PacSec 2014

TENTACLE:
Environment-Sensitive Malware Palpation

FFRI, Inc.
http://www.ffri.jp

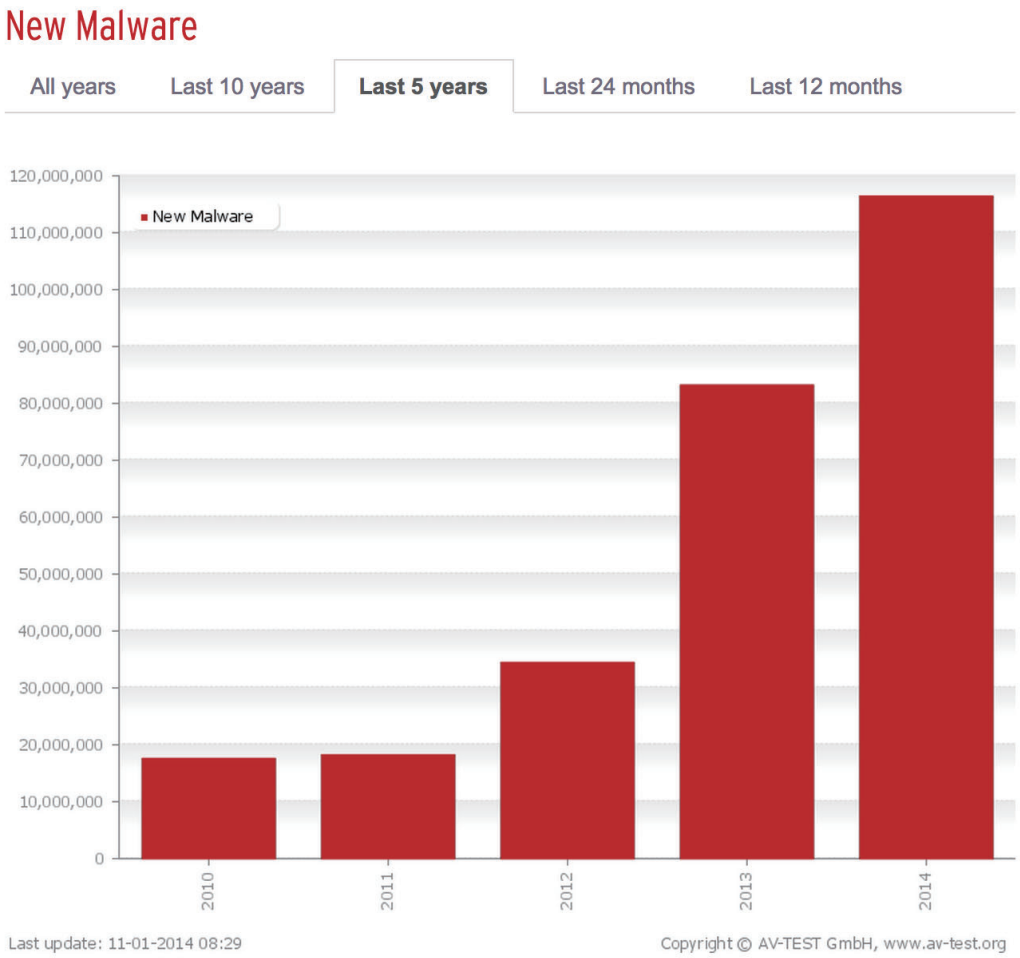
Ver 2.00.01

Contents

- 背景/モチベーション
- アンチサンドボックス技術の分類
- Tentacleの設計思想
- 実装
- 実験
- まとめ

背景

- 日々マルウェアが爆発的に増えている
  - 今年の新種は1.2億体
- アンチウィルスは死んだ？



AV Test: Statistics –New Malware- (Nov. 05 2014 viewed)
http://www.av-test.org/en/statistics/malware/

マルウェア動的解析自動化の必要性

- スケーラビリティは情報大爆発時代の最も重要なファクターの一つ
  - Cloudクラウド
  - ビッグデータ
  - IoT
- マルウェア解析にもスケーラブルな手法が必要
  - 自動化は正義

サンドボックスを使った手動/自動解析

- 解析を行う人間は “壊れてもよい環境”として仮想マシン(VM）やサンドボックスをよく利用する

- 自動解析のために、仮想化技術やアプリケーションサンドボックス技術が使われている
  - Anubis(online sandbox)
  - Cuckoo Sandbox(VirtualBox base)
  - 商用UTMアプライアンス
  - 商用エンドポイントセキュリティ

洗練されたマルウェアの逆襲

- 作り込まれたマルウェアはアンチ解析技術を実装していることが多い
  - 標的型攻撃や、サイバー諜報活動、オンラインバンキングを狙うマルウェアは当然使ってくる

- 研究者の間では、特にアンチ自動解析機能を持っているマルウェアを“evasive malware” と呼ばれている

Related work

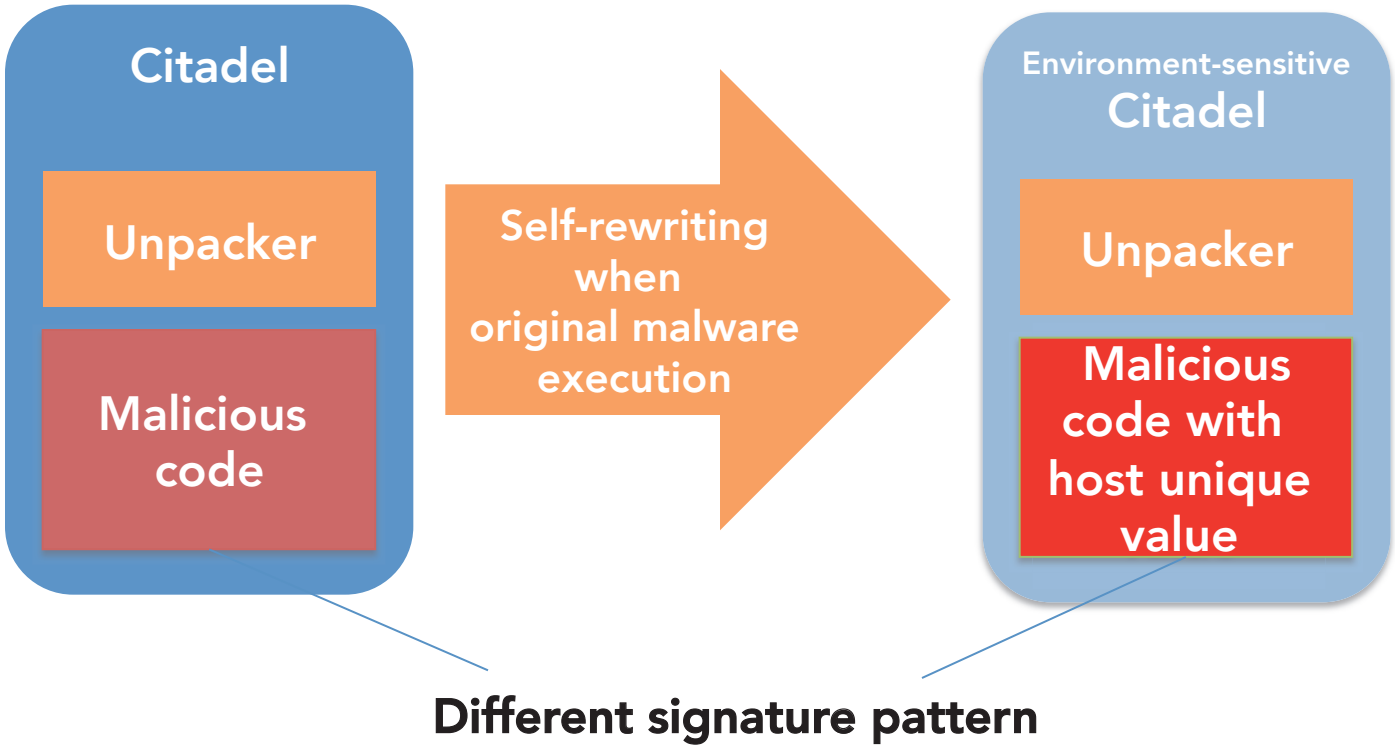
- BareCloud [Dhilung K et al., USENIX SEC'14]
  - "5,835 evasive malware out of 110,005 recent samples"
    - 100,005検体中、evasive malwareを5815検体確認した
- Prevalent Characteristics in Modern Malware [Gabriel et al., BH USA '14]
  - "80% malware detect vmware using backdoor port"
    - 8割のマルウェアがVMWareを検知するロジックを持っている
  - 本当？

ケーススタディ: Citadel

- Citadel検体の中には、感染したホストのみで実行するなど、実行環境によって動作を変えるものがある
  - おそらく動的解析を回避するため
- 実例を2つを紹介
  - Host fingerprinting
  - VM/サンドボックス検知

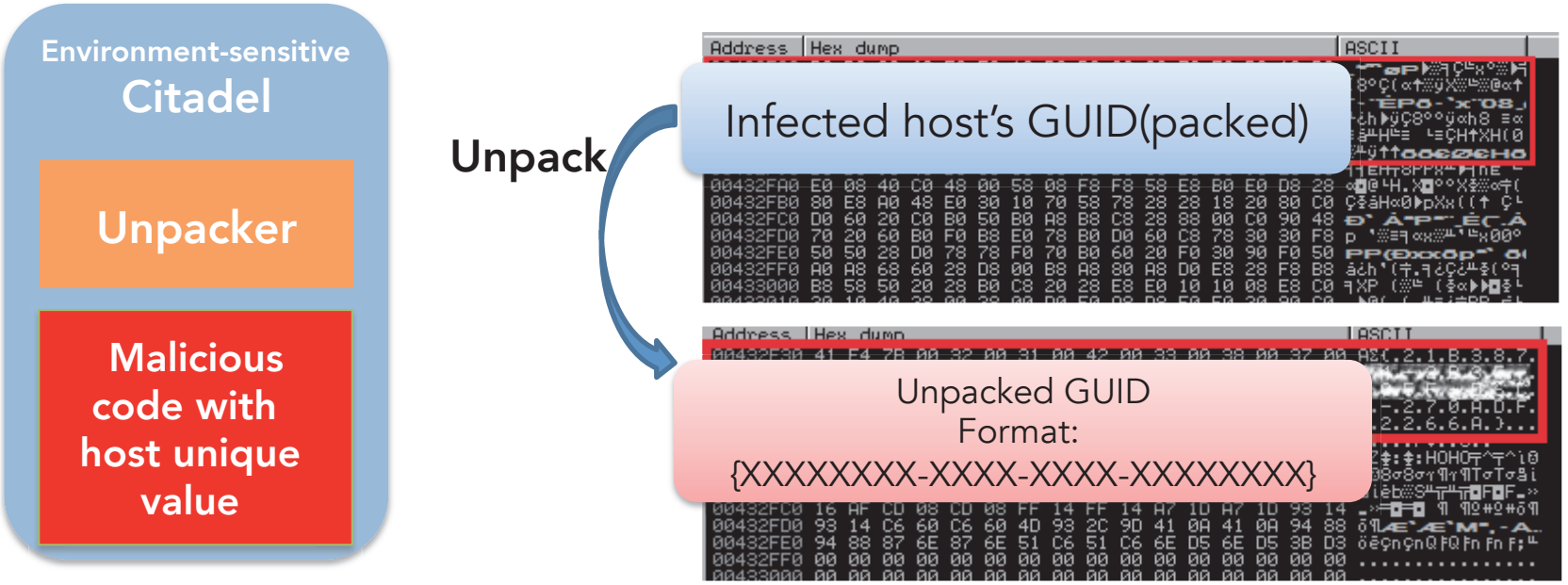
Host fingerprinting

- 感染したホストの環境固有情報を実行ファイルに埋め込む



Host-fingerprinting（続き）

- Citadel系マルウェア(2013年下半期にサンプリング)
  - GetVolumeInformationA()でシステムドライブのGUIDを取得する
  - 環境依存マルウェアは環境依存情報を含んだ自身の実行ファイルをメモリに展開し、取得したGUIDと比較する
  - 一致しなければマルウェアを終了させる
-

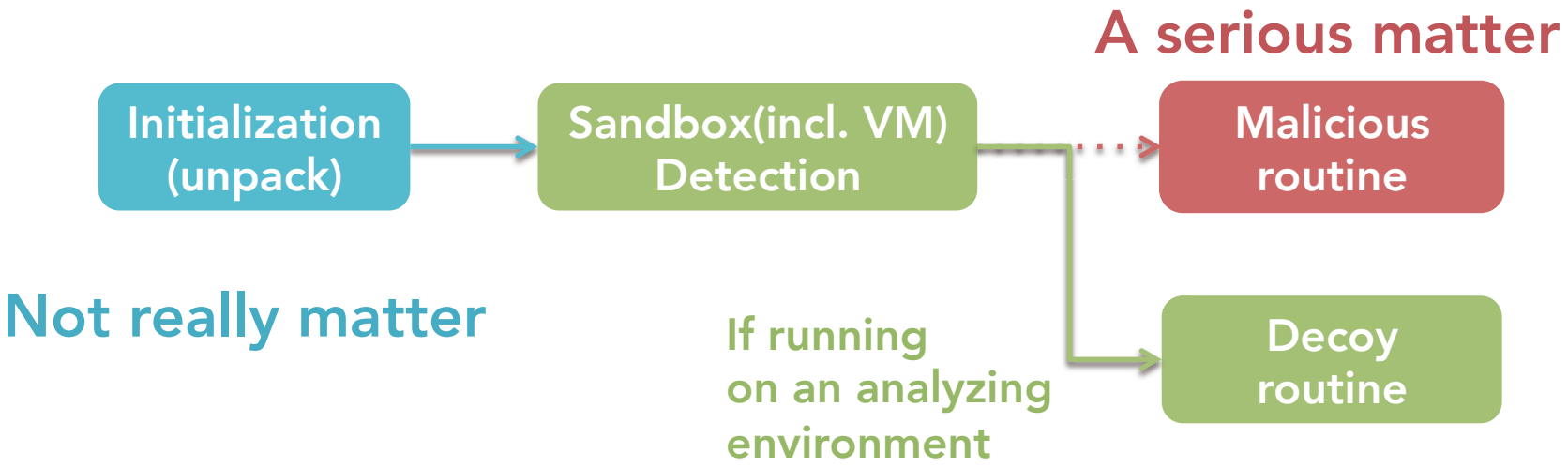


VM/サンドボックス検知

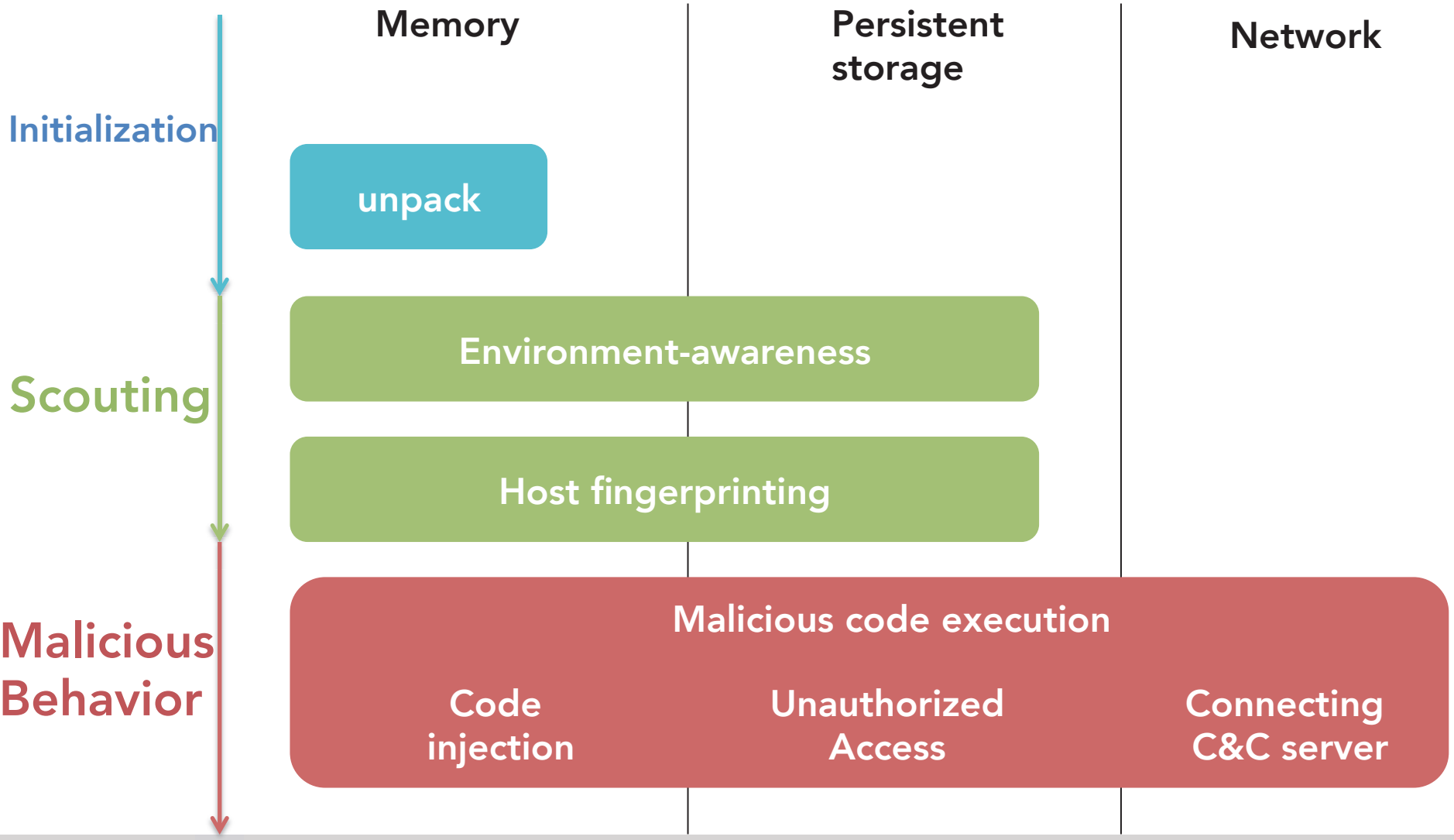
- 動作中のプロセス名に特定の製品に関する文字列がないかチェックする
  - 例えば、”*vmware*”や ”*virtualbox”のような
- 特定のファイルやデバイスが存在するか確認する
  - C:¥popupkiller.exe
  - C:¥stimulator.exe
  - C:¥TOOLS¥execute.exe
  - ¥¥.¥NPF_NdisWanIp
  - ¥¥.¥HGFS
  - ¥¥.¥vmci
  - ¥¥.¥VBoxGuest

検知した場合の動作

- プロセスを終了する
- まったく関係の無い、無害な振る舞いを装う



Citadelの活動分類



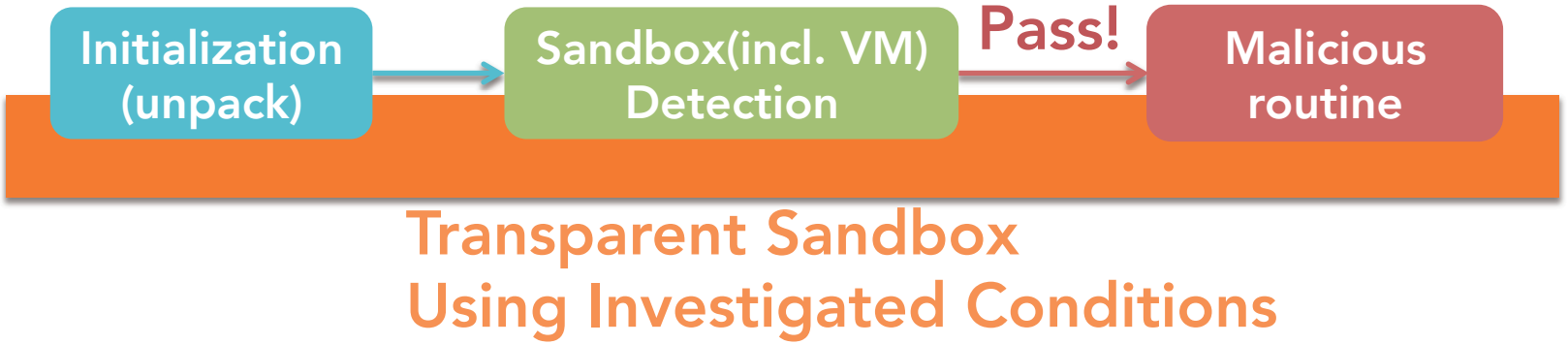
Evasive malwareの狙い

- シグネチャベースのアンチウィルスを回避

- アンチ解析
  - 解析者をてこずらせるため
  - サンドボックスによる自動解析を避けるため

本研究のモチベーション

- マルウェアが用いる、サンドボックス回避に使う条件を自動的に見つけ出したい
  - その条件が分かれば、自動解析サンドボックスは自動的に対解析機能をバイパスすることができる
  - 解析者にとっても非常に有用



“敵を知る”
アンチサンドボックス技術の分類

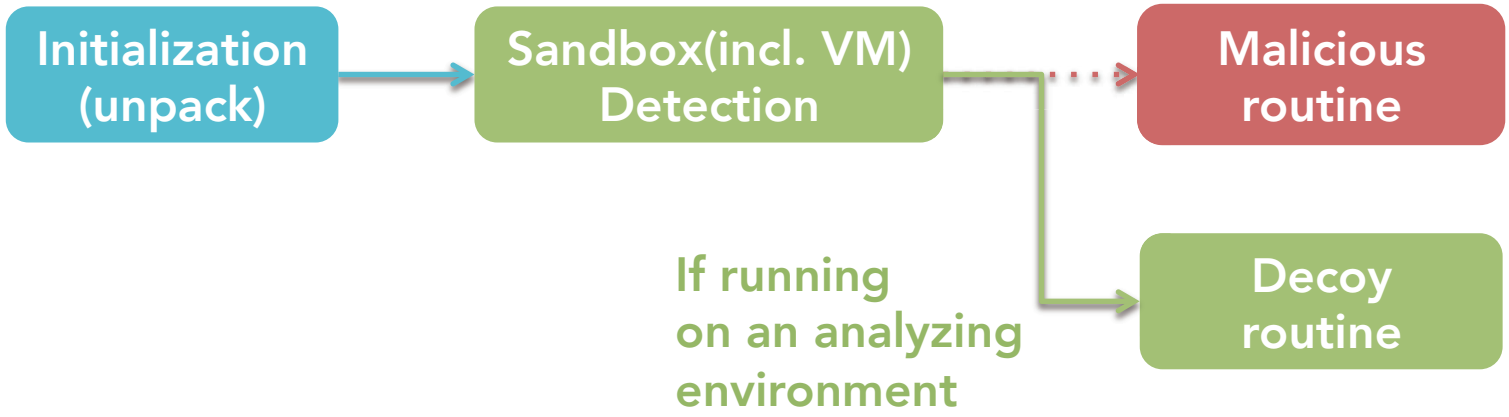
アンチサンドボックス技術の分類

- サンドボックス回避機能
  - Environment awareness
    - Using result of sandbox detection
  - (Stalling code)
  - (User interaction checks)

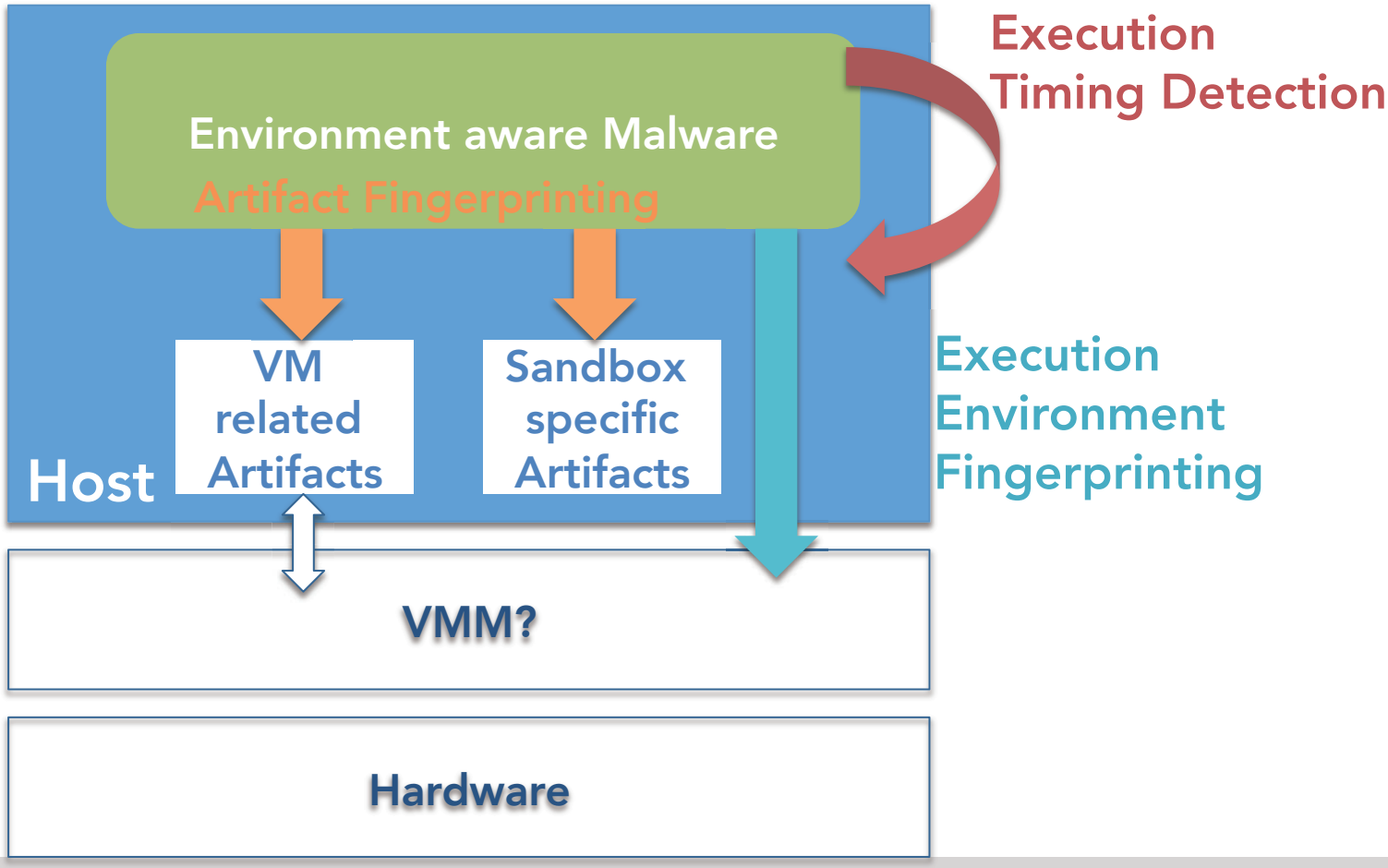
- Sandbox (debug/sandbox/vm) detection
  - Artifact fingerprinting
  - Execution environment fingerprinting
  - Execution timing detection

Environment awareness

- 実行中のホストに関する情報を収集
  - もし解析を目的とした環境なら、無害な振る舞いに切り替える
  - 悪意のある動作は行わない



Sandbox (debug/sandbox/vm) detection



Artifact fingerprinting

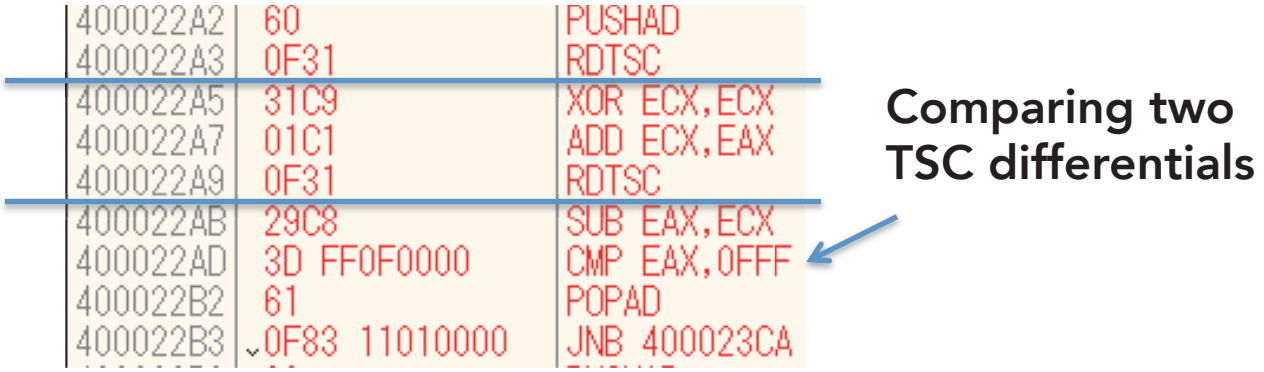
- Sandbox/VM関係のプロセス名を見つける
  - 例えば、vmware, virtualbox
- Sandbox/VM関係のファイルを探す
  - 例えば、vmmouse.sys(Vmware)など
- Sandbox/VM特有のレジストリエントリを探す
- Sandbox/VM特有のデバイスやデバイスの属性（名前など）を探す
- Sandbox/VMが空けるバックドアポートを探す
  - VMWareが使うバックドアポートは有名

Execution environment fingerprinting

- 仮想マシン特有の値や振る舞いを使う
  - CPUID 命令
  - Redpill
    - LDTやGDT, IDTが仮想化環境とベアメタル環境で違う場合があることを利用する

Execution timing detection

- RDTSC命令を2回発行し、その2回の差で仮想化検知をおこなう
  - 昔から、デバッガを検出するために使われている方法だが、仮想化検知にも有効



本研究で対象とするアンチサンドボックス技術

- サンドボックス回避機能
  - ✓ **Environment awareness**
    - Using result of sandbox detection
  - ☐ Stalling code
  - ☐ User interaction checks
- Sandbox (debug/sandbox/vm) detection
  - ✓ **Artifact fingerprinting**
  - ✓ **Execution environment fingerprinting**
  - ☐ Execution timing detection

Automatically disarmament system for armed malware with anti-sandboxing

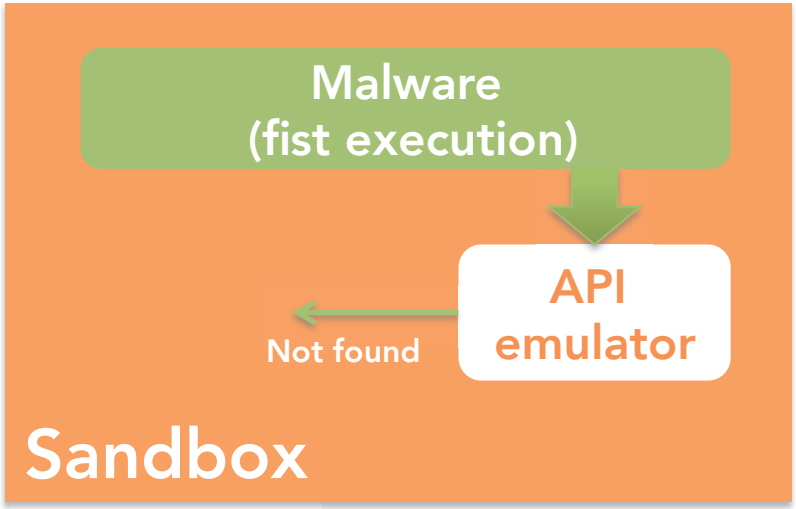
TENTACLEの設計思想

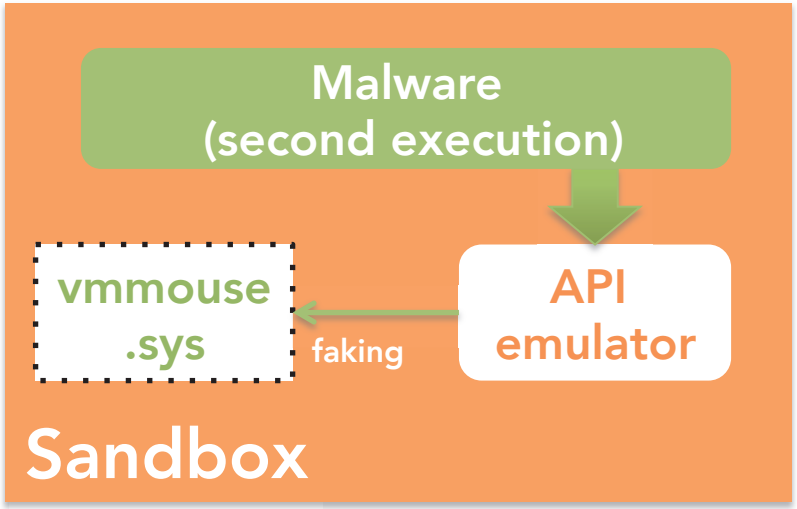
コンセプト: マルウェア触診

1. “Thousand form（千の顔を持つ）” サンドボックスがマルウェアを何度も実行する
   - サンドボックスは実行ごとに異なる仮想的なartifactをマルウェアに見せる
   - コードが分岐した場合、それは仮想化検知を行っているということ
2. Retroactive condition analysis
   - マルウェアが不自然にプロセスを終了させたとき、その終了状態から遡って条件分岐の条件を特定する

マルウェア触診

- サンドボックスが異なる仮想化環境を偽装しつつ、マルウェアを実行する
- 実行中にコードが分岐したかを確かめるため、Code Execution Integrity(CEI)という独自の検証技術を使う





Code Execution Integrity(CEI)

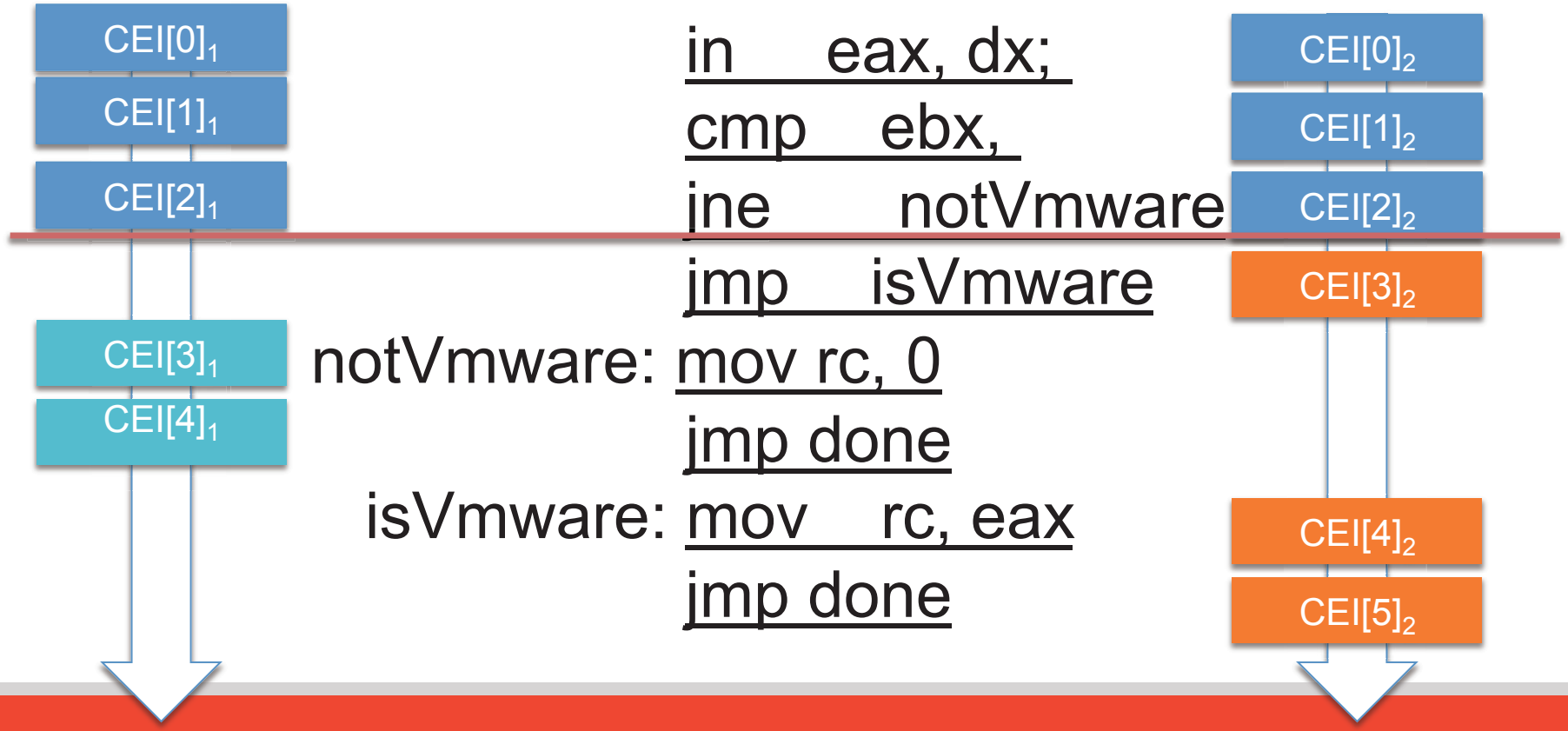
- CEIは以下の計算式で、命令実行ごとに計算する
  - TPM(Trusted computing base)のtrust chain計算を真似している
- 同じ実行ステップ数でかつ、CEIの値が同じ場合、2つの実行履歴は完全に同一

Digest[i] = SHA1(fetched CPU instruction + Digest[i-1])

```
mov    $0x616b6157, %eax   0xb857616b61      d[0] = SHA1(0xb857616b61)
push   %ebx                0x53              d[1] = SHA1(d[0] + 0x53)
push   %eax                0x50              d[2] = SHA1(d[1] + 0x50)
mov    $4, %edx            0xba04000000      d[3] = SHA1(d[2]
mov    $1, %ebx            0xbb01000000                  +0xba04000000)
                                             ...
```

Execution branch detection

- ステップ数が同じでもCEIの値が違う場合、異なる条件分岐を経ていると判断できる



Retroactive condition analysis

- プロセス終了時から遡り、終了直前の分岐とその条件に関連したAPI、引数を列挙

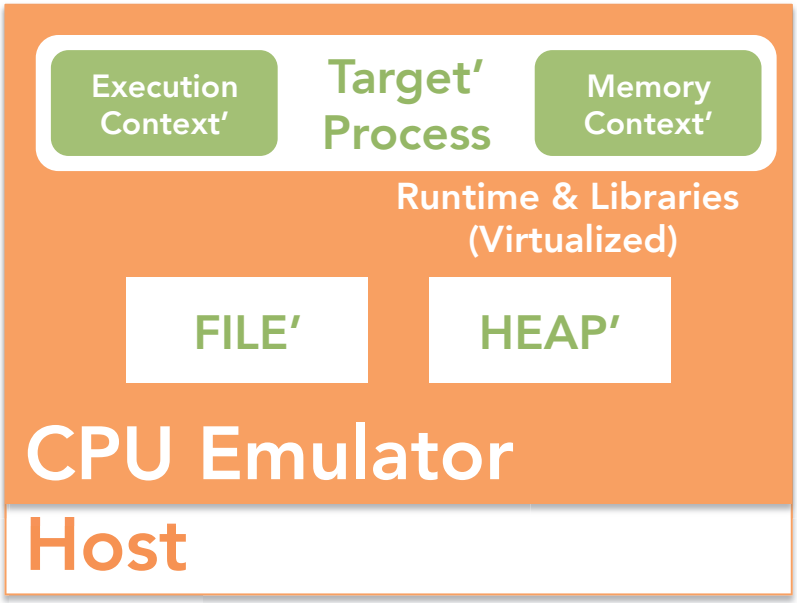
- 不自然なプロセス終了時に解析を行う
  - 非常に少ないステップ数で終了したとき
  - ネットワーク通信を一切行わずに終了したとき

```
sub esp, 1024
mov ebx, esp
push 400h
push ebx
push 0h
call GetModuleFileNameA
lea eax, MyPath
push eax
push ebx
call lstrcmpA
test eax, eax
push 0h
lea eax, MsgCaption
push eax
jz _ok
lea eax, NGMsgText
push eax
push 0h
call MessageBoxA
invoke ExitProcess, NULL
_ok:    lea eax, OKMsgText
```

“己を知る”
TENTACLEの実装

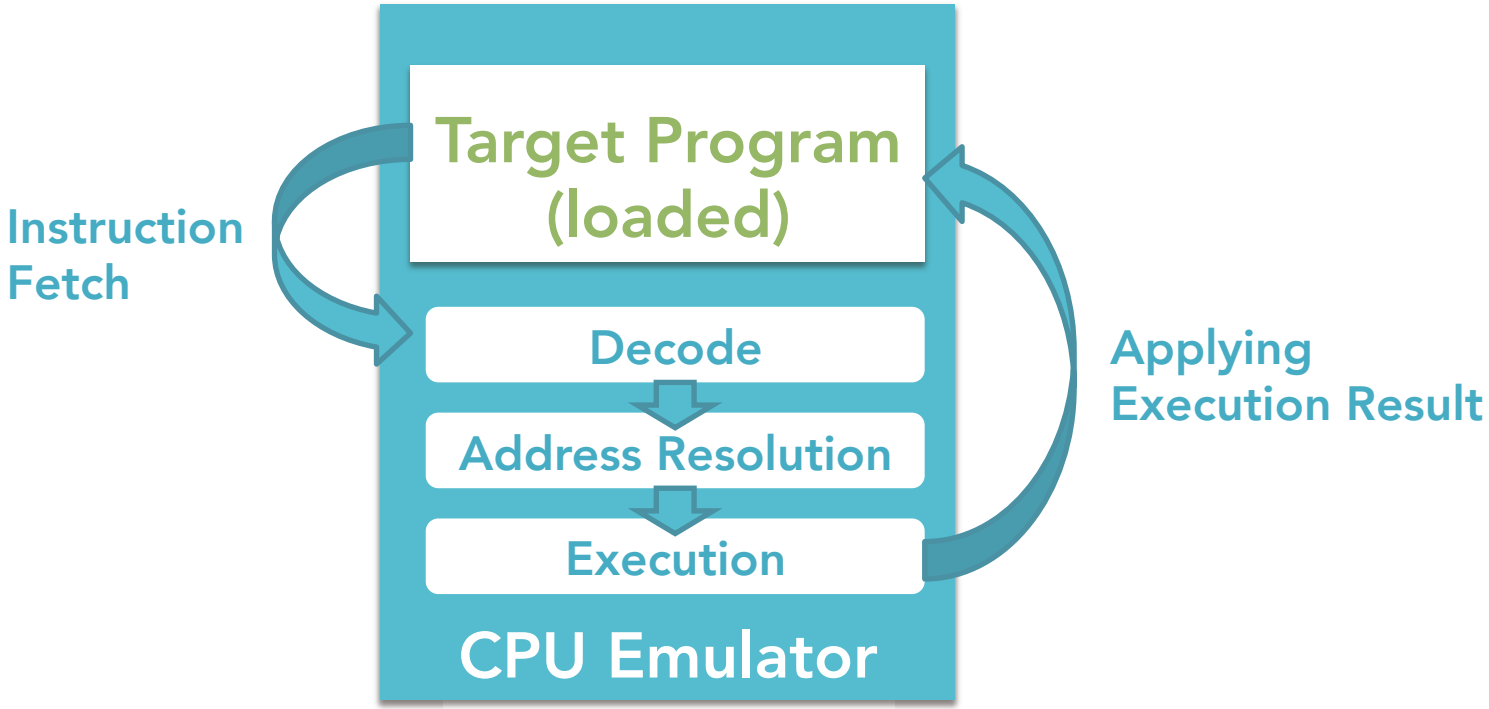
Tentacleの概要

- ソフトウェアベースのCPUエミュレータを使って実装

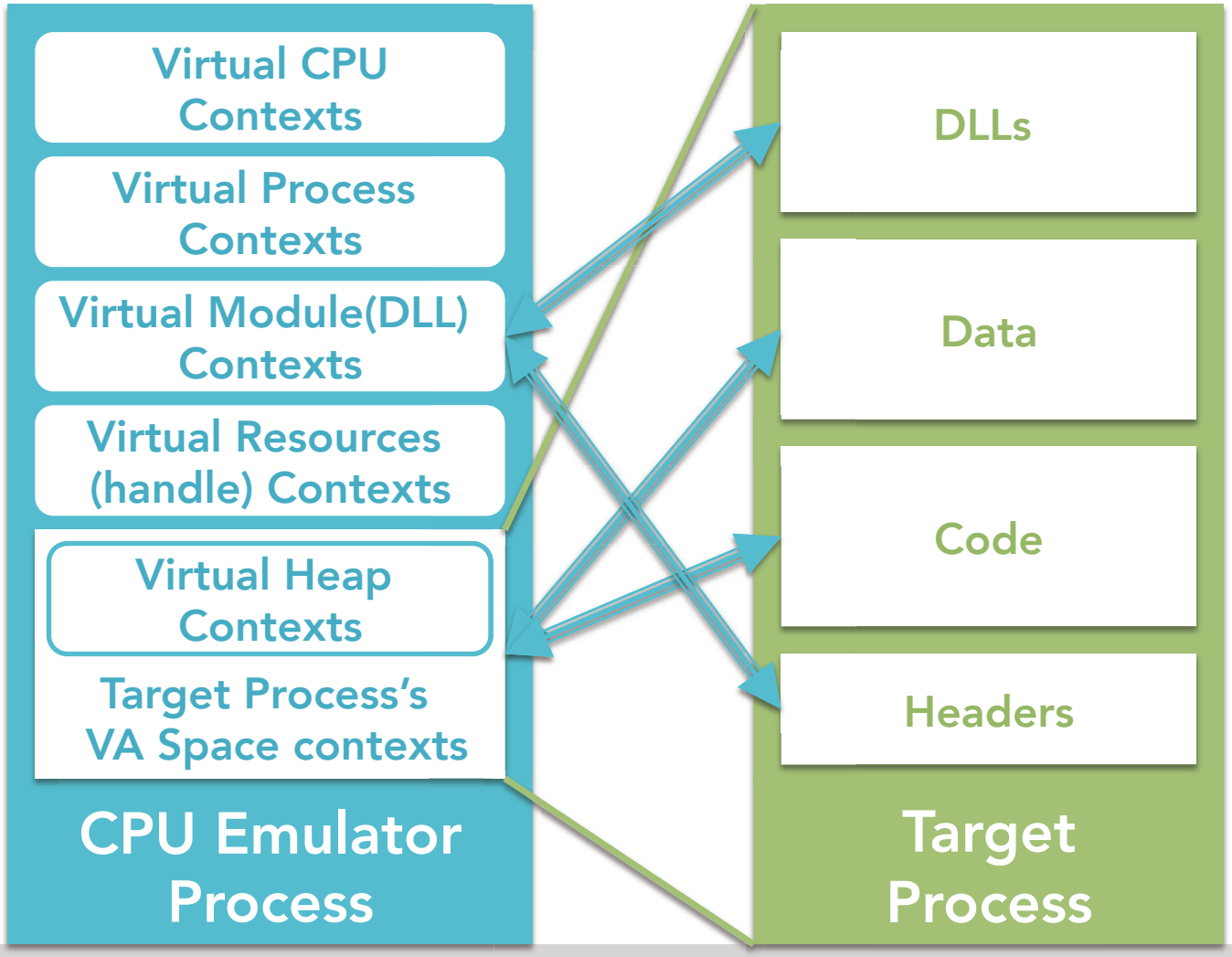


ソフトウェアベースのCPUエミュレータ

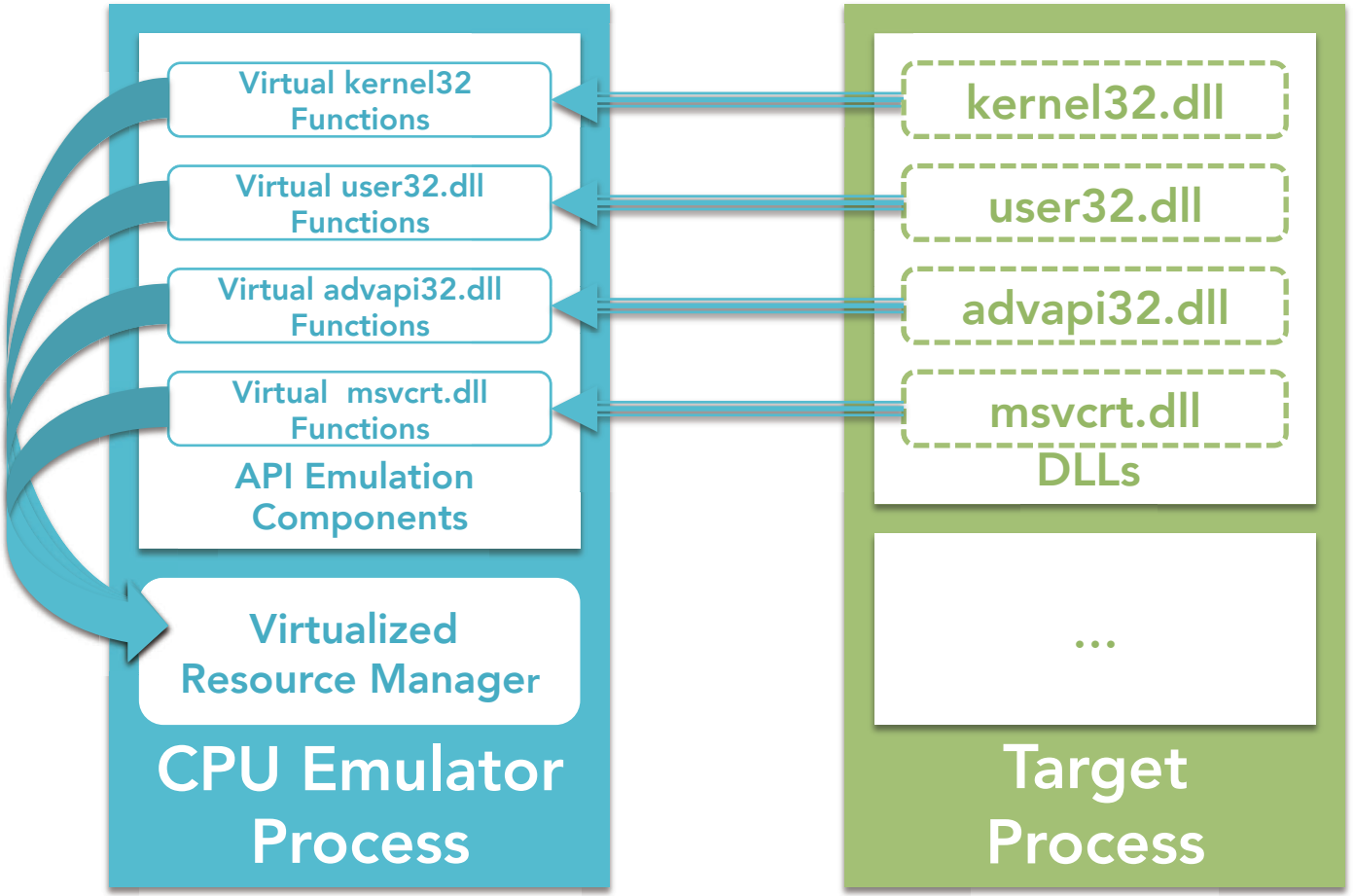
- 社内ツールの一つである独自のIA-32エミュレータを利用
  - 機能はIDA Bochs PE operation modeとだいたい同じ



IA-32 CPU Emulator: 仮想コンテキスト



IA-32 CPU エミュレータ: APIエミュレーション



制約

- ベースとしたIA-32エミュレータは、部分的なAPIエミュレーションしかできていない
- 同じく、サポートしている命令セットも限定的

Tentacleの主要技術

- Nyarlathotep “Thousand Form” Sandbox
  - サンドボックス/VMに偽装する機能を持つ
  - 実行履歴を比較し、異なる条件で分岐が発生したかを検出することが可能
- Retroactive condition analysis
- Necronomicon
  - 上記2つを実現するためのロギングフレームワーク

Necronomicon

- Necrnomiconは実行した命令をロギングする
  - API実行（call命令）の場合はポインタと引数そのものをロギング
  - Retroactive condition analysisに必要な情報を収集
    - 取得するAPIの例：lstrcmp, GetModuleFileName, GetVolumeNameForVolumeMountPoint…
- 同時にCode execution integrity（CEI）を計算し、保持する
- サンドボックスがマルウェアをNステップ実行したときは、Nページ分のログが取得される

Nyarlathotep “Thousand Form” Sandbox

- 今回はVMWareの偽装機能を実装
  - 仮想的にバックドアポートを見せる
  - VMWareのゲストドライバファイルを仮想的に見せる
  - VMWare Tools関係のレジストリエントリを偽装する

- 最初は“すべてのartifactを無効”と “すべてのartifactを有効”にした2つの実行結果を使って、仮想化検知の有無を調査
  - もし検知があった場合、その理由を調べるために “すべてのartifactを無効”と“一つのartifactを有効”にした場合の実行履歴を比較し、検知条件を特定する

実験

- Disarmament #01: Artifact fingerprinting による仮想化検知の理由特定
- Disarmament #02: Disk GUID を用いたサンドボックス回避の条件特定
- Disarmament #03: 実行ファイルのファイルパスを用いたサンドボックス回避の条件特定

Disarmament #01: Artifact fingerprinting による仮想化検知の理由特定

```
int _tmain(int argc, _TCHAR* argv[])
{
        int count = 0;
        delay();
        if ( sbdetect_vmware_backdoor_port() > 0){
                printf("VMWare backdoor port detected.  I am on the virtual.");
        }
        if ( sbdetect_vmware_sysf01() == 0 ){
                printf("vmmouse.sys detected. I am on the virtual.\n");
                exit(1);
        }
        if ( sbdetect_vmware_sysf02() == 0 ){
                printf("vmmouse.sys detected. I am on the virtual.\n");
                exit(1);
        }
        if ( sbdetect_vmware_reg01() == 0 ){
                printf("RegKey: \"SOFTWARE\\VMware, Inc.\\VMware Tools\"
detected. I am on the virtual.\n");
                exit(1);
        }
        //Malicious behavior
        printf("malicious behavior\n”);
        return 0;
}
```

sbdetect_vmware_backdoor_port()

```
int sbdetect_vmware_backdoor_port(void)
{
    int rc = 0;
    __try
    {
        __asm
        {
            mov    eax, 'VMXh'
            mov    ebx, 0;
            mov    ecx, 0xA
            mov    edx, 'VX'  // port
            in     eax, dx; // read port
            cmp    ebx, 'VMXh' // Vmware echo 'VMXh'
            jne      notVmware
            jmp    isVmware
            notVmware:
            mov rc, 0
            jmp done
            isVmware:
            mov    rc, eax
            done:
        }
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        rc = 0;
    }

    return rc;
}
```

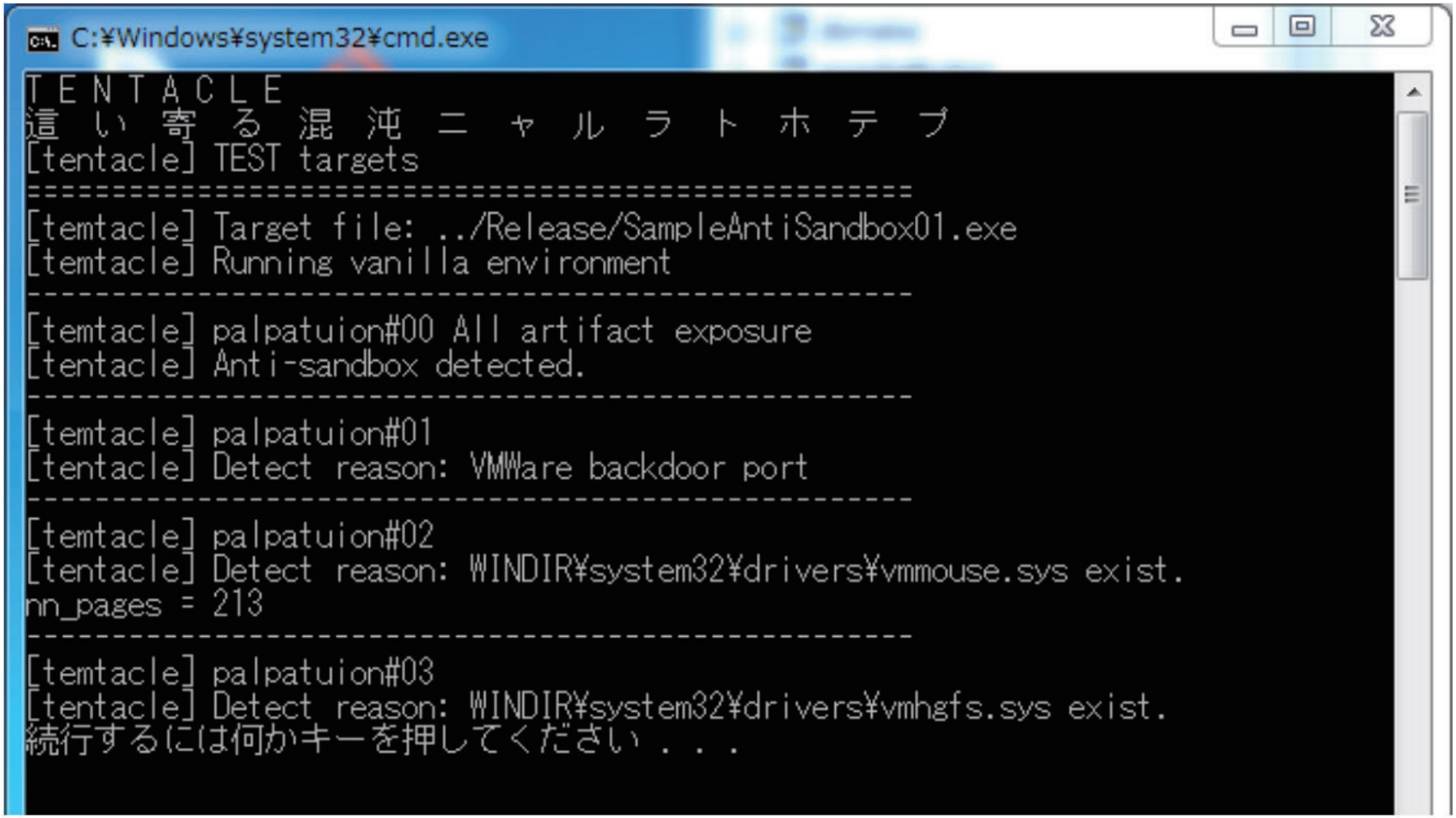
sbdetect_vmware_sysf01()

```
//
// Detect proven: Windows 7 32bit
// Not detected:  Windows 7 64bit
//
int sbdetect_vmware_sysf01() {
    DWORD ret;
            TCHAR target[255] = "%WINDIR%\\system32\\drivers\\vmmouse.sys";
            ret = GetFileAttributes(target);
    if( ret != INVALID_FILE_ATTRIBUTES ){
        return 0;
    }
            else{
        return 1;
    }
}
```

sbdetect_vmware_sysf02()

```
//
// Detect proven: Windows 7 32bit
// Not detected:  Windows 7 64bit
//
int sbdetect_vmware_sysf02() {
    DWORD ret;
            TCHAR target[255] = "%WINDIR%\\system32\\drivers\\vmhgfs.sys";
            ret = GetFileAttributes(target);
    if( ret != INVALID_FILE_ATTRIBUTES ){
        return 0;
    }
            else{
        return 1;
    }
}
```

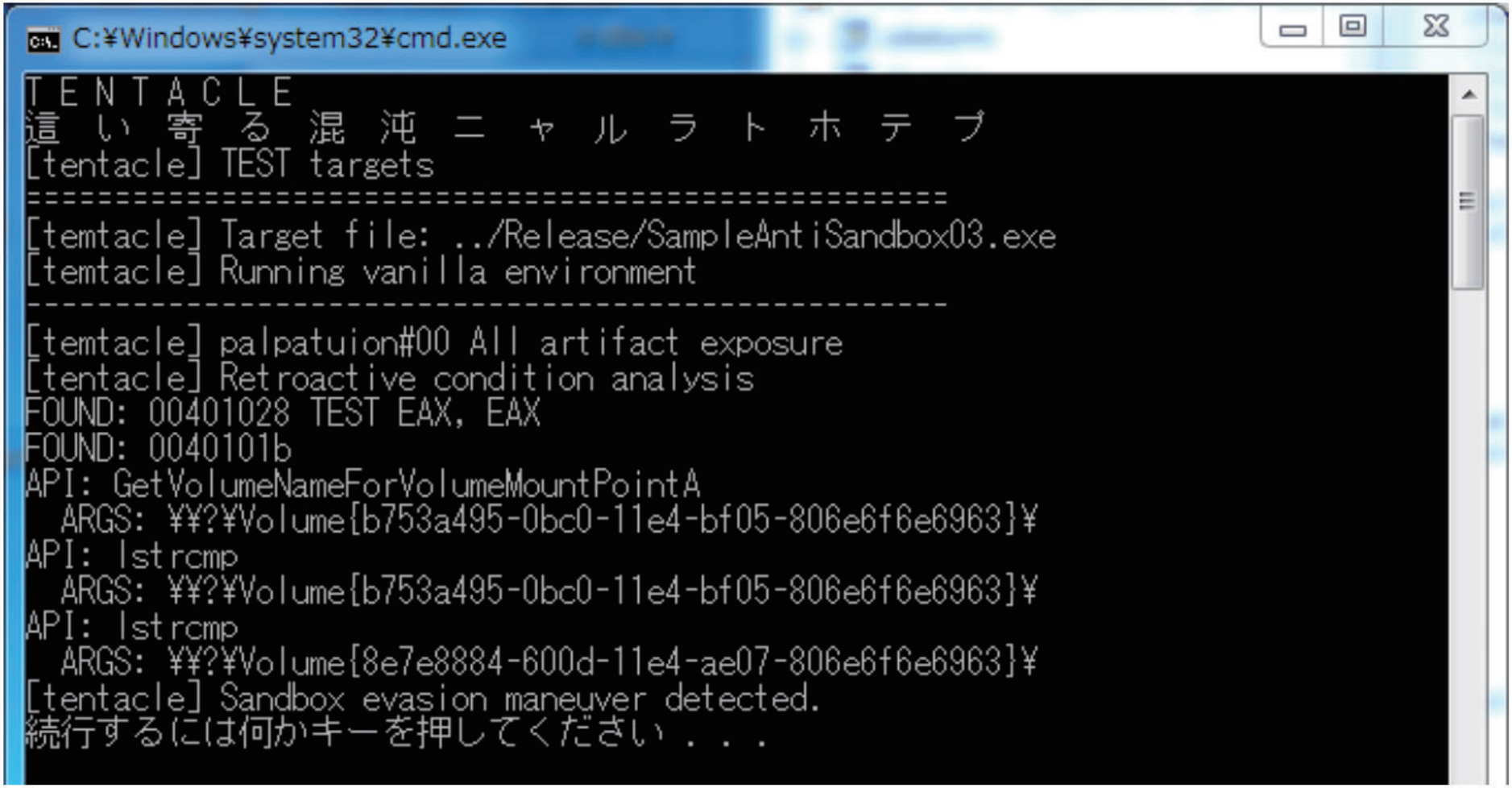
Disarmament #01



Disarmament #02: Disk GUID を用いたサンドボックス回避の条件特定

```
.386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
.data
                DriveC     db "C:\", 0
                VolumeC    db "\\?\Volume{8e7e8884-600d-11e4-
ae07-806e6f6e6963}\", 0
                MsgCaption db "MESSAGE", 0
                OKMsgText  db "Normal Message", 0
                NGMsgText  db "Detect Message", 0
.code
start:
sub esp, 1024
mov ebx, esp
sub esp, 4
mov eax, esp
push eax
push ebx
lea eax, DriveC
push eax
call GetVolumeNameForVolumeMountPointA
lea eax, VolumeC
push eax
push ebx
call lstrcmpA
test eax, eax
push 0h
lea eax, MsgCaption
push eax
jz _ok
lea eax, NGMsgText
push eax
push 0h
call MessageBoxA
invoke ExitProcess, NULL
_ok:
lea eax, OKMsgText
push eax
push 0h
call MessageBoxA
invoke ExitProcess, NULL
end start
```

Disarmament #02



Disarmament #03: 実行ファイルのファイルパスを用いたサンドボックス回避の条件特定

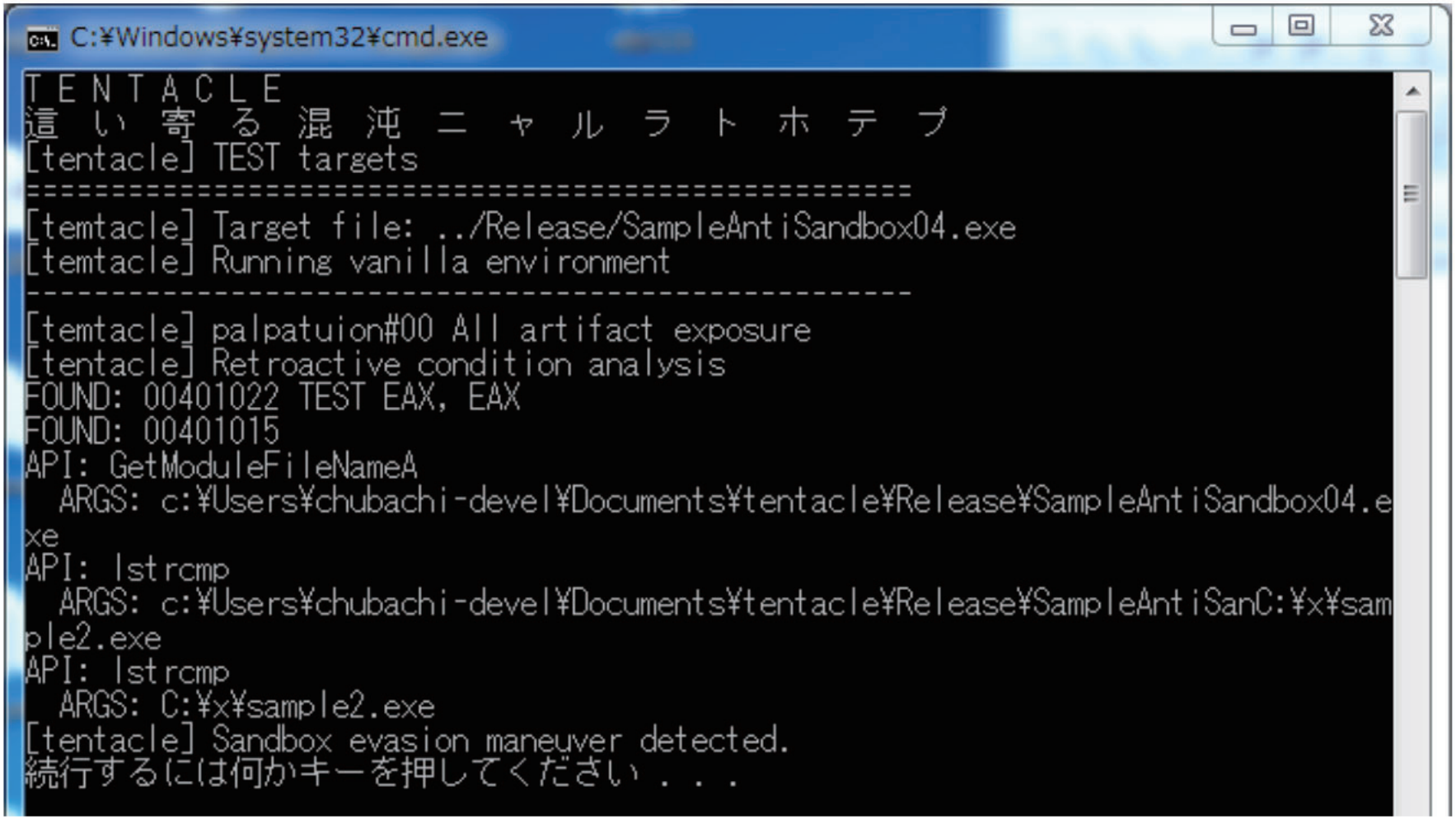
```
.386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
.data

        MyPath     db "C:\x\sample2.exe", 0
        MsgCaption db "MESSAGE", 0
        OKMsgText  db "Normal Message", 0
        NGMsgText  db "Detect Message", 0
```

```
.code
start   sub esp, 1024
        mov ebx, esp
        push 400h
        push ebx
        push 0h
        call GetModuleFileNameA
        lea eax, MyPath
        push eax
        push ebx
        call lstrcmpA
        test eax, eax
        push 0h
        lea eax, MsgCaption
        push eax
        jz _ok
        lea eax, NGMsgText
        push eax
        push 0h
        call MessageBoxA
        invoke ExitProcess, NULL
_ok:    lea eax, OKMsgText
        push eax
        push 0h
        call MessageBoxA
        invoke ExitProcess, NULL

end start
```

Disarmament #03



Future work

- 今回ターゲットとしなかったアンチサンドボックス技術への取り組み
  - Stalling codeの検出、及び克服など
  - 多くの実例を取り込む
- CPUエミュレータのクオリティ向上
  - より多くのマルウェアを調査することが可能になる
  - 本質的には他のCPUエミュレータでもよいので、そちらも検討していく

まとめ

- 自動的にサンドボックス回避条件を解析する手法を提案し、検証のためのプロトタイプを実装した
- アンチサンドボックス技術を部分的に体系化
  - まだまだ追い切れていないアンチサンドボックス技術もあるので、今後も継続してリサーチしていく

“彼を知りて己を知れば、百戦して殆うからず”- 孫子

参考文献

- Analyzing Environment-Aware Malware, Lastline, 2014.05.25(viewed) http://labs.lastline.com/analyzing-environment-aware-malware-a-look-at-zeus-trojan-variant-called-citadel-evading-traditional-sandboxes
- Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. 2011. Detecting environment-sensitive malware. In *Proceedings of the 14th international conference on Recent Advances in Intrusion Detection* (RAID'11). Springer-Verlag, Berlin, Heidelberg, 338-357.
- lemens Kolbitsch, Engin Kirda, and Christopher Kruegel. 2011. The power of procrastination: detection and mitigation of execution-stalling malicious code. In Proceedings of the 18th ACM conference on Computer and communications security (CCS '11). ACM, New York, NY, USA, 285-296.
- Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. 2009. Emulating emulation-resistant malware. In Proceedings of the 1st ACM workshop on Virtual machine security (VMSec '09). ACM, New York, NY, USA, 11-22.
- Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. 2014. Barecloud: bare-metal analysis-based evasive malware detection. In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, Berkeley, CA, USA, 287-301.
- Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. 2009. A view on current malware behaviors. In Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more (LEET'09). USENIX Association, Berkeley, CA, USA, 8-8.
- Aurélien Wailly. Malware vs Virtualization The endless cat and mouse play, 2014.05.25(viewed) http://aurelien.wail.lv/publications/hip-2013-slides.html
- Lorenzo Martignoni, Roberto Paleari, Giampaolo Fresi Roglia, and Danilo Bruschi. 2009. Testing CPU emulators. In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). ACM, New York, NY, USA, 261-272.
- Hao Shi, Abdulla Alwabel and Jelena Mirkovic. 2014. Cardinal Pill Testing of System Virtual Machines. In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, Berkeley, CA, USA,271-285.
- Lorenzo Martignoni, Roberto Paleari, Giampaolo Fresi Roglia, and Danilo Bruschi. 2010. Testing system virtual machines. In Proceedings of the 19th international symposium on Software testing and analysis (ISSTA '10). ACM, New York, NY, USA, 171-182.
- IDA Boch PE operation mode https://www.hex-rays.com/products/ida/support/idadoc/1332.shtml

Thank you !

FFRI, Inc.
http://www.ffri.jp