Computer Security Symposium 2014
22 - 24 October 2014

ネットワーク側で ROP 攻撃コードを検出する手法の提案

田中恭之[†1,†2,a)]　後藤厚宏[†1]

概要：ゼロデイ脆弱性を悪用した標的型攻撃は多くの組織にとって脅威である．一因として一般に入手可能な攻撃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．本稿では，ネットワーク側でのゼロデイ攻撃の検出を目標として，ゼロデイ攻撃で多く用いられる ROP(Return-Oriented Programming)と呼ばれる技法をネットワーク側で検出する方式を提案する．従来方式では ROP 技法はホスト側で検出を行うが，提案方式では ROP 技法の特徴をネットワーク側で検出する．攻撃ツール Metasploit からの攻撃コードサンプルを用いた評価を行った．

キーワード：ゼロデイ脆弱性，攻撃コード，シェルコード，ROP，標的型攻撃

Proposal of a method to detect the ROP attack code on the network

YASUYUKI TANAKA[†1,†2,a)]　ATSUHIRO GOTO[†1]

Abstract: Targeted attacks exploiting a zero-day vulnerability are serious threats for many organizations. One reason is that available attack tools in general are very powerful and easy-to-use for attackers. In this paper, we propose a method that detects ROP (Return-Oriented Programming) attack code on the network. ROP is a core technique is used in zero-day attacks. Novelty of our method is able to detect ROP code efficiently on the network, while most proposed methods are to detect ROP code on the host machines. We evaluated the proposed method by using the attack code samples from attack tool Metasploit.

keywords: zero-day vulnerability, attack code, shell code, ROP, targeted attacks

1. はじめに

ゼロデイ脆弱性とはセキュリティ更新プログラム（パッチ）が未公開な状態の脆弱性である．シマンテック社によるとゼロデイ脆弱性の発生件数は，2011 年の 8 件，2012 年の 14 件に対して 2013 年は 23 件と近年増加傾向にある[1]. これらのゼロデイ脆弱性は標的型攻撃へ悪用されるため，社会的問題となっている[2]. 一例として 2013 年 9 月 17 日に発表されたブラウザ InternetExplorer のゼロデイ脆弱性（CVE-2013-3893）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 年 8 月上旬から攻撃コードが仕掛けられていた．攻撃コードは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[3]. また，このゼロデイ脆弱性は, 2013 年 10 月 9 日に修正パッチが公開されたが，修正パッチ公開前の 2013 年 9 月 30 日に攻撃ツール Metasploit[*1] のコードとして一般にリリースされ，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.

攻撃検出は，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. 企業等で多数のクライアント端末を運用するケースでは，導入及び運用コストの観点からネットワーク側の手法がより望ましい．また標的型攻撃等，高度化・巧妙化するサイバー攻撃に対し，自前ですべて対応するのは困難になってきており，マネージドセキュリティサービス[*2] 等の専門組織によるアウトソーシングが活用されるケースも増えている．マネージドセキュリティサービスでは，まずネットワーク側のセキュリティデバイスの検知ログから異常検出を試みるため, ネットワーク側での攻撃検出のニーズは高い.

本稿では，ネットワーク側でのゼロデイ攻撃の検出を目標として，ゼロデイ攻撃で多く用いられる ROP(Return Oriented Programming)と呼ばれる技法をネットワーク側で検出する方式を提案する. ゼロデイ攻撃では ROP 技法は多用されるためゼロデイ攻撃の検出につなげられると考える. 提案方式では, 攻撃者が ROP 技法を攻撃コードとして用いる場合に ASLR(Address Space Layout Randomization)非対応 dll のメモリ実行権限付与関数の物理アドレスを利用する点，および，ROP コードはシェルコード復号ルーチンの外に配置する必要があるため暗号化対象とならない点の 2 点に着目し，ROP コードを検出する．提案方式の 1 つに対し

†1 情報セキュリティ大学院大学
IISEC, Yokohama, Kanagawa 221–0835, Japan
†2 NTT コミュニケーションズ株式会社
NTT Communications Crop., Gran Park Tower 16F, 3-4-1, Shibaura, Minato-ku, Tokyo, 108-8118, Japan
a) mgs135505@iisec.ac.jp
*1 Metasploit. http://www.metasploit.com/

*2 http://www.lac.co.jp/service/operation/mss.html
http://www.symantec.com/ja/jp/page.jsp?id=TokyoSOC_mss
http://www-935.ibm.com/services/jp/ja/it-services/managed-security-services.html

実際の攻撃コードを用いた True Positive 評価を行い，検出文字列を増やすことで約 84%の ROP 攻撃コードが検出可能であったことを示す.

2. 攻撃コードと防御メカニズム

2.1 攻撃コードの基本構成

攻撃コードは, 図 1 のように, 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制御権をコントロールできるようにするまでの役割を担う．制御権が取られると，シェルコードが実行される．個々の脆弱性により異なるがシェルコードが配置できるメモリ領域はわずかであり，シェルコードに様々な機能を持たせてサイズの大きいコードを配置するのは非現実であるため，多くのシェルコードは攻撃の主目的を担うプログラム（マルウェア）のダウンロードのみを行う数十から数百バイトの小さいコード断片である*3.

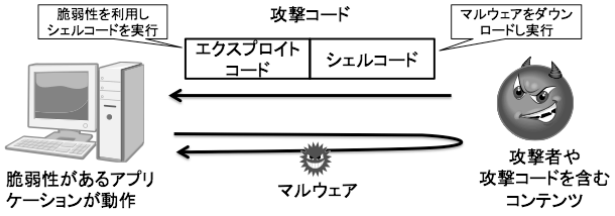

図 1　攻撃コード

2.2 防御メカニズム：DEP

DEP(Data Execution Prevention*4)とは，データ領域内でのコード実行を阻止するもので，Microsoft Windows では WindowsXP SP2 で導入された．制御権を奪ったエクスプロイトコードは，メモリ上のプログラム領域ではなく，スタックやヒープと呼ばれるデータ領域内にシェルコードを配置する．このため，DEP 有効下ではデータ領域での実行権が無くシェルコードは実行できず攻撃者は攻撃に失敗する.

2.3 DEP を回避する ROP 等の Code-Reuse 攻撃

DEP を回避するため Return-to-libc という技法が用いられるようになった．これは dll 等実行権限のあるメモリ中に存在する関数を直接コールする手法であり，スタックやヒープ等データ領域に配置したシェルコードを実行する手法と異なり DEP で防ぐことができない. さらにこの技法を進めた ROP(Return Oriented Programming) [4]という技法が最近の攻撃コードの主流となっている．ROP 技法を用いたコードの一例を図 2 に示す．ROPgadget とは，実行可能領域にある ret 命令で終了する数バイトのコード断片である．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（①). スタックの最上位の値にリターンするように調整しておき（②），最初に gadget No1 が実行される（③）．レジスタ EAX に意図した値(0x50505050)が格納され（④），ret 命令でスタックを参照し（⑤），次の gadget No2 にリターンする形で gadget No2 が実行される（⑥）．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．ROP 技法を進めた技法として JOP(Jump Oriented Programming) [5]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) [6]はフォーマットストリング脆弱性を利用する技法である．尚 ROP,JOP,SOP は Code-Reuse 攻撃とも呼ばれる.

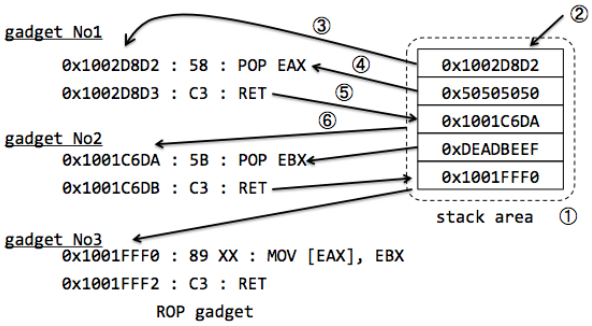

図 2　ROP コードの動作原理

2.4 防御メカニズム：ASLR

ASLR(Address Space Layout Randomization*4)はアドレス空間を OS 起動時にランダム化するもので，WindowsVista から導入されている. Return-to-libc や ROP 技法で見たように，攻撃者が関数やスタック・ヒープの既知の固定アドレスを利用することに対抗した防御方式である.

2.5 ASLR の回避

攻撃者は当該 OS が 32bitOS である場合は，ランダム化されたアドレス空間をスキャンして必要なアドレスを見つけ出すことによって，現実時間・現実試行回数内で ASLR を回避可能である．また dll によってはランダム化が行われない物が存在し，これを悪用した手法が最近の主流となっている．さらに，非 ASLR な dll しか得られない場合でも，脆弱性を利用して特定の dll のベースアドレスを得ることで動的に ROP 攻撃コードを組み立てる手法も登場している[7].

3. 先行研究

シェルコードの検出に関する先行研究には，IDS のようにネットワーク側で検出を行う手法とアンチウイルスソフトのようにホスト側で検出を行う手法があり，前者を 3.1 節及び 3.2 節に，後者を 3.3 節に示す．ホスト側で検出する手法よりネットワーク側の手法の方が，企業等で多数のクライアント端末を運用するケースで，導入及び運用コス

*3 Shellcodes database. http://repo.shell-storm.org/shellcode/

* 4 Microsoft Developer Network. http://msdn.microsoft.com/en-us/library/bb430720.aspx

トの観点から望まれる．

3.1 ネットワーク側での静的解析による検出

T. Toth らは，図 3（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[6]．しかしその方式からシェルコードの難読化や暗号化には対応できない．また，R. Chinchani らは，図 3（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シェルコードを判定している[9]．性能向上と高度な難読化への対応が課題となっている．

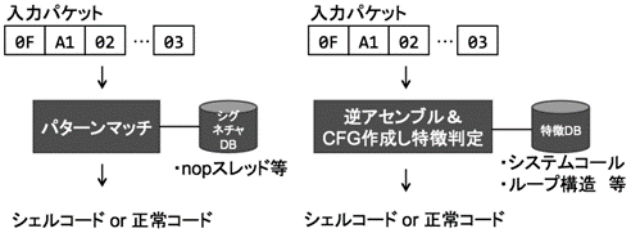

図 3　左：特徴文字列　右：CFG 作成

3.2 ネットワーク側での動的解析による検出

M. Polychronakis らは，図 4 のように，エミュレータでネットワーク上のパケットをコードとして実行を試み，既知の挙動にマッチするかによってシェルコードか否かを判断する[10]．実際に実行するため，高度な難読化や暗号化の影響を受けない．

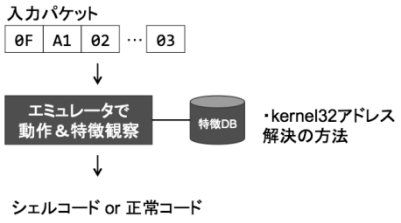

図 4　エミュレータで動作

挙動の一例として，シェルコードが PEB（Process Environment Block）を参照し，kernel32.dll のベースアドレスを解決する動作がある．このとき以下に示すような条件でこの挙動を捕らえることが出来る．

- 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 または InMemoryOrderModuleList または 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. 評価結果として False positives が少ない結果となっているが, 軸となる条件 1 に条件 2,3 を AND 条件を加えている点が効果を発揮していると思われる．さらに J.Khodaverdi は同様の手法で, 新たに egg hunter 等の特徴を捕らえ検出できる挙動を増やしている[11]．ここで egg hunter とは, 最終的に実行したいシェルコードに目印（egg）をつけ egg hunter コードがメモリ走査を行い egg を見つける技法[12]である．

ただし，M. Polychronakis の方式は，ネットワーク側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，実行コストが巨大になり非現実的である．また ROP 技法のような 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検出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が，目的は分析の効率化であり，リアルタイムでの検出ではない[13]．

3.3 ホスト側での検出，他

次にホスト側での検出分野で最新の先行研究を示す．ROP 技法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[14]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. この方式で ROP 技法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 技法の検出をする[15]．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．kBouncer 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 の履歴を活用し次のように ROP を検出する．攻撃者は ROP を用いて API コールやシステムコールを呼び出す．kBouncer は，その時点で過去の分岐履歴をみて正常なコードか ROP かを判定する．LBR の制約から履歴は 16 個に限られるが，V. Pappas らの調査結果では, 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．

その他の ROP 関連研究として K. Lu らの deRop では，ROP コードを通常のアセンブラ命令に置き換える方式を提案・評価している[16]．解析結果は本来同一であることが期待されるが，ASLR 環境の場合は，解析結果が毎回異なる点, および ROP コード部分の特定に動的解析を用いる必要があり解析時の危険を完全に切り離せない点の 2 つが課題である．また deRop の目的は解析の効率化であり ROP コードのリアルタイムの検出ではない．

4. 提案方式

昨今のゼロデイ攻撃でも多用され脅威となっている ROP 攻撃コードをネットワーク側で検出する方式を提案する．冒頭で述べたように，ネットワーク側のセキュリティデバイスの検知ログから異常検出を試みるマネージドセキュリティサービス等のビジネスニーズに対応できること，及び，従来研究では困難であったリアルタイム性が本提案方式の特徴である．

4.1 着眼点

ROP はホスト側のメモリ状態に依存するものであることからホスト側での検出アプローチが主流であり，先行研究では，ネットワーク側での ROP 攻撃コード検出の検討は進んでいない．そこで ROP 攻撃コードに着目してシェルコードを分析した結果，次のような特徴が判明した．図 5 でこの点を説明する．

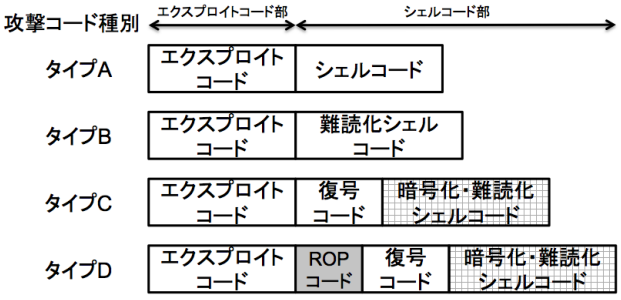

図 5　攻撃コード種別

2 章で示したように，基本となる攻撃コードはエクスプロイトコード＋シェルコードで，図 5 のタイプ A のように構成される．IDS やアンチウイルスソフト等の防御機構を回避するために，攻撃コードはタイプ B,C,D と高度化している．タイプ B は難読化が施されたコードであり，シェルコード部は難読化により元のシェルコードよりサイズが大きくなっている．タイプ C は，さらに暗号化が施されたコードであり，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．最後にタイプ D が ROP 攻撃コードである．DEP や ASLR 等の新たな防御策を回避するために用いられる ROP コードが付加された攻撃コードである．

次にタイプ A,B,C,D の攻撃コードと，ネットワーク側で先行研究（3.1 節，3.2 節）と提案方式の対応を表 1 にまとめた．先行研究ではネットワーク側でタイプ B や C は検出可能だが，タイプ D は検出できない．一方で提案方式はタイプ D に特化して検出を行う．ROP コードは，図 5 のタイプ D で示すように，攻撃者が ROP 攻撃コードを構成する上で，シェルコード復号ロジックの外側に配置する必要がある．これは復号コード自体に実行権限の付与が必要な為である．このため ROP コードをシェルコードの暗号化ロジックの内部に入れることはできない．また難読化の影響も受けない．このためネットワーク側で特徴をつかむのに好都合なのではないかと考え，この ROP コードの内部構成に着目した．

表 1 攻撃コード種別と各方式の対応

方式 / コード	ネットワーク側での先行研究			提案方式
	3.1 節[11]	3.1 節[12]	3.2 節[13]	
タイプ A	○	○	○	×
タイプ B	×	○	○	×
タイプ C	×	×	○	×
タイプ D	×	×	×	○

4.2 ROP コードの内部構成

ROP コードは 2.3 節で示したように実行権限のあるメモリ領域のコード部分を繋いで利用することで DEP を回避して任意のコードを実行可能とする技法である．一般に流通する攻撃コード*5 を調査した結果，攻撃コードとして ROP 技法が用いられる場合， ROP を利用して自由度の高い任意のシェルコードを書くのではなく，図 5 のタイプ D に示すように，ROP コードは後続するコード領域に実行権限付与するのみであった．これは 2.1 節で言及したようにシェルコードのサイズ要求が厳しいことに起因する．

次に実行権限がどのように付与されるかを述べる．Windows ではいくつかの関数をコールすることにより，通常は DEP(2.2 節)によって実行制限されているデータに実行権限を付与することができる．実行権限を付与可能な関数例を表 2 に示す．また表 3 にそのひとつである VirtualProtect 関数についての詳細を示す．

表 2 DEP を制御し実行権限を付与する関数例

関数名	
VirtualProtect	SetProcessDEPPolicy
VirtualAlloc	WriteProcessMemory
HeapCreate	NtSetInformationProcess

表 3 VirtualProtect 関数

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)

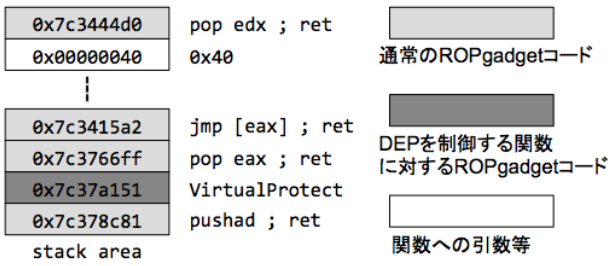

図 6　ROP コードが積まれたスタック

次に VirtualProtect 関数を用いた ROP コードがスタックに積まれた状態例を図 6 に示す．図 6 で DEP を制御して実行権限を付与する関数（表 2）の物理アドレス（図では VirtualProtect 関数）に関する ROPgadget コードを濃いグレー，これらの関数に適切な引数等を準備するのに用いられる ROPgadget コードを通常の ROPgadget コードとして薄い

*5 The Exploit Database. http://www.exploit-db.com/

グレー，関数等への引数を白で示す．攻撃者はスタックにこの 2 種類の ROPgadget コード及び引数等を適切に積んでおき，ROP コードを実行し，シェルコード配置エリアのメモリ領域に実行権限を付与することによって DEP を回避する．DEP が回避されると復号コードの実行等の処理が開始される．

攻撃者は安定して攻撃を成功させるためにメモリ空間の固定領域に配置された ROPgadget コードを用いようとする．2.4 節で示した ASLR が機能している場合は困難となるが 2.5 節で示した ASLR 回避方法がある．実際の攻撃コードを調査したところ，ASLR 非対応 dll の物理アドレスを用いる攻撃方法が多数存在した．ただし ASLR 非対応 dll の数は限られることから用いられる ROPgadget コードの数，特に DEP を制御する関数に対する ROPgadget コードの数は限定的であり，この物理アドレスを特徴文字列として ROP コードを検出する手法が有効であると考えられる．

4.3 提案方式による検出

提案方式は，ネットワーク上に流れるパケットとしてエクスプロイトコード，ROP コード，暗号化・難読化されたシェルコードの順でコードを観測する．攻撃コードの ROP コードは，図 6 で示したようなスタックの状態を作る為，ネットワーク上のパケットでは図 7 のように観測できる．4.1 節で示したとおり，この ROP コードはシェルコードを対象とした暗号化・難読化の影響を受けないため，IDS による侵入検知と同様にネットワーク上に流れるパケットの特徴から検出が可能と考えられる．

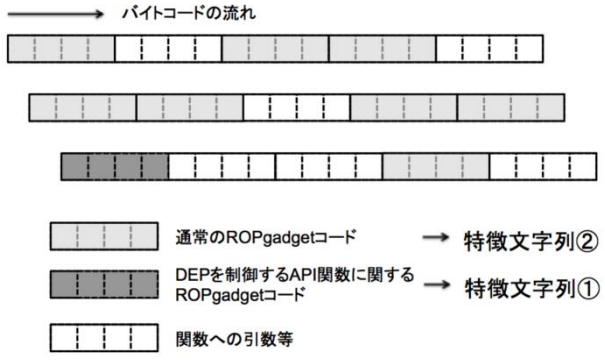

図 7　ネットワーク上の ROP コード

(1) 提案方式 1：DEP 制御関数アドレス利用

図 7 で，DEP を制御する関数の物理アドレスを指す ROPgadget コードを特徴文字列①と定義する．またこの DEP 制御 API 関数への適切な引数等を準備する ROPgadget コードを特徴文字列②と定義する．ここで図 7 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ヒック上に出現する．攻撃コードとして利用される ROP コードを調査すると，大多数が特徴文字列①を固定的に利用していることがわかった．そこで特徴文字列①を検出トリガーとする方法を提案方式 1 とする．具体的には次章で述べる．

(2) 提案方式 2：既知 DLL アドレス空間利用

多くの ROP コードは ASLR 非対応な dll を利用して作られ，非 ASLR-dll の数は少ないことは 4.2 節で示したが，それらの DLL のアドレス空間を特徴文字列として利用する．msvcr71.dll からの ROPgadget コードを用いて組み立てた ROP コードを図 8 に示す．msvcr71.dll のベースアドレスは非 ASLR で 0x7C340000-0x7C396000 にロードされる．このことから作られる Gadget は上位バイトが 0x7C3 となり図のグレーで示すように 4 バイト周期で現れる．表 4 に示すような非 ASLR-dll のアドレス空間を特徴文字列として蓄積しておき検出する方式を提案方式 2 とする．

```
64 4C 36 7C 64 4C 36 7C DC 4E 34 7C 10 02 00 00
05 1E 35 7C E3 36 35 7C FF FF FF FF FF 55 52 34 7C
8E 21 35 7C 37 59 34 7C 00 00 00 00 40 B1 1E 35 7C
B9 C5 36 7C 67 1E 39 7C 58 2E 34 7C 02 D2 34 7C
F4 F8 34 7C A2 15 34 7C DC 4E 34 7C 51 A1 37 7C
81 8C 37 7C 30 5C 34 7C
```

図 8　ROP コードの例

表 4 ROP に用いられる代表的な dll 例

名前	開始アドレス	終了アドレス	サイズ
msvcr71.dll	0x7c340000	0x7c396000	0x56000
hxds.dll	0x51bd0000	0x51ca7000	0xd7000
msvcrt.dll	0x77c10000	0x77c68000	0x58000

(3) 提案方式 3：未知 DLL アドレス空間利用

提案方式 3 は方式 2 を一般化させて未知の DLL に対応できる方式である．図 9 に疑似コードを示す．入力バイト列を先頭から 1 バイト毎にチェックする．先頭 4 バイトが ROP 被疑か否かをチェックし（行:03）被疑であれば，先頭 4 バイト含む 100 バイトをチェック対象とし（行:05），各 4 バイトの差分を取り，差分が閾値を下回る場合スコアを上げる（行:08,09）．スコアが閾値を超えた場合 ROP コードと判定する．（行:13）

```
(01) WHILE offset < file.size DO
(02)   dword = file[offset..offset+3]
(03)   IF dword > 0x7fffffff THEN offset +=1;NEXT
(04)   ELSE
(05)     dword[0..24] = [offset..offset+99]
(06)   ENDIF
(07)   FOR i = 0 to 24 DO
(08)     IF |dword[i]-dword[0] | < 0x100000 THEN
(10)       score += 1
(11)     ENDIF
(12)   ENDFOR
(13)   IF score > 10 THEN print “ROP code found.”
(14)   ENDIF
(15)   offset +=1
(16) ENDWHILE
```

図 9 提案方式 3 の疑似コード

(4) 提案方式のまとめ

提案方式の比較考察結果を表 5 に示す．検出計算量は方式

1 が一番少なく優位である．一方で検出精度は方式 1,2,3 の順に優位性が上がる．方式 2 では方式 1 で特徴文字列①で定義した DEP 制御物理アドレスが意図的に回避される場合でも検知できる．方式 3 では方式 2 が既知 DLL なのに対して未知 DLL も検出可能である．運用コストについては，方式 3 は特徴文字列を用いないので優位である．

表 5 提案方式の比較

提案方式	方式 1	方式 2	方式 3
検出計算量	◎	○	△
検出精度	△	○	◎
運用コスト	○	○	◎

5. 提案方式 1 の具体例

R. Wang らは Metasploit の悪用を脅威ととらえ攻撃を正確に検出することを目的として MetaSymploit というシステムを提案している[17]. 我々も Metasploit の悪用を脅威ととらえ同ツールの攻撃コードを調査したところ提案方式 1 で検出可能なコードが多数存在したため，提案方式 1 にて具体例を示し，評価することとした．

5.1 Metasploit からの特徴文字列抽出

4.3 節で定義した特徴文字列の入手方法は，流通する攻撃コードから抽出する方法や，4.2 節で示したように ASLR 非対応モジュールから該当する ROPgadget コードの物理アドレスを抽出する方法が考えられる．今回は，Metasploit を対象として特徴文字列の抽出を行った．

(1) 特徴文字列①の抽出

ゼロデイ脆弱性が多く発生するブラウザへの攻撃に ROP 技法が多用されることから，Metasploit のプラットフォーム windows のカテゴリ browser における攻撃コード総計 221 個から ROP 技法が使われる 22 個の攻撃コードを抽出した．これらの攻撃コードにおいて，表 2 で示した 6 つの関数を呼ぶと判断できる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 16 アドレス抽出できた．またそれらの物理アドレスに配置された関数は VirtualProtect と VirtualAlloc の 2 つのみであった．表 6 には該当する dll 名とアプリケーション名も併記した．

(2) 特徴文字列②の抽出

特徴文字列②は特徴文字列①と同じく Metasploit のプラットフォーム windows のカテゴリ browser における攻撃コード総計 221 個から ROP 技法が用いられる 22 個のコードを対象に，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が抽出できた．

5.2 実現方式例

これらの特徴文字列を利用して ROP 攻撃コードを検出する方式例を図 10 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のみで検出する場合 False positive がおこる可能性が高いため，ROP コードを組み立てる上で必要になる特徴文字列②を併用して精度を向上させている．

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
0x7c37a151	VirtualProtect	msvcr71.dll	JRE1.6
0x7c37a140	VirtualProtect	msvcr71.dll	JRE1.6
0x77c11120	VirtualProtect	msvcrt.dll	XP sp3
0x77ba1114	VirtualProtect	msvcrt.dll	XP sp3
0x7c801ad4	VirtualProtect	xul.dll	FF3.6
0x6d9fc094	VirtualProtect	jvm.dll	JVM
0x6d6c227c	VirtualProtect	regutil.dll	Java Regutils
0x63f010f4	VirtualProtect	mscorie.dll	.NET2.0
0x77c1110c	VirtualAlloc	msvcrt.dll	XP sp3
0x781a909c	VirtualAlloc	mozcrt19.dll	FF7,8,9
0x51bd115c	VirtualAlloc	hxds.dll	office2007
0x51bd10bc	VirtualAlloc	hxds.dll	office2010
0x1083828c	VirtualAlloc	xul.dll	FF3.6
0x100361a8	VirtualAlloc	FoxitReaderPlugin.dll	FoxitReaderPlugin
0x1001a22c	VirtualAlloc	dwa85.dll	Lotus iNotes ActiveX
0x4401a130	VirtualAlloc	AnnotateX.dll	Quest InTrust ActiveX

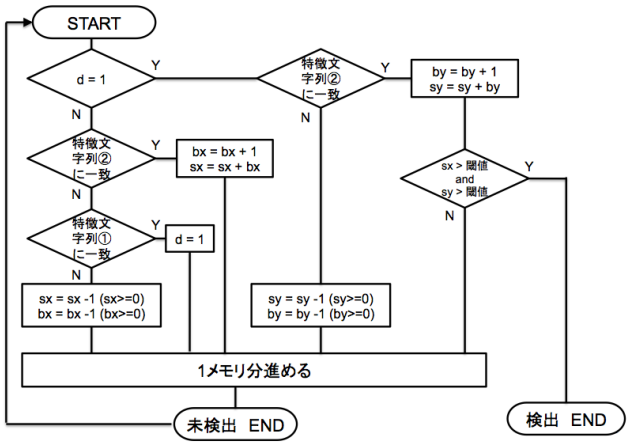

図 10 フローチャート例

6. 提案方式 1 の評価

5 章で抽出した特徴文字列を用いて ROP 攻撃コードが検出可能かどうか True Positive に関する評価を行った．5 章では，Metasploit のプラットフォーム windows，カテゴリ browser から特徴文字列抽出を行ったため，これらの特徴文字列を用いて，browser 以外の他のカテゴリすべてを対象にして評価を実施した．結果を表 7 の結果 1 列に示す．対象とした攻撃コード総数 623 個に含まれる ROP 攻撃コード総数 49 個に対し，ソースコードチェック及びパケットキャプチャの確認により 5 章で示した特徴文字列が攻撃コード内に出現し提案方式で検出可能な場合，検出数としてカウントした．49 個のうち，約 37%である 18 個の ROP 攻撃コードが検出可能であり，30 個の ROP 攻撃コードが検出不可であった．

表 7 評価結果

カテゴリ	コード総数	ROP 総数	結果 1	結果 2
misc	79	19	12	19
emc	2	1	0	0
fileformat	142	18	3	15
ftp	57	3	1	2
http	115	1	1	1
local	18	1	0	0
lotus	4	2	1	2
novel	9	1	0	1
smb	20	2	0	1
ssh	6	1	0	0
その他	171	0	-	-
合計	623	49	18	41

次に検出できなかった 30 個の ROP 攻撃コードについて調査を行った．大半のコードが VirtualProtect 関数と VirtualAlloc 関数を呼び出すものであるが，表 6 の特徴文字列①に示す物理アドレスではなく，アプリケーション固有の dll 等を利用した物理アドレスであった．これらの物理アドレス例を表 8 に示す. 表 8 に示したこれらの物理アドレスを特徴文字列①に追加して再評価を行った結果を表 7 の結果 2 列に示す．特徴文字列を追加したことによってカテゴリ，misc，fileformat，ftp，lotus，novel，smb に関して改善がみられれ，提案方式 1 で検出可能なコード数は約 49 個中 41 個（約 84%）に向上した．ただし，濃いグレーで示すカテゴリ，emc，fileformat，ftp，local，smb，ssh については一部, 検出できていない ROP 攻撃コードが存在することがわかる．

表 8 物理アドレス例

アドレス	関数名	DLL名	アプリケーション
0x10011108	VirtualProtect	SkinScrollBar.dll	dvdx
0x6ce212a4	VirtualProtect	libtaglib_plugin.dll	VideoLAN VLC
0x5f49b260	VirtualProtect	MFC42.dll	absolute_ftp
0x1007d158	VirtualAlloc	zenimgweb.dll	Novell ZENworks

7. 議論

提案方式の特徴を以下にまとめる．

- ROP コードを含む攻撃コードに特化して検出

表 1 に示したように，3 章の先行研究ではタイプ B・C の難読化・暗号化されたシェルコードを捕らえることを目標にしており ROP コードを含むタイプ D の攻撃コードの検出は不可である．一方で 3 つの提案方式はタイプ D に特化して検出を行う．

- ネットワーク上で検出可能

ROP はホスト側のメモリ状態に依存する物であるから先行研究ではホスト側の検出方式が主流である．一方で，企業等で多数のクライアント端末を運用する場合の導入及び運用コストの観点から，また，ネットワーク側のセキュリティデバイスの検知ログから異常検出を試みるマネージドセキュリティサービス等のビジネスニーズへの対応を踏まえるとネットワーク側での検出は有効である．

- アルゴリズムが高速

3 章の先行研究でのネットワーク側での逆アセンブリとコールグラフを作る方式及び 1 バイト毎にエミュレータで実行する方式と比べると，特に提案方式 1 はパターンマッチをベースとする方式であることから，処理コストが少ないことが自明である．

次に, 検出できなかった 8 個の ROP 攻撃コードについてさらに調査を実施した．検出できなかった理由は以下であった．

- プロトコルに応じたエンコードが行われている

FTP プロトコルでの PORT コマンド実装上のバッファオーバーフロー脆弱性をつく ROP 攻撃コードにおいて脆弱性を適切に発動させるためには攻撃者は PORT コマンドに合わせた形のデータを送る必要がある．3 つの提案方式でこれをネットワーク側で検出するには，このエンコード形式を考慮した上で特徴文字列を観測する必要がある．

- ROP コード動的生成

1 個の攻撃コードで，関数のアドレスを動的に取得し，その値に基づき ROP コードを動的生成するものがあった．概要を言及した提案方式 3 で検出可能と考えられるが別途評価が必要である．

- Javascript での難読化

特にブラウザのように入力データの自由度が高い場合，用いられる脆弱性に依存するが攻撃コード全体を Javascript 化しさらに難読化することが可能な場合がある．https 等の暗号化含め，ネットワーク側で観測する 3 つの提案方式は，これには対抗できない．

8. まとめ

本稿では，ニーズが高いネットワーク側でのゼロデイ攻撃への対策を目標として，攻撃コードの構成やホスト防御機構の回避に用いられるコードの調査・分析を行った．従来方式ではネットワーク側での検出検討が進んでいなかった ROP 攻撃コードをネットワーク側で検出する手法を複数提案しその 1 つについて，Metasploit の攻撃コードでの評価を行った．提案方式は，攻撃者が ROP 技法を攻撃コードとして用いる場合に ASLR 非対応 dll のメモリ実行権限付与関数の物理アドレスを利用する点，および，ROP コードはシェルコード復号ルーチンの外に配置する必要があるため暗号化対象とならない点の二つに着目し，ネットワーク上で ROP コードを検出する. 実際の攻撃コードを用いた評価では, 約 84%の ROP 攻撃コードが検出可能であった．さらに，あらかじめ想定できるエンコード方式を検出時に考慮することにより検出率を向上させることができると考える．ただし，Javascript 難読化が施されている場合や https 通信の環境では提案方式による ROP コード検出は難しい．今後は制約事項のインパクトを考慮した上で，提案方式を実装し，判定閾値を調整し False Positive に関する評価が課題である．

参考文献

[1] シマンテック：2014 年 インターネットセキュリティ脅威レポート 第 19 号，シマンテック（オンライン）（2014），入手先 (http://www.symantec.com/ja/jp/security_response/publications/) (参照 2014/08/16)

[2] 情報処理推進機構：2013 年版 10 大脅威 身近に忍び寄る脅威（オンライン）（2014）入手先 (http://www.ipa.go.jp/security/vuln/10threats2013.html) (参照 2014/08/16)

[3] インターネットウォッチ：日本の省庁などへ“水飲み場型攻撃”，IE のゼロデイを突き，8 月に起きていた （オンライン）(2014)，入手先(http://internet.watch.impress.co.jp/docs/news/20131010_618941.html) (参照 2014/08/16)

[4] Hovav 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.

[5] 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.

[6] 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.

[7] http://www.fireeye.com/blog/uncategorized/2014/04/new-zero-day-exploit-targeting-internet-explorer-versions-9-through-11-identified-in-targeted-attacks.html (参照 2014/08/16)

[8] Thomas 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.

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

[10] 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).

[11] Javad Khodaverdi, “Enhancing the Effectiveness of Shellcode Detection by New Runtime Heuristics”, International Journal of Computer Science Research and Application 2013, Vol. 03, Issue. 02, pp. 02-11.

[12] Skape : Safely searching process virtual address space（online）（2004），available from (http://www.hick.org/code/skape/papers/)（accessed 2014/08/16）

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

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

[16] Kangjie Lu, Dabi Zou, Weiping Wen, Debin Gao, “deRop: Removing Return-Oriented Programming from Malware”, ACSAC '11 Proceedings of the 27th Annual Computer Security Applications Conference. Pages 363-372.

[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.