学位論文　博士（工学）

マルウェアの特徴的振る舞いの誘発によるマルウェア対策手法に関する研究

2013年度

慶應義塾大学大学院理工学研究科

糟谷 正樹

マルウェアの特徴的振る舞いの誘発による マルウェア対策手法に関する研究

糟谷 正樹

論文要旨

マルウェア対策はコンピュータセキュリティ上の重要な課題である．コンピュータがマルウェアに感染するとログインアカウントやパスワードなどの個人情報が盗まれたり，詐欺被害に遭うなど様々な被害を受ける．セキュリティベンダの報告によると，2012 年に新たに報告されたマルウェアは 4,000 万件以上に上る．そのため，防御側は増え続けるマルウェアに対して適切に対策を行う必要がある．

マルウェア作成の技術は向上し続けているため，精巧に作られたマルウェアに対処することは困難である．マルウェア作成者は polymorphism や metamorphism を利用してアンチウイルスソフトを回避する．アンチウイルスソフトが利用するシグネチャマッチングと呼ばれる方法は，既知のマルウェアを解析して得たシグネチャと検査対象のプログラムのバイナリ列が一致するかを調べる．そのため，僅かにバイナリを変更することによりマルウェアは容易にシグネチャマッチングを回避できる．一方，振る舞い検出による対策はマルウェア間に共通する振る舞いが検査対象のプログラムに含まれるかを検査する．ここでいう振る舞いとはシステムコールや API (Application Programmers Interface) の呼び出し列のことであり，静的解析や動的解析を利用して振る舞いを抽出する．しかし，マルウェアの多くは難読化を利用して，プログラムの振る舞いを正しく抽出することを困難にしている．動的解析は難読化の影響を受けないもののステルス性の高いマルウェアに対処できていない．ステルス性の高いマルウェアは頻繁な動作を避けて振る舞いを抽出することを困難にしたり，得られた振る舞いから悪意ある振る舞いを含むかどうかの判別を困難にするためである．

本論文では，マルウェアの特徴的な振る舞いを誘発するための作為的な環境を用意して，ステルス性の高いマルウェアに対処する方法を提案する．本手法はマルウェアの種類に応じた入力を作為的に与える環境を用意することにより，マルウェアが実行せざるを得ない状況を作り出す．その結果，頻繁な動作を避けるマルウェアを活性化させたり，得られた振る舞いが悪意ある動作を含むかどうかの

判別を困難にするマルウェアに作為的に特徴的な動作を誘発させることを可能とする．本論文では，提案手法の具体例としてアドウェア・スパイウェアと偽アンチウイルスソフトの 2 種類に対処する．これらは現在においても多くの影響を与えているマルウェアのためである．

アドウェアとスパイウェアは頻繁な動作を避けて自身の悪意ある振る舞いを隠す．そのため，効率良く詳細な解析を行うためには，アドウェア・スパイウェアを活性化させる刺激を外部から与えて，強制的に動作させる環境を用意することが好ましい．これを実現するために，本研究では Blayzard を提案する．Blayzard は Internet Explorer (IE) のアドオンである Browser Helper Object として実現されており，偽装した大量の IE イベントを作り出し，アドウェアやスパイウェアに挿入するシステムである．偽のイベントに騙されたアドウェアやスパイウェアはその挙動を活性化させるため，頻繁な動作を避けるアドウェア・スパイウェアであっても解析を行うことができる．Blayzard の有用性を確認するために，32 個のアドウェア・スパイウェアと 10 個の無害な BHO を利用した結果，Blayzard は全ての検体の挙動を活性化させて，その振る舞いを解析することができた．

偽アンチウイルスソフトは正規のアンチウイルスソフトの挙動を考慮して動作するため，得られた振る舞いが悪意ある動作を含むかどうかを判別することは難しい．この問題を解決するために，正規のアンチウイルスソフトと偽アンチウイルスソフトの違いを誘発する方法と，両者の判別を行う指標を発見することは重要である．本論文では，マルウェアを利用して振る舞いの違いを誘発する．マルウェアのあり・なしの環境を用意して，両方のアンチウイルスソフトを動作させる．その後，環境の違いで振る舞いに差が生じるかどうかを調べ，正規か偽アンチウイルスソフトであるか判別する．実験的な調査から，振る舞いの違いを判別する際にメモリ使用量が最も精度よく判別できることを示す．その違いを機械的に判別するために，統計手法であるリーベン検定を利用する．提案手法の有効性を示すために 39 個の偽アンチウイルスソフトと 8 個の正規のアンチウイルスソフトを用いた結果，各々を正しく分類することができた．正規のアンチウイルスソフトはマルウェアが存在する環境でのみメモリを著しく使用したのに対して，偽アンチウイルスソフトは環境の違いに関わらず，メモリ使用量が変わることはなかった．

A Study on Countermeasures against Malware by Stimulating Behaviors Characteristic of Malware

Masaki Kasuya

Abstract

Tackling malware is an important challenge to computer security. Malware damages computer systems by stealing personal information such as login account and password and/or makes money by deceiving victim users. A security vendor reported that it found over 40 million samples of malware in 2012. To protect computer systems from malware, Defenders and researchers must reveal behaviors of malware, and develop practical countermeasure systems against it.

It is difficult to detect sophisticated malware instances. Malware developers frequently use the technique of polymorphism and metamorphism, and try to evade traditional malware detector, or antivirus software (AV). AV uses signature-based approach that identifies known malware instances by comparing the binary image of a number of uniquely characterizing signatures. It is purely syntactic and ignores semantics of instructions. In contrast to signature-based approach, behavior-based approach is easy to catch semantic information. The approach extracts system calls and API (Application Programmers Interface) calls by using static analysis or dynamic analysis. However, static analysis is evaded by obfuscation technique because it is difficult to make analysts extract correct behaviors. Dynamic analysis is resistant to obfuscation technique but cannot capture behaviors of stealthy malware because stealthy malware avoids showing malicious behaviors frequently and makes it difficult to infer whether the behaviors are benign or not.

This dissertation introduces a novel approach to deal with stealthy malware. To this end, an execution environment to inject "feed" inputs considering kinds of malware is prepared to intentionally stimulate and extract behaviors characteristic of malware. Even stealthy malware is activated and the behaviors are revealed. As a result, this approach can efficiently extract behaviors that stealthy malware executes occasionally and identify whether extracted behaviors are benign or malicious. To show the usefulness, this dissertation conducts two case studies, adware/spyware and fake AV, because

they are known as dominant threats in computer security. In principle, this approach is applicable to other kinds of malware.

Adware and spyware avoid frequent execution to hide their malicious behaviors. To perform the detail analysis, it is needed to extract their behaviors by giving adware and spyware "feed" to stimulate their behaviors. To this end, an environment to inject "feed" inputs is prepared. Since the environment is able to inject a large amount of feed inputs, it is capable to make adware and spyware activate their behaviors. As one example of the idea, Blayzard, a Browser Helper Object (BHO) of Internet Explorer (IE), is implemented. It generates bogus several IEs' events and injects them into adware and spyware instances. As a result, adware and spyware are deceived by the bogus events, and reveal the behaviors characteristic of them. To show the effectiveness, 32 adware/spyware instances and 10 benign BHOs have been used. In this experiment, Blayzard could extract the behaviors of 32 malicious instances and 10 benign samples.

Fake AV mimics behaviors of legitimate AV. Since it is difficult to judge whether AV-like behaviors are benign or not, it is important to find an indicator to distinguish legitimate AV and fake AV, and stimulate behaviors characteristic of fake AV. To this end, two environments, a clean environment including no malware and an infected environment including malware are prepared to bring out differences of behaviors between legitimate AV and fake AV. This approach uses malware as "feed". Experimental investigation shows memory usage is a good indicator. In this experiment, Levene Test, a statistical test, correctly identifies all fake AV samples (39 out of 39) as fake and all legitimate AV products (8 out of 8) as legitimate. All legitimate AV products significantly consume the memory usages in infected environments, but fake AV hardly changes the usages between clean and infected environments.

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景

1.1.1　マルウェアの脅威

マルウェア対策はコンピュータセキュリティ上の重要な課題である．マルウェアとは悪意あるソフトウェアの総称であり，マルウェアがコンピュータに感染することにより様々な被害を発生させることが可能となる．マルウェアが与える代表的な被害例として，サービス停止攻撃，パスワードなどの機密情報の流出，コンピュータユーザへの金銭的な被害などがある．

近年のマルウェアは個人情報の流出や金銭詐欺などの被害を引き起こすために犯罪者が利用している [1–5]．情報の流出を目的としたマルウェアの場合，例えば Market score はクレジットカードやインスタントメッセージの履歴を収集して外部に送信していた [6]．2013 年の McAfee のレポートによると，Android/Kaospy.A と呼ばれるスマートフォンを標的としたマルウェアが SMS メッセージや位置情報を収集して外部に送信していたことが報告されている [7]．金銭詐欺を目的としたマルウェアの場合，Doctor Virus というマルウェアは 3 年間の詐欺行為により 980 万ドル以上の収益をあげていた [8]．また Innovative Marketing Ukraine という会社は詐欺を行うマルウェアの販売により，年間 1 億 8,000 万ドル以上の収益をあげていた [9]．サイバー犯罪者はマルウェアを利用して得られた個人情報やクレジットカード番号などを犯罪者コミュニティで売買していることが Franklin や Holz らの研究により明らかになっている [2, 10]．このように，マルウェアは感染したコンピュータに害を与えるだけでなく，サイバー犯罪者の資金源に繋がることが分かる．

その上，マルウェアの種類は年々増加している．図 1.1 は McAfee Labs のデータベースに登録された年度ごとのマルウェアの種類の総数である [11–15]．2008 年から 2009 年の間に登録されたマルウェアの種類は 1,000 万件程度であったのに対して，2012 年から 2013 年にかけては，その 4 倍の約 4,000 万件に到達している．

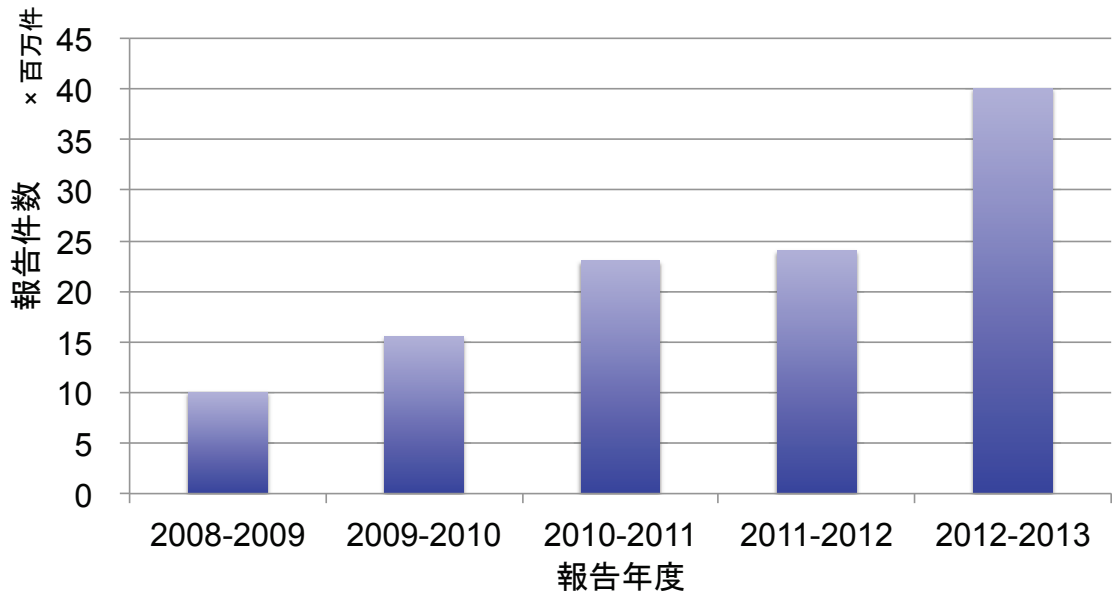

図 1.1: **McAfee Labs** のデータベースに登録された年度ごとのマルウェアの種類の総数．文献 [11–15] より引用．

また，2008 年から 2013 年にかけて合計すると約 1 億 1,000 万件のマルウェアが出現している．従って，コンピュータを安全に利用するためには増加し続けるマルウェアに適切に対応する必要がある．

1.1.2 マルウェア対策手法

一般的なマルウェアへの対策はアンチウイルスソフト [1] を利用する方法である．基本的に，アンチウイルスソフトによるマルウェアの対策はシグネチャマッチングによる判別である．シグネチャマッチングによる検査は，シグネチャと検査対象のバイナリが一致するかを調べてマルウェアであるかどうかを判別する方法である．シグネチャはマルウェアを解析して得られるバイナリ列の一部を抽出したものであり，既知のマルウェアから取得する必要がある．新たに出現したマルウェアに対応するためには，その都度マルウェアの解析を行いシグネチャを作成，更新する必要がある．

しかし，アンチウイルスソフトは直近で出現したマルウェアに十分に対応できていないことが Oberheide らの調査により明らかになっている [16]．その主な原因は解析しなければならないマルウェアの種類が膨大であり，シグネチャの作成が

[1] 本論文ではアンチウイルスソフトを広義の意味で取り扱う．即ち，ウイルスのみならずマルウェア全般に対処するソフトウェアとしてアンチウイルスソフトという言葉を用いる．

追い付かないためである．図 1.1 の 2012 年から 2013 年のマルウェアの報告数を見てみると 1 日あたり約 11 万件のマルウェアが報告されていることになるため，人海戦術により解析結果をシグネチャに反映することは現実的に不可能になっている．マルウェアの種類が増えた理由は，マルウェアの亜種の作成が容易となっているためである．マルウェアの亜種はあるオリジナルのマルウェアと同一の機能を提供するものの，そのバイナリ表現が僅かに異なるマルウェアのことである．

マルウェアの亜種はシグネチャマッチングによる対策を回避できる [17]．基本的に，シグネチャマッチングによる対策はシグネチャが保持しているバイナリ列と一致する検査対象のプログラムをマルウェアとして判別するため，あるシグネチャがあるオリジナルのマルウェアを検出できたとしても，その亜種のバイナリ列はオリジナルのものと僅かに異なるため，マルウェアとして判別することはできない．その結果，亜種ごとにシグネチャを作成する必要がある．実際に，亜種を作るためのツールは広く利用されている [18–20]．2009 年には，Zeus と呼ばれるマルウェア作成ツールを利用することにより数百万以上の亜種を作成していたことから，シグネチャマッチングのみによる対策は現実的ではない [21]．

シグネチャマッチングに変わる判別方法としてマルウェア特有の振る舞いを検出して判別を行う方法がある．振る舞い検出によるマルウェア対策は検査対象のプログラムが悪意ある振る舞いを含むかを調べることにより，マルウェアであるかどうかを判別する方法である．振る舞い検出はバイナリ列の一致による判別を行わないため，亜種の対応が容易になる．亜種はバイナリ表現が異なるものの提供する機能，すなわち振る舞いは変わらないためである．同様に，未知のマルウェアであっても基本的な振る舞いが同じであれば対処することが可能である，こうした恩恵から，現在振る舞いを利用した対策に関する研究が数多く提案されている [22–30]．

振る舞い解析を利用する方法は静的解析と動的解析の 2 つに大別できる．静的解析はディスアセンブラやデバッガなどを用いて，解析対象のバイナリを実行することなくその振る舞いを取得する方法である [22–24]．静的解析を利用する方法は，ディスアセンブラから取得した検査対象のマルウェアの制御フローが検査機構が予め保持している制御フローと一致するかどうかにより判別を行う．先にも述べたように，振る舞い解析はバイナリ列の一致による判別を行わないため，制御フローを正しく取得できれば亜種間のバイナリの僅かな違いの影響は受けない．しかし，難読化を行ったマルウェアは静的解析を利用して正しく判別することができない．難読化はディスアセンブルを行ったプログラムからその挙動を推測す

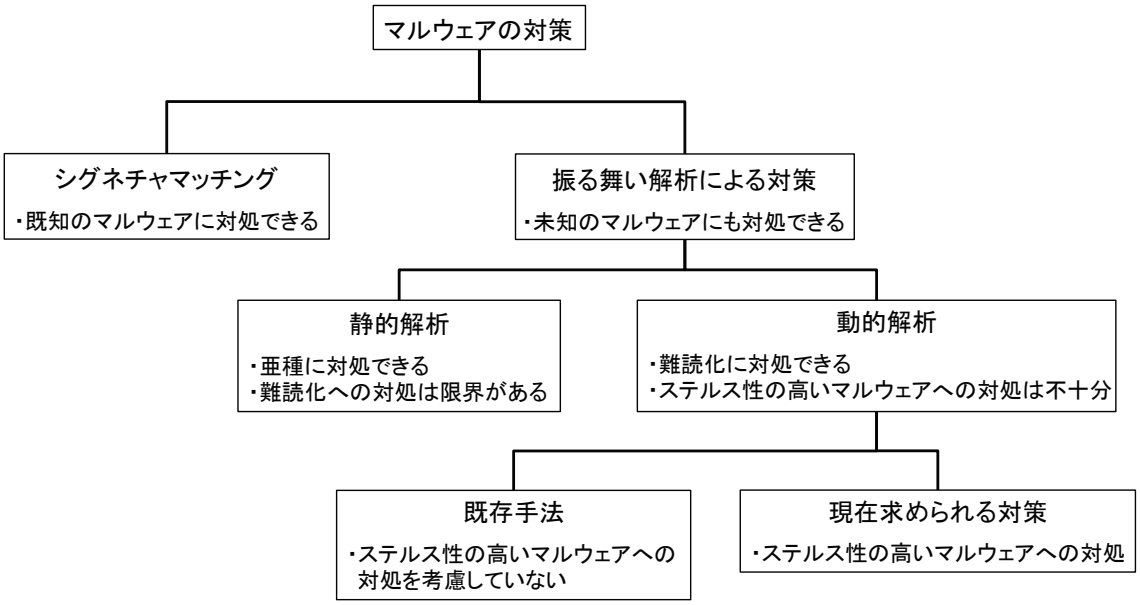

図 1.2: マルウェア対策手法とその対象範囲

ることを困難にする方法である．例えば，静的解析はジャンプ命令の飛び先の番地の値を実行時に決定する難読化を行うマルウェアの制御フローを正しく取得できない．なぜならディスアセンブルを行った時点ではジャンプ命令の飛び先の番地は不定なので，どの番地に飛ぶかを決定できないためである．このような問題から，マルウェアの対策を静的解析のみで対処することはできず，動的解析を利用する必要があることが Moser らにより指摘されている [31].

難読化に対処するために動的解析を利用したマルウェアへの対策方法が数多く提案されている [25–30]．動的解析は実際にマルウェアを動作させることによりマルウェアが発行するシステムコールや API を抽出する方法である．動的解析は実際にマルウェアを実行させて解析を行うため，ディスアセンブルにより取得できるコードに影響を与える難読化の影響は受けない．

1.2 既存の対策手法が抱える問題

現在のマルウェア対策手法が対処することができるマルウェアの分類を図 1.2 に示す．スパイウェアや偽アンチウイルスソフトといったマルウェアの種類に関わらず，一旦シグネチャを作成したマルウェアについてはシグネチャマッチングにより対処することができる．しかし，亜種や新種といった未知のマルウェアは出現した時点ではそれらのシグネチャを保有していないため，対応するシグネチャ

を作成するまではシグネチャマッチングによる対策はできない．

振る舞い解析による対策はバイナリ列の一致による判別を行わないため，未知のマルウェアに対処することができる．静的解析を利用する方法は亜種への対応を可能としているものの，ディスアセンブラを利用した制御フローの取得を困難にする難読化に対応できない．一方，動的解析はディスアセンブラを利用しないため難読化の影響を受けない．

現在の動的解析はステルス性の高いマルウェアに対処できない．ステルス性の高いマルウェアは悪意ある振る舞いを掴ませないように動作する．例えば，1) 頻繁な動作を避ける，2) 害のないソフトウェアの挙動を模倣して，悪意ある振る舞いの判別を困難にする方法を利用する．マルウェアの種類に関わらず，この 2 つの特徴を持つあらゆるマルウェアはステルス性の高いマルウェアとなり得る．頻繁な動作を避けるステルス性の高いマルウェアの場合，そもそもマルウェアが動作しないため，動的解析を利用してもマルウェアの特徴的な振る舞いを取得することができない．害のないソフトウェアの挙動を模倣する場合，システムコールや API 列を取得することができたとしても，それらから悪意ある振る舞いを含んでいるかどうか判別することは難しい．動的解析システムは基本的に検査対象のマルウェアを単純に実行させた振る舞いを利用するため，ステルス性の高いマルウェアに対処するためには，作為的に特徴的な振る舞いを誘発する必要があるといえる．

1.3 特徴的振る舞いの誘発

ステルス性の高いマルウェアに対処するために，本研究では，マルウェアの種類を考慮して，種類ごとの特徴的な振る舞いを誘発するための入力を作為的に与えることによりステルス性の高いマルウェアの動きを活性化させる方法を提案する．提案手法により，ステルス性の高いマルウェアであっても，作為的に仕掛ける入力により特徴的な振る舞いを活性化できるため，ステルス性の高いマルウェアの挙動を誘発することが可能となる．言い換えると，ステルス性の高いマルウェアが動作せざるを得ない環境を意図的に用意することにより，その振る舞いを誘発する．

本研究では，特にアドウェア，スパイウェア及び偽アンチウイルスソフトに対象にした 2 つの特徴的な振る舞いを誘発するアプローチを提案する．第 1 に，ブラ

ウザの状態を偽装する入力を作成・注入することにより，頻繁に動作することを避けるアドウェア・スパイウェアの特徴的な振る舞いを誘発し，抽出するシステムである Blayzard を提案する．Blayzard はブラウザの状態を示すイベントを大量に解析対象のアドウェア・スパイウェアに注入することにより，出現頻度が少ないアドウェアやスパイウェアの振る舞いを意図的に誘発できる．第 2 に，正規のアンチウイルスソフトの挙動を模倣する偽アンチウイルスソフトの特有な動作を誘発するために，作為的にマルウェアを与えて正規のアンチウイルスソフトとは異なる振る舞いを誘発する．これは偽アンチウイルスソフトが正規のアンチウイルスソフトとは異なり，十分なマルウェア検出能力を伴わないことに着目している．正規のアンチウイルスソフトと偽アンチウイルスソフトに対してマルウェアを作為的に与え，マルウェアを検出する際に観察できる特徴的な振る舞いを示すかどうかにより判別を行う．

アドウェア，スパイウェア及び偽アンチウイルスソフトを対象とした理由は，提案手法によるアプローチが実在する影響力の大きいマルウェアに対して有用な方法であることを示すためである．しかしながら，本手法はこれらのマルウェアに限定する方法ではなく，原理的にどのマルウェアに対しても適用できる．スパイウェアやアドウェアにおいて，Moshchuk らは調査を行った 18,000,000 個の URL のうち 13.4% がスパイウェアを含み，そのうち 91% がアドウェアの機能を所持していたと報告している [32]．カスペルスキーによる調査ではアドウェアが継続的に上位 15 位以内に登場していたことを報告している [33–36]．2013 年の McAfee の報告によると，収集したマルウェアのうちの約 1/3 がスパイウェアであったという報告がなされている [37]．偽アンチウイルスソフトについては，Rajab らによると Google のマルウェア収集機構を利用して集めたマルウェアのうち全 15% が偽アンチウイルスソフトであることが明らかにされている [38]．

1.3.1 ウェブブラウザの状態を偽装して挙動を誘発するアドウェア・スパイウェアの解析手法

本論文では Internet Explorer (IE) のアドオンである Browser Helper Object (BHO) として実装されているアドウェアやスパイウェアを対象にしている．BHO はイベント駆動型の IE のアドオンであり，IE は処理内容に応じて様々なイベントを BHO に対して送信するため，対応するイベントハンドラを実装することによって

IE の動作を変更したり，新しい機能を追加したりすることができる．例えば BHO が IE のページ遷移を制御することが可能になる．多くのアドウェアやスパイウェアが BHO として実装されていることが既存研究によって明らかになっているため [32,39]，BHO ベースのスパイウェアやアドウェアに対応できる機構を用意することにより多くのアドウェアやスパイウェアに対処することができるためである．

BHO ベースのアドウェアやスパイウェアには次の 3 つの特徴があり，BHO ベースのアドウェアやスパイウェアの振る舞い解析を行うためにはこれらの問題を解決する必要がある．

1. ブラウザへの寄生: BHO ベースのアドウェアやスパイウェアは IE の一部として動作するため，解析を行った際に得られる振る舞いがブラウザ本体の動作かアドウェア，スパイウェアの動作かを区別しにくい．

2. 頻繁な動作の回避: 自身の存在を悟られないようにするために頻繁な動作を避ける．また，イベント駆動型の IE のアドオンであることから解析を行う際にブラウザを動作させる必要がある．

3. 暗号化や難読化への対処: 暗号化や難読化を施したアドウェアやスパイウェアに対して，デバッガやディスアセンブラを用いた解析は困難となる．

本論文では，上述した 3 つの問題に対して次に述べる 3 つの方法により各々に対処する．

1. ブラウザと BHO の分離: ブラウザと BHO をプロセス単位で分離して解析を行うことによりブラウザの影響を排除する．そのため，分離した BHO のプロセスのみを監視することにより，アドウェアやスパイウェアのみの振る舞いを抽出することができる．またブラウザと BHO はそれぞれ別プロセスに配置されているため，BHO ベースのアドウェアやスパイウェアが IE のメモリを直接操作することを防ぐ．

2. 偽のイベント注入による振る舞いの活性化: 偽のイベント列の生成，注入により，頻繁な動作を行わないステルス性の高いアドウェアやスパイウェアの動作を活性化させる．ブラウザの操作なしに大量のイベント列を容易に生成することができるため，稀にしか動作しない BHO を強制的に作動させることができる．

3. 動的解析: 動的解析により BHO ベースのアドウェアやスパイウェアの振る舞いを抽出する．実際に対象の検体を動作させて，その振る舞いを抽出する．そのため，検査対象のバイナリに暗号化や難読化を行ったとしても，その影響を受けずに解析を行うことができる．

1.3.2 マルウェアを利用して特徴的な挙動を誘発する偽アンチウイルスソフトの判別手法

偽アンチウイルスソフトは正規のアンチウイルスソフトの挙動を真似て動作するマルウェアである．偽アンチウイルスソフトは詐欺行為により利益を得るために，実際にはマルウェアに感染していないにも関わらず嘘のマルウェア感染報告を表示して，感染報告に表示されたマルウェアを除去するためにユーザにワクチンソフトの購入を促す．偽アンチウイルスソフトが表示する嘘の警告に騙されたユーザはワクチンソフトを購入してしまうため，サイバー犯罪者は不正に利益を取得することができる．

偽アンチウイルスソフトであるかどうかの判別を行うために，動的解析を利用した振る舞い解析を行うことは困難である．これは API やシステムコール列などの情報から詐欺行為を判別することができないためである．加えて，より精巧に作られた偽アンチウイルスソフトは正規のアンチウイルスソフトの挙動を考慮して作成されているため， API やシステムコール列による比較では，正規のアンチウイルスソフトと偽アンチウイルスソフトを正しく判別することが困難となる．

そこで，偽アンチウイルスソフトの特有の振る舞いを得るために，マルウェアを利用してその特徴的な振る舞いを誘発する．偽アンチウイルスソフトは正規のアンチウイルスソフトと比べて十分なマルウェア検出能力を有していないため，マルウェアに感染した環境において，偽アンチウイルスソフトと正規のアンチウイルスソフトの間には振る舞いの違いが存在すると考えられる．正規のアンチウイルスソフトの場合，マルウェアを検出した際の詳細なパターンマッチングやエミュレーションなどにより CPU やメモリ使用量といった計算機資源の変化を期待できる．そのため，マルウェアのあり，なしによる振る舞いの違いが存在すると考えられる．一方，偽アンチウイルスソフトの場合，正規のアンチウイルスソフトが行うようなマルウェアスキャンを実施しない．そのため，マルウェアのあり・なしに関わらず振る舞いの違いが生じないと考えられる．

本論文では，まず，実験的な調査からメモリ使用量が正規のアンチウイルスソフトと偽アンチウイルスソフトを判別することができる指標となることを示す．次に，マルウェアのあり，なしの環境において，メモリ使用量に有意な差が生じるかどうかを判別するために，統計手法であるリーベン検定 [40] を利用した判別手法を提案する．リーベン検定を利用した判別では，マルウェアのあり・なしの環境から取得したメモリ使用量の間に統計的な有意差が生じる場合は正規のアンチウイルスソフトとして判別される．逆に，差がない場合は偽アンチウイルスソフトとして判別される．

1.4 本研究の貢献

本研究ではマルウェアの特徴的振る舞いを誘発するアプローチを実現する具体的な 2 つの手法を提案している．まず，第 4 章では大量のイベントを注入することにより，BHO ベースのアドウェアやスパイウェアの振る舞いを活性化して，その動作解析を行う Blayzard を提案している．実験の結果，Blayzard は 32 個の BHO ベースのアドウェアとスパイウェアの振る舞いを解析することができた．特に 10,000 回イベントを注入して活性化したステルス性の高い `autoex.dll` の振る舞い解析を行うことができた．次に，第 5 章ではマルウェアのあり，なしの環境から得られる振る舞いの違いから，正規のアンチウイルスソフトと偽アンチウイルスソフトを判別する手法を提案している．実験の結果，39 個の偽アンチウイルスソフトと 8 個の正規のアンチウイルスソフトを正しく分類することができた．利用した 39 個の偽アンチウイルスソフトのうち，よりステルス性の高い検体が 6 つ存在したものの (第 5 章で述べる粗悪なアンチウイルスソフトのこと)，提案手法はそれらも正しく判別することができた．

本研究の貢献は表 1.1 のように表すことができる．つまり，本研究の特徴であるマルウェアの種類に応じた特徴的な振る舞いを誘発するアプローチにより，既存手法では対処することができなかったステルス性の高いマルウェアの動作解析や検出を行えるようにしたことである．その結果，ステルス性の高いマルウェアへの早期の対策が可能となり，コンピュータセキュリティの向上に寄与できる．本論文では，アドウェア，スパイウェア及び偽アンチウイルスソフトを対象にして特徴的な振る舞いを誘発するアプローチの有用性を示している．

表 1.1: 本研究の貢献

	亜種への耐性	難読化への耐性	ステルス性への耐性
シグネチャを利用する方法	X	X	X
静的解析を利用する方法	✓	X	X
動的解析を利用する方法	✓	✓	X
提案手法	✓	✓	✓

1.5 本論文の構成

本論文は全 6 章からなる．第 1 章では本研究の背景，動機および目的について述べ，本研究の学術的貢献について説明した．

第 2 章では本研究の関連研究をまとめる．シグネチャを利用した対策手法，静的解析や動的解析を利用した方法に関する既存手法をまとめ，本研究との差分を明らかにする．

第 3 章では，本研究の核となる考え方であるマルウェアの特徴的な振る舞いを誘発する方法について，その概念を導入する．

第 4 章と第 5 章では本研究の基本的な考え方である，“マルウェアの特徴を考慮してその振る舞いを誘発する” 手法に関する具体的な 2 つの方法について説明する．第 4 章では，頻繁な動作を避けるスパイウェア・アドウェアを解析するために，ブラウザのイベントを偽装して，そのイベントを大量に挿入して振る舞いを誘発してその振る舞いを取得する Blayzard について述べる．第 4 章では，Blayzard の核となる 3 つの特徴について述べた後，実在する検体を利用した実験を行い，Blayzard が BHO ベースのスパイウェアやアドウェアに対して有効な手段であることを示す．また，実験では頻繁に動作しなかった検体も取り上げ，偽のイベントを挿入する方法が頻繁な動作を避けるアドウェア・スパイウェアの振る舞いを誘発するための有用な方法であることを示す．

第 5 章では，偽アンチウイルスソフト特有の振る舞いを揺さ振り出すために，マルウェアを利用してその特徴的な振る舞いを誘発する手法について述べる．本手法は偽アンチウイルスソフトは正規のアンチウイルスソフトと異なりマルウェアの検出能力が低いことを利用して，マルウェアのあり，なしにおける振る舞いの違いから正規のアンチウイルスソフトか偽アンチウイルスソフトであるかを識別する．第 5 章では，正規のアンチウイルスソフトと偽アンチウイルスソフトの違いが顕著に現れる指標を実験的に調査し，メモリ使用量が両者を判別する最も良い指標であることを示す．その後，統計手法を利用した具体的な判別方法を示す．

最後に第 6 章で本論文をまとめる.

第2章　関連研究

本章では，本研究の関連研究についてまとめる．攻撃側と防御側のマルウェア対策の進歩を通じて，現状のマルウェア対策が既に達成している部分を明らかにする．具体的には，シグネチャマッチングによる対策，振る舞い検出のための静的解析を利用する対策と動的解析を利用する対策について述べる．その後，既存の方法がステルス性の高いマルウェアに対応できないことを述べる．

2.1　シグネチャマッチングを利用する対策

シグネチャはマルウェアの特徴を示すバイナリ列の一部を抽出した定義ファイルであり，検査対象のバイナリ列の一部とシグネチャが一致するかどうかを調べることによりマルウェアを判別する方法である．シグネチャマッチングによる判別方法の動作概要を図 2.1 に示す．アンチウイルスソフトは検体 A がマルウェアであるかどうかを判別するために，検査対象のバイナリ列と一致するシグネチャが存在するかを調べる．図 2.1 中のシグネチャ A とシグネチャ B は検体 A のバイナリ列の一部と一致しない．しかし，シグネチャ C と検体 A のバイナリ列の一部が一致するため検体 A はマルウェアと判別される．

シグネチャマッチングによる対策は商用のアンチウイルスソフト [41–50] が採用している一般的なマルウェア対策であり，広く利用されている方法である．近年では，ヒューリスティックを利用してマルウェアを判別する方法 [51] なども利用されているものの，シグネチャマッチングによる対策はアンチウイルスソフトが利用する方法において現在も主流の方法である [52].

しかし，シグネチャを利用する対策は限界を迎えている．いずれの商用のアンチウイルスソフトも直近で出現したマルウェアを十分に検出できていないためである．シグネチャは人手によるマルウェアの解析や自動生成を支援する手法 [52] を利用して得ることができるため，基本的には後追いの対策となる．そのため，継続的にシグネチャを更新することにより，アンチウイルスソフトが対処できるマ

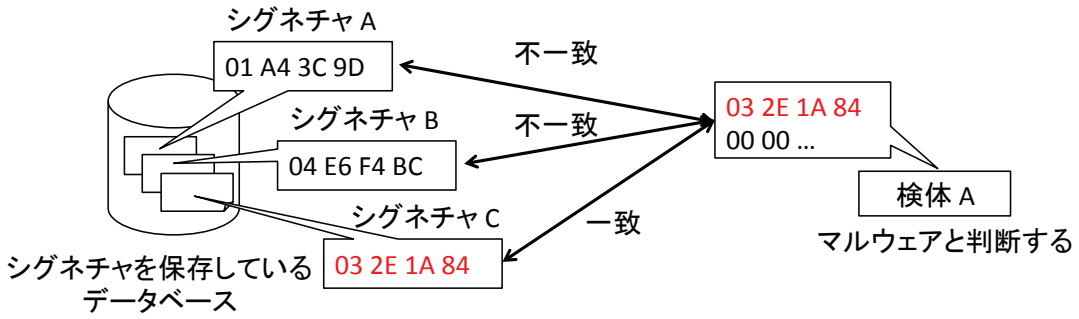

図 2.1: シグネチャマッチングの動作概要

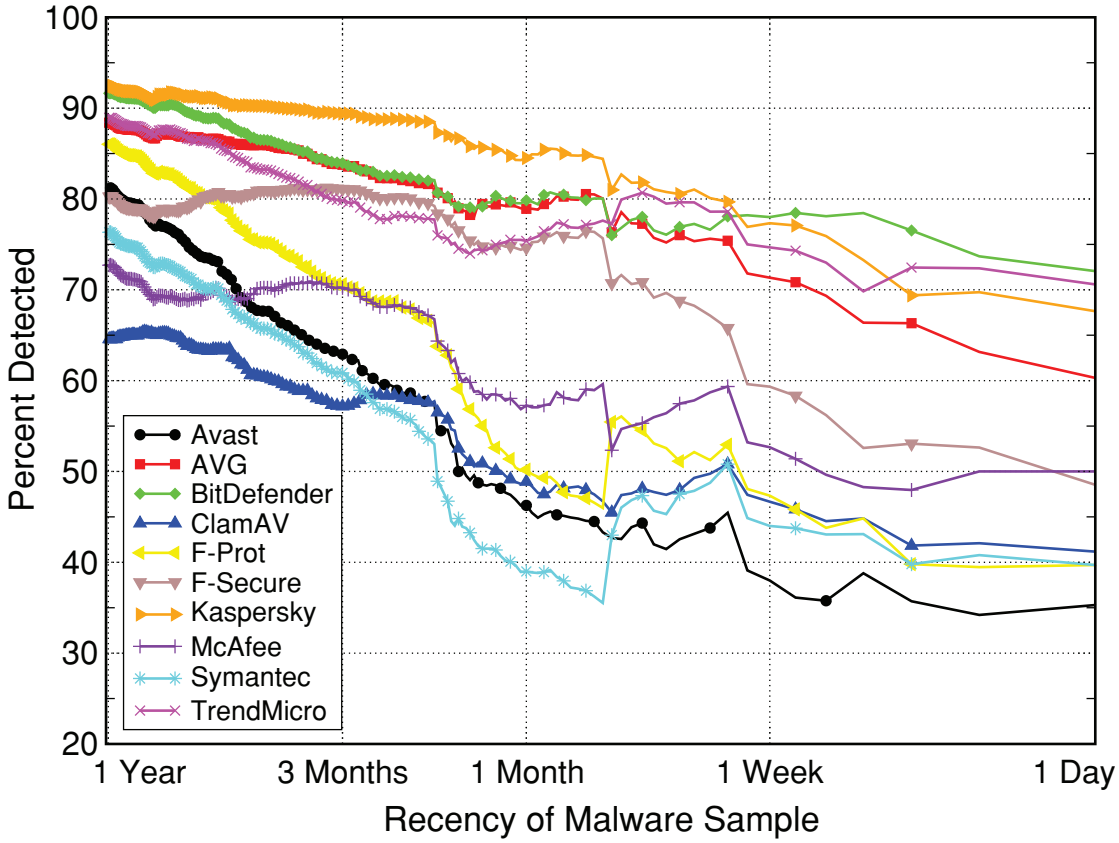

図 2.2: マルウェアの出現時期の違いによる各アンチウイルスソフトのマルウェア検出数の推移．文献 [16]，93 ページ，図 1(b) より引用．

ルウェアの数が増えることになる．Oberheide らは異なる出現時期のマルウェアに対して 10 種類のアンチウイルスソフトを利用した際に，検出率がどのように推移するかを調査している [16]．図 2.2 が示すように，いずれのアンチウイルスソフトを利用しても 1 年前に出現したマルウェアと比べると，直近で出現したマルウェアの検出率が減少していることが分かる．McAfee によると，直近の 1 年間で約 4,000 万件以上のマルウェアが報告されている [15]．つまり，1 日あたり約 11 万

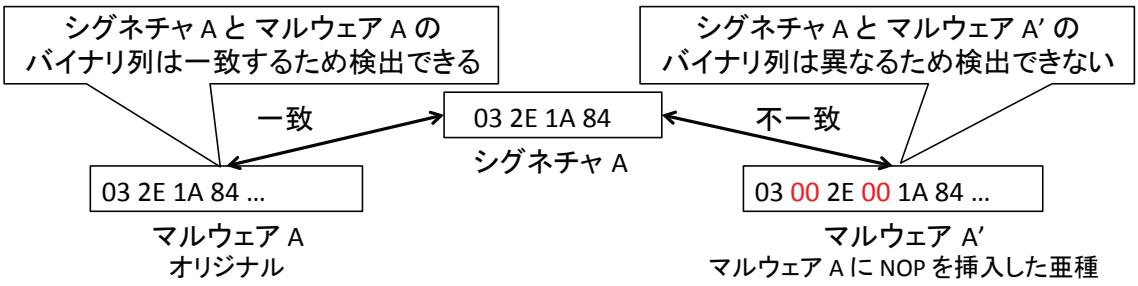

図 2.3: オリジナルのマルウェアとその亜種の関係，及びシグネチャの有効範囲

件の報告が McAfee に送信されることになる．このことからマルウェアの種類の増加に対してシグネチャの作成が追い付いていないといえる．

シグネチャの作成が追い付かない原因の 1 つはマルウェアの亜種の存在である．マルウェアの亜種はオリジナルのマルウェアと同一の機能を提供するものの，そのバイナリ表現が異なるマルウェアである．マルウェア作成者は *polymorphism* や *metamorphism* などを利用して亜種を作成する．Polymorphism は悪意あるコードを暗号化し，実行時に復号する処理を含める方法である．Metamorphism は悪意あるコードへの無意味な命令列の挿入や，その順序を変更するなどの方法を利用してバイナリ表現を変える方法である．

図 2.3 はオリジナルと亜種のマルウェアの関係と，シグネチャの有効範囲を示している．図 2.3 はオリジナルのマルウェア A に対して NOP を挿入することによりその亜種 A' を作成している．このとき，シグネチャ A とマルウェア A のバイナリ列は一致しているため，シグネチャ A はマルウェア A を検出することが可能である．しかし，シグネチャ A を利用してマルウェア A' を検出することができない．NOP の挿入により亜種 A' のバイナリ列がシグネチャ A が保持しているバイナリ列と異なるためである．

このように，シグネチャマッチングによる対策はオリジナルのマルウェアに僅かな変更を加えることにより容易に回避できることが知られており，多くの商用のアンチウイルスソフトが亜種の作成により回避されてしまうことが Christodorescu と Jha により示されている [17]．2009 年にはサイバー犯罪者は Zeus と呼ばれるツールを利用して数百万通りの亜種を作成していることから [21]，亜種の問題は現実に起きている問題であり，適切に対処するべき問題であることがわかる．

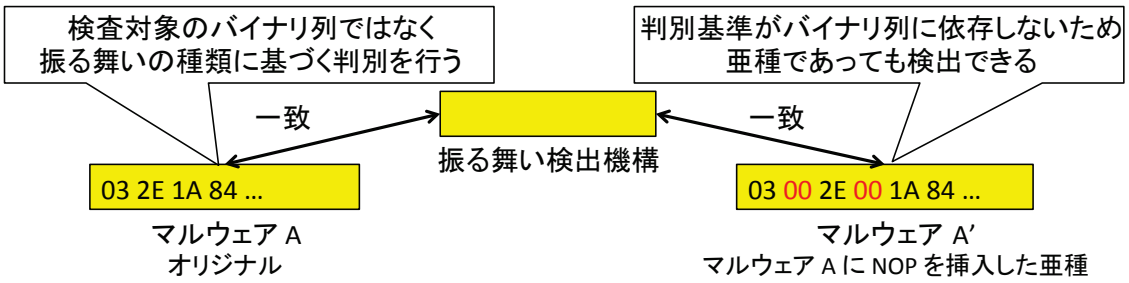

図 2.4: 振る舞い検出の有効範囲

2.2 振る舞い検出による対策

振る舞い検出を利用する対策はマルウェアの種類ごとにその特徴的な悪意ある振る舞いを定義して，定義した振る舞いと同様の振る舞いを持つバイナリをマルウェアとして判別する方法である．ここで述べる振る舞いとはプログラムが呼び出す API やシステムコール列のことである．具体的には，マルウェアの性質を特徴付ける API やシステムコールの呼び出しに関する制御フローやマルウェアがパスワード情報などの重要な情報を取得して外部に送信するといったデータフローを抽出して，その振る舞いと合致するプログラムをマルウェアとして判別する．

振る舞いを利用する方法の利点は亜種の対処が容易になることである．図 2.4 は振る舞い検出の有効範囲を示している．図 2.4 の振る舞い検出機構はマルウェアであるかどうかの判別をバイナリ列の一致による方法ではなく，検査対象のプログラムが実行する振る舞いに基づいて行われる．そのため，振る舞い解析はバイナリ列の僅かな変更による違いに対して耐性がある．図 2.4 のようにオリジナルであるマルウェア A と，A に NOP を挿入した亜種のマルウェア A’ は振る舞い検出機構によりマルウェアとして判別できる．

振る舞い検出を実現するための方法は，静的解析を利用して振る舞いを取得する方法と動的解析を利用して振る舞いを取得する方法に大別できる．静的解析はディスアセンブラやデバッガを利用して検査対象の振る舞いを取得する方法であり，動的解析は検査対象を動作させ，実行時の振る舞いを取得する方法である．以下，静的解析と動的解析を利用する対策手法について述べる．

2.2.1 静的解析

静的解析を利用する方法は検査対象のプログラムが悪意ある振る舞いを含むかどうかを調べるために，商用のディスアセンブラである IDA Pro [53] などを利用

してプログラムの振る舞いを抽出する．例えば，IDA Pro はディスアセンブルを行うと同時に，検査対象のプログラムの制御フローに関するグラフを抽出できるため検査対象のプログラムがどのように動作するのかを詳細に把握できる．以降に取り上げる 3 つの静的解析を利用する手法 [22–24] は IDA Pro を用いて振る舞いを取得している．ディスアセンブラにより得られた振る舞いが，マルウェアが共通して持つ特徴的な振る舞いを含む場合にマルウェアとして検出される．それ故，ディスアセンブラが解析対象のプログラムの振る舞いを正しく抽出できないと静的解析を利用する方法はマルウェアにより回避されることになる．

静的解析を利用するマルウェア対策

Kruegel らはカーネルモジュールとして実現されているルートキットと呼ばれるマルウェアを検出するための手法を提案している [22]．害のないカーネルモジュールと異なり，カーネルモジュールとして実現されているルートキットは頻繁にカーネル内に存在するプロセス構造体やシステムコールテーブルを変更することに Kruegel らは着目している．静的解析により検査対象のプログラムが直接カーネル内のデータ構造を変更する振る舞いを含むどうかを調べることによりカーネルモジュールとして実現されているルートキットを検出している．Kruegel らの手法は 985 個の害のないカーネルモジュールを正しく悪意のないプログラムとして判別し，8 個のルートキットを正しくマルウェアとして判別できている．

Kinder らはモデルチェックを利用したワームの亜種を検出できるシステムを提案している [23]．Kinder らのシステムは 1) 検査対象のバイナリをディスアセンブルして得られる制御フローのグラフと 2) システムが予め保持するマルウェアの振る舞いを記述したモデルが一致するかどうかを調べる．文献 [23] では，自分自身の実行ファイルを，同じファイルシステムの他のディレクトリにコピーする動作をモデルとして記述している．Kinder らのシステムは，そのモデルを利用して *Klez.a*, *Klez.e*, *Klez.g*, *Klez.h*, *NetSky.b*, *NetSky.c*, *NetSky.d*, *NetSky.e*, *NetSky.p*, *MyDoom.a*, *MyDoom.i*, *MyDoom.m* 及び *MyDoom.aa* の亜種を含む 13 個のワームを検出している．

Christodorescu らはマルウェア間に共通する特徴的な振る舞いをテンプレートとして保持して，検査対象をディスアセンブルして得られた制御フローグラフがテンプレートで記述してある振る舞いを含むかどうかを調べてマルウェアであるかを判別する手法を提案している [24]．テンプレートとは制御フロー中に記述され

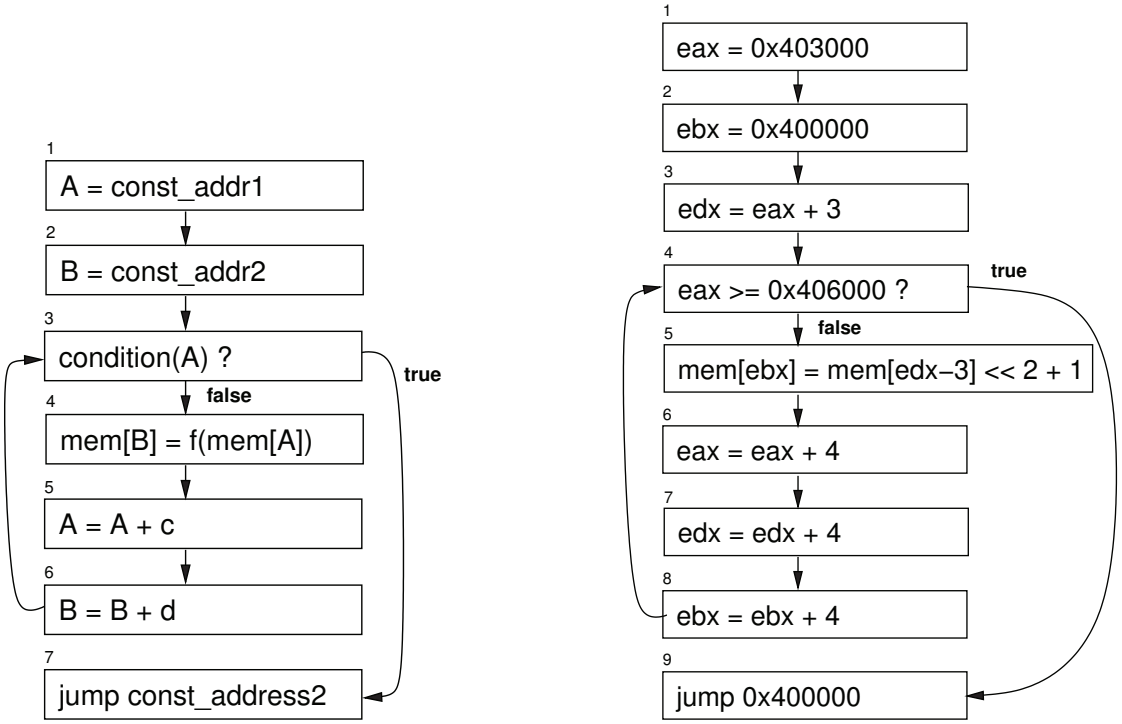

図 2.5: テンプレートとマルウェアから得られる制御フローグラフ．文献 [24]，34 ページ，図 1(a), (b) より引用

るレジスタや定数をシンボルや変数として置き換えた命令列のことを表す．これにより，マルウェアがレジスタの割り当てやメモリブロックの開始アドレスを変更した亜種を作成しても，1 つのテンプレートを利用して同様の振る舞いを含むマルウェアの亜種に対応できる．図 2.5 はテンプレート及びあるマルウェアから抽出した制御フローグラフである．左側のテンプレートはあるマルウェアの亜種に共通する復号ループを示しており，`const_addr1` からメモリに保存されている内容を復号して，その内容を `const_addr2` から順次書き込む振る舞いとなっている．テンプレート内の `const_addr1`, `const_addr2`, `f(X)`, `condition(X)`, `c`, `d` がそれぞれ `0x403000`, `0x400000`, `X` $\geq$ `0x406000`, `X << 2 + 1`, `4`, `4` となると右側のマルウェアから得られる制御フローグラフが一致するため，このプログラムはマルウェアとして判別される．

難読化による静的解析の回避

難読化はディスアセンブルしたコードからマルウェアの振る舞いを把握することを困難にさせる方法である．例えば，制御フローの把握を困難にするために，無

表 2.1: 難読化の分類

単純な難読化	NOP の挿入，コードの実行順序の変更， レジスタの再割当て，同等な命令列への置換
複雑な難読化	定数の値が実行時に定まるようにする方法 [31]

用な間接ジャンプを多用することにより，解析者はディスアセンブルしたコードから悪意ある振る舞いの部分を発見するために意味のないジャンプを辿る必要があるため解析の手間が増える．難読化は 表 2.1 で取り上げているように単純な難読化と複雑な難読化に分けることができる．単純な難読化は Christodorescu らの文献から引用している [54]．マルウェア作成者はこれらの難読化を複数組み合わせることができる．

静的解析による対策は単純な難読化に対処することはできるものの，複雑な難読化に対処することができない．なぜならば実行時に値が定まるような定数が制御フローに関与する場合，静的解析では正しい制御フローを取得できなくなるためである．例えば，ジャンプの飛び先を指定する定数が実行時に定まる場合，静的解析ではどの番地にジャンプするかを把握することができない．実際に，Moser らはバイナリを書き換えるツールを利用することにより，“実行時に値が定まるような定数” による難読化を施したマルウェアを静的解析を利用する対策で検出できないことを示している [31].

2.2.2 動的解析

動的解析はマルウェアを実際に動かして，マルウェアがオペレーティングシステムとのやり取りを行うインターフェース，つまりシステムコールのや API を監視してマルウェアの実行時の振る舞いを抽出する．動的解析を行うためのシステムは Qemu [55] などのエミュレータや仮想化環境などを利用してマルウェアを実行させて，システムコールや API の引数や返り値などを取得するものであり，こうしたシステムは既に広く利用されている [56–58]．エミュレータや仮想化環境を利用することによる利点の 1 つはマルウェアに感染させる前の状態に素早く復帰できるため解析の効率が上がることにある．近年では，エミュレータや仮想化環境を利用しなくても，マルウェアに感染する前の状態に素早く復帰できるシステ

表 2.2: 動的解析を利用する方法の分類

データフローを利用する方法	Egele らの手法 [25]，Panorama [26]
制御フローを利用する方法	Kolbitsch らの手法 [27]，JACKSTRAWS [28]，HOLMES [29]，HookFinder [30]

ムを Kirat らが提案している [59].

難読化を行ったマルウェアに対処するためには動的解析を利用することが 1 つの方法であると Moser らにより指摘されている [31]．動的解析はディスアセンブルを行わずマルウェアを実行させて，その実行時の振る舞いを取得するため難読化の影響は受けない．難読化の問題に加えて，静的解析ではデータフローや制御フローを完全に把握できないことが理論的に証明されている [60, 61].

動的解析を利用するマルウェア対策

動的解析を利用するマルウェア検出を手法は表 2.2 のようにデータフローを利用する方法と制御フローを利用する方法に分けることができる．データフローを利用する方法はパスワード情報やキー入力などの情報がコンピュータの外部に送信されるかどうかを調べる方法であり，制御フローを利用する方法はシステムコールや Win32 API などの発行パターンをマルウェアの検出に利用する方法である．

データフローを利用する方法の 1 つとして，Egele らのスパイウェアを検出する手法がある [25]．ブラウザのアドオンとして実現されたスパイウェアを検出するために，動的解析システムを利用してブラウザの閲覧履歴などの情報がブラウザの本体により処理されるのか，アドオンにより処理されるのかを調べることにより，検査対象のアドオンがスパイウェアであるかどうかを判別する方法である．Panorama [26] は Egele らの手法 [25] とは異なり，データフローの追跡範囲をキーボード入力やファイル，ネットワークアクセスなどシステム全体に拡張することにより，ブラウザのアドオンとして実現されたスパイウェアに限らず，キーボード入力を盗むキーロガーや悪意あるプロセスやプログラムの存在を隠蔽するルートキットなどのより多くのスパイウェアの種類についてその存在を検出すること可能にしている．

制御フローを利用する方法は Kolbitsch らの手法 [27]，JACKSTRAWS [28],

HOLMES がある．これらの方法は動的解析を行いシステムコールや API 列からなる制御フローを自動的に抽出して，検査対象の振る舞いグラフを作成する．その後，クラスタリングを行い，検査対象のプログラムがマルウェアのクラスタに分類されるかどうかにより判別を行う方法である．マルウェア間に共通する振る舞いグラフを抽出する点としては Christodorescu らの手法 [24] と似ているものの，動的解析を利用して振る舞いグラフを抽出するため難読化の影響は受けない．Kolbitsch らの手法はワームに [27]，JACKSTRAWS [28] はボットに，HOLMES [29] はスパイウェア，ワーム，ウイルス，バックドアに対応していることを実験的に示している．

HookFinder [30] はフックの挿入を検出して，そのフックの振る舞いを抽出する手法である．マルウェアはキー入力やブラウザの閲覧履歴などを取得するためにフックの挿入を行うという点に着目して，プログラムの実行経路がマルウェアのコード領域にジャンプする部分を検出することによりフックの検知を行う．加えて HookFinder はマルウェアが利用する新しいフックの技術を解析できるように解析者に対して詳細なレポートを提供する．

2.2.3 動的解析を回避する方法とその対策方法

マルウェア作成者は動的解析を回避するためにエミュレータや仮想化環境の存在を検出する．先に取り上げた動的解析を利用する対策手法 [25–30] はエミュレータや仮想化環境を用いたシステムのため，マルウェア作成者がエミュレータや仮想化環境そのものを検知する方法を利用することにより，マルウェアはすぐに動作を終了したり，悪意ある振る舞いを行わず，害のない動作を示すことにより動的解析を逃れる．例えば，仮想化環境上で動作するマルウェアは，稼働中の OS が利用している仮想デバイスの名前を調べることにより自身が仮想化環境上で実行しているかどうかを推測できる [62, 63]．具体的には，Windows OS の場合はレジストリ内に “VMWare” の文字列が出現するかを調べることにより自身が VMware 上で動作しているかどうかを推測できる．また，特定の I/O ポートにアクセスして，その結果次第で判別する方法がある．マルウェアが “0x5658” の I/O ポートにアクセスにした際に例外が発生しなかった場合は VMWare 上で動作すると判断している．

マルウェアによる動的解析の回避に対処するために，防御側も様々な対策により対処している．Cobra [64] や Ether [65] は I/O ポートや仮想デバイスを調べる

ことにより，エミュレータや仮想化環境を検出するようなマルウェアであっても動的解析を行うことができるシステムである．Cobra や Ether は実ハードウェアとエミュレータや仮想化環境で実行する際の違いを吸収するために，ゲスト OS の命令毎にトラップを行い環境の違いを推測されるような情報を隠蔽することにより，ゲスト OS 上で実行しているマルウェアにあたかも自身が実ハードウェア上で実行しているように見せかける．しかしながら，Cobra や Ether はパフォーマンスの低下が著しいため，数多くのマルウェアを解析する状況には適さない．そこで， Gyung らや Balzarotti らは，このようなマルウェアの解析を行うのではなく，異なる 2 つの環境における振る舞いの違いを検出してエミュレータや仮想化環境上では悪意ある振る舞いを示さないマルウェアの検知を行うシステムを提案している [66, 67]．Gyung らや Balzarotti らの手法により，エミュレータや仮想化環境を識別して振る舞いを変えるマルウェアを適切に発見することができれば，そのようなマルウェアを他の解析手法を適用することにより，解析を逃れることを防ぐことができる．その結果，そのようなマルウェアに対する迅速な対応が期待できる．

マルウェアは悪意ある振る舞いを示すために stalling code を利用して動的解析を回避できることが知られている [68]．stalling code を利用する方法はエミュレータや仮想化環境を検知して，マルウェアの振る舞いを変更する方法ではなく，stalling code 中の処理のみでマルウェアの解析に費やせる時間を枯渇させることを目的としている．動的解析を行うシステムはマルウェアがシステムコールや API を実行すると，その名前や引数を保存するためオーバヘッドが生じる．この点を悪用して，マルウェアは悪意ある動作に入る前に本来の悪意ある振る舞いとは関係のない動作をループで大量に呼び出すことにより，実ハードウェア上で動作する場合は stalling code が数秒で終わるのに対して，解析を行うシステム上では stalling code を抜け出すのに数日掛かるような仕掛けを導入する．図 2.6 は stalling code をプログラムと通常のアプリケーションが，実ハードウェア，仮想化環境，エミュレータにどの程度の影響を与えるかを示すグラフである．MMX，浮動小数点演算，I/O，メモリに関する処理を多少することにより，エミュレータ環境において著しく処理が遅くなることが分かる．HASTEN [68] は stalling code に対処するために，振る舞いグラフを作成して，ループ内に存在する stalling code を検出した際にそれらに関するログの表示をやめることとや，stalling code のループから強制的に抜け出すことにより stalling code を含むマルウェアに対処している．

いずれの対策においても，動的解析を行う際はエミュレータや仮想化環境を利

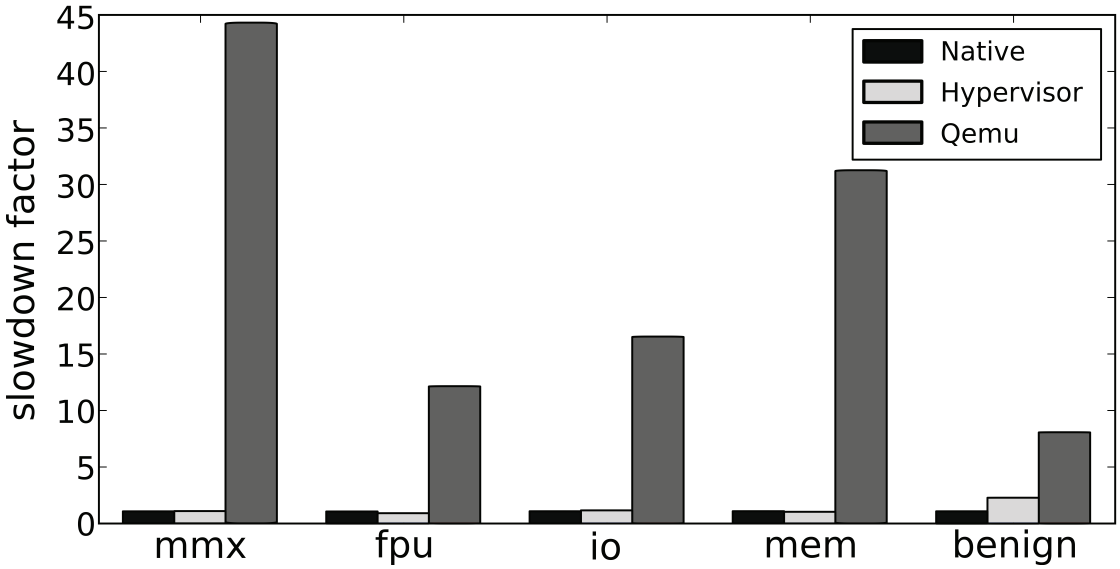

図 2.6: **Stalling code** が及ぼす影響．文献 [68]，287 ページ，図 2 より引用

用していることを考慮して，それらの環境上では悪意ある振る舞いを示さないことにより動的解析を回避している方法である．

2.3 関連研究のまとめ

関連研究を通じて，マルウェアに関する攻撃側と防御側の争いが現在も続いていることを確認した．防御側がシグネチャマッチングの対策を利用すると，攻撃側は亜種を作成することによりシグネチャマッチングの対策を困難にする．防御側が静的解析を適用することにより亜種に対応すると，攻撃側は難読化を利用して静的解析を回避する．防御側が動的解析を利用する方法を利用するようになると，エミュレータや仮想化環境を検出することにより動的解析を回避している．防御側はエミュレータや仮想化環境を検出して悪意ある振る舞いを示さないマルウェアへの対策を考案している．しかし，現状ではステルス性の高い，頻繁な動作を行わないマルウェアやシステムコールや API 列などから悪意ある振る舞いを推測することを困難にするマルウェアに十分に対処できていない．

第3章　マルウェアの特徴的な振る舞いを誘発するアプローチ

第 3 章では，本論文の核となる考え方である “マルウェアの特徴的振る舞いを誘発する” アプローチを提案する．このアプローチを導入することにより，ステルス性の高いマルウェアに対応することを目的としている．第 2 章で述べたように，防御側がシグネチャマッチングによるマルウェア対策，振る舞い検出によるマルウェア対策手法を利用するにつれて，攻撃側は防御側の対策を回避するために亜種や難読化，ステルス性を備えて防御側による対策を困難にしている．

ステルス性の高いマルウェアは悪意ある振る舞いをできる限り隠すように動作する．そのため，防御側はマルウェアの振る舞いを取得することができなくなり，1 つのマルウェアを長期的に利用できるようになり，マルウェアを利用した詐欺行為や個人情報を秘密裏に収集するといった犯罪行為を継続的かつ長期的に行うことができるようになる．ステルス性の高いマルウェアは主に，1) 頻繁な動作を避ける，2) 害のないソフトウェアの挙動を模倣する方法を利用する．

3.1　ステルス性の高いマルウェア

現在，ステルス性の高いマルウェアは動的解析による対策を逃れることができるため，単純にマルウェアを実行させてその振る舞いを抽出するのみではステルス性の高いマルウェアが持つ特徴的な振る舞いを正しく取得できない．その結果，ステルス性の高いマルウェアへの対策が遅れるため，ユーザが多大な損害を被ることになる．

頻繁な動作を避けるステルス性の高いマルウェアは単に動的解析を行うのみではその特徴的な振る舞いを取得することができない．この種類のマルウェアはそもそも悪意ある振る舞いを頻繁に示さないため，特徴的な振る舞いを取得するためにはマルウェアの振る舞いを活性化する必要がある．しかし，既存の対策では動的解析を行う際にマルウェアの振る舞いを活性化することを考慮していない．既

存の対策では，マルウェアが取得するデータフローの追跡やシステムコールや API 列から構成される制御フローを取得できることを前提に動作するためである．また，ステルス性の高いマルウェアに対して動的解析の回避を困難にするアプローチを適用しても，その効果は不十分である．なぜなら，ステルス性の高いマルウェアはエミュレータや仮想化環境を識別して，その振る舞いを変更することはなく，実ハードウェア上においても滅多に動作しない．

害のないソフトウェアの挙動を真似るステルス性の高いマルウェアも同様に単純な動的解析では特徴的な振る舞いを取得することができない．なぜなら，抽出できた振る舞い，つまり，API やシステムコール列は害のないソフトウェアの挙動と似るため，悪意ある振る舞いを含むかどうかの判別を行うことが困難となる．また，データフローや制御フローを利用した判別を行うことはできない．害のないソフトウェアの挙動を真似るマルウェアがユーザに偽の情報を提示して詐欺行為を行うような動作をする場合，動的解析から得られる振る舞いからそのような挙動を把握することができない．

3.2 特徴的な振る舞いを誘発するアプローチ

第 3.1 節で述べたように，ステルス性の高いマルウェアは自身の存在を巧妙に隠蔽して特徴的な動作を防御側に掴ませないように振る舞う．そのため，対策を行うためにはその特徴的な振る舞いを作為的に取得できる機構を用意する必要がある．本論文では，マルウェアの種類（例えばスパイウェアや偽アンチウイルスソフトなど）に応じて，種類ごとの特徴的な挙動を誘発するための入力を与える作為的な環境を用意する．ステルス性の高いマルウェアは自身の存在を隠蔽しつつ動作するため，単に動的解析を行ったとしてもその特徴的な動作を取得することができない場合がある．この問題に対処するために，ステルス性の高いマルウェアが動作せざるを得ない環境を意図的に用意することにより，ステルス性の高いマルウェアの挙動をわざと誘発して解析や検出を行う手法を導入する．

マルウェアの特徴的な振る舞いを誘発するためには，マルウェアの種類を考慮して適切な入力を与える必要がある．マルウェアの種類は多岐に渡り，その目的に応じて実現方法，起動のトリガとなる入力，悪意ある振る舞いが異なる．そのため，唯一の方法であらゆるマルウェアに対処できる方法は存在しない．しかしながら，振る舞いを誘発するアプローチそのものは特定のマルウェアに依存する

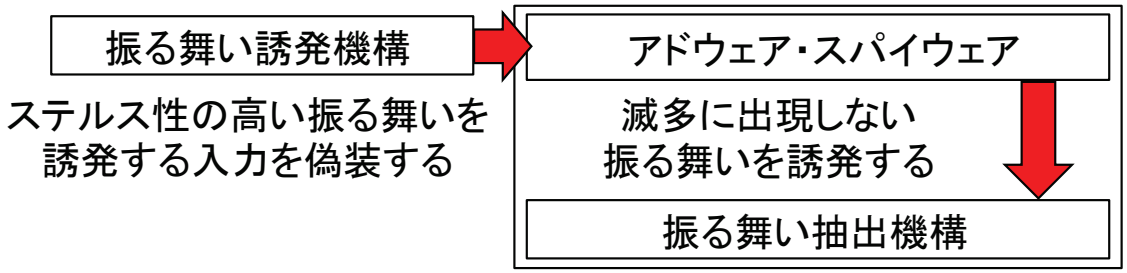

図 3.1: 頻繁な動作を避けるアドウェア，スパイウェアの挙動を誘発するアプローチ

方法ではなく，様々な種類のマルウェアに対応することができる手法である．

本論文では，実在するステルス性の高いマルウェアであるアドウェアやスパイウェア及び偽アンチウイルスソフトを対象に，特徴的な振る舞いを誘発するアプローチがステルス性の高いマルウェアに対して有効な手法であるかを示す．実際に，アドウェアやスパイウェアの中には頻繁な動作を避けて，その特徴的な挙動を掴ませないように動作する場合がある．また，偽アンチウイルスソフトは正規のアンチウイルスソフトの挙動を考慮しつつ動作することにより，API やシステムコール列を取得しても，その振る舞いが悪意ある動作を含むかどうかの推測を困難にする．

まず，頻繁な動作を避けるアドウェアやスパイウェアに対して，図 3.1 のように提案システムが捏造した入力を大量に与えることによりステルス性の高いアドウェアやスパイウェアを活性化させて滅多に動作しない挙動を誘発することができる．ステルス性の高いアドウェアやスパイウェアに感染した場合，これらははユーザの動作と連動して動作を行う．そのため，ユーザの動作を捏造した入力を利用してアドウェアやスパイウェアを騙し，その動作を誘発する．捏造した入力はユーザの操作を伴わないため，容易かつ大量にその入力をステルス性の高いアドウェアやスパイウェアに挿入することができる．その結果，短時間で頻繁に出現しない挙動を誘発することができるため，頻繁な動作を示さないステルス性の高いアドウェアやスパイウェアに効率良く対処できる．このように適切な入力偽装機構を利用することにより，特徴的な振る舞いを意図的に誘発することができるため，ステルス性の高いアドウェアやスパイウェアに対して素早く対処することが可能となる．

次に，正規のアンチウイルスソフトの挙動を模倣して動作するマルウェアである偽アンチウイルスソフトに対しては，正規のアンチウイルスソフトと偽アンチウイルスソフトの違いを誘発する入力を提案システムが与えることにより両者の判別を行う．例えば，正規のアンチウイルスソフトと偽アンチウイルスソフトの

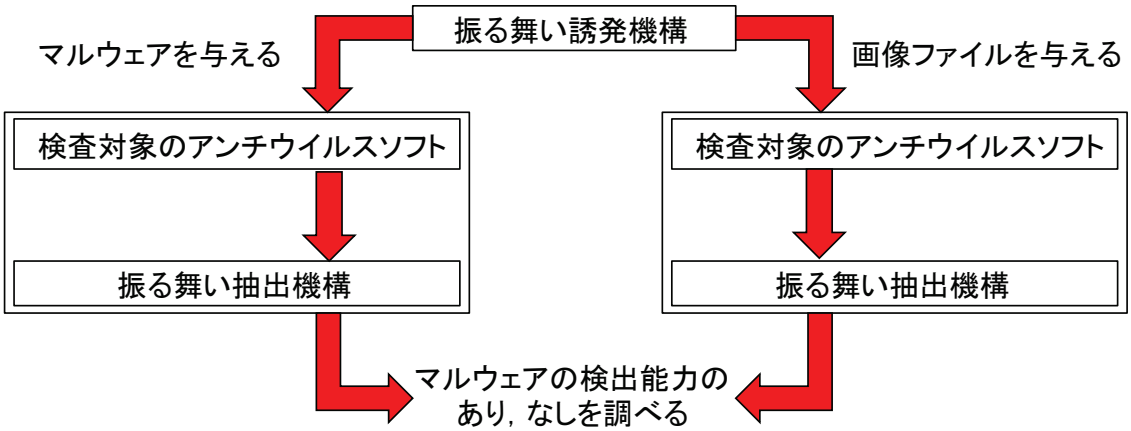

図 3.2: 検査対象のアンチウイルスソフトのマルウェア検出能力のあり，なしを誘発するアプローチ

違いはマルウェアの検出能力を備えているかどうかにある．そのため，アンチウイルスソフトらしき検査対象に対してマルウェアを与えた時の反応を確かめて両者の判別を行う．図 3.2 のように，検査対象のアンチウイルスソフトに対してマルウェアと害のないファイル (画像ファイルなど) をそれぞれ与えて，得られる振る舞いを比較することにより，マルウェアの検出能力のあり，なしを判別することができる．

第4章　アドウェア及びスパイウェアの解析手法

第 4 章では，ステルス性の高いアドウェアやスパイウェアの動作を誘発するために，ウェブブラウザの状態を示すイベントを偽装して，検査対象のアドウェアやスパイウェアにそれらを注入する機構である Blayzard について述べる．

Blayzard は次の 3 つの特徴を備えることにより，BHO ベースのアドウェアやスパイウェアに対処して解析を行う．

1. **IE と BHO の分離：** BHO は Dynamic Link Library (DLL) として IE プロセスに読み込まれるため，BHO が発行するシステムコールは IE が発行するシステムコールと区別することができない．そこで，BHO と IE プロセスを分離するために，Blayzard は両者を別々のプロセスに配置する．プロセスの分離によって，IE の影響を排除することができるため，Blayzard は BHO が発行するシステムコールのみを抽出することができる．また，IE と BHO を別プロセスに配置しているため，Blayzard は悪意ある BHO が IE のメモリを直接操作することを防ぐことができる．

2. **偽のイベントの生成，注入：** ステルス性を有するアドウェアを活性化させるために，Blayzard は偽のイベント列を生成・注入することで，意図的に解析対象のアドウェアを動作させて解析を行う．通常，イベントは IE の動作状況に応じて生成する．しかし，Blayzard は IE の動作を伴わずに偽のイベントを生成できるため，容易に大量のイベント列を生成することが可能となる．その結果，稀にしか動作しない BHO を強制的に動作させることができる．

   また，偽のイベント列を注入することによって，暗号化してある通信内容を推測することができる．通信を行うシステムコールの引数を記録すれば，解析対象のアドウェアが外部に送信する情報を取得することができる．しかし，通信内容が暗号化してある場合は，その内容の判別を行うことができない．そこで，注入するイベント列の内容を工夫することによって，Blayzard

では暗号化された通信内容の推測を支援するようになっている．たとえば，正しい情報（例：Windows のプロダクト ID）を含むイベント列と，偽の情報（例：偽の Windows のプロダクト ID）とを含むイベント列を注入し，解析対象のアドウェアが送信する通信内容の比較を行う．両者の通信内容が異なれば，プロダクト ID を送信している可能性が高い．

3. **動的解析：** Blayzard は BHO ベースのアドウェアやスパイウェアの振る舞いを抽出するために動的解析を行う．Blayzard が生成したイベント列を解析対象のアドウェアやスパイウェアに意図的に注入することで，実際にアドウェアやスパイウェアを動作させて，解析対象のアドウェアやスパイウェアが実行したコールバックメソッドやシステムコール列を抽出する．動的解析を用いる利点の 1 つは，暗号化や難読化に耐性を持つことである．そのため，暗号化や難読化が行われている場合であっても，暗号化や難読化の影響を受けずに解析を行うことができる．

Blayzard の有用性を示すために，Blayzard を実装して，マルウェア収集サイト [69, 70] などから 32 個の BHO ベースのアドウェアやスパイウェアを収集して，Blayzard による振る舞い解析を行った．その結果，収集した全てのアドウェアやスパイウェアの振る舞いを抽出することができた．特に，Blayzard の特徴である偽のイベントを大量に挿入することにより，ステルス性の高いアドウェアやスパイウェアであってもその振る舞いを誘発することができた．また，虚偽の情報を含むイベント列を注入することにより Windows のプロダクト ID を暗号化して送信する振る舞いを解析することができた．Blayzard の解析結果が正しいことを確認するために，商用のディスアセンブラである IDA Pro [53] を用いて検体の解析を行ったところ，Blayzard の解析結果は正しいことが確認できた．

4.1 BHO ベースのアドウェア及びスパイウェア

アドウェアやスパイウェアはユーザが訪れたウェブページの URL やブラウザが所有する Cookie などを収集し，外部のサーバに収集した情報を送信する．サイバー犯罪者は収集した情報を利用して効率的な広告配信を行ったり，第三者に売却して利益を得るために利用する．

第 4.1 節では，BHO ベースのアドウェアやスパイウェアの振る舞いを理解する

ために，BHO の動作概要を説明して，BHO を利用するアドウェア・スパイウェアの動作について説明する．

4.1.1 Browser Helper Object

BHO は IE のプロセスに読み込まれる DLL である．BHO はイベント駆動型のアドオンであり，Windows の Component Object Model [71]（COM）を利用する．IE は動作状況に合わせて様々なイベントを生成し，BHO に送信する．例えば，ブラウザがあるリンクにページ遷移を行う時，`BeforeNavigate2` のイベントが IE によって生成され，BHO に通知される．同様に，リンクへの遷移が完了した時，`NavigateComplete2` のイベントが，ページ遷移が完了し，取得したウェブページが表示できる時，`DocumentComplete` のイベントがそれぞれ生成される．IE によって生成されるイベントは `DWebBrowserEvents2` インターフェースに記載されているイベントである．大半のイベントは引数を伴って BHO に送信される．`BeforeNavigate2` のイベントは 7 つの引数を所有しており，ページ遷移先の URL，HTTP ヘッダや HTTP リクエスト内に含まれる POST データなどが含まれている．また `DocumentComplete` イベントは 2 つの引数を持ち，1 つはダウンロードしたファイルの URL である．

BHO を利用する利点はブラウザの挙動に応じて，その振る舞いを柔軟に変更できることである．例えば，ブラウザの状態やユーザの入力に応じてポップアップの表示を防いだり，自動入力フォームの機能を追加したり，マウスジェスチャの機能を付与するなど，追加できる機能は多岐に渡る．このような機能を実現するために，BHO はコールバックメソッドやシステムコールを利用する．

BHO はイベントを受信した時に実行されるイベントハンドラを登録している．BHO が IE の振る舞いを制御するために，イベントハンドラは IE によって公開されているコールバックメソッドを呼び出すことができる．IE は `Navigate` や `LocationURL` など，非常に多くのメソッドを公開している．`Navigate` メソッドは IE に引数で指定された URL にページ遷移を行わせる．`LocationURL` はブラウザに表示されているウェブページの URL を取得するメソッドである．

BHO はレジストリの値を取得するなどの目的で，IE と同じ権限でシステムコールを発行する．これは BHO が IE と同一プロセス上で動作するためである．結果として，BHO は IE がアクセスできる様々な情報を取得することができる．つまり，BHO が IE の所有する情報にアクセスする適切なインターフェースを持たな

くても，IE がその情報を取得するための権限を持つならば，BHO も同様にシステムコールを利用してその情報を取得することができる．例えば，IE は Windows のレジストリの値を取得するためのコールバックメソッドを公開していない．しかし，IE はシステムコールを利用してレジストリの値を取得できるため，BHO も IE の権限を利用してレジストリの値を取得できる．

このように，BHO はコールバックメソッドやシステムコールを利用することにより，あらゆる機能を実現することができる．このことはブラウザを利用して正当なサービスを提供しようとする開発者に対して有用な手段となる．しかしながら，マルウェア開発者に対しても悪意ある機能を実現させることを容易にする．加えて，BHO はブラウザと連動して動作するため，ブラウザの動作と BHO の動作の区別をすることを困難にさせる．その結果，BHO を利用するアドウェアやスパイウェアはステルス性が高くなり，極めて悪質なものとなり得る．

4.1.2 BHO ベースのアドウェア及びスパイウェアの動作

BHO ベースのアドウェアやスパイウェアの特徴は IE のイベントをトリガとして動作することである．イベントをトリガとして動作するため，BHO ベースのアドウェアやスパイウェアは頻繁に動作を行わず，IE の状態に変化がある時のみに動作することとなり，結果としてステルス性が高くなる．また，イベントは IE に変化がある際に BHO に対して送信することにより，BHO ベースのアドウェアやスパイウェアは IE と連動して動作することになる．そのため，BHO ベースのアドウェアやスパイウェアの挙動は IE の挙動に紛れ込み，BHO ベースのアドウェアやスパイウェアの存在に気付きにくい．また，解析を行った際にも IE の動作と BHO の動作を区別することが困難となる．

BHO ベースのアドウェアやスパイウェアは IE が BHO に対して提供する様々なコールバックメソッドから URL や Cookie を取得できるため，BHO を利用することによりアドウェアやスパイウェアの目的を達成しやすくなる．また BHO は IE の一部として動作するため，システムコールを利用することにより IE の権限でアクセスできる様々な情報を入手できる．

表 4.1: **Google** 検索を行う際に **`xsfer.dll`** が行う動作

入力した検索語	password
本来遷移すべき URL	http://www.google.com/search?hl=en&source=hp &q=password&aq=f&aqi=g10&aql=&oq=&gs_rfai=
`xsfer.dll` が指定した URL	http://synsynsyn.com/search.php?od=google.com &aid=A0003&hl=en&source=hp&q=password &aq=f&aqi=g10&aql=&oq=&gs_rfai=&url= http://www.google.com/search?hl=en[amp]source= hp[amp]q=password[amp]aq=f[amp]aqi=g10 [amp]aql=[amp]oq=[amp]gs_rfai=

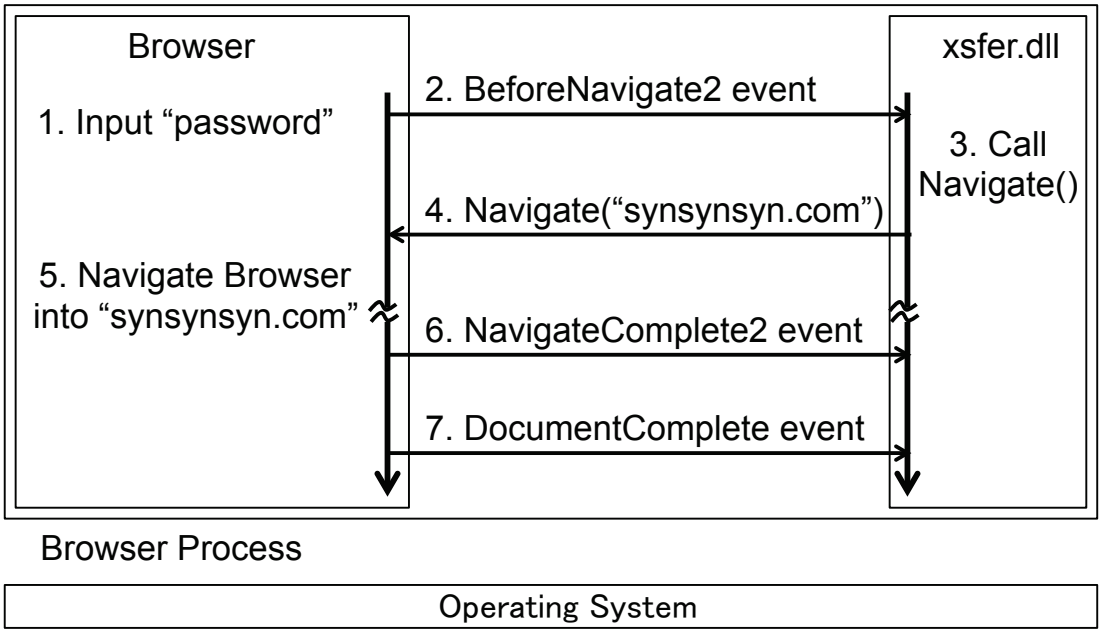

図 4.1: **Navigate** メソッドを利用した検索履歴の流出例

コールバックメソッドを利用する場合の動作例

図 4.1 は実在する BHO ベースのアドウェア・スパイウェアの動作を示している．図 4.1 にて取り上げている `xsfer.dll` の注目するべき動作は 2 つある．1 つ目はユーザが期待していないウェブページに遷移していること．2 つ目はコールバックメソッドに指定する URL に本来訪れるはずであった URL が埋め込まれていることである．`xsfer.dll` に感染したブラウザをユーザが使う場合について説明する．まず，ユーザが検索語としてブラウザに “password” と入力して検索を

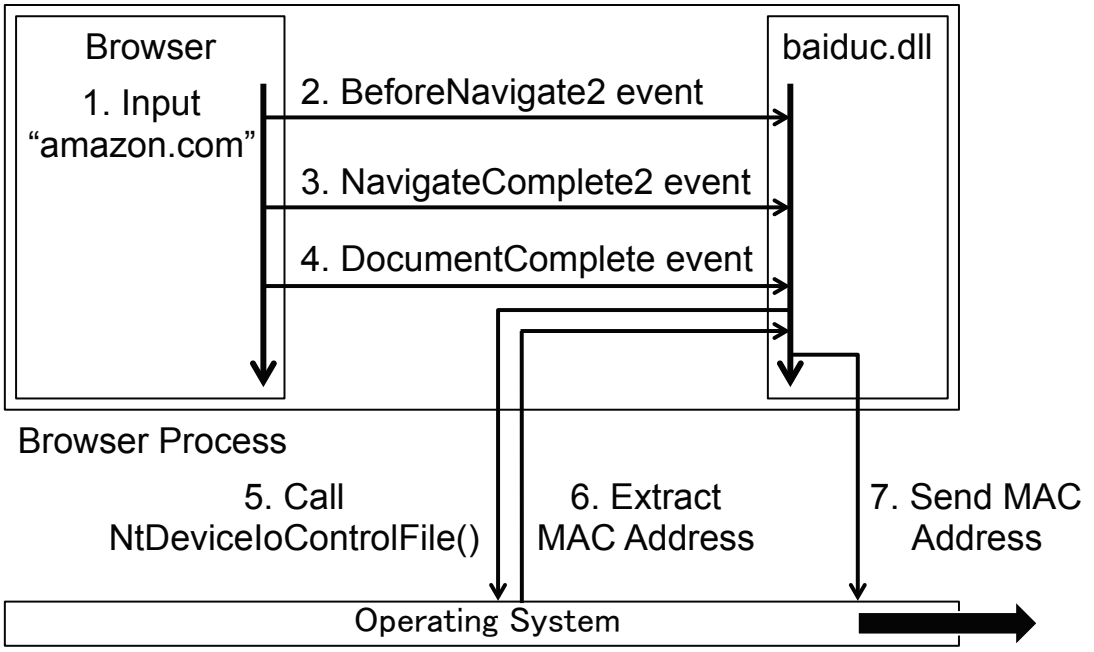

図 4.2: **MAC** アドレスの流出例

行う場合，ページ遷移が行われる前に，ブラウザは登録されている BHO に対して `BeforeNavigate2` のイベントを送信する．`xsfer.dll` がこのイベントを受信した時，`BeforeNavigate2` のイベントの引数にある URL を解析して，その検索が Google 検索を利用して行われているかを調べる．もし，Google 検索が行われていると判断した場合は，`xsfer.dll` はコールバックメソッドである `Navigate` メソッドを発行してブラウザを `synsynsyn.com` に遷移させる．この結果，ユーザは Google 検索の結果を表示するページに辿り着くことなく別のウェブページに移動することになる．加えて `xsfer.dll` は `synsynsyn.com` に遷移する際に，ユーザが本来遷移するはずであった URL を加えることにより，ユーザが訪れようとした URL の収集を同時に試みる．ユーザが入力した検索語が “password” の場合，“本来遷移するべき URL” と “実際に遷移した URL” はそれぞれ表 4.1 になる．表 4.1 内の “本来遷移すべき URL” は Google 検索を行った結果が表示される URL である．しかし，`xsfer.dll` が指定した URL は `synsynsyn.com` のドメインのページとなっている．加えて，その URL 内の “&url” 以降の値は “本来遷移すべき URL が記載されている”．

システムコールを利用する場合の動作例

図 4.2 も xsfer.dll と同様に実在する BHO ベースのアドウェア・スパイウェアの動作を示している．図 4.2 にて取り上げている baiduc.dll は MAC アドレスを外部に送信する検体である．baiduc.dll は DocumentComplete のイベントを受信したタイミングで Windows のシステムコールである NtDeviceIoControlFile を発行する．これにより，MAC アドレスを抽出してそれを外部に送信している．加えて DocumentComplete はそのイベント引数として，ウェブブラウザが訪れた URL を保持している．そのため，baiduc.dll は MAC アドレスと紐付けて URL を外部に送信している．

4.1.3 BHO ベースのアドウェア及びスパイウェアの解析に求められる要件

このように BHO ベースのアドウェア及びスパイウェアはブラウザからの送信されたイベントをトリガとして起動することが分かる．そのため，イベントを受信しない限り，その悪意ある振る舞いを示さない．そのため，イベントを作為的に挿入することができれば，その振る舞いを容易に誘発することができる．また，ブラウザのアドオンとして実装されている性質上，ブラウザの動作に紛れて BHO ベースのアドウェアやスパイウェアは動作することになる．BHO ベースのアドウェアやスパイウェアを効率的に解析するためには，ブラウザの影響を排除することが望ましい．

以上の点を踏まえると，BHO ベースのアドウェアやスパイウェアの解析を行う際には 3 つの要件を満たす必要がある．第 1 に，イベントを作為的に挿入する機構を用意することである．BHO ベースのアドウェアやスパイウェアは IE からのイベントを受信して動作するため，そのイベントを偽装して BHO ベースのアドウェアやスパイウェアに挿入することにより効率良く解析を行うことができる．加えて，イベントを偽装することにより，IE の挙動と関係なくイベントを挿入することができるため，大量のイベントを生成して BHO ベースのアドウェアやスパイウェアに挿入することにより，頻繁な動作を避けるアドウェアやスパイウェアの解析が可能となる．第 2 に，IE と BHO を分離することである．これは両者をプロセス単位で分離することにより，解析結果に IE の影響が及ばないようにするためである．その結果，BHO ベースのアドウェアやスパイウェアのみの振る舞いを容

易に抽出することができる．第 3 に，動的解析によるアプローチを採用する．動的解析を利用することにより難読化を施されたアドウェアやスパイウェアの解析が可能となるためである．

IE 以外のウェブブラウザでも BHO に相当するアドオンを利用することができる．Blayzard は IE と BHO に特化した実装になっているものの，他のブラウザ用のアドオンでも Blayzard と同等の機能を実現することが可能である．これは多くのブラウザが IE と同様にイベントやコールバックによるアドオンを提供しているからである．

4.2 Blayzard

Blayzard は BHO の動作を誘発するイベントを入力し，BHO ベースの悪意あるアドウェア及びスパイウェアの振る舞いを抽出する解析システムである．Blayzard はステルス性のあるアドウェア及びスパイウェアの振る舞いを誘発するために，単純に解析対象のアドウェアを監視するのではなく，偽のイベントを注入してその振る舞いを抽出する．Blayzard が生成する偽のイベントは，悪意あるアドウェアに偽のイベントであることを気付かれないように，ブラウザが送信する実際のイベント列と類似したものになるようにしている．これに加えて，偽のイベントは引数に与える URL を変更して，どのような情報が外部に流出したかを知る手がかりになるように生成している．以下に Blayzard の設計について説明する．

4.2.1 IE と BHO の分離

BHO ベースのアドウェア・スパイウェアは IE と同一プロセス上で動作するため，Blayzard は BHO ベースのアドウェア・スパイウェアと IE の振る舞いを区別することができない．例えば，システムコールを発行する場合，Blayzard はそのシステムコールが IE と BHO のどちらが発行したのかを区別できない．BHO のみの振る舞いを抽出するために，Blayzard は IE と BHO をそれぞれブラウザプロセスと BHO プロセスに配置する．プロセスの分離を行うことにより IE の影響を排除して BHO の振る舞いを調べることができる．

図 4.3 に Blayzard の構成を示す．BHO プロセスは BHO が独立したプロセスとして動作できるようにした実行環境である．BHO プロセスには Event Dispatcher モジュール，Callback Method Invoker モジュールが組み込まれている．IE プロ

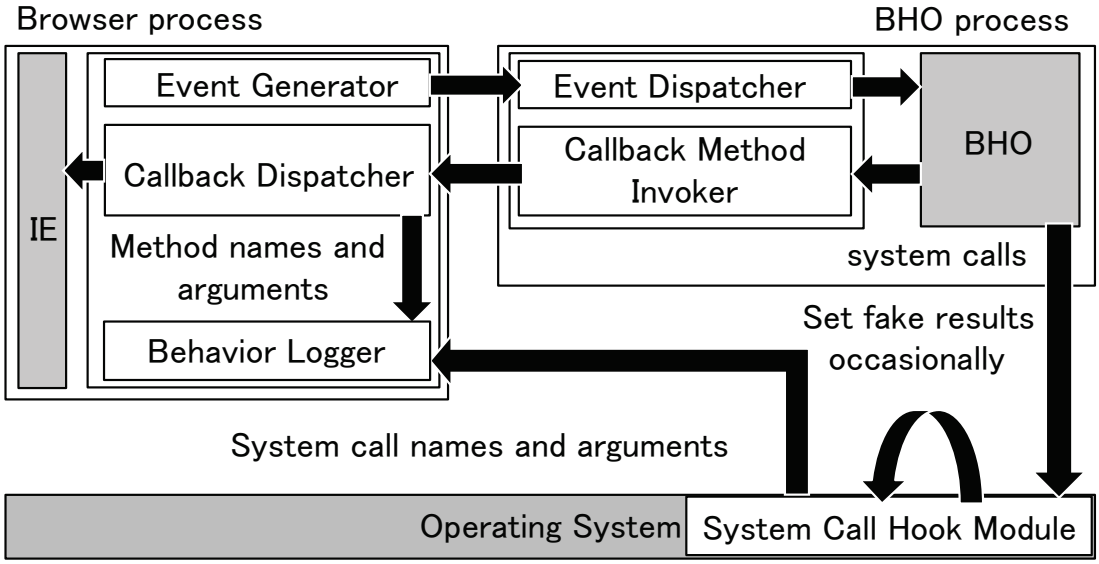

図 4.3: **Blayzard** の構成

セスには BHO として Event Generator モジュール, Callback Dispatcher モジュール，Behavior Logger モジュールが組み込まれている．BHO プロセス内の Event Dispatcher と Callback Method Invoker は，IE が BHO に対して提供するのと全く同じ API を提供しており，BHO プロセスに組み込まれたアドウェア及びスパイウェアからは IE と同等に見える．また，Event Generator，Callback Dispatcher，Behavior Logger は IE の BHO として組み込んであるため，IE からはこれらのモジュールが BHO として見える．Event Generator は偽のイベントを生成し，IPC（inter-process communication）を用いて BHO プロセスにイベントを送信する．BHO プロセス内の Event Dispatcher がそのイベントを受信し，解析対象のアドウェア及びスパイウェアに配信する．イベントを受信したアドウェア及びスパイウェアがコールバックメソッドを起動すると Callback Method Invoker が起動され，IPC により IE プロセス内の Callback Dispatcher に通知される．Callback Dispatcher は IE の提供するコールバックを実際に呼び出す．監視を行なっている BHO の振る舞いを抽出するために，受信したイベント列，コールバックメソッドやシステムコール列は Behavior Logger に送られ，その引数と共に記録される．悪意あるアドウェア及びスパイウェアによるログの改竄が起きないように Behavior Logger は BHO プロセスではなく IE プロセス内に配置している．

BHO ベースのアドウェアやスパイウェアの中には，自身が IE 上で動作しているのかどうかを識別する場合があるため，Blayzard が行う BHO プロセスの分離は BHO ベースのアドウェアやスパイウェアに気付かれないように行う必要がある．

現在，Blayzard は 2 つの対策を施すことにより，この問題に対処している．まず，解析対象のアドウェア及びスパイウェアが IE プロセス内で動作していると思わせるため，BHO プロセスの実行イメージ名は IE と同じ `iexplore.exe` としている．これは，自身が実行するプロセス名が `iexplore.exe` である時のみ，悪意ある動作を行うアドウェア及びスパイウェアが存在しているためである．この他にもプロセスサイズ，実行ファイルのパス名などから実行環境を推測する場合が考えられるものの，32 個の BHO ベースのアドウェアやスパイウェアはプロセス名以外の情報から実行環境を推測して異なる挙動を示す様子は確認できなかった．動作環境の識別を困難にするために，プロセス名以外の偽装についても今後考慮する必要がある．

次に，BHO ベースのアドウェア及びスパイウェアは IE が公開しているインターフェースを備えているかどうかにより，その挙動を変更する可能性がある．そのため， Callback Dispatcher と Callback Method Invoker は IE が備えている数多くのインターフェースを BHO に対して公開する必要がある．現在，Blayzard は BHO ベースのアドウェアやスパイウェアが頻繁に利用する数多くのインターフェースを公開しているため，BHO ベースのアドウェアやスパイウェアが十分なインターフェースを公開していないことによる環境の推測を行うことは難しい．

4.2.2 偽のイベントの生成と挿入

BHO ベースのアドウェアやスパイウェアの振る舞いを抽出するために，Blayzard は偽のイベントを生成し，解析対象のアドウェア及びスパイウェアに注入する．偽のイベントは解析対象のアドウェアやスパイウェアが通常通り動作するように生成する必要がある．これは，適切でないイベントをアドウェアやスパイウェアに送信することで，悪意あるアドウェアやスパイウェアが Blayzard による監視に気付いてしまい，悪意ある振る舞いを行わない可能性があるためである．

BHO ベースのアドウェアやスパイウェアを解析する際に，偽のイベントを利用することは有用である．偽のイベントによって BHO の動作を促すことができればブラウザを操作することなく解析対象の検体の挙動を知ることができる．また，ブラウザを操作することがなくなるため，BHO を動作させるためのイベントを大量に生成することが容易になる．同様のことをブラウザを操作することにより実現する場合，イベントを生成するために解析者がブラウザを使い続ける必要があり解析者の負担が増すことになる．加えて，滅多に挙動を示すことがないアドウェ

表 4.2: **Blayzard** が自動注入を行うイベント

BeforeNavigate	NavigateComplete	SetSecureLockIcon
StatusTextChange	Quit	DownloadComplete
DownloadBegin	NewWindow	CommandStateChange
ProgressChange	PropertyChange	TitleChange
FrameBeforeNavigate	FrameNavigateComplete	TitleIconChange
BeforeNavigate2	NewWindow2	DocumentComplete
NavigateComplete2	OnQuit	OnAddressbar
OnVisible	OnToolbar	OnMenubar
FileDownload	NavigateError	OnStatusbar
PrivacyImpactedStateChange	NewWindow3	SetPhishingFilterStatus
WindowStateChanged		

表 4.3: **Blayzard** が手動注入を行うイベント

OnFullScreen	OnTheaterMode	WindowSetResizable
WindowClosing	WindowSetLeft	WindowSetTop
WindowSetWidth	WindowSetHeight	ClientToHostWindow

アやスパイウェアを作動させることが容易となる．仮にページ遷移直前のイベントを示す `BeforeNavigate2` を 100 回受信したタイミングでアドウェアやスパイウェアが収集した情報を外部に送信するような場合であっても，わざわざブラウザを操作して 100 回のページ遷移を起こす必要はない．

しかしながら，BHO ベースのアドウェアやスパイウェアに対して無秩序にイベントを送信することは望ましくない．これはユーザがブラウザを操作する際の典型的な動作内に必ず呼び出されるイベントが存在するためである．例えば，ページ遷移直前のタイミングを示す `BeforeNavigate2` や遷移先のウェブページの表示が完了したことを示す `DocumentComplete` などである．仮にこれらのイベントの存在を考慮せず，無秩序にイベントを BHO ベースのアドウェアやスパイウェアに送信した場合，イベントの順序を監視しているアドウェアやスパイウェアは Blayzard の存在に気付く可能性がある．

そのため，Blayzard はブラウザが実際に発行するイベントの順番を真似て，偽のイベントをあたかも IE が生成したように作成している．実際，ブラウザはその実行状態に応じて様々なイベントを生成し BHO に送信する．Blayzard ではブラウザの実行状態として，初期化，ページ遷移，タブ及びウィンドウの生成，ブラウザの終了という 5 つの実行状態を用意し，それぞれの状態に応じて実際のブラウ

表 4.4: 未実装のイベント

WindowMove	WindowResize	WindowActivate
FrameNewWindow	ViewUpdate	

ザが送信するのと同等のイベント列を生成するようにしている．

Blayzard が偽装するイベント

Blayzard は IE の状態を表す全 45 種類のイベントに全て対応している．Blayzard では，IE が BHO に送信する 45 種類のイベントのうち仕様が公開されている 40 種類のイベントを送信することができる．Blayzard はブラウザの初期化やページ遷移，ブラウザの終了など，ユーザが通常利用する状況にて頻繁に観察される表 4.2 が示す 31 種類のイベントを自動注入することができる．表 4.3 が示す自動注入されることのない 9 (= 40 - 31) 種類のイベントは通常のブラウジング動作で発行されることがまれであるため，必要に応じて Blayzard の利用者が手動で注入する必要がある．表 4.4 の実装していない 5 種類のイベントは，そもそも IE で実装されていないイベントや，仕様の詳細が公開されていないイベントである [72,73]．これらのイベントを組み合わせることにより，様々なイベント列を作ることが可能であるものの，IE が発行する順序を考慮しないイベント列を BHO ベースのアドウェアやスパイウェアに挿入すると Blayzard の存在に気付かれる可能性があるため，典型的なブラウジングパターンを表すイベント列を利用すれば十分な解析を行うことができる．Blayzard は初期化，ページ遷移，タブ及びウィンドウの生成，ブラウザの終了の 5 つの実行状態の際に観察されるイベント列を自動注入することができ，実験では，全ての BHO ベースのアドウェアやスパイウェアの振る舞いを活性化させることができている．

例えば，ブラウザのページ遷移を模すために，ページ遷移を行う前に生成される `BeforeNavigate2` イベント，ページ遷移が完了したときに生成される `NavigateComplete2` イベント，ドキュメントの表示が完了したときに生成される `DocumentComplete` イベントなどを順に生成する．ただし，実際にブラウザがページ遷移を行うときには，タイトルの変化を通知する `TitleChange` イベントや，より詳細な状態変化を通知する `StatusTextExchange` イベントなどが図 4.4 のように通知される．こうしたイベントも適切に再現することによって，BHO ベースのアドウェア・スパイウェアから見れば，実際のウェブブラウザが動作しているように見える．

```
DISPID_BEFORENAVIGATE2 (250)
DISPID_BEFORENAVIGATE (100)
DISPID_DOWNLOADBEGIN (106) * 2
DISPID_DOWNLOADCOMPLETE (104) * 2
DISPID_STATUSTEXTCHANGE (102) *2
DISPID_DOWNLOADBEGIN (106) * 2
DISPID_PROGRESSCHANGE (108) * 2
DISPID_COMMANDSTATECHANGE (105) * 2
DISPID_PROPERTYCHANGE (112) * 4
DISPID_STATUSTEXTCHANGE (102) * 6
DISPID_COMMANDSTATECHANGE (105) * 2
DISPID_TITLECHANGE (113) * 2
DISPID_NAVIGATECOMPLETE2 (252)
DISPID_NAVIGATECOMPLETE (101)
DISPID_SETSECURELOCKICON (269)
DISPID_WINDOWSTATECHANGED (283) * 2
DISPID_SETSECURELOCKICON (269)
DISPID_SETPHISHINGFILTERSTATUS (282)
DISPID_COMMANDSTATECHANGE (105) * 4
DISPID_STATUSTEXTCHANGE (102) * 2
DISPID_SETPHISHINGFILTERSTATUS (282)
DISPID_COMMANDSTATECHANGE (105) * 2
DISPID_DOCUMENTCOMPLETE (259)
DISPID_TITLECHANGE (113) * 2
DISPID_STATUSTEXTCHANGE (102) * 6
DISPID_PROGRESSCHANGE (108) * 4
DISPID_DOWNLOADCOMPLETE (104) * 2
DISPID_PROGRESSCHANGE (108) * 2
```

図 4.4: ページ遷移の際にブラウザが発行するイベント列

既に第 4.1 節で述べたように IE が BHO に通知するイベントには引数を伴うものが多い．これらの引数の多くは毎回同じ値が設定されていることが多いため，Blayzard もその値を設定するようにしている．しかし，例えば遷移先の URL などを引数として伴うイベントの場合，Blayzard のほうで適切な値を設定する必要がある．現在の Blayzard の実装ではあらかじめ定めておいたいくつかの値から選択した値を設定する．いくつかの値から選択することで，引数が特定の値の場合に

```
HRESULT Invoke(
    DISPID  dispIdMember,
    REFIID  riid,
    LCID  lcid,
    WORD  wFlags,
    DISPPARAMS FAR*  pDispParams,
    VARIANT FAR*  pVarResult,
    EXCEPINFO FAR*  pExcepInfo,
    unsigned int FAR*  puArgErr
);
```

図 4.5: **Invoke** メソッドの定義

のみ動作するようなアドウェア・スパイウェアでも活性化できる可能性が高まる．実際，第 4.3 節の実験では，Google など特定のサイトを訪問したときにのみ活性化するアドウェア・スパイウェアがあった．より現実的なイベント列を生成するためには，IE の操作履歴等を多数集め，その履歴に沿った形でイベント列を生成するのが望ましい．

Blayzard ではステルス性の高いアドウェア・スパイウェアを活性化させるため，ブラウザの初期化から終了までの一連のイベントを数回から百数十回に渡って注入するようにしている．このようにすることで，何回かに 1 度だけ活性化するアドウェア・スパイウェアであっても活性化させることができる．実際に，ウェブの閲覧履歴を収集するあるスパイウェアはコールバックメソッドを利用して URL を取得した後にファイルに保存して，一定量そのログを書き溜めた後に，その情報を外部に送信するものがあった．このようなアドウェア・スパイウェアであっても Blayzard は大量のイベントを挿入することにより，その動作を揺さ振り出すことが可能となる．また，ブラウザの初期化から終了までのイベント注入を繰り返す際，イベントに伴う引数は毎回違った値を選択するようにしている．これは引数が特定の値の時のみ活性化するアドウェア・スパイウェアでも活性化させるためである．なお，このようなアドウェア・スパイウェアの場合，Blayzard を用いても活性化できないことがある．それについては，後の章で議論する．

```
typedef struct tagDISPPARAMS {
    VARIANTARG *rgvarg;
    DISPID *rgdispidNamedArgs;
    UINT cArgs;
    UINT cNamedArgs;
} DISPPARAMS;
```

図 4.6: **DISPPARAMS** 構造体の定義

Invoke メソッドによるイベント注入

IE は Invoke メソッドを利用して，様々なイベントを BHO に送信する．そのため，偽のイベントを BHO に注入する際には Invoke メソッドの引数に適切な値を与える必要がある．図 4.5 は Invoke メソッドの定義である．Blayzard は Invoke メソッドの引数のうち dispIdMember と PDispParams を偽装することにより偽のイベント注入を行う． 引数の dispIdMember はイベント名を表す整数を指定することにより Invoke が送信するイベントの種類を指定することができる．引数の PDispParams は各イベントの引数に関する情報を格納した構造体へのポインタを指定する．その構造体の定義は図 4.6 である．Blayzard はイベントの引数が格納されている配列である rgvarg とその個数を指定する cArgs を偽装している．Blayzard は注入するイベントに応じて引数に指定する値とその数を適切に与えるために，45 種類全てのイベントに関する引数の数と与えるべき引数を調べ，どのような BHO に対して偽のイベントを注入しても正しく動作するようにしている．より精巧なイベントを生成して注入を行うためには他の引数についても，注意深く設定を行う必要がある．しかし，実験において他の引数による動作への影響は確認できなかったため，Blayzard はイベントを偽装する際の重要な引数を適切に設定できている．

4.2.3 BHO の振る舞いの抽出

Blayzard はアドウェア及びスパイウェアの振る舞いとして，解析対象のアドウェア及びスパイウェアが受信したイベント列，起動したコールバックおよびシステムコールとその引数を記録する．図 4.3 に示したように，IE が送信するイベント列，起動されたコールバックとその引数は Event Generator, Callback Dispatcher で

取得する．現在，700 以上のコールバック関数に対応しており，BHO が利用するコールバックメソッドを抽出するのに十分である．BHO プロセスが実行したシステムコール列は，オペレーティングシステムに組み込んだシステムコールのフック・モジュールによってフックされる．このフック・モジュールは System Service Descriptor Table (SSDT) のエントリを書き換えることにより，システムコールのフックを行う．このフック・モジュールはシステムコールとその引数を IE プロセス内の Behavior Logger に通知するだけではなく，特定のシステムコールについては適宜，虚偽の返値を返す．

BHO ベースのアドウェア及びスパイウェアが別のプロセスを立ち上げ，アドウェア及びスパイウェアが取得した情報をそのプロセスを使って外部に送信する場合がある．Blayzard では新しいプロセスを作成する `NtOpenProcess`，`NtCreateProcess` をフックし，BHO プロセスが起動したプロセスについてもシステムコールのフックを行う．このようにすることで，BHO プロセスの振る舞いを追跡することが可能となる．

第 4.1 節で述べたように BHO ベースのアドウェアやスパイウェアは IE と同じ権限でシステムコールを実行することができる．そのため，BHO ベースのアドウェアやスパイウェアはシステムコールを利用して Windows のプロダクト ID やシステム情報など様々な情報を取得することができる．BHO ベースのアドウェアやスパイウェアの中にはシステムコールから得られる情報に暗号化を行って外部に送信する可能性がある．このような場合に対処するため，Blayzard では特定のシステムコールの返値を改竄するようにしている．例えば，レジストリの値を取得するシステムコールを実行した場合，数回に 1 度，虚偽の値を返すようにしている．この工夫により，実験で利用した `xml2u32h.dll` の暗号化された文字列に Windows のプロダクト ID の内容が含まれていることを推測することができている．

コールバック・メソッドのフック

Blayzard は 700 以上のコールバックメソッドを抽出することができ，BHO が利用するコールバックメソッドに十分に対応できる．各コールバックメソッドを所有するインターフェースは表 4.5 に示す通りであり，文献 [74] に記載されているインターフェースに加えて，更に数種類のインターフェースを加えており，BHO が発行する大半のコールバックメソッドに対応することができる．

表 4.5: **Blayzard** が抽出できるコールバック・メソッドを所有するインターフェース

IWebBrowser	IWebBrowser2	IWebBrowserApp
IOleWindow	IOleCommandTarget	IInputObjectSite
IInputObject	IOmNavigator	ITravelLogStg
IContextMenu	IObjectWithSite	IDispatch
IConnectionPoint	IHTMLDocument	IHTMLDocument2
IHTMLDocument3	IHTMLDocument4	IHTMLDocument5
IShellBrowser	IOleContainer	IHTMLElement
IHTMLElement2	IHTMLElement3	IHTMLElement4
IHTMLLocation	ICustomDoc	IHTMLFramesCollection
IHTMLFramesCollection2	IHTMLElementCollection	IHTMLElementCollection2
IHTMLElementCollection3	IHTMLEventObj	IHTMLEventObj2
IHTMLEventObj3	IHTMLEventObj4	IHTMLWindow2
IHTMLWindow3	IHTMLWindow4	IHTMLWindow5

表 4.6: **Blayzard** が監視するシステムコール

NtOpenFile	NtCreateFile	NtWriteFile
NtReadFile	NtClose	NtDeviceIoControlFile
NtOpenKey	NtCreateKey	NtQueryValueKey
NtCloseKey	NtOpenProcess	NtCreateProcess

システムコールのフック

Blayzard はアドウェアやスパイウェアがシステムコールを利用して所望の情報を得る動作を監視するために，ファイルアクセス，ソケット関連，レジストリ関連，プロセス生成関連のシステムコールを監視する．表 4.6 は Blayzard が監視するシステムコールの一覧である．ファイルアクセス，ソケット関連，レジストリ関連のシステムコールを監視は解析対象のアドウェアやスパイウェアがプロダクト ID やパスワードなどの情報を収集して，その情報を外部に送信する挙動を知るために十分である．加えて，プロセス関連のシステムコールは BHO ベースのアドウェアやスパイウェアが別プロセスを生成して，そのプロセスに悪意ある振る舞いを行わせる際の振る舞いを追跡する場合に対応できる．

ソケット関連のシステムコールについて Windows は専用のシステムコールを用意せず，代わりに `NtDeviceIoControlFile` を利用している．`NtDeviceIoControlFile` は TCP や UDP，またデータの送受信に関わらずあらゆるソケット関連の処理を行うシステムコールとなっている．そ

```
NTSTATUS WINAPI NtDeviceIoControlFile(
    __in   HANDLE FileHandle,
    __in   HANDLE Event,
    __in   PIO_APC_ROUTINE ApcRoutine,
    __in   PVOID ApcContext,
    __out  PIO_STATUS_BLOCK IoStatusBlock,
    __in   ULONG IoControlCode,
    __in   PVOID InputBuffer,
    __in   ULONG InputBufferLength,
    __out  PVOID OutputBuffer,
    __in   ULONG OutputBufferLength
);
```

図 4.7: **NtDeviceIoControlFile** の定義

のため，アドウェアやスパイウェアによるデータの送受信を監視するためには NtDeviceIoControlFile を監視して適切なタイミングで情報を取得する必要がある．NtDeviceIoControlFile の定義を図 4.7 を示す．NtDeviceIoControlFile がデータの送受信を行うタイミングは IoControlCode を調べることにより知ることができる．IoControlCode が 0x1201F の場合，NtDeviceIoControlFile は TCP を利用したデータの送信を行っていることを表す．同様に，0x12017 の場合は TCP を利用したデータの受信を行っていることを表す．そのため，IoControlCode の値が 0x1201F と 0x12017 の場合は InputBuffer を調べることにより，BHO がどのようなデータを送受信しているのかを抽出することができる．

現在，Blayzard は TCP 通信のデータの送受信のみ監視している．なぜならブラウザは HTTP プロトコルを利用してデータを送受信しており，BHO ベースのアドウェアやスパイウェアがブラウザの挙動に紛れて取得したデータを外部に送信する場合は HTTP プロトコルを利用することが予想できるためである．そのため，Blayzard が TCP 通信を監視できれば大多数の BHO ベースのアドウェアやスパイウェアに対処できる．UDP など他のプロトコルをサポートする場合は，別途 IoControlCode の値を監視して，実際に送受信されるデータを取得する必要がある．

InputBuffer は IoControlCode の値に応じて適切な構造体を利用してデータを格納する．TCP を用いたデータの送受信の際は AFD_SEND_INFO と

```
typedef struct _AFD_SEND_INFO {
    PAFD_WSABUF BufferArray;
    ULONG BufferCount;
    ULONG AfdFlags;
    ULONGTdiFlags;
} AFD_SEND_INFO, *PAFD_SEND_INFO;
```

図 4.8: **AFD_SEND_INFO** 構造体の定義

```
typedef struct _AFD_WSABUF {
    UINT len;
    PCHAR buf;
} AFD_WSABUF, *PAFD_WSABUF
```

図 4.9: **AFD_WSABUF** 構造体の定義

AFD_RECV_INFO 構造体を利用して InputBuffer 内のデータを正しく取得することができる．なお，AFD_SEND_INFO と AFD_RECV_INFO は同一の構造体である．その定義は図 4.8 に示す通りである．また，AFD_WSABUF の構造体は図 4.9 となっている．AFD_WSABUF 実際に送受信されるデータとそのデータサイズを格納する構造体である．

4.2.4 Blayzard の制限

Blayzard では適切なイベントを解析対象のアドウェア及びスパイウェアに注入することによってそれらの振る舞い解析を行っている．そのため，解析対象のアドウェア及びスパイウェアに与えるイベントが不適切であると，正しくその振る舞いを解析することができない．たとえば，イベントの引数が特定の値の時のみ活性化するタイプの場合，その特定の値を注入することができなければ対象の検体を活性化させることができず，振る舞い解析を行うことはできない．たとえば，特定の URL を訪問したときだけ活性化するアドウェアやスパイウェアの場合，その URL をイベントの引数として注入する必要がある．

Blayzard の現在の実装では，イベントの引数としてあらかじめ定めたいくつかの値のみを利用している．たとえば，イベントの引数として使われる URL には google.com, yahoo.com 等を用意している．第 4.3 節でも示すように，多くの

ユーザが訪問する URL を用意しておくだけでも十分に効果的である．しかし，あらゆる URL を事前に準備しておくことは現実的ではなく，稀にしか訪問されることのない URL に対してのみ活性化するアドウェア及びスパイウェアについては Blayzard での解析には適さない．このような場合，Fuzzing や静的解析を援用することが効果的であると考えられる．

また，ブラウザが表示しているウェブページの内容によって動作を変えるアドウェアやスパイウェアが存在する場合，Blayzard では適切な解析結果を得ることができない．BHO からウェブページの内容を取得するためには，`get_innerHTML` というコールバックメソッドを呼び出す．このようなタイプの解析を行うためには `get_innerHTML` の返値としてさまざまな内容を返さなければならない．しかし，Blayzard を用いて `get_innerHTML` が呼び出されていることは解析できるため，それを手がかりに静的解析などを援用することにより，より詳細な解析結果を得ることができる．

静的解析と動的解析の網羅性と簡易性はトレードオフの関係にある．Blayzard は動的解析を行っているため，全ての実行経路を網羅することは難しい．静的解析を行えば理論上は全ての実行経路を網羅できる．しかし，アドウェアやスパイウェアのバイナリが暗号化や難読化を行っている場合，IDA Pro などを用いた静的解析によって暗号化や難読化を行ったアドウェア及びスパイウェアの動作を理解するためには相応の技量と時間を必要とする．一方，動的解析を行うとバイナリの暗号化や難読化によらず，実行時の振る舞いを抽出することができる．

4.3 実験

実在するアドウェアやスパイウェアとよく利用される BHO を対象に，Blayzard の解析結果を示す．評価に用いるアドウェアやスパイウェアは表 4.7 に示す 32 個を Offensive Computing [69] や VX Heavens [70] などのマルウェア収集サイトから収集した．また，Google Toolbar などの表 4.8 で示す，よく利用される BHO を 10 個収集した．実験は Windows XP SP3 上の IE 7 (ver. 7.0.5730.13) と IE 8 (ver. 8.0.6001.18702) を用いた．また，意図的にアドウェアやスパイウェアに感染した状態で実験を行うために仮想化環境である VMware Fusion 3.1.1 上で実験を行った．仮想化環境を利用することにより感染前の状態に戻すことが容易となる．

表 4.7: 実験で用いたアドウェアとスパイウェア

xsfer.dll	autoex.dll	C Java Object
vvengygavnorr.dll	ini.dll	nada64.dll
navfilter.dll	iefltr.dll	nvgf.dll
tasdgim.dll	dhofozr.dll	sdsheol.dll
gofpa.dll	uslpazhfbcbafrssa.dll	skype Helper Object
BhoApp Class	IeHelperEx.dll	msfacat32.dll
ylsvevnbhjdaj.dll	jwixfukqsgxwy.dll	acrobat.dll
UCMTSAIE.dll	BrowserAccelerator.dll	Platrium.dll
cpush.dll	marwin32.dll	baiduc.dll
mqodcuwucedvr.dll	QQMenu	LsVpd
xml2u32h.dll	mws31209.dll	

表 4.8: 実験で用いたよく利用される **BHO**

Google Toolbar	Earth link
Yahoo! Toolbar	Bing Toolbar
MicroGuarden	Spybot S&D
Super Ad Blocker	Alexa Toolbar
Spyware Guard	MoreGoogle

4.3.1 解析を行ったアドウェア及びスパイウェアの振る舞い

表 4.9 に実験に用いたアドウェア及びスパイウェアとその振る舞いの一覧を示す．表 4.9 に示したように，Blayzard を用いて全てのアドウェアやスパイウェアの振る舞い解析を行うことができた．動作確認のために商用のディスアセンブラである IDA Pro を利用して検証を行ったところ，Blayzard は実験で利用した全ての BHO ベースのアドウェアやスパイウェアの特徴的な振る舞いを抽出できていることが分かった．一部のステルス性の高いアドウェアやスパイウェアを除いて，Blayzard による 1 回のイベント列 (例えば，ページ遷移時に発行されるイベント列) の注入により特徴的な振る舞いを示した．しかし，`navfilter.dll` や `autoex.dll` は特徴的な振る舞いを抽出するために，繰り返しイベント列を注入する必要があった．特に，`autoex.dll` は 10,000 回のページ遷移に関するイベント列を注入する必要があった．同様のことを Blayzard を用いずに実現する場合，これはブラウザを利用して 10,000 ページに訪れる必要がある．このようなことから，Blayzard はステルス性の高いアドウェアやスパイウェアに対しても効果的に特徴的な振る舞いを抽出できるといえる．

```
GET /qinfr2?|検索語|暗号化文字列|検索エンジンを表す数字 HTTP/1.1
```

図 4.10: **xml2u32h.dll** が情報を外部に送信するために利用する **GET** メソッド

この解析結果からわかるように，アドウェアやスパイウェアの振る舞いは，URL を流出しつつページ遷移を行う，広告表示のためのスクリプトやテキストの挿入，Windows のプロダクト ID や Cookie の流出など，多岐に渡る．なお，IE 7 と IE 8 のブラウザのバージョンの違いによる解析結果の違いは確認できなかった．以下，Blayzard を利用して特にステルス性の高い振る舞いを示した 5 つの検体の詳細な解析結果を示す．

xml2u32h.dll の振る舞い

このスパイウェアは Google などの検索エンジンに入力した検索語と Windows のプロダクト ID を外部に送信する．これらの情報を外部に送信するために，xml2u32h.dll は HTTP の GET メソッドを用いている．そのメッセージフォーマットは 図 4.10 の通りである．

Blayzard がページ遷移直前の状態を表す BeforeNavigate2 のイベントを xml2u32h.dll に注入すると，このスパイウェアは BeforeNavigate2 のイベント引数にある URL を調べて，その内容に応じて適宜外部に送信するメッセージの内容を変更する．具体的には，特定の検索エンジンを使った際に送信されるメッセージと BeforeNavigate2 のイベントを受信するごとに送信されるメッセージの 2 つがある．

Blayzard が Google，Yahoo!，MSN の検索エンジンを利用した際の検索結果が表示される URL を BeforeNavigate2 のイベント引数として xml2u32h.dll に注入した時に得られる結果は表 4.10 のようになる．Blayzard は BeforeNavigate2 のイベント引数として Google などのよく知られている URL を与えるようになっているため，このような振る舞いを抽出することができている．表 4.10 では，検索エンジンを利用した際の検索語，暗号化された Windows のプロダクト ID，暗号化された乱数，検索エンジンの種類を表す数字をそれぞれ表示している．

図 4.10 の “検索語” の部分は Blayzard が注入するイベント引数を xml2u32h.dll が解析することにより取得している．例えば，Google 検索

表 4.9: 評価を行ったアドウェア，スパイウェアとその振る舞い

名前	抽出できた振る舞い
xsfer.dll	`Navigate` を用いて URL を流出させる
autoex.dll	`Navigate` を用いて URL を流出させる
C Java Object	アダルトサイトにアクセスする
ini.dll	`write` を用いてスクリプトを挿入する
nada64.dll	`write` を用いてスクリプトを挿入する
navfilter.dll	`write` を用いてスクリプトを挿入する
iefltr.dll	`write` を用いてスクリプトを挿入する
nvgf.dll	`write` を用いてスクリプトを挿入する
tasdgim.dll	`write` を用いてスクリプトを挿入する
dhofozr.dll	`write` を用いてテキストを挿入する
sdsheol.dll	`write` を用いてテキストを挿入する
gofpa.dll	`write` を用いてテキストを挿入する
uslpazhfbcbafrssa.dll	`showBrowserBar` を用いて検索語を外部に送信する
skype Helper Object	定期的にブラウザの起動情報を送信する
BhoApp Class	Cookie と URL を流出する
IeHelperEx.dll	レジストリに値を追加する
msfacat32.dll	警告メッセージを読み込む
ylsvevnbhjdaj.dll	HTML ファイルをダウンロードする
jwixfukqsgxwy.dll	テキストファイルをダウンロードする
acrobat.dll	コンピュータ名を流出させる
UCMTSAIE.dll	URL を収集する
BrowserAccelerator.dll	URL 流出する
Platrium.dll	URL を流出する
cpush.dll	広告を表示する
marwin32.dll	広告を表示する
baiduc.dll	広告を表示する
mqodcuwucedvr.dll	`get_innerHTML` と `put_innerHTML` を呼び出す
QQMenu	イベントを受信後，表示しているウィンドウを廃棄する
LsVpd	`get_cookie` を用いて Cookie を収集する
xml2u32h.dll	検索語とプロダクト ID を外部に送信する
mws31209.dll	検索語とプロダクト ID を外部に送信する

を利用して “password” と入力する場合，その検索結果を表示するウェブページの URL はユーザが入力した検索語，つまり “password” を含んでいる．Blayzard がこの URL を偽のイベント引数として与えた場合は図 4.10 の “検索語” の部分が

表 4.10: **xml2u32h.dll** が検索エンジンの表示結果を表す **URL** を検出した際に送信されるメッセージ

検索語	暗号化したプロダクト ID	暗号化した乱数	種類
password	VGNzY0OTQtNjQwLT Q5MzIxODEtMjMyMzI	5NDkzMTk3NDc1	1 (Google)
car+insurance	VGNzY0OTQtNjQwLT Q5MzIxODEtMjMyMzI	3MjU4NTcyNjkx	2 (Yahoo!)
car+insurance	VGNzY0OTQtNjQwLT Q5MzIxODEtMjMyMzI	4Mjg3NzM0NDEx	3 (MSN)

“password” となる．

図 4.10 の “暗号化文字列” の一部はプロダクト ID を含んでいることが Blayzard を利用することにより分かった．暗号化された文字列から Windows のプロダクト ID の存在を推測することができた理由は，`xml2u32h.dll` がプロダクト ID を取得するために発行する `NtQueryValueKey` の引数に保持している値を意図的に変更したためである．具体的には，`NtQueryValueKey` の引数の 1 つである `ValueName` の値が “Product ID” の場合に，もう 1 つの引数である `KeyValueInformation` が保持している末尾の値を変更した (例えば，0 から 1 に変更)．表 4.10 では，実験環境から得られる正しいプロダクト ID を `xml2u32h.dll` に返した際の結果を表示しているため，いずれも同一の結果が表示されている．もちろん，Blayzard が捏造した結果を返す場合は，暗号化された文字列は表中の結果と異なったものとなる．しかしながら Blayzard は表 4.10 の “暗号化した乱数” の部分がどのような情報に依存しているのかを判別することはできなかった．

最後に，利用した検索エンジンの種類に応じて，異なる数が外部に送信されるメッセージの末尾に付与されることが分かった．実際に Google，Yahoo!，MSN の検索結果に関する URL を Blayzard が注入すると，`xml2u32h.dll` はそれぞれ 1, 2, 3 の値を末尾に付与した．

注入するイベント引数に関わらず，`BeforeNavigate2` のイベントを受信した時に外部に送信されるメッセージは表 4.11 に示すとおりである．この場合，検索語の部分は “userinit” と書き込まれている．また暗号化された文字列の部分は検索

表 4.11: **xml2u32h.dll** が **BeforeNavigate2** を受信したときに外部に送信されるメッセージ

検索語	暗号化したプロダクト ID	暗号化した乱数	種類
userinit	VGNzY0OTQtNjQwLT Q5MzIxODEtMjMyMzI	xNzk3MzE5Mzc5	none

エンジンの結果を示す URL をイベント引数として与えた場合のときと同様に，プロダクト ID と乱数が付与される．種類を表す部分には何も記載されない．

Blayzard による振る舞い解析の結果を検証するために，商用のディスアセンブラである IDA Pro [53] を利用して確認を行った．`BeforeNavigate2` メソッドの引数から URL, 検索語を得ていること，検索エンジンによって 1, 2, 3 の数値を選んでいること，暗号化された文字列は Windows のプロダクト ID を含んでいることが確認できた．また，Blayzard を利用するのみでは分からなかった暗号化された文字列の一部はシステム時間を種とした乱数が付与されていることが IDA Pro を利用した解析結果より判明した．

`BhoApp Class` の振る舞い

`BhoApp Class` の特徴的な振る舞いは大きく分けて 2 つある．1 つはコールバックメソッドを利用して IE が保持している Cookie と URL を取得すること．もう 1 つは得られた Cookie と URL を一定量蓄えた後に外部に送信していることである．

`BhoApp Class` が Cookie と URL を取得していることは Blayzard が `NavigateComplete2` のイベントを注入するときにコールバックメソッドである `get_cookie` と `get_LocationURL` をそれぞれ呼び出していることから推察できる．また，システムコールである `NtCreateFile` と `NtWriteFile` を監視した結果，取得した Cookie と URL は `system32` ディレクトリに `perfz9368.dat` というファイルを作成して暗号化されて保存されることが分かった．図 4.11 は `NtWriteFile` を監視した際に得られた文字列である．図 4.11 は 2 種類の暗号化した文字列を表示している．上部の暗号化された文字列は `get_cookie` と `get_LocationURL` の結果を `BhoApp Class` に渡さなかった場合に得られた文字列である．下部の暗号化された文字列は `get_cookie` と `get_LocationURL` の結果を `BhoApp Class` に渡した場合に得られる文字列である．両者の違いは

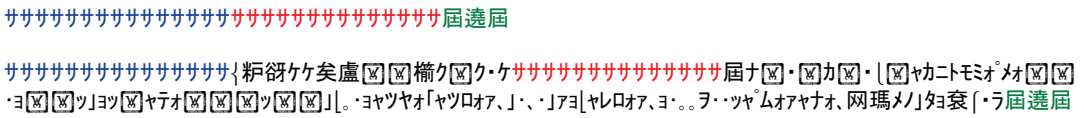

図 4.11: **BhoApp Class** が暗号化して保存するメッセージの例

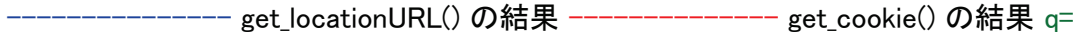

図 4.12: **BhoApp Class** が送信するメッセージを復号した結果

暗号化された黒文字の部分があるかどうかである．そのため，暗号化された文字列の一部は get_cookie と get_LocationURL の結果に依存することが Blayzard を利用することにより分かった．

BhoApp Class は perfz9368.dat に保存されている Cookie と URL のサイズが一定量を超えると，その内容を HTTP の POST メソッドを利用して外部に送信する．この振る舞いはステルス性を高めることにより自身の存在を検知されないようにするためであると考えられる．しかしながら，Blayzard は大量のイベント列を BhoApp Class に注入することにより，get_cookie と get_LocationURL の呼び出しを誘発して perfz9368.dat の書き込みを促す．その結果，情報を外部に送信する振る舞いを素早く誘発することができた．

IDA Pro [53] を用いて解析を行ったところ，xor 0x96 による暗号化を行って URL と Cookie を含む文字列を外部に送信していることが確認できた．復号した文字列は図 4.12 のようになる．そのため，確かに外部に送信されるメッセージは get_cookie と get_LocationURL の内容に依存していることが分かった．このことから，Blayzard による解析が正しいことが確認できる．

navfilter.dll の振る舞い

navfilter.dll は NavigateComplete2 を受信した際に，コールバックメソッドの write を呼び出して，直接 IE が表示しているウェブページを変更するアドウェアである．その後，ユーザが navfilter.dll が表示しているウェブページから他のウェブページに遷移を行う際にはダイアログボックスを表示して偽のマルウェア感染警告を表示する振る舞いを示す．navfilter.dll が write のコールバックメソッドを呼び出した際に IE に挿入される HTML は図 4.13 に示すとおりである．

navfilter.dll のもう 1 つの特徴的な振る舞いとしてダイアログボックスを

```
<html>
   <frameset cols='*'>
      <frame src='http://free-viruscan.com/00/00/00/
       error.php' frameborder=0 noresize scrolling=no>
   </frameset>
</html>
```

図 4.13: navfilter.dll が write メソッドを利用して IE に挿入する HTML

利用した警告メッセージは NavigateComplete2 を受信するごとに表示されないことである．そのため，navfilter.dll は Blayzard の偽のイベントを大量に挿入することにより特徴的な振る舞いを誘発することができた検体である．このように，頻繁な動作を避けるアドウェアやスパイウェアであっても Blayzard によるアプローチは効果的に働く．

この頻繁な動作を避ける原因を調べるために，商用のディスアセンブラである IDA Pro を用いて検査を行った結果，rand 関数が返す値が一定以上となった場合に警告メッセージが表示されることが分かった．そのため，Blayzard は静的解析を行わなくても navfilter.dll を活性化させて，特徴的な振る舞いを解析することができたといえる．

baiduc.dll の振る舞い

baiduc.dll は広告表示を行い，MAC アドレス，URL 及びこのアドウェアが生成した ID を外部に送信している．baiduc.dll を BHO プロセスに読み込むと，system32 ディレクトリに mprmsgse.axz というファイルを作成し，そのファイルに何らかの値を書き込んでいることが確認できる．その後，ドキュメントの表示完了を知らせる DocumentComplete イベントを受信すると，get_LocationURL メソッドをコールバックし，ドキュメントの URL を取得していることが確認できる．その後，MAC アドレス及び mprmsgse.axz に書き込んだ値を外部に送信していることが解析できる．

なお，baiduc.dll を含め，広告を表示させるタイプのアドウェアには共通する特徴的な動作がある．それは，IE のウィンドウがフォーカスされているときのみ広告が表示されるという点である．Blayzard では偽のイベントを送り込む際には，IE のウィンドウをフォーカスするようにしている．

```
http://202.75.49.168/go/go.php?``get_locationURL の値''
```

図 4.14: **autoex.dll** が呼び出す **Navigate** メソッドの遷移先

autoex.dll の振る舞い

autoex.dll は極めてステルス性が高い BHO ベースのスパイウェアである．なぜならば，ページ遷移のイベント列を 10,000 回繰り返し注入した際に BeforeNavigate2 のタイミングで Navigate メソッドを呼び出したためである．このことから，autoex.dll は Blayzard のイベント注入機構が効果的に働いた検体の 1 つであるといえる．

図 4.14 が示すように，autoex.dll は Navigate メソッドを呼び出す際に get_LocationURL を利用して得た URL を付与している．このことから，ページ遷移を行うタイミングで本来訪れようとした URL の値を取得していることが Blayzard を利用することにより分かった．

4.3.2 無害な BHO の振る舞い

アドウェアとは見なされていない BHO の振る舞いを調べるために，Google Toolbar などの 10 個の BHO の振る舞い解析を行った．表 4.12 に解析結果を示す．8 個の BHO は広告を表示したり情報を流出させることはなかったものの，Google Toolbar と Earthlink は情報漏洩を行うアドウェアに似た振る舞いをすることが分かった．以下，Google Toolbar と Earthlink について詳細を述べる．

Google Toolbar

Google Toolbar は訪問した URL や OS やブラウザのバージョンを外部に送信するなど，情報漏洩を行うアドウェアと似た振る舞いをすることが分かった．表 4.13 に Google Toolbar が送信した情報の一部を示す．これから分かるように，clients4.google.com に対してブラウザのバージョンを送信し，www.google.com に対して OS のバージョンやブラウザのバージョンなどを送信している．また，ページ遷移が完了したときに送信される NavigateComplete2 イベントを受信した時，引数として受け渡される toolbarqueries.google.co.jp の URL を送信している．

表 4.12: 評価を行ったよく利用される **BHO** とその振る舞い

名前	抽出できた振る舞い
Google Toolbar	URL やバージョン情報などを送信する
Earth link	OS の種類を流出する
Yahoo! Toolbar	バージョン情報などは送信しない
Bing Toolbar	バージョン情報などは送信しない
MicroGuarden	バージョン情報などは送信しない
Spybot S&D	バージョン情報などは送信しない
Super Ad Blocker	バージョン情報などは送信しない
Alexa Toolbar	バージョン情報などは送信しない
Spyware Guard	バージョン情報などは送信しない
MoreGoogle	バージョン情報などは送信しない

表 4.13: **Google Toolbar** が送信したシステム情報

ホスト名	外部に送信された通信内容
clients4.google.com	POST /tbproxy/...&v=6.5.708.1000&*browser=7.0.5730.13* &rlz=1T4GGLL_jaJP375JP376&... HTTP/1.1
www.google.com	GET /tools/...&*osver=5.1*&*ossp=3.0*&*osarch=32* &*browser=7.0.5730.13*&*browserarch=32*&... HTTP/1.1
toolbarqueries.google.co.jp	GET /tbr?client=navclient-auto&... &q=info:*http://www.a-blog.jp/*&... HTTP/1.1

Earthlink

Earthlink は使用している OS の種類を HTTP の GET メソッドを用いて外部に送信している．また，ドキュメントの表示が完了したことを通知する `DocumentComplete` イベントを受信した時，広告表示に似た振る舞いを行うことが分かった．`DocumentComplete` イベントを受信すると，数回に 1 度，`insertAdjacentHTML` メソッドをコールバックしポップアップを表示しようとしている．ただし，その URL はすでに無効であった．

4.4 議論

4.4.1 シグネチャの作成の支援

現在，数多くのアドウェア検出ツールや除去ツールが存在しており，実際に，Ad-Aware [75]，Spybot S&D [76]，Snort [77] や Bro [78] などのツールがある．これら

のツールを利用するためには，予め該当するアドウェアを解析して得られる定義ファイルを用意する必要がある．定義ファイルは，主に人手による解析を行って作成している．しかし，マルウェアの数は年々増加し続けており，アドウェアにおいてもこのことは例外ではない．そのため解析を行う作業は煩雑となるため，効率よく解析を行う必要がある．Blayzard は BHO ベースのアドウェアの解析を行うために，ブラウザと BHO を分離し，偽のイベントを解析対象のアドウェアに注入し，その振る舞いを抽出する動的解析を行うことで，アドウェアが発行したコールバックメソッド，システムコール列の呼び出しや，どのような情報を取得し，外部に送信したかを容易に知ることができる．Blayzard を利用することで，BHO ベースのアドウェアの解析を容易に行うことができる．そのため，定義ファイルの作成が容易になったり，より詳細な解析を行うための手がかりを得ることができる．

4.4.2 Blayzard の回避方法に関する議論

近年，マルウェア解析を行うために VMware [79] などの仮想化環境や Qemu [55] などのエミュレータを用いて，解析を行う手法を様々な研究者が提案している．これらを用いた解析手法はマルウェアのみならず，OS を含めたシステム全体を監視することができる．そのため，マルウェアは様々な方法 [63,80–82] を用いて仮想化環境やエミュレータ環境を識別し，そのような環境でマルウェアを動作することを避ける．解析環境を識別するマルウェアに対抗するために，Cobra [64] や Ether [65] は解析環境を識別するマルウェアをそのような環境上で動作していないように騙し，解析を行うシステムである．現在のマルウェアは Cobra [64] や Ether [65] に対抗できないといわれているものの，解析環境の識別を回避する時間のオーバーヘッドが大きいため，数多くのマルウェアを解析することには向いていない．Blayzard は仮想化環境やエミュレータに依存したシステムではないため，解析環境を識別する BHO ベースのアドウェアによる影響は受けない．

BHO ベースのアドウェアやスパイウェアは個々のイベント間の受信タイミングを考慮することにより Blayzard による解析を回避できる．なぜなら，Blayzard はイベント列を挿入する際に個々のイベントの挿入間隔を考慮していないため，BHO ベースのアドウェアやスパイウェアが通常のブラウジングを行う場合と比べて大量のイベントを受信した際には Blayzard の存在を推測できるためである．しかし，本論文で取り扱った 32 個のアドウェアやスパイウェアは個々のイベントの受信間隔を計測して挙動を変える検体は存在しなかった．Blayzard による解析を行った限

りでは，BHO ベースのアドウェアやスパイウェアはイベントの受信間隔よりも受信するイベントの種類の順序を考慮して自身の挙動を変える傾向が見られた．そのため，BHO ベースのアドウェアやスパイウェアはイベントの受信間隔を現在考慮して作成されていないといえる．

Blayzard を改良して，通常のブラウジングで発生するイベントの送信間隔と同じになるように，Blayzard が個々のイベントを挿入する間隔を調整することは可能である．しかし，個々のイベントの挿入間隔が増える分，解析の効率は落ちるため，Blayzard の回避の難しさと解析効率はトレードオフの関係がある．そのため，Blayzard を利用する解析者は解析の効率を優先するか，回避の難しさを優先するか適切に判断する必要がある．

第5章　偽アンチウイルスソフトの判別手法

偽アンチウイルスソフトはコンピュータの利用者に嘘の情報を提供して詐欺行為を働くマルウェアである．偽アンチウイルスソフトの主な動作は 2 つに大別することができる．まず，偽アンチウイルスソフトに感染すると偽のマルウェアスキャンを行い，マルウェアに感染していないにも関わらず嘘のマルウェア感染報告をコンピュータの利用者に表示する [83]．次に，その脅威に対処するために偽アンチウイルスソフトの商用版を購入するように被害者に促す．この警告に騙された被害者が偽アンチウイルスソフトの商用版を購入することにより詐欺行為が成立する．また，偽アンチウイルスソフトは正規のアンチウイルスソフトの振る舞いを考慮しつつ動作する．例えば，偽アンチウイルスソフトの 1 つである Security Antivirus は感染したコンピュータ内に実在するファイル名や正規のアンチウイルスソフトが頻繁にアクセスするディレクトリファイルにアクセスする．

加えて，偽アンチウイルスソフトと似たマルウェアとして粗悪なアンチウイルスソフトが存在する．粗悪なアンチウイルスソフトは偽アンチウイルスソフトと同様に詐欺行為を行うマルウェアである．しかし，偽アンチウイルスソフトと異なり粗悪なアンチウイルスソフトはマルウェア検出能力を僅かに備えているものの，マルウェアの検出率は正規のアンチウイルスソフトに及ばない．以降，区別する必要がある場合を除いて，偽アンチウイルスソフトと粗悪なアンチウイルスソフトを単に偽アンチウイルスソフトとして表記する．

正規のアンチウイルスソフトと偽アンチウイルスソフトの違いがどの部分に生じるかという議論は十分に行われていない．両者の違いを明確にすることができれば，偽アンチウイルスソフトの対策を素早く行うことが可能となる．現在，正規のアンチウイルスソフトベンダである McAfee は目視による偽アンチウイルスソフトの判別方法に関するホワイトペーパを提供している [84]．しかし，目視による判別方法であることから自動判別を行う指標として利用することに適していない．

第 5 章では，正規のアンチウイルスソフトの振る舞いを考慮して動作する偽アンチウイルスソフト特有の振る舞いを取得するために，マルウェアを利用してその特徴的な振る舞いを誘発する．これは正規のアンチウイルスソフトと偽アンチウイルスソフトはマルウェアを検出する際にそれぞれ異なる挙動を示すことが考えられるためである．正規のアンチウイルスソフトはマルウェアを検出する際に詳細な解析やシグネチャマッチングを行うことから CPU やメモリ使用量などの計算機資源を多量に消費すると考えられる．一方で，偽アンチウイルスソフトはマルウェアの検出能力が不十分であることから，正規のアンチウイルスソフトに表れる計算機資源の消費はない．

本研究では，マルウェアを利用して効率よく偽アンチウイルスソフト特有の振る舞いを誘発するために，マルウェアのあり，なしの環境を利用する．検査対象のアンチウイルスソフトらしき検体を両方の環境で実行して，各環境での振る舞いを習得した後に，得られた振る舞いに違いが生じるかどうかを調べることにより，正規のアンチウイルスソフトであるか偽アンチウイルスソフトであるかの判別を行う．正規のアンチウイルスソフトは十分なマルウェア検出能力を備えているため，マルウェアに感染した環境から得られる振る舞いとマルウェアに感染していない環境から得られる振る舞いには差が生じる．一方で，偽アンチウイルスソフトはマルウェアの検出を行うことはないため，マルウェアのあり，なしに関わらず振る舞いに違いが生じない．

実験的な調査から，メモリ使用量が正規のアンチウイルスソフトと偽アンチウイルスソフトを精度良く判別できる指標であることを示す．マルウェア検出能力を僅かに備えている粗悪なアンチウイルスソフトであっても，正規のアンチウイルスソフトのようにメモリを消費しないため，メモリ使用量は正規のアンチウイルスソフトと偽アンチウイルスソフトを正しく判別できるだけなく，正規のアンチウイルスソフトと粗悪なアンチウイルスソフトも正しく分類できる．

この違いを機械的に判別するために，本章ではリーベン検定と呼ばれる統計手法を利用する [40]．リーベン検定は異なるサンプル間において分散が等しいかどうかを判別する統計手法である．本研究はマルウェアのあり，なしの環境において，マルウェアを検出した際のメモリ使用量の分布に違いが生じるかどうかという部分に着目しているため分散の違いを利用している．リーベン検定を利用して，マルウェアに感染した環境と感染していない環境において取得したメモリ使用量の分散の違いを，マルウェアの検出能力のあり，なしに関連付けることにより判別を行う．リーベン検定により統計的な違いがないと判別された場合は，マルウェ

アの検出能力を伴わないことが原因である．そのため，提案手法は検査対象の検体を偽アンチウイルスソフトとして判別する．一方で，統計的な違いが生じる場合は，マルウェアの検出能力を備えていることが原因である．そのため，提案手法は検査対象の検体を正規のアンチウイルスソフトとして判別する．

提案手法が正規のアンチウイルスソフトと偽アンチウイルスソフトを正しく判別できることを確認するために，39 個の実在する偽アンチウイルスソフトと 8 個のよく利用されている正規のアンチウイルスソフトを対象に実験を行った結果，提案手法は 39 個の偽アンチウイルスソフトを正しく偽物と判別し，8 個の正規のアンチウイルスソフトを正しく本物と判別することができた．

提案方法は偽アンチウイルスソフトによる回避を困難にする．偽アンチウイルスソフトが提案手法を回避するためには，正規のアンチウイルスソフトと同等の品質となるように作成する必要があり，単純にメモリ使用量の傾向を変更するだけでは回避を行うことは難しい．実験では，複数回のリーベン検定を組み合わせることにより，偽アンチウイルスソフトがランダムにメモリ使用量を変更しても提案手法を回避できないことを示している．

5.1 偽アンチウイルスソフトによる被害

偽アンチウイルスソフトの被害は現実の脅威となっている．実際に，サイバー犯罪者は偽アンチウイルスソフトを数多くのコンピュータに感染させて，詐欺行為により莫大な収益を得ている．シマンテックは 2008 年の 7 月から 2009 年の 6 月にかけて，4,300 te 件の偽アンチウイルスソフトがインストールされたことを報告している [8]．Rajab らは Google のマルウェア検出機構 [85] を利用した調査を行った結果，検出したマルウェアのうち約 15% が偽アンチウイルスソフトであったことを明らかにしている [38]．Stone-Gross らは 3 種類の偽アンチウイルスソフトの金銭の流れを追跡した結果，それらが 1 億 3,000 万ドル以上の収益を得ていることを報告している [4]．同様に，McAfee は Innovative Marketing Ukraine という会社が偽アンチウイルスソフトの販売により 1 億 8,000 万ドル以上の利益を生んだことを明らかにしている [9]．偽アンチウイルスソフトを配布するための手段の 1 つとして，サイバー犯罪者はソフトウェアダウンロードサイトを利用する．実際に，RegGenie と呼ばれる偽アンチウイルスソフトは 2012 年に CNET というソフトウェアダウンロードサイトを介して配布されていた [86]．

5.2 偽アンチウイルスソフトと粗悪なアンチウイルスソフト

第 5.2 節では悪意あるアンチウイルスソフトである偽アンチウイルスソフトと粗悪なアンチウイルスソフトの特徴を詳述する．加えて，現状の対策は悪意あるアンチウイルスソフトの特徴を適切に取得できていないことを述べて，解決するべき課題を明らかにする．

5.2.1 偽アンチウイルスソフト

偽アンチウイルスソフトは正規のアンチウイルスソフトの振る舞いを考慮して動作し，感染したコンピュータのファイルシステムを検査することなく虚偽のマルウェア感染報告を表示する．偽アンチウイルスソフトは感染したコンピュータ上に存在するディレクトリ名やファイル名を偽のマルウェア感染報告として利用することにより，あたかもマルウェアスキャンを実行しているかのように見せる．例えば，偽アンチウイルスソフトの 1 つである Security Antivirus [87] は図 5.1 が表示する内容の警告メッセージを表示する．警告メッセージが表示しているパス名である `C:\Documents ... snl2w.dll` は感染したコンピュータ内に実際に存在するファイル名である．このように，偽アンチウイルスソフトは感染したコンピュータ内に実際に存在するパス名を表示することにより，あたかも実際に存在するファイルがマルウェアに感染しているかのようにコンピュータのユーザを騙す．最終的に，偽アンチウイルスソフトは報告されたマルウェアの除去を行うために偽アンチウイルスソフトの製品版を購入するようにコンピュータのユーザを促す．

偽アンチウイルスソフトによるディレクトリからファイル名を収集する挙動は正規のアンチウイルスソフトとの判別を困難にする．なぜなら正規のアンチウイルスソフトも個々のファイルにアクセスするために，ディレクトリからファイル名を収集するためである．そのため，一見すると正規のアンチウイルスソフトと偽アンチウイルスソフトが同じ振る舞いをしているかのように見える．実際に，Security Antivirus は多くの正規のアンチウイルスソフトが共通してアクセスする多くのディレクトリからファイル名を収集しており，その類似度は 99.7% (= 2393/2400) である．このことから，Security Antivirus は正規のアンチウイルスソフトの挙動を考慮したアクセスパターンであるといえる．このような性質があるため，偽アンチ

```
Virus name:
  Virus.Win32.Faker.a
Infected file:
  C:\Documents and Settings\Kasuya\Recent\snl2w.dll
Description:
  These programs steal MSN Messenger passwords using
  a fake dialogue box for entering MSN password. The
  program terminates connection and advises re-conn-
  ecting, and info entered is sent to the virus writer.
```

図 5.1: 偽アンチウイルスソフトが表示する警告メッセージ

ウイルスソフトと正規のアンチウイルスソフトの判別を行う際に，ディレクトリアクセスの類似度から判別することはできない．

5.2.2 粗悪なアンチウイルスソフト

粗悪なアンチウイルスソフトはマルウェアの検出精度が低いアンチウイルスソフトである．偽アンチウイルスソフトとは異なり，粗悪なアンチウイルスソフトはファイル名を収集した後に，各ファイルにアクセスしてマルウェアスキャンを行ってアクセスしたファイルがマルウェアであるかどうかの検査を行う．しかしながら，マルウェアの検出精度は十分に備わっていないため，実在するほとんどのマルウェアを検出することができないことから実用に耐えるものとならない．加えて，偽アンチウイルスソフトと同様に，粗悪なアンチウイルスソフトは感染したコンピュータのユーザに対して自身の製品版を購入するように促す．多くの場合，この製品版も十分なマルウェア検出率を伴わない粗悪品である．

粗悪なアンチウイルスソフトのマルウェア検出精度が低いことを確認するために，粗悪なアンチウイルスソフトの 1 つである Anti-virus Elite と正規のアンチウイルスソフトである Kaspersky のマルウェア検出精度をそれぞれ確かめた．Windows XP SP3 上に 905 種類のマルウェアをインストールして Anti-virus Elite と Kaspersky にそれぞれマルウェアスキャンを実行させたところ，この 905 種類のマルウェアに対する Anti-virus Elite の検出率は 8.2% (= 74/905) であることが分かった．一方で，Kaspersky は 905 種類の全てのマルウェアを検出することができた．すなわち，その検出率は 100% である．

粗悪なアンチウイルスソフトはマルウェア検出能力を僅かに備えているものの，多くの正規のアンチウイルスソフトは組閣なアンチウイルスソフトをマルウェアとして認識する．オンラインのマルウェアスキャンサービスである VirusTotal [88] を利用して Anti-virus Elite がマルウェアであるかどうかを調べたところ，VirusTotal に登録されている 65 % (= 28/43) の正規のアンチウイルスソフトが Anti-virus Elite をマルウェアとして判別している．

粗悪なアンチウイルスソフトは正規のアンチウイルスソフトと偽アンチウイルスソフトの境界を曖昧にする．なぜなら，粗悪なアンチウイルスソフトは，ファイル名を収集するディレクトリアクセスに加えて，個々のファイルにアクセスして，検査対象のファイルがマルウェアであるかどうかを検査するためである．検出率の低さを除くと粗悪なアンチウイルスソフトは正規のアンチウイルスソフトと似た振る舞いをすることからアンチウイルスソフトらしき検体が正規のアンチウイルスソフトであるか悪意あるアンチウイルスソフトであるかの判別を行うことは困難となる．

5.2.3 現在の正規のアンチウイルスソフトとの判別基準

セキュリティベンダのマカフィーは，目視による偽アンチウイルスソフトと正規のアンチウイルスソフトの判別方法をホワイトペーパとして公開している [84]. ユーザが文献 [84] で示す方法を参考にすることにより，両者の判別を行う際の手助けとなる．文献 [84] では，偽アンチウイルスソフトは膨大な数のマルウェア感染報告の表示を行ったり，コンピュータがマルウェアに感染している旨を知らせるポップアップウィンドウを頻繁に表示したり，製品版の購入を促すなどの振る舞いは偽アンチウイルスソフトの特有の振る舞いであると述べている．

この判別基準は人手を利用して偽アンチウイルスソフトを識別する際に役立つ指標となるものの，偽アンチウイルスソフトと正規のアンチウイルスソフトを自動的に判別するための有用な方法とはならない．例えば，マルウェア感染報告数を自動的に数える場合を想定すると，感染報告は自然言語で記述されているため，コンピュータがその報告から適切に感染報告数を数えることは容易ではない．また，あらゆる偽アンチウイルスソフトが文献 [84] で示す振る舞いを示すわけではない．例えば，偽アンチウイルスソフトの 1 つである Anti Spyware Expert は 18 件の虚偽のマルウェア感染報告のみを表示している．加えて，粗悪なアンチウイルスソフトは文献 [84] で示す方法を利用しても判別することは困難である．粗悪

なアンチウイルスソフトは十分なマルウェア検出率を伴わないということ以外では正規のアンチウイルスソフトと似た振る舞いを示すため，文献 [84] に従って判別を行ったとしても，その特徴的な振る舞いを発見することができない．

5.3 判別指標の発見

正規のアンチウイルスソフトと偽アンチウイルスソフトの判別を行うためには両者を正しく判別するための指標が必要である．しかし，現状の対策では正規のアンチウイルスソフトと偽アンチウイルスソフトの適切な違いを捉えることができていない．そのため，両者の判別を行うことができる指標を発見する必要がある．

正規のアンチウイルスソフトと偽アンチウイルスソフトの違いを効率良く誘発するために，本研究ではマルウェアを利用する．具体的には，マルウェアに感染した環境と感染していない環境をそれぞれ用意して，正規のアンチウイルスソフトと偽アンチウイルスソフトを実行させる．そして判別指標に従う振る舞いを抽出する．この指標は以下に示す振る舞いを識別する必要がある．正規のアンチウイルスソフトの場合は十分なマルウェア検出能力を備えていることから，マルウェアに感染した環境と感染していない環境では顕著な違いが出ると考えられる．一方で，偽アンチウイルスソフトの場合は，マルウェア検出能力を十分に備えていないため環境の違いにおいて違いが生じないと考えられる．

5.3.1 指標に求められる要件

偽アンチウイルスソフトを判別するための指標は 3 つの要件を満たす必要がある．第 1 に，様々な種類の偽アンチウイルスソフトに対応できる必要がある．特に，本指標は未知の検体においてもその特徴的な振る舞いを識別できる必要がある．

第 2 に，偽アンチウイルスソフトが本指標を回避することが困難となるようにする必要がある．なぜなら偽アンチウイルスソフトの作成者は本指標を回避するためにより精巧な偽アンチウイルスソフトを作成しなければならないためである．偽アンチウイルスソフト作成者が精巧な偽アンチウイルスソフトを作成することは多大な労力を必要とする．正規のアンチウイルスソフトはその品質を保つために，数多くのマルウェアを解析して様々なシグネチャを用意したり，製品そのもののアップデートを行うことにより，その品質を向上させている．そのため，正

規のアンチウイルスソフトと同等の品質を得るためには，偽アンチウイルスソフトも同様のことを行う必要があるためである．

第 3 に，正規のアンチウイルスソフトと偽アンチウイルスソフトの判別を自動で行う必要がある．判別を自動化することにより，正規のアンチウイルスソフトにその機能を組み込むことができたり，ソフトウェアダウンロードサイト [89,90] を利用した偽アンチウイルスソフトの配布を未然に防ぐといったことが可能となる．既存の対策である文献 [84] による判別は偽アンチウイルスソフトとして考えられるの疑わしい振る舞いについて，人目による判別を行う必要がある．本指標を利用した判別は人目による判別を行わず，機械的に処理できる方法を採用する．例えば，正規のアンチウイルスソフトと偽アンチウイルスソフトのシステムコールの発行パターンの違いやコンピュータの計算機資源の使用量の違いを利用した判別を行う．

判別指標を取得するための 1 つの方法として，バイナリ解析を利用する方法がある．一般的に，マルウェアのソースコードは公開されていないため，ソースコードから振る舞いを取得する方法を仮定することは適切とはいえない．バイナリ解析により，マルウェアが実行する多くの振る舞いを取得することができるものの，静的解析を利用する方法はマルウェアによる難読化の問題が生じるため，この問題を完全に解決することは困難である．また，動的解析による方法は難読化の問題を考慮する必要がなくなるものの，得られた振る舞いから悪意ある動作を含むかどうかを判別することは困難である．こうした背景を考慮して，本指標はこれらの方法に依存しない方法を採用する．

5.3.2 基本的なアプローチ

提案手法は正規のアンチウイルスソフトと偽アンチウイルスソフトの違いをシステムコールの発行パターンや計算機資源量を指標として判別する．本研究ではマルウェアを利用してこの違いを効率良く抽出する．具体的には，マルウェアに感染した環境と感染していない環境から得られる指標を比較することである．以降マルウェアに感染した環境を *infected environment*，マルウェアに感染していない環境を *clean environment* として表す．Clean environment は DVD-ROM などの read-only のインストールメディアから OS をインストールした直後の状態を表す．この環境に無害なファイル (例えば画像ファイル) をインストールしても同様に clean environment のままである．こうした環境は仮想機械技術を利用して作成する．

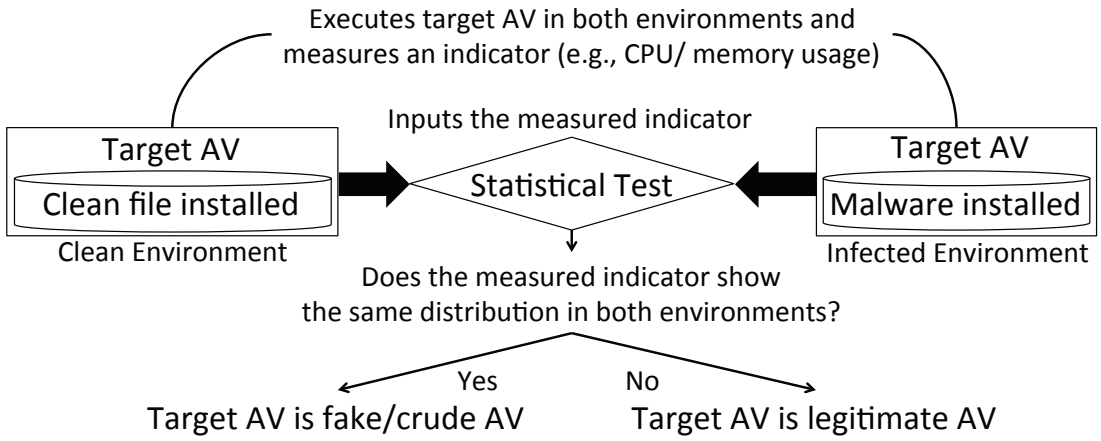

図 5.2: 提案手法の概略図

Clean environment を用意した直後に，仮想マシンのイメージ (スナップショット) としてこの状態を保存することにより容易に clean environment を再現できる．Clean environment にマルウェアをインストールすることにより，infected environment を作成する．Infected environment も同様にスナップショットを取得することにより，検査前の状態に容易に復元できる．

図 5.2 は提案手法を利用する判別方法の概略図を表している．提案手法は判別指標取得フェーズと統計による判別フェーズの 2 段階に分かれる．まず，判別指標取得フェーズでは，検査対象の検体を clean environment と infected environment でそれぞれ実行させて判別指標を取得する．ここで取得する判別指標は後述するファイルアクセスパターン，CPU 利用率，メモリ使用量などである．その後，統計による判別フェーズでは，clean environment と infected environment から得られた指標に差が生じるかどうかを調べることにより，マルウェアの検出能力があるかどうかを判別する．

この判別方法の核となる考え方はマルウェアを発見した際に，正規のアンチウイルスソフトと偽アンチウイルスソフトはそれぞれ異なる振る舞いを示すことが考えられることにある．なぜなら，正規のアンチウイルスソフトはマルウェア検出能力があり，偽アンチウイルスソフトはマルウェア検出能力がないためである．正規のアンチウイルスソフトはマルウェアを発見した際に詳細な解析やシグネチャマッチングを行うことから clean environment から得られる指標と infected environment から得られる指標は統計的に差が生じると考えられる．一方で，偽アンチウイルスソフトはマルウェアの検出能力を十分に備えていないため clean environment と infected environment から得られる指標に統計的な差が生じないことが期待される．

提案手法は先に述べた 3 つの要件を満たす方法である．第 1 に，本指標は特定の偽アンチウイルスソフトの振る舞いに依存する判別方法ではない．本指標はマルウェア検出能力が備わっているかどうかの違いを捉える指標であるため，様々な偽アンチウイルスソフトに利用することができる．第 5.4 節で示すように，提案手法は実験で用いた 39 個の偽アンチウイルスソフトの全てを正しく偽物として判別できて，8 個の正規のアンチウイルスソフトの全てを本物として判別できている．第 2 に，偽アンチウイルスソフトが提案指標を回避することは現実的ではない．仮に回避を行う際には，偽アンチウイルスソフトは infected environment に存在するマルウェアを正しく識別して，あたかも正規のアンチウイルスソフトのように動作する必要がある．つまり，正規のアンチウイルスソフトが持つマルウェア検出能力と同等の機能を所有する必要がある．これを実現するためには，サイバー犯罪者は正規のアンチウイルスソフトそのものを作成する必要がある．回避方法に関する議論は第 5.5.1 項で行う．第 3 に，提案指標による判別は機械的に処理することが可能である．得られた指標は統計的な方法によって処理され，統計的な差が生じる場合は正規のアンチウイルスソフトとして判別され，統計的な差が生じない場合は偽アンチウイルスソフトとして処理される．そのため，目視による判別に依存しない．

5.3.3 指標の候補

正規のアンチウイルスソフトと偽アンチウイルスソフトを効率良く判別できる指標を見つけるために，機械的に取得できる 3 つの指標について，実験的に調査する．本論文では，1) ファイルアクセスパターン，2) CPU 利用率，3) メモリ使用量の 3 つを取り扱う．これらを得するために，ファイルアクセスパターンの取得にはシステムコールのフックを利用してファイルのアクセス履歴を取得する．CPU 及びメモリ使用量の取得には Windows OS がデフォルトでインストールしている Performance Monitor と呼ばれるアプリケーションを利用している．

ファイルアクセスパターン

ファイルアクセスパターンによる正規のアンチウイルスソフトと偽アンチウイルスソフトの判別はファイルアクセスの網羅率に着目する．ファイルアクセスパターンに関して，正規のアンチウイルスソフトと偽アンチウイルスソフトの違い

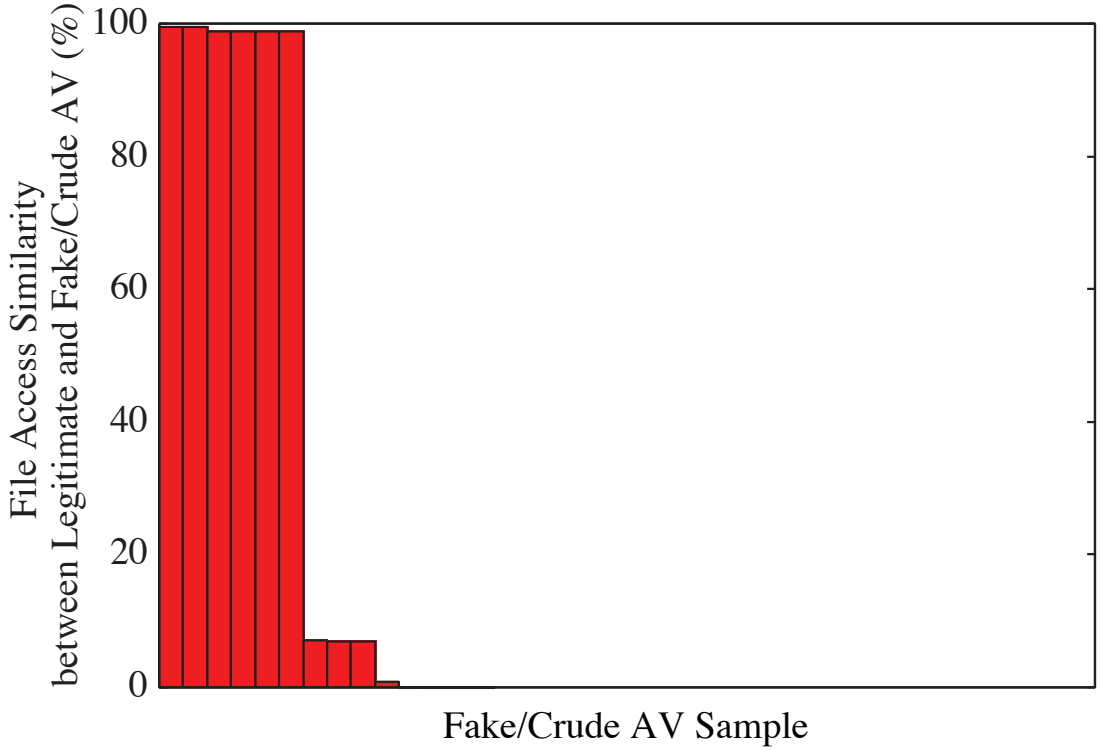

図 5.3: 正規のアンチウイルスソフトと偽アンチウイルスソフト間のファイルアクセスパターンの類似度．39 個中 6 個の偽アンチウイルスソフトは 8 個の正規のアンチウイルスソフトが共通してアクセスするファイルにアクセスするため判別指標として不適切である．

は以下のように考えられる．正規のアンチウイルスソフトはファイルシステムを網羅的に検査して，マルウェアに感染しているかどうかを調べる．一方で，偽アンチウイルスソフトはマルウェアを検出する必要がないため，正規のアンチウイルスソフトのように個々のファイルへのアクセスを行わない．そのため，ある検体のファイルアクセスが十分な場合は正規のアンチウイルスソフトと考えられ，そうでない場合は偽アンチウイルスソフトと考えられる．

しかしながら，ファイルアクセスパターンは正規のアンチウイルスソフトと偽アンチウイルスソフトを判別するための指標とはならない．なぜならば偽アンチウイルスソフトの中でも正規のアンチウイルスソフトのようにファイルアクセスを行う検体が存在するためである．図 5.3 は 8 個の正規のアンチウイルスソフトが共通してアクセスするファイルに，39 個の偽アンチウイルスソフトがどの程度アクセスしているのかを表すグラフである．各棒グラフは個々の偽アンチウイルスソフトを表している．100% に近づいている棒グラフは，正規のアンチウイルスソフトがアクセスするほぼすべてのファイルにアクセスしていることを表す．図 5.3

が示すように，6 個の偽アンチウイルスソフトは正規のアンチウイルスソフトがアクセスするほぼ全てのファイルにアクセスしていることが分かる．図 5.3 は 10 個の偽アンチウイルスソフトについてのみ表示している．なぜなら残りの 29 個の偽アンチウイルスソフトは個々のファイルにアクセスすることがなかったためである．

CPU 利用率

CPU 利用率による正規のアンチウイルスソフトと偽アンチウイルスソフトの判別は clean environment と infected environment から取得した CPU 利用率のユーザモードの占有率に差が生じるかどうかを調べることにより行う．正規のアンチウイルスソフトが行うマルウェアスキャンは 1 種のアプリケーションであるため，シグネチャマッチングや検査対象のファイルの詳細な解析を実行している間は CPU はユーザモードで実行すると考えられるためである．そのため，正規のアンチウイルスソフトと偽アンチウイルスソフトは CPU 利用率について次のような違いが表れると考えられる．正規のアンチウイルスソフトの場合は，マルウェア検出能力があることから infected environment においてユーザモードの占有率が増えることが予想される．そのため，clean environment と infected environment から取得できる CPU 利用率のユーザモードの占有率に違いが生じると考えられる．一方で，偽アンチウイルスソフトはマルウェアの検出能力を持たないことため，clean environment と infected environment から得られる CPU 利用率のユーザモードの占有率に差が生じないと考えられる．

調査の結果，infected environment と clean environment から得られる CPU 利用率のユーザモードの占有率の差を利用して判別する方法は，正規のアンチウイルスソフトと偽アンチウイルスソフトの違いを適切に捉えていないことが分かった．図 5.4 は 8 個の正規のアンチウイルスソフトと 39 個の偽アンチウイルスソフトについて，infected environment と clean environment から得られた CPU 利用率のユーザモードの占有率における平均の差を表示したグラフである．縦軸の U_i と U_c はそれぞれ infected environment と clean environment から得られた CPU 利用率のユーザモードの占有率の平均を表す．図 5.4 中の緑と赤の棒グラフはそれぞれ正規のアンチウイルスソフトと偽アンチウイルスソフトを表す．図 5.4 が示す結果から正規のアンチウイルスソフトであろうと偽アンチウイルスソフトであろうと，CPU 利用率のユーザモードの平均の差は多様になることが分かった．当初の予想

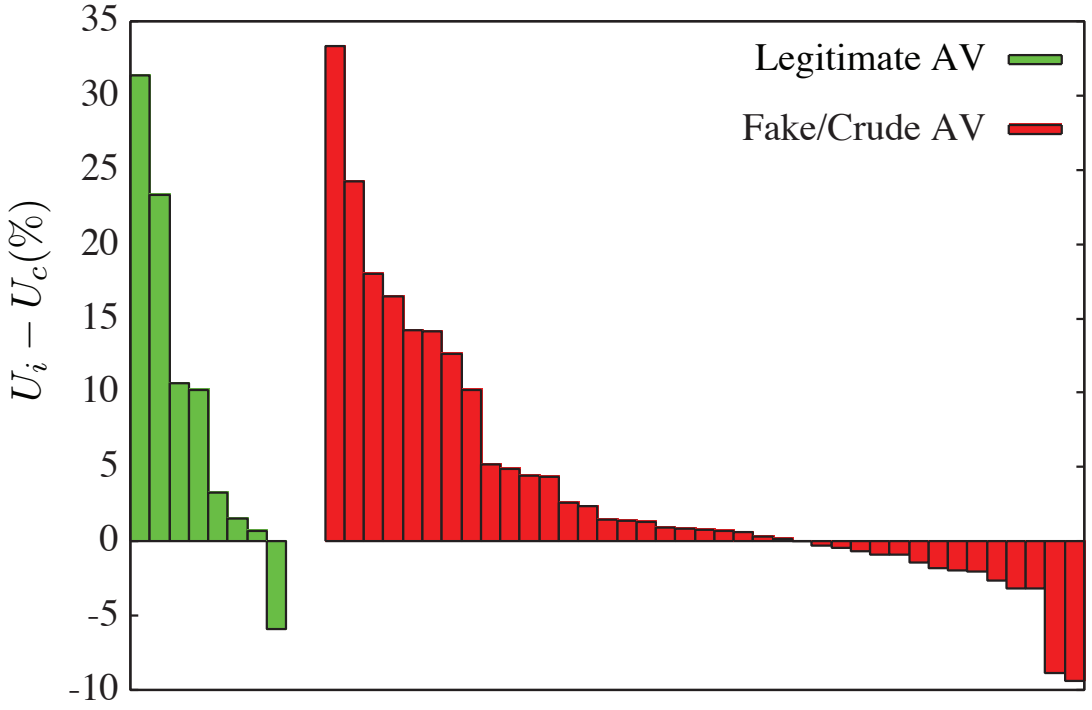

図 5.4: **Clean environment** と **infected environment** から得られたユーザモードの **CPU** 利用率の比較．U_i と U_c は infected environment と clean environment から得られたユーザモードにおける CPU 利用率の平均の差を表す．緑と赤の棒グラフはそれぞれ正規のアンチウイルスソフトと偽アンチウイルスソフトを表す．アンチウイルスソフトの種類に関わらず，$U_i - U_c$ の値は -10% から 30% の間を推移している．

とは異なり，正規のアンチウイルスソフトの中にも infected environment と clean environment から得られる CPU 利用率のユーザモードの占有率の差がほとんどない検体が存在する一方，偽アンチウイルスソフトの中にも，その差が大きい検体が存在する．正規のアンチウイルスソフトと偽アンチウイルスソフトに関わらず，infected environment と clean environment 間の CPU 利用率のユーザモードにおける平均の差は -10% から 30% の幅がある．

メモリ使用量

メモリ使用量における正規のアンチウイルスソフトと偽アンチウイルスソフトの期待される違いは次のようになる．正規のアンチウイルスソフトの場合は，マルウェアを検出する際に詳細な解析やシグネチャマッチングのためにメモリを多く消費することが予想される．そのため，infected environment と clean environment か

ら得られるメモリ使用量は異なると考えられる．しかし，偽アンチウイルスソフトの場合は，十分なマルウェア検出能力を備えていないことから infected environment と clean environment のメモリ使用はほとんど変化がないと考えられる．

調査の結果，マルウェア検出能力のあり，なしはメモリ使用量に表れることが分かった．つまり，メモリ使用量は正規のアンチウイルスソフトと偽アンチウイルスソフトの判別を行うための指標になる．図 5.5，図 5.6，図 5.7 は infected environment と clean environment から得られた正規のアンチウイルスソフト，偽アンチウイルスソフト，粗悪なアンチウイルスソフトのメモリ使用量を示している．図 5.5 が示すように正規のアンチウイルスソフトは infected environment と clean environment のメモリ使用量は著しく異なる．これはマルウェアを検出した際に，詳細な解析やシグネチャマッチングを行っているため正規のアンチウイルスソフトに追加でメモリが割り当てられていることを表している．図 5.6 が示す偽アンチウイルスソフトは環境の違いに関わらずメモリ使用量に変化がない．加えて，図 5.7 の粗悪なアンチウイルスソフトも偽アンチウイルスソフトと同様に infected environment と clean environment から得られるメモリ使用量に違いがない．僅かながらマルウェア検出能力を備えているものの，粗悪なアンチウイルスソフトはメモリ使用量が著しく変わるほどの挙動を示していないことが分かった．

本研究では infected environment と clean environment から得られるメモリ使用量の分散に違いが生じるかどうかを調べてマルウェアの検出能力のあり，なしを判別することにより，正規のアンチウイルスソフトか偽アンチウイルスソフトかの判別を行う．図 5.8 は infected environment と clean environment から得られたメモリ使用量の分散について，縦軸を V_i/V_c とした割合を正規のアンチウイルスソフトと偽アンチウイルスソフトごとにまとめたグラフである．V_i と V_c はそれぞれ infected environment と clean environment から得られたメモリ使用量である．

V_i/V_c の値を利用することにより，正規のアンチウイルスソフトと偽アンチウイルスソフトを適切に判別することができる．図 5.8 では，全ての正規のアンチウイルスソフトは V_i/V_c が表す分散の割合が 4 以上となっている．しかし，偽アンチウイルスソフトは V_i/V_c が表す分散の割合がいずれも 1 前後に留まっている．このように，いずれの正規のアンチウイルスソフトも Infected environment ではメモリ使用量の分布が変わるものの，偽アンチウイルスソフトの場合は，infected environment と clean environment においてメモリ使用量の分布は変わらないことが分かる．そのため，図 5.8 が示すように infected environment と clean environment から得られるメモリ使用量の分散を判別指標とすることにより，正規のアンチウ

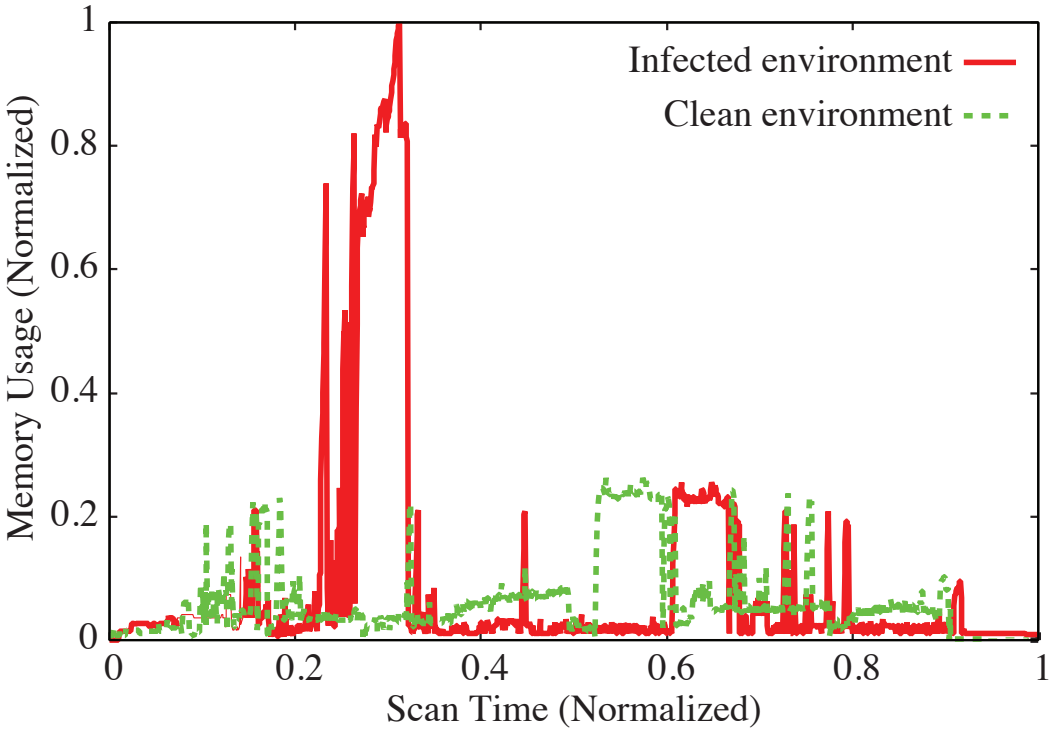

図 5.5: **Infected environment** と **clean environment** から得られた **McAfee** のメモリ使用量 (正規のアンチウイルスソフト)

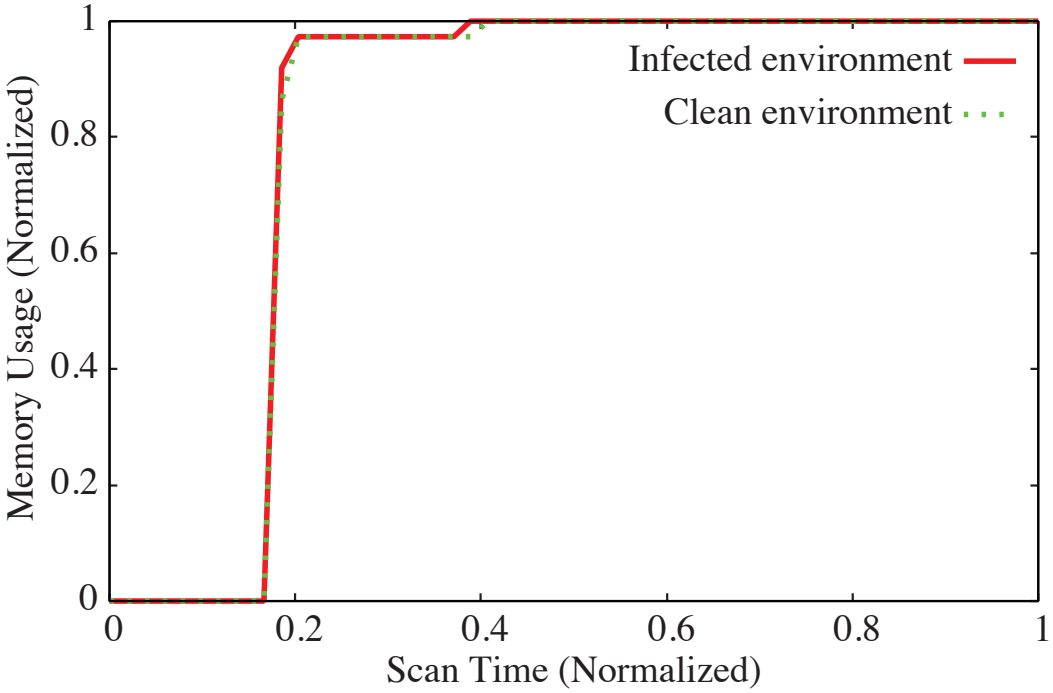

図 5.6: **Infected environment** と **clean environment** から得られた **Major Defense Kit** のメモリ使用量 (偽アンチウイルスソフト)

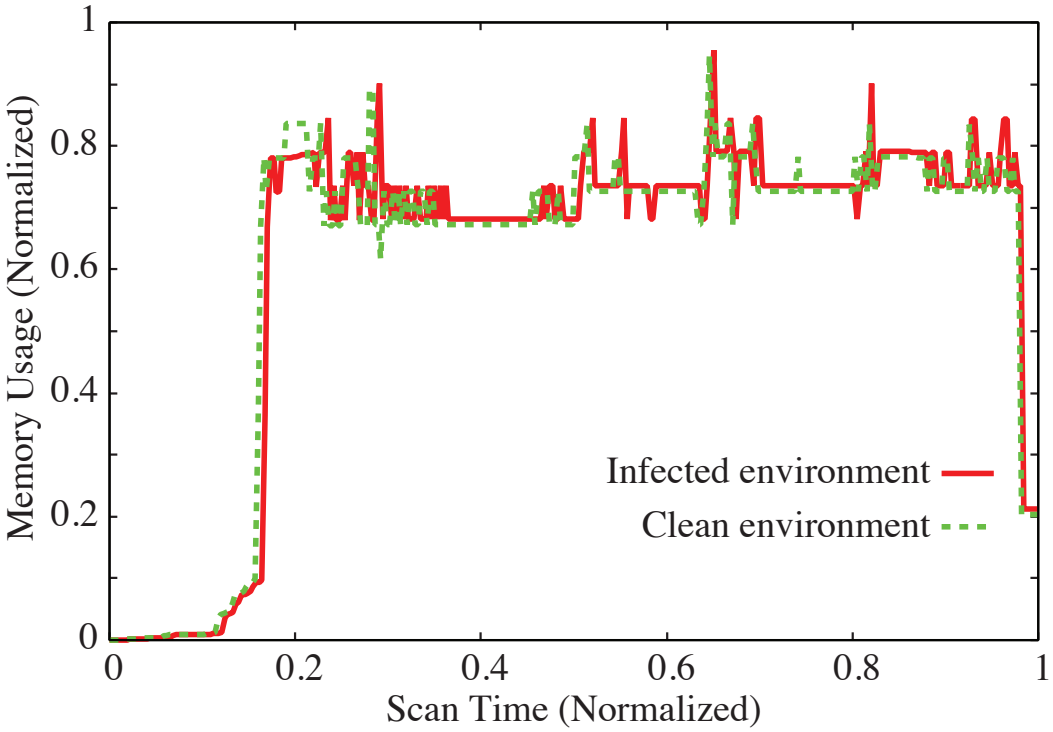

図 5.7: **Infected environment** と **clean environment** から得られた **Anti-virus Elite** のメモリ使用量 (粗悪なアンチウイルスソフト)

表 5.1: **3** つの指標候補に関する調査結果

指標候補	比較対象	監視方法	結果
ファイルアクセスパターン	アクセスするファイルの類似度	システムコールのフック	X
CPU 利用率	CPU 利用率のユーザモードの占有率	毎秒 CPU 利用率を取得	X
メモリ使用量	メモリ使用量の分散	毎秒メモリ使用量を取得	✓

イルスソフトと偽アンチウイルスソフトの判別をすることができる．

表 5.1 は調査した 3 つの指標に関する調査結果である．各指標候補に対して，比較対象，監視方法及び結果を記載している．3 つの指標候補を調査した結果，メモリ使用量を利用することにより，正規のアンチウイルスソフトと偽アンチウイルスソフトを適切に判別できることが分かった．ファイルアクセスパターンや CPU 利用率は当初の予想とは異なり正規のアンチウイルスソフトと偽アンチウイルスソフトの判別に利用できない．本研究では，メモリ使用量の分散の違いを統計手法を用いて，その差が統計的に有意であるかを調べる．

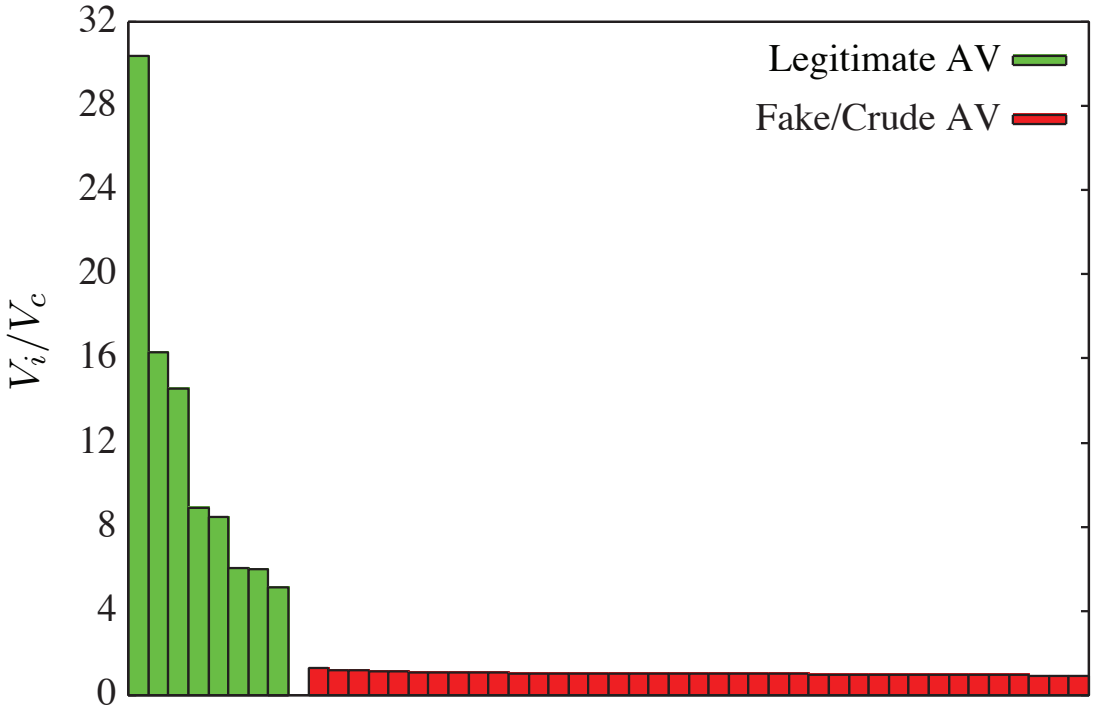

図 5.8: **Infected environment** と **clean environment** から得られたメモリ使用量の分散の比較．V_i と V_c は infected environment と clean environment から得られたメモリ使用量の分散を表す．緑と赤はそれぞれ正規のアンチウイルスソフトと偽アンチウイルスソフトを表す．全ての正規のアンチウイルスソフトは infected environment においてメモリ使用量が著しく増加した結果，分散が大きくなっている．しかし，偽アンチウイルスソフトは両環境から得られる分散にほとんど違いが表れない．

5.3.4 メモリ使用量を利用する偽アンチウイルスソフトの判別手法

正規のアンチウイルスソフトと偽アンチウイルスソフトの判別を行うためにメモリ使用量を判別のための指標とした具体的な方法を詳述する．

検査の流れは以下のようになる．まず，検査対象となるアンチウイルスソフトを infected environment と clean environment にそれぞれインストールを行う．Infected environment は 905 種類の合計 500 MB のマルウェアをインストールする．Clean environment は 905 個の合計 500 MB の画像ファイルをインストールする．Infected environment と clean environment にインストールされる個々のマルウェアと画像ファイルは，同一のディレクトリ構造と同一のサイズで構成されており，両環境の唯一の違いはマルウェアが含まれているかいないかに限定される．

検査対象のアンチウイルスソフトのマルウェアスキャン実行中のメモリ使用量の分布を取得するために，メモリ使用量を infected environment と clean environment

のそれぞれの環境から毎秒ごとに取得する．Infected environment から取得するメモリ使用量の分布を Y_1，clean environment から取得するメモリ使用量の分布を Y_2 と仮定した場合，提案手法が Y_1 と Y_2 を統計的に異なると判断する場合は検査対象の検体を正規のアンチウイルスソフトと判別する．一方で，Y_1 と Y_2 を統計的に等しいと判別する場合は偽アンチウイルスソフトとして判別する．

Infected environment と clean environment から取得するメモリ使用量の分散が統計的な有意差が生じるかどうかを判別するために，本研究では統計検定の 1 つであるリーベン検定 [40] を利用する．リーベン検定は異なるサンプル間における分散が統計的に等しいかどうかを判定する際に用いる検定である．Infected environment から取得するメモリ使用量の分布と clean environment から取得するメモリ使用量の分布をリーベン検定の入力として与えた際に，その結果が統計学においてよく利用される値である 0.05 未満となるかどうかにより両環境から取得したメモリ使用量の分散に差に有意な差があるかどうかを判断する．リーベン検定による結果が 0.05 未満となった場合は，統計的に有意な差が生じているため，提案手法は検査対象のアンチウイルスソフトをを正規のアンチウイルスソフトとして判別する．一方で 0.05 以上となった場合は統計的に有意な差がないため，提案手法は検査対象のアンチウイルスソフトを偽アンチウイルスソフトとして判別する．

偽アンチウイルスソフトによる提案手法の回避を困難にするために，infected environment と clean environment から複数個のメモリ使用量を取得してリーベン検定を複数回実施する．なぜならランダムにメモリ使用量を変更する偽アンチウイルスソフトは，組み合わせ次第では infected environment と clean environment から取得するメモリ使用量の分布が異なる場合があり 1 回のリーベン検定による判定では不十分なためである．複数回のリーベン検定による正規のアンチウイルスソフトと偽アンチウイルスソフトの判別は 1) 全ての組み合わせで統計的な有意差が生じる場合か 2) いずれかの組み合わせで統計的な有意差が生じない場合の 2 つで判別を行う．複数回のリーベン検定の結果が全て統計的な有意差 (0.05 未満) が生じる場合は，提案手法は検査対象のアンチウイルスソフトを正規のアンチウイルスソフトと判別する．一方で，いずれかの組み合わせでリーベン検定の結果が統計的な有意差を生じない場合，提案手法は検査対象のアンチウイルスソフトを偽アンチウイルスソフトとして判別する．しかしながら，メモリ使用量を複数回計測するのは時間が掛かるため，$\lceil\sqrt{M}\rceil$ 回ずつ各環境でメモリ使用量を計測して，異なる環境同士でペアを作成することにより合計 M 回のリーベン検定を実施する．

表 5.2: 実験で利用した正規のアンチウイルスソフト

Avast Pro Antivirus 7.0.1426	G Data Antivirus 2011 21.1.0.1
AVG Antivirus 2012.0.1913	Kaspersky Anti-Virus 2011 11.0.2.556
McAfee VirusScan 15.0.294	ESET NOD32 Antivirus 4.2.71.2
Norton AntiVirus 18.7.0.13	Panda Antivirus Pro 2011 10.00.00

表 5.3: 実験で利用した偽アンチウイルスソフト

XP Internet Security 2011	XP Internet Security 2012 6.0.2900.2180
XP Home Security 2011	XP Home Security 2012 6.0.2900.2180
XP Anti Spyware 2011	XP Anti Spyware 2012 6.0.2900.2180
XP Antivirus 2011	XP Antivirus 2012 6.0.2900.2180
XP Security 2011	XP Security 2012 6.0.2900.2180
XP Total Security 2011	PC Privacy Cleaner 1.0.22.4
Patchup Plus	Virus Remover 2008 1.0.15.2
Security Tool	Virus Remover 2009 1.0.9.0
System Security	Anti Spy Safeguard 1.0.0.0
XL Guarder	Security Antivirus 2.0.2.18
Security Shield	Major Defense Kit 1.0.0.0
Protect Code	Anti Spyware Bot 9.6.9
Adware Bot 12.0.6	Security Defender 1.6.812.0
Reg Clean 1.0.0.1	Malware Removal Bot 12.0.6
Onescan 1.0.0.1	Anti Spyware Expert 1.0.22.2
Anti-Spyware 12.0.6	Anti-Virus Elite v5.0
Error Sweeper 2.8.0	Pest Detector 1.0.0.0
Registry Smart 2.10.0	Netcom3 PC Cleaner 9.1.10
Red Cross 1.0.0.0	Peak Protection 1.0.0.0
Privacy Control 2.6.0.0	

5.4 実験

複数回のリーベン検定を利用する提案手法が正規のアンチウイルスソフトと偽アンチウイルスソフトを正しく判別できるかを確かめるための実験を行う.

5.4.1 実験環境

実在する検体に対して有用な方法であることを示すために，表 5.2 に示す 8 個の正規のアンチウイルスソフトと 表 5.3 で示す 39 個の偽アンチウイルスソフトを利

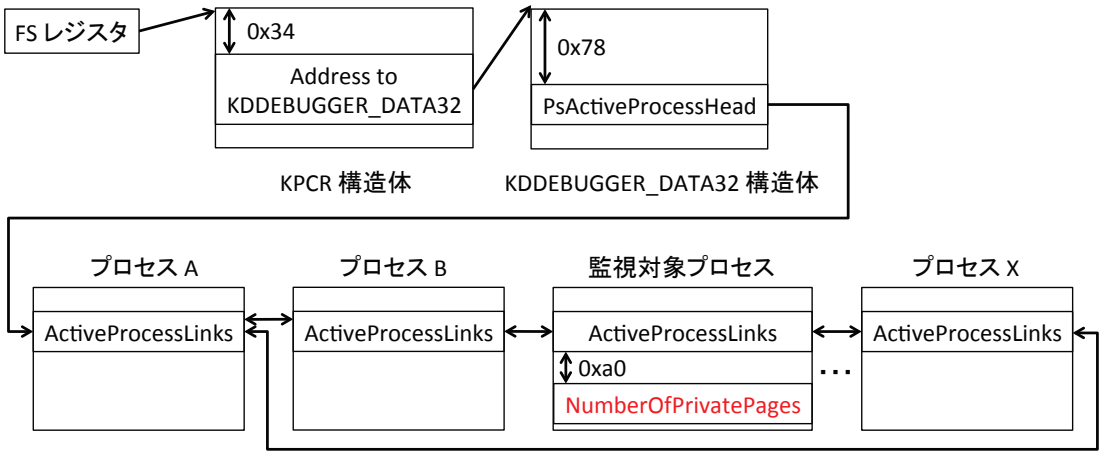

図 5.9: プロセス構造体からメモリ使用量を取得する方法

用している．偽アンチウイルスソフトはマルウェア収集サイト [69,91] や Malware Domain List [92] と呼ばれるマルウェアの配布があった URL を公開しているウェブサイトから収集している．Infected environment と clean environment は KVM を伴う Qemu 1.1.1 に Windows XP SP3 をインストールして，メモリを 1GB，Intel Xeon 2.4 GHz の CPU を 1 つ割り当てている．本実験においては Windows XP を利用しているものの，提案手法は OS に依存する方法ではない．そのため，他の OS を利用する場合においても，メモリ使用量を取得することができれば，どの OS にでも適用できる．

5.4.2 メモリ使用量の取得

メモリ使用量の取得は偽アンチウイルスソフトによる悪影響を防ぐために，Windows のプロセス構造体である EPROCESS 構造体のエントリから直接検査対象のプロセスのメモリ使用量を取得する．偽アンチウイルスソフトは API やシステムコールを監視して結果を改ざんすることが可能であるため Performance Monitor などのアプリケーションから得られる値が不正確になる場合があるためである．

EPROCESS 構造体のエントリからメモリ使用量を取得するまでの流れは 図 5.9 が示すように FS レジスタの値から辿ることが可能である．Windows XP を利用する場合，FS レジスタは KPCR 構造体の先頭アドレス (`0xffdff000`) を保持している．KPCR 構造体の先頭アドレスに `0x34` のオフセットを足すと KDDEBUGGER_DATA32 構造体への先頭アドレスを取得できる．この先頭アドレスに `0x78` のオフセットを足すと PsActiveProcessHead と呼ばれるカーネル変数のアドレスを

取得できる．PsActiveProcessHead は現在実行中のプロセスの EPROCESS 構造体の ActiveProcessLinks エントリのアドレスを保持している．WIndows XP を利用する場合，PsActiveProcessHead は System Process の ActiveProcessLinks のアドレスを保持している．ある EPROCESS 構造体から別の EPROCESS 構造体を得るためには ActiveProcessLinks を辿り続けることにより，全ての EPROCESS 構造体にアクセスすることができる．監視対象の EPROCESS 構造体を発見したら 該当プロセスの ActiveProcessLinks を保持しているアドレスに `0xa0` のオフセットを足すことにより NumberOfPrivatePages を保持しているアドレスを取得できる．NumberOfPrivatePages は該当プロセスのメモリ使用量を保持しているエントリである．実際には NumberOfPrivatePages はページ数を保存されているため，その値を 4,096 倍することによりメモリ使用量を得ることができる．

監視対象のアンチウイルスソフトの EPROCESS 構造体を発見するために，Windows がデフォルトで起動するプロセスは予め監視対象から外しておくというヒューリスティックを用いている．これにより，検査対象のアンチウイルスソフトを infected environment と clean environment にインストールして実行する際には新たなプロセスが作成されるため，それを監視対象としている．

5.4.3 実験方法

本実験では，リーベン検定を 9 回実施する．すなわち，M が 9 となり，$\lceil\sqrt{M}\rceil$ が 3 となるため，infected environment と clean environment からそれぞれメモリ使用量を 3 回ずつ取得する．本実験における判別の流れは大別すると図 5.10 と図 5.11 の 2 つに大別できる．図 5.10 のように 9 回全ての検定結果が統計的な有意差を生じる場合，提案手法は検査対象のアンチウイルスソフトを正規のアンチウイルスソフトとして判別する．一方，図 5.11 のように，いずれかの組み合わせにおいて統計的な有意差を生じない場合に提案手法は検査対象のアンチウイルスソフトを偽アンチウイルスソフトとして判別する．

5.4.4 実験結果

提案手法は実験で利用した 39 個の偽アンチウイルスソフトを正しく偽物として判別し，8 個の正規のアンチウイルスソフトを正しく本物として判別することができた．実験結果の一覧は表 5.4 に示す通りである．表 5.4 において，*class* は検査対

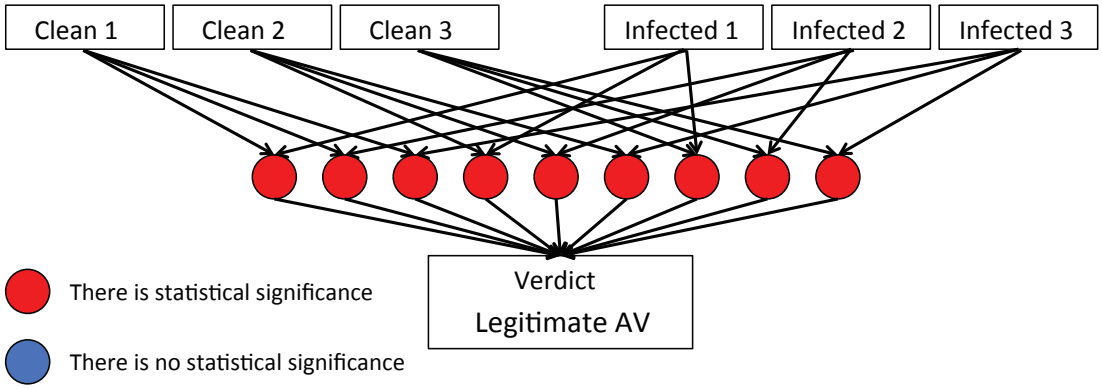

図 5.10: 複数回のリーベン検定により正規のアンチウイルスソフトとして判別される場合

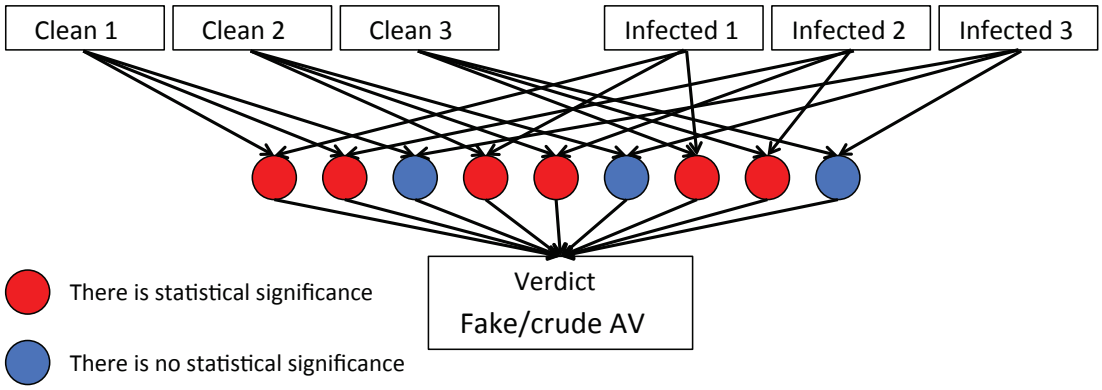

図 5.11: 複数回のリーベン検定により偽アンチウイルスソフトとして判別される場合

象のプログラムが正規のアンチウイルスソフト (legitimate) であるか偽アンチウイルスソフト (fake/crude) であるかを表している．C 及び I は検査対象のアンチウイルスソフトが clean environment と infected environment においてマルウェアスキャン中に取得したメモリ使用量を表している．例えば，C_1 は clean environment 1 から取得したメモリ使用量のことである．各環境から C_1，C_2，C_3，I_1，I_2，I_3 のメモリ使用量を取得し，異なる環境から得られたメモリ使用量同士でペアを作成することにより合計 9 回のリーベン検定を行う．表 5.4 では，あるペアのリーベン検定の結果が統計的な有意差を示す場合は ✓ として，統計的な有意差を示さない場合は X として表示している．全てのペアで統計的な有意差を示した検査対象のプログラムは正規のアンチウイルスソフト (legitimate) として判別され，いずれかのペアが統計的な有意差を含まない場合は，偽アンチウイルスソフト (fake/crude) のとして判別される．その結果は表 5.4 の *verdict* に記載している．

表 5.4: **9** 回のリーベン検定の結果と提案手法による判別結果

Name	Class	C_1 & I_1	C_1 & I_2	C_1 & I_3	C_2 & I_1	C_2 & I_2	C_2 & I_3	C_3 & I_1	C_3 & I_2	C_3 & I_3	Total of ✓	Verdict
Avast	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
AVG	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
McAfee	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
NOD32	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
G Data	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
Norton	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
Kaspersky	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
Panda	Legitimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	Legitimate
Adware Bot	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Anti Spy Safeguard	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Anti-Spyware	Fake/Crude	✓	X	X	X	✓	✓	X	✓	✓	5	Fake/Crude
Anti Spyware Bot	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Anti-Virus Elite	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Anti Spyware Expert	Fake/Crude	✓	X	X	X	X	✓	X	X	✓	3	Fake/Crude
Error Sweeper	Fake/Crude	X	X	X	✓	X	X	X	X	X	1	Fake/Crude
Major Defense Kit	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Malware Removal Bot	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Netcom3	Fake/Crude	✓	X	✓	✓	X	✓	X	✓	X	5	Fake/Crude
Onescan	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Patchup Plus	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
PC Privacy Cleaner	Fake/Crude	X	✓	✓	X	✓	✓	✓	X	X	5	Fake/Crude
Peak Protection	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Pest Detector	Fake/Crude	X	X	✓	X	X	✓	X	X	✓	3	Fake/Crude
Privacy Control	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Red Cross	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Reg Clean	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Registry Smart	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Security Antivirus	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Protect Code	Fake/Crude	X	X	X	X	X	✓	X	X	✓	2	Fake/Crude
Security Defender	Fake/Crude	✓	X	X	✓	X	✓	✓	✓	✓	6	Fake/Crude
Security Shield	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Security Tool	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
System Security	Fake/Crude	✓	X	✓	✓	X	✓	X	✓	X	5	Fake/Crude
Virus Remover 2008	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
Virus Remover 2009	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
XL Guarder	Fake/Crude	X	X	X	X	X	X	X	X	X	0	Fake/Crude
XP AntiSpyware 2011	Fake/Crude	✓	✓	X	X	✓	X	✓	✓	X	5	Fake/Crude
XP AntiSpyware 2012	Fake/Crude	X	X	X	✓	X	✓	✓	✓	X	4	Fake/Crude
XP AntiVirus 2011	Fake/Crude	X	✓	X	✓	✓	✓	✓	✓	X	6	Fake/Crude
XP AntiVirus 2012	Fake/Crude	X	X	X	X	X	X	✓	✓	X	2	Fake/Crude
XP HomeSecurity 2011	Fake/Crude	X	X	✓	X	X	✓	X	X	X	2	Fake/Crude
XP HomeSecurity 2012	Fake/Crude	X	X	✓	X	X	✓	✓	✓	X	4	Fake/Crude
XP InternetSecurity 2011	Fake/Crude	X	X	X	X	X	X	✓	✓	X	2	Fake/Crude
XP InternetSecurity 2012	Fake/Crude	X	X	✓	X	X	✓	X	X	X	2	Fake/Crude
XP Security 2011	Fake/Crude	X	✓	✓	X	X	✓	✓	X	X	4	Fake/Crude
XP Security 2012	Fake/Crude	X	X	✓	X	X	X	✓	✓	X	3	Fake/Crude
XP TotalSecurity 2011	Fake/Crude	X	X	X	X	X	X	✓	✓	X	2	Fake/Crude

“✓” は統計的な有意差があることを表す．“X” は統計的な有意差がないことを示す．

正規のアンチウイルスソフトの判別

表 5.4 内の全ての正規のアンチウイルスソフト (表内上位 8 件) はいずれも ✓ の数が 9 つとなった．これは正規のアンチウイルスソフトの場合は infected environment と clean environment の環境から得られるメモリ使用量は常に統計的な有意差を生じるということを表しており，マルウェアを解析，検出する際に著しくメモリを消費することを意味している．つまり，clean environment から得られるメモリ使用量と infected environment から得られるメモリ使用量の分布は常に異なる．図 5.12 と

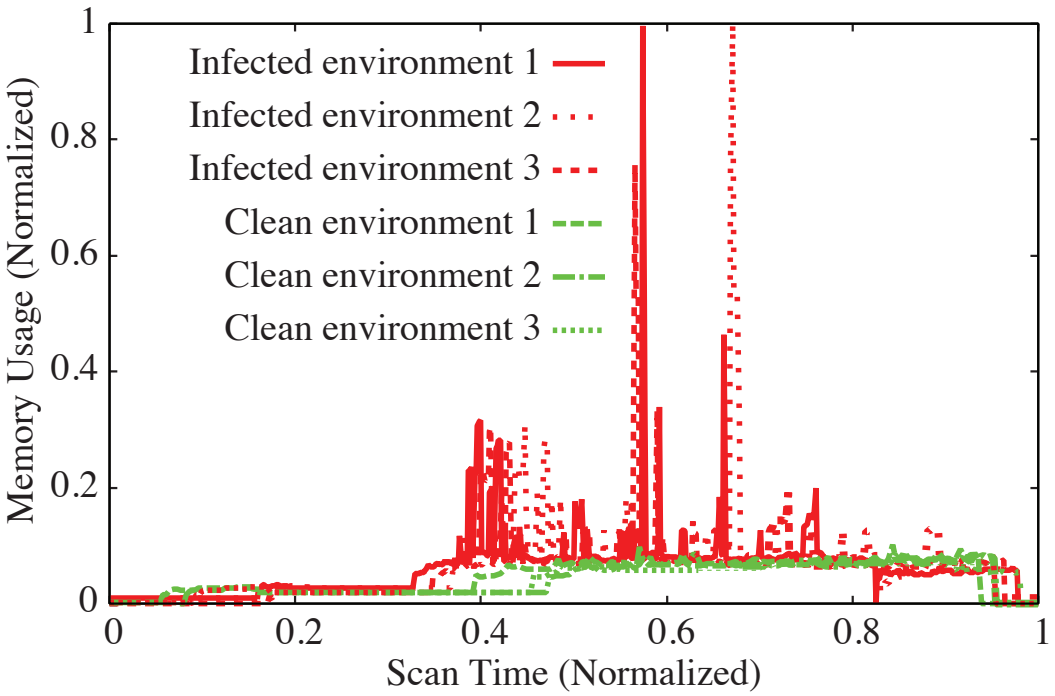

図 5.12: **AVG** のメモリ使用量 (正規のアンチウイルスソフト)．マルウェアを検出した際にメモリ使用量が増加している．

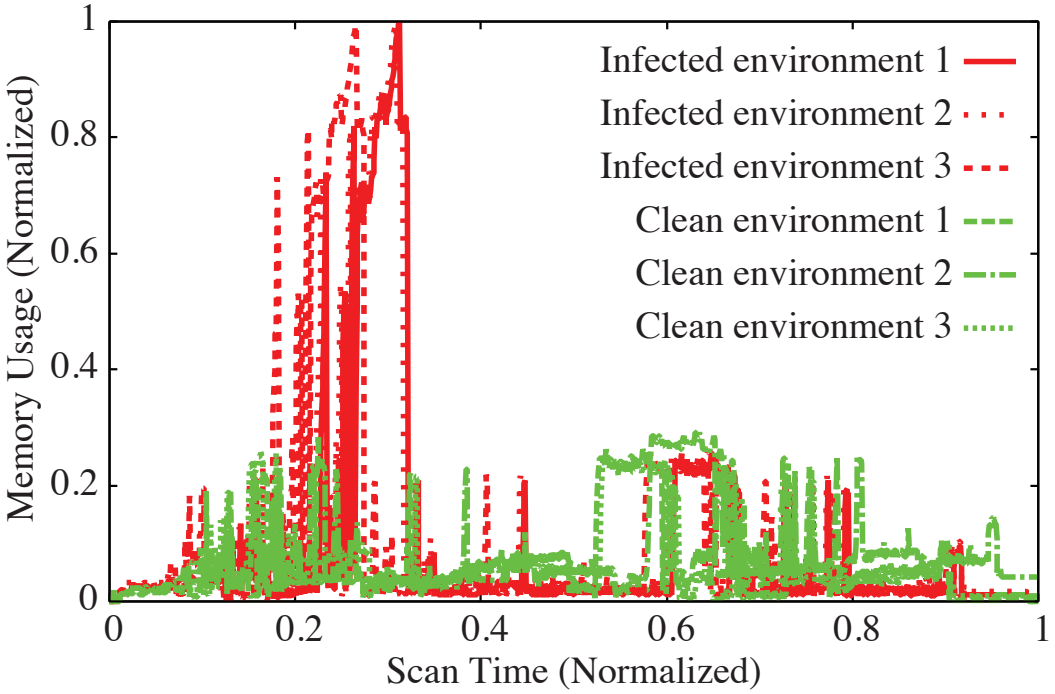

図 5.13: **McAfee** のメモリ使用量 (正規のアンチウイルスソフト)．Infected enviornment のみメモリ使用量が増加している．

図 5.13 は AVG と McAfee のメモリ使用量を表すグラフである．AVG と McAfee はどちらも infected environment，特に，マルウェアを検出した際に追加でメモリを割り当てられている．AVG と McAfee に加えて，他の 6 つの正規のアンチウイルスソフトも同様の傾向を示している．

偽アンチウイルスソフトの判別

偽アンチウイルスソフトについては，39 個中 19 個の偽アンチウイルスソフトは ✓ の数が 0 となり，39 個中 29 個は ✓ の数が 3 以下となった．そのため，ほとんどの偽アンチウイルスソフトは infected environment と clean environment の環境の違いによりメモリ使用量に変化がないことが分かった．図 5.14 は Red Cross のメモリ使用量を表すグラフであり，環境の違いによりメモリ使用量に大きな差がないことが分かる．図 5.15 は Pest Detector のメモリ使用量を表すグラフである．表 5.4 において Pest Detector の ✓ の数は 3 となっている．図 5.15 において，infected environment 3 のみ明らかに他のメモリ使用量と分布が異なっていることが分かる．その結果，infected environment 3 から得られたメモリ使用量と clean environment 1, 2, 3 から得られたメモリ使用量のペアはいずれも統計的な有意差を生じている．

図 5.16 と図 5.17 は ✓ の数がそれぞれ 5 と 6 となったサンプルのメモリ使用量のグラフを取り上げている．図 5.16 は粗悪なアンチウイルスソフトである Netcom3 のメモリ使用量である．粗悪なアンチウイルスソフトであっても，そのメモリ使用量の分布は正規のアンチウイルスソフトのように infected environment で極端に増減することはなく，メモリ使用量分布の変化は僅かに留まる程度であった．図 5.17 は XP AntiVirus 2011 のメモリ使用量である．図 5.14 や図 5.15 と比べると XP AntiVirus 2011 はメモリ使用量の計測の度に，その分布を変更していることが分かる．そのため，XP AntiVirus 2011 は実行するたびにメモリ使用量を変更していることが推測できる．

しかしながら，偽アンチウイルスソフトは提案手法をランダムなメモリ使用量を利用することにより回避することはできない．なぜなら，提案手法は複数回のリーベン検定を行っているからである．偽アンチウイルスソフトが提案手法を回避するためには正規のアンチウイルスソフトのように，infected environment に存在するマルウェアを正しく検出し，メモリ使用量を適切に変化させなければならない．

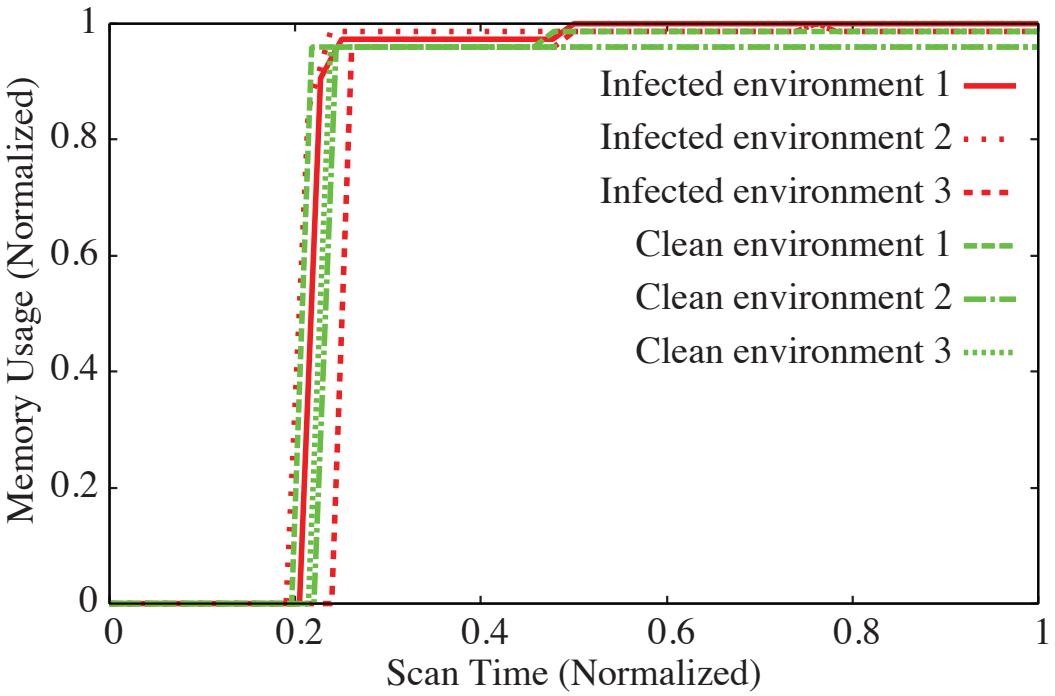

図 5.14: **Red Cross** のメモリ使用量 (偽アンチウイルスソフト)．環境の違いに関わらず，メモリ使用量はほぼ同一である．

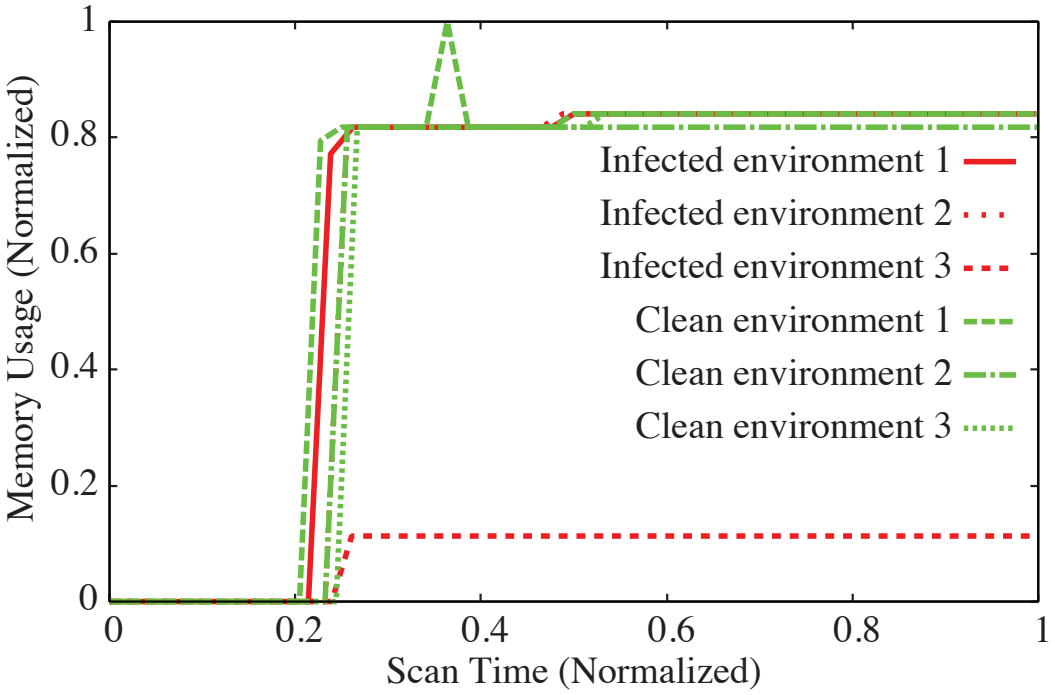

図 5.15: **Pest Detector** のメモリ使用量 (偽アンチウイルスソフト)．Infected environment 3 のみメモリ使用量が異なる．

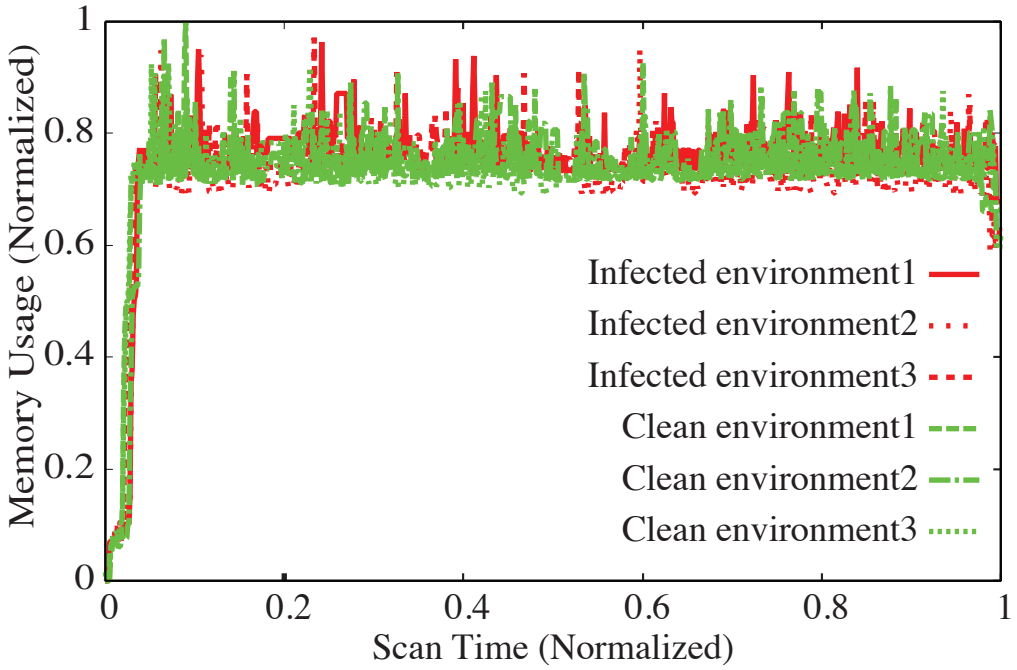

図 5.16: **Netcom3** のメモリ使用量 (粗悪なアンチウイルスソフト)．マルウェア検出能力を持つ粗悪なアンチウイルスソフトであっても正規のアンチウイルスソフトほどメモリ使用量は増加しない．

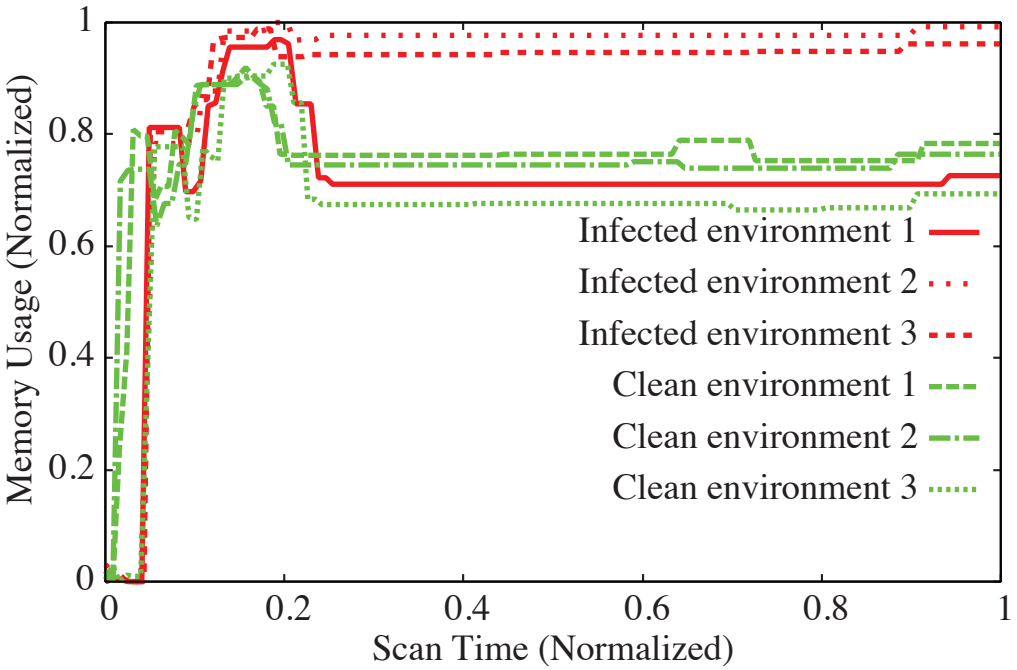

図 5.17: **XP AntiVirus 2011** のメモリ使用量 (偽アンチウイルスソフト)．計測を行う度にメモリ使用量を少しずつ変えている．

5.5 議論

5.5.1 回避手法に関する考察

既に述べてきたように，メモリ使用量をランダムに変更しても提案手法を回避することはできない．Infected environment と clean environment からそれぞれ 1 回ずつのみメモリ使用量を取得してリーベン検定を行う場合は，メモリ使用量をランダムに変えることにより偽アンチウイルスソフトは提案手法を回避することができてしまう．そのため，複数回のリーベン検定を実施することにより，ランダムなメモリ使用量による提案手法の回避を防いでいる．偽アンチウイルスソフトが提案手法の回避を試みる際には，infected environment と clean environment で得られるメモリ使用量の分布が完全に異なるようにする必要がある．結局それは正規のアンチウイルスソフトを作成しなければならない．

偽アンチウイルスソフトの作成者は Clam AV [93] といったオープンソースのアンチウイルスソフトや流出した正規のアンチウイルスソフトのソースコードを流用することにより，正規のアンチウイルスソフトと同様の振る舞いを実現できる．このような偽アンチウイルスソフトは提案手法を回避することができる．しかし，アンチウイルスソフトのベンダはこうした偽アンチウイルスソフトに素早く対処できる．なぜなら，アンチウイルスソフトのベンダはオープンソースや流出した正規のアンチウイルスソフトについて詳細な挙動を把握しているため，これらの偽アンチウイルスソフトに関するシグネチャを早期に作成できるためである．このような点から，オープンソースのアンチウイルスソフトや流出した正規のアンチウイルスソフトのソースコードを流用して偽アンチウイルスソフトを作成したとしても，サイバー犯罪者はその恩恵を十分に受けることができないと考えられる．

5.5.2 提案手法の利用場面に関する考察

提案手法を利用する場面の 1 つとして，ソフトウェアダウンロードサイトが候補として考えられる．CNET [89] や PCMAG [90] などのソフトウェアダウンロードサイトは多くのユーザが利用するため，このようなサイトで偽アンチウイルスソフトの配布を行うべきではない．そのため，ソフトウェアダウンロードサイトではセキュリティ専門期間の品質テストに合格した信頼できるアンチウイルスソフトを配布するべきである．しかしながら，現状ではそのような取り組みがなさ

れておらず，サイバー犯罪者は CNET を悪用して RegGeine と呼ばれる偽アンチウイルスソフトを 2012 年の 9 月に配布していた．こうした背景から，提案手法はソフトウェアダウンロードサイトが所有するアンチウイルスソフトらしき検体を配布してよいかどうかを検査するための方法として利用することにより，偽アンチウイルスソフトの配布を未然に防ぐといった使い方が考えられる．

ソフトウェアダウンロードサイトにおいて利用する場合，本手法が仮定する条件と合致するため親和性が高い．本手法は正規のアンチウイルスソフトと偽アンチウイルスソフトを判別する方法であり，検査対象はアンチウイルスソフトらしき検体に限定する必要がある．ソフトウェアダウンロードサイトは多くの場合，“AntiVirus” のように配布対象のソフトウェアをタグ付けしたインデックスを持つ．そのため，“AntiVirus” とタグ付けされたソフトウェアに限定して本手法を適用することにより，アンチウイルスソフトと関係ないソフトウェアを偽アンチウイルスソフトとする誤判別を避けることができる．

Rajab らは偽アンチウイルスソフトの配布サイトに関する調査を行った結果，偽アンチウイルスソフトは脆弱性のあるウェブサイトを悪用して，偽アンチウイルスソフトのバイナリの配信を行っているという報告をしている [38]．そのようなウェブサイトを検出する方法として deSEO [94] や SURF [95] などがある．これらは *search poisoning* と呼ばれるユーザを悪意あるウェブサイトに誘導する方法を検出する．提案手法はアンチウイルスソフトらしきプログラムのバイナリを正規のアンチウイルスソフトであるか偽アンチウイルスソフトかどうかを判別する方法のため，deSEO や SURF を利用してアンチウイルスソフトを配布しているサイトから得られたバイナリを判別するといった利用方法も考えられる．

5.5.3 他の判別指標に関する考察

本研究では指標の候補としてファイルアクセスパターン， CPU 利用率，メモリ使用量について調査して，メモリ使用量を利用する判別指標が最も精度良く正規のアンチウイルスソフトと偽アンチウイルスソフトを判別できることを確認した．しかしながら，本手法による判別は複数回メモリ使用量を取得する必要がある．偽アンチウイルスソフトのマルウェアスキャンに要する時間は種類により数秒から十数分になることが分かっているため，infected environment と clean environment をそれぞれ 1 つずつ用意して各環境から 3 回ずつメモリ使用量を集める場合，検体によっては 30 分以上の時間を要する．前処理の段階でメモリ使用量以外に正規

のアンチウイルスソフトと異なる振る舞いを示す偽アンチウイルスソフトを除去することができれば，複数回メモリ使用量を取得する必要はなくなるといえる．

そこで，前処理としてファイルアクセスパターンを利用する方法が考えられる．図 5.3 が示すように，実験で利用した 39 個の偽アンチウイルスソフトのうち，33 個の偽アンチウイルスソフトは正規のアンチウイルスソフトがアクセスするファイルにアクセスしていない．この点を考慮して，検査対象のアンチウイルスソフトが十分なファイルアクセスを伴わない場合は偽アンチウイルスソフトとして判別することにより，1 度のマルウェアスキャンのみで検査を行うことができると考えられる．この方法が前処理として有効であるかどうかはより多くの検体のファイルアクセスパターンを収集して議論を行う必要がある．

第6章　結論

6.1　本研究のまとめ

マルウェア対策はコンピュータセキュリティの重要な課題である．マルウェアの被害を防ぐために，アンチウイルスソフトによる対策，静的解析や動的解析を用いた振る舞い検出による対策を講じることによりマルウェアに対処している．しかし，現在のマルウェアはサイバー犯罪者が個人情報を収集したり，収益を得るための手段として利用されるため，極めて悪質に振る舞う．加えて，マルウェアの悪意ある振る舞いを防御側に察知されないようにするために，様々な工夫を行いマルウェア対策手法を回避する．特に，近年ステルス性の高いマルウェアが出現しており，これに対応する必要がある．ステルス性の高いマルウェアは頻繁な動作を避けたり，害のないソフトウェアの振る舞いを真似て動作することにより，防御側にできる限り悪質な振る舞いを示さないように工夫されたマルウェアである．そのため，ステルス性の高いマルウェアへの効率的な対策手法が必要とされるものの，既存の対策手法では，ステルス性の高いマルウェアに対応できていない．

本論文では，マルウェアの特徴的な振る舞いを誘発するアプローチにより，ステルス性の高いマルウェアに対処する 2 つの手法を提案した．本手法の核となる考え方はステルス性の高いマルウェアの振る舞いを活性化させるための作為的な環境を用意して，その環境でステルス性の高いマルウェアを動作させることにより，ステルス性の高いマルウェアがその特徴的な振る舞いを示さざるを得ないようにすることである．提案手法により，頻繁な動作を避けるマルウェアを活性化することができるため，その特徴的な振る舞いを取得することが容易となる．また，害のないソフトウェアの挙動を真似るようなマルウェアに対しても，害のないソフトウェアのみが活性化するような入力を作為的に与えることにより，ステルス性の高いマルウェアを揺さ振り出すことが可能となる．本論文では，マルウェアの特徴的な振る舞いを誘発するアプローチの概念をアドウェア及びスパイウェア，偽アンチウイルスソフトに適用することにより，その有用性を確認している．し

かしながら，提案手法のアプローチは特定のマルウェアに限定するものではなく，他の種類のマルウェアにも適用可能である．

まず，頻繁な動作を避けるステルス性の高いアドウェア及びスパイウェアを解析する具体的な方法として Blayzard を導入した．Blayzard はステルス性の高いアドウェアやスパイウェアはブラウザのアドオンとして実現されていることを利用して，ブラウザの状態を偽装したイベントを大量にブラウザのアドオンとして実現されたアドウェアやスパイウェアに注入することにより，作為的に特徴的な振る舞いを抽出し，解析するシステムである．Blayzard の有用性を確認するために，実在するアドウェアとスパイウェアを 32 個とより利用される害のないアドオンを 10 個収集して実験を行ったところ，偽のイベントを利用することにより，32 個のアドウェアやスパイウェア，10 個の害のないアドオンの振る舞いを解析することができた．加えて，Blayzard は滅多に動作しない悪質な振る舞いを効率よく抽出することができた．

次に，害のないソフトウェアの挙動を真似るマルウェアの 1 つである偽アンチウイルスソフトの判別手法を導入した．偽アンチウイルスソフトは正規のアンチウイルスソフトの挙動を模倣するマルウェアである．サイバー犯罪者は収益を得るために，偽アンチウイルスソフトを利用して，偽のマルウェア感染警告を表示して，その対策のために偽アンチウイルスソフトの製品版を購入することを促す．単純な振る舞い解析を行ったとしても偽アンチウイルスソフトは正規のアンチウイルスソフトの挙動を模倣しているため，その特徴的な振る舞いを抽出することは難しい．そこで，害のないソフトウェアとステルス性の高いマルウェアの違いを如実に表す振る舞いを誘発するために，敢えてマルウェアを利用した．正規のアンチウイルスソフトと偽アンチウイルスソフトの違いはメモリ使用量であることを実験的に示し，マルウェアが存在する環境と存在しない環境においてメモリ使用量に統計的な違いが生じるかどうかを調べることにより，正規のアンチウイルスソフトと偽アンチウイルスソフトの判別を行った．実験では，39 個の偽アンチウイルスソフトと 8 個の正規のアンチウイルスソフトを利用したところ，本判別手法は 39 個の偽アンチウイルスソフトを正しく偽物として判別して，8 個の正規のアンチウイルスソフトを正しく本物として判別することができた．

以上のように，マルウェアの特徴的な振る舞いを誘発するアプローチを 2 つの具体的な手法として実現し，その有効性の検証を行った結果，実在するステルス性の高いマルウェアに対処できることを示した．

6.2 今後の展望

第4章と第5章で示したマルウェアの特徴的な振る舞いを誘発してマルウェア対策を行う具体的な二つの方法はどちらも一般ユーザが利用することを想定していない．今後の課題の1つとして，提案手法の考え方を実現したマルウェア対策手法を一般ユーザが利用できるように研究を発展させる方向性がある．Blayzardの発展的な使い方として，ユーザがコンピュータを操作していないタイミングで，ステルス性の高いマルウェアの挙動を誘発する入力を与えて，マルウェアの検出を行うといった利用方法が考えられる．例えば，偽のイベントの注入により勝手にウェブページの遷移が発生する場合はアドウェアやスパイウェアによる影響のためと考えられる．また，第5章にて述べた方法も，マルウェア以外の入力を与えることにより，特徴的な振る舞いを誘発することができると考えられる．例えば，商用のアンチウイルスソフトはネットワークの監視を行っているため，害のないパケットと攻撃を模倣するパケットの2種類を挿入することにより，正規のアンチウイルスソフトと偽アンチウイルスソフトでは異なる振る舞いを示すことが予想できる．

謝辞

本論文は著者が慶應義塾大学大学院理工学研究科開放環境科学専攻の後期博士課程に在籍中の研究成果をまとめたものです．本研究を遂行するにあたり，また本論文をまとめるにあたり，多くの方々からご指導，ご協力を賜りました．お世話になった全ての方々にこの場を借りて御礼を申し上げます．

まず，本論文の主査であり，著者の指導教員である慶應義塾大学理工学部情報工学科の河野健二准教授に深く感謝いたします．河野健二准教授には，著者が学部 4 年時から修士，博士課程の合計 6 年間もの長きにわたり，筆記試験の良し悪しにより評価される世界だけしか知らなかった著者に，研究活動の意義，読み手を説得するための論文の執筆方法，聴衆に伝わるプレゼンテーションの作り方や口頭発表の方法など，基礎的なことを徹底的に指導していただき，何より研究活動の面白さを教えていただきました．また，学士と修士の学位を得て就職するものだと思っていた著者に，博士課程に進学するという全く考えてもいなかった道に進むきっかけを作ってくださりました．教育期間の最後の時期に河野健二准教授と共に研究を行い，成果を出せたことは著者にとって大きな財産となりました．加えて，著者の研究成果が学生論文賞，最優秀学生発表賞を受賞できたことは河野健二准教授の指導なしには決して達成できなかったことと確信しております．

次に，本論文の副査を担当していただいた，慶應義塾大学理工学部情報工学科の重野寛教授，高田眞吾准教授，及びシステムデザイン工学科の西宏章准教授に感謝いたします．副査のみなさまには貴重なお時間を割いていただき，本論文を丁寧に査読していただきました．副査のみなさまとの有意義な議論により，本論文の完成度が向上したと実感しております．深く感謝いたします．

嶋村誠博士，花岡美幸博士のお二人に感謝いたします．嶋村誠博士，花岡美幸博士には，著者が学部 4 年から修士 1 年の 2 年間にわたり，研究の方向性や実装に関する議論，発表資料の添削など様々な面からご指導をいただきました．嶋村誠博士からはリサーチプログラムに関して多くのことを学びました．嶋村誠博士から指導を受けることなしに，研究活動における実装の勘所を捉えることはでき

なかったと考えております．花岡美幸博士からは発表資料の添削を数多く行っていただきました．研究室に配属した直後，発表資料の作り方を把握していなかった時期に丁寧に指導していただきました．お二人より指導をいただけたことは著者が研究活動を遂行するにあたり大きな支えになったと感じております．

同時期に博士課程に進学した堀江光氏に感謝いたします．思うように成果が出ず，辛く苦しかった時期を乗り越えることができたのは，博士の学位取得に向けて厳しい道を共に歩いている同期がいたためと考えております．また，慶應義塾大学理工学部情報工学科河野研究室の皆様に感謝いたします．優秀な仲間から受ける刺激は著者の研究活動の励みとなりました．著者にとって河野研究室は教育期間の最後を過ごす場所として，これ以上ない環境であったと確信しております．

本論文の研究を遂行するにあたって使用した実験機材の一部，および原著論文の掲載，国際会議や国内学会の発表にあたっては科学技術振興機構 CREST の支援をいただきました．また，慶應義塾先端科学技術研究センターの KLL 後期博士課程研究助成金から本研究に対する支援をいただきました．特に，国際会議の発表や聴講は著者にとって大変有意義なものとなりました．

一般財団法人の慶応工学会による給費奨学金は著者の研究活動を遂行するにあたり大きな支えとなりました．修士課程 2 年から博士課程 3 年までの 4 年間にわたり奨学生として経済的なご支援をいただけたことにより，著者の経済的な不安を解消して，研究活動一つに集中できる環境を整えることができました．心より感謝いたします．

最後に，博士課程への進学を支持し，経済的な支援を惜しまず，現在まで暖かく見守っていただきました両親に心より感謝いたします．

参考文献

[1] Mary Landesman. Malware Revolution: A Change in Target. http://technet.microsoft.com/en-us/library/cc512596.aspx, Mar. 2007.

[2] Jason Franklin, Vern Paxson, Adrian Perrig, and Stefan Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 375–388, Nov. 2007.

[3] Marco Cova, Corrado Leita, Olivier Thonnard, Angelos D. Keromytis, and Marc Dacier. An Analysis of Rogue AV Campaigns. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID '10)*, pp. 442–463, Sep. 2010.

[4] Brett Stone-Gross, Ryan Abman, Richard A. Kemmerer, Christopher Kruegel, Douglas G. Steigerwald, and Giovanni Vigna. The Underground Economy of Fake Antivirus Software. In *Proceedings of the 10th Workshop on Economics of Information Security (online) (WEIS '11)*, Jun. 2011.

[5] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring Pay-per-Install: The Commoditization of Malware Distribution. In *Proceedings of the 20th USENIX Security Symposium*, pp. 187–202, Aug. 2011.

[6] Symantec. Techniques of Adware and Spyware. http://www.symantec.com/avcenter/reference/techniques.of.adware.and.spyware.pdf.

[7] McAfee Labs. McAfee Threats Report: Second Quarter 2013. http://www.mcafee.com/au/resources/reports/rp-quarterly-threat-q2-2013.pdf, 2013.

[8] Marc Fossi, Dean Turner, Eric Johnson, Trevor Mack, Teo Adams, Joseph Blackbird, Mo King Low, David McKinney, Marc Dacier, Angelos D. Keromytis,

Corrado Leita, Marco Cova, Jon Orbeton, and Olivier Thonnard. Symantec Report on Rogue Security Software. http://www4.symantec.com/Vrt/wl?tu_id=TeCm125590003756772344 [retrieved: Jul, 2013], Oct. 2009.

[9] François Paget. Running Scared: Fake Security Software Rakes in Money Around the World. http://www.mcafee.com/us/resources/white-papers/wp-running-scared-fake-security-software.pdf [retrieved: Jul, 2013], 2010.

[10] Thorsten Holz, Markus Engelberth, and Felix Freiling. Learning More about the Underground Economy: A Case-Study of Keyloggers and Dropzones. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS '09)*, pp. 1–18, Sep. 2009.

[11] McAfee Labs. McAfee Threats Report: Fourth Quarter 2009. http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2009.pdf, 2009.

[12] McAfee Labs. McAfee Threats Report: Fourth Quarter 2010. http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2010.pdf, 2010.

[13] McAfee Labs. McAfee Threats Report: Fourth Quarter 2011. http://www.mcafee.com/uk/resources/reports/rp-quarterly-threat-q4-2011.pdf, 2011.

[14] McAfee Labs. McAfee Threats Report: Fourth Quarter 2012. http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2012.pdf, 2012.

[15] McAfee Labs. McAfee Threats Report: Second Quarter 2013. http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2013.pdf, 2013.

[16] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Cloud AV: N-Version Antivirus in the Network Cloud. In *Proceedings of the 17th USENIX Security Symposium*, pp. 91–106, Jul. 2008.

[17] Mihai Christodorescu and Somesh Jha. Testing Malware Detectors. In *Proceedings of the 19th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '04)*, pp. 34–44, Jul. 2004.

[18] Automated reverse engineering: Mistfall engine. http://vxheaven.org/lib/vzo21.html.

[19] Frédéric Perriot, Péter Ször, and Peter Ferrie. Striking Similarites: Win32/Simile and Metamorphic Virus Code. http://www.symantec.com/avcenter/reference/striking.similarities.pdf.

[20] Masud Khafir. Trident Ploymorphic Engine. http://vxheaven.org/vx.php?id=et06.

[21] Symantec. Cybercrime's Financial and Geographic Growth Shows No Slowdown during the Global Economic Crisis. http://www.symantec.com/about/news/release/article.jsp?prid=20100419_02, Apr. 2010.

[22] Christopher Kruegel, William Robertson, and Giovanni Vigna. Detecting Kernel-Level Rootkits Through Binary Analysis. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04)*, pp. 91–100, Dec. 2004.

[23] Johannes Kinder, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. Detecting Malicious Code by Model Checking. In *Proceedings of the 2nd international conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '05)*, pp. 174–187, Jul. 2005.

[24] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. Semantics-Aware Malware Detection. In *Proceedings of the 26th IEEE Symposium on Security and Privacy (S&P '05)*, pp. 32–46, May. 2005.

[25] Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Song. Dynamic Spyware Analysis. In *Proceedings of USENIX Annual Technical Conference*, pp. 233–246, Jun. 2007.

[26] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 116–127, Nov. 2007.

[27] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and Efficient Malware Detection at the End Host. In *Proceedings of the 18th USENIX Security Symposium*, pp. 351–366, Aug. 2009.

[28] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *Proceedings of the 20th USENIX Security Symposium*, pp. 443–458, Aug. 2011.

[29] Matt Fredrikson, Somesh Jha, Mihai Christodorescu, Reiner Sailer, and Xifeng Yan. Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors. In *Proceedings of the 31st IEEE Symposium on Security and Privacy (S&P '10)*, pp. 45–60, May. 2010.

[30] Heng Yin, Zhenkai Liang, and Dawn Song. HookFinder: Identifying and Understanding Malware Hooing Behaviors. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS '08)*, Feb. 2008.

[31] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of Static Analysis for Malware Detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC '07)*, pp. 421–430, Dec. 2007.

[32] Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A Crawler-based Study of Spyware on the Web. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS '06)*, Feb. 2006.

[33] Kaspersky Lab. Monthly Malware Statics, October 2010. http://www.securelist.com/en/analysis/204792146/Monthly_Malware_Statistics_October_2010, 2010.

[34] Kaspersky Lab. Monthly Malware Statics, September 2010. http://www.securelist.com/en/analysis/204792141/Monthly_Malware_Statistics_September_2010, 2010.

[35] Kaspersky Lab. Monthly Malware Statics, August 2010. http://www.securelist.com/en/analysis/204792135/Monthly_Malware_Statistics_August_2010, 2010.

[36] Kaspersky Lab. Monthly Malware Statics, July 2010. http://www.securelist.com/en/analysis/204792130/Monthly_Malware_Statistics_July_2010, 2010.

[37] McAfee Labs. Mobile Security: McAfee Consumer Trends Report. Trends in risky apps, mobile misbehavior, and spyware. http://www.biztositasiszemle.hu/files/201302/rp-mobile-security-consumer-trends.pdf, 2013.

[38] Moheeb Abu Rajab, Lucas Ballard, Panayiotis Mavrommatis, Niels Provos, and Xin Zhao. The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution. In *Proceedings of the 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '10)*, Aug. 2010.

[39] Yi-Min Wang, Roussi Roussev, Chad Verbowski, Aaron Johnson, Ming-Wei Wu, Yennun Huang, and Sy-Yen Kuo. Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management. In *Proceedings of the 18th Large Installation System Administration Conference (LISA '04)*, pp. 33–46, Nov. 2004.

[40] Howard Levene. *Robust Tests for Equality of Variances*. Stanford University Press, 1960.

[41] NORTON - Antivirus Software and Spyware Removal. http://us.norton.com/.

[42] McAfee. McAfee - Antivirus, Encryption, Firewall, Email Security, Web Security, Risk & Compliance. http://www.mcafee.com/us/.

[43] G DATA. G Data. Security Made in Germany. G Data Software, Inc. http://www.gdata-software.com/.

[44] Kaspersky Lab. Kaspersky Lab Antivirus Protection & Internet Security Software. http://www.kaspersky.com/.

[45] Trend Micro. Content security software - Internet Security & Cloud - Trend Micro USA. http://www.trendmicro.com/us/index.htm.

[46] F-Secure. Home F-Secure. http://www.f-secure.com/en/web/business_us.

[47] Panda Security. Panda Security, the Cloud Security Company. http://www.pandasecurity.com/.

[48] Avast. AVAST 2014 Download Free Antivirus Software for Virus Protection. http://www.avast.com/en-us/index.

[49] AVG. AVG Free Antivirus. http://free.avg.com/us-en/homepage.

[50] ESET. ESET Antivirus, Internet Security Software & Virus Protection. http://www.eset.com/us/.

[51] William Arnold and Gerald Tesauro. Automatically Generated Win32 Heuristic Virus Detection. In *Proceedings of the 10th Virus Bulletin Conference*, pp. 51–60, Sep. 2000.

[52] Kent Griffin, Scott Schneider, Xin Hu, and Tzi cker Chiueh. Automatic Generation of String Signatures for Malware Detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID '09)*, pp. 101–120, Sep. 2009.

[53] Hex-Rays. IDA Pro Disassembler - multi-processor, windows hosted disassembler and debugger. http://www.hex-rays.com/idapro/.

[54] Mihai Christodorescu and Somesh Jha. Static Analysis of Executables to Detect Malicious Patterns. In *Proceedings of the 12th USENIX Security Symposium*, pp. 169–186, Aug. 2003.

[55] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of USENIX Annual Technical Conference*, pp. 41–46, Apr. 2005.

[56] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. TTAnalyze: A Tool for Analyzing Malware. In *Proceedings of the 15th European Institute for Computer Antivirus Research Conference (EICAR)*, Apr. 2006.

[57] Dawn Song, David Brumley, 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 (ICISS)*, pp. 1–25, Dec. 2008.

[58] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy*, Vol. 5, No. 2, pp. 32–39, 2007.

[59] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. BareBox: Efficient Malware Analysis on Bare-Metal. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*, pp. 403–412, Dec. 2011.

[60] William Landi. Undecidability of Static Analysis. *ACM Lett. Program. Lang. Syst.*, Vol. 1, pp. 323–337, December 1992.

[61] G. Ramalingam. The Undecidability of Aliasing. *ACM Trans. Program. Lang. Syst.*, Vol. 16, pp. 1467–1471, Septermber 1994.

[62] Peter Ferrie. Attacks on Virtual Machine Emulators. In *Proceedings of the 9th Annual Anti-Virus Asia Researchers Conference*, Dec. 2006.

[63] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. Detecting System Emulators. In *Proceedings of the 10th Information Security Conference (ISC '07)*, pp. 1–18, Oct. 2007.

[64] Amit Vasudevan and Ramesh Yerraballi. Cobra: Fine-grained Malware Analysis using Stealth Localized-executions. In *Proceedings of the 27th IEEE Symposium on Security and Privacy (S&P '06)*, pp. 264–279, May. 2006.

[65] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: Malware Analysis via Hardware Virtualization Extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08)*, pp. 51–62, Oct. 2008.

[66] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. Emulating Emulation-Resistant Malware. In *Proceedings of the Workshop on Virtual Machine Security (VMSec '09)*, pp. 11–22, Nov. 2009.

[67] Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Efficient Detection of Split Personalities in Malware. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS '10)*, Mar. 2010.

[68] Clemens Kolbitsch, Engin Kirda, and Christopher Kruegel. 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), pp. 285–296, Oct. 2011.

[69] Open Malware. Open Malware — Community Malicious code researh and analysis. http://offensivecomputing.net [retrieved: Jul, 2013].

[70] VX Heavens. Welcome to VX Heavens! (VX Heavens). http://vx.netlux.org.

[71] Microsoft. The Component Object Model. http://msdn.microsoft.com/en-us/library/ms694363(VS.85).aspx.

[72] MSDN Library. DWebBrowserEvents. http://msdn.microsoft.com/en-us/library/aa768309(v=vs.85).aspx.

[73] MSDN Library. DWebBrowserEvents2. http://msdn.microsoft.com/en-us/library/aa￥￥768283(v=vs.85).aspx.

[74] Zhuowei Li, XiaoFeng Wang, and Jong Youl Choi. SpyShield: Preserving Privacy from Spy Add-Ons. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID '07)*, pp. 296–316, Sep. 2007.

[75] Lavasoft AB. Ad-Aware. http://www.lavasoft.com/.

[76] Spybot Search & Destroy. http://www.safer-networking.org/en/home/index.html.

[77] Snort. http://www.snort.org/.

[78] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proceedings of the 7th USENIX Security Symposium*, pp. 31–52, Jan. 1998.

[79] VMware. http://www.vmware.com.

[80] Tobias Klein. ScoopyNG - The VMware detectioin tool. http://www.trapkit.de/research/vmm/scoopyng/index.html.

[81] Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: How to automatically generate procedures to detect CPU emulators. In *Proceedings of the 3rd USENIX Conference on Offensive Technologies (WOOT '09)*, Aug. 2009.

[82] Joanna Rutkowska. Red Pill... or how to detect VMM using (almost) one CPU instruction. http://www.invisiblethings.org/papers/redpill.html, 2004.

[83] Sophos. What is FakeAV. http://www.bt.bt/forms/sophos-what-is-fakeav-wpna.pdf [retrieved: Jul, 2013], May. 2010.

[84] Abhishek Karnik, Jr. Avelino C. Rico, Amith Prakash, and Shinsuke Honjo. Identifying Fake Security Products. http://www.mcafee.com/us/resources/white-papers/wp-identifying-fake-security-products.pdf [retrieved: Jul, 2013], 2009.

[85] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All Your iFRAMEs Point to Us. In *Proceedings of the 17th USENIX Security Symposium*, pp. 1–16, Jul. 2008.

[86] Sean Doyle. How To Remove RegGenie Rogue Antivirus Software – Uninstall RegGenie Malware (Identity Theft Protection). http://botcrawl.com/how-to-remove-reggenie-rogue-antivirus-software/ [retrieved: Jul, 2013], Sep. 2012.

[87] Ugnius Kiguolis. Remove Security Antivirus. http://www.2-spyware.com/remove-security-antivirus.html [retrieved: Jul, 2013], Mar. 2010.

[88] Virus Total. https://www.virustotal.com [retrieved: Jul, 2013].

[89] CNET | Download.com. http://download.cnet.com [retrieved: Jul, 2013].

[90] PCMAG.COM. http://www.pcmag.com/downloads [retrieved: Jul, 2013].

[91] Fakeavs (dashke's blog). http://www.fakeavs.com [retrieved: Jul, 2013].

[92] MDL. Malware Domain List. http://www.malwaredomainlist.com/ [retrieved: Jul, 2013].

[93] Clam AntiVirus. http://www.clamav.net/lang/en/.

[94] John P. John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martîn Abadi. deseo: Combating search-result poisoning. In *Proceedings of the 20th USENIX Security Symposium*, pp. 299–313, Aug. 2011.

[95] Long Lu, Roberto Perdisci, and Wenke Lee. SURF: Detecting and Measuring Search Poisoning. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*, pp. 467–476, Oct. 2011.

論文目録

定期刊行誌掲載論文

- Masaki Kasuya, Kenji Kono: “Screening Legitimate and Fake/Crude Antivirus Software”, *IPSJ Transactions on Advanced Computing System (ACS 45)*, To Appear.

- 糟谷 正樹，河野 健二: “ブラウザのアドオンを利用したアドウェアのイベント注入による振舞い解析”，情報処理学会論文誌，Vol.52, No.12, pp.3775-3785, Dec. 2011. (情報処理学会コンピュータセキュリティシンポジウム 2010 推薦論文)

国際会議論文

- *Masaki Kasuya, Kenji Kono: “Distinguishing Legitimate and Fake/Crude Antivirus Software”, In *Proceedings of the 7th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE ’13)*, pp.109–116, Aug. 2013.

国内学会発表

- *Masaki Kasuya, Kenji Kono: “An Analysis of Fake Antivirus Behaviors”, 第 120 回システムソフトウェアとオペレーティング・システム研究会報告，Vol.2012-OS-120, No.6, 9 pages, Feb. 2012. (最優秀学生発表賞受賞)

- *糟谷 正樹，河野 健二: “ブラウザのアドオンを利用したアドウェアの振る舞い解析”，第 13 回コンピュータセキュリティシンポジウム (CSS 2010), pp.651–656, Oct. 2010. (学生論文賞受賞)