SECURITY

BeyondCorp
Building a Healthy Fleet

HUNTER KING, MICHAEL JANOSKO, BETSY BEYER, AND MAX SALTONSTALL

Hunter King is an Engineer on the Security Operations team at Google. Currently, he focuses on endpoint integrity and identity. Hunter has also been a Lead Engineer in the BeyondCorp effort for the last seven years. Prior to Google, he was a Security Researcher at SecureWorks. He enjoys hiking, tinkering, and making lights blink. Hunter holds a bachelor's degree in computer science from Colgate University. hunterking@google.com

Michael Janosko is a Security Engineer Manager in Google's Enterprise Infrastructure Protection group, where he helps secure the way Google works. On weekends, he enjoys a good cup of coffee while building forts with his son. janosko@google.com

Betsy Beyer is a Technical Writer for Google Site Reliability Engineering in NYC, and the editor of *Site Reliability Engineering: How Google Runs Production Systems* and the forthcoming *Site Reliability Workbook*. She has previously written documentation for Google Datacenter and Hardware Operations teams. bbeyer@google.com

Max Saltonstall is a Technical Director in the Google Cloud Office of the CTO in New York. Since joining Google in 2011, he has worked on video products, internal change management, IT externalization, and coding puzzles. He has a degree in computer science and psychology from Yale. maxsaltonstall@google.com

Any security capability is inherently only as secure as the other systems it trusts. The BeyondCorp project helped Google clearly define and make access decisions around the platforms we trust, shifting our security strategy from protecting services to protecting trusted platforms. Previous BeyondCorp articles discussed the tooling Google uses to confidently ascertain the provenance of a device, but we have not yet covered the mechanics behind how we trust these devices.

Our focus on platform security is supported by a wealth of evidence [1] in the industry that end users are the number one target of a wide range of attacks that also vary in sophistication. Attackers can devise quite advanced social engineering attacks as mechanisms to deliver malicious code onto devices, where they can then exploit the large attack surface of modern operating systems. Advanced attackers aim to reuse trust inherent in the device, the credentials on the device, or the trust granted to the user to further exploit systems.

To successfully prevent compromise in environments with a constant mix of trusted (enterprise web apps, corporate credentials) and untrusted content (external software repos, social media, personal email, etc.), the platforms themselves must have a layered and consistent set of controls. As a result, the platforms that make up the fleet are the new perimeter.

Building upon Previous Work

The work we describe in this article builds upon the work described in the white paper "Fleet Management at Scale" [2] and the previous five BeyondCorp articles [3]. Building on this foundation, our team aimed to further strengthen the BeyondCorp model by:

1. Defining what a healthy fleet looks like from a common control perspective
2. Ensuring that these controls are consistently and comprehensively applied, measured, and enforced
3. Using these measurements to drive continuous improvement in our control set

Defining the Threats against Your Environment

As with any defensive security effort, it's important to first define the threats against the environment you're trying to protect. When creating this list of threats, it's helpful to think of classes of attacks instead of all the variants of a single attack. Attackers are constantly discovering new variants of attacks, which makes defining the entire tactical threat environment impossible. However, if you successfully mitigate a class of attacks, then variants within that class should be less concerning [4].

At a very high level, some classes of threats to consider against your platforms include:

1. **Unknown devices**: sensitive systems accessed by unknown or unmanaged devices
2. **Platform compromise**: exploitation of a misconfigured operating system or software on the platform
3. **Security control bypass**: system compromise through unused or misconfigured security policy

4. **Privilege escalation**: code execution resulting in privileged system controls takeover and persistence on the system
5. **Software compromise**: installation and persistence of malware
6. **Attack persistence**: prolonged persistence of attackers due to lack of inspection
7. **Authentication bypass**: compromise of the platform through password theft or authentication bypass
8. **Data compromise**: unauthorized access to sensitive data on disk, memory, or in transit
9. **Attack concealment**: prolonged persistence of attackers due to lack of logging and monitoring
10. **Attack repudiation**: hampered investigations due to attackers' ability to cover their tracks

Addressing These Threats through Improved Fleet Health

With these threats defined, you can better identify the classes of controls you need to mitigate these threats. Then you can measure the state of these controls (their effectiveness, and whether they are on or off) through device inspection at service access time. Table 1 maps each of the categories of threats outlined above to the qualities ("Control") one would expect to see in an ideal trusted platform.

#	Threats	Control
1	Unknown devices	Fleet inventory and asset management
2	Platform compromise	OS & base software configuration management
3	Security control bypass	Security policy management & enforcement
4	Privilege escalation	Resilience against system takeover & persistence
5	Software compromise	Software control and anti-malware
6	Attack persistence	Remotely verifiable platform state
7	Authentication bypass	Robust authentication of platform and user
8	Data compromise	Data protection
9	Attack concealment	Logging and log collection for detection capability
10	Attack repudiation	Response capability on platform/ Detection & response

Table 1: Threat classes and potential mitigations

Characteristics of a Healthy Device

A healthy fleet is composed of healthy devices supported by tooling, processes, and teams to maintain fleet health. We consider a device to be healthy if:

- It can withstand most attacks.
- It provides sufficient telemetry to contain a compromise when one occurs.

Let's take a deeper look into the reasons why each of the qualities of an ideal trusted platform we enumerated above are important.

Fleet Inventory and Asset Management

Hardware is the foundation on which the OS and applications run. Limiting hardware configuration variations allows you to more effectively reason about the capabilities and limitations of the devices in your fleet. An inventory system places an upper bound on the number of devices able to connect to sensitive systems through device access provisioning.

OS and Software Configuration Management

Software management is a key component to maintaining a healthy fleet. A centralized management infrastructure should drive a consistent platform configuration to ensure that instances of the trusted platform:

- Are secure by default, with minimal drift over time
- Continue to benefit from security improvements over time

The ability to patch the running OS, the sensitive software stack, and protective agents is paramount to a healthy security posture. It's equally important to manage configurations (e.g., software auto-update policy) in a central location.

Security Policy Enforcement

Trusted platforms should enforce security policies consistently, and report and log any deviations from expected policy. Security policy is often intertwined with the general OS management and configuration policies mentioned above. However, security policy is unique because it's a *mandatory access control policy that users cannot subvert*. For example, consider minimally inclusive login policies: this strategy lessens the threat of lateral movement, and removing root privileges by default helps mitigate the damage a rogue process can inflict.

Resilience against System Takeover and Persistence

The goal here is to layer defenses so that malware execution doesn't necessarily compromise the security of the system. Ensure that hosts can report abnormal behavior before advanced malware can silence a host's logging subsystem.

Software Integrity and Control
You should be able to restrict unauthorized code execution on the platform. Common strategies include either only allowing known good software and explicitly blocking suspected bad software. We generally prefer an allowed list strategy: it's possible to define the applications you need to accomplish your work, but the potentially bad actors or software you need to block are infinite.

Remotely Verifiable Platform State
The platform should have a cryptographically verifiable integrity mechanism that provides guarantees on the underlying platform—from the firmware up to and including the running OS. Some examples include first-command-execution control [5], secure boot, and remote attestation.

Robust Authentication of Platform and User
Wherever possible, credentials should be hardware-backed or hardware-isolated on a system. Windows Defender Credential Guard [6] is one example of this capability.

Data Protection
We assume that any user's system has some sensitive data; therefore, sensitive data should be encrypted both at rest and in transit. To handle lost or stolen devices, devices should support remote wipes that destroy any data stored on the system and any long-term credentials.

Logging and Log Collection for Detecting Threats
To provide defense in depth, the platform threat model should assume that attackers will bypass preventative controls and that machines will be compromised. To mitigate this risk, your platforms should be able to log such incidents. Logging should include user- and device-attributable audit records for all sensitive data accesses or modifications, including changes to the platform's security controls, state, and behavior. This information should be streamed to a centralized logging facility. The ideal logging strategy prevents unauthorized processes from tampering with the logs.

Response Capability on Platform / Detection and Response
If a threat is detected, platform capabilities should facilitate remote incident response by authorized intrusion analysts. Tools like GRR can provide remote accessibility for performing this analysis [7]. We prefer to keep device-in-hand forensics to a minimum, as this strategy can't scale to respond to a widespread breach. Ideally, authorized analysts should be able to create a forensically sound timeline of an incident and augment the investigation with one-off pulls from the affected systems. By re-creating an event, the Detection and Response team can obtain a thorough picture of what happened and respond accordingly.

Maintaining a Healthy Fleet

A group of client devices with the controls detailed above make for a generally healthy and secure fleet. To reach that state, we first needed to figure out how to bootstrap our platform trust.

Building Up Trust

Sensitive services should only be accessed by trusted devices. We divide system trust into tiers. Devices can earn different levels of trust based on their characteristics and behavior [8].

Unfortunately, this approach results in a chicken and egg problem: transitioning a device into a trustworthy state requires access to a client software repository, yet a client software repository *is* a sensitive system. To resolve this issue, we introduce an *Identified* state in the journey from untrusted to trusted. An identified device is one our inventory system believes to be in good standing but is not trusted for some reason. These devices can access a subset of our client software repository in order to install remediation software. This software enables a machine to report device state, download and apply required patches, and take all necessary steps to fulfill the requirements of a trusted platform.

As you work towards building a healthy fleet, you achieve a better understanding of your environment. As a result, you're in a stronger position to grant access confidently. The next challenge is maintaining that state as technology and your business continue to change. The following section discusses how to keep the fleet in a good state of health as you evolve, and how to correct quickly when health degrades.

Combating Device Entropy

Once in the hands of users, devices are prone to becoming less secure as security guarantees atrophy over time. We've found a few strategies useful in our fight against entropy.

The first and most powerful strategy is to integrate access decisions with an inventory system. All machines should be known and trusted before they're granted access to internal resources. At Google, we add every machine in our fleet to our corporate inventory during the receiving and imaging process. We promptly remove access from any devices reported as missing, stolen, or lost. To encourage timely reporting of lost or stolen devices, we require users to self-report before they can receive a replacement device.

It's also important to have strong telemetry around the state of any machine that accesses your environment. Facebook's OS Query [9] is an excellent open source telemetry tool for Linux, OS X, and Windows: it allows you to measure device properties such as a machine's OS version, patch level of critical software, and encryption status.

Finally, patch and configuration management tools [10] enable you to change the security state of a machine—transitioning an untrusted machine into a trustworthy one. BeyondCorp uses access restriction to help drive user actions such as rebooting or accepting updates.

Detecting Unhealthy Hosts

Throughout the lifecycle of a host, certain actions or inactions might cause a device to transition to an unhealthy state. Our trust inference system [11] detects state changes by performing continuous trust evaluations. When a device fails to meet our trust criteria, we downgrade its trust level to *Identified*. We notify the machine's owner and provide instructions for remediating their device.

Our Detection and Response Team acts as an additional datasource for trust decisions. This team can remove trust from any machine that's acting maliciously.

Providing Flexible Policies

At a quick glance, defining fleet healthiness is a straightforward task. However, like most IT environments, the devil is in the details (and the exceptions). When dealing with a plethora of different OSes and a wide variety of use cases, you encounter many of these details.

As we roll out controls to the fleet, we always attempt to introduce thresholds of policy compliance rather than institute absolute requirements. This strategy allows users greater flexibility to operate within a good state and avoids draconian rule sets that break many of our users (causing them to seek out workarounds or overrides). For example, if a user needs to apply a non-critical patch, we give them a grace period before downgrading their access.

We also believe it's important to design preventative controls to provide signal to your incident detection and response capabilities. To that end, we work to integrate these controls into our security information and event management pipeline so that they can report and log relevant policy-related data. Capturing data about when we allow access and when we block access according to policy can aid in future forensics and incident detection.

Rolling Out and Scaling These Principles

A typical development process and rollout by the Security Team and its partners starts with the design and prototype phases, followed by a period to gather feedback across the fleet and from our users. Over time, we've arrived at a strategy of first rolling out controls in monitor mode and crafting our dogfood [12] populations to facilitate debugging. For instance, we might push a new USB auditing agent to a subset of a hardware engineering organization, as this population often interacts with custom USB components. As a result, we'll uncover edge cases that will likely crop up in a less concentrated form across a broader sample size. Alternately, we might slice the dogfood geographically and prepare local support staff in advance of the change.

When rolling out new controls, clear communication helps build understanding of the new policies and why they exist. Mapping each control to the threats it addresses helps everyone understand why the Security team has chosen a particular action. High transparency and explicit explanations of our criteria have increased understanding among our users and helped us build consensus among stakeholders. When they saw we had no concealed objectives or motives, we could bring them fully on board with our vision of the future and our timeline to get there. Often, teams tasked with making security-driven changes can benefit from seeing the big picture goal, which increases the credibility of the request and therefore also increases buy-in from partner teams. This buy-in often leads to a virtuous cycle of feedback about how you can make the fleet even more secure.

Platform Measurement and Control Parity

Once you define your baseline expected qualities, you'll find you can't apply controls universally—capabilities vary (sometimes widely) among platforms, both in terms of the device itself and in the management/policy layer. For example, Chome OS's Secure Access provides robust software control, but Linux has no out-of-the-box capabilities that prevent malware. To ensure consistency in security across our fleet, we needed to normalize security evaluations. While it's probably not appropriate to expect 100% parity across different platforms (as capabilities and threat models differ), we aim to be consistent when classifying a control as sufficient versus a security risk that requires action.

To accomplish normalized evaluations, we analyzed the current state of all relevant platforms with respect to how well they met our control ideal state. We then evaluated the gaps from ideal in totality. We created an overall fleet health report for each platform managed at Google—not a report card, but a shared understanding of capabilities. For each platform, we evaluated the following:

- Can the platform support the control?
- Is the control turned on by default?
- Can we measure the state of the control?
- Is the fleet in compliance?

To drive objective measurement and equivalencies, you might consider:

- Anchoring these strategies in a shared measurement unit: time since patch released, geo-location, count

- Driving your measurements from a relative reference point: versions from current, features supported vs. implemented

Setting these standard measurements is the hard part. Once you have equivalency, your ability to discuss fleet health will greatly improve.

Where preventative controls are lacking or only partially effective, you can look for other ways to mitigate risk—for instance, higher monitoring/detection signal confidence or a compensating control that is more effective on a platform. You may find that you're relying on a subjective overall sense of robustness of the platform against attack. Modern operating systems have very complex attack surfaces, capabilities, and threat models; the best way we've found to aggregate all this information still boils down to manually comparing the desired characteristics of the device versus its actual characteristics. This comparison allows us to make high-level recommendations around projects to fill gaps and to prioritize those projects. No matter the source of the data driving these conclusions, it's important to document the rationale for the conclusion or at least the process that generated it. Doing so allows people beyond the immediate security engineers to understand the fleet state.

Deviations from Ideal

Despite all the best efforts to define, roll out, measure, and enforce controls, you may inevitably face the harsh reality that 100% uniform control deployment is a mythical state where unicorns frolic unconcerned about malware and state-sponsored attackers. You need to have a plan for deviations from the ideal state, root cause analysis, and exception handling.

Many deviations are naturally occurring, resulting from broken processes, faulty management tooling, flaky releases, and other root causes. For instance, there are often delays in applying patches on a system. It's important to understand when it makes sense to grandfather in exceptions fleetwide, and preventing the growth of the exception group versus when you should instigate hard corrections in control states. If you're clear about the threat model and user impact tradeoffs, you can drive good decisions here.

Exceptions should be measurable and time-based. We recommend you classify root causes in a consistent fashion across the fleet so that you can drive understanding around any gaps and identify places where controls are not suited to the fleet or certain classes of users. If an exception is perpetually renewed (or otherwise never expires), the control is not working. You should redesign the control or revisit your assumptions about its role in the fleet.

Getting Started

How do you start putting the BeyondCorp principles discussed in this article into practice on your own fleet? A general approach involves four main steps:

1. Define the security controls you care about.
2. Find a way to measure those controls.
3. Determine where your fleet isn't in compliance.
4. Fix workflows that don't work with your defined security stance or define exceptions.

The first essential step is defining the goals you want to achieve. You shouldn't create a set of desired security controls in a vacuum–these controls should be specific responses to threats you need to defend against. Explicitly enumerating threats provides you a heuristic to measure effectiveness and a framework to reason about the priority of individual properties. Consult partner teams (see "Lessons Learned," below) when defining and ranking desired qualities. As you clarify your threats and the controls that will mitigate them, build in tests such as unit tests or end-to-end red team assessments to evaluate how effective those controls are. Then you can determine whether they actually meet your security goals in practice.

In order to ascertain a device's security posture, you must be able to measure its current state versus the ideal state. If you haven't already, you'll need to roll out instrumentation software to your fleet to collect relevant data. However, raw data is only half of the story: you also need to define the ideal state your devices will be measured against. As a large fleet guarantees variation, you need to define multiple ideal states in order to cover all potential valid use cases.

Once you can measure the security stance of your fleet, you can start examining devices with deviations from the ideal. Some deviations might pose no security risk (as they're mitigated by compensating controls), but other deviations will uncover gaps. We focused our initial efforts on ensuring that new machines are in compliance with a control from the first moment employees use them. Once we knew that all new devices began their lives in a known good state, we could turn our attention to the rest of the machines in our fleet to improve overall fleet health.

Establishing an exception framework so you can create exceptions for the existing fleet when enforcing a new control is equally important. The deviation in the fleet will thus remain static, allowing you to remediate existing machines while keeping new machines in compliance. Once you isolate the problem to a grandfathered portion of the fleet, you can cluster failure reasons. These clusters will uncover problems shared by entire classes of devices or workflows. Tackling the largest and most risky of these clusters first will provide the largest security win for the smallest amount of effort. Repeat this clustering and remediation process until you have resolved the main issues in the fleet. One-off issues may need explicit exceptions if a user's workflow is explicitly not compatible with a desired security property.

While this system requires a lot of collaboration and hard work from many different teams, completing the effort gives you and your organization a more resilient position in the face of constant attack.

Lessons Learned

Instituting a coherent program for measuring and evaluating trust and fleet health is not a short-term project. Fully achieving the goals outlined in this paper (and the more general goals of BeyondCorp) requires significant resources. That being said, some lessons we've learned over the past couple of years can save you some time and headaches.

Set Milestones Early

Set key milestones sooner rather than later. Determine which properties you care about and rank them (at least roughly). This exercise helps you allocate resources efficiently and provides the motivation to implement large-scale projects. Incorporating data from a fleet management system into your authorization decisions is an excellent initial milestone. This alone will keep unknown devices from reaching your services and has the side benefit of providing a known good device inventory.

Decide How to Handle Exceptions

Define your approach to exceptions early in the project. Every fleet contains devices that cannot fully comply with the ideal security stance. Determining the procedural and technical implementation of exception management is key to a successful rollout. Define the reasons an exception can be granted, how to document those reasons, the maximum length of time an exception can exist before it must be reexamined, and the review process for existing exceptions.

Engage with Partner and Impacted Teams Early

A successful implementation of BeyondCorp requires work from the entire IT organization. Engaging with partner and impacted teams early in the process will dramatically streamline the enforcement portion of a rollout. For example:

- The device procurement and onboarding teams will need to ensure they keep the fleet management system up to date as devices are added or retired from the fleet.
- Other security teams will provide valuable input while defining machine security properties and potential inputs into the overall system.
- Traditional IT support teams will field the vast majority of user escalations. It is essential they understand the goals of the project and are able to help troubleshoot user issues.

You also need a way to communicate with the users who will be directly impacted by this change. Ensuring that the average user can actually follow and complete self-remediation steps reduces the load on IT and time wasted on troubleshooting.

Conclusion

Securing your employees' machines is a cornerstone to securing the crucial information your company handles. To this end, we thoroughly evaluate and regularly inspect all corporate devices to validate their health. Only known healthy devices can access critical internal systems and information.

Employees and their devices have already earned the attention of malicious actors, and it's up to you to defend employees while keeping them productive. To do that, you need a strong sense of fleet health, clear policies and measurements, and a process for handling deviations from the goal state. With consistent controls and enforcement, we believe every enterprise can simultaneously boost fleet health and security, improving resilience to an ever-increasing variety of attacks and threats.

Acknowledgments

While this continues to be a large cross-functional effort across Google and there are many contributors to this project, we want to acknowledge Cyrus Vesuna for his work on defining common trusted controls across our platforms.

References

[1] See Verizon, "2018 Data Breach Investigations Report: Executive Summary": https://www.verizonenterprise.com/resources/reports/rp_DBIR_2018_Report_execsummary_en_xg.pdf; Mandiant, *M-Trends 2018*: https://www.fireeye.com/content/dam/collateral/en/mtrends-2018.pdf.

[2] Google, "Fleet Management at Scale," November 2017: https://services.google.com/fh/files/misc/fleet_management_at_scale_white_paper.pdf.

[3] https://cloud.google.com/beyondcorp/#researchPapers.

[4] New variants often stretch the common understanding of classes of attacks, so you can't ignore variants completely. For instance, the industry thought we had a good grasp on micro-architecture security up until 2018—see Jann Horn, Project Zero (Google), "Reading Privileged Memory with a Side-Channel," January 3, 2018: https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html.

[5] Such as Intel's Boot Guard: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/4th-gen-core-family-mobile-brief.pdf.

[6] Microsoft's Defender Credential Guard: https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard.

[7] https://github.com/google/grr.

[8] For a description of trust levels and calculation, see B. Osborn, J. McWilliams, B. Beyer, M. Saltonstall, "BeyondCorp: Design to Deployment at Google": https://ai.google/research/pubs/pub44860.

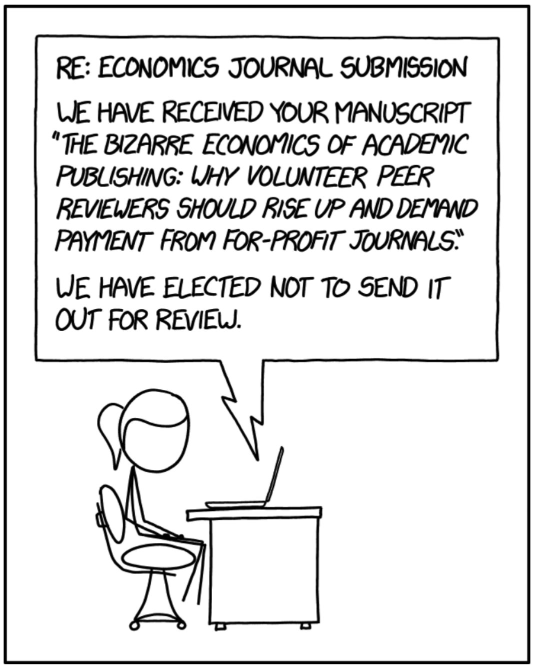
[9] https://osquery.io/.

[10] For more on the tools we use at Google, see "Fleet Management at Scale: How Google Manages a Quarter Million Computers Securely and Efficiently": https://ai.google/research/pubs/pub46587.

[11] For more on the trust inference system and the other moving parts of our BeyondCorp model, see B. Osborn, J. McWilliams, B. Beyer, M. Saltonstall, "BeyondCorp: Design to Deployment at Google": https://ai.google/research/pubs/pub44860.

[12] Dogfood: early release of products to employees to get feedback and catch bugs before a wider release.

XKCD xkcd.com


RE: ECONOMICS JOURNAL SUBMISSION
WE HAVE RECEIVED YOUR MANUSCRIPT "THE BIZARRE ECONOMICS OF ACADEMIC PUBLISHING: WHY VOLUNTEER PEER REVIEWERS SHOULD RISE UP AND DEMAND PAYMENT FROM FOR-PROFIT JOURNALS."
WE HAVE ELECTED NOT TO SEND IT OUT FOR REVIEW.