マルウェア対策研究人材育成ワークショップ2008

ステルスデバッガを利用したマルウェア解析手法の提案

NTT情報流通プラットフォーム研究所

◎川古谷裕平　岩村誠　伊藤光恭

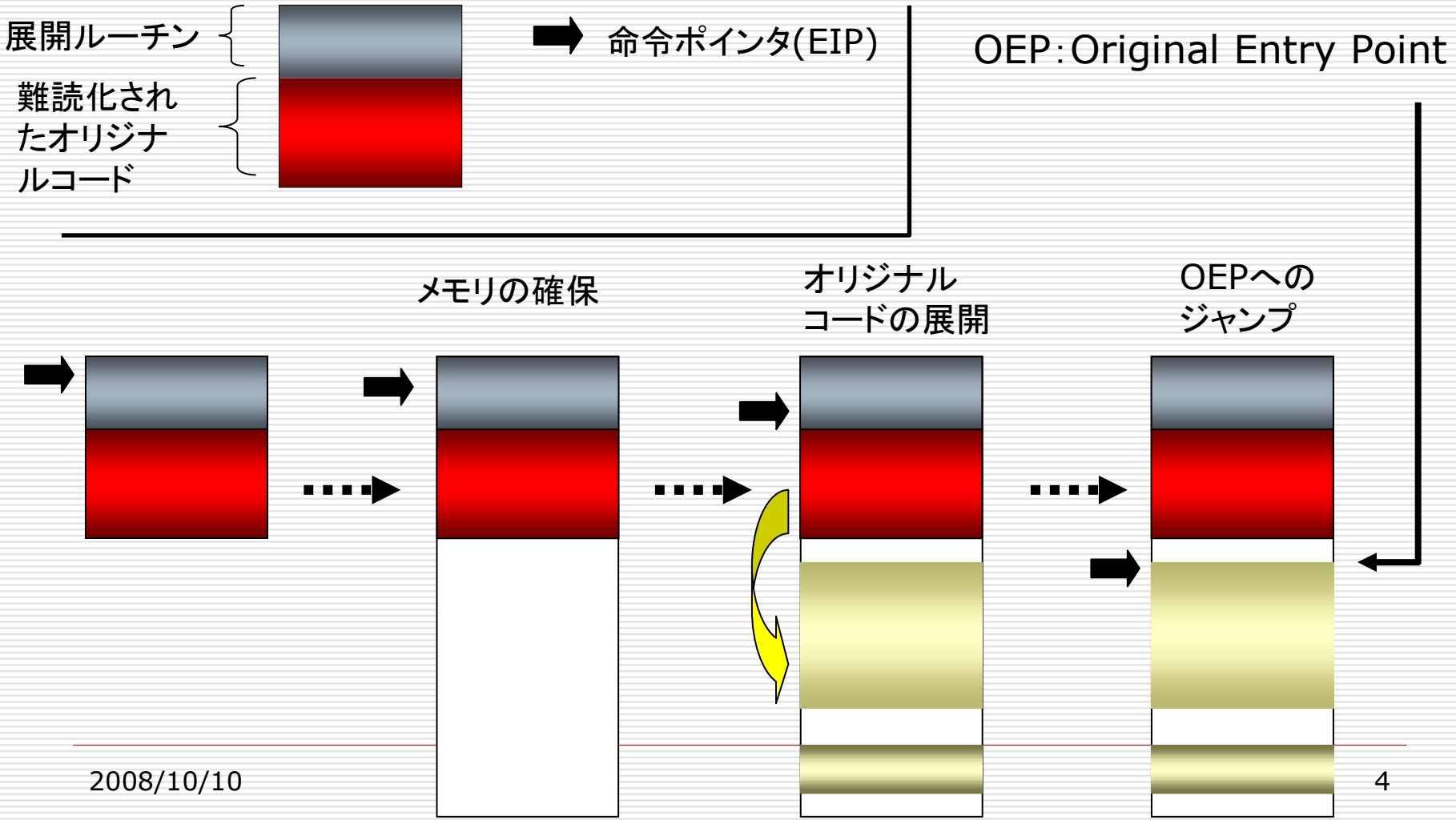
目次

- 背景
- ステルスデバッガの提案
- CCC Dataset 2008検体による評価
- 考察
- まとめ

背景

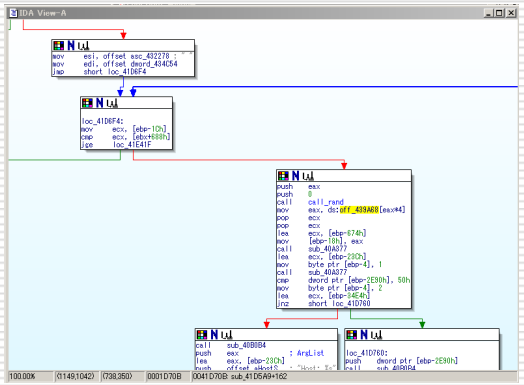
- マルウェアの高度化、高機能化
  - 柔軟な機能追加
  - 自身の隠蔽化
  - 耐解析機能（Anti Debug機能）
  - …
- 難読化ツールの利用
  - パッカー/プロテクタ/難読化ツール

難読化されたマルウェアの基本的な動作

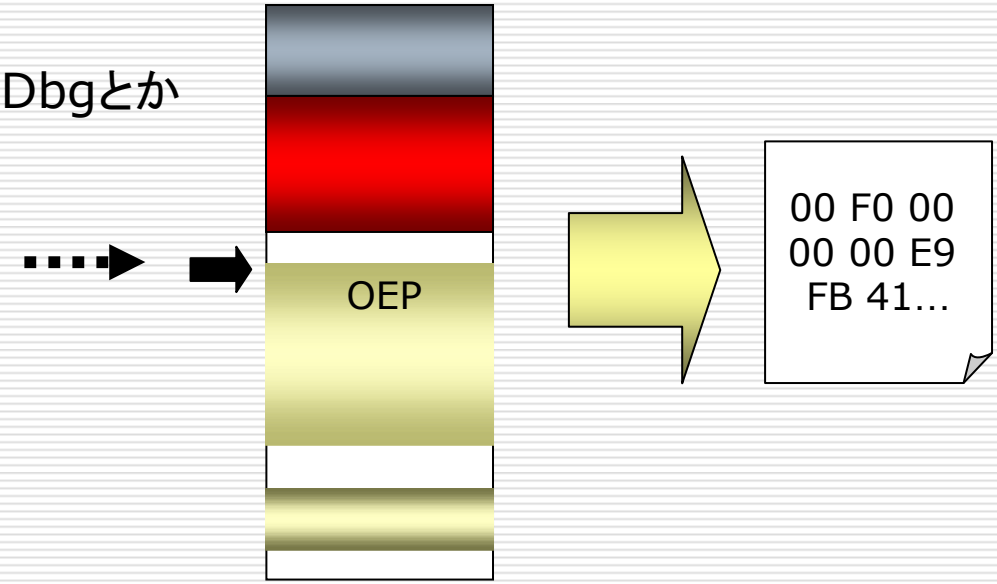


マルウェアの基本的な解析手法

- デバッガで実行してOEPでとめる
- メモリ上の値をダンプしてファイルに保存
- ダンプされたバイナリをIDA proなどで逆アセンブル
- がんばって読む



OllyDbgとか

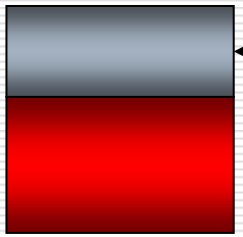


00 F0 00
00 00 E9
FB 41…

C&C Server
感染手法
パスワード
etc…

アンパッキング

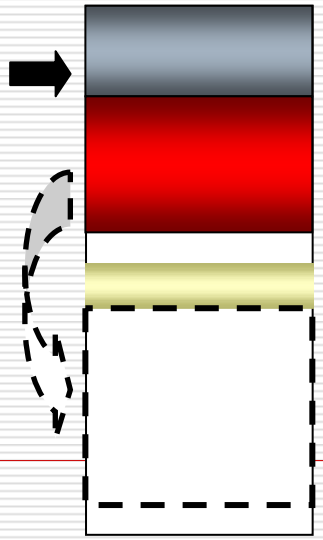
Anti Debug機能



展開ルーチン+Anti Debug機能

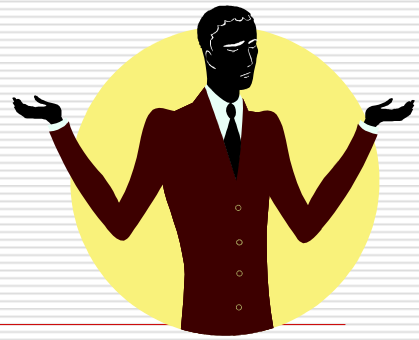
- デバッガ検知
- ブレークポイント検知
- シングルステップ検知
- 仮想マシン検知
- 時間制限
- …

オリジナル
コードの展開



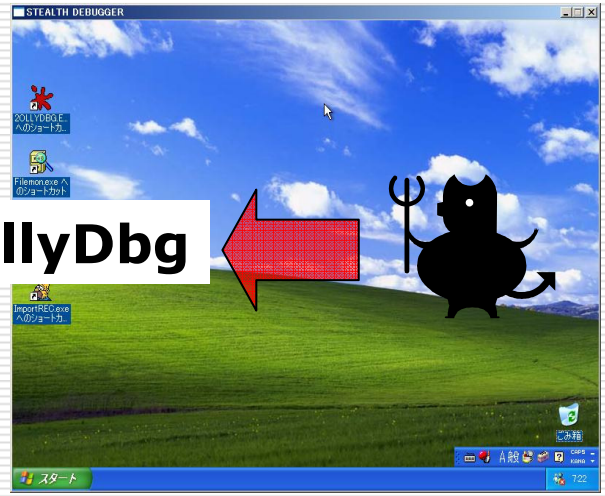
OllyDbg

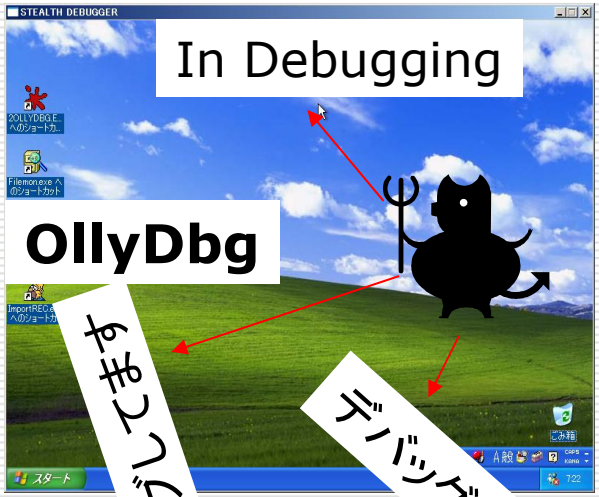
終了
異常動作



従来デバッガの問題点

1. 同環境上にデバッガとデバッギが存在するため、デバッガが攻撃を受けやすい(見つかりやすい)
2. CPUやOSのデバッグ支援機構はステルス性を考慮して設計されていない
   - いろんな箇所にデバッグの痕跡を残してしまう

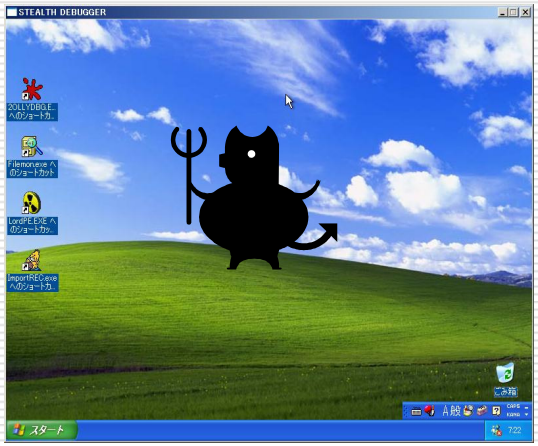




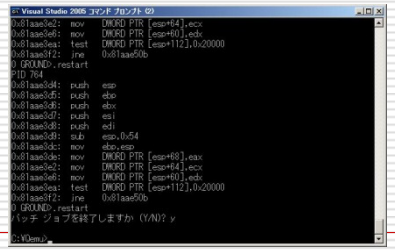
ステルスデバッガの提案

- 仮想マシンを利用してマルウェアの実行環境から隔離された環境からデバッグを行える機構の提供(ring -1)
- 従来のCPUやOSのデバッグ支援機構に頼らないデバッグ機構の提供

ゲストOS



Controller



制御

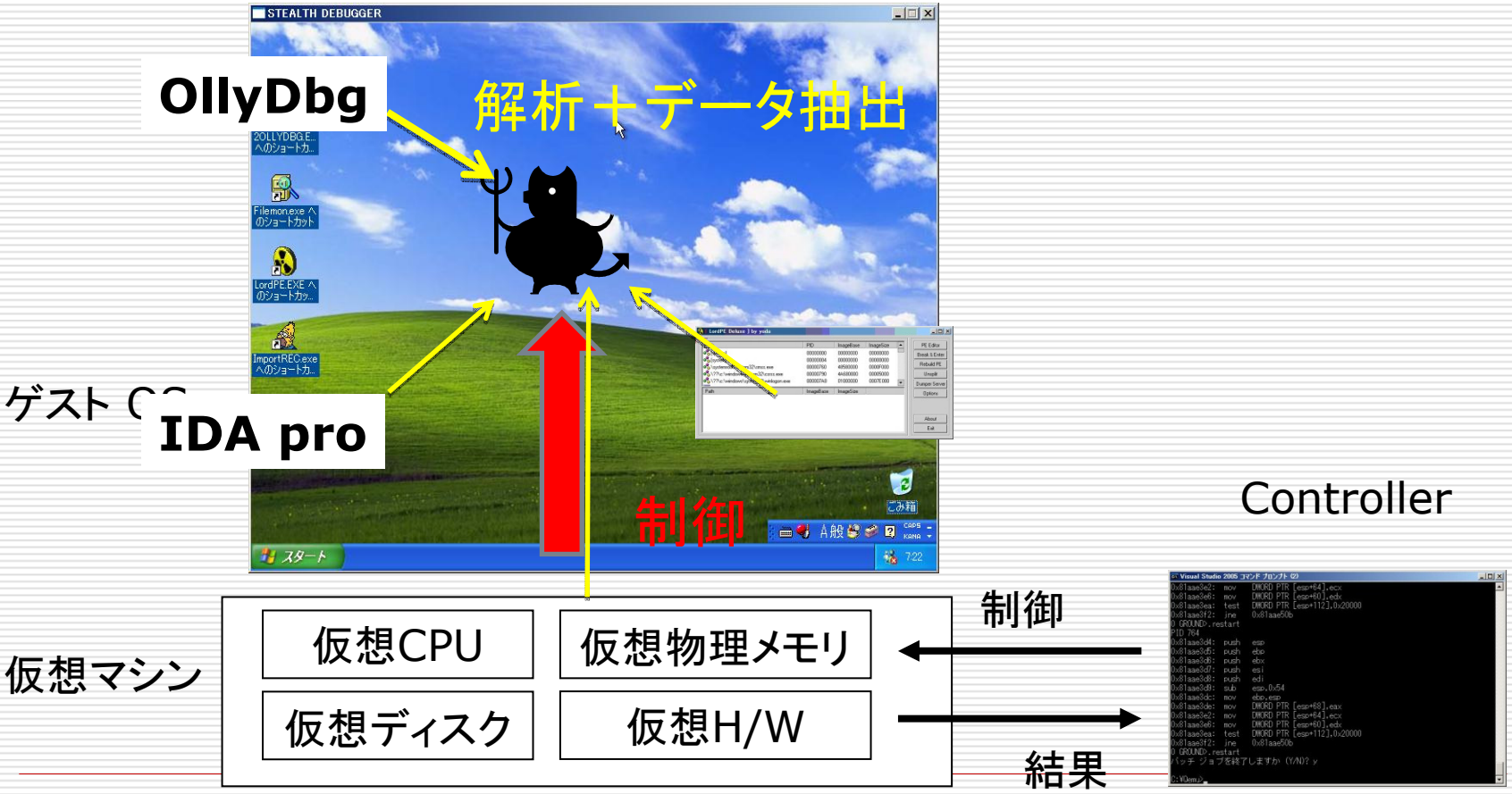
結果

仮想マシン
Based on
Qemu

仮想CPU | 仮想物理メモリ

仮想ディスク | 仮想H/W

実際の解析環境


OllyDbg
IDA pro
解析＋データ抽出
制御
ゲストOS
仮想マシン
仮想CPU
仮想物理メモリ
仮想ディスク
仮想H/W
制御
結果
Controller

ステルスデバッガの特徴

- 仮想H/W制御によるデバッガ機能
  - 従来のデバッガとは異なる仕組みでのデバッグ機能の提供
  - 通常のデバッガが持つデバッグ機能とほぼ同等の機能を実現
- タイムコントロール
  - ゲストOS内の時間をコントロールしながらデバッグを行う。デバッグ中はゲストOS内の時間は完全に停止する。
- 命令内容に基づいたブレークポイント
  - 特定の命令でブレークさせることが可能
  - 特定の命令列でブレークさせることが可能
- リソースアクセスモニタ
  - ゲストOSの外側から特権モードに移行する命令の実行を捉えることでゲストOSの内部で行われているファイルシステムへのアクセスを監視

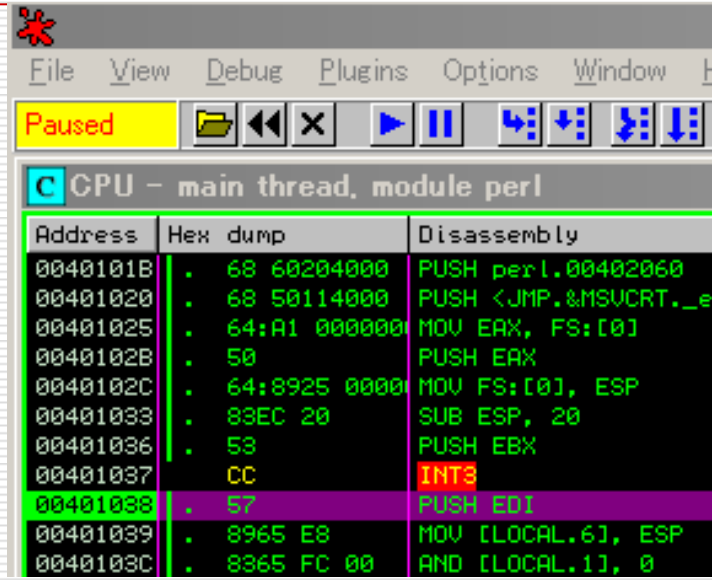
ステルスデバッガコマンド一覧

分類	コマンド	概要
ブレークポイント	.setbp [address]	ブレークポイントのセット
	.delbp [address]	ブレークポイントの削除
	.showbp	ブレークポイントの一覧
トレース	.trace start [pid] [filename]	実行トレース開始
	.trace end [pid] [filename]	実行トレース終了
ステップ実行	.ss	シングルステップ実行
	.tb	Translation Blockステップ実行
メモリ操作	.mem [address] [bytes]	addressからbytes分メモリ表示
	.write [address] [value]	addressに値(value)をセット
ファイルモニタ	.mon start	ファイルアクセス監視を開始
	.mon end	ファイルアクセス監視を終了
レジスタ操作	.reg	レジスタ値一覧
	.setreg [reg] [value]	レジスタ(reg)に値(value)をセット

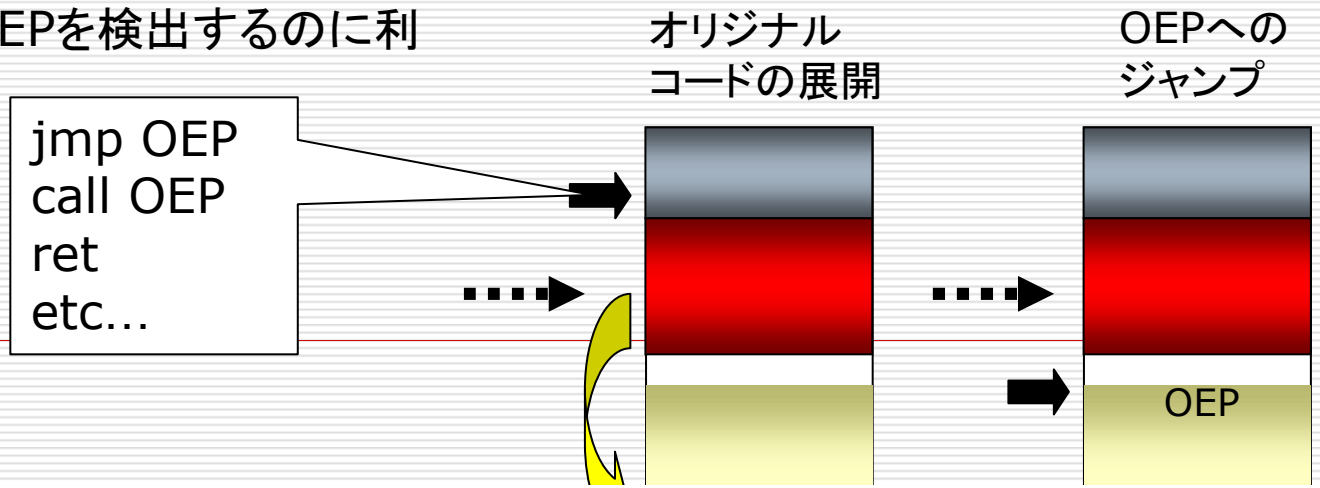
※その他…ゲストOS内でCtrl+Shiftでブレーク

実行命令に基くブレークポイント

- 従来のブレークポイントはアドレスに基き設定される
- 実行命令に基くブレークポイント
  - 実行命令に基きデバッギをブレークさせるブレークポイント
  - 例1 ret命令が実行されたブレーク
  - 例2 pop,pop,retの命令列が実行されたらブレーク



OEPへジャンプする部分のジャンプコードをブレークさせてOEPを検出するのに利用する



リソースアクセスモニタ

- ゲストOSの外側からゲストOS内でのリソース(ファイル)アクセスを監視する
- ユーザモードからカーネルモードへ移行する命令の実行を契機にゲストOSのメモリやファイルシステムの情報を調べることで、リソースへのアクセスを監視する

例) hoge.txtをCreateFileした場合

```
Kernel32.dll      CreateFile(C:¥hoge.txt,…);
Ntdll.dll         NtCreateFile(C:¥hoge.txt,…);
…
sysenter
Ntoskrnl.exe      KiSystemService();
…
Ntoskrnl.exe      NtCreateFile(C:¥hoge.txt, …);
```

仮想CPUでここを捉えて、引数に渡されている値をログ出力する

実装環境

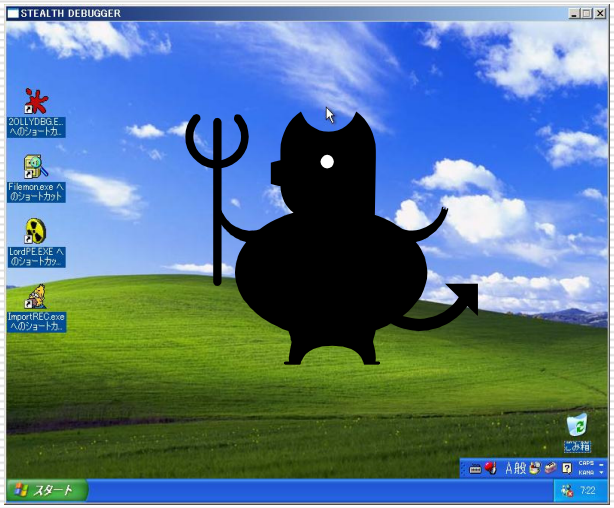
CPU	Intel Core2 Quad 2.66GHz
メモリ	SDRAM 2GB
Host OS	Windows XP SP3 Pro
Guest OS	Windows XP SP0 Pro
VMM	Qemu 0.9.1 without KQEMU

CCC Dataset 2008検体による評価

前調査

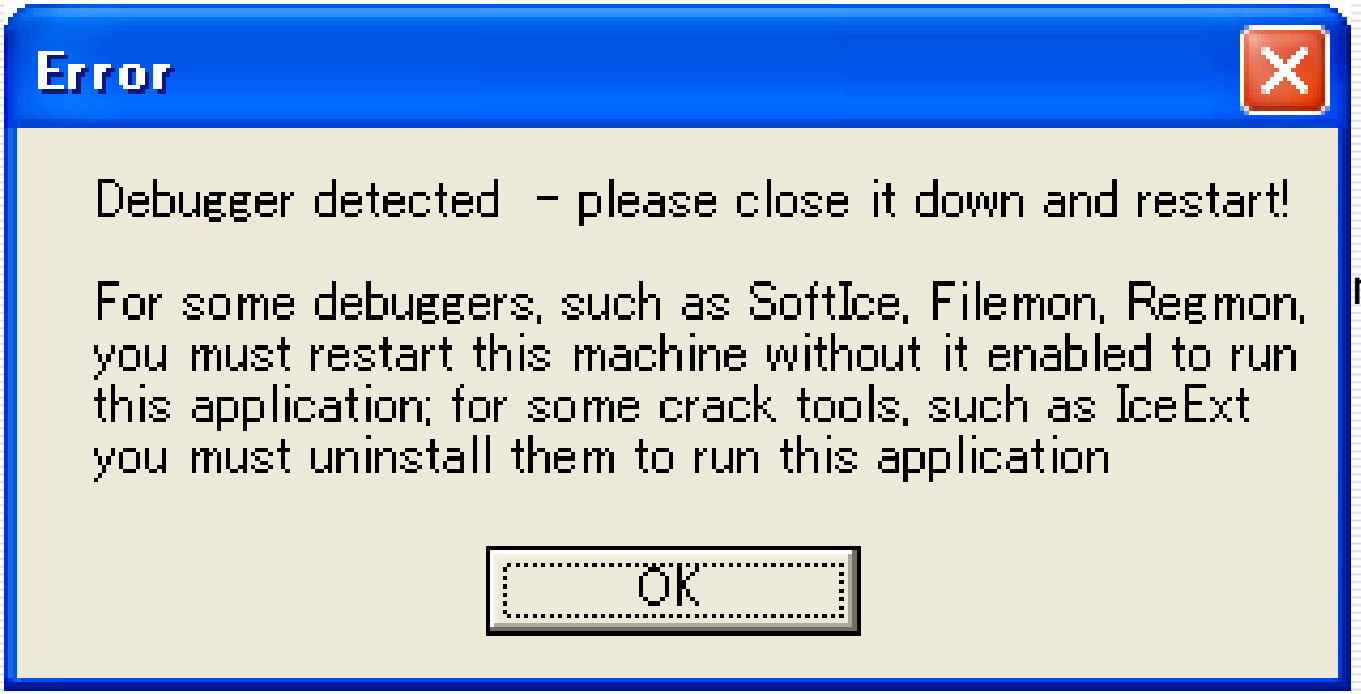
OllyDbg

LordPE



RegMon　FileMon

前調査



FileMon

nitor

IDA pro

高機能パッカーA

ステルスデバッガを利用したCCC Dataset 2008検体の解析

1. 動的解析
    1. リソースアクセスモニタ
    2. トレース実行
2. アンパッキング
    1. 実行命令に基くブレークポイントでOEP候補を検出
    2. デバッギをOEPで停止させる
    3. デバッギのメモリをダンプ
3. 静的解析 by IDA pro
    1. がんばる

リソースアクセスモニタによる動的解析
#>.mon start

プロセスの起動順序(PID) 2020→2028→112→120

[pid=2020] C:¥DOCUME~1¥UserName¥LOCALS~1¥Temp¥~temp02023628362.tmp ← ？

[pid=2028] C:¥DOCUME~1¥UserName¥LOCALS~1¥Temp¥~temp02023628362.tmp
[pid=2028] C:¥share¥ccc_malware.exe
[pid=2028] ¥SICE
[pid=2028] ¥NTICE
[pid=2028] ¥SIWDEBUG
[pid=2028] ¥SIWVID
[pid=2028] ¥FILEMON
…
[pid=2028] ¥NTICE
[pid=2028] C:¥share¥ccc_malware.exe
[pid=2028] C:¥WINDOWS¥System32¥NVCOM.EXE

ドライバファイルにアクセス

本体？

[pid=112] C:¥DOCUME~1¥UserName¥LOCALS~1¥Temp¥~temp02023628362.tmp

[pid=120] C:¥DOCUME~1¥UserName¥LOCALS~1¥Temp¥~temp02023628362.tmp
[pid=120] C:¥WINDOWS¥System32¥NVCOM.EXE
[pid=120] ¥SICE
[pid=120] ¥NTICE
[pid=120] ¥SIWDEBUG
[pid=120] ¥SIWVID
…
[pid=120] ¥NTICE
[pid=120] C:¥WINDOWS¥System32¥drivers¥etc¥hosts
…

hostsファイルにアクセス

FileMonなどでは取得できなかったファイルアクセスモニタが可能

全実行命令トレース
>.trace start [pid]

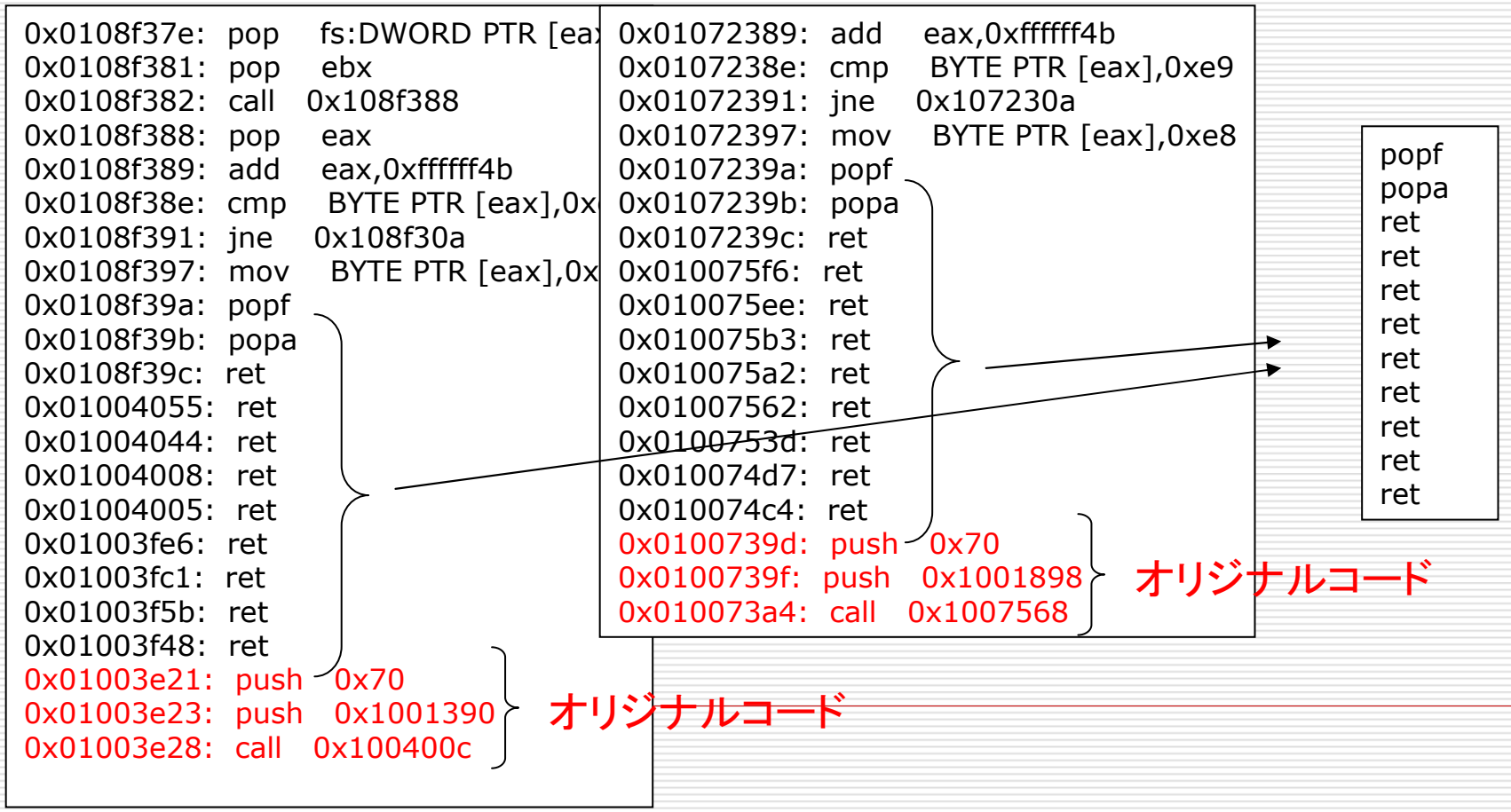
```
0x00405c29:  inc    edx
0x00405c2a:  test   al,al
0x00405c2c:  jne    0x405bff
0x00406d5f:  cmp    eax,0x1
0x00406d62:  jne    0x406e67
0x00406d68:  mov    DWORD PTR [ebp-32],esi
0x00406d6b:  mov    DWORD PTR [ebp-36],esi
0x00406d6e:  cmp    DWORD PTR [ebp+24],esi
0x00406d71:  jne    0x406d7b
0x00406d7b:  push   esi
0x00406d7c:  push   esi
0x00406d7d:  push   DWORD PTR [ebp+16]
0x00406d80:  push   DWORD PTR [ebp+12]
0x00406d83:  xor    eax,eax
0x00406d85:  cmp    DWORD PTR [ebp+32],esi
0x00406d88:  setne  al
0x00406d8b:  lea    eax,[eax*8+1]
0x00406d92:  push   eax
0x00406d93:  push   DWORD PTR [ebp+24]
0x00406d96:  call   ds:0x408030
0x00406d9c:  mov    edi,eax
0x00406d9e:  mov    DWORD PTR [ebp-40],edi
0x00406da1:  test   edi,edi
```

数々のAnti Debug技術
- rdtscによる時間計測
- シングルステップチェック
- 例外による状態遷移
- API呼び出しのモニタチェック
- VMware検知

ステルスデバッガではこれらのAnti Debug技術を回避して検体をデバッグすることが可能

高機能パッカーAのOEPジャンプの特徴

二つのオリジナルバイナリを高機能パッカーAでパッキングして、トレースを取得



OEPの検出

- 「popf,popa,2回以上のret」でブレークさせるシグネチャを作成して検体を実行

```
0x0049f389:  add    eax,0xffffff4b
0x0049f38e:  cmp    BYTE PTR [eax],0xe9
0x0049f391:  jne    0x49f30a
0x0049f397:  mov    BYTE PTR [eax],0xe8
0x0049f39a:  popf
0x0049f39b:  popa
0x0049f39c:  ret
0x0042bdcb:  ret
0x0042bdab:  ret
0x0042bd8b:  ret
0x0042bd60:  ret
0x0042bd35:  ret
0x0042bd20:  ret
0x0042bc64:  ret
0x0042bc30:  ret
…
```

```
…
0x00427eab:  ret
0x00427eaa:  ret
0x00427ea7:  ret
0x00427e6b:  ret
0x00427e10:  ret
0x00427dfd:  ret
0x00427d20:  call   0x4a5f52
0x004a5f52:  call   0x49f3be
0x0049f3be:  call   0x49f3c4
0x004a5f57:  mov    eax,DWORD PTR [eax+200]
```

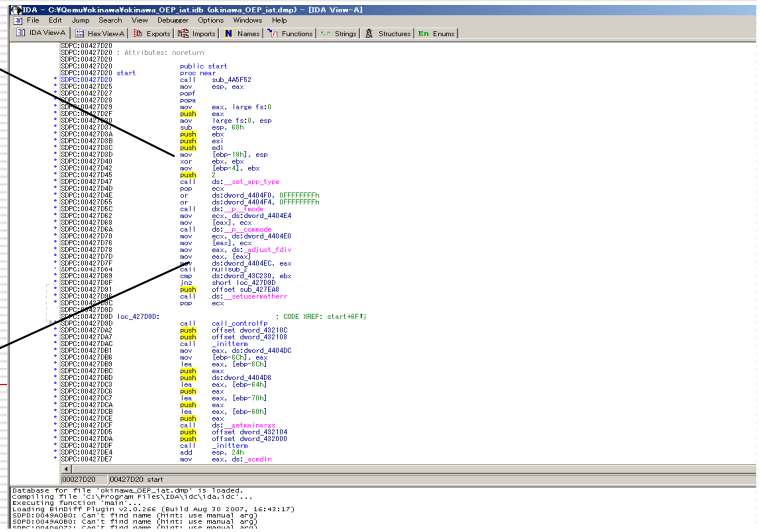
OEP?

アンパッキング

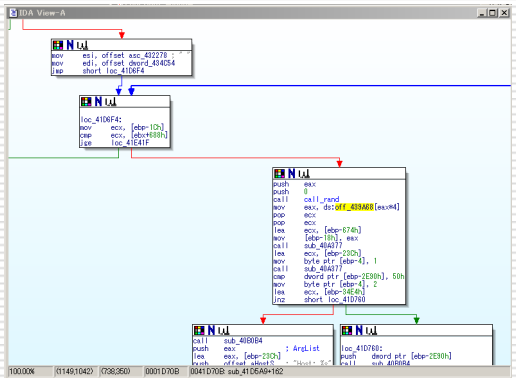
- OEPでブレークしたらメモリをパッチして対象プロセスを無限ループに突入させる
  - #>.write 0x427d20 0xeb
  - #>.write 0x427d21 0xfe
- ゲストOS内からダンプツールを利用して対象プロセスをダンプ
- ImportRECを利用してIATを再構築

従来デバッガでは解析することが困難な検体を容易にアンパッキング

```
SDPC:00411BE7 push    esi
SDPC:00411BE8 push    esi
SDPC:00411BE9 push    esi
SDPC:00411BEA push    [ebp+lpCommandLine]
SDPC:00411BED mov     [ebp+Dst], 44h
SDPC:00411BF4 push    esi
SDPC:00411BF5 mov     [ebp+var_24], si
SDPC:00411BF9 call    ds:CreateProcessA
SDPC:00411BFF test    eax, eax
```



IDA proによる静的解析



- Hostsファイルの書き換え、アンチウィルスベンダ、セキュリティベンダ、OSベンダなどへのWebアクセスの禁止
- HKLM¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run/¥RunServicesへの自身のバイナリの登録
- バイナリ配布用のFTPサーバの立ち上げ
- DLサーバの立ち上げ
- MS03-026/MS02-061?(SQLExec脆弱性)を狙った攻撃コード
- 各種プロセスの停止 F-AGOBOT.EXE, HIJACKTHIS.EXE, _AVPM.EXE, _AVPCC.EXE, _AVP32.EXE, VSCHED.EXE,その他、Anti Virus、セキュリティソフトなどの非常に多くのプロセス
- “login”, “!login”,”hashin”,”!hashin”,”!secure”,”secure”,etcの文字列を含むパケットを盗聴
- その他まだまだ大量の機能

考察

- Qemu検知技術
  - 既存Qemu検知手法の1部(3/5)対処済み
  - 未知の手法により検知される可能性がある
  - 仮想マシンすべてをソフトウェアで記述しているため、理論上は検知されない仮想マシン環境が作れるはず？

- OEPジャンプ箇所検出の有効性(他のパッカーでは？)
  - 単純なパッカーでは誤検知が発生する場合がある
    - 複数の候補の中から人間が目視
  - 未知のパッカーの場合、検出はできない
    - Genericな命令列でとらえるという手はあるが,,,
  - パッカーの振る舞いをもう少し詳細にみる
    - ループの回数＋ジャンプ箇所

まとめ

- 様々なAnti Debug機能を回避しつつデバッグを行うことが可能なステルスデバッガを提案、実装を行った。
  - OSやCPUのデバッグ機能支援機構に頼らないデバッグ機能
  - タイムコントロール機能
  - リソースアクセスモニタ
  - 実行命令に基くブレークポイントによるOEPの検出
- CCC Dataset 2008検体による評価を行い、従来のデバッガでは解析困難な高機能なAnti Debugを搭載した高機能パッカーAをアンパッキングしオリジナルコードの抽出に成功した。