SECURITY ARK

# ソフトウェアを美味しく解析する方法

Will@ Security Ark
http://security.symphonic-net.com/

# 自己紹介

- 世を忍ぶ仮の姿はとある大学の学生
- しょぼいソフトウェアの開発や解析などを少々
- 金穴につき、ただ今大学の生協にてバイト中
- 全く関係ないが二日前まで長野でスノボーしてた
- さらに関係ないが金もないのに今年は何故か三回もスノボーに行った。

# 解析とは

リバースエンジニアリングとは、機械を分解
したり、製品の動作を観察したり
ソフトウェアの動作を解析するなどして
製品の構造を分析し、そこから製造方法や
動作原理、設計図、ソースコードなどを
調査する事である。

http://ja.wikipedia.org/wiki/リバースエンジニアリング

取　り　敢　え　ず

今回はソフトウェアのReverse engineeringということで

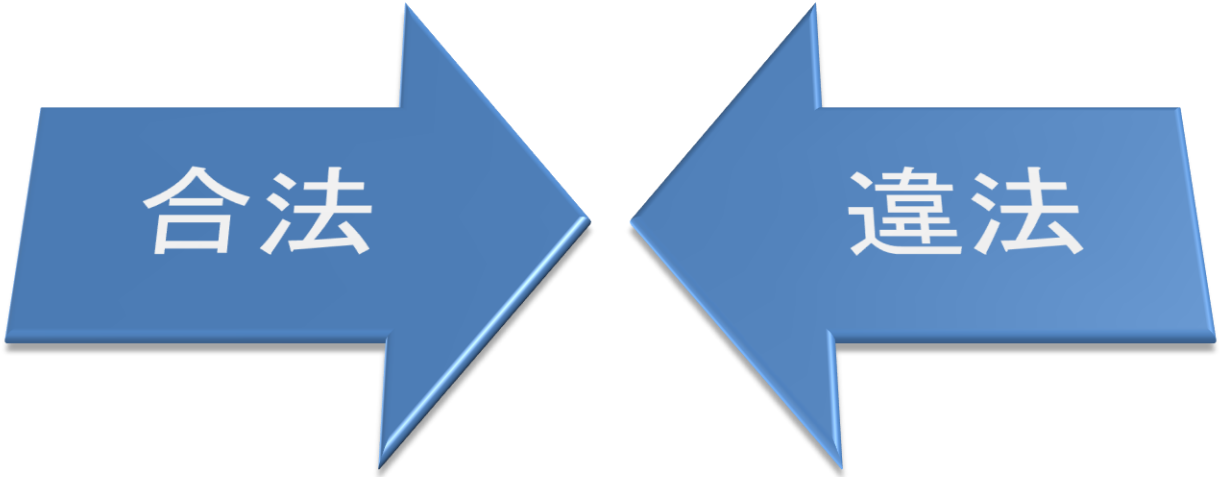
# 解析の目的

- 脆弱性の調査
- ウイルスの調査
- 互換性の確保
- 独自機能の追加

…etc

# 解析の必要性

使い方によっては善にも悪にもなる技術だが
Securityの分野では重要な技術！



# 解析を行うには

♪アセンブリ言語に関する知識

♪Win32APIに関する知識

♪PE Formatに関する知識

**挫けない心** ←これ超重要

# アセンブリ言語について

アセンブリ言語ではデータを保存する場所として以下の四つがあります。

✓レジスタ
✓スタック
✓ヒープ領域
✓データセクション領域

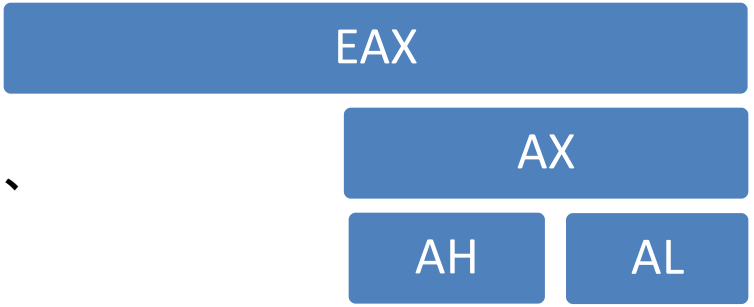
☆レジスタについて☆

　レジスタはCPUの内部に存在するメモリで、高速なので計算を行う際に一時的に使用されます。各レジスタには大まかな役割が割り当てられていますが無視しても大丈夫です。
　しかし、命令によってはある特定のレジスタの値に依存した処理が行われる事があるので、出来る限りその方針にしたがった使い方をした方がいいでしょう。

# レジスター覧

| レジスタ名 | 使用目的 |
|---|---|
| EAX | 何にでも使っても良い、関数の戻り値はこれに入る。 |
| EBX | アドレス演算用。ポインタみたいな感じ。 |
| EDX | 何にでも使って良い。 |
| ECX | ECXは特定条件化ではカウンタとなるが<br>基本的にはEAX,EDX同様に何に使ってもよい |
| ESI | extended source indexの略で転送元アドレスの指定に使われる |
| EDI | extended destination indexの略で転送先アドレスの指定に使われる |
| ESP | スタックの現在アドレスを保持している<br>（基本的にこれは別用途で使用してはいけない） |
| EBP | ローカル変数や引数の参照などにつかう |

✓レジスタはすべて32bitで、最初のEをとった
ax,bxなどで下位16bitにアクセスできる。
✓さらに、EAX、EBX、EDX,ECXに至ってはahおよび、
alでさらにその半分の8bitにアクセスできる。



# 命令一覧

| 命令 | 説明 |
| --- | --- |
| mov | add dest,src　dest = dest + srcという処理が行われます。 |
| add | sub dest,src　dest = dest - srcという処理が行われます。 |
| push | push src　srcがスタックに積まれます。 |
| pop | pop dest　destにスタックから取り出した値が格納されます。 |
| imul | Iml eax,edx　eaxにeax*edxの結果が格納されます。 |
| test | test src1, src2　src1とsrc2の論理積(AND)をとって、<br>その結果をステータスレジスタに書き込みます。<br>ステータスレジスタは変わりますが、<br>src1とsrc2のあたいは変わらずに保持されます。 |
| j* | j* src　*の中に入る文字で処理が変わります。<br>eの時はequalの略で比較命令の結果が等しればジャンプします。<br>neはnot equalの略で比較命令の結果が等しくなければジャンプ。 |
| lea | lea dest,src　実行アドレスの計算。他に簡単な計算などに使われる。 |

# 関数の呼び出し規約

C言語では主にcdeclが用いられている。

1. cdeclでは関数への引数は右から左の順でスタックに積まれる。
2. 関数の戻り値は EAX（80x86のレジスタの一つ）に格納される。
3. スタックポインタの処理は呼び出し側で行う。

1.の例　　foo(int a1,int a2,int a3)という関数を呼び出す場合は

```
---------------------------------------------
push a3
push a2
push a1
call foo
---------------------------------------------
```

2.の例　　1を戻り値として返す場合

```
---------------------------------------------
mov eax,1
ret
---------------------------------------------
```

# これな～んだ？

56 8B 74 24 08 85 F6 75 07 B8 01
00 00 00 5E C3 8D 46 FF 50 E8 E7
FF FF FF 0F AF C6 83 C4 04 5E C3

これで分かる人はマジで凄い！

とりあえず逆アセンブルしてみよう♪

# 逆アセンブルしてみよう♪

```
00941000        PUSH ESI
00941001        MOV ESI,DWORD PTR SS:[ESP+8]
00941005        TEST ESI,ESI
00941007        JNZ SHORT test.00941010
00941009        MOV EAX,1
0094100E        POP ESI
0094100F        RETN
00941010        LEA EAX,DWORD PTR DS:[ESI-1]
00941013        PUSH EAX
00941014        CALL test.00941000
00941019        IMUL EAX,ESI
0094101C        ADD ESP,4
0094101F        POP ESI
00941020        RETN
```

目標はCのソースに変換！

# ブロックで分ける

```
00941000    PUSH ESI
00941001    MOV ESI,DWORD PTR SS:[ESP+8]
00941005    TEST ESI,ESI
00941007    JNZ SHORT test.00941010
00941009    MOV EAX,1
0094100E    POP ESI
0094100F    RETN
00941010    LEA EAX,DWORD PTR DS:[ESI-1]
00941013    PUSH EAX
00941014    CALL test.00941000
00941019    IMUL EAX,ESI
0094101C    ADD ESP,4
0094101F    POP ESI
00941020    RETN
```

第一ブロック

第二ブロック

第三ブロック

# 第一ブロックの理解

00941000　　PUSH ESI
↑下で使うので今現在の値をスタックに待避

00941001　　MOV ESI,DWORD PTR SS:[ESP+8]
↑ESIに第一引数の値を格納

00941005　　TEST ESI,ESI
↑ESIの値を評価

00941007　　JNZ SHORT test.00941010
↑ESIの値が0ではなかったらジャンプ！

# 第二ブロックの理解

00941009　　MOV EAX,1
个EAXに1を代入

0094100E　　POP ESI
个待避していた値をESIに戻す

0094100F　　RETN
个りた～ん

# 第三ブロックの理解

00941010　　LEA EAX,DWORD PTR DS:[ESI-1]

↑EAXにESI-1の結果を格納

00941013　　PUSH EAX

↑EAXをスタックに放り込む

00941014　　CALL test.00941000

↑0x941000番地の関数を呼び出す

00941019　　IMUL EAX,ESI

↑EAX=EAX*ESIの計算を行う

0094101C　　ADD ESP,4

↑スタックの調整

0094101F　　POP ESI

↑待避していた値をESIに戻す

00941020　　RETN

# ちょっとずつCにしていくよ

```
00941000        PUSH ESI   ←ESIを使うのでスタックに待避
00941001        MOV ESI,DWORD PTR SS:[ESP+8]
00941005        TEST ESI,ESI
00941007        JNZ SHORT test.00941010←第三ブロックへ
```

```
hoge(int arg)
{
        if(arg == 0){
                [第二ブロック]
        }
        [第三ブロック]
}
```

# 第二ブロック

| | | |
|---|---|---|
| 00941009 | MOV EAX,1 | 戻り値に1を指定 |
| 0094100E | POP ESI | ESIに値を戻す |
| 0094100F | RETN | |

Return 1;

これだけですｗ

# 第三ブロック

```
00941010    LEA EAX,DWORD PTR DS:[ESI-1]    ←    ESIはarg
00941013    PUSH EAX
00941014    CALL test.00941000              ← 0x941000は関数の先頭
00941019    IMUL EAX,ESI
0094101C    ADD ESP,4
0094101F    POP ESI
00941020    RETN
```

return hoge(arg-1)*arg

これだけですｗ

# まとめちゃいます

```
hoge(int arg)
{
        if(arg == 0){
                return 1;
        }
                return hoge(arg-1)*arg;
}
```

Nの階乗を計算する
再帰関数でした

# まとめ

解析（リバースエンジニアリング）をするためには☆

✓アセンブリ言語の知識
✓Win32APIに関する知識
✓PE Formatに関する知識
✓根性
✓カン＆慣れ

そして…

**0と1を愛する心♪**

# 終わり

＼(^o^)／オワタ

ご静聴
ありがとうございました♪