Linuxディストリビューションのバイナリにおけるメモリ破壊攻撃の対策技術の適用状況の調査

菅原 捷汰[1]　近藤 秀太[1]　渡辺 亮平[1]　横山 雅展[1]　中村 慈愛[2]　須崎 有康[3]　齋藤 孝道[2]

概要：現在も一定数報告があるメモリ破壊脆弱性は，端末の制御の奪取などを目的としたメモリ破壊攻撃に悪用される可能性がある．これまでにメモリ破壊攻撃に対して多くの対策技術が考案され，現在のLinuxディストリビューションのバイナリではコンパイラのセキュリティオプションを指定して対策技術が含まれている．しかし，対策技術が機能しているかは不明である．そこで本論文は，3種類の32bit Linuxディストリビューション（CentOS，Ubuntu，openSUSE）の3世代に含まれる計10000個以上のバイナリに対して，4つの対策技術（RELRO，SSP，PIE，Automatic Fortification）の適用状況を調査した．本論文では，調査の結果，ディストリビューションごとに普及度が違うことや一部の対策技術しか機能していないケースが存在したことを示す．

キーワード：メモリ破壊攻撃，対策技術の適用状況，普及度の違い

A Survey of Application Status of Prevention against Memory Corruption Attacks in Linux Distribution's binarys

Shota Sugawara[1]　Shuta Kondo[1]　Ryohei Watanabe[1]　Masahiro Yokoyama[1]　Jiai Nakamura[2]
Kuniyasu Suzaki[3]　Takamichi Saito[2]

Abstract: The memory corruption vulnerability, which has a certain number of reports, can be abused for memory corruption attacks which aim at hijacking an application control flow and so on. Until today, many countermeasures against memory corruption attacks has been proposed. The current Linux distribution binaries include countermeasures specifying the compiler security options. However, it is unclear whether countermeasures function. In this paper, we surveyed the application status of more than 10,000 binaries of 4 prevention technologies for 3 generations of 3 Linux distributions. As a result of a survey, we revealed that the spread is different from each distributions and there are a certain number of binaries that only some countermeasures are applied.

Keywords: Memory Corruption Attacks, Application Status of Prevention, Difference of Spread

1. はじめに

CWE-119[1] に分類されるメモリ破損脆弱性は，現在も一定数が報告され続けている．この脆弱性は，メモリ上に配置されるプログラムの制御情報を書き換えることでサービスのクラッシュや端末の制御の奪取を引き起こすメモリ破損攻撃に悪用される．これまでにメモリ破損攻撃に向けて様々な対策技術が考案されてきた．既に一部の対策技術はOSや主要なコンパイラに標準で組み込まれ，容易に適用できるようになっている．

一方，近年はメモリ破損攻撃の多様化も進んでいる．ROP（Return Oriented Programming）に代表されるコード再利用攻撃は，先行研究 [2] や [3] のようにさらに高度

[1] 明治大学大学院
Graduate School of Meiji University
[2] 明治大学
Meiji University
[3] 国立研究開発法人産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

な手法に派生しており，既存の対策技術を回避する攻撃が次々に発見されている．このように多様化するメモリ破損攻撃を防御，緩和する観点では，一つの対策技術を適用するだけでは不十分であり，複数の対策技術を組み合わせて適用することが望ましいとされている．

現在の Linux ディストリビューションのバイナリは，コンパイラのセキュリティオプションを指定してビルドされる．しかし，セキュリティオプションを指定しても対策技術が機能しているかどうかは不明である [4].

既存研究 [5] において，Ubuntu，Debian，CentOS の 3 種類の 32bit Linux ディストリビューションに標準で含まれる ELF バイナリを解析することで，GCC に組み込まれている対策技術のうち SSP，RELRO，PIE の 3 つについて適用状況の調査が行われた．その結果，RELRO と PIE は全体的に適用率が低く，安全とは言い切れないバイナリが一定数存在することが示された．

本論文では，3 つのディストリビューション（CentOS，OpenSUSE，Ubuntu）の 3 世代において，4 つのセキュリティ対策技術（RELRO，SSP，PIE，Automatic Fortification）の導入傾向や個々のディストリビューションのバージョン間での対策技術の適用状況の変化を明らかにすることを目的として調査を行った．調査対象とした CentOS，OpenSUSE，Ubuntu のバイナリ数は，それぞれ，4,711 個，6,396 個，3,385 個で，総数 14,492 個である．調査は，各ディストリビューションに標準で含まれる ELF バイナリを解析した．

その結果，以下のことがわかった．

(1) ディストリビューションごとに，対策技術の適用状況が違っていた
(2) 対策技術が，バイナリに一旦適用されたのち，後の世代で意図的に，非適用・弱化されたケースが散見された
(3) コンパイルオプションは指定されていたが，実際には，機能していないケースがあった

特に，(3) については，対策技術が特定の条件下でしか適用されないことに起因するが，その適用条件については，プログラマへの周知が必要であると言える．

2. 調査対象とする GCC で実装されている対策技術

メモリ破損攻撃への対策技術はこれまでに様々なものが提案され，一部の対策技術は OS や，GCC に代表される主要なコンパイラに標準で組み込まれている．本章では，調査対象とする 4 つの対策技術について取り上げる．

2.1 RELRO（RELocation Read-Only）

RELRO は，仮想アドレス空間内の.got セクションを読み込みのみ可能にするリンカによる対策技術である．一般的な関数のアドレス解決は，初回の関数の呼び出し時に行うが，RELRO が有効の場合，実行ファイルのロード時にシンボルのアドレス解決を行い，.got セクションを読み込みのみ可能にする．そのため，GOT 書き換え攻撃 [6] に対して有効である．コンパイル時にリンカオプションを指定することで，この対策技術が適用される．

RELRO は遅延バインドが有効な場合は Partial RELRO が適用され，仮想アドレス空間内に.got セクションとは別に読み書き可能な.got.plt セクションが生成される．Partial RELRO の場合は，.got.plt セクションへの書き込みが可能なので.got.plt への書き換えを許してしまう．遅延バインドが無効の場合は Full RELRO が適用され，.got.plt セクションは生成されず，データセグメント以外読み込み専用となる．Full RELRO では，.got セクションへの書き込みができないので GOT 書き換え攻撃は行えない．しかし，Full RELRO は，ロード時に全てのシンボルのアドレス解決を行うので，実行時のオーバーヘッドが高くなる．

2.2 SSP（Stack Smashing Protector）

SSP は，関数の呼び出し時にスタック領域内の変数とフレームポインタの間に canary という値を挿入し，関数の終了時に canary の値の書き換えの有無をチェックすることで Stack-based Buffer Overflow 攻撃を検知する対策技術である [7]．canary の値が書き換えられていた場合は，プログラムの実行を停止する．この対策技術は，適用対象のプログラムのコンパイル時に関数の先頭に canary を挿入する命令を挿入し，関数の末尾に canary の書き換えをチェックする命令を挿入する．SSP は GCC のバージョン 4.1 からデフォルトで適用される．

これらの検査コードは，ローカル変数に文字配列がない関数や，文字配列が 8 バイト未満である関数には挿入されない [8]．デフォルトの文字配列の閾値は 8 バイトであり，GCC の `--param ssp-buffer-size=N` オプションを用いて変更できる．また，全ての関数に検査コードを挿入する，`-fstack-protector-all` オプションも提供されている．なお，Ubuntu では，バージョン 10.10 以降，デフォルトの文字配列の閾値は 4 バイトとなっている [9].

2.3 PIE（Position Independent Executable）

PIE は，ASLR（Address Space Layout Randomization）によるテキスト領域のランダマイズ化を実現する対策技術である．実行ファイルのアドレス参照を相対アドレスにすることで，その実行ファイルが仮想アドレス空間のどこに配置されても正常に実行できるようにする．PIE は ASLR と併せて適用することで，テキスト領域，データ領域，ヒープ領域，スタック領域のベースアドレスがランダマイズ化されるので，ROP に代表されるコード再利用攻撃の緩和に一定の効果がある．PIE は GCC のバージョン 3.4 から導入された．

表 1　調査対象のディストリビューションとバージョン

ディストリビューション	バージョン		
Red Hat 系 CentOS	CentOS5.0 リリース日：2007-04-12 サポート終了日：2017-03-31	CentOS6.0 リリース日：2010-11-09 サポート終了日：2020-11-30	CentOS7.3 リリース日：2016-12-12 サポート終了日：2024-06-30
Slackware 系 OpenSUSE	openSUSE12.1 リリース日：2011-11-16 サポート終了日：2013-05-06	openSUSE13.1 リリース日：2013-11-19 サポート終了日：2016-02-03	openSUSE13.2 リリース日：2014-11-04 サポート終了日：2017-01-17
Debian 系 Ubuntu	Ubuntu10.04 リリース日：2010-08-17 サポート終了日：2013-05-09	Ubuntu12.04 リリース日：2012-04-26 サポート終了日：2017-04-28	Ubuntu14.04 リリース日：2014-04-17 サポート終了日：2019-04

2.4 Automatic Fortification

Automatic Fortification はコンパイル時に，バッファオーバーフロー脆弱性の原因となりうるライブラリ関数を書き込み先のバッファの境界検査を行う安全な代替関数に置換する対策技術である．置換された関数によって，バッファオーバーフローを検出した場合，プログラムの実行を停止する．

この対策技術は GCC のバージョン 4.0 以降および，glibc のバージョン 2.3.4 以降を必要とし，コンパイル時に `-O1` 以上の最適化と `-D_FORTIFY_SOURCE=N`（`N=1`，`2`）を指定した場合に有効となる．特に，`-D_FORTIFY_SOURCE=2` を指定した場合，フォーマット文字列攻撃も検出可能となる．`-D_FORTIFY_SOURCE` を有効にした時のコンパイラによる関数の置換は，書き込み先のバッファサイズおよび書き込むデータサイズによって以下のように変化する [10][11].

(1) 書き込み先のバッファサイズと書き込むデータサイズを判定でき，書き込み先のバッファサイズが書き込むデータサイズより大きい場合，境界検査を行う関数へ置換しない．
(2) 書き込み先のバッファサイズを判定でき，書き込むデータサイズを判定できない場合，境界検査を行う関数へ置換する．
(3) 書き込み先のバッファサイズと書き込むデータサイズを判定でき，書き込むデータサイズが，書き込み先のバッファサイズより大きい場合，警告を表示すると共に，境界検査を行う関数へ置換する．
(4) 書き込み先のバッファサイズを判定できない場合，境界検査を行う関数へ置換しない．

上記のように，Automatic Fortification は，対象のライブラリ関数の全てを置換するわけではない．(4) によって置換されていないライブラリ関数に起因するバッファオーバーフローは防ぐことができない．

3. 調査方法

3.1 調査対象

本論文で，調査の対象とする Linux ディストリビューションとそのバージョンを**表 1** に示す．

ディストリビューションは，Red Hat 系から CentOS，Slackware 系から openSUSE，Debian 系から Ubuntu を選定した．各ディストリビューションは 32bit のデスクトップバージョンである．

CentOS は，サポートが切れている 5.0，現在サポートされている 6.0 と 7.3 の 2 種類の計 3 種類を選定した．CentOS 7.3 は Alternative Architecture Special Interest Group によってサポートされているものを選定した．openSUSE はサポートが切れている 3 種類のバージョン（12.1，13.1，13.2）を選定した．Ubuntu はサポートが切れている 10.04 と 12.04 の 2 種類と現在サポートされているバージョンで最も古い 14.04 の計 3 種類を選定した．

ウィンドウマネージャーは，それぞれのディストリビューションでデフォルトのウィンドウマネージャーを使用した．CentOS の 3 バージョンと Ubuntu10.04 は GNOME，openSUSE の 3 バージョンは KDE，Ubuntu12.04 と Ubuntu14.04 は Unity である．さらに，CentOS の 3 バージョンと Ubuntu の 3 バージョンは OS をインストールした後に追加でソフトウェアをインストールしていない状態である．openSUSE は，調査に必要な readelf コマンドが存在しなかったので，binutils をインストールした．

上記の 9 種類のディストリビューションに含まれるデフォルトで PATH が通っているディレクトリ内のバイナリについて，RELRO，SSP，PIE，および，Automatic Fortification の 4 種類の対策技術の適用状況を調査した．

3.2 調査手法

3.2.1 全体の進め方

調査手法は各ディストリビューションの root ユーザにおける環境変数 PATH が通っているディレクトリ内のバイナリに対して，trapkit[12] の checksec を参考に作成したスクリプトを実行するというものである．スクリプトの実行結果から，ディストリビューションおよびバージョンごとで対策技術の適用状況を比較する．

3.2.2 RELRO の調査方法

RELRO の有効および無効の分類は対象のバイナリの GNU_RELRO セグメントの有無で行っている．GNU_RELRO セグメントが存在すれば，RELRO が有効であり，さらに対象のバイナリの.dynamic セクションのエントリタイプに BIND_NOW が存在すれば Full RELRO，存在しなければ Partial RELRO として分類する．

3.2.3 SSP の調査方法

SSP の有効および無効の分類は，canary の検査コードとして追加される `__stack_chk_fail` 関数の有無で行っている．検査コードはローカル変数に文字配列がない関数や，文字配列が 8 バイト未満である関数では挿入されない．対象のバイナリの全ての関数で検査コードが挿入されていない場合，SSP が有効でコンパイルされていても，本論文の調査では無効に分類される．

3.2.4 PIE の調査方法

PIE の有効および無効の分類は対象のバイナリの ELF ヘッダーのタイプが DYN かどうかで行っている．DYN であれば PIE が有効でコンパイルされているので，有効として分類し，それ以外は無効として分類している．

3.2.5 Automatic Fortification の調査方法

Automatic Fortification の有効および無効の分類は，書き込み先のバッファの境界検査を行う安全な代替関数として追加される `_chk` を接尾辞に持つ関数の有無で行っている．Automatic Fortification の安全な代替関数の置換条件を満たしておらず，置換が行われなかったバイナリは，Automatic Fortification が有効としてコンパイルされている場合でも，本論文の調査では無効に分類される．

4. 調査結果

ディストリビューションごとのバイナリ数を**表 2** に示す．CentOS では，5.0 が他のバージョンに比べ総バイナリ数が一番多く，6.0 が他のバージョンに比べ総バイナリ数が一番少ない．openSUSE では，バージョンが上がるにつれて総バイナリ数が増えている．Ubuntu では，12.04 が他のバージョンに比べ総バイナリ数が一番少なく，14.04 が他のバージョンに比べ総バイナリ数が一番多い．

表 2 調査したバイナリの総数

CentOS5.0	CentOS6.0	CentOS7.3
1,700	1,432	1,579
openSUSE12.1	openSUSE13.1	openSUSE13.2
2,033	2,169	2,194
Ubuntu10.04	Ubuntu12.04	Ubuntu14.04
1,131	1,094	1,160

4.1 各ディストリビューションの対策技術の適用状況とバージョン間の共通バイナリの調査

調査した 4 つの対策技術の適用状況のグラフを図 1〜4 に示す．

4.1.1 RELRO の適用状況

図 1 は RELRO の適用状況である．

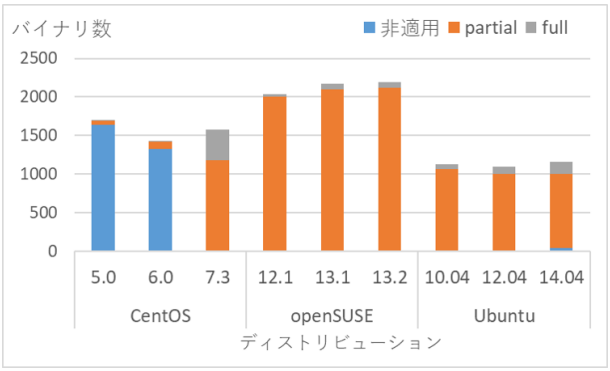

図 1 RELRO の適用状況

CentOS5.0 と CentOS6.0 では，RELRO が適用されていないバイナリがそれぞれ 1,639 個（96%），1,329 個（93%）であり，他の 7 種類のディストリビューションに比べその割合は大きかった．一方で，これら 7 種類のディストリビューションでは，Partial RELRO が適用されているバイナリが多く，openSUSE の 3 バージョンでは，それぞれ 2,000 以上，Ubuntu の 3 バージョンでは，それぞれ 1,000 前後であった．

Full RELRO が適用されているバイナリの数は CentOS7.3 以外の 8 種類のディストリビューションにおいては少なかった．適用率が一番高いもので Ubuntu14.04 の 153 個（13%）であった．CentOS7.3 ではその数は 400 個（25%）であった．

4.1.2 SSP の適用状況

図 2 は SSP の適用状況である．

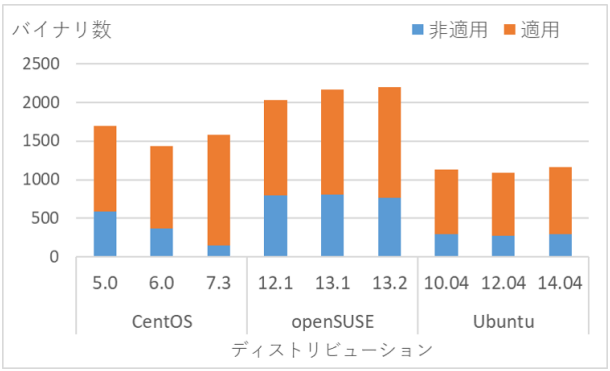

図 2 SSP の適用状況

どのバージョンにおいても，SSP が適用されているバイナリの方が多かった．特に，CentOS7.3 では SSP が適用

表 3　CentOS の対策技術の適用状況の変化

	CentOS5.0 と CentOS6.0				CentOS6.0 と CentOS7.3			
共通のバイナリの総数	957				1,004			
対策技術	RELRO	SSP	PIE	Automatic Fortification	RELRO	SSP	PIE	Automatic Fortification
変化しているバイナリの数	33	80	30	78	977	182	127	28
強化されているバイナリの数	25	59	18	64	977	178	127	12
弱化されているバイナリの数	8	21	12	14	0	5	0	16

されているバイナリの数が 1,432 個（91%）と適用率は一番高かった．

4.1.3　PIE の適用状況

図 3 は PIE の適用状況である．

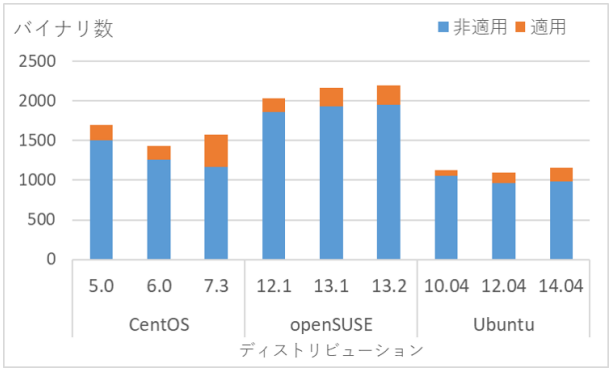

図 3　PIE の適用状況

どのバージョンにおいても，PIE が適用されているバイナリの方が少なく，特に Ubuntu10.04 では 75 個（7%），openSUSE12.1 では 177 個（9%）であった．CentOS7.3 では，その数は 409（26%）であり，適用率が一番高かった．

4.1.4　Automatic Fortification の適用状況

図 4 は Automatic Fortification の適用状況である．

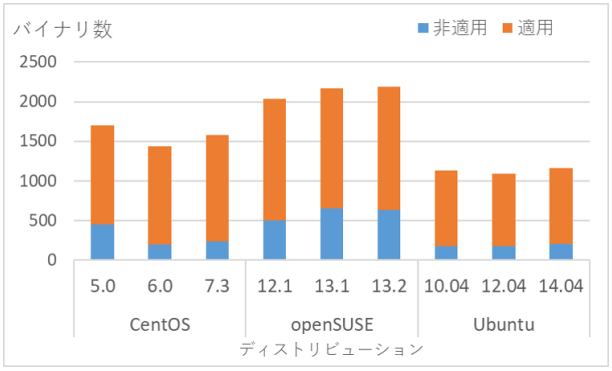

図 4　Automatic Fortification の適用状況

どのバージョンにおいても，Automatic Fortification が適用されているバイナリの方が多く，CentOS6.0 では 1,243 個（87%），CentOS7.3 では 1,340 個（85%），Ubuntu10.04 では 960 個（85%）であった．

4.1.5　適用状況の変化

本論文では，各ディストリビューションのバージョン間で共通するバイナリについて対策技術の適用状況の変化についても調査した．以降，非適用から適用または Partial RELRO から Full RELRO への変化を強化，適用から非適用または Full RELRO から Partial RELRO への変化を弱化と言う．表 3〜5 にディストリビューションのバージョン間で共通するバイナリの総数と適用状況が変化したバイナリの数を示す．

表 3 に CentOS における調査した対策技術の適用状況の変化を示す．CentOS5.0 から CentOS6.0 では，対策技術の適用状況が弱化されたバイナリの数よりも強化されたバイナリの数の方が多かった．CentOS6.0 と CentOS7.3 では，CentOS5.0 と CentOS6.0 の場合と比較すると RELRO，SSP，PIE の適用状況が変化しているバイナリの数は多くなっている．特に，RELRO では，変化しているバイナリの数が 977 個あり，それらすべてで適用状況が強化されていた．一方，Automatic Fortification は適用状況が強化されたバイナリの数よりも適用状況が弱化されたバイナリの数が多くなっていた．

表 4 に openSUSE における調査した対策技術の適用状況の変化を示す．openSUSE12.1 と openSUSE13.1 では，RELRO，SSP，PIE の適用状況が弱化されたバイナリの数よりも強化されたバイナリの数の方が多かった．一方，Automatic Fortification は弱化されたバイナリの数の方が多かった．openSUSE13.1 と openSUSE13.2 では，RELRO，SSP，Automatic Fortification の適用状況が強化されたバイナリの数の方が多かった．PIE は適用状況が変化しているバイナリすべてで適用状況が弱化されていた．

表 5 に Ubuntu における調査した対策技術の適用状況の変化を示す．Ubuntu10.04 と Ubuntu12.04 では，RELRO，SSP，PIE の適用状況が弱化されたバイナリの数よりも強化されたバイナリの数の方が多かった．一方で Automatic Fortification は適用状況が弱化されたバイナリの数の方が多かった．Ubuntu12.04 と Ubuntu14.04 では，RELRO と PIE の適用状況が強化されたバイナリの数の方が多く，SSP と Automatic Fortification では適用状況が弱化されたバイナリの数の方が多かった．

表 4　openSUSE の対策技術の適用状況の変化

	openSUSE12.1 と openSUSE13.1				openSUSE13.1 と openSUSE13.2			
共通のバイナリの総数	1,623				2,052			
対策技術	RELRO	SSP	PIE	Automatic Fortification	RELRO	SSP	PIE	Automatic Fortification
変化しているバイナリの数	28	88	45	21	10	33	3	22
強化されているバイナリの数	28	74	45	7	8	24	0	14
弱化されているバイナリの数	0	14	0	14	2	9	3	8

表 5　Ubuntu の対策技術の適用状況の変化

	Ubuntu10.04 と Ubuntu12.04				Ubuntu12.04 と Ubuntu14.04			
共通のバイナリの総数	965				1,006			
対策技術	RELRO	SSP	PIE	Automatic Fortification	RELRO	SSP	PIE	Automatic Fortification
変化しているバイナリの数	61	33	38	28	100	15	39	60
強化されているバイナリの数	36	23	38	7	54	4	32	20
弱化されているバイナリの数	25	10	0	21	46	11	7	40

表 6　ビルド日とリリース日の差の平均

CentOS			openSUSE		
5.0	6.0	7.3	12.1	13.1	13.2
64	257	403	17	48	30

4.2　パッケージのビルド日の調査

本論文では，ディストリビューションにインストールされているパッケージのビルド日がリリース日とどのくらい離れているのかを調査した (**表 6** 参照)．調査対象は，CentOS の 3 バージョンと openSUSE の 3 バージョンにインストールされているすべてのパッケージである．

調査方法は，それぞれのディストリビューションのリリース日と，パッケージのビルド日との差を求め，平均を取った．

CentOS ではどのバージョンにおいてもリリース日とパッケージのビルド日の差の平均が 2 ヶ月以上であった．特に，CentOS7.3 では 403 日であった．

他方，openSUSE はどのバージョンも平均が 2 ヶ月以内となった．特に，openSUSE12.1 では 17 日であった．

本論文ではさらに，CentOS の一部のパッケージに関して，パッケージのビルド日をプロットした．選定したパッケージは，Ubuntu の security-critical packages[13] に挙げられているものである．プロット図を**図 A·1** に示す．図 A·1 から，CentOS7.3 の一部のパッケージはリリース日から 2 年以上前にビルドされたパッケージを使用していることがわかった．

5.　考察

5.1　ソースコード解析とバイナリ解析の比較

本論文の調査では，各ディストリビューションに標準で含まれる ELF バイナリを解析した．これは，”WYSINWYX: What You See Is Not What You eXecute”，すなわち，ソースコード解析だけでは，セキュリティに関する機能が実行時にどのように振る舞うのかを示せないからである [4]. さらに，コンパイル時に，ある対策技術 X を有効にするオプション指定をしても，その対策技術 X が適用されたバイナリが必ずしも生成されるわけではない．例えば，Ubuntu 14.04 のソースパッケージ（cups-1.7.2）に含まれる cupsaccept は，ビルド時に Automatic Fortification を有効にするオプションが指定されるが，生成されたバイナリを解析すると関数の置換が行われていなかった．同様に，ソースパッケージ（util-linux-2.20.1）に含まれる ldattach は，ビルド時に SSP を有効にするオプションが指定されているが，生成されたバイナリには，SSP の検査コードは挿入されていなかった．

5.2　バージョン間における対策技術の適用状況の変化

表 3，表 4，表 5 にあるように，ディストリビューションのバージョンが上がる一方，対策技術が非適用となるバイナリが一定数存在することがわかった．特に，Ubuntu12.04 と Ubuntu14.04 間では，Automatic Fortification が非適用に変化しているバイナリが 40 個存在した．

これら 40 個のうち，33 個のバイナリは，Automatic Fortification が非適用となると同時に，RELRO も Plartial RELRO から非適用となっていた．この 33 個のバイナリは，3 つのパッケージ（x11-apps，x11-xfs-utils，x11-xkb-utils）に属し，いずれも，X Window System に関係するものであった．これらのバイナリのビルド時のコンパイルオプションを確認した．その結果，Ubuntu12.04 では，Automatic Fortification と Partial RELRO のオプションが明示的に指定されていたが，Ubuntu14.04 では，そのどちらのオプションも指定されてないことがわかった．また，残りの 7 個のバイナリのうち 5 個のバイナリは，1 つのパッケージ（xfonts-utils）に属する．これらのバイナリのビルド時のコンパイルオプションを確認したところ，Ubuntu12.04 でも Ubuntu14.04 でもコンパイルオプションが指定されていなかった．しかし，Ubuntu12.04 で

は，Automatic Fortification による関数の置換が行われていた．残り 2 個のバイナリは，2 つのパッケージ（evince，nautilus）に属する．これらのバイナリは，Ubuntu12.04 と Ubuntu14.04 のどちらもビルド時のコンパイルオプションが指定されていた．しかし，Ubuntu14.04 においては置換対象の関数のシンボルが存在しなかった．

さらに，PIE が非適用に変化したバイナリも調査した．Ubutun12.04 と Ubuntu14.04 間では，PIE が非適用に変化しているバイナリが 7 個あり，全て dbus パッケージに属していることがわかった．Automatic Fortification のケースと同様に，このパッケージのコンパイルオプションを確認したところ，Ubuntu14.04 では，PIE を有効にするコンパイルオプションが指定されていなかった．この原因を調査したところ，Ubuntu14.04 の dbus パッケージでは PIE を有効にすると正しく動作しないという理由 [14] から，明示的に PIE が無効化されていることが分かった [15]．

最後に，SSP が非適用に変化したバイナリも調査した．Ubuntu12.04 と Ubuntu14.04 間で，SSP が非適用に変化しているバイナリが 11 個存在した．これら 11 個のうち 2 個のバイナリは，2 つのパッケージ（intel-gpu-tools，x11-xkb-utils）に属する．これらのバイナリのビルド時のコンパイルオプションを確認した．Ubuntu12.04 では，コンパイルオプションが指定されていたが，Ubuntu14.04 ではコンパイルオプションが指定されていなかった．残りの 9 個のバイナリでは，どちらのディストリビューションもコンパイルオプションが指定されていた．9 個のうち 3 個のバイナリは，2 つのパッケージに分類することができる．これら 3 個のバイナリで，ソースコード中にバッファが存在するかどうかを確認した．その結果，これらのバイナリではバッファが存在しなかった．これは，SSP のコンパイルオプションを指定しても，ソースコード中にバッファがないという理由で SSP の検査コードが挿入されなかったことが考えられる．残りの 6 個のバイナリは，どちらのディストリビューションもコンパイルオプションが指定されており，ソースコード中にバッファが存在しているにもかかわらず，SSP の検査コードが挿入されていなかった．

5.3 PIE の適用率が低い理由

図 3 が示すように，ディストリビューションの種類によらず，PIE の適用率は低いことがわかった．

これは，セキュリティの効果が高くないうえに，パフォーマンスへの悪影響があることが原因であると推察される．

セキュリティの効果の観点では，32bit の Linux 環境における ASLR のエントロピーは低く，ブルートフォース攻撃によって回避されることが知られている [16]．

x86 環境において，PIE 形式のバイナリは通常のバイナリと比較して実行速度が大きく低下してしまう．GCC で PIE を適用したバイナリは，実行時にランダマイズ化したテキスト領域のベースアドレスを汎用レジスタの 1 つに保持する実装となっているので，結果として演算に使用できるレジスタが 1 つ少なくなる．x86 は汎用レジスタの数が 8 個と少ないので，これにより実行速度に大きな影響を受ける．先行研究 [17] は，x86 対象の Ubuntu11.04 を実験環境として，SPEC CPU 2006 が提供する 19 個のバイナリについて PIE 適用時と非適時の実行時間の比較を行っており，適用時は非適用時と比較して平均約 10%のオーバーヘッドが生じたと述べている．また，Ubuntu Wiki[18] においても，x86 環境では PIE 形式のバイナリの実行時に通常の 5〜10%のオーバーヘッドが生じるので，デフォルトではセキュリティ的に重要度の高いパッケージにのみ PIE を適用していることが述べられている．

CentOS と openSUSE において PIE の適用率が低くなった理由も Ubuntu と同様の理由だと推測する．

5.4 パッケージのビルド日の調査

図 A·1 から，CentOS において security-critical packages[13] で挙げられているパッケージは，リリース日から 1 年以内にビルドされたものが多かった．このことから，これらのパッケージはリリースに伴い更新されていることがわかった．しかし，security-critical packages 以外のパッケージの中には更新が行われていないパッケージが存在する可能性がある．また，本論文ではディストリビューション間でリリース日が離れているものを選定している．リリース日が近いディストリビューション間で調査を行えば，security-critical packages のパッケージでも更新されていないパッケージを使用しているディストリビューションが存在している可能性がある．

5.5 対策技術とセキュアプログラミング

Automatic Fortification の調査結果を用いて，セキュアプログラミングの観点で考察する．

たとえば，Ubuntu14.04 において，cups-1.7.2 の Makedefs ファイルを確認したところ，コンパイルオプションにて，Automatic Fortification を有効にするものが指定されていた．しかし，本論文におけるバイナリの解析によると，このパッケージに含まれる cupsaccept というバイナリ中で_chk を接尾辞にもつ関数が存在せず，Automatic Fortification による関数の置換が行われていなかった．このことは，コンパイラではオプションが有効になっているが，Automatic Fortification は機能しておらず，コンパイラは無駄なビルド作業を行っているともいえる．この現象の理由は，セキュアなプログラミングが浸透してきており，Automatic Fortification が変換の対象とする脆弱な関数を，プログラマが使うケースが減ってきたとが推察される．

一方で，Automatic Fortification は，すべての対象のライブラリ関数を置換できるわけではない．コンパイル時に書

き込み先のバッファサイズを決定できない関数は置換されず，その関数呼び出しに起因するバッファオーバーフローは検知できない．具体的な例として，CVE-2009-2957 に報告されている dnsmasq[19] の Heap-based Buffer Overflow 脆弱性がある [20]．この脆弱性は，strncat 関数における TFTP パケットの処理の不備に起因する．本論文での検証実験において，Automatic Fortification を明示的に有効にし，この dnsmasq をコンパイルしたが，strncat 関数は置換されず，バッファオーバーフローを防ぐことはできなかった．実験環境は，Ubuntu14.04 32bit，dnsmasq-2.49，gcc-4.8.4，glibc2.19 である．以上より，Automatic Fortification の適用の有無に関わらず，アプリケーションプログラマは適切な教育を受けるなどして，安全でない関数を利用しないことが望ましいといえる．

6. まとめ

本論文では，主要な Linux ディストリビューションにおける対策技術の適用状況を調査した．調査対象とした CentOS，OpenSUSE，Ubuntu のバイナリ数は，それぞれ，4,711 個，6,396 個，3,385 個で，総数 14,492 個である．その結果，以下のことがわかった．

(1) ディストリビューションごとに，対策技術の適用状況が違っていた
(2) 対策技術が，バイナリに一旦適用されたのち，後の世代で意図的に，非適用・弱化されたケースが散見された
(3) コンパイルオプション指定はされていたが，実際には，機能していないケースがあった

特に，(3) については，対策技術が特定の条件下でしか適用されないことに起因するが，その適用条件については，プログラマへの周知が必要であるといえる．

参考文献

[1] CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer, http://cwe.mitre.org/data/definitions/119.html
[2] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazieres, and D. Boneh. Hacking blind. In Proc. of IEEE Symposium on Security and Privacy. 2014.
[3] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.R. Sadeghi. Just-in-time Code-Reuse: On the effectiveness of fine-grained address space layout randomization. In Proc. of IEEE Symposium on Security and Privacy. 2013.
[4] Balakrishnan G., Reps T., Melski D., and Teitelbaum T. WYSINWYX: What You See Is Not What You eXecute. In: Meyer B., Woodcock J. (eds) Verified Software: Theories, Tools, Experiments. Lecture Notes in Computer Science, vol 4171. Springer, Berlin, Heidelberg (2008)
[5] T. Saito, H. Miyazaki, T. Baba, Y. Sumida, and Y. Hori. Study on Diffusion of Protection/Mitigation against Memory Corruption Attack in Linux Distributions, In Proc. of the 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS) 2015. 2015
[6] Mller, Tilo. ASLR smack & laugh reference. In Proc. of Seminar on Advanced Exploitation Techniques. 2008.
[7] IPA オープンソース・ソフトウェアのセキュリティ確保に関する調査報告書, https://www.ipa.go.jp/files/000013695.pdf
[8] IPA ISEC セキュア・プログラミング講座：C/C++言語編 第 10 章 著名な脆弱性対策， https://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/c905.html
[9] ubuntu wiki CompilerFlags, https://wiki.ubuntu.com/ToolChain/CompilerFlags
[10] [PATCH] Object size checking to prevent (some) buffer overflows, http://gcc.gnu.org/ml/gcc-patches/2004-09/msg02055.html
[11] Rober C.Seacourd，C/C++セキュアコーディング 第 2 版
[12] TRAPKIT checksec.sh, http://www.trapkit.de/tools/checksec.html
[13] BuiltPIE, https://wiki.ubuntu.com/SecurityTeam/KnowledgeBase/BuiltPIE
[14] enables PIE, which often doesn't work on odd platforms, https://bugs.freedesktop.org/show_bug.cgi?id=16621
[15] D-Bus 1.11.14 (2017-06-29), https://dbus.freedesktop.org/doc/NEWS
[16] Shacham, Hovav, et al. On the effectiveness of address-space randomization. In Proc. of the 11th ACM conference on Computer and communications security. ACM, 2004.
[17] Mathias Payer. Too much PIE is bad for performance. In Proc. of ETH Zurich, Department of Computer Science Technical Report 766. 2012
[18] Ubuntu Wiki: Security/Features, https://wiki.ubuntu.com/Security/Features
[19] Dnsmasq, http://www.thekelleys.org.uk/dnsmasq/doc.html
[20] CVE-2009-2957, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2957

付　　録

図 **A·1**　CentOS のパッケージのビルド日のプロット図