Vol.2014-DPS-158 No.18
Vol.2014-CSEC-64 No.18
2014/3/6

システムコール呼び出し回数を削減したegghunter

山内 志織[1,a)]　齊藤 泰一[1,b)]

概要：egghuningとは、メモリ脆弱性のあるアプリケーションに対する攻撃手法の一つである。先行研究としてSkapeの提案したegghunterが存在するが、そこでは、システムコール呼び出し回数が多く実行時間が長くなるという問題点が存在した。そこで本稿では、Skapeのegghunterを改良しシステムコールの呼び出し回数を削減した方式を提案する。

1. はじめに

Staged shellcodeとは、メモリ脆弱性(buffer overflow脆弱性、integer overflow脆弱性、 format string脆弱性等)を持つアプリケーションに対する攻撃において、攻撃に利用できるバッファのサイズが小さいためshellcodeを格納できないときに用いられる技術である。

このStaged shellcodeに分類される技術としてegghuntingというものがある。egghuntingでは、egghunterと呼ばれる小さいサイズのshellcodeを使用する。まず、egghunterはアプリケーションのプロセスのバッファに格納され、実行される。その後、アプリケーションのプロセスの仮想アドレス空間内で、実際に攻撃を行うshellcode本体を探索し実行する。

Skape [1]のLinux版egghunterでは、`access`システムコール、`sigacion`システムコールを用いることにより、安全に仮想アドレス空間の探索を行っている。しかし、一般にシステムコールの呼び出し回数が多いため実行時間が長くなってしまう、という問題点があった。本稿では、Skapeのegghunterを改良し、システムコールの呼び出し回数を削減した方式を提案する。

2. 準備

2.1 開発環境

本稿で提案するegghunterの開発は以下の環境で行った。

Kernel version：Linux 2.6.38

Architecture：i686

Processor： Intel(R) Core(TM) i5-3570K CPU @ 3.40 GHz

2.2 egghuntingの概要

egghuntingは、仮想アドレス空間に格納されているshellcode本体をeggと呼ばれる目印を手がかりに探し出し実行する技術である。まず、egghuntingで必要になる要素のtagとeggについて説明する。

tag：任意の4byte。shellcode本体を仮想アドレス空間から探すときの目印となる。

egg：tagを2つ並べた8byte。shellcode本体はeggの直後に格納される。

また、egghunterは以下の3つの条件を満たさなければならない。

(1) 頑健性 (robustness) を持つ
(2) サイズが小さいこと
(3) 処理が速いこと

egghunterが頑健 (robust) であるためには、egghunterは、不適切に参照した場合に領域違反を起こしたりアプリケーションを破壊したりしてしまう可能性がある無効 (invalid) なメモリ領域に、安全にアクセスできる必要がある。

また、egghunterはshellcodeの一部であるので以下の条件も満たさなければならない。

- bad charecters(`x00`,`x0a`,`x0d`等)を含まないこと
  `x00`は文字列の終端を表す`NULL`文字である。また、`x0a`と`x0d`は共にコマンドの最後を表す文字である。
- `jmp`命令には相対jumpを使用すること
  絶対jumpでは、jump先のアドレスを特定しjumpしなければならない。しかし、一般に目的のアドレスを特定するのは難しい。したがって、`jmp`命令には相対jumpを使用する。

[1] 東京電機大学情報通信工学科
暗号方式・暗号プロトコル研究室
Tokyou Denki Uniersity
[a)] 10ec122@ms.dendai.ac.jp
[b)] taiichi@c.dendai.ac.jp

2.3 提案方式構成のための仮定

今回の提案方式は Skape の方式を改良して処理速度を向上させたものである。提案方式の構成は以下の仮定と実験結果に基づく。

仮定:ページの中に 1 つでも有効なアドレスがあれば、そのページの全アドレスは有効。

この仮定が成り立てば、システムコール呼び出し回数が削減でき、処理速度を向上させることができる。まず、この仮定の説明をする為にページとアドレスについて述べる。

ページは仮想アドレス空間を構成するものである。通常使用されるページサイズは 32 ビットシステムでは 4KB であり、本稿の実験環境でのページサイズも 4KB である。また、ページには有効・無効という概念がある。[1] 有効なページとは対応する物理メモリ、またはスワップ領域やディスク上のファイルなどの二次記憶装置上の領域が存在するページである。無効なページは対応する物理メモリなどが存在せず、未使用またはアドレス空間に割り当てられてないことを表す。

次にアドレスの説明をする。仮定の文中で使われているアドレスは仮想アドレスを意味している。Skape はアドレスの有効・無効を検査する際、システムコールを用いる。システムコールには、無効なメモリアドレスに遭遇したとき EFAULT エラーを返却するものがある。このことがアドレスが有効であるかを判別する際に役立つ。EFAULT とは、POSIX.1 で定義されているエラーで”bad address”を意味する。

Skape の egghunter では、アドレスの検査のために access システムコール (あるいは sigaction システムコール) が使用されており、検査対象のアドレスに対し、access システムコール (あるいは sigaction システムコール) が EFAULT エラーを返さない場合はそのアドレスを有効な (valid) アドレス、EFAULT エラーを返す場合はそのアドレスを無効な (invalid) アドレスとして定義しているようである。本稿でもこの定義を採用する。また、この定義で有効とされたアドレスは、有効なページ上の読み込み可能なアドレスであることが、次節の実験の中で確認された。

Skape の egghunter は、アドレスを一つずつシステムコールによって検査し、ページ内で無効なアドレスに遭遇した場合、そのアドレスを含むそれ以降の同じページの全アドレスを無効なアド レスと判断して検査を中止し、次のページのアドレスの検査に処理を移す。つまり、Skape の egghunter では、無効なアドレスが含まれるページに含まれるアドレスは無効であると仮定しているようである。 本稿でもこの仮定を採用する。それに加え、本稿ではページの中に一つでも有効なアドレスがあればそのページの全アドレスは有効であるという上の仮定を立てた。

仮定が成り立つ場合、有効なアドレスに 1 つでも遭遇すればそのページは有効となり、そのページのそれ以降のアドレスについてシステムコールを呼び出して調査する必要がない。よって、Skape の egghunter よりもシステムコールの呼び出し回数が減るので処理が速くなることが期待される。

2.4 仮定の検証

ここでは 2 つの事柄について検証する。

- システムコールによって EFAULT エラーが返されないアドレスは、読み込み可能なアドレスか。
- ページに一つでも有効なアドレスが存在するとき、同一ページのその他のアドレスも有効であるか。

Skape の egghunter は、アドレスの有効・無効を検査する際、access システムコールにより EFAULT エラーが返されるかどうかで有効・無効を判定している。

access システムコールは、本来はファイルに対する実ユーザでのアクセス権をチェックするためのものである。引数にはファイルのアクセス権をチェックするフラグをセットすることができ、引数指定のアクセス権が無い場合に EFAULT エラーを返す。

Skape の egghunter と提案方式では、access システムコールを本来の使用目的とは異なる使い方をして、アドレスの有効・無効を検査している。Skape の egghunter では access システムコールのフラグを F_OK にセットしている。本来は、F_OK はファイルそのもののアクセス権に関係なくファイルが存在するかどうかのチェックのみしか行わないため、Skape の egghunter のアドレスの検査では、アドレスが存在するかどうかの判定しかしていないように見える。

そこで、access システムコールでアドレスの存在を判定した後、そのアドレスが読み込み可能かどうか判定するプログラムを用いて実験を行なった。この結果、access システムコールで存在すると判定されたアドレスは必ず読み込み可能であることが確認できた。

また、有効なアドレスに遭遇した場合、そのアドレスから 1 ページ分のアドレスが有効なアドレスであることが確認できた。このことから、ページの先頭アドレスが有効なアドレスの場合、このページの全アドレスは有効なものであり、このページは有効なページだということが言える。したがって、Linux Kernel 2.6.38 の i686 アーキテクチャ版の実装では、この仮定に妥当性があると言える。

3. 関連研究

2004 年 Skape[1] によって提案された egghunter がある。

3.1 概要

Skape の egghunter は以下のような構成を持つ。

- 仮想アドレス空間からアドレスを取り出し以下の検査を行う。

– そのアドレスが有効であるかどうかの検査
– そのアドレスから始まる 8 バイトに egg が格納されているかどうかの検査

- いずれかの検査に失敗したとき、アドレスを取り直す処理に戻る。
- 両方の検査に成功したとき、egg(あるいは egg の直後に格納されている shellcode 本体) に制御を移す。

また、仮想アドレス空間からアドレスを取り出しアドレスが有効であるか検査するために、Skape の Linux 版 egghunter では、access システムコールと sigactioin システムコールを使用している。

- access システムコール
  int access
  (const char *pathname, int mode);
  システムコール番号は 0x21 であるので、access システムコールを呼び出す際は eax にシステムコール番号の 0x21、ebx に第一引数の pathname ポインタ、ecx に第二引数の mode の値を格納する。検査対象のアドレスは、本来アクセス権をチェックするファイルのパス名を指す第一引数の pathname に格納される。本稿の提案方式では、この access システムコールを用いる。
- sigaction システムコール
  int sigaction (int signum,
  const struct sigaction *act,
  struct sigaction *oldact);
  システムコール番号は 0x43 であるので、sigaction システムコールを呼び出す際は eax にシステムコール番号の 0x43、ebx に第一引数の signum の値、ecx に第二引数の act ポインタ、edx に第三引数の oldact ポインタを格納する。検査対象のアドレスは、本来シグナル動作を設定する sigaction 構造体へのポインタ値が格納される第二引数の act に格納される。本稿の提案方式では sigaction システムコールを使用しないが、付録として Skape の sigaction システムコール使用した Linux 版 egghunter を改良したものを掲載する。

3.2 access を用いた egg hunter 解説

以下に Skape の egghunter を記載する。

```
00000000 <_start>:
  0: 31 d2              xor   edx,edx

00000002 <pageinc>:
  2: 66 81 ca ff 0f     or    dx,0xfff

00000007 <addrinc>:
  7: 42                 inc   edx
  8: 8d 5a 04           lea   ebx,[edx+0x4]
  b: 6a 21              push  0x21
  d: 58                 pop   eax
  e: cd 80              int   0x80
 10: 3c f2              cmp   al,0xf2
 12: 74 ee              je    2 <pageinc>
 14: b8 90 50 90 50     mov   eax,0x50905090
 19: 89 d7              mov   edi,edx
 1b: af                 scasd
 1c: 75 e9              jne   7 <addrinc>
 1e: af                 scasd
 1f: 75 e6              jne   7 <addrinc>
 21: ff e7              jmp   edi
```

まず、最初の 3 つの命令で edx の初期化とページの先頭アドレスの設定を行っている。

lea 命令では、access システムコールの第一引数への値が ebx に格納される。ebx にはこれから検査するアドレスが格納されていて、このアドレスが access システムコールの返却値によって有効か無効か判定される。

push、pop 命令では、access のシステムコール番号を eax に格納している。その後、int 0x80 でソフトウェア割り込みを行い access システムコールを呼び出している。EFAULT エラーが起こった場合の返却値は 0xffffff2f であり、この返却値は eax に格納される。

次の cmp 命令では、返却値が EFAULT エラーが起こったときのものかを調べている。この cmp 命令では、egghunter のサイズを小さくするため、返却値が EFAULT エラーの値と一致するかどうかの比較を eax の低位バイト al のみで行っている。比較の結果、現在調べているアドレスが無効なものであったら、現在のページは無効なページと判断し、次のページの先頭アドレスを取得する処理にジャンプする。

次に、システムコールで検査したアドレスが有効なものであったとき、egghunter はプロセスの仮想アドレス空間で shellcode 本体を見つけるために egg を探す。この例では tag は 0x50905090 である。現在検査しているアドレスから始まる 8 バイトに egg が格納されているか調べるために、eax に tag、edi に現在検査しているアドレスを格納する。scasd 命令は edi に格納されているアドレスからの 4 バイトと eax の値を比較する。これによって、eax に格納された tag の値と edi に格納された現在検査しているアドレスを比較できる。また、scasd 命令は比較の後 edi の値を 4 インクリメントする。したがって、2 回目の scasd 命令では、1 回目に比較した edi に格納されているアドレスを 4 インクリメントした値と eax の値を比較する。scasd 命令を 2 回行い、edi に格納されているアドレスから 8 バイトと egg の値が一致したとき egghunter は egg を発見できたことになり、egg の直後に格納されている shellcode 本体に処理を移す。scasd 命令で egg を発見できない場合は、次のページの先頭アドレスを取得する処理にジャンプする。

4. 提案方式

4.1 概要

本稿で提案する方式は、Skape の egghunter を改良し、処理速度を速くしたものである。Skape の egghunter と異なる点は以下の 2 つである。

- システムコールを呼び出すタイミングはページの先頭アドレスを検査するときのみ。

- 有効なページが複数連続して存在する場合 (単独で存在する場合も含む)、有効なページ群の先頭アドレスと末尾のアドレスを保存し、その有効なページ群の中で一気に egg の探査を行う。

提案する egghunter は以下のような構成を持つ。

- 仮想アドレス空間からページの先頭アドレスを取り出し以下の処理を行う。
  - ページの先頭アドレスが有効か無効かの検査
  - 有効なページ群の先頭アドレスと末尾のアドレスの保存
- 有効なページ群の中で、検査対象のアドレスから始まる 8 バイトが egg と一致するか検査する。
- 検査に失敗した場合は、有効なページ群の中で次のアドレスを取り直す。
- 有効なページ群の先頭アドレスまで検査を行なった場合は、次の有効なページ群の先頭アドレスと末尾のアドレスを取り直す。

4.2 解説

今回提案する egghunter は以下のようになっている。

```
00000000 <_start-0x2>:
  0:  31 db                xor   ebx,ebx

00000002 <_start>:
  2:  31 f6                xor   esi,esi

00000004 <pageinc>:
  4:  66 81 cb ff 0f       or    bx,0xfff
  9:  43                   inc   ebx

0000000a <syscall>:
  a:  31 c9                xor   ecx,ecx
  c:  6a 21                push  0x21
  e:  58                   pop   eax
  f:  cd 80                int   0x80
 11:  3c f2                cmp   al,0xf2
 13:  74 08                je    1d <addrend>
 15:  85 f6                test  esi,esi
 17:  75 eb                jne   4 <pageinc>

00000019 <addrstart>:
 19:  89 de                mov   esi,ebx
 1b:  eb e7                jmp   4 <pageinc>

0000001d <addrend>:
 1d:  85 f6                test  esi,esi
 1f:  74 e3                je    4 <pageinc>
 21:  8d 4b f8             lea   ecx,[ebx-0x8]

00000024 <addrdec>:
 24:  49                   dec   ecx
 25:  39 f1                cmp   ecx,esi
 27:  7c d9                jl    2 <_start>

00000029 <search>:
 29:  b8 90 50 90 50       mov   eax,0x50905090
 2e:  89 cf                mov   edi,ecx
 30:  af                   scasd
 31:  75 f1                jne   24 <addrdec>
 33:  af                   scasd
 34:  75 ee                jne   24 <addrdec>
 36:  ff e7                jmp   edi
```

この egghunter は大まかに 2 つのセクションに分かれている。プログラム先頭から addrend ラベル内の処理まででは、有効なページ群 (単独で存在する有効なページも含む) の先頭アドレスと末尾のアドレスの取得をしており、それ以降では取得したアドレスを使って有効なページ群の末尾のアドレスから先頭のアドレスまでに egg が無いか探査する。

まず、ebx、esi の初期化を行う。そして pageinc ラベルではページの先頭アドレスを取得する処理を行う。

次に、syscall ラベルでは、access システムコールの呼び出しを行い、ページの先頭アドレスが有効か無効の検査をする。ここでは ecx の初期化を行なっている。ecx の値は access システムコールの第二引数になるため初期化しなくてはならない。次にシステムコール呼び出しを行う。この手順は Skape のものと同様である。システムコールから EFAULT エラーが返ってきたとき、つまり検査しているアドレスが無効であった場合、このページが無効なページであると判定し、addrend ラベルへジャンプする。一方、検査しているアドレスが有効であった場合、このページは有効なページと判定する。ここで、esi は有効なページ群の一番最初のアドレスを格納するものとして利用する。そして、esi の値が 0、つまり esi に有効なページ群の先頭アドレスが格納されていないとき addrstart ラベルに進み、esi の値が 0 でない、つまり esi に有効なページ群の先頭アドレスが格納されているとき pageinc ラベルにジャンプする。

addrstart ラベルでは、有効なページ群の先頭アドレスを取得をしている。このとき ebx は有効なページの先頭アドレスを格納しているので、この値を esi にコピーしている。その後 pageinc ラベルにジャンプする。

addrend ラベルでは、有効なページ群の末尾のアドレスを取得している。まず、esi の値を調べ値が 0 であった場合、まだ有効なページに出会っていないということなので pageinc ラベルへジャンプする。

seach ラベルでは、scasd を使用して egg を探す。この処理は Skape のものと同様である。この提案方式でも tag は 0x50905090 としている。

5. 提案方式の評価

表 1 Skape の egghunter と提案方式の比較

	サイズ [byte]	実行時間 [s]	call 数
Skape	36	0.0590	48409.7
提案方式	56	0.0294	36544.1

テストプログラムを作成し、Skape の egghunter と提案方式の egghunter の比較を行なった。システムコール呼び出し回数、実行時間の項目についてはそれぞれのプログラムを 10 回実行し、その平均をとった値を比較している。また、shellcode 本体は実行されるとシステムコールの exit を呼び出すものにし、heap 領域に格納した。システムコールの exit を呼び出すものにした理由は、exit がこれを呼んだプロセスを直ちに終了させるというものであり、実行時間の比較がしやすいからである。また、テストプログラ

ム作成の際は gcc を用いて実行ファイルを作成した。gcc のオプションは以下のようにした。

```
$gcc -fno-stack-protector -z exestack -o
egghunter egghunter.c
```

5.1 頑健性

Skape の egghunter では、acccess システムコールを呼び出す前に ecx の初期化をしていない。ecx に格納されている値は access システムコールの第二引数に渡される値なので、初期化を行わないと ecx におかしな値が入って egghuunter が正常に動作しないことがあることを実験で確認した。したがって、Skape の access システムコールを用いた egghunter は少々頑健性に欠けている。ちなみに、Skape の egghunter に初期化の処理を追加すると、egghunter のサイズは 2 バイト増え、38 バイトになる。

それに対し、提案方式では access システムコールを呼び出す前に ecx の初期化を行っている。したがって、Skape の egghunter よりも頑健性があると言える。

5.2 バイト数

今回、Skape 方式が 36byte なのに対し、提案方式は 56byte になった。提案方式ではシステムコール呼び出し回数を削減することに重点を置いたので Skape のものよりも大きいサイズになってしまった。

5.3 システムコール呼び出し回数

システムコール呼び出し回数の計測には Linux の strace コマンドを使用した

測定結果は Skape 方式が 48409.7[回]、提案方式が 36544.1[回] となり、提案方式の方がシステムコール呼び出し回数が少ないことが確認できた。

5.4 実行時間

実行時間の計測には Linux の time コマンドを使用した。

測定結果は Skape 方式が 0.0590[s]、提案方式が 0.0296[s] となり、提案方式の方が速いことが確認できた。

このことから、システムコール呼び出し回数が少ない方が処理速度が速いということが言える。また、現在はサイズの小さいテストプログラムの中で egghunter を動作させているが、より大きいアプリケーションの中では更に速さの差がつくと考えられる。

5.5 executable な egg か

Skape の egg と同様に提案方式の egg も executable な egg である必要はない。これは Skape の egghunter と提案方式の egghunter が仮想アドレス空間から tag を使用して egg を探査する際に、scasd 命令を使用しているからである。scasd 命令では eax と edi の比較を行った後、edi の値が 4 バイトインクリメントれさる。そして、scasd を用いて 2 回比較を行うと edi は 8 バイトインクリメントされて shellcode 本体の先頭を指す。2 回の scasd 命令で仮想アドレス空間から egg を発見出来た後、egghunter は jmp edi 命令を実行するが、このとき edi は shellcode 本体の先頭を指しているので実行を shellcode 本体に移すことができる。したがって、egg は executable である必要はない。

6. おわりに

本稿では、Skape の access システムコールを用いた Linux 版 egghunter を改良し、システムコール呼び出し回数を削減して処理速度を向上させた egghunter を提案した。まず、ページの中に 1 つでも有効なアドレスがあれば、そのページの全アドレスは有効という仮定を立て、この仮定の妥当性を実験を行い検証した。その結果、この仮定は妥当なものであることがわかったので、この仮定に基づき提案方式を構成した。そして、実際に Skape の egghunter と提案方式の egghunter を比較し、システムコール呼び出し回数を削減することで、実行時間を短くすることができることを示した。

参考文献

[1] Matt Miller(Skape) : Safely Searching Process Virtual Address Space(2004)
[2] Daniel P.Bovet,Marco Cesati : 詳解 Linux カーネル第3版 (2007)
[3] Robert Love : Linux システムプログラミング (2008)

付　　録

A.1 sigaction システムコールを用いた改良版 egghunter

ここでは、付録として Skape の sigaction システムコールを用いた egghunter を改良したプログラムを掲載する。このプログラムも今回の提案方式と同様にシステムコールの呼び出し回数を削減することで処理速度を向上させている。

しかし、Skape の sigaction システムコールを用いた egghunter を改良したこの方式は、提案方式の access システムコールを用いた egghunter よりも実行時間が長くなる。sigaction システムコールを用いて、仮想アドレス空間からページの先頭アドレスを取り出してアドレスが有効かどうか検査する際、このアドレスは sigaction システムコールの第二引数に渡される。sigaction システムコールは、第二引数に渡されたアドレスから 16 バイトの検査をまとめて行う。しかし、今回の提案方式で、システムコールを用いたアドレス有効･無効の検査をする必要があるアドレスはページの先頭アドレスだけであり、sigaction システムコールを用いてページ内の 16 バイトもの検査をする必要は無い。したがって、実行時間も access システムコールを用いた方式の方が sigaction システムコールを用いた方式よりも速くなる。

また、sigaction システムコールを用いた egghunter には頑健性においても劣る点がある。それは、sigaction システムコールが 0x00000000 から 0x00000fff までの 1 ページを検査できない点である。これは sigaction システムコール自体の仕様であるので egghunter の開発者側で改善することができない。

```
00000000 <_start-0x2>:
  0: 31 c9              xor   ecx,ecx

00000002 <_start>:
  2: 31 f6              xor   esi,esi

00000004 <pageinc>:
  4: 66 81 c9 ff 0f     or    cx,0xfff
  9: 41                 inc   ecx

0000000a <syscall>:
  a: 31 db              xor   ebx,ebx
  c: 6a 43              push  0x43
  e: 58                 pop   eax
  f: cd 80              int   0x80
 11: 3c f2              cmp   al,0xf2
 13: 74 08              je    1d <addrend>
 15: 85 f6              test  esi,esi
 17: 75 eb              jne   4 <pageinc>

00000019 <addrstart>:
 19: 89 ce              mov   esi,ecx
 1b: eb e7              jmp   4 <pageinc>

0000001d <addrend>:
 1d: 85 f6              test  esi,esi
 1f: 74 e3              je    4 <pageinc>
 21: 8d 59 f8           lea   ebx,[ecx-0x8]

00000024 <addrdec>:
 24: 4b                 dec   ebx
 25: 39 f3              cmp   ebx,esi
 27: 7c d9              jl    2 <_start>

00000029 <search>:
 29: b8 90 50 90 50     mov   eax,0x50905090
 2e: 89 df              mov   edi,ebx
 30: af                 scad
 31: 75 f1              jne   24 <addrdec>
 33: af                 scasd
 34: 75 ee              jne   24 <addrdec>
 36: ff e7              jmp   edi
```