DNP

Symbiote

検出がほぼ不可能な Linux の新たな脅威の分析

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

本調査は、Intezer 社のセキュリティ・リサーチャーである Joakim Kennedy 氏と、BlackBerry Threat Research & Intelligence チームとの共同研究に依るもので、BlackBerry 社のブログにも掲載されています。

生物学において Symbiote（共生生物）とは、他の生物と共生している生物のことです。この共生関係は、両方の生物にとって相互に有益である場合もありますが、一方が利益を得て、他方が害を受ける寄生的な場合もあります。数ヶ月前、私たちは、この寄生的な性質を持つ行為を行う、未検出の新しい Linux®マルウェアを発見しました。私たちは、このマルウェアを Symbiote と名付けました。

Symbiote が、他の Linux マルウェアと異なるのは、感染したマシンに被害を与える為に、他の実行中のプロセスを感染させる必要がある点です。マシンを感染するために実行されるスタンドアロン実行ファイルではなく、共有オブジェクト（SO）ライブラリとして、**LD_PRELOAD（T1574.006）**を使用して実行中のすべてのプロセスにロードされ、マシンに寄生して感染させます。実行中のすべてのプロセスに感染すると、ルートキット機能、資格情報の採取機能、リモートアクセス機能を脅威アクターに提供します。

Symbiote の誕生

Symbiote の最初の検出は、2021 年 11 月からで、ラテンアメリカの金融セクターをターゲットとして書かれたようです。マルウェアがマシンに感染すると、マルウェア自身と脅威アクターが使用する他のマルウェアが隠され、感染の検出が非常に難しくなります。感染したマシンでライブフォレンジックを実行しても、ファイル、プロセス、ネットワーク、全てのアーティファクトがマルウェアによって隠されているため、何も発見できない場合があります。ルートキット機能に加えて、このマルウェアは、ハードコードされたパスワードを使用してマシン上の任意のユーザとしてログインし、最高権限のコマンドを実行するバックドアを脅威アクターに提供します。

Symbiote は極めて回避的であるため、Symbiote 感染は "レーダーの下を飛ぶ" 可能性が高いのです。私たちの調査では、Symbiote が高度な標的型攻撃や広範な攻撃に使われているかどうかを判断するのに十分な証拠は見つかっていません。

Symbiote の興味深い技術的側面の 1 つは、Berkeley Packet Filter（BPF）フック機能です。Symbiote は、BPF を使用する最初の Linux マルウェアではありません。たとえば、Equation Group によるものとされている高度なバックドアは、秘密の通信のために BPF を使用してきました。しかし、Symbiote は感染したマシン上の悪意のあるネットワークトラフィックを隠すために BPF を利用します。管理者が感染したマシン上で任意のパケットキャプチャツールを起動すると、どのパケットをキャプチャすべきかを定義するカーネルに BPF バイトコードが注入されます。このプロセスでは、Symbiote が最初にバイトコードを追加して、パケットキャプチャソフトウェアに見せたくないネットワークトラフィックを除外できるようにします。

回避技術

Symbiote は非常に高いステルス性が特徴です。このマルウェアは、**LD_PRELOAD** 環境変数を介してリンカーによってロードされるように設計されています。これにより、他のどの共有オブジェクトよりも先にロードされるようになります。最初にロードされる事により、アプリケーション用にロードされる他のライブラリファイルから「インポートをハイジャック」することができます。Symbiote はこれを利用して、**libc** と **libpcap** の関数をフックすることにより、マシン上の自分の存在を隠します。下の画像は、このマルウェアの回避方法の概要を示しています。

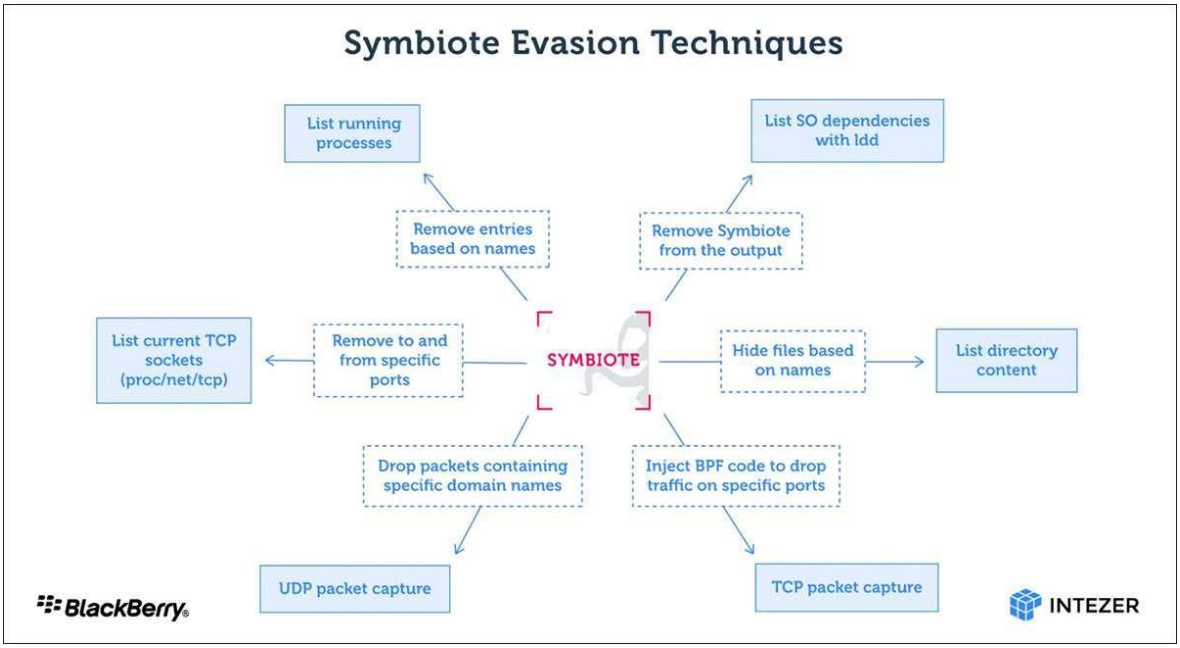

図 1：Symbiote の回避技術

ホストアクティビティ

Symbiote マルウェアは、マシン上の自身の存在を隠すだけでなく、一緒に展開された可能性が高いマルウェアに関連する他のファイルも隠します。バイナリ内には、RC4 で暗号化されたファイルリストが存在します。フックされた関数が呼び出されると、マルウェアはまず libc を動的にロードし、オリジナルの関数を呼び出します。このロジックは、すべてのフックされた関数で使用されます。その例を以下の図 2 に示します。

```
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]=0x9545f1510bb
│   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
```

図 2：libc から readdir を解決するためのロジック。

呼び出し元のアプリケーションが**/proc** 以下のファイルやフォルダにアクセスしようとする場合、マルウェアは、そのリストにあるプロセス名からの出力をこすり落とします。以下のリストにあるプロセス名は、私たちが発見したサンプルから抽出したものです。

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

呼び出したアプリケーションが**/proc** 以下の何かにアクセスを試みていない場合、マルウェアは代わりにファイルリストから結果をこすり落とします。私たちが調査したすべてのサンプルから抽出されたファイルは、以下のリストに示されています。ファイル名のいくつかは、Symbiote が使用するものと一致し、他のものは、感染したマシン上で脅威アクターが使用するツールと疑われるファイルの名前と一致します。このリストには、以下のファイルが含まれています。

- 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

Symbiote が **LD_PRELOAD** 経由でプロセスにロードされる結果の 1 つは、各プログラムに必要な共有ライブラリを印刷するユーティリティである **ldd** などのツールが、マルウェアを、ロードされたオブジェクトとしてリストアップすることです。これに対抗するため、マルウェアは **execve** をフックし、環境変数 **LD_TRACE_LOADED_OBJECTS** を 1 に設定してこの関数への呼び出しを探します。この理由を理解するには、**ldd** のマニュアルページが役立ちます。

> 通常の場合、ldd は LD_TRACE_LOADED_OBJECTS 環境変数を 1 に設定して標準のダイナミックリンカー（ld.so(8) 参照）を呼び出す。 これによりダイナミックリンカーはプログラムの動的依存関係を検査し、（ld.so(8) に記述されている規則に従って）それらの依存関係を満たすオブジェクトを見つけてロードする。ldd は依存関係ごとに、一致するオブジェクトの場所と、それがロードされる（16 進数の）アドレスを表示する。(linux-vdso と ld-linux の共有依存関係は特別で、 vdso(7) と ld.so(8) を参照のこと)。

マルウェアはこれを検知すると、**ldd** と同様にローダーを実行しますが、その結果から自分自身のエントリをこすり落とします。

ネットワーク活動

また、Symbiote は感染したマシン上のネットワーク活動を隠す機能も持っています。これを実現する為に、3 つの異なる方法を使用します。**最初の方法**は、**fopen** と **fopen64** をフックするものです。呼び出したアプリケーションが**/proc/net/tcp** を開こうとすると、マルウェアは一時ファイルを作成し、そのファイルに最初の行をコピーします。その後、各行をスキャンして、特定のポートの存在を確認します。マルウェアは、スキャンしている行に探しているポートを見つけた場合、次の行にスキップします。そうでない場合、その行は一時ファイルに書き込まれます。元のファイルが完全に処理されると、マルウェアはそのファイルを閉じ、一時ファイルのファイル記述子を呼び出し元に戻します。基本的に、これにより呼び出しプロセスにこすり落とされた結果が与えられ、マルウェアが隠したいネットワーク接続のすべてのエントリが除外されます。

Symbiote がネットワーク活動を隠すのに使用する **2 番目の方法は**、注入されたパケットフィルタリングバイトコードをすべてハイジャックすることです。Linux カーネルは拡張 BerkeleyPacket Filter（eBPF）を使用して、ユーザーランドプロセスから提供されるルールに基づいてパケットフィルタリングを行うことができるようにします。フィルタリングルールは、eBPF バイトコードで、カーネルが仮想マシン（VM）上で実行します。これにより、カーネルとユーザーランド間のコンテキストスイッチングが最小限に抑えられ、カーネルが直接フィルタリングを実行するため、パフォ

ーマンスが向上します。

感染したマシン上のアプリケーションが eBPF でパケットフィルタリングを行おうとすると、Symbiote はフィルタリングプロセスをハイジャックします。まず、**libc** の関数 **setsockopt** をフックします。この関数が、ソケット上でパケットフィルタリングを実行するために使用される **SO_ATTACH_FILTER** オプションで呼び出された場合、呼び出したアプリケーションが提供する eBPF コードの前に、自身のバイトコードを先頭に付加します。

コードスニペット 1 は、Symbiote サンプルの 1 つによって注入されたバイトコードの注釈付きバージョンを示しています。バイトコードは、以下の条件にマッチするとパケットを「ドロップ」します。

- IPv6(TCP または SCTP)かつ src ポート(43253 または 43753 または 63424 または 26424)
- IPv6(TCP または SCTP)かつ dst ポート 43253
- IPv4(TCP または SCTP)かつ src ポート(43253 または 43753 または 63424 または 26424)
- IPv4(TCP または SCTP)かつ dst ポート(43253 または 43753 または 63424 または 26424)

このバイトコードはポート番号に基づいてパケットをドロップするだけですが、IPv4 アドレスに基づくトラフィックのフィルタリングも確認されています。いずれの場合も、フィルタリングはマシンからのインバウンド・トラフィックとアウトバウンド・トラフィックの両方に対して動作し、トラフィックの両方向を非表示にします。条件を満たさない場合、呼び出し元のアプリケーションが提供するバイトコードの先頭にジャンプするだけです。

コードスニペット 1 に示すように、サンプルの 1 つから抽出したバイトコードは 32 個の命令で構成されています。このコードは、この後さらにバイトコードが存在することを前提としているため、単独でカーネルに注入することはできません。このバイトコードには、呼び出し元プロセスが提供するバイトコードの先頭にスキップするジャンプがいくつかあります。呼び出し元のバイトコードがなければ、注入されたバイトコードは境界外にジャンプしますが、これはカーネルでは許可されていません。このようなバイトコードは、手書きで作成されるか、コンパイラが生成したバイトコードにパッチを適用する必要があります。いずれにせよ、このマルウェアは熟練した開発者によって作成されたことを示唆しています。

```
; 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.
```

コードスニペット 1：Symbiote サンプルの 1 つから抽出された注釈付きバイトコード。

Symbiote がネットワークトラフィックを隠すために使う **3 番目の方法は**、**libpcap** 関数をフックすることです。この方法は、マルウェアが、リストにあるドメイン名への UDP トラフィックをフィルタリングするために使用されます。これは、このタスクを達成するために、関数 **pcap_loop** と **pcap_stats** をフックしています。Symbiote は、受信したパケット毎に、フィルタリングしたいドメインの部分文字列について UDP ペイロードをチェックします。一致するものが見つかると、マルウェアはそのパケットを無視し、カウンターを 1 増やします。**pcap_stats** はこのカウンターを使用して、真のパケット処理数からカウンター値を減算することによって、パケット処理数を「修正」します。パケットのペイロードが、そのリストにある文字列のどれも含んでいない場合、オリジナルのコールバック関数が呼び出されます。このメソッドは UDP パケットをフィルタリングするために使用され、バイトコードメソッドは TCP パケットをフィルタリングするために使用されます。これらの 3 つの方法をすべて使用することで、マルウェアはすべてのトラフィックを確実に隠蔽します。

Symbiote の目的

このマルウェアの目的は、マシン上での悪意ある活動を隠すことに加え、資格情報を採取し、脅威アクターに対してリモートアクセスを提供することです。資格情報の取得は、**libc の read** 関数をフックすることで行われます。**ssh** や **scp** のプロセスがこの関数を呼び出すと、資格情報が取得されます。資格情報は、まず埋め込まれた鍵を使って RC4 で暗号化され、その後ファイルに書き込まれます。例えば、このマルウェアの 1 つのバージョンでは、キャプチャした資格情報を **/usr/include/certbot.h** ファイルに書き込みます。

資格情報をローカルに保存するだけでなく、資格情報を流出させます。データは 16 進数でエンコードされ、塊にされ、脅威アクターが制御するドメイン名への DNS アドレス（A）レコード要求によって流出されます。A レコードのリクエストは次の通りです。

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

コードスニペット 2：Symbiote がデータを流出させるために使用する DNS リクエストの構造。

マルウェアは、マシンが**/etc/resolv.conf** に設定されたネームサーバを持っているかどうかを確認します。ない場合は、Google の DNS（8.8.8.8）が使用されます。ドメイン名へのリクエストの送信とともに、Symbiote は UDP ブロードキャストとして送信します。

感染したマシンへのリモートアクセスは、いくつかの Linux Pluggable Authentication Module (PAM) 関数をフックすることで実現されます。あるサービスが PAM を使用してユーザを認証しようとすると、マルウェアは提供されたパスワードをハードコードされたパスワードと照合します。提供されたパスワードが一致した場合、フックされた関数は成功の応答を返します。このフックは PAM 内にあるため、脅威アクターは PAM を使用するあらゆるサービスでマシンに認証されるようになります。これには、Secure Shell (SSH) などのリモートサービスも含まれます。

入力されたパスワードがハードコードされたパスワードと一致しない場合、マルウェアは、キーロギング機能の一部として、そのパスワードを保存し、流出させます。さらに、マルウェアは、コマンド＆コントロール（C2）ドメインに DNS の TXT レコード要求を送信します。この TXT レコードの形式は、**%MACHINEID%.%C2_DOMAIN%** です。応答を取得すると、マルウェアはコンテンツを base64 デコードし、コンテンツが正しい ed25519 秘密鍵で署名されているかどうかを確認し、コンテンツを RC4 で復号化し、生成された bash プロセスでシェルスクリプトを実行します。この機能は、通常のプロセスが機能しない場合に、マシンへのアクセスを回復するためのブレークグラス方式（緊急アクセス）と

して動作させることができます。

脅威アクターが感染したマシンに認証されると、Symbiote は root 権限を得るための機能を提供します。共有オブジェクトが最初にロードされると、環境変数 **HTTP_SETTHIS** があるかどうかを確認します。この変数にコンテンツが設定されている場合、マルウェアは有効なユーザとグループ ID をルートユーザに変更し、システムコマンドでコンテンツを実行する前にこの変数をクリアします。

このプロセスでは、SO に setuid 許可フラグが設定されていることが必要です。システムコマンドが終了すると、Symbiote は元のプロセスが実行されないように、プロセスも終了させます。下の図 3 は、実行されたコードを示しています。これにより、シェル内の任意のユーザとして **HTTP_SETTHIS="/bin/bash -p" /bin/true** を実行することにより、ルートシェルを生成できます。

```
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
```

図 3：root 権限でコマンドを実行するためのロジック

ネットワークインフラ

Symbiote マルウェアが使用するドメイン名は、ブラジルのいくつかの大手銀行になりすましていま

す。このことは、これらの銀行またはその顧客が潜在的な標的であることを示唆しています。このマルウェアが利用するドメイン名を利用して、VirusTotal に **certbotx64** という名前でアップロードされていた関連サンプルを発見することに成功しました。このファイル名は、私たちが最初に入手した Symbiote のサンプルの 1 つで隠蔽するファイルとしてリストアップされたものと一致します。このファイルは、dnscat2 というオープンソースの DNS トンネリングツールであると確認されました。

このサンプルには、バイナリに **git[.]bancodobrasil[.]dev** ドメインを C2 サーバとして使用する設定がありました。2 月から 3 月にかけて、このドメイン名は、Njalla の仮想プライベートサーバ（VPS）サービスにリンクされている IP アドレスに付与されました。パッシブ DNS レコードは、同じ IP アドレスが数ヶ月前に **nsl[.]cintepol[.]link** と **ns2[.]cintepol[.]link** に解決されたことを示しました。

Cintepol は、ブラジル連邦警察が提供する情報ポータルサイトです。このポータルでは、警察官が捜査の一環として、連邦警察が提供するさまざまなデータベースにアクセスできます。このなりすましドメイン名に使用されているネームサーバは、2021 年 12 月中旬から 2022 年 1 月末までの間、アクティブになっていました。

また、2022 年 2 月以降、ドメイン **caixa[.]wf** のネームサーバが、別の Njalla VPS IP を指していました。以下の図 4 は、これらのイベントのタイムラインを示したものです。ネットワークインフラに加えて、ファイルが VirusTotal に提出された時点のタイムスタンプも含まれています。これら 3 つの Symbiote サンプルは、ブラジルの同じ提出者によってアップロードされました。このファイルは、インフラがオンラインになる前に VirusTotal に提出されたようです。

これらのファイルは、インフラがオンラインになる前に VirusTotal に提出され、サンプルの中にはローカル IP アドレスを非表示にするルールが含まれていたことから、サンプルは使用前に Antivirus の検出をテストするために VirusTotal に提出された可能性があります。さらに、開発中と思われるバージョンが 11 月末にブラジルから提出されており、Symbiote の背後にいる脅威アクターまたはグループが検出テストのために VirusTotal を使用していたことがさらに示唆されています。

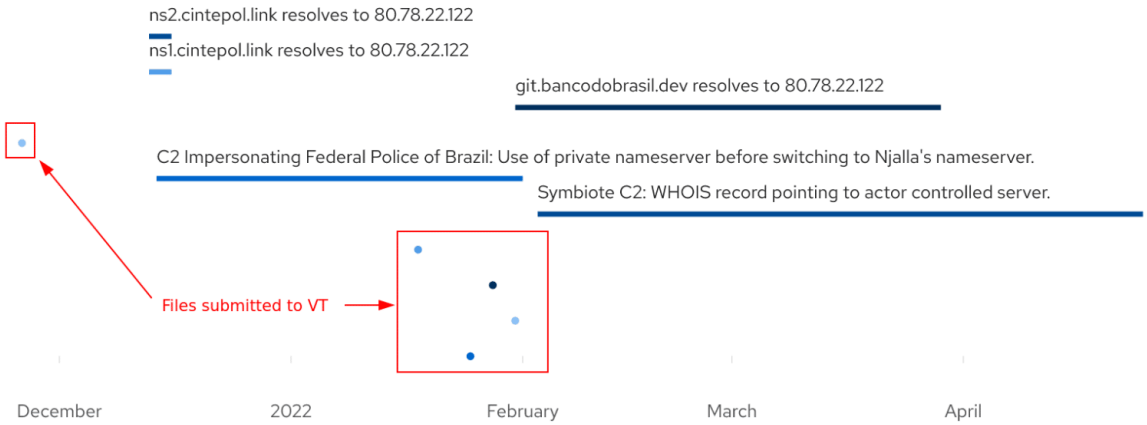

図 4：VirusTotal にファイルが送信された時刻と、ネットワークインフラがアクティブになった時刻を示す時系列図。

他のマルウェアとの類似性

Symbiote は、資格情報を盗むことと、感染した Linux サーバへのリモートアクセスを提供することの両方を目的として設計されているようです。Symbiote は、この目的のために開発された最初の Linux マルウェアではありません。2014 年、ESET は、資格情報の盗用も実行する OpenSSH バックドアである Ebury の詳細な分析を発表しました。両マルウェアファミリーが使用する手法には、いくつかの共通点があります。どちらもフック機能を使用して資格情報を取得し、取得したデータを DNS リクエストとして流出させます。しかし、2 つのマルウェアファミリーが使用するバックドアへの認証方法は異なっています。最初に Intezer Analyze でサンプルを分析したところ、(他のマルウェアファミリーには含まれない)ユニークなコードのみが検出されました（図 5)。Symbiote と Ebury/Windigo やその他の既知のマルウェアの間で共有されているコードはないため、Symbiote は未発見の新しい Linux マルウェアであると明確に結論付けることができます。

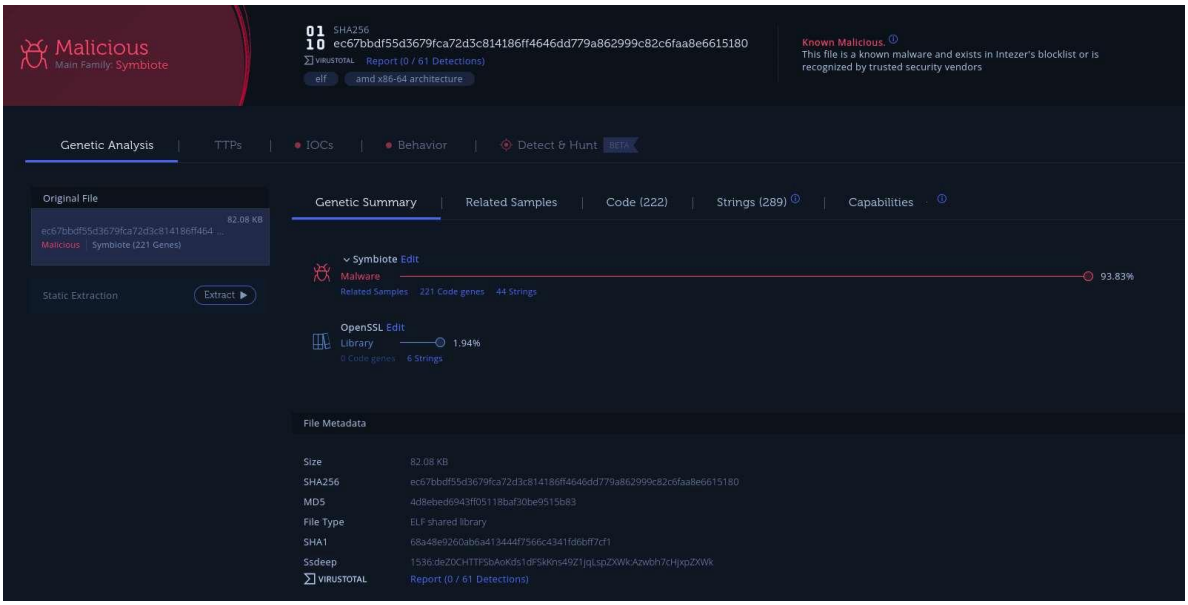

図 5：Symbiote に分類される遺伝子のみを表示した Symbiote サンプルの Intezer 解析。

結論

Symbiote は、高度に回避的なマルウェアです。その主な目的は、資格情報を取得し、感染したマシンへのバックドアアクセスを容易にすることです。このマルウェアは、ユーザーランドレベルのルートキットとして動作するため、感染の検出が困難な場合があります。ネットワークテレメトリにより異常な DNS リクエストを検出し、アンチウイルス（AV）やエンドポイント検出・応答（EDR）などのセキュリティツールを静的にリンクさせ、ユーザーランドルートキットに「感染」していないことを確認する必要があります。

侵入の痕跡　（IoC:Indicators of Compromise）

ハッシュ

ハッシュ値	注記
121157e0fcb728eb8a23b55457e89d45d76aa3b7d01d3d49105890a00662c924	“kerneldev.so.bkp.”初期の開発ビルドのようです
f55af21f69a183fb8550ac60f392b05df14aa01d7ffe9f28bc48a118dc110b4c	“mt64_.so.” DNS を介したクレデンシャルの漏えいがありません
ec67bbdf55d3679fca72d3c814186ff4646dd779a862999c82c6faa8e6615180	“search.so.” DNS のクレデンシャル漏えいを使用した最初のサンプル
a0cd554c35dee3fed3d1607dc18debd1296faaee29b5bd77ff83ab6956a6f9d6	“liblinux.so.”
45eacba032367db7f3b031e5d9df10b30d01664f24da6847322f6af1fd8e7f01	“certbotx64.”dnscat2

隠されたポート

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

隠されたドメイン

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

- webfirewall[.]caixa[.]wf
- caixa[.]wf

隠されたプロセス名

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

隠されたファイル名

- 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

クレデンシャル漏えい ドメイン

- *.x3206.caixa.cx
- *.dev21AW

■原文

Intezer Blog

Symbiote Deep-Dive: Analysis of a New, Nearly-Impossible-to-Detect Linux Threat (June 9, 2022)

https://www.intezer.com/blog/research/new-linux-threat-symbiote/

本稿は Intezer 社の許可を得て、同社のブログを翻訳したものです。翻訳を許諾いただいた Intezer 社に感謝します。

日本語版作成：2022 年 6 月

■免責事項

本稿は原文にできるだけ忠実に翻訳するよう努めていますが、完全性や正確性を保証するものではありません。翻訳監修主体は、本翻訳物に記載されている情報より生じる損失または損害に対して、いかなる人物あるいは団体にも責任を負うものではありません。原文の内容を理解する必要がある場合、上記の原文をお読み下さい。

翻訳監修主体：大日本印刷株式会社　AB センター サイバーセキュリティ事業開発ユニット