SCIS 2014 The 31st Symposium on
Cryptography and Information Security
Kagoshima, Japan, Jan. 21- 24, 2014
The Institute of Electronics,
Information and Communication Engineers

攻撃コードの特徴からみた対策の検討

田中 恭之† 後藤 厚宏†

†情報セキュリティ大学院大学
221-0835 神奈川県横浜市神奈川区鶴屋町 2-14-1
†{mgs135505,goto}@iisec.ac.jp

概要: 水飲み場型攻撃と呼ばれる標的型攻撃が昨今行われ脅威となっている．一因として一般に入手可能な攻撃コード生成ツールを利用することで攻撃が容易に実行可能である点がある．攻撃コードはいくつかの要素から成り立つが，本稿では，攻撃コードの中でも特にシェルコードに着目した．攻撃コード実行のメカニズムを説明し，シェルコードの要件から見た特徴を整理した．関連研究の比較考察を行い，従来方式で対応が困難であった ROP と呼ばれる攻撃手法をネットワーク上で検出する方式を提案し評価を行った．

Study of Countermeasures based on the Characteristic of Recent Attack Code

Abstract: It has become a threat targeted attacks called Watering Hole Attack is carried out these days. One reason is that available attack tools in general are very powerful to make these attacks. As attack codes consists of several elements, We are focusing on the Shell-Code in this paper. We've explained execution mechanism of attack code and classified the characteristic of the Shell-Code. We've compared characteristic of the Shell-Code with methods of related works. And We've proposed and evaluated a method for detecting network attacks on a technique called ROP that has been difficult in the conventional method.

1. はじめに

脆弱性修正パッチがリリースされていない状態のゼロデイ脆弱性は未だ減ることはなく毎年発生している. 特に 2013/9/17 に発表された InternetExplorerVer6-11 のゼロデイ脆弱性[1]は，水飲み場型攻撃と呼ばれる標的型攻撃に利用され，日本のある情報提供サイトに 2013/8 上旬から仕掛けられていた(発見は 8 月下旬)．標的とされた特定の IP アドレス帯域(国内の中央官庁や重要インフラなど 20 程度の組織)のみアクセス可能かつ特定の OS やブラウザバージョンのみ攻撃を発動する仕掛けとなっており, 第三者には気づかれにくく，確実に標的をとらえられるように工夫されていた[2].

また，このゼロデイ脆弱性は，2013/10/9 に修正パッチが公開された[3]が，無償利用できる攻撃ツールである Metasploit Framework[4]の攻撃コードが 2013/9/30 にリリースされ，技術の無い人間でもこの悪質な水飲み場を作りゼロデイ攻撃を成功させることが可能となっていたのも問題である．Metasploit Framework では，各アプリケーションの脆弱性毎に攻撃コードをリリースしており，多数の攻撃コードが入手できる．

本稿では，攻撃コードの中でも，特にシェルコードに着目した．定義は後述するが，エクスプロイトコードは脆弱性のバリエーション分存在し，マルウェアについては亜種が次々に作成される一方，シェルコードは Metasploit Framework 等からリリースされる物含め世の中に流通する種類が少ないため攻撃をとらえるのに有利ではないかと考えたためである．流通量が少ない点については本稿で解説するシェルコードの要件や特徴に起因すると考えられる．仮にシェルコードを特定できればこのようなゼロデイ検出にも効果を発揮できると考えた．まず攻撃コードのメカニズムを理解する上で種々の関連知識を説明し，シェルコードの特徴を整理した上で, 関連研究で提案された方式において, どの特徴を検出することが可能か机上検討による比較を行った．また比較結果，従来方式で対応が困難であった ROP と呼ばれる攻撃を NW 上で検出する手法を提案し評価を行った.

2. 攻撃コードと実行メカニズム

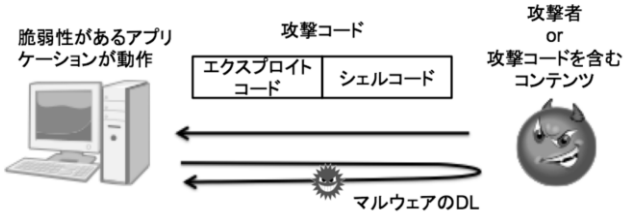

図 1 攻撃コード

2.1 攻撃コード

図 1 のように，攻撃コードの中身は，一般に，エクスプロイトコード（エクスプロイトとも呼ばれる）及びシェルコード（ペイロードとも呼ばれる）から構成

される．エクスプロイトコードは，脆弱性のあるアプリケーションから実行権をコントロールできるようにするまでの役割を担う．実行権が取られると，シェルコードが実行される．多くのシェルコードは攻撃の拡大を目的として次のプログラム（主にマルウェア）のダウンロード（及び実行）を行う．

本稿執筆時では，Metasploit Framework から，エクスプロイトコードが 1201 個，Windows 向けのシェルコードとして 129 個リリースされている．

2.2 攻撃コード実行のメカニズム

攻撃コード実行のメカニズムを見る．ここでは一番単純な例として，スタックバッファオーバーフローによるシェルコードの実行を説明する．図 2 にスタックバッファオーバーフロー脆弱性のあるサンプルプログラムとスタックの状態を示す．

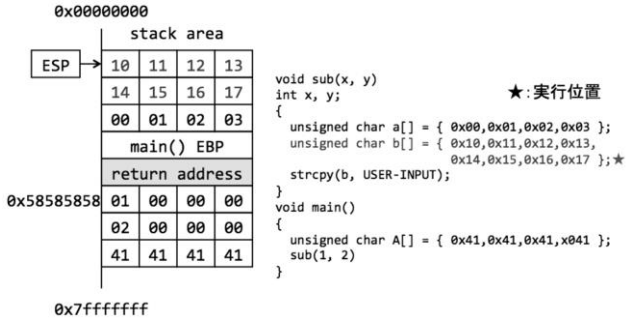

図 2 スタックの解説①

main()関数から引数を 2 個とる sub()関数が呼ばれる簡単なプログラムになっている．図 2 で現在の実行位置は★であり，変数 b[]に値が格納された状態である．このときスタック領域には，アドレスの高位から，main()関数で格納された変数 A[]，sub()関数への引数，main()関数に戻るためのリターンアドレス，sub()関数が呼ばれるときの EBP レジスタの値が積まれている．さらに sub()関数での変数 a[]及び b[]が積まれている．

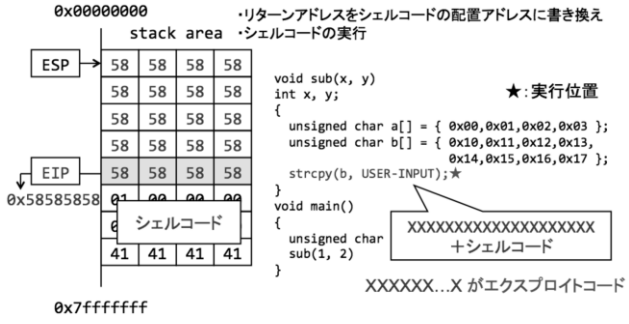

図 3 スタックの解説②

次に，プログラムを 1 ステップ進めた状態が図 3 である．strcpy 関数でユーザからの入力文字列を変数 b[]に格納を終えた状態のスタック領域を表している．変数 b[]は 8 バイトの配列として定義されているが，strcpy 関数はそれを意識せずコピーを行うため問題が発生する．入力文字列として，X（文字コード 0x58）を 20 文字＋任意の文字列（シェルコードに相当）とした場合のスタック状態を示す．main()関数へのリターンアドレスが 0x58585858 に書き換えられ，次の実行アドレスを示す EIP レジスタの値を 0x58585858 に書き換えることに成功している．同アドレスにはシェルコードが存在するため，リターン時にシェルコードが実行されることがわかる．

3. シェルコードの要件から見た特徴

シェルコードに求められる要件から特徴を見ていく．一般にシェルコードには以下の特徴がある．

特徴① API 関数アドレスの自己解決

通常の Windows 実行形式の exe ファイルは PE(Portable Executable)形式を取っており IAT(Import Address Table)に外部関数のアドレスが登録されるため意識する必要はないが，シェルコードは PE 形式ではなく単なるバイト列であるため，呼び出したい関数アドレスを自己解決する必要がある．これはいくつかのステップを踏むこととなるが，まず kernel32.dll のアドレスを解決することが多い．これは kernel32.dll のアドレスがわかれば，kernel32.dll 内の LoadLibrary 関数と GetProcAddress 関数を使ってすべての関数にアクセス可能となるからである．kernel32.dll のアドレスを知るには以下のようなアプローチがある．

- Process Environment Block(PEB)を調べる
- Thread Environment Block(TEB)を調べる
- Structured Exception Handling(SEH)を用いて調査する

ここでは Process Environment Block(PEB)を調べる方法の詳細の一例を説明する．PEB は Windows 上のすべてのプロセスそれぞれに割り当てられるデータ構造体であり，そのプロセスがロードするモジュールの情報等が格納されている．

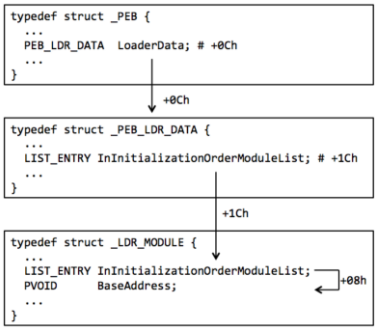

図 4 各構造体の関係

図 4 に示すように，PEB データ構造体から PEB_LDR_DATA 構造体をたどることで最終的に kernel32.dll の baseaddress を見つけることができる．

また，このとき対応するコードを図 5 に示す．Windows では仕様として fs:0x30 に PEB アドレスが格納されている．ここからの先ほど見た各構造体の関係を考慮して相対アドレスを求めていく．lodsd 命令はア

ドレス ds:esi のダブルワードを eax にロードする命令である．最終的にレジスタ ebx に kernel32.dll の baseaddress が格納される．そのため多くのシェルコードは冒頭に図 5 に示すコードを含む特徴が現れる．

```
mov eax, fs:[ebx+0x30] ; eax = PEB
mov eax, [eax+0x0C]    ; eax = PEB.LoaderData
mov esi, [eax+0x1C]    ; esi = InInitializationOrder
                             ModuleList.Flink
lodsd                  ; eax = kernel32.dll's
                             LDR_MODULE entrypointer
mov eax, [eax+0x08]    ; eax = BaseAddress
mov ebx, eax           ; ebx = kernel32.dllbaseaddr
```

図 5 kernel32.dll アドレスを求めるコード

その他のアプローチとして，直接固定の関数アドレスを呼び出すというものがあるが，Windows のサービスパックや言語が異なると別アドレスとなるため攻撃コードとしては安定性に欠ける．

特徴② シェルコード起動の確実さ

攻撃者は，シェルコードを確実に起動させるために以下の特徴を持たせることがある．

- NOP スレッドを持つ
  大量の NOP 命令（何も動作しないという命令）の後にシェルコードを配置することを NOP スレッドとよぶ. エクスプロイトコードで正確に EIP レジスタをコントロールできなくとも NOP 命令群に EIP レジスタを制御できれば NOP コードを実行後シェルコードにたどり着くことができる．
- 禁則文字の考慮
  前述した strcpy 関数の例だと, エクスプロイトコード及びシェルコードの中に，0x00 が入ると strcpy 関数が文字列終端だと認識しコピーが終了となる．また例えば eax レジスタを 0 にしたい場合, 対応するアセンブリ(Intel 形式)と機械語は以下のようになる．

  ```
  mov eax,0      B8 00 00 00 00
  ```

  0x00 が入ってしまうので,

  ```
  xor eax,eax    33 c0
  ```

  のように表現することで，同じ動作を 0x00 無しで可能とする. また機械語サイズの削減効果を目的として同等の工夫がなされる場合もある．

特徴③ サイズを小さく

攻撃コード実行のメカニズムで見たように，攻撃に用いられる脆弱性のプログラム構造に起因したメモリの状態に依存して，シェルコードが配置できるサイズが決まる．汎用性を考えるとできる限り小さい方が望ましい．そのために攻撃者は以下のようなテクニックが用いることがある．

- Stager 及び Stage 構成を取る
  最初に動作する極小コード Stager が，Stage コードのための領域確保と Stage コードの書き込みを行う．次に Stage コードが Stage コードから呼び出され起動する．
- Egg 及び Egg Hunter[5], Omelette 構成を取る
  最終的に実行したいシェルコードに目印(Egg)をつけ，Egg Hunter コードがメモリ内を操作して，シェルコードを発見しシェルコードに動作を移す．Omelette はその発展版であり，Egg が複数になる等の工夫をしている．

特にこれらのテクニックは，使用できるバッファが小さく，また分断されているときに有効である．

特徴④ 難読化コードを用いた検出回避

一般に IDS/IPS はネットワーク上を流れるパケットに対し，シグネチャを用いたパターンマッチにより検出を行う．攻撃者はこれを回避するため難読化コード（メタモーフィックコード）を用いることがある．特徴②でも見たように，最終的に同じ動作をするコードでも様々な記述が可能である．これによりシグネチャを回避する．また，コードの解析自体を困難にさせることを目的として実施されることもある．他の難読化手法として，レジスタの値にジャンプする間接ジャンプという手法があり，実行してみないとジャンプ先がわからない．

特徴⑤ 暗号化コードを用いた検出回避

類似の手法として，暗号化コード（ポリモーフィックコード）がある．本来のシェルコードを暗号化し，複合化するデコーダルーチンを暗号化コードに付加する方式である．デコードルーチンを実際に実施しない限り，本来のシェルコードのバイト列はわからずシグニチャ検知ができない. また, 復号化をするためには, 暗号化コード位置の絶対アドレスが必要になるが，アドレスを算出するために GetPC コードと呼ばれるコードを呼ぶことで算出する手法が多く取られる．

特徴⑥ DEP・ASLR の回避

攻撃者は DEP や ASLR を回避するために Code-Reuse 攻撃の一種である ROP というテクニックを用いる．

- Retun-to-libc による DEP 回避
  DEP(Data Execution Prevention)とは，データ領域内での実行を阻止するもので，WindowsXP の場合，SP2 で導入された．DEP 有効下では，これまで見たようにデータ領域であるスタック領域に配置したシェルコードは実行権が無く実行できない. これに対抗して, Return-to-libc というテクニックが用いられるようになった. これはメモリ中に存在する API 関数を直接コールする手法で, API 関数への引数を適切にスタックに積んでコールすることで API 関数を動作させ目的を達成することができる．
- ROP による DEP 回避
  さらにこのテクニックを進めた ROP(Return Oriented Programming)[6]というテクニックが最近主流となっている．ROP コードの一例を図 6 に示

す．実行可能領域にある ROPgadget と呼ばれる ret 命令で終了する数バイトのコード断片を利用する．この例では，最終的にアドレス 0x50505050 に値 0xDEADBEEF を格納することを実現している．まず攻撃者は目的が達成できるように適切にスタックを積んでおく．スタックの最上位の値にリターンするように調整し，最初に gadget No1 が実行される．レジスタ EAX には意図した値(0x50505050)が格納され，ret 命令でスタックを参照し，次の gadget No2 にリターンする形で gadget No2 が実行される．このようにして，最終的に gadget No3 が実行されることで目的が達成できる．

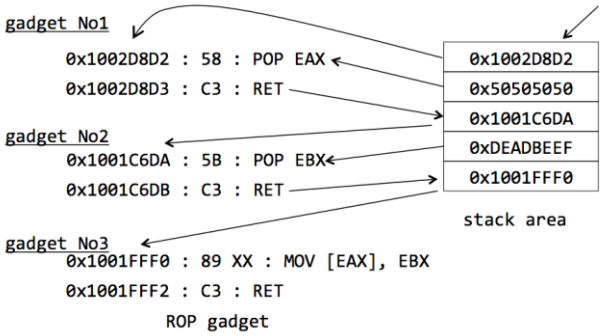

図 6 ROP コードの動作原理

ROP を進めたテクニックとして JOP(Jump Oriented Programming)[7]が提案されている．ret の代わりに jmp を用いる物だが，gadget をコントロールする Dispatcher を用意し，jmp 先をこの Dispatcher にするように工夫している．また SOP(String Oriented Programming)[8]はフォーマットストリング脆弱性を利用するテクニックである．ただ本稿執筆時では JOP や SOP については具体的な攻撃コードを目にすることは無かった．尚 ROP,JOP,SOP は Code-Reuse 攻撃とも呼ばれる．

- Code-Reuse 攻撃による ASLR 回避

次に ASLR(Address Space Layout Randomization)について説明する．Return-to-libc や ROP で見たように，攻撃者が関数やスタック・ヒープの固定な既知アドレスを利用することに対抗して，アドレスを Windows 起動時毎にランダマイズする方式である．これは WindowsVista 以降であり，XP では実装は無い．ASLR 回避策としては，32bitOS であればブルートフォースが現実時間で可能であることと，dll によってはランダムかが行われない dll が存在しそれを狙うという手法がよく用いられる．冒頭で示した IE のゼロデイに対応した Metasploit 攻撃コードも ASLR を回避するために，MicrosoftOffice が提供する dll 内のコードを利用して ROP コードを構成している．同攻撃コードは Windows7 ベースであり Windows7 の標準 dll はたいてい ASLR がかかっているが，MicrosoftOffice の dll の一部で ASLR が有効な状態でコンパイルされていなものがあり，該当する hxds.dll を用い巧みに利用して攻撃を成功させている[9]．

特徴⑦ 最終目標：マルウェアのダウンロード

シェルコードという名称は歴史的に UNIX の sh を取得して操作権を奪うというところから来ているが，近年一般には，マルウェアをダウンロードして実行することを最終目的とされることが多い．そのため表 1 のような特定の API がコールされることが多い．

表 1 マルウェアダウンロードに用いられる関数名

関数名
LoadLibrary
GetProcAddress
URLDownloadToFileA
CreateProcess

特に URLDownloadToFileA はソケットを開く処理を行うよりもシンプルに目的を達成できるので好まれる．

4. 関連研究 1

ネットワーク側でのシェルコード検出の分野で関連研究の調査を行った．攻撃コードの検出は IDS のようにネットワーク側で行う手法とアンチウイルスソフトのようにホスト側で行う手法があるが，ネットワーク側の方が，一般に実運用を踏まえると導入障壁が低いと考えられるためである．例えば企業等で複数のクライアント端末を管理運用するケースで望まれると考えられる．また，各研究における検出方式とシェルコートの対応についての考察を行った．

4.1 静的解析による検出

T. Toth らは，図 7（左）のように特徴文字列として NOP スレッドをパターンマッチにて検出することで効率的にシェルコードの検出を行っている[10]．しかしその方式から難読化や暗号化には対応できない．

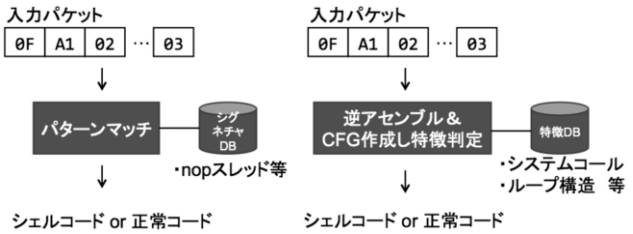

図 7 左:特徴文字列 右:CFG 作成

また，R. Chinchani らは，図 7（右）のようにバイト列を機械語とみなし逆アセンブルし制御フローグラフを構成し，システムコールやループ構造等既知の特徴にマッチするかで判定している[11]．性能向上と高度な難読化への対応が課題となっている．

4.2 動的解析による検出

M. Polychronakis らは，図 8 のようにエミュレータで実際に実行を行い，既知の挙動にマッチするかで判断を行う[12]．実際に実行するので，高度な難読化や暗号化が施されていても対応することができる．特徴

の例として，PEB アドレスを用いた kernel32.dll の baseaddress の解決をシェルコード判定の規準としている．関連知識の説明で示したように，PEB アドレスを知るためには一定のアプローチが見られたが，その特徴をとらえる形を取っている．

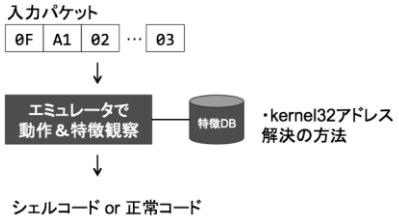

図 8 エミュレータで動作

- 条件 1 : fs:0x30 のメモリ参照，かつ，直近で fs セグメントレジスタ操作を伴う命令の出現
- 条件 2：PEB_LDR_DATA の参照
- 条件 3：
  InLoadOrderModuleList or InMemoryOrderModuleList or InInitializationOrderModuleList の参照

上記の 3 つの条件がすべて発生した場合にシェルコードと判定している. 本方式は非常に False positives が少ない評価結果となっているが，条件 1 に条件 2,3 を AND 条件として加えていることが効果を出している．

ただ，本方式では，ネットワーク上のパケットストリームで，どこを起点にしてエミュレータで動作させればよいかは不明のため，1 バイト毎にすべてのバイトを開始位置として実行を行う必要があり，性能が極端に悪くなる．また ROP のような Code-Reuse を使うコードの場合，単純なエミュレータでは完全にホストのメモリ状態まで持ち合わせないので対応することができない．

動的解析の関連研究として，神保らは，動的解析結果に基づき，シェルコードをエンコード方式や使用 API 等の観点での自動分類を試みている[13].

4.3 検出技術の考察

ネットワーク側が実装場所となる関連研究[10,11,12]で提案されている方式が，前述した特徴①から特徴⑦を受けたシェルコードの特徴のどれに対応可能かについて表 2 にまとめた．

表 2 シェルコードの特徴と対策方式の対応

実装場所	ネットワーク側		
方式	静的解析		動的解析
詳細方式	特徴文字列[10]	制御フロー解析[11]	エミュレータ[12]
特徴①	(④⑤無なら○)	(⑤無なら○)	○
特徴②	○	○	○
特徴③	×	(△)	○
特徴④	×	△	○
特徴⑤	×	×	○
特徴⑥	×	×	×
特徴⑦	(④⑤無なら○)	(⑤無なら○)	○
性能	○	△	×

表で（）表記としているものは予測値である．つまり該当論文中で評価はされていないが，その方式から推測するにこうなるであろうという値を入れている．表で「(④⑤無なら○)」の表記は，特徴④難読化コードを用いた検出回避及び特徴⑤暗号化コードを用いた検出回避がされていなければ該当方式でも検出できることを示し，() をつけているので，当方による推測である. 表に示す通り, エミュレータを使う方式[12]は，性能が大きな課題となるが，特徴⑥DEP・ASLR の回避，以外の特徴には対応可能となっている．パターンマッチによる方式[10]は特徴④⑤が障壁となるが性能は良いので，他で得た知見をうまく生かすことで発展させられる可能性がある．[11]については，特徴③サイズを小さく，で説明した Egg Hunter コードにも対応可能と考えられる．特徴⑥が，いずれの方式でも対応が不可である．これは，ROP に代表される Code-Reuse 攻撃はホスト側のメモリの状態に依存するため，ホスト側のメモリ状態を持ち合わせないこれらの方法では検出ができない.

5. 関連研究２

攻撃トレンドとして焦点となっている ROP 対策と Metasploit Framework 対策について最新の研究動向を調査した.

5.1 ROP 対策

これまで見てきたように，攻撃コードの検出での鍵となっているのが ROP 対策である. ここでは本稿執筆時の ROP 対策における最新動向を説明する．ROP の動きは関連知識の整理で説明したが，ROP を防ぐには関数が呼ばれてリターンした場合に適切に呼び元のアドレスの次のアドレスに戻っているかをチェックすれば良い．これは従来から提案されている方式で，L. Davi らの ROPdefender では，shadow stack という領域を本来の stack 領域とは別にもうける．この領域を用いリターンアドレスを記録し適切にリターンしているかをチェックする[15]．この方式で ROP の検出は可能となるが，shadow stack の管理のオーバヘッドが大きくなり性能に影響が出る.

そこで V. Pappas らの kBouncer では，最近の Intel プロセッサで提供される機能である LBR（Last Branch Recording）を用い ROP 検出を実現している[16]．LBR には過去 16 回分と限られはするが直近の分岐情報が記録されている．これの利点は LBR は CPU が提供するレジスタであるため，この記録にかかる性能劣化はゼロと見なすことができる点である. この LBR の履歴を活用し ROP を検出する．具体的には，攻撃者は最終的に API コールやシステムコールを目的とするのでその時点で過去の分岐履歴をみて正常なコードか ROP コードかを判定する. 16 回という短い履歴でどうやって検出するかが焦点となるが，ROP による攻撃が

どのように行われるかを V. Pappas らは詳細に調べており，API コール，システムコールを目的とした ROP gadget は通常 10 個程度であるため問題ないと結論づけている．

5.2 Metasploit Framework 対策

冒頭で述べたように Metasploit Framework はセキュリティ研究者にとってすばらしいツールであるが，一方で悪用される脅威も無視できない状況である．ここで Metasploit Framework を悪用した攻撃を正確に検出することを目的とした本稿執筆時で最新の研究事例を紹介する．R. Wang らは MetaSymploit というシステムで, Metasploit Framework で提供される攻撃スクリプト群をインプットとして，それらの実行により行われた攻撃を正確に検出する IDS 用シグネチャを自動生成するものである[17]．MetaSymploit は Symbolic Execution Layer と Signature Generator という 2 つの機能部に分かれており，実際の攻撃コードの解析結果，攻撃コード実行時のログ，攻撃パケットを利用し，シグネチャの自動作成を可能としている．

6. 着眼点

関連研究 1,2 で示したように従来研究では NW 側での ROP 対策の検討は進んでいない．そこで ROP コードに着目し攻撃コードを分析すると，ある特徴があることがわかった．図 9 でこの点を説明する．2 章で示したように，攻撃コードはエクスプロイトコード＋シェルコードで構成され，図 9 の「通常の攻撃コード」のように構成される．3 章でみた特徴①から⑦のうち，特徴④難読化が施されたコードは「難読化攻撃コード」のように構成される．特徴④及び特徴⑤暗号化が施されたコードは，「暗号化・難読化攻撃コード」に示すように，復号コード＋暗号化された元の難読化シェルコードで構成される．次に DEP や ASLR 等の新たな防御策を回避するために付加される，特徴③の EggHunter コード及び特徴⑥の ROP コードを，pre コードと定義する．

図 9 pre コードの定義

この pre コードは，攻撃コード構成の都合上，図 9 の「ROP や EggHunter コード」に示すように，シェルコード復号ロジックの外側に配置する必要があり，⑤の暗号化の対象とできない．また④の難読化の影響も受けないため特徴をつかむのに好都合なのではないかと考えた．

6.1 ROP コードの目的と詳細

ROP コードは，前述したとおり実行権限のあるコード部分を繋いで利用するテクニックであるが，最終的な目的は DEP 回避でありシェルコード配置部分に対して実行権限を付与することである．実行権限を付与可能な API 関数を表 3 に列挙する．

表 3 実行権限付与 API 関数

関数名	
VirtualProtect	SetProcessDEPPolicy
VirtualAlloc	WriteProcessMemory
HeapCreate	NtSetInformationProcess

また VirtualProtect 構造体について表 4 及び図 10 に示す．

表 4 VirtualProtect 構造体

名前	説明
lpAddress	アクセス権限を変えたいページ領域のアドレス
dwSize	領域のサイズ
flNewProtect	アクセス権限．実行権を付与するには 0x40(PAGE_EXECUTE_READWRITE)

```
BOOL WINAPI VirtualProtect(
  __in LPVOID lpAddress,
  __in SIZE_T dwSize,
  __in DWORD flNewProtect,
  __out PDWORD lpflOldProtect );
```

図 10 VirtualProtect 構造体

6.2 EggHunter コードの目的と詳細

EggHunter コードの目的は前述した通りシェルコードを発見することであるが, [5]によると Windows の例外処理機構である SEH(Structured Exception Handling)を用いる手法や特定の関数を用いメモリサーチの効率化を図る手法がある．前者は SEH の悪用に対する保護機能である SafeSEH で抑止されてしまうことから最近用いられる傾向が少なく，後者の手法が多く用いられる．表 5 の関数を利用して，ページ単位でメモリをチェックし，アクセス違反であれば次のセグメントに移る方式である．

表 5 EggHunter で用いられる API 関数

関数名
IsBadReadPrt
NtDisplayString
NtAccessCheck

7. 提案方式

NW 上に流れるトラフィックの特徴から，pre コードである ROP コードや EggHunter コードを検出し，未知の攻撃コードをとらえることを目的とする．

7.1 NW 上での ROP コードの検出

NW 上での ROP コードの流れを図 11 に示す．3 章で示したように ROP コードはその目的から DEP を制御する API

関数の物理アドレスを指すROPgadgetコードをコールする．これを特徴文字列①と定義する．またROPgadgetコードを複数用い，この DEP 制御 API 関数への適切な引数等を準備する．この ROPgadget コードを特徴文字列②と定義する．ここで図 11 の□は 1byte を示し，特徴文字列①及び②は32bit環境の場合 4byte の物理メモリアドレスである．このとき特徴文字列①は特徴文字列②及び関数への引数等の値に挟まれるような形でトラフィック上に出現する．

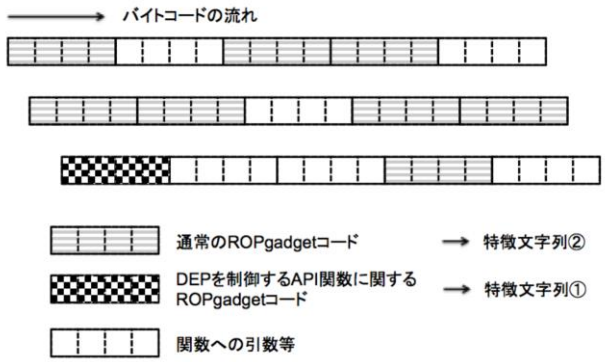

図 11 トラフィック上のコード

特徴文字列①は次のように抽出した．1 章や 5 章で示した通り Metasploit Framework の攻撃コードが脅威となっていることから同ツールを対象とし，今回はブラウザへの攻撃回避を主眼としたため，windows:browser エクスプロイトの攻撃コード総計 221 個から表 3 で示した 6 つの関数を呼ぶ物理アドレスを取り出しそれらを特徴文字列①とした．結果を表 6 に示す．物理アドレスは 10 アドレス抽出された．また，対象関数は VirtualProtect と VirtualAlloc の 2 つのみであった．一部非常に古く，マイナーであると判断できるアプリケーション dll からの少数の物理アドレスは効果が薄いと考え除外した．また，該当した攻撃コードの数を exploit 数として示した．1つの攻撃コード内で複数の物理アドレスが用いられるケースがあったため表 6 での exploit 数合計は 31 個であるが実際のユニーク攻撃コード数は 22 個であった．

表 6 特徴文字列①

アドレス	関数名	DLL名	アプリケーション	exploit数
0x7c37a151	VirtualProtect	msvcr71.dll	JRE1.6	10
0x7c37a140	VirtualProtect	msvcr71.dll	JRE1.6	4
0x77c11120	VirtualProtect	msvcrt.dll	XP sp3	2
0x77ba1114	VirtualProtect	msvcrt.dll	XP sp3	2
0x7c801ad4	VirtualProtect	xul.dll	FF3.6	1
0x77c1110c	VirtualAlloc	mscvrt.dll	XP sp3	7
0x781a909c	VirtualAlloc	mozcrt19.dll	FF7,8,9	2
0x51bd115c	VirtualAlloc	hxds.dll	office2007	1
0x51bd10bc	VirtualAlloc	hxds.dll	office2010	1
0x1083828c	VirtualAlloc	xul.dll	FF3.6	1

次に特徴文字列②は次のように抽出した．同じく Metasploit Framework の windows:browser エクスプロイトの攻撃コード総計 221 個から ROPgadget となるすべてのアドレスから特徴文字列①を除外したものを取り出した．合計 500 個程度の物理アドレスであった．

これらの特徴文字列を利用して ROP コードを検出する方式例を図 12 に示す．特徴文字列②が特徴文字列①の前後に連続して出現することでスコア値を上げていき，特徴文字列①が出現しかつスコア値が一定の閾値以上の際に検出と判断する．特徴文字列①のみの場合 False positives がおこる可能性があるので，特徴文字列②を併用し精度を向上させる．

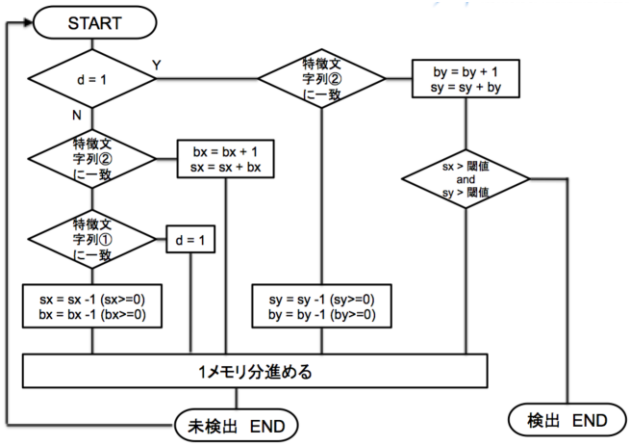

図 12 フローチャート

ブラウザに対する攻撃の場合，エンコードに注意する必要がある．今回見た一部の攻撃コードでは表 7 に示すようにHTML 上で用いるためにバイト列を%u エンコードしていた．ネットワーク上で検出する場合これらを考慮する必要がある．

表 7 エンコード

コード種別	形式
元のバイトコード	66 4C 36 7C
HTML 上	%u367C%u664c
ネットワーク上のバイトコード	25 75 33 36 37 63 25 75 36 36 34 63

7.2 NW 上での EggHunter コードの検出

ROP コードと同様に Metasploit Framework から特徴文字列を抽出した．すべてのエクスプロイトコード（現時点で 1201 個）を対象に EggHunter が使われているコードを確認したところ 52 個が該当し，すべてが NtAccessCheck 関数を用いる物であった．引数に 2 を指定し 0x2e でシステムコールを行う．図 13 のバイト列（左の列）を特徴文字列として利用する．

```
6A02  push 0x02
58    pop eax
CD2E  int 0x2e
3C05  cmp al,0x05
5A    pop edx
```

図 13 EggHunter コード

8. 評価

8.1 NW 上での ROP コードの検出

Metasploit Framework から該当する 7 章で示した 22 個

の攻撃コードを対象とし以下の方法で評価した.

- ソースコードチェックによる確認
  全 22 個すべてで特徴文字列を用いて攻撃コードが組み立てられることを確認.
- パケットキャプチャによる確認
  全22個から3個抜き出し攻撃コードが動作する環境でパケットキャプチャを実施し確認. 3 個すべてで特徴文字列を確認.

8.2 NW 上での EggHunter コードの検出

同様に該当 52 個の攻撃コードを以下の方法により評価した.

- ソースコードチェックによる確認
  全 52 個すべてで特徴文字列を用いて攻撃コードが組み立てられることを確認.
- パケットキャプチャによる確認
  全52個から3個抜き出し攻撃コードが動作する環境でパケットキャプチャを実施し確認. 3 個すべてで特徴文字列を確認.

9. 制限事項

提案方式はブラウザに限らず Windows ベースの攻撃コードの検出を可能とするものであり, シェルコード自体の暗号化, 難読化に対して影響を受けないが, 特にブラウザへの攻撃では, 攻撃コード全体を Javascript で難読化するケースがあり, これには対抗できない. また https 等の暗号化も同様である.

10. まとめ

本稿では，ゼロデイ対策を最終目的として攻撃コードの中でシェルコードに着目した．まず，攻撃コードや種々の関連知識を説明し，シェルコードの特徴を 7 個に整理した上で，関連研究で提案された方式において，どの特徴を検出することが可能か机上検討による比較と考察を行った. その中で技術的課題となる, ROP 対策や Metasploit Framework 対策における最新の研究動向を説明した. さらに ROP コードや EggHunter コードの特徴に着目し従来方式では NW 上での検出が困難であった同コードを検出する方式を提案し評価を行った．今後は制約事項のインパクトを考察し本方式の実装評価もしくは適用領域の再検討を行う.

参考文献

[1] http://www.ipa.go.jp/security/ciadr/vul/20130918-ms.html (2013/12/1)

[2] http://internet.watch.impress.co.jp/docs/news/20131010_618941.html (2013/12/1)

[3] https://technet.microsoft.com/ja-jp/security/bulletin/ms13-080 (2013/12/1)

[4] http://www.metasploit.com (2013/12/1)

[5] Skape. Safely searching process virtual address space, 2004. http://www.hick.org/code/skape/papers/

[6] H. Shacham. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In Proceedings of the 14th ACM conference on Computer and Communications Security (CCS), 2007.

[7] Tyler Bletsch, Xuxian Jiang, and Vince Freeh, “Jump-Oriented Programming: A New Class of Code-Reuse Attack,” in Proceeding ASIACCS. ACM, 2011, pp. 30-40.

[8] Mathias Payer, Thomas R. Gross, “String oriented programming: when ASLR is not enough”, in Proceeding PPREW '13 Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop.

[9] http://blog.trendmicro.co.jp/archives/7861 (2013/12/1)

[10] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In Proceedings of the 5th Symposium on Recent Advances in Intrusion Detection (RAID), Oct. 2002.

[11] R.Chinchani and E. van den Berg, “A Fast Static Analysis Approach To Detect Exploit Code Inside Network Flows,” RAID 2005, pp. 284 – 308, 2005.

[12] Michalis Polychronakis, Kostas G. Anagnostakis, and Evangelos P. Markatos. “ Comprehensive Shellcode Detection using Runtime Heuristics.” In Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC).

[13] 神保千晶，吉岡克成，四方順司，松本 勉，衛藤将史，井上大介，中尾康二，CPU エミュレータと Dynamic Binary Instrumentation の併用によるシェルコード動的分析手法の提案, ISCC2010-54.

[14] http://support.microsoft.com/kb/2458544/ja (2013/12/1)

[15] Lucas Davi, Ahmad-Reza Sadeghi, and Marcel Winandy. ROPdefender: A practical protection tool to protect against return-oriented programming. In Proceedings of the 6th Sym- posium on Information, Computer and Communications Security (ASIACCS), 2011.

[16] Vasilis Pappas, Michalis Polychronakis, and Angelos D. Keromytis. Transparent ROP Exploit Mitigation Using Indirect Branch Tracing. USENIX Security, 2013.

[17] Ruowen Wang, Peng Ning, Tao Xie, and Quan Chen. MetaSymploit: Day-One Defense against Script-based Attacks with Security-Enhanced Symbolic Analysis. USENIX Security, 2013.