Vol.2021-CSEC-94 No.2
Vol.2021-SPT-43 No.2
2021/7/19

IoT機器のWebUIを模したハニーポットの自動生成フレームワーク

山本 萌花[1,a)]　掛井 将平[1]　齋藤 彰一[1]

概要：IoT 機器に搭載される Web インターフェース（組み込み WebUI）を狙った攻撃を観察するために，実システムの挙動を模倣して攻撃情報を収集するハニーポットが利用される．組み込み WebUI は動作がベンダーの実装に依存することや，ログイン機能を有することを特徴とする．よってハニーポットの作成においては，WebUI を備えた様々な IoT 機器のエミュレーション環境の用意と，ログインを前提とした挙動再現が求められる．本研究ではこれらを達成する，組み込み WebUI を模したハニーポットを自動生成するフレームワークを提案する．コンテナ仮想化技術と QEMU によって IoT 機器のファームウェアを動作させ，起動した WebUI の挙動を機械学習によって再現する．オープンソースである OpenWrt をベースとした無線 LAN ルータを対象として実験を行い，提案手法によって 9 種類の CPU アーキテクチャおよび 7 ベンダーの WebUI を模したハニーポットを生成できることを確認した．

A Framework for Automatic Generation of Honeypots Emulating Embedded Web Interface

Abstract: Honeypots, mimicking the behavior of real systems, are used to collect attack information to observe attacks on Web interfaces (embedded WebUI) installed in IoT devices. The embedded WebUI is characterized by the fact that its behavior depends on the vendor's implementation and a login function. Therefore, there are challenges in creating a honeypot: preparing an emulation environment for various IoT devices with WebUIs and reproducing the behavior of WebUIs that assume login. This study proposes a framework for automatically generating honeypots that emulate embedded WebUIs to solve these problems. We use container virtualization technology and QEMU to run the firmware of IoT devices and reproduce the behavior of the launched WebUI using a machine learning model. We conducted experiments on a wireless LAN router based on the open-source OpenWrt. We confirmed that the proposed method could generate honeypots that mimic nine different CPU architectures and seven vendor's WebUIs.

1. はじめに

IoT 機器は低価格化と機能向上が大きく進んでいる．例えば，多くの IoT 機器にはブラウザを通じて設定変更を行うための Web インターフェース（組み込み WebUI）が用意されており，IoT 機器の状態をインターネットから確認できる．組み込み WebUI は機器の種類やベンダーによって特有の実装が行われているために静的解析や動的解析による脆弱性の発見が困難であり，セキュリティ対策が行われにくい [1]．そのため，IoT 機器に対するサイバー攻撃の中でも組み込み WebUI に存在する脆弱性が狙われている．

このような状況において，正規のシステムを装って攻撃者の侵入および操作を監視するハニーポットは，脆弱性分析や脅威分析を行う手法として有用である．ハニーポットは対話レベルによって高対話型と低対話型に分類される．高対話型ハニーポットは脆弱性を持つ実際のアプリケーションを公開して攻撃を観察するため，高度な情報を獲得できる．しかし安全性に問題があり，高対話型ハニーポットがマルウェアに乗っ取られ，DDoS 攻撃に加担した事例がある [2]．対して，低対話型ハニーポットはアプリケーションの挙動を模倣したシステムである．再現した範囲の限られた応答しかできないために攻撃者にハニーポットであることを検出されやすいが，攻撃コマンドをハニーポットで実行することがないために安全であり，本研究では低対話型ハニーポットに注目する．

[1] 名古屋工業大学
Nagoya Institute of Technology
[a)] 32414105@stn.nitech.ac.jp

Web を対象にした低対話型ハニーポットは，実際のアプリケーションロジックは実装されず，受信した HTTP リクエストに対して適切な HTTP レスポンスを返す仕組みが実装される．例えば，一般的な WebUI を模した低対話型ハニーポットは 3 段階で作成される [3]．はじめに，模倣対象となる WebUI を用意する．次に，WebUI を操作して発生する通信を記録する．そして，記録した通信を基に，受信したリクエストに対する応答ルールを設定する．一方，組み込み WebUI の場合はそれぞれの段階で課題が生じる．

1 つ目の課題は，組み込み WebUI のエミュレート環境を用意することである．組み込み WebUI にはハードウェアに関する情報が表示されるため，実機を利用することが望ましいが，多種にわたる実機の購入は経済的負担となる．そこで，ベンダーが配布する IoT 機器のファームウェアを起動して組み込み WebUI を用意する手法 [4], [5] が提案されているが，いずれも動的解析に特化されている．

2 つ目の課題は，組み込み WebUI に存在する様々な機能を操作することである．例えば，組み込み WebUI は認証機能を備えており，ログインを行うことで IoT 機器の設定を変更するページへ到達する．しかし，認証や設定変更の操作は複雑であり，既存手法 [3], [6] ではこれらの機能を再現していない．

3 つ目の課題は，組み込み WebUI の実装毎に最適な応答ルールを設定することである．既存手法では研究者の経験則に基づく応答ルールが設定され，攻撃に対して固定された対応が実施される．しかし，組み込み WebUI の挙動は多様であり [7]，その実装に合わせて逐一応答ルールを用意するのは労力を要する．

以上より，組み込み WebUI のハニーポットを作成するには，実装方法に制限されることなく WebUI の機能を再現するための手法が求められる．現状，この条件を満たすアプローチは存在しない．

そこで本研究では，組み込み WebUI の低対話型ハニーポットを自動生成するフレームワークを提案する．提案フレームワークは，ファームウェアを動作させるためのコンテナイメージを作成し，組み込み WebUI をコンテナ上で実行する．そして，組み込み WebUI へのログイン操作およびページ探索を自動化するブラウザ操作ツールによって発生させた通信を，自然言語処理における対話モデルに学習させる．すなわち，組み込み WebUI の応答ルールを学習した対話モデルが攻撃者とやり取りを行う．

提案フレームワークを実装し，オープンソースである OpenWrt[8] をベースとした無線 LAN ルータを対象として実験を行ったところ，提案手法によって 9 種類の CPU アーキテクチャおよび 7 ベンダーの組み込み WebUI を模したハニーポットを生成できることを確認した．また，生成されたハニーポットをインターネットに公開した結果，ログイン試行などの WebUI を操作する攻撃を観測した．

本研究の貢献は以下の通りである．

1. 実機同等のパフォーマンスで動作するファームウェアのエミュレート環境を用意し，組み込み WebUI を模したハニーポットの作成を効率化する手法を提案する．
2. 提案手法によって作成したハニーポットをインターネットに公開し，対話モデルが模倣する IoT 機器を狙った攻撃を観測した．

本稿では，第 2 章でハニーポット作成における課題を関連研究とともに説明する．次に，第 3 章で課題の解決方針および提案について述べる．そして，第 4 章で評価結果を示し，第 5 章で考察と今後の課題を述べる．第 6 章で倫理的配慮について述べ，最後に全体のまとめを行う．

2. 関連研究

本章では，関連研究の説明とともに，組み込み WebUI のハニーポット作成における課題を述べる．

2.1 組み込み WebUI の用意

組み込み WebUI は IoT 機器のファームウェアに搭載される Web アプリケーションによって提供される．ファームウェアはカーネルやファイルシステムから構成されており，圧縮された形でベンダーから配布される．ファイルシステムに搭載されたアプリケーションは，GPIO ピンや機器の構成情報が書き込まれた不揮発性メモリなどのハードウェアにアクセスすることが特徴である．そのため，CPU や周辺機器をエミュレートできる QEMU[9] を利用して，ファームウェアを動作させる研究が行われている．

QEMU を利用するアプローチには，カーネルを含めたシステム全体をエミュレートするフルシステムエミュレーションと，ユーザランドを非特権モードでエミュレートするユーザモードエミュレーションがある．既存のファームウェアエミュレータの多くは脆弱性分析を目的としており，フルシステムエミュレーションを採用している．その中でも代表的なツールである Firmadyne[4] は，ブートプロセスを通じてファームウェアの各アプリケーションを起動する．しかし，Firmadyne はエミュレート成功率の低さが指摘されており，Honware[2] と FirmAE[5] は経験則的に対処することで成功率を上げている．

これらのエミュレータの共通点は，CPU アーキテクチャが MIPS の big および little エンディアン，ARM の little エンディアンのファームウェアを対象とすることである．我々の調査では，PowerPC や MIPS64 のファームウェアが存在するため [10]，これらのエミュレータの汎用性は低い．また，FirmAFL[11] の評価によると，フルシステムエミュレーションはユーザモードエミュレーションに比べて実行速度が約 10 倍低速であることが示されている．本提案においてこの性質はハニーポット作成時間の増加に繋がる．

2.2 通信の収集

組み込み WebUI の挙動再現では，攻撃者による事前調査が行われてもハニーポットであることを検出されないような HTTP リクエストと HTTP レスポンスの対応を獲得すべきである．攻撃者はアクセスしたシステムの正体を知るために，Web アプリケーション上の様々なパスへアクセスすることで内部構造を調べる [12]．また，攻撃者は WebUI を操作して得られる応答の内容から，そのシステム上で動作するプログラムの種類やバージョン，OS などの情報を特定する．例えば，HTTP レスポンスに含まれる認証情報を参照して脆弱性情報を獲得する行為が観測されている [6]．このような偵察行為に対応するために，Web アプリケーション上に存在するすべてのパスを発見することに加え，画面遷移や認証などの機能を再現する必要がある．

2.3 応答ルールの設定

低対話型ハニーポットに設定される応答ルールは，ハニーポットが受信した HTTP リクエストの内容を分析し，事前の通信収集で記録した中から最適な HTTP レスポンスを選択する．すなわち，低対話型ハニーポットの偽装性は応答ルールの緻密さによって決まる．

既存の応答ルールとして，HTTP リクエストに含まれるメソッドやヘッダなどの各要素に重要度を付けて HTTP レスポンスを決定する手法 [3] や，ハニーポットが受信した HTTP リクエストをデータベースに問い合わせる形式に変換して HTTP レスポンスを検索する手法 [13] が提案されている．しかし，複雑な挙動をする組み込み WebUI の場合，経験則による応答ルールでは正確な挙動を再現できない．さらに，既存手法では未知の HTTP リクエストに対する応答が 404-not-found. などのエラーメッセージに固定されている．この場合，攻撃者から脆弱性がないと判断されて攻撃対象から除外されることがある [14].

一方，ルールベースではない方式として，機械学習を利用したアプローチが提案されている．機械学習によって攻撃に対して最適な応答をするモデルが作成され，未知の HTTP リクエストを受信した場合にも柔軟に対応する．IoTCandyJar[6] は，インターネットスキャンにより収集した HTTP レスポンスの中から攻撃者が期待しているものを見つける応答選択問題にマルコフ決定プロセスモデルを利用する．また，自己適応型と呼ばれるハニーポットは，ゲーム理論や強化学習を利用して攻撃者からのコマンドに対して実行するアクションを決定する [15]．これらのアプローチは攻撃者との一連のやり取りを学習する．よって，実際の攻撃を再現したデータセットを用意しなければならないが，攻撃手法を事前に想定することは難しい．

3. 提案

本章では，IoT 機器のファームウェアを利用してハニーポットを自動生成するフレームワークについて述べる．

3.1 設計目標

本研究では，多種多様な組み込み WebUI の挙動を正確に再現したハニーポットを生成するフレームワークを用意することで，多様な攻撃を効率よく収集することを目指す．そのために，フレームワークの設計目標として次に示す 3 つを掲げる．

- **汎用性**：CPU アーキテクチャやベンダーの実装に依存せずにハニーポットが生成されること
- **偽装性**：模倣対象の組み込み WebUI を正確に再現したハニーポットが生成されること
- **効率性**：利用者の手動操作を必要とせずに短い時間でハニーポットが生成されること

3.2 対象とするファームウェア

世の中には様々な機種の IoT 機器が存在する．本稿では実験対象の IoT 機器として，家庭用として広く普及しており，また攻撃対象ともなっている無線 LAN ルータを選択する．さらに，無線 LAN ルータの中でも OpenWrt[8] をベースとして作成されたファームウェアを実験対象とする．OpenWrt は組み込み機器を対象としたオープンソースの Linux ディストリビューションであり，多くのベンダーが OpenWrt をベースとしたファームウェアを GPL コードとして公開している．

3.3 組み込み WebUI を模した低対話型ハニーポットの自動生成フレームワーク

組み込み WebUI を対象とした低対話型ハニーポットを作成するためには，組み込み WebUI の用意，通信の収集，応答ルールの設定の 3 つの段階を必要とする．組み込み WebUI の用意では，ユーザモードエミュレーションを利用したファームウェア実行環境を構築し，高い汎用性を実現する．通信の収集では，挙動再現に必要なログイン操作とページ探索を行うことでハニーポットの偽装性を向上する．さらに応答ルールの設定では，自然言語処理における対話モデルを応用することで偽装性を向上する．提案フレームワークではそれぞれの段階を，**起動段階**，**通信収集段階**，**学習段階**と呼び，すべての処理を自動化することで効率性を達成する．提案フレームワークの全体像を**図 1** に示す．利用者はファームウェアを手元に用意し，フレームワークに投入することで自動的にハニーポットが生成される．生成されたハニーポットには組み込み WebUI の挙動を学習した対話モデルが組み込まれ，攻撃者とやり取りを行う．

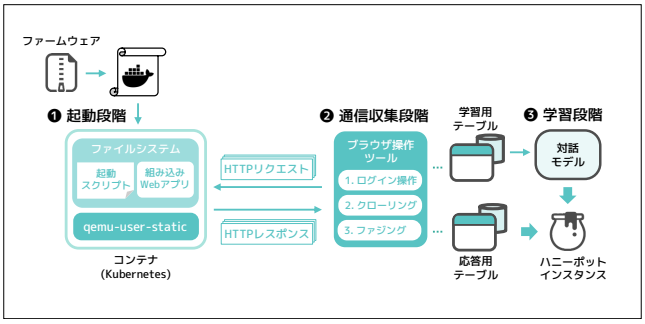

図 1 提案フレームワークの全体像
Fig. 1 Overview of our framework.

3.3.1 起動段階

組み込み WebUI を用意するために，コンテナを利用してファームウェアに搭載された Web アプリケーションを起動する．コンテナはホストマシンとは隔離されたネットワークやファイルシステムを持つ軽量な仮想環境であり，環境設定や起動時の動作を指定したコンテナイメージから作成される．提案手法では，ファームウェアから抽出したファイルシステムと QEMU ユーザモードエミュレータを含めたコンテナイメージを作成し，コンテナ起動時に Web アプリケーションを起動するためのスクリプトを実行するように指定する．このスクリプトは OpenWrt が提供するファームウェアのブートプロセスを解析して作成したものであり，ブートプロセスで発生するネットワーク設定や周辺機器へのアクセスを避けて Web アプリケーションを起動する．そして，Linux カーネルの機能である binfmt_misc を利用して，実行ファイルの形式に合わせて QEMU ユーザモードエミュレータを起動するように設定することで，コンテナ内はホストマシンと異なる CPU アーキテクチャの実行ファイルが動作する空間となる．

本提案ではコンテナ仮想化プラットフォームとして Docker[16] を利用しており，さらにコンテナは Kubernetes[17] によって管理する．Kubernetes はコンテナの管理および運用を自動化するツールであり，通信収集中にコンテナが停止した場合は自動的に再起動する．また，大量の HTTP リクエストを送信した場合は複数のコンテナに負荷分散させるため，組み込み WebUI の並列化を実現する．

3.3.2 通信収集段階

起動段階で用意した組み込み WebUI の構造や機能を再現するために，SeleniumWire[18] を基にして作成したブラウザ操作ツールによって必要な通信を収集する．ブラウザ操作ツールが送信した HTTP リクエストと，組み込み WebUI から返される HTTP レスポンスはデータベースへ保存する．データベースへ書き込む際には，ヘッダやボディに含まれている日時や IP アドレスを削除し，ホスト側のハードウェア情報が含まれていた場合は IoT 機器の相当する値に置換する．データベースには，学習段階で利用する学習用テーブルと，ハニーポットで利用する応答用テーブルがある．**表 1** に示す学習用テーブルは HTTP リクエストと HTTP レスポンスの対応を保存する．このテーブルには HTTP リクエストのメソッド，パス，クエリ，ヘッダ，ボディと，対応する “レスポンス ID”(res_id) が含まれる．“レスポンス ID” とは，HTTP レスポンスのステータスコード，ヘッダ，ボディを含めた全体を 1 つの数値に変換したものであり，**表 2** に示す応答用テーブルにその対応を記録する．ブラウザ操作ツールは，ログイン操作を行った後に Web アプリケーション全体をクローリングし，発見した全てのパスに対してファジングを実施する．そして，これらの操作中に発生した通信をデータベースへ保存する．

表 1 学習用テーブルの例
Table 1 An example of table for learning.

method	path	query	headers	body	res_id
POST	/login.html	a=1	Conn···	Pass···	1
GET	/favicon.ico	-	Acce···	-	2
GET	/index.php	b=2	Cont···	-	3

表 2 応答用テーブルの例
Table 2 An example of table used by the responder.

res_id	status	headers	body
1	200	Set-cookie···	<html >Welcome···
2	404	Content-le···	Page Not Found.
3	503	Last-modif···	503-Service Unava···

1. ログイン操作

攻撃者がログインページにアクセスし，入力フォームにユーザ名とパスワードを指定して，送信ボタンをクリックするまでの一連の挙動を SeleniumWire によって再現する．初回利用時にユーザ名やパスワードを登録する必要がある場合は事前に指定した値を設定する．ログイン成功時に得られる Cookie はクローリングおよびファジングを行う際に利用する．

2. クローリング

アクセスした Web ページに存在するアンカーリンクを探してそのリンク先へアクセスする処理を繰り返すことでページ探索を行う．しかし，この手法では Web アプリケーション上に存在するすべてのパスを発見できない可能性がある．そこで，起動段階でファームウェアから抽出したファイルシステムを参照し，Web アプリケーションのディレクトリ以下に含まれるファイルのパス情報も利用する．

3. ファジング

HTTP リクエストの内容によって，HTTP レスポンスのヘッダやボディが変化する特徴に対応するためにファジングを実施する．事前に 5 ベンダーの組み込み WebUI を調査したところ，HTTP リクエストヘッダの内容によって HTTP レスポンスヘッダに含まれる値が変化することを確認した．よって本提案では，HTTP リクエストヘッダに変化を付けるファジングを実施する．今回ファジングの対象

表 3　変化させるリクエストのヘッダフィールド
Table 3　Request header fields for fuzzing.

フィールド	説明	値の例
Accept	受入可能なコンテンツ	text/html
Accept-Charset	受入可能な文字セット	utf-8, SJIS
Accept-Encoding	受入可能な圧縮方式	gzip, compress
Accept-Language	受入可能な言語	en-US, ja
Connection	接続情報	keep-alive, close

にしたリクエストヘッダは**表 3** に示す 5 つのフィールドであり，これらはクライアントが要求する情報を表すために HTTP レスポンスが変化する可能性があると考えた．これらのフィールドの値はインターネット番号割当機関によって定義されており，その情報に従ってフィールドの値や数を変化させる．

3.3.3　学習段階

ハニーポットが攻撃者と対話を行うことに注目し，近年機械翻訳やチャットボットで成果を上げている対話モデルを利用して，通信収集段階で記録した HTTP リクエストと HTTP レスポンスの対応を学習する．本提案では，発言に基づいて決まった応答の中から適当なものを選択する検索型対話モデルの仕組みを利用し，学習用テーブルに記録された HTTP リクエストを入力として，レスポンス ID を予測するモデルを作成する．対話モデルとして，入力文と出力文に相関のある特徴を抽出する Attention 機構を適用した Sequece-to-Sequnce モデル [19] を利用しており，そのイメージを**図 2** に示す．

学習を行う前に，HTTP リクエストのメソッド，パス，クエリ，ヘッダ，ボディに含まれる文字列を数値に置き換える．パス，クエリ，ボディのそれぞれは文字列全体を 1 つの数値に置き換えるが，ヘッダは各フィールドの Key と Value をセットにしたものを 1 つの数値に置き換える．この文字列と数値の対応を記録した辞書を用意し，ハニーポットインスタンスに搭載する．

3.3.4　ハニーポットインスタンス

フレームワークによって最終的に生成されるハニーポットインスタンスには，学習済みの対話モデル，辞書，応答用テーブル，そして Web サーバが含まれる．Web サーバが HTTP リクエストを受信すると，その HTTP リクエストを辞書に従って数値化する．次に，数値化した HTTP リクエストを学習済みモデルに入力し，レスポンス ID を予測する．そして，レスポンス ID を主キーとして応答用テーブルから該当する HTTP レスポンスを検索し，IP アドレスやホスト名をハニーポットの情報に置き換えたものを Web サーバが通信相手へ送信する．最後に，受信した HTTP リクエストと予測した HTTP レスポンスは分析のためにログに記録する．

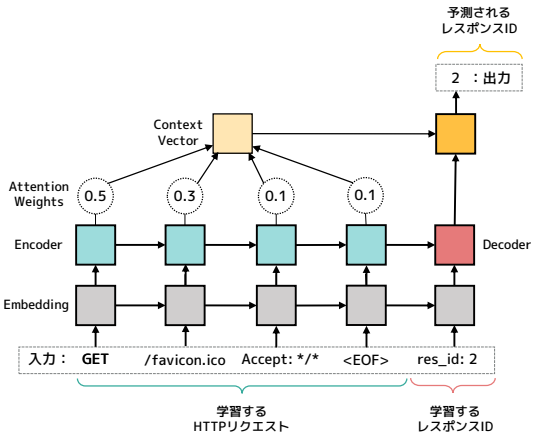

図 2　対話モデルの仕組み
Fig. 2　Our interaction model.

表 4　評価環境
Table 4　Experiment setup.

実験マシン		ソフトウェア	
OS	Ubuntu 20.04.3 LTS	QEMU	4.2.1
Kernel	Linux 5.8.0-48 x86_64	Docker	19.3.13
CPU	i7-10700K@3.80GHz	Minikube	1.17.1
GPU	GeForce RTX3060	SeleniumWire	2.1.2
Memory	32GB	TensorFlow	2.3.1

4.　評価

本章では，提案フレームワークの評価結果を示す．本提案の起動段階でファームウェアを動作させるコンテナは，ローカルに Kubernetes 環境を構築する Minikube[20] を利用して管理する．また，複数のコンテナを並列化させて通信収集を行うことができるが，評価においてはコンテナ 1 つのみを利用した．学習段階では，学習に利用するデータセットに合わせて適切なパラメータを設定した対話モデルを TensorFlow[21] によって作成し，いずれもデータセットを 50 エポック学習させた時点のモデルを評価に利用した．評価環境を**表 4** に示す．

4.1　エミュレータの性能評価

本提案で採用する，コンテナを利用したユーザモードエミュレーションがハニーポット作成に適しているか調べるために評価実験を行った．

4.1.1　エミュレータの汎用性

はじめに，用意したエミュレータが，設計目標の 1 つである汎用性を満たすことを調べるために，FirmAE が提供するデータセット [22] を利用して評価を行った．このデータセットには無線 LAN ルータと IP カメラのファームウェアが合計 1,124 個含まれている．この中から OpenWrt をベースとして作成された無線 LAN ルータのファームウェア全 58 個を評価に利用した．そして，それぞれのファー

表 5 起動に成功した FirmAE データセットに含まれるファームウェアのベンダー毎の内訳

Table 5 Vendors of successfully emulated firmware in the FirmAE dataset.

ベンダー	合計	起動成功
ベンダー A	22	19
ベンダー B	18	18
ベンダー C	12	10
ベンダー D	6	5
合計	58	52

表 6 起動に成功したファームウェアの CPU アーキテクチャ

Table 6 CPU architecture of successfully emulated firmware.

CPU アーキテクチャ	ビット	エンディアン
MIPS	32	big, little
ARM	32	big, little
PowerPC	32	big
Intel 80386	32	little
MIPS64	64	big
Aarch64	64	little
x86-64	64	little

ムウェアをエミュレータで起動し，Web アプリケーションが動作する 80/tcp または 443/tcp に HTTP リクエストを送信して応答があった場合に起動成功とする．

起動に成功したファームウェアをベンダー毎にまとめた結果を**表 5** に示す．58 個のうち 52 個のファームウェアの起動に成功した．起動に失敗した 6 個のファームウェアのうち，1 個は Web アプリケーションが含まれていなかった．また，別の 1 個はファームウェアに含まれる実行ファイルの命令が QEMU に対応していなかったことで失敗した．残りの 4 個は Web アプリケーションの TLS 設定に用いる証明書や鍵に関連する問題でエラーが発生した．

次に，エミュレータで動作する CPU アーキテクチャの種類について述べる．OpenWrt の公式 Web サイトから入手した**表 6** に示す CPU アーキテクチャのファームウェア全てが，Web アプリケーションの起動に成功した．合計で 9 種類の CPU アーキテクチャが動作し，第 2.1 節で述べた既存エミュレータ [2], [4], [5] よりも 6 種類多く対応した．

4.1.2 エミュレータの実行速度

エミュレータの実行速度を評価するために，実機およびフルシステムエミュレーションを採用する FirmAE[5] と比較実験を行った．実機には第 4.2 節で利用したベンダー B のファームウェアが搭載されており，本提案および FirmAE では Web サイトから入手した実機と同じファームウェアを動作させる．

それぞれを利用してハニーポット生成を行った結果を**表 7** に示す．Web アプリケーションの起動時間は提案手法が最も速く，実機と約 2 倍の差が表れた．この差はハニーポット作成時間の短縮に繋がる．また，組み込み WebUI に HTTP リクエストを送信して，HTTP レスポンスが返されるまでの 1 通信にかかる時間は実機と提案手法で大きな差が見られなかった．一方，FirmAE は他の手法に比べて約 1.6 倍の通信時間がかかっている．さらに FirmAE はハニーポット生成の途中で通信不可能な状態に陥ることを確認した．以上より，本提案は実機と変わらない通信速度で，フルシステムエミュレーションよりも安定して動作するため，ハニーポット作成の利用に適している．

表 7 実機およびフルシステムエミュレーションとの速度比較

Table 7 Comparison of time taken by our framework with our emulator, a real device, and FirmAE.

	Web アプリ起動時間 [s]	1 通信の時間 [s]	ハニーポット生成時間 [s]
提案	44.4	0.058	1815.2
実機	85.2	0.055	1655.6
FirmAE	388.7	0.091	-

4.2 フレームワーク全体の性能評価

本フレームワークが，設計目標である偽装性と効率性を満たすことを確認するために評価実験を行った．**表 8** に示す 5 種類のファームウェアを独自にベンダーの Web サイトよりダウンロードし，フレームワークによってハニーポットを生成した．通信収集段階でのログイン操作に必要となるユーザ名とパスワードは，No.2，3 はデフォルトのまま，No.1，4 は同一のものを独自に設定した．No.5 はログイン操作が不要である．

4.2.1 ハニーポットの生成時間とインスタンスのサイズ

表 9 に，通信収集段階におけるクローリングで発見したパス数，ブラウザ操作ツールによって発生した通信数および応答数，そしてハニーポット生成に要した時間を示す．ハニーポット生成に要した時間は最短が約 6 分，最長が約 60 分であった．この時間に最も影響しているのは通信収集段階であるが，発生した通信数とハニーポット生成時間は比例していない．ファームウェアによってエミュレータ側の設定を変更していないため，通信収集時間に影響を与えるのは組み込み WebUI の実装方法であると考えられる．本手法では組み込み WebUI を並列化させて通信収集ができるため，実装方法を変更せずとも時間短縮が可能である．

また，フレームワークによって作成された各ハニーポットインスタンスのサイズを，表 9 に示す．ハニーポットインスタンスのサイズには学習モデルや応答用テーブルが含まれており，これらがサイズの大小に影響している．応答用テーブルは組み込み WebUI が送信する実際の HTTP レスポンスが含まれるため，部分的に削除してサイズを減らすことは難しい．一方，学習モデルのサイズは学習パラメータに依存しており，調整することでサイズの削減が可能であると考えられる．

表 8 ハニーポットの作成に利用した 5 つのファームウェア
Table 8 Five firmware samples for the evaluation.

No.	ベンダー	公開日	アーキテクチャ	Web サービス	ポート	OpenWrt バージョン	言語	ログイン機能
1	ベンダー B	2018.05	ARM little	uhttpd	80,443	Attitude Adjustment 12.09	英語	有り
2	ベンダー D	2017.07	MIPS big	uhttpd	80	Barrier Breaker 14.07	英語	有り
3	ベンダー E	2020.10	Aarch64 little	lighttpd	80	LEDE 17.01	英語	有り
4	ベンダー F	2019.01	MIPS little	lighttpd	80	OpenWrt 18.06.1	英語	有り
5	ベンダー G	2020.10	MIPS little	uhttpd	80	Chaos Calmer 15.05.1	日本語	無し

表 9 フレームワークによるハニーポット生成の結果
Table 9 Results of the evaluation with the five firmware samples.

No.	発見した パス数	発生した 通信数	収集した 応答数	ハニーポット生成時間 起動段階 [s]	通信収集段階 [s]	学習段階 [s]	合計時間 [s]	ハニーポットの サイズ [M]
1	498	20,019	6,956	45.2	1815.2	123.8	1984.2	51
2	352	13,571	4,892	38.8	222.3	116.4	377.5	72
3	181	5,446	1,812	8.7	3491.0	151.7	3651.4	17
4	58	1,678	624	14.6	1305.6	117.0	1437.1	37
5	440	18,479	5,688	19.0	239.7	137.9	396.6	34

4.2.2 ハニーポットの偽装性

ハニーポットの偽装性を調べるために，フレームワークによって生成されたハニーポットをインターネットに設置して攻撃を観察した．ハニーポットが受信する HTTP リクエストの中には WordPress などのコンテンツ管理システムを狙ったものや，Google などの検索エンジンによるものが含まれる．ハニーポットの偽装性を調べるためには，これらの HTTP リクエストと模倣対象の機器を狙った HTTP リクエストを判別する必要がある．本評価では各機器に存在する固有のパスに対してアクセスが来た場合に，当該機器を狙った攻撃であると判定する手法 [7] を採用する．

学内に設置した公開サーバおよびクラウドサービスのそれぞれに 5 つのハニーポットを設置し，2021 年 3 月 11 日から 6 月 1 日の 83 日間に渡って攻撃観測を行った．この期間に受信したリクエスト数を**表 10** に示す．ログイン機能を持つ No.1 から No.4 のハニーポットでは，いずれもログイン試行攻撃を観測した．受信したユーザ名およびパスワードの組み合わせはハニーポット間で異なっており，ログイン機能を持っていない No.5 のハニーポットでは観測されなかった．このことから，攻撃者は WebUI の機能や特徴を確認した上で攻撃していることが予想される．

また，各ハニーポットを 2 箇所に設置しているが，受信したリクエストにはハニーポット毎に特有の傾向があった．例えば，No.1 のハニーポットに対するログイン試行攻撃は，2 箇所ともログインを試すパスワードの順番や回数が一致していた．一方，No.1 以外のハニーポットではそのような HTTP リクエストを観測していない．すなわち，HTTP レスポンスに含まれる特定のキーワードに反応して挙動を変えている可能性が考えられる．また，ブラウザを利用してハニーポットにアクセスした場合は，JavaScript ファイルや CSS ファイルなどのファイルが連続して要求されるが，そのようなアクセスはほとんど観測されなかった．以上より，特定の IoT 機器を狙ったボットによるアクセスであり，HTTP レスポンスに含まれる特定のキーワードに反応して挙動を変えている可能性が考えられる．一方，No.5 のハニーポットはログインせずに設定変更ページへアクセスできるのにも関わらず固有パスへのアクセスが少ないため，HTTP レスポンスに含まれる言語も関係する可能性がある．

表 10 攻撃観測の結果
Table 10 Results of the attack observation.

No.	受信した リクエスト	固有パスへの リクエスト	ログイン試行 (成功)
1	26,268	312	45(12)
2	54,594	77	2(1)
3	52,588	63	7(3)
4	16,230	15	5(0)
5	25,590	49	0(0)

さらに分析した結果，攻撃者は “GET / HTTP/1.1” をはじめに送信し，ハニーポットから返される HTTP レスポンスの内容を確認してから次の行動を決定していることが分かった．このとき，HTTP レスポンスのステータスコードとして “200 OK” を返すことが一般的であるが，No.4 のハニーポットは “503 Service Unavailable” を返す．このハニーポットは他のハニーポットに比べて受信したリクエスト数および固有パスへのリクエスト数が少なく，攻撃者からサービスが利用できないと判断されて攻撃対象から除外された可能性が高い．その一方で，直接ログインページへアクセスする攻撃も観測しており，これは特定の機器を狙った攻撃であることを示唆する．

5. 考察と今後の課題

今回の実験では，OpenWrt をベースとした無線 LAN ルータに限定した場合に，異なるベンダーや CPU アーキテクチャであっても組み込み WebUI ハニーポットを自動生成できることを確認した．今後は他機器のファームウェアや一般的な Web アプリケーションにも適用可能である，普遍的なハニーポット生成フレームワークの実現を目指す．

また，本フレームワークによって生成されたハニーポットをインターネットに設置した結果，攻撃者によるログイン操作を観測し，ハニーポット間で異なる HTTP リクエストが送信されていることを確認した．これはログイン機能を持たない WebUI を対象とした既存の低対話型ハニーポットでは観測が困難である．しかし，設定変更やマルウェアの挿入などの高度な攻撃は観測されなかった．その理由として，挙動再現のための通信収集や対話モデルの学習が不十分であった可能性が考えられる．今後は模倣対象となる WebUI の再現度を高めるとともに，ハニーポットが攻撃を受けたときの挙動再現を課題とする．

6. 研究倫理

今回実験や評価に利用したファームウェアは，各ベンダーの Web サイトで提供されているものである．また，ハニーポット生成においてファームウェアそのものをインターネットに公開することはなく，外部との通信は行わない．さらに，通信収集段階によって得られる応答の中に機密情報が含まれていた場合は，別の値に変換することで安全性を保証する．今後ハニーポット作成時や攻撃観測時に脆弱性を発見したときは，対象となるベンダーにすみやかに報告し，対処を依頼する．

7. まとめ

本研究では，組み込み WebUI を狙った攻撃を効率よく収集することを目指し，IoT 機器のファームウェアを利用してハニーポットを生成するフレームワークを提案した．多種多様な組み込み WebUI の挙動を再現するためのブラウザ操作ツールを作成し，機械学習を利用することによって自動化を達成した．本提案で用意したエミュレータは 9 種類の CPU アーキテクチャおよび 7 ベンダーのファームウェアを動作させることができた．さらに，5 ベンダーの組み込み WebUI の挙動を模したハニーポットを作成して実際に攻撃観測を行った．

今後はハニーポットの偽装性を高めるために，通信収集範囲の拡大や学習モデルのパラメータ調整を行う．さらに，対象機器を無線 LAN ルータ以外の IoT 機器に広げることを検討する．

謝辞 本研究の一部は，JSPS 科研費基盤研究 (C)19K11962 による助成を受けたものです．

参考文献

[1] Xie, W., Jiang, Y., Tang, Y., Ding, N. and Gao, Y.: Vulnerability Detection in IoT Firmware: A Survey, *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, pp. 769–772 (2017).
[2] Vetterl, A. and Clayton, R.: Honware: A Virtual Honeypot Framework for Capturing CPE and IoT Zero Days, *APWG Symposium on Electronic Crime Research (eCrime)*, IEEE, pp. 1–13 (2019).
[3] Musch, M., Härterich, M. and Johns, M.: Towards an Automatic Generation of Low-Interaction Web Application Honeypots, *Proceedings of the International Conference on Availability, Reliability and Security*, pp. 1–6 (2018).
[4] Chen, D. D., Woo, M., Brumley, D. and Egele, M.: Towards Automated Dynamic Analysis for Linux-based Embedded Firmware., *NDSS*, Vol. 1, pp. 1–1 (2016).
[5] Kim, M., Kim, D., Kim, E., Kim, S., Jang, Y. and Kim, Y.: FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis, *Annual Computer Security Applications Conference*, pp. 733–745 (2020).
[6] Luo, T., Xu, Z., Jin, X., Jia, Y. and Ouyang, X.: IotCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices, *Black Hat*, pp. 1–11 (2017).
[7] 江澤優太，田宮和樹， 中山颯， 鉄穎，吉岡克成，松本勉：実機を用いたハニーポットによる組込み機器の WebUI に対するサイバー攻撃の分析，コンピュータセキュリティシンポジウム論文集，Vol. 2017, No. 2 (2017).
[8] : OpenWrt, `https://openwrt.org/`.
[9] : QEMU, `https://www.qemu.org/`.
[10] 山本萌花，掛井将平，齋藤彰一：ファームウェアの挙動を事前収集する IoT ハニーポットの提案および基礎調査，コンピュータセキュリティシンポジウム論文集，pp. 346–353 (2020).
[11] Zheng, Y., Davanian, A., Yin, H., Song, C., Zhu, H. and Sun, L.: FIRM-AFL: High-Throughput Greybox Fuzzing of IoT Firmware via Augmented Process Emulation, *USENIX Security Symposium*, pp. 1099–1114 (2019).
[12] Roy, S., Sharmin, N., Acosta, J. C., Kiekintveld, C. and Laszka, A.: Survey and Taxonomy of Adversarial Reconnaissance Techniques, *arXiv:2105.04749* (2021).
[13] Zhou, Y.: Chameleon: Towards Adaptive Honeypot for Internet of Things, *Proceedings of the ACM Turing Celebration Conference - China*, pp. 1–5 (2019).
[14] 毅 八木，直人谷本，剛男針生，光恭伊藤：高対話型 Web ハニーポットにおける攻撃情報収集方式の改善，コンピュータセキュリティシンポジウム論文集，Vol. 2009, pp. 1–6 (2011).
[15] Pauna, A., Iacob, A.-C. and Bica, I.: Qrassh-a Self-Adaptive SSH Honeypot Driven by Q-Learning, *International Conference on Communications (COMM)*, IEEE, pp. 441–446 (2018).
[16] : Docker, `https://www.docker.com/`.
[17] : Kubernetes, `https://kubernetes.io/`.
[18] : Selenium Wire, `https://github.com/wkeeling/selenium-wire`.
[19] Luong, M., Pham, H. and Manning, C. D.: Effective Approaches to Attention-based Neural Machine Translation, *CoRR*, Vol. abs/1508.04025 (2015).
[20] : Minikube, `https://minikube.sigs.k8s.io/docs/`.
[21] : TensorFlow, `https://www.tensorflow.org/`.
[22] : FirmAE Dataset, `https://github.com/pr0v3rbs/FirmAE`.