バックドア検知技術の調査と今後の展望

嶋田 有佑[1,*] 佐々木 貴之[1]

概要：近年，インフラや企業システムを構成する際に，単一の企業のデバイスやソフトウェアだけでなく，外部の企業が製造したものを組み合わせるようになり，外部調達したソフトウェアからバックドアが発見されるインシデントが多数報告されている．バックドアの検出は簡単ではない．なぜならバックドアは通常のソフトウェアの一部に組み込まれた不正な機能であり，その有無を検査するにはソフトウェア全体を静的解析する必要がある．また特定の条件でのみ起動するため，単純な動的解析も効果的でない．さらに，ソフトウェアごとに組み込まれるため，別のソフトウェアで使用したものを再利用することが少なくブラックリスト化が難しい．現在までにバックドアを検知する手法がいくつか研究されてきたが，研究者によりバックドアの定義や分類，検知手法が対象とする種類は様々である．本論文では，ソフトウェアに挿入されたバックドアに対する検知手法を調査し，バックドアの定義と分類，構成要素を整理した．また各手法が対象とするバックドアの種類や制約を整理した．整理の結果，すべての種類を検出できる単一の手法は無く，複数の手法の組み合わせが必要であることが分かった．

キーワード：バックドア，検知

A Survey on Backdoor Detection and Future Work

Yusuke Shimada[1,*] Takayuki Sasaki[1]

Abstract: In recent years, a number of incidents about backdoors embedded in software have been reported. Unfortunately, backdoor detection is a challenging and unsolved problem. The reason is that we must statically analyze the whole software because a backdoor is just a part of the benign software. Moreover, a simple dynamic analysis is ineffective because the backdoor can be activated only when a specific condition is satisfied. Some researchers have tackled these problems. However, definitions and classifications of backdoors are different between researchers, and also they targeted different types of backdoors. In this paper, we survey backdoor detection techniques. We first introduce their definitions and classifications of backdoors, and we explain the common backdoor components. Next, we compare the backdoor detection methods and their limitations. As a result, we clarify that there is no silver bullet that can detect all backdoor types, and combination of detection methods is required.

Keywords: backdoor, detection

1. はじめに

インターネットルーターやネットワークスイッチ，センサーやアクチュエータなどの組込みデバイスは，日常の暮らしを支えるインフラの一部や，企業システムの一部として重要な役割を担っている．近年，このようなインフラや企業システムは複雑化しており，単一の企業のデバイスだけで構成されるのではなく，様々な企業のデバイスを外部から調達し，それらを組み合わせて構築されている．デバイスをシステムに組み込む際に，我々はデバイスの製造者及び製造・流通チェーンを信頼している．

しかしながら，ここ数年間でこれらの組込みデバイスにおいて，ソフトウェア（またはファームウェア）およびハードウェアの両面でユーザが認知していない隠された機能や予期しない機能が発見されるインシデントが多数報告されている．従って，デバイスの製造者及び製造・流通チェーンが信頼できるという前提が成り立たなくなってきた．多くの場合，こうしたユーザの認知していない追加的な機能のことをバックドアと呼んでいる．バックドアは，ハードウェアに埋め込まれるものと，ソフトウェアに埋め込まれるものに大別できる．前者の場合，ハードウェアとして埋め込まれるため，攻撃者は高度な技術に加え，国家単位での関与あるいはチップの製造者という特殊な立場が要求される．防御の観点から考えると，ハードウェアは外部からの解析が困難であるため，バックドアがハードウェアに挿入された場合は検知が難しい．ソフトウェアに組み込まれたバックドアの場合，攻撃者はハードウェアのバックドアよりも容易に埋め込むことが可能であると考えられる．ソフトウェアのバックドアの検知は，ハードウェアと同様に困難であるが，いくつかの検知手法が提案されている．

本論文では，ソフトウェアのバックドアに対する検知手法を調査し，各手法が対象とするバックドアの種類や手法の制約について整理する．具体的には，ソフトウェアに挿入されるバックドアの分類として次に示すものが一般的であり，主にこの分類に基づいて検知手法の整理を行った．

- 認証機構の回避を行うもの

1 NEC セキュリティ研究所
Security Research Laboratories, NEC Corporation.
* y-shimada@gj.jp.nec.com

- 公開されていない，あるいは隠されたコマンドや機能
- バックドアを起動するための脆弱性

一概にバックドアと言っても，研究者により対象としているバックドアの種類は異なり，バックドアの定義および分類も様々である．そこでまず 2 章では，バックドアに関してどのような定義あるいは分類がされてきたのかを紹介し，次に Thomas らによるバックドアの定義を紹介する．3 章では，Thomas らによるバックドアの構成定義に基づいて，バックドアの共通する構成要素について説明する．そして 4 章では，バックドアに対する既存の検知手法を紹介する．5 章では，既存の検知手法の分類と制約について述べる．6 章では，既存の検知手法を分類，整理した結果に基づき，バックドアの検査手法についての今後の展望を述べる．7 章で本論文をまとめる．

2. バックドアの定義および分類

この章では既存研究においてバックドアがどのように定義および分類されてきたのかを述べる．まず Zhang らをはじめとした研究者たちによるバックドアに対する略式の定義と分類を紹介し，次に Thomas らによる定義を紹介する．Thomas らは [1]において，バックドアは基本的に 4 つの構成要素に分解できると述べている．

2.1 既存研究におけるバックドアの定義

Zhang らは [2]で，バックドアを次のように定義している．

「バックドアとは，システムに対して不正アクセスを可能にするため秘かに入れられた機構のことである．」

後述する通り，Zhang らはネットワークセキュリティの観点からのみバックドアの検知を検討している．この論文でのバックドアの定義はやや抽象度が高いが，他の著者による定義との共通部分が多い一般的な定義である．

Wysopal らは [3]の中で，バックドアを次のような 3 種類に分類している．

- システムバックドア

データやプロセスに対し，システムレベルのアクセスを可能にするバックドアで，ルートキットやリモートアクセスソフトウェア，さらには攻撃者による意図的なシステムの誤設定がこのタイプである．

- アプリケーションバックドア

正規のソフトウェアが改変され，ある条件下でセキュリティ機構を回避できてしまうバックドア．

- 脆弱な暗号技術によるバックドア

意図的に弱い暗号鍵やメッセージを作成し，攻撃者がクリアテキストにアクセスできるようにするバックドア．

本論文で紹介する検知手法では，主に組込みデバイスのファームウェアに対するバックドアを対象としており，それらは総じて Wysopal らの述べる「アプリケーションバックドア」に分類される．

Schuster らは [4]で，バックドアを次の通り定義している．

「バックドアとは，隠された，仕様書にない，あるいは望まれないプログラムやその変更・操作である．そして何かしらのトリガーにより，備わっているセキュリティ機構を回避する，あるいはユーザの望まない，知らされていない悪意のある行動をする．」

Schuster らの定義は Zhang らの定義よりも具体的であり，現在知られているバックドア要素の大部分を記述している．

Shoshitaishvili らは [5]の中で，認証回避の脆弱性がバックドアの一つのカテゴリであると述べている．後述する通り，Shoshitaishvili らが提唱した Firmalice というツールでは，このカテゴリのバックドアに対してのみ検知を行うため，他のバックドアの分類および定義は行っていない．

Papp らは [6]の中で，特定の入力に対して仕様書に記載のない機能を実行する「トリガーに起因する動作」にバックドアとロジックボムが含まれると述べている．一般的にバックドアは常時動いているプログラムではなく，一定の条件を満たした際に初めて起動するプログラムであり，起動するための入り口のことをトリガーと呼ぶ．そしてこうした動作のことを「トリガーに起因する動作」という．バックドアの構成要素の中でもトリガーは極めて重要であり，Papp らはバックドアの定義や分類は行っていないものの，トリガーに起因する動作にはあらゆるバックドアの種類が含まれることを示唆している．

Tien らが [7]で言及しているバックドアは ssh や telnet などの遠隔ログインインターフェースに含まれるものとしており，やや限定的である．またバックドアが挿入されるタイミングとして組込みシステムベンダによる開発段階に限定している．

Thomas らは [1]で，バックドアを次の通り定義している．

「バックドアは，システムに含まれる意図的な構造物で，本来は権限が必要な機能や情報にアクセスできるようにすることで，期待されるセキュリティ機構を回避してしまう．」

定義自体は Zhang らと同様にやや抽象的であるが，Thomas らによるバックドアの構成要素への分解はバックドアを理解し，その検知手法を模索する上で極めて参考になる．Thomas らはバックドアの実装において，次のように 4 つの構成要素へ分解できることを述べている．

- 入力元

バックドアを起動する上で，ユーザによる入力などの何かしらのパラメータを必要とする．

- トリガー機構

入力に対して何かしらの操作を行い，バックドアを起動するかどうかを決定する．

- ペイロード

起動したバックドアが実際に実行する操作部分である．

- 権限を持った状態

多くのバックドアでは，最終的に権限のない攻撃者が権限を持った状態に至る．

表 1　研究者によるバックドアの様々な定義及び分類

Table 1　Various definitions and classifications of backdoors.

	定義および分類
Zhang ら	やや抽象度の高い，バックドア全般に共通する概念を定義している
Wysopal ら	バックドアを 3 種類に分類し，それぞれの定義を行っている
Schuster ら	バックドア全般に対して，挙動の面からより具体的な定義を行っている
Shoshitaishvili ら	認証回避の脆弱性をバックドアの一つのカテゴリと定義している
Papp ら	「トリガーに起因する動作」の一種としてバックドアを位置付けている
Tien ら	遠隔ログイン用インターフェースなどバックドアの対象が限定されている
Thomas ら	定義自体の抽象度はやや高いが，バックドアが 4 つの構成要素から成ることを示している

3. バックドアの構成

3.1 入力元

図 1 のように，バックドアのトリガー機構をある一つの関数と考えると，引数に取るパラメータの内の一つが入力元となる．入力元は，バックドアトリガーをアクティブにする値ではなく，それらを受け取るものであり，ネットワークソケットや標準入力，システムクロックなどである．

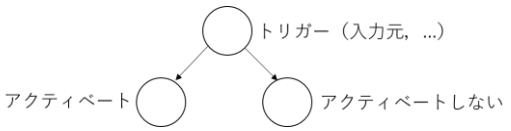

図 1　単純化したバックドアのトリガー

Figure 1　A simple backdoor trigger mechanism.

3.2 トリガー機構

バックドアのトリガーは一定の条件を満たした際，バックドアのペイロード部分の実行を引き起こし，このペイロードにより攻撃者は権限昇格などを実現する．トリガーの挙動はいくつかのパターンが存在する．図 1 のようにバックドアのトリガーをある boolean 関数と考えると，関数の返り値が真の場合にペイロードへと明示的に状態遷移することもあれば，図 2 のように，通常の CFG には現れないようなペイロードへの遷移も存在する．バッファオーバーフローなどの脆弱性をエクスプロイトすることで，トリガー関数のリターンアドレスを書き換えることなどにより，ペイロードへ正常でない遷移が起こる場合である．こうした状態遷移は制御フローにおいて，明示的でない遷移である．

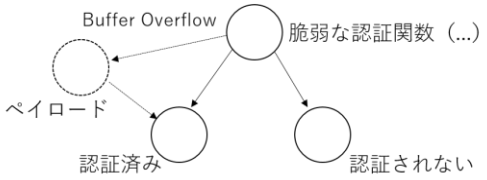

図 2　脆弱性を利用したバックドアのトリガー

Figure 2　A bug-based backdoor trigger mechanism.

3.3 ペイロード

トリガーの条件を満たしたのちに，本来権限のない攻撃者がどのように権限を持った状態へと至るのかを解決するための手段がこのペイロード部分である．ペイロードへ明示的な遷移をするのか，そしてペイロード自体が制御フローにおいて，明示的に存在しているかによりいくつかのパターンが存在する．

図 3 の例では，トリガーはハードコードされた認証情報に対するチェックであり，この結果が真であった場合に，明示的にペイロードへ遷移する．またその後の実行部分であるペイロードも明示されている．

```
/* Trigger */
if (strcmp(user._name, "backdoor") == 0) {
    // Payload
    user._is_admin = true;
    /* Transition to privileged state */
    open_shell(&user);
}
```

図 3　明示的ペイロードへ明示的な遷移（ [1]より引用）

Figure 3　Explicit transition to the explicit payload.

ROP ベースのペイロードを持つバックドアでは，攻撃者の入力によって遷移先が異なるため，明示的ではない遷移が起こる．Return-oriented Programming (ROP)とは，ROP ガジェットと呼ばれる ret 命令で終わる命令列の先頭へジャンプを繰り返すことで，任意の命令列を実行する手法である．図 4 では脆弱性のあるトリガーをエクスプロイトすることにより制御フローをハイジャックし，ペイロードへ明示的でない遷移する．その後，明示的あるいは明示的でないペイロードにより攻撃者は権限を与えられた状態へと至る．

```
void some_function() {
    char buf[80];
    /* Trigger */
    strcpy(buf, input);
    return;
}
void other_function() {
    g_user._is_admin = true;
    open_control_panel();
}
```

図 4　ペイロードへの明示的でない遷移（ [1]より引用）

Figure 4　Non-explicit transition to the payload.

他のケースとして，ペイロードが単体の遷移となっている場合がある．このようなケースでは通常の認証以外にバックドアのトリガーとなる認証を回避するための機構を用意していることが多い．

3.4 権限を持った状態

トリガーによるバックドアのアクティベートとペイロードからの遷移により,システムは権限を持った状態へと至る.この状態へは，正常なユーザによる通常の実行によっても至ることができる場合と，バックドアのトリガーによってのみ至ることができる場合がある.

4. バックドアに対する検知手法

この章ではバックドアに対する既存の検知手法を紹介する.

4.1 Zhang らによる検知手法

Zhang ら [2]はネットワークセキュリティの観点からバックドアを検知する手法を提案している．Zhang らが検知の対象としているのは「インタラクティブバックドア」と呼ばれるもので，攻撃者はキーストロークを通してバックドアとやり取りを行う．検知手法としては IDS によりネットワークのトラフィックを監視し，接続の方向性（インバウンド接続かアウトバウンド接続か)，パケットサイズ，パケットのインターバルタイムを調べることでインタラクティブバックドアを発見する．また，SSH や Rlogin，Telnet，FTP など一般的によく利用されるサービスに対してはプロトコル特有のアルゴリズムも考案している.

4.2 Wysopal らによる検知手法

Wysopal ら [3]はアプリケーションバックドアに対する静的検知のアプローチを提案している．アプリケーションバックドアには「特別な認証情報」「隠し機能」「意図的でないネットワークアクティビティ」「セキュリティ上重要なパラメータの細工」の４種類が存在するとしている.

「特別な認証情報」とは，攻撃者が特別な認証情報とそれに対する認証ロジックをプログラム内に仕込んでおくものである.認証情報としては特別なユーザ名やパスワード,パスワードのハッシュ値などが存在する．認証ロジックとしては主にこれらの特別な認証情報と比較する機構である.「特別な認証情報」を検知する手法としては，特定の意味を示していそうな静的な変数を見つけることで，具体的にはユーザ名やパスワード,ハッシュ値,暗号鍵などである.

「隠し機能」は，攻撃者が隠しコマンドを実行したり，正規の認証を通らずに認証されたりするもので，特別なパラメータを使用することで，そうした特殊なロジックをトリガーする．他にも仕様書に記載のないコマンドや，残されたデバッグ用コードなどがこのタイプのバックドアである.「隠し機能」は特別な IP のチェックと併用することで，他のユーザがバックドアを使用できないようにすることもある．この「隠し機能」に分類されるバックドアは様々なタイプが存在するので，静的な検知手法としてはそれぞれに特有の観点から分析を行う必要がある．例えば，何かしらのコマンドを実行するという観点から，C 言語での exec 系関数や system 関数，PHP 言語の popen 関数や eval 関数などに注目する，あるいはアプリケーションのコマンドのような静的な変数を調べたりすることが有効である.

「意図的でないネットワークアクティビティ」は代表的なバックドアの特徴の一つであり，いくつかの技術の総称であるが，仕様書に記載のないポートで listen している，外部の攻撃者がバックドアを含むノードに対して実行させるコマンドを送り込むために，アウトバウンドな command & control チャネルを確立させる，あるいは様々なプロトコルを利用してネットワーク越しに機密情報を漏洩させるなどである．検知手法としては，ネットワークユーティリティを用いた動的検知がメインとなるが，静的検知も可能である．具体的には，ハードコードされた IP アドレスやポートに対してアウトバウンド接続を確立しようとしているものに注意を払い，分析を行うなどである.

「セキュリティ上重要なパラメータの細工」は，システムのセキュリティ上重要であるパラメータを直接細工する,あるいはそれらのパラメータとの比較に欠陥のあるロジックを入れることで，トリガーを知る攻撃者は自身の権限レベルを引き上げるなどできる．こうしたパラメータは OS カーネルやアプリケーションの中に様々に存在するため，一概に検知手法を述べることは難しい.

総じてこの論文では，特定のバックドアに対して，事前にそれぞれ固有のパターンを特定しておく必要があることを示唆している.

4.3 Schuster らによる検知手法

Schuster ら [4]はクライアント・サーバーモデルにおけるサーバーアプリケーションに対して，バックドアを自動的に検知する手法を提案している．対象のバックドアは「欠陥のある認証ルーチン」と「隠しコマンド，隠し機能」である．前者は，認証機構を完全に回避してしまうためのコードをバイナリに含む場合である．こうした場合に，バックドアが混入していないバージョンのバイナリが入手できれば，過去の研究 [8] [9]のように CFG を調べ，追加されている部分を分析することにより検知できる可能性がある．しかし一般にこうしたバージョンのバイナリが入手できるケースは少ないため，この論文においては，バイナリ単体を対象にバックドアを検知する手法を提案している.

手順としては，まずバイナリの中から攻撃に利用されやすい特定の部分を自動的に見つけ出す．サーバーアプリケーションにおいては，認証ルーチンとコマンドディスパッチング機能である．認証ルーチンにおいては，認証情報の検証を行い，その結果を処理する．また コマンドディスパッチング機能の場合は，コマンドと引数に対してパースを行い，コマンドのディスパッチを行った後，それぞれに応じた特定の機能を実行する．認証ルーチンとコマンドディスパッチング機能のいずれの場合も「decider」と「handler」という２つの部分に分けることができる．例えば認証ルーチンであれば，認証情報の検証を行う部分が decider で，検証結果を処理する部分が handler である．コマンドディス

パッチング機能では，最終的にディスパッチするそれぞれの機能が handler である．この論文では WEASEL というアルゴリズムにより，バイナリの中から decider と handler を特定している．特定した decider と handler に対して，IDA Pro などのツールを用いて詳細に静的あるいは動的な解析を行い，バックドアが存在するかどうかを調べる．

最終的にバックドアであるかどうかについては，手動かつヒューリスティックに判断する．そして最後に，見つかった脆弱な機能や怪しいコマンドを監視，あるいは無効化するために自動でバイナリに instrumentation を行うようにしている．バイナリへの instrumentation とは，対象のバイナリにいくつかの命令を挿入し，挙動を監視したり，パフォーマンスを測定したりする技術のことである．

4.4 Shoshitaishvili らによる検知手法

Shoshitaishvili ら [5]は Firmalice というツールを実装し，ファームウェアのバイナリに対して，認証回避の問題がないかを自動的に解析している．この論文では，認証回避にかかわる問題として，意図的なハードコードされた認証情報や隠された認証インターフェース，意図的ではない，認証回避につながってしまう脆弱性を検知の対象としている．

手順としてはまずファームウェアをロードし，バイナリを中間表現に変換する．入力としてファームウェアとともにセキュリティポリシーを受け取るが，これはそれぞれのデバイスごとに，認証によりどのような特権操作を行うのかが異なるため，その情報を事前に知らせるためのものである．このポリシーに従って「特権プログラムポイント」を抽出する．例えば論文では図 5 のプログラムを考えている．

```
int auth(char *u, char *p) {
  if ((strcmp(u, "GO") == 0) && (strcmp(p, "ON") == 0))
    return SUCCESS;
  char *store_u = get_username();
  char *store_p = get_password();
  if ((strcmp(u, store_u) == 0) && (strcmp(p, store_p) == 0))
    return SUCCESS;
  else return FAIL;
}
int main() {
  if (auth(input("User:"), input("Password:")))
    system(input("Command:"));
}
```

図 5　認証機構回避の例（ [5]から引用，一部省略）

Figure 5　An example of authentication bypass.

このプログラムではセキュリティポリシーとして，認証されていないユーザには system 関数を実行してはならないとしており，system 関数を実行するコード部分が「特権プログラムポイント」となる．そしてバイナリから CFG を作成し，静的解析によりエントリポイントから先ほどの「特権プログラムポイント」への「認証スライス」を作成する．図 6 の CFG のうち白い部分がこのスライスを示している．

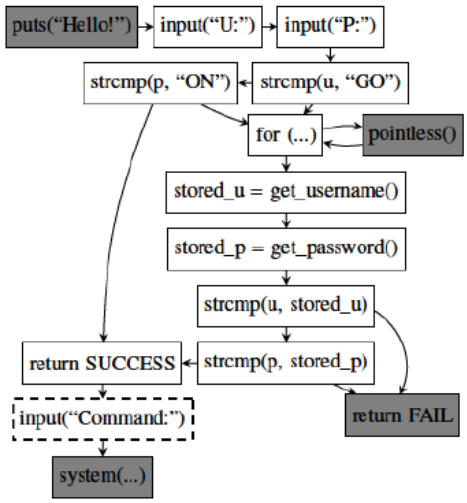

図 6　Firmalice による CFG（ [5]より引用）

Figure 6　CFG created by Firmalice.

次に Firmalice では，このスライスに対してシンボリック実行を行い，input("Command:") ブロックに至ることのできるユーザ入力を見つけ出す．シンボリック実行については，4.5 にて説明する．このプログラムの場合，正しいユーザの認証情報により認証されている場合と，バックドアにより認証を回避している場合のいずれかで特権状態になっている．Firmalice では「認証回避チェックモジュール」にこれらの特権状態に至るためのユーザ入力の制約条件を調べ，ユニークな値のみであった場合，それを認証回避として結論付ける．

4.5 Papp らによる検知手法

Papp ら [6]はコンクリート実行とシンボリック実行を併用することにより，バックドアやロジックボムといったトリガーに起因する動作を半自動的に検知する手法を提案している．バイナリを解析対象とするブラックボックステストではなく，ソースコードをもとに解析を行うホワイトボックステストである．トリガーに起因する動作を検知するためには次のようなアプローチが必要である．

- 入力された値がどのようにハンドルされるのかを理解する．これは入力された値がプログラムの挙動をどう変えるかを理解することでもある．
- その理解に基づいて自動的に入力値を生成する．
- コードのあらゆる箇所にたどり着くための，入力値などに関する条件を特定する．
- 怪しい条件を検知し，その実行パスが悪意あるものかどうかを判断する．

最初の 3 つの要件はシンボリック実行によって達成することができる．シンボリック実行とは，プログラムがあるパスを通る際の入力に関する制約（これをパス条件という）を計算する技術である．シンボリック実行では入力値とし

て具体的な値を与えずに，シンボルとして制約を特定していく．プログラムのブランチにおいては，シンボリック実行はブランチ条件を満たすようなシンボルと満たさないシンボルの２つのインスタンスに分割する．得られたパス条件を解くことで得られる具体的な値は，この後用いるテストケースに使用できる．しかしシンボリック実行においてシンボルだけで処理しようとするとパス爆発（ブランチごとにプログラムのパスが急増していく現象）などの問題が生じる．これらを解決する手段として，部分的にシンボルではなく具体値を与えるコンクリート実行と，シンボリック実行を併用したハイブリッドな手法(コンコリック実行)が用いられることが多い．既存のコンコリック実行ツールでは，ソフトウェアにおいてどの変数をシンボリックとして扱うかを特定する追加の操作が必要である．この論文では，はじめにソースコードに対して，自動的に解析用のコードを挿入する．これにより，隠された振る舞いをガードしている変数や関数呼び出しを特定する．ある種の関数呼び出しはトリガーの潜在的なエントリポイントとなっており，これらの関数をダミー関数に置き換えることにより，その関数呼び出しがその後の実行パスにどれだけ影響を与えるかを調べていく．最終的には，候補として出力した怪しいパス条件に対して，人間が手動で解析を行うことで，本当に悪意のある実行パスであるかどうかを判断する．

4.6 Thomas らによる検知手法

Thomas らは，ファームウェアのバイナリに対して，隠し機能あるいはユーザの期待しない機能を可能な限り自動的に見つけるための HumIDIFy [10]というツールと，ハードコードされた認証情報に対するチェックや隠された，あるいは仕様書に記載のないコマンドを特定するための Stringer [11]というツールを提唱し，実装を行っている．

まずは HumIDIFy について説明する．この論文では，対象のファームウェアイメージに対して，どのバイナリが本来期待されている機能から逸脱しているのかを可能な限り自動的に調べる．その後，実際に埋め込まれている機能が悪意のあるものであるかどうかの解析はマニュアルで行う必要がある．解析の流れとしては次の通りである．

- ファームウェアイメージを「アンパックエンジン」によりアンパックする．
- 得られた個々のバイナリに対して，クラス分類器により機能ごとのクラス分けを行う．
- 個々のバイナリに対して，「プロファイル評価器」により該当する機能のプロファイルを参照し，静的解析により不審な機能がないかなどの結果を出力する．

まずアンパックエンジンにより，ファームウェアイメージのアンパックを行う．次にクラス分類器により個々のバイナリに対してクラス分けを行う．このクラス分けにおいては該当する機能のカテゴリと，その分類がどれだけ正確であるかという信頼値を与える．分類についてはファイル名をもとに行うのが簡単であるが，ファイル名をきれいに抽出できない場合や，特定のベンダから提供されるサービスへのバイアスがかかり過学習の問題が起こる．過学習は学習モデルがテストデータに対して具体的になりすぎて，あらゆる入力に対しての汎用性を欠いている状態である．そこでこの論文では教師あり学習と半教師あり学習を併用している．この手法ではある程度の量のラベリングされたサンプルデータが必要であり，また先の過学習状態を避けるために十分な量のサンプルデータを使用して学習を行っている．バイナリと得られた機能カテゴリに対して，プロファイル評価器にかける．このフェーズではまず個々のバイナリに対して静的解析を行い，プロファイルのデータベースを参照することで，該当する機能カテゴリの通常の機能と照らし合わせて，不審な機能がバイナリに含まれないかを調べる．ここで不審な機能が含まれている場合には手動の解析に回す．

次に Stringer について説明する．Stringer では，バックドアのトリガー部分に注意を払い，解析を行うことを目的としている．1 章で紹介した通り，Thomas らによればバックドアは４つの構成要素に分解することができるが，ユーザの期待しない挙動はバックドアのペイロード部分であり，そこに遷移するために特別なキーワードとの比較を行うことが多く，これがバックドアのトリガー部分である．具体的な手法としては，まず静的なデータと比較を行っている関数をすべて特定する．一般的には strcmp や strncmp 関数が使用されるが，静的リンクやシンボルを削除されている場合にはこうした関数名で探すことはできない．そこで，渡されている引数のうち,最低一つが静的なデータであり，その関数の返り値（基本的にブール値）が次の制御フローに影響を与えるものを抽出することを行う．

似たようなケースとして，プロトコルパーサーが存在する．パーサーも同様に比較関数を用いて異なる静的なデータの引数と何度も比較を行う．そして隠しコマンドや仕様書に記載のないプロトコルメッセージはしばしばこうしたパースルーチンの中から見つかる．そこでこうしたパースルーチンの中から見つかる静的なデータの比較関数のコールサイトも解析の対象としている．これらの条件を満たす関数のコールサイトを調べ，比較されている静的なデータを抽出していく.抽出したデータを使用する関数に対して，関数内の各ベーシックブロックに至るために比較されているデータシーケンスのセットを作成する．これらのセットをベースにして関数ごとのスコアを算出することで，どの程度その関数の条件処理が静的なデータの比較に依存しているのかを調べる．最後に，このスコアをもとにして関数の解析の重要度を決定する．

4.7 Tien らによる検知手法

Tien ら [7]は Universal Firmware vulnerability Observer(UFO)というシステムを提案し，ファームウェアの脆弱性を発見

する．このシステムでは，まずファームウェアのバイナリから組込みファイルシステムを抽出する．RESTful API を提供しており，コマンドラインやウェブアプリケーションからサンプルのファームウェアバイナリを入力することができる．ファームウェアの抽出器としては基本的に Binwalk を使用している．得られたファイルシステムに対して，脆弱性を検査するが，これは 4 つのカテゴリからなる．はじめに，既知の脆弱性を集めたデータベースを参照することで，対象のファイルシステムの中に既知の脆弱性が存在しないかを検査する．同時にセキュリティセンシティブなファイルや情報を見つけるために特定のシステムプロファイルを探す，あるいは特定のキーワード検索をかける．そしてファイルシステムの中から怪しい実行パスのシェルスクリプトを見つけることで，バックドアが存在しないかをチェックする．怪しい実行パスとは主に，書き込み可能なディレクトリに配置されている実行パスであり，またシェルスクリプト内で telnetd や sshd など疑わしいコマンドを実行しているものを調べる．こうした脆弱性の分析を行い，最終的にはレポートを出力するシステムである．

5. 考察

5.1 検知手法の分類

はじめに，4 章で説明したバックドアに対する検知手法を，次の３つ観点で考察した．なお，Zhang らの手法は IDS によるネットワークセキュリティをベースとしており，また Wysopal らは検知手法の具体的な実装をしているわけではないので表には記載していない．

- 検知対象のバックドアの種類
- 検知システムに入力として必要な情報
- 自動化されているかどうか

表 2　検知手法と対象のバックドアの種類

Table 2　Detection methods and their target backdoors.

	認証回避	脆弱性	隠しコマンド・隠し機能
Schuster らの手法	✓		✓
Firmalice	✓		
Papp らの手法			✓
HumIDIFy			✓
Stringer	✓		✓
UFO		✓	✓

表 2 において，代表的なバックドアである認証回避において利用される，ハードコードされた認証情報や脆弱性といったものを検知する手法と，隠しコマンドや隠し機能を検知するものに大別できる．前者はバックドアのトリガー部分を対象としており，後者はペイロード部分を対象とする手法と分類することができる．バックドアの種類は多岐にわたるが，こうしたバックドアの構成要素ごとに効果的なアプローチをすることが重要であることを示唆している．ハードコードされた認証情報や既知の脆弱性といったトリガー部分に関しては，4 章で見てきたように静的なデータの比較など，特定の条件に絞ることで検知できる．ペイロードに関しては大きく二つの手法に分かれる．一つは仕様書などで公開されている機能と照らし合わせて，対象にそれ以外の隠しコマンドや隠し機能が存在しないかを直接調べる手法である．これはバックドアが仕込まれていない正規のバイナリとの比較や，HumIDIFy のように機能を事前に推定し，正しい挙動からの逸脱を検知するアプローチが存在する．もう一つは，隠し機能や隠しコマンドが，基本的にトリガーに起因することに着目したアプローチである．これは Papp らの手法のように，特定の入力などにより，はじめて隠し機能などが動作することに着目して，コンクリート実行やシンボリック実行を活用することで，こうした特定の条件を満たすケースを発見するものである．Thomas らの提唱したバックドアの構成要素への分解は極めて有効かつ重要な概念であるが，バックドアの中にはトリガー部分とペイロード部分を切り分けることが難しいものもあり，それぞれの構成要素ごとに検知を行うアプローチが有効でない場合もある．この問題に対して Papp らが提唱した手法のように，バックドアの特徴を踏まえて効果的にソフトウェア解析技術を用いるアプローチは有用であると考えられる．

表 3　検知手法と入力として必要な情報

Table 3　Detection methods and their input files.

	ソースコード	バイナリ
Schuster らの手法		✓
Firmalice		✓
Papp らの手法	✓	
HumIDIFy		✓
Stringer		✓
UFO		✓

次にそれぞれの検知手法が，検知システムの入力としてどのような情報を必要としているのかを考察する．解析を行う上で，ソースコードを入手することにより，より効果的に解析や検査を行うことができるのは確かである．しかしながら組込みデバイスのバックドアを検知する検査においては，現実的にソースコードが入手できる可能性は高くなく，表 3 を見ても，実際に多くの研究者がバイナリを入力として，解析ツールの実装を行っている．

最後に，検知手法がどの程度手続きを自動化しているのかを考察する．近年のバックドアに対する検知手法を提唱した論文の多くは，手動解析の手間を削減するために極力自動化する方法を模索している．共通する特徴としては，バックドアの構成要素ごとの特徴をなるべく自動的に見つ

け出し，場合によってはスコアリングなどで優先順位付けを行う．そして高い優先度のものから最終的に手動で解析や判断を行うというアプローチが一般的であるようだ．

6. 今後の展望

バックドアに対する既存の検知手法は，特定の種類のバックドアに対して有効なアプローチを取り，検知効率と精度を高めている．本論文で紹介した研究では，認証回避やトリガーとなる脆弱性，隠しコマンド・隠し機能をそれぞれ対象にしたものであった．メーカーが外部調達したソフトウェアに対してバックドアの検査をする際には，あらゆる種類のバックドアに対応する検査手法を確立する必要があり，既存の検知手法単一では特定のバックドアしか対象としないので，複数の手法を組み合わせることが必要である．

しかしながら，これらの手法を単純に組み合わせても検査効率，効果の両面で十分な成果を期待できない．一つは，各検査を別々に行うと，検査過程の重複が数多く存在し，無駄が発生する．はじめに対象のソフトウェアの構造解析を行うものや，特定の構造体や情報を抽出するものは，その過程を共通化できる．二つ目は，これらの手法を組み合わせた際に，あらゆるバックドアに対して検査したことを保証できない問題である．既存の検知手法では代表的な認証回避や隠しコマンド・隠し機能を検査するが，これらの手法が対象としないバックドアが含まれていても検知できない．また，手法によってはバックドアがソフトウェアのどの部分から検出されたのか不明瞭な場合もある．

効率的に，かつ効果的に検査を行うためには，トップダウンにソフトウェアを解析，検査することが有効である．構造解析や機能ごとの分割などのプロセスを共通化し，ソフトウェアを構造から網羅的に検査することで，バックドアが存在しやすい機能（認証機能やパーサー機能など）だけでなく，あらゆる制御フローや機能を検査し，隠れ潜むバックドアを検出，あるいは存在しないことを保証できる．

7. おわりに

近年では，組込みデバイスがインフラや企業システムにおいて重要な役割を担っている．結果としてシステムは外部から調達したデバイスやソフトウェアで構成されている．しかし，こうしたデバイスからバックドアが発見されるインシデントが数多く報告され，効果的なバックドアの検知手法を確立することが重要な課題となっている．本論文では，既存研究におけるバックドアの定義や分類を紹介し，Thomas らによるバックドアの構成要素への分解とそれぞれの要素を説明した．そしてバックドアを検知する手法を提唱している既存研究を紹介し，いくつかの観点から比較，整理した．今後，整理の結果に基づいて，効率的かつ効果的なバックドア検査技術を開発する予定である．

参考文献

[1] S. L. Thomas , A. Francillon, “Backdoors: Definition, deniability and detection,” 21st International Symposium on Research in Attacks, Intrusions and Defenses, 2018.

[2] Y. Zhang and V. Paxson, "Detecting Backdoors," SSYM'00 Proceedings of the 9th conference on USENIX Security Symposium, 2000.

[3] C. Wysopal , C. Eng, “Static Detection of Application Backdoors,” Black Hat USA, 2007.

[4] F. Schuster , T. Holz, “Towards reducing the attack surface of software backdoors,” Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013.

[5] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel , G. Vigna, “Firmalice－Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware,” Network and Distributed System Security Symposium, 2015.

[6] D. Papp, L. Buttyán , Z. Ma, “Towards Semi-automated Detection of Trigger-based Behavior for Software Security Assurance, ” Proceedings of the 12th International Conference on Availability, Reliability and Security, 2017.

[7] C. Tien, T. Tsai, I. Chen , S. Kuo, “UFO - Hidden Backdoor Discovery and Security Verification in IoT Device Firmware,” 2018 IEEE International Symposium on Software Reliability Engineering Workshops, 2018.

[8] H. Flake, “Structural Comparison of Executable Objects,” In Proceedings of the IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment, 2004.

[9] D. Gao, M. K. Reiter , D. Song, “BinHunt: Automatically Finding Semantic Differences in Binary Programs, ” International Conference on Information and Communications Security, 2008.

[10] S. L. Thomas, F. Garcia , T. Chothia, “HumIDIFy: A Tool for Hidden Functionality Detection in Firmware, ” International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2017.

[11] S. L. Thomas, T. Chothia , F. D. Garcia, “Stringer: Measuring the Importance of Static Data Comparisons to Detect Backdoors and Undocumented Functionality, ” European Symposium on Research in Computer Security, 2017.