A Quick Look at ELF Bifrose (Part 1)

cyberandramen.net/2022/12/30/a-quick-look-at-elf-bifrose/

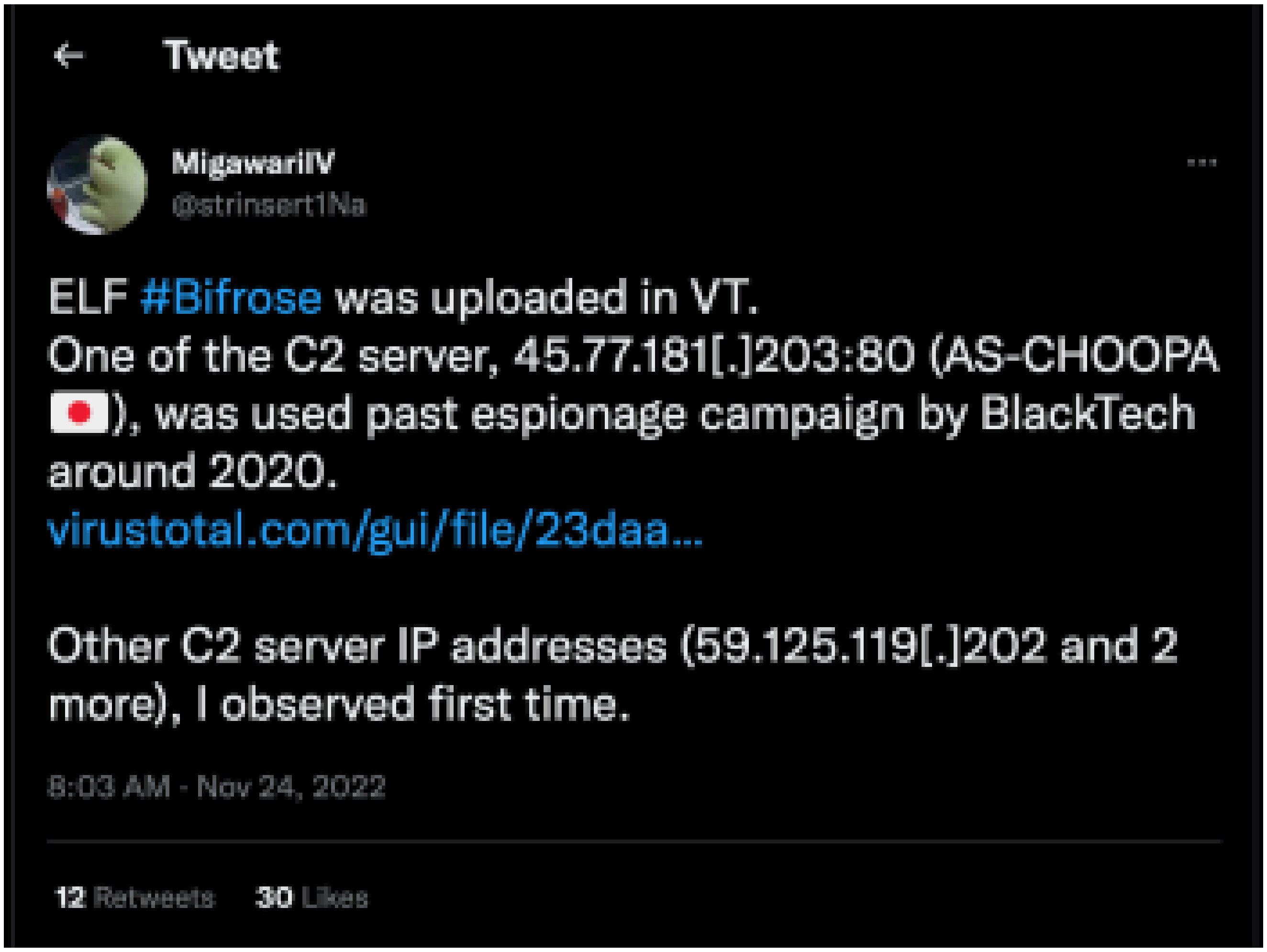
December 30, 2022

Bifrose or Bifrost is a backdoor initially targeting Windows systems with a long history. First identified in the early 2000’s, it is believed a hacking group (likely BlackTech), purchased the source code or gained access to it around 2010, and enhanced the malware for use in its own campaigns.

BlackTech has long targeted both Windows and Unix-based systems with a variety of malicious software, tailoring different malware to each campaign.

It Started With A Tweet

On 24 November, Twitter user @strinsert1Na tweeted that a recent ELF Bifrose sample had been uploaded to VirusTotal.


Tweet

MigawariIV
@strinsert1Na

ELF #Bifrose was uploaded in VT.
One of the C2 server, 45.77.181[.]203:80 (AS-CHOOPA 🇯🇵), was used past espionage campaign by BlackTech around 2020.
virustotal.com/gui/file/23daa...

Other C2 server IP addresses (59.125.119[.]202 and 2 more), I observed first time.

8:03 AM · Nov 24, 2022

12 Retweets 30 Likes

Figure 1: Tweet courtesy of @strinsert1Na

While the reuse of command and control (C&C) infrastructure is nothing new for BlackTech, the operators have consistently added new features to the backdoor, while seemingly not changing the targets of their attacks.

“udevd-10.138.61.156”

As of the time of writing, the latest Bifrose sample is detected by about half of the vendors on VirusTotal, scoring a 36 out of 64.

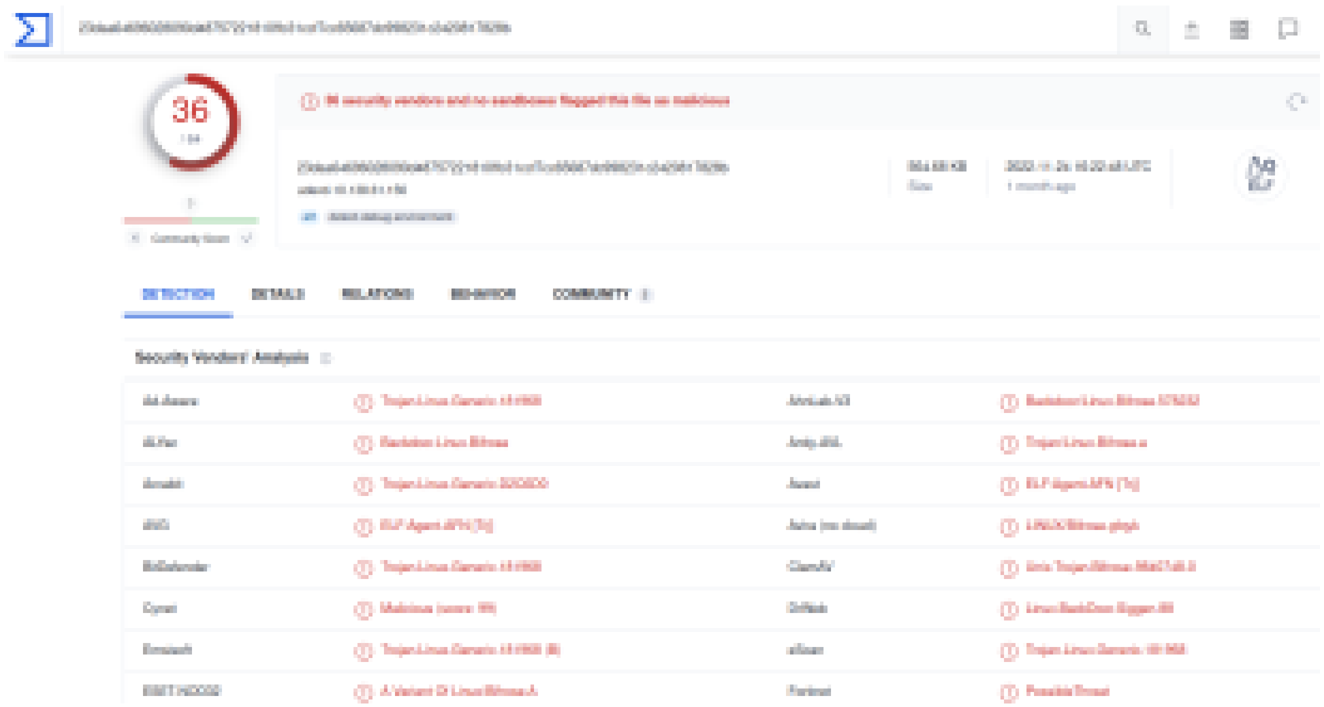
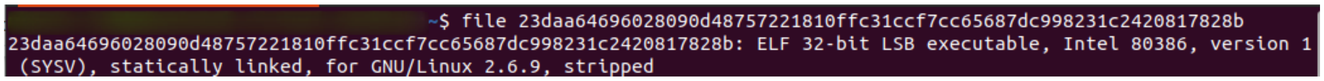


Figure2: VirusTotal Results

Although we have a good idea the file in question is an ELF file, running the *file command* will provide us with confirmation of the file type as well as if the file has been stripped.



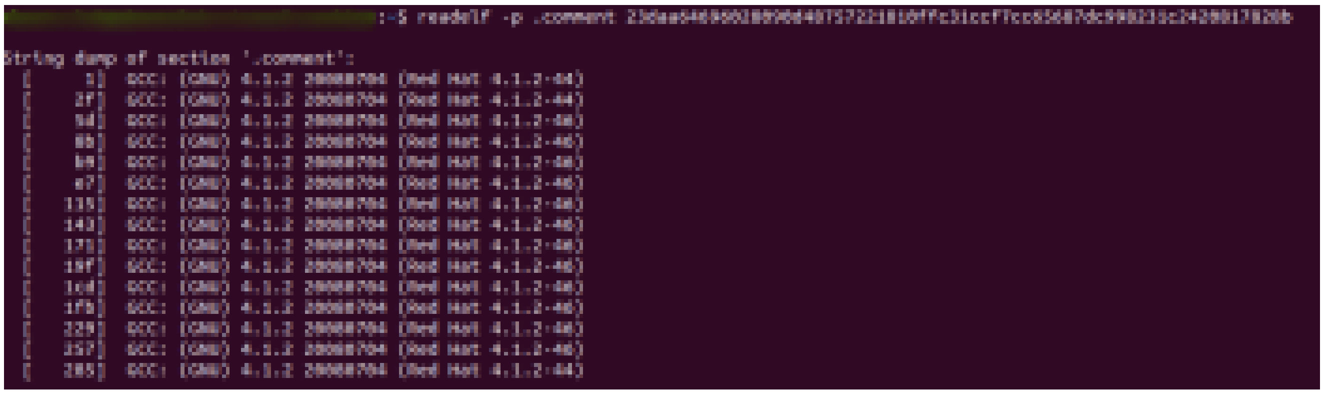
```
[redacted]~$ file 23daa64696028090d48757221810ffc31ccf7cc65687dc998231c2420817828b
23daa64696028090d48757221810ffc31ccf7cc65687dc998231c2420817828b: ELF 32-bit LSB executable, Intel 80386, version 1
 (SYSV), statically linked, for GNU/Linux 2.6.9, stripped
```

Figure 3: Output of file command

Sure enough, the output identifies the executable has been stripped; in other words, the symbols containing human-readable function names have been removed to slow down analysis.

If you’re still unsure the file is stripped, try running *readelf -s filename*. In this case, no output confirms the file’s symbols have been tampered with.

Running readelf with the “-p” argument on the .comment section will provide the compiler version and development environment.



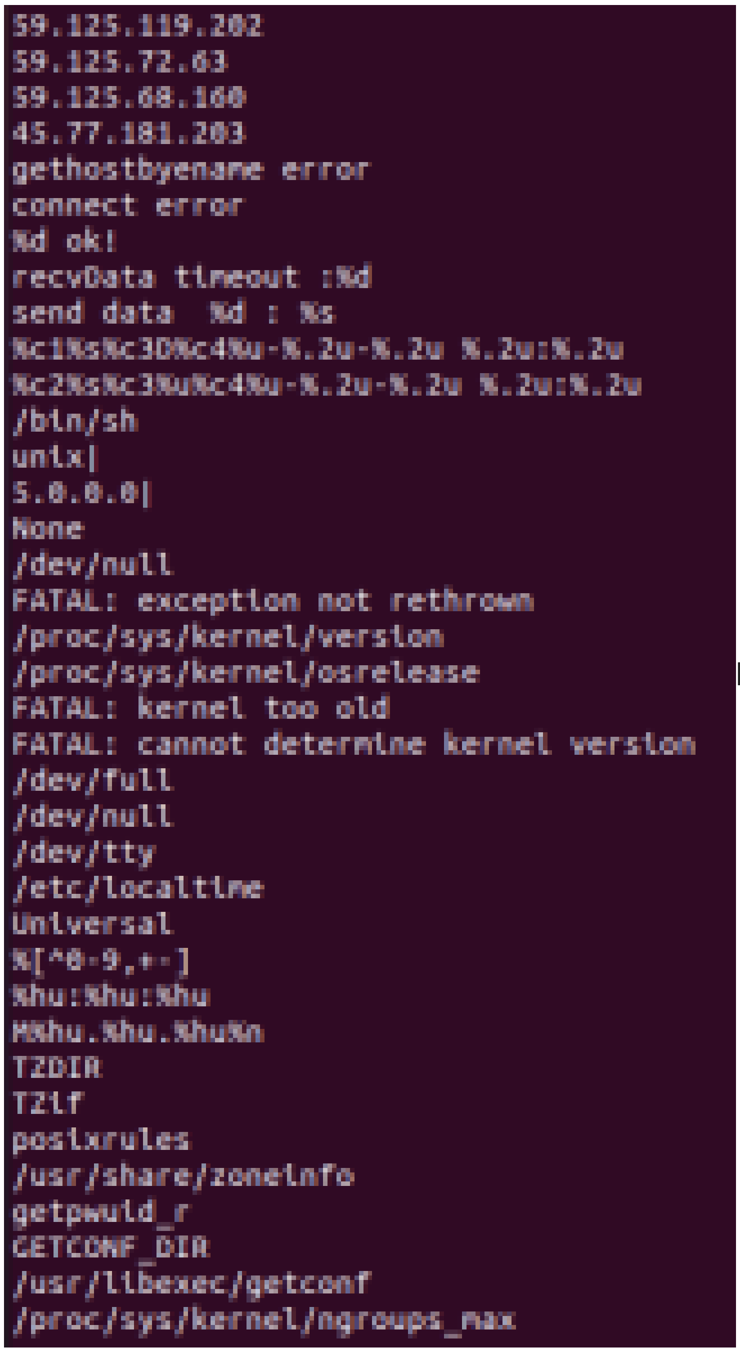
```
[redacted]:~$ readelf -p .comment [illegible]

String dump of section '.comment':
  [     1]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
  [    2f]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
  [    5d]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [    8b]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [    b9]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [    e7]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   115]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   143]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   171]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   19f]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   1cd]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   1fb]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   229]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   257]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
  [   285]  GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
```

Figure 4: Output of *readelf -p .comment filename*

From the above output, we can infer that this sample is likely targeting Red Hat distributions.

Probably one of the best analysis tools, the strings command can assist in identifying the functionality of the executable, as well as indicators (think Windows APIs for PEs, & syscalls for Unix). The output in Figure 5 provides a small snapshot of running strings.



```
59.125.119.202
59.125.72.63
59.125.68.160
45.77.181.203
gethostbyename error
connect error
%d ok!
recvData timeout :%d
send data  %d : %s
%c1%s%c3D%c4%u-%.2u-%.2u %.2u:%.2u
%c2%s%c3%u%c4%u-%.2u-%.2u %.2u:%.2u
/bin/sh
unix|
5.0.0.0|
None
/dev/null
FATAL: exception not rethrown
/proc/sys/kernel/version
/proc/sys/kernel/osrelease
FATAL: kernel too old
FATAL: cannot determine kernel version
/dev/full
/dev/null
/dev/tty
/etc/localtime
Universal
%[^0-9,+-]
%hu:%hu:%hu
M%hu.%hu.%hu%n
TZDIR
TZif
posixrules
/usr/share/zoneinfo
getpwuid_r
GETCONF_DIR
/usr/libexec/getconf
/proc/sys/kernel/ngroups_max
```

Figure 5: Output of strings

In addition to the hard-coded IP addresses, standard strings indicating first contact with the C&C server, notably *unix*|, *5.0.0.0*|, and what appear to be C&C commands (recvData and send data), are visible in the output.

Additionally, we can see signs of reconnaissance of the infected system, viewing the version and OS release, as well as the kernel version, and the timezone the target is located in.

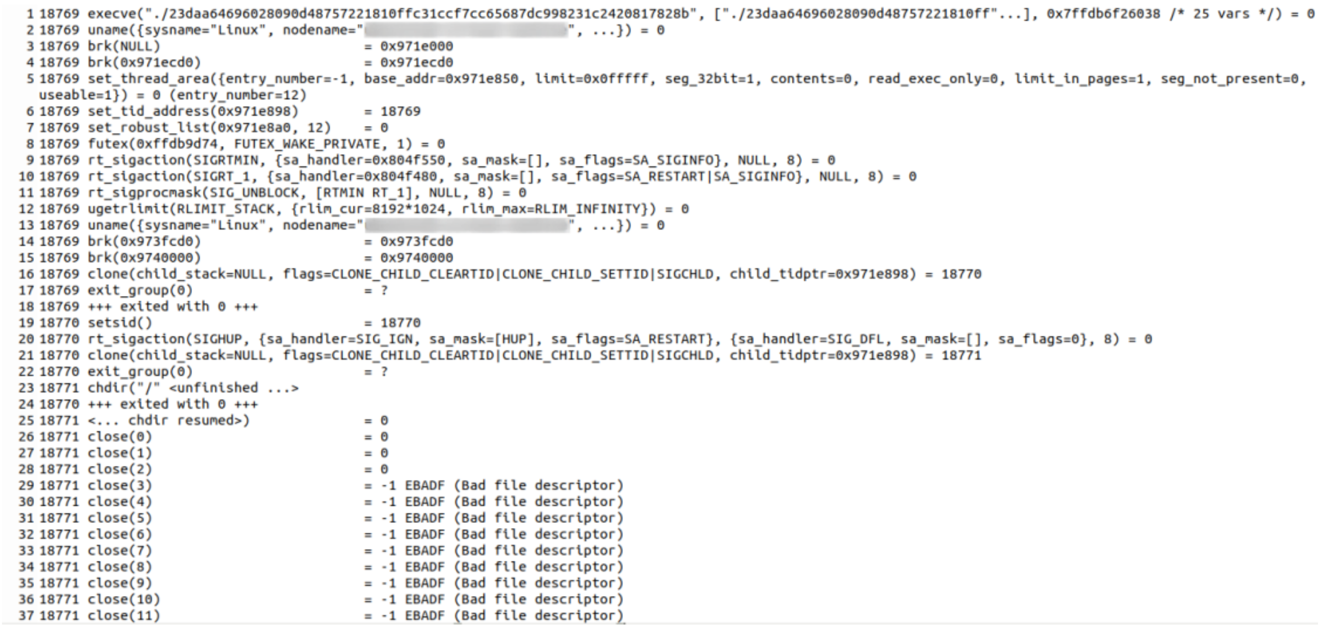
Bifrose Capabilities

If you don't have Sysmon for Linux setup in a VM, or aren't quite ready to upload the sample to a public sandbox, one great option is to utilize strace to run the sample and redirect the output to a separate file.

strace output will include operations such as any network connections or attempts, system calls, file read and write operations, etc., all information that is extremely valuable to understand the program's behavior.

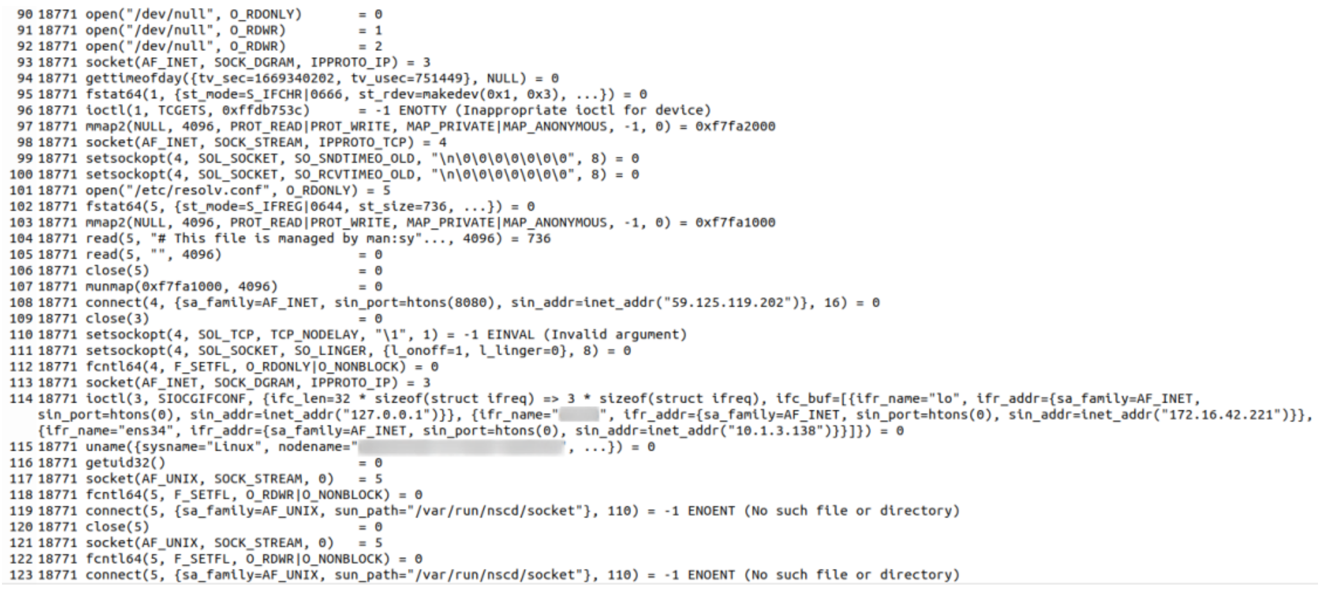
The command *strace -o strace_results.txt ./elf_file* is all you need, along with Wireshark, TCPDump, or any other tool that can capture network traffic. Explaining the syscalls identified in the strace output would be an article or two, and I would like to keep this short. If your interested in strace, see the below links section.

Figure 6 and 7 show snippets of interesting system calls Bifrose makes when run.



```
18769 execve("./23daa64696028090d48757221810ffc31ccf7cc65687dc998231c2420817828b", ["./23daa64696028090d48757221810ff"...], 0x7ffdb6f26038 /* 25 vars */) = 0
18769 uname({sysname="Linux", nodename="[redacted]", ...}) = 0
18769 brk(NULL)                         = 0x971e000
18769 brk(0x971ecd0)                    = 0x971ecd0
18769 set_thread_area({entry_number=-1, base_addr=0x971e850, limit=0x0fffff, seg_32bit=1, contents=0, read_exec_only=0, limit_in_pages=1, seg_not_present=0,
 useable=1}) = 0 (entry_number=12)
18769 set_tid_address(0x971e898)        = 18769
18769 set_robust_list(0x971e8a0, 12)    = 0
18769 futex(0xffdb9d74, FUTEX_WAKE_PRIVATE, 1) = 0
18769 rt_sigaction(SIGRTMIN, {sa_handler=0x804f550, sa_mask=[], sa_flags=SA_SIGINFO}, NULL, 8) = 0
18769 rt_sigaction(SIGRT_1, {sa_handler=0x804f480, sa_mask=[], sa_flags=SA_RESTART|SA_SIGINFO}, NULL, 8) = 0
18769 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
18769 ugetrlimit(RLIMIT_STACK, {rlim_cur=8192*1024, rlim_max=RLIM_INFINITY}) = 0
18769 uname({sysname="Linux", nodename="[redacted]", ...}) = 0
18769 brk(0x973fcd0)                    = 0x973fcd0
18769 brk(0x9740000)                    = 0x9740000
18769 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x971e898) = 18770
18769 exit_group(0)                     = ?
18769 +++ exited with 0 +++
18770 setsid()                          = 18770
18770 rt_sigaction(SIGHUP, {sa_handler=SIG_IGN, sa_mask=[HUP], sa_flags=SA_RESTART}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
18770 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x971e898) = 18771
18770 exit_group(0)                     = ?
18771 chdir("/" <unfinished ...>
18770 +++ exited with 0 +++
18771 <... chdir resumed>)              = 0
18771 close(0)                          = 0
18771 close(1)                          = 0
18771 close(2)                          = 0
18771 close(3)                          = -1 EBADF (Bad file descriptor)
18771 close(4)                          = -1 EBADF (Bad file descriptor)
18771 close(5)                          = -1 EBADF (Bad file descriptor)
18771 close(6)                          = -1 EBADF (Bad file descriptor)
18771 close(7)                          = -1 EBADF (Bad file descriptor)
18771 close(8)                          = -1 EBADF (Bad file descriptor)
18771 close(9)                          = -1 EBADF (Bad file descriptor)
18771 close(10)                         = -1 EBADF (Bad file descriptor)
18771 close(11)                         = -1 EBADF (Bad file descriptor)
```

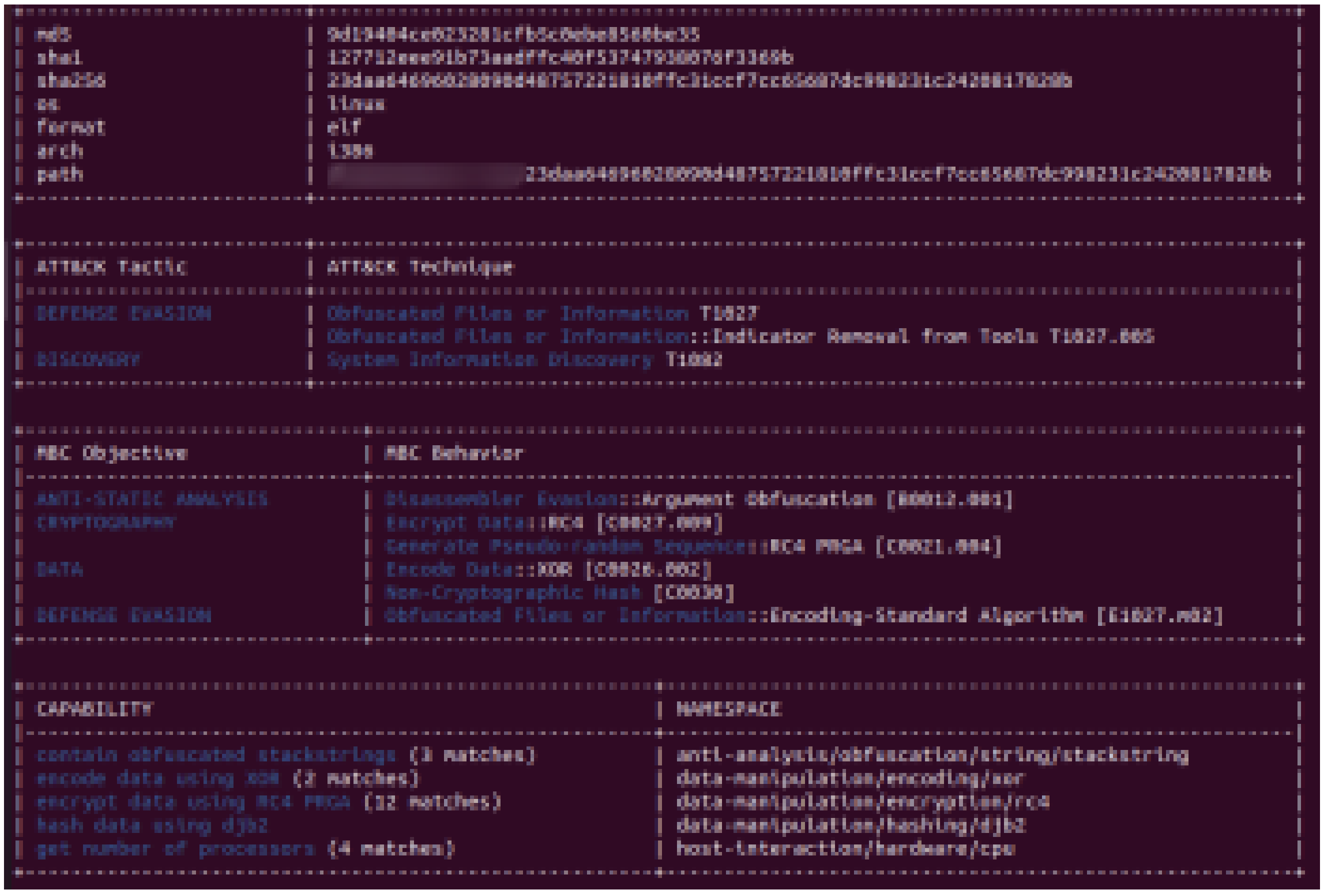
Figure 6: strace output (1)



```
18771 open("/dev/null", O_RDONLY)       = 0
18771 open("/dev/null", O_RDWR)         = 1
18771 open("/dev/null", O_RDWR)         = 2
18771 socket(AF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
18771 gettimeofday({tv_sec=1669340202, tv_usec=751449}, NULL) = 0
18771 fstat64(1, {st_mode=S_IFCHR|0666, st_rdev=makedev(0x1, 0x3), ...}) = 0
18771 ioctl(1, TCGETS, 0xffdb753c)      = -1 ENOTTY (Inappropriate ioctl for device)
18771 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xf7fa2000
18771 socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 4
18771 setsockopt(4, SOL_SOCKET, SO_SNDTIMEO_OLD, "\n\0\0\0\0\0\0\0", 8) = 0
18771 setsockopt(4, SOL_SOCKET, SO_RCVTIMEO_OLD, "\n\0\0\0\0\0\0\0", 8) = 0
18771 open("/etc/resolv.conf", O_RDONLY) = 5
18771 fstat64(5, {st_mode=S_IFREG|0644, st_size=736, ...}) = 0
18771 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xf7fa1000
18771 read(5, "# This file is managed by man:sy"..., 4096) = 736
18771 read(5, "", 4096)                 = 0
18771 close(5)                          = 0
18771 munmap(0xf7fa1000, 4096)          = 0
18771 connect(4, {sa_family=AF_INET, sin_port=htons(8080), sin_addr=inet_addr("59.125.119.202")}, 16) = 0
18771 close(3)                          = 0
18771 setsockopt(4, SOL_TCP, TCP_NODELAY, "\1", 1) = -1 EINVAL (Invalid argument)
18771 setsockopt(4, SOL_SOCKET, SO_LINGER, {l_onoff=1, l_linger=0}, 8) = 0
18771 fcntl64(4, F_SETFL, O_RDONLY|O_NONBLOCK) = 0
18771 socket(AF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
18771 ioctl(3, SIOCGIFCONF, {ifc_len=32 * sizeof(struct ifreq) => 3 * sizeof(struct ifreq), ifc_buf=[{ifr_name="lo", ifr_addr={sa_family=AF_INET,
 sin_port=htons(0), sin_addr=inet_addr("127.0.0.1")}}, {ifr_name="[redacted]", ifr_addr={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("172.16.42.221")}},
 {ifr_name="ens34", ifr_addr={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("10.1.3.138")}}]}) = 0
18771 uname({sysname="Linux", nodename="[redacted]", ...}) = 0
18771 getuid32()                        = 0
18771 socket(AF_UNIX, SOCK_STREAM, 0)   = 5
18771 fcntl64(5, F_SETFL, O_RDWR|O_NONBLOCK) = 0
18771 connect(5, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory)
18771 close(5)                          = 0
18771 socket(AF_UNIX, SOCK_STREAM, 0)   = 5
18771 fcntl64(5, F_SETFL, O_RDWR|O_NONBLOCK) = 0
18771 connect(5, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory)
```

strace output (2)

To keep things simple, we'll use Mandiant's CAPA tool to get an idea of what Bifrose is up to.



```
+------------------------+------------------------------------------------------------------------------------+
| md5                    | 9d19484ce023281cfb5c0ebe8560be35                                                   |
| sha1                   | 127712eee91b73aadffc40f53747930076f3369b                                           |
| sha256                 | 23daa44696028009d48757221810ffc31ccf7cc65687dc998231c2420817828b                   |
| os                     | linux                                                                              |
| format                 | elf                                                                                |
| arch                   | i386                                                                               |
| path                   | [redacted]23daa44696028009d48757221810ffc31ccf7cc65687dc998231c2420817828b         |
+------------------------+------------------------------------------------------------------------------------+

+------------------------+------------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                   |
|------------------------+------------------------------------------------------------------------------------|
| DEFENSE EVASION        | Obfuscated Files or Information T1027                                              |
|                        | Obfuscated Files or Information::Indicator Removal from Tools T1027.005            |
| DISCOVERY              | System Information Discovery T1082                                                 |
+------------------------+------------------------------------------------------------------------------------+

+-----------------------------+-------------------------------------------------------------------------------+
| MBC Objective               | MBC Behavior                                                                  |
|-----------------------------+-------------------------------------------------------------------------------|
| ANTI-STATIC ANALYSIS        | Disassembler Evasion::Argument Obfuscation [B0012.001]                        |
| CRYPTOGRAPHY                | Encrypt Data::RC4 [C0027.009]                                                 |
|                             | Generate Pseudo-random Sequence::RC4 PRGA [C0021.004]                         |
| DATA                        | Encode Data::XOR [C0026.002]                                                  |
|                             | Non-Cryptographic Hash [C0030]                                                |
| DEFENSE EVASION             | Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]      |
+-----------------------------+-------------------------------------------------------------------------------+

+--------------------------------------------------------+----------------------------------------------------+
| CAPABILITY                                             | NAMESPACE                                          |
|--------------------------------------------------------+----------------------------------------------------|
| contain obfuscated stackstrings (3 matches)            | anti-analysis/obfuscation/string/stackstring       |
| encode data using XOR (2 matches)                      | data-manipulation/encoding/xor                     |
| encrypt data using RC4 PRGA (12 matches)               | data-manipulation/encryption/rc4                   |
| hash data using djb2                                   | data-manipulation/hashing/djb2                     |
| get number of processors (4 matches)                   | host-interaction/hardware/cpu                      |
+--------------------------------------------------------+----------------------------------------------------+
```

Figure 8: CAPA output

In the next post, I'll use Cutter to look at some of the capabilities identified in the above image and see if we can map out the execution of Bifrose, to help defenders get an idea of what indicators will assist in identifying a possible intrusion.

Links

https://man7.org/linux/man-pages/man2/syscalls.2.html

> ELF Malware Analysis 101: Part 3 – Advanced Analysis

https://www.pentesteracademy.com/video?id=881