エミュレーションに基づく シェルコード検知手法の改善

Oct. 20, 2010

横浜国立大学大学院　環境情報学府

藤井 孝好　　吉岡 克成　　四方 順司　　松本 勉

マルウェア対策研究人材育成ワークショップ 2010 (MWS 2010)

はじめに（1/2）

- マルウェアによる被害の深刻化
  - ネットワークを介して感染を行う方法の一つに，**シェルコード**による攻撃が挙げられる
- シェルコード
  - リモートエクスプロイト攻撃として，ターゲットのコンピュータのプロセスにおける脆弱性を突いた後に実行される機械語コード
  - →攻撃を防ぐため，侵入検知システム(IDS)として，通信内容を監視して検知する研究が進められてきた
  - 一方，IDSの検知を逃れるために自己書き換えを行う**ポリモーフィックコード**が出現

はじめに（2/2）

- 近年，ポリモーフィックコードに対して，通信データを機械語とみなしてエミュレータ上で実行し，その動作の特徴により検知を行う**動的検知手法**が提案されている
- 既存の動的検知手法はポリモーフィックコードのみを対象としており，**非ポリモーフィックコード**は検知できない．また，非ポリモーフィックコードを対象とする既存の検知手法は，コードの難読化への対応が十分でない．

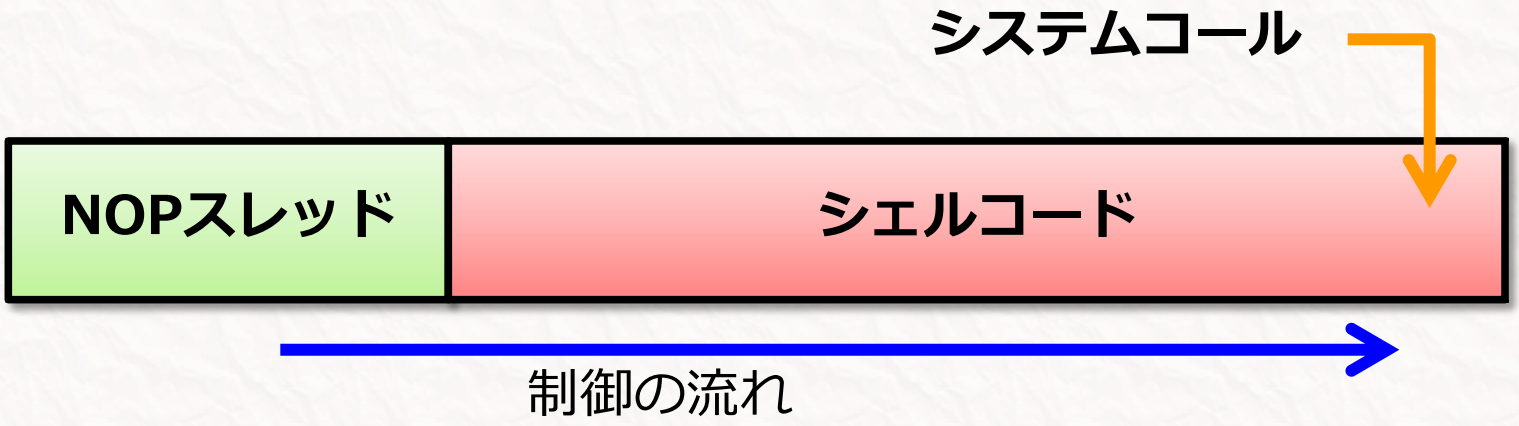
ポリモーフィックコード，および（難読化された）**非ポリモーフィックコード**の両方を，動的検知を用いて検知する手法を提案

発表の流れ

- はじめに
- シェルコードの種類と実行
  - 難読化のない通常のシェルコード
  - ポリモーフィックシェルコード
  - その他の難読化によるシェルコード
- 検知手法
  - 既存の検知手法
  - 提案手法
- 評価実験
- まとめと今後の課題

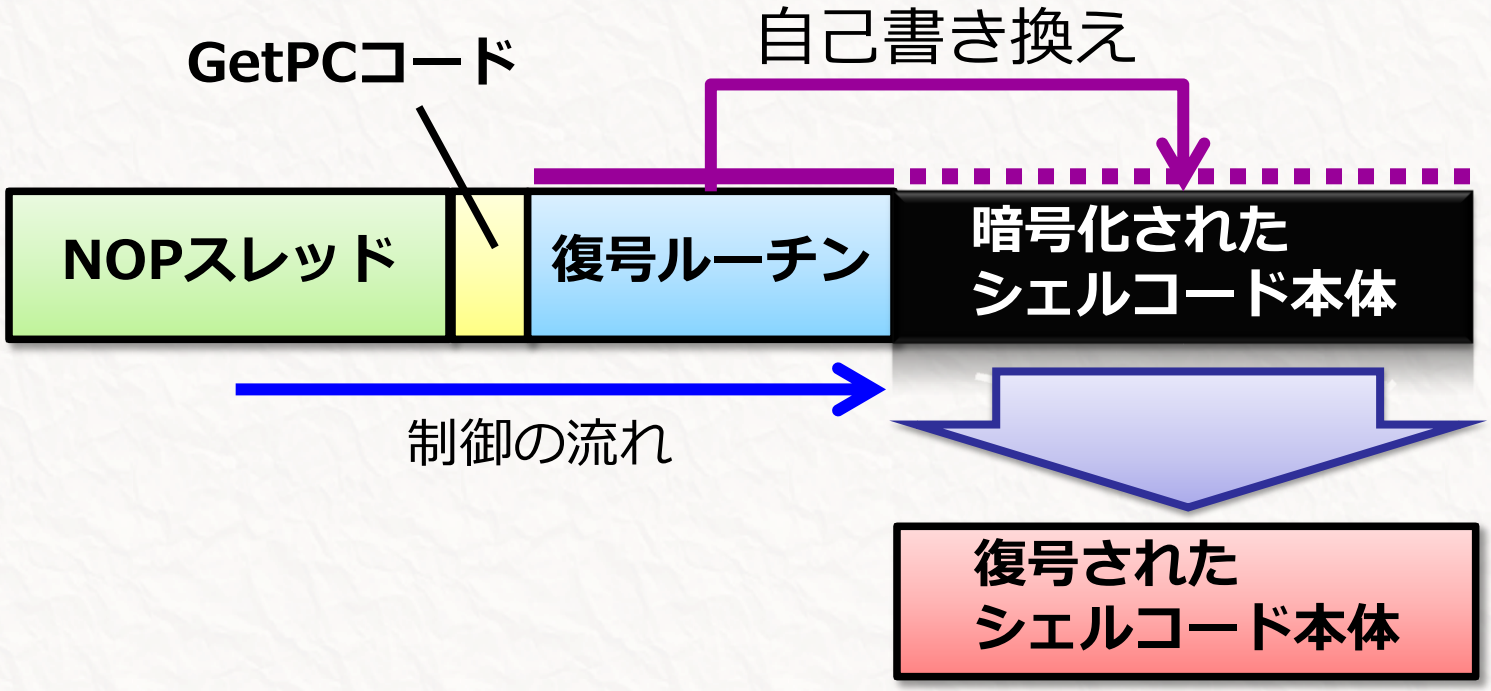
難読化されていない通常のシェルコード

- 脆弱性を突いた後，制御をコード本体に移す
  - ただし，直接コード先頭に制御を移すのは難しいため，まず，特定の命令列（**NOPスレッド**）上に制御を移す
  - NOPスレッドは，その中の任意の位置に制御が移っても，コード先頭に制御を渡すことができるような命令列である
- コード本体では，レジスタやメモリの値を設定した後，割り込み命令・ライブラリ呼び出し（**システムコール**）を実行する



ポリモーフィックコード

- 制御を奪取した後，実行されているコードの位置を**GetPCコード**により取得し，復号位置を求める
- 次に，暗号化されたコード本体を自己復号（書き換え）する
- その後，復号されたコードを実行する



その他の難読化手法

- メタモーフィズム
  - 命令の種類や実行順を変更する
    - コードの外見を多様化できる

```
xor eax,  eax
```

```
mov eax,  0x0
```

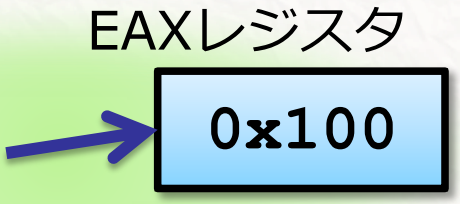
◆EAXレジスタを0に初期化

- 間接ジャンプ
  - ジャンプ先のアドレスを直接指定せずに，レジスタやメモリの値を用いる
    - ジャンプ先のアドレスが直接見えないようになる

```
jmp 0x100
```

◆直接ジャンプ

```
jmp eax
```



◆間接ジャンプ

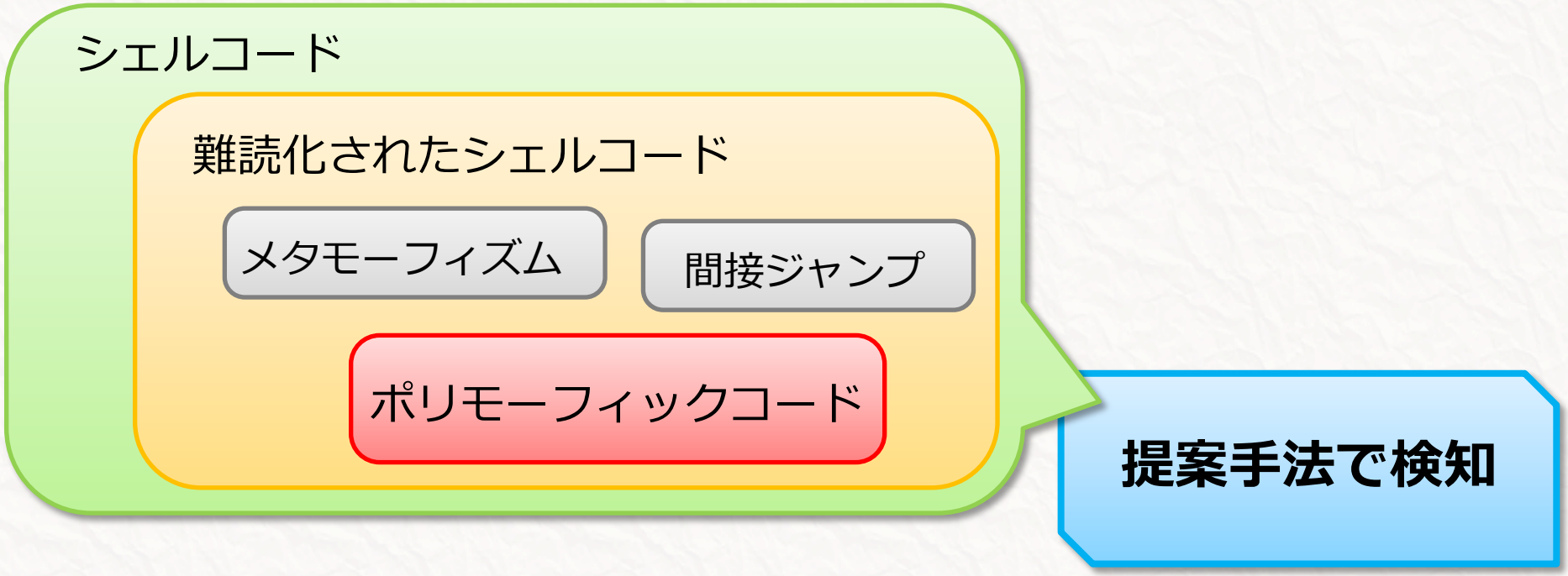
既存のシェルコード検知手法

- シグネチャ検知
  - 通信内容が，特定のバイトパターンに一致するかを調べる
  - → ポリモーフィックコードなど，コードの外見を多様化できる手法に対応が困難
- 静的検知（逆アセンブル・制御フロー解析）
  - 通信内容を機械語とみなして逆アセンブル・制御フローグラフを生成して解析する
  - → ポリモーフィックコードなどの自己書き換えや逆アセンブルを妨害する手法への対応が不十分
- 動的検知（エミュレーション）
  - 通信内容を機械語とみなしてCPUエミュレータで模擬実行を行い解析する
  - 既存研究では，ポリモーフィックコードに対して，自己書き換え処理の挙動に着目して検知を行う方法が提案

問題点と提案手法での対策

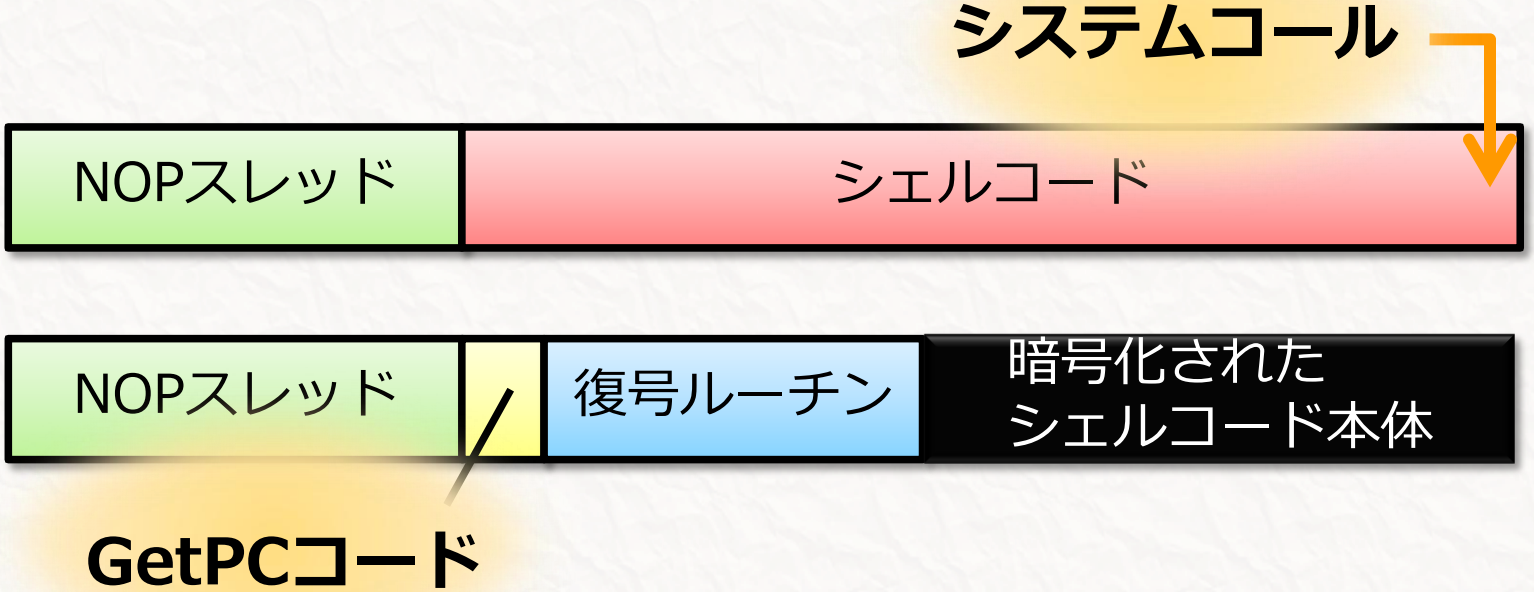
- 問題点
  - **難読化された非ポリモーフィックコード**の対応が不十分
    - シグネチャ・静的検知では，様々な難読化に対応しきれない

➔エミュレーションに基づく検知（動的検知）を考える



提案：ポリモーフィック・非ポリモーフィックコードを検出可能な動的検知手法（1/5）

- 着目点
  - ポリモーフィック・非ポリモーフィックコードそれぞれに，難読化の施されない命令（**平文命令**）が存在する
  - → 検査対象のデータを静的に走査することで平文命令を検出できる
  - さらにその周辺を動的に検査する



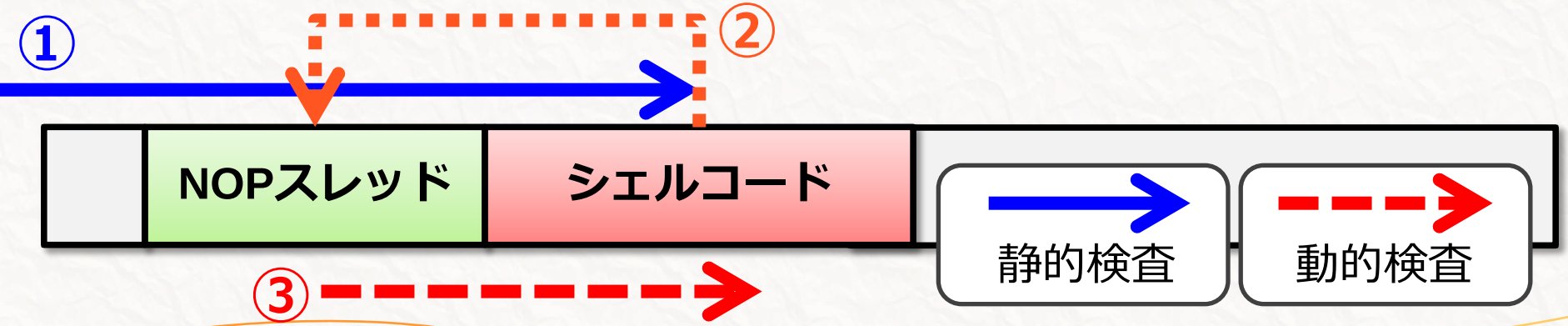
提案：ポリモーフィック・非ポリモーフィックコードを検出可能な動的検知手法（2/5）

- 検知の流れ
  - 各TCPストリーム・UDPパケットを検査対象として，以下の処理を行う

① • **静的検査**（平文命令の検出）

② • **エミュレーション開始位置の決定**

③ • **動的検査**（シェルコードにおける挙動の検出）



提案：ポリモーフィック・非ポリモーフィックコードを検出可能な動的検知手法（3/5）

①

- 検査対象を1バイトずつ順に検査し，**平文命令**の有無をチェック
- 平文命令が検出されたら，**②エミュレーション開始位置の決定**へ

- 検出する平文命令の種類
  - **GetPCコード**（ポリモーフィックコード）
    - `call,fnstenv`等
  - **システムコール**（非ポリモーフィックコード）
    - `int,call`等

提案：ポリモーフィック・非ポリモーフィックコードを検出可能な動的検知手法（4/5）

②

- 平文命令の検出位置から，オフセットを用いて開始位置を決定
- 特に，開始位置がNOPスレッド内にあるようにする
- 開始位置を決定したら，**③動的検査**へ

- 実験で用いたオフセットは，-512~512 byteの24種類の定数値

提案：ポリモーフィック・非ポリモーフィックコードを検出可能な動的検知手法（5/5）

③

- エミュレーションを行い，**共通規則**，および**ポリモーフィックコード検知規則**または**非ポリモーフィックコード検知規則**を満たしたらシェルコードとして検知する

共通規則

エミュレーションが検出された平文命令まで続くこと

ポリモーフィックコード検知規則

自己書き換えの実行が行われたこと

非ポリモーフィックコード検知規則

システムコールの実行時に，割り込み番号や呼び出しアドレスが適切であること

評価実験の流れ（1/2）

1.脆弱性検証ツールを用いた検知実験

- ✓実際にシェルコードを検知できるか
- 脆弱性検証ツール[1]を用いて，309個のシェルコードを生成
  - 非ポリモーフィック9個+ポリモーフィック300個
  - 攻撃の種類，暗号化の種類を変えて生成

2.擬似乱数ストリームに対する検知実験

- ✓シェルコードが含まれないデータに対して，どの程度フォールスポジティブ（誤検知）があるか
- 1ストリームのサイズを8KBとし，計2.3GBの擬似ストリーム（計294400個）を生成

[1] “The Metasploit Project,” http://www.metasploit.com/.

評価実験の流れ（2/2）

3. 実際のトラフィックに対する検知実験

- ✓ 実際の通信データではどのように検知されるか
- 以下の通信データを使用
  - 大学で稼働中のWebサーバの通信データ
  - CCC DATAset2010[2]（ハニーポットによる攻撃通信データ）

Webサーバへの通信データ	
ストリーム・パケットサイズ合計	69[MB]
TCP(HTTP)ストリーム数	2288[個]
CCC DATAset2010の攻撃通信データ	
pcapファイルサイズ合計	約3.4[GB]
ストリーム・パケットサイズ合計	162[MB]
TCPストリーム数	15910[個]
UDPパケット数	272814[個]

[2] 畑田充弘, 中津留勇, 秋山満昭, 三輪信, "マルウェア対策のための研究用データセット～MWS2010Datasets～," *MWS2010, 2010.*

実験結果（1/4）

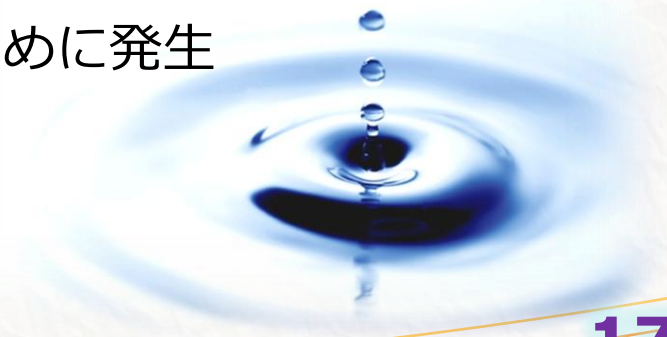
1. 脆弱性検証ツールを用いた検知実験

- すべてのシェルコード309個を検知

2. 擬似乱数ストリームに対する検知実験

- シェルコードとして誤検知された回数は合計9回
  （294400個のストリーム中）
  - ポリモーフィックコード　　1個
  - 非ポリモーフィックコード　　8個

- 偶然 機械語列として解釈されたために発生

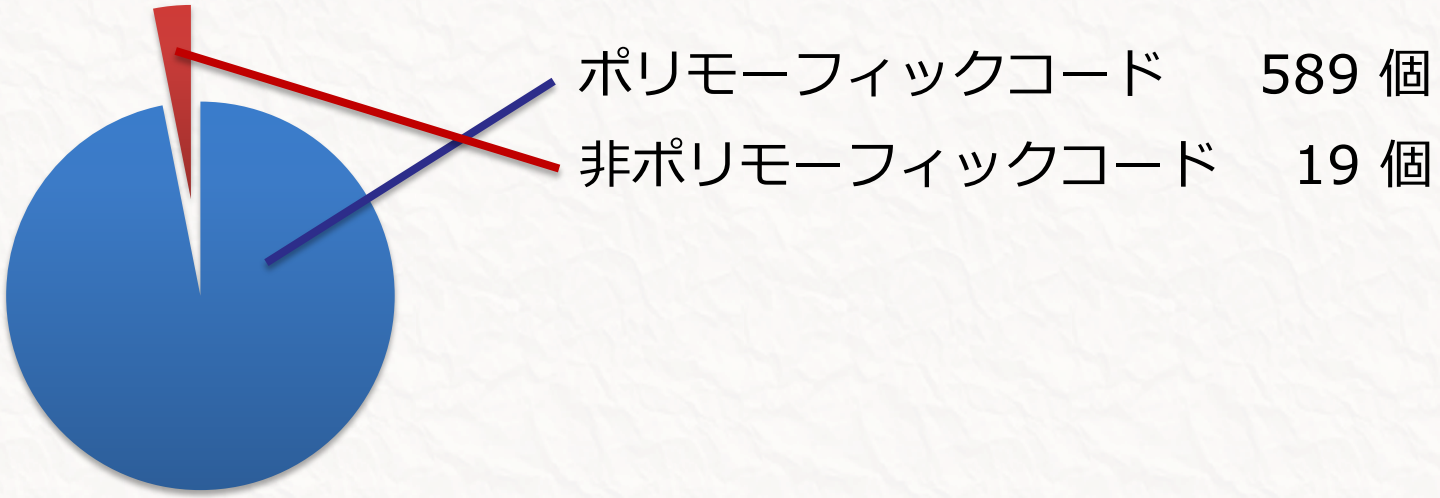


実験結果（2/4）

3. 実際のトラフィックに対する検知実験

- Webサーバのデータからは検知なし
- 攻撃通信データでは，計608個が検知された
  - いずれも手動により，実際のシェルコードであることを確認

- ポリモーフィックかどうかの内訳

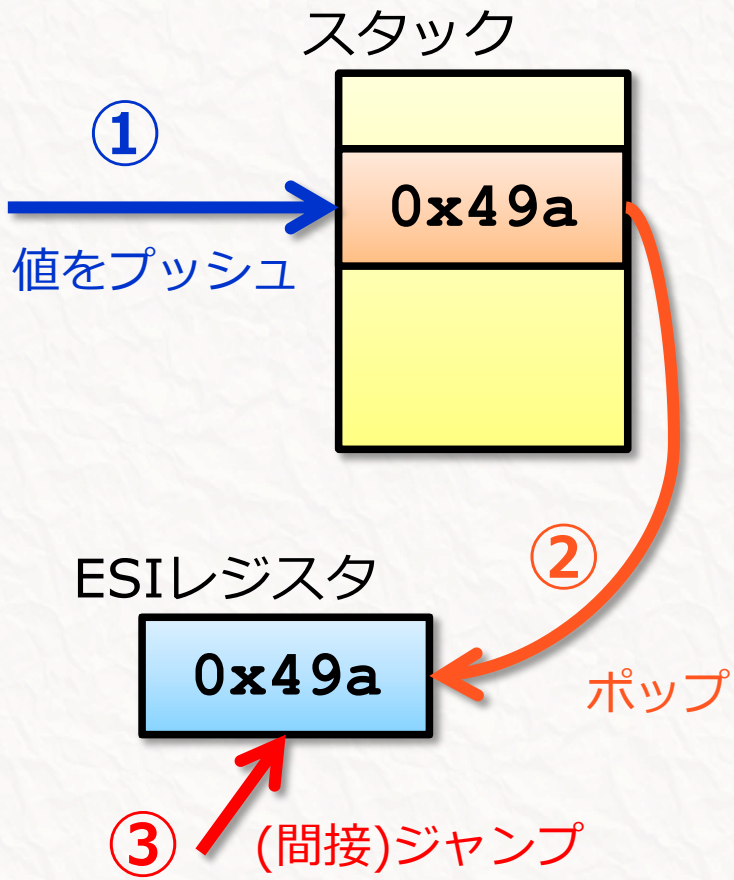


実験結果（3/4）

- 観察された非ポリモーフィックコードには，スタック内のデータを参照した間接ジャンプが使用

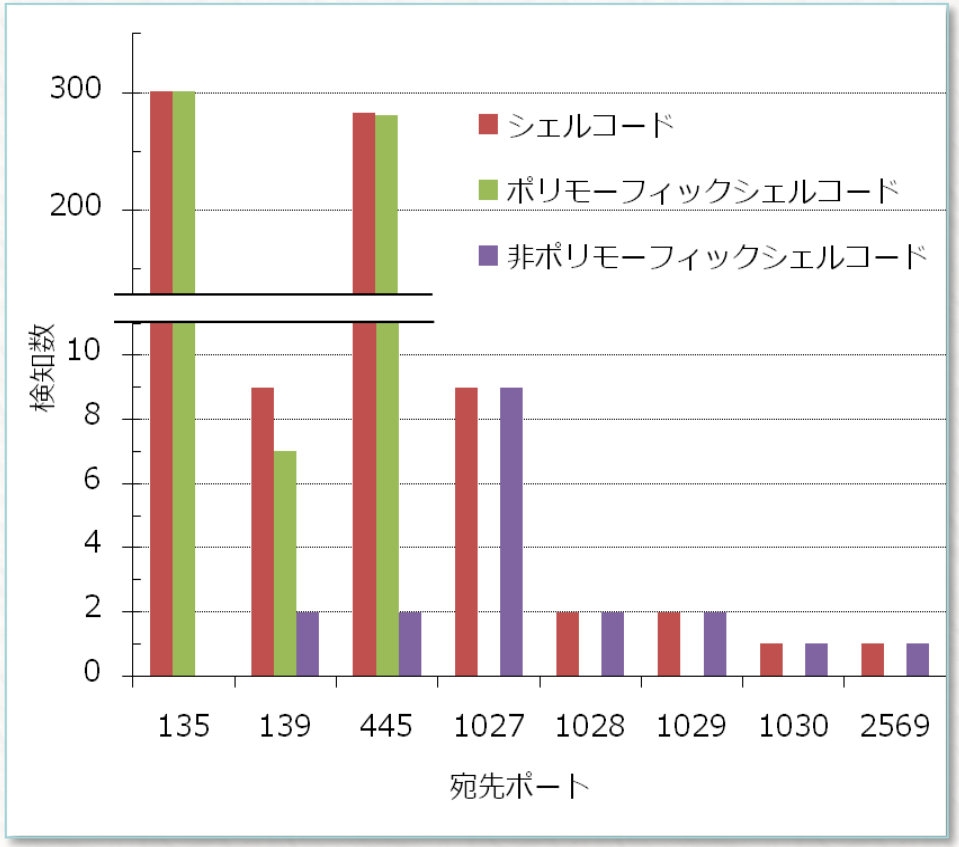
→ 本提案手法で検知が有効に働く

```
0495   call 0xfffffffe ..... ①
0493   jmp  0x6d
0500   pop  esi ..... ②
0501   push byte 0x30
              :
0522   push edi
0523   push dword 0xe88a49ea
0528   call esi ..... ③
049a   push ebx
049b   push ebp
              :
```



実験結果（4/4）

- 宛先ポート別に対するシェルコード数
  - UDPパケットからは検知されず
  - 135，445番ポートに攻撃が集中



- ポリモーフィック・非ポリモーフィックによって，宛先ポートの種類に違いが生じている

まとめと今後の課題

- まとめ
  - ポリモーフィック・非ポリモーフィックのどちらのシェルコードも検知可能な提案手法を示した
  - 実験によって，以下のことを示した
    - 脆弱性検知ツールにより生成したシェルコードを検知
    - フォールスポジティブ（誤検知）は9回/294400ストリーム
    - 攻撃通信データから，実際のシェルコードを検知
- 今後の課題
  - 検体を増やしたうえで，フォールスネガティブ（攻撃の見逃し率）の評価
  - ほかの検知手法に対して，検知精度および動作速度の比較および改善

ご清聴ありがとうございました