ステガノグラフィー技術を用いたマルウェアの解析耐性強化と検討

長嶺 優大[1]　矢内 直人[1]　岡村 真吾[2]　藤原 融[1]

概要：解析耐性をもつマルウェアの研究について，発火条件を限定することで挙動を隠蔽する Trigger-Based Malware(Dennis ら，DIMVA'14) がある．しかし，この研究における静的解析対策は解除が容易であり，また動的解析についても，実行箇所が特定されると容易になる．そこで本研究では，Trigger-Based Malware にステガノグラフィー技術を用いることで静的・動的解析耐性を強化する手法とその対策について検討する．強化手法としては解析時に検出されないように悪性コードを埋め込み，かつ，悪性コード部を分割し実行を段階的に行うことである．

キーワード：Format-Trnasformation Encryption, Trigger-Based Malware, Evasive Malware, 難読化

A study on steganography for malware

Nagamine Yudai[1]　Yanai Naoto[1]　Okamura Shingo[2]　Fujiwara Toru[1]

Abstract: There is Trigger-Based Malware (Dennis et al., DIMVA '14) that limits firing conditions for research on malware having analysis tolerance. However, countermeasures against static analysis are easily avoidable, and dynamic analysis is also facilitated when the execution location is specified. Therefore, we examine methods and countermeasures to strengthen static and dynamic analysis tolerance by using steganography technology for Trigger-Based Malware. As a strengthening method, malicious code is embedded so as not to be detected at the time of analysis, and malicious code part is divided and execution is performed stepwise.

Keywords: Format-Trnasformation Encryption, Trigger-Based Malware, Evasive Malware, Obfuscation

1. はじめに

1.1 序論

近年 IT 技術が浸透し，クレジットカード情報や個人情報がデータベース上に格納されるようになった結果，これらの情報を狙ったマルウェアによる被害が急増している．既存のマルウェアの対策を行うためには，解析することでその挙動を明らかにする必要が望ましい．解析方法としては大きく分けて二つある．一つは検体自体を資源の隔離された環境上でマルウェアを実行し，挙動を見ることで解析する動的解析である．この方式の利点は直接実行することにより，実際に実行されているコードを中心として解析が容易である．欠点としては，マルウェアを実行する必要があるため，環境の用意や解析環境の外へ被害が広がる可能性があることである．

もう一つはマルウェアを実行することなく，バイナリのコードをディスアセンブルして解析する静的解析である．利点は詳細な解析ができること，欠点は実行されるコード部の特定をする必要があるため，手間がかかることである．

攻撃に使われるマルウェアの総数はここ数年で急増傾向にあり，解析者による手動の解析が間に合っていない現状があるため，積極的な自動化が試行錯誤されている．しか

[1] 大阪大学
Osaka University, 1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan
[2] 奈良工業高等専門学校
National Institute of Technology, Nara College, 22 Yata, Yamatokoriyama, Nara, 639-1080, Japan

し，マルウェアの中には各種の解析や検知を妨害・回避する解析耐性を持ったマルウェア (Evasive Malware[1], [2]) が出現している．これらはコード内に不要な命令を複数持っていることや，バイナリのシンボル名として本来の用途とは全く別の名前をつけているなど様々な改変が施されている．攻撃者はこのような手法をマルウェアに施すことにより，解析者が一つのマルウェアを解析する時間を肥大化し，場合によっては解析を諦めざるを得ない状況も起こり得る．新しい解析妨害手法を持つマルウェアが出た場合はその目的や仕組みを詳細に解析し，それを踏まえた対処策を考える必要があるが，多様化する各手法全てに対処することは難しい．そこで，考え得る手法を研究者自身が安全性評価用ツールとして実装することで，根本的な対処策を考える研究が必要となっている．本稿では，その前段階として解析に耐性を持つソフトウェアについて，ステガノグラフィー技術に着目して検証する．ステガノグラフィー技術は情報を隠蔽したまま埋め込む技術である．暗号化では特定の情報を権限のない人に読まれないようにすることが目的であるが，ステガノグラフィーの目的は何らかの情報が埋め込まれている事実自体を隠蔽することにある．埋め込まれていることに気づかせないことで，第三者に疑われることなく情報の伝達を行うこともできる [3], [4]．実世界での使われ方としては，画像データ内に他の画像データを埋め込む手法が多いが，本稿では，「情報が埋め込まれている事自体を隠蔽する」ことを目的とした，広義のステガノグラフィー技術を取り扱う．

1.2 貢献

本稿での貢献は，既存の静的・動的解析手法への耐性を持つ手法として，特殊な入力があったときのみ実行される trigger-dependent bug [5] に着目し，その解析耐性をステガノグラフィー技術を用いて強化したことである．具体的には，Dennis らの trigger-dependent bug[5] の弱点であった静的解析耐性を Format-Transformation Encryption(FTE)[3] を用いることで強化したこと，及び動的解析耐性を複数回暗号化することで強化したことである．また，この静的解析耐性と動的解析耐性を両立させるために，gadget(悪性コード) の記述方法も新たに提案している．

コード偽装： Dennis らの研究 [5] における静的解析対策は，unaligned x86 code[6] を挿入していた．これは本来のバイトコードの先頭に特定のバイトを挿入することにより，ディスアセンブラのオペコード読み込みを妨害し，異なるアセンブラにディスアセンブルさせる手法である．この手法は仕組みが単純であるため比較的容易に利用することができることから，現実のマルウェアの中でも使われている．一方，解析者はディスアセンブル開始バイトを数バイトずらすだけでこの手法を回避することができる．簡単にディスアセンブルできないようにするために，該当コード部を FTE を用いて暗号化した．FTE は出力される暗号文のフォーマットを指定できる．出力フォーマットとして良性関数のバイトコードに擬態することにより，存在そのものを隠蔽した．

多段階実行： Dennis らの研究 [5] では，入力のハッシュ値を用いることで悪性コード実行のトリガとなる入力と悪性コード自体の場所を隠蔽している．この欠点としては，利用しているハッシュ関数が単純であるために逆算が簡単であること及びブルートフォースを使うことで解析できた．本稿では悪性コード部を複数の段階に分ける，かつ暗号化することにより悪性コード部を特定できたとしても鍵がなければ再現することができず，次の段階のコードを実行・解析をできなくしている．

gadget(悪性コード) の記述方法： 本稿では，提案ソフトウェアにトリガとなる入力があった場合に実行されるコード (gadget) を関数とシェルコードの形で挿入した．複数回暗号化したシェルコードは共有メモリに格納する．各実行段階では，入力を用いて暗号文を復号し，mmap で実行権限が付与された領域にコピー，実行する．gadget はソースコードレベルでの挿入を行っているため，OS の防御機構である Address Space Layout Randomization(ASLR) や NX bit による実行の支障はきたさないことが期待できる．Dennis らの研究 [5] では，ASLR は無効の状態で実行することが前提となっていたが，本稿では共有メモリを用いることでこの問題も解決している．

本稿の構成は以下のようになっている．2 章では脅威モデル及び要素技術について示す．3 章では提案する手法の設計方針とその具体的な構成について解説する．4 章に提案手法を実装したソフトウェアを用いて行った実験とその結果を記述し，それらについて考察する．5 章では本稿に関連する研究を挙げる．最後に，6 章に本稿のまとめと今後の展望を述べる．

2. 要件定義

2.1 前提条件

本稿で対象とする内容は提案ソフトウェアに感染した際の挙動や解析者による解析耐性についてであり，実際の感染のさせ方については議論しない．つまり，被害者はあらかじめ本ソフトウェアに感染しているものとする．

想定するシナリオとしては，2017 年 8 月に起こった Web ブラウザ Google Chrome のブラウザ拡張 Web Developer for Chrome の作者アカウントが流出 [7] し，本家 HP で配布しているソースコードが改変されたものが配布された場合を想定する．

感染対象の PC は外部へ Web サービスを提供する目的で構築された，Linux OS が動作しているサーバ用 PC であり，80 番ポートでサーバプログラムが動くものとする．加えて，攻撃者から悪性コード部を実行する特殊な入力を受け取る

ため，ネットワークに接続されている必要がある．攻撃者はサーバ PC へ接続し，自由に HTTP リクエストを送れるものとする．通常のネットワークでは，サーバ PC や接続を行う攻撃者 PC 上ではネットワークに関する監視をするツール Intrusion Detection System/Intrusion Prevention System(IDS/IPS) や Deep Packet Inspection(DPI) が動いていることが想定される．本稿では提案ソフトウェアがサーバプログラムであるため，悪性コード実行の引き金 (トリガ) として，特殊な HTTP ヘッダー行を含んだ HTTP リクエストを受け取るものとする．また，提案ソフトウェアが動作するために必要なライブラリは予め全てインストールされているものとする．

2.2 良性コードと悪性コード

本稿における良性コードとは，ソフトウェアの動作に必要であるもの，もしくはその開発目的を体現する機能を指す．一方，悪性コードは利用者に何らかの不利益を与えるために組み込まれ，該当部分が実行されることによってソフトウェアの利用者の意図しない挙動をする機能を指す．

2.3 脅威モデル

本稿では，企業などで運用されている外部に Web サービスを提供するサーバ PC 上に置いて，外部の攻撃者からの接続によりサーバ PC 管理者の意図しない挙動をするソフトウェアを作成する．本稿の趣旨はこのソフトウェアが悪性のものであると疑われた際に，解析に対する耐性の検証と評価である．このためには改変されたサーバプログラムを動作させている Web サーバ，そして Web サーバに接続できる攻撃者が操作する PC の二つが必要である．通常では入力されることのない特殊な入力がない限り，改変されたサーバプログラムは改変前と全く同じ挙動をするため，早期の発見を遅らせる．以上を行うことにより，端末に感染し，長時間潜伏しながら情報を収集，送信しつつも検知を困難にすることが見込まれる．

3. 要素技術

本章では提案するソフトウェアの設計にあたり必要となる要素技術について紹介する．

3.1 trigger-dependent bugs

Trigger dependent bug(trigger bugs)[5] は，特定の入力があったときのみ実行される特殊なコードフロー (control transfer) である．目的は，解析者から実行の引き金 (トリガ) となる入力の隠蔽と入力があった場合に実行される部位の相互的な隠蔽である．つまり，トリガとなる入力と実行されるコード部のどちらかが特定されたとしてももう片方の特定は困難なままである．

Dennis らの研究 [5] では，トリガとなる入力は特定の時刻や外部からの入力などとしている．これらの入力があった場合，入力されたデータをハッシュ関数の入力として用いてハッシュ値を算出し，ハッシュ値を悪性コード部のアドレスとして利用している．この場合，手元にソフトウェアのバイナリのみ手元にある解析者による解析では，悪性コード実行の分岐処理やハッシュ関数の方式自体を見つけることができるが，トリガ入力がない限り悪性コード部への分岐処理の再現や悪性コード部の確定は困難となる．
図 1 に trigger-dependent bug の直観を示す．

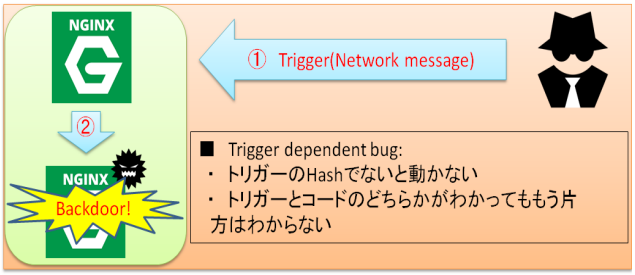

図 1 trigger-depedent bug

3.2 フォーマット変形暗号 (FTE)

FTE は入力として平文 (M) とフォーマット (F)，そして鍵 (K) を与え，出力としてフォーマットに沿った暗号文を得る暗号方式である．鍵はメッセージ自体を始めに AES で暗号化するために与えられる．フォーマットの記述には正規表現を利用でき，ユーザ自身が柔軟なフォーマットを作成できる．フォーマットにプロトコルを模したものを与えれば，暗号化されたデータは HTTP(S) や SSH，FTP など任意のプロトコルに擬態することができる．よって，DPI の検知ルールの記述に正規表現を用いた DPI(regular expressions based DPI, regexDPI) の検知ルールと同じ記述が可能となる．この特性を利用すると，検知ルールの設定ファイルがわかれば，そのルールで暗号化することで，DPI の検知を容易に回避することができる．Kevin らの研究 [3] によると，実際に FTE-powered Tor Browser Bundle を用いて，中国の様々なサービスを禁止する DPI システムの The Great Firewall of China(CFG) を回避し，中国では禁止されているサービスの Youtube，Facebook へのアクセスが成功している．本稿ではこの暗号方式を用いて，本来の x86 アーキテクチャのコードを別のコードに暗号化することで，悪性コード部の存在の隠蔽を行う．

3.3 unaligned x86 code

unaligned x86 code[6] は Intel x86 アーキテクチャのディスアセンブラによるディスアセンブルを妨害する手法のことである．通常ディスアセンブラはバイトコードの先頭を読み込むことでオペコードを識別し，オペランドを決定，ディスアセンブルする．しかし，このバイトコードの先頭に本来のコードとは関係のないバイトを挿入することで

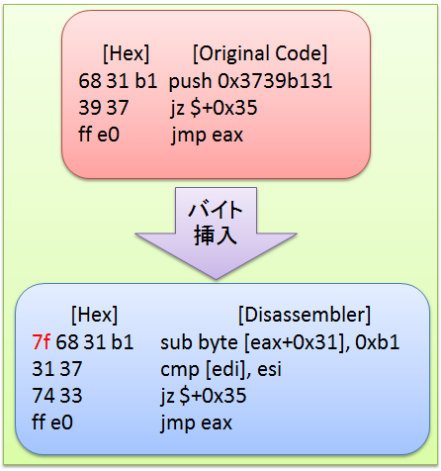

図 2 unaligned x86 code example

ディスアセンブラを誤認させたディスアセンブル結果を得ることができる．この手法が施された場合の対策は，挿入されたバイトコードを避け，本来のコード部からディスアセンブルし直すことである．図 2 に unaligned x86 code の例を示す．この図では，本来のコードの先頭部分に 16 進数の値 0x7f を挿入することで異なるコードにディスアセンブルされている．実行時には挿入したバイトを無視する形で実行する．

4. 提案手法

本節では提案手法について，その設計方針について述べる．本稿では，Dennis らの研究 [5] のもつ解析解析耐性をステガノグラフィー技術を用いて強化した場合の脅威の評価を目的とする．このために Dennis らの提案に複数の技術を施すことで耐性を向上させる．

提案手法について，Dennis らのソースコード [5] を基に実装した．これはサーバプログラム nginx のバージョン 1.5.8 をベースに trigger-bug を挿入したソースコードを使用したものである．特殊な HTTP ヘッダーを持った HTTP リクエストを受信したときのみに実行されるコードを持つ．このプログラムに本稿で提案する静的・動的解析妨害手法を適用した．動作環境は Linux の 32 ビット OS となっている．

コード偽装： 静的解析では，従来は悪性コードの先頭に特定のバイトを挿入することでコードを難読化していた．これに加えて，悪性コード部を暗号化する．静的解析妨害として挿入する悪性コードの暗号化には FTE を用いる．FTE のフォーマットとしては良性の関数を含んだバイト列を正規表現で与え，出力は良性関数や通常のデータのように見える形の暗号文を得る．攻撃者は細工した HTTP リクエスト内に暗号化の際に使用したフォーマットを与え，受け取ったサーバプログラム側ではこれを用いて暗号文を復号し，実行する．手元にバイナリのみある解析者は静的解析ではコード内に悪性とわかるコードやデータは見つけられず，違和感を感じた場合でも確信に到ることへの妨害が期待できる．暗号化の手法には，Format-Transformation Encryption(FTE)[3] を用いる．FTE では出力となる暗号文のフォーマットを任意に与えて操作することができるため，フォーマットとして良性の関数を含むバイト列を与える．これにより，悪性コード部自体を別の関数に見えるように暗号化する．解析者からは通常のコードと見えるため，悪性コード部の存在を隠蔽し，該当部分の特定を妨害する機能が期待できる．本稿では FTE 暗号化を行うためのライブラリとして，libfte[8] を利用した．

同じ領域の多段階実行： 動的解析では，悪性コード部を複数に分割することで解析耐性を改善する．具体的には，悪性コード部の FTE による暗号化を複数回繰り返すことで，前の平文が次の暗号文となる仕組みを組み込む．従来は悪性コードが特定できればデバッガで強制的にジャンプさせることで，挙動の解析ができる可能性があった．これを解決するためにソフトウェア内に複数のトリガを持ち，各トリガを正しい順序で送信しなければ正常に動作しないことで，解析者は部分的なトリガーだけでは解析できなくする．多段実行の流れとしては，図 3 のようになる．FTE は平文よりも暗号文 (悪性コード部) が長くなる性質がある．従って，復号が進むに連れて暗号文は短くなってゆく．結果，暗号文はマトリョシカ状になり，次の鍵がない限り解析者が復号することは困難となる．

gadget(悪性コード) の記述方法： 悪性コードはアセンブラで記述することで，高い自由度を担保する．実行前に mmap 関数を呼び，実行権限の付与された領域を確保する．確保した領域に gadget をコピーし，処理を移す．

図 3 に提案手法の動作例を示す．偽装した暗号文は鍵が送られてくる度に復号され，次の実行コードに変化していく．

提案手法では，gadget は関数内で実行フラグを与えた領域を mmap で確保する．その後実行したいコードを確保した領域にコピーし，実行する形で挿入する．元の論文 [5] では NX bit に関しては議論していないが，提案手法は NX bit は有効のままでも問題なく動作する．

トリガーとなる入力： 提案手法において，トリガとなる入力のフォーマットは表 1 のものとする．

表 1 トリガとなる HTTP リクエストのフォーマット

GET HTTP 1.1<CR><LF>
Host: 192.168.179.5<CR><LF>
特定のハッシュ値になる文字列<0x00>:FTE 復号用のフォーマット

IDS/IPS や DPI が動作するネットワーク上でも，トリガとなる入力を通常の HTTP ヘッダー行と同じ形式を使うなどすることにより，提案手法の動作に支障は来さないと考えられる．

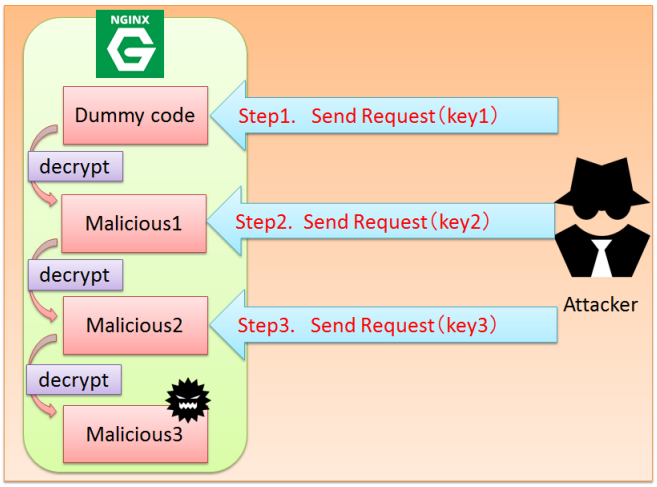

図 3　提案手法の動作例

5. 評価

本章では，4 節で述べた手法のプロトタイプを実装・検証を行う．その後今後の課題について述べる．

5.1 実験内容

実験は本プロトタイプが Web サービスを提供しているサーバ PC と Web サービスにアクセスできる攻撃者 PC によって構成されるネットワーク上で行う．攻撃者はサーバ PC 上へ HTTP 接続でき，特殊な HTTP ヘッダーを追加した HTTP レスポンスを送信できるものとする．簡単のため，ASLR は無効とし， IDS/IPS や DPI は動作していないものとする．また，FTE 復号用のフォーマットはプロトタイプ内に組み込んであり，HTTP リクエスト内にはトリガ入力の文字列のみが入っているものとする．静的解析対策として挿入されているバイトは可読性のために削除する．

本実験の目的は，提案手法が取りが入力を受信した場合に， gadget 部がどのように変化していくのかを確認することである．埋め込む gadget は execve 関数によって ”touch /tmp/phase?” と段階に応じたファイルを作成するコードとした．表 2 に実際に用いたコードを示す．Intel x86 アーキテクチャのアセンブラである Netwide Assembler(Nasm)[9] の記法に沿って記述した．0x00 バイトは安定な動作を妨げる可能性があるため，除いた．文書スペースの省略のため，一部改行を省いている．

実験で用いた FTE 暗号化フォーマットは表 3 のものを利用した．これらは正規表現で記述されている．一番最後の FTE 暗号化に利用したフォーマットは，Nginx 内の関数 ngx_error_log を参考に作成した．この関数を選択した理由は，アセンブルした結果に 0x00 を持っていないため，FTE 暗号化をする際に比較的安定した動作が見込めるためである．

表 2　touch /tmp/phase1 を実行するコード

```
BITS 32
; char *argv[] = ”/usr/bin/touch”, ”./phase1”, NULL;
; char *envp[] = NULL;
; execve(argv[0], argv, envp);
xor edx,edx
push edx
; /tmp/phase1
push 0x31657361 push 0x68702f2f push 0x706d742f
mov ecx, esp
push edx
; /usr/bin//touch
push 0x6863756f push 0x742f2f2f push 0x6e69622f push 0x7273752f
mov ebx,esp
push edx push ecx push ebx
mov ecx,esp
lea eax,[edx+0xb]
int 0x80
```

表 3　FTE 暗号化フォーマット雛形

```
^<擬態または実行コード部>(\\x90|\\x66\\x90)+\$
```

表 4　実験環境

CPU	Memory	OS
Intel Core i5 M540 2.53GHz	1GB	Ubuntu 14.04 32bit

5.2 実験結果

gadget 部の遷移の流れを図 4，5，6，7 に示す．各図は gadget 部を 16 進数で表現したものである．括弧で囲まれている数値は全体の大きさである．データ数が大きいものについては，画像上では一部省略している．FTE の暗号文は平文よりもデータ量が大きくなる性質上，最初の暗号文が大きく，復号されてゆくに連れて小さくなっていることが確認できる．図 5，図 6，図 7 ではほぼ同じ処理なため，データに差異はほとんど見受けられないが，図 4 と図 5 では，先頭の実行コード部が大きく異なることが確認できる．従って，正しい鍵がない限り，次の実行コード部を類推することは困難であると考えられる．

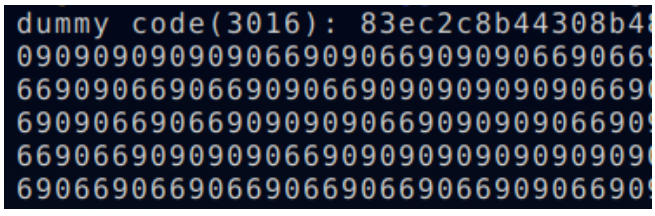

図 4　偽装コード

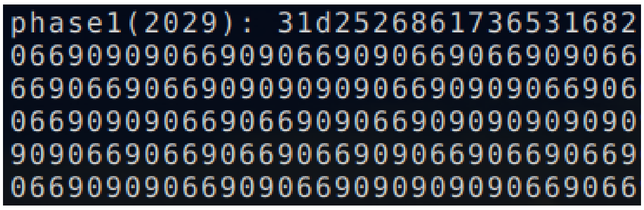

図 5　第一段階目 (touch /tmp/phase1)

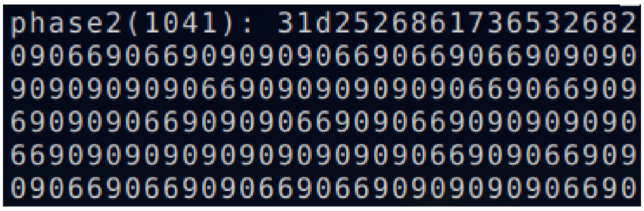

図 6　第二段階目 (touch /tmp/phase2)

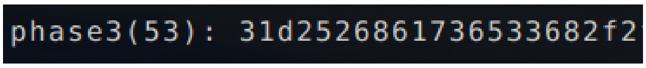

図 7　第三段回目 (touch /tmp/phase3)

5.3　今後の課題

現在の実装では，悪性コード部は実行用の関数が作成されているため，これを探すことで解析は容易になると考えられる．しかし，この点については関数のインライン化や従来の難読化により容易に隠蔽することが見込める．

6.　関連研究

最後に，本稿に関連した研究について紹介する．

6.1　ステガノグラフィー技術

6.1.1　プロトコル擬態化

擬態化は，本来の通信内容を他のプロトコルに偽装する，暗号通信の亜種である．擬態化の方法は，プロトコル自体を擬態化するフォーマット変形暗号 (FTE)[3]，StegoTorus[10] がある．FTE は平文と任意のフォーマットを与え，暗号化することで，そのフォーマットに沿った形で暗号化する手法である．Kevin[3] の研究では，FTE を使った通信で，中国で使われている DPI システムである Great Firewall をすり抜けることが確認されている．また，同じ著者により FTE 暗号を提供する API が提供されている [8]．単純にフォーマットを変換して送信するだけでは，トラフィック上でプロトコルに偏りが生まれて検知しやすくなる．この対策として，レスポンスの内容をより現実的な振る舞いをするようユーザが操作できる Marionette[4] がある．StegoTorus ではヘッダに擬態データを埋め込み，ステガノグラフィで HTTP に見せる手法を取られている．DPI では擬態しているプロトコルを禁止することにより対策ができるが，対象のサービスによっては禁止できない場合もある．フォーマットは柔軟性を持つため，様々な形に変換することができるが，通常のトラフィックとは違うデータになりがちである点が問題となっている．本稿では，トラフィックを暗号化するために使われている FTE を実行コードを暗号化するために用いている．

6.1.2　Trigger-Based Malware

Trigger-Based Malware は特定の入力以外では発火しない悪性コードを仕込んだソフトウェアである．解析対策は静的・動的解析両方の妨害手法が施されている．静的解析対策としては unaligned x86 code[6] の手法を，動的解析対策としては，特定の入力のときのみに発火する悪性コードを仕込む Trigger-Bug をそれぞれ用いている．Trigger-Bug は特定の入力のハッシュ値を用いることで，呼び出される部分の特定，及び呼び出される入力の特定を困難にしている．ハッシュ値は未初期化のスタック値を取り扱う必要があるため，ソースコードレベルでの改変が必要となっている．本稿での提案ソフトウェアは，この研究成果を基にステガノグラフィー技術を加えることで解析耐性を強化している．

6.2　マルウェア解析

6.2.1　静的解析

静的解析はコードを実行しないまま解析を行う特性上，難読化がされた場合の解析をどのように行うかが重要となる．Mihai らの研究 [11] では，不要な命令の挿入や細かな jmp 命令によるフロー制御がなされている場合でも検出する手法について議論し，検出器を開発している．Moser らの研究 [12] は難読化はその柔軟性故にパターンでの検出が極めて困難であることについて議論している．Barak らの研究 [13] は，難読化における論理的な限界を議論し，結果として非常に単純な公式の下であっても難読化は不可能であることを示している．静的解析手法はプログラムを実行しない特性上，単体では難読化による解析妨害と相性が悪い．そのため，動的解析と組み合わせることで対処することが多い．本稿で行っている実行コード部の暗号化も難読化の一種である．

6.2.2　動的解析

日々大量に生成されるマルウェアの自動解析に関する研究は盛んに行われている．実行中のプログラムにコードを挿入し，メモリの情報の取得や制御の変更などを行うことができる技術に Dynamic Binary Instrumentation(DBI) がある．解析対象のプログラムにエージェントと呼ばれる情報収集用のプログラムを挿入し，解析する DBI には Intel Pin[14] がある．これは豊富な API を提供し，かつマルチプラットフォームな解析を可能とする．

また，QEMU[15] をベースにした DBI では，解析対象の OS をフルエミュレーションすることで上位のリングで動作するマルウェアの解析も行得る利点がある．これには QEMU にテイントと呼ばれるタグを付けることでデータ

の伝搬を追跡することができるテイント解析機能をつけたTEMU[16]や，Pin の API を使うことができる PEMU[17]がある．DBI の解析を回避する手法として，エミュレーター上で実行するときの特徴から検知する手法がある．本稿では仮想化に関する妨害手法は施していないため，DBI による解析は通常通りに行えると考えられる．

シンボリック実行をサポートするライブラリとしてはGithub でオープンソースとして公開されている angr[18][19]がある．シンボリック実行は特定のアドレスへ到る入力や出力を含めたパスを求めることができる．提案ソフトウェアは悪性コード部へのパスはプログラム内には存在しないため，シンボリック実行での解析は困難であると考えられる．

7. まとめ

本稿では，静的・動的解析について耐性のあるソフトウェアの対策を考えるために，その前段階として Dennis らの研究 [5] をステガノグラフィー技術を用いて，その耐性を強化したプロトタイプを作成した．解析耐性の強化には gadget 部分を FTE を用いて実行可能なコードに暗号化し，同じ暗号文を複数回暗号化することで多段階の復号・実行が可能な gadget を作成した．プロトタイプはサーバプログラムとして動作し，特定の HTTP ヘッダー行を含む HTTP リクエストを受け取ると通常は実行されない gadget コード部を実行する．今後は既存の静的・動的解析ツールを用いて解析耐性の評価を行っていく．

8. 謝辞

本研究は JST ACT-I の助成を受けたものである．

参考文献

[1] Christopher Kruegel. Full system emulation: Achieving successful automated dynamic analysis of evasive malware. In *Blackhat USA*, 2014.

[2] Christopher Kruegel. Understanding and fighting evasive malware. In *RSA CONFERENCE EUROPE*, 2013.

[3] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *2013 ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany, November 4-8, In Proceedings of 2013*, pages 61–72, 2013.

[4] Kevin P. Dyer, Scott E. Coull, and Thomas Shrimpton. Marionette: A programmable network traffic obfuscation system. In *Proceedings of the 24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 367–382, 2015.

[5] Dennis Andriesse and Herbert Bos. Instruction-level steganography for covert trigger-based malware - (extended abstract). In *Proceedings of the 11th DIMVA International Conference, Egham, UK, July 10-11, 2014. Proceedings*, pages 41–50, 2014.

[6] Cullen Linn and Saumya K. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, pages 290–299, 2003.

[7] Chrome web dev plugin with 1m+ users hijacked, crams ads into browsers. https://www.theregister.co.uk/2017/08/02/chrome_web_developer_extension_hacked/.

[8] Daniel Luchaup, Kevin P. Dyer, Somesh Jha, Thomas Ristenpart, and Thomas Shrimpton. Libfte: A toolkit for constructing practical, format-abiding encryption schemes. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 877–891, 2014.

[9] Netwide assembler. http://www.nasm.us/.

[10] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the ACM Conference on Computer and Communications Security, Raleigh, NC, USA, October 16-18, 2012*, pages 109–120, 2012.

[11] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*, 2003.

[12] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *23rd ACSAC 2007, Miami Beach, Florida, USA, December 10-14, 2007,*, pages 421–430, 2007.

[13] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012.

[14] Vijay Janapa Reddi, Alex Settle, Daniel A. Connors, and Robert S. Cohn. PIN: a binary instrumentation tool for computer architecture research and education. In *Proceedings of the WCAE@ISCA 2004, Munich, Germany, June 19, 2004*, page 22, 2004.

[15] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*, pages 41–46, 2005.

[16] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. BitBlaze: A new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security. Keynote invited paper.*, Hyderabad, India, December 2008.

[17] Junyuan Zeng, Yangchun Fu, and Zhiqiang Lin. PEMU: A pin highly compatible out-of-vm dynamic binary instrumentation framework. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Istanbul, Turkey, March 14-15, 2015*, pages 147–160, 2015.

[18] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*, 2016.

[19] angr github. https://github.com/angr/angr.