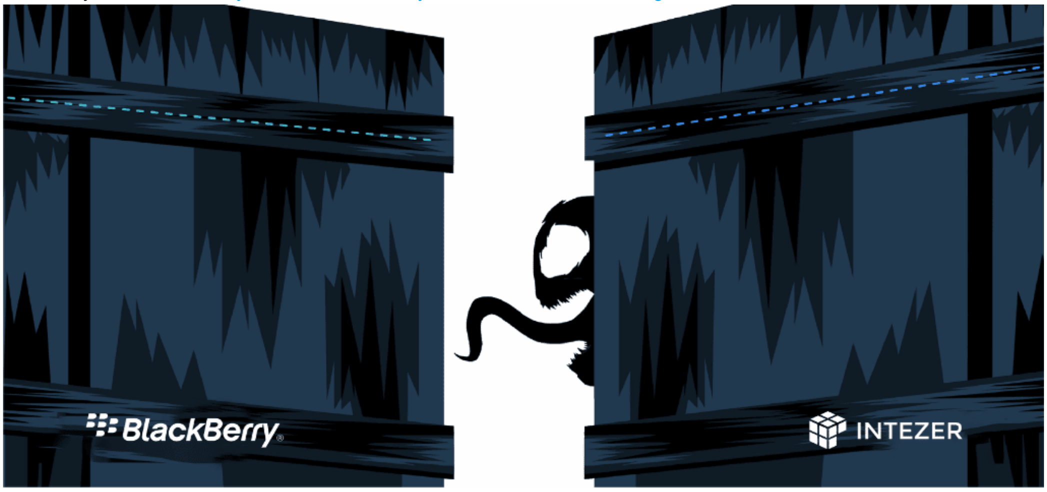
Symbiote Deep-Dive: Analysis of a New, Nearly-Impossible-to-Detect Linux Threat

6/9/2022

Written by Joakim Kennedy and The BlackBerry Threat Research & Intelligence Team - 9 June 2022



This research is a joint effort between Joakim Kennedy, Security Researcher at Intezer, and the BlackBerry Threat Research & Intelligence team. It can be found in the BlackBerry blog here as well.

In biology, a *symbiote* is an organism that lives in symbiosis with another organism. The symbiosis can be mutually beneficial to both organisms, but sometimes it can be parasitic when one benefits and the other is harmed. A few months back, we discovered a new, undetected Linux® malware that acts in this parasitic nature. We have aptly named this malware Symbiote.

What makes Symbiote different from other Linux malware that we usually come across, is that it needs to infect other running processes to inflict damage on infected machines. Instead of being a standalone executable file that is run to infect a machine, it is a shared object (SO) library that is loaded into all running processes using **LD_PRELOAD** (T1574.006), and parasitically infects the machine. Once it has infected all the running processes, it provides the threat actor with rootkit functionality, the ability to harvest credentials, and remote access capability.

The Birth of a Symbiote

Our earliest detection of Symbiote is from November 2021, and it appears to have been written to target the **financial sector** in **Latin America**. Once the malware has infected a machine, it hides itself and any other malware used by the threat actor, making infections very hard to detect. Performing live forensics on an infected machine may not turn anything up since all the file, processes, and network artifacts are hidden by the malware. In addition to the rootkit capability, the malware provides a backdoor for the threat actor to log in as any user on the machine with a hardcoded password and to execute commands with the highest privileges.

Since it is extremely evasive, a Symbiote infection is likely to “fly under the radar.” In our research, we haven’t found enough evidence to determine whether Symbiote is being used in highly targeted or broad attacks.

One interesting technical aspect of Symbiote is its Berkeley Packet Filter (BPF) hooking functionality. Symbiote is not the first Linux malware to use BPF. For example, advanced backdoors attributed to the Equation Group have been using BPF for covert communication. However, Symbiote utilizes BPF to hide malicious network traffic on an infected machine. When an administrator starts any packet capture tool on the infected machine, BPF bytecode is injected into the kernel that defines which packets should be captured. In this process, Symbiote adds its bytecode first so it can filter out network traffic that it doesn’t want the packet-capturing software to see.

Evasion Techniques

Symbiote is very stealthy. The malware is designed to be loaded by the linker via the **LD_PRELOAD** directive. This allows it to be loaded before any other shared objects. Since it is loaded first, it can “hijack the imports” from the other library files loaded for the application. Symbiote uses this to hide its presence on the machine by hooking **libc** and **libpcap** functions. The image below shows a summary of the malware’s evasions.

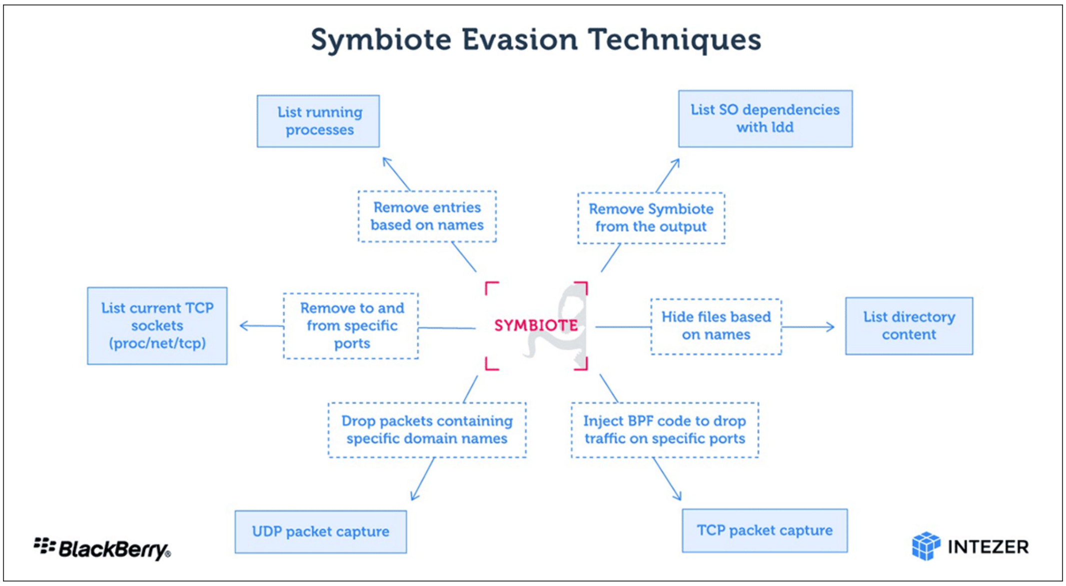


Figure 1: Symbiote evasion techniques.

Host Activity

The Symbiote malware, in addition to hiding its own presence on the machine, also hides other files related to malware likely deployed with it. Within the binary, there is a file list that is RC4 encrypted. When hooked functions are called, the malware first dynamically loads libc and calls the original function. This logic is used in all hooked functions. An example is shown in Figure 2 below.

```
sym.readdir (int64_t arg1);
     ; var int64_t var_18h @ rbp-0x18
     ; var int64_t var_10h @ rbp-0x10
     ; var uint32_t var_8h @ rbp-0x8
     ; arg int64_t arg1 @ rdi
     0x0000d99c      55              push rbp
     0x0000d99d      4889e5          mov rbp, rsp
     0x0000d9a0      4883ec20        sub rsp, 0x20
     0x0000d9a4      48897de8        mov qword [var_18h], rdi     ; arg1
     0x0000d9a8      488b05814120.   mov rax, qword [obj.orig_readdir.10823]    ; [0x211b30:8]=
     0x0000d9af      4885c0          test rax, rax
  ┌─< 0x0000d9b2      7539            jne 0xd9ed
  │  0x0000d9b4      488b05852b00.   mov rax, qword [0x00010540]    ; [0x10540:8]=0x9545f1519bb
  │  0x0000d9bb      488945f0        mov qword [var_10h], rax
  │  0x0000d9bf      488d45f0        lea rax, [var_10h]
  │  0x0000d9c3      ba07000000      mov edx, 7                   ; int64_t arg3
  │  0x0000d9c8      4889c6          mov rsi, rax                 ; int64_t arg2
  │  0x0000d9cb      488d3d1c2300.   lea rdi, [0x0000fcee]        ; int64_t arg1
  │  0x0000d9d2      e84547ffff      call sym.rc4                 ;[2]
  │  0x0000d9d7      4889c6          mov rsi, rax
  │  0x0000d9da      48c7c7ffffff.   mov rdi, 0xffffffffffffffff
  │  0x0000d9e1      e87245ffff      call sym.imp.dlsym           ;[3]
  │  0x0000d9e6      488905434120.   mov qword [obj.orig_readdir.10823], rax    ; [0x211b30:8]=
  │  ; CODE XREF from sym.readdir @ 0xd9b2
  └─> 0x0000d9ed      48c745f80000.   mov qword [var_8h], 0
```

Figure 2: Logic for resolving readdir from libc.

If the calling application is trying to access a file or folder under **/proc**, the malware scrubs the output from process names that are on its list. The process names in the list below were extracted from the samples we have discovered.

- certbotx64
- certbotx86
- javautils
- javaserverx64
- javaclientex64
- javanodex86

If the calling application is not trying to access something under **/proc**, the malware instead scrubs the result from a file list. The files extracted from all the samples we examined are shown in the list below. Some of the file names match those used by Symbiote, while others match names of files suspected to be tools used by the threat actor on the infected machine. The list includes the following files.

- apache2start
- apache2stop
- profiles.php
- 404erro.php
- javaserverx64
- javaclientex64

- javanodex86
- liblinux.so
- java.h
- open.h
- mpt86.h
- sqlsearch.php
- indexq.php
- mt64.so
- certbot.h
- cert.h
- certbotx64
- certbotx86
- javautils
- search.so

One consequence of Symbiote being loaded into processes via **LD_PRELOAD** is that tools like **ldd**, a utility that prints the shared libraries required by each program, will list the malware as a loaded object. To counter this, the malware hooks **execve** and looks for calls to this function with the environment variable **LD_TRACE_LOADED_OBJECTS** set to 1. To understand why, it's worth looking at the manual page for **ldd**:

In the usual case, ldd invokes the standard dynamic linker (see ld.so(8)) with the LD_TRACE_LOADED_OBJECTS environment variable set to 1. This causes the dynamic linker to inspect the program's dynamic dependencies, and find (according to the rules described in ld.so(8)) and load the objects that satisfy those dependencies. For each dependency, ldd displays the location of the matching object and the (hexadecimal) address at which it is loaded. (The linux-vdso and ld-linux shared dependencies are special; see vdso(7) and ld.so(8).)

When the malware detects this, it executes the loader as **ldd** does, but it scrubs its own entry from the result.

Network Activity

Symbiote also has functionality to hide network activity on the infected machine. It uses three different methods to accomplish this. The first method involves hooking **fopen** and **fopen64**. If the calling application tries to open **/proc/net/tcp,** the malware creates a temp file and copies the first line to that file. After that, it scans each line for the presence of specific ports. If the malware finds a port it's searching for on a line it's scanning, it skips to the next line. Otherwise, the line is written to the temp file. Once the original file has been completely processed, the malware closes the file and returns the file descriptor of the temp file back to the caller. Essentially, this gives the calling process a scrubbed result, which excludes all entries of the network connections that the malware wants to hide.

The second method Symbiote uses to hide its network activity is by hijacking any injected packet filtering bytecode. The Linux kernel uses extended Berkeley Packet Filter (eBPF) to allow packet filtering based on rules provided from a userland process. The filtering rule is provided as eBPF bytecode that the kernel executes on a virtual machine (VM). This minimizes the context switching between kernel and userland, providing a performance boost since the kernel performs the filtering directly.

If an application on the infected machine tries to perform packet filtering with eBPF, Symbiote hijacks the filtering process. First, it hooks the **libc** function **setsockopt**. If the function is called with the option **SO_ATTACH_FILTER**, which is used to perform packet filtering on a socket, it prepends its own bytecode before the eBPF code provided by the calling application.

Code Snippet 1 shows an annotated version of the bytecode injected by one of the Symbiote samples. The bytecode "drops" if they match the following conditions:

- IPv6 (TCP or SCTP) and src port (43253 or 43753 or 63424 or 26424)
- IPv6 (TCP or SCTP) and dst port 43253
- IPv4 (TCP or SCTP) and src port (43253 or 43753 or 63424 or 26424)
- IPv4 (TCP or SCTP) and dst port (43253 or 43753 or 63424 or 26424)

While this bytecode only drops packets based on ports, we have also observed filtering of traffic based on IPv4 addresses. In all cases, the filtering operates on both inbound and outbound traffic from the machine, to hide both directions of the traffic. If the conditions are not met, it just jumps to the start of the bytecode provided by the calling application.

The bytecode extracted from one of the samples, as shown in Code Snippet 1, consists of 32 instructions. This code can't be injected into the kernel on its own, because it assumes that more bytecode exists after it. There are a few jumps in this bytecode that skip to the beginning of the bytecode provided by the calling process. Without the caller's bytecode, the injected bytecode would jump out-of-bounds, which is not allowed by the kernel. Bytecode like this either has to be handwritten or by patching compiler generated-bytecode. Either option suggests that this malware was written by a skilled developer.

```
; Load Ether frame type from the packet.
0x00:    0x28    0x00    0x00    0x000c         ldabsh    0xc
```

```
; Jump if it's not IPv6 (0x86DD)
0x01:    0x15    0x00    0x0b    0x86dd          jeq    r0,    0x86dd,    +0, +0x0b
(jump to 0xd)
; Load IPv6 next header into register.
0x02:    0x30    0x00    0x00    0x0014          ldabsb    0x14
; Short jump if SCTP
0x03:    0x15    0x02    0x00    0x0084          jeq    r0,    0x84,    +0x2
(jump to 0x6)              ; SCTP
; Short jump if TCP
0x04:    0x15    0x01    0x00    0x0006          jeq    r0,    0x6,    +0x1    (jump
to 0x6)                  ; TCP
; Jump to original byte code if UDP
0x05:    0x15    0x00    0x1a    0x0011          jeq    r0,    0x11,    +0x1a
(jump to 0x20)           ; UDP

; Load TCP src port into register.
0x06:    0x28    0x00    0x00    0x0036          ldabsh    0x36
; Jump to drop the packet if port 43253.
0x07:    0x15    0x17    0x00    0xa8f5          jeq    r0,    0xa8f5,    +0x17
(jump to 0x1f)    ; 43253
; Jump to drop the packet if port 43753.
0x08:    0x15    0x16    0x00    0xaae9          jeq    r0,    0xaae9,    +0x16
(jump to 0x1f)    ; 43753
; Jump to drop the packet if port 63424.
0x09:    0x15    0x15    0x00    0xf7c0          jeq    r0,    0xf7c0,    +0x15
(jump to 0x1f)    ; 63424
; Jump to drop the packet if port 26424.
0x0a:    0x15    0x14    0x00    0x6738          jeq    r0,    0x6738,    +0x14
(jump to 0x1f)    ; 26424

; Load TCP dst port into register.
0x0b:    0x28    0x00    0x00    0x0038          ldabsh    0x38
; Jump to drop packet if port 43253 else jump to 0x1c.
0x0c:    0x15    0x12    0x0f    0xa8f5          jeq    r0,    0xa8f5,    +0xf12
(jump to 0x1f)    (jump to 0x1c) ; 43253

; Ether frame type check for IPv4 (0x0800)
0x0d:    0x15    0x00    0x12    0x0800          jeq    r0,    0x800,    +0x1200
(jump to 0x20)
; Load IPv4 next header field into register.
0x0e:    0x30    0x00    0x00    0x0017          ldabsb    0x17
; Short jump if SCTP.
0x0f:    0x15    0x02    0x00    0x0084          jeq    r0,    0x84,    +0x2
(jump to 0x12)           ; SCTP
; Short jump if TCP.
0x10:    0x15    0x01    0x00    0x0006          jeq    r0,    0x6,    +0x1    (jump
to 0x12)         ; TCP
; Jump to original byte code if UDP.
0x11:    0x15    0x00    0x0e    0x0011          jeq    r0,    0x11,    +0xe00
(jump to 0x20)     ; UDP

; Load IPv4 flag into register.
0x12:    0x28    0x00    0x00    0x0014          ldabsh    0x14
; Jump to original byte code if flags are set.
0x13:    0x45    0x0c    0x00    0x1fff          jset    r0,    0x1fff,    +0xc
(jump to 0x20)

; Load Internet Header Length into x.
0x14:    0xb1    0x00    0x00    0x000e          ldxmsh 0x0e
; Load TCP src port into register.
0x15:    0x48    0x00    0x00    0x000e          ldindh    r0,    0xe
; Jump to drop the packet if port 43253.
0x16:    0x15    0x08    0x00    0xa8f5          jeq    r0,    0xa8f5,    +0x8
(jump to 0x1f) ; 43253
; Jump to drop the packet if port 43753.
0x17:    0x15    0x07    0x00    0xaae9          jeq    r0,    0xaae9,    +0x7
(jump to 0x1f)    ; 43753
; Jump to drop the packet if port 63424.
```

```
0x18:    0x15    0x06    0x00    0xf7c0          jeq    r0,    0xf7c0,    +0x6
(jump to 0x1f)    ; 63424
; Jump to drop the packet if port 26424.
0x19:    0x15    0x05    0x00    0x6738          jeq    r0,    0x6738,    +0x5
(jump to 0x1f)    ; 26424

; Load TCP dst port into register.
0x1a:    0x48    0x00    0x00    0x0010          ldindh    r0,    0x10
; Jump to drop the packet if port 43253.
0x1b:    0x15    0x03    0x00    0xa8f5          jeq    r0,    0xa8f5,    +0x3
(jump to 0x1f) ; 43253
; Jump to drop the packet if port 43753.
0x1c:    0x15    0x02    0x00    0xaae9          jeq    r0,    0xaae9,    +0x2
(jump to 0x1f) ; 43753
; Jump to drop the packet if port 63424.
0x1d:    0x15    0x01    0x00    0xf7c0          jeq    r0,    0xf7c0,    +0x1
(jump to 0x1f)    ; 63424

; Jump to drop packet if true otherwise jump to original byte code.
0x1e:    0x15    0x00    0x01    0x6738          jeq    r0,    0x6738,    +0x100
(jump to 0x20); 26424
; Drop packet by returning 0.
0x1f:    0x06    0x00    0x00    0x0000          ret 0
0x20: // Original byte code.
```

Code Snippet 1: Annotated bytecode extracted from one of the Symbiote samples.

The third method Symbiote uses to hide its network traffic is to hook **libpcap** functions. This method is used by the malware to filter out UDP traffic to domain names it has in a list. It hooks the functions **pcap_loop** and **pcap_stats** to accomplish this task. For each packet that is received, Symbiote checks the UDP payload for substrings of the domains it wants to filter out. If it finds a match, the malware ignores the packet and increments a counter. The **pcap_stats** uses this counter to “correct” the number of packets processed by subtracting the counter value from the true number of packets processed. If a packet payload does not contain any of the strings it has in its list, the original callback function is called. This method is used to filter out UDP packets, while the bytecode method is used to filter out TCP packets. By using all three of these methods, the malware ensures that all traffic is hidden.

Symbiote Objectives

The malware’s objective, in addition to hiding malicious activity on the machine, is to harvest credentials and to provide remote access for the threat actor. The credential harvesting is performed by hooking the **libc read** function. If an **ssh** or **scp** process is calling the function, it captures the credentials. The credentials are first encrypted with RC4 using an embedded key, and then written to a file. For example, one of the versions of the malware writes the captured credentials to the file **/usr/include/certbot.h**.

In addition to storing the credentials locally, the credentials are exfiltrated. The data is hex encoded and chunked up to be exfiltrated via DNS address (A) record requests to a domain name controlled by the threat actor. The A record request has the following format:

```
%PACKET_NUMBER%.%MACHINE_ID%.%HEX_ENC_PAYLOAD%.%DOMAIN_NAME%
```

Code Snippet 2: Structure of DNS request used by Symbiote to exfiltrate data.

The malware checks if the machine has a nameserver configured in **/etc/resolv.conf**. If it doesn’t, Google’s DNS (8.8.8.8) is used. Along with sending the request to the domain name, Symbiote also sends it as a UDP broadcast.

Remote access to the infected machine is achieved by hooking a few Linux Pluggable Authentication Module (PAM) functions. When a service tries to use PAM to authenticate a user, the malware checks the provided password against a hardcoded password. If the password provided is a match, the hooked function returns a success response. Since the hooks are in PAM, it allows the threat actor to authenticate to the machine with any service that uses PAM. This includes remote services such as Secure Shell (SSH).

If the entered password does not match the hardcoded password, the malware saves and exfiltrates it as part of its keylogging functionality. Additionally, the malware sends a DNS TXT record request to its command-and-control (C2) domain. The TXT record has the format of **%MACHINEID%.%C2_DOMAIN%**. If it gets a response, the malware base64 decodes the content, checks if the content has been signed by a correct ed25519 private key, decrypts the content with RC4, and executes the shell script in a spawned bash process. This functionality can operate as a break-glass method for regaining access to the machine in case the normal process doesn’t work.

Once the threat actor has authenticated to the infected machine, Symbiote provides functionality to gain root privileges. When the shared object is first loaded, it checks for the environment variable **HTTP_SETTHIS**. If the variable is set with content, the malware changes the effective user and group ID to the root user, and then clears the variable before executing the content via the system command.

This process requires that the SO has the setuid permission flag set. Once the system command has exited, Symbiote also exits the process, to prevent the original process from executing. Figure 3 below shows the code executed. This allows for spawning a root shell by running **HTTP_SETTHIS="/bin/bash -p" /bin/true** as any user in a shell.

```
0x0000236f      55              push rbp
0x00002370      4889e5          mov rbp, rsp
0x00002373      4883ec30        sub rsp, 0x30
0x00002377      c745d01f82ae.   mov dword [var_30h], 0x65ae821f
0x0000237e      c745d4ca7fa2.   mov dword [var_2ch], 0xe2a27fca    ; HTTP_SETTHIS
0x00002385      c745d891076f.   mov dword [var_28h], 0xc06f0791
0x0000238c      c645dc00        mov byte [var_24h], 0
0x00002390      488d45d0        lea rax, [var_30h]
0x00002394      ba0c000000      mov edx, 0xc                  ; int64_t arg3
0x00002399      4889c6          mov rsi, rax                  ; int64_t arg2
0x0000239c      488d3d4bd900.   lea rdi, sym.rc4_key          ; 0xfcee ; int64_t arg1
0x000023a3      e874fdffff      call sym.rc4                  ;[1]
0x000023a8      4889c7          mov rdi, rax                  ; const char *name
0x000023ab      e8e8faffff      call sym.imp.getenv           ;[2] ; char *getenv(const char *name)
0x000023b0      488945f8        mov qword [string], rax
0x000023b4      48837df800      cmp qword [string], 0
0x000023b9      746d            je 0x2428
0x000023bb      bf00000000      mov edi, 0
0x000023c0      e813fcffff      call sym.imp.setgid           ;[3]
0x000023c5      bf00000000      mov edi, 0
0x000023ca      e849f9ffff      call sym.imp.setuid           ;[4]
0x000023cf      bf0a000000      mov edi, 0xa                  ; int c
0x000023d4      e8bff8ffff      call sym.imp.putchar          ;[5] ; int putchar(int c)
0x000023d9      c745e01f82ae.   mov dword [var_20h], 0x65ae821f
0x000023e0      c745e4ca7fa2.   mov dword [var_1ch], 0xe2a27fca    ; HTTP_SETTHIS
0x000023e7      c745e891076f.   mov dword [var_18h], 0xc06f0791
0x000023ee      c645ec00        mov byte [var_14h], 0
0x000023f2      488d45e0        lea rax, [var_20h]
0x000023f6      ba0c000000      mov edx, 0xc                  ; int64_t arg3
0x000023fb      4889c6          mov rsi, rax                  ; int64_t arg2
0x000023fe      488d3de9d800.   lea rdi, sym.rc4_key          ; 0xfcee ; int64_t arg1
0x00002405      e812fdffff      call sym.rc4                  ;[1]
0x0000240a      4889c7          mov rdi, rax
0x0000240d      e856fbffff      call sym.imp.unsetenv         ;[6]
0x00002412      488b45f8        mov rax, qword [string]
0x00002416      4889c7          mov rdi, rax                  ; const char *string
0x00002419      e8aaf8ffff      call sym.imp.system           ;[7] ; int system(const char *string)
0x0000241e      bf00000000      mov edi, 0
0x00002423      e8c0f8ffff      call sym.imp._exit            ;[8]
; CODE XREF from sym.init_method @ 0x23b9
```

Figure 3: Logic used to execute a command with root privileges.

Network Infrastructure

The domain names used by the Symbiote malware are impersonating some major Brazilian banks. This suggests that these banks or their customers are the potential targets. Using the domain names utilized by the malware, we managed to uncover a related sample that was uploaded to VirusTotal with the name **certbotx64**. This file name matches one of those listed as a file to hide in one of the Symbiote samples we originally obtained. The file was identified as an open-source DNS tunneling tool called dnscat2.

The sample had a configuration in the binary that used the **git[.]bancodobrasil[.]dev** domain as its C2 server. During the months of February and March, this domain name resolved to an IP address that is linked to Njalla's Virtual Private Server (VPS) service. Passive DNS records showed that the same IP address was resolved to **ns1[.]cintepol[.]link** and **ns2[.]cintepol[.]link** a few months earlier. Cintepol is an intelligence portal provided by the Federal Police of Brazil. The portal allows police officers to access different databases provided by the federal police as part of their investigations. The nameserver used for this impersonating domain name was active from the middle of December 2021 to the end of January 2022.

Also starting in February of 2022, the name servers for the domain **caixa[.]wf** were pointing to another Njalla VPS IP. Figure 4 below shows a timeline of these events. In addition to the network infrastructure, the timestamps of when the files were submitted to VirusTotal are included. These three Symbiote samples were uploaded by the same submitter from Brazil. It appears that the files were submitted to VirusTotal before the infrastructure went online.

Given that these files were submitted to VirusTotal prior to the infrastructure going online, and because some of the samples included rules to hide local IP addresses, it is possible that the samples were submitted to VirusTotal to test Antivirus detection before being used. Additionally, a version that appears to be under development was submitted at the end of November from Brazil, further suggesting VirusTotal was being used by the threat actor or group behind Symbiote for detection testing.

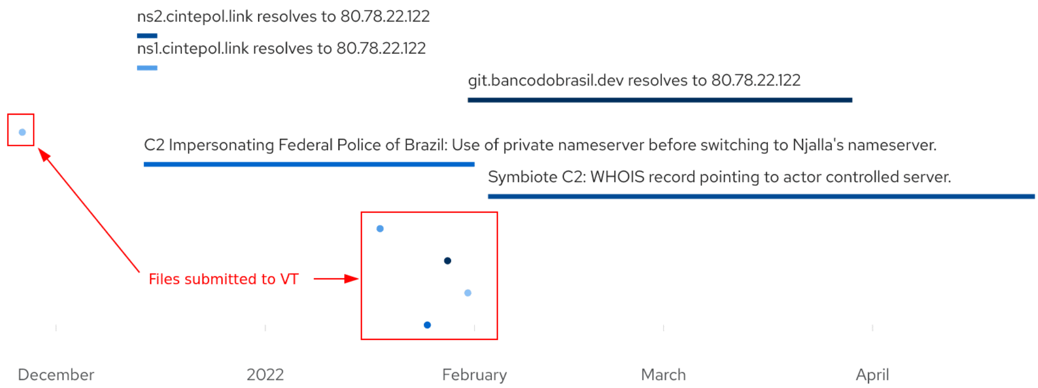


Figure 4: Timeline showing when files were submitted to VirusTotal and when network infrastructure went active.

Similarity to Other Malware

Symbiote appears to be designed for both credential stealing and to provide remote access to infected Linux servers. Symbiote is not the first Linux malware developed for this goal. In 2014, ESET released an in-depth analysis of Ebury, an OpenSSH backdoor that also performs credential stealing. There are some similarities in the techniques used by both malware families. Both use hooked functions to capture credentials and exfiltrate the captured data as DNS requests. However, the authentication method to the backdoor used by the two malware families is different. When we first analyzed the samples with Intezer Analyze, only unique code was detected (Figure 5). As no code is shared between Symbiote and Ebury/Windigo or any other known malware, we can confidently conclude that Symbiote is a new, undiscovered Linux malware.

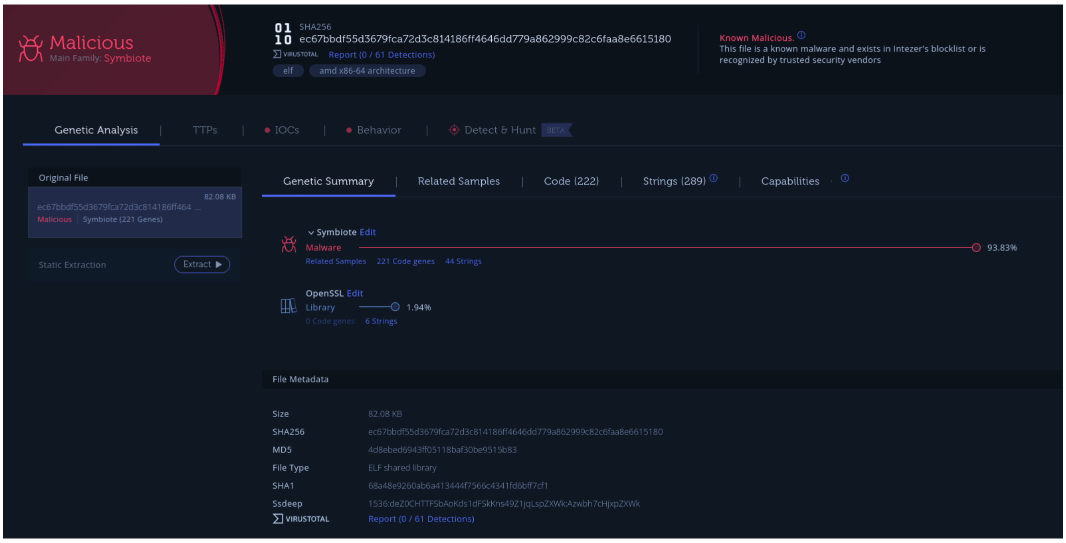


Figure 5: Intezer analysis of a Symbiote sample showing only genes classified as Symbiote.

Conclusion

Symbiote is a malware that is highly evasive. Its main objective is to capture credentials and to facilitate backdoor access to infected machines. Since the malware operates as a userland level rootkit, detecting an infection may be difficult. Network telemetry can be used to detect anomalous DNS requests and security tools such as antivirus (AVs) and endpoint detection and response (EDRs) should be statically linked to ensure they are not "infected" by userland rootkits.

Indicators of Compromise (IoCs)

Hashes

Hash	Notes
121157e0fcb728eb8a23b55457e89d45d76aa3b7d01d3d49105890a00662c924	"kerneldev.so.bkp." Appears to be an early development build.
f55af21f69a183fb8550ac60f392b05df14aa01d7ffe9f28bc48a118dc110b4c	"mt64_.so." Missing credential exfiltration over DNS.
ec67bbdf55d3679fca72d3c814186ff4646dd779a862999c82c6faa8e6615180	"search.so." First

	sample with credential exfiltration of DNS.
a0cd554c35dee3fed3d1607dc18debd1296faaee29b5bd77ff83ab6956a6f9d6	“liblinux.so.”
45eacba032367db7f3b031e5d9df10b30d01664f24da6847322f6af1fd8e7f01	“certbotx64.” dnscat2

Ports Hidden

- 45345
- 34535
- 64543
- 24645
- 47623
- 62537
- 43253
- 43753
- 63424
- 26424

Domains Hidden

- assets[.]fans
- caixa[.]cx
- dpf[.]fm
- bancodobrasil[.]dev
- cctdcapllx0520
- cctdcapllx0520[.]df[.]caixa
- webfirewall[.]caixa[.]wf
- caixa[.]wf

Process Names Hidden

- javaserverx64
- javaclientex64
- javanodex86
- apache2start
- apache2stop
- [watchdog/0]
- certbotx64
- certbotx86
- javautils

File Names Hidden

- apache2start
- apache2stop
- profiles.php
- 404erro.php
- javaserverx64
- javaclientex64
- javanodex86
- liblinux.so
- java.h
- open.h
- mpt86.h
- sqlsearch.php
- indexq.php
- mt64.so
- certbot.h
- cert.h
- certbotx64
- certbotx86
- javautils
- search.so

Credential Exfil Domains

- *.x3206.caixa.cx
- *.dev21.bancodobrasil.dev