Computer Security Symposium 2017
23 - 25 October 2017

不要な実行バイナリコードの削除による任意コード実行リソースの削減

池上 祐太[1]　正力 達也[2]

概要：脆弱性攻撃において，実行プログラムのコードやリンクされたライブラリ内部のコードを利用し，任意コードを実行するケースがある．プログラムの利用において，プロセスのメモリ上に展開されたすべてのコードを実行するケースは少ない．プログラムが実行しない，不要なコードを削除することで，攻撃者が利用できるコードリソースが削減されるため，任意コード実行を困難にすることができる．また，不要なコードを削除することで，脆弱性を含んだコード自体の削減も期待できる．本研究では，Linux の実行形式ファイル（ELF）を対象とし，不要なコードを削除する手法を提案する．本手法を適用することで，利用者が望むプログラム動作を実行しつつ，攻撃に利用できるコードリソースを削減でき，攻撃の難易度を上げることができると考える．

キーワード：任意コード実行防止，バイナリコード削減

Reduction of arbitrary code execution resources by deleting unnecessary execution binary code

Yuta Ikegami[1]　Tatsuya Shoriki[2]

Abstract: In vulnerability attacks, there are cases where arbitrary code is executed by using executable program code or code inside the linked library. In the use of the program, there are few cases where all the code expanded in the memory of the process is executed. Deleting unnecessary code that the program does not execute reduces the code resources available to attackers, which makes it difficult to execute arbitrary code. Also, by deleting unnecessary code, we can expect reduction of code itself including vulnerability. In this research, we propose a method to delete unnecessary code for Linux executable file (ELF). By applying this method, it was possible to reduce the code resources available for attack while executing the desired program operation by the user, and to increase the difficulty level of attack.

Keywords: Prevention arbitary code execution, Deleting binary code

1. はじめに

脆弱性攻撃において，実行プログラムの実行バイナリコード (以降，コードという) やリンクされたライブラリ内部のコードを利用し，任意コードを実行するケースがある．これは，OS やプログラムでのセキュリティ対策が進んでおり，任意コードを設置するスタック領域やヒープ領域において，任意コードを実行できないためである．このため，多くの任意コード実行は，実行可能領域のコードを再利用する攻撃（ret2lib, ROP など）が用いられている．この攻撃は，実行プロセス中の実行可能なバイナリコードをうまく組み合わせてることで，任意の処理を実行できる．また，この任意コードから，mprotect(2) のようなメモリに実行属性を付与する関数を呼び出すことも多い．再利用されるコードは，本体プログラムやロードしたライブラリのコードを使用する．既存の任意コード実行を防止する手

1　中国電力株式会社
The Chugoku Electric Power Co., Inc.
2　株式会社ケイ・オプティコム
K-Opticom Co., Inc.

© 2017 Information Processing Society of Japan

法は，数多く存在するが，最新の CPU や OS が必要，プログラムのソースコードが必要であり，導入コストが高い．本研究では，プログラムを改変することなく，不要な実行バイナリコードを削除することで，任意コード実行に利用できるコードリソースを削減する．これにより，任意コード実行を困難にできると考える．不要なコードとは，実行プロセス中には存在するが，実行されないコードを指す．不要なコードは，プログラムの利用者によって異なるという考えの基で研究に着手している．これは，同一プログラムでも，そのプログラムの一部分のみしか使用しない利用者や多くの機能を使用する利用者と様々存在し，実行されるコードも異なる．また，プログラムは，その本体コードだけでなく，多数のライブラリをロードする．ライブラリは，プログラム自体に組み込まれる静的リンクと他プロセスと共有する動的リンクがある．物理メモリの使用領域を削減するため，多くのプログラムは動的リンクを利用する．動的リンクを利用した場合，プログラムは，ライブラリ関数の一部しか使用しないにもかかわらず，ライブラリ全体がプロセスメモリにマッピングされる．このため，攻撃者は，ライブラリがマッピングされたメモリアドレスを取得することで，ライブラリの任意の関数を呼び出し，任意コード実行を行うこともある．本手法では，利用者にとって必要なコードのみを残し，不要なコードはライブラリも含めてすべて削除する．これにより，攻撃者が利用できるコードリソースが削減されるため，任意コード実行を困難にすることができる．また，本手法では，必要な命令コードのみを残すため，脆弱性そのものを含んだコードを削除できる効果も期待できる．

2. 関連研究

2.1 概要

本章では，既存の任意コード実行を防止する研究を述べる．

2.2 コード再利用攻撃を防止する研究

コード再利用攻撃を防止する研究として，ガジェット（コード再利用攻撃で利用されるコード断片のこと）の配置をランダマイズ化したり，命令そのものを変化させる手法がある．

文献 [1] は，バイナリの命令を等価な命令に置き換えることで，任意コード実行を防止している．文献 [2] は，コンパイラベースの防止手法であり，ガジェットに使用されうる命令をコンパイル時に改変している．文献 [3] は，実行コード，ライブラリなどの命令を分割し，配置をランダマイズ化する．LD_PRELOAD 機能で命令を書き換えるライブラリを挿入し，実現している．文献 [4] は，コンパイラベースの対策であり，Windows バイナリで，ASLR が適用されていない命令に対して，ランダマイズ化を適用させている．call や ret の前の命令を jmp 命令に置き換え，ret までをすべて int 0x3 命令にする．jmp 命令は，新規にコンパイラで作成した .ropf セクションへいき，そこに元の命令が配置させている．文献 [5] は，x86 バイナリを静的解析し，新しいセクションと命令を改変し，攻撃に対処する．プログラムのソースコード，デバッグ情報，再配置情報なしでバイナリをランダム化できる．しかし，本体バイナリのみのランダマイズ化で，ライブラリまではランダマイズ化していない．文献 [6] は，コンパイラベースの手法であり，直接ガジェットを実行させないよう，間にトランポリンコードを埋め込むことで，任意コード実行を防止する．

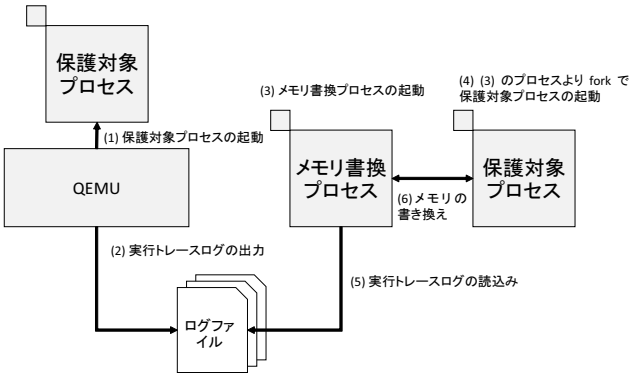

図 1　システムの全体像
Fig. 1　System overview

2.3 コードサイズを削減する研究

文献 [7] は，Linux カーネルのサイズが大きくなるに伴い，脆弱性が増加していることを問題として取り上げ，利用者に応じて，カーネルサイズを最適化する手法を提案している．Linux カーネルの再構築は，11,000 もの設定をする必要があるため，負担が大きい．そこで，自動的にカーネル再構築の設定を行う手法を提案している．これにより，50〜80%ほど，攻撃される割合を削減できた．手順としては，ftrace でカーネルコードを追跡し，設定ファイルに反映させている．

3. 提案手法

3.1 概要

本章では，提案手法の構成と実装方法について説明する．

3.2 システム構成

システム構成図を図 1 に示す．このシステムは，具体的に以下の手順で実行する．

(1) qira 上で保護対象プログラムを実行
(2) 保護対象プログラムが実行していないコードをログとして保存

© 2017 Information Processing Society of Japan

(3) メモリ書換えのプロセスを実行

(4) (3) で起動したプロセスから子プロセスとして，保護対象プログラムを実行

(5) 実行トレースログを読込み

(6) 保護対象プロセスのメモリを書換え，不要コードを削除

(7) 子プロセスをデタッチし，保護対象プロセスの処理を継続

まず，qira[8] という QEMU を利用したデバッガで，保護対象プログラムを動作させ，実行されたコードの情報をログファイルとして出力する．続いて，保護対象プログラムの実行されていないコードのログファイルを作成する．ここまでを qira が行う．続いて，メモリ書換えのプロセスを実行させ，保護対象プログラムを子プロセスとして実行させる．保護対象プログラムから子プロセスへアタッチし，エントリポイントまで実行させる．エントリポイントまで実行した後，ログファイルを読込み，子プロセスの不要なコードを削除する．不要コードを削除した後，子プロセスをデタッチする．

3.3 qira

本手法では，実行命令トレースを保存するため，qira というタイムレスデバッガを使用している．タイムレスデバッガとは，実行命令トレースをすべて保存しておき，プログラム解析を促進させるデバッガのことをいう．提案手法では，不要コードを発見するため，タイムレスデバッガを使用する．ここでは，qira というタイムレスデバッガを使用した．qira は，QEMU 上で解析対象プログラムを動作させ，効率的な解析を提供するデバッガである．QEMU を利用しているため，様々のアーキテクチャのバイナリでも解析できるという利点を持つ．なお，qira を使用しなくても，後述する ptrace などを使用することで，同様のプログラムを作成できるが，多数アーキテクチャへ対応できる利点を活用するため，qira を用いた．

本手法では，qira を改変し，本体プログラム自体の実行されたコードとリンクされたライブラリごとの実行されたコードをログファイルに出力させた．ログファイルは，各実行ファイルのベースアドレスからのオフセットとその実行コードのバイト数保存する．これにより，各モジュールごとに，実行されたコードのアドレスとそのバイト数が分かる．また，保護対象プログラムと各ライブラリの全命令を逆アセンブルし，命令コードのベースアドレスからのオフセットとそのバイト数をログファイルとして出力する処理も追加した．この 2 つのログファイルの差分を取ることで，実行されていないコードが分かる．その際に取得する必要な情報は，実行メモリアドレスのベースアドレスからのオフセットとゼロクリアする際に必要な命令コードのバイト数である．

表 1 ls の不要コードの削減数
Table 1 Number of unnecessary codes of ls

	全体のコード数	実行コード数	不要コード数
本体	17743	9954	16789
glibc	334039	40997	203042

3.4 不要コードの削除

不要コードの削除は，メモリ書換えプロセスで実施する．具体的には，ptrace を用い，保護対象プログラムを子プロセスとして動作させ，デバッギーとして実行する．通常，プロセスの実行可能領域は，書込み不可領域となっているため，メモリページのアクセス属性を変更する必要がある．これに対応するため，保護対象プログラムをデバッギーとして動作させることで，任意のメモリアドレスに任意データを書き込むことができる．これを達成できるのが ptrace であり，GDB のようなデバッガや strace で使用されている．

メモリ書換えプロセスは，qira で取得した不要コード情報を保存したログファイルと /proc/¡保護対象プロセスの pid¿/maps から読み込んだ各実行ファイルのベースアドレスを取得し，不要コードを削除する．今回は，ゼロクリアすることで，不要コードを削除した．

不要コードを削除するタイミングは，保護対象プログラムのエントリポイントに処理が移ったときに実施する．エントリポイントに処理が移った際，共有ライブラリの読込みはすべて終了している (dlsym のように，動的ロードされるライブラリは除く)．その後，いくつかのライブラリの関数を実行した後に，本体プログラムに処理が移る．エントリポイントまで ptrace で処理を実行させ，エントリポイントに達した際に，各モジュールの不要コードを削除させる．このタイミングで不要コードを削除することで，攻撃者に脆弱性を利用される前に，不要コードを削除できる．また，シンボル情報が存在するプログラムの場合，エントリポイントより main 関数の実行時点に，不要コードを削除することで，より多くの不要コードを削除できる．エントリポイントから main 関数までは，攻撃に利用されるライブラリのコードが実行される．これらのコードは，main 以降では実行されることはないため，main 関数呼び出し時に削除することで，任意コード実行をより困難にできる．

4. 提案手法の評価

4.1 概要

本章では，複数のプログラムに提案手法を適用し，ls コマンドを実行した際の不要コードの削減数を測定した．評価環境は，CPU は Intel Core i7-3520M (2.90 GHz)，メモリは 8 GB，OS は Linux 4.4.0-92/x86 64，ディストリビューションは Ubuntu 16.04 である

© 2017 Information Processing Society of Japan

4.2 不要コードの件数

本評価では，使用される機会の多い４つのプログラムを用い，不要コード数を測定した．コードは，.text セクション内のコードのみに限定している結果を表??に示す．実行トレースログから不要コードの件数を測定した．

5. 考察と課題

今後は，提案手法の更なる評価が必要である．提案手法は，Linux の実行ファイルを対象としている．今後の課題として，Windows での適用があげられる．また，提案手法をバイパスする攻撃として，必要なコードは残す必要があるので，それらのコードを利用された場合には，攻撃を防げない．あくまで，攻撃を緩和させる手法であるため，その他の脆弱性対策ツールなどと併用して利用することが望まれる．また，それらを適用した際の評価も必要である．さらに，dlsym のように，動的ロードされるライブラリへの適用も課題である．エントリポイント後にライブラリを動的ロードする場合，エントリポイント時に不要コードを実施するタイミングでは，このライブラリの不要コードを削除できない．この問題に対処するには，カーネル側から dlopen などの関数をフックし，不要コードを削除する処理を追加する必要がある．

6. おわりに

本研究では，脆弱性攻撃において，実行プログラムのコードやリンクされたライブラリ内部の不要なコードを削除する手法を提案した．今後は，更なる評価と課題を解決していく．また，不要なコードを削除することで，脆弱性を含んだコード自体の削減も期待できる．本研究では，Linux の実行形式ファイル（ELF）を対象とし，不要なコードを削除する手法を提案する．本手法を適用することで，利用者が望むプログラム動作を実行しつつ，攻撃に利用できるコードリソースを削減できることを示した．

参考文献

[1] Georgios Portokalidis, Angelos D. Keromytis: Fast and practical instruction-set randomization for commodity systems, Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC 2010, pp.41–48, (2010).

[2] Kaan Onarlioglu, Leyla Bilge, Andrea Lanzi, Davide Balzarotti, Engin Kirda: G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries, Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC 2010, pp.49–58, (2010).

[3] Lucas Vincenzo Davi, Alexandra Dmitrienko, Stefan Nurnberger, Ahmad-Reza Sadeghi: Gadge me if you can: secure and efficient ad-hoc instruction-level randomization for x86 and ARM, Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2013, pp.299–310, (2013).

[4] Hyungjoon Koo, Michalis Polychronakis: Juggling the Gadgets: Binary-level Code Randomization using Instruction Displacement, Proceedings of the 11th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2016, pp.23–34, (2016).

[5] Richard Wartell, Vishwath Mohan, Kevin W. Hamlen, Zhiqiang Lin: Binary Stirring: Self-randomizing Instruction Addresses of Legacy x86 Binary Code, Proceedings of the 2012 ACM conference on Computer and communications security, CCS 2012, pp.157–168, (2012).

[6] Stephen J. Crane, Stijn Volckaert, Felix Schuster, Christopher Liebchen, Per Larsen, Lucas Davi, Ahmad-Reza Sadeghi, Thorsten Holz, Bjorn De Sutter, Michael Franz: It's a TRAP: Table Randomization and Protection against Function Reuse Attacks, Proceedings of the 2015 ACM conference on Computer and communications security, CCS 2052, pp.243–255, (2015).

[7] Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Daniel Lohmann: Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring, Proceedings of the Network and Distributed System Security Symposium, NDSS 2013, pp.1–18, (2013).

[8] George Hotz: Timeless Debugging, USENIX Association, (2016).

© 2017 Information Processing Society of Japan