# Linuxアプリケーションバイナリにおけるコンパイラセキュリティオプションの有効性の静的解析

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

**概要**：現在の Linux アプリケーションはコンパイラで複数のセキュリティオプションを有効にして，バイナリが作成される．しかしながら，このセキュリティオプションにより期待されるセキュリティ機能が，すべてのバイナリに対して有効であるかは不明である．本論文では，4 つのセキュリティ対策技術（スタックを保護する SSP, ELF バイナリの got セクションを読み出し専用にする RELRO，メモリのランダム配置を有効にする PIE，脆弱なライブラリ関数を置き換える Automatic Fortification）がどの程度有効化しているかを 3 つの Linux ディストリビューション（CentOS, Ubuntu, openSuse）の 4 世代に対して総数 30,000 以上のバイナリおよびパッケージの調査を行った．この結果，セキュリティ対策技術が開発者の期待通り適用されていないこと，普及度に違いがあること，また，過去にビルドされた古いパッケージを利用し続けているディストリビューションが存在することが分かった．

**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

## 1. はじめに

CWE-119[1] に分類されるメモリ破壊脆弱性は，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壊攻撃に悪用される．これまでにメモリ破壊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，近年はメモリ破壊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壊攻撃を防御，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．

しかしながら，既存研究 [4] において，CentOS, Ubuntu, Debian 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 バイナリを解析し，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結果，RELRO と PIE は全体的に適

[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

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．ただし，既存研究では各ディストリビューションの 1 世代，32bit のみを対象としており，調査範囲が狭いという欠点がある．

そこで，本論分では，既存研究を拡張し，3 つのディストリビューション（CentOS，openSUSE，Ubuntu）の 4 世代の 32/64bit 版に標準で含まれる ELF バイナリを解析し，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パッケージ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．さらに，パッケージのビルド日とパッケージに含まれるバイナリの関係を明らかにするために，各ディストリビューションに標準で含まれるパッケージのビルド日も調査した．調査対象とした CentOS，Ubuntu，openSUSE のバイナリ数は，それぞれ，10,542 個，8,284 個，15,424 個で，総数 34,250 個である．

本論分の貢献は新たなセキュリティ対策技術を考案する際の助けになることだと考えている．バイナリのセキュリティ対策技術の適用状況を明らかにすることで，脆弱性を悪用される可能性があるバイナリが可視化される．結果として，対策技術の有効性を評価する際の一つの指標になると考えている．

本調査ではソースコード解析ではなくバイナリ解析を行う．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 [5].

調査の結果，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
(4) 過去にビルドされたパッケージを利用し続けているディストリビューションが存在した

## 2. 調査対象とする GCC で実装されている対策技術

メモリ破壊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．本節では，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専用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専用にする．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ようになった．

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．

### 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 調査対象のディストリビューションとバージョン

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のバージョンの選定は，異なるディストリビューション間の比較および最新のバージョンの対策技術の適用状況の調査の観点から行なった．

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 | 調査対象のビット | リリース日 | サポート終了日 |
|---|---|---|---|---|
| Red Hat 系 CentOS | 6.3 | 32/64bit | 2012-07-09 | 2020-11-30 |
| | 6.5 | 32/64bit | 2013-12-01 | 2020-11-30 |
| | 6.6 | 32/64bit | 2014-10-28 | 2020-11-30 |
| | 7.4 | 64bit | 2017-09-14 | 2024-06-30 |
| Debian 系 Ubuntu | 12.04 | 32/64bit | 2012-04-26 | 2017-04-28 |
| | 13.04 | 32/64bit | 2013-04-25 | 2014-01-27 |
| | 14.04 | 32/64bit | 2014-04-17 | 2019-04 |
| | 17.04 | 64bit | 2017-04-13 | 2018-01 |
| Slackware 系 openSUSE | 12.2 | 32/64bit | 2012-09-05 | 2014-01-15 |
| | 13.1 | 32/64bit | 2013-11-19 | 2016-02-03 |
| | 13.2 | 32/64bit | 2014-11-04 | 2017-01-17 |
| | 42.3 | 64bit | 2017-07-26 | N/A |

異なるディストリビューション間で比較を行うために，連続した 3 年間（2012 年，2013 年，2014 年）にリリースされた 32/64bit 版を各ディストリビューションにつき 3 バージョン選定した．2015 年，2016 年にリリースされたバージョンには 32bit 版が存在しないディストリビューションがあるので，今回は選定の対象外とした．さらに，比較的新しいバージョンの対策技術の適用状況を調査するために各ディストリビューションにつき，2017 年にリリースされたバージョンを 1 つ選定した．Ubuntu17.04 には 32bit 版も存在するが，CentOS と openSUSE には存在しない．調査条件を同様にするために，今回は，64bit 版のみを選定した．調査対象の各ディストリビューションはデスクトップバージョンである．

ウィンドウマネージャーは，各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 4 バージョンは GNOME，Ubuntu の 4 バージョンは UNITY，openSUSE の 4 バージョンは KDE である．さらに，CentOS の 4 バージョンと Ubuntu の 4 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 の 4 バージョンは，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．

本論文では，上記 21 種類のディストリビューションに対して以下の調査を行なった．

(1) 対策技術の適用状況の調査
(2) パッケージのビルド日の調査

### 3.2　対策技術の適用状況の調査手法

対策技術の適用状況の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およびシンボリックリンクが指す先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に対策技術の適用状況を比較する．

#### 3.2.1　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．

#### 3.2.2　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する．

#### 3.2.3 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．

#### 3.2.4 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する．

### 3.3 パッケージのビルド日の調査手法

調査手法は，デフォルトでインストールされているパッケージに対して，rpm コマンドを用いることでビルド日を取得するというものである．CentOS の 32bit および openSUSE の 32bit に対して調査を行なった．

## 4. 調査結果

4.1 節では調査対象のバイナリの数および各対策技術の適用状況の調査結果を示す．4.2 節では調査対象のパッケージの数およびパッケージのビルド日の調査結果を示す．

### 4.1 各対策技術の適用状況の調査結果

各ディストリビューションのバイナリ数を**表 2** に示す．32bit と 64bit のバイナリ数は概ね同数であった．Ubuntu，CentOS，openSUSE の順にバイナリ数が多いことがわかる．

#### 4.1.1 RELRO の適用状況

**図 1**〜**図 3** は各ディストリビューションにおける RELRO の適用状況である．

CentOS6.3，6.5 および 6.6 では非適用の割合が他のディストリビューションに比べて高いが，

**表 2** 調査したバイナリの総数

| ディストリビューション | バージョン | バイナリの数 32bit | 64bit |
|---|---|---|---|
| CentOS | 6.3 | 1,497 | 1,499 |
| | 6.5 | 1,485 | 1,487 |
| | 6.6 | 1,492 | 1,494 |
| | 7.3 | N/A | 1,588 |
| Ubuntu | 12.04 | 1,103 | 1,102 |
| | 13.04 | 1,152 | 1,152 |
| | 14.04 | 1,161 | 1,164 |
| | 17.04 | N/A | 1,450 |
| openSUSE | 12.2 | 2,178 | 2,181 |
| | 13.1 | 2,199 | 2,199 |
| | 13.2 | 2,228 | 2,226 |
| | 42.3 | N/A | 2,213 |

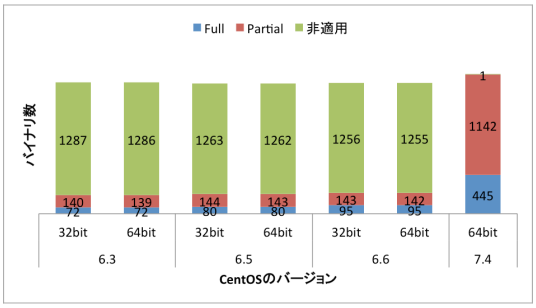


**図 1** CentOS における RELRO の適用状況

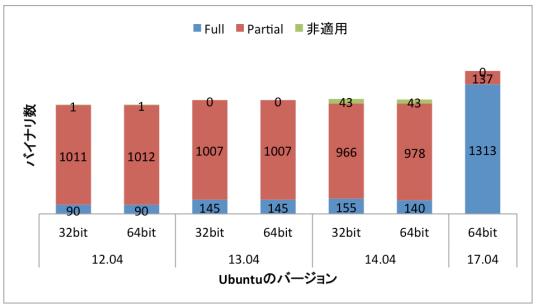


**図 2** Ubuntu における RELRO の適用状況

最新バージョンである 7.4 で改善されている．Full RELRO が適用されているバイナリの割合は Ubuntu17.04 のみ 90%を上回っていた．

#### 4.1.2 SSP の適用状況

**図 4**〜**図 6** は SSP の適用状況である．

どのディストリビューションにおいても，SSP が適用されているバイナリの割合は非適用に比べて高かった．CentOS7.4，Ubuntu17.04 以外のディ

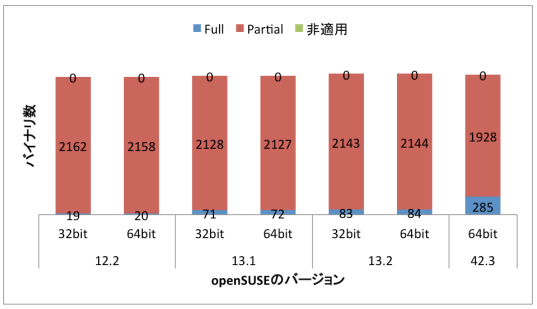


図 3　openSUSE における RELRO の適用状況

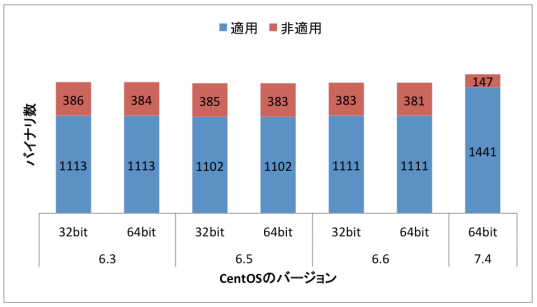


図 4　CentOS における SSP の適用状況

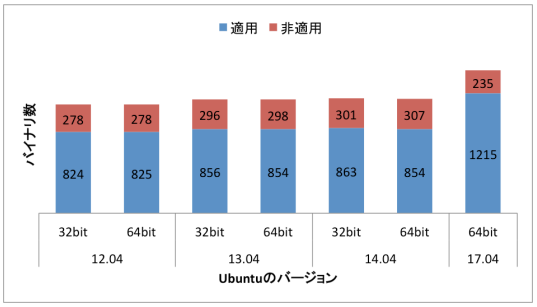


図 5　Ubuntu における SSP の適用状況

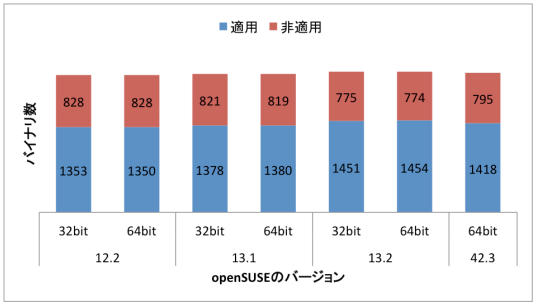


図 6　openSUSE における SSP の適用状況

ストリビューションにかけては適用率が横這いであることがわかる．

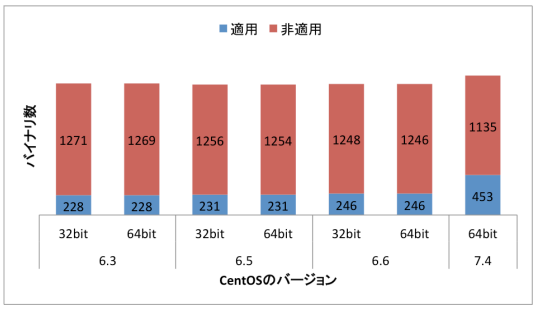


図 7　CentOS における PIE の適用状況

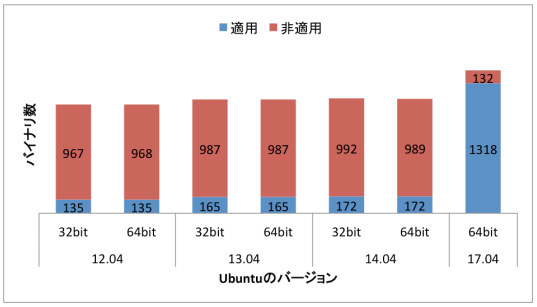


図 8　Ubuntu における PIE の適用状況

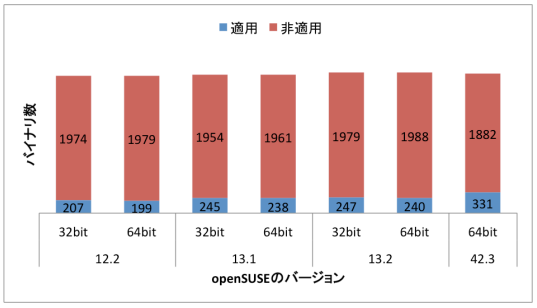


図 9　openSUSE における PIE の適用状況

#### 4.1.3　PIE の適用状況

図 7〜図 9 は PIE の適用状況である．

Ubuntu17.04 以外のディストリビューション全てで，PIE が適用されているバイナリの割合は非適用に比べて低かった．適用率の変遷は概ね横這いであるが，各ディストリビューションの最新バージョンでは適用率が上昇している傾向にある．

#### 4.1.4　Automatic Fortification の適用状況

図 10〜図 12 は Automatic Fortification の適用状況である．

どのディストリビューションにおいても，Automatic Fortification が適用されているバイナリの

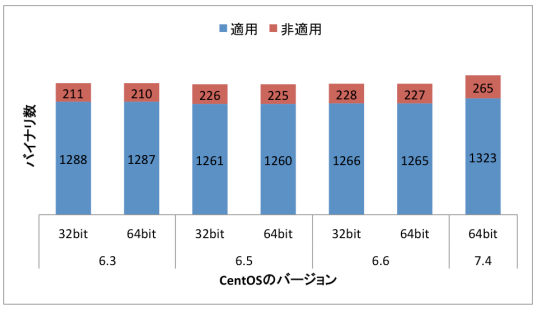


図 10　CentOS における Automatic Fortification の適用状況

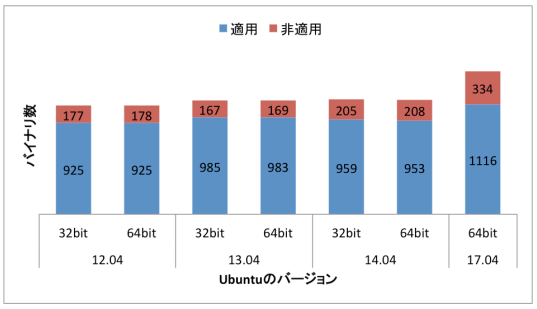


図 11　Ubuntu における Automatic Fortification の適用状況

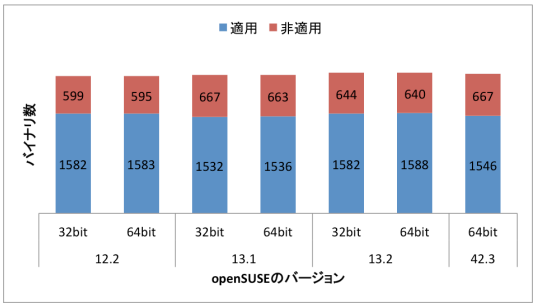


図 12　openSUSE における Automatic Fortification の適用状況

割合は非適用に比べて高かった．他の対策技術の適用状況と違い，最新バージョンでの適用率の上昇は見られなかった．

#### 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 への変化を弱化と呼ぶ．

**表 3** に 32bit における強化したバイナリの数を，**表 4** に 32bit における弱化したバイナリの数を示す．一定数のバイナリが強化されている一方で，弱化されているバイナリも一定数存在することがわかる．

### 4.2　パッケージのビルド日の調査結果

各ディストリビューションのパッケージ数を**表 5** に示す．CentOS は約 1100 個のパッケージがあり，openSUSE は約 1500 個のパッケージがあった．

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程度離れているのかを調査した (**表 6** 参照)．

CentOS のパッケージのビルド日はリリース日から平均 1 年以上離れており，openSUSE のパッケージのビルド日はリリース日から平均 50 日以内であることがわかる．

本論文ではさらに**図 A·1**，**図 A·2** としてパッケージのビルド日をプロットした．

対象のディストリビューションは CentOS の 32bit の 3 バージョン，openSUSE の 32bit の 3 バージョンである．縦軸には調査対象のディストリビューションの各バージョンに含まれるパッケージの和集合をとり，得られたパッケージを並べている．各点が 1 つのパッケージのビルド日を示している．

CentOS のプロット図から，一部のパッケージはリリース直前にビルドされていることがわかる．しかし，一定数のパッケージは過去にビルドされたパッケージを利用している．具体的には，調査対象の CentOS の各バージョンに含まれるパッケージの和集合をとり，得られた 1123 個のパッケージの内，522 個のパッケージは CentOS6.3 がリリースされる以前にビルドされたパッケージであり，CentOS6.5 および 6.6 は過去にビルドされたパッケージを使用しつづけていることがわかる．

openSUSE のパッケージは CentOS とは違い，リリース直前にビルドされていることがわかる．

表 3　32bit における強化されたバイナリの数

| ディストリビューション | 比較バージョン | RELRO | SSP | PIE | Automatic Fortification | 共通のバイナリ数 |
|---|---|---|---|---|---|---|
| CentOS | 6.3 と 6.5 | 11 | 6 | 2 | 0 | 1438 |
| | 6.3 と 6.6 | 25 | 7 | 16 | 0 | 1438 |
| | 6.5 と 6.6 | 14 | 1 | 14 | 0 | 1477 |
| Ubuntu | 12.04 と 13.04 | 52 | 0 | 29 | 16 | 1048 |
| | 12.04 と 14.04 | 54 | 4 | 32 | 20 | 1002 |
| | 13.04 と 14.04 | 3 | 5 | 3 | 9 | 1096 |
| openSUSE | 12.2 と 13.1 | 25 | 17 | 25 | 5 | 1915 |
| | 12.2 と 13.2 | 34 | 45 | 24 | 22 | 1817 |
| | 13.1 と 13.2 | 10 | 32 | 2 | 21 | 2075 |

表 4　32bit における弱化されたバイナリの数

| ディストリビューション | 比較バージョン | RELRO | SSP | PIE | Automatic Fortification | 共通のバイナリ数 |
|---|---|---|---|---|---|---|
| CentOS | 6.3 と 6.5 | 0 | 3 | 0 | 4 | 1438 |
| | 6.3 と 6.6 | 0 | 3 | 0 | 4 | 1438 |
| | 6.5 と 6.6 | 0 | 0 | 0 | 0 | 1477 |
| Ubuntu | 12.04 と 13.04 | 6 | 7 | 6 | 2 | 1048 |
| | 12.04 と 14.04 | 46 | 11 | 6 | 40 | 1002 |
| | 13.04 と 14.04 | 43 | 6 | 0 | 41 | 1096 |
| openSUSE | 12.2 と 13.1 | 1 | 8 | 0 | 1 | 1915 |
| | 12.2 と 13.2 | 1 | 9 | 1 | 12 | 1817 |
| | 13.1 と 13.2 | 0 | 1 | 1 | 13 | 2075 |

表 5　調査したパッケージの総数

| CentOS | | | openSUSE | | |
|---|---|---|---|---|---|
| 6.3 | 6.5 | 6.6 | 12.2 | 13.1 | 13.2 |
| 1078 | 1100 | 1111 | 1442 | 1508 | 1580 |

表 6　ビルド日とリリース日の差の平均

| CentOS | | | openSUSE | | |
|---|---|---|---|---|---|
| 6.3 | 6.5 | 6.6 | 12.2 | 13.1 | 13.2 |
| 369 | 603 | 729 | 46 | 48 | 30 |

ただし，gpg-pubkey というパッケージだけは他のパッケージと違い，リリース日の数年前にビルドされている．偶然このパッケージだけがリリース前にビルドされていなかった可能性，タイムスタンプの値が正しくない可能性がある．

## 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 [5]．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 32bit 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，Ubuntu14.04 32bit のソースパッケージ（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 4 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際に，弱化するバイナリが一定数存在することがわかった．

#### 5.2.1 Automatic Fortification が弱化した理由の考察

特に，Ubuntu12.04 と Ubuntu14.04 間では，Automatic Fortification が弱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 が弱化すると同時に，RELRO も弱化していた．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．

#### 5.2.2 PIE が弱化した理由の考察

さらに，PIE が弱化したバイナリも調査した．Ubutun12.04 と Ubuntu14.04 間では，PIE が弱化しているバイナリが 6 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から，明示的に PIE が無効化されていることが分かった [13][14]．

#### 5.2.3 SSP が弱化した理由の考察

最後に，SSP が弱化したバイナリも調査した．Ubuntu12.04 と Ubuntu14.04 間で，SSP が弱化したバイナリが 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（colord，gcalctool[*1]）に属する．これら 3 個のバイナリの，ソースコードを確認した．その結果，SSP の適用条件に当てはまらないことがわかった．このことから，SSP のコンパイルオプションを指定しても 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．これはバグである可能性が高い．

### 5.3 PIE の適用率が低い理由

図 7〜図 9 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，32bit の Linux 環境における PIE 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 における

[*1] Ubuntu14.04 では gnome-calculator に変更された．

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 [15].

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 [16] 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[17] 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のは Ubuntu と同様の理由だと推測する．

### 5.4 32bit 版と 64bit 版の比較

図 1〜図 12 が示すように，32/64bit 版で対策技術の適用率に大きな差はなかった．この結果から，今回調査したディストリビューションでは，32/64bit 版で同一のコンパイルオプションを適用していると推察できる．5.3 節で述べられているように，32bit では PIE 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を招くが，64bit ではその限りではない．そのため，本来は 64bit 環境では PIE を有効にするべきである．

### 5.5 パッケージのビルド日の調査

図 A·1，図 A·2 からパッケージのビルド日の分布がわかる．openSUSE ではディストリビューションのリリース前に大多数のパッケージをビルドし直していることがわかる．

しかし，CentOS ではリリース前にビルドを行なっていないパッケージが多数存在していることがわかる．コンパイラは日々新たなセキュリティ技術を導入しており，デフォルトで適用される対策技術も変化する．そのため，新たにビルドし直さない限りセキュリティ技術を享受することはできない．以上より，CentOS は openSUSE より，この点で安全性が低いといえる．

### 5.6 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ことが推察される．

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[18] の Heap-based Buffer Overflow 脆弱性がある [19]．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. 今後の課題

今後の課題として，今回行えていない Ubuntu のビルド日の調査がある．さらに，Ubuntu12.04 と 14.04 以外の弱化したバイナリの調査がある．また，ソースコードと実際のバイナリの違いの調査がある．具体的には，ソースコード解析およびコンパイラオプションの調査を行い，実際のバイナリとの差異を見ることで，WYSINWYX を示すことができると考えている．

## 7. まとめ

本論文では，主要な 32/64bit の 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，Ubuntu，openSUSE のバイナリ数は，それぞれ，10,542 個，8,284 個，15,424 個で，総数 34,250 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
(4) 過去にビルドされたパッケージを利用し続けているディストリビューションが存在した

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] 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

[5] 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)

[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] enables PIE, which often doesn't work on odd platforms, https://bugs.freedesktop.org/show_bug.cgi?id=16621

[14] D-Bus 1.11.14 (2017-06-29), https://dbus.freedesktop.org/doc/NEWS

[15] 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.

[16] Mathias Payer. Too much PIE is bad for performance. In Proc. of ETH Zurich, Department of Computer Science Technical Report 766. 2012

[17] Ubuntu Wiki: Security/Features, https://wiki.ubuntu.com/Security/Features

[18] Dnsmasq,
http://www.thekelleys.org.uk/dnsmasq/doc.html
[19] 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

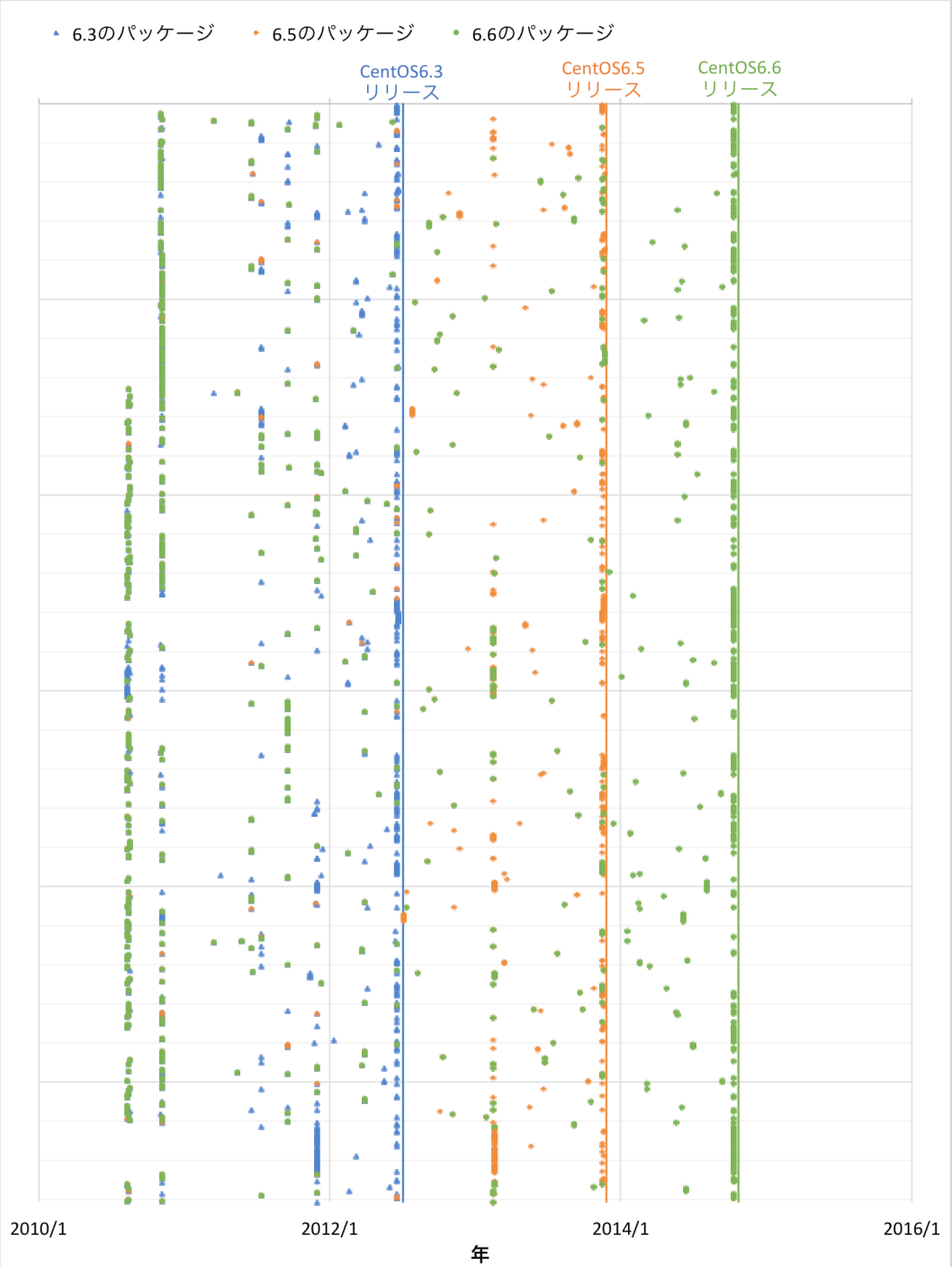

図 A·2　openSUSE のパッケージのビルド日のプロット図