Computer Security Symposium 2024
22-25 October 2024

Linuxファイルレスマルウェア検知手法に対するeBPFを用いた回避手法の提案とその対策

高林 裕太[1,a)]　満保 雅浩[1]

概要：ファイルレスマルウェアはメモリ上にペイロードを配置するため，検知やフォレンジックが困難である．近年では Linux をターゲットとした検体も確認されており，その対策として複数の検知手法が考案されている．しかし，こうした検知手法は感染端末に用意されたコマンドやファイルに依存しており，それらが改ざんされた場合を考慮していない．たとえば Linux に存在する eBPF という技術を悪用することで，システムコールやネットワークパケットの改ざんが可能である．そこで本論文では，現在考案されている Linux ファイルレスマルウェアの検知手法に対して，eBPF を用いた検知回避手法を提案し，実装を通してその実現性を示す．さらに，提案した検知回避手法に対する防御策についても考察を行い，既存の検知手法の堅牢性を向上させる手法を示す．

キーワード：eBPF，ファイルレスマルウェア，Linux，検知回避

Proposal of eBPF-based Evasion Methods Against Detection of Linux Fileless Malware and Their Countermeasures

Yuta Takabayashi[1,a)]　Masahiro Mambo[1]

Abstract: It is difficult to detect and analyze fileless malware because its payload resides in memory. Recently, samples of fileless malware have been found on Linux and several detection methods have been proposed to detect them. However, these detection methods rely on commands and files provided on the infected computer and do not consider potential tampering with them. For example, we can tamper with system calls and network packets by using “eBPF”, a technology available on Linux. In this paper, we propose eBPF-based evasion methods against existing detection methods for Linux fileless malware, and show these feasibilities through implementation. In addition, we also discuss countermeasures against the proposed evasion methods and improve robustness of the existing detection methods.

Keywords: eBPF, Fileless Malware, Linux, Detection Evasion

1. はじめに

ファイルレスマルウェアは通常のマルウェアと異なり，メモリ上にペイロードを配置し，ディスク上にファイルを残さない．そのため，従来のマルウェア検知システムでは対処が難しく，フォレンジックも困難である．こうしたファイルレスマルウェアは主に PowerShell を用いて攻撃を行うため，これまでは Windows をターゲットとする検体がほとんどであった．

ところが，2022 年頃を起点として，Linux 向けのファイルレスマルウェア検体も確認されるようになった．これらの検体の多くは memfd_create システムコールと fexecve 関数を用いることで実装されていることから，その特徴を利用した検知手法が提案されている [1], [2]．しかし，現在提案されている検知手法は感染端末のコマンドやファイルに依存しており，それらが改ざんされた場合は正常な動作が期待できない．つまり，検知を回避されるリスクが存在

[1] 金沢大学
Kanazawa University
[a)] ytakabayashi@stu.kanazawa-u.ac.jp

© 2024 Information Processing Society of Japan

This paper is work in progress and not peer-reviewed.

する．

コマンドやファイルの改ざんに繋がりうる技術として，ここではLinuxに搭載されている“eBPF”という技術に着目する．近年，eBPFの悪用に関する発表が数多く行われており[3], [4], [5]，悪用によってシステムコールの入出力の改ざんや，通信パケットの改ざんを行えることが報告されている．Linuxファイルレスマルウェアの検知はシステムコールの使用や通信ログの確認によって行われることが多いことから，eBPFの悪用例を応用することで従来の検知手法を回避されるおそれがある．

そこで本論文では，既存のLinuxファイルレスマルウェア検知手法に対して，eBPFを用いた検知回避手法を提案する．そして提案手法を実装し，疑似ファイルレスマルウェアに組み込むことで，従来の検知手法をバイパスできることを示す．さらに，提案した検知回避手法の対策についても考察を行い，既存の検知手法のロバスト性を向上させる方法を示す．

2. 予備知識

2.1 eBPF

eBPF（extended Berkeley Packet Filter）とは，Linuxカーネルへの機能追加を安全かつ容易にする技術である．

eBPFの登場以前は，カーネルに機能を追加する方法としてはカーネルモジュールが主流であったが，その開発にはカーネルプログラミングに関する知識が必要であった．さらに，作成したモジュールにバグがあるとカーネル全体がクラッシュする可能性があったため，開発者はその安全性にも細心の注意を払う必要があった．

eBPFはこうした問題を解決し，安全かつ容易に機能追加を行うための技術として，Linux3.18から導入された．eBPFにより得られるメリットは次の通りである．

安全性向上 カーネルにロードされる前に，eBPF検証器によってプログラムの安全性チェックが行われる

移植性向上 CO-RE（Compile Once - Run Everywhere）という機能により，異なるバージョンのカーネルでも動作する

作成が容易 C言語だけでなく，PythonやGoでもプログラムを記述できる

eBPFは現在，システムのトレースやネットワークスタックの性能向上，セキュリティの向上といった用途に用いられており，今後もその利用は広がっていくものと考えられる．

3. 関連研究

3.1 eBPFの悪用について

Hogan[3]は，eBPFを用いてreadシステムコールを改ざんし，偽のデータを出力させる手法について解説した．さらに複数の悪用手口の実装をGitHubにリリースし，悪意のあるプロセスのPIDを隠すプログラム“Pid-Hide”や，/etc/sudoersファイルをreadシステムコールで読み込んだ内容を書き換えることで特権昇格を行うプログラム“Sudo-Add”等を公開した[6]．

Fournierら[4]は，eBPFを悪用したルートキットの作成を目標とし，永続化やC&C，ネットワーク探索の手口について解説を行った．

Philippartら[5]は，当時考案されていた悪用手口の取りまとめを行い，eBPFを悪用したルートキットの要件を分析した．さらに，eBPFベースのマルウェアの対処法の考案も行った．

以上はeBPFを用いたルートキットやマルウェアの実現に関する研究であり，検知回避を対象としていない．一方，本論文はファイルレスマルウェア検知を回避する方法に関する論文であり，提案されてきた検知手法をeBPFで回避することを目的としている点が異なる．

3.2 linuxファイルレスマルウェアの手口について

金井ら[2]は，Linuxファイルレスマルウェアの手口を2つに分類している．1つ目はptraceシステムコールを用いた方法であり，ターゲットとするプロセスのメモリを書き換えることで，ペイロードを実行する手法である．2つ目はmemfd_createシステムコールを用いてメモリ上にファイルを作成し，fexecve関数を用いて実行する手法である．

このうち，前者で使用しているptraceシステムコールはカーネル側で機能の無効化が可能なうえ，メモリを直接書き換える必要があり攻撃難易度が比較的高い．一方，メモリ上に作成したファイルは一般的なファイルと同様に読み書きが可能なため，後者の実装難易度は低い．さらに，memfd_createは一般的なソフトウェアでも使用されることがあるため，影響を調べてからでないと無効化することができない．

こうした理由から，本論文ではmemfd_createシステムコールを用いるファイルレスマルウェアのみを対象として，その検知や検知回避を議論することとする．

3.3 Linuxファイルレスマルウェアの検知手法について

田中ら[1]は，memfd_createシステムコールを用いたLinuxファイルレスマルウェアを取り上げ，その検知手法としてmemfd_createシステムコールとfexecve関数をフックする方法を提案した．さらに，検知と同時にファイルレスマルウェアのバイナリを取得する方法も考案した．そして，擬似的なファイルレスマルウェアに対して提案手法を利用し，検知とバイナリの取得に成功した．

金井ら[2]は，さまざまな手法で取得したファイルディスクリプタについて，それがfexecveで実行可能かどうかを網羅的に調査し，ファイルレスマルウェアに使用されうるディスクリプタ取得手法がshm_openシステムコールと

© 2024 Information Processing Society of Japan

memfd_create システムコールの 2 つであることを示した．そして，これらの手法によって取得されたファイルの実行を制限するしくみをセキュア OS 上に実装し，性能評価を行った．実装においては，田中らと同様に execve/execveat システムコールの実行をカーネル側で検知し，それを起点としてファイルレスマルウェアかどうかの判定を行っている．

The Sandfly Security Team[7] は，コマンドライン上で Linux ファイルレスマルウェアの検知を行う手法として，/proc/*/exe に残るファイルを検索する方法を紹介した．これは execve の実行時に，memfd_create により作成されたファイルのシンボリックリンクが/proc/*/exe に生成されることを用いた検知手法である．

また，対象の OS にかかわらず，ネットワークパケットの監視や分析を行う手法も有効であると考えられている．たとえば Poston[8] は，パケットキャプチャを行うことでトラフィックログからファイルレスマルウェアのバイナリを取得する手法を示した．

4. 提案手法

4.1 本論文で扱う検知手法

関連研究で述べた通り，本論文では memfd_create システムコールを用いる Linux ファイルレスマルウェアのみを対象とする．そして，関連研究で提案されている Linux ファイルレスマルウェア検知手法を踏まえ，以下の 3 つの検知手法を回避の対象として扱うこととする．

4.1.1 フックによる検知

本手法では，LD_PRELOAD 環境変数を用いて memfd_create と fexecve をフックし，双方の引数に同じファイルディスクリプタが与えられた場合にファイルレスマルウェアであると検知する．これは田中ら [1] により提案されたものである．

LD_PRELOAD 環境変数に指定されたライブラリは，他の共有ライブラリよりも先に読み込まれる．そのため，フック対象の関数と同名の自作関数を定義したライブラリを作成し，そのパスを LD_PRELOAD 環境変数に代入することで，既存のシステムコールや関数を自作関数と入れ替えることが可能である．

4.1.2 /proc による検知

本手法では，すべてのプロセスに対して ls コマンドで/proc/*/exe の情報を取得し，そのファイル名が memfd_create システムコールにより作成されたことを意味する “memfd:” から始まることを調べることで，ファイルレスマルウェアを検知する．これは The Sandfly Security Team[7] によって紹介されたものである．

fexecve 関数によってファイルが実行されると，その実行可能ファイルへのシンボリックリンクが/proc/*/exe に作成される．これは memfd_create システムコールが作成したファイルであっても同様である．さらに，memfd_create により作成されたファイルの名前は “memfd:” から始まるという性質がある．そのため，/proc/*/exe に “memfd:” から始まるファイルがあるか検索することで，memfd_create で作成されたファイルが実行中かどうかを調べることができる．

4.1.3 パケット監視による検知

本手法では，デバイス上でネットワークパケットを監視し，HTTP レスポンス内にペイロードが含まれるかを調査することで，ファイルレスマルウェアの検知を行う．これは Poston[8] が紹介した手法をヒントにしている．

ファイルレスマルウェアは外部からペイロードを取得する必要があるため，ネットワークパケットの監視を行うことでその検知が可能である．パケットの監視にあたっては通信相手の IP アドレスや通信プロトコルなどさまざまな観点で調査を行うことが想定されるが，本手法では HTTP レスポンス内にペイロードが含まれるかどうかという観点で監視を行う．つまり，悪性サーバから HTTP 通信でペイロードを受け取るような手口を検知することを目的とする．また，デバイスの外部，すなわちルータの手前などでパケットの監視を行うのではなく，デバイス上で監視ソフトを起動して監視することを前提とする．

4.2 提案する検知回避手法

前項で述べた検知手法それぞれに対し，以下の回避手法を提案する．

4.2.1 フックによる検知の回避

本手法では，eBPF を用いて openat システムコールの引数を改ざんし，LD_PRELOAD に指定されたライブラリを読み込まないようにすることで，memfd_create と fexecve のフックを回避する．

LD_PRELOAD によるプリロードは，内部的には以下の手順で行われる．まず，プログラムの実行時に execve システムコールが呼ばれ，環境変数として LD_PRELOAD が渡される．次に，LD_PRELOAD にライブラリが指定されている場合，そのライブラリが openat システムコールで開かれ，read システムコールで読み込まれる．これにより指定されたライブラリが他のライブラリよりも先にロードされる．

この手順を踏まえると，フックを回避するためには openat システムコールを改ざんすればよいことがわかる．そこで本手法では，まず execve システムコールと openat システムコールに eBPF プログラムをアタッチする．そして execve の引数から LD_PRELOAD の値を読み取り，そのライブラリを openat で開こうとした際に，openat の引数を glibc（GNU C ライブラリ）のパスに改ざんする．これによりフックを阻止し，代わりに glibc を開かせることができる．なお，ここで glibc に書き換えたのは，存在しな

© 2024 Information Processing Society of Japan

いファイルを引数に指定するとエラーとなるためである．

以上の方法を用いることで，フックによる検知を無効化できると期待される．

4.2.2 /proc による検知の回避

本手法では，eBPF を用いて，悪性プロセスの PID に対応する/proc/<PID>ディレクトリを非表示にすることで，/proc による検知を回避する．

/proc による検知手法では，ls コマンド等で/proc/*/exe の情報を確認する必要がある．ls コマンドの内部では getdents システムコールが呼ばれており，これによりディレクトリ内のファイル一覧を取得している．ディレクトリ内のファイル情報は linux_dirent64 構造体の配列として保持されており，この構造体は自身のサイズ “d_reclen” やファイル名 “d_name” をメンバとして持つ．このうち d_reclen は，linux_dirent64 構造体の配列が与えられた際，自身の次の構造体のポインタを得るために用いられる．つまり，ある構造体のポインタに，それ自身の d_reclen を加えることで，次の構造体を指すポインタを取得している．

ここで，ある構造体の d_reclen を，自身の d_reclen と次の構造体の d_reclen の和に書き換えることを考える．こうすることで次の構造体が読み飛ばされることになるため，特定のファイルをスキップすることが可能となる．本手法ではこれを利用し，getdents システムコールにより取得した linux_dirent64 構造体の配列について，悪性プロセスの情報を含むディレクトリ（/proc/<PID>）を読み飛ばすように d_reclen の値を改ざんする．これにより，ls や ps といったコマンドを使用しても該当のプロセスが存在しないように見えるため，/proc による検知を回避できると期待される．

4.2.3 パケット監視による検知の回避

本手法では，eBPF でパケットを書き換え，HTTP レスポンスに含まれるペイロードを除去することで，パケット監視による検知を回避する．

前述の通り，本論文ではデバイス上でパケットの監視を行い，HTTP レスポンス内のペイロードを確認することによる検知手法を対象として考えている．これを回避するには，監視ソフトに届く前にパケットを書き換え，正常なパケットに偽装する必要がある．これは eBPF の XDP という機能を用いることで可能となる．XDP プログラムはネットワークデバイス上で実行することができるため，パケットがネットワークデバイスに到達した時点で書き換えを行えば，ソフトウェアによる監視を回避することができる．

しかし，パケットの書き換えを行うことで元のパケットに含まれるペイロードも消えてしまうため，ファイルレスマルウェアがペイロードを利用できないという別の問題が発生する．そのため，書き換えを行うと同時に，ペイロードを別のメモリ領域に避難させる必要がある．これについては eBPF Map という機能を用いることができる．Map はカーネル上で動作する eBPF プログラムとユーザ空間との間でデータのやり取りを行うためのデータ構造であり，そのデータはカーネルのメモリ領域に置かれる．そのため，ディスクに書き込みを行わないというファイルレスマルウェアの性質を損なうことなく，ペイロードの避難が可能となる．

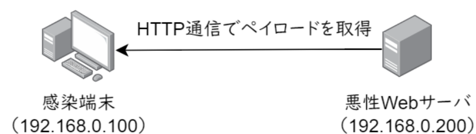

図 1 実験環境の構成
Fig. 1 Structure of an Experimental Environment

そこで本手法では，eBPF XDP プログラムを用いて 2 段階の処理を行う．まず悪性パケットからペイロードを抜き取り，eBPF Map に保存することで，ユーザ空間で動作中のファイルレスマルウェアが使用できるようにする．そのうえでパケットの内容を “204 No Content” という正常なレスポンスに置き換え，パケット監視による検知の回避を試みる．なお，ここで置き換え後のレスポンスに “204 No Content” を使用したのは，サイズが固定長であり改ざんがしやすく，そこで通信を打ち切っても不自然ではないことが理由である．

5. 評価

提案した検知回避手法の実現可能性を示すために，提案手法の実装と評価を行う．

5.1 実験環境

評価にあたっては, Intel Core i7-12700 (2.10 GHz) CPU を搭載した Windows11 ホストマシン上に，**表 1** に示す Ubuntu 仮想環境を 2 台立ち上げて実験を行う．**図 1** に示す通り，1 台はファイルレスマルウェアに感染したマシンとして，もう 1 台は nginx が動作する悪性 Web サーバとして用いる．

悪性サーバとして扱うマシンには，HTTP リクエストに対してペイロードを返すように設定を施す．なおペイロードには，“Hello, World!” と表示した後 5 秒間待機（スリープ）するだけの ELF バイナリファイルを用いる．そして感染端末として用いるマシンには，各検知手法と検知回避手法の実装，ファイルレスマルウェアの疑似検体を含める．各検知手法の実装については後述する．

検知対象の Linux ファイルレスマルウェアについては，実際の攻撃を模倣するために作成した疑似検体を用いる．これは memfd_create システムコールでメモリ上にファイルを作成し，そこに悪性サーバから HTTP 通信で取得したペイロードを書き込み，fexecve 関数を用いて実行するものである．

© 2024 Information Processing Society of Japan

表 1 実験環境
Table 1 An Experimental Environment

項目	内容
OS	Ubuntu 22.04.3 LTS
カーネルバージョン	5.15.0-92-generic
CPU プロセッサ数	2
メモリ	2GB

```
1721 : memfd_create, fileless_sample
1721 : fexecve, fileless_sample
```

図 2 フックによる検知ログ
Fig. 2 A Log of Detection by Hook

```
35899 0 lrwxrwxrwx 1 root root 0 Aug 19 07:08 /proc/1721/exe
     -> /memfd:testfile\ (deleted)
35899 0 lrwxrwxrwx 1 root root 0 Aug 19 07:08 /proc/1721/exe
     -> /memfd:testfile\ (deleted)
35899 0 lrwxrwxrwx 1 root root 0 Aug 19 07:08 /proc/1721/exe
     -> /memfd:testfile\ (deleted)
35899 0 lrwxrwxrwx 1 root root 0 Aug 19 07:08 /proc/1721/exe
     -> /memfd:testfile\ (deleted)
35899 0 lrwxrwxrwx 1 root root 0 Aug 19 07:08 /proc/1721/exe
     -> /memfd:testfile\ (deleted)
```

図 3 /proc による検知ログ
Fig. 3 A Log of Detection by /proc

5.2 検知手法の実装と動作確認

まず，本論文で扱う 3 つの検知手法と Linux ファイルレスマルウェアの疑似検体を実装する．フックによる検知では，田中らの実装を参考にして memfd_create と fexecve の処理を置き換えるための共有ライブラリを作成し，これらのシステムコール・関数が使用された場合にログファイルを作成するようにする．そして，作成したライブラリのパスを LD_PRELOAD 環境変数に設定することでフックを有効にする．/proc による検知では，watch コマンドを用いて 1 秒おきに/proc/*/exe を調査し，“memfd:” から始まるファイルが存在する場合はログに記録する．パケット監視による検知では，tcpdump を用いてパケットキャプチャを行い，80 番ポートの TCP パケットをすべてログに記録する．

そして，これらの検知手法を実行したうえで疑似ファイルレスマルウェアを実行したところ，すべての検知手法で正常に検知することができた．それぞれの検知手法において，検知の際に得られたログを**図 2**，**図 3**，**図 4** に示す．以上のことから，検知回避を行わない疑似検体に対しては，従来の検知手法で検知できることが確認できた．

5.3 検知回避手法の実装とその実現可能性評価

次に，提案した 3 つの検知回避手法を eBPF プログラムで実装し，Linux ファイルレスマルウェアの疑似検体に組み込む．各回避手法は libbpf を用いて C 言語で実装を行い，clang で eBPF バイトコードにコンパイルする．またそれらをカーネルにロードするためのプログラムを別途作成し，このプログラムを疑似ファイルレスマルウェアの内部で実行するようにすることで，疑似検体に検知回避手法を組み込む．

```
07:08:26.389905 eth1 In ifindex 3 08:00:27:d6:be:03
     ethertype IPv4 (0x0800), length 1158: (tos 0x0, ttl 64,
     id 50067, offset 0, flags [DF], proto TCP (6), length
     1138)
   192.168.0.200.80 > 192.168.0.100.49630: Flags [P.],
        cksum 0xd692 (correct), seq 1:1087, ack 57, win
        509, options [nop,nop,TS val 2979987327 ecr
        638260876], length 1086: HTTP, length: 1086
       HTTP/1.1 200 OK
       Server: nginx/1.18.0 (Ubuntu)
       Date: Mon, 19 Aug 2024 06:52:00 GMT
       Content-Type: application/octet-stream
       Content-Length: 824
       Last-Modified: Fri, 09 Aug 2024 05:57:27 GMT
       Connection: keep-alive
       ETag: "66b5afc7-338"
       Accept-Ranges: bytes
```

図 4 パケット監視による検知ログ
Fig. 4 A Log of Detection by Packet Monitoring

実装後，検知手法を実行した状態で，検知回避手法を組み込んだ擬似ファイルレスマルウェアを実行した．その結果，フックによる検知では LD_PRELOAD に示されたライブラリの読み込みができず，memfd_create や fexecve の実行検知に失敗した．また/proc による検知では，/proc/*/exe を検索しても memfd から始まるファイルはヒットせず，ログの生成が行われなかった．さらに，パケット監視による検知ではパケットログの生成はされたものの，**図 5** に示す通り改ざんされたパケットしか取得できておらず，悪性サーバから受信したペイロードはログに残っていなかった．また改ざん前のペイロードが eBPF Map を介して疑似ファイルレスマルウェアに渡され，正常に実行されていることも確かめられた．

以上より，すべての検知手法の回避に成功したことが確認できた．

5.4 検知回避による実行時間への影響評価

最後に，提案した検知回避 eBPF プログラムをカーネルにロードすることで，以降のコマンド実行時間にどのような影響があるかを調べる．実行時間の計測には multitime コマンドを使い，実行時間とシステム CPU 時間の双方について，100 回計測した平均値と標準偏差を算出する．

回避プログラムの有無によるシステムへの影響を調べるために，各回避手法をロードした際の特定コマンドの実行時間変化を調べることにする．具体的には，“フックによる

© 2024 Information Processing Society of Japan

```
07:12:28.318791 eth1 In ifindex 3 08:00:27:d6:be:03
     ethertype IPv4 (0x0800), length 87: (tos 0x0, ttl 64,
     id 10598, offset 0, flags [DF], proto TCP (6), length
     67)
    192.168.0.200.80 > 192.168.0.100.41620: Flags [P.],
        cksum 0x57c0 (correct), seq 1:28, ack 57, win 509,
        length 27: HTTP, length: 27
      HTTP/1.1 204 No Content
```

図 5　検知回避手法を実行した状態での，パケット監視による検知ログ

Fig. 5　A Log of Detection by Packet Monitoring When Using the Proposal Evasion Method

表 2　"フックによる検知の回避" の有無による ps コマンドの実行時間

Table 2　Execution Time of ps with/without Evasion against Detection by Hook

	実行時間（sec）	システム CPU 時間（sec）
無し	0.022 ± 0.008	0.009 ± 0.006
有り	0.017 ± 0.007	0.006 ± 0.005

表 3　"/proc による検知の回避" の有無による find コマンドの実行時間

Table 3　Execution Time of find with/without Evasion against Detection by /proc

	実行時間（sec）	システム CPU 時間（sec）
無し	0.059 ± 0.015	0.026 ± 0.014
有り	0.064 ± 0.014	0.039 ± 0.017

検知の回避" は LD_PRELOAD 環境変数に指定されたライブラリの読み込みを妨害する手法なので，LD_PRELOAD 環境変数にフック用共有ライブラリを指定したうえで，回避プログラムの有無による ps コマンドの実行時間変化を調べる．また，"/proc による検知の回避" は/proc/<PID>ディレクトリを非表示にする手法なので，find コマンドを用いて/proc/*/exe を検索する際の実行時間変化を調べる．そして，"パケット監視による検知の回避" は受信パケットを改ざんする手法なので，curl コマンドで図 1 に示した悪性 Web サーバからペイロードを取得した際の実行時間変化を調べる．

それぞれの実験結果を**表 2**，**表 3**，**表 4** に示す．どの結果を見ても，回避プログラムをロードした際の実行時間やシステム CPU 時間の増加は 0.01 秒未満であった．また，表 2 や表 4 においてはロードしたにも関わらず実行時間が減少しているが，これはプリロード処理が削減されたことや，通信パケットの改ざんによりパケットサイズが減少したことが影響していると考えられる．

以上の結果から，回避プログラムの有無によるシステムへの影響は非常に少ないことがわかった．

表 4　"パケット監視による検知の回避" の有無による curl コマンドの実行時間

Table 4　Execution Time of curl with/without Evasion against Detection by Packet Monitoring

	実行時間（sec）	システム CPU 時間（sec）
無し	0.030 ± 0.009	0.012 ± 0.007
有り	0.027 ± 0.011	0.011 ± 0.007

6. 考察

6.1 提案手法を用いた攻撃シナリオの実現可能性

Linux はサーバ用途や IoT 機器に用いられることが多いため，Windows マルウェアのようにユーザ操作を起点とするシナリオは考えにくい．そのため，攻撃者起点のシナリオを考える必要がある．また，後述するように eBPF の有効化や eBPF プログラムのロードには root 権限が必要なため，場合によっては特権昇格も行う必要がある．

これらを踏まえると，提案手法を用いた攻撃シナリオとして以下のようなものが考えられる．

(1) 資格情報の窃取などにより，対象のマシンに侵入する
(2) 特権ユーザでない場合，脆弱性等を利用して特権昇格を行う
(3) 提案した検知回避用 eBPF プログラムをロードし，ファイルレスマルウェアによる攻撃を開始する

これは一般的な攻撃シナリオとほぼ同じ手順であり，十分実現可能であると考えられる．違いは eBPF の利用だけであるが，その eBPF もカーネルビルド時に無効化していない限りは管理者権限で有効化できるため，実現の可否に大きな影響はない．

ただし，eBPF は開発が盛んな技術のため，対象マシンのカーネルバージョンが低い場合は特定の機能を使用することができない．eBPF 自体は 2014 年末にリリースされた Linux3.18 から導入されており，現在ではほとんどの Linux マシンで動作する．しかし，近年導入された機能はカーネルによっては使えず，例えば eBPF プログラムを異なるカーネルバージョンに対応させるための "CO-RE" という機能については Linux5.17 以降でないと使用できない．

以上の考察から，eBPF の最新機能を使う場合や eBPF 自体が無効化されている場合を除けば，提案手法を用いた攻撃シナリオは十分に実現可能であるといえる．

6.2 提案手法の制限事項

提案した検知回避手法には，eBPF に起因する複数の制限事項が存在する．まず，eBPF プログラムのロードには管理者権限が必要であり，一般ユーザは eBPF を悪用できない．Linux の設定によりロードに必要な権限を変更可能ではあるが，デフォルトでは特権ユーザのみがロード可能となっている．また，カーネルのビルド方法によってはそ

© 2024 Information Processing Society of Japan

もそも eBPF が使用できないように設定することもでき，その場合は特権ユーザであっても eBPF を使用することはできない．

次に，eBPF によってファイルレスマルウェアの痕跡を隠せたとしても，カーネルにロードされた eBPF プログラム自体は “bpftool” というツールを用いることで簡単に確認可能である．その上，eBPF プログラムがユーザメモリの書き換えといった危険な機能を用いた場合は，カーネルのログに警告が記録される． そのため，完璧なステルス性を獲得するためには，こうしたツールやログの対策も必要となる．

このように，eBPF を用いた回避手法には多くの制限がある．しかしながら，システムコールやパケットの改ざんが可能という性質上，一度マシンに侵入されてしまうと検知が非常に難しくなる．そのため，侵入される前，もしくは eBPF プログラムがロードされる前に対策を行うことが重要である．

6.3 提案した検知回避手法の防御策

制限事項を踏まえ，提案手法の防御策を複数提示する．eBPF の悪用を止めるには，カーネルビルド時に eBPF を無効化するのが最も効果的である．しかし，eBPF は今後も多くの分野で活用が見込まれるため，ここでは eBPF 自体の無効化を行わないことを前提とする．

1 つ目は bpf システムコールへのアクセスを制限する方法である．eBPF プログラムは bpf システムコールによってカーネルにロードされるので，ここを制限することで不正な eBPF プログラムのロードを防ぐことができる．具体的な手法としては，ロードの可否を判断する eBPF プログラムをあらかじめ bpf システムコールにアタッチしておく方法が挙げられる．ただし，その eBPF プログラムを攻撃者に取り外されたり，悪意のある eBPF プログラムが先にロードされたりするといった問題も考えられる．

2 つ目は eBPF の一部機能を無効化する方法である．Linux カーネルのビルド時に設定を行うことで，ネットワーク操作や関数フックといった機能ごとに有効・無効を選択することができる．そのため，使わない機能を無効化することでセキュリティリスクの大幅な軽減が可能である．ただし，カーネルのビルドを行う手間がかかるため，あまり現実的な手段とはいえない．

3 つ目は eBPF プログラムの署名を活用する方法である．現時点では構想段階であるが，eBPF プログラムにデジタル署名を含めることが検討されている．実現すれば，署名を検証することでプログラムの提供元が正しいか，そしてプログラムが改ざんされていないかを確認することが可能になる．これにより，不正なプログラムのロードを防ぎ，eBPF の悪用を止めることが期待される．しかし，前述の通りまだ実装されておらず，今すぐ使用することはできない．

これらの防御策を用いることで，提案した検知回避手法の無力化が期待できる．しかし，各手法の説明で述べたような弱点も存在するため，今後の検証が必要である．

7. 倫理的配慮

本論文は検知回避手法の提案となるため，倫理的配慮に細心の注意を払う必要がある．そのため，まずサイバーセキュリティ研究倫理に関するチェックリストを用いてセルフチェックを行った．関係者に対する周知の必要性については，eBPF の悪用が既知の話題であるため，必要性はないと判断した．

そしてその上で，ネガティブな影響を最小限に抑えるための努力を行った．具体的には，提案手法の制限事項や防御策について考察したほか，具体的な実装については必要最小限の説明にとどめることで，第三者による安易な悪用を防止した．

8. おわりに

本論文では，Linux ファイルレスマルウェアに関する従来の検知手法に対して，eBPF を用いた検知回避手法を提案した．そして提案手法の実装を行い，検知の回避が実際に可能であることを示した．さらに，提案した検知回避手法の防御策を考察し，複数の案を提示した．今後は考察で提示した防御策の有効性について詳しい調査を行い，実装・評価を行うことが課題である．

参考文献

[1] 田中紘世，齊藤泰一：Linux におけるファイルレスマルウェア対策，コンピュータセキュリティシンポジウム 2018 論文集， Vol. 2018, No. 2, pp. 601–606 (2018).

[2] 金井遵，内匠真也，花谷嘉一：Linux 向けファイルレスマルウェアに対する OS レベル保護方式，*SCIS2019*, pp. 1–8 (2019).

[3] Hogan, P.: Warping Reality – Creating and countering the next generation of Linux rootkits using eBPF, *DEF CON 29*, (online), available from ⟨https://media.defcon.org/DEF%20CON%2029/DEF%20CON%2029%20presentations/PatH%20-%20Warping%20Reality%20-%20creating%20and%20countering%20the%20next%20generation%20of%20Linux%20rootkits%20using%20eBPF.pdf⟩ (2021).

[4] Fournier, G., Afchain, S. and Baubeau, S.: eBPF, I thought we were friends!, *DEF CON 29*, (online), available from ⟨https://media.defcon.org/DEF%20CON%2029/DEF%20CON%2029%20presentations/Guillaume%20Fournier%20Sylvain%20Afchain%20Sylvain%20Baubeau%20-%20eBPF%2C%20I%20thought%20we%20were%20friends.pdf⟩ (2021).

[5] Philippart, T., van Namen, S. and Hendriks, C.: eBPF-based Malware, (online), available from ⟨https://www.researchgate.net/publication/372499508_eBPF-based_Malware⟩ (2023).

[6] pathtofile: bad-bpf, GitHub (online), available from ⟨https://github.com/pathtofile/bad-bpf/tree/main⟩ (ac-

© 2024 Information Processing Society of Japan

cessed 2024-07-23).
[7] The Sandfly Security Team: Detecting Linux memfd_create() Fileless Malware with Command Line Forensics, Sandfly Security (online), available from ⟨https://sandflysecurity.com/blog/detecting-linux-memfd-create-fileless-malware-with-command-line-forensics/⟩ (accessed 2024-07-29).
[8] Poston, H.: Network traffic analysis for IR: Analyzing fileless malware, Infosec (online), available from ⟨https://www.infosecinstitute.com/resources/incident-response-resources/network-traffic-analysis-for-ir-analyzing-fileless-malware/⟩ (accessed 2024-07-30).

© 2024 Information Processing Society of Japan