IoT機器の埋め込みログイン情報の検出と脆弱性解析の環境改善に関する研究

依田 みなみ

電気通信大学大学院情報理工学研究科

博士（工学）の学位申請論文

2024年3月

IoT機器の埋め込みログイン情報の検出と脆弱性解析の環境改善に関する研究

博士論文審査委員会

主査　　大須賀 昭彦　教授

委員　　南 泰浩　教授

委員　　広田 光一　教授

委員　　大坐畠 智　教授

委員　　田原 康之　准教授

著作権所有者

依田 みなみ

2024

Detection of Embedded Login Information in IoT Devices and proposal of improving the environment for vulnerability analysis

Minami Yoda

Abstract

IoT (Internet of Things) devices have become popular in recent years and are expected to be utilized further in the future. Examples include routers that connect Internet-enabled devices such as computers and smartphones to the Internet, camera devices that monitor rooms while the user is away from home, and devices that automatically adjust daily activities and room temperature. While the use of IoT devices has made our lives more convenient, cyberattacks targeting vulnerabilities in IoT devices are also on the rise. For example, a malware called "Mirai" hijacks vulnerable IoT devices and instructs them to access a large number of specific servers, overloading the servers. This caused social damage. As shown above, some attacks targeting vulnerabilities in IoT devices cause significant social damage, so early detection of vulnerabilities is important.

In addition, various manufacturers are entering the IoT device market due to the low cost of IoT device components. Since different manufacturers have different security policies and responses to vulnerability diagnostics, there is a possibility that unintended vulnerabilities may be included due to omission of vulnerability detection and forgotten deletion of development code. Therefore, it is important to develop a vulnerability detection tool that allows IoT device users to easily diagnose vulnerabilities in their IoT devices, similar to malware diagnostic software for computers. However, existing public tools are complicated to use even for those with expertise, and their licenses are expensive and inaccessible.

In this study, we propose a vulnerability detection method for IoT devices and a vulnerability detection tool that is free and easy to use. For the vulnerability detection method for IoT devices, we propose three methods to detect "embedded login information", which is the No. 1 risk

in the vulnerability ranking of IoT devices. Based on the proposed methods, we developed our own vulnerability detection tool and evaluated its vulnerability detection capability through experiments. In the evaluation experiments, we analyzed IoT devices actually sold using the proposed methods and found that the proposed methods have the same vulnerability detection capability as existing research, and also found vulnerable communication programs that have not been discovered so far. These results contribute to the detection of "embedded login information", which is one of the most important vulnerabilities in IoT devices.

In developing the vulnerability detection tool, we investigated a problem with existing vulnerability detection tools, developed a tool that solves the problems, and released it for free on Github. The problems with existing tools include the complexity of environment construction and pre-processing for vulnerability analysis, which require man-hours, and the long analysis time. In this method, we first defined requirements to solve common problems, and then developed tools to meet the requirements. Evaluation experiments were conducted. In the evaluation experiment, we measured the environment construction, preprocessing man-hours, and analysis time to confirm whether the proposed tool met the requirements. In both cases, the proposed tool completed in less time than the existing tools, confirming the usefulness of the proposed tool.

With these two research results, we have successfully developed a method and tool that allows anyone to easily detect an embedded login information of IoT devices.

IoT 機器の埋め込みログイン情報の検出と脆弱性解析の環境改善に関する研究

依田 みなみ

概要

近年，IoT（Internet of Things）機器が普及しており，今後もさらなる利活用が期待されている．IoT 機器とは，インターネットを活用してサービスを提供するデバイスである．例えば，コンピュータやスマートフォンなどのインターネット接続機能を持つデバイスをインターネットに接続するルーター，留守中の部屋をモニタリングするカメラデバイス，日常動作や室内の温度を自動調整するデバイスなどがある．IoT 機器の活用によって生活の利便性が向上する一方，IoT 機器の脆弱性を狙った攻撃も増加している．例えば「Mirai」と呼ばれるマルウェアは，脆弱な IoT 機器を乗っ取って特定のサーバーに大量アクセスするよう命令し，サーバーに負荷を与えるマルウェアである．Mirai に感染した大量の IoT 機器によって，大規模 SNS や通販サイトのサーバーがダウンし，サービスが停止する社会的被害をもたらした．このように，IoT 機器の脆弱性を狙った攻撃は社会的被害が大きいものもあり，脆弱性の早期発見が重要である．中でも,「埋め込みログイン情報」に起因する脆弱性は，IoT 機器で留意すべき脆弱性のランキング 1 位となっており，早期発見が重要視される脆弱性である．埋め込みログイン情報とは，IoT 機器のプログラムに ID とパスワードなどのログイン情報となる文字列が埋め込まれており，このログイン情報を知る第三者ならば誰でも IoT 機器に不正ログインできてしまう脆弱性である．埋め込みログイン情報を検出する既存研究では，未知の脆弱性を発見しているものの，条件文で使用されている埋め込みログイン情報のみを検出対象としているため，その他の部分で使用されている場合には検出漏れが生じる．また，手法を提案するだけでは脆弱性は検出できないため，提案手法を実際の IoT 機器に適用して脆弱性を検出する過程も重要である．そのため，手法の提案に加えて脆弱性検出ツールが必要となる．

脆弱性の検出ツールにおいては，一般公開されている既存の IoT 機器脆弱性検出ツールは，専門知識を有する者にとっても利用方法が複雑で作業時間が長時間化することや，ラ

イセンスが高額で手が届きにくい問題がある．近年はIoT機器の部品の低コストがすすみ，IoT機器市場に様々なメーカーが参入している．メーカーによってセキュリティガイドラインや脆弱性診断の対応が異なるため，脆弱性の検出漏れや開発用コードの消し忘れ等による意図せぬ脆弱性が含まれる可能性も高まる．メーカーの多様化による安全基準のばらつきを対策する手段の一つとして，コンピュータのマルウェア診断ソフトウェアのように，IoT機器の購入者であるユーザ自身が簡単に手持ちのIoT機器の脆弱性診断できるしくみづくりが必要である．

そこで本研究では，埋め込みログイン情報の検出手法と，無償で簡単に利用できる埋め込みログイン情報の検出ツールを提案する．埋め込みログイン情報の検出手法では， IoT機器で留意すべき脆弱性のランキング1位となっている「パスワードの埋め込み」を検出対象とし，本研究では3つの検出手法を提案する．提案手法をもとに独自の検出ツールを開発し，実験で脆弱性の検出能力を評価した．評価実験では実際に販売されているIoT機器を提案手法で解析し，提案手法が既存研究では発見できなかった埋め込みログイン情報を1件，未報告の脆弱な通信プログラムを1件発見した．これらの研究成果により，本手法はIoT機器の脆弱性で最も重要視されている「パスワードの埋め込み」の検出に有用であることがわかった．

埋め込みログイン情報の検出ツールの開発では，既存の脆弱性検出ツールを調査し，課題を解決するツールを開発して，GitHub上に無償公開した．GitHubは，世界中で幅広く使用されている，オンライン上でソフトウェアを公開・共有できるソフトウェア開発プラットフォームである．既存ツールの課題として，環境構築と脆弱性解析の前処理が複雑で工数がかかること，解析時間の長時間化がある．本手法では，まず共通課題を解決する要件を定義し，次に要件を満たすツールを開発した．評価実験では，提案ツールが要件を満たすか確認するため，環境構築，前処理作業工数ならびに解析時間を計測した．いずれの結果においても，提案ツールが既存ツールよりも短時間完了し，提案ツールの有用性を確認した．最後に，提案ツールが脆弱性検出ツールの共通機能を提供するミドルウェアとして展開できるか検討し，評価を行った．結果として，提案ツールが既存研究の検出ツールよりもミドルウェアとしての有用性が高いことがわかり，提案ツールの今後の拡張性の高さについても示すことができた．

以上の研究成果により，本研究では誰もが無料かつ短時間で IoT 機器の埋め込みログイン情報を検出できる手法とツールの開発に成功した.

目 次

図 目 次

表 目 次

第1章　序論

本章では，本研究の背景を述べた後，本論文の目的と貢献を説明する．その後，本論文の構成について述べる

1.1　本研究の背景

2018 年から 2025 年の IoT（Internet of Things）機器市場の年平均成長率は 21%と予測されており，今後も普及が期待されている．一方で，IoT 機器の脆弱性を狙った攻撃も増加している．2018 年前半期のハニーポットへの攻撃件数は 2 億 3100 万件であるのに対し，2020 年下半期は 42 億件と約 18 倍となった [1]．ハニーポットとは意図的に脆弱性を含ませたシステムで，攻撃者の手口や行動調査に使用されている．

IoT 機器の脆弱性を狙った攻撃の一例として,「Mirai」と呼ばれるマルウェアがある [2]. Mirai は脆弱な IoT 機器を乗っ取って特定のサーバーに大量アクセスするよう命令し，サーバーに負荷を与えるマルウェアである．Mirai に感染した大量の IoT 機器によって，大規模 SNS や通販サイトのサーバーがダウンし，様々なサービスが停止する社会的被害が報告されている．他にも，Zyxel 社製のルーターのプログラムにログイン情報が埋め込まれていた事例がある．ファームウェアと呼ばれるルーターの本体プログラムに ID とパスワードの文字列が埋め込まれており，ログイン情報を知り得た第三者が不正ログインできる状態であった．ルーターは 10 万台以上の販売実績があったため社会的被害が大きく，深刻な脆弱性として共通脆弱性識別子（CVE）に CVE-2020-29583 として登録されている．埋め込みログイン情報は，OWASP による IoT 機器で留意すべき脆弱性ランキングでも 1 位となっており，検出すべき重要な脆弱性である．このように，IoT 機器の脆弱性を狙った攻撃は社会的被害が大きいものもあり，これまでも IoT 機器の脆弱性検出を目的とした様々な手法が提案されている．また，手法を提案するだけでは脆弱性は検出できないため，提案手法を

実際の IoT 機器に適用して脆弱性を検出する過程も重要である．そのため，手法の提案に加えて脆弱性検出ツールが必要となる．

脆弱性の検出ツールにおいては，一般公開されている既存の IoT 機器脆弱性検出ツールは，専門知識を有する者にとっても利用方法が複雑で作業時間が長時間化することや，ライセンスが高額で手が届きにくい問題がある．近年は IoT 機器の部品の低コストがすすみ，IoT 機器市場に様々なメーカーが参入している．メーカーによってセキュリティガイドラインや脆弱性診断の対応が異なるため，脆弱性の検出漏れや開発用コードの消し忘れ等による意図せぬ脆弱性が含まれる可能性も高まる．メーカーの多様化による安全基準のばらつきを対策する手段の一つとして，コンピュータのマルウェア診断ソフトウェアのように，IoT 機器の購入者であるユーザ自身が簡単に手持ちの IoT 機器の脆弱性診断できるしくみづくりが必要である．

1.2　本研究の目的と貢献

本研究では，埋め込みログイン情報の検出手法と，無償かつ短時間で埋め込みログイン情報が検出できるツールを提案する．本研究では 3 つの検出手法を提案する．提案手法をもとに独自のツールを開発し，実験で脆弱性の検出能力を評価した．評価実験では実際に販売されている IoT 機器を提案手法で解析した．その結果，提案手法が既存研究では検出していない既知の埋め込みログイン情報を 1 件と，未報告の脆弱な通信プログラムを 1 件発見した．これらの研究成果により，本研究は IoT 機器の脆弱性で最も重要視されている「パスワードの埋め込み」の検出に貢献することがわかった．

検出ツールの提案では，既存の脆弱性検出ツールの課題を調査し，課題を解決する検出ツールを開発して GitHub 上に無償公開した．既存ツールの課題として，環境構築と脆弱性解析の前処理が複雑で工数がかかること，解析時間の長時間化がある．評価実験では．提案ツールが要件を満たしているか確認するため，環境構築，前処理作業工数ならびに解析時間を計測した．いずれの結果においても，提案ツールが既存ツールよりも短時間で作業が完了でき，提案ツールの有用性を確認した．これら 2 つの研究成果により，誰もが無償かつ短時間で IoT 機器の埋め込みログイン情報を検出できる手法とツールの開発に成功した．

1.3 本論文の構成

本論文の構成は以下である．まず，IoT 機器とりまく脆弱性について，現状と解析環境について述べる．次に，現状の問題点を洗いだし，本研究が取り組む課題について述べる．そして，埋め込みログイン情報の検出手法を 3 つ提案し，提案手法の検出能力について評価と考察を述べる．次に，検出ツールを開発するため，ツールの要件を定義し，検討ツールを構築する．そして，検出ツールの評価と考察を述べる．更に，検出ツールのミドルウェア化について検討するため，ミドルウェアの試作と評価及び考察を述べる．最後に，本研究のまとめと今後について述べる．

第2章　IoT機器の脆弱性と検出手法の現状

本章では，IoT 機器の脆弱性とそれらを検出する手法の現状について説明する．まず，2.1 節では，IoT 機器における脆弱性について説明する．2.2 節では，検出手法の種別について説明する．2.3 節では，多くの検出手法で活用されている逆解析について説明する．2.4 節では，IoT 機器の脆弱性で最も留意すべきとされている「埋め込みログイン情報」について説明する．2.5 節では，埋め込みログイン情報の実例について説明する．2.6 節は，埋め込みログイン情報を検出する既存研究について説明する．

2.1　IoT 機器の脆弱性

IoT 機器で特に留意すべき脆弱性の指標として，表 2.1 に示す 2018 OWASP Internet of Things Top 10 が有用である [15]．この指標は， The Open Web Application Security Project（OWASP）が提案したものである．OWASP とは，様々なコンピュータ技術に関する脆弱性や課題の解決を目的としたオープンコミュニティである．2018 OWASP Internet of Things Top 10 の作成手順は，まずコミュニティメンバーが，IoT に関する著名なセキュリティプロジェクトや実際の脆弱性情報を収集し，脆弱性の内容と優先順位を決定づける．調査対象のセキュリティプロジェクトは，CSA IoT Controls Matrix，CTIA，Stanford’s Secure Internet of Things Project，NISTIR 8200，ENISA IoT Baseline Report 及び Code of Practice for Consumer IoT Security である．そして，脆弱性の内容と優先順位について，コミュニティ内レビューとコミュニティ外のパブリックレビューの両面で議論し，複数回の修正を重ねてリリースされた．

OWASP の指標では「強度が弱い，予測が容易，または埋め込まれたパスワード」が，最も留意すべき脆弱性であると述べている．IoT 機器の脆弱性事例のうち，大きな社会的被害をもたらした事例としてあげられるマルウェア「Mirai」も，初期状態の脆弱なパスワー

表 2.1: Top 10 OWASP vulnerabilities in 2018

順位	脆弱性の内容
1 位	強度が弱い，予測が容易，または埋め込まれたパスワード
2 位	不安全な通信サービス
3 位	不安全なエコシステムのインターフェース
4 位	ソフトウェア更新の安全なしくみの欠如
5 位	不安全・保障切れのソフトウェアコンポーネントの使用
6 位	不十分なプライバシー保護
7 位	不安全なデータの転送や保存
8 位	サポートなどのデバイス管理の欠如
9 位	不安全な標準設定
10 位	不十分な物理的堅牢化

ドを使用する IoT 機器を狙ったものである．また，10 万台の販売実績がある Zyxel 社製のルーターには，ID とパスワードのログイン情報が埋め込まれていた．これにより，ログイン情報を知り得た第三者が不正ログインできる状態であった．ルーターは 10 万台以上の販売実績があったため社会的被害が大きく，深刻な脆弱性として共通脆弱性識別子（CVE）に CVE-2020-29583 として登録されている．事例が多く，社会的被害も大きい事例があることから，最も留意すべき脆弱性である．

2.2　動的解析と静的解析

IoT 機器の脆弱性検出手法は，大別すると動的解析と静的解析がある．動的解析は，エミュレータを用いた仮想環境上で IoT 機器を動作させ，挙動や機能を調査して脆弱性を検出をする．静的解析は，IoT 機器のプログラムを復元し，IoT 機器を動作させずに復元プログラムを解析して脆弱性を検出する．動的解析は，動作した機能のみが解析対象となるため，IoT 機器の設計書がないブラックボックスの状態では，全機能を網羅することは困難である．一方，静的解析は元プログラムが正しくを復元されていれば，プログラム全体を解析できるため，現在は多くの手法が静的解析を採用している．

2.3 逆解析を用いた静的解析

静的解析では，解析の前処理として逆解析をおこなう．逆解析とは，バイナリ形式の IoT 機器のファームウェアから，プログラムなど様々な情報を取り出し，解析しやすくするものである．逆解析はリバースエンジニアリングとも呼ばれる．

逆解析を用いた静的解析の流れを示す．まず，IoT 機器のメーカーの WEB サイトなどからファームウェアを入手する．ファームウェアはバイナリ形式が一般的である．次に，ファームウェアを展開して構成ファイルをとりだす．構成ファイルも同様にバイナリ形式である．とりだした構成ファイルのうち，解析対象となるファイルをリバースエンジニアリングソフトウェアにインポートする．リバースエンジニアリングソフトウェアは，構成ファイルを逆解析し，プログラムを復元するものである．この段階で，ようやく脆弱性検出アルゴリズムが適用可能となる．最後に，脆弱性検出アルゴリズムを適用し，脆弱性を検出する．

```
(base) root@797a83384f5e:~/Documents/firmware-mod-kit# ./extract-firmware.sh firmware/wbd-500-firmware/wbd-500-firmware.bin
Firmware Mod Kit (extract) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake

Scanning firmware...

Scan Time:     2021-01-26 15:07:09
Target File:   /root/Documents/firmware-mod-kit/firmware/wbd-500-firmware/wbd-500-firmware.bin
MD5 Checksum:  79a555bb41d626553542bb6dc593be04
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             TRX firmware header, little endian, image size: 6660124 bytes, CRC32: 0x6701E239, flags: 0x1, version: 1, header size: 28 bytes, loader offset: 0x1C, linux kernel offset: 0x0, rootfs offset: 0x0
28            0x1C            LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes, uncompressed size: 2939620 bytes
917532        0xE001C         Squashfs filesystem, big endian, DD-WRT signature, version 3.0, size: 5741926 bytes, 875 inodes, blocksize: 131072 bytes, created: 2013-01-24 04:30:58

Extracting 917532 bytes of trx header image at offset 0
Extracting squashfs file system at offset 917532
Extracting squashfs files...
```

図 2.1: IoT 機器のファームウェアから構成ファイルを取り出す過程

図 2.1 は，ファームウェアから構成ファイルを取り出す様子である．専用のソフトウェアを用いて構成ファイルを取り出す．静的解析では，取り出した構成ファイルを 1 つずつ逆解析する．図 2.2 に，ファームウェアと構成ファイルの対応を示す．D-Link 社の 9 製品に含まれる構成ファイルの平均数を調査したところ 1,894 個であった．このように，ファー

図 2.2: IoT 機器のファームウェアには多数の構成ファイルが含まれている

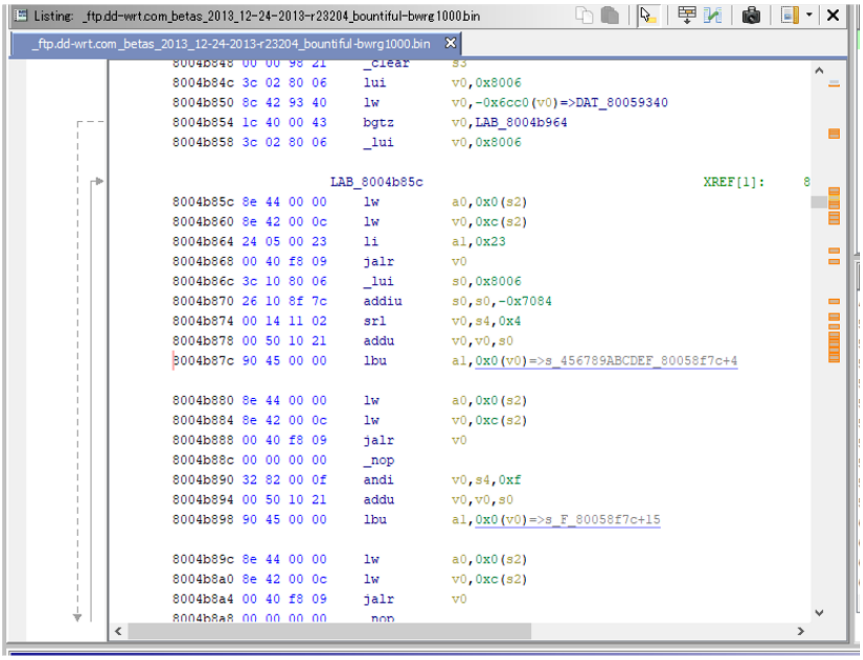

図 2.3: 構成ファイルを逆解析して得られたアセンブリコード

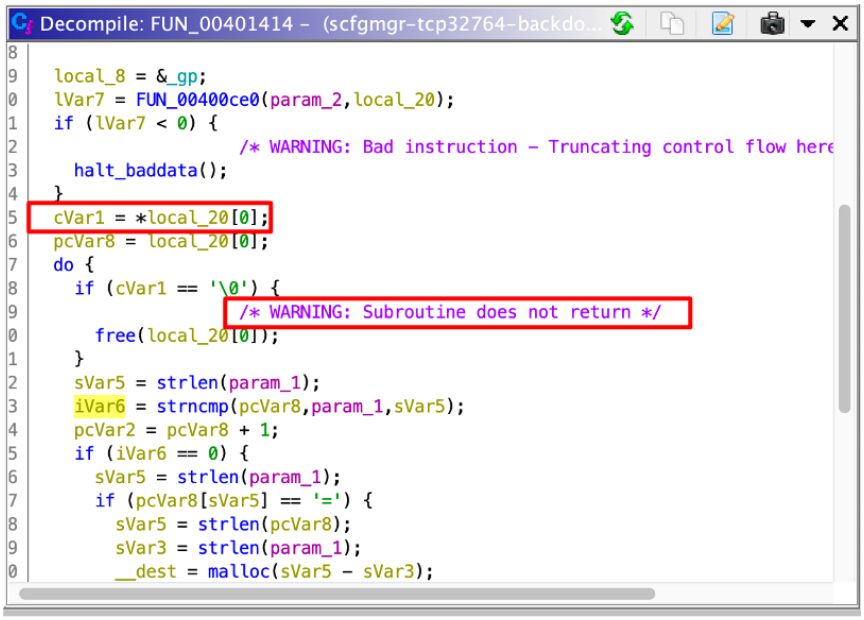

図 2.4: 構成ファイルを逆解析して得られたソースコード

ムウェアに含まれる構成ファイルは多数であるため，手がかりなしに脆弱性を検出するのは現実的とは言えない．そのため，脆弱性検出アルゴリズムの提案が必要である．

逆解析で得られる結果の一部を図 2.3，2.4 に示す．逆解析結果によって，構成ファイルのアセンブリコードやプログラムが復元される．図 2.4 では，構成ファイルのソースコードが復元できたものの，上部赤枠のように変数名が規則的で変数名からプログラムの内容を予測できなくなっていたり，下部赤枠のように復元エラーによるプログラムの欠損もみられる．このように，IoT 機器の解析では不完全なプログラムをもとに解析する場合が多いため，WEB アプリケーションなどの完全なソースコードがあるソフトウェアの解析に比べて複雑である．

2.4 本手法の検出対象と埋め込みログイン情報の定義

2.1 節において，IoT 機器で最も留意すべき脆弱性は「強度が弱い，予測が容易，または埋め込まれたパスワード」であることがわかった.「強度が弱い，予測が容易」なパスワードは，ユーザーによって対策可能である．例えば，IoT 機器の初期設定でユーザーにパスワードの変更を促すフローを追加するなどで対策できる．他にも，メーカーが初期設定の

パスワードを複雑にし，更に機器ごとに設定することで対策できる．しかし,「埋め込まれたパスワード」は IoT 機器のファームウェアのプログラムにパスワードの文字列が直接埋め込まれているため，ユーザーは変更できない．情報を知り得た第三者なら誰でもが不正ログインできてしまい危険である．ファームウェアは IoT 機器の本体プログラムを指す．そのため，本手法では「プログラムに直接埋め込まれたパスワード」の検出手法を提案する．また，パスワードの付近には ID に相当する文字列が埋め込まれていることが多いため，本手法では ID も検出対象とする．これらをまとめて「埋め込みログイン情報」と称する．まとめると，埋め込みログイン情報とは，IoT 機器のファームウェアのプログラムに直接埋め込まれた ID とパスワードの文字列を指す．

2.5　埋め込みログイン情報の実例

```
int FUN_00060118(Backdoor)(int param_1,char *param_2,char *param_3)

{
  int iVar1;
  char *__s;
  int iVar2;
  int iVar3;
  int iVar4;

  FUN_000cef88(DAT_004a6788,*(undefined4 *)(param_1 + 0x334),0);
  iVar1 = strcmp(param_3,"664225");
  if ((iVar1 == 0) && (iVar1 = strcmp(param_2,"root"), iVar1 == 0)) {
    puts("LINE[1] REMOTE USER LOGIN IN!");
    return 1;
  }
```

図 2.5: Q-see 社のドライブビデオレコーダー（DVR）で発見された埋め込みログイン情報

図 2.5 に，埋め込みログイン情報の例を示す．このプログラムは，IoT 機器のファームウェアを逆解析して取り出したものである．ID は `strcmp(username, "root")`，パスワードは `strcmp("664225", password)` に埋め込まれている．`strcmp` は C 言語の文字列比較関数で，第一引数と第二引数の文字列が一致するかを判定する関数である．このような関数や変数は，ファームウェア内ではシンボルと呼ばれる．図 2.5 では，`strcmp` シンボルを使って，埋め込まれた ID/パスワードを変数と比較されている．変数にはユーザーが入力し

た値が格納されている．このように，埋め込みログイン情報は strcmp シンボルを使って，ユーザーが入力した値と比較される特徴を持つ．

2.6 既存の埋め込みログイン情報検出手法

Stringer

Thomas らは，ファームウェアに埋め込まれたパスワード文字列を発見する Stringer を提案した [25]．この方法は静的文字列の比較関数に重み付けをし，重みの大きい関数をバックドアのパスワード候補と判断する．重みは各関数に到達するために必要な静的データ列や分岐の数などから算出し，静的データが関数の分岐に与える影響度を測定する．実験では 30 社が提供する 2,451,532 個のバイナリのデータに手法を適用し，未知のバックドアを 3 つ発見した．しかし，Stringer は分岐以外の埋め込みログイン情報の検出に非対応であるため，分岐の前で埋め込みログイン情報の比較を結果を格納した変数を用意し，その変数が分岐で使用されている場合などで検出漏れが発生する．

第3章　IoT機器の脆弱性検出ツールの現状

本章では，IoT 機器の脆弱性検出ツールの現状と，既存ツールの調査結果について述べる．まず，3.1 節では，IoT 機器の脆弱性検出ツールについて説明する．次に 3.2 節では，既存の脆弱性検出ツールを実際にインストールし，使用した結果と考察を述べる．最後に，3.3 節で既存の脆弱性検出ツールの課題を整理する．

3.1　脆弱性検出ツールの現状

IoT 機器における脆弱性検出ツールとは，IoT 機器を解析して特定の脆弱性検出アルゴリズムを適用し，その結果を出力するものである．IoT 機器の脆弱性を検出するには，IoT 機器を解析できる状態に変換する必要があるため，脆弱性検出アルゴリズムの提案のみでは脆弱性は検出できない．そのため，脆弱性検出アルゴリズムの提案に加えて，脆弱性検出ツールの構築も必要不可欠である．従来の脆弱性検出ツールの想定ユーザーは研究機関や企業であったが，最近は GitHub などのソフトウェア開発プラットフォームで様々な検出ツールが公開されており，誰もが脆弱性検出ツールを利用できる環境が広まっている．そのため，IoT セキュリティの研究機関や企業だけではなく，自身の IoT 機器の脆弱性を解析したい一般ユーザーも脆弱性検出ツールを利用するようになった．

3.2　既存の脆弱性検出ツール

本節では，IoT 機器の脆弱性検出手法に関する研究を 10 件調査し，研究内容と検出ツールを使用した考察を述べる．調査対象の研究は未知の脆弱性を発見した論文，引用数が多い著名論文や，最新の論文とした．検出ツールが公開されている研究については，検出ツールのインストールとサンプル実行の可否についても調査した．検出ツールのインストール

環境は，検出ツールが Docker などの仮想環境で配布されている場合は配布された環境を使用する．環境が配布されていない検出ツールは，Ubuntu22.04 の Docker コンテナ上でインストールをした．表 3.1 に，関連研究が提案した検出ツールのサンプル実行の可否をまとめる．

表 3.1: 既存研究が提案する検出ツールのサンプル実行の可否

ツール名	サンプル実行の可否	実行不可の理由
FIRMADYNE	-	依存ライブラリの未解決
Stringer	-	入力データ形式が不明瞭
HumIDIFy	-	使用方法が未記載
FirmFuzz	-	依存ライブラリの未解決
KARONTE	✓	-
Number of ✓	1	-

Firmalice

Shoshitaishvili らはシンボリック実行を用いてファームウェアを解析し，認証回避（バックドア）を検出する Firmalice を提案した [19]．バックドアとはプログラムの設計上，意図しない経路を経由してファームウェアに入り込み，特権操作や不正操作を可能とする脆弱性である．Firmalice はファームウェアの認証回避が起こりうるポイント（特権プログラムの実行ポイント）をセキュリティポリシーとして記述し，このポリシーに基づいてバックドアを検出する．セキュリティポリシーは主に「意図的に埋め込まれたログイン情報」「意図的に隠された認証インターフェース」「意図的でないバグ」の 3 種類の情報が記述されている．解析の第一段階として，静的解析でファームウェアのコントロールフローグラフを構築する．そして構築したグラフを用いてプログラムのシンボリック実行を行い，プログラムの特権実行箇所を検出する．特権操作の実行点を検出した場合，その実行点がバックドアになりうるかをチェックして脆弱性を発見する．ツールは非公開である．

PIE

PIE はファームウェアのバイナリコードを LLVM を用いて中間言語に変換し，個々の関数やコンポーネント単位でプログラムを分割する．そして実行された全コマンドを抽出することで，バグや隠れたコマンドを発見する手法である [7]．機能の特定は，LLVM がコア機能のプログラムや複数のサーバを持つ複雑なプロトコルなどのサンプルから学習し，基本ブロック数，分岐数（if-then-else やループなど），条件文の数などで特徴づけて行う．PIE ではこの方法ではコントロールフローグラフ（CFG）とデータフローグラフ（DFG）も解析に使用する．ツールは非公開である．

FIRMADYNE

FIRMADYNE は QEMU エミュレータを使用した，ファームウェアの動的解析による脆弱性検出ツールである [5]．検出ターゲットは以下の 3 つが定義されている．(1）ファームウェアイメージの LAN インターフェースからアクセスできる Web ページの検出, (2）認証されていない SNMP 情報をすべて検出する Snmpwalk ツールを用いた SNMP（Simple Network Management Protocol）情報の検出, (3）既知の脆弱性（Metasploit フレームワークから取得した 60 件の脆弱性）FIRMADYNE はツールが公開されている [4]．しかし，ツールの環境構築は完了できず，サンプル実行も不可能であった．理由としては，マニュアルとライブラリの依存関係のメンテナンス不足で，マニュアルに記載されたライブラリがインストールできず，環境構築が完了できなかったためである．

Stringer

Thomas らは，ファームウェアに埋め込まれたパスワード文字列を発見する Stringer を提案した [25]．この方法は静的文字列の比較関数に重み付けをし，重みの大きい関数をバックドアのパスワード候補と判断する．重みは各関数に到達するために必要な静的データ列や分岐の数などから算出し，静的データが関数の分岐に与える影響度を測定する．実験では 30 社が提供する 2,451,532 個のバイナリのデータに手法を適用し，未知のバックドアを 3 つ発見した．Stringer はツールが公開されている [23]．ツールはインストールできたが，サンプル実行は不可能であった．理由としては，マニュアルの記載された入力データのファ

イル形式が不明確で，用意すべきファイルが不明なためである．また，Stringer の動作には SRE ツールの IDA Pro 6.8 以上が必要であり，IDA Pro の有料ライセンスが別途必要となる．マニュアルには IDA Pro との連携方法は記載されていないため，解析途中で連携エラーが発生する可能性がある．

HumIDIFy

Thomas らは半教師あり学習の分類器を用いてバックドアを検出する HumIDIFy を提案した [24]．ファームウェアのシンボル情報を収集して関数の機能ごとにラベル付けし，サポートベクターマシンでバックドア検出モデルを作成した．関数の機能は似ているにもかかわらず，正解データとの差分が大きいと判定された関数はバックドアの対象として検出される．実験では Tenda ルーターのファームウェアでバックドアを発見した．HumIDIFy はツールが公開されている [22]．ツールはインストールできたが，サンプル実行は不可能であった．理由としては，マニュアルの記載が不十分で，サンプル実行例が記載されていないためである．

D-Taint

D-Taint は，FIRMADYNE を応用してプログラムのテイント解析をして脆弱性を検出する手法である [6]．テイント解析では，入力元が特定できない（信頼できない）値とその値がプログラムに与える影響が追跡できる．D-Taint では，strcpy()，memcpy()，system() などのファイルやネットワークに関するシンボルや，ループバッファのコピーなどの安全でないコードパターンとデータパスを追跡することで脆弱性の検出を試みる．ツールは非公開である．

Firmup

Firmup は，共通脆弱性識別子（CVE）に登録されたプログラムに類似するファームウェアプログラムを検出する手法である [9]．この方法はファームウェアをアセンブリ解析し，バイナリの類似度で脆弱性を検出する．ツールは非公開である．

FirmFuzz

FirmFuzz は，QEMU ベースのエミュレータを使用して，Linux ベースの IoT ファームウェアを分析する動的解析手法である [21]．まずファームウェアに付属している Web アプリのユーザ名とパスワードを収集し，偽のデバイスドライバを利用して，Web アプリからの攻撃や脆弱性を検出する．検出する脆弱性は 4 つで（1）コマンドインジェクション,（2）バッファオーバーフロー,（3）XSS に関する脆弱性,（4）NULL ポインタである．FirmFuzz はツールが公開されている [20]．ツールはインストールが完了できず，サンプル実行は不可能であった．理由としては，セットアップスクリプト内のライブラリが見つからず，ビルドエラーが発生したためである．

John et al.

John らは，グラフ畳み込みニューラルネットワーク（GCN）を利用した Android のマルウェア検出方法を提案した [12]．ネットワーク関連のシステムコールはマルウェアによく利用される点に着目し，システムコール，ソケットコール（read/send）に関するプログラムを重点的に解析した．システムコールの流れでグラフを構築し，関連するプログラムのフローグラフをマルウェアの候補として検出する．ツールは非公開である．

Karonte

Karonte は，ファームウェア内のバイナリ同士のプロセス間通信に着目した脆弱性検出システムである [18]．Karonte は，近年の静的解析手法では，最も多くの未知の脆弱性を検出した手法である [27] [10]．バイナリ単位でフローグラフを構築し，データ通信とメモリ位置でテイント解析で行うことで，バイナリ間のデータ通信を監視して不安全なデータの検出を試みる手法である．監視ターゲットとなるのは，ファイル，共有メモリ，環境変数，ソケット通信，コマンド引数などである．Karonte はツールが公開されている [17]．ツールは Docker で配布されており，インストールは用意であった．しかし，Karonte の Docker コンテナを起動させる際，一般的な Docker コマンドでは起動せず，起動オプションを指定する必要があった．このオプションはマニュアルには記載されておらず，エラー対応に時間を要した．また，Karonte は，論文の機能紹介と実際の機能に差分があり，非搭載の機能

はユーザーが手作業で完了しなければならず，作業時間が長くなることがわかった．非搭載の機能の具体的な内容としては，論文では，ツールの入力としてファームウェア本体をそのまま入力にできると説明している．しかし，実際に受付ける入力は，逆解析に必要なファームウェアの情報をまとめた JSON 形式のファイルであった．逆解析に必要な情報をまとめる作業，つまり前処理作業は，ユーザーがファームウェアを分解して構成ファイルを取り出したり，解析始点となるファームウェアのベースアドレスを調査しなければならない．この前処理作業は，専門知識を要する外部ツールの使用が必須であり，作業の複雑さに起因する工数が高い．

3.2.1　まとめ

10 件の既存研究のうち，検出ツールを公開してる研究は 5 件，更に公式マニュアル通りにサンプル実行できた検出ツールは 1 件であった．4 件の検出ツールがインストールまたはサンプルの実行が不可だった理由としては，1 件が実行方法の未記載，1 件が不明瞭な入力データ，2 件が使用ライブラリのインストールエラーであった．

実行方法の未記載と不明瞭な入力データは，マニュアルを整備することで解決可能である．使用ライブラリのインストールエラーについては，使用ライブラリを含めた仮想環境を提供することで解決可能である．

サンプル実行ができた Karonte においても，起動と前処理工程において作業工数の負荷を確認した．具体的には，Karonte の Docker コンテナの起動に特殊なオプションが必要な場合があるが，マニュアルには記載されておらず，環境構築の作業負荷があった．他には，サンプル以外のファームウェア解析では，前処理として設定ファイルを作成しなければならず，作業工数の負荷を確認した．この場合，Docker コンテナの起動の複雑さについては，マニュアルの整備で解決可能である．前処理の複雑さについては，検出ツール内で前処理を自動化し，ユーザーの手作業を減らすことで解決可能である．

3.3 既存の脆弱性検出ツールの課題整理

Karonte における課題は，前処理の複雑さである．Karonte で任意のファームウェアを解析する場合は設定ファイルを作成する必要がある．設定ファイルの作成には，Karonte 以外の外部ツールを使用してファームウェアを分割したり，ファームウェアの解析アドレスを特定する必要があり，設定ファイルづくりの前処理工程は複雑であった．

他にも，Karonte を含む既存の検出ツールの共通課題は 2 つある．1 つ目は，環境構築の複雑さである．著者らの調査で，既存の解析ツールの多くはマニュアルやツールのメンテナンス不足でインストールが完了できないことが確認されている [5,21,24,25]．既存の解析ツールのうち，正常にインストールが完了し，解析実行できたのは Karonte のみであった．

環境構築の複雑さは，不便さのみならず，網羅的な脆弱性解析の妨げにつながる．なぜならば，各ツールが検出対象とする脆弱性は限られているため，網羅的な脆弱性解析には複数のツールを併用する必要があるからである．例えば，Karonte の検出対象はバッファオーバーフローと DDos 攻撃の 2 種類である．そのため，IoT の脆弱性ランキングの 1 位「ログイン情報の埋め込み」を検出対象とする場合，Karonte 以外の解析ツールの使用が必須である [11]．

2 つ目は，解析時間が長いことである．既存の解析ツールの解析時間は長く，ファイルサイズの大きいファームウェアの解析は 1 日以上かかる場合もある [18,19]．通常，1 つのファームウェアには数百〜数千の構成ファイルが含まれており，その 1 つずつに脆弱性検出アルゴリズムを適用して脆弱性を検出する．解析時間の長時間化は，大量の構成ファイルに影響するため，解析時間を削減する工夫が必要である．

第4章　埋め込みログイン情報の検出手法の提案

本章では，埋め込みログイン情報の検出手法を提案する．まず，4.1 節で提案手法の概要を説明する．そして，4.2 節，4.3 節，および 4.4 節で各手法の具体的な説明する．

4.1　概要

本手法では，IoT 機器のファームウェアに含まれる埋め込みログイン情報を検出するため，3 つの手法を提案する．まず，1 つ目の Stringe Search では，埋め込みログイン情報によく使用される文字列比較関数に着目し，文字列比較関数を使用する関数の行を埋め込みログイン情報の候補とする手法を提案する．2 つ目の Socket Search では，遠隔ログインはネットワーク機能関数を経由することを踏まえ，`socket` シンボルを参照している関数の周辺で文字列比較関数を使用する関数の行を埋め込みログイン情報の候補とする手法を提案する．3 つ目の User Input Search では，文字列比較関数の引数となる変数がユーザーの入力値である場合，その文字列比較を参照している行を埋め込みログイン情報の候補として出力する手法を提案する．

いずれの提案手法も，文字列比較関数で使用されている文字列に着目している．各手法の違いは解析範囲である．String Search の解析範囲は，文字列比較関数を使用する関数である．Socket Search と User Input Search は，String Search の部分集合の関係である．Socket Search の解析範囲は，String Search の解析範囲のうち，ネットワーク関数とその周辺関数のみである．User Input Search の解析範囲は，String Search の解析範囲のうち，文字列比較関数でユーザが入力した値が使用されている場合のみである．

4.2 String Search の提案

本節では String Search を提案する．String Search は，文字列比較関数である strcmp または strncmp シンボルを参照する関数を検出し，文字列比較関数が使用されている行を埋め込みログイン情報の候補として出力する手法である．

4.2.1 String Search のアルゴリズム

Algorithm 1 Main Program of String Search

```
1: SymbolTable ← getSymbolTable()
2: while SymbolTable.hasNext() do
3:     Symbol ← SymbolTable.getSymbol()
4:     if Symbol.isMatched(strn?cmp) then
5:         SymAddress ← Symbol.getAddress()
6:         SymFunction ← getFunctionFromAddr(SymAddress)
7:         FunctionList ← getReferenceFunctions(SymFunction)
8:         for ChildFunction ∈ FunctionList do
9:             printHardCoded(ChildFunction)
10:        end for
11:    end if
12: end while
```

Algorithm 2 printHardCoded function

```
1: result ← getDecompiledFunction(Function)
2: lines ← result.eachLine.matches(".*strn?cmp.*")).toList()
3: if lines.size() > 0 then
4:     lines.forEach(line− > println(line))
5: end if
```

String Search のメインアルゴリズムを Algorithm 1 に示す．メインプログラム (Algorithm 1) では，IoT 機器のファームウェアのシンボルテーブルを読み込み，その中に strcmp または strncmp のシンボルがあるかどうかをチェックする (4 行目)．そして，シンボルが含まれていれば、strcmp または strncmp シンボルのアドレスを取得し，関数情報を取得する（5-6 行目）．続いて，取得した関数情報を使って，strcmp または strncmp シンボルが参照する

関数リストを作る（7 行目）．そして，その関数はリストを埋め込みログイン情報の候補として出力する（9 行目）．

Algorithm 2 の printHardCoded 関数では，メインアルゴリズムの 9 行目で使用される．この関数は，引数として渡された関数プログラムで埋め込みログイン情報を含むプログラム行のみを出力する．

4.3 Socket Search の提案

本節では Socket Search を提案する．Socket Search は，`socket` 関数の周辺で文字列比較関数を使用している関数と該当する行を見つける手法である．埋め込みログイン情報を知り得た第三者が IoT 機器にログインする場合，インターネット経由でのアクセスが考えられる．つまり，TCP/UDP などを用いたネットワーク機能と，埋め込みログイン情報を照合する関数は参照関係にある場合がある．そこで本手法は，ファームウェアのネットワーク機能関数を起点として，関数周辺の文字列比較関数を埋め込みログイン情報の候補として出力する．

図 4.1 に Socket Search の探索順序を示す．この図は `socket` シンボルのコールグラフである．各ノードは関数を意味する．矢印は関数間の参照関係を意味する．hop では探索範囲を設定する．図では hop が 2 なので，起点から 2 つ先の関数までを範囲とする．探索が進むと hop が 1 つ減る．そして，hop がゼロになるまでを続ける．探索例として，関数 A は `socket` シンボルを参照しているので，Socket Search は関数 A から探索を開始する．この例では hop が 2 なので，探索起点から 2 つ先の関数までを探索範囲とする．2 つ先の関数はオレンジ色と緑のノードである．探索の結果，関数 B で文字列比較関数が使用されていたため，関数 B を埋め込みログイン情報の候補として出力する．

4.3.1 Socket Search のアルゴリズム

Socket Search のアルゴリズムを説明する．Algorithm 3 にメインプログラムを示す．まず，`Hop` の標準値を 5 に設定する（1 行目）．メインプログラム実行時に引数で深さの値が与えられた場合，`Hop` の値を書き換える．次に，ファームウェアのシンボルテーブルを

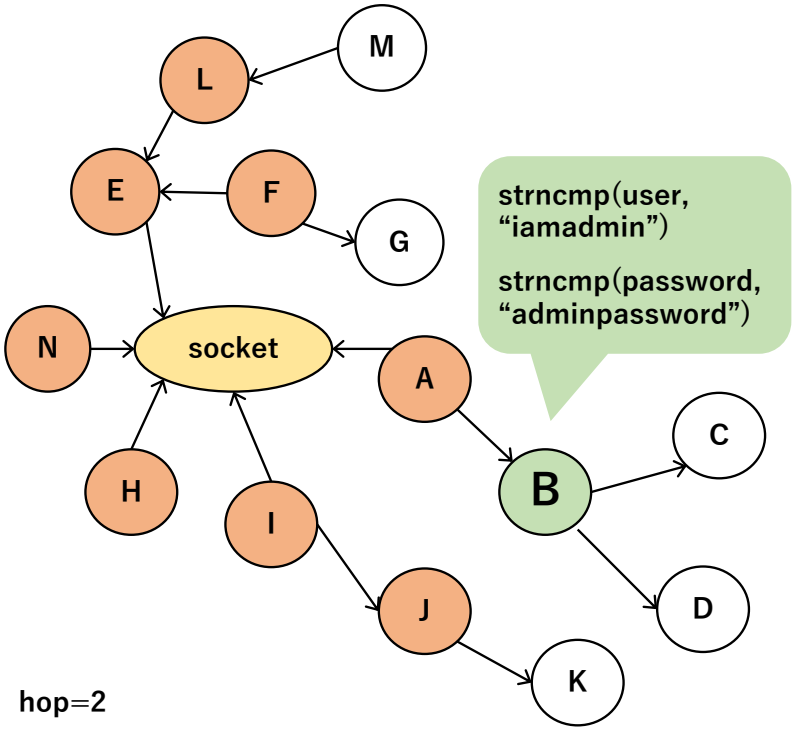

図 4.1: Socket Search の概要

読み込み，socket シンボルがシンボルテーブルにあるかチェックする（5 行目）．そして，socket シンボルがある場合は，シンボルアドレスを取得する．socket シンボルを使用している関数をネットワーク関数と定義し，ネットワーク関数の情報を取得する（6 行目-7 行目）．次に，ネットワーク関数を参照する関数のリストを作成する（8 行目）．次の探索先を見つけるため，関数リストのそれぞれの関数を参照する関数を取得する（10 行目）．次に，関数リストの関数内で文字列比較関数が使用されている場合はその箇所を出力し，次の探索に進む (11 行目).

Algorithm 3 Main Program of Socket Search

```
 1: Hop ← 5
 2: SymbolTable ← getSymbolTable()
 3: while SymbolTable.hasNext() do
 4:     Symbol ← SymbolTable.getSymbol()
 5:     if Symbol.isSocket then
 6:         SymAddress ← Symbol.getAddress()
 7:         SymFunction ← getFunctionFromAddr(SymAddress)
 8:         FunctionList ← getReferenceFunctions(SymFunction)
 9:         for ChildFunction ∈ FunctionList do
10:             Incoming ← getIncomingCalls(ChildFunction)
11:             printReference(Incoming, Hop)
12:         end for
13:     end if
14: end while
```

Algorithm 4 の printReference 関数では，引数で渡された関数内で文字列比較関数を使用している部分を出力する．更に，次の探索を実行する．

Algorithm 4 printReference Function (Function, Hop)

```
1: FunctionList ← getReferenceFuncsFrom(Function)
2: for Function ∈ FunctionList do
3:     printHardCoded(Function)
4:     printIncomingCalls(Function, Hop)
5:     printOutgoingCalls(Function, Hop)
6: end for
```

Algorithm 5 printIncoming(Outgoing)Calls function (Function, Hop)

```
1: if Hop == 0 then
2:     ↵ false
3: else
4:     Hop− = 1
5: end if
6: FunctionList ← getReferenceFuncsFrom(Function)
7: for Function ∈ FunctionList do
8:     printHardCoded(Function)
9: end for
10: ↵ true
```

Algorithm 6 printHardCoded function

```
1: result ← getDecompiledFunction(Function)
2: lines ← result.eachLine.matches(". ∗ strn?cmp. ∗ ")).toList()
3: if lines.size() > 0 then
4:     lines.forEach(line− > println(line))
5: end if
```

printIncomingCalls 関数は，引数で渡された関数が参照する関数に文字列比較関数が含まれているかを確認する関数である．Algorithm 5 にアルゴリズムを示す．関数リストを printHardCoded 関数（Algorithm 6）に渡し，printHardCoded 関数は文字列比較関数を使用しているプログラムを出力する．printOutgoingCalls 関数は printIncomingCalls 関数と同じ働きをするが，参照の方向が逆となる．

4.4　User Input Search の提案

本節では，User Input Search を提案する．埋め込みログイン情報はユーザーが入力した値と比較される．そこで，ユーザー入力値が文字列比較関数で使用されている場合，その関数を使用する行を埋め込みログイン情報の候補として出力する手法である．

4.4.1 ユーザー入力値の例

通常，ファームウェアはユーザーが入力した値をメモリ，スタック，またはヒープに格納する．図 4.2 に，実際のファームウェアでユーザー入力値が格納されている例を示す．ユーザーが入力した値はアドレス `0x58d4e0` に格納され，`uStack4` にユーザー入力値が格納されている．

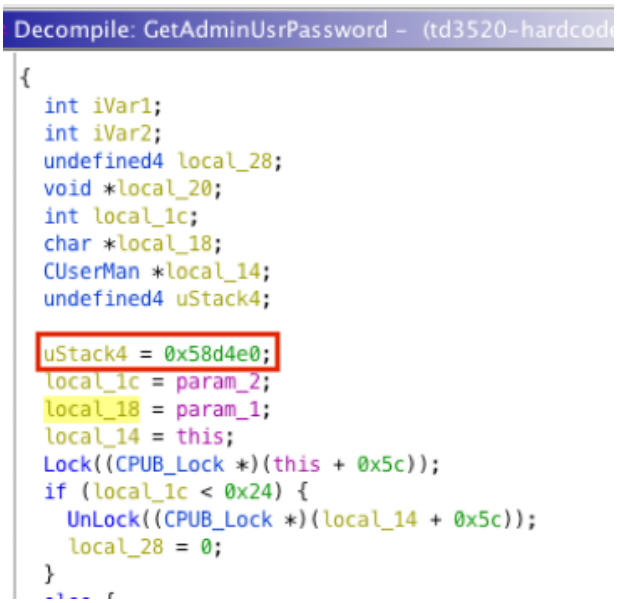

図 4.2: ユーザー入力値の例

4.4.2 User Input Search の探索フロー

図 6.1 に User Input Search の概要を示す．ファームウェアのプログラムを入力とし，`strcmp` または `strncmp` シンボルを参照する関数のリストを作成する．次に，関数リストの関数を一つずつ確認し，文字列比較関数を使用している行を抽出する．そしてその中から，文字列比較の引数の組み合わせが，変数と埋め込み文字列であるものを絞り込む．例えば，`strcmp("apple", param_1)` のように，埋め込み文字列と変数が引数となっているケースと指す．その後，引数に使用されている変数がユーザー入力値であるかどうか，代入関係を遡る．最後に，変数がユーザー入力値であると判別できた場合，その値を埋め込みログイン情報の候補として出力する．

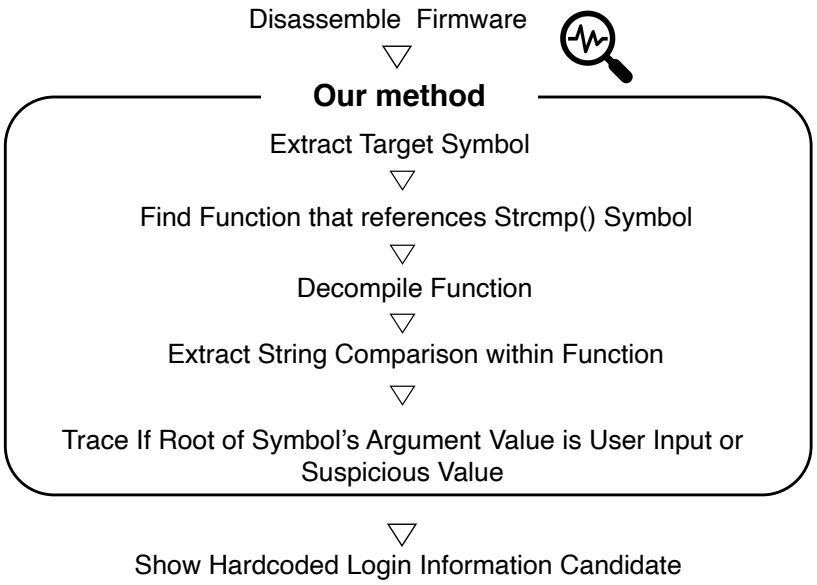

図 4.3: User Input Search の概要

4.4.3　User Input Search のアルゴリズム

Step 1. 文字列比較関数を使用する関数の抽出

本ステップでは，strcmp または strncmp シンボルを参照する関数のリストを作成する（図 4.4 のステップ 1）．Algorithm 7 に step1 のコードを示す.

ファームウェアのシンボルテーブルを読み込み，その中に strcmp または strncmp のシンボルがあるかどうかをチェックする（Algorithm 7 の 4 行目）．シンボルが含まれていれば、strcmp または strncmp シンボルのアドレスを取得し，関数情報を得る（5 行目-6 行目）．この関数情報を使用して，strcmp または strncmp 関数が参照している関数リストを作成する（7 行目）．そして，作成したリストの関数を次のステップで解析する（9 行目）.

Steps 2 及び 3. 逆解析とユーザー入力値の特定

step2 のアルゴリズムを Algorithm 8 に示す．step1 の関数リストから関数を一つずつ解析する．そのうち，埋め込みログイン情報の候補となりうる文字列比較関数を含む関数の行を抽出する.（7 行目）．文字列比較関数のうち，下記すべての条件を満たすものを候補と

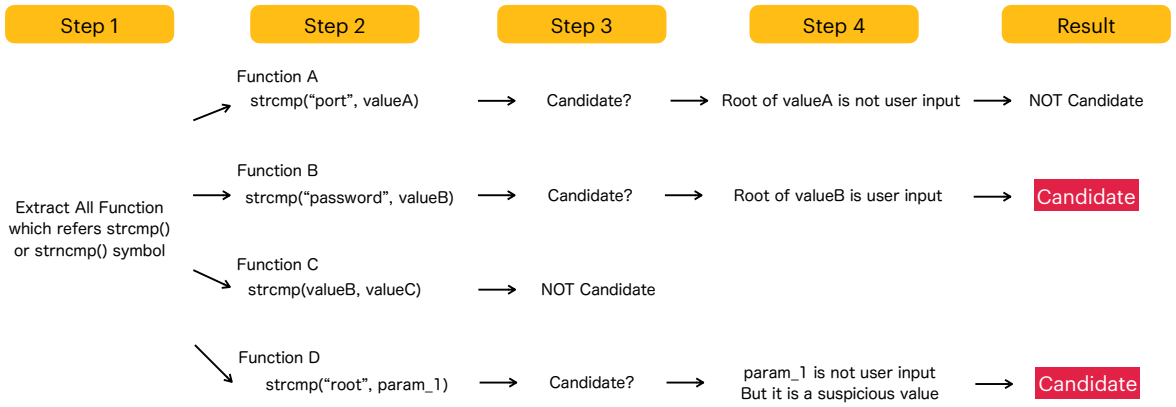

図 4.4: User Input Search の解析の流れ

する．

(1) `strcmp` または `strncmp` シンボルの引数が，変数と文字列の組み合わせである (8 行目)．例えば，図 4.4 の関数 A，B，および D は条件を満たす．それぞれ，`"port"`，`password` および `root` は埋め込み文字列，`valueA`，`valueB` および `param_1` は変数である．一方，関数 C は引数がいずれも変数であるため候補からは外れる．

(2) 変数名が，‘param‘，‘stack‘，‘DAT‘および ‘local‘のいずれを含むものであること (13 行目)．SRE ツールでは，メモリに関する変数名には param, stack, DAT, local のいずれかが自動 で付与されるため，これらの変数名はユーザー入力の特徴とみなし，候補として次のステップで解析する．

(3) 埋め込み文字列の長さが 4 より大きいこと (18 行目)．埋め込みパスワードは数字，単語や複数の文字列が混在しているため，単語の長さが 4 文字以上となる場合が多い．例えば,「EN」や「JP」のような環境変数は候補から外れる．

これらすべての条件を満たす場合，該当する行は埋め込みログイン情報の候補リストに追加される (21 行目)．例外として，(1) と (3) にマッチするが (2) にマッチしない場合，次のステップで候補レベルを確認するため，一時的な候補リストに追加される (29 行目)．これは，ユーザー入力値の変数名が (2) の特徴を持たない場合も稀にあるためである．

Algorithm 7 Main program of User Input Search

```
 1: SymbolTable ← getSymbolTable()
 2: while SymbolTable.hasNext() do
 3:     Symbol ← SymbolTable.getSymbol()
 4:     if Symbol.isMatched(strn?cmp) then
 5:         SymAddress ← Symbol.getAddress()
 6:         SymFunction ← getFunctionFromAddr(SymAddress)
 7:         FunctionList ← getReferenceFunctions(SymFunction)
 8:         for ChildFunction ∈ FunctionList do
 9:             decompileFunction(ChildFunction)
10:         end for
11:     end if
12: end while
```

Step 4. ユーザー入力値の判定

本ステップでは，埋め込みログイン情報の候補が実際にユーザー入力値であるかを確認する．アルゴリズムを Algorithm9 に示す．step3 で候補となったの参照関係を遡り，変数の値が代入される行を見つけ，ユーザー入力の値であるかを確かめる．5-6 行目では，変数と値を分割する．`right` は数式の右側，`left` は数式の左側を表す．例えば，`valueA = "password"` という行の場合，`right` は`"password"`，`left` は `valueA` となる．

もし `right` の値に “memset” または “0x” が含まれていれば，その値がユーザーの入力値であると判定し，埋め込みログイン情報の候補とする（9-12 行目）．図 4.2 にあるように，ユーザーの入力値はメモリアドレスから変数に格納されるので，11 行目で「memset」か「0x」が右辺に含まれるか確認する．0x はメモリアドレスの頭文字に用いられる． memset はメモリ領域を指定した文字で埋める関数である．もし `right` が param や他の変数を含んでいたら，さらにその参照元を遡り，前述のフローを繰り返す．この再帰処理は decompileFunctionRecursive() で実行される (13–17 行目)．Algorithm 10 に decompileFunctionRecursive() を示す．

Algorithm 8 decompileFunction(Function f)

```
1: Stream < String > resLines = f.getDecompiledFunction().getC().lines()
2: List < String > decompiled = resLines.collect(Collectors.toList())
3: List < String > resultStrncmp = newArrayList < String > ()
4: List < String > resultStrncmpEtc = newArrayList < String > ()
5: booleanfound = false
6: Stringmatched = ""
7: for Stringstr : decompiled do
8:     Stringregex = ".*strn?cmp(([']".*["'], [[^"']".*[[^"']|[[^"']".*[[^"'], ['"].*["'])).*"
9:     Patternp = Pattern.compile(regex)
10:    Matcherm = p.matcher(str)
11:    if m.find() then
12:        Stringmatchstr = m.group()
13:        if m.group(1).contains("param"||"Stack"||"DAT"||"local") then
14:            Stringregex2 = "['"].*["']"
15:            Patternp2 = Pattern.compile(regex2)
16:            Matcherm2EmbeddedString = p2.matcher(m.group(1))
17:            if m2EmbeddedString.find() then
18:                if m2EmbeddedString.group(0).length() > 4 then
19:                    matched = m.group(1)
20:                    if !resultStrncmp.contains(var) then
21:                        resultStrncmp.add(var)
22:                        found = true
23:                    end if
24:                end if
25:            end if
26:        else
27:            Stringvar = m.group(1)
28:            if !resultStrncmpEtc.contains(var) then
29:                resultStrncmpEtc.add(var)
30:            end if
31:        end if
32:    end if
33: end for
34: for Stringvar : resultStrncmpEtc do
35:    rootValue = checkParentValue(var, decompiled, function)
36: end for
```

Algorithm 9 checkParentValue (String var, List(String) decompiled, Function f)

```
1: for Stringline : decompiled do
2:     Stringright = ""
3:     Stringleft = ""
4:     if var.length() > 2&&line.matches("[^(] * " + var + ".* = . * ") then
5:         right = getVariableAndContent(line, true)
6:         left = getVariableAndContent(line, false)
7:         if right.length() > 0 then
8:             addCondidateCount()
9:             if right.contains("memset") then
10:                returnright
11:            else if left.contains("uStack")&&right.contains("0x") then
12:                returnright
13:            else if right.matches("param_d") then
14:                StringchildFunctionName = f.getName()
15:                intN = 0
16:                N = parseInt(right.replaceAll("param", ""))
17:                returndecompileFunctionRecursive(right, f.Name(), N)
18:            else
19:                returndecompileFunctionRecursive(right, f.Name(), 0)
20:            end if
21:            break
22:        else
23:            continue
24:        end if
25:    end if
26:    returnnull
27: end for
```

Algorithm 10 decompileFunctionRecursive(Function f, String childFunctionName, int childVar)

```
1: Stream < String > resLines = decompRes.getDecompiledFunction().getC().lines()
2: List < String > decompiled = resLines.collect(Collectors.toList()
3: List < String > resultStrncmp = newArrayList < String > ()
4: booleanfound = false
5: Stringmatched = ""
6: String[]params = newString[10]
7: StringvarToBackdoor = ""
8: for Stringstr : decompiled do
9:     if str.contains(childFunctionName) then
10:        params = str.split(",")
11:        var = params[childVar]
12:        Stringres = checkParentValue(var, decompiled, function)
13:    end if
14: end for
15: if found then
16:    addSearchedCount()
17: end if
18: returndecompRes
```

第5章　埋め込みログイン情報の検出手法の評価

本章では，埋め込みログイン情報の検出手法の評価を行う．まず，5.1 節で評価の概要について説明する．続いて，5.2 節で評価と考察を行う．最後に，5.3 節で手法のまとめと今後を述べる．

5.1　評価の概要

提案した 3 手法を，実際に IoT 機器のファームウェアに適用し，脆弱性の検出能力を確かめる．本評価では，埋め込みログイン情報が報告されている 6 つの IoT 機器のファームウェア解析対象とする．表 5.1 に，解析対象の機器名と埋め込みログイン情報を示す．

3 つの提案手法の他に，ベースラインとして Whole Search を採用した．Whole Search では，すべての関数を探索し，`strcmp` または `strncmp` シンボルを使用している関数の行を見つける．他の 3 つの提案手法と異なるのは探索開始点の有無である．Whole Search は探索開始点を持たず，すべての関数を一つずつ探索する．String Search は，探索開始点が `strcmp` または `strncmp` シンボルとなる．Socket Search は `socket` が探索開始点である．User Input Search は文字列比較関数に含まれる変数が探索開始点となる．

提案手法の実装手法と評価環境

各提案手法は，Ghidra のプラグイン (Ver.9.1) として開発した．Ghidra は，National Security Agency が開発した，オープンソースのリバースエンジニアリングソフトウェアである [13]．プラグインは，Java SE Development Kit 11 (JDK 11) で開発した．評価に使用したコンピュータは，64 ビットの Windows 10、3.60GHz の Intel Core i7-7700、8GB の RAM を搭載した Windows PC である．

表 5.1: 検出対象の埋め込みログイン情報

機器名	埋め込みログイン情報
D-Link Router	`strcmp(ua, "xmlset_roodkcableoj28840ybtide")`
Q-See DVR	`strcmp("6036huanyuan",password)`
TRENDnet Router	`strcmp("emptyuserrrrrrrrrrrr", password)`
Tenda Router	`strcmp("w302r_mfg",packet->magic)`
TCP32764 Router	`"ScMM"`
Ray Sharp DVR	`strcmp("519070",password)`
Ray Sharp DVR	`strcmp("664255",password)`

5.1.1 評価指標

本手法では，評価指標として，Accuracy，Precision，Recall そして F-Score を採用した．

Accuracy は，手法の精度をあらわす.（式 5.1）

Precision（適合率）は手法の正確性をあらわす．手法が検出した埋め込みログイン情報候補が，実際に埋め込みログイン情報である割合である．埋め込みログイン情報候補とは，各関数の文字列比較関数を含むプログラム行のうち，各手法が埋め込みログイン情報と判定したものである.（式 5.2）

Recall（再現率）は手法の網羅性を示す．これらの値は，埋め込みログイン情報を本手法が検出できた割合を示す.（式 5.3）

F 値は，正確性と網羅性の総合的な評価を示すものである．F 値は，適合率と再現率の調和平均である.（式 5.4）

表 5.2 に各指標の計算に使用する値を示す．真陽性 (TP) は，埋め込みログイン情報候補のうち，実際に埋め込みログイン情報であった数を示す．偽陽性（FP）は，埋め込みログイン情報候補のうち，埋め込みログイン情報ではないものの数を示す．偽陰性（FN）は，手法は検出しなかったが，実際は埋め込みログイン情報であったものの数を示す．真陰性（TN）は，手法が埋め込みログイン情報候補と判定しなかったもののうち，実際に埋め込みログイン情報ではないものの数を示す.

表 5.2: 評価指標

	True Candidate	False Candidate
Method Found	True Positive (TP)	False Positive (FP)
Method Missed	False Negative (FN)	True Negative (TN)

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

$$F - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5.4}$$

5.2　評価結果と考察

Whole Search

Whole Search はファームウェア内のすべての関数を一つずつ探索し，`strcmp` または `strncmp` シンボルを使用している関数の行を埋め込みログイン情報の候補として出力する手法である．表 5.3 に結果を示す．検出能力は，7 件の埋め込みログイン情報のうち 4 件を発見した．TCP32764 Router は，文字列比較関数で値が使用されていなかったため検出できなかった．TCP32764 Router は，TCP 32764 番で特定文字列を送ることで，外から端末の操作が可能となる脆弱性である．このライブラリは多くのルーターに含まれており，社会的被害が大きい脆弱性であった．埋め込みログイン情報ではないが，特定の文字列が埋め込まれているという観点から，本手法および既存研究で検出できるかを確かめるため，今回採用した．Ray Sharp DVR は，文字列比較関数を使用しているが検出できなかった．理由

表 5.3: Whole Search（ベースライン）の結果

機器名	サイズ (KB)	解析時間 (分)	検出
D-Link Router	619	2.43	yes
Q-See DVR	7200	38.83	yes
Trendnet Router	318	1.45	yes
Tenda Router	566	1.83	yes
TCP32764 Router	18	0.11	no
Ray Sharp DVR	4900	16.31	no

表 5.4: String Search の結果

機器名	時間 (秒)	検出	TP	FP	FN	TN	Accuracy	Precision	Recall	F-score
D-Link Router	38	Yes	1	44	0	1576	0.973	0.0222	1.0	0.0435
Q-See DVR	278	Yes	1	102	0	18465	0.995	0.0097	1.0	0.0192
Trendnet Router	79	Yes	1	83	0	532	0.865	0.0119	1.0	0.0235
Tenda Router	37	Yes	1	63	0	1314	0.954	0.0156	1.0	0.0308
TCP32764 Router	3	No	0	2	1	132	0.978	0.0	0.0	N/A
Ray Sharp DVR	113	No	0	262	2	12913	0.980	0.0	0.0	N/A

としては，関数の数が膨大であるため，Ghidra による解析漏れが考えられる．解析漏れの有無は，文字列比較関数をリストアップする段階で判別できるが，後述する他の検出手法ではリストアップの段階で該当関数が含まれていることが確認できたため，文字列比較関数をリストアップするプログラムは正常に動作している．解析時間に関しては，ファームウェアのサイズが 4MB を超えると解析時間が 10 分以上かかることが確認された．ファームウェアサイズが最も大きい Q-see DVR の解析完了時間は 38 分であった．

String Search

String Search は，`strcmp` と `strncmp` のシンボルを使用する関数を抽出し，`strcmp` と `strncmp` シンボルを使用している行を，埋め込みログイン情報の候補として出力する手法である．表 5.4 に結果を示す．検出能力は 7 件中 4 件と Whole Search と同等であった．解析時間，ファームウェアのサイズにもよるが，全てのファームウェアの解析時間は 3 秒-5

表 5.5: Socket Search の結果

機器名	時間 (秒)	検出	TP	FP	FN	TN	Accuracy	Precision	Recall	F-score	深さ
D-Link Router	79	Yes	1	44	0	1576	0.973	0.0222	1.0	0.0435	4
Q-See DVR	402	Yes	1	97	0	18470	0.995	0.0102	1.0	0.0202	4
Trendnet Router	50	Yes	1	83	0	532	0.865	0.0119	1.0	0.0235	4
Tenda Router	14	Yes	1	4	0	1373	0.997	0.20	1.0	0.3333	1
TCP32764 Router	1	No	0	2	1	132	0.978	0.0	0.0 N/A	5	
Ray Sharp DVR	303	No	0	262	2	12913	0.980	0.0	0.0	N/A	5

深さは埋め込みログイン情報を検出した時点の探索の深さを示す

表 5.6: User Input Search の結果

機器名	検出	TP	FP	FN	TN	Accuracy	Precision	Recall	F-score
D-Link Router	Yes	1	18	0	1602	0.989	0.053	1.0	0.100
Q-See DVR	Yes	1	39	0	18528	0.998	0.025	1.0	0.048
Trendnet Router	Yes	1	33	0	582	0.946	0.029	1.0	0.057
Tenda Router	No	0	10	1	1367	0.992	0.0	0.0	N/A
TCP32764 Router	No	0	0	1	134	0.993	N/A	0.0	N/A
Ray Sharp DVR	Yes	2	39	0	13136	0.997	0.049	1.0	0.093

分以内となり，Whole Search よりも短縮できた．探索対象を strcmp と strncmp シンボルを使用する関数のみに絞り込むことで，解析時間を短縮することができた．

Socket Search

Socket Search は，socket シンボルを使用するネットワーク関数を抽出し，その関数を参照する関数のうち strcmp または strncmp シンボルを使用する関数の行を埋め込みログイン情報の候補として出力する．本評価では探索範囲を指定する hop を 5 とした．

結果を表 5.5 に示す．検出能力は 7 件中 4 件であった．埋め込みログイン情報の候補数は String Search よりも少なく，より少ない候補から埋め込みログイン情報を発見できた．Socket Search は Tenda Router に対して有効であった．探索範囲の値が 1 の段階で埋め込みログイン情報を発見した．これは，ネットワーク関数の周辺に埋め込みログイン情報が存在することを示す．一方，他の 3 件は深さ 4 で発見されたため，候補数が多くなった．

User Input Search

Decompile: FUN_000a8028 - (raysharp_dvr-hardcoded-backdoor)

```
83      }
84    }
85    sprintf(acStack1104,"http://%s:%d/%d?trans=tcp&action=play&media=%s&username=%s&password=%s",
86            (char *)&local_8,param_7,param_8,(char *)&local_490,&stack0x00000010,&stack0x00000030);
87    local_4c0 = 0xffffffff;
88    local_4c8 = FUN_00138b54(*(void **)(param_1 + 0xc),acStack1104,(uint)*(byte *)(param_1 + 8),
89                             &LAB_000a87f4,&LAB_000a7f14,&LAB_000a7f1c,&local_50);
90    if (local_4c8 == (char *)0x0) {
91      *in_stack_00000060 = local_50;
92      *(undefined4 *)(param_1 + 0x9605c) = 0;
93      if (local_44 != 1) {
94        puVar6 = in_stack_00000060;
95      }
```

図 5.1: User Input Search が発見したログイン情報を含む平文通信

User Input Search は，文字列比較関数の引数に使用されている変数がユーザー入力値であった場合，その関数が使用されている行を埋め込みログイン情報の候補として出力する手法である．結果を表 5.6 に示す．検出能力については，7 件中 5 件検出した．Tenda Router が検出できなかった理由として，変数が構造体の packet->magic であったため，検出対象

表 5.7: 提案手法と既存研究の検出能力の比較

機器名	Socket Search + User Input Search	Stringer
D-Link Router	1	0
Q-See DVR	1	1
Trendnet Router	1	1
Tenda Router	1	0
TCP32764 Router	0	0
Ray Sharp DVR	2	1
Total	6	3

に含まれなかったためである．Tenda Router のように，引数が構造体である場合など変数の種類によって検出漏れが生じることがわかった．今後は多様な変数を検出対象にできるように，手法を改善する必要がある．しかし，他の 3 手法が発見できなかった，Ray Sharp DVR の埋め込みログイン情報を検出した．

また，他の 3 手法よりも少ない候補数から埋め込みログイン情報を検出でき，F-score の値は最も高くなった．検出できたケースで Accuracy が 100 %でない理由としては，False Positive が多く，埋め込みログイン情報ではないものが候補として出力されたためである．その他，検出結果を手動で調査したところ，Ray Sharp DVR に ID とパスワードを平文で通信するプログラムを発見した．これらの事例は未報告であり，本手法が初めての発見となる．図 5.1 に通信箇所を示す．FUN_000a8028() では，ユーザー名とパスワードが含まれた URL が使用されている．http 通信は暗号化されていないため，ネットワーク傍受によってユーザー名とパスワードが漏洩してしまう．例えば，各値をハッシュ化して送信するなどの工夫が必要である．

提案手法と既存研究の検出能力の比較

表 5.7 に，提案手法と既存研究の検出能力を比較した結果を示す．提案手法は評価で検出能力が高かった，Socket Search と User Input Search の和である．Thomas らによって提案された Stringer を比較対象とする [25]．Stringer は，静的文字列の比較関数に重み付けを

し，重みの大きい関数を埋め込みログイン情報の候補とする手法である．新規の脆弱性を3つ発見し，埋め込みログイン情報に関する手法においては，脆弱性の検知数が多い手法である．各検出数の合計数は，既存研究よりも提案手法のほうが多いことから，提案手法の検出能力の高さが示される．

5.3 まとめと今後

本手法では，IoT機器のファームウェアに含まれる埋め込みログイン情報を検出するため，3つの手法を提案した．まず，1つ目のStringe Searchでは，埋め込みログイン情報によく使用される文字列比較関数に着目し，文字列比較関数を使用する関数を埋め込みログイン情報の候補とする手法を提案した．2つ目のSocket Searchでは，遠隔ログインはネットワーク機能関数を経由することを踏まえ，`socket`シンボルを参照している関数の周辺で文字列比較関数を使用する関数を埋め込みログイン情報の候補とする手法を提案した．3つ目のUser Input Searchでは，文字列比較関数の引数となる変数がユーザーの入力値である場合，埋め込みログイン情報の候補として出力する手法を提案した．

各手法の検出能力を確かめるため，実際に埋め込みログイン情報の脆弱性が報告されているIoT機器のファームウェア6件を収集し，ベースラインのWhole Searchを含む4手法に適用した．Socket Searchは，Tenda Routerにおいて最もF-scoreが高く，ネットワーク機能周辺に埋め込みログイン情報が含まれている場合は有用な手法であることがわかった．User Input Searchは，7件中5件のファームウェアにおいてF-scoreが最も高く，未知の脆弱性も発見したことから，全手法の中で最も有用な手法だといえる．

現状は，User Input SearchとSocket Searchを併用することで，高い精度と検出能力を維持できるといえる．今後は，User Input Searchが対象とする変数の種類を増やし，検出精度の向上につとめる．

第6章　埋め込みログイン情報検出ツールの提案と構築

本章では，既存の脆弱性検出ツールの課題を解決する，埋め込みログイン情報の検出ツールの提案と構築について述べる．まず，6.1 節で提案ツールの要件となる課題や関連要件についてまとめる．そして，6.2 節で，提案ツールの要件をまとめる．次に，6.3 節では，定義した要件をもとに提案ツールのソフトウェア設計について説明する．そして，6.4 節で，提案ツールの構築方法について述べる．

6.1　要件となる課題と関連要件の整理

6.1.1　既存ツールの課題解決のための要件

既存ツールでは，インストールエラー，高額なライブラリの必要性，そしてインストール・前処理・および解析の長時間化が課題となった．本手法では，これらを解決するべく，無料かつ短時間で脆弱性が検出できる検出ツールを提案する．利用対象者は，IoT セキュリティの研究者や企業などのビジネスユースだけではなく，IoT 機器の脆弱性解析に関心のある一般ユーザーとする．一般ユーザーも利用できるように，一般的なコンピュータにインストール可能な検出ツールを構築する．

また，各作業時間の短縮は，ソフトウェアのユーザビリティ向上にも役立つことが示されており，提案ツールではユーザビリティの向上も見込める．ユーザビリティとは，ユーザーがシステムなどを利用したときの使いやすさを示す．ユーザビリティを定義している国際標準機構の国際規格である ISO 9241-11 では，システムの「特定のユーザが特定の利用状況において，システム，又はサービスを利用する際に，効果，効率及び満足を伴って特定の目標を達成する度合い」について説明されており．そのうち，ユーザーが目標を達

成する際の正確さと完全さに関連して費やした資源が少ないことが，良いユーザビリティの指標となっている [3].

6.1.2　IoT 環境におけるミドルウェアの要件調査

IoT 脆弱性検出ツールの要件を定義した既存研究は，著者らの調査では見つかっていない．そこで，まずツールの下支えとなるミドルウェアの要件について既存研究を調査した．ミドルウェアとは，システムやハードウェアの複雑さを抽象化した上で，アプリケーションに共通するサービスを提供するものである [16]．本研究では，アプリケーションは脆弱性検出ツールを指す．本節ではミドルウェアの議論が活発かつ，類似分野である IoT のミドルウェア要件を調査した．次に，一般的なミドルウェアの定義と静的解析ツールの課題を踏まえた上で，提案ツールが備えるべき要件を定義する．

IoT を取り巻く環境のミドルウェア要件はこれまで議論されている [16,26,28]．Razzaque らは，61 件の IoT 環境のミドルウェアに関する研究を調査し，IoT 環境のミドルウェアが備えるべき要件を定義した [16]．網羅的な調査が評価され，現在でも IoT 環境のミドルウェア開発の手引として支持されている．しかし，Razzaque らの調査対象は 2014 年以前の研究であり，それ以降の研究や技術進歩による IoT 環境の変化も考慮した要件についても確認する必要がある．

Zhang らは，2021 年までの IoT 環境のミドルウェアの研究を調査対象としており，近年の IoT 環境に対応したミドルウェア要件を定義している [28]．Zhang らは，調査対象の研究の選出に厳格な調査基準を設け，WoS と ISI にインデックスされている査読付き論文誌や論文のみを調査対象とした．基準をクリアした 20 件の研究をもとに IoT 環境のミドルウェアの要件や実現技術をまとめた．他にも，Vikash らは，IoT 環境の実現に必須技術である WSN（Wireless Sensor Networks）のミドルウェアについて要件をまとめている [26].

これらの研究では，特に重要な IoT 環境のミドルウェアの要件として，導入の容易さ，リアルタイム性，相互運用性，軽量さ，スケーラビリティ，セキュリティとプライバシー，そして信頼性と可用性が重要と言及している．

導入の容易さは，ユーザーが専門知識やサポートなしに，容易にミドルウェアのインストールや使用ができることを示す．リアルタイム性は，アプリケーションやユーザーが要

求した処理を迅速に実行する能力を示す．相互運用性は，ミドルウェアが多様な IoT デバイスにサービスを提供する能力である．軽量さは，ミドルウェアが様々なデバイス上 で動作するように，コンピュータリソースの使用を抑える工夫や，ミドルウェアのファイルサイズを軽量化することである．スケーラビリティは，ミドルウェアがシステムの規模増大や機能追加に対応できる能力である．セキュリティとプライバシーは，ミドルウェアが保持する個人情報とユーザーのプライバシーを守ることや，ミドルウェアが保持する個人情報のコンテキストアウェアネスの開示することである．信頼性と可用性は，ミドルウェアが障害の発生時もサービスを提供する能力と停止時間を最小限にする能力である．

6.2 提案ツールの要件定義

既存ツールの課題と IoT 環境のミドルウェアの要件をもとに，提案ツールの要件を定義する．まず，6.1.2 節で述べた一般的なミドルウェアの定義より，提案ツールは，ユーザーがミドルウェアの複雑さを意識せず利用できる必要がある．このことから，既存の静的解析ツールの課題である環境構築と前処理の複雑さやを抽象化する「導入の容易さ」は，処理の複雑さを抽象化するために重要な要件である．次に，既存ツールの共通課題を解決するサービスを提供することは，よりよい脆弱性検出ツールの実現に重要である．そこで，既存ツールの共通課題である解析時間を解決する「リアルタイム性」は重要な要件である．

以上から，IoT ファームウェアの静的解析のツールの要件は下記とする．

1. リアルタイム性

2. 導入の容易さ

その他，静的解析ツールにおいて，脆弱性の検出精度も重要な要件である．しかし，検出精度は脆弱性検出アルゴリズムに依存するため，共通サービスの提供が責務であるツールの要件としては対象外となる．本ツールでは，脆弱性検出アルゴリズムによらない解析時間の削減と導入の容易性を実現する．

6.3　ソフトウェア設計

本節では，6.2 節で定義したそれぞれの要件に対して，要件を満たすソフトウェア設計について議論する．

6.3.1　導入の容易さ

導入の容易さを担保するため，2 つの機能を導入する．1 つ目は，使用の容易さを担保するために，ファームウェア本体をそのまま入力データとして取り扱えるしくみを導入である．既存ツールは，入力として決められたデータ形式しか受け付けないが，本ツールは多様な入力データに対応する．

2 つ目は，環境構築の容易さを担保するために，ツールのベースは仮想環境で提供する．これにより，ユーザー環境の違いによって生じるソフトウェアの挙動や環境設定の違いを吸収し，ユーザーに統一したサービスを提供する．環境の配布にあたり，ユーザーがツールのインストールと使用に必要十分なマニュアルも作成する．

6.3.2　リアルタイム性

リアルタイム性を担保するため，3 つの機能と導入する．1 つ目は，解析時間を短縮するため，シンボリックリンクによる重複解析をふせぐ仕組みを導入する．具体的には，ファームウェアのシンボリックリンクは解析対象とせず，実体のみを解析対象とすることで，解析時間を削減するしくみを導入する．

2 つ目は，逆解析結果を機能ごとに API として提供するしくみである．これにより，必要な解析結果だけを利用してアルゴリズム開発ができるため，不要な解析処理が減り，解析時間の削減につながる．

3 つ目は，ファームウェアに含まれる複数の構成ファイルに対し，連続で逆解析および検出アルゴリズムを適用できる技術である．逆解析および解析環境の基盤を提供する Ghidra では，手動で全てのファイルに逆解析および検出アルゴリズムを適用しなければならず，作業工数の負荷量は現実的ではない．そこで，本ツールでは複数のファイルに対して，連続して逆解析および検出アルゴリズムを適用できる技術を開発した．

6.4 埋め込みログイン情報検出ツールの構築

本節では，埋め込みログイン情報検出ツール「Yielding Optimal Device Analyzer（YODA）」を開発し，ツールの機能について具体的に説明する．表 6.1 に本ツールの概要図を示す．

6.4.1 ベース開発

表 6.1 で示す，Our Middleware のベース技術について説明する．本ツールでは，Docker によるコンテナ型仮想化技術を採用する．Docker は Linux のコンテナ技術を使用したもので，アプリケーションやランタイム環境を含むソフトウェアの実行に必要なファイルをパッケージ化する技術である．ユーザーはツールの Docker コンテナを起動すればサービスが利用できるため，複雑なライブラリのインストール作業が削減できる．これは関連研究の解析ツールで問題になっていた，環境構築のライブラリ管理の複雑さを解決する．

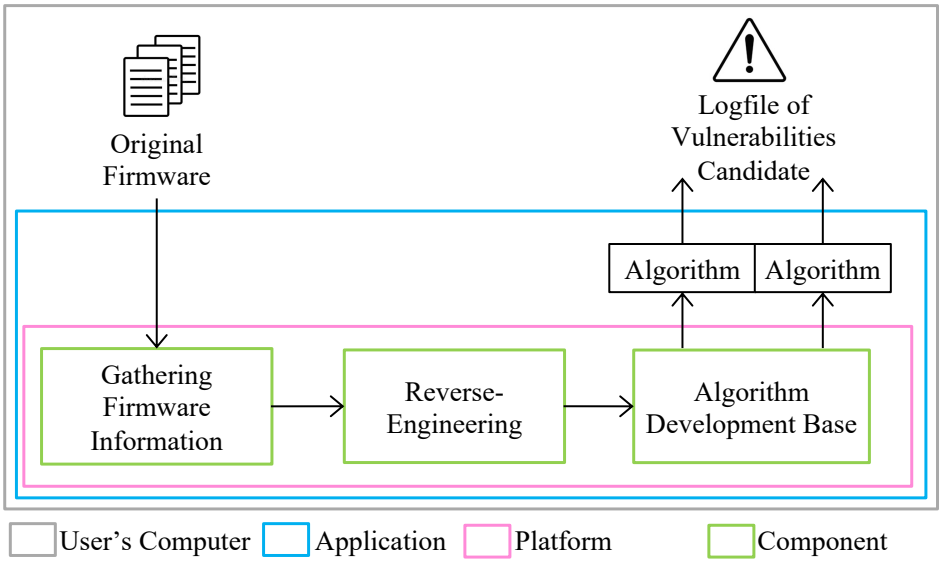

図 6.1: 提案ツールの概要図

6.4.2 コンポーネントの開発

表 6.1 に示す，Component について説明する．本ツールでは，各サービスをコンポーネントとして提供する．

ファームウェア情報の収集

本コンポーネントでは，ファームウェアから逆解析および解析工程に必要な情報を自動で収集するサービスを提供する．本コンポーネントによって，これまで課題となっていた前処理の複雑さが抽象化される．

本コンポーネントを利用する準備として，ユーザーは解析したいファームウェアのバイナリを指定のディレクトリに格納する．このときユーザーは複数のファームウェアを格納できる．

本コンポーネントを実行すると，まず，ファームウェアの構成ファイルが展開される．次に，本コンポーネントは展開した構成ファイルの中から，解析対象のファイルリストを作成する．一般的に，静的解析は ELF ファイルに対して解析を行うため，本ツールも ELF ファイルを解析対象とした．

ファイルリストの作成時，解析対象の重複を排除することで解析時間の削減を実現する．例えば，busybox に含まれるコマンドの多くはシンボリックリンクで実現されているが，逆解析ではシンボリックリンクも実体とみなすため，busybox が複数回逆解析されてしまう．busybox のようにシンボリックリンクで参照される実体は，複数のコマンドを抱えていることからファイルサイズが大きくなる傾向がある．そのため，重複した逆解析は解析時間に影響を及ぼす．そのため，ファイルリストの作成による重複の排除は重要な作業である．構成ファイルの取り出しには，Firmware-mod-kit [8] を採用した．

Algorithm 11 にプログラムの流れを示す．本コンポーネントを実行すると，1 行目でファームウェアの構成ファルが抽出される．ファイルの抽出方法は確立されているため，本手法で新たな機能は実装せず，firmware-mod-kit [8] を利用した．

次に，本手法で独自に開発したプログラムを 2 から 13 行目実行する．まず，解析対象となるコンポーズファイルのリストを作成し，次の行の処理に渡す．一般的に，静的解析は ELF（Executable and Linkable Format）ファイルを解析対象とするため，本コンポーネントでも ELF ファイルのリストを作成する．次の解析処理で，ELF ファイル以外が渡されると処理が停止してしまうため，本作業は作業の自動化に重要である．重複ファイルの排除は 7 行目で実行される．

Algorithm 11 Gathering Firmware Information

```
1: FirmwareBinaryList ← getFirmwareBinaryList()
2: while FirmwareBinaryList.hasNext() do
3:     ELFList.open(ThisFirmwareLogFile)
4:     ExtractedFiles ← FirmwareBinaryList.getFirmwareFiles()
5:     while ExtractedFiles.hasNext() do
6:         File ← ExtractedFiles.getFile()
7:         if File.getFormat() = ELF then
8:             ELFList.write(File.getFilePath())
9:             File.ImporttoGhidra(this.projectPath)
10:        end if
11:    end while
12:    ELFList.close()
13: end while
```

解析基盤

本コンポーネントでは，ファームウェアを静的解析するベース環境を提供する．具体的には，逆解析機能，脆弱性検出アルゴリズムの開発と実行機能を提供する．

本ツールでは解析基盤のベースとして Ghidra を採用した．解析基盤としてシェアを占めているのは，Ghidra，IDAPro，および angr である [29]．IDAPro は有料ライセンスが必要であるため，ユーザーがツールを導入する障壁になりうる．一方，Ghidra は Apache License 2.0 を採用しているため，その制限は発生しない．angr は Ghidra 同様に無償で利用が可能であるが，ELF ファイルの解析カバレッジは Ghidra のほうが高いことがわかっている [14]．本ツールでは Ghidra が最も解析基盤として適していると判断し，Ghidra を採用した．

解析基盤では，逆解析および脆弱性検出アルゴリズムの開発と実行機能の共通機能として，ELF ファイルの一括解析機能を提供する．一般的に，1 つのファームウェアに含まれる ELF ファイルは数百個以上に及ぶが，Ghidra では一括で逆解析および脆弱性検出アルゴリズムを適用する機能がなく，手作業で実施する必要があるため，一括解析機能を開発した．

逆解析機能

Algorithm 12 setUpDecompiler()

```
1: decompApi ← FlatDecompilerAPI(firmware)
2: if decompApi.getDecompiler() == null then
3:     decompApi.initialize()
4: end if
5: decomplib ← decompApi.getDecompiler()
6: options = DecompileOptions()
7: decomplib.toggleCCode(true)
8: decomplib.toggleSyntaxTree(true)
9: decomplib.setSimplificationStyle("decompile")
10: returndecompApi;
```

Algorithm Development Base　逆解析機能では，ELF ファイルを逆解析し，元プログラムを復元したり，グラフを生成するなどして，ELF ファイルから様々な情報を取得する．逆解析結果を利用することで，プログラムやデータの流れを用いた高度な脆弱性解析が可能となる．逆解析結果は検出アルゴリズムの適用先として，アルゴリズムのプログラム内で参照される．

本機能では，逆解析結果を API として提供する．本来，ユーザーは逆解析結果を取得するプログラムを開発なければならない．しかし，逆解析結果を取得する方法は Ghidra の公式ガイドラインには記載されていないため，Ghidra の内部プログラムを読み解く必要があり，複雑なプログラム開発が必要となる．本 API を利用することで，ユーザーは別途複雑なプログラムを開発する必要がなく，API を呼び出すだけで逆解析結果を検出アルゴリズム内で利用できる．

更に，本 API では，逆解析結果を機能ごとに分けて提供する．機能ごととは，ELF ファイルの元プログラムのみを提供する API，グラフ生成結果をのみを提供する API などである．機能ごとに API を切り分けて提供することで，検出アルゴリズムに必要な逆解析結果のみ使用できるため，不要な処理が減り，解析時間の削減につながる．Algorithm 12 で API を呼び出す例を示す．この API は，ファームウェアのバイナリを逆解析し，元プログラムを復元する．

脆弱性検出アルゴリズムの開発と実行機能 脆弱性検出アルゴリズムの開発と実行機能では，検出アルゴリズムの開発環境と，検出アルゴリズムを実行する機能を提供する．開発環境では，検出アルゴリズムのテンプレートを Java ファイルで提供する．テンプレートファイルには，逆解析結果 API の参照プログラムやログファイルへの結果出力プログラムなど，アルゴリズムのベースに必要なサンプルプログラムが記載されている．テンプレートファイルを使用することで，ユーザーは本ツールのサービスを容易に利用できるため，すみやかに本質的なアルゴリズム開発にとりくむことができる．

Algorithm 13 Example of analysis tool

```
1: logPath ← "/mnt/example/logs"
2: stringsSearch ← StringSearchHeadless()
3: stringsSearch.getVals(targetFirmware, logPath)
```

Algorithm 13 は，提案したテンプレートを使って構築した，Socket Search のプログラム例を示す．1 行目で結果を記録するのログファイルのパスを指定，プログラムを実行するだけで解析が実行できる．

本ツールを活用した解析ツールの開発例 本ツールを活用したサンプル解析ツールとして，既存研究である Karonte の脆弱性解析アルゴリズムを一部実装する解析ツールを開発した．

ネットワーク通信通信検出アルゴリズム 本アルゴリズムは，関連研究である Redini らが提案した Karonte の脆弱性検出機能の一部である [18]．本アルゴリズムは，ELF ファイル内にネットワーク通信関連の文字列が含まれているか検出する．Redini らはネットワーク通信に関連する文字列を定義し，これらを含む ELF ファイルを検出する．検出対象の文字列は，`QUERY_STRING`，`username`，`http_`，`REMOTE_ADDR`，`boundary=`，`HTTP_`，`query`，`remote`，`user-agent`，`soap`，`index.`，
`CONTENT_TYPE`，`Content-Type` である．Redini らは，これらの文字列を含む ELF ファイルは，攻撃者からのデータ入力を受付ける機能の可能性があり，DoS 攻撃を受ける可能性があるファイルとして注目している．

KARONTE は，もう一つの検出対象であるバッファオーバーフローは，マルチバイナリ解析と呼ばれる，ELF バイナリ間のデータ通信を利用してする．本ツールは，現状，単一

バイナリの解析を提供しているため，単一バイナリのデータ Dos 攻撃に対するアルゴリズムをサンプルとして開発した.

第7章　埋め込みログイン情報検出ツールの評価

本章では，提案した埋め込みログイン情報検出ツールの評価と考察について述べる．まず7.1節で，提案ツールの評価について述べ，7.2節で考察を述べる．最後に，7.3節で，提案ツールのまとめと今後について述べる．

7.1　評価

7.1.1　導入の容易さ

導入の容易さ，つまりツールの導入とサービス使用時の工数の少なさを評価するため，環境構築と前処理作業の作業工数を計測した．環境構築は，インストールから起動までの工程を示す．前処理作業は，起動から解析実行にかかるまでの工程を示す．評価対象は，提案ツール，Karonte，およびベースラインの3ツールである．ベースラインは，提案ツールの要件は考慮せず，提案ツールの前処理までに必要なサービスを提供する環境を指す．

具体的な実験内容は，被験者に担当ツールのマニュアルを渡し，インストールから前処理作業まで行ってもらい，それぞれの作業にかかった時間を計測した．作業時間は最大8時間とした．被験者は合計19人で，各ツールの被験者の内訳は，提案ツールは6人，Karonteは7人，そしてベースラインは6人である．

被験者の属性は，情報系専攻の学部生，大学院生（前期・後期），およびソフトウェア開発歴5年以上のソフトウェアエンジニアである．各ツールの被験者の属性や技能レベルの公平性を担保するため，学年内で使用ツールが異なるように振り分けた．本実験は電気通信大学倫理審査会の承認を受けており（管理番号: 22090），各被験者から同意の署名を得た． Karonteは公式マニュアルが英語であったため，マニュアル解読時間の公平さを保つ

ため，日本語に翻訳したマニュアルを被験者に渡した．Karonte は実施時期が異なるため，他 2 ツールと人数が異なる．

表 7.1 に各作業時間の結果を示す．環境構築においては，提案ツールは Karonte よりも，平均値は 53 分，最大値は 1 時間 22 分早く作業が完了した．最小値は Karonte が 15 分早かった．ベースラインは，提案ツールよりも早く作業が終了した．

前処理においては，提案ツールは Karonte よりも，平均値は 2 時間 31 分，最大値で 3 時間 51 分，最小値は 45 分の作業時間の短縮がみられた．提案ツールはベースラインよりも，平均値は 2 時間 19 分，最大値は 4 時間 43 分，そして最小値は 17 分の作業時間短縮がみられた．表 7.2 に全作業の完了者数の比較を示す．8 時間以内に全作業を終えた被験者は，提案ツールは 6 人全員，Karonte は 7 人中 2 人，そしてベースラインは 6 人中 5 人であった．

表 7.3 に全作業の合計時間の結果を示す．環境構築と前処理作業の全体の作業時間では，提案ツールは Karonte よりも，平均値は 3 時間 24 分，最大値は 2 時間 18 分，そして最小値は 1 時間 6 分早く作業が終了した．ベースラインとの比較では，提案ツールが平均値 1 時間 9 分，最大値は 1 時間 38 分，最小値は 12 分早く作業が終了した．

表 7.1: 環境構築時間と前処理作業時間の比較

ツール名	環境構築（時間：分）			前処理（時間：分）		
	最小値	最大値	平均値	最小値	最大値	平均値
YODA	0:27	3:33	1:53	0:29	2:09	1:04
Karonte	0:12	4:55	2:46	1:14	6:00*	3:36*
Baseline	0:22	1:15	0:43	0:46	7:32*	3:30*

* 環境構築と前処理作業の合計作業時間は 8 時間に制限した．8 時間の制限時間を設けない場合，作業時間は増加する．

7.1.2　リアルタイム性

第 6.4.2 章で開発したサンプルツールと Karonte の解析時間を比較した．解析対象として，Karonte のデータセットである 4 ベンダー 22 件のファームウェアを利用した．公平性

表 7.2: 全作業の完了者数の比較

ツール名	被験者数（人）	時間内に全作業完了した被験者数（人）
YODA	6	6
Karonte	7	2
Baseline	6	5

表 7.3: 全作業の合計時間の比較

ツール名	最小値	最大値	平均値
YODA	0:56	5:42	2:58
Karonte	2:02	8:00*	6:22*
Baseline	1:08	8:00*	4:13*

* 8 時間で作業が完了しなかった場合，作業時間を 8 時間とみなした．もし 8 時間の制限時間を設けない場合，作業時間は増加する．

を保つため，Karonte とサンプルツールの検出範囲が同等になるように，Karonte プログラムの解析範囲を改変した．解析時間を計測したコンピュータのスペックは，OS が Ubuntu 22.04.02，CPU が i9-9900K，RAM16GB である．

表 8.4 に解析時間の結果を示す．22 件全てのファームウェアで，提案ツールの解析時間が Karonte よりも早いことがわかった．最もサイズの大きい NETGEAR の 304MB のファームウェアでは，提案ツールは Karonte よりも，10 時間 55 分早く解析が終了した．重複ファイル数は 232 個，合計のデータサイズは 57MB であった．重複解析の回避機能によって，57MB の解析が削減された．最もファイルサイズが小さい TP-Link の 28MB のファームウェアでは，提案ツールは Karonte よりも，14 分早く解析が終了した．重複ファイル数は 64 個，合計のデータサイズは 10MB であった．重複解析の回避機能によって，10MB の解析が削減された．

7.2　考察

7.2.1　導入の容易さ

環境構築作業では，提案ツールと Karonte と比較したとき，平均値と最大値において，提案ツールのほうが作業時間が短いことを確認した．最小値では Karonte のほうが早かった理由として，インストールする外部ツール数の違いがある．Karonte は Docker のインストールのみであるが，提案ツールは Docker と Git コマンドのインストールが必要である．そのため，提案ツールでは Git コマンドのインストール作業時間が計上されたため，最小値が Karonte より長かった．しかし，Docker と Git は一般的な技術であり，被験者によっては日頃から使用しており，実験前からインストール済の場合もある．そのため，外部ツールのインストールによる作業時間の差異は，解析ツール特有のものではないことから，環境構築の本質ではないといえる．

ベースラインは提案ツールよりも短い時間で作業が完了している．これは，ベースラインの環境構築作業が，Ghidra のインストールのみであったため，作業時間が短くなった．一方，提案ツールと Karonte は，ツール本体のダウンロード，インストール，および起動確認の複数工程が必要なため，作業時間が長くなった．しかし，提案ツールと Karonte は，ツール本体がパッケージ化されており，前処理から解析に必要な全てのツールが揃っているため，追加のインストール作業は不要である．環境構築の作業時間の最大値については，Docker 未経験の被験者が，Docker インストール方法と起動方法に時間がかかったため，時間がかかっている．今回の環境構築の範囲は前処理作業までに必要なツールのインストールであったため，ベースラインは解析作業以降に必要なツールは別途インストールが必要である．ベースラインで全てのツールをインストールした場合，環境構築の作業時間は増加する見込みである．

前処理作業と全作業の合計時間においては，Karonte とベースラインと比較し，全ての値において提案ツールの作業時間が短いことがわかった．また，8 時間以内に被験者全員の作業が終了したのは，3 ツール中，提案ツールのみであった．ベースラインは 6 人中 1 人，Karonte は 7 人中 5 人が，時間内に全工程を終えられなかった．そのため，作業の上限時間を設けなかった場合，ベースラインと Karonte の作業時間は増加する．提案ツールは．全

員が時間内に作業を完了したことからも，確実に各工程の作業工数が削減できたことがわかる．

次に，全作業における被験者の作業報告について考察する．提案ツールを使用した被験者の作業報告では，環境構築では一部の被験者が Docker の使用方法の確認時間を要していた．しかし，提案ツールのインストールそのものに関するエラーは報告されなかった．また，前処理作業においてもサービス自体のエラーは報告されず，被験者全員が作業時間内に作業を終了した．前処理作業でも，マニュアル不備等による実行エラーは報告されなかった．しかし，操作がわかりにくいという報告を受けた．内容としては．提案ツールは GUI 環境と CLI 環境を提供しており，どちらでサービスを実行するのかわかりにくいという旨である．これについては，どちらの環境からもサービスが使用できる設計になっており，サービス使用不可による作業停止の恐れはないため，導入の容易さは担保される．しかし，さらなる導入の容易さを追求するため，いずれの環境においてもサービスが利用可能であること，もしくは推奨環境である CLI 環境を利用する旨をマニュアルに明記が必要である．

ベースラインを使用した被験者の作業報告では，環境構築の際に，Ghidra の動作に必要な Java 環境の構築と，ファームウェアを分解する外部ツールに必要なライブラリのインストールがつまづいたという報告があった．一方，提案ツールはツール内に動作に必要な全ての外部ツールが含まれているため，ベースラインで発生したエラーは生じない．前処理作業では,「ファームウェア分割ツールをインストールはしたが使用方法がわからない」や「ファームウェアが分解できず構成ファイルが取り出せない」ことから，作業時間を要していることがわかった．この点に関し，提案ツールは前処理作業を抽象化しているため，提案ツールを使用することでベースラインで発生した問題は解決できる．

Karonte を使用した被験者の作業報告では，環境構築作業で Karonte の Docker コンテナを起動する際，通常の Docker コマンドではなく特有のオプションが必要で，その記載がマニュアルになかったことから，起動に時間がかかる被験者が多かった．一方，提案ツールでは起動時の実行エラーや不明点の報告はなく，マニュアルに従ってすみやかに作業できたことが被験者の報告で確認できている．この点において，提案ツールを使用することで，Karonte で発生した問題は解決可能である．その他，提案ツールを同じく，Docker のイン

ストールや使い方の理解に時間を要する被験者もみられた．前処理作業では，ベースライン同様，外部ツールの使い方，および入力となる設定ファイルに必要なファームウェア情報の収集方法がわからず，前処理作業に時間を要する被験者が多かった．前処理作業では，7 人中 5 人が時間内に作業を完了できなかったことからも，前処理作業の複雑さが確認できる．一方，提案ツールでは前処理作業は不要で，Karonte のような設定ファイルを作成する必要がない．そのため，提案ツールを使用することで，Karonte で発生した前処理作業の複雑さを解消できる．

考察をまとめる．提案ツールは，3 ツールにおいて唯一，全ての被験者が時間内作業が終了し，全ての作業において Karonte よりも早い時間で作業が完了した．また，提案ツールの被験者からは，Karonte やベースラインで発生した前処理作業の複雑さの報告はなく，更に，マニュアル不備によるインストールと使用に関するエラーによる作業の中断報告はうけなかった．この結果から，提案ツールは導入と使用が容易かつ，作業時間も少ないと言えるため，ミドルウェア要件である「導入の容易さ」を満たすといえる．

7.2.2　リアルタイム性

1 つのファームウェアには数百〜数千の構成ファイルが含まれており，1 つずつ解析する必要がある．解析時間が長時間化する非効率さは，すべての構成ファイルの解析時間に影響を及ぼしてしまう．また，機関や解析者など，一度に数千のファームウェアを解析したいユーザーにとっては，解析時間の長時間化は耐えがたいものとなる．22 件のファームウェア全てにおいて，提案ツールは Karonte よりも早い時間で解析が終了した．特に，ファームウェアサイズが大きくなるにつれて解析時間にひらきがあり，最もサイズの大きい NETGEAR の 304MB のファームウェアでは，10 時間 55 分もの解析時間の差があった．提案ツールの解析時間が短かった理由として，提案ツールは逆解析結果を API 化することでアルゴリズムが使用する機能を選べるようになっているため，不要な処理が減り，解析時間の大幅な短縮が実現できた．一方，Karonte は，基本解析機能として関数のフローグラフを構築する処理等を含んでいるため，関数の数が多くなるほどグラフ生成に時間がかかり，解析時間が長くなった．Karonte は提案ツールのように，逆解析結果を選択する機能を備えていないため，全ての逆解析結果を使用した解析が行われてしまい，解析時間が長くなった．

リアルタイム性の評価にベースラインを含まなかった理由としては，ベースラインは多くの作業が手動であることから，解析時間以外の作業も計上されてしまい，正確な計測が困難であったためである．例えば，1 件のファームウェアを解析するためには数百〜数千の ELF ファイルに対して脆弱性解析アルゴリズムを適用する必要がある．ベースラインでは，1 つ 1 つの ELF に手作業で解析アルゴリズムを適用しなければならないため，正確な解析時間の計測は困難である．一方，提案ツールは一括で解析する機能を提供するため，この問題は発生しない．

仮にベースラインにおける作業工数の問題が解決され，解析できるようになった場合でも，ベースラインでは重複解析が発生する．そのため，重複解析が発生しない提案ツールよりも，ベースラインのほうが解析時間が増加することが予想される．以上から，提案ツールは要件となっている「リアルタイム性」，つまり要求された迅速に処理する能力を有しており，要件を満たすといえる．

7.3 まとめと今後

7.3.1 まとめ

本手法では，既存の IoT 脆弱性検出ツールで課題となっている，前処理作業と環境構築の複雑さ，および解析時間の長時間化を解決するため，IoT 脆弱性検出ツールの構築と評価を行った．ツールを構築するために，まず要件定義を実施した．要件定義では，既存の解析ツールの課題，一般的なミドルウェアの定義，類似分野かつミドルウェアの提案が盛んな IoT 環境のミドルウェア要件を洗い出し，提案ツールに必要な要件を定義した．ツールの構築では，要件を満たすツールを実現するために，4 つの新規技術開発と 1 つの環境構築に関する工夫を取り入れた．評価では，環境構築，前処理作業，および解析時間について，ベースラインと既存研究のうち最も支持されている Karonte と比較し，提案ツールが要件を満たすか評価した．結果として，提案ツールは Karonte と比較したときに，すべての作業項目において提案ツールが性能を上回ることが確認でき，本研究では要件を満たすツールの開発に成功した．

表 7.4: Karonte と提案ツールの解析時間

Vendor	Firmware Name	Unpacked Firmware Size(MB)	YODA (hour:min)	Karonte (hour:min)
NETGEAR	R8500	304	0:37	11:32
NETGEAR	AC1450	71	0:13	0:40
TP_Link	Archer_C2	54	0:08	0:56
TP_Link	Archer_C50	53	0:08	0:55
TP_Link	Archer_C3200	96	0:13	1:31
TP_Link	Archer_D2	100	0:09	1:07
TP_Link	TD_W9970	57	0:08	0:54
TP_Link	TL-MR3020	57	0:04	0:18
TP_Link	TL-WA830RE	28	0:04	0:18
TP_Link	TL-WR1043ND	39	0:06	0:49
TP_Link	TX-VG1530	106	0:19	1:24
D-Link	DIR-868	71	0:11	8:52
D-Link	DIR-880	86	0:20	0:56
D-Link	DIR-885	104	0:12	0:42
D-Link	DIR-895	192	0:12	0:41
D-Link	DIR-118	40	0:10	0:40
D-Link	DIR-826	46	0:10	1:15
D-Link	DIR-890	105	0:13	0:43
Tenda	US_AC6V	33	0:07	1:13
Tenda	US_AC9V	37	0:09	2:07
Tenda	US_AC15V	64	0:12	2:55
Tenda	US_WH450AV	37	0:15	1:39

第8章　ミドルウェア化の展望と評価

本章では，提案ツールがIoT機器の脆弱性検出におけるミドルウェアとして拡張できるか検討し，試作と評価を行う．8.1節では，既存ツールの共通機能を調査し，ミドルウェアに搭載すべき機能を述べる．8.2節では，ミドルウェアの構築について説明する．8.3節，8.4及び8.5節では，ミドルウェアの評価を考察を述べる．最後に，8.6節でまとめと今後について述べる．

8.1　既存ツールにおける共通機能の調査

ミドルウェア構築のため，既存の脆弱性検出ツールで使用されている機能を調査した．これにより，脆弱性検出ツールが備えるべき機能がわかるため，ミドルウェア構築の参考にする．ミドルウェアとは，システムやハードウェアの複雑さを抽象化した上で，アプリケーションに共通するサービスを提供するものである．表8.1にまとめを示す．共通機能は8つに分類された．分類方法は，関連研究が使用している解析準備や脆弱性検出アルゴリズムの技術を集計し，集計結果で得た各技術に名称をつけて分類した．

ファームウェアの分割

ファームウェアの分割は，ファームウェアからファイルシステムの構成ファイルを分割する技術である．通常，ファームウェアは複数のファイルが圧縮されている．そのため，ファームウェア解析をおこなうSREツールはそのまま読み込むことができない．そのため，多くの研究ではSREツールにファームウェアを読み込ませるために，ファームウェアを分割する作業が必要となる．

表 8.1: 既存の脆弱性検出ツールで使用されている機能の分類

Research	Splitting Firmware	Finding Static Strings	Finding Memory Value	Using Symbolic Execution	Using Emulator	Generating Graph	Using Machine Learning	Finding Network Function
Firmalice	✓	✓	✓	✓	-	✓	-	-
PIE	✓	-	-	-	-	✓	-	-
FIRMADYNE	✓	-	-	-	✓	-	-	✓
Stringer	✓	✓	-	-	-	✓	-	-
HumIDIFy	✓	-	-	-	-	-	✓	-
DTaint	✓	✓	✓	-	✓	-	-	✓
FirmUp	-	-	-	-	-	✓	-	-
FirmFuzz	✓	✓	-	-	✓	-	-	✓
Johnetal.	✓	-	-	-	-	✓	✓	✓
KARONTE	✓	✓	✓	-	-	✓	-	✓
Number of ✓	10	5	3	1	3	6	2	5

静的文字列の探索

静的な文字列とは，ファームウェアに埋め込まれた文字列の値を意味する．ファームウェアにパスワードや ID などのログイン情報が埋め込まれていると，パスワードを知り得た第三者がファームウェアに不正ログインし，ファームウェア機器を制御できてしまう脆弱性が発生する場合がある．

メモリ値の探索

メモリ値とは，メモリ上に格納されている値のことである．メモリにはユーザーからの入力値が格納される．このため，ユーザまたは攻撃者が何らかの値を入力するとその値がメモリに一時保存される．メモリ上に保存されたユーザ入力値が安全であるかテイント解析などで活用される．

シンボリック実行

シンボリック実行とは，ある入力値がプログラムの実行に与える影響や経路をトレースする技術である．

エミュレータの使用

エミュレータは，ファームウェアの動的解析分野で使用される機能である．ファームウェア解析は大きく分けて 2 種類あり，動的解析と静的解析がある．動的解析では，エミュレータを用いて仮想環境上で IoT 機器を動かすことで脆弱性を検出をする．静的解析では，プログラムを動作させずに脆弱性解析を行う．

コントロールフローグラフの生成

グラフとはプログラムの流れを表すものである．例えば，代表的なグラフとしてコントロールフローグラフがある．ファームウェアのプログラムの流れを把握することで，プログラムの機能理解に役立つ．

機械学習の使用

機械学習を脆弱性解析に使用する場合，学習データには脆弱性やバグの特徴がみられる関数，シンボルやプログラムフローを使用する．バイナリをモデルへの入力とし，脆弱性と考えられるプログラム箇所を結果として出力する．関連研究では半教師あり学習や深層学習を用いたモデルが使用されている．

ネットワーク機能の探索

ネットワーク機能とは，socket シンボルやネットワークポートなど，ファームウェアの情報を外部通信する際に使用する機能を指す．ネットワーク関数は，攻撃者の侵入経路となるため，ファームウェアの脆弱性解析でよく活用される．

8.2 ミドルウェアの試作

本節では，既存の脆弱性検出ツールで共通してよく機能をもつ脆弱性検出ツールと提案をおこなう．

表 8.2: 試作した脆弱性検出ツールのモジュール一覧

モジュール名	呼び出し方法	入力（引数）	出力
ファームウェアの分割	splitFirmware()	ファームウェア単体, ディレクトリ名	ファームウェアから取り出されたファイル群
静的文字列の探索	staticStrings()	ELF ファイル	文字列比較関数を参照する関数一覧, 文字列比較関数を参照するプログラム箇所
メモリ値の探索	memoryValues()	ELF ファイル	メモリ値を参照する関数一覧, メモリ値を参照するプログラム箇所
コントロールフローグラフの生成	controlFlowGraph()	ELF ファイル	ELF ファイルのフローグラフ
ネットワーク関数の特定	identifyingNetwork()	ELF ファイル, 探索の深さ（整数）	ネットワーク関数の一覧, ネットワーク関数を参照する周辺関数

8.2.1　概要

共通機能のまとめを踏まえて，本試作では (1) ファームウェア分割，(2) 静的文字列の探索，(3) メモリ値の探索，(4) コントロールフローグラフの使用，(5) ネットワーク機能特定の 5 つの機能を提供する脆弱性検出ツール（以下，ツール）を提案する．ツールの概要を，表 8.2 にツールのモジュール一覧を示す．

またツールの利用例を Algorithm 14 に示す．研究者は脆弱性検出アルゴリズムを開発する際，プログラム内で各モジュールを呼び出して使用する．本ツールを使用しない場合，研究者は各機能を実装する必要があるが，本ツールを活用することで脆弱性解析の事前準備と頻出機能の結果を得ることができ（3-11 行目），アルゴリズムのコア開発に即時に取り組むことができる（12 行目）．

8 機能のうち 5 つを選択した理由は，シンボリックリンク実行は 13 研究のうち 1 つでしか使用されておらず，利用頻度が低いためである．エミュレータの使用については，本ツールは静的解析手法をターゲットとしており，動的解析手法であるエミュレータは解析方法が根本的に異なるため対象としなかった．機械学習の使用については，機械学習モデルは数多く存在しており，現状では脆弱性検出に最も有効な機械学習モデルの検討が済んでいないためである．

Algorithm 14 Main Program of Firmware Analysis

```
RawBinaries ← getRawBinaries()
while RawBinaries.hasNext() do
    RawBinary ← RawBinaries.getBinary()
    ELFs ← splitFirmware(RawBinary)
    while ELFs.hasNext() do
        imported ← importELFintoSRE(ELFs.getELF())
        if imported then
            strings ← staticStrings(importedELF)
            memory ← memoryValues(importedELF)
            graphs ← controlFlowGraph(importedELF)
            networks ← identifyingNetwork(importedELF, 3)
            algorithmByResearch(strings, graphs, networks)
        end if
    end while
end while
```

表 8.3: 本手法の適用によって削減できるコード行数

Research	Total Code Lines	Splitting Firmware	Finding Static Strings	Finding Memory Value	Using Graph	Finding Network Function
FIRMADYNE	5111	28	-	-	-	0
Stringer	1467	0*	188	-	201	-
HumIDIFy	17539	0*	-	-	-	-
FirmFuzz	2198939	190	0	-	-	0
KARONTE	33938	162	180	439	968	801

* ツールの入力として分割済みのバイナリが求められていたため，置き換え対象となるプログラムが存在しなかった.

8.3　コード数の削減

8.3.1　評価

既存研究のうち，公開されているソフトウェアを実験対象とし，本ツールで書き換え可能なプログラムの行数を調査した．13 個の関連研究のうちソフトウェアを公開している研究は 5 つであった．本実験では，本ツールと実験対象のソフトウェアの言語が異なる場合でも，処理内容が同じ場合は書き換え可能と判断した．調査結果を表 8.3 に示す．Research はツール名で，Total Code Lines は各ソフトウェアの総コード数を示している．

5 つのソフトウェアのうち本ツールで一部書き換え可能なソフトウェアは，FIRMADYNE, Stringer，FirmFuzz と KARONTE であった．書き換え対象とならないソフトウェアは HumIDIFy であった．

FIRMADYNE は，ファームウェア分割のプログラムが 28 行置き換え可能で，ネットワーク関数の特定機能は置き換え対象とならなかった．コードの削減率は 0.5%であった．

Stringer は，静的文字列の探索機能は 188 行，グラフの使用は 201 行置き換え可能で，ファームウェア分割のプログラムは置き換え対象とならなかった．コードの削減率は 26%であった．

HumIDIFy はどの機能も置き換え対象とならなかった．

FirmFuzz は，ファームウェア分割機能が 190 行置き換え可能で，静的文字列の探索とネットワーク関数の特定機能は置き換え対象にならなかった．

KARONTE は全ての機能で置き換え可能であった．ファームウェア分割は 162 行，静的文字列の探索は 180 行，メモリに格納された値の探索は 439 行，グラフの活用は 968 行，ネットワーク関数の特定では 801 行のプログラムが置き換え対象となった．コードの削減率は 7.5%であった．

8.3.2　考察

FIRMADYNE は，ファームウェア分割部分が本ツールで置き換え可能であった．ファームウェア分割ツールとして，binwalk[1]を外部呼び出ししており，binwalk 呼び出し箇所が書

[1]binwalk. https://github.com/ReFirmLabs/binwalk

き換え対象となった．一方，ネットワーク関数の特定が置き換え対象とならなかった理由は，FIRMADYNE は動的解析ツールであり，仮想環境の LAN インターフェースを用いて Web ページを収集するなど，動的解析特有の機能を用いてネットワーク関数の特定を行っているため，静的解析である本ツールの書き換え対象にはならなかったからである．

Stringer は，静的文字列の探索とコントロールフローグラフの使用に関するプログラムが置き換え可能であった．一方，ファームウェア分割は，別途分割ツールを用いているため置き換え対象となるプログラムが存在しなかった．理由として，Stringer は論文上でファームウェア分割ツールとして BAP[2]を使用していると記述しているが，実際のツールではファームウェア分割機能は提供されておらず，ツールの入力として分割済みのバイナリが求められていたためである．ファームウェアの分割にかかる準備や設定は開発環境に依存する部分が多く，仕様書の手順では見えないトラブルシューティングが多いため，実際の準備にかかる開発工数は大きい．そのため，分割ツールを導入する環境設定まで含めて考えると，本ツールを導入することによって更に開発工数を削減できる見込みがある．

HumIDIFy も Stringer 同様に，ファームウェア分割は，別途分割ツールを用いているため置き換え対象となるプログラムが存在しなかった．論文では，ファームウェア分割に binwalk を使用していると記述されているが，ツールではファームウェア分割機能は提供されておらず，分割済みのバイナリが入力として求められていたためである．HumIDIFy でも分割ツールを導入することで，開発工数を削減できる可能性がある．

FirmFuzz は，ファームウェア分割部分が置き換え可能であった．FirmFuzz はファームウェア分割として FIRMADYNE つまり binwalk を使用しており，その呼出部分が置き換え可能となった．一方，静的文字列の探索とネットワーク関数の特定は置き換え不可であった．理由としては FirmFuzz は動的解析であり，仮想環境上でファームウェアを動作させて脆弱性解析をおこなっているため，静的解析の本ツールとはアルゴリズムが異なるため置き換えが不可となった．

KARONTE は，すべての機能で置き換え可能であった．KARONTE は静的解析ツールであり，同じく静的解析である本ツールと親和性が高かった．

全体の結果として，5 つのうち 4 ツールで置き換え可能であることがわかった．特にファー

[2]BAP. https://github.com/BinaryAnalysisPlatform/bap

ムウェア分割は，動的解析・静的解析のいずれでも置き換え可能であり，解析手法が異なっても本ツールが活用できることがわかった．

実験対象のツールは，各自のアルゴリズムに従った結果のみを出力するようにソフトウェア設計・開発されており，プログラム再利用がしにくく汎用性に欠ける．一方，本ツールは共通機能の呼び出しが容易であり再利用性が高い．煩雑な前処理や共通機能の記述が不要となるため，ゼロベースでアルゴリズムを開発をする必要がなくなり，研究者に限らず，脆弱性解析に詳しくないユーザでもアルゴリズム開発に着手しやすくなり，今後の IoT 機器のセキュリティ対策の底上げに貢献できる．

8.4　解析カバレッジ

8.4.1　評価

解析カバレッジは，どれだけ網羅的にファームウェアを解析できるかの指標となり，ミドルウェアの要件として重要である．提案ツールを用いて Karonte のアルゴリズムを搭載した解析ツールと Karonte の解析カバレッジを比較した．表 8.4 に，提案ツールと Karonte が解析した ELF ファイル数を示す．解析対象は実製品のファームウェアとし，3 ベンダー 16 個のファームウェアを解析した．解析時間を計測したコンピュータの性能は，OS が Ubuntu 22.04.02，CPU が i9-9900K，RAM16GB である．

結果から，16 個全てのファームウェアにおいて，提案ツールの解析ファイル数が多いことが示された．

8.4.2　考察

提案ツールが Karonte よりも解析ファイル数が多かった理由として，ELF ファイルの収集方法が異なることがあげられる．Karonte は，Linux の find コマンドを用いて，ファームウェアの構成ファイルから ELF ファイルを取り出している．しかし，Karonte の取得方法では取りこぼしがあり，ELF ファイルである Linux Kernel Module（.ko）や shared library(.so) の一部が正常に取得できていないことがわかった．さらに，解析対象ではない画像ファイル（.png, .bmp）やテキストファイル（.xml, .js, .conf）ファイルも取得しており，False Positive

表 8.4: 解析した ELF ファイルの数

Vendor	Firmware Name	Unpacked Firmware Size(MB)	YODA	Karonte
Tenda	FH-1201	35	103	62
Tenda	FH-1206	35	103	62
Tenda	WH-450	37	149	134
TP_Link	Archer_C50	53	123	88
TP_Link	Archer_C2	54	124	89
TP_Link	Archer_C20	54	136	89
TP_Link	TD-W8970	58	175	108
TP_Link	TD-W9970	57	150	95
TP_Link	TL-MR3020	39	165	77
TP_Link	TL-MR3040	28	165	77
TP_Link	TL-WA701ND	33	153	72
TP_Link	TL-WA830RE	28	145	72
TP_Link	TL-WR1043ND	39	190	84
D-Link	DWR-118	40	226	180
D-Link	DIR-826	46	128	101
D-Link	DIR-842	38	91	79

が多いことがわかった．例えば，D-Link 製品は WEB ブラウザで動作する管理システムを提供しており，管理画面を構成するプログラムや画像などが収集されていた．False Positive のファイル，つまり解析対象外のファイルを取得することで，次工程の逆解析フェーズにおいて読み込みエラーが発生し，解析が停止する恐れがあるため，False Positive は発生しないことが望ましい．

一方，提案ツールはファームウェアの構成ファイルを一つずつ判定し，ELF ファイルと判定したのみを取り出している．そのため，網羅的に ELF ファイルを抽出し，且つ False Positive を発生させずに解析を実行できた．以上から，提案ツールを活用することで解析カバレッジが向上することがわかり，提案ツールの有用性が確認できた．

8.5　スケーラビリティ

8.5.1　評価

スケーラビリティは，ミドルウェアがどれだけ様々な脆弱性検出アルゴリズムを搭載できるか能力を示す指標である．実装可能な脆弱性検出アルゴリズムの数を，提案ツールと Karonte で実際にプログラムを実装し，搭載できるアルゴリズム数を比較した．

表 8.5 に結果を示す．提案ツールが 5 つの脆弱性検出アルゴリズムを搭載できるのに対し，Karonte の搭載数は 2 つであった．1 つ目はメモリの破損を検出するアルゴリズムである．不正なメモリの書き換えや操作によってデータを破壊する脆弱性を検知する．これは IoT だけでなく汎用コンピュータにも影響を及ぼすため，最も重要な脆弱性の一つといえる．2 つ目は DoS 攻撃の被害をうける可能性があるプログラムの検出である．DoS 攻撃は，IoT 機器の脆弱性を悪用した大規模な Mirai 攻撃の原因でもあり，重要な検知対象である．3 つ目は，ログイン情報の埋め込みです．ログイン情報である ID やパスワードがファームウェアに直接書き込まれており，ログイン情報を知っている第三者が誰しもファームウェアにログインできてしまうため，重要な検出対象である．

表 8.5: 搭載できた脆弱性検出アルゴリズムの比較

Vulnerability	YODA	Karonte
メモリ破損	✓	✓
DoS 攻撃	✓	✓
埋め込みログイン情報 (String Search, Socket Search, User Input Search)	✓	-

8.5.2 考察

スケーラビリティにおいて，本ツールのほうが Karonte よりも搭載できる脆弱性検出アルゴリズムが多いことがわかった．Karonte が埋め込みログイン情報の検知アルゴリズムに対応できない理由は，検出に必要な機能が内蔵されていないためである．この脆弱性を検知するためには，プログラムからパスワードとなる文字列を検出するしくみが必要である．しかし，Karonte のプログラムを確認したところ，文字列検出の機能がコメントアウトされており，利用できない状態となっていた．コメントアウトされたプログラムを実行しようとしたところ，エラーが発生し，プログラムを実行することができなかった．このような理由から，Karonte では埋め込みログイン情報の検出に対応していない．一方，提案ツールは文字列解析の API を提供しているため，埋め込みログイン情報の検出アルゴリズムの実装が可能である．

また，Karonte は，ファームウェアの入出力プログラムが複雑であるため，脆弱性検出アルゴリズムの組み込みや機能拡張に課題がある．例えば，Karonte には解析結果を出力するログプログラムがいくつか含まれているが，特定の結果ファイルに出力する方法や，どのプログラムを呼び出すべきなのか不明確であった．こういったプログラムの複雑さは，脆弱性検知アルゴリズムを追加することを困難する理由になる．

Karonte の優れた点として，マルチバイナリ解析機能がある．提案ツールはシングルバイナリ解析のみの対応となるため，今後はマルチバイナリ解析も対応できるように拡張する．

8.6 まとめ

提案ツールをミドルウェアとして展開できるかの可能性を検討するため，既存研究の共通機能を洗い出したミドルウェアを提案し，コードの削減数，および解析カバレッジとスケーラビリティについて既存研究を比較を行った．結果として，すべての結果において提案ツールが優れていることを確認した．今後は，提案ツールのミドルウェア化と普及にむけ，逆解析結果の API の拡充，マニュアルの改善および多言語化をすすめる．また，今後提案される解析ツールのキャッチアップを継続し，ミドルウェアよって解決できる課題については，要件と機能追加を検討し，ミドルウェアのさらなる発展につなげる．

第9章　結論

本章では，本研究の課題，提案内容，評価内容についてまとめる．そして最後に，今後の課題について述べる．

9.1　まとめ

埋め込みログイン情報検出の手法提案では，IoT 機器のファームウェアに含まれる埋め込みログイン情報を検出するため，3 つの手法を提案した．まず，1 つ目の Stringe Search では，埋め込みログイン情報によく使用される文字列比較関数に着目し，文字列比較関数を使用する関数を埋め込みログイン情報の候補とする手法を提案した．2 つ目の Socket Search では，遠隔ログインはネットワーク機能関数を経由することを踏まえ，`socket` シンボルを参照している関数の周辺で文字列比較関数を使用する関数を埋め込みログイン情報の候補とする手法を提案した．3 つ目の User Input Search では，文字列比較関数の引数となる変数がユーザーの入力値である場合，埋め込みログイン情報の候補として出力する手法を提案した．各手法の検出能力を確かめるため，実際に埋め込みログイン情報の脆弱性が報告されている IoT 機器のファームウェア 6 件を収集し，7 件の脆弱性を検出できるか評価した．ベースラインの Whole Search を含む 4 手法を適用した．Socket Search は，Tenda Router において最も F-score が高く，ネットワーク機能周辺に埋め込みログイン情報が含まれている場合は有用な手法であることがわかった．User Input Search は，脆弱性を 7 件中 5 件のファームウェアにおいて F-score が最も高く，未知の脆弱性も発見したことから，全手法の中で最も有用な手法だといえる．現状は，User Input Search と Socket Search を併用することで，高い精度と検出能力を維持できるといえる．

埋め込みログイン情報検出ツールの提案と構築では，既存の IoT 脆弱性検出ツールで課題となっている，前処理作業と環境構築の複雑さ，および解析時間の長時間化を解決する

ため，埋め込みログイン情報検出ツールの構築と評価を行った．ツールを構築するために，まず要件定義を実施した．要件定義では，既存の解析ツールの課題，一般的なミドルウェアの定義，類似分野かつミドルウェアの提案が盛んな IoT 環境のミドルウェア要件を洗い出し，提案ツールに必要な要件を定義した．ツールの構築では，要件を満たすツールを実現するために，4 つの新規技術開発と 1 つの環境構築に関する工夫を取り入れた．評価では，環境構築，前処理作業，および解析時間について，ベースラインと既存研究のうち最も支持されている Karonte と比較し，提案ツールが要件を満たすか評価した．結果として，提案ツールは Karonte と比較したときに，すべての作業項目において提案ツールが性能を上回ることが確認でき，本研究では要件を満たすツールの開発に成功した．

また，提案ツールをミドルウェアとして展開できるか検討するため，既存研究の共通機能を洗いだし，ミドルウェアの構築を行った．そして，既存研究と比較したときのコード削減数，解析カバレッジ，及びスケーラビリティについて既存研究を比較を行い，提案ツールがよりミドルウェアとして優れていることを確認し，ミドルウェア化の展望を示した．

9.2 今後

今後は，User Input Search が対象とする変数の種類を増やし，検出精度の向上につとめる．埋め込みログイン情報検出ツールの提案と構築では，ツールのミドルウェア化と普及にむけ，逆解析結果の API の拡充，マニュアルの改善および多言語化をすすめる．また，今後提案される解析ツールのキャッチアップを継続し，ミドルウェアよって解決できる課題については，要件と機能追加を検討し，ミドルウェアのさらなる発展につなげる．

謝辞

本研究にあたり，ご多忙の中適切なご指導をくださった大須賀 昭彦 教授，田原 康之准教授，清 雄一教授に感謝いたします．先生方とは，修士号を取得して就職した後も，業務や研究活動などを通して交流させていただきました．

博士前期課程の入学から今日まで，研究だけではなく，考え方や価値観の面でも大きな学びと刺激を受けました．先生方のお言葉とお気遣いに何度励まされたかわかりません．日々の業務を続ける中で，また研究活動に励みたいと悩んでいたところ，このように再び快く迎え入れてくださり本当にありがとうございました．

更に，実験など様々な協力をしてくださった研究室の先輩，同期，後輩，秘書の五領田 令奈さんに感謝の意を表します．他にも，投稿した論文等に対して，国内外の多くの査読者から様々なコメントをいただき，よりよい研究に向けて日々取り組むことができました．

そして，審査を快く引き受けてくださいました大学院 情報理工学研究科の南 泰浩 教授，広田 光一 教授，大坐畠 智 教授に感謝申し上げます．先生方には，論文のまとめ方や技術の評価方法などに関して多大なご指導をいただきました．

最後に，様々な面で支えてくださった大切な家族と友人に心から御礼申し上げます．

参考文献

[1] Mohammed Alsheikh, Liam Konieczny, Michael Prater, Gabe Smith, and Suleyman Uludag. The state of iot security: Unequivocal appeal to cybercriminals, onerous to defenders. *IEEE Consumer Electronics Magazine*, Vol. 11, No. 3, pp. 59–68, 2022.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *Proc. the 26th USENIX Security Symposium*, Vancouver, Canada, Aug. 2017.

[3] Nigel Bevan, James Carter, and Susan Harker. Iso 9241-11 revised: What have we learnt about usability since 1998? In Masaaki Kurosu, editor, *Human-Computer Interaction: Design and Evaluation*, pp. 143–151, Los Angeles, CA, USA, Aug. 2015. Springer.

[4] Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. Firmadyne. `https://github.com/firmadyne/firmadyne`. accessed on May. 05. 2023.

[5] Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *Proc. 23rd Annual Network and Distributed System Security Symposium*, San Diego, USA, Feb. 2016.

[6] Kai Cheng, Qiang Li, Lei Wang, Qian Chen, Yaowen Zheng, Limin Sun, and Zhenkai Liang. Dtaint: Detecting the taint-style vulnerability in embedded device firmware. pp. 430–441, Luxembourg, Luxembourg, 2018.

[7] Lucian Cojocar, Jonas Zaddach, Roel Verdult, Herbert Bos, Aurélien Francillon, and Davide Balzarotti. PIE: parser identification in embedded systems. In *Proc. the 31st Annual Computer Security Applications Conference*, pp. 251–260, Los Angeles, USA, Dec. 2015.

[8] Jeremy Collake. Firmware mod kit. `https://github.com/amitv87/firmware-mod-kit`. accessed on May. 05. 2023.

[9] Yaniv David, Nimrod Partush, and Eran Yahav. Firmup: Precise static detection of common vulnerabilities in firmware. In *Proc. the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, p. 392–404, New York, USA, Mar. 2018.

[10] Xiaotao Feng, Xiaogang Zhu, Qing-Long Han, Wei Zhou, Sheng Wen, and Yang Xiang. Detecting vulnerability on iot device firmware: A survey. *IEEE/CAA Journal of Automatica Sinica*, Vol. 10, No. 1, pp. 25–41, 2023.

[11] Pietro Ferrara, Amit Kr Mandal, Agostino Cortesi, and Fausto Spoto. Static analysis for discovering iot vulnerabilities. *Int. J. Softw. Tools Technol. Transf.*, Vol. 23, No. 1, p. 71–88, 2021.

[12] Teenu S. John, Tony Thomas, and Sabu Emmanuel. Graph convolutional networks for android malware detection with system call graphs. In *Proc. Third ISEA Conference on Security and Privacy*, pp. 162–170, Guwahati, India, Feb. 2020.

[13] The National Security Agency of the United States. Ghidra. `https://ghidra-sre.org/`. accessed on May. 05. 2023.

[14] Chengbin Pang, Ruotong Yu, Dongpeng Xu, Eric Koskinen, Georgios Portokalidis, and Jun Xu. Towards optimal use of exception handling information for function detection. In *Proc. 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 338–349, Online, 2021.

[15] The Open Web Application Security Project. Owasp-iot-top-10-2018. https://www.owasp.org/images/1/1c/OWASP-IoT-Top-10-2018-final.pdf. accessed on May. 05. 2023.

[16] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, Vol. 3, No. 1, pp. 70–95, 2016.

[17] Nilo Redini, Aravind MacHiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Karonte. https://hub.docker.com/r/badnack/karonte. accessed on May. 05. 2023.

[18] Nilo Redini, Aravind MacHiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Karonte: Detecting insecure multi-binary interactions in embedded firmware. In *Proc. 2020 IEEE Symposium on Security and Privacy*, pp. 1544–1561, Online, May. 2020.

[19] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware. In *Proc. 22rd Annual Network and Distributed System Security Symposium*, San Diego, USA, Feb. 2015.

[20] Prashast Srivastava, Hui Peng, Jiahao Li, Hamed Okhravi, Howard Shrobe, and Mathias Payer. Firmfuzz. https://github.com/HexHive/FirmFuzz. accessed on May. 05. 2023.

[21] Prashast Srivastava, Hui Peng, Jiahao Li, Hamed Okhravi, Howard Shrobe, and Mathias Payer. Firmfuzz: Automated iot firmware introspection and analysis. In *Proc. the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, p. 15–21, London, United Kingdom, Nov. 2019.

[22] Sam L. Thomas, Tom Chothia, and Flavio D. Garcia. Humidify. https://github.com/BaDSeED-SEC/HumIDIFy. accessed on May. 05. 2023.

[23] Sam L. Thomas, Tom Chothia, and Flavio D. Garcia. Stringer. `https://github.com/BaDSeED-SEC/strngr`. accessed on May. 05. 2023.

[24] Sam L. Thomas, Tom Chothia, and Flavio D. Garcia. Humidify: A tool for hidden functionality detection in firmware. In *Proc. 24rd Annual Network and Distributed System Security Symposium*, pp. 279–300, San Diego, USA, Feb. 2017.

[25] Sam L. Thomas, Tom Chothia, and Flavio D. Garcia. Stringer: Measuring the importance of static data comparisons to detect backdoors and undocumented functionality. In *Proc. 22nd European Symposium on Research in Computer Security*, pp. 513–531, Copenhagen, Denmark, Sept. 2017.

[26] Vikash, Lalita Mishra, and Shirshu Varma. Middleware technologies for smart wireless sensor networks towards internet of things: A comparative review. *Wireless Personal Communications*, Vol. 116, pp. 1539–1574, Feb. 2021.

[27] Joobeom Yun, Fayozbek Rustamov, Juhwan Kim, and Youngjoo Shin. Fuzzing of embedded systems: A survey. *ACM Comput. Surv.*, Vol. 55, No. 7, pp. 1–33, 2022.

[28] Jingbin Zhang, Meng Ma, Ping Wang, and Xiao dong Sun. Middleware for the internet of things: A survey on requirements, enabling technologies, and solutions. *Journal of Systems Architecture*, Vol. 117, p. 102098, 2021.

[29] Zhuo Zhang, Yapeng Ye, Wei You, Guanhong Tao, Wen-chuan Lee, Yonghwi Kwon, Yousra Aafer, and Xiangyu Zhang. Osprey: Recovery of variable and data structure via probabilistic analysis for stripped binary. In *Proc. the 42nd IEEE Symposium on Security and Privacy*, pp. 813–832, San Francisco, USA, May. 2021.

研究業績

学術雑誌

1. 櫻庭秀次，**依田みなみ**，清雄一，田原康之，大須賀昭彦：送信ドメイン認証を用いた送信者レピュテーションの構築手法とフィードバックループの提案，情報処理学会論文誌，Vol.64,No.1,pp.13-23，2023 年 1 月．

2. **Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, :Detection of the hardcoded login information from socket and string compare symbols(AETiC), 2021 Annals of Emerging Technologies in Computing, vol.5, no.1, pp.28-39, 2021.

3. 櫻庭秀次，**依田みなみ**，清雄一，田原康之，大須賀昭彦：送信ドメイン認証を用いた送信者レピュテーション構築手法の提案，情報処理学会論文誌, Vol.62, No.5, 1173-1183, 2021 年 5 月．

国際会議

4. **Minami Yoda**, Shigeo Nakamura, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : A Scalable Middleware for IoT Vulnerability Detection, Proceedings of 26th IEEE/ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter 2023), Springer, December 2023.

5. **Minami Yoda**, Shigeo Nakamura, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : A Middleware to Improve Analysis Coverage in IoT Vulnerability Detection, Proceedings of

The 2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS 2023), IEEE, November 2023.

6. **Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : Detecting hardcoded login information from user input, Proceedings of IEEE 40th International Conference on Consumer Electronics (ICCE), pp.104-105, IEEE, January 2022.

7. **Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : Proposal of a middleware to support development of IoT firmware analysis tools, Proceedings of the 14th International Joint Conference on Knowledge-Based Software Engineering (JCKBSE2022), pp.3-14, IEEE, August 2022.

8. **Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : Detection of plaintext login information in firmware, Proceedings of 2022 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW), pp.1-2, July 2022.

9. Shuji Sakuraba, **Minami Yoda**,Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : Sender Reputation Construction method using Sender Authentication, Proceedings of 1st IEEE International Conference on Data Science and Computer Application (ICDSCA 2021), pp.369-373, IEEE, December 2021.

10. Shuji Sakuraba, **Minami Yoda**,Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : Improvement of Legitimate Mail Server Detection Method using Sender Authentication, Proceedings of 18th IEEE/ACIS International Conference on Software Engineering, Management and Applications (SERA 2021), pp.10-14, IEEE, June 2021.

11. **Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga, : Detection of the Hardcoded Login Information from Socket Symbols ,Proceedings of 3rd IEEE International Conference on Computing, Electronics & Communications Engineering, Electrical & Communication Engineering (IEEE iCCECE '20), pp.33-38, IEEE, July 2020.

関連論文の印刷公表の方法及び時期

学術雑誌

1. 全著者名：**Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga
   論文題目：Detection of the hardcoded login information from socket and string compare symbols
   印刷公表の方法及び時期：2021 Annals of Emerging Technologies in Computing(AETiC), vol.5, no.1, pp.28-39, 2021.
   （第 2 章及び第 4 章及び第 5 章に関連）

国際会議

2. 全著者名：**Minami Yoda**, Shigeo Nakamura, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga
   論文題目： A Scalable Middleware for IoT Vulnerability Detection
   印刷公表の方法及び時期：Proceedings of 26th IEEE/ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter 2023), Springer, December 2023.
   （第 3 章及び第 6 章及び第 8 章に関連）

3. 全著者名：**Minami Yoda**, Shigeo Nakamura, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga
   論文題目： A Middleware to Improve Analysis Coverage in IoT Vulnerability Detection
   印刷公表の方法及び時期：Proceedings of The 2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS 2023), IEEE, November 2023.
   （第 3 章及び第 6 章及び第 8 章に関連）

4. 全著者名：**Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga
論文題目： Detecting hardcoded login information from user input
印刷公表の方法及び時期:Proceedings of IEEE 41st International Conference on Consumer Electronics (ICCE), pp.104-105, IEEE, October 2022.
（第 4 章及び第 5 章に関連）

5. 全著者名：**Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga
論文題目： Proposal of a middleware to support development of IoT firmware analysis tools
印刷公表の方法及び時期：Proceedings of the 14th International Joint Conference on Knowledge-Based Software Engineering (JCKBSE2022), pp.3-14, Springer, Aug 2022.
（第 8 章に関連）

6. 全著者名：**Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga
論文題目： Detection of plaintext login information in firmware
印刷公表の方法及び時期：Proceedings of 2022 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW), pp.1-2, July 2022.
（第 4 章及び第 5 章に関連）

7. 全著者名：**Minami Yoda**, Shuji Sakuraba, Yuichi Sei, Yasuyuki Tahara, Akihiko Ohsuga
論文題目： Detection of the Hardcoded Login Information from Socket Symbols
印刷公表の方法及び時期：Proceedings of 3rd IEEE International Conference on Computing, Electronics & Communications Engineering, Electrical & Communication Engineering (IEEE iCCECE ’20), pp.33-38, IEEE, July 2020.
（第 4 章及び第 5 章に関連）

本論文との関連の詳細

章	節	関連論文番号	関連する内容
2 章	2.5〜2.6 節	1	埋め込みログイン情報の定義
3 章		2, 3	IoT 機器の脆弱性検出ツールの現状
4 章		1, 4, 6, 7	埋め込みログイン情報の検出手法の提案
5 章		1, 4, 6, 7	埋め込みログイン情報の評価
6 章		2, 3	脆弱性検出ツールの提案と構築
8 章		2, 3, 5	既存ツールの共通機能の調査，ミドルウェアの評価

著者略歴

依田 みなみ（よだ みなみ）

1990 年 11 月 22 日	長野県千曲市（旧：更埴市）に生まれる
2010 年 3 月	長野県立 長野西高等学校 卒業
2010 年 4 月	東京工科大学 メディア学部 メディア学科 入学
2014 年 3 月	東京工科大学 メディア学部 メディア学科 卒業
2014 年 4 月	国立電気通信大学 大学院 情報システム学研究科 社会知能情報学専攻 博士前期課程 入学
2017 年 3 月	国立電気通信大学 大学院 情報システム学研究科 社会知能情報学専攻 博士前期課程 入学
2019 年 10 月	国立大学法人 電気通信大学 大学院 情報理工学研究科 情報学専攻 博士後期課程 入学
2024 年 3 月	国立大学法人 電気通信大学 大学院 情報理工学研究科 情報学専攻 博士後期課程 修了予定