SCIS 2025 2025 Symposium on
Cryptography and Information Security
Kokura, Japan, Jan. 28 – 31, 2025
The Institute of Electronics,
Information and Communication Engineers

機械語命令フックによる耐解析機能の特性を利用したマルウェア対策
Endpoint protection utilizing anti-analysis function through machine language instruction hooks

小松 蒼樹 * 毛利 公一 * 瀧本 栄二 * †
Soju Komatsu Koichi Mouri Eiji Takimoto

あらまし マルウェアによる感染経路の多様化により，端末を保護するエンドポイントセキュリティが不可欠である．マルウェアには，デバッガや仮想環境などの解析環境特有の特徴を検知する耐解析機能が実装されている．マルウェアの耐解析機能は，解析環境であると検知した場合，自己動作を停止させる性質を持つ．そこで端末保護手法として，マルウェアにおける耐解析機能の性質に着目した活動抑止手法が提案されている．この手法は，耐解析機能に解析環境を示す偽情報を与え，耐解析機能による実行環境の誤検知を発生させることで，マルウェアの活動を抑止する．耐解析機能は複数手法が存在するが，我々は，既存手法では扱われていない特定の機械語命令の実行結果を用いる耐解析機能に着目した．本論文で提案する手法は，カスタムデバッガを用いてターゲットプロセスをデバッグ対象とし，ターゲットプロセス上の特定の機械語命令にブレークポイントを設置する．これにより，カスタムデバッガが処理をフックし，返り値となるレジスタの値を改ざんすることで耐解析機能を逆用する．さらに，本手法のプロトタイプ実装とその有効性に関する評価結果とオーバヘッドについて報告する．

キーワード マルウェア対策，命令エミュレーション

1 はじめに

企業活動におけるテレワーク導入などによるネットワーク境界があいまいになっている背景や，マルウェアによる感染経路の多様化により，従来のゲートウェイセキュリティだけでなく，端末を保護するエンドポイントセキュリティが不可欠になっている．マルウェアには，デバッガや仮想環境などの解析環境を検知する耐解析機能が実装されている．マルウェアにおける耐解析機能は，解析環境であると検知した場合，解析を妨害する目的で自己動作を停止させる性質を持つ．そこで端末保護手法として，マルウェアにおける耐解析機能の性質に着目した活動抑止手法が文献 [1] で提案されている．この手法は，マルウェアの耐解析機能に解析環境を示す偽情報を与えることで，耐解析機能による実行環境の誤検知を発生させ，マルウェアの活動を抑止する．

耐解析機能には，検知対象と検知方法の 2 つの視点で大きく分類できる．検知対象による分類では，大まかに，仮想環境を検知する Anti-VM と，デバッガを検知する Anti-Debug に分類される．また，検知方法による分類では，Windows API を用いる手法，特定の機械語命令を用いる手法，直接メモリを参照する手法に分類される．文献 [1] では，Windows API を用いた Anti-Debug 手法について焦点を当てており，そのほかの既存研究では，Anti-Debug と Anti-VM における Windows API を用いる手法と直接メモリを参照する手法に焦点を当てている．そのため既存研究では，特定の機械語命令を用いる手法について検討されていない．加えて，文献 [2] では，2010 年から 2019 年の間に実際に確認されたマルウェアを分析しており，耐解析機能について議論されている．具体的には，2010 年から 2019 年の四半期ごとに耐解析機能の各手法における割合を調査しており，特定の機械語命令を用いる手法が耐解析機能を実装しているマルウェアで利用されている割合は，平均値で約 24%と算出されている．

そこで本論文では，特定の機械語命令を用いる耐解析機能手法について着目し，Windows 環境において，機械語命令フックを用いて機械語命令の返り値であるレジスタを改ざんし偽の情報を与える．具体的には，カスタムデバッガを用いてターゲットプロセスをデバッガの制御下に置き，ターゲットプロセス上のフックしたい機械

* 立命館大学, 〒 567-8570 大阪府茨木市岩倉町 2-150, Ritsumeikan University, 2-150 Iwakura-cho, Ibaraki, Osaka 567-8570 Japan.
† 奈良女子大学, 〒 630-8506 奈良県奈良市北魚屋東町，Nara Women's University, Kitauoya-higashimachi, Nara-shi, Nara 630-8506 Japan.

語命令の先頭アドレスにブレークポイントを設置する．ブレークポイント到達後，デバッガにて処理をフックし，機械語命令の返り値となるレジスタの値を改ざんする手法を提案する．提案手法により，特定の機械語命令を用いる耐解析機能に偽の情報を与えることが可能となり，マルウェア活動の抑止が期待できる．

以下，本論文では，2 章で関連研究について述べる．その後，3 章で耐解析機能における機械語命令フックについて述べ，4 章で提案手法とそのプロトタイプ実装について述べる．5 章で性能評価について述べ，6 章にて考察を述べ，7 章にて今後について述べる．

2 関連研究

マルウェアの耐解析機能の性質を逆手に取ることで，防御としてのマルウェア動作の妨害をする研究が行われている．松木ら [1] は，マルウェアにおける耐解析機能では，解析環境を検知すると自己動作を停止させる性質に着目し，マルウェアの活動を抑止する手法を提案している．文献 [1] では，デバッガの存在を確認可能な Windows API である IsDebuggerPresent API に着目し，この API をフックすることにより返り値を改ざんしている．これにより，マルウェアに対してデバッガがアタッチされていると誤認させ，マルウェアの動作を抑止している．

志倉ら [3] は，マルウェアに感染した後の被害抑制を目的として，Windows API を用いる Anti-Debug 手法だけでなく，直接メモリを参照する手法に対しても対応可能なマルウェア動作妨害手法を提案している．具体的には，Windows API のフックなどによるデバッガの存在偽装ではなく，ターゲットプロセスに対して実際に軽量デバッガをアタッチすることで，多くの Anti-Debug を逆用し，マルウェア活動を抑止している．

文献 [1] や文献 [3] では，Anti-Debug にアプローチしているのに対し，J. Zhang ら [4] は，Anti-VM に対して主にアプローチしている．文献 [4] では，耐解析機能を持つマルウェアが，ユーザ端末で悪意のあるコードを実行する前に，耐解析機能を実行する特徴を持つことから，ユーザ端末を仮想環境であるかのように偽装することで，Anti-VM に対して逆用アプローチを行っている．文献 [4] では，軽量の欺瞞機構を DLL インジェクションによる関数フックを用いて構築し，評価している．

関連研究においては，Anti-Debug および Anti-VM の両方に対し，性質を逆手に取ることでマルウェアの動作妨害を行っている．しかしながら，検知手法の観点から見ると，Windows API を用いる手法や，直接メモリを参照する手法に対してはアプローチがなされているが，特定の機械語命令を用いた耐解析機能に対しては，アプローチされていない．そこで特定の機械語命令を用いた耐解析機能への動作妨害アプローチとして，本論文では，特定の機械語命令を用いる耐解析機能を機械語命令フックにより，機械語命令の返り値であるレジスタ値を改ざんし，偽の情報を与える手法を提案する．

3 耐解析機能における機械語命令フック

3.1 特定の機械語命令を用いた耐解析機能

耐解析機能における手法の 1 つである，特定の機械語命令を用いる耐解析機能は，解析環境への検知方法が Anti-VM と Anti-Debug によって大きく異なる．

まず Anti-VM では，仮想環境特有の情報を特定の機械語命令における返り値が格納されるレジスタから値を取得して検知する．たとえば，Anti-VM の代表的な手法には，CPUID 命令を利用した手法が挙げられる．CPUID 命令は，プロセッサ情報やその機能を取得可能な機械語命令であり，EAX レジスタに 0x1 を格納した状態で CPUID 命令を実行すると，ECX レジスタ，EDX レジスタから機能フラグが取得可能である．この，ECX レジスタの 31 ビット目は，hypervisor bit と呼ばれるビットであり，現在の動作環境がハイパーバイザ上であるかを示すフラグになっている．そこで，CPUID 命令を用いた Anti-VM では，ECX レジスタの hypervisor bit を参照することで，仮想環境上で動作しているかを判定する．

次に Anti-Debug では，機械語命令の実行後に例外が発生するかどうかでデバッガの存在を検知する．たとえば，Anti-Debug の代表的な手法には，INT3 命令を利用した手法が挙げられる．INT3 命令は，割り込みを発生させる機械語命令であり，デバッガがアタッチされた状態で INT3 命令を実行すると，割り込みをデバッガが受け取り，デバッガに制御が移る．その後，デバッガが INT3 命令の実行元プロセスへ制御を戻して処理は継続される．しかし，デバッガがアタッチされていない状態で INT3 命令を実行すると，割り込みが発生した後に，この割り込みに対応する割り込みハンドラが存在しないため，例外が発生する．そこで，INT3 命令を利用した Anti-Debug では，あらかじめ例外が発生する前提でプログラムを記述し，INT3 命令実行後，例外が発生しなければデバッガが存在すると判定する．このほかにも，特定の機械語命令を用いた耐解析機能は存在し，表 1 にて，その代表例を示す．

3.2 User Mode Instruction Prevention

User Mode Instruction Prevention(以下，UMIP)[7] は，特定の命令をスーパーバイザモード (リング 0) でのみ実行できるようにする，x86 に導入されているセキュリティ機能である．UMIP の利用有無は，各 OS が選択可能であり，利用を選択する場合，SMSW，STR，SIDT，SGDT，SLDT 命令をユーザモードで実行しようとする

表 1: 特定の機械語命令を用いた耐解析機能の例

機械語命令	実行時レジスタ値	参照レジスタ	検知手法
CPUID	EAX=0x1	ECX	hypervisor bit のセット有無
CPUID	EAX=0x40000000	ECX，EDX	仮想化ベンダ文字列
IN	EAX='VMXh' EDX='VXh'	EBX	EBX='VMXh'(VMware) [5]
SMSW	EAX=0xCCCCCCCC	EAX	未文書化ビット (VMware) [6]
STR,SIDT,SGDT,SLDT			実環境と VM での値の違い [6]
INT3,INT2D			ブレークポイント例外発生の有無

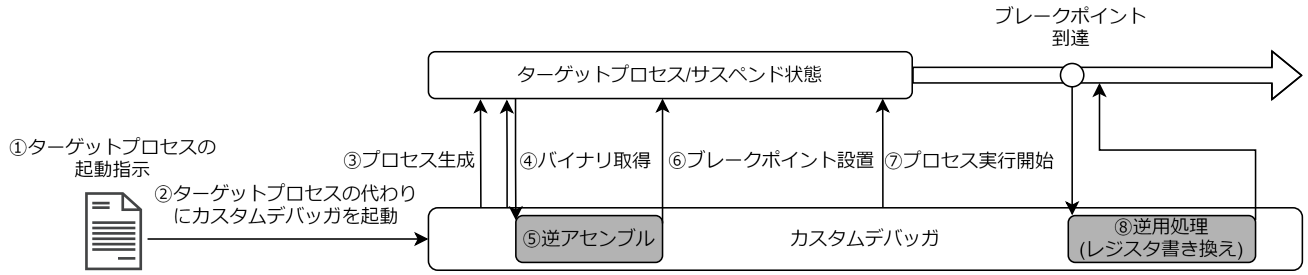

図 1: カスタムデバッガを用いた提案手法

と，一般保護例外 (#GP) が発生し，実行できない．しかしながら，文献 [2] によると，これら命令を用いる Anti-VM の割合は，2010 年から 2019 年の間において，STR 命令で 1.74%，SLDT 命令で 1.29%と実際のマルウェアで一定数実装されていることが示されており，これらの命令に対しても，機械語命令フックによりレジスタ値を改ざんすることで，マルウェア動作抑止の効果が期待できる．

3.3 機械語命令フックに対する課題

表 1 から分かるとおり，複数手法において機械語命令の返り値となるレジスタ値を参照して解析環境を検知している．その為，実行環境を誤認させるためには，機械語命令をフックし，参照されるレジスタ値を書き換える必要がある．しかしながら，機械語命令のフックには大きく 2 つの課題が存在する．

まず 1 つ目の課題として，フックしたい命令にジャンプ命令などを上書きする点である．フックしたい命令に対してジャンプ命令などを上書きすると，上書きした後ろの命令列のアドレスがずれてしまう．このアドレスのずれを解決するために，アドレスの再計算が必要になり，正確なアドレス再計算が求められる．もし，アドレスの再計算に失敗すると，ターゲットプロセスの処理に影響を与えてしまうため，正確に再計算可能な手法が求められる．

次に 2 つ目の課題として，ターゲットプロセスの処理中に実行される命令列を監視する必要がある点である．特定の機械語命令を用いた耐解析機能では，複数手法において機械語命令実行前にレジスタに特定の値を格納してから機械語命令を実行する．その為，フックしたい機械語命令における直前のレジスタ値を確認した後にフックする必要があり，動的でのフックが望ましい．しかし，ターゲットプロセス全体の命令数が増加すると，監視に伴う処理時間の増加により，セキュリティ対策として利用するには無視できない程のオーバヘッドになる恐れがある．そこで本論文では，上記 2 つの課題を解決した機械語命令をフックする手法を提案し，この手法により特定の機械語命令を用いた耐解析機能に偽の情報を与える．

4 提案手法

本論文では，ターゲットプロセス上の機械語命令をフックし，カスタムデバッガが処理を受け取った後に，機械語命令の返り値となるレジスタ値を改ざんする．また本手法により，耐解析機能で利用される機械語命令をフックし，偽の情報を与えることで，マルウェアの活動抑止を行う．

4.1 耐解析機能の逆用における機械語命令フック手法

本手法の全体像を，図 1 に示す．まず，本手法では，ターゲット実行可能ファイルが読み取られると，Windows レジストリである Image File Execution Options[8](以下，IFEO) を用いてターゲットプロセスの代わりにカスタムデバッガプロセスを起動する．IFEO は，特定の実行可能ファイルの挙動を変更するレジストリであり，設定することでさまざまな起動オプションを付与可能である．そこで，この IFEO を用いた提案手法では，ターゲット実行可能ファイルを起動する代わりに，カスタムデバッガを起動する．その後，カスタムデバッガからターゲ

ットプロセスを，CREATE_SUSPENDED，DEBUG_-ONLY_THIS_PROCESS をプロパティとして設定した状態で CreateProcess API により生成する．生成後，カスタムデバッガは，メモリ上に読み込まれたターゲットプロセスのバイナリデータを読み込み，そのバイナリデータを逆アセンブルする．逆アセンブルした結果をもとに，フックしたい機械語命令の先頭アドレスにソフトウェアブレークポイントである INT3 命令を上書きした後，ターゲットプロセスを実行開始する．ターゲットプロセスがカスタムデバッガによって設置された INT3 命令へ到達すると，カスタムデバッガに処理が移行し，直前のレジスタ値を取得する．取得したレジスタ値が，耐解析機能で利用されるレジスタ値と一致する場合，機械語命令の返り値となるレジスタ値を解析環境を示す値に書き換える．その後，プログラムカウンタである EIP レジスタを，次の命令の先頭アドレスを指すように書き換えた後，ターゲットプロセスを実行再開する．

4.2 提案手法における各処理

4.2.1 ターゲットプロセスへのアタッチ処理

本手法では，ターゲットプロセスが起動するまでにカスタムデバッガの制御下に置く必要がある．なぜなら，耐解析機能は自身の解析を防ぐ目的で実装されている理由から，ターゲットプロセスの比較的早い段階で耐解析機能が実行される恐れがあるためである．そこで，IFEO を用いることで，カスタムデバッガからターゲットプロセスを生成し，カスタムデバッガの制御下に置くこととした．また，IFEO では，引数を設定でき，IFEO によって起動されたプロセスがその引数を取得可能である．そこで，引数に自身の実行可能ファイル名を設定することで，複数の実行可能ファイルに IFEO が設定されたとしても，カスタムデバッガはどの実行可能ファイルの IFEO によって起動されたかを識別できるようにし，適切にターゲットプロセスを子プロセスとして生成できるようにした．

4.2.2 ブレークポイント設置処理

カスタムデバッガによってサスペンド状態でターゲットプロセスが生成された後，カスタムデバッガがターゲットプロセスのメモリデータを取得し，逆アセンブルする．逆アセンブル結果より，耐解析機能で用いられる機械語命令にソフトウェアブレークポイントである INT3 命令 (0xCC) を，機械語命令の先頭アドレスに上書きする．INT3 命令の上書き後，ターゲットプロセスのサスペンド状態を解除し，実行を開始する．

4.2.3 逆用処理

ターゲットプロセス上に設置したブレークポイントに到達すると，カスタムデバッガに制御が移行される．制御移行後，カスタムデバッガでは，その機械語命令の実行目的が耐解析機能であるかを確認するために，直前のレジスタ値を確認する．耐解析機能で利用されるレジスタ値であった場合，機械語命令の返り値であるレジスタに偽の情報を書き込む．その後，プログラムカウンタをフックした次の機械語命令を指すように変更した後，ターゲットプロセスに制御を戻し，実行を再開させる．例として，図 2 に CPUID 命令におけるフック処理を示す．

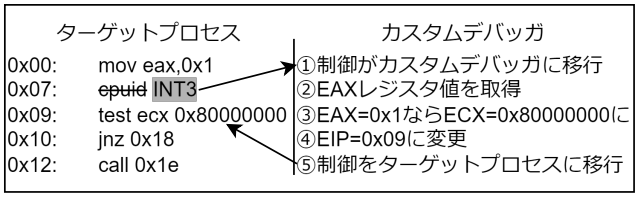

図 2: CPUID 命令 (EAX=0x1) へのフック処理

まず，CPUID 命令上に設置された INT3 命令により，ターゲットプロセスからカスタムデバッガに制御が移行される．制御移行後にカスタムデバッガは，現在の EAX レジスタが 0x1 であるかを確認する．これは，CPUID 命令の hypervisor bit を確認する耐解析機能は，EAX レジスタ値を 0x1 の状態で実行するためである．EAX レジスタ値が 0x1 であれば，耐解析機能の恐れがあると判断し，hypervisor bit がセットされるようにレジスタ値を書き換える．その後，カスタムデバッガはプログラムカウンタを次の機械語命令の先頭アドレスに変更する．この操作により，CPUID 命令はスキップされる．その後，ターゲットプロセスに制御を戻し，実行再開する．

4.3 実装

提案手法について，プロトタイプを実装した．プロトタイプ実装では，CPUID 命令の hypervisor bit を用いた Anti-VM に偽の情報を与える．今回のプロトタイプ実装では，逆アセンブラとして，C 言語コードに直接組込み可能なライブラリである udis86[9] を利用し，x64 向けにプロトタイプを実装した．その為，x64 では，汎用レジスタを x86 の 32 ビットから 64 ビットに拡張されており，EAX は RAX，ECX は RCX レジスタへと対応付いている．そこで，本章以降では，x64 のレジスタ名を用いて述べていく．

5 評価

本章では，提案手法適用時の耐解析機能に対する有効性の検証およびシステムオーバヘッド計測結果について述べる．今回使用した，評価環境を表 2 に示す．

表 2: 評価環境

要素	仕様
CPU	Intel Core i7-12700K 3.61GHz
メモリ	32.0GB
OS	Windows 10 Education 22H2

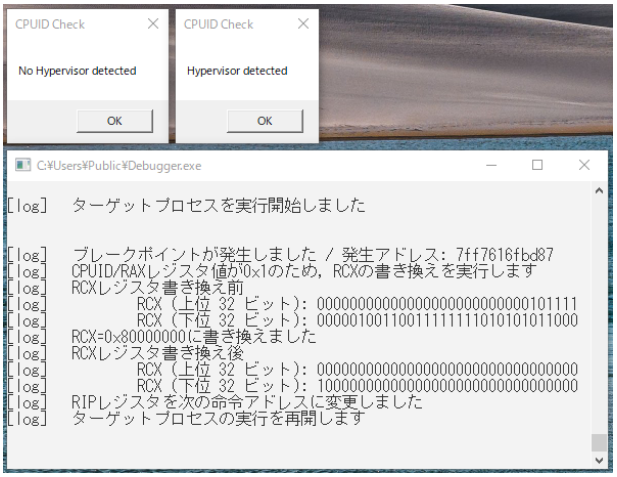

図 3: 提案手法適用時と未適用時の実行結果

5.1 耐解析機能への有効性

まず，CPUID 命令の hypervisor bit による Anti-VM に対する提案手法の有効性を確認する．CPUID 命令による Anti-VM を実装した検証プログラムを作成し，プロトタイプ実装の動作検証を行った．検証プログラムは，C/C++の標準ライブラリ関数である__cpuid 関数を用いて，RAX レジスタ値が 0x1 の状態で CPUID 命令を実行し，その結果を Windows API である MessageBoxA API で出力する．図 3 にその結果を示す．図 3 の上段には，検証プログラムの出力結果が示されており，上段左ウィンドウが，通常起動時の出力結果であり，右ウィンドウが提案手法適用時における検証プログラムの出力結果である．下段には，カスタムデバッガのログ出力を示している．まず，上段にある検証プログラムの出力結果から，CPUID 命令を用いた Anti-VM に対して，提案手法により解析環境への誤認が成功していることが確認できる．次に，カスタムデバッガのログ出力から，ブレークポイントの発生により，デバッガへ制御が移り，RAX レジスタ値が 0x01 であることを確認し，RCX レジスタの書き換え後に，ターゲットプロセスへ制御を戻していることが確認できる．

次に，Anti-Debug に対する有効性を確認する．今回は，多数の Anti-Debug を実装したオープンソースツールである al-khaser[10] を用いて，有効性を検証した．al-khaser に実装されている，UnhandledExcepFilterTest については，Visual Studio 付属のデバッガをアタッチした状態で実行しても，正しく処理が終了しなかったため，今回はコメントアウトした．また，al-khaser を実行しデバッガを検知した場合は [BAD]，未検知の場合は [GOOD] と表示されるが，各検知結果の表示が 15 文字分，左側で表示されるように変更した．カスタムデバッガをアタッチした状態で起動した al-khaser の出力結果を図 4 に示す．図 4 から，複数の手法でデバッガの存在を検出されていることが分かる．この実行結果によると，OutoutDebugString 項目では，デバッガの存在が検出されていないことが確認できるが，この手法は Windows XP/2000 でのみ動作するため，デバッガが検出されなかったと推測できる．デバッガが検出されていない Anti-Debug 手法に関しては，それぞれの手法を 1 つずつ対応することで，検出される可能性があり，今後議論していく必要がある．デバッガが検出された Anti-Debug 手法として，注目すべきは，If Parent Process is explorer.exe 項目である．これは，親プロセスによりデバッガの存在を確認する Anti-Debug 手法である．通常，ダブルクリックなどによりプロセスが生成されると，生成されるプロセスの親プロセスは explorer.exe となる．しかしながら，デバッガから生成されたプロセスの親プロセスはデバッガプロセスとなるため，本 Anti-Debug 手法で検出されている．デバッガからプロセスを生成するのではなく，すでに生成されたプロセスに対して，デバッガをアタッチした場合の親プロセスは，デバッガにはならないため，IFEO を用いることで，親プロセスを確認する Anti-Debug に対しても有効であると言える．

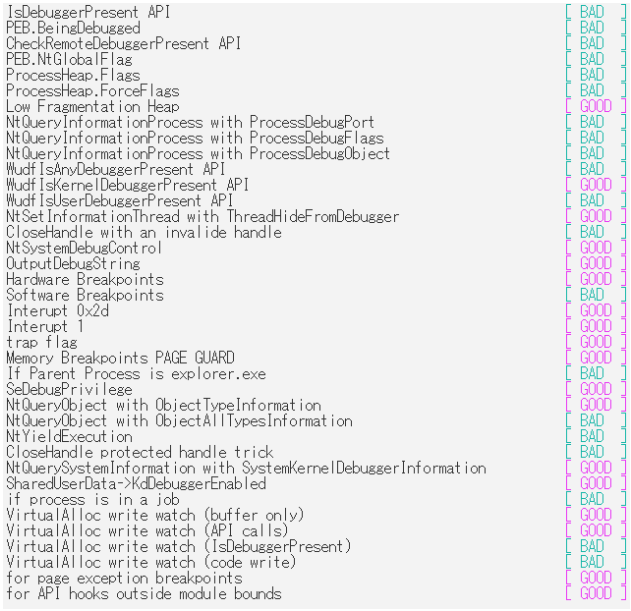

図 4: 提案手法適用時における al-khaser の実行結果

表 3: ターゲットプロセス実行開始前の平均処理時間

区間	未適用時 [ms]	適用時 [ms]
1a	5.101	110.646
1b		767.38258

5.2 オーバヘッド評価

次に，提案手法を適用した場合のシステムオーバヘッドを計測した．結果に偏りが反映されないように，各 10000 回計測し，その平均値を計測値とした．提案手法におけるオーバヘッドを処理ごとに分類すると大きく以下のように分類可能である．

(1) ターゲットプロセス実行開始前

(1.a) ターゲットとなる実行可能ファイルの読み取り後，カスタムデバッガによってターゲットプロセスが生成されるまでのオーバヘッド

(1.b) 逆アセンブルに要する時間に加え，特定の機械語命令を探索し，ブレークポイントを設置するまでのオーバヘッド

(2) ターゲットプロセス実行開始後

(2.a) デバッグイベント発生時のオーバヘッド

(2.b) 逆用に伴う機械語命令フックのオーバヘッド

まず初めに，(1) におけるオーバヘッドを計測した．(1a) では，Process Monitor[11] にて取得したログを用いて計測した．Process Monitor を用いて，ターゲットとなる実行可能ファイルにおける CreateFile ログと，Process Create ログに記録された Time 項目の差分により処理時間を算出した．この方法を用いて，通常時のターゲットプロセス生成にかかる時間と，提案手法適用時のターゲットプロセス生成にかかる時間の両方を計測した．(1b) では，カスタムデバッガのコード内に，C++の標準ライブラリである chrono を用いて計測した．また，今回は 50000 バイトに対し，逆アセンブリおよび CPUID 命令の探索，ブレークポイントの設置を行った．計測プログラムの命令列には，3 つの CPUID 命令が存在していたため，カスタムデバッガは 3 つのブレークポイントの設置を行っている．(1a) および (1b) の計測結果を表 3 に示した．

表 3 の (1a) により，IFEO を用いてカスタムデバッガからターゲットプロセスを生成した場合，通常時の生成までの時間と比べ，約 21.7 倍の処理時間を要することが分かった．次に，表 3 の (1b) によると，50000 バイトの逆アセンブリおよびブレークポイントの設置には，かなりの時間を要することが確認できる．(1b) に発生する処理時間が増加するにつれて，ターゲットプロセス実行開始までの時間が長くなる．その為，今回は，50000 バイトに対する処理時間であるが，さらなるバイト数に対して処理を行う場合などを考慮すると，改善策を考える必要がある．

表 4: 各デバッグイベント発生時の平均処理時間

デバッグイベント	未適用時 [μ s]	適用時 [μ s]
Load_Dll	4.3053	4.3083
Unload_Dll	0.1201	0.1193
Create_Thread	19.7072	25.0617
Exit_Thread	5.6845	10.1869
Output_Debug_String	20.7371	75.5958
Exception(Breakpoint)		980.8297

表 5: CPUID(EAX=0x1) 実行時の平均処理時間

	実行時間 [μ s]
適用時	517.4322

次に，(2) ターゲットプロセス実行開始後のオーバヘッドについて計測した．まず，ターゲットプロセスの実行開始後に，(2a) デバッガがターゲットプロセスの処理を捕捉する各デバッグイベント発生時の処理時間を，提案手法未適用時と適用時のそれぞれに対し，C++の標準ライブラリである chrono を用いて計測した．また今回，EXCEPTION_DEBUG_EVENT は，例外発生時に発生するデバッグイベントであるが，今回は，Microsoft Visual Studio のコンパイラである MSVC で提供される__debugbreak 関数を用いて，ブレークポイントを発生させ，計測している．各デバッグイベントの計測結果を表 4 に示す．表 4 から，各デバッグイベントにおける処理時間が，提案手法未適用時に比べ，適用時の方が，ほぼ同等かそれ以上の処理時間を要していることが分かる．今回計測したデバッグイベントのほかに，CREATE_PROCESS_DEBUG_EVENT や EXIT_PROCESS_DEBUG_EVENT，RIP_DEBUG_EVENT が存在する．CREATE_PROCESS と EXIT_PROCESS は，複数プロセス間での計測では正確な計測が困難である点から計測対象外とし，また，RIP_DEBUG_EVENT については，発生可能性が低いと考えられる為，計測対象外とした．

次に，(2b) における処理時間として，今回は，CPUID 命令における hypervisor bit を用いた耐解析機能の実行時における提案手法適用時の処理時間を計測した．

提案手法適用時における CPUID 命令は，CPUID 命令上にブレークポイントが設置されており，カスタムデバッガに制御が移行した後，RCX レジスタ値の書き換えなどが行われてから処理がターゲットプロセスにに戻ってくるまでの処理時間を計測した．cpuid 命令を実行する際には，__cpuid 関数を用いた．表 5 にその計測結果を示す．表 4 における Exception(Breakpoint) の項目と

比較すると，表 4 の Breakpoint 項目の方が処理時間を要していることが分かる．

これは，今回の計測において，それぞれ__debugbreak 関数および__cpuid 関数の処理時間を計測している．そのため，単純な INT3 命令や CPUID 命令の機械語命令における実行時間を計測しているものではない為と考える．

6 考察

耐解析機能に対する有効性評価により，機械語命令フックを用いたレジスタ書き換えによる Anti-VN への有効性と，デバッガを実際にアタッチすることによる Anti-Debug への有効性を示した．これらにより，実際のマルウェアの耐解析機能に対しても同様に，提案手法により解析環境への誤認が可能である可能性が高いと言える．また，それによりマルウェアの動作抑止が可能と考える．

提案手法のオーバヘッド評価により，提案手法適用時におけるターゲットプロセス実行開始までのオーバヘッドを表 3 に，ターゲットプロセス開始後のオーバヘッドを表 4 や表 5 に示した．オーバヘッド計測の結果から，特に (1b) と (2b) の処理時間が顕著であることが明らかになったため，それらオーバヘッドについて考察する．

まず (1b) について考察する．今回は，ターゲットプロセスの main 関数のアドレスから 50000 バイトを取得し，CPUID 命令を探索している．具体的には，PE ファイルフォーマットにおける，Standard COFF Fields の AddressOfEntryPoint が指すオフセット形式 (RVA) のアドレスに，実行可能ファイルがロードされている先頭アドレスを示す，プロセス環境ブロック (PEB) の ImageBaseAddress を加算することで main 関数のアドレスを取得している．今回は，50000 バイトを取得しているが，実際のマルウェアではさらなるバイト数の取得が必要な可能性がある．しかしながら，耐解析機能には，解析を防ぐ目的で実装されているため，マルウェア処理の比較的早い段階で実行される事が想定される．そこで，命令の探索範囲を特定の箇所のみに絞り込むことで，逆アセンブリするバイト数が減少し，処理時間を削減可能である．またそもそも，実行可能ファイルがメモリ上にロードされた後に命令を探索しているが，ロードされる前にあらかじめ命令の探索を済ませておき，ブレークポイントを設置したい命令のオフセットアドレスを保持しておくことで，ターゲットの実行可能ファイルがロードされた段階での探索を回避する手法も考えられる．この場合は，事前に取得してあるブレークポイントを設置したい命令のオフセットの情報をもとに，ブレークポイントを設置するだけでよい．

次に，(2b) について考察する．ブレークポイントに到達すると，ターゲットプロセスの処理は中断され，デバッガに制御が移行する．ターゲットプロセスの処理は，デバッガが制御をターゲットプロセスに受け渡さないと再開されないため，デバッガでの処理内容が多くなればなるほど，ブレークポイント発生時の処理時間は長くなる．しかし，ターゲットプロセス上のデバッガに制御が移行しない処理については，処理時間のオーバヘッドはデバッガがアタッチされていない場合と比較して，大きくは変わらない．また，特定の機械語命令を用いた耐解析機能で利用される機械語命令は，CPUID 命令を除いて，ユーザレベルでは頻繁に利用されないものが多い．その為，ブレークポイントが多発し，明確な影響が出るほどのオーバヘッド増加にはつながらないと考える．

それら考察に加え，計測したオーバヘッド結果から，提案手法を実運用で用いる場合に，考えられる課題がいくつか挙げられる．

まず 1 つ目として，複数プロセスを生成するプロセスに対する課題が挙げられる．(1b) におけるオーバヘッドが多いため，複数のプロセスで同時に (1b) に関する処理を実行すると計算資源が一斉に必要とされ，コンピュータの動作が遅くなる可能性がある．その為，(1b) の処理を一斉に実行しないような工夫が今後求められる．

次に 2 つ目として，デバッグイベントが多く発生するプロセスに対する課題が挙げられる．デバッグイベントが発生すると，表 4 に示したオーバヘッドがかかる．その為，多くのデバッグイベントが発生するようなプロセスにデバッガをアタッチすると，ターゲットプロセスの実行時間に影響を与える可能性がある．デバッグイベントが多く発生する可能性があり，また高度な処理を必要とするプロセス，ゲームソフトや動画編集ソフト，仮想化ソフトウェアなどにはアタッチしないような工夫が求められる．

耐解析機能は，多くのマルウェアで利用されていることが文献 [2] で示されており，今後も実装されると予想される．しかしながら，耐解析機能を実装するのは，マルウェアだけではない．リバースエンジニアリングなどからソフトウェアを守る目的で，商用ソフトウェアなどにも実装されていることが知られている．その為，本手法をすべてのプログラムに適用すると，良性ソフトウェアが正しく起動しない可能性が考えられる．そこで，今回は特定の実行可能ファイルにのみにカスタムデバッガをアタッチするように IFEO を用いてプロトタイプ実装を行ったが，今後，デバッガをアタッチする対象の選定方法については，検討する必要がある．また，プロトタイプ実装で用いた IFEO は，実際のマルウェアが IFEO レジストリの情報を取得することで，提案手法の存在が検知される恐れがある．しかしながら，提案手法は仮想環境においても適用可能であり，提案手法の存在により解析環境でない保証はない．その為，マルウェアが IFEO により提案手法を検知すると本来の挙動の通り，自身の

動作を停止させると考えられ，端末保護手法としての目的は達成される．しかし，マルウェアによって IFEO レジストリに書き込みされた場合には，マルウェアが複数回起動する場合に，適切にデバッガが起動しない可能性が考えられる為，対策を検討する必要がある．

7 おわりに

本論文では，マルウェアの耐解析機能に偽情報を与えて，実行環境の誤検知を発生させる手法として，カスタムデバッガを用いる手法を提案し，本手法のプロトタイプを実装した．提案手法は，ターゲットプロセスに対してデバッガの制御下におき，耐解析機能で利用される特定の機械語命令にブレークポイントを設置する．ブレークポイントに到達後，機械語命令をフックし，機械語命令の返り値となるレジスタ値を改ざんすることで耐解析機能の実行環境検知を誤認させる．プロトタイプに対して，Anti-VM や Anti-Debug が実装されたプログラムをアタッチすることで有効性を確認し，オーバヘッドの計測も行った．

今後の課題として，提案手法適用時のオーバヘッドの削減やターゲットプロセスの絞り込み方法の検討などが挙げられ，今後も議論する必要がある．

謝辞

本研究は JSPS 科研費 JP22K12037 の助成を受けたものです．

参考文献

[1] 松木隆宏，新井悠，寺田真敏，土居範久，マルウェアの耐解析機能を逆用した活動抑止手法の提案，情報処理学会論文誌，Vol 50，No 9，pp.2118–2126 (2009).

[2] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, Stefano Zanero, A Systematical and longitudinal study of evasive behaviors in windows malware, Computers & Security,Volume 113,pp.102550 (2022).

[3] 志倉大貴，西村俊和，瀧本栄二，軽量デバッガを用いたマルウェア動作妨害機構の実装と評価，コンピュータセキュリティシンポジウム 2022 論文集，pp.1156-1162 (2022).

[4] J. Zhang et al., Scarecrow: Deactivating Evasive Malware via Its Own Evasive Logic, 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 76-87,(2020).

[5] ”VMware Backdoor I/O Port - VM Back”，https://web.archive.org/web/20230325103442/https://sites.google.com/site/chitchatvmback/backdoor，(参照 2024-12-11)

[6] Michael Sikorski, Andrew Honig, ”Practical Malware Analysis”, pp.369-380, No Starch Press, (2012).

[7] ”User Mode Instruction Prevention”,https://edc.intel.com/content/www/jp/ja/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/004/005/user-mode-instruction-prevention/，(参照 2024-12-11)

[8] ”Image File Execution Options”，https://learn.microsoft.com/ja-jp/previous-versions/windows/desktop/xperf/image-file-execution-options，(参照 2024-12-13)

[9] ”vmt/udis86: Disassembler Library for x86 and x86-64”,https://github.com/vmt/udis86，(参照 2024-12-11)

[10] ”ayoubfaouzi/al-khaser”，https://github.com/ayoubfaouzi/al-khaser，(参照 2024-12-11)

[11] ”プロセス モニター-Sysinternals”，https://learn.microsoft.com/ja-jp/sysinternals/downloads/procmon，(参照 2024-12-12)