L-011

アンチデバッグ機能に着目したマルウェア検出手法
Malware Detection Method Focusing on Anti-Debugging

吉崎広太 † 山内利宏 †
Kota Yoshizaki Toshihiro Yamauchi

1. はじめに

近年，マルウェアの数は増え続け [1]，マルウェアによる被害は国内外を問わず社会問題となっている．

マルウェアの判定手法には，動的解析手法がある．動的解析手法は，デバッガを用いてマルウェアを実行し，大まかな動作を把握する手法であり，解析にかかる時間は短い．素早くマルウェアを見つけるために，セキュリティベンダは，時間のかからない動的解析した後，静的解析を行う [2]．しかし，アンチデバッグ機能と呼ばれる解析の妨害機能を持つマルウェアが存在する．

そこで，本稿では，アンチデバッグ機能に着目し，マルウェアを検出する手法を検討する．本手法は，アンチデバッグ機能に用いられる Windows API（以降，API と呼ぶ）の戻り値を変更する．また，マルウェアと正規のアプリケーション（以降，AP と呼ぶ）は，解析されていないと認識した場合に挙動の差異がある．本手法はこの挙動の差異に着目する．

2. アンチデバッグ機能に着目したマルウェア検出手法

2.1. アンチデバッグ機能

表 1 は，API を用いるアンチデバッグ機能一覧である．アンチデバッグ機能とは，リバースエンジニアリングからソフトウェアを保護する目的で，ソフトウェアに付与される耐解析機能である．多くのマルウェアは，この機能をマルウェア解析者による解析の妨害に利用する [3]．

表 1 アンチデバッグ機能一覧

動作の種類	使用される API
プロセスの検知	CreateToolhelp32Snapshot Process32First Process32Next
ウィンドウの検知	EnumWindows GetClassName
デバイスファイルの検知	CreateFile
実行時間の計測	GetTickCount timeGetTime QueryPerformanceCounter
マルウェア自身の再起動	CreateProcess
BeingDebugged フラグの取得	IsDebuggerPresent
デバッガの特定のプロセスへのアタッチ	ChechRemoteDebuggerPresent
スレッドのコンテキスト取得	GetThreadContext
API の戻り値取得	OutputDebugString
デバッガのアタッチ	CreateProcess DebugActiveProces

2.2. 基本方式

本手法は，解析されていると認識させる状態と解析されていないと認識させる状態のそれぞれの状態で AP を実行する．正規の AP とマルウェアは，解析されていないと認識した場合に挙動の差異がある．このため，本手法は，この挙動の差異に着目してマルウェアを検出する．

アンチデバッグ機能を持つ AP は，アンチデバッグ機能を呼び出した結果，自身が解析されていないと判断した場合は，本来の動作を行い，解析されていると判断した場合は，本来の動作とは異なる動作を行う．この本来の動作とは異なる動作を回避動作と呼ぶ．回避動作には，プロセス自身の動作の停止や AP 自身の実行ファイルの削除がある．

本手法では，API を用いるアンチデバッグ機能を捕捉し，解析されていると認識させ，AP を実行する．解析されていると認識した AP は，回避動作を行う．また，同様に，解析されていないと認識させ，AP を実行する．解析されていないと認識した AP は，マルウェアであれば，悪意ある動作を行い，正規の AP であれば，悪意ある動作を行わない．このため，回避動作と悪意ある動作を行う AP はマルウェアであるといえる．

本手法では，API フック機構とマルウェア検出機構によってマルウェアを検出する．以下は，解析されていると認識させる状態でマルウェアを実行した場合の処理の流れである．

(1) マルウェア検出機構は，解析されているとマルウェアに認識させる．
(2) マルウェアは，回避動作を行う．
(3) マルウェア検出機構は，マルウェアが行った回避動作をログファイルに書き出す．

以下は，解析されていないと認識させる状態でマルウェアを実行した場合の処理の流れである．

(1) マルウェア検出機構は，解析されていないとマルウェアに認識させる．
(2) マルウェアは，悪意ある動作を行う．
(3) マルウェア検出機構は，ログファイルの回避動作の情報と悪意ある動作の情報をもとに，マルウェアを検出する．
(4) マルウェア検出機構は，計算機利用者にマルウェアの検出を通知する．

本手法では，スタートアップに登録するためのレジストリキーへのエントリの追加を悪意ある動作と定義する．これは，多くのマルウェアは，計算機の起動時に自身が自動で実行されるように，レジストリキーにエントリを追加するためである [4]．レジストリキーへのエントリの追加には，API が用いられる．このため，本手法では，レジストリキーにエントリを追加する API をフックした日付，時刻，プロセス ID，API 名，および API の引数を悪意ある動作の情報と定義する．

2.3. 実現方式

2.3.1. 課題

以下に，本手法を実現するための課題を示す．

(課題 1) アンチデバッグ機能に用いられる API をフックできること

†岡山大学大学院自然科学研究科，Graduate School of Natual Science and Technology, Okayama University

(課題 2) 回避動作に用いられる API をフックし，フックした API に関するログを出力できること
(課題 3) スタートアップに登録するためのレジストリキーを開くか生成する API をフックできること
(課題 4) 書き出された回避動作のログと悪意ある動作の情報をもとにマルウェアを検出できること

すべての課題に対処するために，API フック機構とマルウェア検出機構を実現した．以下に述べる (1)〜(3) は，次節の番号と対応している．(課題 1) は (1) と (2) の API のフックにより解決する．また，(課題 2) は，(3) の API をフックし，この API に関するログの出力により解決する．(課題 3) は，レジストリキーを開くか新たに生成する (2) の API のフックにより解決する．(課題 4) は，マルウェア検出機構が書き出したログと悪意ある動作の情報をもとにマルウェアを判定することにより解決した．

2.3.2. API フック機構

API フック機構は，以下の API をフックし，フックで得た回避動作の情報をログファイルに書き出し，悪意ある動作の情報をマルウェア検出機構に渡す機構である．

(1) IsDebuggerPresent
(2) RegCreateKeyExA
(3) ExitProcess

本手法は，回避動作として呼び出される API をフックした場合，フックした API の情報 (以降，回避動作の情報と呼ぶ) として，API をフックした日付，時刻，プロセス ID，API 名，および API の引数をログファイルに書き出す．

2.3.3. マルウェア検出機構

マルウェア検出機構は，以下の 2 つの条件を満たしているとき，AP をマルウェアとして検出する．

(1) ログファイルに回避動作の情報が書き出された
(2) API フック機構が悪意ある動作の情報を検出した

マルウェアを検出した場合，計算機利用者にメッセージボックスを用いて通知する．

2.4. 期待される効果

本手法の実現により，以下の 2 つの効果を期待できる．

(1) 素早くマルウェアを検出できる．
(2) ウイルス対策ソフトによるヒューリスティック検出として利用できる．

3. 評価

3.1. 評価環境

Windows Vista SP2 上で，マルウェアに回避動作もしくは悪意ある動作を行わせ，評価した．評価では，アンチデバッグ機能を持つボットである Agobot を評価検体に用いた．Agobot は，IsDebuggerPresent API によってデバッガを検知するアンチデバッグ機能を持つ．Agobot は，外部ネットワークとのコネクションを確立できない場合は，動作を中止する．このため，評価では，コネクションの確立前にアンチデバッグ機能を呼び出すように，Agobot を改変した．

```
2013/2/7,16:01:21:294,1752,IsDebuggerPresent,VOID
2013/2/7,16:01:22:567,1752,ExitProcess,VOID
```

図 1 解析されていると認識させる状態で Agobot を実行したときのログ

```
2013/2/7,16:01:21:294,1752,IsDebuggerPresent,VOID
2013/2/7,16:01:22:567,1752,ExitProcess,VOID
2013/2/7,16:15:25:802,3780,IsDebuggerPresent,VOID
2013/2/7,16:15:26:673,3780,RegCreateKeyExA,SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run
```

図 2 解析されていないと認識させる状態で Agobot を実行したときのログ

3.2. 結果

はじめに，解析されていると認識させる状態で Agobot を実行した．具体的には，IsDebuggerPresent API が常に TRUE を返す状態で Agobot を実行した．このときのログを図 1 に示す．図 1 より，ExitProess API の呼び出しを確認できる．また，IsDebuggerPresent API を呼び出してから ExitProcess API によって自身の動作を停止するまでの時間は，約 1 秒であることが分かる．本手法は，アンチデバッグ機能の呼び出しから 5 秒以内に自身の動作を停止した場合に，この動作を回避動作と判定する．このため，Agobot は，アンチデバッグ機能を呼び出し，解析されていると認識し，回避動作を行ったといえる．次に，解析されていないと認識させる状態で Agobot を実行した．具体的には，IsDebuggerPresent API が常に FALSE を返す状態で Agobot を実行した．このときのログを図 2 に示す．図 2 より，RegCreateKeyExA API の呼び出しを確認できる．この API の引数は `SOFTWARE\Microsoft\Windows\CurrentVersion\Run` である．これは，計算機の起動時の自動実行を設定するレジストリキーである．このため，4 行目の RegCreateKeyExA API の呼び出しは，悪意ある動作である．マルウェア検出機構は，回避動作の情報を図 1 のログファイルから読み込み，悪意ある動作の情報を API フック機構から受け取り，マルウェアを検出した．

4. おわりに

解析されている状態と解析されていない状態で，アンチデバッグ機能を持つ正規の AP とマルウェアに挙動の差異に着目したマルウェア検出システムについて述べた．また，評価により，マルウェアを用いた評価結果について報告した．

参考文献

[1] AVTEST The Independent IT-Security Institute: Statistics, Malware, available from 〈`http://www.av-test.org/en/statistics/malware/`〉 (参照 2013-06-26).
[2] FFRI BLOG : マルウェア解析の実体，入手先 〈`http://www.fourteenforty.jp/blog/2013/02/2013-02-18-1.htm`〉 (参照 2013-06-26).
[3] 株式会社フォティーンフォティ技術研究所 : 2012-08-15 Black Hat 技術報告 後編，入手先 〈`http://www.fourteenforty.jp/blog/2012/08/2012-08-15-1.htm`〉 (参照 2013-06-26).
[4] F-Secure: News from the Lab, available from 〈`http://www.f-secure.com/weblog/archives/00001207.html`〉 (参照 2013-06-26).