How To CISO

Applying Zero Trust Principles in a Cloud-Centric World

A How to CISO Handbook

Zero Trust … but to Which Cloud?

In the cloud-centric world enterprises increasingly operate in, there are different *interesting* environments that zero trust principles should be applied to. One of them we've somewhat talked about the evolution of the enterprise IT network. That network is becoming more and more obsolete, slowly being replaced with different environments. One environment is the cluster of the user and all of their devices (laptop, tablets, phone): the ~~*user-*~~*cyborg*. Another is the *SaaS-native cloud*, as applications and business processes are increasingly operated and hosted by third parties in shared cloud environments. And the final is the *public cloud*, where enterprises build and deploy their own applications, sometimes for consumption by their own employees, but even more often in service of their external users.

Bringing our zero trust principles to each of these clouds will have slightly different implementations – some of them easier, some of them harder – but they can happen at independent paces in many cases.

Zero Trust Principles

- Individualized, strong authentication
- Limited assumption of privilege
- Minimize unused privilege

From *Zero Trust Principles: a How to CISO Handbook*

The Zero Trust Enterprise Network

Applying zero trust in the core enterprise applications has basically been the thrust of the Zero Trust Network Access (ZTNA), Secure Access Service Edge (SASE), and Security Service Edge (SSE): upleveling the security practices around how enterprise networks control access to their network assets. Predominantly, enterprise network approaches have focused on protecting servers and applications from malicious users that might be (or masquerade as) users. The last decade of breaches have revealed a much greater issue: endpoints are required to trust far too many security and administrative tools. Breaches and failures of these administrative tools are the source (or at least a key contributor) to the massive blast radii that have led to many a bad corporate day.

The Zero Trust Cyborg

Most modern enterprises think about identifying *users*, which is an interesting difficult challenge when you don't have a human in front of you. The mental model underlying that

thought process is one in which the *devices* that a user possesses and uses are owned and administered by the enterprise and can be used to authenticate a user. In this model, only *humans* do interesting things as users.

This model often breaks down when we consider that a user's device might be compromised by an adversary, and all authentication steps are functionally passing through the control of an adversary. It's further complicated by the idea of *bring your own device (BYOD)*, which may be rare for laptops, but is quite prevalent for phones, tablets, and wearables.

If we define this group of devices, including the human who uses them, as an entity in their own right (the cyborg) instead of as two distinct sets of entities (a human and computing assets of a company), it becomes easier to think about how we can apply our zero trust principles in defense of the user environment.

Individualized, strong authentication. This principle is going to apply in two directions.

Internally, we want the cyborg to be able to authenticate its own members: a laptop authenticating the human that uses it, for instance. Modern systems provide us with fairly robust capabilities, from facial recognition to fingerprint identification, strong PINs and even passwords. While we should frown on the use of passwords for authentication *across* identity spaces, authentication *within* a cyborg identity is less problematic. To make this work, local passwords on devices should be just that: local. The password that a human uses to log into a laptop must be used nowhere else, and absolutely cannot be used for remote access.

Externally, we want a cyborg to be able to strongly authenticate itself to remote entities. We *might* make a requirement that at least two devices inside a cyborg must collaborate in this authentication step, perhaps with an initial connection-based authentication via an X.509/TLS certificate stored on a laptop, and then with an application-specific challenge to a smartphone that meets the FIDO2 specification. Note the absence of a requirement for human-known password-based authentication; the cyborg should use the knowledge of the human to approve its own use of authentication credentials, but sending that knowledge across the network weakens the long-term safety of the authentication (besides, ubiquitous password managers functionally remove human knowledge from the loop on most authentication steps).

A cyborg might operate in multiple contexts, and browser profiles and enterprise browsers offer a way to segregate authentication between those contexts. Many users might have a corporate profile for logging into company applications, and a personal profile[1] for checking on the applications for their life.

Limited assumption of privilege. If we're going to let the cyborg be considered as one entity, especially one that is *capable* of continuously authenticating over the network, then it is critical

[1] Chrome, I'd love it if you'd build a link dispatcher, so I can click on a link and explicitly choose which profile you send it to, instead of having to see a link, then focus on a Chrome window, switch to the right profile, then focus back to the original app to click the link.

that we limit how many *other* entities can assume the identity (and thus the privileges) of a given cyborg.

Obviously, the manufacturer of the devices in use presents that risk. Operating system vendors like Apple, Google, and Microsoft are the greatest hazard to the user cloud, as are the hardware manufacturers, but any piece of software on those devices, especially those with administrative control, are potential threat vectors. The Solarwinds and Kaseya breaches are obvious cautionary tales here, but so is every piece of ransomware. Remote administrative software and access are the greatest danger to the user cloud.

Minimize unused privilege. Within a cyborg, one of the most dangerous privileges is the ability to install new software, especially with administrative privileges. Enterprises often react to this by taking away the ability to install software from their users, requiring often convoluted processes for users to get access to applications that will make their jobs easier. Most modern operating systems have taken the opposite approach: removing the ability of *applications* to install themselves, requiring explicit user interaction to install new software. *Enterprise browsers* also seek to reduce unused privilege by creating enclaves for enterprise data, to limit which other applications are capable of accessing specific data.

Within a cyborg, if we think of it as a single entity, there become fewer useful opportunities to reduce unused privilege, *especially* once we remove the remote administrative accounts and services that had unused major privileges. Most of the relevant privileges for the user cloud exist elsewhere, as the user cloud interacts with applications over the Internet.

The Zero Trust SaaS-Native Cloud

More and more applications that power modern businesses are operated as websites by other entities, and often they communicate with each other on behalf of an enterprise. Companies being created today may have no on-premise applications *at all*. All of their business processes (and sensitive data) operate between multiple vendors, from HR to payroll, document management to graphic design, security operations to application development. As enterprises become SaaS-native, zero trust principles need to be applied *between* SaaS vendors, as well as *within* each SaaS vendor as they evaluate end-user capabilities. This isn't just as simple as "which identity, human or non-human, has access to an application," as data can often be directly shared with various applications, users, or even with the world.

Individualized, strong authentication. SaaS applications should know who they are communicating with at all times.

For end users, this is surprisingly complex. It isn't sufficient to merely validate a user cloud; part of the authentication step needs to validate that the user is *still* an employee. Single Sign On (SSO) systems are often used to make this easier: a user cloud will strongly authenticate itself to a centralized Identity Provider (IdP), which will hand the user cloud a secret token that

validates that user cloud's identity to a specific SaaS application. This makes provisioning "easy" at scale (by comparison to the alternatives, but anyone who has been tasked with setting up an SSO integration for the first time would be unlikely to call it easy), but creates weaknesses in the authentication flow. The token being exchanged is functionally a password, even if it's only known to the user cloud's browser and expires after 12 hours. If compromised, an adversary can use it to become the user for long enough to do harm. If the user leaves a company, SaaS applications may still grant access until the token expires. A stronger model might be to pair the user cloud's X.509 certificate and token; OCSP (expand) can be used to quickly validate that a user remains in the employ of the company, and cross-validating the token to the X.509 certificate limits theft.

SaaS-to-SaaS communications often happen via an application programming interface (API). While one might casually think this makes authentication an easy problem to solve (a non-human identity now means no humans in the loop!), that's rarely the case. Rather than using strong authentication, SaaS APIs often merely use API keys for authentication. While the term *key* evokes the idea of a cryptographic challenge, instead an API key is usually just a very long password, copied from one SaaS interface to another, sometimes via an OAuth token, but often by a human (who now has the API key sitting in their user cloud). API keys rarely expire, are annoying to rotate (rarely are APIs designed to support a rolling authentication when you update a key, so changing a key *requires breaking the ability to use the API* for the duration of the change), and are easily copyable. Non-human, third-party apps would ideally have a way to negotiate and upgrade their own authentication. A human *might* be required to authorize two SaaS apps to talk to one another the first time, but the applications should not rely on storing secrets negotiated by humans to engage in their own authentication.

Limited assumption of privilege. Designers of many SaaS systems don't actually support the *idea* of a client of their API being a third-party automated system. Often API keys are affiliated with user accounts, so when the key is provisioned for a single API, the client SaaS service now has all the access of the original user. The most likely user to provision an API key is often an administrative user, which means that the API key they give to another application can assume *a lot* of privilege.
When one SaaS application needs access to another, it should be provisioned with its own tightly restricted account. Account management is a challenge facing SaaS-native administrators, as keeping track of which SaaS application has accounts in which other SaaS applications is a hefty burden. And unlike users, where we can tie their authentication process all the way back to their employment status, with access being removed automatically when they become no longer employed, modern enterprise procurement systems have no comparable connections to technical systems.

Minimize unused privilege. Configuring privileges in SaaS systems is often challenging. Every vendor creates their own access control system, and it may have a handful of simple roles, or require you to configure access from thousands of primitive permissions. Ideally, you'll build your own mapping over SaaS systems, ensuring that users are tied to the permissions that they need, and that API clients are tightly tied to the APIs that they access.

The harder challenge may be in *identifying* unused privilege across hundreds of SaaS applications. Working across each one to identify permissions in its own configuration language, hopefully correlating with examining access logs that can be meaningfully tied to those privileges, is a large task that enterprises must tackle.

Zero Trust in the Public Cloud

More and more of the development that powers modern enterprise revenue streams is happening in public cloud service providers (CSPs) like AWS, GCP, AliCloud, Azure, and OCI. These CSPs bring an ease of deployment: rather than an engineering team needing to consider all of the infrastructure to support servers that run applications, CSPs provide much of that infrastructure automatically, and all that developers need to do is focus on critical application features. It's a beautiful model … if only it worked that smoothly. CSPs often default to the *shared responsibility model*, which requires an enterprise to *correctly* configure all of the services that they rely on.

When developer teams are becoming more agile and spending less time managing release pipelines, the opportunity to correctly configure security systems can be missed. From a zero trust perspective, what are we aiming for?

Individualized, strong authentication. Much like SaaS environments, user authentication can sometimes be the strongest, especially where users are connecting via SSH, or have already been strongly authenticated to the CSP's administrative console. Applications often have even stronger options, even if they are rarely used. Various cloud environments have workload-based permissions, where authentication is extremely strong: only the *workload* in question has that permission, and it can't be stolen and used elsewhere (although an adversary who has taken control of the workload can assume the role).

In many cases, developers have just brought forward the same system authentication architectures from the data center world: SSH identities stored on disk, TLS certificates stored on disk, shared secrets in configuration files (need I say … stored on disk). Cloud environments offer numerous key and identity management services to better secure and protect these authentication credentials, but enabling them safely is often a burden that enterprises have no plan to take on.

Limited assumption of privilege. In a cloud environment, we must accept that the cloud administrator does have the ability to assume virtually any privilege present in the cloud environment. The first step in limiting assumption abilities isn't to try to protect these privileges from the cloud administrator account. It's to reduce as much as possible the use of the cloud administrator account *at all*. It can't be completely eliminated, as you'll need the cloud

administrator account to provision other privileged accounts, but the number of times an administrator account is used should be minimized and carefully tracked.

Workloads should be provisioned with their own identities to hook entitlements to, rather than using shared identities (often tied to administrators). Creating a required assumption of privilege is directly opposite the goal we are trying to achieve.

Minimize unused privilege. Configuring entitlements in the public cloud is so challenging that there is already a market space *just* to solve one aspect of this one issue: cloud infrastructure entitlements management (CIEM). This can seem like an extremely daunting task to get *right*, and often that's because security teams start with an unattainable goal: eliminate *all* unnecessary privilege. That's a challenging task, not least because the teams you're going to work with aren't necessarily going to be trusting that you'll correctly reduce privileges down to what they actually need. Complications like infrequent automated processes, devops incident management, and job transfers all get into the mix.

But in many cases, the level of privilege exceeds the actual usage not by double-digit percentages … but by two orders of magnitude or more. The first step should be to quickly implement policies that reduce the completely unused permissions, which *often* correlate to administrative privileges that shouldn't be left lying around anyway.

There's Zero Reason to Wait

Implementing zero trust principles in your cloud-centric world – whether you focus on users, the SaaS-native cloud, or the public cloud – doesn't have to be a massive project. While some sub-projects might be large for your enterprise, you can quickly make progress on some of the tasks, often by checking with your existing vendors if they can quickly identify the low-hanging fruit for you, whether it's in your user cloud, your SaaS mesh, or your public cloud infrastructure.