博士論文

IoT機器に対するマルウェア持続感染の分析に関する研究

A Study on Analysis of Persistent Malware Infection Targeting IoT Devices

国立大学法人 横浜国立大学大学院
環境情報学府

原 悟史
Satoshi HARA

2020年9月

概要

近年は，IoT(Internet of Things) と称されるモノのインターネットが注目を浴びている．これまでスタンドアロンで用いられていた組込み機器すなわち IoT 機器がインターネットに接続され，データやサービスを連携することにより高い付加価値を生み出している．一方で，IoT 機器のセキュリティリスクも顕在化している．脆弱な IoT 機器をインターネットに接続することで，機器自体が攻撃を受けたり，機器を踏み台として外部もしくは IoT の内部システムへの攻撃に悪用される事例が増加している．IoT 機器は安価で大量に製造・販売されており，これらの機器が攻撃対象となることで，従来と比べて大規模なサイバー攻撃を引き起こしている．たとえばマルウェア Mirai は多数の IoT 機器に感染し，当時史上最大規模の DDoS 攻撃を行うなど，大きな社会問題となっている．

従来，組込み機器のソフトウェアは，OS や通信のような低レイヤに至るまで機器独自に設計・開発されていたため，サイバー攻撃の対象となりにくかった．近年は Linux のような汎用 OS の搭載や，オープンソースソフトウェアの利用，汎用的な通信プロトコルを使用する機器が増加しており，このような共通性を狙って多くの機器に感染して動作するマルウェアが登場した．このように IoT 機器は，PC と共通するソフトウェア部品を使用して開発されるようになってきたものの，その目的や性質の違いから，機器間の互換性には制約がある．PC はユーザーが機能を追加して使用する前提のため，ハードウェアリソースを潤沢に搭載しており，OS やコマンド群は汎用性を持たせた互換性の高い構成となっている．一方，IoT 機器は限られた機能に特化して安価に製造されており，必要最低限のハードウェアリソースのみを搭載している．リソースの制約により，IoT 機器のソフトウェアは OS コマンドの削減や軽量化のような省リソース化が行われるほか，機器独自の方法による機能拡張が行われることがあり，互換性を下げる要因となる．このような機器独自の特徴を悪用した，特定の機器に特化された攻撃が登場しており，効率的に解析する技術が必要とされている．

本研究では，特定の IoT 機器に特化された攻撃のうち，マルウェア持続感染に着目した分析を行う．Mirai に代表される IoT マルウェアの多くは，感染した機器の電源を再起動することで消滅することが知られている．しかしながら，特定の機器を狙った，電源の再起動のみでは消えない“持続感染”を引き起こす IoT マルウェアの事例も報告されている．IoT 機器はユーザーによる機能追加が不要であることに加え，いきなり電源を切る使い方が多いことから，システムを保護するために読み取り専用ファイルシステムが用いられることが多い．ユーザー設定を電源再起動後も保持し続けるための不揮発ストレージがあるものの，実現手法は機器により異なる．持続感染はこのような機器独自の仕組みを悪用することにより実現されている．持続感染をしたマルウェアを駆除するためには，そのマルウェアが持続感染する仕組みを正しく解析する必要がある．

まず，IoT 機器の特性や機能に着目し，マルウェア持続感染の成立条件を分析する．持続感染を実現可能な攻撃手法として，“システム設定改変攻撃”と“ファームウェア不正書

換攻撃”の 2 つを定義する．IoT 機器の実機とマルウェア検体を用いて持続感染の有無および持続感染の成立条件をどの程度満たしているか調査を行った結果，いずれの機器・マルウェア検体の組み合わせにおいても，持続感染は確認されなかった．しかしながら，概念実証の結果，“システム設定改変攻撃”と“ファームウェア不正書換攻撃”により持続感染を実現可能な機器が存在することを示した．これらの概念実証をもとに，持続感染防止に効果的なファームウェア構成について議論を行う．

続いて，IoT 機器の実機を用いたマルウェア動的解析手法を提案する．機器の構成に依存した振る舞いを行うマルウェアや，仮想環境検知を備えるマルウェアを動的解析する場合，IoT 機器の実機を用いる必要がある．IoT 機器は後から機能をインストールすることができない場合が多く，機器が元々有する機能のみを用いて解析を行う方法を検討した．さらに，持続感染型マルウェアの解析を行った後の，解析環境の復元方法の検討を行った．IoT 機器の実機とマルウェア検体を用いた提案手法の評価を行った結果，ファイル・ディレクトリ操作，プロセスの起動のような機器内部に対する挙動を解析することにより，特定の機器のみで生じる持続感染の仕組みを解析できることを示した．加えて，マルウェアによる自身のバイナリ削除やプロセス名の偽装による感染を隠蔽する行為や，他機器へのスキャンや感染拡大，C&C サーバーとの通信のような挙動を解析できることを示した．

最後に，多くの IoT 機器のディレクトリ構成の和集合を持つ複合ディレクトリ型サンドボックスを用いた，特定の IoT 機器のみに通用する攻撃の動的解析を行う手法を提案する．メーカーのホームページに公開されている IoT 機器向けファームウェア更新用ファイルを大量に収集し，抽出したファイルシステムの和集合となる構成を持つ複合ディレクトリ型サンドボックスを仮想環境上に構築した．実在するマルウェアを用いた評価を行い，特定の機器のみに持続感染するマルウェアを解析できることを示した．本解析手法により，持続感染のメカニズムの究明に加えて，持続感染し得る機器の一覧をリストアップすることが可能であり，解析の大幅な効率化が見込まれる．

Abstract

In recent years, the Internet of Things called IoT has been rapidly developing. The embedded devices were often used as stand-alone devices, but more and more devices are connected to the Internet, and they are called IoT devices. These devices provide high added value by linking data and services through the Internet. On the other hand, the security risks of IoT devices are also becoming obvious. By connecting vulnerable IoT devices to the Internet, there are an increasing number of cases in which the IoT devices themselves are attacked or are abused to attack external or IoT internal systems via the devices. IoT devices are mass-produced and sold at low cost, and the attacks on these devices cause a larger cyberattack than before. For example, the malware Mirai infects many IoT devices and causes the largest DDoS attack in history at the time, which is a big social problem.

In past, software for embedded devices has been developed specially not only for applications but also for lower layers such as operating system and communication libraries, therefore these devices have been difficult to be targeted by cyberattacks. In recent years, general-purpose operating systems such as Linux, open source software, and general-purpose communication protocols are increasingly used in IoT devices, and malware that attacks such commonality has infected many devices. Although IoT devices have come to be developed using software components that are common with PCs, there are restrictions on the compatibility between devices due to the differences in their purposes and characteristics. The PC is equipped with sufficient hardware resources because the user adds functions as a premise, and the operating systems, operating systems commands, and software libraries are general-purpose and highly compatible. Opposingly, IoT devices are equipped with the minimum necessary hardware resources, since they are manufactured at low cost with limited functions. Due to the limited resources, the software of IoT devices is reduced in weight, for example, by reducing the number of operating systems commands and replacing them with alternative lightweight commands. Further, instead of installing general-purpose application, the function is expanded by using lightweight application originally developed. These are factors that reduce the compatibility of IoT devices. Attacks targeting specific devices that exploit such device-specific functions have appeared, and techniques for efficient analysis are needed.

This dissertation first focus on malware persistent infection among the attacks targeted at specific IoT devices. It is known that most of IoT malware such as Mirai disappears by restarting the infected IoT devices. However, there have been reports of IoT malware that targets a specific device and causes “persistent infection” that cannot be disappeared by restarting the device. Read-only file systems are often used to protect device storage

because IoT devices do not require software installation by users and are often used to turn off the power suddenly. Although IoT devices have a non-volatile memory for holding user settings, the implementation method differs depending on the device. Persistent infection is realized by abusing such a device-specific mechanism. In order to get rid of persistently infected malware, it is necessary to correctly analyze how the malware achieved persistent infection.

Chapter 4 focus on the characteristics and functions of IoT devices and analyze the conditions for persistent infection of malware on IoT devices. Two attack methods that can achieve persistent infection are defined: "system setting modification attack" and "unauthorized firmware overwriting attack". As a result of an experiment using a real IoT devices and malware samples to determine whether persistent infection occurred or how much the condition for causing the persistent infection is satisfied, persistent infection was not confirmed in any combination of device and malware sample. However, as a result of the proof of concept, it was shown that there are devices that can realize persistent infection by "system setting modification attack" or "unauthorized firmware overwriting attack". Based on these proofs of concept, the design of firmware to prevent persistent infection was considered.

Chapter 5 proposes a malware dynamic analysis method using real IoT devices. For dynamic analysis of malware that changes the behavior depending on a specific device, or that detects the virtual environment and changes its behavior, it is necessary to use the real IoT devices. Since many IoT devices do not allow users to install software, a method to analyze using only the functions originally provided in the devices was considered. Moreover, a method of restoring the analysis environment after analysis of persistent infection malware was considered. As a result of evaluating the proposed method using real IoT devices and malware samples, it is possible to analyze the mechanism of persistent infection that occurs only in a specific device by analyzing the behavior inside the device such as file / directory operation and process startup. In addition, it was shown that behaviors such as the act of concealing infection by malware's own binary deletion or process name disguise, scanning to other devices, spreading infection, and communication with C&C server can be analyzed.

Section 6 proposes a dynamic analysis method using a "composite directory sandbox" that has a union of the directory structure of many IoT devices to analyze malware that changes the behavior in a specific IoT device. A lot of firmware update files for IoT devices released on the manufacturer's homepage were collected, and the file system that combines the directory structure extracted from those firmware update files was constructed as a "composite directory sandbox" in the virtual environment. As a result of evaluation of the proposed method using existing malware, it was shown that malware that persistently infects only a specific device can be analyzed. In addition to investigat-

ing the mechanism of persistent infection, the proposed method can list the devices that are potentially persistently infected. The composite directory sandbox realizes a great efficiency in analysis of persistent infections.

目 次

図 目 次

表 目 次

第 1 章
序論

1.1 背景と目的

近年は，IoT(Internet of Things) と称されるモノのインターネットが注目を浴びている．これまでスタンドアロンで用いられていた組込み機器すなわち IoT 機器がインターネットに接続され，データやサービスを連携することにより高い付加価値を生み出している．一方で，IoT 機器のセキュリティリスクも顕在化している．脆弱な IoT 機器をインターネットに接続することで，機器自体が攻撃を受けたり，機器を踏み台として外部もしくは IoT の内部システムへの攻撃に悪用される事例が増加している．IoT 機器は安価で大量に製造・販売されており，これらの機器が攻撃対象となることで，従来と比べて大規模なサイバー攻撃を引き起こしている．たとえばマルウェア Mirai は多数の IoT 機器に感染し，当時史上最大規模の DDoS 攻撃を行うなど，大きな社会問題となっている．

従来，組込み機器のソフトウェアは，OS や通信のような低レイヤに至るまで機器独自に設計・開発されていたため，サイバー攻撃の対象となりにくかった．近年は Linux のような汎用 OS の搭載や，オープンソースソフトウェアの利用，汎用的な通信プロトコルを使用する機器が増加しており，このような共通性を狙って多くの機器に感染して動作するマルウェアが登場した．このように IoT 機器は，PC と共通するソフトウェア部品を使用して開発されるようになってきたものの，その目的や性質の違いから，機器間の互換性には制約がある．PC はユーザーが機能を追加して使用する前提のため，ハードウェアリソースを潤沢に搭載しており，OS やコマンド群は汎用性を持たせた互換性の高い構成となっている．一方，IoT 機器は限られた機能に特化して安価に製造されており，必要最低限のハードウェアリソースのみを搭載している．リソースの制約により，IoT 機器のソフトウェアは OS コマンドの削減や軽量化のような省リソース化が行われるほか，機器独自の方法による機能拡張が行われることがあり，互換性を下げる要因となる．このような機器独自の特徴を悪用した，特定の機器に特化された攻撃が登場しており，効率的に解析する技術が必要とされている．

本研究では，特定の IoT 機器に特化された攻撃のうち，マルウェア持続感染に着目した分析を行う．Mirai に代表される IoT マルウェアの多くは，感染した機器の電源を再起動することで消滅することが知られている．しかしながら，特定の機器を狙った，電源の再起動のみでは消えない“持続感染”を引き起こす IoT マルウェアの事例も報告されている．IoT 機器はユーザーによる機能追加が不要であることに加え，いきなり電源を切る使い方が多いことから，システムを保護するために読み取り専用ファイルシステムが用いられることが多い．ユーザー設定を電源再起動後も保持し続けるための不揮発ストレージがあるものの，実現手法は機器により異なる．持続感染はこのような機器独自の仕組みを悪用す

ることにより実現されている．持続感染をしたマルウェアを駆除するためには，そのマルウェアが持続感染する仕組みを正しく解析する必要がある．

まず，IoT 機器の特性や機能に着目し，マルウェア持続感染の成立条件を分析する．持続感染を実現可能な攻撃手法として，“システム設定改変攻撃”と“ファームウェア不正書換攻撃”の 2 つを定義する．IoT 機器の実機とマルウェア検体を用いて持続感染の有無および持続感染の成立条件をどの程度満たしているか調査を行った結果，いずれの機器・マルウェア検体の組み合わせにおいても，持続感染は確認されなかった．しかしながら，概念実証の結果，“システム設定改変攻撃”と“ファームウェア不正書換攻撃”により持続感染を実現可能な機器が存在することを示した．これらの概念実証をもとに，持続感染防止に効果的なファームウェア構成について議論を行う．

続いて，IoT 機器の実機を用いたマルウェア動的解析手法を提案する．機器の構成に依存した振る舞いを行うマルウェアや，仮想環境検知を備えるマルウェアを動的解析する場合，IoT 機器の実機を用いる必要がある．IoT 機器は後から機能をインストールすることができない場合が多く，機器が元々有する機能のみを用いて解析を行う方法を検討した．さらに，持続感染型マルウェアの解析を行った後の，解析環境の復元方法の検討を行った．IoT 機器の実機とマルウェア検体を用いた提案手法の評価を行った結果，ファイル・ディレクトリ操作，プロセスの起動のような機器内部に対する挙動を解析することにより，特定の機器のみで生じる持続感染の仕組みを解析できることを示した．加えて，マルウェアによる自身のバイナリ削除やプロセス名の偽装による感染を隠蔽する行為や，他機器へのスキャンや感染拡大，C&C サーバーとの通信のような挙動を解析できることを示した．

最後に，多くの IoT 機器のディレクトリ構成の和集合を持つ複合ディレクトリ型サンドボックスを用いた，特定の IoT 機器のみに通用する攻撃の動的解析を行う手法を提案する．メーカーのホームページに公開されている IoT 機器向けファームウェア更新用ファイルを大量に収集し，抽出したファイルシステムの和集合となる構成を持つ複合ディレクトリ型サンドボックスを仮想環境上に構築した．実在するマルウェアを用いた評価を行い，特定の機器のみに持続感染するマルウェアを解析できることを示した．本解析手法により，持続感染のメカニズムの究明に加えて，持続感染し得る機器の一覧をリストアップすることが可能であり，解析の大幅な効率化が見込まれる．

1.2 本論文の構成

本論文の構成を図 1.1 に示す．まず，2 章では，本研究に関連する先行研究について紹介する．3 章では，IoT 機器に対するマルウェア持続感染の概論を述べる．4 章では，IoT 機器へのマルウェアの持続感染の成立条件を分析し，IoT 機器の実機および実際のマルウェア検体を用いて概念実証を行った結果を示す．この研究は電子情報通信学会論文誌 (Volume J102-B No.8 2019) に発表されている．5 章では，IoT マルウェアの動的解析環境として，実機を用いた動的解析手法の要件を分析したうえで，IoT 機器の実機およびマルウェア検体を用いて検証を行った結果を示す．この研究は電子情報通信学会論文誌 (Volume J103-B

No.8 2020) に発表されている．6 章では，複合ディレクトリ型サンドボックスを用いた動的解析手法の検証を行った結果を示す．7 章では，本稿の内容を総括し，今後の課題を述べる．

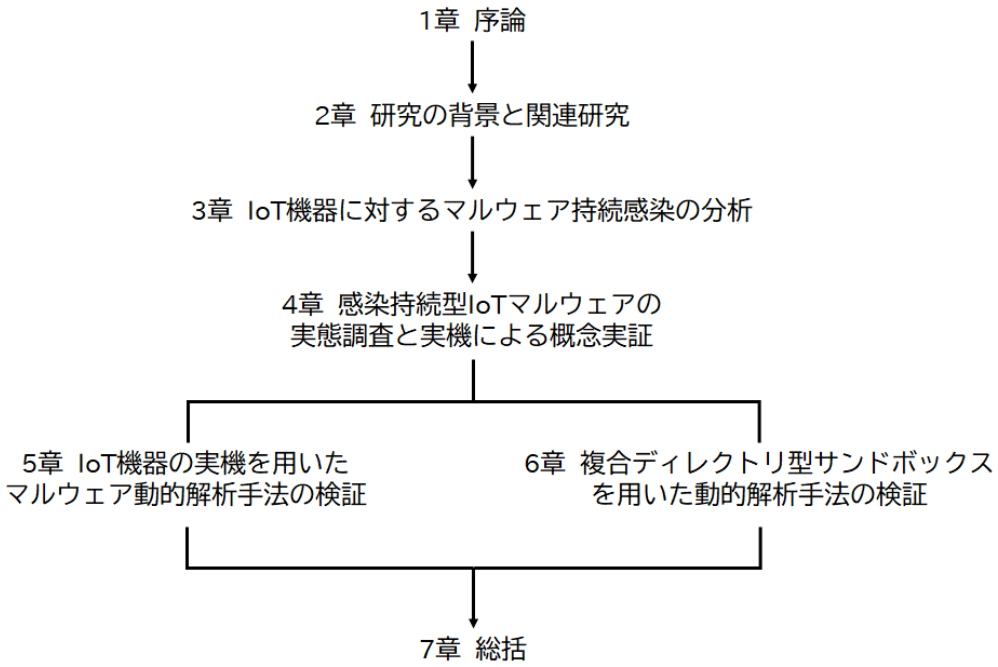

図 1.1 論文の構成

第 2 章
本研究に関する先行研究

本章では，本研究と関連する先行研究および技術について述べる．

2.1 IoT 機器

近年，インターネットにつながる組込み機器すなわち IoT 機器は増加し続けている．特にこれまでインターネットに接続されていなかった輸送機器，医療機器，産業用機器分野のインターネット接続の増加が見込まれ，2018 年の 307 億台から 2021 年の 448 億台への増加が予測されている [1]．IoT 機器の増加に伴い，これらの機器を狙うサイバー攻撃も活発化している．インターネット上に流れる攻撃パケットは 2015 年から 2018 年の 3 年間で 3.7 倍に急増しており，約半数は IoT 機器を狙ったものと推測されている [2]．

従来，組込み機器のソフトウェアは，OS や通信のような低レイヤに至るまで機器独自に設計・開発されていたため，サイバー攻撃の対象となりにくかった．近年は汎用 OS の搭載や，オープンソースソフトウェアの利用，汎用的な通信プロトコルを使用する機器が増加しており，サイバー攻撃の主な対象となっている．文献 [3] では 26,275 もの IoT 機器向けファームウェア更新用ファイルを収集し分析することにより，IoT 機器の特性を示した．この分析によると，IoT 機器には ARM [4] や MIPS [5] といった CPU アーキテクチャが多く用いられており，搭載される OS の多くは Linux [6] である．

一般に IoT 機器は PC と比べて長期間使われることが多い一方で，販売時に製品ライフサイクルが示されていないことが多い問題がある．加えて，搭載するハードウェアリソースの制約や OS のサポート終了などによりセキュリティアップデートの提供が困難になる可能性がある．たとえば，日本国内の家庭用 Wi-Fi ルーターメーカーの業界団体では，サポートの保証期間終了後も可能な限りの脆弱性対応の意向を示しているものの，具体的な期間は明言されておらず，セキュリティアップデートを行えなくなる可能性について言及されている [7]．このように，IoT 機器の長期利用はセキュリティリスクを抱えている．

2.2 IoT マルウェア

IoT 機器をターゲットとしたマルウェアが多く登場している．多くの場合，マルウェアに感染した IoT 機器はボットネットを構築し，DDoS 攻撃に悪用される．最も初期の IoT マルウェアはオープンソースのボットネットフレームワークの Hydra であると言われており，2008 年に登場した．以降，Psyb0t や Chuck Norris，Tsunami，Bashlite といった IoT 機器をターゲットとしたマルウェアが続々と登場した [8][9][10]．特に有名な IoT マルウェアは 2016 年に登場した Mirai であり，Telnet の脆弱な認証を悪用して最大で約 60 万台の IoT

機器に感染し，600Gbps を超える当時史上最大規模の DDoS 攻撃を引き起こした．Mirai のソースコードはインターネット上に公開されており，現在に至るまで多数の亜種が出現している [11]．これらの IoT マルウェアは RAM 上にのみ存在するため，機器の電源を切ることで消滅した．しかし，近年では Hajime [12][13]，VPNFilter [14][15] といった，機器の電源を切っても消滅をしない，持続感染するマルウェアが登場している．

PC をターゲットとしたマルウェアは，以前から持続感染するものが知られており，実現手法が分析されている [16][17][18]．OS のスタートアップ機能の悪用や，タスクスケジューラーを改変することにより，機器の電源再起動後にマルウェアのバイナリを自動起動することで持続感染を実現する．しかし，IoT 機器には squashfs [19] や cramfs [20] のような読み取り専用型のファイルシステムが多く使われており，PC と同様の手法で持続感染を試みても機器の再起動により復元され，持続感染に至らないことが多い．このように，IoT 機器へのマルウェアの持続感染は，マルウェアの実装のみならず機器の設計思想や実装との関係が深いが，体系立てた分析は行われていない．そこで，4 章で IoT 機器における持続感染の成立条件を分析のうえ概念実証を行う．

2.3 IoT マルウェアの動的解析

マルウェアの解析手法は，主に静的解析と動的解析の 2 つに分類される．静的解析は，マルウェアのバイナリファイルをアセンブリコードにリバースエンジニアリングをすることにより，マルウェアを実行することなく解析する手法である．実行可能なすべての処理を解析可能な利点がある一方で，詳細な解析を行うためには熟練した技術が必要であり解析に時間を要するため，特定の機器に依存するような複雑な処理を解析することは困難である．そのため，IoT マルウェアの解析に動的解析が多く用いられている．

動的解析を行うためのフレームワークやサンドボックスがいくつか提案されている．Cuckoo Sandbox [21] はオープンソースの自動マルウェア解析システムである．API コールのトレース，ネットワークトラフィックのダンプのようなマルウェアの振る舞いを記録する機能が提供されているものの，マルウェアを実行するためのサンドボックスはユーザー自身で用意しなければならない．サンドボックスに用いるエミュレーターとして，オープンソースエミュレーターの QEMU [22] が多く用いられている．QEMU は組込み機器で広く使用されている ARM や MIPS を含め，多くの種類の CPU をエミュレート可能である．

オープンソースのサンドボックスとして，Detux [23] が提供されている．QEMU を用いて MIPS，MIPSEL，ARM を含む 5 つの CPU アーキテクチャがサポートされている．バイナリファイルに含まれる文字列解析や ELF ファイル解析のような静的解析機能，および動的解析としてネットワークトラフィック解析が提供される．しかし，ファイルやディレクトリ操作のような機器内部に対する振る舞いを解析する機能は提供されていない．文献 [24] では，Cuckoo Sandbox 上で QEMU と buildroot [25] を用いて IoT 機器を模したマルウェア動的解析環境を構築する手法が提案されている．しかし，提案手法を用いてどのような解析が行われたかは示されておらず，結果の分析や考察が行われていない．本手法で

サンドボックスに使用するルートファイルシステムは，buildroot を用いて生成した汎用的なものであるため，例えば特定の機器のディレクトリ構成に依存した動作のような，特定の機器に依存する振る舞いを解析することができない問題が考えられる．“IoTBOX”[26] は，組込み機器向けディストリビューションの OpenWRT [27] を用いたサンドボックスである．エミュレーターとして QEMU を用いており，MIPS，ARM，MIPSEL を含む 8 つの CPU アーキテクチャがサポートされている．“IoTBOX”は C&C サーバーからのコマンドや DoS 攻撃の通信などのネットワーク解析を目的としており，機器内部に対する振る舞いの解析には触れられていない．これらのサンドボックスは IoT 機器の CPU をエミュレートするものの，ルートファイルシステムは汎用的なものであり，機器の内部構成を再現できていない問題がある．そのため，マルウェアの持続感染のような，マルウェアの振る舞いと機器の内部構成に密接に関係して実現される攻撃は，これらのサンドボックス上で正しく動作しない可能性がある．

一方で，より実機に近いサンドボックスが提案されている．サンドボックス“Avatar[28] は，エミュレーターの実行を実機と連動させることにより，ハードウェアへのアクセスも忠実にエミュレートしようと試みた．しかし，DMA によるハードウェア的なメモリ書き換えをエミュレートできない問題や，実機がハードウェアへアクセスする速度と比較しエミュレーターとハードウェア間の同期処理が非常に遅くシステムがクラッシュする問題が指摘されている．“Firmadyne”[29] では，機器のファームウェアを QEMU 上で実行する手法が用いられている．機器の持つ機能を再現しており，機器の web インターフェースが持つ脆弱性を検出可能であることを示した．しかし，エミュレート対象は IoT 機器のルートファイルシステムのみであり，カーネルをあらかじめ用意した別のものに差し替えて解析を行うため，カーネルに依存する処理をエミュレートできない問題がある．そのため，収集した 23,035 のファームウェアのうち正しくエミュレートできたものは 10%以下であった．文献 [30] では，サンドボックス“F-Sandbox”を提案した．“F-Sandbox”は“Firmadyne”をベースにシステムコールの観測機能を強化し，マルウェアが使用するシステムコールの統計をもとにマルウェアのファミリの分類を試みた．しかし，“Firmadyne”のエミュレートの問題を解消していない．このようにハードウェア周りの厳密なエミュレートが難しく，機器の内部構成の厳密な再現は困難である．

これらのサンドボックスはいずれも仮想環境上で動作するものであるが，IoT マルウェアの中には解析環境を検知して振る舞いを変えることにより，解析を妨害するものがあることが知られている．デジタルビデオレコーダーを標的としたマルウェア Amnesia は仮想環境を検知して回避する最初の Linux マルウェアであると言われており，代表的な仮想環境である VirtualBox [31]，VMware [32]，QEMU を用いたサンドボックス上での実行を検知すると，仮想環境のファイルシステムを破壊する性質を持つ [33]．文献 [18] では，Linux をターゲットとしたマルウェアを多数解析し，マルウェアの持つ様々な特性を明らかにし，一部の Linux マルウェアが，仮想環境を検知し振る舞いを変える性質を持つことを指摘している．文献 [34] では，組込み機器向け評価基板である Raspberry Pi [35] の環境に対するサンドボックス検知の議論を行った．実行環境の属性情報と性能指標を用いたサンドボッ

クス検知方法であり，Raspberry Pi 以外の組込み機器に対しても応用可能であると考えられる．このように，仮想環境検知により振る舞いを変えるマルウェアが，IoT 機器向けにも実在し，多くの機器で実現し得ることが示されている．これまでに紹介した動的解析の手法は，すべて仮想環境上で動作するものであり，仮想環境を検知し振る舞いを変えるマルウェアを正しく解析することはできない．

マルウェア持続感染を解析するためのサンドボックスとして，これらの問題を解決する 2 つのアプローチを提案する．5 章では，IoT 機器の実機を用いた解析手法の提案と評価結果を示す．6 章では，仮想環境上にさまざまな IoT 機器のディレクトリ構成の和集合を持つ “複合ディレクトリ型サンドボックス” を用いた解析手法の提案と評価結果を示す．

第 3 章
IoT機器に対するマルウェア持続感染の分析

本論文は，IoT 機器に対するマルウェア持続感染の分析に関する研究である．従来の IoT マルウェアは機器の電源を再起動することで消滅したものの，近年は機器の電源を切っても消えない持続感染をするマルウェアが登場している．持続感染型マルウェアは，長期にわたり活動を行えることに加えて，ゆっくりと感染を広げるステルス性の高い攻撃を実現可能であることから，非常に脅威である．IoT マルウェアの持続感染は，マルウェア自体の振る舞いだけでなく，IoT 機器の設計思想や仕組みにも依存するが，これらを体系立てた分析は行われていない．IoT 機器の機能や内部構成は機器ごとの差異が大きく，持続感染の実現手法は機器ごとに異なる場合がある．そのため，機器独自の特徴を悪用した新たな手法により持続感染を実現するマルウェアが登場し得る．マルウェア持続感染を解析して実現手法を明らかにすることにより，持続感染型マルウェアの駆除手法の導出，持続感染しない堅牢な設計の導出，潜在的に持続感染し得る機器のリストアップのような対策が求められる．従来の汎用的な仮想環境を用いた解析手法では，マルウェア持続感染を再現して解析することが困難である．本研究では，持続感染型マルウェアの解析手法を提案し，これらの対策に貢献することを目的とする．

本研究のアプローチを図 3.1 に示す．まず，持続感染の成立条件を分析し，持続感染の実現手法に関する概念をモデル化し，実証を行う．3.1 節で，本アプローチの詳細を述べる．続いて，解析手法の議論を行う．解析が想定される場面として，1) マルウェアと感染する機器の組み合わせが分かっている場合，2) ある機器に対して持続感染が発生し得るか調査を行う場合，3) あるマルウェア検体がどの機器に感染するか分からない場合，の 3 つが挙げられる．検証対象の IoT 機器が明確であり入手可能な場合は，その機器を用いて解析することが考えられる．しかし，機器が元々備えている機能のみを用いて解析をする必要があるなど，制約が考えられる．3.2 節で，本アプローチの詳細を述べる．マルウェアがどの機器に感染するか分からない場合，市販の IoT 機器を大量に収集したりエミュレートして解析することは非現実的である．そこで，多くの IoT 機器の特徴を模擬したサンドボックスである，“複合ディレクトリ型サンドボックス”を用いた解析手法を提案する．これは，検証対象の IoT 機器を入手できない，もしくは解析に使用できない場合の代替手段としての活用も可能である．3.3 節で，本アプローチの詳細を述べる．

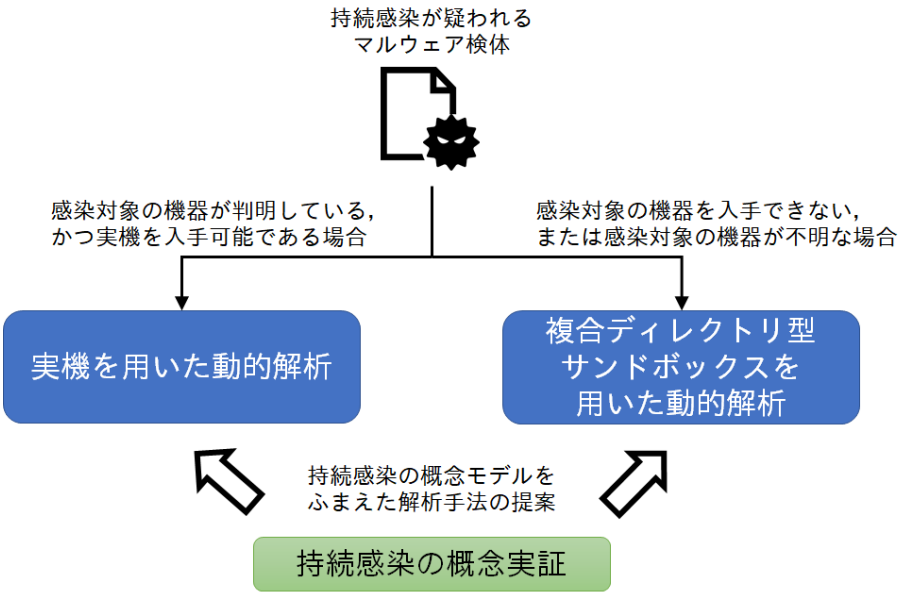

図 3.1 本研究のアプローチ

3.1 IoT 機器に対するマルウェアの持続感染の概念実証

PC をターゲットにしたマルウェアの中には，電源を再起動しても消えない持続感染をするマルウェアの存在が以前より指摘されている．PC に対する持続感染の主な実現手法は，元々 OS が備える機能，たとえば起動スクリプトやタスクスケジューラー等の仕組みを悪用して，機器の電源起動時もしくは定期的にマルウェアを実行する方法である．PC のファイルシステムはストレージに直接反映されるため通常は不揮発であり，攻撃者が特別な操作をすることなく，マルウェアのバイナリや OS の設定変更は電源を再起動した後にも変更が保持される (図 3.2)．

一方で，IoT 機器はストレージに関するハードウェア構成や設計思想が PC と異なるため (図 3.3)，持続感染の実現手法も PC とは別物と考えなければならない．そこで，機器の特性をふまえた 2 つの攻撃シナリオを検討し，実証を行った．最初のシナリオは，機器の OS のデフォルトもしくは機器独自に用意した起動スクリプトやタスクスケジューラーの設定を改変する攻撃シナリオ“システム設定改変攻撃”である．機器のファイルシステムが読み取り専用化されていない，もしくは起動スクリプトやタスクスケジューラーの設定が読み取り専用領域外にある場合に，本攻撃が通用すると考えられる．もう一つのシナリオは，機器の不揮発メモリ上に格納されたファームウェアを，マルウェア入りのファームウェアに書き換える攻撃シナリオ“ファームウェア不正書換攻撃”である．市販の IoT 機器を用いて実証実験を行い，これらの攻撃が実際に通用しうることを示す．

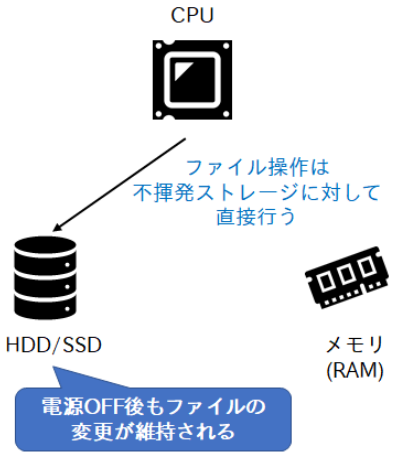

図 3.2 PC のデータ保存の仕組み

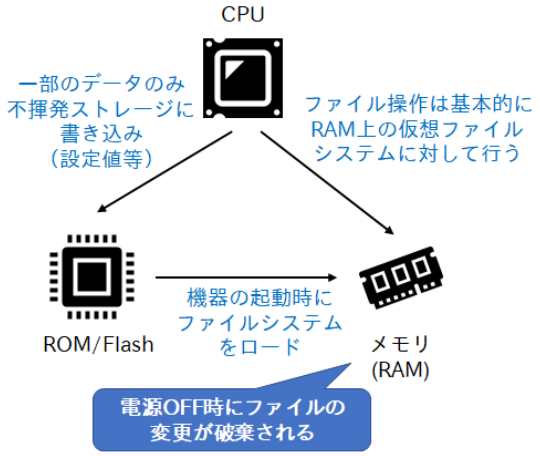

図 3.3 一般的な IoT 機器のデータ保存の仕組み

3.2 IoT 機器の実機を用いた動的解析

マルウェアの持続感染の振る舞いを再現するためには，ターゲットとなる機器を可能な限り忠実に再現した環境が必要である．マルウェアと持続感染する機器の組み合わせが分かっている場合，機器が入手可能であれば，IoT 機器の実機そのものを用いて解析する方法が考えられる．しかし，IoT 機器は特定の用途向けに作られており，ユーザーが後から機能を追加することが困難である．PC と比べて外部インターフェースが限られており，さらにはユーザーが OS の機能を直接操作する手段が提供されていないことが多い．以下に，IoT 機器の実機をサンドボックスとして使用する際の課題と，解決のためのアプローチを記載する．

管理者権限による OS コマンド実行

マルウェアのダウンロード・実行や解析に使用するコマンドの実行の目的で，解析者が機器の OS コマンドを管理者権限で実行する必要がある．通常，IoT 機器は OS コマンドを直接ユーザーが実行できる機能を提供していない．

マルウェアが感染する際，通常は機器のリモートコマンド実行の脆弱性を悪用される．マルウェアの感染方法が分かっていれば，同様の方法で外部から OS コマンドを実行可能であると考えられる．また，IoT 機器にはデバッグ目的で Telnet や UART のようなの通信や端子が残されていて，OS コマンドを実行可能な場合がある．本論文では，Telnet および UART を用いて実証実験を行った．

マルウェアの振る舞いの観測手法

IoT 機器はリソースの制約やファイルシステムが読み取り専用化されているなど，後から解析に必要なプログラムをインストールすることが困難である．通信のような外部に対する振る舞いは，機器の外側で通信をキャプチャすることにより観測可能である．一方で，機器内部に対する振る舞いの観測には，機器にあらかじめ備えられて

いるプログラムや OS コマンドを利用する必要がある．そこで，OS コマンドのうち多くの機器に備えられているものを使用して，マルウェア実行前後におけるディレクトリ・ファイルリストの差分や，プロセスリストの差分をもとにマルウェアの振る舞いを分析する方法を検討した．本手法を用いて実際の IoT 機器とマルウェア検体を用いた実証実験を行い，マルウェアによる自身のバイナリ削除やプロセス名の偽装による感染を隠蔽する行為や，マルウェアによる機器のファイル改変の行為，他機器のスキャンや C&C サーバーとの通信の挙動を解析可能であることを示した．特に，マルウェアによる機器のファイル改変の行為を詳しく解析することにより，持続感染の実現手法の解析が可能であった．

解析環境の復元

IoT 機器の実機をサンドボックスとして用いる場合，マルウェア感染後に元の状態に復元可能である必要がある．持続感染に至らなくても，機器内のファイルや設定がマルウェアにより書き換えられている可能性が考えられる．機器の復元方法として，機器のソフトウェアの再起動，電源の再起動，工場出荷状態への初期化，ファームウェア更新によりどの程度復元可能か検証を行った．

実証実験の結果，持続感染型マルウェアのターゲットとする機器を用いて解析することにより，機器独自の機能を悪用してマルウェアのバイナリおよびプロセスを自動起動する，持続感染の実現手法を解析可能であった．解析により改変対象のファイルとディレクトリが明確となり，駆除手法の検証を行うことが可能であった．また，マルウェアが持続感染のターゲットとする機器以外を用いて解析をした場合には，持続感染の試みを観測できないことを示した．

3.3 複合ディレクトリ型サンドボックスを用いた解析

IoT 機器の実機を解析に用いる場合，マルウェアに応じて多くの機器を買い集めなければならず，コスト面の課題がある．また，IoT 機器は長年に渡り使用されることが多く，古い機器の入手性の問題が考えられる．入手できた場合でも，機器の備える機能によってはマルウェアの動的解析に使用できない場合も想定される．そこで，機器の特徴を最低限模擬したサンドボックスを用いて，特定の機器をターゲットにしたマルウェアの振る舞いの解析を試みた．

IoT 機器を模擬する方法として，機器のディレクトリ構成に着目をした．IoT 機器のファームウェア更新用ファイルはメーカーのホームページ上に無償で公開されている場合が多く，これらの一定数は展開してディレクトリやファイルを抽出可能であることが知られている．そこで，多くの IoT 機器のファームウェア更新用ファイルを収集し抽出したディレクトリ構成の和集合となる“複合ディレクトリ型サンドボックス”を用いた解析手法を提案した．まず，インターネット上から 3,199 のファームウェア更新用ファイルを収集し，308 製品・1,103 ファイルからディレクトリ構成を抽出した．これらのディレクトリ構成の和集合を

持つ“複合ディレクトリ型サンドボックス”を構築し，実証実験として 87 のマルウェア検体を解析した．持続感染の可能性が指摘される 4 検体のうち，2 検体で実際に持続感染を試みるためのファイル生成の振る舞いを解析可能であった．提案手法により持続感染の実現手法を解析することが可能であり，駆除手法の検討や堅牢な設計の対策に活かすことが可能である．マルウェアが悪用するディレクトリが存在しない場合には，持続感染の振る舞いを観測できず，本手法の有効性が示された．加えて，“複合ディレクトリ型サンドボックス”を構成する 308 製品・1,103 のファームウェア更新用ファイルから，マルウェアが悪用するディレクトリを有するものを抽出することにより，潜在的に持続感染する可能性のある機器の一覧を抽出した．これらの機器のメーカーへの注意喚起を行うことにより，持続感染の被害拡大を防ぐことが可能である．

第 4 章
持続感染型IoTマルウェアの実態調査と実機による概念実証

Mirai に代表される，組込み機器を攻撃対象とした IoT マルウェアのほとんどは，感染した機器の電源を再起動することで消滅することが知られている．しかしながら，特定の機器を狙った，電源の再起動のみでは消えない“持続感染”を引き起こす IoT マルウェアの事例も報告されている．このようなマルウェアが流行した場合，被害の深刻化が容易に想定される．本研究では，まず IoT 機器の実機とマルウェア検体を用いて持続感染の有無の調査を行った．7 種類の IoT 機器と計 45 の IoT マルウェア検体による実験の結果，いずれの機器，マルウェア検体についても持続感染は確認されなかった．しかし，1 つの機器においては，“システム設定改変攻撃”により，容易に持続感染に至る危険な状態であることが分かった．他の 2 つの機器においても，“ファームウェア不正書換攻撃”による，IoT マルウェアの持続感染が成立することを実証した．これらの実証実験をもとに，持続感染が成立する要因を分析し，持続感染防止に効果的なファームウェア構成について議論を行う．

4.1 はじめに

近年，世界中の様々なモノがインターネットに接続されるようになり，このような状態はモノのインターネット (IoT) と称されている．インターネットに接続する組込み機器は増加の一途をたどり，2018 年の 307 億台から 2021 年の 448 億台への増加が予測されている [1]．一方で，組込み機器を狙った攻撃は，機器の増加のペースを上回る毎年 4〜5 倍のペースで増加 [36] しており対策が急務である．従来，組込み機器は，機器ごとに専用の OS やプログラムが搭載されることが多かったが，近年は汎用コンピューターと同様に，Linux[6] をはじめとするオープンソースソフトウェアを活用する事例が増えている．組込み機器のファームウェア更新用ファイルを収集し静的解析を行った大規模調査 [3] によると，86%の IoT 機器の OS は Linux であった．オープンソースソフトウェアの活用は，開発の期間短縮・コスト削減の観点で非常に効果的である．一方で，オープンソースソフトウェアはソースコードが公開されており，攻撃者にとっても有益な情報となり得る．さらには，オープンソースソフトウェアに脆弱性がある場合に複数機器にまたがる脆弱性となり得る．このように，オープンソースソフトウェアはセキュリティ面のリスクを内包している．

IoT を議論するうえで，モノとネットワーク機器を区別するかは解釈が分かれるところである．“モノのインターネット”という言葉より，厳密にモノだけを指す考え方がある一方，文献 [37] のように，セキュリティを語るうえではモノとネットワーク機器をあわせて IoT 機器と称する考え方がある．モノとネットワーク機器は，いずれも本質的にはイン

ターネットに接続された組込み機器であり，マルウェア感染の観点において共通した攻撃が通用すると考えられる．組込み機器を模したハニーポットを用いた攻撃元の調査 [26] によると，モノとネットワーク機器の両方から攻撃を受けることが示されている．組込み機器をターゲットとしたマルウェア “Mirai” はモノとネットワーク機器のいずれにも感染することが示されている [11][38] ．このような背景より，本論文では，インターネットに接続された組込み機器を総称して IoT 機器と定義する．

IoT 機器の多くは購入後すぐに利用できるよう設定されて出荷されていることから，管理画面等が初期パスワードのまま公開され放置されているケースが多く，不正侵入やマルウェア感染の原因となっている [39][40]．2016 年に流行したマルウェア “Mirai”[38] は，Linux を搭載した IoT 機器に対して，汎用的な攻撃が通用することを示す結果となった．このように，十分なマルウェア対策をしていない多くの IoT 機器がマルウェアに感染しており，感染した機器は攻撃者により大規模な DDoS 攻撃等に利用されている [41].

Mirai に代表される，IoT 機器を攻撃対象とするマルウェアの多くは，感染した機器の電源を再起動することで消滅することが知られている．一方で，特定の機器を対象とした，電源の再起動のみでは消えない“持続感染”を引き起こすマルウェア SYNFul Knock[42]，機器を故障に至らせるマルウェア Bricker Bot[43] が報告されている．

IoT 機器の用途によっては，常時通電の機器も多数存在しており，このような機器に対しては持続感染の有無に関わらず長期間にわたる感染が可能である．一方で，機器の再起動による IoT マルウェアの駆除方法が公的機関より周知されている [37]．将来，駆除方法が一般に広まった状況下において，持続感染をしないマルウェアは，大規模なボットネットを構築・維持するために短期間で大量の機器を感染させる必要があり，感染拡大時の特徴的な通信の増加を観測する目的として，ダークネットのトラフィック観測 [44] や，ハニーポットによる観測が引き続き効果的である．しかしながら，持続感染を引き起こすマルウェアは長期にわたりゆっくりと感染を広げることが可能になり，観測が困難になると考えられる．現に，2018 年 5 月に一般に認知されたマルウェア “VPNFilter” は 2016 年頃から活動していると考えられるものの，長期にわたり多くのセキュリティ研究者に認知されることなく 50 万台以上への感染を拡大した [14]．このように持続感染の脅威は大きく，対策が急務である．

本研究では，まず IoT 機器の実機とマルウェア検体を用いた感染実験を行った．主にハニーポットを用いてマルウェア検体の収集と攻撃元機器の特定を行い，7 種類の機器と 45 のマルウェア検体を用意した．これらの機器・マルウェア検体を用いた感染実験を行い，いずれの機器，マルウェア検体についても，機器の電源の再起動後にマルウェアのプロセスが消滅，すなわち持続感染が生じないことを確認した．

続いて，概念実証用の疑似マルウェアを用い，これらの機器に対して 2 種類の攻撃 1) システム設定改変攻撃 2) ファームウェア不正書換攻撃 により持続感染を実証した．システム設定改変攻撃では，前述の 7 種類の機器のうち，1 種類の機器において持続感染が実現し得ることを実証した．本機器と同様の特徴のファイルシステムを持つ機器を追加で 3 種類用意し，いずれの機器においても持続感染が実現し得ることを実証した．システム設定改

変攻撃において持続感染が生じなかった 6 種類の機器に対してファームウェア不正書換攻撃を実施し，2 種類の機器で持続感染が実現し得ることを実証した．最後に，これらの実証実験をもとに持続感染の成立条件を分析したうえで，対策方法について議論する．

本研究により実証した攻撃手法は，Linux の標準的な機能に着目した手法であり，本研究で実証した機器にとどまらず，一般的に通用する攻撃である．特にシステム設定改変攻撃は現在流行するマルウェアの感染方法のわずかな変更で実現が可能であり，非常に大きな脅威である．本研究はこのような危機的状況と対策方法を IoT 機器開発者に示し，IoT 機器のセキュリティ改善を促すものである．

本研究の貢献を以下に示す．

- IoT 機器に対するマルウェア持続感染の成立条件を分析した．
- 持続感染の成立条件をもとに攻撃モデルを定義のうえ，疑似マルウェアを用いた概念実証を行い，市販の機器に持続感染を実現し得ることを実証した．
- マルウェアの持続感染を防ぐためのファームウェア設計思想を議論した．

4.2 持続感染の攻撃モデル

持続感染を実現するためには，下記のマルウェア持続感染の成立条件をすべて満たす必要がある．

1. 機器上にマルウェアのバイナリファイルを保存する
2. 機器の電源再起動後にマルウェアのプロセスが自動起動するよう，機器の起動スクリプトを改変する
3. 上記 2 つの変更が，機器の再起動後も維持されるようにする

持続感染が実現するかどうかは，感染後のマルウェアの振る舞いと関係なく，感染時に機器のどの程度の深さまで感染するかに依存する．本条件を満たす攻撃モデルを以下に 2 つ定義する．

4.2.1 システム設定改変攻撃

本攻撃モデルを図 4.1 に示す．動作中の機器に対し何かしらの脆弱性を悪用して管理者権限によるリモートコマンド実行を行い，マルウェアを格納したうえでマルウェアを自動起動するよう機器の設定を改変する方法である．マルウェアを自動起動する手法は，“initd”[45] や “bashrc”[46] のような機器の起動時に自動的に実行されるスクリプトを改変する，もしくは “crontab”[47] のようなタスクスケジューラーを用いる方法が挙げられる．IoT 機器は，PC やサーバーと違い，機器の電源再起動時に一部の設定情報を除き変更を破棄する設計思想で作られている．機器自体が起動時もしくはシャットダウン時にリカバリ処理を

行っている可能性があり，電源再起動後に変更が維持されるかどうかは，機器の設計に依存すると考えられる．

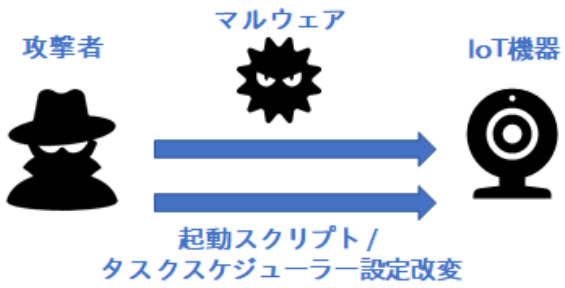

図 4.1 システム設定改変攻撃

4.2.2 ファームウェア不正書換攻撃

ファームウェア不正書換攻撃を説明するにあたり，本論文における“ファームウェア”の定義を行う．“ファームウェア”という用語は，「機器に組み込まれたソフトウェア」の意味で使用される場合と，「機器に組み込まれたソフトウェアを書き換えるためのデータ」の意味で使用される場合がある．本論文では下記のとおり明示的に使い分ける．

ファームウェア
: 機器に所望の動作をさせるために不揮発メモリに書き込まれているソフトウェア一式を指す．

ファームウェア更新用ファイル
: ファームウェアを更新するために，機器に入力するデータを示す．ファームウェアと等価な場合もあるが，ファームウェアの一部のみを更新するものや，アップデーターやチェックサムのような付加データを含む場合もある．

本攻撃モデルを図 4.2 に示す．マルウェアの格納および起動スクリプト改変を行った不正なファームウェア更新用ファイルを生成し，機器のファームウェアを書き換える方法である．IoT 機器の多くは購入後にファームウェアを書き換える機能があり，正規のファームウェア更新用ファイルはメーカーのウェブサイトに公開されている場合が多い．

4.3 既知の持続感染型マルウェアの考察

持続感染型マルウェア SYNful Knock[42] の攻撃対象の OS は Linux ではなくメーカー独自の OS であるものの，改変したファームウェア更新用ファイルを用いた攻撃であり，ファームウェア不正書換攻撃である．一方で，持続感染型マルウェア VPNFilter[14][15] の

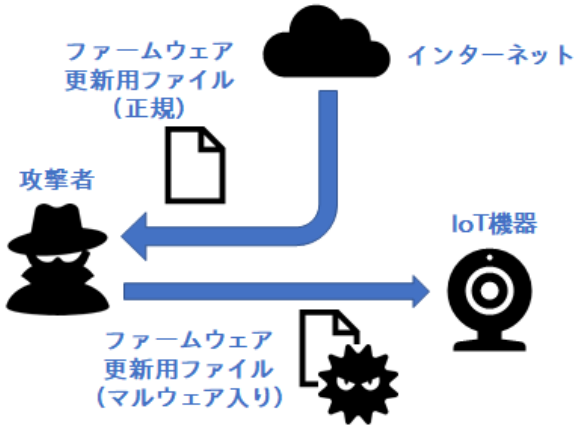

図 4.2 ファームウェア不正書換攻撃

感染方法は本論文執筆時点で明らかになっていない．しかし，プログラム自動起動の仕組みである crontab の設定ファイルを改変し持続感染を実現していることが分かっている．そのため，システム設定改変攻撃もしくはファームウェア不正書換攻撃のいずれかであると推測される．

4.4 実証実験

最初に実験 1 では，市販の機器と実際のマルウェア検体を用いて持続感染の有無の調査を行う．続いて実験 2 では，システム設定改変攻撃における持続感染の実現性を実証する．最後に実験 3 では，ファームウェア不正書換攻撃における持続感染の実現性を実証する．本実証実験で用いる機材を表 4.1 に示す．機器 A〜G は実験 1〜実験 3 で使用する．機器 H〜J は実験 2 および実験 3 で使用する．

4.4.1 実験 1

まず，概念実証に使用する機器に対する既存マルウェアによる持続感染の有無を確認するため，実機およびマルウェア検体を用いた感染実験を実施する．IoT 機器の内部システムはメーカーや機器の種類により異なり，マルウェアの挙動も異なると推測されるため，7 種類の実機と計 45 のマルウェア検体を用いて実証実験を行う．

4.4.1.1 実験方法

実験手順を以下に示す．

1. 機器内部のファイルシステム (全ファイル名とファイルサイズ) とプロセスを感染前の状態として記録する

表 4.1 機材一覧

機器	種類	国	アーキテクチャ
A	IP Camera	台湾	ARM
B	プリンター	アメリカ	ARM
C	ルータ	台湾	MIPS
D	Wi-Fi ストレージ/ポケット Wi-Fi	日本	MIPSEL
E	Wi-Fi ストレージ/ポケット Wi-Fi	日本	MIPSEL
F	Wi-Fi ストレージ/ポケット Wi-Fi	アメリカ	MIPSEL
G	衛星放送受信機	ドイツ	SH
H	モバイル Wi-Fi ルータ	日本	ARM
I	モバイル Wi-Fi ルータ	フランス	ARM
J	モバイル Wi-Fi ルータ	中国	ARM

2. 事前に入手したマルウェアバイナリファイルを機器に転送し実行する
3. マルウェア実行後，プロセスの増加や攻撃者の C&C サーバーに対する通信を開始するなどにより機器がマルウェア感染状態になったことを確認する
4. 機器の電源を再起動する
5. 手順 1 で記録した感染前の機器のファイルシステムやプロセスと比較し，感染の持続性を評価する

実験環境を図 4.3 に示す．通信制御・観測用マシンは IoT 機器からインターネットへの通信の制御と観測を行う他，IoT 機器に対するマルウェアバイナリファイルの転送と，機器のファイルシステム及びプロセスの記録も行う．

4.4.1.2 実験用機器と検体の準備

本項では実験に使用した機器と検体の準備について記す．まず，実験に使用するための機器では何かしらの手段で OS コマンドのリモート実行ができる必要がある．そこで，Telnet 接続が可能な機器の情報を当研究室のハニーポット [26][48][49] を用いて収集した．具体的には，ハニーポットで 23/tcp 宛の攻撃を観測した際に，攻撃元の IP アドレスに対してスキャンすることで攻撃元ホストの Telnet のバナー情報を取得し，マルウェアに感染したと考えられる機器の製品名や型番を特定した．そして，同機種をいくつか購入し実際に Telnet でログイン可能なことを確認した機器を実験に使用した．本手法により，表 4.1 の機器 A ～機器 G の 7 種類の機器を用意した．

実験に使用した検体についても，主にハニーポットを利用して準備を行った．まず，ハニーポットを用いて攻撃者が実行するマルウェアをダウンロードするシェルコマンドを

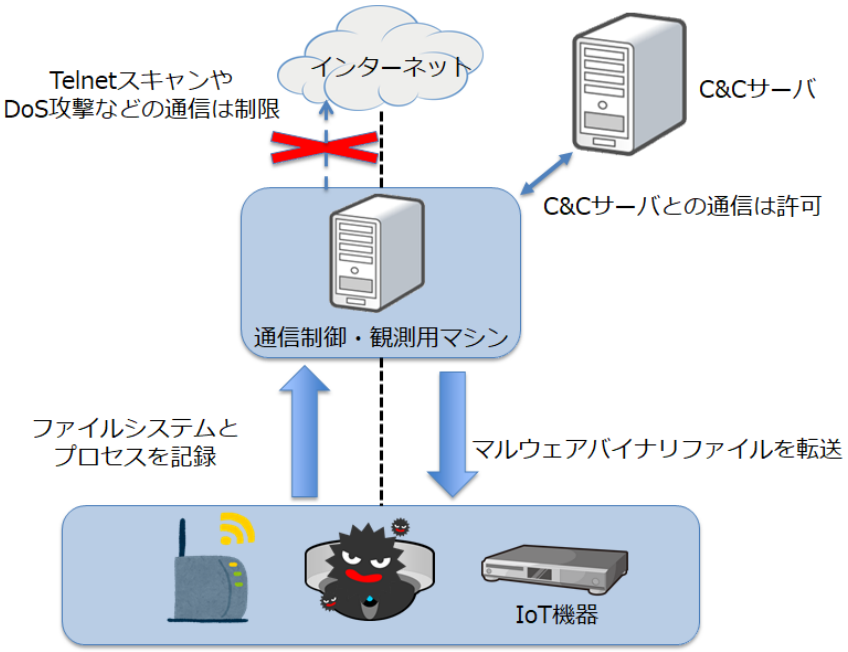

図 4.3 実験環境

観測する．その後，別マシンでこのコマンドを再現し攻撃者のダウンロードサーバーからマルウェアをダウンロードした．また，一部の検体についてはマルウェア解析 Web サイト「VirusTotal」[50] から検体をダウンロードした．機器の CPU アーキテクチャ毎に，ARM[4] で動作するものを 11 検体 (表 A.1)，MIPS[5] で動作するものを 12 検体 (表 A.2)，MIPSEL[5] で動作するものを 12 検体 (表 A.3)，SH(Super H)[51] で動作するものを 10 検体 (表 A.4) の計 45 検体を使用した．マルウェア検体 1〜7，12〜18，24〜30，36〜43 の 29 検体については，我々が運用する IoT ハニーポットで取得したものであり，残りの 16 検体は「VirusTotal」から取得した．いずれの検体も 2016 年 9 月から 12 月にかけて収集を行った．

4.4.1.3 実験結果

本項では感染実験の結果を表 4.2〜表 4.8 に示す．結果表中の「○」は機器の電源再起動により，悪性プロセスもしくはマルウェアのバイナリファイルが残留したことを示す．「×」は機器の電源再起動により，悪性プロセスもしくはマルウェアのバイナリファイルが削除されたことを示す．

機器 A の実験結果を表 4.2 に示す．検体 7 のマルウェアは動作しなかったが，動作した他の検体ではいずれも機器の電源の再起動により，マルウェアのバイナリとプロセスはともに消滅した．

機器 B の実験結果を表 4.3 に示す．この機器では，検体 4,7,8 のマルウェアは動作しなかった．動作した他の検体ではいずれも機器の電源の再起動により，マルウェアのプロセ

表 4.2 実験 1 結果 (機器 A)

検体	マルウェアの バイナリの残留	マルウェアの プロセスの残留
検体 1	×	×
検体 2	×	×
検体 3	×	×
検体 4	×	×
検体 5	×	×
検体 6	×	×
検体 7	動作せず	動作せず
検体 8	×	×
検体 9	×	×
検体 10	×	×
検体 11	×	×

表 4.3 実験 1 結果 (機器 B)

検体	マルウェアの バイナリの残留	マルウェアの プロセスの残留
検体 1	○	×
検体 2	○	×
検体 3	○	×
検体 4	動作せず	動作せず
検体 5	○	×
検体 6	○	×
検体 7	動作せず	動作せず
検体 8	動作せず	動作せず
検体 9	×	×
検体 10	○	×
検体 11	○	×

スは消滅したが，バイナリは残留した．検体 9 では，マルウェアのバイナリが残っていなかったが，これはマルウェアの実行後に自身のバイナリファイルを削除するという挙動によるものである．

機器 C の実験結果を表 4.4 に示す．実験に用いた 12 検体すべてにおいて，機器の電源の

表 4.4 実験 1 結果 (機器 C)

検体	マルウェアの バイナリの残留	マルウェアの プロセスの残留
検体 12	×	×
検体 13	×	×
検体 14	×	×
検体 15	×	×
検体 16	×	×
検体 17	×	×
検体 18	×	×
検体 19	×	×
検体 20	×	×
検体 21	×	×
検体 22	×	×
検体 23	×	×

再起動により，マルウェアのバイナリとプロセスはともに消滅した．

機器 D の実験結果を表 4.5 に示す．検体 32 のマルウェアは動作しなかったが，動作した他の検体ではいずれも機器の電源の再起動により，マルウェアのバイナリとプロセスはともに消滅した．

機器 E の実験結果を表 4.6 に示す．検体 26 のマルウェアは動作しなかったが，動作した他の検体ではいずれも機器の電源の再起動により，マルウェアのバイナリとプロセスはともに消滅した．

機器 F の実験結果を表 4.7 に示す．実験に用いた 12 検体すべてにおいて，機器の電源の再起動により，マルウェアのバイナリとプロセスはともに消滅した．

機器 G の実験結果を表 4.8 に示す．検体 37 と 40 は動作しなかったが，動作した他の検体ではいずれも機器の電源の再起動により，マルウェアのバイナリとプロセスはともに消滅した．

4.4.2 実験 2

実験 1 では，いずれの機器・マルウェアの組み合わせにおいても，持続感染は発生しなかった．実験 2 では，これらの機器に対してシステム設定改変攻撃を行い，持続感染の実現性を実証する．

表 4.5 実験 1 結果 (機器 D)

検体	マルウェアの バイナリの残留	マルウェアの プロセスの残留
検体 24	×	×
検体 25	×	×
検体 26	×	×
検体 27	×	×
検体 28	×	×
検体 29	×	×
検体 30	×	×
検体 31	×	×
検体 32	動作せず	動作せず
検体 33	×	×
検体 34	×	×
検体 35	×	×

表 4.6 実験 1 結果 (機器 E)

検体	マルウェアの バイナリの残留	マルウェアの プロセスの残留
検体 24	×	×
検体 25	×	×
検体 26	動作せず	動作せず
検体 27	×	×
検体 28	×	×
検体 29	×	×
検体 30	×	×
検体 31	×	×
検体 32	×	×
検体 33	×	×
検体 34	×	×
検体 35	×	×

表 4.7 実験 1 結果 (機器 F)

検体	マルウェアの バイナリの残留	マルウェアの プロセスの残留
検体 24	×	×
検体 25	×	×
検体 26	×	×
検体 27	×	×
検体 28	×	×
検体 29	×	×
検体 30	×	×
検体 31	×	×
検体 32	×	×
検体 33	×	×
検体 34	×	×
検体 35	×	×

表 4.8 実験 1 結果 (機器 G)

検体	マルウェアの バイナリの残留	マルウェアの プロセスの残留
検体 36	×	×
検体 37	動作せず	動作せず
検体 38	×	×
検体 39	×	×
検体 40	動作せず	動作せず
検体 41	×	×
検体 42	×	×
検体 43	×	×
検体 44	×	×
検体 45	×	×

表 4.9 実験 2 結果

機器	疑似マルウェアのバイナリの残留	起動スクリプト改変の残留	疑似マルウェアのプロセスの残留
A	×	×	×
B	○	○	○
C	×	×	×
D	×	×	×
E	×	×	×
F	×	×	×
G	×	×	×
H	○	○	○
I	○	○	○
J	○	○	○

4.4.2.1 実験方法

持続感染は感染後のマルウェアの振る舞いと関係なく，感染時に機器のどの程度の深さまで感染するかに依存する．そのため，機器上でプロセスを起動するだけの疑似マルウェアを用意し実験を行う．下記の実験手順による感染により，各機器で持続感染に至るかどうかを検証する．実験手順を以下に示す．

1. 疑似マルウェアのバイナリファイルを機器に転送し実行する
2. マルウェア実行後，疑似マルウェアのプロセスが開始し感染状態となったことを確認する
3. 機器の起動時に疑似マルウェアのプロセスが自動起動するよう，起動スクリプトもしくはタスクスケジューラーの設定を改変する
4. 機器の電源を再起動する
5. 再起動後，疑似マルウェアのプロセスが開始したことを確認する

管理者権限によるリモートコマンド実行の手段として，本実験では機器の持つ telnet/SSH のパスワードの脆弱性を使用する．改変対象の起動スクリプトは，Linux で一般的に使われる”Init Script”[45] とする．機器が何かしらの復旧機能を有していない場合，手順 3. により持続感染が生じると考えられる．

4.4.2.2 実験結果

実験結果を表 4.9 に示す．機器 B のみ再起動後に疑似マルウェアのプロセスが残留している状態であり，持続感染が生じている．一方，他の機器においては，疑似マルウェアのバ

表 4.10 ファイルシステム

機器	ファイルシステム
A	不明
B	UBIFS（読み書き可能）
C	squashfs（読み取り専用）
D	squashfs（読み取り専用）
E	squashfs（読み取り専用）
F	squashfs（読み取り専用）
G	cramfs（読み取り専用）
H	YAFFS2（読み書き可能）
I	YAFFS2（読み書き可能）
J	YAFFS2（読み書き可能）

イナリや起動スクリプト改変といった，機器に対する変更が復旧されていることが分かる．

各機器のファイルシステムを調査したところ，表 4.10 のとおりであった．機器 A はファイルシステムが特定できなかったものの，挙動より読み取り専用ファイルシステムと考えられる．持続感染とファイルシステムの関連を明確にするため，読み書き可能ファイルシステムを有する機器を表 4.1 の機器 H〜機器 J のとおり用意し同様に実験を行ったところ，表 4.9 に示すとおり，機器 H〜機器 J すべてにおいて持続感染を確認した．本結果により持続感染を引き起こすかどうかはファイルシステムの性質に依存するといえる．読み書き可能ファイルシステムを有する機器に対しては，現状のマルウェアの感染方法に加え，起動スクリプトを改変する処理を加えるのみで持続感染を引き起こす可能性が高く，非常に危険な状況といえる．

4.4.3 実験 3

実験 3 ではファームウェア不正書換攻撃による持続感染の実現性を実証する．

4.4.3.1 実験方法

本実験では，実験 2 と同じ疑似マルウェアを使用する．実験手順を以下に示す．

1. メーカーのホームページより，正規のファームウェア更新用ファイルを入手する
2. 正規のファームウェア更新用ファイルの中からファイルシステムを抽出・展開する
3. ファイルシステム上に疑似マルウェアを格納する
4. 機器の起動時に疑似マルウェアのプロセスが自動起動するよう，ファイルシステム上の起動スクリプトを改変する

5. 改変したファイルシステムをもとに，ファームウェア更新用ファイルを再構築する
6. 改変したファームウェア更新用ファイルを用いて機器のファームウェアを書き換え，機器の電源を再起動する
7. 再起動後，疑似マルウェアのプロセスが開始したことを確認する

4.4.3.2 実験結果

本実験の結果，2 種類の機器で持続感染を確認した．他の 1 種類の機種では，機器への書き込みは行っていないものの，ファームウェア更新用ファイルの改変まで成功した．

以下に機器ごとの結果を示す．機器 A はメーカーのホームページよりファームウェア更新用ファイルを入手することができず，実験を行えなかった．機器 B,C,G は正規のファームウェア更新用ファイルを入手でき，またファイルシステムの抽出も実施できた．しかし，ファームウェア更新機能における，ファームウェア更新用ファイルの正当性判定処理を抽出できなかったため，ファームウェア更新用ファイルの再構築を行えなかった．機器 D,E は手順通り実行することにより，持続感染を生じることを確認した．機器 F はファームウェア更新用ファイルを改変し，機器に受け入れられるようチェックサムの改変まで行っており，持続感染を実現できる可能性が高い．しかし，機器の入手性の問題があるため，改変したファームウェア更新用ファイルの機器への書き込みは行わなかった．機器 H,I,J はファームウェア更新機能はモバイル回線を通じたオンライン更新機能しか提供されておらず，ファームウェア更新用ファイルを入手できなかった．

4.5 考察

4.4 節の実証実験では，実際にマルウェアの持続感染が実現されることを，実機を用いて実証した．システム設定改変攻撃とファームウェア不正書換攻撃のいずれにおいても，Linux の標準的なファイルシステムおよびシステム起動スクリプトに着目した持続感染を実証しており，他機器へも同様の攻撃が通用する可能性が高いと言える．以下では，それぞれの攻撃モデルに対する持続感染の成立条件および対策について考察を行う．IoT 機器は機器の構成や用途により，記載の対策を適用できない場合が考えられる．いずれの攻撃モデルにおいても，持続感染の成立条件は AND 条件であるため，一つ以上の条件に対する対策を行うことにより，持続感染の成立を防止することが可能である．

4.5.1 システム設定改変攻撃

下記の条件をすべて満たす場合に，システム設定改変攻撃による持続感染が成立する．

(a) 機器上にマルウェアのバイナリファイルを保存できること
(b) 機器の電源再起動後にマルウェアのプロセスが自動起動するよう，機器の起動スクリプトを改変できること

(c) 上記 2 つの変更が，機器の再起動後も維持されるようにすること

(a)(b) の対策として，攻撃者にリモート接続をさせないことが挙げられる．具体的には，下記の対策が効果的である．

(1) リモート接続を許可しない

(2) アカウントロック機能を導入する

(3) 機器の初回仕様時にパスワード変更を義務付ける

(c) の対策として，読み取り専用ファイルシステムの活用が挙げられる．ファイルシステム全体に適用することが望ましいが，機器の機能上困難な場合は，最低限として起動スクリプトやタスクスケジューラーの領域に適用すべきである．

4.5.2 ファームウェア不正書換攻撃

下記の条件をすべて満たす場合に，ファームウェア不正書換攻撃による持続感染が成立する．

(A) 機器のファームウェア更新用ファイルを入手可能であること

(B) 機器のファームウェア更新用ファイルを解析・改変可能であること

(C) 改変後のファームウェア更新用ファイルが機器に受け入れられること

(D) ファームウェア更新機能を攻撃者が利用できること

(A) の対策として，ファームウェア更新用ファイルの隠蔽が挙げられる．具体的には，web 上でのファームウェア更新用ファイルの配布をせず，機器自体が暗号化された通信経路を用いて更新サーバーからファームウェア更新用ファイルを取得するという方法が挙げられる．しかしながら，オフライン環境でファームウェア更新が行えないなどの制約も発生し，対応が困難な場合も多いと考えらえる．

(B) の対策として，ファームウェア更新用ファイルの暗号化が挙げられる．暗号の不適切な使用は脆弱性の原因になり得るため，使用方法には十分な配慮が必要である．暗号の不適切な使用の具体例の一つとして，XOR 暗号の使用が挙げられる．XOR 暗号は既知平文攻撃により容易に暗号キーを推定できる問題がある [52]．平文のファームウェア更新用ファイルでは，ファイルシステムの空き領域や，ファームウェア更新用ファイルのサイズ調整の領域が 0 埋めなど容易に推測可能な bit 配列であることが多く，既知平文攻撃が通用する恐れがある．また，このように平文のファームウェア更新用ファイルは偏りのあるデータの可能性があり，ECB モードのようなブロック暗号では平文漏洩のリスクがある．ファームウェア更新用ファイルの暗号化は，ECB モード以外の暗号モードを使用するべきである．

(C) の対策として，ファームウェア更新用ファイルにメーカーの電子署名を付与したうえで，ファームウェア更新機能で署名検証を行う方法が挙げられる．

(D) の対策として，ファームウェア更新機能をリモートから実行できないよう制限する方法が考えられる．しかしながら，ソーシャルエンジニアリングのように，機器の管理者自身に悪意のあるファームウェア更新用ファイルを書き込ませる方法も考えられ，対策としては不十分である．

4.5.3 研究倫理的考察

当該研究はメンロレポート [53] に規定される研究倫理原則に基づき実施した．IoT マルウェアの持続感染の有無は，対策を検討する上で重要な要素であるにも関わらず，これまで十分な研究が実施されておらず，正確な情報が提供されていないのが現状である．特に 2016 年に大流行した Mirai やその亜種が持続感染機能を有していないことから，一般に IoT マルウェアが持続感染しないという誤った認識に基づき対策が検討される恐れがある．本研究は，IoT マルウェアの持続感染の可能性について，機器の性質をふまえて体系的に検討すると共に複数の具体的対策方法を示すものであり，今後出現する可能性がある持続感染型の IoT マルウェアへの対策に資すると考える．一方，本研究成果の悪用の影響を最小化するため，具体的な機器の情報の匿名化を行うことで直接的な悪用を防ぐと共に，情報処理推進機構・JPCERT/CC に当該研究成果に関する情報提供済である．本論文の執筆にあたり新たに持続感染の可能性が明らかとなった機器に関して情報処理推進機構を窓口として JPCERT/CC 及び機器のメーカーへ情報提供済である．このように本研究により得られる恩恵は，その悪用による潜在的な危害を大きく上回ると考える．

4.6 まとめと本章における課題

本研究では，Linux を使用した IoT 機器において，7 種類の IoT 機器と計 45 の IoT マルウェア検体による感染実験を行い，これらの機器・マルウェアにおいて持続感染が発生していないことを示した．続いて，“システム設定改変攻撃”と“ファームウェア不正書換攻撃”の 2 種類の手法により，これらの機器に持続感染が実現可能であることを示した．特に，“システム設定改変攻撃”では持続感染能力を持たないマルウェアに対して，わずかに変更を加えるのみで持続感染が生じる危険性を示した．これらの実証をもとに，“システム設定改変攻撃”と“ファームウェア不正書換攻撃”のそれぞれにおいて持続感染の成立条件を分析したうえで，持続感染の対策について考察を行った．

本研究の課題として，感染実験に使用したマルウェア検体の網羅性が挙げられる．本論文の執筆時点では本研究で使用したマルウェア検体の亜種による攻撃が依然として多く観測されており，この点は現状の脅威を反映しているものと言える．しかしながら，現状の観測環境で VPNFilter のような新種のマルウェアの検体の収集や感染活動の観測を行えない点に問題がある．今後はハニーポットを改善のうえ，攻撃の観測および感染実験の網羅

性を高めていきたい.

第 5 章
IoT機器の実機を用いたマルウェア動的解析手法の検証

IoT 機器の増加に伴い，これらの機器をターゲットとしたマルウェアが急増し多様化している．従来の IoT マルウェアは多くの種類の機器に共通する機能を悪用することが多かったため，仮想環境上に構築した汎用的な解析環境で動的解析が可能であった．近年，特定の機器の機能に依存したマルウェアが観測されており，汎用的な環境ではこのようなマルウェアの挙動を正しく解析することができない．また，IoT 機器は構成が多種多様かつハードウェアの挙動と密接に関係することが多いが，このような点を攻撃するマルウェアが登場した場合，機器を忠実に再現した解析環境が求められる．しかし，仮想環境上でハードウェアの挙動を忠実にエミュレートすることは困難である．さらに，仮想環境を検知して振る舞いを変える IoT マルウェアの存在が知られている．このような IoT マルウェアを動的解析する場合に IoT 機器の実機を用いる必要があるが，これまで実機を用いた動的解析に関して詳しく検証されていない．本章では，IoT マルウェアの実機動的環境に最低限必要な要件を分析する．続いて，5 種類の IoT 機器および仮想環境上で，計 87 の IoT マルウェア検体を用いて実証実験を行った．最後に，実証実験をもとに実機による動的解析手法の問題点や制限の議論を行った．本手法を用いることにより，マルウェアの IoT 機器に対する持続感染の実現の仕組みやプロセスの隠蔽，ファイル改変といった機器内部に対する挙動，機器外部に対する通信挙動を解析することが可能であった．

5.1 はじめに

近年，IoT(Internet of Things) と称されるモノのインターネットが注目を浴びている．これまでスタンドアロンで用いられていたモノがインターネットに接続されデータやサービスを連携することにより高い付加価値を生み出している．インターネットにつながるモノすなわち IoT 機器は急増し続けており，2018 年の 307 億台から 2021 年の 448 億台への増加が予測されている [1]．IoT 機器は PC と比べて脆弱な機器が多く，機器の増加の一方でこれらの機器を狙うサイバー攻撃も活発化している．インターネット上に流れる攻撃パケットは 2015 年から 2018 年の 3 年間で 3.7 倍に急増しており，約半数は IoT 機器を狙ったものと推測されている [2]．このように，IoT 機器に対するサイバー攻撃への対策は急務である．

IoT 機器は特定の用途向けに作られる機器が多く，安価に製造されており，機器の用途に応じた必要最低限のハードウェアリソースで構成される．このように制限された構成で動作するよう，OS やソフトウェアの軽量化が行われることが多い．IoT 機器の OS として

Linux が使われる場合が多いが，このような制約のため PC 向け Linux と思想が異なる．たとえば，不要な OS コマンドやライブラリや標準ソフトウェアを除去したり，Busybox[54] のような，機能制約のある軽量 OS コマンドへの置き換えが行われる．またユーザーインターフェースはメーカーが独自に用意した web 画面や制御用通信に限定されるなど，一般ユーザーが機器上の Linux を直接操作できない構成となっていることが多い．更にはストレージも一部の設定領域を除いて読み取り専用ファイルシステムによる保護が行われていることがある．このようなカスタマイズのため，IoT 機器は Linux のような汎用 OS を使用している場合においても，機種ごとの差異が大きいといえる．

2016 年に流行したマルウェア “Mirai” は，出現当初は汎用的なプロトコルである Telnet[55] を経由して感染を拡大した [11]．多くの IoT 機器の Linux に最低限備えられている機能や，共通するディレクトリ構成を悪用することで，多くの種類の機器に感染した．一方で，特定の IoT 機器にのみ通用する攻撃を行うマルウェアが登場している．たとえば，2018 年に特定の機器のディレクトリ構成に依存した動作を行うマルウェアを観測した [13]．また，IoT マルウェアの中には解析環境を検知して振る舞いを変えることにより，解析を妨害するものがあることが知られている [33]．このように，機器の構成に依存した振る舞いを行うマルウェアや，仮想環境検知を備えるマルウェアを動的解析する場合，IoT 機器の実機を用いる必要がある．

そこで本研究では，Linux を搭載した IoT 機器を対象として，実機を用いた動的解析手法を検討のうえ，実際の IoT 機器とマルウェア検体を用いて検証を行った．実機を用いた動的解析環境の機能として，「マルウェア検体の転送と実行」「マルウェア検体の挙動の観測」「解析環境の復元」の 3 つを必須機能と考える．まず，実機上でマルウェアの動的解析を行うにあたり，機器へマルウェア検体を転送し実行する必要がある．ネットワークを介して感染する IoT マルウェアは，機器の持つリモートコマンド実行の脆弱性の悪用により機器の OS コマンドが実行されることで，機器へ転送・実行されることが多い．多くのマルウェアが実際に感染する時と同様に，OS コマンドを実行することによりマルウェア検体の転送と実行を実現可能である．続いて，実機上でのマルウェア検体の挙動を観測する必要がある．他機器へのスキャンや感染拡大，C&C サーバーとの通信のような外部挙動の観測は，機器の通信を外側で収集することで比較的容易に観測可能である．一方，プロセスの起動・終了やファイル操作，環境設定の変更といった内部挙動を観測するためには，解析環境内に観測機構を用意する必要がある．IoT 機器のリソース面や環境依存性の観点から，利用者が機器に解析ツールを追加インストールできないケースが多いと想定し，機器が元々有する機能のみを用いて解析を行う方法を検討した．そのため，多くの機器に採用可能である反面，観測面で制約が考えられる．最後に，解析環境をマルウェア検体実行前の状態に復元する必要がある．そこで，多くの機器で用意されている機能である「機器の再起動」「工場出荷状態へのリセット」「ファームウェア更新」を用いた復元方法を検討した．

続いて，実証実験を通じて実際に実機上でマルウェアの動的解析が可能であることを示した．具体的には，5 種類の IoT 機器の実機を用いて動的解析環境を構築し，87 のマルウェア検体を用い，延べ 316 ケースの解析実験を行った．加えて，実機との比較のため，IoT 機

器を模した仮想環境を用いて解析実験を行い，結果の比較・考察を行った．実機上での OS コマンド実行の手段として，リモートコマンド実行の脆弱性のひとつである，容易な認証のまま公開されている Telnet を用いた．また，解析時は機器への物理アクセスが可能であると想定し，デバッグ用途として機器上に使用可能なまま残っていることがある UART[56] を OS コマンド実行の手段として検討した．

マルウェア検体の内部挙動の観測において，マルウェア検体の実行前後で機器上のプロセスリストを比較することにより，マルウェア検体によるプロセスの生成や，元々動作していた telnet のプロセスを終了する挙動が観測できた．同様にファイルリストを比較することにより，マルウェア検体が自身の実行ファイルを削除する挙動を観測した．さらには，同じマルウェア検体であっても機器により挙動が変わる現象を確認した．たとえば，特定の機器のみで，機器の電源再起動後もマルウェアによるファイル改変やプロセスが残り続ける持続感染を確認した．実機を用いた動的解析により，このマルウェアが持続感染を実現した仕組みを解析することが可能であった．外部挙動の観測においては，C&C サーバーとの通信やランダムなスキャン活動とみられる通信を観測した．解析環境の復元の検証において，5 機種中 4 機種では，reboot コマンドを使用した機器の再起動でマルウェア検体実行前の状態に復元することができた．残りの 1 機種はファームウェア更新機能により復元することができた．このように，今回実験に用いた IoT 機器とマルウェア検体においては，マルウェア検体を実行して通信挙動や基本的な内部挙動の観測ができ，マルウェア検体の実行後にシステムの復元が可能であることをできた．

大半の機材・マルウェアの組み合わせにおいては，実機と仮想環境で振る舞いが同じであった．4 検体では実機と仮想環境での振る舞いが異なり，実機を用いた動的解析により，これらのマルウェアの解析が可能であった．3 検体は，仮想環境で実行した場合のみ，機器のシャットダウンを行い，解析を阻害する振る舞いを観測した．1 検体は，特定の実機上でのみ，機器の電源再起動後もマルウェアによるファイル改変やプロセスが残り続ける持続感染を確認した．実機を用いた動的解析により，このマルウェアが持続感染を実現した仕組みを解析することが可能であった．

以降，5.2 節で実機を用いた動的解析手法の基本アイディアを説明する．5.3 節では IoT 機器の実機とマルウェア検体を用いた検証実験について説明する．仮想環境上でも同様の検証実験を行い，結果の比較・考察を行う．最後に 5.4 節でまとめと今後の課題を述べる．

本研究の貢献を以下に示す．

- IoT 機器の実機を用いたマルウェア動的解析を行うために最低限必要な機器の機能を分析した．
- IoT 機器の実機を用いたマルウェア動的解析手法を提案した．
- 実際のマルウェア検体と市販の IoT 機器を用いて提案手法の評価を行い，持続感染型マルウェアの解析において汎用的な解析環境と比較した提案手法の優位性を示した．

表 5.1 機材一覧

機器	種類	メーカー 本社所在地	CPU アーキテクチャ	CPU クロック (MHz)	RAM (MB)	ストレージ (MB)	Linux カーネル
A	ルータ	台湾	MIPSEL	600	256	128	2.6.22
B	Wi-Fi ストレージ	日本	MIPSEL	360	32	16	2.6.21
C	プリンター	アメリカ	ARM	400	128	128	2.6.31
D	ルータ	台湾	MIPS	400	256	16	2.6.30
E	Wi-FI ルータ	ラトビア	MIPS	600	128	128	3.3.5

表 5.2 仮想環境一覧

仮想環境	CPU アーキテクチャ	CPU クロック	RAM (MB)	ストレージ (MB)	Linux カーネル
X	MIPSEL	200	256	128	3.18.23
Y	ARM	125	256	256	4.4.50
Z	MIPS	200	256	128	3.18.23

5.2 マルウェアの実機動的解析環境

5.2.1 基本アイディア

本論文では，実機を用いた IoT マルウェアの動的解析に最低限必要な機能を以下の 3 つに分類する．

5.2.1.1 マルウェア検体の転送と実行

IoT 機器は PC と異なり特定の用途で用いられるため，ユーザーがマルウェアを含む任意のプログラムを機器上で実行するための機能を有していない場合がほとんどである．攻撃者は機器のリモートコマンド実行の脆弱性を悪用し，マルウェアをダウンロードし実行することが多い．マルウェアの動的解析においても，同様に OS コマンドを実行することにより，マルウェアのダウンロード・実行を実現可能である．また，解析時に機器への物理アクセスが可能な場合，OS コマンド実行の手段として，デバッグ用途として機器上に使用可能なまま残っていることがある UART が使用可能な場合がある．

5.2.1.2 マルウェア検体の挙動の観測

マルウェアの挙動は，他機器へのスキャンや感染拡大，C&C サーバーとの通信のような外部挙動と，機器内のプロセスの起動・終了やファイル操作のような内部挙動に分けら

れる．外部挙動の観測は，機器の通信を外側で収集することで比較的容易に観測可能である．一方，プロセスやファイルアクセス，環境設定の変更といった内部挙動を観測するためには，解析環境内に観測機構を用意する必要がある．しかし，IoT 機器は PC での解析と異なり，リソース制約が厳しい上，そもそも任意のプログラムの追加やシステム変更を行う機能が提供されておらず，高度な内部挙動観測機構を構築することは難しい．そこで，本研究では機器が元々有する OS コマンドを使用し，プロセスリストやディレクトリ構成，ファイルの内容などの基本的な内部状態を把握する．

5.2.1.3　解析環境の復元

マルウェア検体実行後，解析環境を感染前の状態に復元する必要がある．Bashlite[57] や Mirai 等の初期的な IoT マルウェアは持続感染する機能を有していないことが知られており，IoT 機器の電源を再起動することで復元することが可能である．しかし，昨今，持続感染機能を有するマルウェアも報告されており，電源を再起動するだけでは解析環境が適切に復元されたとはいえない [13][14][15]．そのため，感染前の機器のファイルシステムやプロセスリストと比較し，復元結果を評価する必要がある．適切に復元が行われるまで，下記の順に復元を試みる．

i. reboot コマンド実行による再起動
ii. 主電源による再起動
iii. 工場出荷状態へのリセット
iv. ファームウェア更新機能を用いて同じバージョンのファームウェアで上書き

5.2.2　機器が有するべき機能

5.2.1 項で述べた 3 つの機能を実現するために必要な，解析に利用する IoT 機器が有するべき機能を以下に示す．

a. マルウェア検体を機器内に転送，実行する手段があること．具体的には解析者が利用可能なリモートコマンド実行の脆弱性を有する，または UART 等のデバッグ機能を有することにより，OS コマンドを実行できること．
b. 機器の内部挙動を観測するための OS コマンドや機能が備わっていること．または，a. で示した方法により，観測機能 (観測用スクリプトなど) を追加できること．
c. reboot コマンドが存在し，解析者が実行できること．工場出荷状態へリセットする機能を有すること．ファームウェア更新機能を有すること．

5.3 実証実験

本節では，5.2 節で分析した実機を用いた IoT マルウェアの動的解析に必要な 3 つの機能が，IoT 機器の実機と実際の IoT マルウェアに対してどの程度利用可能かどうかを検証する．加えて，IoT 機器を模した仮想環境を用いた実験を実施し，実機を用いた場合との比較・考察を行う．

5.3.1 実験用機器の準備

本項では実験に使用した機器の準備について記す．実験に使用するための機器では，何かしらの手段で解析者が OS コマンドを実行できる必要がある．本実験では，比較のために Telnet による接続と UART の接続の両方で OS コマンドを実行可能な機器を選定した．まず，Telnet 接続が可能な機器の情報を当研究室のハニーポット [26][48][49] を用いて収集した．具体的には，ハニーポットで 23/tcp 宛の攻撃を観測した際に，攻撃元の IP アドレスに対してスキャンすることで攻撃元ホストの Telnet のバナー情報を取得し，マルウェアに感染したと考えられる機器の製品名や型番を特定した．そして，同機種をいくつか購入し実際に Telnet でログイン可能なことを確認した．続いて，機器を分解することにより，UART 端子が存在して UART 経由で OS コマンド実行が可能なことを確認した．本手法により，表 5.1 の機器 A〜機器 D の 4 種類の機器を用意した．加えて，機器 E として機器固有の仕組みをマルウェアに悪用された事例がある機器を用意した [13].

5.3.2 仮想環境の準備

実機および仮想環境における動的解析結果の比較にあたり，5.3.1 項で選定した機器のファームウェアをそのまま仮想環境上でエミュレートし実験できることが理想的である．しかしながら，先行研究 [29] においても，本実験に必要な外部通信が可能なレベルのエミュレートの成功率は 10%に満たないことが示されている．表 5.1 の 5 機器に関しても，先行研究 [29] の手法ではいずれもエミュレートできなかった．そのため，組込み機器向けディストリビューションである OpenWRT[27] を用いた仮想環境を構築した．表 5.2 に示す通り，CPU アーキテクチャごとにそれぞれ仮想環境 X,Y,Z を用意した．

5.3.3 実験用検体の準備

実験に使用した検体についても，主にハニーポットを利用して準備を行った．まず，ハニーポットを用いて攻撃者がマルウェアをダウンロードするために実行するシェルコマンドを観測する．その後，別マシンでこのコマンドを再現し攻撃者のダウンロードサーバーからマルウェアをダウンロードした．この手法で 83 検体を用意した．加えて，特定の機器 (機器 E) に依存した挙動を示すマルウェアを 1 つ用意した．

使用した検体を，Mipsel[5],Mips[5],Arm[4] の 3 つの CPU アーキテクチャ毎に，それぞれ表 B.5，表 B.6，表 B.7，表 B.8 に示す．マルウェアのファミリ名として，マルウェア解析 Web サイト「VirusTotal」[50] で検索した結果を併記する．検体 1〜42,44〜84 は 2016 年 12 月から 2019 年 1 月にかけて我々が運用する IoT ハニーポットで取得したものである．検体 43,86,87 は特定の機器に持続感染する可能性のある VPNFilter の 1st stage であり，文献 [15] にハッシュ値が記載されている検体を，2020 年 2 月に VirusTotal から取得した．検体 85 は特定の機器 (機器 E) に持続感染するマルウェアであり，2018 年 10 月に取得した．

5.3.4 動的解析環境

5.2.1 項で述べた 3 つの機能を実現可能な実機動的解析環境を図 5.1 に，仮想動的解析環境を図 5.2 にそれぞれ示す．

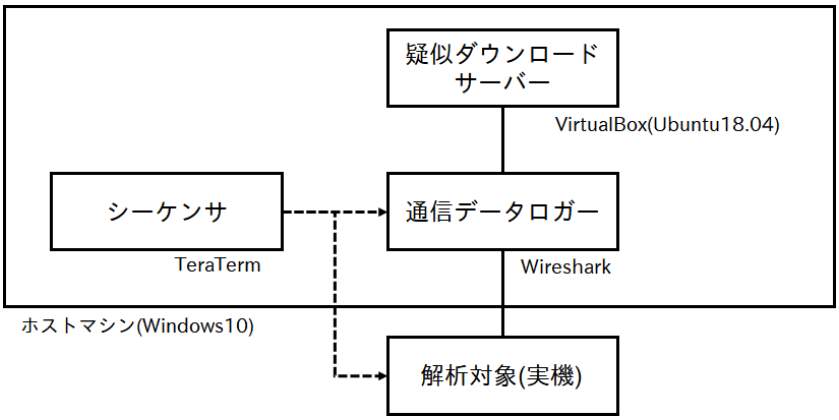

図 5.1 実機動的解析環境

シーケンサ

5.3.5 項に示す手順 1 から手順 6 における一連の OS コマンドを自動実行する．また，実行したコマンドおよびコマンド実行に伴う出力をログとして保存する．実機動的解析環境においては，ターミナルソフトである TeraTerm[58] のマクロ機能を用いて実現する．仮想動的解析環境においては，tmux[59] とシェルスクリプトにより実現する．

疑似ダウンロードサーバー

本実験で使用するマルウェア検体をダウンロード可能な web サーバーを仮想マシン上に構築する．web サーバーとして Apache2[60] を使用する．

通信データロガー

ネットワークプロトコルアナライザを用いてマルウェア検体が試みる通信データを

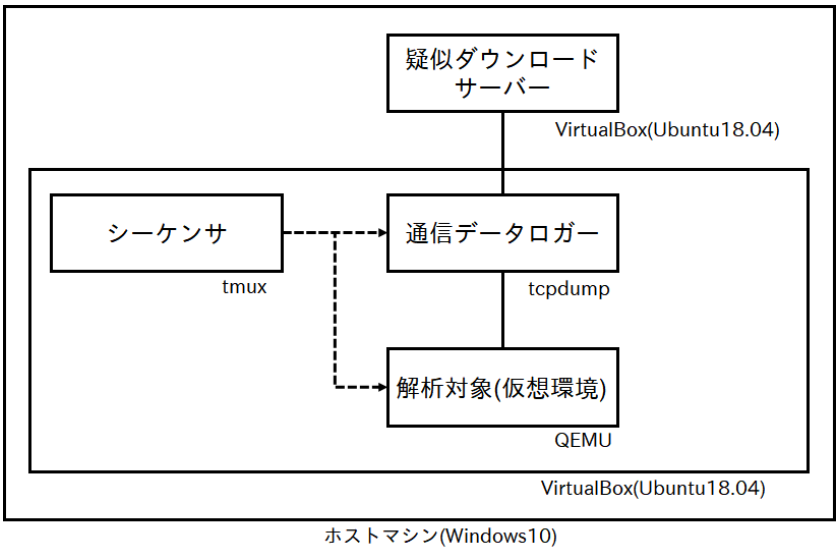

図 5.2 仮想動的解析環境

キャプチャする．実機動的解析環境においては Wireshark[61] を，仮想動的解析環境においては tcpdump[62] を用いて実現する．

マルウェア検体が外部へ攻撃活動を行う可能性を考慮し，本実験環境はインターネットとは接続しないものとした．そのため，外部挙動の観測は通信のコネクションの試行のみ観測可能である．

5.3.5 実験手順

実験手順を以下に示す．

1. **検体実行前の状態の記録**
   解析対象機器の OS コマンドを実行することにより，機器内部のファイルシステム (全ファイル名，ファイルサイズ，タイムスタンプ) とプロセスリストを感染前の状態として記録する．
2. **検体の実行**
   解析対象機器の OS コマンドを実行することにより，事前に入手したマルウェア検体のバイナリファイルを，疑似ダウンロードサーバーからダウンロードし実行する．
3. **感染の確認**
   マルウェア実行後，解析対象機器の OS コマンドを実行しプロセスの増加の有無を確認する．また，通信データロガーで解析対象機器と外部との通信の有無を確認する．プロセスの増加や攻撃者の C&C サーバーに対する通信を開始するなどにより機器がマルウェア感染状態になったことを確認する．

4. マルウェア検体の外部挙動の観測
   通信データロガーで解析対象機器の通信挙動を 5 分間記録する．
5. 検体実行後の状態の記録
   マルウェア実行の 5 分後に，再度解析対象機器の OS コマンドを実行することにより，機器内部のファイルシステム (全ファイル名，ファイルサイズ，タイムスタンプ) とプロセスリストを感染後の状態として記録する．
6. 機器の状態の復元（実機動的解析環境のみ実施）
   reboot コマンドを実行して機器を再起動する．
7. 復元後の機器の状態確認（実機動的解析環境のみ実施）
   手順 1 で記録した感染前の機器のファイルシステムやプロセスリストと比較し，復元結果を評価する．機器の状態を感染前の状態に復元できていない場合，主電源の再起動，工場出荷状態へリセット，ファームウェア更新機能により同じバージョンのファームウェアに上書きすることにより復元を試みる．
8. マルウェア検体の内部挙動の抽出
   最後に手順 1 および手順 5 の記録を比較し，差分をマルウェアの内部挙動として抽出する．

上記手順を，実機動的解析環境と仮想動的解析環境でそれぞれ実施する．実機動的解析環境においては，OS コマンド実行方法として Telnet を用いる場合と UART を用いる場合の両方を実施する．

仮想環境はその仕組み上容易に復元・複製が行えるため，復元に関する実験は行わない．

5.3.6 実験結果

5.3.6.1 内部挙動の解析

マルウェアの内部挙動に関する実験結果を示す．

内部挙動の観測にあたり，通信切断や通信異常が発生し内部挙動を正しく観測できない機器・マルウェア検体の組み合わせがあった．Telnet を用いた内部挙動の解析において，マルウェア検体実行に伴う Telnet の切断の有無を表 5.3 に示す．全 148 ケース中 18 ケースで Telnet の切断が発生し，解析を継続することができなかった．UART を用いた内部挙動の解析において，マルウェア検体実行に伴う UART の異常の有無を表 5.4 に示す．全 148 ケース中 6 ケースで UART の異常が発生し，解析を継続することができなかった．異常の内容はすべて UART 通信の欠落であり，正常に OS コマンド実行をすることができなくなった．本現象は機器 D のみで発生しており，同じ検体を機器 E で実行したところ，特に異常は生じなかった．仮想環境を用いた内部挙動の解析において，マルウェア検体実行に伴う QEMU ターミナルの異常の有無を表 5.5 に示す．検体 20,21,22 の 3 検体で検体実行直後にターミナルにシャットダウンを示すメッセージが表示された後，エミュレーショ

表 5.3 マルウェア検体実行後の Telnet の切断

機器	コマンド実行方法	Telnet 切断あり	Telnet 切断なし
A	Telnet	3	19
B	Telnet	3	19
C	Telnet	0	21
D	Telnet	7	37
E	Telnet	5	39

表 5.4 マルウェア検体実行後の UART 通信異常

機器	コマンド実行方法	UART 異常あり	UART 異常なし
A	UART	0	22
B	UART	0	22
C	UART	0	21
D	UART	6	38
E	UART	0	44

表 5.5 マルウェア検体実行後の仮想端末の通信異常

機器	UART 異常あり	UART 異常なし
X	3	19
Y	0	21
Z	0	44

ンが終了した．これらの通信異常が発生したケースでは内部挙動の解析を正しく行えないため，以降の内部挙動の解析結果は「検証不可」として示す．

次に，ファイル操作に関する内部挙動の実験結果を示す．まず，マルウェア検体による自身のバイナリの削除の挙動を表 5.6 に示す．実機と仮想環境での差異はなく，35%のマルウェア検体が自身のバイナリを削除した．続いて，マルウェア検体実行前後での機器のディレクトリおよびファイルの差異を表 5.7 に示す．14%のマルウェア検体が機器のディレクトリおよびファイルの変更を行った．実機と仮想環境の挙動を比較したところ，検体 85 のみ異なる結果となった．この検体は，機器 E を用いた解析において，Linux 標準のディ

表 5.6 マルウェア検体による自身の検体削除

機器	コマンド実行方法	検体削除あり	検体削除なし	検証不可
A	Telnet	7	12	3
B	Telnet	7	12	3
C	Telnet	1	20	0
D	Telnet	12	25	7
E	Telnet	14	25	5
A	UART	10	12	0
B	UART	10	12	0
C	UART	1	20	0
D	UART	13	25	6
E	UART	19	25	0
X	仮想端末	7	12	3
Y	仮想端末	1	20	0
Z	仮想端末	19	25	0

レクトリ構成 [63] に定められていない，機器固有のディレクトリ “/flash” へのマルウェアのバイナリのコピーを行ったうえ，コピーしたバイナリを実行するためのスクリプトファイルの生成を行った．一方，仮想環境 Z ではこれらのファイル生成の挙動を示さなかった．

最後に，プロセスに関する内部挙動の実験結果を表 5.8 に示す．すべてのマルウェア検体が何かしらのプロセスを起動した．通常 Linux では実行ファイル名がそのままプロセス名となる．しかし，多くのマルウェア検体が何かしらのプロセス名の偽装を行った．43%の検体は，マルウェア自身のプロセス名を “sshd”, “dropbear”, “telnet”, “ifconfig”, “tftpboot”, “bash” のように，機器に組み込まれていることが多いプログラムへの偽装を行った．28%の検体は，マルウェア自身のプロセス名の消去を行った．18%の検体は，マルウェア自身のプロセス名をランダムな文字列に変更した．マルウェア自身のプロセス名の偽装を行わない検体は 11%であった．プロセス名を偽装する挙動において，実機と仮想環境での差異はなかった．

5.3.6.2 外部挙動の解析

続いて，マルウェアの外部挙動に関する実験結果を示す．マルウェア検体実行後に機器が行った外部通信の特徴を表 5.9 に示す．80%の検体が，何かしらの通信を試みた．実験環境はインターネットへの接続がない環境であるが，コネクション試行や名前解決の傾向から，下記のように分類を行った．

表 5.7 マルウェア検体実行後の機器のファイルの差異

機器	コマンド実行方法	ファイル差異あり	ファイル差異なし	検証不可
A	Telnet	0	19	3
B	Telnet	0	19	3
C	Telnet	0	21	0
D	Telnet	8	29	7
E	Telnet	9	30	5
A	UART	0	22	0
B	UART	0	22	0
C	UART	0	21	0
D	UART	8	30	6
E	UART	14	30	0
X	仮想端末	0	19	3
Y	仮想端末	0	21	0
Z	仮想端末	14	30	0

表 5.8 マルウェアのプロセス名の偽装

機器	コマンド実行方法	プロセス名偽装	プロセス名消去	プロセス名ランダム化	偽装なし	検証不可
A	Telnet	5	5	7	2	3
B	Telnet	5	5	7	2	3
C	Telnet	3	11	1	5	1
D	Telnet	20	8	6	4	6
E	Telnet	21	8	6	4	5
A	UART	8	5	7	2	0
B	UART	8	5	7	2	0
C	UART	3	11	1	5	1
D	UART	19	8	7	4	6
E	UART	25	8	7	4	0
X	仮想端末	5	5	7	2	3
Y	仮想端末	3	11	1	5	1
Z	仮想端末	25	8	7	4	0

表 5.9 外部通信の特徴

機器	コマンド実行方法	スキャン+C&C 通信	スキャンのみ	C&C 通信のみ	P2P 通信のみ	通信なし
A	Telnet	7	1	10	0	4
B	Telnet	7	1	11	0	3
C	Telnet	0	0	8	0	13
D	Telnet	9	0	21	12	2
E	Telnet	9	0	21	12	2
A	UART	7	1	10	0	4
B	UART	7	1	11	0	3
C	UART	0	0	8	0	13
D	UART	9	0	21	12	2
E	UART	9	0	21	12	2
X	仮想端末	7	1	11	0	3
Y	仮想端末	1	0	7	0	13
Z	仮想端末	9	0	21	12	2

スキャン

不特定多数の IP に対する，特定のポートへのコネクション

C&C 通信

特定 IP の特定のポートの組み合わせに対するコネクション

P2P 通信

bittorrent[64] のような P2P 通信サービスの名前解決

外部通信の特徴において，実機と仮想環境での差異はなかった．

5.3.6.3 解析環境の復元

マルウェア検体が作成したファイルやプロセス，およびマルウェア検体自身のバイナリに着目し，実機の復元に関する検証を行った．

reboot コマンドを用いた再起動による復元結果を表 5.10 に，プロセスの復元結果を表 5.11 にそれぞれ示す．機器 E において，reboot コマンドを用いた再起動によるファイルの復旧が行えない事例が 12 検体でみられた．11 検体は，マルウェアが実行ディレクトリに空ディレクトリおよび空ファイルを作成したものが，reboot コマンドを用いた再起動後にも残留したものであった．残り 1 検体は検体 85 であり，マルウェアが生成したファイルとプロセスの両方が機器の再起動後にも残留する，持続感染状態となった．これらの 12 検体について，電源の再起動および工場出荷状態へのリセットを行ったが，復旧は行えなかった．

表 5.10 reboot コマンドによる解析環境の復元

機器	コマンド実行方法	復元対象	再起動後のファイル改変の残留	再起動後のマルウェアのバイナリの残留
A	UART	12	0	0
B	UART	12	0	0
C	UART	19	0	0
D	UART	31	0	0
E	UART	37	12	1

表 5.11 reboot 後のマルウェアのプロセスの生存

機器	コマンド実行方法	復元対象	プロセスの残留あり	プロセスの残留なし
A	UART	12	0	12
B	UART	12	0	12
C	UART	19	0	19
D	UART	31	0	29
E	UART	37	1	34

ファームウェア更新機能を用いて同じバージョンのファームウェアに上書きをしたところ，復旧が可能であった．検体 85 のファームウェア更新機能は，機器の web 管理画面から更新を行う手法と，ネットワークブート機能を用いて機器の起動時に更新を行う方法の複数の手法が用意されていた．web 管理画面から更新を行う手法を用いた更新では復旧できなかったものの，ネットワークブート機能を用いた更新により復旧が可能であった．

5.3.7 考察

本実験では 87 のマルウェア検体を用い，5 種類の IoT 機器の実機および仮想環境上で動的解析実験を行った．多くのケースで実機と仮想環境で同様の振る舞いを観測できたものの，一部検体では解析環境により異なる振る舞いを観測した．

検体 85 では，機器 E で実行した場合のみ，電源を再起動した後もマルウェアのプロセスが残り続ける持続感染を生じた．5.3.6.1 項で示す通り，機器 E でのみ機器固有のディレクトリ “/flash” へのマルウェアのバイナリのコピー，およびコピーしたバイナリを実行するためのスクリプトファイルの生成を行った．機器 E は起動時に “/flash” 内の特定のディレクトリに格納されたスクリプトを実行する機能を有しており，この機能が悪用されたこと

により持続感染が生じた．機器 E は Linux 標準の起動スクリプトである rc スクリプトを模した起動スクリプト機能を “/flash” 内の特定のディレクトリに有しており，この機能が悪用されたことにより持続感染が生じた．標準の rc スクリプトの格納先は読み取り専用ファイルシステム内にあり保護されていることから，機器の何かしらの機能のために開発者が意図的に不揮発メモリ上に起動スクリプトの機能を用意したものと推測される．機器 E はファームウェア更新方法を複数提供しているが，一部の更新方法では復旧できないことが示された．検体 85 は仮想環境 Z ではバイナリのコピーやスクリプト生成は行っておらず，持続感染の挙動を解析することはできなかったといえる．

続いて，5.3.6.1 項において仮想環境を用いた解析時に異常を生じた検体 20,21,22 の 3 検体について分析を行った．これらの 3 検体は，仮想環境上で実行した直後に，機器をシャットダウンする挙動が見られた．実機を用いた Telnet による解析では，Telnet を切断する挙動が見られた．しかし，機器のシャットダウンは行われておらず，Telnet を再接続が可能であり，機器の内部状態も Telnet 切断前と同じであった．Telnet を再接続することで，解析を継続することが可能である．実機を用いた UART による解析では，通信の切断や機器をシャットダウンする挙動は見られず，解析を継続可能であった．このように，検体 20,21,22 の 3 検体は解析を阻害する振る舞いを行い，仮想環境上では解析が行えないものの，実機を用いた解析は可能であった．

これらの結果より，解析対象のマルウェア検体が感染する機器が分かっている場合は，感染する実機を用いた動的解析を検討するべきであると言える．

5.4　まとめと本章における課題

本研究では，IoT 機器の実機を用いたマルウェア動的解析を行うために最低限必要な機器の機能の分析を行い，実機を用いた動的解析方法を提案した．続いて，IoT 機器の実機および仮想環境を用いた動的解析に関して比較を行った．実証実験として 87 のマルウェア検体を用い，5 種類の IoT 機器の実機および仮想環境を用いた動的解析実験を行った．大半の機材・マルウェアの組み合わせにおいては，実機と仮想環境で同じ挙動を観測可能であったが，一部のマルウェアは，実機のみで観測可能な挙動を示した．3 検体は，仮想環境で実行した場合に機器のシャットダウンを行い，仮想環境を用いた解析を阻害する振る舞いを観測した．このように，実機に依存する挙動を示すマルウェアや，仮想環境での解析を阻害するマルウェアが実在し，これらの特性を持つマルウェアの解析において，実機を用いた解析環境が有用であることを示した．1 検体は，特定の実機上でのみ，機器の電源再起動後もマルウェアによるファイル改変やプロセスが残り続ける持続感染を確認した．本解析手法は CPU アーキテクチャに依存しない解析手法であり，今後新たに販売される製品や，実験で示した ARM，MIPS，MIPSEL 以外の CPU アーキテクチャ，たとえば今後増加が見込まれる RISC-V を用いた機器に対しても，5.2.1 節の条件を満たす場合に解析に利用可能である．

マルウェアの動的解析結果はマルウェア対策への活用が可能である．たとえば，スキャ

ン通信を観測することで，攻撃者の狙うサービスを把握することが可能であり，C&C サーバーとの通信を観測することで，ブラックリストとしての活用や，サーバーのテイクダウンに活用可能である．持続感染を引き起こす内部挙動を解析することにより，マルウェアの駆除方法の解明につながる．解析対象のマルウェア検体が感染する機器が分かっている場合は，実機を用いて解析することで，より正しい挙動を観測し，対策に活かすことが可能である．

一方で，環境依存性のマルウェアが多く登場すると，解析のために多くの機器を集めなければいけない問題が生じる．環境依存性のあるマルウェアを，異なる環境で簡易的に解析する手法の検討が，今後の課題である．

第 6 章
複合ディレクトリ型サンドボックスを用いたマルウェア動的解析手法の検証

Mirai に代表される IoT マルウェアの多くは，感染した機器の電源を再起動することで消滅することが知られている．しかしながら，特定の機器を狙った，電源の再起動のみでは消えない“持続感染”を引き起こす IoT マルウェアの事例も報告されている．マルウェアが持続感染機能を有するかどうか，およびどの機器に持続感染をし得るかどうかを効率的に解析する技術が必要とされている．

本章では，複合ディレクトリ型サンドボックスを用いた動的解析手法を提案する．持続感染を実現するためには，「機器上にマルウェアのバイナリを格納する」「機器のアプリケーション自動起動の仕組みを悪用する」の 2 つを不揮発に行う必要がある．機器のファイルシステムに何かしらの操作を行うことが考えられ，これらを観測することで持続感染の有無やそのメカニズムを解明できる可能性がある．IoT 機器のファイルシステムは機器によりカスタマイズされていることが多く，特定の機器のみに存在するディレクトリ構成を再現する必要がある．そこで，メーカーのホームページに公開されている IoT 機器向けファームウェア更新用ファイルを大量に収集し，抽出したファイルシステムの和集合となる構成を持つ複合ディレクトリ型サンドボックスを仮想環境上に構築した．続いて，実在するマルウェアを用いた評価を行い，特定の機器のみに持続感染するマルウェアを解析できることを示した．本解析手法により，持続感染のメカニズムの究明に加えて，潜在的に持続感染し得る機器をリストアップすることが可能であり，解析の大幅な効率化が見込まれる．

6.1 はじめに

近年，IoT 機器の普及が著しく，それらの機器を標的としたサイバー攻撃が増加している．PC の出荷台数は年間 3 億台前後である一方，IoT 機器は 2018 年の 307 億台から 2021 年には 448 億台への増加が予測されており，インターネットの主役となりつつある [1]．Mirai に代表される従来の IoT マルウェアは，感染した機器の電源を再起動することで消滅することが知られている．しかしながら，特定の機器を狙った，電源の再起動のみでは消えない“持続感染”を引き起こす IoT マルウェアの事例も報告されている．持続感染型マルウェアは長期間の活動が可能であることに加え，ボットネットを維持するための活発なスキャン活動が不要であり，ステルス性の高い感染拡大を実現可能である．持続感染型マルウェアである VPNFilter は，多くのセキュリティ技術者に気付かれることなく，50 万台の IoT 機器に感染した [14][15]．PC をターゲットとしたマルウェアの中には持続感染するものが知られているが，IoT 機器はその設計思想から，PC と同じ手順による持続感染が生じない

場合が多い．近年はIoT機器のOSとしてLinuxが用いられることが多いが，ファイルシステムの性質はIoT機器とPCで大きく異なる．IoT機器はユーザーによる機能追加が不要であることに加え，いきなり電源を切る使い方が多いことから，システムを保護するために読み取り専用ファイルシステムが用いられることが多い．そのため，IoT機器に持続感染を行うためには機器ごとの設計の違いを理解し，機器特有の機能を悪用する必要がある．このようなマルウェアは，汎用的な解析環境では解析できず，個別のIoT機器を再現した解析環境が必要であり，解析環境の構築が困難である．

マルウェアの解析手法は，主に静的解析と動的解析の2つに分類される．静的解析は，マルウェアのバイナリファイルをアセンブリコードにリバースエンジニアリングをすることにより，マルウェアを実行することなく解析する手法である．実行可能なすべての処理を解析可能な利点がある一方で，詳細な解析を行うためには熟練した技術が必要であり解析に時間を要する．特に特定の機器に依存するような複雑な処理を解析することは困難である．そのため，IoTマルウェアの解析に動的解析が多く用いられている．動的解析環境としては汎用的なディレクトリ構成を持つ解析環境が用いられることが多いが，持続感染のような機器特有の仕組みを悪用する振る舞いを観測することは困難である．マルウェアが機器に存在しないディレクトリの操作を試みた際に，痕跡が残らない問題がある．動的解析環境 “Firmadyne”[29] では，機器のファームウェアを仮想上に再現する試みが行われているものの，エミュレーションの成功率は10%以下と効率的ではない．一方で，約40%弱のファームウェア更新用ファイルからファイルシステムを抽出できることが示されており，比較的多くの機器のディレクトリ構成を知ることができることが示されている．本研究では，IoT機器のディレクトリ構成を模擬したサンドボックスを用いた解析を試みる．多数のIoT機器のディレクトリ構成の和集合を持つ，複合ディレクトリ型サンドボックスを用いた解析手法を提案する．実際のマルウェア検体の解析を行い，汎用的な解析環境では解析できない機器特有のディレクトリ構成に対する振る舞いを，本手法を用いて解析できることを示す．

本研究の貢献を以下に示す．

- 複合ディレクトリ型サンドボックスを用いた，マルウェア持続感染の実現方法を解析する手法を提案した．
- 提案手法により，機器特有のディレクトリ構成を悪用した持続感染の実現方法の解析を行えることを実証した．
- 複合ディレクトリ型サンドボックスを構成するファームウェア更新用ファイルを分析することにより，潜在的に持続感染が生じうる機器のリストアップを行えることを実証した．

6.2　マルウェアの動的解析環境

6.2.1　基本アイディア

4 章の分析より，マルウェアが持続感染を実現するには，下記 3 点をすべて満たす必要がある．

1. 機器上にマルウェアのバイナリファイルを保存する
2. 機器の電源再起動後にマルウェアのプロセスが自動起動するよう，機器の起動スクリプトを改変する
3. 上記 2 つの変更が，機器の再起動後も維持されるようにする

IoT 機器のルートファイルシステムは読み取り専用ファイルシステムが使用されていることが多く，保護された領域のファイル書き込みおよび改変は，電源の再起動時に復元される．機器によっては不揮発領域を持つ，もしくは機器独自の仕組みによりプログラムを自動起動する機能を有する場合があり，このような仕組みが持続感染に悪用されると考えられる．そのため，マルウェアが持続感染を試みる際に何かしらのディレクトリ操作またはファイル操作を観測できる可能性がある．しかし，これらのディレクトリ・ファイルが解析環境に存在しなければ，形跡が残らない可能性がある．

そこで，本研究では IoT 機器が持ちうるディレクトリをあらかじめ有するサンドボックスである“複合ディレクトリ型サンドボックス”を用いた動的解析を提案する．図 6.1 に複合ディレクトリ型サンドボックスの概念を示す．まず，多くの IoT 機器のファームウェア更新用ファイルを収集し，ディレクトリ構成を抽出する．続いて，これらの機器のディレクトリ構成の和集合となる構成を複合ディレクトリ型サンドボックス上に構築する．複数の機器で同一パスに内容の異なるファイルが存在する場合があるため，ファイルの再現は空ファイルを作成することによるファイルパスの再現までとした．

6.2.2　ファームウェア更新用ファイルの収集・展開

IoT 機器メーカーのホームページに公開されている IoT 機器向けファームウェア更新用ファイルをクローラーで収集し，3,199 ファイルを収集した．

続いて，これらのファームウェア更新用ファイルの展開によるファイルシステムの抽出を試みた．ファームウェア更新用ファイルを展開してファイルシステムを抽出するツールとして binwalk [65] が多く用いられる．しかし，binwalk の再帰展開機能は，展開に時間を要するうえに大量のストレージが必要である．さらには，無限ループに陥り展開が終わらない場合がある問題が指摘されている．Firmadyne [29] の Extractor は binwalk の API をラッピングして，展開するファイル種別や再帰展開時の階層およびファイル数を制限することにより，binwalk の再帰展開機能が持つ問題を解決している．しかし，これらの制限により，本来 binwalk で展開可能なファームウェア更新用ファイルを Firmadyne の Extractor

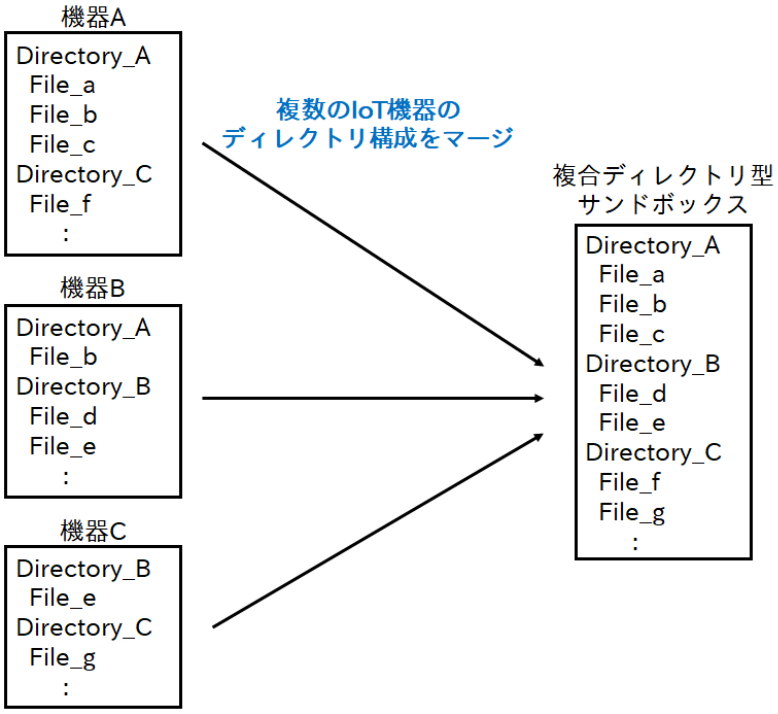

図 6.1 複合ディレクトリ型サンドボックスの概念

で展開できなくなる問題がある．本研究では Firmadyne の Extractor をベースに，下記の改良を加えてファームウェア更新用ファイルの展開に使用した．

展開するファイル種別の制限

Firmadyne では Linux を OS として使用する機器のみを対象としているため，Extractor は，Windows の実行ファイルを除外する目的で，拡張子が “exe” のファイルを展開対象から除外している．しかし，本研究で集めたファームウェア更新用ファイルの中には，Windows の自己解凍形式で配布されているものがあるが，これらのファームウェア更新用ファイルの拡張子も “exe” であるため，展開対象から除外されてしまう問題があった．そのため，拡張子が “exe” のファイルも展開を行うよう改良した．

再帰展開の階層とファイル数の制限

Firmadyne の Extractor は，時間短縮のために再帰展開の際の階層，および同一階層内で再帰展開するファイル数に制限を設けている．これらの条件を緩和し，より多くのファームウェア更新用ファイルの展開を行うよう改良した．

表 6.1 ファームウェア更新用ファイル データセット

Vendor	Wireless router	Network camera	Wired router	Switch	NAS	NVR	Wireless adapter	HDD	Printer server	Card Reader Writer	TV tuner	未分類	SUM
A	83 (6)	0 (0)	0 (0)	25 (6)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	108 (12)
B	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
C	170 (12)	0 (0)	69 (7)	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	60 (0)	300 (20)
D	0 (0)	28 (26)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	25 (0)	53 (26)
E	17 (6)	0 (0)	1 (1)	21 (9)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	40 (17)
F	88 (35)	4 (2)	2 (2)	13 (6)	1 (1)	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)	16 (5)	125 (52)
G	54 (20)	50 (9)	8 (3)	19 (5)	14 (5)	0 (0)	2 (2)	13 (3)	0 (0)	0 (0)	2 (1)	94 (33)	256 (81)
H	0 (0)	0 (0)	18 (7)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	10 (6)	28 (13)
I	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	22 (3)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	2 (1)	24 (4)
J	81 (37)	33 (13)	21 (14)	0 (0)	12 (5)	0 (0)	4 (3)	0 (0)	2 (2)	0 (0)	0 (0)	13 (7)	166 (81)
K	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
L	2 (1)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	3 (2)
SUM	495 (117)	115 (50)	119 (34)	78 (26)	27 (11)	22 (3)	6 (5)	14 (4)	2 (2)	1 (1)	2 (1)	222 (54)	1,103 (308)

() 内は製品数を示す

表 6.2 複合ディレクトリサンドボックスの第一階層ディレクトリ一覧

/.bcm-debug-info	/.svn	/00_2015-01-13	/NFW	/PreOSGi
/TM	/accsh	/addon	/app	/appfs
/apps	/atheros	/auth	**/bin**	**/boot**
/cgi	/cgroup	/conf	/dalvikvm	/data
/dev	/dev.tmp	/dni_boot	/dpi_backup	/drivers
/dump_sadb	**/etc**	/etc.tmp	/etc_ro	/ffmpeg
/flash	/flash2	/fonts	/framework	/gm
/home	/htdocs	/html	/https	/image
/include	/jffs	/lang	**/lib**	**/lib32**
/lib64	/libexec	/log	/lost+found	/man
/media	**/mnt**	/mnt_flash	/mnt_flash_ex	/mnth
/mntlog	/model	/modsqfs	/mtd	/mtdrw
/mydlink	/nashttpd	/nfwdir	/nvdata	**/opt**
/overlay	/preinstall	/proc	/ram	/ramfs
/rboot	/recovery	/rl3	/ro	/rom
/root	/rw	**/sbin**	/selinux	/shadow
/share	/skel	/splashbin	/sqfs	**/srv**
/sslvpn	/storage	/sys	/system	/test
/tm_key	/tm_pattern	**/tmp**	/tmp_orig	/tts
/udev	/userapps	/userdata	/usr	/var
/var.radius	/vc03	/wa_www	/web	/webroot
/wifi	/www	/www-ms	/xbin	

ファームウェア更新用ファイルの展開の結果，表 6.1 に示す 308 製品・1,103 ファイルからファイルシステムの抽出に成功した．これらのファームウェア更新用ファイルから，全 8,289 ディレクトリ，80,838 ファイルを抽出した．抽出したディレクトリ構成のうち，第一階層の 109 ディレクトリを表 6.2 に示す．太字の 15 ディレクトリは Linux 標準のディレクトリ構成 [63] で定められたものである．第一階層のディレクトリだけを見ても，IoT 機器のディレクトリ構成の多彩さがよく示される結果となった．

表 6.3 仮想環境一覧

仮想環境	CPU アーキテクチャ	CPU クロック	RAM (MB)	Linux カーネル
x	MIPSEL	200	256	3.2.51-1
y	ARM	200	256	3.2.51-1
z	MIPS	200	256	3.2.51-1

6.3 実証実験

6.3.1 仮想環境の準備

表 6.3 に示す通り，CPU アーキテクチャごとにそれぞれ QEMU を用いた仮想環境 x,y,z を用意した．

複合ディレクトリ型サンドボックスを用いた動的解析環境を図 6.2 に示す．

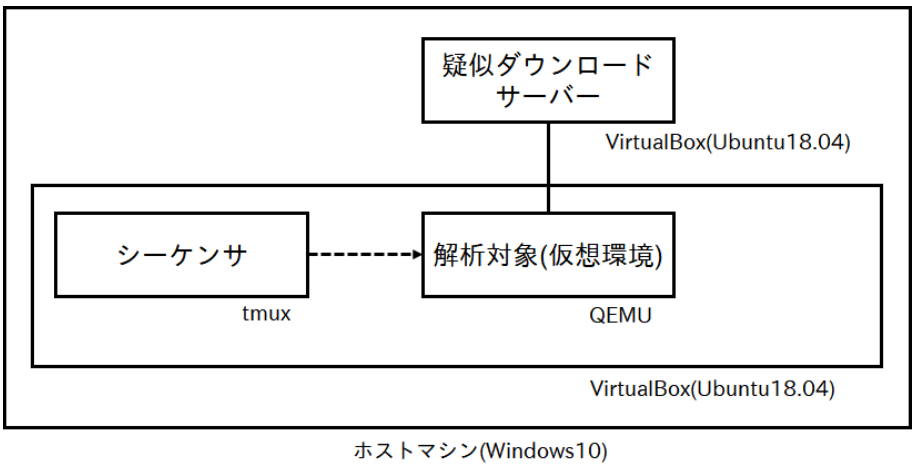

図 6.2 複合ディレクトリ型仮想動的解析環境

解析対象（仮想環境）

解析対象のマルウェアがターゲットとする CPU アーキテクチャに応じて，仮想環境 x,y,z を使用する．仮想環境 x,y,z はベース環境として debian の QEMU 向けイメージを使用し，6.2.2 節で収集した 308 製品・1,103 ファイルのディレクトリ構成をあらかじめ再現しておく．“/proc” ディレクトリはシステムの状態を示すファイルが動的に生成される領域のため，ディレクトリ構成の再現の対象外とする．

シーケンサ

6.3.3 節に示す手順 1 から手順 4 における一連の OS コマンドを自動実行する．また，

実行したコマンドおよびコマンド実行に伴う出力をログとして保存する．tmux[59]とシェルスクリプトにより実現する．

疑似ダウンロードサーバー

本実験で使用するマルウェア検体をダウンロード可能な web サーバーを仮想マシン上に構築する．web サーバーとして Apache2[60] を使用する．

マルウェア検体が外部へ攻撃活動を行う可能性を考慮し，本実験環境はインターネットとは接続しないものとした．

6.3.2 実験用検体の準備

実験に使用した検体は，主にハニーポットを利用して準備を行った．まず，ハニーポットを用いて攻撃者がマルウェアをダウンロードするために実行するシェルコマンドを観測する．その後，別マシンでこのコマンドを再現し攻撃者のダウンロードサーバーからマルウェアをダウンロードした．この手法で 83 検体を用意した．加えて，特定の機器に持続感染する可能性が指摘されているマルウェアを 4 つ用意した．

使用した検体を，Mipsel[5],Mips[5],Arm[4] の 3 つの CPU アーキテクチャ毎に，それぞれ表 B.5，表 B.6，表 B.7，表 B.8 に示す．マルウェアのファミリ名として，マルウェア解析 Web サイト「VirusTotal」[50] で検索した結果を併記する．検体 1〜42,44〜84 は 2016 年 12 月から 2019 年 1 月にかけて我々が運用する IoT ハニーポットで取得したものである．検体 43,86,87 は特定の機器に持続感染する可能性のある VPNFilter の 1st stage であり，文献 [15] にハッシュ値が記載されている検体を，2020 年 2 月に VirusTotal から取得した．検体 85 は特定の機器に持続感染するマルウェアであり，2018 年 10 月に取得した．

6.3.3 実験手順

実験手順を以下に示す．

1. **検体実行前の状態の記録**
   解析対象機器の OS コマンドを実行することにより，機器内部のファイルシステム (全ファイル名，ファイルサイズ，タイムスタンプ) とプロセスリストを感染前の状態として記録する．
2. **検体の実行**
   解析対象機器の OS コマンドを実行することにより，事前に入手したマルウェア検体のバイナリファイルを，疑似ダウンロードサーバーからダウンロードし実行する．
3. **感染の確認**
   マルウェア実行後，解析対象機器の OS コマンドを実行しプロセスの増加の有無を確認する．また，通信データロガーで解析対象機器と外部との通信の有無を確認する．

表 6.4 マルウェア検体実行後の機器のファイルの差異

仮想環境	ファイル差異あり	ファイル差異なし	検証不可
x	0	19	3
y	1	20	0
z	14	30	0

プロセスの増加や攻撃者の C&C サーバーに対する通信を開始するなどにより機器がマルウェア感染状態になったことを確認する．

4. **検体実行後の状態の記録**
   マルウェア実行の 5 分後に，再度解析対象機器の OS コマンドを実行することにより，機器内部のファイルシステム (全ファイル名，ファイルサイズ，タイムスタンプ) とプロセスリストを感染後の状態として記録する．

5. **マルウェア検体の内部挙動の抽出**
   最後に手順 1 および手順 4 の記録を比較し，差分をマルウェアの内部挙動として抽出する．

6.3.4 実験結果

マルウェア検体実行前後での機器のディレクトリおよびファイルの差異を表 6.4 に示す．

検体 20,21,22 の 3 検体で検体実行直後にターミナルにシャットダウンを示すメッセージが表示された後，エミュレーションが終了した．これらの異常が発生したケースでは内部挙動の解析を正しく行えないため，解析結果は「検証不可」として示す．検体 74〜84 の 11 検体では iptables に関連するファイルの改変が見られた．iptables はネットワークのパケットフィルタリング機能であり，持続感染に悪用されたとは考えにくい．検体 43,85,86,87 ではそれぞれ特徴的なファイル操作を検出したため，個別に詳細に記載する．

検体 43

検体 43 は ARM をターゲットとしたマルウェア VPNFilter であり，特定の機器に持続感染する可能性が指摘されている．提案手法による解析の結果，以下の 5 ファイルの生成を確認した．

- /etc/config/crontab
- /run/client_ca.crt
- /run/client.crt

- /run/client.key
- /run/msvf.pid

図 6.3 /etc/config/crontab

```
*/5 * * * * ./f3309f143cca7954dba2887aadd2b4cb
```

特に “/etc/config/crontab” は Linux のタスクスケジューラー crontab の設定と考えられ，持続感染に悪用された可能性がある．ファイル “/etc/config/crontab” の処理は図 6.3 に示すとおり，ホームディレクトリにあるマルウェアのバイナリファイルを 5 分おきに起動するためのスケジューラーの設定であった．crontab の設定ファイルは通常は “/etc/crontab” に保存されるため，“/etc/config/crontab” は機器独自の仕様であると考えられる [47]．ディレクトリ “/etc/config” が存在しない環境では，ファイル “/etc/config/crontab” は生成されなかった．複合ディレクトリ型サンドボックスの元となるファームウェア更新用ファイルのうち，“/etc/config/” を持つものは 49 製品 360 ファイルであり，全体の 15.9%の製品である．これらの機器に潜在的に持続感染する可能性を持つ．ただし，実際に持続感染し得るかは，該当機器のファイルシステムの種類，および crontab の設定ファイルのパスに関する機器の仕様次第である．

“/run/client_ca.crt”，“/run/client.crt”，“/run/client.key” はファイル名からクライアント認証に用いられるものと推測される．C&C サーバーとの通信に用いられる用途が考えられる．

検体 85

検体 85 は MIPS をターゲットとしたマルウェア Hajime であり，特定の機器に持続感染することが知られている [13]．提案手法による解析の結果，以下の 3 ファイルの生成を確認した．

- /flash/bin/.telnetd
- /flash/etc/rc.d/run.d/S99telnetd
- /flash/bin/.p/atk.mipseb

ファイル “/flash/bin/.telnetd” はマルウェアのバイナリファイルのコピーである．ファイル “/flash/etc/rc.d/run.d/S99telnetd” の処理は図 6.4 に示すとおりマルウェアのバイナリファイル “/flash/bin/.telnetd” を起動する bash スクリプトであった．Linux の起動スクリプトを格納するディレクトリは通常は “/etc/rc.d/” であるが，本検体は “/flash/etc/rc.d/” を起動スクリプト格納ディレクトリとして扱う機器を対象に持続感染を狙ったものと考え

られる．ディレクトリ “/flash” が存在しない環境では，これらのファイルは生成されなかった．このマルウェアがターゲットとする機器を用いて実証実験を行った結果，本検体が持続感染を行うことが示された．

図 6.4 /flash/etc/rc.d/run.d/S99telnetd

```
#!/bin/bash
cd /flash/bin
./.telnetd
```

複合ディレクトリ型サンドボックスの元となるファームウェア更新用ファイルのうち，“/flash/” を持つものは 12 製品 31 ファイルであり，全体の 3.9%の製品である．これらの機器に潜在的に持続感染する可能性を持つ．ただし，実際に持続感染し得るかは，該当機器のファイルシステムの種類，および機器が “/flash/etc/rc.d/” を起動スクリプト格納ディレクトリとして扱うかどうかの機器の仕様次第である．

検体 86，87

検体 86，87 は MIPS をターゲットとしたマルウェア VPNFilter であり，特定の機器に持続感染する可能性が指摘されている．提案手法による解析の結果，以下の 3 ファイルの生成を確認した．

- /tmp/client_ca.crt
- /tmp/client.crt
- /tmp/client.key

“/tmp/client_ca.crt”，“/tmp/client.crt”，“/tmp/client.key” はファイル名からクライアント認証に用いられるものと推測される．C&C サーバーとの通信に用いられる用途が考えられる．これらのファイルが生成されたディレクトリ “/tmp” は Linux 標準のディレクトリ構成 [63] でも規定されており，多くの機器で挙動を観測できると考えられる．5 章の動的解析環境では実機と仮想環境ともに同様にこれらのファイルが生成されることを確認した．本検体では，持続感染が疑われるようなファイル操作は観測されなかった．

6.3.5 考察

複合ディレクトリ型サンドボックスを用いて実際のマルウェアの動的解析を行った結果，87 検体中 15 検体でファイルの改変を確認した．特に持続感染の可能性が指摘される 4 検体のうち，2 検体で持続感染の振る舞いを解析可能であった．これらの 2 検体は，いずれも Linux 標準ではない，特定の機器にのみ存在するディレクトリに対するファイル生成を行うことにより，持続感染を実現していた．対象となるディレクトリが存在しない環境で

は，これらのファイル生成は観測できなかった．実証実験の結果，提案手法が持続感染型マルウェアの解析に有効であることが示された．

続いて，持続感染の振る舞いを観測した 2 検体がターゲットとしたディレクトリについて，複合ディレクトリ型サンドボックスを構成するファームウェア更新用ファイルのデータセットの分析を行った．検体 43 の対象ディレクトリ “/etc/config/” を持つものは，データセット全体の 15.9%の製品にあたる 49 製品 360 個のファームウェア更新用ファイルで確認された．検体 85 の対象ディレクトリ “/flash/” を持つものは，データセット全体の全体の 3.9%の製品にあたる，2 製品 31 個のファームウェア更新用ファイルで確認された．複合ディレクトリ型サンドボックスを構成するファームウェア更新用ファイルのデータセットから，対象ディレクトリを持つ機器をリストアップすることで，潜在的に持続感染をする可能性のある機器を抽出可能であることを示した．

6.4 まとめと本章における課題

本研究では，複合ディレクトリ型サンドボックスを用いた動的解析環境を提案した．実際のマルウェア検体を用いた検証を行い，Linux 標準ではない，特定の機器にのみ存在するディレクトリに対するマルウェアのアクションを検出可能であることを示した．これにより，特定の機器のディレクトリ構成を悪用して持続感染するマルウェアの検出と，持続感染のメカニズムおよび潜在的に持続感染しうる機器のリストアップを行うことができた．本解析手法は，QEMU のサポートする CPU アーキテクチャであれば，実験で示した ARM，MIPS，MIPSEL 以外の CPU アーキテクチャ，たとえば今後増加が見込まれる RISC-V に関しても，複合ディレクトリ型サンドボックスとして構築可能である．また，新しい機器のディレクトリ構成を複合ディレクトリ型サンドボックスに組み入れていくことで，継続的に発展可能である．

本手法は前提としてあらかじめ多くの機器のファームウェア更新用ファイルを展開してディレクトリ構成を知る必要がある．より多くの機器のファイルシステムの構成を再現したサンドボックスを構築するためには，収集するファームウェア更新用ファイルを増やすことと，ファームウェア更新用ファイルの展開技術を向上して展開可能なファームウェアを増やす必要がある．ファームウェア更新用ファイルは解析防止のための難読化が行われている場合も多いため，ファームウェア更新用ファイルの展開だけでなく，リモートログイン可能な機器から動的にディレクトリ構成を収集するような別のアプローチも必要である．また，本提案手法は仮想環境を用いており，仮想環境を検知して振る舞いを変えるマルウェアを正しく解析できない問題が考えられる．開発者向けに販売されている CPU の評価基板を活用した，実機版複合ディレクトリ型サンドボックスの検討が必要である．

第 7 章
総括

本研究では，持続感染型マルウェアの対策への貢献を目的として，IoT 機器に対するマルウェア持続感染の概念のモデル化，および解析手法の提案を行った．

4 章では，持続感染の実現手法に関する概念のモデル化を行った．IoT 機器の特性や機能に着目し，マルウェア持続感染の成立条件を分析し，持続感染を実現可能な攻撃手法として，“システム設定改変攻撃”と“ファームウェア不正書換攻撃”の 2 つを定義した．市販の IoT 機器を用いた概念実証を行い，これらの攻撃手法が実際に通用することを示した．これらの実証実験をもとに，持続感染防止に効果的なファームウェア構成について議論を行った．

続いて，マルウェアが持つ持続感染の性質を解析する手法として，IoT 機器の実機を用いた動的解析手法，および複合ディレクトリ型サンドボックスを用いた動的解析手法の 2 つを提案し，実証実験を行った．5 章では，IoT 機器の実機を用いた動的解析手法を提案した．持続感染型マルウェアは特定の実機上でのみ持続感染の振る舞いを示すことが多く，このようなマルウェアを正しく解析して持続感染の実現手法を明らかにできることを示した．6 章では，複合ディレクトリ型サンドボックスを用いた動的解析手法を提案した．インターネット上から収集した多数の IoT 機器のファームウェア更新用ファイルからディレクトリ構成を抽出し，和集合のディレクトリ構成を仮想環境上に再現した複合ディレクトリ型サンドボックスを構築した．本手法により，特定の機器に対してのみ持続感染の振る舞いを示すマルウェアを解析することが可能であった．加えて，複合ディレクトリ型サンドボックスを構成するファームウェア更新用ファイルを分析することにより，潜在的に持続感染が生じうる機器のリストアップを行えることを実証した．これらの解析手法は新しい機器・マルウェアに対しても適用可能であり，また持続感染型マルウェアの解析にとどまらず，IoT 機器をターゲットにしたマルウェアの解析全般に応用可能である．CPU アーキテクチャに依存しない解析手法であり，今後増加が見込まれる RISC-V に対しても有用である．

今後の課題は，実機を用いた複合ディレクトリ型サンドボックスの構築である．仮想環境を用いた複合ディレクトリ型サンドボックスでは，仮想環境検知型マルウェアの解析を行うことができない．市販の IoT 機器では仮想環境検知型のマルウェアを解析できるものの，リソース制限やファイルシステムの読み取り専用の制限のために複合ディレクトリ型サンドボックスの構築を行えない．開発者向けに CPU の評価基板が販売されている．市販の IoT 機器と比較してリソースを潤沢に搭載しており，またファイルシステムの構築を自由に行うことが可能である．このような CPU の評価基板を用いて，実機版の複合ディレクトリ型サンドボックスの構築が実現可能であると考える．

謝辞

本論文を執筆するにあたり，ご支援頂いた全ての皆様に深く感謝致します．特に，本研究を進めるにあたり，多大なご指導とご助言を頂きました，横浜国立大学大学院環境情報研究院 松本勉 教授，吉岡克成 准教授に深く感謝申し上げます．

加えて，いつも本研究に関する活発な意見を頂いた，四方順司 教授，田辺瑠偉 特任助教，吉田直樹 特任助教，藤田彬 連携研究員に深く感謝致します．本論文および公聴会発表資料の査読をいただき，大変有意義なご意見をいただいた森辰則 教授，白川真一 講師に深く感謝致します．

また，ミーティングでご協力，ご意見を頂いた ハニーポットユニット，IoT マルウェアユニットの皆様に感謝いたします．毎日の研究室でお世話になりました秘書の成松美央氏，技術補佐員の石舘知子氏，高山宏明氏をはじめ，吉岡研究室，松本研究室，四方研究室の皆様に感謝の意を表します．博士論文を書き上げられたことに対して，これまでお世話になったすべての方々に改めて謝意を表します．

私に研究の機会を与えてくださり，大学院通学を支援して頂きました，富士ソフト株式会社 渡辺露文 部長をはじめセキュリティマネジメント部の皆様に深く感謝致します．

最後に，研究活動を支えてくれた妻と子供，妻の実家の家族と私の両親に深く感謝致します．

参考文献

[1] 総務省，“令和元年版情報通信白書”(2019) .

[2] “NICTER 観測レポート 2018”，https://www.nict.go.jp/cyber/report/NICTER_report_2018.pdf,（参照 2019-4-22）.

[3] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti, “A Large-Scale Analysis of the Security of Embedded Firmwares”, 23rd USENIX Security Symposium(2014).

[4] “ARM”, https://www.arm.com/ja/,（参照 2019-4-30）.

[5] “MIPS Processors – Imagination Technologies”, https://imgtec.com/mips,（参照 2019-4-30）.

[6] “Linux Foundation”, https://www.linuxfoundation.org/,（参照 2018-5-28）.

[7] “ご家庭で Wi-Fi ルーターをより安全にお使い頂くために”，https://dlpa.jp/pdf/dlpa_pr_20191218.pdf,（参照 2020-6-5）.

[8] Marta Janus, “Heads of the Hydra. Malware for Network Devices”, https://securelist.com/heads-of-the-hydra-malware-for-network-devices/36396/,（参照 2020-3-31）.

[9] Kishore Angrishi, “Turning Internet of Things (IoT) into Internet of Vulnerabilities (IoV): IoT Botnets”, arXiv:1702.03681, 13 Feb. 2017, https://arxiv.org/pdf/1702.03681.pdf,（参照 2020-3-31）.

[10] Muhammad Junaid Bohio, “Analysis of a MIPS Malware”, Information Security Reading Room, 2015, https://www.sans.org/reading-room/whitepapers/malicious/analyzing-backdoor-bot-mips-platform-35902,（参照 2020-3-31）.

[11] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan. K. Thomas, Y. Zhou, “Understanding the Mirai Botnet”, 26th USENIX Security Symposium (2017).

[12] Sam Edwards, “Hajime: Analysis of a decentralized internet worm for IoT devices”, Rapidity Networks. Oct. 2016, http://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf,（参照 2020-3-31）.

[13] “電源を切っても消えない持続感染型 IoT マルウェアについて”，http://ipsr.ynu.ac.jp/iotmalware/index.html,（参照 2019-6-26）.

[14] “New VPNFilter malware targets at least 500K networking devices worldwide”, https://blog.talosintelligence.com/2018/05/VPNFilter.html,（参照 2018-5-26）.

[15] “VPNFilter Update - VPNFilter exploits endpoints, targets new devices”, https://blog.talosintelligence.com/2018/06/vpnfilter-update.html,（参照 2020-2-13）.

[16] Harlan Carvey, “The Windows Registry as a Forensic Resource”, Digital Investigation Volume 2, Issue 3, September 2005, http://www.campus64.com/digital_learning/data/windows_forensics/registry/registry_forensics.pdf,（参照 2020-3-31）.

[17] Patrick Wardle, “Methods of malware persistence on Mac OS X”, Proceedings of the Virus Bulletin Conference (2014), https://pdfs.semanticscholar.org/d093/387297c46493c1f14b18b9fee8911b803e21.pdf,（参照 2020-3-31）.

[18] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, Davide Balzarotti, “Understanding Linux Malware”, 2018 IEEE Symposium on Security and Privacy(2018).

[19] “SQUASHFS 4.0 FILESYSTEM”, https://www.kernel.org/doc/Documentation/filesystems/squashfs.txt,（参照 2020-4-7）.

[20] “Cramfs - cram a filesystem onto a small ROM”, https://www.kernel.org/doc/Documentation/filesystems/cramfs.txt,（参照 2020-4-7）.

[21] “Cuckoo Sandbox - Automated Malware Analysis”, https://cuckoosandbox.org/,（参照 2019-6-13）.

[22] “QEMU”, https://www.qemu.org/,（参照 2019-6-13）.

[23] “Detux”, https://github.com/detuxsandbox/detux,（参照 2020-3-31）.

[24] Andrei Costin, Jonas Zaddach, “IoT Malware: Comprehensive Survey, Analysis Framework and Case Studies”, Blackhat USA(2018).

[25] “Buildroot - Making Embedded Linux Easy”, https://buildroot.org/,（参照 2019-6-13）.

[26] Yin Minn Pa Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, C. Rossow, “IoTPOT: Analysing the Rise of IoT Compromises”, USENIX/WOOT’15，2015.

[27] “OpenWrt Project”, https://openwrt.org/,（参照 2020-2-9）.

[28] Jonas Zaddach, Luca Bruno, Aurelien Francillon and Davide Balzarotti, “Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems’ Firmwares”, The Network and Distributed System Security Symposium(2014).

[29] Daming D. Chen, Manuel Egele, Maverick Woo, and David Brumley, “Towards Automated Dynamic Analysis for Linux-based Embedded Firmware.”, Network and Distributed System Security Symposium 2016.

[30] Tran Nghi Phu, Kien Hoang Dang, Dung Ngo Quoc, Nguyen Tho Dai, and Nguyen Ngoc Binh, “A Novel Framework to Classify Malware in MIPS Architecture-Based IoT Devices”, Hindawi Security and Communication Networks(2019).

[31] “Oracle VM VirtualBox”, https://www.virtualbox.org/,（参照 2020-4-7）.

[32] “VMware - Official Site”, https://www.vmware.com/,（参照 2020-4-7）.

[33] “Paloaltonetworks: New IoT/Linux Malware Targets DVRs, Forms Botnet”, https://unit42.paloaltonetworks.com/unit42-new-iotlinux-malware-targets-dvrs-forms-botnet/,（参照 2019-6-15）.

[34] 大山 恵弘, “Raspberry Pi 環境のためのサンドボックス検出ツール”, 日本ソフトウェア科学会第 35 回大会 (2018).

[35] “raspberrypi.org”, https://www.raspberrypi.org/,（参照 2020-2-23）.

[36] “IoT Malware Activity Already More Than Doubled 2016 Numbers”, https://threatpost.com/iot-malware-activity-already-more-than-doubled-2016-numbers/126350/,（参照 2017-10-30）.

[37] 独立行政法人 情報処理推進機構 セキュリティセンター , “情報セキュリティ 10 大脅威 2018” , 2018.

[38] “JVNTA#95530271 Mirai 等のマルウェアで構築されたボットネットによる DDoS 攻撃の脅威”, https://jvn.jp/ta/JVNTA95530271/,（参照 2017-7-24）.

[39] Cui Ang, Stolfo Salvatore, “A Quantitative Analysis of the Insecurity of Embedded Network Devices: Results of a Wide–Area Scan”, 26th Annual Computer Security Applications Conference: Proceedings: Austin, Texas, USA: 6-10 December(2010).

[40] 中里 純二, 島村 隼平, 衛藤 将史, 井上 大介, 中尾 康二, “nicter によるネットワーク観測および分析レポート～組み込みシステムに感染するマルウエア～”, 電子情報通信学会信学技報 vol．113 no．135，ISEC2013-48，pp．1-5，2013.

[41] “Security issues within the IoT: The problem with DDoS – トレンドマイクロセキュリティブログ”, http://blog.trendmicro.com/security-issues-within-iot-problem-ddos/,（参照 2016-12-11）.

[42] “SYNful Knock – A Cisco router implant – Part I”, https://www.fireeye.com/blog/threat-research/2015/09/synful_knock_-_acis.html,（参照 2017-7-11）.

[43] “IoT 機器を「使用不能」にするマルウェア,「BrickerBot」”, http://blog.trendmicro.co.jp/archives/14757,（参照 2017-7-24）.

[44] 笠間 貴弘, 井上 大介, “大規模ダークネット観測と能動的スキャンによるマルウェア感染 IoT 機器の分類”, 情報処理学会論文誌 58 巻 9 号 pp 1388–1398, 2017.

[45] “How To Configure a Linux Service to Start Automatically After a Crash or Reboot – Part 2: Reference”, https://www.digitalocean.com/community/tutorials/how-to-configure-a-linux-service-to-start-automatically-after-a-crash-or-reboot-part-2-reference,（参照 2018-5-18）.

[46] “Bash Startup Files (Bash Reference Manual)”, https://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html,（参照 2020-4-17）.

[47] “Man page of CRON”, https://linuxjm.osdn.jp/html/cron/man8/cron.8.html,（参照 2020-4-12）.

[48] 鈴木将吾，インミンパパ，江澤優太，鉄穎，中山颯，吉岡克成，松本勉，“組込み機器への攻撃を観測するハニーポット IoTPOT の機能拡張”，電子情報通信学会信学技報 vol．115 no．488，ICSS2015–47，pp．1–6，2016.

[49] 中山 颯, 鉄 穎, 楊 笛, 田宮 和樹, 吉岡 克成, 松本 勉, “IoT 機器への Telnet を用いたサイバー攻撃の分析”, 情報処理学会コンピュータセキュリティシンポジウム 2016, セッション 3E1,2016.

[50] “VirusTotal”, https://www.virustotal.com/ja/,（参照 2019-4-22）.

[51] “SuperH RISC engine ファミリ – RENESAS”, https://www.renesas.com/ja-jp/products/microcontrollers-microprocessors/superh.html,（参照 2016-12-11）.

[52] ISO, “ISO / IEC TS 29167-15 Information technology -Automatic identifidation and data capture techniques – Part 15: Crypto suite XOR security services for air interface comunications”,2017.

[53] “The Menlo Report (August 2012)”, https://www.dhs.gov/sites/default/files/publications/CSD-MenloPrinciplesCORE-20120803_1.pdf,（参照 2019-1-11）.

[54] BUSYBOX, https://busybox.net/,（参照 2019-4-30）.

[55] “TELNET PROTOCOL SPECIFICATION”, https://tools.ietf.org/html/rfc854,（参照 2019-5-24）.

[56] “universal asynchronous receiver transmitter (UART)”, https://www.jedec.org/standards-documents/dictionary/terms/universal-asynchronous-receiver-transmitter-uart,（参照 2019-6-11）.

[57] “BASHLITE Family Of Malware Infects 1 Million IoT Devices”, https://threatpost.com/bashlite-family-of-malware-infects-1-million-iot-devices/120230/,（参照 2019-5-16）.

[58] “Tera Term Home Page”, https://ttssh2.osdn.jp/,（参照 2019-10-13）.

[59] “tmux”, https://github.com/tmux/tmux,（参照 2020-2-9）.

[60] “The Apache HTTP Server Project”, https://httpd.apache.org/,（参照 2019-10-21）.

[61] “Wireshark”, https://www.wireshark.org/,（参照 2019-10-21）.

[62] “TCPDUMP/LIBPCAP public repository”, https://www.tcpdump.org/,（参照 2020-2-9）.

[63] “Filesystem Hierarchy Standard”, http://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf,（参照 2019-10-13）.

[64] “The Basics of BitTorrent”, http://help.bittorrent.com/customer/portal/articles/178790-the-basics-of-bittorrent,（参照 2019-10-21）.

[65] “GitHub - ReFirmLabs/binwalk: Firmware Analysis Tool”, https://github.com/ReFirmLabs/binwalk,（参照 2020-4-7）.

公表論文リスト

査読付き論文誌

（主論文）

1. 原 悟史，田宮 和樹，鉄 穎，渡辺 露文，吉岡 克成，松本 勉，“感染持続型 IoT マルウェアの実態調査と実機による概念実証”，電子情報通信学会論文誌，Volume J102-B No.8 pp.524-535，Aug. 2019

2. 原 悟史，熊 佳，玉井 達也，田宮 和樹，田辺 瑠偉，藤田 彬，吉岡 克成，松本 勉，“IoT 機器の実機を用いたマルウェア動的解析手法の検証”，電子情報通信学会論文誌，Volume J103-B No.8 (掲載予定)，Aug. 2020

（その他関係論文）

3. Chun-Jung Wu, Ying Tie, Satoshi Hara, Kazuki Tamiya, Akira Fujita, Katsunari Yoshioka, Tsutomu Matsumoto, “IoTProtect: Highly Deployable Whitelist-based Protection for Low-cost Internet-of-Things Devices”, Journal of Information Processing 26(0), 662-672, 2018

査読付き国際会議ポスター発表

1. Satoshi Hara, Jia Xiong, Tatsuya Tamai, Kazuki Tamiya, Rui Tanabe, Akira Fujita, Katsunari Yoshioka, Tsutomu Matsumoto, “Verification of malware dynamic analysis method using real and virtual IoT devices”, The 14th International Workshop on Security(IWSEC), 2019

国内学会発表

1. 原 悟史，渡辺 露文，田宮 和樹，吉岡 克成，松本 勉， “IoT マルウェアの持続的感染の成立要因の分析と実機による検証，情報処理学会コンピューターセキュリティシンポジウム 2017, セッション 2A4-2, 論文集, PP795-799.（CSS2017 奨励賞）

2. 熊 佳，玉井 達也，田宮 和樹，原 悟史，田辺 瑠偉，藤田 彬，吉岡 克成，松本 勉，“IoT 機器を用いたマルウェア動的解析環境の構築方法の検討”, 電子情報通信学会暗号と情報セキュリティシンポジウム 2019, 論文集, PP7-12.

3. 森下 瞬，小川 航汰，原 悟史，田辺 瑠偉，吉岡 克成，松本 勉，“広域スキャンとダークネット観測に基づく IoT マルウェア感染状況の分析”, 信学技報, vol. 119, no. 437, ICSS2019-81, pp. 79-84, 2020 年 3 月.

付録

A　4 章の実験に用いたマルウェア検体一覧

4.4 節の実験に使用したマルウェア検体の一覧を表 A.1，表 A.2，表 A.3，表 A.4 に示す．

表 A.1 マルウェア検体 (ARM)

検体	md5hash 値	trend micro	Avg	Kaspersky
1	06ffde55395f4508c2a8edd6fb8c2a83	ELF_BASHLITE.SM	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
2	1e1a9668e86dfec845893968760dcdb6	ELF_BASHLITE.DHD	a variant of Linux/ Gafgyt.MT	Linux/Fgt
3	4493a098787ef436308ef1ed571a6ab9	ELF_BASHLITE.DIB	a variant of Linux/ Gafgyt.SR	Linux/Fgt
4	a021d25c75f2a0dfa6a1103ac812c3e1	ELF_BASHLITE.DIA	a variant of Linux/ Gafgyt.QE	Linux/Fgt.BP
5	ac8733aa6c973564643236ee2535083b	検知されず	a variant of Linux/ Gafgyt.C	Linux/Fgt.CE
6	1a3480286d6f6b4e7fe42fa11ee1122d	検知されず	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
7	c7c3a9c7a73c9f8dcd160a86be65740b	検知されず	a variant of Linux/ Mirai.A	Linux/Fgt.CI
8	59e59d762e1eb8838d33c4e133c6a546	検知されず	検知されず	Linux/Fgt.CI
9	238a67e6f9b129680b618a3c579a8c6c	ELF_MIRAI.A	a variant of Linux/ Mirai.A	Linux/Fgt.CI
10	8d5419f45c1074713325a5f414bcb347	検知されず	a variant of Linux/ Tsunami.NAL	Linux/Tsunami.BK
11	ae68fa96792645d7e8ef40031330f4f3	ELF_SONEX.SMA	Linux/Dnsamp.B	Linux/MrBlack.A

表 A.2 マルウェア検体 (MIPS)

検体	md5hash 値	trend micro	Avg	Kaspersky
12	4ee0f67ed642ea4b1b81ce15e6063e29	ELF_BASHLITE.SM	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
13	9133cb25d9df542a131c23043d0b3e87	ELF_BASHLITE.DHD	a variant of Linux/ Gafgyt.MT	Linux/Fgt
14	6b70af7e7b23a4308d8cc998e7a719d4	ELF_BASHLITE.DIB	a variant of Linux/ Gafgyt.SR	Linux/Fgt
15	33f9b8c338b23a5101c8368a6cf55890	ELF_BASHLITE.DIA	a variant of Linux/ Gafgyt.QE	Linux/Fgt.BP
16	97688fd70bfcf344a9ab1ea1cd7c3b6a	検知されず	a variant of Linux/ Gafgyt.C	Linux/Fgt.CE
17	e8b7ba10d79ba9803391996dc548233c	BASHLITE	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
18	933364b18742a0bbd7b9b5ca1b0adea6	検知されず	a variant of Linux/ Mirai.A	Linux/Fgt.CI
19	be01698c9dfa07c0e4f9d93b89ef3b82	検知されず	検知されず	Linux/Fgt.CI
20	9ba4401c2a4faa8975175498dc1fbfd4	ELF_MIRAI.A	a variant of Linux/ Mirai.A	Linux/Fgt.CI
21	cfec82edb6a96d43667741ab62d63a44	検知されず	a variant of Linux/ Tsunami.NAL	Linux/Tsunami.CT
22	5afdcceb2fc5fc1c15d7fdbef674c6a5	検知されず	a variant of Linux/ PNScan.A	Linux/Generic_c.ZB
23	811351fd15c2c4c355543c4bac3d7cc9	ELF_SONEX.SMC	Linux/Dnsamp.A	Linux/MrBlack.D

表 A.3 マルウェア検体 (MIPSEL)

検体	md5hash 値	trend micro	Avg	Kaspersky
24	7a1bace1b58a1619fe4122ee8cfce3a5	ELF_BASHLITE.SM	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
25	c214ddb267d339229f71875cc1839261	ELF_BASHLITE.DHD	a variant of Linux/ Gafgyt.MT	Linux/Fgt
26	7474bec1a0427efe1a46d79293a1b706	ELF_BASHLITE.DIB	a variant of Linux/ Gafgyt.SR	Linux/Fgt
27	dcd9d22066509902647e4a40741a9dcd	ELF_BASHLITE.DIA	a variant of Linux/ Gafgyt.QE	Linux/Fgt.BP
28	6541826fd6428a99ef77a8188073192c	検知されず	a variant of Linux/ Gafgyt.C	Linux/Fgt.CE
29	65cdb4c8a76dd0cc75f0aa15f13590fd	検知されず	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
30	f6e7d8574469ed5f76f23945efaeb0cc	検知されず	a variant of Linux/ Mirai.A	Linux/Fgt.CI
31	d55e3cccf128eb0f6b1ec176fdce2573	検知されず	検知されず	Linux/Fgt.CI
32	5849bb9ceefee5ef295e7e966d0ba2b5	ELF_MIRAI.A	a variant of Linux/ Mirai.A	Linux/Fgt.CI
33	e0f1b98d6778f0eba97be0a3cafc4d52	検知されず	a variant of Linux/ Tsunami.NAL	Linux/Tsunami.CT
34	ac93f826c3031859f4a71529794fe7f9	検知されず	Linux/Dofloo.A	Linux/Dofloo.A
35	856f14251f643bac62b9193c54449472	検知されず	Linux/PNScan.A	Linux/Generic_c.AKS

表 A.4 マルウェア検体 (SH)

検体	md5hash 値	trend micro	Avg	Kaspersky
36	65aaa70a0986597196cb7416fab7de9b	ELF_BASHLITE.SM	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
37	02f72a165304074e9e437800c35154da	ELF_BASHLITE.DHD	a variant of Linux/ Gafgyt.MT	Linux/Fgt
38	4b49b439ff434f0aad0c36efd22378a9	ELF_BASHLITE.DIB	a variant of Linux/ Gafgyt.SR	Linux/Fgt
39	e62aa5444b24a36848d012c67ee3bcf9	ELF_BASHLITE.DIA	a variant of Linux/ Gafgyt.QE	Linux/Fgt.BP
40	9f9cc8c71822ca872de2c28e9c41f44a	検知されず	a variant of Linux/ Gafgyt.C	Linux/Fgt.CE
41	e892b8e008aeb50f2634d7c602b8fb77	検知されず	a variant of Linux/ Gafgyt.C	Linux/Fgt.AB
44	c44e92168d07a637fa8943dd95f2a462	検知されず	a variant of Linux/ Mirai.A	Linux/Fgt.CI
42	e867f3fc4bffe17bfd8cdcbfbfb28dc5	検知されず	検知されず	Linux/Fgt.CI
43	a490bb1c9a005bcf8cfe4bdffe7b991f	ELF_MIRAI.A	a variant of Linux/ Mirai.A	Linux/Fgt.CI
45	cc33124ee92cc9b9b120cd54fee94d9b	検知されず	a variant of Linux/ Tsunami.NAL	Linux/Tsunami.CT

B　5 章および 6 章の実験に用いたマルウェア検体一覧

5.3 節および 6.3 節の実験に使用したマルウェア検体の一覧を表 B.5，表 B.6，表 B.7，表 B.8 に示す．

表 B.5 マルウェア検体 (MIPSEL)

No.	md5hash 値	ファミリー
1	23efccb92c175c3335ac0c37ca671d87	Mirai
2	0f2192a61d637e55726a9d60ac8a661c	Mirai
3	1ca9ea33f1c582a996b69310aaef7c28	Mirai
4	2e1031c4aa5b129ac523c9d775f2798a	Mirai
5	e336138493bf7ff772431542001524be	Mirai
6	f5c070bfac6a95e9d9c23068c166cee2	Mirai
7	7afa8de836a328cd0b6098029a901759	Mirai
8	21fad4f163a892d1a1cbc0f827361828	Mirai
9	705b61a5443e720a6277207228af6553	Bashlite
10	34d65bcf7beb2b30b9ec2bab9d7c7eac	Bashlite
11	d9bd10edc15a4c260360a6e5f26db71c	Bashlite
12	53fbe92b5e50bee9c01ce2bf5dd160dd	Bashlite
13	7ae081e089bc74217fc27da4e85d6eb9	Bashlite
14	45f4630e26dce8d44feb09546bd8d86b	Bashlite
15	7faebdc2ac4131b274dd0b7b31b5fd80	Tsunami
16	09ebdeff43ed606b946bc38e17300ec8	Tsunami
17	394a45f62d0c10632b220607ef3abcf6	Tsunami
18	75c66f96ec9d48121439ea30363a4b7a	Tsunami
19	cd7035a2f06574eeb68b1a083b67f5a1	Tsunami
20	4b42e368292c71533f78adab1e7affa9	Hajime
21	b992e3b402c290c43f89b92ad21eba48	Hajime
22	ca484bb9df12919b4ddb52c30ebf6d6f	Hajime

表 B.6 マルウェア検体 (ARM)

No.	md5hash 値	ファミリー
23	e7430db8a6e0462203cddf34b2d3efd2	Mirai
24	0a2d5fd18ecb30595502f91b983ed274	Bashlite
25	6bb847315bcceaf2b65b137323139cde	Bashlite
26	4d87c296cae43271e1ea3dd0bd42eab6	Bashlite
27	dec8235a9636ece0e045ba6981715de5	Bashlite
28	e37a4905fe1fdfc5100b71396152d1cb	Bashlite
29	151aeea82c1b53196a75131758ba02d1	Bashlite
30	16dc9f2e07786a4e0d5f1ef90482b0b3	Bashlite
31	25552139bce4f600412e9a0586ab63f0	Bashlite
32	6a677671902ebcf19ffd4240624d13c3	Bashlite
33	893f5e9e1af80177036f496b219aa1f0	Bashlite
34	8d581205858b4889884afd045db4d271	Bashlite
35	fe41d3f4beef77bde4e2190405a1cfff	Bashlite
36	02211b6c2b94ed22a44d8cf83ee62d55	Tsunami
37	2e725f347ea875cf1b647583f6f9f79a	Tsunami
38	6c52af748ebe2939aa873451a2985ce8	Tsunami
39	c59541af3f28440b765d4f5f8f622323	Tsunami
40	d4de4f34c21173fa25fedeb3f6f912e1	Tsunami
41	d9d266ae5552f1474ed73e05bab0761f	Tsunami
42	f6e9e791267c506acefdae6a07a7efe6	Tsunami
43	f3309f143cca7954dba2887aadd2b4cb	VPNFilter

表 B.7 マルウェア検体 (MIPS(1))

No.	md5hash 値	ファミリー
44	bda73d9297e77758f5e67c1b9285c3d2	Mirai
45	0d4b1eadb32495809d16d599e5ae4a83	Mirai
46	6f555e8b10e8dbc72b7b25c5c3560b6b	Mirai
47	6fddad057473660f9db69e1de4cde981	Mirai
48	9d48aced2e367f36c795dcebb45b1485	Mirai
49	bd4fc570e5118bca633f920c52f7f1c5	Mirai
50	c37702ad250f2fe3e2bdb5472f433cec	Mirai
51	c76bde42a558baf3503be8ea36818656	Mirai
52	0ca78871d506bc48ee8da72f5ce2e925	Bashlite
53	052ed497a2b3253e45633cf2c9f80de7	Bashlite
54	3f6a6cf5f5d7a1d6e770eacacfd9ff3d	Bashlite
55	4357a64efc1cce5027693e9dbeb7c944	Bashlite
56	6799bdb006f0cfda1ff3edeafa474137	Bashlite
57	d82bd2c63df257470cadaf52d2f34b2e	Bashlite
58	d872a5b18b3a87d820639b45f33562e4	Bashlite
59	e38a80320ae50b83ca4e156a7cff2fd2	Bashlite
60	e52e18491b8bacc88db992b8b90bc68b	Bashlite
61	06488dab48127cac3a80d7536b772ff1	Tsunami
62	043f4e84c2ea37456319baccd21d0b00	Tsunami
63	245c9cc38b412e8f44d3b92fc2daa732	Tsunami
64	0ec3f75940588679c5f886c8487edb13	Tsunami
65	1380ccf9bf702f3df6f68be452492e21	Tsunami
66	4decb63973245dbdced2a0aaafebde25	Tsunami
67	7d993783b0dd0bdba4ae95cee4d5827f	Tsunami
68	912fcdf63527c7e3a4cfb605a4eac771	Tsunami
69	9f54ad2ab5f23d70b2f19caf06f4811d	Tsunami
70	a28163237b79fca42ba35b22cc82a6c0	Tsunami
71	ccdcbdcf23d5aadd3192c3109a92c165	Tsunami
72	cfec82edb6a96d43667741ab62d63a44	Tsunami
73	d96fdf5692b920a7b755893d66b6abf7	Tsunami
74	00344f59cd58358657bb461c8600c342	Hajime
75	11e02d5125bc58f493ce45b4cd3f6e52	Hajime
76	0a5a4371519904b7e9ae3670c682b3d4	Hajime
77	400ecd03db7bde9479f23562f9afcb84	Hajime
78	4eacdd6f8ab0ce3ea10855877918cb4e	Hajime
79	505573950c2333feabe583af980f9cfe	Hajime
80	598be394d467c6528422e624c496dbae	Hajime

表 B.8 マルウェア検体 (MIPS(2))

No.	md5hash 値	ファミリー
81	680bd0849cffc422e336acc3de4c4f92	Hajime
82	8a94ff6711d0565459e50cb2a16df75c	Hajime
83	90d43edb00108f51717b1678c2e50071	Hajime
84	ec1263f9ba78d57e6f50292a8fb059c9	Hajime
85	5377e8f2ebdb280216c37a6195da9d6c	Hajime
86	45871bad3a9b4594fc3de39e4b5930ad	VPNFilter
87	d0dbbd31fe5205295301b2241610ef9c	VPNFilter