us. Combined with additional defense-in-depth measures, security keys have enabled us to successfully defend against sophisticated and serious attacks.

[4] Eligible consumer users are users that have provided us with enough information to deliver MFA as a service, for example, a phone number for SMS-based MFA. If a user has not provided us with this information (as is the case for many early accounts), then we ask the user for this information in the course of further protecting the account.

Pledge Goal 2: Default Passwords

Default passwords used in software and hardware can be easily discovered by threat actors. This basic vulnerability is an area Google has addressed by implementing best practices across our hardware and software ecosystem, and including it as a component of our security reviews. If a default password was present in our products, we would treat that as a vulnerability and handle it through our established processes for remediating issues.

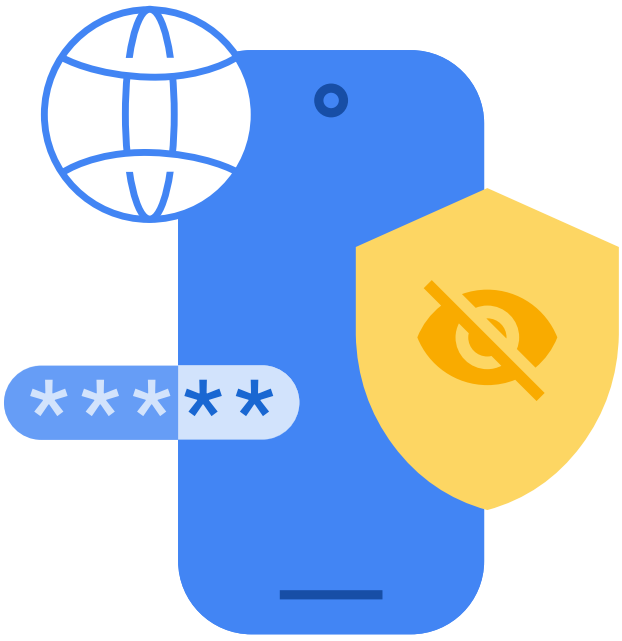
Hardware

When designing retail hardware-based product lines, Google uses a system setup and maintenance life cycle that links the device to the user's Google account and does not rely on preconfigured passwords on these devices. For instance, configuring a new Nest smart home device, Google Pixel phone, Google TV streamer, or Fitbit wearable requires the user to log in with an individual Google account. Additionally, smart home device setup using a mobile app such as Google Nest or Google Home requires the device to be within Bluetooth range of the mobile phone and the device needs to be on the local WiFi network. For devices like Google Nest cameras, routers, and Chromecast streamers, a code on the device (e.g., a QR code) needs to be scanned or typed in by the user to prove physical possession before being able to link the product to a home data structure.

Smart home devices are managed by the user via a Google mobile app (e.g., Nest, Google Home) and can be accessed via the mobile app or a Web app. This access relies on a Google account and device linking, not on default passwords. In addition, there is no support for remote administration, for example, in a corporate environment. Factory reset logic is available and allows the user in possession of the device to wipe user data and unlink devices from their account. Return Merchandise Authorization (RMA) servicing for these devices uses factory reset within processes defined by Google. Similarly, RMA for Pixel devices is driven by systems owned by Google that enable an RMA agent to safely handle returned devices, including the removal of user data and factory reset.

Software and Services

Google's software-based services are similarly set up and accessed using a Google Account. Enterprise Cloud-based services such as Workspace and Google Cloud are managed by organization administrators. The setup process for these accounts does not involve default passwords. Cloud-based API services leverage a centralized Cloud IAM service that relies on industry-standard authentication mechanisms (e.g., OAuth 2.0, OpenID Connect), eliminating the need for additional credentials. When a domain administrator creates a new user in the Google Workspace Admin Console, Google automatically generates a strong password.



Pledge Goal 3: Reducing Entire Classes of Vulnerability

Google's products and services are built on top of platforms, such as our custom production environment, Google Cloud, and Android, as well as platforms defined by public standards, like the Web Platform and foundational Internet protocols, all of which have enabled our ability to address vulnerabilities at scale over time. As part of our platform work, Google has built simple, safe, and reliable libraries, abstractions, and application frameworks for our developers to use with the goal of eliminating classes of vulnerabilities in the code they write.

Safe Coding

The principle of Safe Coding is based on the idea that APIs and platform features should be inherently safe in any reasonable usage, and not only when developers carefully adhere to complex and difficult-to-reason-about secure-coding guidelines. A key observation informing the Safe Coding approach is that common classes of vulnerabilities in software applications tend to arise from the design of APIs, libraries, and platforms that developers use when building and deploying these applications. The design of an API or platform feature can be inherently risky, in that it requires the developer to carefully write the code that uses the API so as to ensure a critical security property or invariant; if the developer makes a subtle mistake and their code fails to ensure the security property, an exploitable vulnerability might be present.

In other words, each use of the risky API or platform feature in question is potentially vulnerable, unless the developer carefully adhered to secure-coding and -configuration guidelines and avoided making any mistake. Given a substantial number of potential vulnerabilities, some actual vulnerabilities tend to sneak into code or configuration. And once a vulnerability has been introduced, efforts to discover and fix it (such as code review, static or dynamic code analysis) are inherently incomplete and unable to find every single instance.

At Google, we've found that the most effective approach to address classes of vulnerabilities due to potentially pervasive coding or configuration errors (such as bugs in code relying on widely-used APIs or platform features) is to replace risky, mistake-prone APIs and platform features with functionally equivalent APIs that are designed to be safe by design, and which protect developers from the risk of accidentally introducing vulnerabilities – thereby enabling Safe Coding.

Safe Developer Ecosystems

To fully realize the benefits of Safe Coding, it is helpful to consider all aspects of the developer ecosystem in which applications are designed, developed, and deployed. [5] This includes programming languages, software libraries, application frameworks, source repositories, build and deployment tooling, as well as the deployment platform and its configuration surfaces.

Most of Google's large scale user-facing services, including Search, Ads, Gmail, Docs, as well as core control- and data-plane components of Google Cloud, are developed in a shared repository using a trunk-based development paradigm.[6] Foundational components of this developer ecosystem are developed and maintained centrally by teams of domain experts. This includes security-critical and -relevant libraries (such as cryptographic primitives and protocols, authentication, authorization, RPC servers and clients, and so forth), as well as higher-level application frameworks that provide opinionated assemblies of vetted components for classes of applications and services, such as Web frontends and microservice backends.

This centralization enables domain experts to instill best practices across classes of applications, in domains including security and privacy, but also reliability, scalability, code health, and maintainability. For critical properties, the centrally managed toolchain can be leveraged to ensure adherence to best practices with a high degree of assurance. For example, certain risky, subtle low-level APIs and framework features should not be used in general application code unless truly necessary, and if so, only when subject to domain expert review. In Google's shared repository, this practice is upheld through a feature of the central build system, which supports restrictions on packages that may depend on such a low-level API; the repository's code review workflow ensures that additions to the allowlist are reviewed by appropriate domain experts. Similarly, custom, domain-specific code conformance checks are enforced through plugins in the centrally managed compiler toolchain. Higher-level application frameworks are structured to require expert review before an application developer can modify critical safe default configurations. Finally, the shared repository provides tooling for automated large scale changes which is used to retrofit improvements, such as migrating uses of risky APIs to safer alternatives.

Google's developer ecosystem is designed to support development of common classes of applications, including Web applications, backend microservices, API frontends, and mobile applications, with 100s to 1000s of individual applications and services in each class. This broadly used developer ecosystem is complemented by bespoke developer ecosystems tuned to the specific needs of products like Android, Chrome, and ChromeOS, and certain components of Google Cloud. These bespoke developer ecosystems similarly provide libraries and frameworks in support of security properties, for example, isolation of and mediated communication between subsystems in Chrome,[7, 8] memory safety in Android and Chrome (discussed below), or third-party dependency management in Cloud products.

It is not straightforward to (re)-design developer ecosystems and their components, APIs, and platforms to provide a Safe Coding environment.

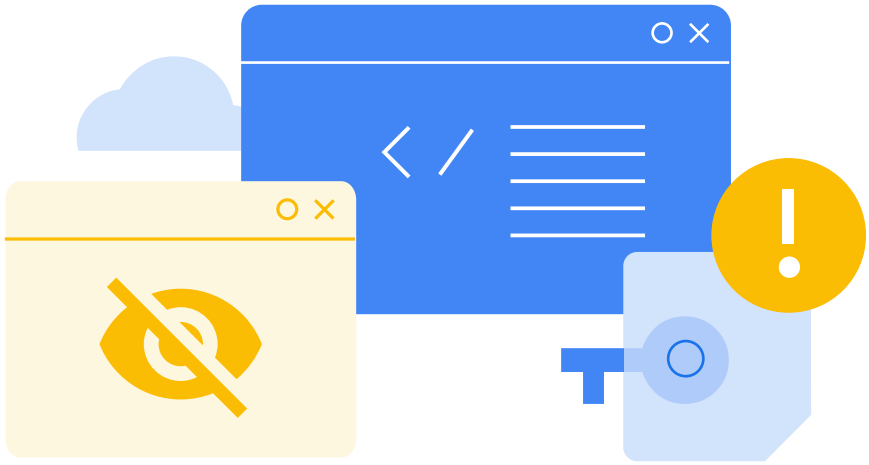
> Google's experience applying Safe Coding to several classes of security defects shows that it is possible, and can be done in a cost-effective manner.

However, Google's experience applying Safe Coding to several classes of security defects over the past decade shows that it is possible, and can be done in a cost-effective manner.

We discuss many of these methodologies in Building Secure and Reliable Systems and highlight some key areas of long-term investment in the following sections.

[5] Kern, Christoph. "Developer Ecosystems for Software Safety." Communications of the ACM 67.6 (June 2024), 52-60. https://doi.org/10.1145/3651621.
[6] Potvin, Rachel and Levenberg, Josh. 2016. "Why Google stores billions of lines of code in a single repository." Commun. ACM 59, 7 (July 2016), 78–87. https://doi.org/10.1145/2854146.

Cross-Site Scripting (XSS)



Cross-site scripting (XSS) has been one of the major web security vulnerabilities for over a decade, ranking 2nd in the Stubborn Weaknesses in the CWE Top 25, and being the most commonly reported vulnerability class across several popular online bug bounty platforms. This is why Google focused on this vulnerability class and proactive measures to drastically reduce its occurrence in our major products. Because core web technologies allow unsafely mixing code and data (for example, HTML permits the loading of <script> elements which execute arbitrary JavaScript code alongside harmless presentational markup), XSS is common whenever developers compose web pages using data outside of their control without first escaping, sanitizing, or otherwise validating it. Any malicious scripts injected into a web page execute in users' browsers with the privileges of the affected web origin, giving the attacker full control over a user's session and allowing them to view or edit the user's data, or perform other malicious actions in the vulnerable application.

At Google, our approach to addressing XSS in sensitive services is two-fold and relies on hardening internal application development frameworks using the Safe Coding approach, and developing and enabling defense-in-depth anti-XSS mechanisms built directly into web browsers. Google's internal frameworks replace unsafe APIs with inherently secure alternatives, such as those provided by strictly autoescaping template systems or client-side frameworks, such as the safevalues library. These safe APIs are designed to ensure the absence of specific vulnerability classes and their usage is verified during compile-time (e.g., through tools such as the safety-web plugin and internal equivalents).

Our hardened web frameworks enable these protections by default and are designed to prevent developers from disabling them without undergoing a review by security experts.

To complement these compile-time security features, Google's application development frameworks also make use of core web platform defenses enforced at run-time by users' browsers, such as Content Security Policy and Trusted Types. Google actively collaborates with the W3C and web browser makers to develop and enhance these web platform security features and enable them by default for all of our recommended application development frameworks. Furthermore, Google proactively backports these security features at scale to protect existing applications (brownfield applications) from XSS vulnerabilities. This ensures that even applications launched before the widespread adoption of safe defaults benefit from enhanced security measures.

These compile-time and run-time security controls work together to provide defense-in-depth protection against XSS in sensitive services, ensuring that even if one security mechanism were to fail, others remain active and will prevent or mitigate the impact of a vulnerability. Data supports the conclusion that our proactive approach to addressing XSS has been highly effective. In the past three years, **for hundreds of complex web applications that are built on Google's hardened and safe-by-design frameworks, we've averaged less than one XSS report per year in total**. As an example, Google Photos was developed on secure-by-design frameworks from the outset, and has had no XSS vulnerabilities discovered in its codebase during its full lifetime. Additionally, by proactively backporting these security features at scale to existing applications, we've reduced XSS vulnerabilities in core Google products by 90% over the past decade.

> Google focused on this vulnerability class and proactive measures to drastically reduce its occurrence in our major products.

[7] Barth, Adam et al. 2008. "The security architecture of the Chromium browser." Technical report. Stanford University. https://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf.
[8] Reis, Charles et al. 2019. "Site isolation: Process separation for web sites within the browser." 28th USENIX Security Symposium (USENIX Security 19).https://www.usenix.org/system/files/sec19-reis.pdf.

Additional Classes of Web Application Vulnerabilities

Beyond injection attacks such as XSS, a wide range of common web application vulnerabilities stem from inadequate isolation guarantees provided by the web platform itself. This includes well-known threats such as Cross-Site Request Forgery (CSRF) and clickjacking, as well as emerging attack vectors such as cross-site leaks and microarchitectural issues that allow bypassing browser-enforced web security boundaries, including Spectre and its variants.

Such isolation issues occur when distinct web applications opened by the same browser lack logical separation, allowing malicious websites to interact with sensitive services in unexpected ways. This enables malicious actors to exploit the interplay between these web applications, leading to unauthorized access to user data or execution of unintended actions on behalf of users logged into a sensitive web application.

Certain vulnerabilities, such as CSRF, can be mitigated through custom protections such as requiring CSRF tokens for HTTP methods that modify state. However, many isolation-based vulnerabilities necessitate the use of "native" web platform security features such as SameSite cookies, X-Frame-Options (XFO), Cross-Origin Opener Policy (COOP), Cross-Origin Resource Policy (CORP), and Fetch Metadata Request Headers.

Google has participated in the design and enhancement of many of these web platform mechanisms, made best practices available to the industry (example), and, as with all other web application security controls, enabled them by default within our hardened frameworks.

To facilitate targeted remediation, we utilize our Security Signals infrastructure to identify endpoints that require protections but haven't yet enabled them. This strategic approach allows us to prioritize our efforts based on application sensitivity and deploy security controls precisely where they are most needed. This data-driven approach also informs Google's large-scale efforts to backport security controls, ensuring that even legacy applications benefit from the latest web platform defenses.

SQL Injection (SQLi)

SQL injection (SQLi) is a common class of security vulnerability, ranking 3rd in the Stubborn Weaknesses in the CWE Top 25. In 2013, Google embarked on an effort to address the risk of SQL injection vulnerabilities in applications built on our large-scale production databases in the shared developer ecosystem described in the above section on Safe Developer Ecosystems. As a result, we have not seen any SQL injection vulnerabilities across these 100s of applications over the past decade.

SQLi vulnerabilities arise from a coding error whereby untrustworthy string fragments are incorporated into a SQL query, potentially allowing attackers to inject malicious query fragments that result in unauthorized access to or modification of data.

The systemic root cause for this vulnerability lies in the typical design of database query APIs, which accept the SQL query in the form of a general-purpose string. This API design places responsibility on developers writing application code that dynamically constructs a SQL query (a common pattern in web applications) to ensure that all query fragments are trustworthy and safe to use as part of a query. In complex applications, it's easy to make a mistake resulting in a vulnerability, and after-the-fact code review and static or dynamic analysis are inherently unable to reliably discover all vulnerabilities.

Google addressed this systemic root cause by re-designing the API: We changed database query APIs available to developers in our shared repository, in particular the SQL query APIs for the large-scale databases (Spanner[9] and F1[10]) that are widely used as persistence layers for Google's user-facing applications. These APIs no longer accept SQL queries in the form of a simple string. Instead, secure-by-design SQL APIs require developers to supply queries in the form of a custom data type, TrustedSqlString, that represents safely constructed queries. The custom type's constructors and builder APIs are designed to ensure this property for all instances of the type.[11] The implementations of the data type and its constructors are curated and reviewed by security experts, and our shared repository and its build system ensure that application code indeed uses these secure-by-design Spanner and F1 query APIs, while occasionally necessary uses of lower-level, potentially unsafe query APIs are subject to security review and approval.

If a developer were to write code that supplies an unsafely constructed query string to a secure-by-design database API, it would be rejected by the compiler's type checker, and fail to build. Therefore, Google gains a high degree of confidence that every application in our shared repository that uses secure-by-design database APIs (such as the Spanner SQL query API) is free of SQL injection vulnerabilities.

[9] Bacon, D. F. et al. 2017. "Spanner: Becoming a SQL System." In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17). Association for Computing Machinery, New York, NY, USA, 331–343. https://doi.org/10.1145/3035918.3056103.
[10] Shute, Jeff et al. "F1: A distributed SQL database that scales." Proceedings of the VLDB Endowment 6.11 (2013): 1068-1079. https://doi.org/10.14778/2536222.2536232.
[11] An implementation in the Go language that illustrates this approach is available at https://github.com/google/go-safeweb/tree/master/safesql.

Memory Safety Vulnerabilities

Memory safety vulnerabilities have consistently represented around two-thirds of software vulnerabilities in memory-unsafe languages like C and C++. [12] The CWE project's list of 15 most stubborn software weaknesses includes five classes of memory safety defects.

Google's journey with memory safety is intertwined with the evolution of the software industry itself. In our early days, we recognized the importance of balancing performance with safety. This led to the early adoption of memory-safe languages like Java and Python, and the creation of Go. Today these languages comprise a large portion of Google's code base, providing memory safety among other benefits. We continue to invest in our memory-safe language offerings to prevent the introduction of new memory safety vulnerabilities by design, using Safe Coding principles.

A large component of this push is to expand the adoption of Rust in places where C++ was previously the language of choice, due to high performance demands. Google is also investing in improved interoperability between memory-safe languages and C++ to accelerate this transition through tools like Crubit and experimental languages like Carbon.

> We continue to invest in our memory-safe language offerings to prevent the introduction of new memory safety vulnerabilities by design.

We have outlined our perspective on memory safety in the past and our recent blog post shows our strategy for advancing memory safety at Google.

Briefly, Google's strategy takes a two-pronged approach:

1. Enabling high-performance future code to be written in a memory safe language, combined with targeted rewrites of security-critical or problematic components. To that end, we are investing to expand Rust usage at Google. This will unlock the use of memory safe languages in low-level code environments where C and C++ have typically been the language of choice.
2. Mitigating the risk of memory-unsafe code. Alongside proactive bug detection, workload isolation, and exploit mitigation, we are prioritizing the elimination of subclasses of memory-safety vulnerabilities in our memory-unsafe code to the extent possible, using secure-by-design principles. For instance, we are working to eliminate spatial safety vulnerabilities by retrofitting bounds checking.

Google has seen benefits of making these improvements over time. For example, Android has seen a decrease in the number of memory safety vulnerabilities reported between 2019 - 2024 (from 76% to 24% of Android's total vulnerabilities). In Chrome, we have been rolling out MiraclePtr, a new smart pointer that quarantines allocations that have known pointers. This has mitigated 57% of use-after-free vulnerabilities in privileged processes, and has been linked to a decrease in in-the-wild exploits. [13] MiraclePtr is considered a declarative security boundary and a valid submission of a MiraclePtr bypass is now eligible for a vulnerability reward of $250,128.

As we advance in our pursuit of memory safety, we will continue to share updates on our progress.

[12] https://www.memorysafety.org/docs/memory-safety/#how-common-are-memory-safety-vulnerabilities
[13] https://security.googleblog.com/2024/01/miracleptr-protecting-users-from-use.html and https://blog.google/technology/safety-security/a-review-of-zero-day-in-the-wild-exploits-in-2023/

Insecure Use of Cryptography

As we wrote in Building Secure and Reliable Systems, cryptographic code is particularly prone to subtle mistakes. Many cryptographic primitives (such as cipher and hash algorithms) have failure modes that are difficult for non-experts to recognize. For example, in certain situations where encryption is combined improperly with authentication (or used without authentication at all), an attacker who can only observe whether a request to a service fails or is accepted can nevertheless use the service as a so-called "decryption oracle" and recover the clear text of encrypted messages. A non-expert who is not aware of the underlying attack technique has little chance of noticing the flaw: the encrypted data looks perfectly unreadable, and the code is using a standard, recommended, and secure cipher like AES. Nevertheless, because of the subtly incorrect usage of the nominally secure cipher, the cryptographic scheme is insecure. In our experience, code involving cryptographic primitives that was not developed and reviewed by experienced cryptographers commonly has serious flaws.

This led Google to develop Tink: a library that enables engineers to use cryptography safely in their applications. Tink was born out of our extensive experience working with Google product teams, fixing vulnerabilities in cryptography implementations, and providing simple APIs that engineers without a cryptographic background can use safely.

Tink also provides a solution for key management, integrating with Google's Cloud Key Management Service (KMS) and AWS Key Management Service. Many cryptographic libraries make it easy to store private keys on disk, and make adding private keys to your source code even easier – practices that are strongly discouraged. Even if you run "keyhunt" and "password hunt" activities to detect and scrub secrets from your codebase and storage systems, they are point-in-time and will be incomplete, leading to repeated key management-related incidents. In contrast, Tink's API encourages use of a key management service. Using key material directly is only possible using special APIs which are easily audited (or controlled by an allowlist).

Google uses Tink to secure the data of many products, and it is now the recommended library for protecting data within Google and when communicating with third parties. By providing abstractions with well-understood properties (such as "authenticated encryption") backed by well-engineered implementations, it allows security engineers to focus on higher-level aspects of cryptographic code without having to be concerned with lower-level attacks on the underlying cryptographic primitives. We use our build system's constraints on package dependencies to block un-reviewed use of certain cryptographic libraries, and we have implemented custom static checks to flag certain common unsafe usage patterns.[14]

Google also employs formal verification to produce high-assurance cryptographic code for use in our cryptographic libraries. [15, 16, 17] This provides mathematical proof that cryptographic operations are functionally correct and free of critical implementation vulnerabilities. Formal verification allows us to conclusively rule out bugs and vulnerabilities early in the development process.

Tink reduces the potential for common cryptography pitfalls, and provides secure APIs that are easy to use correctly and hard(er) to misuse. The following principles guided Tink's design and development:

- **Secure by Default:** The library provides an API that's hard to misuse. For example, the API does not permit reuse of nonces in Galois Counter Mode – a fairly common but subtle mistake that was specifically called out in RFC 5288, as it allows authentication key recovery that leads to a complete failure of the AES-GCM mode's authenticity.
- **Usability:** The library has a simple and easy-to-use API, so a software engineer can focus on the desired functionality – for example, using block and streaming Authenticated Encryption with Associated Data (AEAD) primitives.
- **Readability and Auditability:** Functionality is clearly readable in code, and Tink maintains control over employed cryptographic schemes.
- **Agility:** Tink has built-in key rotation and supports deprecation of obsolete/broken schemes. This further facilitates migration to new algorithms, like post-quantum cryptography.
- **Interoperability:** Tink is available in many languages and on many platforms.

[14] https://errorprone.info/bugpattern/InsecureCryptoUsage, note that in our shared repository's toolchain, suppressing this check is subject to domain expert review.
[15] Erbsen, A. et al. 2020. "Simple high-level code for cryptographic arithmetic: With proofs, without compromises." ACM SIGOPS Operating Systems Review, 54(1), 23-30. https://doi.org/10.1145/3421473.3421477.
[16] Kuepper, Joel et al. 2023. "CryptOpt: Verified Compilation with Randomized Program Search for Cryptographic Primitives." Proc. ACM Program. Lang. 7, PLDI, Article 158 (June 2023), 25 pages. https://doi.org/10.1145/3591272
[17] https://bughunters.google.com/blog/6038863069184000/formally-verified-post-quantum-algorithms

Android-specific Vulnerabilities

Android leverages vulnerability data from internal and external sources (like Google's vulnerability reward programs) to identify classes of vulnerabilities. We then work closely with feature teams to harden the Android platform by building mitigations into major releases.

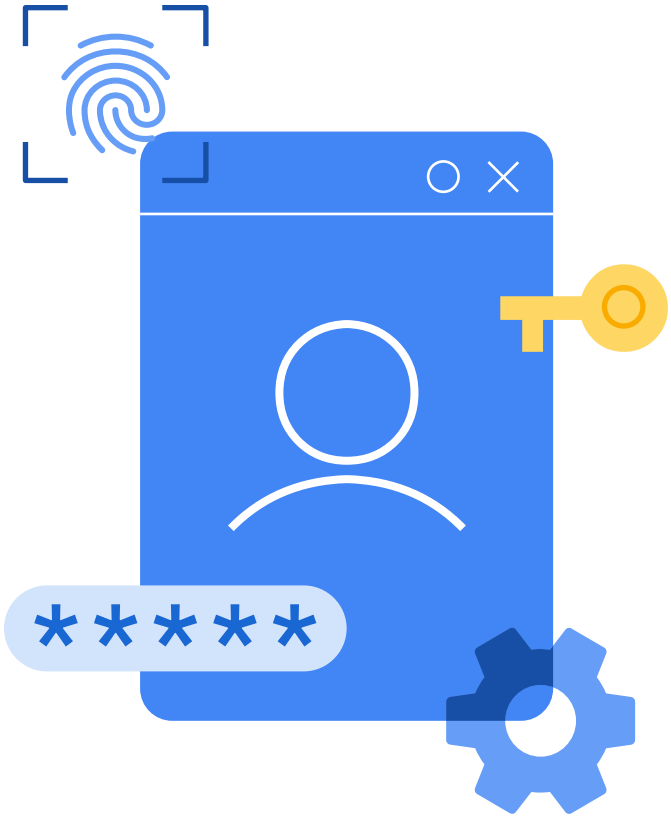
For example, Android has introduced **Safer Parcel Deserialization APIs**. Parcelable implementations can have vulnerabilities that could be exploited by malware to enable silent package installation and arbitrary code execution. We have implemented security hardening solutions to the Android Parcel mechanism to make parcel deserialization safer. These include: deprecating untyped parcel container APIs, checking that there are no bytes left to be read on the parcel, specifying allowed types before deserializing, and enforcing boundary checks for items in Bundle. Please refer to our presentation at Black Hat Europe for technical details.[18]

Other examples of mitigations include **Safer Dynamic Code Loading** which prevents an app from being exploited by loading and executing untrusted code, and the **Safer Zip Path Traversal API** which validates ZIP file entry paths via a new public API called ZipPathValidator. The API throws a ZipException if ZIP file entry names contain ".." or start with "/".

Finally, through the App Security Improvement Program, we provide developers with tips and recommendations for building more secure apps and identify potential security enhancements when apps are uploaded to Google Play. To date, the program has helped developers fix over 1,000,000 apps on Google Play. In 2022 alone, the App Security Improvements program helped developers fix ~500K security issues affecting ~300K apps with a combined install base of approximately 250B installs.

[18] Ke, Hao et al. 2022. "Android parcels: the bad, the good and the better – Introducing Android's Safer Parcel." Blackhat Europe. https://i.blackhat.com/EU-22/Wednesday-Briefings/EU-22-Ke-Android-Parcels-Introducing-Android-Safer-Parcel.pdf

Pledge Goal 4: Security Patches



While Google strives to minimize the number of issues in our products before they are released, we do know that errors – whether they're related to functionality, reliability, or security – are inevitable when building complex systems. We understand that addressing issues in our products is critical to their ongoing trustworthiness, and the trust pact with our users. As such, Google has developed strategies over time to ensure high rates of uptake for our deployed fixes, especially in cases where reducing the window of opportunity for exploitation is safety critical. We describe some examples below.

Web-Based and Cloud Services

Many of Google's products are delivered through web-based services. These include some of our most widely used and popular products such as Search, YouTube, and Gmail. One benefit of online web-based services is that end-users and customers do not need to take any action to update the software should a vulnerability occur. As such, when Google deploys a fix for a functionality-, reliability-, or security-related issue, it is addressed for our entire user base once fully deployed according to our safe release procedures.

When developers run on top of the managed Google Cloud platform, they gain the same benefits:

Google can take responsibility for patching and updating the infrastructure they run on, so they and their end users benefit from the scalability and immediacy of a central solution.

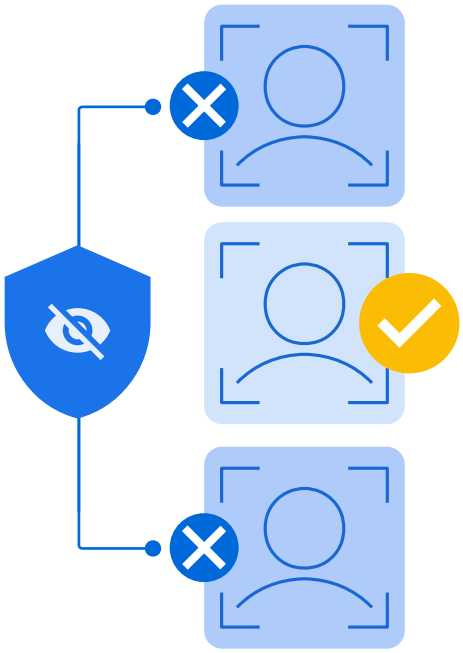
For example, Google Cloud was able to protect all hosted developers from the Spectre and Meltdown vulnerabilities without action on their part.

We developed shared fate in Google Cloud to start addressing the challenges that the shared responsibility model doesn't address. Shared fate focuses on how all parties can better interact to continuously improve security. Shared fate builds on the shared responsibility model because it views the relationship between cloud provider and customer as an ongoing partnership to improve security.

Enabling Enterprise Administrators

In September 2024, Google Workspace launched Security Advisor for small and medium-sized businesses which enables blocking outdated OS versions and disables access when an OS is missing appropriate security patch updates. Before blocking access, enterprise end users are given a warning upfront to allow them to act before losing access. This mechanism helps enterprises remain safe and secure with the latest operating system versions and security patches.

Google Workspace enterprise customers can also set a minimum Operating System(OS) version across MacOS, Windows, Linux, ChromeOS, iOS, and Android via Context Aware Access (CAA). In addition, domain administrators can enforce a Chrome Browser minimum version with CAA. Finally, customers can use monitor mode to see the potential impact before enforcing and blocking users.



ChromeOS

ChromeOS is built from the ground up with security as a top priority. Multiple layers of protection, including Verified Boot, sandboxing, blocked executables, and user space isolation, work together to create a defense against malware and other threats. This layered approach, combined with automatic and seamless updates, has allowed ChromeOS to remain free of viruses and ransomware for over a decade.

A key element of this security strategy is the automatic update system. Unlike traditional operating systems that may rely on infrequent, user-initiated updates, ChromeOS proactively patches vulnerabilities.

> ChromeOS proactively patches vulnerabilities.

This eliminates the need to wait for scheduled updates and ensures that devices are always protected against the latest threats. ChromeOS achieves this by storing two images of the OS: the active version currently in use and a new version that downloads silently in the background. When ready, the system seamlessly installs the update and, with a simple reboot, switches to the new, secure image. This process, combined with Verified Boot which ensures the integrity of the boot sequence, provides users with a secure and consistently updated computing experience. As of 2023, all Chromebook platforms receive regular automatic updates for 10 years after release.

Chrome Browser

Chrome updates happen in the background whether or not the browser is running. Restarting a running browser may be necessary to complete the update, but if no browser is running, Chrome will update without any user interaction. Chrome Browser was built with secure-by-design technology to provide security patches regularly and automatically to Chrome users, reducing the window of vulnerability for exploits.

Chrome Browser has a rapid release cycle that ensures security patches are deployed frequently, typically every week. This rapid response time helps address vulnerabilities promptly. Chrome moved from a two-weekly to a weekly security update cycle in 2023.

Android

Android is an open source operating system that powers both Google's Pixel product line and devices from hundreds of manufacturers worldwide. Device makers and carriers are responsible for deploying patches in their environment, and to facilitate consistency and speed of patch delivery, Android has worked closely with partners to develop safe mechanisms for updates.

Android has instituted a Security Patch Level (SPL) that drives the ecosystem to patch devices regularly. Android issues regular partner preview bulletins (to drive patch adoption with partners), regular public security bulletins to inform users and the ecosystem of what patches are available, and advisories for high-risk issues. System updates with security fixes are pushed to user devices and staged for installation upon the next device reboot. Android also has mandatory requirements for real-time patching of emergency-class vulnerabilities.

Android also builds and maintains tooling to detect when partners are missing security patches and inform OEMs, as well as tooling, like Security Hub, to help users understand patch update status and other important security characteristics.

A key strategy in scaling security updates across a broad range of hardware products at varying price points is to make it easier and more cost-effective for device manufacturers to update software. To this end, Google has spent years rearchitecting Android, with a focus on increased compatibility and centralization of updates by Google.

> Google has spent years rearchitecting Android, with a focus on increased compatibility and centralization of updates by Google.

Major milestones of this journey include improving hardware compatibility across updates through Project Treble, and better collaboration with system-on-chip (SoC) manufacturers to deliver pre-tested system images via GMS Express (2017). In 2019, major Android components became Google-updateable through Project Mainline. Next, in 2020, Generic Kernel Image (GKI) unified the core kernel and moved hardware-specific code into loadable vendor modules, while Linux Protected KVM (pKVM) Hypervisor created a harmonized Android trusted execution environment (TEE). GKI was improved in 2021 with GKI 2.0, which requires signed kernel images that are patched regularly by Google with Long-Term Support (LTS) and critical bug fixes. In 2021, we also enabled standardized Android binaries for Secure Element (SE) applets, including over-the-air updates (Android ReadySE). Finally, in 2023 we introduced Android Virtualization Framework (AVF) to provide standardized interfaces for Android TEEs and virtual machines.

Pixel

Google's Pixel phones and watches powered by Android are designed to prioritize security and provide users with the latest features through automatic updates. In enterprise environments, IT administrators have the option to disable automatic updates, allowing them to test new software versions before deploying them across their fleet of Pixel phones. This flexibility ensures compatibility and minimizes potential disruptions.

Additionally, Google maintains transparency by publishing Pixel security phone, tablet and watch bulletins and the security support periods for Pixel phones and watches, empowering users to make informed decisions about their devices' lifespan and ongoing protection. Software updates for Pixel customers are released regularly and include security patches. This rollout proceeds gradually and the updates become available to 100% of customers within two weeks. The latest Pixel 9 family of devices are guaranteed Android and security updates until at least August 2031.

Fitbit

While automatic updates are the easiest way to keep devices updated, things get far more complicated in highly constrained devices. For Fitbit trackers, these devices are small in size, but need to last a long time before getting charged. Thus, automatically pushing an update to the device could result in a failed update if the battery is not fully charged, or cause the device's battery to die at an inconvenient time for the user. Therefore, for these devices, the user is notified that an update is available, but the user controls when it is applied.

Nest

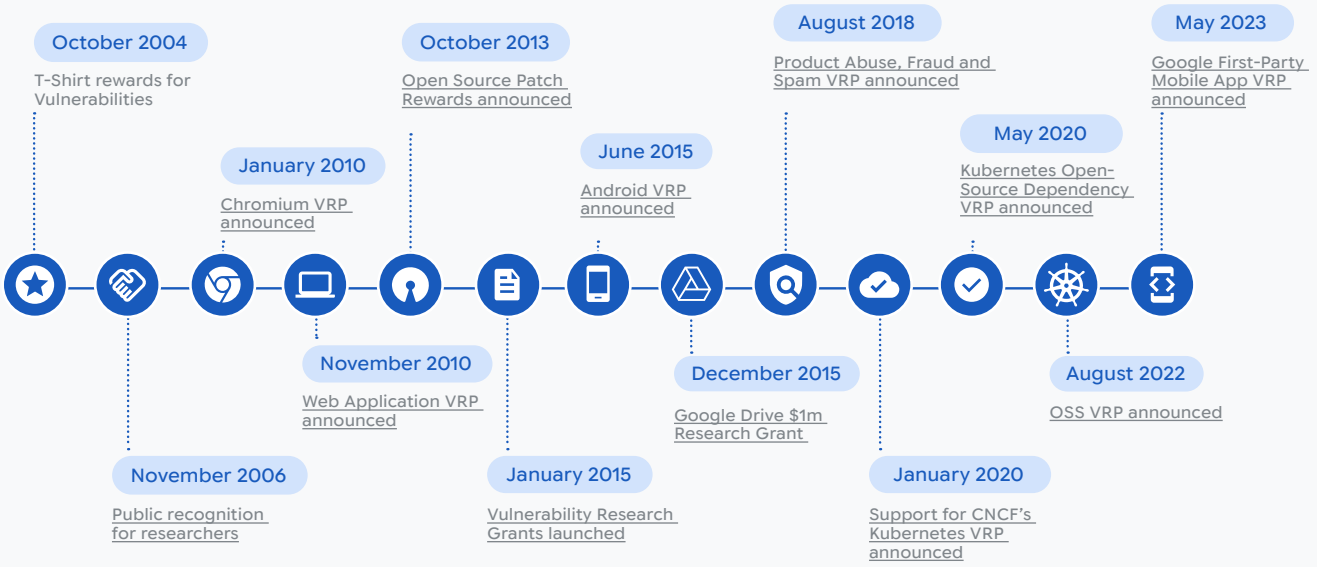
In 2019, Nest established a set of security and privacy commitments, further strengthened in 2021 by guaranteeing that all Nest devices will receive automatic security updates for a minimum of 5 years. Google deploys updates in a rolling fashion, which typically complete within a three-week period, and publishes the security support periods for each Nest device, enabling users to make informed decisions. Additionally, Google collaborates with security researchers who play a vital role in identifying vulnerabilities in our devices. Upon discovery and confirmation of a risk to users, these vulnerabilities are promptly patched, and the corresponding updates are automatically deployed to safeguard our users' homes and data.

Pledge Goal 5: Vulnerability Disclosure Policy

Google's belief is that building secure and reliable systems requires monitoring for issues that require study and remediation. This includes security vulnerabilities that might exist in our products. While Google has proactive measures to detect issues internally, we are also committed to receiving reports from external sources and working with the broader Internet community to improve our systems.

Google's Vulnerability Disclosure Policy reflects our beliefs, providing a clear and accessible channel for security researchers to report potential issues in our products.

A Closer Look at Google's Vulnerability Reward Programs



Google has been collaborating with external security researchers since 2004. Our reporting process encourages direct engagement, fostering a community-based approach to address security concerns.[19] Security researchers are encouraged to report issues to us that are discovered responsibly (i.e., Google discourages actions that could disrupt or harm users). From there, a panel of Google security experts reviews each vulnerability report, assessing its potential impact and the sensitivity of the affected service.

Based on this evaluation, rewards are assigned, ranging from $100 to $1 million based on the complexity and severity of the issue.

Over time, Google has evolved the program by expanding what kinds of issues qualify for a reward as well as increasing reward payments.

[19] For more information on our Bug Hunters program, see also Hacking Google Episode 004.

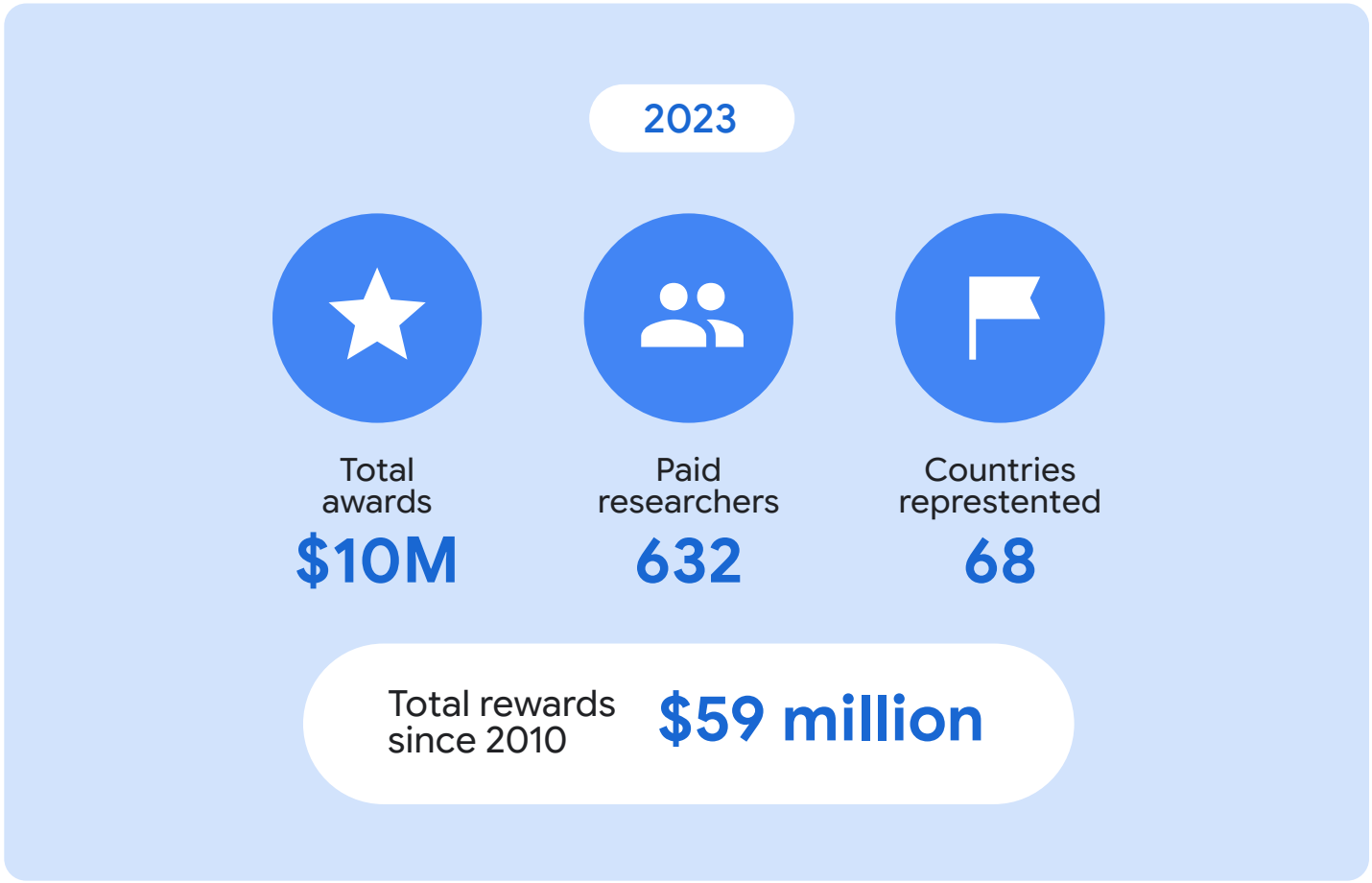
The vulnerability reports submitted by external researchers inform and validate our proactive efforts to address and eliminate classes of vulnerabilities, and thus provide an important feedback loop in our end-to-end security process.

In addition, beyond being a channel for responsible disclosure, Google also uses the VRP to incentivize security research into emerging risk areas that we believe are of interest to adversaries. Our bugSWAT events are a good example: we invite external security researchers to search for vulnerabilities in our products side-by-side with our security team and support learning by accompanying these activities with presentations from our engineering teams. Google also regularly issues vulnerability research grants to accomplished bug hunters and domain experts – these grants provide an up-front financial reward for researching areas of specific interest to us; any discovered vulnerabilities are eligible for additional reward by our VRP.

In 2023, working with our dedicated bug hunter community, Google awarded close to $10 million to 600+ researchers based in 68 countries. The highest single reward exceeded $113,000. Over the lifetime of the program, more than 18,500 individual rewards have been given totaling nearly $59 million.

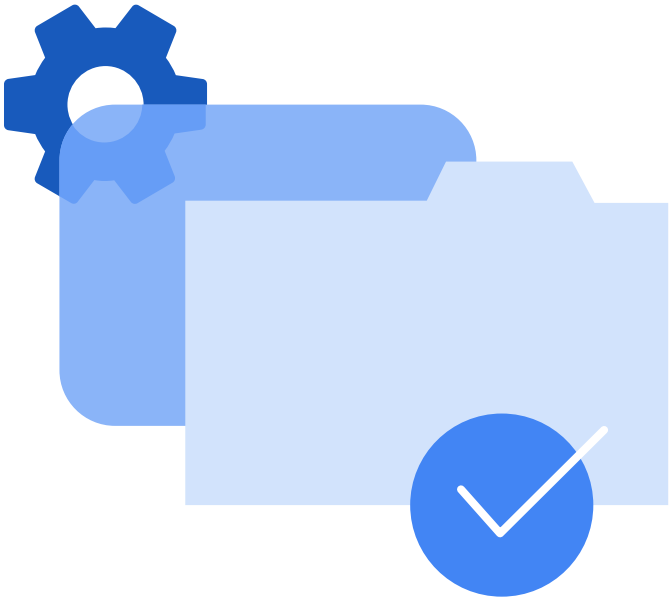
> The highest single reward exceeded $113,000.

While Google has a broad VRP to cover all of Google and Alphabet, we have also established product-specific VRPs, allowing us to develop custom guidelines and tailor rewards for specific areas of interest. An overview of VRPs at Google and links to detailed program rules (including scope and rewards) can be found on the Google Bug Hunters site. Also see our blog post reviewing VRPs in 2023.



Pledge Goal 6: CVEs

In 1999, the project to enumerate and share Common Vulnerabilities and Exposures (CVE) was conceived as a way to normalize disparate vendor vulnerability databases, with the purpose of simplifying the implementation of security assessment tools.[20] At the time, almost all software was still released "off the shelf" and available only by download or hard-copy (e.g., CD-ROM). Users wishing to protect themselves from security issues had to be notified that they had required actions to take. CVEs were intended to fill the knowledge gap for end users.



Over the subsequent decades, industry experts and academics have debated CVE issuance, especially as attacks and software delivery mechanisms have evolved. For example, alternatives to the CVE database were instantiated to address perceived weaknesses in the original proposal (e.g., National Vulnerability Database, the defunct Open-Source Vulnerability Database (not to be confused with OSV.dev)).

Severity scoring was conceptually introduced, most notably via the Common Vulnerability Scoring System (although there are many others).[21] Taxonomies were proposed, such as the Common Weakness Enumeration (CWE) to enable data analytics and the Common Platform Enumeration (CPE) to standardize product names. Federated CVE Numbering Authorities (CNAs) now give entities operational flexibility in issuing their own CVEs. This flexibility has introduced unresolved challenges around CVE record consistency between CNAs.

While most major software producers globally, including Google, engage in use of some or all of these schemes, there are differing views amongst experts on their return on investment, scalability, efficiency, and usefulness. For example, there is no consensus on, nor consistent use in practice of, CVEs for web-based service vulnerabilities that do not require user or customer interaction to upgrade. Known issues exist with scoring vulnerability severity, namely needing to know how a piece of technology is used in a target environment to rate the issue accordingly.[22] And some of the programs for maintaining databases, scoring mechanisms, and taxonomies are maintained on a best-effort basis.

At Google we have led efforts to address these problems for open source software via the OSV Schema and OSV.dev vulnerability database. This involved working with many open source ecosystems (e.g. Python Software Foundation, Rust Secure Code Working Group) and large entities such as GitHub, Canonical, and Red Hat on adopting a common standard and distributed vulnerability database for open source software. The OSV Schema is now broadly supported across most major programming language ecosystems as well as Linux distributions, which makes OSV.dev a comprehensive source of vulnerabilities for open source software.

As part of this effort, Google also ensured that OSV maintained interoperability with CVEs. We've collaborated with CVE working groups on the CVE 5.0 standard, and aim to continue to work with them to ensure continued interoperability to help improve CVE's processes and standards.

[20] https://cve.mitre.org/docs/docs-2000/cerias.html; https://cve.mitre.org/docs/docs-2001/Development_of_CVE.html
[21] The CVSS has undergone several iterations, now on version 4: https://nvd.nist.gov/vuln-metrics/cvss. NIST is currently only "prioritizing analysis of the most significant vulnerabilities."
[22] A recent study of different vulnerability scoring systems highlights the complexities. Milousi, Konstantina et al. 2024. "Evaluating Cybersecurity Risk: A Comprehensive Comparison of Vulnerability Scoring Methodologies." In Proceedings of the 19th International Conference on Availability, Reliability and Security (ARES '24). Association for Computing Machinery, New York, NY, USA, Article 52, 1–11. https://doi.org/10.1145/3664476.3670915

Issuance of CVEs

Google prioritizes issuing CVEs for our products when users need to take action. This also helps meet the original goal of CVEs: enable security assessment software in a customer's environment to sufficiently identify fixes that have not been applied. Google issues CVEs for its products in the following circumstances:

- Consumer and Enterprise products that require user or customer actions to update – even if just to restart – such as Chrome, Android, ChromeOS, Google Cloud and Google Devices. We have issued over 8,000 CVEs in these products over 13 years.
- Open-source software that we publish through our Github repositories. We have issued over 600 CVEs in these projects over 4 years.
- Vulnerabilities reported to us through our VRPs. CVEs add transparency to our VRP programs and recognize researchers for their work.

In 2011, Chrome & Android became our first CVE Numbering Authorities (CNA), one of a small group of a few hundred entities worldwide that can issue their own CVEs.

In 2022, Google expanded its partnership with MITRE to become one of the 4 Roots.

We continue to listen to our customers on what will be most helpful to them and evaluate how we are transparent about how we resolve issues in our products. This includes engagement with government and industry. Google applauds CISA for furthering the discussion on vulnerability scoring (CVSS) and taxonomies (CWE, CPE) and we value the partnership in resolving outstanding gaps in their feasibility. As our journey progresses, we will continue to provide updates.

Product Bulletins

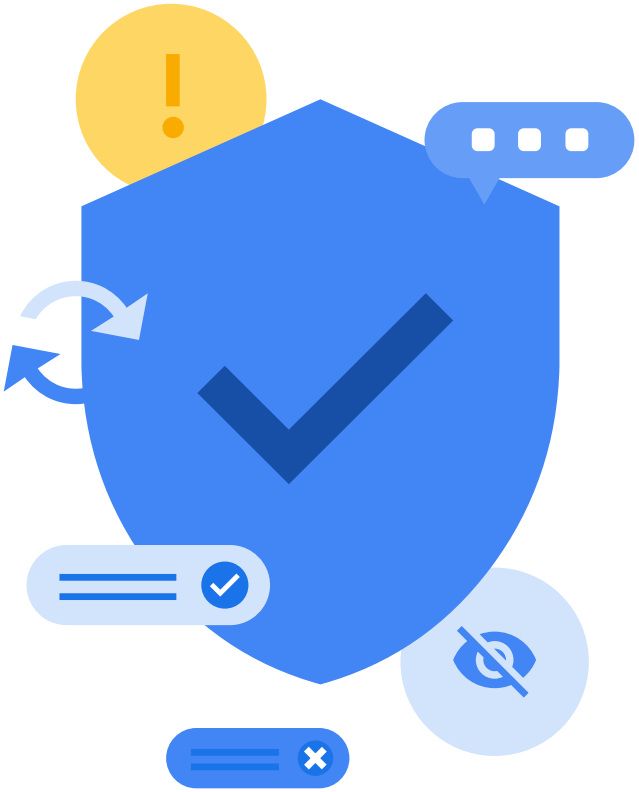
In addition to issuing CVEs, Google provides security bulletins for many of its products that are delivered through software update mechanisms. We provide a brief list of examples below.

- Android: Google publishes a monthly Android Security Bulletin that details vulnerabilities discovered and patched in the Android operating system and its components (including partner-specific components). Android also publishes a Transparency Report on Android ecosystem security, providing data on the prevalence of potentially harmful applications and the effectiveness of security measures.
- Chrome Browser: Chrome's automatic update functionality covers both milestone releases and security updates. We additionally provide information for other browsers that share Chromium's core browser engine (via the security-notify list) as an extra step in openness. We've recently enhanced some of our CVE records to include CWE information, and have ongoing projects to include CWE and CPE information in them all.
- ChromeOS: ChromeOS was designed from the ground up to update automatically, in the background, with no user interaction. Google has been tracking CPE information for third-party packages included in ChromeOS since 2019. We use this to automatically identify publicly disclosed vulnerabilities in third-party packages shipped with ChromeOS. Moreover, we're currently revamping our release notes process to more comprehensively identify fixed security bugs, including listing CVE identifiers.
- Cloud: Google Cloud's Security Bulletins provide detailed information about security vulnerabilities affecting products and services in Cloud. These bulletins typically include a description of the vulnerability, its potential impact, affected products and versions, and recommended mitigation steps or patch updates. They serve as an essential resource for Google Cloud users to stay informed about security risks and take appropriate action to protect their systems and data.

Pledge Goal 7: Evidence of Intrusions

Google believes it's important for customers to be able to have insight into whether cybersecurity intrusions are affecting the use of our products, and support CISA's seventh pledge goal of providing customers and users evidence of intrusions. Many of Google's products take into account how users and customers receive information about issues that may be impacting them.

Our work here is grounded in providing the right kind of insights while not overloading customers with too much irrelevant or inactionable information. We outline some examples where we've achieved this balance below, and are continuing to invest in additional insights and resources.



Google Accounts

Google provides visibility and alerting mechanisms to protect Google consumer accounts. Security Checkup provides personalized security recommendations for Google Accounts, including account recovery options, 2-step verification, removing risky access to data, and screen locks. We send Security Alerts if we detect unusual account activity on an account. And we allow users to see which devices have account access, and where they are signed in from.

Google Safe Browsing

Google Safe Browsing warns users before they visit dangerous sites and protects users from web-based threats like malware, unwanted software, social engineering, phishing, and deceptive sites. By leveraging verdicts from our detection systems, Google protects users by showing them warnings before they visit dangerous sites, or download malicious files. We've made Safe Browsing services free and publicly available for developers and other companies to use in their applications and browsers.

For users who require or want a more advanced level of security, Google offers Enhanced Safe Browsing (ESB) which provides additional AI-powered protections from the newest online threats. In Chrome, ESB users benefit from on-device and server-side models that look for signals commonly associated with malicious behavior. Furthermore, additional file protections like deep scans for suspicious files protect users from the latest malware. ESB users also get additional protection in Gmail from spam related to malicious files.

Android

Android has proactive measures and real-time monitoring and controls that alert a user to suspicious activity that could be related to potential intrusions on their device. In consumer use cases, for example, Google Play Protect (GPP) is a built-in anti-malware solution that runs on 3+ billion Google Mobile Services-enabled devices and alerts users to potential threats through a combination of on-device app scanning and cloud-based backend infrastructure. Android also has out-of-the-box detection for potential scam and phishing attacks received via Google Messages. Malicious Android apps often exhibit unusual behaviors, so we provide real-time alerting to users for apps that request Runtime Permissions, access to Android's clipboard, microphone and camera, and Background Location Access. If an app has not been used for a few months, the system protects user data by initiating a Permission Auto-Reset.

Additionally, enterprises managing a mobile phone fleet can leverage Android Enterprise capabilities to look for evidence of intrusions. Specific items include Security Audit Logs that record the device configuration and changes to it, application installations, device reboots, and other audit events. Network Event Logs and various other risk signals enable Unified Endpoint Vendors, Mobile Threat Defense Vendors, SIEM, and other security vendors to monitor network activity on the device and understand whether a device or application is running in a compromised environment.

Android also offers app developers various signals that they can leverage. For example, when an app is used on an Android device with the Google Play Store, and powered by Google Play services, the Play Integrity API provides a response that helps developers determine whether the user is interacting with a genuine, unmodified version of the app binary that Google Play recognizes. The API also provides the ability to determine whether the current user account installed or paid for their app or game on Google Play and whether the app is running on a genuine Android device powered by Google Play services (or a genuine instance of Google Play Games for PC). Developers can also choose to receive information about whether apps are running that could be used to capture the screen, display overlays, or control the device, and whether Google Play Protect is turned on and has found risky or dangerous apps installed on the device.

Google Cloud

Google Cloud operates a shared fate model which emphasizes collaboration with customers to achieve a common security and risk management goal. Google builds a foundation with built-in, always-on, and immutable controls aligned to Google's opinionated best practices, and provides actionable intelligence on security posture and risk, as well as continuous threat protection.

As a baseline for security, admin activity audit logs that capture actions that modify the configuration or metadata of resources are available for Cloud products. For example, these logs record when users create VM instances or change Identity and Access Management permissions. These logs are stored for 400 days, and users cannot configure, exclude, modify, or disable them. System event audit logs are the equivalent for Google-generated actions, and no configuration by users is required. In addition to these default audit logs, users have the option to select additional products for access to even more verbose logs, either from the Google Cloud service or from their own applications.

Cloud Logging allows for the centralization and retention of logs starting at 30 days for general logs without additional charge. Customers can additionally opt-in to keep logs for longer, up to 10 years in Cloud Logging or indefinitely in cold storage. Customers can choose the log management tool of their choice. We allow customers to use the log management tool of their choice and offer routing of logs to multiple destinations (GCS, Pub/Sub) to support other tools (Splunk, Elastic, Datadog, etc.) without Cloud Logging charges.

Google Workspace

Google offers the ability to review recent Gmail activity, including the dates, times, and IP addresses of sessions used to access the Gmail service. Users can also use Google Takeout to download "Access Log Activity," which includes multiple weeks of log information for Gmail and other Google services. In addition, Gmail users can move from this dialog to the security checkup to secure their account and device.
Gmail issues alerts to consumer users and Workspace domain administrators if a possible government-backed attack is detected.

Google Workspace domain administrators can use the audit and investigation tool and Reports API to review user and administrator activity in their organization.

> Gmail issues alerts to consumer users and Workspace domain administrators if a possible government-backed attack is detected.

Google Workspace creates log events for relevant end-user actions across Google Workspace services like Gmail, Drive, Docs, and Chat, and also logs all admin actions. The default retention of these audit logs is 6 months with various audit log export capabilities.

Domain administrators can use this information to track actions performed by users and admins, and for security purposes.

For email, Google Workspace supports additional email security sandboxing, malware protection, and phishing protection, and allows admins to take actions like finding and erasing malicious emails, marking emails as spam or phishing, or sending notifications to users' inboxes.

Finally, Google Workspace domain administrators can use alert center to view notifications about potential issues within their domain, and take action (like end-user education or updates to existing policies or settings) to resolve the issues and protect their organization from security threats.

Conclusion: Building and Sustaining a Security Culture

As we outline in our book, Building Secure and Reliable Systems, security is largely an emergent property[23] of the developer ecosystem in which software is designed, implemented, and deployed. As such, it is a shared responsibility across the entire organization, not just the domain of security specialists. Everyone, from developers and Site Reliability Engineers to managers and executives, plays a role in maintaining the security and reliability of systems. Google believes that integrating security considerations throughout the entire software development lifecycle is crucial to building resilient systems.

To foster a security-first culture, Google promotes a culture of review, where changes to code and configurations undergo peer scrutiny before deployment. We leverage automation to streamline processes and minimize errors, and encourage the use of Secure by Design APIs and frameworks to guide developers toward creating inherently safer code.

Google's approach to security culture also emphasizes the importance of transparency and open communication, regularly engaging with the broader security community through initiatives like our Vulnerability Reward Programs, and encourages knowledge sharing and collaboration both internally and externally. By fostering a culture where security is everyone's responsibility and integrating security practices into the development process, Google aims to build systems that are both highly secure and reliable, ultimately providing a safer and more trustworthy experience for its users and enterprise customers. All of this is reinforced by a postmortem philosophy.

Google recognizes that creating a truly secure digital ecosystem requires a collaborative approach – one that identifies common threats and develops shared solutions that protect users across the world. We invite industry partners, policymakers, and security experts to join us in this critical endeavor. By working together, we can establish common standards, share best practices, and develop innovative solutions to combat evolving threats. We believe that through collective action – collaborating with everyone from security experts to competitors, governmental bodies, policy makers, and everyday citizens – we can build a more secure and resilient digital future for everyone.

Google supports CISA in their efforts on Secure by Design and believes that the practices outlined in this paper can help other security experts build truly defensible systems. While we have been successful in evolving and improving Google's security posture, we do not intend to rest on our laurels. We will continue to innovate and push the boundaries of what's possible in the security space.

[23] Kern, Christoph. "Developer Ecosystems for Software Safety." Communications of the ACM 67.6 (June 2024), 52-60. https://doi.org/10.1145/3651621.

Contributors

Thank you to the following people, and so many others, for contributions to this paper and the work it represents.

Adam Bacchus
Adam Samet
Adam Wu
Adrian Taylor
Alex Rebert
Amy Ressler
Andres Erbsen
Andrew Eames
Andrew Pollock
Andrew Whalley
Artur Janc
Bill Creasey
Bobby Jen
Bobby Norberg
Brad Ree
Camillus Cai
Casey Sakima
Chandler Carruth
Charley Snyder
Christiaan Brand
Christoph Kern
Dave Kell
Dave Kleidermacher
David Klein
David Monsees
Deeksha Kaul
Diane Tang
Dirk Göhmann
Ed Fernandez
Elias Levy
Eugene Liderman
Hao Ke
Harini Parthasarathy
Harold Chun
Heather Adkins
Iain Mulholland
Javier Leon
Jeanette Manfra
Jeff Ma
Jeffrey Vander Stoep
Jen Engel
Jeroen Kemperman
John Gronberg
John Solomon
Jonathan Hirsch
Jonathan Li
Jonathan Rubin
Jorge Lucangeli Obes
Juan Vasquez
Kate Charlet
Keira Li
Kimberly Samra
Krzysztof Kotowicz
Lukas Weichselbaum
Mary Koes
Matthew Flegal
Matthew Riley
Max Grifka
Melanie Lombardi
Michael Groover
Mike Burr
Nicolas Lidzborski EOT
Nithan Sannappa
Oliver Chang
Pankaj Rohatgi
Parisa Tabriz
Peter Valchev
Phillip Carter
Piumi Arachchige
Rocío Vives
Roger Piqueras Jover
Ross Richendrfer
Sachin Parsewar
Salvador Mandujano
Sarah Morales
Shenaz Zack
Shuvo Chatterjee
Sławek Goryczka
Sophie Schmieg
Sri Tulasiram
Srilekha Krishnamurthy
Sriram Karra
Stefan Kölbl
Stephanie Kiel
Tatyana Bolton
Thomas Holenstein
Thyla van der Merwe
Tim Dierks
Tony Ureche
Wendy Dembowski
Will Beers
Yang Yang