PRISMA CLOUD | UNIT 42 BY PALO ALTO NETWORKS

CLOUD THREAT REPORT
2H 2021

Secure the Software Supply Chain to Secure the Cloud

Table of Contents

Foreword

What do misconfigurations and cloud supply chain insecurity have in common? Both are risks that increase as organizations shift more applications to the cloud. High-profile attacks like those involving SolarWinds and Kaseya have brought these risks top of mind for everyone, executive boards, DevOps, and security teams alike.

While risks related to supply chains have received a lot of attention in the media recently, what the discussions often overlook is that attackers don't necessarily modify source code repositories to facilitate these breaches. They don't have to. They **find weaknesses in the software development pipeline** and attack those.

> "The threat actor **did not modify our source code repository.** The malicious activity occurred within the automated build environment for our Orion Platform software."
> **– Sudhakar Ramakrishna, CEO, SolarWinds**[1]

Proactively addressing these threats is of the utmost importance. To this end, Unit 42's elite cloud threat intelligence team conducted a red team exercise on a customer's software development pipeline. **Posing as malicious insiders, Unit 42 researchers were able to infiltrate the development environment and show how an advanced persistent threat (APT) could carry out another SolarWinds/Kaseya-style attack** with the intent of compromising cloud assets and potentially impacting thousands of the customers' consumers.

What can be done to manage this growing threat in your cloud environments? Put simply: shift security left. Security teams can no longer ignore their build environments or ignore those of third parties that do software development on their behalf.

To provide guidance on shifting security left, the Unit 42 Cloud Threat Report, 2H 2021 draws on the Cloud Native Computing Foundation's (CNCF) recent paper on Software Supply Chain Best Practices to detail practical steps companies can take now to improve their security posture. Read on and find out how to get started.

Matthew Chiodi

Chief Security Officer, Cloud

Palo Alto Networks

@mattchiodi

1. Sudhakar Ramakrishna, "An Investigative Update of the Cyberattack," SolarWinds, May 7, 2021, https://orangematter.solarwinds.com/2021/05/07/an-investigative-update-of-the-cyberattack/.

Executive Summary

Despite the media coverage afforded to the SolarWinds and Kaseya breaches, our research indicates supply chain security in the cloud continues its growth as an emerging threat. Much remains misunderstood about both the nature of these attacks and the most effective means of defending against them. To better understand how supply chain attacks occur in the cloud, Unit 42 researchers analyzed data from a variety of public data sources around the world and, at the request of a large SaaS provider, executed a red team exercise against their software development environment. Overall, the findings indicate that many organizations may still be lulled into a false sense of supply chain security in the cloud. Case in point: Even with limited access to the customer's development environment, **it took a single Unit 42 researcher only three days to discover several critical software development flaws that could have exposed the customer to an attack similar to that of SolarWinds and Kaseya.**

Drawing on Unit 42's analysis of past supply chain attacks, the report explains the full scope of supply chain attacks, discusses poorly understood details about how they occur, and recommends actionable best practices that organizations can adopt today to help protect their supply chains in the cloud.

Supply Chain Flaws Are Difficult to Detect

During a red team exercise with a large SaaS provider, Unit 42 researchers were able to leverage misconfigurations in their software development environment that allowed researchers to control the customer's software development processes. With this level of access, researchers were able to control the flow of software, allowing them to perform a supply chain attack. They were able to do this by exploiting process gaps and critical security flaws like hardcoded credentials.

Red team exercises, like the one performed by the Unit 42 team, demonstrate an example of how poor security hygiene in the supply chain can impact cloud infrastructure. The customer, in this case, a large SaaS provider, maintains what most would consider a mature cloud security posture. However, Unit 42 researchers found that 21% of the security scans they ran against the customer's development environment resulted in misconfigurations or vulnerabilities (a number that squarely lines up with the industry average of 20%[2]). Researchers believe it is highly likely that the techniques employed during the red team exercise could be successfully performed against many organizations developing applications in the cloud.

Third-Party Code Poses a Hidden Risk

Unit 42 researchers found that 63% of third-party code templates used in building cloud infrastructure contained insecure configurations, based on a global analysis. However, **more shockingly, they found that 96% of third-party container applications deployed in cloud infrastructure contain known vulnerabilities**. In most supply chain attacks, an attacker compromises a vendor and inserts malicious code in software used by customers. Cloud infrastructure is susceptible to a similar approach in which unvetted code could introduce security flaws into cloud infrastructure and give attackers access to sensitive data in the cloud environment. An infrastructure flaw can be riskier than a software flaw, as an infrastructure flaw may directly impact hundreds of cloud workloads, such as virtual machines and data storage.

The challenge with third-party code is that it could come from anyone, including an advanced persistent threat (APT). This raises the stakes for code that's intended to be shared and used by others. Given modern cloud software development practices for sharing and incorporating third-party code—and creating complex structures that depend on many other building blocks—if an attacker compromises third-party developers or their code repositories, it's possible to infiltrate thousands of organizations' cloud infrastructures.

2. In July of 2021, Unit 42 researchers analyzed thousands of public repositories and found the global average to be 20%.

01

Supply Chain Insecurity

The SolarWinds incident was the first major software supply chain attack to make international headlines, but it was hardly the first breach of its kind. In 2017, Unit 42 published an article titled "The Era of Software Supply Chain Attacks Has Begun," which identified software supply chain attacks as a growing threat and predicted an increased focus on attacking "trusted" developers. The article also analyzed significant attacks that had occurred to date, including:

- **September 2015** – XcodeGhost: An attacker distributed a version of Apple's Xcode software (used to build iOS and macOS applications) that injected additional code into iOS apps built using it. This attack resulted in thousands of compromised apps identified in Apple's App Store®.
- **March 2016** – KeRanger: Transmission, a popular open-source BitTorrent client, was compromised through the injection of macOS ransomware into its installer. Users who downloaded and installed the program would be infected with malware that held their files for ransom. Attackers injected the ransomware by taking control of the servers used to distribute Transmission.
- **June 2017** – NotPetya: Attackers compromised a Ukrainian software company and distributed a destructive payload with network-worm capabilities through an update to the "MeDoc" financial software. After infecting systems using the software, the malware spread to other hosts in the network and caused a worldwide disruption that affected thousands of organizations.
- **September 2017** – CCleaner: Attackers compromised Avast's CCleaner tool, used by millions to help keep their PCs working properly. The compromise was used to target large technology and telecommunications companies worldwide with a second-stage payload.

In each of these breaches, attackers compromised software development pipelines, then used the trust placed in them to gain access to other networks. In this respect, supply chain attacks represent a fundamentally different type of threat because attackers target upstream developers to breach multiple targets through a single attack, rather than focusing on individual end-user organizations.

Red Team Exercise: Testing Cloud Supply Chain Security

While Unit 42 has been following software supply chain threats for years, the SolarWinds incident provided an opportunity to gain a deeper understanding of the specific nature of such incidents and test response strategies.

Prompted by the SolarWinds attack, a customer commissioned the Unit 42 team to attack their continuous integration (CI) environment in the cloud in order to help test the resiliency of their own supply chain. To initiate the attack, Unit 42 researchers masqueraded as malicious developers with limited access to the customer's CI environment and attempted to gain administrative rights to the larger cloud environment. This operation, although somewhat different from the SolarWinds attack path, illustrates how a malicious insider could harvest a CI repository to access sensitive information.

Path of Attack

Researchers were assigned a DevOps role commonly given to all developers in the customer's DevOps environment, including access to internal GitLab® repositories. This is where the customer's developers normally built all their applications. Had the customer followed security best practices around separation of duties for each developer, the Unit 42 exercise would not have been quite so efficient.

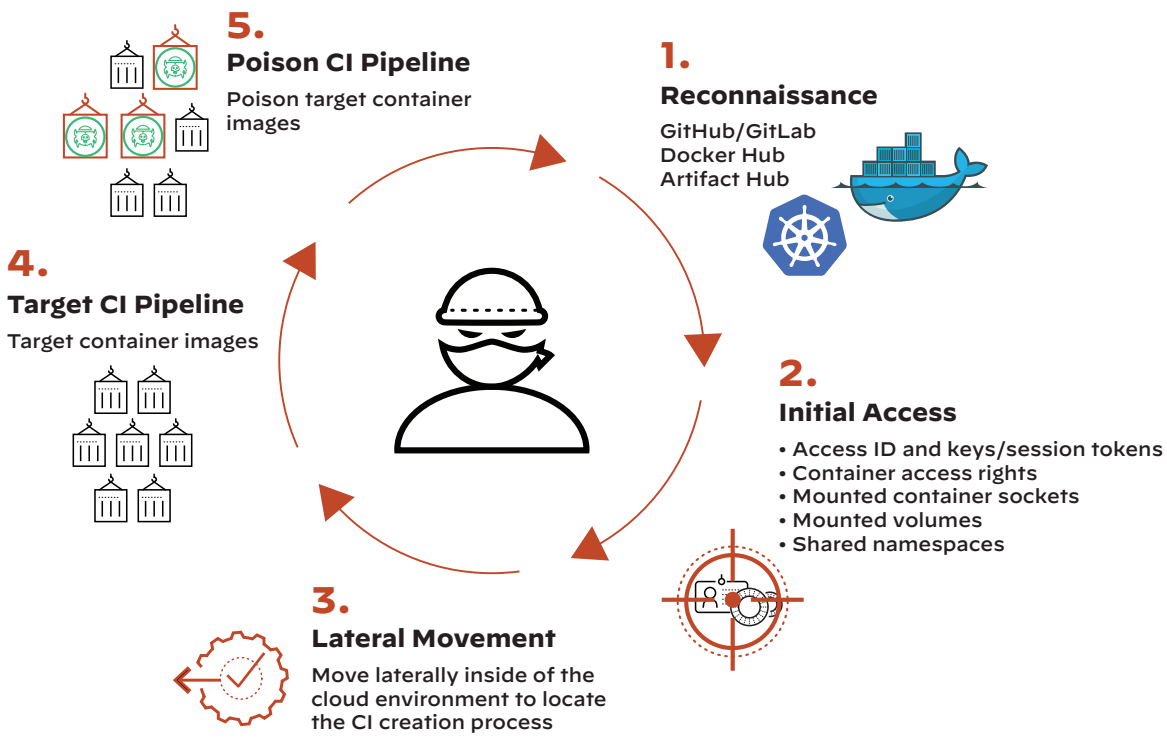

Figure 1: Steps to poison a CI pipeline

Without proper access controls to stop them, however, Unit 42 researchers were able to comb through the customer's GitLab resources. The team executed the following steps to escalate their privileges and gain admin-level access to the customer's software development pipeline (see figure 1):

1. **Reconnaissance**: Find, pull-down, and scan any known infrastructure as code (IaC) repositories associated with the customer and employees who work for the organization:
   a. Researchers wrote a custom script to efficiently download every accessible repository to their cloud lab environment. **In total, researchers exfiltrated more than 150 unique repositories**.
   b. Attackers and researchers alike will use a combination of custom scripts and open-source tools, like Checkov (a cloud infrastructure security scanning tool) to search for hardcoded identity and access management (IAM) credentials, misconfigurations, or service vulnerabilities. Each finding is then cataloged and assessed for future offensive operations.
2. **Initial access**: Use the findings to attempt initial access to the customer's cloud environment, for example, successfully exploiting a vulnerability to gain initial access, such as in a vulnerable web-based application, exposed SQL database, or exposed Docker® Daemon or Kubernetes® API, or leverage a misconfigured cloud entity:
   a. Researchers gained initial access by leveraging several of the hardcoded IAM credentials identified during the reconnaissance phase, which allowed the team to perform additional enumeration operations.
   b. Attackers could leverage the successful compromise of a container management host system, such as Docker or Kubernetes, via the exploitation of a misconfigured or vulnerable container that allowed attackers to "escape" from that container and gain access to the host.

3. **Lateral movement**: After a successful initial compromise of the organization's cloud environment, begin the process of moving laterally within the platform to acquire the final target:
   a. Researchers were successful in moving laterally, allowing them to escalate their access privileges with the iam:PassRole technique. This presented the researchers with additional access to the customer's cloud environment and CI pipeline.
   b. Attackers will continually cycle through reconnaissance, initial access, and lateral movement operations. Further access to the cloud environment can be achieved with the goal of compromising the CI pipeline.
4. **Target CI pipeline**: Locate and compromise the customer's CI pipeline by targeting developer accounts and CI pipeline applications, such as automation servers (for example, Jenkins®, JFrog®, or CircleCI®):
   a. Researchers used identified IaC templates located within repository managers, such as GitLab, and repository storage applications, such as HashiCorp's Vault, to identify the leakage of sensitive information, such as hardcoded IAM credentials. **Researchers found 26 IAM credentials hardcoded within IaC templates and configuration and log files** using this technique, which allowed them to escalate their access to allow for CI pipeline modifications.
   b. Attackers will use vulnerability and misconfiguration scanning tools on CI pipeline applications, such as Jenkins, JFrog, or CircleCI to expose weaknesses allowing for further compromises, such as access to the CI pipeline codebase.
5. **Poison CI pipeline**: Modify select CI packages to build poisoned applications. With the access obtained from the previous step, researchers now could poison the client's CI pipeline. However, since code modification was not permitted within the researchers' scope of operations, the following two techniques illustrate which steps an attacker could take against a CI pipeline:
   a. **Poisoning core application codebase**: This is the process of modifying the vendor's source code to create a backdoor network connection to malicious command and control (C2) infrastructure. Typically, vendors scan and review their core source code for misconfigurations and vulnerabilities frequently. This process will often catch attempts to modify source code directly. This is one of the reasons why, as in the case of SolarWinds, attackers went after the supporting environment plugins instead of the core code base.
   b. **Poisoning support plugins**: This is the process of modifying the code of plugins that support the infrastructure for the core application. These modified supporting plugins can result in the establishment of backdoor network connections to malicious C2 infrastructure within victim environments. Unlike modifying an application's core source code directly, modifying support plugins may allow an attacker's malicious modifications to go undetected. In the case of the SolarWinds malicious plugin, SUNBURST, the plugin was signed with a legitimate SolarWinds digital signature, which likely allowed the plugin to go unnoticed for a long period of time.

Using this approach—which parallels the one attackers would follow in a real-world supply chain breach—the Unit 42 team was able to download every GitLab repository from the customer's cloud software storage location. From this codebase, Unit 42 researchers identified nearly 80,000 individual cloud resources (such as cloud virtual machines, databases, and cloud storage instances as well as network infrastructure like virtual private cloud (VPC) and network gateway entities) within 154 unique CI repositories. Further, within the repositories, **researchers found 26 hardcoded IAM key pairs, five of which were session tokens instead of access keys**. Session tokens traditionally only remain active for a short period of time (one hour by default), making them less likely to incur substantial security risk over long durations. By contrast, access keys can stay active for months, or even years, and as such, pose a more significant risk to cloud environments over longer periods of time.

Using these hardcoded IAM key pairs, researchers were able to escalate their privileges via the iam:PassRole functionality, allowing them to create an EC2 instance granting administrative access to the customer's larger cloud environment. Notably, this is the same attack technique that researchers detailed within the 2H 2020 Cloud Threat Report support blog. Having now witnessed the same vulnerability within two separate environments, Unit 42 researchers can further infer this technique would be successful in many other cloud environments. See figure 2 for an example of a misconfiguration that could allow for the malicious use of the PassRole functionality.

```
"Effect": "Allow",
"Action": "iam:PassRole"
"Resource": [
    "arn:aws:iam::*:/role/flowlogs-*",
],
"Condition": {
    "StringEquals": {
        "iam:PassedToService": [
            "ec2.amazonaws.com"
        ]
    }
}
```

Figure 2: PassRole example

Once Unit 42 researchers were able to escalate their privileges to the administrator level, they could perform any number of actions within the cloud environment, including poisoning CI operations to insert malicious code into the customer's software development pipeline. Researchers stopped at this point, however, because the scope of the engagement did not include actual code modifications.

Caught by the SOC—but It Was Already Game Over

After breaching the customer's CI resources and gaining full access to their cloud environment, the Unit 42 team looked more broadly at the customer's security operations and response capabilities to help them improve their security posture and prevent similar breaches by malicious parties.

To further assess the organization's security posture, Unit 42 researchers employed the open-source Palo Alto Networks reconnaissance tool IAMFinder, as described in detail in this blog. The IAMFinder tool allowed researchers to enumerate AWS user and role accounts without API logs and error messages appearing within the customer's AWS environment. Since this technique does not send error messages of attempted enumeration or reconnaissance efforts to the customer's logging operations, the customer was not able to detect the reconnaissance efforts. It was not until researchers switched to authenticated API requests initiated by the command-line interface tool, awscli, that the customer's properly configured Amazon GuardDuty® service detected the researchers' malicious activity. GuardDuty is an AWS security tool that continuously monitors configured AWS accounts for malicious activity. What's more, GuardDuty detection occurred on only one of more than 50 identified AWS accounts used by the customer and its developers. Unit 42 researchers noted that GuardDuty did flag the account based on a malicious User-Agent string used by one researcher's test machines. However, it was unclear if any of the other AWS accounts were properly configured by the customer for GuardDuty monitoring.

The customer's integration of Amazon GuardDuty with a cloud security posture management platform—in this case, Palo Alto Networks Prisma Cloud—was essential to the detection of the attack. Still, because the customer properly configured *only one of the accounts* to be monitored by GuardDuty, only a small fraction of the overall malicious activity came to light in the SOC.

> **Note:** In the cloud shared responsibility model, configuration of cloud service provider (CSP) offered security service belongs firmly with the customer, not the CSP. In the case of our customer, while they used GuardDuty, they failed to configure it according to AWS best practices.

Once alerted to the malicious actions, the customer's security operations center (SOC) team was able to identify the usage of each compromised IAM access key enumerated by researchers. The SOC team also effectively employed a security orchestration, automation, and response (SOAR) tool that automated the deactivation of the identified compromised keys in near-real time, thus effectively halting the second phase of the red team exercise.

Lessons Learned

At the conclusion of the exercise, Unit 42 researchers worked with the organization's SOC, DevOps, and red and blue teams to develop a plan of action on how to shift security left. A big part of this was the early identification of suspicious or malicious operations within their software development pipeline. The customer adopted the following short-term, tactical steps, after which the Unit 42 Security Consulting team helped to define a larger, longer-term cloud security strategy:

- Prevent the exfiltration of DevOps repositories from GitLab environments for non-essential user accounts by restricting access:
  - » Developers should only have access to the specific repositories relevant to their work.
  - » Security measures should be put in place to limit the access and download capabilities of repositories outside of their working requirements.
  - » Role-based access control (RBAC) should be employed to limit access to repositories to only the user accounts required to maintain or modify those repositories.
- Implement cloud platform detection rules for sensitive API requests originating from outside of the organization's network range. The customer confirmed access to certain sensitive APIs typically only takes place internally, and therefore external access should immediately be deemed suspicious.
- Implement cloud platform detection rules for user-specific API requests directed toward IAM service accounts:
  - » For example, if an IAM service role is created for a Fluentd service, that service account should only maintain API access to read and write data (log information) from select cloud resources. This service account should not be allowed to perform API requests for reading or listing IAM users, roles, or policy information.

02

Infrastructure as Code: A Key to Supply Chain Protection

In most known supply chain attacks, attackers compromised software vendors' continuous integration and continuous delivery/deployment (CI/CD) pipelines and injected malicious code into packages that millions of downstream users depend on.

Defending against this risk is difficult for two reasons. First, there are hundreds of dependent packages maintained by open-source developers in a typical modern cloud native application. Each of these dependencies carries a certain risk and can be a vector to another supply chain attack. Second, third-party packages are routinely imported into supply chains via infrastructure as code (IaC) templates that organizations don't always inspect sufficiently. As a result, security vulnerabilities can creep in and remain undetected.

For example, when creating an IaC template, developers commonly bootstrap infrastructure using multiple third-party packages, such as VPC networks and Kubernetes clusters. Then, when deploying the infrastructure, all the dependent packages need to be downloaded and integrated. As cloud native applications grow more complex, an organization's cloud infrastructure will rely on more and more dependent packages. This chain of dependency, similar to the concept of the supply chain, can soon become troublesome and difficult to manage. Indeed, our research shows a positive correlation between the number of security flaws and dependencies.

Figure 3 illustrates how a simple cloud infrastructure built with IaC can inherit many misconfigurations and vulnerabilities from the dependent packages. This deployment pipeline contains three main stages: *cloud infrastructure provisioning, Kubernetes Helm chart deployment* (Helm is a Kubernetes deployment tool for automating the packaging and deployment of apps and services to Kubernetes clusters), and *container image instantiation*. At each stage, different dependent packages are deployed, and various security issues can be subsequently introduced. The following are a few examples of how security issues can be introduced in each stage:

- **Cloud infrastructure provisioning**: The IaC uses three types of Terraform modules: VPC, Kubernetes, and storage. The storage module also depends on two other modules. After all five modules are deployed, a total of **nine misconfigurations** are introduced.
- **Kubernetes application deployment**: A Kubernetes application is deployed using a Helm chart. This main Helm chart depends on two other Helm charts, and each Helm chart also depends on two container images. After all three Helm charts are deployed, a total of **five Kubernetes misconfigurations** are introduced.
- **Container image instantiation**: Each container image is instantiated to one or more containers across the cluster. Inside each container, the application code and all its dependent packages are loaded into memory. After all eight containers are deployed, a total of **40 application vulnerabilities** are introduced.

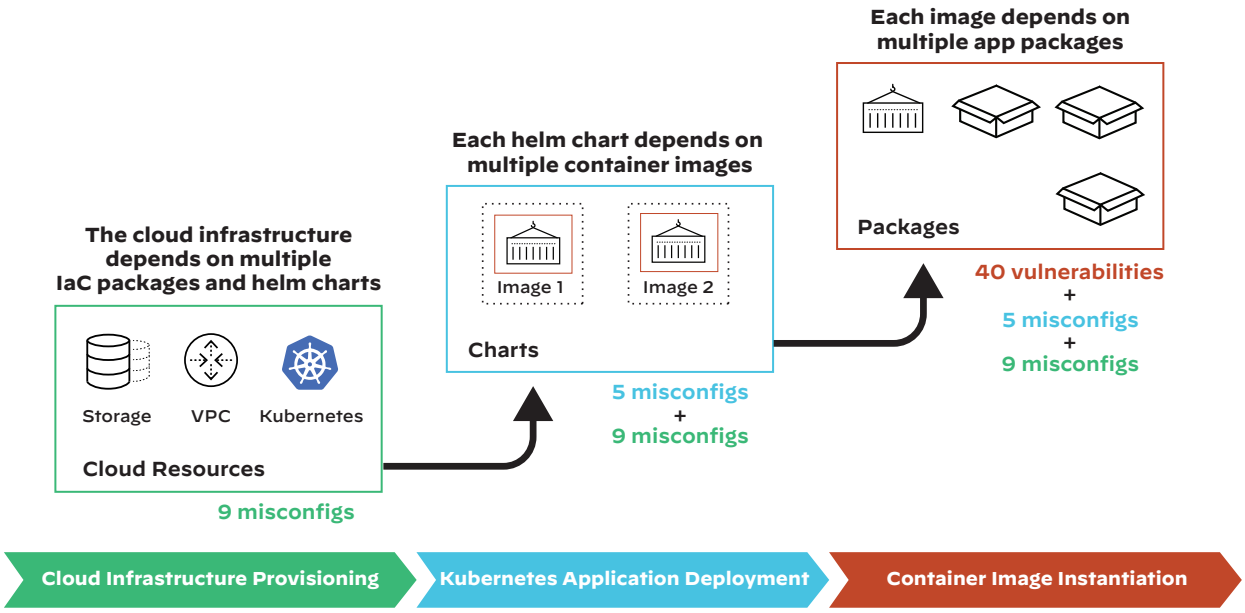

Figure 3: Chain of dependencies in a modern cloud native application

This example highlights how vulnerabilities and misconfigurations can quickly snowball within the context of supply chain dependencies. Although the deployed infrastructure in the examples will be fully functional, the default configurations of the dependent packages may not be secure. **If any of the dependent packages are compromised, millions of cloud environments relying on them could become vulnerable to attacks like those in recent history.**

To dive deeper into the scope of IaC-related security problems at each stage of the cloud native software dependency chain, Unit 42 researchers analyzed thousands of Terraform scripts, Helm charts, and Docker images.

Note: The following IaC misconfigurations are created by the cloud user, not by CSPs or IaC providers. In the context of the shared responsibility model, IaC template configuration belongs entirely to the cloud user. The challenge for organizations is ensuring that secure IaC configurations are consistently enforced across multiple cloud accounts, providers, and software development pipelines.

Security Teardown: Terraform Modules

Unit 42 researchers used Bridgecrew's Checkov to analyze 4,055 Terraform templates and 38,480 Terraform files in popular open-source Terraform repositories. The owners of these templates can be a CSP, a vendor, or any open-source developer. Checkov is an open-source static code analysis tool for infrastructure as code. Overall, 63% of the Terraform templates contain one or more insecure configurations, and 49% of the templates contain at least one critical or highly insecure configuration. **Considering the number of times each module has been downloaded, 64% of the downloads result in at least one high or critical insecure configuration.**

Anyone can publish a module on open-source Terraform repositories, and all the Terraform modules are open-sourced and available on GitHub. **The advantage of open-source tools is that anyone can scrutinize the code, but the downside is that no one is obligated to maintain or secure the code.** While new cloud features and services are released on a daily basis, some modules have not been updated for years, like third-party modules, artifactory (not from JFrog), and fpc-ops.

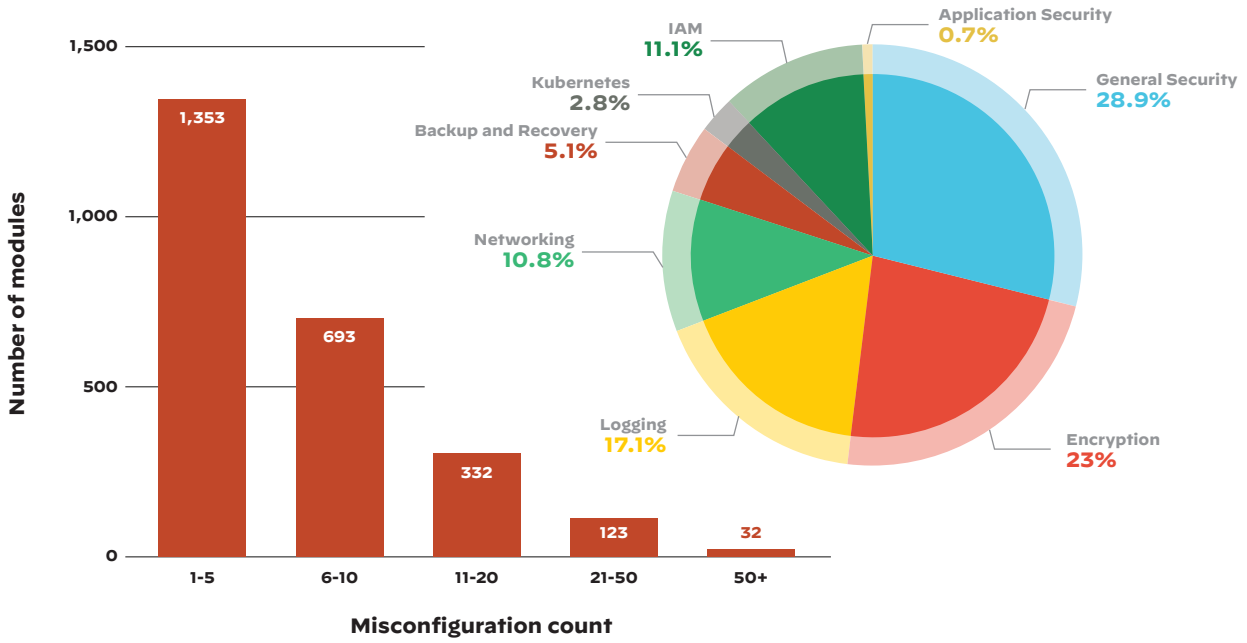

Figure 4: Public Terraform modules by number of misconfigurations (left); types of misconfigurations and their percentages (right)

We found similar misconfigurations across different CSPs, such as unencrypted data in cloud storage, disabled logging for cloud workloads, and publicly exposed remote management services, as shown in table 1. The detailed explanation and remediation process for each misconfiguration is available in the AWS policies for Palo Alto Networks Bridgecrew.

Table 1: Top Five Most Common Alerts Addressing Insecure Configurations for Each CSP	
AWS	Ensure all data stored in the Launch configuration EBS is securely encrypted
	Ensure all data stored in the S3 bucket has versioning enabled
	Ensure all data stored in the S3 bucket is securely encrypted at rest
	Ensure no security groups allow ingress from 0.0.0.0:0 to port 22
	Ensure VPC flow logging is enabled in all VPCs
Azure	Ensure default network access rule for Storage Accounts is set to deny
	Ensure storage for critical data is encrypted with Customer Managed Key
	Ensure that the expiration date is set on all secrets
	Ensure Storage Account is using the latest version of TLS encryption
	Ensure that 'Secure transfer required' is set to 'Enabled'
GCP	Ensure that VPC Flow Logs is enabled for every subnet in a VPC Network
	Ensure 'Block Project-wide SSH keys' is enabled for VM instances
	Ensure VM disks for critical VMs are encrypted with Customer Supplied Encryption Keys (CSEK)
	Ensure that instances are not configured to use the default service account
	Ensure all Cloud SQL database instances require all incoming connections to use SSL

Security Teardown: Kubernetes Helm Charts

Unit 42 researchers analyzed 3,155 Helm charts and 8,805 YAML files in Artifact Hub using helm-scanner. Overall, 99.9% of the Helm charts contain one or more insecure configurations, and 6% of Helm charts contain at least one critical or highly insecure configuration.

Similar to Terraform modules, each Helm chart can depend on other charts, and each dependent chart can also depend on others. Twenty-nine percent of the Helm charts in Artifact Hub have one or more dependencies. **For charts with dependencies, 62% of the misconfigurations come from the dependent charts.** One of the charts we analyzed pulled in a total of 23 dependent charts when deploying, and 92% of the misconfigurations were in the dependent charts. Figure 5 (bottom) shows a positive correlation between the number of dependencies and the number of misconfigurations.

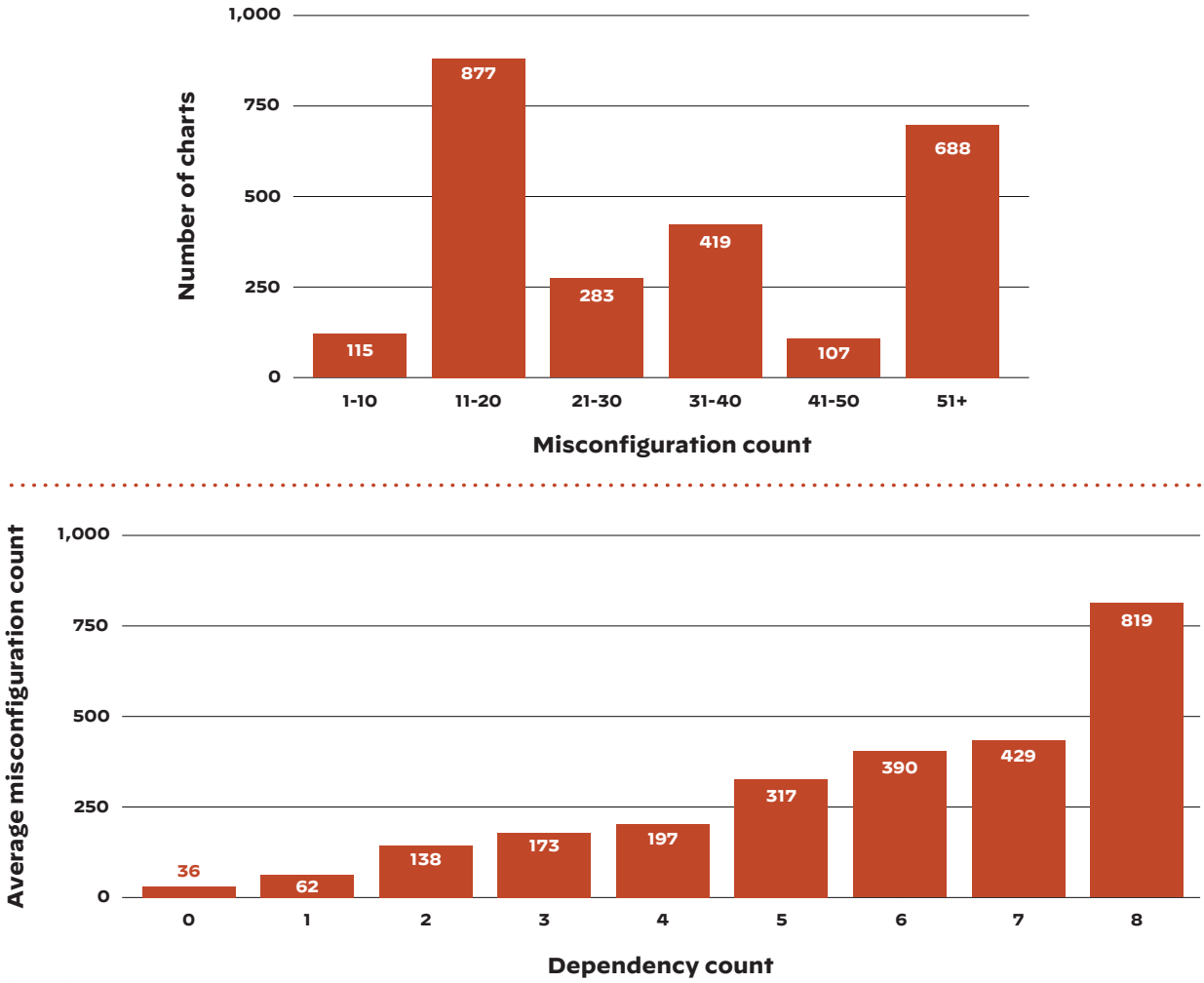

Figure 5: Kubernetes Helm charts categorized by number of misconfigurations they contain (top); average number of misconfigurations for Helm charts with different numbers of dependencies (bottom)

The most common and concerning misconfigurations we saw were over-privileged containers, as shown in table 2. Most containers don't need to run in privileged mode or require system administration capability. Privileged containers can access devices on the host and bypass namespace isolation. If a privileged container is compromised, the attackers can easily gain access to the host or even the entire cluster. For a deeper dive into securing Kubernetes and Helm charts, check out this series of blog posts.

Table 2: Top Five Critical or High Kubernetes Violations
Container should not be privileged
Do not use the CAP_SYS_ADMIN Linux capability
Do not admit privileged containers
Ensure that the --bind-address argument is set to 127.0.0.1
Ensure that the --kubelet-certificate-authority argument is set as appropriate

Security Teardown: Container Images

Unit 42 researchers analyzed the container images used in the Kubernetes Helm charts. A total of 1,544 distinct images were studied. These container images were hosted in various public registries, such as Docker Hub, Quay®, and Google Container Registry (GCR). **Overall, we found known vulnerabilities in 96% of the images, and 91% of the images contain at least one critical or high vulnerability.**

Vulnerabilities can be introduced if a Helm chart maintainer fails to update the charts or a container image maintainer fails to update the images. Many widely used libraries, such as libgcrypt20 and openssl, are well-maintained and always have vulnerability-free versions available. However, most users don't download and compile these libraries from the source code. Instead, they rely on package managers such as rpm, dpkg, and container image maintainers to build the source code into packages that can be easily used on different platforms. The patched source code can reach the end users only when all the intermediate package managers have all integrated the updates.

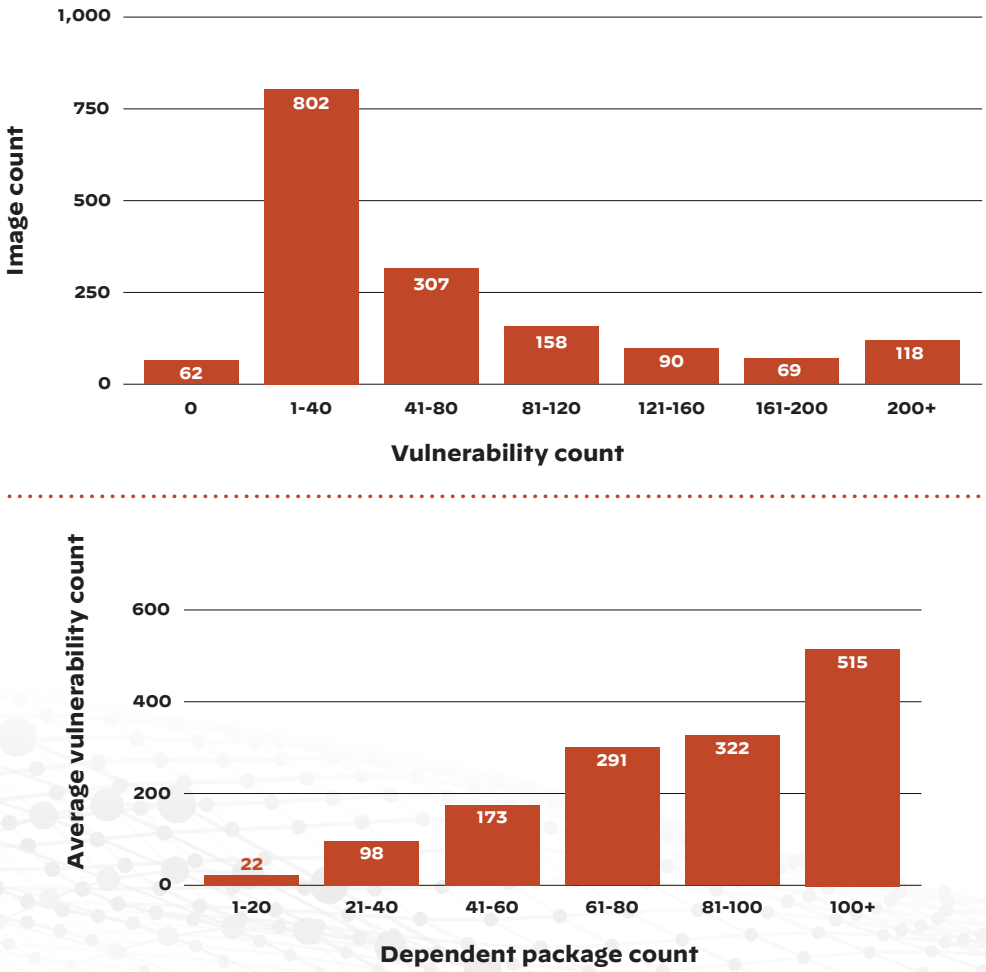

Figure 6: Container images categorized by the number of vulnerabilities they contain (top); vulnerability count of container images with different numbers of dependent packages (bottom)

03

Conclusion and Recommendations

The key takeaway from our analysis is that despite much talk in the security community about shifting left, organizations are still very much neglecting DevOps security, due, in part, to lack of attention to supply chain threats. Cloud native applications have a long chain of dependencies, including open-source tools and components from other vendors and developers. Complicating matters, each dependency may also have other dependencies. This leads us to the conclusion that **DevOps and security teams must gain visibility into the bill of materials in every cloud workload**. Only with this visibility is it possible to evaluate the risk in every stage of the dependency chain and create guardrails to contain the impact if any stage is compromised.

However, **believing that code scanning at the end of the development lifecycle is sufficient, many organizations have a false sense of security in the cloud**. This has led to development environments becoming the vector of choice for APTs. This was certainly the case for the SolarWinds breach as well as what *could have been* a major breach for our customer had the Unit 42 team not identified the customer's supply chain vulnerabilities before a malicious attacker did.

Securing Your Supply Chain

To address cloud software supply chain security holistically, the Cloud Native Computing Foundation (CNCF) released a best practices white paper in 2021. The white paper focuses on five key areas: securing the source code, securing materials, securing build pipelines, securing artifacts, and securing deployments. The Unit 42 team recommends an in-depth read of this document and embedding these best practices into your cloud security strategy.

The challenge many organizations will face, however, is turning theory into practice. If your organization does not have a defined cloud security strategy, we recommend starting with the Big Cloud 5 framework. This includes some key aspects from the CNCF's best practices guide. A focal point of the framework is embedding security into the software development pipeline. Your team can get started by working through the three steps that follow.

1. Define Your Shift-Left Security Strategy

The first step of any journey is to define where you intend to go. Do not underestimate the power of a concisely written strategy document (ideally one page). It is critical to define what shift-left security means in your organization. This is about painting the most vivid picture possible for your teams so they know what success looks like. Key items to include in this document are vision, ownership/responsibility, milestones, and metrics. Expect the strategy document to mature over time, and don't spend too much time trying to perfect it. Iteration over time is essential.

2. Understand Where and How Software Is Created in Your Organization

Perhaps one of the most challenging aspects of shifting security left is getting a handle on how and where software is created in your organization. Depending on the size of your company, this could run the gamut from straightforward to extremely challenging. This step is significant because the end result is what allows the security team to understand where they can move security closer to development. Large organizations that have not undertaken this process will likely spend a few months digging. Oftentimes, development is outsourced to multiple vendors, which will require additional work and sometimes contract reviews. Small and medium-sized organizations will find this step relatively straightforward but equally rewarding.

The goal of this step is to first look organization-wide and document the overall flow of software in your company. Medium to large organizations will want to start at the macro level and then drill into individual business units. It is highly likely that each business unit will have its own software development process and tools. Key items to identify in this phase include who is developing code (people), how it flows from development laptops to production (process), and which systems they are using to enable the process (technology).

3. Identify and Implement Security-Quality Guardrails

Quality assurance has always been part of the software development lifecycle. However, software quality has not historically included security. This must change, and the work done in the previous steps will arm you to do this. Every step of the software development process is an opportunity to give feedback and look for security issues. The most effective security teams start small. They arm development teams with simple and effective tools that become part of the daily development routine. Open-source tools that fit well into this category include Bridgecrew's Checkov, Yor, and AirIAM.

What Shift-Left Security Looks Like

It's quite common to have a security strategy that focuses almost entirely on the "right" side of the software development pipeline. Using the visual that follows, which represents what we will call Scenario 1 (figure 7), development starts without security, and software quality is only checked during runtime. This often results in an uneasy conversation between security and development when vulnerabilities are found, not to mention a software development pipeline that is more open to attack from APTs.

Scenario 1

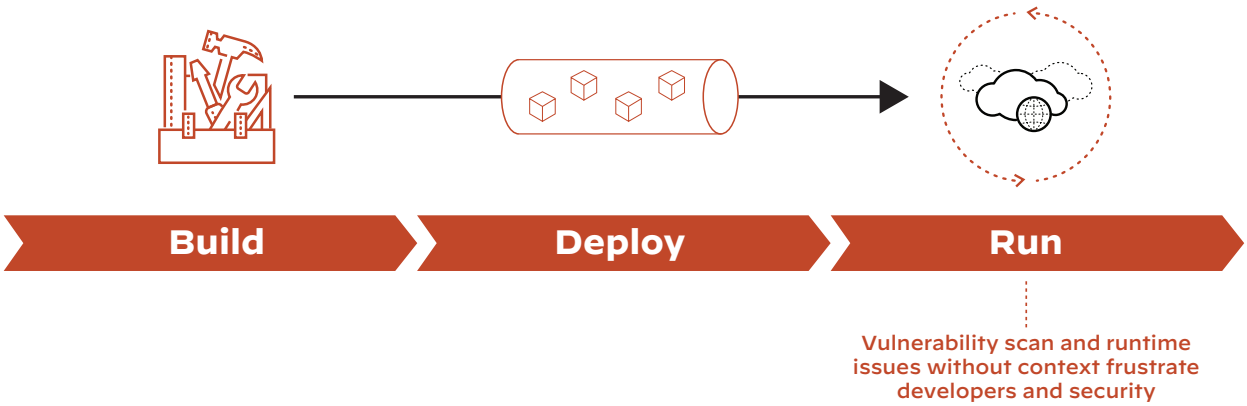

Figure 7: Development without security

In Scenario 2 (figure 8), however, security teams have shifted security left by investing the time to understand the development process within their organization. They have also embedded security processes and tools into all stages of the CI/CD pipeline, resulting in automated security guardrails.

Scenario 2

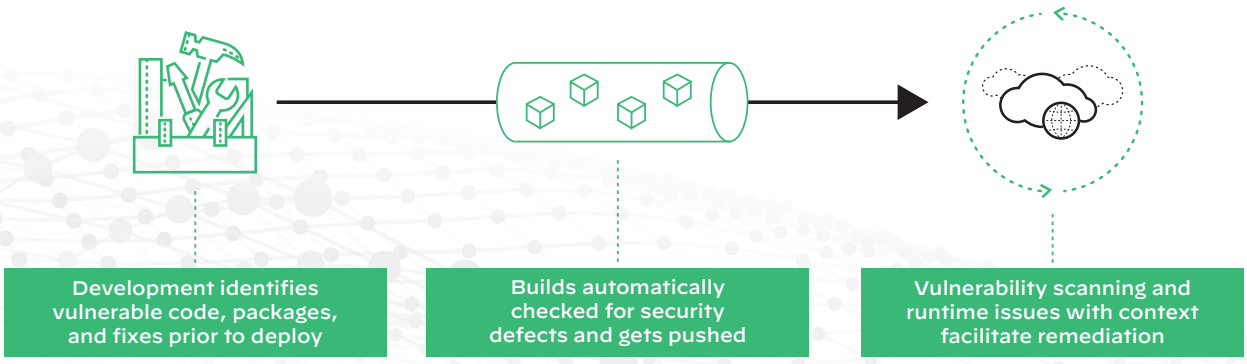

Figure 8: Development with security integrated

Every organization today needs to strive for Scenario 2. As headline-grabbing supply chain breaches have made all too clear, the cloud supply chain remains a critical vulnerability for many businesses. The nature of this vulnerability is not always fully understood by stakeholders, in large part due to a failure to recognize the expansive risks that can arise from complex, multilayered dependency chains within software development pipelines.

Utilizing the three previous steps will put your organization on a solid path toward not only shifting security left but making security synonymous with development. As your organization moves toward shift left as part of its cloud journey, it is critical that security controls be automated and part of every phase of the software development lifecycle.

Ready to Identify the Threats in Your Cloud?

Prisma® Cloud analyzes more than 10 billion events every month. By proactively detecting security and compliance misconfigurations as well as triggering automated workflow responses, Prisma Cloud helps ensure you continuously and securely meet the demands of your dynamic cloud architectures. To get started with Prisma Cloud, request your free trial today.

Methodology

In July of 2021, Unit 42 researchers used the publicly available data from open-source Terraform repositories, Artifact Hub, Docker Hub, Quay, and Google Container Registry (GCR) to evaluate the security posture of the modern cloud native software supply chain. The data in these public repositories are global and not restricted to specific regions. Palo Alto Networks IaC scanning tool Checkov and helm-scanner were used to analyze the downloaded Terraform templates, Kubernetes Helm charts, and Kubernetes YAML templates. Palo Alto Networks Prisma Cloud container vulnerability scanner was then used to analyze the components and vulnerabilities in each container image.

Bridgecrew by Prisma Cloud

Developed and expanded in collaboration with customers and partners, Bridgecrew has been vital in connecting code to the cloud and engineering to security. By providing security feedback in code, with code, and making it accessible through existing developer tools and workflows, the Bridgecrew platform has transformed the way hundreds of teams secure and govern infrastructure.

Palo Alto Networks WildFire

The cloud-based WildFire® malware prevention service employs a unique multi-technique approach, combining dynamic and static analysis, innovative machine learning techniques, and a groundbreaking bare metal analysis environment to detect and prevent even the most evasive threats.

Palo Alto Networks AutoFocus

The AutoFocus™ contextual threat intelligence service provides the intelligence, analytics, and context required to understand which attacks require immediate response, as well as the ability to make indicators actionable and prevent future attacks.

About

Prisma Cloud

Prisma® Cloud is a comprehensive cloud native security platform with the industry's broadest security and compliance coverage—for applications, data, and the entire cloud native technology stack—throughout the development lifecycle and across hybrid and multicloud deployments. Prisma Cloud's integrated approach enables security operations and DevOps teams to stay agile, collaborate effectively, and accelerate cloud native application development and deployment securely.

Unit 42

Unit 42 brings together our world-renowned threat researchers with an elite team of security consultants to create an intelligence-driven, response-ready organization. The Unit 42 Threat Intelligence team provides threat research that enables security teams to understand adversary intent and attribution, while enhancing protections offered by our products and services to stop advanced attacks. As threats escalate, Unit 42 is available to advise customers on the latest risks, assess their readiness, and help them recover when the worst occurs. The Unit 42 Security Consulting team serves as a trusted partner with state-of-the-art cyber risk expertise and incident response capabilities, helping customers focus on their business before, during, and after a breach.

Authors

Nathaniel "Q" Quist, Senior Threat Researcher, Public Cloud Security, Palo Alto Networks

Jay Chen, Senior Threat Researcher, Public Cloud Security, Palo Alto Networks

Matthew Chiodi, Vice President and Chief Security Officer, Cloud, Palo Alto Networks

Contributors

Matt Johnson, Bridgecrew Developer Advocate Lead, Palo Alto Networks

Stephen Giguere, Bridgecrew Developer Advocate, Palo Alto Networks

paloalto® NETWORKS

3000 Tannery Way
Santa Clara, CA 95054

Main: +1.408.753.4000
Sales: +1.866.320.4788
Support: +1.866.898.9087

www.paloaltonetworks.com