社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DPS－122（58）
2005－CSEC－28（58）
2005／3／23

改良例外ハンドラによる実時間オーバーフロー防御システム

安藤類央　武藤佳恭
慶應義塾大学政策メディア研究科
〒２５２０８１６　神奈川県藤沢市遠藤５３２２
{ruo,takefuji}@sfc.keio.ac.jp

あらまし　本論文では、例外ハンドラを改良し、オーバーフローを起こしたプロセスを検出し、実時間で停止する手法を提案する。プロセス構造を変更することで、適切な個所にブレークポイントを設定し、分岐命令処理前後のレジスタの遷移を検査することにより、脆弱性の種類に関わらず不正アクセスによるプロセスの変異を制御する。例外ハンドラを改良しているため、検出に加えて、実時間での対応処理を組み込むことができる。提案手法は、ソフトウェアとカーネルのリビルドを行わず、同時に感染したプロセスの停止を可能なところに特徴がある。評価実験では、検出・防御実行時の負荷を測定し、適切な負荷で提案システムが稼動することが明らかになった。

Improced exception handler for real-time overflow prevention

Ruo Ando Yoshiyasu Takefuji
Graduate School of Media and Governance,Keio University,
5322 Endo Fujisawa, Kanagawa, 252 Japan
{ruo,takefuji}@sfc.keio.ac.jp

Abstract: In this In this paper, we introduce a improved exception handler, based on debug and instruction trace technology for real-time malicious process nullification. Proposal system, which is relied on the concept of process debugging is applied for the prevention of buffer overflow, which makes it possible to nullify the infected process without rebuilding the application. Previously, there is no technology that the some kind of control such as stopping for malicious process without recompiling source code. In experiment, it has been validated that the load stress testing of detecting buffer overflow is reasonably low in the proposal system.

１　はじめに

ここ数年の各産業のインターネットの利用率の増加は著しい。情報化社会とは、一面ではＩＴ産業が新規産業として位置付けられるということだけでなく、電子政府への取り組み、企業のERP,CRM等の積極的な導入に象徴されるように、産業全体がインターネットという開放的な情報通信システムによって有機的に接合され、その連関を再編成していると解釈することができる。しかし同時に、軍や学術機関の研究から端を発したインターネットが民間の産業と生活に普及にしていくにつれて、これからの課題は情報通信システムのセキュリティをどのように確保していくかに重点がおかれつつある。特に、ソフトウェアのバグを利用した不正アクセスが社会問題化しているケースが多くなっている。

２　関連研究

不正プロセスの検出には、静的に解析をする方法と、動的に検出する方法があり、前者は、対象となるコードがメモリにロードされる前に、データマイニングやヒューリスティックなどの手法を使って解析する方法で、後者はコードをメモリにロードし、シミュレーションあるいはランタイムで検出を行う手法である。

２．１　静的検出法

静的検出法は、プログラムを実行せずに検出が可能な方法であり、パターンマッチング法を中心として、比較方式とヒューリスティック方式に分かれる。コンペア法／チェックサム法／インテグリティチェック法[3]は、未感染のプログラムの情報を格納しておき、監視対象のコードの変化を検出する。パターンマッチング法[4]は、感染コードの特徴的なバイトコードが、対象となるプログラムに存在するか検査する方式である。ヒューリスティック方式[5]は、感染後の振る舞いを引き起こすと見なされ易いコードを解析することで、プログラムの異常を検出するものである。一般に、静的検出法は、ステルス技法の施されたコードや、ポリモーフィック／メタモーフィック化したコードに対しては、解析検出が困難であるという欠点が指

摘されることが多い。

２．２　動的検出法

ビヘイビア法は、ダイナミックヒューリスティック法とも呼ばれ、監視対象となるコードを実際に動作させ、あらかじめ定めておいたルールに適合する挙動を検出する方法であり、実行環境は実マシンあるいはエミュレータ[7]両方の場合がある。そのなかでも、バッファオーバーフローを引き起こすプロセスを検出する方法としては、コンパイラベース、カーネルベース、そしてプロセッサ機能と併用する非実行スタック防御の３種類が挙げられる。コンパイラベースの検出技術は、フレームポインタの隣接にカナリアと呼ばれる値を設置し、この値がバッファオーバーフロー時に改変されることを利用して検出を行うものである。代表的なものにStackGuard[8]とSSP[9][10]があるが、前者は実コードで検出を行うのに対して、後者は中間言語において検出を行うことから、StackGuardがインテルCPUに特化しているのに対し、サポートするアーキテクチャに幅があり、また、引数やフレームポインタの書き換えにも対応できる。以上２つに代表されるコンパイラベースの防御策には、バッファオーバーフローの脆弱性が発見されたアプリケーションやカーネルのリビルドを行わないといけない、また、バッファオーバーフローの検出はできるがこれに対して、プロセスの停止などの措置は取れないなどの欠点が指摘されている。

同様に、コンパイラベースの手法として、メモリオブジェクトのトレーシング機能を利用しているのが、Bounds Checking[11]である。Bounds Checkingではすべてのメモリ上のオブジェクトはその利用がプログラムによって追跡できるようにコンパイラが拡張されている。これによって、理論的には、バグの種類にかかわらずメモリの不正利用を検出することができ、
リターンアドレスの改善防止などの防御が可能になる反面、他の防御手法に比べると負荷が高く、脆弱性が発見されたアプリケーションやカーネルに対してはBounds Check用のライブラリをリンクし、アプリケーションをリビルドする必要がある。

Openwall[12][13]はバッファオーバーフローをカーネルの拡張で防御するもので、ユーザメモリエリアにおけるスタックの非実行化機能に特徴がある。カーネスベースの防御では、スタックのデータの実行を禁止することができ、バッファオーバーフローが発生しても不正なコードを実行することができないようになっている。カーネルレベルで保護機能が発揮されるため、アプリケーションを際コンパイルする必要がなく、実行時の負荷が比較的小さいという特徴があるが、カーネルのリビルドは必須で、StackGuard系の防御システムと同じくバッファオーバーフローの検出はできても不正プロセスの停止などの措置は取れないことがボトルネックとなっている。

３　提案システム

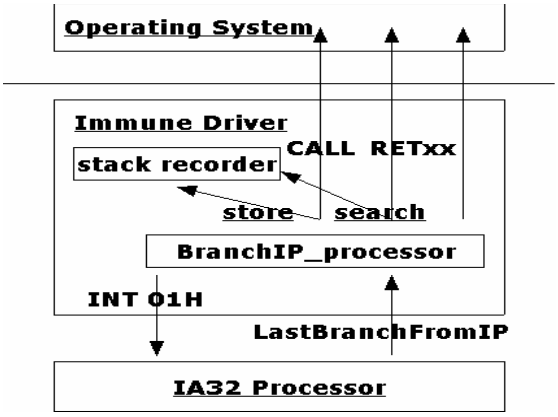
本論文では、オーバーフローに特化したデバッガをＯＳに組み込み、リアルタイムでオーバーフローを検出し、実時間でプロセスを停止させることが目標である。そのために、デバッグポイントをプログラムのロード時に適切な場所にインサートし、例外ハンドラを改良し、レジスタの遷移を検査するためのデータ構造を用意することでシステムの要件を達成した。これによって、

３１　改良例外ハンドラ

ＩＮＴＥＬプロセッサのマニュアルでは、同期割り込みのことを例外、非同期割り込みのことを割り込みと定義しており、ＩＡ３２系では数十種類の例外を発生するため、カーネルは個々の種類に対応した例外ハンドラを用意しており、提案システムではINT01Hに対応するデバッグ例外ハンドラを改良したものを実装している。これに併行して、割り込みディスクリプタにおけるベクタとハンドラのアドレスの対応関係の記述も変更する。本論文では、INT01Hを改良し、検出に必要な情報が発生する分岐命令だけをフィルタするためのImmuneINT01Hを実装した。

３.２　分岐命令レコーダ

前節で説明したトレースコールバック関数により、提案システムでは分岐命令実行時には、常に LastBranchFromIP が取得できるようになっている。分岐命令処理フィルタとは、この LastBranchFromIP から、リターンアドレス改竄に利用されるＣＡＬＬ／ＲＥＴ命令をフィルタし、ＥＩＰを保存検索するためのドライバである。その意味では、本フィルタは、カーネルデバッガと似た機能を持っているが、リターンアドレスの改竄チェックに特化しているという点に特徴がある。カーネルデバッガは、ＣＰＵ とＯＳの間で動作し、デバッグの対象となっているプロセスが停止すると、制御がＯＳからデバッガに移り、主にプロセッサのステータス情報などを取得することができる。提案システムでは、分岐命令処理フィルタを用いてこのＯＳの停止中にプロセスに対する処理を行うことができ、その結果として、不正プロセスの制御が可能となる。



以下に、分岐命令処理レコーダの構造体を示す。

```
Typedef struct IMMUME_STACK_LIST
{
 LIST_ENTRY m_ListEntry;
 ULONG ThreadID;
 ULONG *StackPointer;
 LONG CurrentStackLocation;
};
```

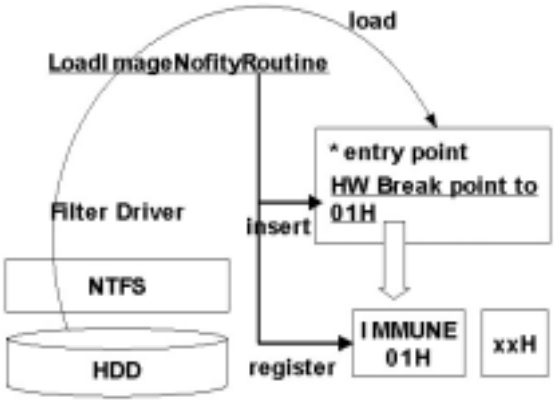
３３　プロセス構造ルーチン

改良例外ハンドラの適用を、防御対象のプロセスにのみ適用するために、本論文ではプロセス構造ルーチンを用いてロード時に登録されるコールバック関数を通じて上の操作を行う。ここで、プロセス構造ルーチンとは、実行ファイルがメモリにロードされプロセスとなる際に、コールバック関数やブレークポイントを挿入するために用いるドライバ関数のことを指す。

図３は、コールバック関数内での操作を示したものである。提案システムでは、プロセス構造ルーチンを用いて、実行ファイルのロード時にコールバック関数を登録する。Win32のドライバ関数を用いると、イメージが実行時にロードされるときに呼び出されるコールバック関数を登録することができる。

```
Vvoid LoadImageNotifyRoutine{
        PUNICODE_STRING FullImageName,
        HANDLE ProcessId,
        PIMAGE_INFO ImageInfo
}
```

上の関数は、提案システムで LoadImageNotifyRoutineを示したものである。同コールバック関数内で行われる作業は、５節で詳述するが、エントリポイントのアドレスの取得し、ハードウェアブレークポイントを挿入するというものである。これにより、分岐命令実行時には常に改良例外ハンドラが呼び出されることになる。この改良例外ハンドラ内で、次節で説明する分岐命令処理フィルタリングが行われる。



４　検出手順

提案システムの検出手順は、初期化、プロセスのロード、追跡、感染プロセスの停止の４つのフェーズがある。

４１　初期化

提案システムの初期化の手順は以下の通りである。

手順１－１：プロセスの指定

防御対象のプロセス名をレジストリから読み込む。

手順１－２：コールバック関数の登録

プロセス起動時に読み込まれるコールバック関数を登録する。同関数は、プロセスのロード時に時節で述べる処理を行う。

手順１－３：分岐命令レコーダの初期化

改良例外ハンドラ内で、（ＥＩＰ）実行アドレスの格納と検索を行う。

４２　プロセスのロード

ここでは、前節でのべたコールバック関数内での処理手順を示す。

手順２－１：ターゲットプロセスの判定

手順１－１から、読み込んだ名前をもとに、起動するプロセスが防御対象のものであるか判定する。

手順２－２：改良例外ハンドラの登録

割り込みディスクリプタを書き換えて、改良例外ハンドラを登録する。分岐命令処理フィルタは、この改良例外ハンドラ内で実行される。

手順２－３：エントリポイントの取得とHWブレークポイントの挿入

実行プロセスの最初に実行される関数(エントリポイント)のアドレスを取得し、その直後にINT01Hを呼び出すハードウェアブレークポイントを設定する。これにより、分岐命令実行時には常に、ImmuneINT01Hがコールされることになる。

４３　追跡

ここでは、分岐命令処理フィルタによって、関数呼び出しを監視し、スタックレコーダを用いて不正な実行アドレス遷移が起きないかチェックする。

手順３－１：関数呼び出し

関数呼び出しの際に、例外処理を発生させ、saved EIP(リターンアドレス)をスタックレコーダに保存する。

手順３－２：関数からの復帰

スタックレコーダ内での検索のためのコードを載せる。スタックレコーダに現在の実行アドレスが格納されていれば、バッファオーバーフロー発生はないとして、レコードを１つ削除する。検索の結果、実行アドレスが見つからなければ、saved EIP（リターンアドレス）と現在の実行アドレスが異なっており、バッファオーバーフローが起こったとして、次節の処理へ進む。

４．４　プロセスの停止

提案システムの特徴は、ローカル関数実行からの復帰の際に、例外処理を発生させているため、バッファオーバーフローが起こった時点でＯＳにプログラムコントロールを渡さずに、対応策を実施できる点にある。本論文では、バッファオーバーフローが起こったプロセスが不正な処理を行う前に、当該プロセスを停止させるために、ＩＮＴ０３Ｈ命令を適用した。具体的には、ＲＥＴ命令の次に実行される命令のアドレスを取得し、そこをＩＮＴ０３Ｈ命令に書き換える。

５　負荷テスト

本論文では、提案システムの評価実験として、ローカルバッファオーバフローの検出時の負荷と、防御対象をデータベースサーバのプロセスに設定し、処理内容に応じてシステムの負荷を測定した。提案システムは、Intel IA-32の命令トレース機能を有しているPentiumPro以上のプロセッサの中から、PentiumIIIを選択して実装を行った。OSは、Microsoft Windows 2000 SP0である。本節では、上記の実験環境におけるリモートバッファオーバーフローの検出時と、提案システム稼動時のシステムの負荷率を示した。提案システムの負荷は防御対象となるプロセスの処理内容によって決定する部分が大きく、プロセスを停止する処理自体の負荷は非常に小さいことがわかった。

提案システム稼動	**2.86**
提案システム停止	**2.57**

表１　オーバーフロー検出時の負荷率
（ＣＰＵ時間：%）

６　まとめと今後の課題

本論文では、CPUの例外デバッグ／命令トレース機能を用いて、分岐命令処理をフィルタし、実行アドレスの遷移をチェックすることで不正プロセスを実時間で制御する手法を提案した。提案システムは、バッファオーバーフローの防御に適用され、脆弱性を孕むソフトウェアのリビルドを行うことなく、不正プロセスの停止が可能であることが明らかになった。表２に、分岐命令処理フィルタを用いた本手法と関連研究でのべた他の手法との比較を示した。

	リビルド	制御(停止)	負荷
StackGuard/SSP	要	不可	中
Bounds Checking	要	可	大
OpenWall	不要	不可	小
提案手法	不要	可	中

表２　提案手法と他の防御手法との比較

再構築の不要な防御手法は、OpenWallと提案手法であるが、OpenWallはバッファオーバーフローを検出するのみで、攻撃を受けたプロセスの停止は不可能である。一方、不正プロセスの制御が可能な手法はBounds Checkingと提案手法である。この両者との比較において、提案手法はリビルドを必要としないところに特長がある。評価実験では、ローカルバッファオーバーフロー検出時の負荷率と、防御対象にデータベースサーバのプロセスを設定し、クエリ処理内容に応じて実験測定を行い、適切な負荷で提案システムが稼動することが明らかになった。今後の課題は、更に適切なブレークポイントの設定や、リターンアドレスの修復などが挙げられる。

謝辞

本論文の提案システム研究は、US Air Force Office of Scientific Research（助成番号AOARD 03-4049）の支援を受けている。また、文部科学省２１世紀ＣＯＥプログラム次世代メディア・知的社会基盤のメンバーから頂いた助言に謝意を記す。

参考文献

[1]Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 2A: Insruction Set Reference A-M (2004).
[2]Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 2B: Insruction Set Reference N-Z(2004).
[3]Intel Corporation: IA-32 IntelR Architecture Software Developer's Manual, Volume 3: System Programming Guide(2004).
[4]Gene H. Kim and Eugene H. Spafford:Tripwire A File System Integrity Checker,
ACM Conference on Computer and Communications Security , pp. 18-29(1994).
[5]Roesch, M:Snort - lightweight intrusion detection for networks.
Proceedings of Thirteenth Systems Administration Conference (LISA '99), pp. 229-238(1999).
[6]Symantec Corporation: Bloodhound Technology.
http://securityresponse.symantec.com/
[7]三宅崇之,白石義明,森井昌克 :仮想ネットワークを使った未知ウィルス検知システム,情報処理学会研究報告 ,No.022-016(2003).
[8]C.Cowan, C.Pu, D.Maier, J.Walpole, P.Bakke, S.Beattie, A.Grier, P.Wagle, Q.Zhang, and H.Hinton:StackGuard Automatic adaptive detection and prevention of buffer-overflow attacks,
In Proc. 7th USENIX Security Conference, pp 63--78(1998).
[9]Hiroaki Etoh: GCC extension for protecting applications
from stack-smashing attacks.
http://www.trl.ibm.com/projects/security/ssp
[10]江頭博明: Propolice スタックスマッシング攻撃検出法の改良,情報処理学会コンピュータセキュリティ,(2001).
[11]Richard W M Jones,Paul H J Kelly: Backwards-compatible bounds checking for array and pointers in C programs,
AADEBUG97(1997).
[12]Zeshan Ghory:Openwall Improving security with the openwall patch ,securityfocus(2002).
[13]Linux Openwall project.\\
http://www.openwall.com/
[14]Stephen Pearce:Viral Polymorphism,technical paper submitted for GSEC version 1.4b(2003).
[15]Szor, Peter and Ferrie, Peter:Hunting for Metamorphic,
Virus Bulletin Conference(2001).
[16]Y. Takefuji,K. Shoji, H. Miura, T. Kawade, T. Nozaki:
Security strategy and a proposal of driverware,JNSA Journal(2003).
[17]Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole: Buffer Overflows - Attacks and Defenses for the Vulnerability of the Decade,
DARPA Information Survivability Conference and Expo(2000).
[18]A. Pasupulati, J. Coit, K. Levitt, S.F. Wu, S.H. Li, R.C. Kuo, K.P. Fan:
Buttercup On Network-Based Detection of Polymorphic Buffer Overflow Vulnerabilities,9th IEEE/IFIP Network Operation and Management Symposium(2004).
[19]Ruo Ando, Hideaki Miura, Yoshiyasu Takefuji:File system driver filtering against metamorphic viral coding,
WSEAS transactions of information science and applications,Issue 4, Volume 1(2004).
[20]独立行政法人:情報処理推進機構:未知ウィルス検出技術に関する調査,
１５情経第１６７５号(2004)