Computer Security Symposium 2018
22 - 25 October 2018

Linuxにおけるファイルレスマルウェア対策

田中 紘世[1]　齊藤 泰一[2]

概要：ファイルレスマルウェアでは，ダウンロードされたマルウェア本体 (ペイロード) はハードドライブ上に格納されることはない．ダウンロードされたペイロードは，OS の機能により直接メモリ上に展開され，実行された後，削除される．メモリ上から削除されること，ファイルとしての実体を持たないことが，ペイロードのフォレンジックを困難とする．ハードディスク上に存在するドロッパー・ローダーは，ペイロードをダウンロード・実行するのみであり，これを解析してもマルウェア全体としての動作を解析できない．我々は Linux において想定されるファイルレスマルウェアに使われる技術を分析し，その対策法について考察した．本項では Linux システムコール memfd_create を利用したファイルレスマルウェアへの対策手法を述べる．

キーワード：ファイルレスマルウェア Linux システムコール

Mitigaton of fileless-malware in Linux

Kousei Tanaka[1]　Taiichi Saito[2]

Abstract: In a fileless malware, a dropper downloads the main part of malware called payload from network and does not store it into the hard drive. It stores the payload directly into memory, activates and deletes it. It is difficult to analyze the payload with existing digital forensics methods since the payload is temporarily placed in the memory and finally deleted. In this paper, we investigate a new kind of fileless malware that uses memfd_create systemcall and consider mitigation for it.

Keywords: fileless-malware Linux SystemCalls

1. はじめに

ファイルレスマルウェアとはファイルとして実体を持たないマルウェアである．ファイルとしてハードドライブ上に存在するドロッパーはマルウェア本体 (ペイロード) をダウンロードおよび実行するのみである．ダウンロードされたペイロードはメモリ上に直接展開される．ペイロードは実行後，ファイルレスマルウェア本体および OS の機能によって削除される．メモリ上から削除され，ハードドライブ上には置かれないことからペイロードのフォレンジックは困難である．ドロッパーはペイロードをダウンロードし，実行する機能しか持たないため，これを解析してもマルウェア全体としての動作を解析することができない．

本稿は Linux におけるファイルレスマルウェアに関するものである．Windows に比べて，Linux におけるファイルレスマルウェアはあまり確認されていない．しかし，それを可能とする技術は存在しており，いずれ多く見られるようになると予想される．本稿では，最近提案されたファイルレスマルウェア技術 [6] を調査し，その対策方法を考察した．利用されるシステムコールをフックすることで，ダウンロードされたペイロードを抽出することを可能とした．

[1] 工学研究科 情報通信工学専攻，東京電機大学 〒 120-8551 東京都足立区北千住旭町 5. Tokyo Denki University,5,Senju-Asahicho,Adachi,Tokyo,120-8551,Japan
[2] 東京電機大学 〒 120-8551 東京都足立区北千住旭町 5. Tokyo Denki University,5,Senju-Asahicho,Adachi,Tokyo,120-8551,Japan

2. 背景

2.1 ファイルレスマルウェアとは

ファイルレスマルウェアとはファイルとしての実体を持たないマルウェア (ウィルス) のことである．既存のマルウェアはファイルとして実体を持ち，ハードドライブ上に設置される．アンチウィルスソフトはマルウェアが設置される場所を監視し，マルウェアを検知すると隔離する．または，ハードドライブ上をスキャンすることでマルウェアを発見する．しかし，ファイルレスマルウェアは既存のアンチウィルスソフトによって検知されず，ファイルとして実体を持たないがために解析も困難であるという特徴を持つ．ファイルレスマルウェアはハードドライブ上にはドロッパーのみを設置する．ドロッパーはメモリ上にマルウェアをダウンロードし，実行する．マルウェアの実行が終わり次第，そのマルウェアはメモリ上から削除されてしまうため，マルウェア解析に必要な検体として残すことを困難にしている．

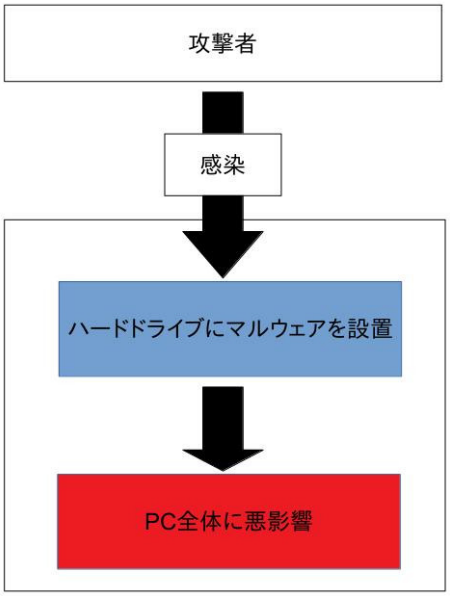

図 1 既存のマルウェアの構造
Fig. 1 structure of generally malware

3. 関連技術

3.1 Linux におけるファイルレスマルウェア

Linux におけるファイルレスマルウェアは process injection を利用したものが一般的である．process injection としては多くの方法があり，その中で ptrace システムコールを用いたものが最も簡単である．ptrace システムコールを使うと，あるプロセスが別のプロセスの実行を監視および制御できるようになり，メモリーとレジスタの変更を可能にする．このシステムコールは主に開発者がデバッグする際に使用する [1]．ptrace システムコールを用いて，攻撃者は特定のプロセスにアタッチし，そのプロセスにコードを注入することで攻撃を行う．しかし，実運用システムでは ptrace の動作を制限したカーネルが用いられることが多い．

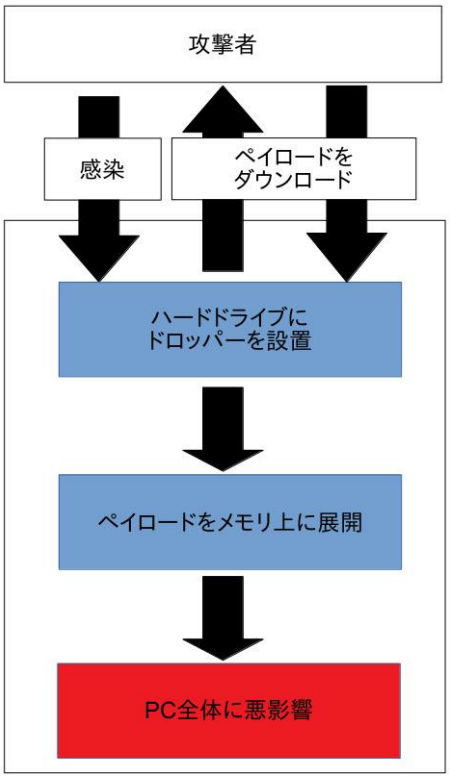

図 2 ファイルレスマルウェアの構造
Fig. 2 structure of fileless malware

Linux では Windows における PowerShell を利用したファイルレスマルウェアのようなマルウェアはあまり確認されていない．しかし，そのようなファイルレスマルウェアに転用可能な技術が 0x00sec[6] により発表された．この技術では Linux に存在するいくつかのシステムコールやライブラリ関数を組み合わせることで，メモリ上にマルウェアを設置し，実行することを可能とする．

3.2 Linux システムコール

0x00sec.org[6] で発表されたファイルレスマルウェアのアイディアでは memfd create システムコールと fexecve 関数が利用される．

memfd_create システムコールはメモリ上にファイルを生成し，そのファイルディスクリプタを返却するシステムコールである．memfd_create() システムコールは Linux 3.17 で初めて登場した [2]．

fexecve 関数は execve システムコールと同じ作業を行う [3]．execve はファイルパスを引数にとり，それが指し示すファイルを実行するシステムコールであるが，fexecve 関数はファイルディスクリプタを引数にとり，それが指し

示すファイルを実行する．2 つのシステムコールと関数を併用することにより，メモリ上に作成されたファイルを直接実行することを可能とする．

memfd_create システムコールの書式は下記の通りである．

```
#include <sys/memfd.h>
int memfd_create(const char *name, unsigned int
    flags);
```

ソースコード 1 memfd_create システムコールの書式

memfd_create システムコールは無名ファイルをメモリ上に作成する．name に指定された名前はファイル名として使用される．flags に指定される値により動作を変更できる，指定可能な値は下記の通りである．

- MFD_CLOEXEC
- MFD_ALLOW_SEALING

MFD_CLOEXEC は新しいファイルディスクリプタに close-on-exec フラグをセットする．MFD_ALLOW_SEALING はファイルに対して sealing 操作を許可する．memfd_create システムコールの返り値はメモリ上に作成したファイルを示すファイルディスクリプタである．

fexecve 関数の書式は下記の通りである．

```
#include <unistd.h>

int fexecve(int fd, char *const argv[], char *
    const envp[]);
```

ソースコード 2 fexecve 関数の書式

fexecve 関数は fd で指定されたファイルを実行する．fexecve 関数の返り値は呼び出しに成功した場合は戻る事はない．エラーの場合は-1 が関数に戻る．

3.3 ファイルディスクリプタ

ファイルディスクリプタとはプログラムがファイルにアクセスするための記述子である．ファイルディスクリプタは数値で表現され，その上限は OS により決まっている．ファイルディスクリプタの 0，1，2 番はそれぞれ標準入力，標準出力，標準エラー出力となっている．プロセスがファイルを開くとファイルディスクリプタ 3 番以降が割り当てられる．プロセスはファイルディスクリプタにアクセスすることでファイルを読み書きすることができる．

ファイルディスクリプタがファイル名で， 実際のファイルへのシンボリックリンクになっている [4]．例えば，PID100 番のプロセスがファイルを開き，ファイルディスクリプタ 3 番が割り当てられた時，”/proc/100/fd/” 以下に ’3’ というシンボリックリンクが作成される．このシンボリックリンクは PID100 番のプロセスが開いたファイルへのリンクとなっている．これがファイルディスクリプタによりファイルへとアクセスすることができる仕組みである．

4. 提案手法

我々は Linux におけるファイルレスマルウェア対策として，該当するシステムコールおよびライブラリ関数をフックする手法を提案する．提案する手法では memfd_create システムコールと fexecve 関数をフックすることでファイルレスマルウェアであることを検知する．fexecve 関数をフックすることにより，fexecve 関数が実行するファイルディスクリプタが指し示すファイルを抽出することを可能とする．環境変数 LD_PRELOAD にバイナリを指定することで対応したライブラリ関数への関数フックが可能となる．memfd_create はシステムコールであるが，実際にプログラムから呼び出されるシステムコールは libc のラッパールーチンである．このため，ラッパールーチンをフックし動作を付け加えることでシステムコールに対してもフックすることが可能となる．また，関数をフックすることにより，その関数に渡された引数を確認することも可能となっている．これを利用し，memfd_create システムコールと fexecve 関数に渡されたファイルディスクリプタが一致している場合のみマルウェアと検知することができる．この場合，マルウェアを抽出後，本来のライブラリ関数に処理を返さないことで fexecve 関数の動作を無効にできる．

4.1 提案手法のソースコード

提案手法のソースコードは下記の通りである．

```
#include<dlfcn.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<errno.h>
#include<sys/syscall.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<arpa/inet.h>

int (*fexecve2)(int fd, char *const argv[], char
    *const envp[]);
int (*memfd_create2)(const char *name, unsigned
    int flags);

__attribute__((constructor))
static void init_hook()
{
  fexecve2 = (int (*)(int, char *const *, char *
      const *))dlsym(RTLD_NEXT, "fexecve");
    memfd_create2 = (int (*)(const char *,
        unsigned int))dlsym(RTLD_NEXT, "
        memfd_create");
}

__attribute__((destructor))
static void fini_hook()
{
}
```

```
int fexecve(int fd, char *const argv[], char *
    const envp[])
{
  // ログファイル作成
  FILE *fp;
  fp = fopen("/tmp/log-fexecve.log","a+");
  pid_t pid;
  pid = getpid();
  char prg_name[64];
  char prg_path[256];
  readlink("/proc/self/exe",prg_path,sizeof(
      prg_path));
  sprintf(prg_name, "%s", basename(prg_path));
  // pidとプロセス名をログ
  fprintf(fp, "%d : %s, %s\n", pid, __func__,
      prg_name);
  fclose(fp);

  // memfdで書かれたファイルを/tmpに出力
  char command[32];
  sprintf(command, "/proc/%d/fd/%d", pid, fd);
  FILE *input_binary;
  if((input_binary = fopen(command, "rb")) ==
      NULL){
    exit(1);
  }
  char buf[10240];
  fread(buf, 1, 10240, input_binary);

  FILE *output_binary;
  if((output_binary = fopen("/tmp/output_binary",
       "wb")) == NULL){
    exit(1);
  }
  fwrite(buf, sizeof(char), 10240, output_binary
      );

  (int)fexecve2(fd, argv, envp);
  return 0;

}

int memfd_create(const char *name, unsigned int
    flags)
{
    FILE *fp;
    fp = fopen("/tmp/log-fexecve.log","a+");
    pid_t p_pid;
    p_pid = getpid();
    char prg_name[64];
    char prg_path[256];
    readlink("/proc/self/exe", prg_path, sizeof(
        prg_path));
    sprintf(prg_name, "%s", basename(prg_path));
    fprintf(fp, "%d : %s, %s\n", p_pid, __func__,
         prg_name);
    fclose(fp);

    return syscall(__NR_memfd_create, name, flags
        );
}
```

ソースコード 3 関数フックのソースコード

このソースコードは，memfd_create システムコールのラッパー関数と fexecve 関数をフックする．この関数フックは，それぞれの関数に一定の動作を付け加える．提案手法はそれぞれの関数を利用したアプリケーションの PID とプロセス名をログとして記録し，fexecve 関数の引数となっているファイルディスクリプタを辿る．ファイルディスクリプタは/proc/<pid>/fd/FD のシンボリックリンクとなっているため，このリンクを辿ることで実行しようとしているファイルを確認できる．その参照するファイルを読み込み，その読み込んだ内容を/tmp 以下に書き出すことで動作するペイロードを保持する．このソースコードでは行なっていないが，fexecve 関数本体に処理を渡さないことでファイルディスクリプタで指定されているペイロードを実行させないことも可能となる．正規のアプリケーションが fexecve 関数を利用している可能性も考えられる．この場合は，memfd_create システムコールを検知したのち，その指し示すファイルディスクリプタが close されることなく fexecve 関数に渡される場面を検知することでファイルレスマルウェアと推定することが可能である．ファイルレスマルウェアの場合は，memfd_create システムコールで確保されたファイルディスクリプタに対し，ダウンロードしてきたペイロードを書き込む動作が入るため，それを検知することでも検知精度の向上を測ることができる．

4.2 調査

提案手法を用いて，ファイルレスマルウェアを検知，およびそのペイロードを取得することができるか調査した．調査環境は下記の通りである．

表 1 実行環境

ディストリビューション	ArchLinux
カーネルバージョン	4.17.6-1-ARCH

利用したファイルレスマルウェアのコードは下記の通りである．

```
#include <fcntl.h>
#include <linux/memfd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

extern char        **environ;

int main(void){
  int fd,malware;
  malware = open("./shellcode.so",O_RDONLY);
  char buf[10240];
  read(malware,buf,10240);

  fd = memfd_create("memfd_test",0);
  write(fd,buf,sizeof(buf));
  char *args[2]= {"[kworker/u!0]", NULL};
  fexecve(fd,args,environ);
  return 0;
}
```

ソースコード 4 利用したファイルレスマルウェア

このソースコードでは，memfd_create システムコールを利用しシェルコードをメモリに書き，fexecve 関数により，それを実行する．これをコンパイルしたバイナリを fileless-malware という名前にし実行した結果，取得できた

ログが下記である．

```
903 : memfd_create, xfwm4
1116 : memfd_create, pactl
1115 : memfd_create, wrapper-2.0
1159 : memfd_create, pactl
1233 : memfd_create, TVGuiSlave.64
1233 : memfd_create, TVGuiSlave.64
10852 : memfd_create, chrome
11083 : memfd_create, firefox
11753 : memfd_create, fileless-malware
11753 : fexecve, fileless-malware
```

図 3 取得したログ
Fig. 3 Obtained log

このうち, fexecve システムコールを利用した PID 11753, プロセス名 fileless-malware を/tmp 以下に output_binary として書き込むことでマルウェアのバイナリを取得することに成功した．取得したバイナリに実行可能権限を付与し，実行したところソースコード 4 で利用していた shellcode.so と同等の動作をすることが確認された．

また，提案手法を用いて，正規のアプリケーションが memfd_create システムコールと fexecve 関数を利用しているか調査した．/etc/profile に LD_PRELOAD=/usr/bin/hook.so とする事でシステムが起動した時に読み込まれる環境変数に関数フックを設定することができる．これにより，環境変数を読み込み後に起動したプロセスに対し，関数フックをすることができる．

```
LD_PRELOAD=/usr/bin/hook.so
export LD_PRELOAD
```

著者の環境で調査したところ，memfd_create システムコールを利用しているアプリケーションの一例が下記である．

- pulseaudio 12.2
- firefox 56.0.2
- google-chrome 64.0.3282.186

PulseAudio は GNOME や KDE などのデスクトップ環境で一般的に使われているサウンドサーバーである [5]. firefox や google-chrome はブラウザである．

一方で fexecve 関数を利用しているアプリケーションはこの調査では観測されなかった．このことから fexecve 関数は頻繁に利用されるような関数でなく，memfd_create システムコールの後，fexecve 関数を検知することでファイルレスマルウェアであると特定することが可能であると言える．

5. 今後の展望

攻撃者によりシステムコールやライブラリ関数をインターポジショニングされた場合，提案した手法ではフックすることが出来ない．これは LD_PRELOAD が動的ライブラリ関数にしか働かないためである．関数がインターポジショニングされた場合，その関数は攻撃者により再定義され，事前にフックしようとした関数とは別物となる．/proc/<pid>/fd 以下のシンボリックリンクを辿ることにより，ペイロードのダンプは可能であることから，別の検知手法が必要になると考えらえる．この場合，strace による検知は可能である．システムコール番号は再定義されても変更されないため，システムコール検知により memfd_create システムコールは検知することができる．しかし，fexecve 関数は内部では execveat システムコールが利用されており，このシステムコールは正規のアプリケーションでもよく利用されるものである．そのため，fexecve 関数に関してはインターポジショニングにより検知することが困難となる．複数の方法を組み合わせて検知するといった別の手法が必要になると考えられる．

6. まとめ

本稿では Linux におけるファイルレスマルウェアの分析を行い，それに対する対策を考察した．このマルウェアは memfd_create システムコールと fexecve 関数を利用することにより実装していた．我々はこの 2 つの関数に対し，それぞれフックすることでマルウェアの動作を検知し，さらにダウンロードされたペイロードを抽出することに成功した．システムコールをフックすることにより，検知漏れすることも考えづらく，攻撃自体を阻害することも可能である．しかし，攻撃者に関数をインターポジショニングし直されてしまうとフックできないことから，別のアプローチも必要であると考えられる．

Linux におけるファイルレスマルウェアは現在確認されていないが，今後出現すると予想される．今後 Linux を攻撃対象としたファイルレスマルウェアが出現した場合，提案手法により単純なファイルレスマルウェアを無効とすることが可能である．

参考文献

[1] ptrace() の無効化 - Red Hat Customer Portal https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/sect-security-enhanced_linux-working_with_selinux-disable_ptrace [2018/08/18 accessed]
[2] Man page of MEMFD_CREATE https://linuxjm.osdn.jp/html/LDP_man-pages/man2/memfd_create.2.html [2018/08/02 accessed]
[3] Man page of FEXECVE https://linuxjm.osdn.jp/html/LDP_man-pages/man3/fexecve.3.html [2018/08/02 accessed]
[4] Man page of PROC https://linuxjm.osdn.jp/html/LDP_man-pages/man5/proc.5.html [2018/08/10 accessed]
[5] PulseAudio - ArchWiki https://wiki.archlinux.jp/index.php/PulseAudio

[2018/08/02 accessed]
[6] Super-Stealthy Droppers
https://0x00sec.org/t/super-stealthy-droppers/3715
[2018/08/03 accessed]

付　　録

A.1 malware-code

```
#include <stdio.h>
#include <stdlib.h>

#include <sys/syscall.h>

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define __NR_memfd_create 319
#define MFD_CLOEXEC 1

static inline int memfd_create(const char *name,
    unsigned int flags) {
    return syscall(__NR_memfd_create, name, flags
        );
}

extern char        **environ;

int main (int argc, char **argv) {
  int                fd, s;
  unsigned long      addr = 0x0100007f11110002;
  char               *args[2]= {"[kworker/u!0]",
      NULL};
  char               buf[1024];

  // Connect
  if ((s = socket (PF_INET, SOCK_STREAM,
      IPPROTO_TCP)) < 0) exit (1);
  if (connect (s, (struct sockaddr*)&addr, 16) <
      0) exit (1);
  if ((fd = memfd_create("a", MFD_CLOEXEC)) < 0)
      exit (1);

  while (1) {
      if ((read (s, buf, 1024) ) <= 0) break;
      write (fd, buf, 1024);
    }
  close (s);

  if (fexecve (fd, args, environ) < 0) exit (1);

  return 0;

}
```

ソースコード 5 ファイルレスマルウェアのソースコード例

このソースコードは 0x00sec[6] からの引用であり，本稿の実験に使用したものは一部を改変した下記のものである．実際に使用したファイルレスマルウェアでは，memfd_create システムコールのラッパー関数の再定義を行なっておらず，標準ライブラリに存在するものをそのまま利用している．これは 5 節でも述べた通り，提案手法が再定義された関数をフックすることができないためである．