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

2008－SLDM－134／2008－EMB－8（5）
2008 / 3 / 27

文字列に着目した情報フロー追跡によるインジェクション攻撃の検出

勝沼 聡[†] 塩谷 亮太[†] 入江 英嗣[††] 五島 正裕[†] 坂井 修一[†]

† 東京大学大学院 情報理工学系研究科
†† 科学技術振興機構
E-mail: †{katsu05,shioya,ern,goshima,sakai}@mtl.t.u-tokyo.ac.jp

あらまし クロスサイト・スクリプティング，SQLインジェクションなどを突く高次のインジェクション・アタックがセキュリティ上の脅威となっている．本稿で提案するSWIFT（String-Wise Information Flow Tracking）は，このようなアタックを防ぐハードウェア方式である．SWIFTでは，既存手法とは異なり，セマンティックな文字列操作の単位で処理することで，高い精度でアタックを検出する．

キーワード セキュリティ，脆弱性，インジェクション・アタック，プロセッサ，DIFT，文字列

String-Aware Information Flow Tracking to Detect Injection Attacks

Satoshi KATSUNUMA[†], Ryota SHIOYA[†], Hidetsugu IRIE[††], Masahiro GOSHIMA[†], and Shuichi SAKAI[†]

† Graduate School of Information Science and Technology, The University of Tokyo
†† Japan Science and Technology Agency
E-mail: †{katsu05,shioya,ern,goshima,sakai}@mtl.t.u-tokyo.ac.jp

Abstract High-level injection attacks, which exploit SQL injection and cross-site scripting, have a significant effect on computer security. We propose String-Wise Information Flow Tracking (SWIFT), which detects a wide range of these attacks. SWIFT propagates tags by string operations, which leads to high propagation accuracy.

Key words security, vulnerability, injection attack, processor, DIFT, string

1. はじめに

近年，サーバに対する不正アクセスによる被害が深刻になっている．Webページやシステムが改ざん，破壊されたり，サーバ内の機密情報が盗まれたりしている．このような不正なアクセスとしては，バッファオーバーフロー・アタックがよく知られているが，最近では，クロスサイト・スクリプティング，SQLインジェクションなどを突く，よりセマンティックなアタックが，特に深刻になっている．このようなバイナリの構造に依らないアタックは，バッファオーバーフロー・アタックなどのメモリ破壊系のアタックと対比して，高次の（high-level）インジェクション・アタックと呼ばれる[4]．図1は，CVE[3]に報告されたアプリケーションの脆弱性の中で，高次のインジェクション・アタックを引き起こす脆弱性の全報告数に占める割合を示したものであり，年々，増加傾向にあることが分かる．

このような高次のインジェクション・アタックを検出する手法として，Perlのテイントモードを発展させた，動的な情報フロー追跡方式（Dynamic Information Flow Tracking，以下では，**DIFT**と略す．）[6][5][4]が最近，研究されている．DIFTでは，ネットワーク等を介して入力されたデータにテイントと印付けし，そのテイント情報を，プログラムのデータの依存関係に従って伝播させる．そして，出力時に，テイントと印付けされたデータを検査することで，アタックを検出する．なお，この処理を行うのは，プロセッサのようなハードウェアであっても，インタープリタやVMのようなソフトウェアであってもよい．

ハードウェア・ベースのDIFTでは，ソフトウェア・ベースのDIFTとは異なり，適用できるプログラムがほとんど限定されず包括的である．また，実行速度のオーバーヘッドも小さい．しかし，伝播の精度は十分とは言えず，誤検出や検出漏れを引き起こしている．本稿では，SWIFT（String-Wise Information Flow Tracking）を提案する．SWIFTでは，ハードウェアで伝播を行うことで，既存のハードウェア・ベースのDIFT[4]と同様に，包括的かつ，実行速度のオーバーヘッドも小さい．さらに，既存手法とは異なり，テイント情報を命令単位ではなく，よりセマンティックな文字列操作の単位で伝播させることで高精度な伝播を実現する．

本稿の構成としては，まず，2章で，高次のインジェクショ

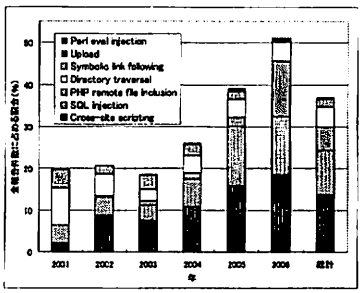

図 1 CVE に報告されたアプリケーションの脆弱性
Fig. 1 Vulnerabilities of applications in CVE

ン・アタックについて具体的に説明する．3 章で，DIFT について具体的な手法を挙げつつ，その利点や欠点について論じる．次に，4 章で，本稿で提案する SWIFT の基本的な方針について述べ，5 章で，詳細な動作を説明する．そして，6 章で，SWIFT の伝播精度に関する評価の結果と考察について述べる．

2. 高次のインジェクション・アタック

2.1 概　　要

脆弱性があるサーバ・アプリケーションでは，クライアントから入力されたデータを，ネットワークや，データベースなどのシステム・リソースへの出力の一部として使われる．すなわち，図 2 のように，入力データはプログラムに文字列として取り込まれた後，複製，結合，変換などの文字列操作で，文字列から文字列に移動し，その移動先の文字列が出力として使われる．高次のインジェクション・アタックでは，このようなプログラムに対し，その入力検査をすり抜けることで，プログラムが意図していない出力を行わせる．

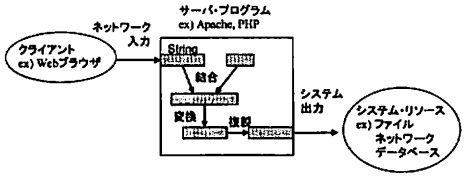

図 2 脆弱性があるサーバ・アプリケーション
Fig. 2 Vulnerable server application

2.2 SQL インジェクション

サーバ・アプリケーションでは，クライアントから受け取ったデータから SQL クエリを生成し，その SQL クエリをサーバ側のデータベースに送信されることがよく行われている．その時に，クライアントからのデータに対し適切なチェックを行っていないと，攻撃者によって SQL コマンド等を注入され，データベース内のデータが盗み見や改ざんされたりする．

例えば，サーバ・アプリケーションの一部に，

```
$cmd = "SELECT price FROM products
    WHERE name '".$name."'"
```

のような PHP コードがあったとする．このコードでは，クライアントからの入力を受け取った文字列 name がシステムが生成した文字列と結合され，文字列 cmd を生成される．そして，文字列 cmd を SQL クエリとしてデータベースに転送する．攻撃者は，

```
tmp';UPDATE products SET price = 0
WHERE = 'house
```

のような SQL コマンドを含んだ入力をすることで，

```
SELECT ... WHERE name = '
tmp';UPDATE ... 'house'
```

のような SQL 文が生成され，これがクエリとしてデータベースに転送されることで，データベースの改ざんが行われる．

3. 動的な情報フロー追跡（DIFT）

3.1 動　　作

動的な情報フロー追跡（DIFT）は，高次のインジェクション・アタックを統一のアルゴリズムで検出する．DIFT に関して様々な手法が提案されているのが，基本的に，DIFT では，実行しているプログラムのワード（バイト）に対して，テイントか否か識別を行う．まず，ネットワークなどを介して外部からプログラムに入力されたデータに，ソフトウェアでテイントと印付けする．そして，プログラム実行時に，そのテイント情報をデータの依存関係に従って伝播させる．例えば，

```
a = b;
  // b から a に伝播
a = b + c;
  // b と c から a に OR 伝播
```

のように，データ転送や演算が行われたときに，参照された変数から代入される変数へテイント情報を伝播させる．テイント情報の伝播は，手法によって，ソフトウェアまたは，ハードウェアによって行われる．そして，プログラムの出力時，すなわち，ファイルやデータベース等へのアクセスや，ネットワークとの通信などが行われた時に，表 1 のように，テイント情報が付加したデータの検査を行い，条件を満たしたらアタックとして検出する．

表 1 DIFT の出力検査
Table 1 Checkings in DIFT

出力先	検査項目	検出するアタック
ネットワーク	テイントな スクリプトタグ	Cross-site scripting
データベース	テイントな SQL コマンド	SQL injection
ファイル	公開ファイル以外の テイントなパス （システムコール open, execve 等）	PHP remote file inc. Directory traversal Symbolic link follow. Command injection

3.2 伝播精度

DIFT のテイント情報伝播の精度について述べる．テイント情報の伝播は，データの依存関係に従って行う．しかし，データの依存関係には，データの転送や演算など，直接的なデータの依存関係がある場合だけでなく，アドレス依存，暗黙的依存と呼ばれる間接的な依存関係が存在する．以下，それぞれについて説明する．

まず，アドレス依存とは，

```
dest = translation_table[index];
  //index から dest に依存
```

のような，変換テーブルのインデックスから，変換先のデータへの依存である．アドレス依存は，エンコード，デコード等の文字コード変換や，大文字小文字変換など多くの変換で存在する．特に，ネットワーク・アプリケーションでは，入力データに対するデコードや，出力データに対するエンコードが多く行われる．次に，暗黙的依存とは，

```
if(cond == '+')
  dest = ' '; // cond から dest に依存
```

のような，分岐条件として用いられる変数から，分岐中に代入される変数への依存である．暗黙的依存も，URL エンコード・デコード等の変換において見られる．このような依存関係は，直接的なデータの依存関係と異なり，テイント情報を全ての依存先に伝播させると誤検出が生じるため対処が難しく，[4] [7] [8] などの既存の DIFT でも問題点として挙げられている．

3.3 既存手法

既存手法としては，インタプリタ方式とハードウェア方式が挙げられる．以下で，それぞれについて述べる．

3.3.1 インタプリタ方式

インタプリタ方式としては，Perl のテイントモードが知られているが，最近では，高次のインジェクション・アタックの検出を目的とした手法が検討されている．例えば，PHP インタプリタ上に実現したものとして，Pietraszek らの手法 [6]，Nguyen-Tuong らの手法 [5] などがある．インタプリタでは，データの属性や，標準関数の操作などを把握しているため伝播の精度は比較的高い．しかし，間接的なデータの依存関係を根本的に解決しているわけではなく，伝播精度は十分とは言えない．

3.3.2 ハードウェア方式

ハードウェア・ベースの方式としては，当初，バッファ・オーバーフロー等の低次のインジェクション・アタックを検出する手法が提案されたが，最近では，Raksha [4] など，高次のインジェクション・アタックも検出する手法が提案されている．

Raksha は，メモリ，レジスタ，キャッシュ等の記憶領域に対してワード単位のタグを追加し，そのタグによって，データのテイント情報の記録する．まず，入力データが取り込まれたメモリ領域のタグに，OS 等のソフトウェアで印付けする．そして，データ転送命令や演算命令時に，

```
load R1 = [R2]    //[R2] から R1 に伝播
add R1 = R2 + R3 //R2，R3 から R1 に OR 伝播
```

のように，ソースからデスティネーションオペランドに，ハードウェアでテイント情報の伝播を行い，出力時に，ソフトウェアで検査を行う．

Raksha は，ハードウェアで伝播を行うことで，ソフトウェア・ベースの方式とは異なり，プログラムがインタプリタ上で動作しているかどうかや，ソースコードが入手可能であるかに依存しない．したがって，ほとんど全てのプログラムに対して適用可能であり，包括的である．また，ハードウェアで命令実行と並行してテイント情報が伝播されるので，実行速度のオーバーヘッドも小さい．

しかし，伝播精度に関しては，他の手法と同様に，アドレス依存，暗黙的依存のため不十分である．Raksha は，暗黙的依存に対しては全く伝播を行わない．また，アドレス依存に関しては，

```
load R1 = [R2] // R2 から R1 に伝播
```

のように，アドレス伝播と呼ばれる，ソースアドレスからデスティネーションオペランドに伝播させる機構をハードウェアとして提供している．しかし，誤検出の多さから，標準では使用されていない．

4. SWIFT の基本方針

4.1 方　　針

本稿で提案する SWIFT は，テイント情報の伝播方法以外は，Raksha と同様の方法を取る．SWIFT では，Raksha と同様に，ハードウェアでテイント情報を伝播させる．このことで，包括的，かつ，実行速度のオーバーヘッドを小さくする．また，Raksha とは異なり，命令のソースからデスティネーションに対し伝播を行わない．その代わり，よりセマンティックな文字列操作を識別し，文字列から文字列へデータが移動する時に，その文字列が格納されたメモリ領域のテイント情報を伝播させる．このことで，SWIFT は，命令レベルのデータの依存の仕方に透過的な伝播を行う．つまり，Raksha で伝播精度を下げる要因になったアドレス依存や暗黙的依存に対しても対応可能にする．

SWIFT では，まず，プログラムに対して外部から入力が行われたら，その入力データが取り込まれたメモリ領域に対して，ソフトウェアでテイントと印付けする．そして，そのテイント情報を伝播するために，SWIFT では，まず，インオーダなメモリアクセス（ロード及びストア）のパターンを，動的に取得する．そして，そのパターンに基づき，アクセスされたロード及びストアデータが文字列の要素か否か識別する．また，文字列の識別と同時に，ロードされた文字列からストアされた文字列に移動が行われているかを識別する．そして，移動が行われていると識別されたら，その移動元から移動先にテイント情報を伝播させる．そうして，出力時に，ソフトウェアでテイント情報の検査を行う．

表 2　メモリアクセス・パターン
Table 2　Memory access pattern

コマンド	ld	ld	ld	st	st	ld	st	ld	st	ld	st	st	st	st
アドレス	40	32	33	14	64	20	65	34	52	35	76	34	66	67
サイズ	4	1	1	4	1	1	1	1	4	1	4	4	1	1
テイント	0	1	1	-	-	0	-	1	-	1	-	-	-	-

4.2 文字列の識別

一つの文字列の要素は，一般的にメモリの連続領域に格納されており，文字列要素へのメモリアクセスは連続的に行われる．そこで，本手法では，アドレスなどのメモリアクセスのパター

ンから，連続的に増加または減少するアドレスの列を検出し，そのアドレス列を文字列へのアクセスとみなす．例えば，表 2 のような，メモリアクセスが行われた時，連続的に増加するロードアドレスの列 32〜35 を，同じ文字列の要素のロードと識別し，また，ストアアドレスの列 64〜67 を，同じ文字列要素へのストアを識別する．

4.3 文字列操作の識別

文字列から文字列への移動は，文字列の複製や，抽出，結合のような単純なデータの転送から，文字コード変換や，大文字・小文字変換のような計算を要する変換まで様々であるが，その特徴として，移動元の文字列の各要素から，移動先の文字列の各要素が生成されるという点が挙げられる．すなわち，メモリレベルでは，移動元の文字列要素のロードと，移動先の文字列要素へのストアが繰り返し行われる．そこで，本手法では，ある文字列の要素のロードと，他の文字列の要素へのストアが一定範囲内のサイズで交互に行われたら，その二つの文字列間で移動が行われているとみなす．

例えば，表 2 では，アドレス 32〜35 から成る文字列とアドレス 64〜67 から成る文字列は，データサイズ 2 バイトずつ，交互にロードとストアが行われていることから，文字列操作が行われているとみなす．そして，アドレス 32，33，34，35 のデータの移動先を，各々，アドレス 64，65，66，67 として，テイント情報を伝播させる．

5. SWIFT の詳細な動作

5.1 ハードウェア構成

SWIFT では，メモリやキャッシュに対してバイト当たりのタグを追加し，バイトデータのテイント情報を記憶する．また，テイント情報の伝播のために，図 3 のように，Load String Table（LST），Store String Table（SST）という二つの CAM のテーブルを追加する．LST は，ロード命令時にその情報を格納するテーブルで，その各エントリには，それぞれ異なる文字列の情報が格納される．また，SST は，ストア命令時にその情報を格納するテーブルである．その各エントリには，LST と同様に，それぞれ異なる文字列の情報が格納される．そして，SST のエントリ内の各 ID のブロックには，その ID に対応づけられた文字列（LST に情報が格納）との文字列操作に関する情報が格納される．

5.2 文字列の識別

文字列の識別は，連続的に増加または減少するアドレスの列を検出することで行う．検出方法については，プリフェッチのためのアドレスのストライド予測機 [2] に基づいている．ストアされた文字列を識別するために，SST では，各々のエントリに，文字列で最後にアクセスされた要素のアドレスとデータサイズ，その文字列のアクセス順（昇順または降順）を記憶し，ストア命令が行われたときに，そのアドレス，データサイズと各エントリの上記の項目を比較することで，そのストアデータが文字列の次の要素であるかを判定する．ロードされた文字列の識別も同様に，LST を用いて行う．ただし，ロードされた文字列は，テイント情報を持つ要素を含む場合のみ識別される．

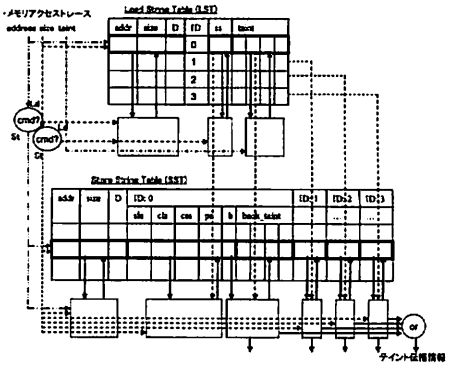

図 3 SWIFT のハードウェア構成
Fig. 3 Hardware structure of SWIFT

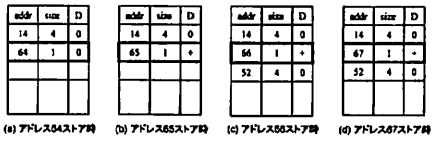

図 4 文字列の識別
Fig. 4 String identification

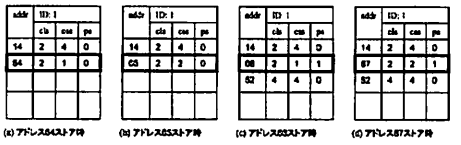

図 5 文字操作の識別
Fig. 5 String operation identification

例えば，表 2 のようなメモリアクセスが行われた時の，SST の更新の様子が図 4 である．SST の *addr*，*size* は，文字列で最後にアクセスされた要素のアドレス，データサイズを記録し，*D* は，その文字列のアクセス順を記録する．アドレス 64 へのストア時に，同じ文字列が SST に存在しないので，図 4 (a) のように，エントリが初期される．そして，アドレス 65 へのストア時に，アドレス 64 の文字列の次の要素と判定され，図 4 (b) のように，対応するエントリが更新される．さらに，アドレス 66，67 へのストア時にも，同じ文字列の次の要素と判定され，図 4 (c) (d) のように，先程と同様のエントリが更新される．このようにして，ストアされた文字列の識別が行われる．

5.3 文字列操作の識別

文字列操作の識別は，文字列要素のロードとストアが一定範囲内のサイズで交互に行われるのを検出することで行う．厳密には，次のステップ 1，2 がその順序で 2 回行われると，文字列 A と B は文字列操作が行われているとみなす．

ステップ 1． 文字列 A が 1〜4 バイト，ロード．

ステップ 2． 文字列 B に 1〜4 バイト，ストア．

ステップ 1，2 の文字列 A，B へのメモリアクセスを合わせて，操作フェーズと呼ぶ．例えば，図 5 (a) では，アドレス 32，33，64，65 へのアクセスや，アドレス 34，35，66，67 へのアクセ

スが，それぞれ操作フェーズである．文字列操作を識別するために，SST のエントリの各 ID ブロックに，そのエントリに対応した文字列と，その ID に対応した文字列の関係を記憶しておく．すなわち，現在及び一つ前の操作フェーズのロード及びストアサイズを記録する．そして，その値から文字列操作と識別されたならば，現在及び一つ前の操作フェーズのストア先にテイント情報を伝播させる．

例えば，表 2 のようなメモリアクセスが行われた時の，SST における ID:1 のブロックの更新の様子が図 5 である．SST の各 ID ブロックの *cls*，*css* は，現在の操作フェーズのロードサイズ，ストアサイズを記録し，*ps* は，一つ前の操作フェーズのロード及びストアサイズの情報を記憶する．LST の ID が 1 のエントリには，アドレス 32 を基点とした文字列（以下，文字列 I）が対応付けされているとする．アドレス 64 へのストア時，文字列 I では，既にアドレス 32，33 がロードされているので，図 5 (a) のように，エントリが初期される．そして，アドレス 65 へのストア時には，文字列 I はアクセスされていないので，操作フェーズは変わらず，図 5 (b) のように，そのエントリが更新される．さらに，アドレス 66 へのストア時には，文字列 I では，新たに，アドレス 34，35 がロードされたので，操作フェーズが遷移し，図 5 (c) のようにエントリが更新される．この時，文字列 I から，アドレス 64 を基点とした文字列へ移動が行われているとみなされる．

表 3　文字列操作関数による評価 1
Table 3　Evaluations by String Operation Functions 1

	関数	操作	伝播の有無		
			Rn	Ra	S
a	substr	抽出	○	○	○
b	"."演算子	結合	○	○	○
c	ereg_replace	挿入	○	○	○
d	ereg	照合	−	−	−
e	strtoupper	大文字変換	×	○	○
f	strtolower	小文字変換	×	○	○
g	urlencode	URL encode	×	○	○
h	urldecode	URL decode	○	○	○
i	base64_encode	base64 enc.	×	○	○
j	base64_decode	base64 dec.	×	○	○

Rn: Raksha_n　Ra: Raksha_a　S: SWIFT

6. 評　　価

6.1 方　　法

SWIFT の評価は，x86 エミュレータ Bochs 3.5 上で Red Hat Linux 8.0，apache 2.0，PHP 4.2 などを実行させることで行った．Bochs を変更し，SWIFT と Raksha を実装した．Raksha は，ハードウェア・ベースの DIFT で，最も精度が高い手法である．SWIFT は，LST，SST のエントリ数をそれぞれ 4，64 とした．また，Raksha については，アドレス伝播を行わない方式（以下，Raksha_n）とアドレス伝播を行う方式（以下，Raksha_a）の両方を実装した．Raksha は，ワード単位のタグを用いているが，公平な評価を行うため，SWIFT に合わせバイト単位のタグを実装した．評価プログラムは，文字列操作の関数とネットワーク・アプリケーションを用い，伝播精度に関して評価を行った．

6.2 結　　果

6.2.1 文字列操作

まず，表 3 のような PHP の文字列操作関数を用いて，伝播精度の評価を行った．評価方法としては，システムコール read 時に，入力データの特定箇所（表 4 の下線部分）にテイントと印付けし，各関数を実行した後，システムコール write で出力されるデータのテイント情報を検査することで行った．その結果，表 3 表 4 のようになった．表 4 では，出力への正しい依存の仕方と，各手法による出力への伝播の様子（ただし，"−" は正しい依存の仕方と同じ．）を表している．下線部分が，テイントな箇所である．

(a) ～ (c) のようなデータ転送のみを行うプログラムについては，全ての手法で正確に伝播させることができた．また，(d) のような，入力と出力に依存関係がないプログラムでは，全ての手法で伝播しないことが確認できた．一方，(e) ～ (j) のような計算を要する変換については，各々の手法で異なる出力結果が得られた．Raksha_n に関しては，多くの検出漏れが生じた．これは，これらのプログラムの変換の過程で，アドレス依存や暗黙的依存が生じているからである．大文字・小文字変換，base64 エンコード・デコードでは全てのデータの変換でアドレス依存が生じ，また，URL エンコードではアルファベット以外のデータの変換でアドレス依存が生じている．また，URL デコードでは，"＋" の変換で暗黙的依存が生じている．Raksha_a に関しては，全てのプログラムで伝播させることができた．ただし，URL エンコードについては，Raksha_n と同様に，暗黙的依存がある箇所で，部分的に検出漏れが生じた．一方，SWIFT では，全てのプログラムで伝播を確認することができた．しかし，(g) (i) では部分的に伝播できない箇所が存在した．

6.2.2 ネットワーク・アプリケーション

次に，実際のネットワーク・アプリケーションを用いて評価を行った．評価に用いたプログラムは，表 5 のようなプログラムであり，各々，高次のインジェクション・アタックを引き起こす脆弱性を持っている．各プログラムに対して，その脆弱性を突くアタックを行い，各手法で検出，誤検出の有無を調べた．なお，出力の検査は，全ての手法で，表 1 のように行った．

その結果，表 5 のようになった．全ての手法で全てのアタックを検出することができた．また，SWIFT 及び，Raksha_n では誤検出が生じなかった．表 6 は，SWIFT において，アタックが検出された出力部分を表している．下線部分が，テイント付けされた箇所である．全てのプログラムで，入力に依存した出力箇所のみテイント付けされ，それ以外の箇所にはテイント付けされなかった．一方，Raksha_a では，プログラムの多くのメモリ領域がテイント付けされ，結果として，全てのプログラムで誤検出が生じた．

6.3 考　　察

Raksha_a では，単純なプログラムで，テイント情報を入力データに限定的に付加する場合には，高精度な伝播を行う．しかし，現実的なプログラムでは，多くの誤検出が生じ，現実的な手法とは言えない．このアドレス依存による誤検出は，[7] [4]

表 4　文字列操作による評価 2

Table 4　Evaluations by String Operation Functions 2

	入力	正しい依存の仕方	Raksha_n による伝播	Raksha_a による伝播	SWIFT による伝播
a	abcdefrhirklmno	defrhirk	–	–	–
b	abcde	PPPabcdePPP	–	–	–
c	abcdefrhirklmno	abcdePPPPPklmno	–	–	–
d	tonight	tomorrow	–	–	–
e	abcdefrhi	ABCDEFRHI	ABCDEFRHI	–	–
f	abcdefrhi	abcdefrhi	abcdefrhi	–	–
g	a<b>c;	a%3Cb%3Ec%3B	a%3Cb%3Ec%3B	a%3Cb%3Ec%3B	a%3Cb%3Ec%3B
h	a%62+%64e%66r%68i	ab defrhi	ab defrhi	ab defrhi	–
i	abcdefghi	YWJjZGVmZ2hp	YWJjZGVmZ2hp	–	YWJjZGVmZ2hp
j	YWJjZGVmZ2hpamts	abcdefghijkl	abcdefghijkl	–	–

表 5　ネットワーク・アプリケーションによる評価 1

Table 5　Evaluations by network applications 1

CVE	Program	Attack type	Raksha_n		Raksha_a		SWIFT	
			検出	誤検出	検出	誤検出	検出	誤検出
2005-0870	phpsysinfo 2.3	Cross site scripting	あり	なし	あり	あり	あり	なし
2003-0486	phpBB 2.0.5	SQL injection	あり	なし	あり	あり	あり	なし
2006-0983	Qwikiwiki 1.4.1	Directory traversal	あり	なし	あり	あり	あり	なし
2005-2380	phpSurveyor 0.98	Cross site scripting	あり	なし	あり	あり	あり	なし
2005-2398	phpSurveyor 0.98	SQL injection	あり	なし	あり	あり	あり	なし
2003-1435	PHPnuke 6.0	SQL injection	あり	なし	あり	あり	あり	なし

表 6　ネットワーク・アプリケーションの評価 2

Table 6　Evaluations by network applications 2

CVE	出力
2005-0870	: <script>alert('XSS');</script> is not
2003-0486	id = -1;DELETE * FROM table AND
2006-0983	data/../config.php
2005-2380	sid=1<script>alert('XSS');</script>
2005-2398	sid=1;DELETE * FROM table
2003-1435	<= 1000;DELETE * FROM table ORDER

などの既存の DIFT でも議論されていたことである．また，Raksha_n は，本稿で試したアタックを全て検出することができたものの，多くの典型的な変換で伝播させることができなかった．ネットワーク・アプリケーションにおいて，エンコードやデコードは，それぞれ，入力されたデータや出力されるデータに対して頻繁に行われる．また，入力データを全て小文字に変換するようなアプリケーションも存在する．したがって，このような変換に対応できないのは，本評価で全てのアタックが検出されたとはいえ，安全とは言いがたい．

一方，SWIFT では，本評価で試したアタックを誤検出なく全て検出することができた．また，多くの典型的な変換で伝播させることができた．したがって，SWIFT は，Raksha の両方式よりも伝播精度が高いと言える．しかし，SWIFT にも問題点がある．表 3 (g) (i) では部分的に伝播できない箇所が存在した．一般に，文字列操作で移動元と移動先の対応関係を正確に求めるのは難しく，表 3 (g) (i) のような，移動元と移動先のデータサイズが異なる場合や，文字列操作の最後の操作フェーズでは伝播の精度が下がる．文字列操作の識別精度を向上させることに関しては今後の課題としたい．

7. おわりに

本稿では，高次のインジェクションアタックを防ぐために，文字列操作の単位でテイント情報を伝播させる方式 (SWIFT) を提案した．SWIFT では，文字列操作を識別し，文字列から文字列へのメモリデータの移動にしたがって，テイント情報を伝播させる．評価は，x86 エミュレータ Bochs 上に SWIFT を実装し，既存のハードウェア・ベースの DIFT である Raksha と伝播精度について比較することによって行った．その結果，SWIFT は，Raksha よりも伝播の精度が高いことが示された．

謝　辞

本論文の研究は，一部，半導体理工学研究センター (STARC) 及び，JST CREST，科学研究費補助金（特定領域研究）No. 19024020 による．

文　献

[1] J. Allen. Perl version 5.8.8 documentation - perlsec. http://perldoc.perl.org/perlsec.pdf, 2006.

[2] J. L. Baer and T. F. Chen. Effective hardware-based data prefetching for high-performance processors. *IEEE Computer Society*, Vol. 44, No. 5, pp. 609–623.

[3] S. Christey and R. A. Martin. Vulnerability type distributions in cve. http://cve.mitre.org/docs/vuln-trends/index.html, 2007.

[4] M. Dalton, H. Kannan, and C. Kozyrakis. Raksha: A flexible information flow architecture for software security. In *34th International Symposium on Computer Architecture (ISCA)*, pp. 482–493, 2007.

[5] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, and D. Evans. Automatically hardening web applications using precise tainting. In *20th IFIP International Information Security Conference (SEC)*, pp. 295–307, 2005.

[6] T. Pietraszek and C. V. Berghe. Defending against injection attacks through context-sensitive string evaluation. In *8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pp. 124–145, 2005.

[7] W. Xu, S. Bhatkar, and R. Sekar. Practical dynamic taint analysis for countering input validation attacks on web applications. In *Technical Report SECLAB-05-04*, 2005.

[8] W. Xu, S. Bhatkar, and R. Sekar. Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In *15th USENIX Security Conference*, Vol. 15, pp. 121–136, 2006.