3T-6

スタックの改ざん検知による侵入検知システムの一考察

長野　文昭†　鑪　講平††　田端　利宏†††　櫻井　幸一†††
†九州大学工学部電気情報工学科　††九州大学大学院システム情報科学府
†††九州大学大学院システム情報科学研究院　http://itslab.csce.kyushu-u.ac.jp/index-j.html

1. はじめに

既存のスタックオーバフローを防ぐ方法として，カナリアを利用した方法が提案されている．しかし，この方法には，メモリ上からカナリアの値を読み，その値を利用することでスタックオーバフローを回避できるという欠点がある[1]．その原因としてカナリアを用いた方式ではスタック上のデータの完全性を保証しないことがあげられる．そこで本論文では，スタック上のデータの改ざんを検出することで侵入を検知する手法を提案する．また，本方式は改ざんされたデータを復元し，攻撃を無効化できる特徴を持つ．

2. 提案方式

2.1. 概要

スタックオーバフローを利用した侵入行為とは，CやC++により書かれた実行コードに存在する攻撃方法で，プログラマが割り当てたバッファ領域より大きなデータを攻撃者が入力することで，攻撃者による任意のコマンドが実行可能になってしまう攻撃方法のことである．

このため，セキュリティの観点からはバッファ領域に入力があるすべての場合にスタック上のすべてのデータの完全性を検証するのが望ましい．しかし，最も脆弱なデータは，戻りアドレスのような本来サブルーチン内で値が変更しない値である[2]．そこで，本方式ではサブルーチン内で値が変わらないスタック上のデータの改ざんについて検証する．

サブルーチンの先頭でスタック上のデータについて検証子と呼ぶデータ構造体を作成し，サブルーチン終了直前に検証子を検証することでデータの改ざん検知をする．また，改ざんを検出されたデータにおいて，データを改ざん前の値に戻すことも可能である．データを改ざん前の値に戻すことによって，攻撃を無効化することができる．以下，まずこの方式において使用する検証メタデータについて説明し，次にその利用方法について説明する．

2.2. 検証子

検証に使用する検証子として，図1に示す検証子を採用する[3]．検証メタデータを構成する単位は検証子構造体配列(VSA: Verifier Structure Array)であり，それは検証子構造体(VS)を要素とする配列である．以下，VSのそれぞれの要素について説明する．

- 検証データ　検証対象ポインタが指しているデータのハッシュ値，もしくは検証対象ポインタが指しているデータそのもの．
- Alteredフラグ　データ完全性の検証時にデータの改ざんが検出されると立つフラグ．
- Data Linkフラグ　検証対象ポインタが指してい

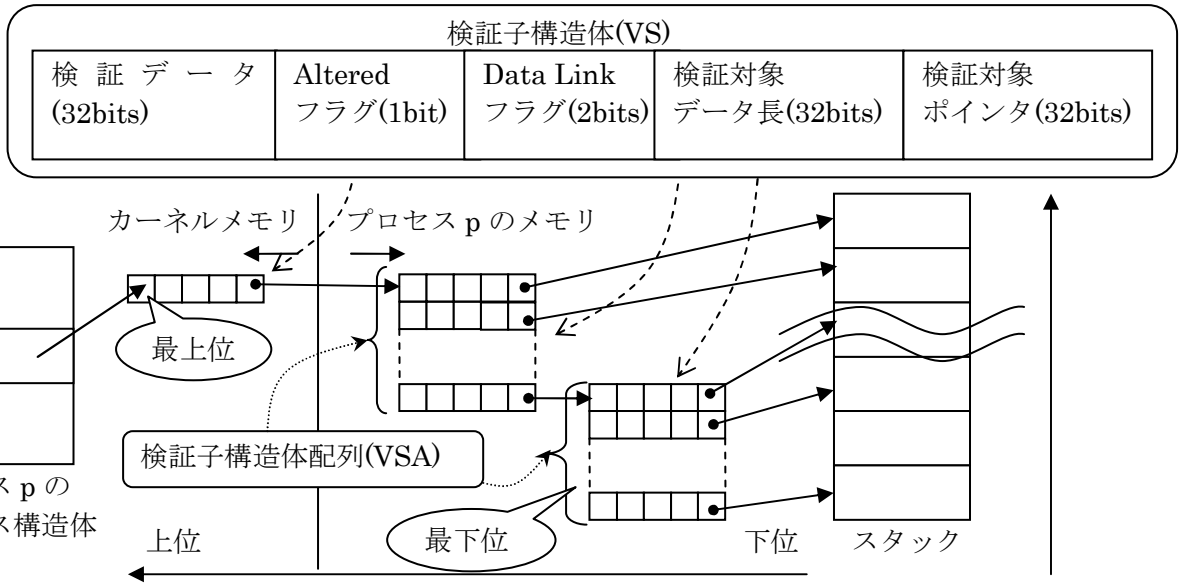

図 1　検証子

るデータがVSAであるか，スタック上のデータのハッシュ値であるか，スタック上のデータのそのものであるかを区別するフラグ．

- 検証対象データ長　検証対象データの長さ．
- 検証対象ポインタ　検証対象データへのポインタ．

2.3. スタック上のデータの完全性検証法

ここで，説明の都合上カーネルメモリ側を上位，ユーザメモリ側を下位と呼ぶ．よって，カーネルメモリに存在する検証子構造体が最上位，下位に検証子構造体配列をリンクしない検証子構造体配列が最下位となる．(図1参照)

サブルーチンの先頭で検証子を作成し，サブルーチンの終了直前に検証子の検証を行う．検証子作成方法，検証方法は次のように行う．

検証子登録方法

サブルーチンの先頭でスタック上のデータに対して最下位のVSAの要素VSを作成する．次にその上位のVSAの要素VSを作成し，カーネルに存在させる最上位のVSAに至るまで行う．この際カーネルに存在させる最上位のVSAに関してはシステムコールにより登録する．カーネル以外のVSAは，プロセスのヒープ領域に格納する．ただし，(今から述べる条件を例外条件と呼ぶ事にする．) Alteredフラグがたっているについては無視し，その際Data Linkフラグにより下位にVSAを持つことがわかった場合は，そのVSで連鎖される下位のVSAから最下位のそれに至るまでの全てのVSAにつて無視する．これは，下位のVSA改ざんによる検証コストの増大を引き起こす一種のDoS攻撃を回避するためである．

検証子の検証方法

サブルーチンの終了直前に最上位のVSAの要素VS全てについて検証データを作成し，格納されている検証データとの比較を行う．この二つのデータが同じであったら下位のVSAの完全性が保証されることとなる．次に最上位の下位のVSAで完全性が確認されたものについて同様に行い，最下位に至るかまたは完全性が損なわれたものが現れるまで順次下位のVSAについて同様に行う．改ざんが検出されたVSに関してはそのVSのAlteredフラグにチェックをつける．ただし，例外条件を考慮する．

上記のように行うことにより，たとえ，スタック上の値やプロセスのヒープ領域の検証メタデータを書き換えられたとしてもカーネルメモリのVSAを書き換えられない限り，スタック上のデータの完全性は保証される．なぜなら，カーネル上のVSAによりヒープ上のVSAの完全性が保証され，保証されたヒープ上のVSAによりスタック上のデータの完全性を保証することができるからである．つまり，スタックオーバフローが検出可能となるのである．

検証子利用方法

上記の検証方法が改ざんを検知した時にはユーザアプリケーションに対してシグナルが送られる．そのときの処理はユーザアプリケーションのポリシ次第である．また，改ざんが検知されたVSのData Linkフラグにより，このVSの検証データの欄に検証データそのものがある場合は検証データに格納されている値を使用し，データを改ざん前に戻すことも可能である．

3. 実装方法

本方式を実装には以下の4点が必要である．

1. カーネルにVSを登録するシステムコールの実装．
2. カーネルに登録したVSの完全性を検証するシステムコールの実装．
3. サブルーチン実行直後にユーザ領域に上記の検証子を作成する関数の実装
4. サブルーチン終了直前にユーザ領域の検証子を検証する関数の実装

1, 2はOSに新たにシステムコールを追加することで可能となる．3，4はコンパイラを改良し，サブルーチンの先頭に3の関数を，サブルーチンの終了直前に4の関数を実行させるようにすることで可能となる．また，上記の条件をみたすことによりユーザが望む任意の位置で任意のデータの完全性を検証することが可能となる．なぜならユーザが望む位置で上記の検証子を作成することができるためである．

4. おわりに

本論文では，関数の戻りアドレスやローカル変数を検証し，かつ，攻撃を無効化可能なスタックオーバフロー検知システムを提案した．今後の課題として，実装，本方式のヒープオーバフローの検知やフォーマットストリングバグを利用した攻撃の検知への適用がある．

5. 参考文献

[1]A. Zuquete , Stackfences: A RUN-TIME APPROACH FOR DETECTING STACK OVERFLOW，ICETE，2004.

[2]C. Cowan，S. Beattie，J. Johansen，and P. Wagle，PointGuard™: Protecting Pointers From Buffer Overflow Vulnerabilities， USENIX，2003.

[3]江頭徹，稲村雄，竹下敦，マルチタスク OS におけるメモリの保護-データ改ざん検出手法の提案-，電子情報通信学会技術研究報告，Vol.104，No.200，p27-34，2004.