シェルコード解析と分類による
不正侵入防止システムへの適応可能性の評価

齋藤　彰一† 　　上原　哲太郎‡ 　　下平　哲也§ 　　松尾　啓志†

† 名古屋工業大学大学院 　　§ 名古屋工業大学工学部
465-8555 名古屋市昭和区御器所町
{shoichi,matsuo}@nitech.ac.jp, shimodaira@matlab.nitech.ac.jp
‡ 京都大学　学術情報メディアセンター
606-8501 京都市左京区吉田本町
uehara@media.kyoto-u.ac.jp

あらまし

ある種のホストベース不正侵入防止システムでは，プログラムが実行する Windows API やシステムコールを監視し，異常な動作を検知することにより，マルウェアの侵入を防止している．多くのマルウェアは，まずシェルコードと呼ばれるシステムの脆弱性を利用する小さなプログラムを最初に侵入させる．このシェルコードがプロセスの制御を奪い，マルウェア本体を取り込む．この際，シェルコードが行う動作が異常検知の対象となる．

本稿では，そのような侵入防止システムの開発の基礎データとするため，研究用データセット CCC DATAset 2008 において収集された実際のシェルコードを解析，分類する．

Feasibility Assessment of IPS based on Analyzing and Classifying Shellcode

Shoichi SAITO† 　　Tetsutaro UEHARA‡ 　　Tetsuya SHIMODAIRA§
Hiroshi MATSUO†

†Graduate School of Engineering, Nagoya Institute of Technology
§Faculty of Engineering, Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya 465-8555, Japan
{shoichi,matsuo}@nitech.ac.jp, shimodaira@matlab.nitech.ac.jp
‡Academic Center for Computing and Media Studies, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
uehara@media.kyoto-u.ac.jp

Abstract

Some Host-based Intrusion Prevention Systems (IPS) observe Windows APIs and system calls issued by the target programs and detect their anomalous behaviour to prevent intrusions by malwares. Most of such malwares firstly exploit vulnerability of the system to feed small programs called shell codes. These shell code preempt processes to pull the main bodies of malwares. The target to detect anomalies is such behavior of the malware.

In this paper, the authors analize and classify actual shell codes obtained from CCC DATAset 2008 as the basis to develop an IPS.

1 はじめに

計算機システムに対するマルウェア侵入を防止するシステムとして，ホスト型の侵入防止システムが数多く提案され，商用化もなされている．侵入防止システム（以下，IPS という）にはいくつかの実現手法があるが，その中に各プロセスの発する OS へのシステムコール (Windows の場合は Windows API と呼ばれるがこれを含め以下では単にシステムコールと呼ぶ) を監視し，特定のプロセスが想定と異なるシステムコールを発行した場合に異常と判断するシステムがある．

システムコールは，ユーザアプリケーションがオペレーティングシステム（以下，OS という）の機能を利用する際に使用する．現在主流の保護機能を備えた OS においては，システムコールを使用することなく OS が管理する資源の利用（例えばファイルへのアクセスやネットワークを介しての他の計算機との通信）を直接行うことはできない．よってマルウェアも，外部との通信やファイルアクセスにはシステムコールを使用する必要がある．各プロセスは，通常当該プログラムに記述されたとおりにシステムコールを発するはずであるが，マルウェアはこのプロセスの制御を奪って動作するため，侵入を受けたプロセスの発行するシステムコールは通常と異なる．よって，あらかじめ各プログラムが発行すると予測されるシステムコールの数や種類，引数，順序等を調べておき，実行時に OS がそのプロセスのシステムコールを監視することによって，想定されないシステムコールが実行された場合に異常と判断することができる．なお，事前のシステムコールの調査には，プロセスを実際に動作させて OS に学習させる方法や，プログラムを静的に解析する方法などが考えられる．

筆者らは既に，Linux 上においてシステムコールを監視し，その異常を検出することによってマルウェア侵入を阻止する IPS を開発している [1]．しかし，実際のマルウェアにおいてシステムコールがどのように発行されるかの確認はこれまで十分に行えていなかった．そこで本稿では，研究用データセット CCC DATAset 2008[2]（以下，CCC2008 データという）に基づいて得られた実際のマルウェア感染記録を利用して，各マルウェアがシステムコールを発行する過程を解析し，IPS による検知可能性を検証する．なお，本稿では，Intel IA-32 アーキテクチャに基づいた計算機を対象としている．

本稿では，2 章でマルウェアの概要について述べ，3 章でマルウェアが感染当初に実行されるシェルコードの概要について述べる．また，4 章では，通信記録から得られたシェルコードの解析と分類結果についてまとめ，5 章でまとめを述べる．

2 マルウェア感染時の動作の概要

本章では，マルウェアがシステムに侵入する過程をまとめ，IPS がマルウェアを検出するタイミングについて検討する．一般に，マルウェアの多くは，以下の 3 つの段階を経てシステムに侵入することが知られている．

1. 脆弱性利用による小さなコード（シェルコード）の送り込み
2. 送り込んだシェルコードの実行による，マルウェア本体の送り込み
3. マルウェア本体の実行

まず，マルウェアは攻撃対象のシステム内で動作しているプログラムの欠陥，すわなち脆弱性を利用して，外部からプログラムの断片 (コード) を送り込み，当該プログラムが利用していたプロセスの制御を奪って実行させる．この際，この段階ではマルウェア本体を送り込むことをせず，一般にシェルコードと呼ばれる小さなコードを侵入させる場合が多い．このシェルコードの役割も多様であるが，その多くが「マルウェア本体プログラムをシステム内にダウンロードする」ことを実施する[1]．

[1] このような構造になっているのは，シェルコードが脆弱性を利用する際，スタックやヒープの限られたメモリサイズで動作する必要があることと，シェルコードはアセンブラで作成する必要があるため，多機能なマルウェアを C 言語等で作成してシェルコードにダウンロードさせるさせる方が開発効率が上がるためと考えられる．

シェルコードは，マルウェア本体を受信するために，TCP セッションの接続（connect）や待ち受け（listen）等を実施する必要がある．受信完了後には当該マルウェア本体を実行するが，この際新しいプロセスとして実行する場合が多い．

以上から，シェルコードは 1）脆弱性を利用してプロセスの制御を奪った直後に動作し，2）マルウェア本体を取得するために各種システムコールを使用する必要がある．これらから，IPS がプロセスの発行するシステムコール監視により侵入を検知し防止する際には，マルウェア本体ではなくシェルコードの動作を検知する必要がある．以下，IPS による検知のための，シェルコードの解析と分類について述べる．

3　シェルコードの特徴

本章では，シェルコードの特徴について我々が調査した結果をまとめる．

シェルコードは，攻撃対象のプログラムの脆弱性，特にバッファオーバーフロー等を用いて，自身のコードを当該プロセスのデータ領域に送り込むとともに，スタック上の関数戻りアドレスやヒープ上の割り込みベクタ等の関数エントリアドレスを書き換えて，当該プロセスの制御を奪う．この際，書き換えるべきアドレス（シェルコードの実行開始アドレス）はオーバーフローさせるバッファのアドレスに依存するが，これは当該プログラムの細かなバージョンの違いやプロセスの状態によって変化しうる．したがって，シェルコードの実行開始アドレスに一定の幅を持たせることを可能にするために，シェルコードの実行開始アドレス周辺には大量の NOP（No Operation）命令が挿入されている．本来のシェルコード本体は，この NOP 命令列に続いて配置される．この性質は，シェルコードの検出に利用可能である．

この他，今回調査したシェルコードは XOR 演算を用いた自身の暗号化や，Windows API のアドレス解決なども行っていたがここでは詳細を割愛する．

4　攻撃通信データ解析

本章では，CCC2008 データの攻撃通信データを解析し，シェルコードの特定，解析した結果について述べる．次に，シェルコードの分類とマルウェアとの関係について述べる．さらに特定したシェルコードから 2 つを選び，詳細な動作の分析について述べる．最後に，IPS による異常検知への適用について考察する．

4.1　シェルコードを含むパケットの特定

今回，調査対象とした CCC2008 データの攻撃通信データ（以下，攻撃通信データという）と攻撃元データ（以下，攻撃元データという）を使用してシェルコードの解析と分類を行った．攻撃通信データはハニーポットに対する攻撃時の通信を含むため，その中にはシェルコードが含まれている．この攻撃通信データ (2 日間分) を利用してシェルコード特定と分析を行った．また，初日の 0 時から 12 時までの 12 時間分のデータから，対応するマルウェアの特定と分析も行った．

攻撃通信データ内からシェルコードを取り出すためには，前章で述べたとおりシェルコードが NOP 列を含むことを利用している．NOP は数十バイトから数キロバイトの間，連続して配置されていた．この連続した NOP を検出した場合，当該パケットはシェルコードである可能性があると判断した．なお，多くのシェルコードは NOP に 0x90 を使用しているが，文献 [3] に示されるように 1 バイトのコードには NOP に相当することが可能な命令が他にもある（ただし，特定のレジスタの値の保存ができないなどの影響がある）．これらについても考慮し，文献 [3] に示された 1 バイト NOP 互換命令が連続した場合にもシェルコード候補と判断した．以下，NOP 命令という場合は，これらの NOP 互換命令も含まれる．以上の方式による特定の結果，1865 個のパケットに NOP 命令が含まれるという結果を得た．これらをシェルコード候補として解析の対象とした．

表 1: 特定したシェルコードの分類

ShellCode	特定数	動作タイプ	暗号化	NOP	構成の特徴
A	816	connect, jmp	有	90	
B	348	bind, CreateProcessA	有	90	
C(B)	226	bind, CreateProcessA	有	90	UNICODE
E	214	tftp, exec	無	90	
I	80	WinExec, ftp, exec	無	41-44	
G	54	URLDownloadToFileA, WinExec	有	90	NOP 前にコード
H(G)	32	URLDownloadToFileA, WinExec	有	90	
J	28	listen, jmp	有	90	
X	8	listen, create file, WinExec	無	90	
S	7	(connect, WinExec)	有	90	
AK(X)	2	listen, create file, WinExec	無	41	NOP 前にコード
AV(E)	1	tftp, exec	有	90	

注）ShellCode 欄の括弧内は，同一動作タイプだが暗号化や構成が異なる ShellCode である．

4.2 シェルコードの解析

シェルコードの解析は，1）パケットからのシェルコード部分の取り出し，2）逆アセンブルによる暗号化部分の特定と解析，3）復号，4）シェルコード本体の逆アセンブルによる解析の順で行った．逆アセンブラには NASM[4] を用いた．シェルコード本体の解析は，文献 [6] を参考にして行った．

シェルコード本体の解析では，使用される Windows API の特定を行った．API の特定処理は，シェルコードに含まれる Windows API のアドレス解決コードを解析することで行うことができる．アドレス解決コードには，当該 Windows API を指し示すキーとなる情報が記載されており，大きく分けて 2 種類ある．1 つ目は，Windows API 名をキーとしたアドレス解決が行われている場合で，この場合には当該 API の特定は容易である．2 つ目は，キーに Windows API 毎のハッシュ値を利用する場合である．この場合は，API 検索コードに渡されるハッシュ値を解析によって探しだし，対応する Windows API を特定する必要がある．このハッシュ値は，mwcollect.org[5] 等で確認することができる．これらの分析作業を通じ，各シェルコードが利用している Windows API を特定することができた．

4.3 シェルコードの分類

シェルコードの分類作業は，以下の手順によって行った．

1. 4.1 節によって特定したすべてのシェルコード候補について，NOP 命令が連続する数と連続した NOP 命令群の出現パターンとパケットサイズにより分類を行った．ここで，NOP 命令連続数と出現パターンとパケットサイズが完全に同一の場合は，同じシェルコードであると仮定した．

2. 各シェルコード群からそれぞれ複数個を選び，逆アセンブルによって動作解析を行った．この作業を通じて，各シェルコードが利用するシステムコールを特定した．

3. 上記の解析結果から，シェルコードの内，同一の動作タイプであるものをまとめた．

特定したシェルコードの分類結果を表 1 に示す．分類の対象としたシェルコードを含むと思われるパケットは，4.1 節で特定した 1865 個中の 1816 個であった[2]．各シェルコードは動作タイプと構成の違いによって分類し，便宜上アルファベットによる名前を付けた[3]．以降，各シェ

[2]残りの 49 個は，不明もしくは他のプロトコルの誤認識であった．

[3]名前は，NOP 命令の出現パターンによる分類によって A〜Z,AA〜AZ の名前を付けた後，動作パターンによる取りまとめを行った．その結果，集約された ShellCode 名は表 1 には現れていない．

表 2: シェルコードとマルウェアの対応

ShellCode	マルウェア
A	BKDR_VANBOT.(AD,AX,FM,MF), PE_VIRUT.(A,D,D-1,D-2,AV,YE), WORM_IRCBOT.AHX, TROJ_SYSTEMHI.BQ, TROJ_IRCBRUTEAT, TROJ_QQPASS.CC, WORM_KOLABC.(M,BQ), Mal_Virut-2, PE_BOBAX.(AK,AH), WORM_RBOT.(YF,TEM)
B	PE_VIRUT.GEN
S	PE_VIRUT.XY
C	BKDR_AGENT.ANHZ
X	Mal_Allaple

注）攻撃通信データ 2 日分の内の，初日の 0 時から 12 時までを対象として調査した．

ルコードを参照する場合は，表 1 の ShellCode 欄の名前を用いる．表 1 では，特定数，動作タイプ，XOR による暗号化の有無と NOP に使用された 1 バイト命令の種類を示している．また，構成に特徴があるシェルコードは備考欄に特徴を示す．

特定したシェルコードは，以下で述べる「動作タイプ」に基づいて分類した．シェルコードがマルウェア本体を取得し，実行する際には，攻撃元やダウンロードサイトにネットワーク接続し，制御を当該プログラムに移す．接続の際には，Windows API 等として connect, listen, tftp, ftp, URLDownloadToFileA を用いる．制御を当該プログラムに移す際には，WinExec, exec 等の Windows API を用いるか，直接 jmp 命令によって起動する．ここでシェルコードの動作タイプとは，これらのマルウェア本体等の第 2 プログラムの取得方法と制御移行方法で定義した．上記で述べたシェルコードの分類の結果，12 種類の動作タイプに分けることができた．

シェルコードは，検出を困難にするためか自身のコードを XOR によって暗号化しているものが多いが，今回の解析の結果，一部のシェルコードに XOR による暗号化を行っていないものが見つかった．また NOP 命令は，0x90 を利用したものが大半であったが，**ShellCode I** のみ，0x41〜0x44 を使用していた．

ShellCode C は，文字列処理を UNICODE により行っていた．また，実行開始アドレスが複数ありうるらしく，NOP 列を間に挟んで，2 つのシェルコード（中身は同一）が含まれていた．

4.4 シェルコードとマルウェア

4.3 節で特定したシェルコードが呼び出す（取得する）マルウェアの対応を表 2 に示す．表中の ShellCode 欄は，表 1 の ShellCode 欄に対応する．マルウェア名は，攻撃元データに記載されたものである（すなわち，独自にマルウェアの特定を行ってはいない）．なお，表 2 にない ShellCode については，当該シェルコードに直接結びつくようなマルウェアが検出されていないことを意味する．

マルウェアとの関係が判明したシェルコードは 4 種類である．**ShellCode A** 以外は，1 つのマルウェアのみとの関連している．**ShellCode A** は，多種多様なマルウェアのシェルコードとして再利用されていると考えられる．

4.5 シェルコードの詳細分析

本節では，今回検出したうち 2 つのシェルコード，**ShellCode A および I** の動作の詳細について述べる．

ShellCode A は，今回特定したシェルコードの中で最多の利用数のシェルコードである．**ShellCode A** は，以下の各 Windows API を実行する．

```
loadlibraryA("ws2_32")
socket()
connect(ip, port)
recv()
```

このうち，connect() の接続先 (ip) は攻撃元の IP アドレスである．recv() では，マルウェ

```
cmd /k echo open ip.ip.ip.ip port >> o &
echo user 1 1 >> o &
echo binary >> o &
echo get winscrvs.exe >> o &
echo quit >> o &
ftp -n -v -s:o &
del /F /Q o &
winscrvs.exe
```

図 1: **ShellCode I** が生成するバッチファイル

ア本体となるべきプログラムを受信する．このプログラムは先頭より直接実行可能なバイナリになっており，その実行は，recv() の受信バッファの先頭に直接 jmp することで行っている．このように，**ShellCode A** では単純に TCP ソケットを作成し，直接外部のマルウェア本体の供給元に向けて TCP 接続を行いダウンロードを行った上で，受信バッファ内に読み込まれたマルウェア本体を起動する．

なお，マルウェア本体のダウンロードが始まる前には，一種の認証のためか **ShellCode A** に埋め込まれたキーとなる値が connect の先に送信されていた．マルウェア本体のダウンロードは，このキーを送信した後から始まる仕様である．

ShellCode I は，NOP 命令に 0x41〜0x44 を使用するシェルコードである．また，XOR による暗号化を行っておらず，パケットで直接シェルコードを見ることができる．

ShellCode I は，バッチファイルを動的に作成して WinExec() を用いて実行し，FTP コマンドファイルを作成して FTP によるマルウェア本体のダウンロードを行い，実行する．パケットに記載されたバッチファイルの内容を図 1 に示す．IP アドレスとポート番号は，攻撃元を示している．なお，取得する実行ファイル名はマルウェアの種類に応じて変化する．

4.6 IPS による異常検知への適用

表 1 から，今回調査した全てのシェルコードが何らかのシステムコールを発行していることが明らかになった．これらは感染元プログラムに記述されていない，もしくは，実行されるタイミングが異なるシステムコールであるため，[1] で提案したシステムで容易に異常として検出できる．以上から，提案システムによる侵入防止が可能であることが，実際のシェルコードの解析結果からも検証された．

5 まとめ

本稿では，IPS によるシステムコールの監視による侵入防止の適応性を評価するために，侵入直後に動作するシェルコードの解析と分類を行い，提案システムによる侵入防止が実環境においても有効であることを確認できた．また，シェルコードとマルウェアの関係では，マルウェアの種類に比してシェルコードの種類が少ないことも明らかになった．今後の課題として，提案システムの環境である Linux におけるシェルコードの入手と解析を行い，実際に我々が開発している IPS での侵入検知実験を行う予定である．また，今回比較的少数にシェルコードが集約できることから，このことを利用してより高精度かつ低コストで侵入検知を行う手法も開発してゆきたい．

参考文献

[1] 鶴田浩史, 槙本裕司, 齋藤彰一, 松尾啓志, ライブラリ関数毎のシステムコール監視による侵入検知システムの開発, FIT2008 第 7 回情報科学技術フォーラム (2008).

[2] CCC DATAset 2008，サイバークリーンセンター (2008).

[3] Hsiang-Lun Huang, Tzong-Jye Liu, Kuong-Ho Chen, Chyi-Ren Dow, and Lih-Chyau Wuu, A polymorphic shellcode detection mechanism in the network, In *Proceedings of the 2nd international conference on Scalable information systems* (June, 2007).

[4] Simon Tatham, Julian Hall, H. Peter Anvin, The Netwide Assembler, `http://nasm.sourceforge.net/`.

[5] Paul Bacher, Markus Kotter, Georg Wicherski and Tillmann Werner, Nepenthes, `http://nepenthes.mwcollect.org/`.

[6] 愛甲健二, ハッカー・プログラミング大全 攻撃編, データハウス (2006).