

形式整合性と脆弱性誘発要因を考慮した LLMベースのIoTファジング検査値作成手法

中西 響^{1,a)} 長谷川 健人^{2,b)} 披田野 清良^{2,c)} 福島 和英^{2,d)} 橋本 和夫^{1,e)} 戸川 望^{1,f)}

概要：近年、IoT デバイスの普及が進む反面、その脆弱性を狙った悪意のある攻撃が深刻な問題となっている。ファジングは未知の脆弱性を発見する手法として有効であり、AFL や Snipuzz などの手法が存在する。しかし、多種多様な IoT デバイスの中にはセキュリティの観点でレスポンスにおけるエラーメッセージの詳細を隠蔽したり、ソースコードが非公開であったりするために、既存手法が適用できない場合が少なくない。本稿では、大規模言語モデル（LLM）を活用し、上記のようなデバイスにおいても有効な検査を行うためのファジングの検査値作成手法を提案する。提案手法は、IoT デバイスの通信ログから取得したデータを利用し、入力データ形式の整合性を保持する。また、LLM の知見により、検査対象デバイスにおける特定の脆弱性の発生条件を推論させ、その結果をもとに脆弱性を効率的に誘発する検査値の作成を実現する。評価実験の結果、提案手法は、従来手法や比較手法では検出に失敗した脆弱性の検出に成功した。

キーワード：大規模言語モデル、ファジング、脆弱性検査、IoT デバイス、セキュリティ、CoT（Chain of Thought）

LLM-based Test Input Creation Method for Effective Fuzzing

HIBIKI NAKANISHI^{1,a)} KENTO HASEGAWA^{2,b)} SEIRA HIDANO^{2,c)} KAZUhide FUKUSHIMA^{2,d)}
KAZUO HASHIMOTO^{1,e)} NOZOMU TOGAWA^{1,f)}

Abstract: Fuzzing is an effective testing method for IoT devices, where attacks are frequently observed. In this paper, we propose a method for effectively creating test values even for devices where it was difficult to apply conventional methods using LLM.

1. はじめに

近年、IoT（Internet of Things）デバイスは社会インフラの一部として定着し、家庭用のスマートホーム機器から医療機器、交通制御システム、産業用ロボットに至るまで、幅広い分野で導入が進んでいる。全世界のIoTデバイスは2024年時点で177億台を超え、その数は今後10年以内に倍増すると予測されている[1]。これらのIoTデバイスは

常にネットワークに接続され、センシングや制御、データ処理といった機能により利便性を提供している一方で、IoTデバイスを狙ったサイバー攻撃も増大している[2]。IoTデバイスのセキュリティ侵害に起因する1件あたりの平均損害額は33万ドルを超えるとされ、IoTデバイスのセキュリティ対策は喫緊の課題となっている[3]。しかし、IoTデバイスは処理能力や電力供給に制約があることが多く、自動更新機能を持たない製品も少なくない。また、開発コストの観点から、サードパーティのコンポーネントが組み込まれることが多いため、製品の内部構造を完全に把握するのは困難である[4]。上記の背景から、内部構造に依存せずに製品の出荷前に未知の脆弱性を発見する検査として、ファジングが有効なアプローチとされている。

ファジングは、検査対象のデバイスやプログラムに対して大量の検査値を自動作成・送信し、その実行時の挙動を

¹ 早稲田大学基幹理工学研究所 〒169-8555 東京都新宿区大久保3-4-1
² 株式会社 KDDI 総合研究所, 埼玉県ふじみ野市大原 2-1-15
^{a)} hibiki.nakanishi@togawa.cs.waseda.ac.jp
^{b)} kt-hasegawa@kddi.com
^{c)} se-hidano@kddi.com
^{d)} ka-fukushima@kddi.com
^{e)} kazuo.hashimoto@togawa.cs.waseda.ac.jp
^{f)} ntogawa@waseda.jp

監視することで、クラッシュなどセキュリティ上の欠陥を検出する検査手法である。先述した IoT デバイスのブラックボックス性やソースコードの入手困難性から、既存のファジングツール（ファザー）の多くは初期値（初期シード）をもとに、ミューテーションベースでこれに変異を加えることで大量の検査値を作成している [5, 6]。

IoT デバイスをブラックボックスに検査するファザーとしてこれまでいくつか提案されているが、中でも近年提案された Snipuzz [7] は、データ形式を保持したミューテーションが可能で、そのためのミューテーション単位の手動分割を必要としない点において、これまで提案されたファザーに比較して優位性を持つ。Snipuzz は、デバイスから返されるレスポンスメッセージをもとに初期シードを複数のスニペットに分割することで、データ形式に違反せず効果的なミューテーションを実現する。しかし、IoT デバイスの中には、セキュリティ保護の観点からレスポンス中にエラーメッセージの詳細を明示しない場合も多く、このような場合には適切なスニペット分割が困難になる。上記の結果、Snipuzz の利点であるデータ形式に違反しないミューテーションが機能せず、有効性は限定的となる。さらに、Snipuzz を含む従来のミューテーションベースのファザーの多くは、検査値作成時に加えるビット反転やバイト削除などの変異操作（ミューテーションオペレータ）を事前に定義しており、検査状況に応じて動的に変更されることはない [7, 8]。このようなアプローチでは、検査値の作成が初期シードやミューテーションオペレータに依存するため、探索範囲の偏りが生じやすく、脆弱性の検出効率が限定的となる場合がある。

ここで大規模言語モデル (LLM: Large Language Model) に注目する [9]。LLM は大量のテキストデータを学習済みであり、自然言語だけでなくソースコードや構造化データの文法や規則を推論する能力を有している。LLM の特性を活かすことで、エラーメッセージが観測困難なデバイスに対しても、構文的に適切なミューテーションを実現できる可能性がある。また、LLM の知見の活用により、検査対象の脆弱性を効果的に誘発、顕在化させる検査値を作成し、脆弱性検査効率の向上が期待される。

本稿では、IoT デバイスのブラックボックスファジングの効率向上のため、LLM ベースの検査値作成手法を提案する。提案手法は、IoT デバイスの通信ログを使用してデータ形式仕様に準拠し、LLM の知見をもとに脆弱性を誘発するデータの特徴（脆弱性誘発要因）を考慮して検査値を作成する。

本稿の貢献を以下に示す。

- (1) LLM を活用し、IoT ファジングにおいて、各デバイスの仕様に合わせた検査値を自動で作成する手法を提案する。
- (2) 提案手法は、通信ログを活用することによりデータ形式の整合性の向上や探索範囲の拡大を図る。また、

LLM の知見をもとに脆弱性を誘発する条件を推論や、Chain-of-Thought (CoT) [10] に基づく二段階推論を取り入れることで、脆弱性検査に有効な値の作成を促す。

- (3) 評価実験の結果、提案手法は 2 種類の IoT デバイスに対して、比較手法では検出できなかった脆弱性の検出に成功した。また、通信ログや CoT を除いたベースライン手法との比較から、これらの要素の重要性および必要性を確認した。

2. 先行研究

本章では、2.1 節で既存のファザー、2.2 節で LLM を活用した既存のファザーを説明する。

2.1 既存のファザー

従来の IoT デバイスのファザーには、Peach Fuzzer [11] や AFL (American Fuzzy Lop) [12], Boofuzz [8], Snipuzz [7] などがある。

Peach Fuzzer は、XML (Extensible Markup Language) 形式の設定ファイルにデータ構造やミューテーションでの変更箇所を記述し、それに基づいてファジングを行うファザーである。ファジング対象が幅広く、仕様が明確な場合は有効だが、製品の内部構造が不明な場合は設定ファイルの作成が難しく、変更箇所も手動で設定する必要がある。

AFL は、ソースコードをもとに検査値の実行経路を計測して検査を行うグレーボックスファジングである。コードのカバレッジを元に探索可能なファザーであるが、多くのブラックボックスデバイスのように、ソースコードや実行時のカバレッジ情報を取得できない場合は適用が難しい。

Boofuzz は、ネットワークを監視しながらファジングを行うネットワークプロトコルファザーである。Boofuzz も Peach Fuzzer と同様の設定を Python で記述する必要がある。手動作業の必要性やファジング対象の仕様の入手困難性の観点で課題がある。

Snipuzz は、デバイスからのレスポンスメッセージに基づいて初期シードをスニペットに分割し、スニペット単位でミューテーションを行うファザーである。スニペットに基づくことで、データ形式を考慮したミューテーションが可能となるが、1 章で述べたようにレスポンスメッセージの詳細を隠蔽するデバイスにおいては有効性が発揮されない。また、レスポンスメッセージが隠蔽されないデバイスであっても、ミューテーションオペレータが事前定義されているために、脆弱性検出の効率が初期シードや定義されたオペレータに依存するという課題がある。

2.2 LLM を活用した既存ファザー

LLM を活用した既存のファザーには、TitanFuzz [13], Fuzz4all [14], LLMIF [15], LLMUZZ [16], ChatHTTP-Fuzz [17] などがある。

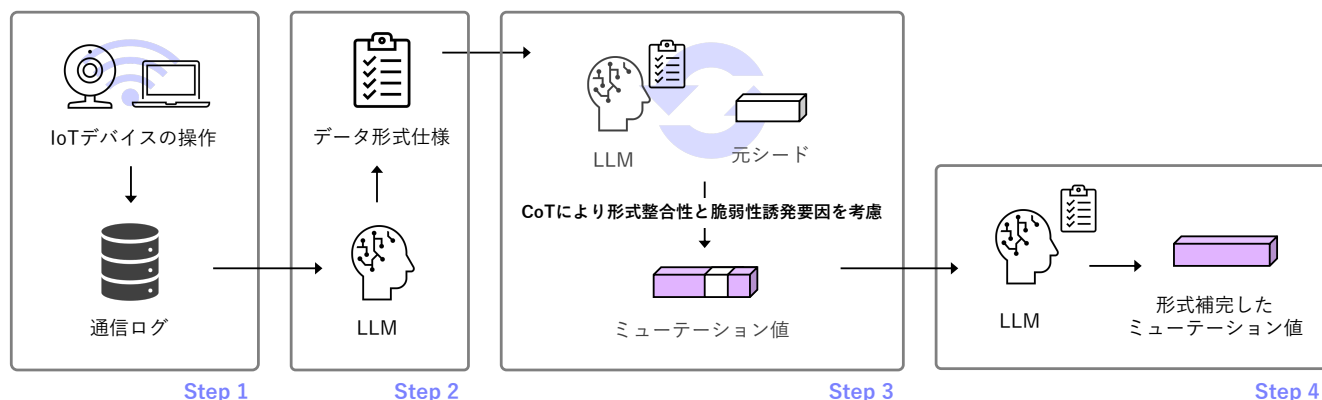


図 1: 提案手法の流れ.

TitanFuzz は、LLM を活用して深層学習ライブラリをファuzzingする手法である。TitanFuzz は、Codex [18] などの Generative LLM を使用し、ライブラリのインポートやターゲットとなる API シグネチャをプロンプトで指定して、初期シードとなるプログラムを作成する。また、シードの一部をマスクし、その領域を InCoder [19] などの LLM により置き換える。しかし、TitanFuzz は LLM によりプログラムを生成する手法であり、ソースコードが非公開であることが多い IoT デバイスへの適用は限定的である。

Fuzz4all は、LLM を使った多言語対応のファザーであり、主に自動プロンプト生成とファuzzingグループで構成される。自動プロンプト生成では、LLM はユーザから与えられたドキュメント、コード例、仕様などを要約し、ファuzzingで生成させる値を指示するプロンプトを作成する。続くファuzzingグループでは、作成されたプロンプトをもとに、別の LLM に検査値を生成させる。しかし、ファuzzingグループにおいて適用されるミューテーションオペレータは 3 種類であり、いずれも仕様等をベースにした値の作成や、作成した値の微小な変更、同一の意味での書き換えを LLM に指示するものであり、特定の脆弱性の誘発を目的とした値の生成は行わない。

LLMIF は、Zigbee [20] や Z-WAVE [21] などの IoT プロトコルのファuzzingを、プロトコル仕様書と LLM の組み合わせにより効率化する手法である。しかし、ファuzzingの対象が OSI 参照モデル [22] のアプリケーション層のペイロード等であった場合、その詳細な仕様書はプロトコル仕様書と異なり入手が難しい。上記の場合、LLM が仕様書から背景知識を取得することができず、LLMIF の効果を得られない。

LLMUZZ は、IoT デバイスのための LLM ベースのファuzzing手法であり、初期シード生成や隠しキーワード分析、ミューテーションに LLM を用いる。LLMUZZ におけるミューテーションでは、LLM は変異すべきフィールドをブレースホルダで置換するだけであり、実際の値の挿入は外部エンジンが行う。これは、1 章で述べたミューテーションオペレータの事前定義による課題を解決していない。ま

た、初期シード生成や隠しキーワード分析でファームウェアを利用するが、ファームウェアは暗号化されていることが多く、一般的に入手が困難である。したがって、外部で製造されたデバイスを検査する場合、LLMUZZ の適用は難しい。

ChatHTTPFuzz は、LLM により HTTP (Hyper Text Transfer Protocol) 構造とコードを解析する IoT 向けのファザーである。LLM は、HTTP リクエストの可変フィールドの抽出や、バックエンドのコード等の解析から新規パケットの作成を行う。ソースコードが非公開である IoT デバイスへ適用する場合には、コードの分岐条件等を解析できないため、十分な効果が得られない。

上述のように、LLM を活用した既存ファザーを内部構造が不明な IoT デバイスに適用する際の課題は、ソースコードや、アプリケーションレベルの詳細な仕様書の入手困難性、さらにミューテーションにおける LLM の効果的な活用にある。特に、詳細な仕様書が得られない場合、入力形式をデバイスの期待するものと一致させるのが困難となり、ファuzzing中に作成される多くの検査値が無効となる。提案手法は、LLM の活用により通信ログからデータ形式仕様書を作成してそれを満たす検査値を作成したり、特定の脆弱性を誘発する検査値を作成したりすることで、ファuzzing効率の向上を図る。

3. 提案手法

本章では提案手法を説明する。提案手法の流れを図 1 に示す。

提案手法は、以下の手順で構成される。

Step 1: 検査対象 IoT デバイスの通信ログを取得する。

Step 2: 通信ログをもとに、LLM によりデータ形式仕様書を作成する。

Step 3: LLM によりミューテーション値を作成する。

Step 4: データ形式の欠陥があれば、LLM により補完する。

Step 4 の後、ミューテーション値を IoT デバイスに送信し、Step 3 に戻って処理を繰り返す。

以降、手法の各ステップを説明する。

3.1 Step 1：通信ログ取得

検査対象デバイスの仕様を LLM に認識させるために用いる通信ログを取得する。通信ログは、Wireshark [23] や Burp Suite [24] などのツールを用いて取得できる。なお、LLM により対象デバイスを網羅的に検査する値を作成可能にするため、通信ログはデバイスの操作の全種類を少なくとも 1 つずつ含むものを用いる。

3.2 Step 2：データ形式仕様の生成

Step 2 では、Step 1 で取得した通信ログをもとに、検査対象デバイスのデータ形式の仕様を LLM により生成する。LLM の多様なテキストの文法や規則を推理を推論する能力により、詳細な仕様書が存在しない場合でもデータ形式仕様を生成でき、デバイス仕様に準拠した検査値の生成が可能となる。

Step 2 で用いるプロンプトテンプレートを以下に示す。

プロンプトテンプレート 1

Please answer the user's question with reference to the following communication logs for an IoT device.

Communication logs:
{Step 1 の通信ログ}

Question:
In generating data to be sent to a device, please provide a brief description of any special specification regarding the data format.

Answer:

データ形式仕様をテキストで生成することで、以降のミューテーションにおいて LLM が出力する検査値の形式整合性の確保や、検査範囲の拡大が図られる。

3.3 Step 3：ミューテーション値作成

Step 3 では、Step 2 で生成されたデータ形式仕様を用いて、LLM による脆弱性誘発要因を考慮したミューテーションを行う。プロンプトには以下の情報を含める。

- (1) データ形式仕様 (Step 2 の成果物)
- (2) 検査対象脆弱性の名称
- (3) ミューテーションの元となるシード

脆弱性については、名称を与えるだけで、LLM が持つ事前知識に基づきその脆弱性を効率的に検査するための値が自動で作成される。

また、CoT [10] による二段階の推論を行う。まずデータ形式仕様をもとに、検査対象脆弱性がどのような場合に発生しうるか推論させる。続いて、その推論結果をもとに元シードをミューテーションする。二段階推論によるミューテーション値作成を行うことで、LLM の思考過程を誘導し、出力値による脆弱性検出率を向上させる。

プロンプトテンプレートを以下に示す。

プロンプトテンプレート 2

The following is the communication command specification for fuzzing the target device.

Specification: {Step 2 のデータ形式仕様}

Based on the above, comprehensively explain the characteristics of the data that may cause {検査する脆弱性の名称}.

Answer:

プロンプトテンプレート 3

Based on the above, please mutate {元シード} to cause {検査する脆弱性の名称}.

Rules:

- Please output each data in the following format:

BEGIN_PAYLOAD
<Mutation data here>
END_PAYLOAD

- **Please do not use any explanatory text, numbers, or markdown notation (such as “”).**
- Only the payload should be output.

Output:

3.4 Step 4：データ形式補完

Step 4 では、Step 3 で得たミューテーション値のデータ形式を検証し、不完全な場合に補完を行う。特に、LLM の回答生成過程で最大生成長 (max_new_tokens) を超過すると、出力が途中で切れ、フォーマットが欠落することがある。例えば、JSON (JavaScript Object Notation) 形式のデータの末尾が括弧 (}) で閉じられていない場合などが該当する。

提案手法では、プロンプトテンプレート 3 に示すように、ミューテーション値の出力フォーマットを明示的に指示している。本実験では、出力されるミューテーション値を一つずつ ### BEGIN_PAYLOAD と ### END_PAYLOAD で囲む形式を指定している。ミューテーション値の後ろに ### END_PAYLOAD が欠落している場合は、途中で出力が切れた不完全なデータ形式と判定し、以下のプロンプ

表 1: 実験環境.

リソース名	型番
OS	Rocky Linux 8.7
CPU	Intel Xeon Gold 5218*4
メモリ	251GB

トによりデータ形式を補完する.

プロンプトテンプレート 4

The following are the communication command specifications for fuzzing the target device.

Specification:

{Step 2 のデータ形式仕様 }

The following string is incomplete data that was terminated midway through generation. Please ****ONLY ADD**** any missing characters. ****DON'T CHANGE EXISTING TEXT EVEN IF THE VALUE IS INVALID.****

Incompleted data:

{ データ形式に欠陥のあるミューテーション値 }

Please complete the above data.

Output format:

Completed: <completed data>

4. 評価実験

本章では, 提案手法の有効性を検証する. 4.1 節で実験環境, 4.2 節で検査対象の IoT デバイス, 4.3 節で提案手法によるファuzzingの結果, 4.4 節で比較実験を説明し, 4.5 節で実験結果を考察する.

4.1 実験環境

提案手法を Python 3.8.10 を用いて実装し, 表 1 の実行環境で実行した. LLM は OpenAI の API で提供される GPT-4o モデルを使用し, API 呼び出しには openai ライブラリ (v0.28.0) を用いた. また, 検査対象 IoT デバイス (4.2 節を参照) へのデータ送信には, HTTP ライブラリである Requests の post メソッドを使用した.

4.2 検査対象の IoT デバイス

本実験では, 検査対象 IoT デバイスとして, [25], [26] や [27], [28] など, 市販のデバイスを参考にしたスマート電球と IP カメラをそれぞれ Raspberry Pi 4 上に実装した. 各デバイスは, Flask を通じて HTTP 通信が可能な API を持つ.

スマート電球は, Command キーと Parameter キーから構成される JSON 形式のデータを受信する. IP カメラは, Request タグ内にネストされた Command タグと Parameter タグを持つ XML 形式のデータを受信する.

```
{ "Command": "turnOn", "Parameter": "" }
{ "Command": "turnOff", "Parameter": "" }
{ "Command": "toggle", "Parameter": "" }
{ "Command": "setBrightness", "Parameter": "98" }
{ "Command": "setColor", "Parameter": "25:25:25" }
{ "Command": "setColorTemperature", "Parameter": "6000" }
```

図 2: スマート電球の通信ログ.

提案手法の有効性を検証するため, スマート電球には buffer overflow と null pointer exception の脆弱性をそれぞれ 4 つずつ, IP カメラには 6 つずつ実装した. 簡単のため, スマート電球では, Command や Parameter キーに対応する値の長さが 256 を超えた場合に buffer overflow が発生し, 値が null である場合に null pointer exception が発生する. また, IP カメラでは, Command や Parameter タグ内の値の長さが 256 を超えた場合に buffer overflow が発生し, これらのタグが存在しない場合に null pointer exception が発生する.

4.3 IoT デバイスでの検証

本節では, 4.2 節で説明した 2 種類の IoT デバイスに対して, 提案手法によるファuzzingを適用した結果を説明する.

4.3.1 スマート電球での検証

スマート電球では, Step 1 において図 2 に示す通信ログが取得された.

図 2 の通信ログをもとに, Step 2 で生成されたデータ形式仕様の一部を以下に示す.

スマート電球のデータ形式仕様

- **Command Structure**:** Each command is represented as a JSON object with two fields: 'Command' and 'Parameter'.
- **Command Field**:** This field specifies the action to be performed by the device. It is a string that must match one of the predefined commands exactly. The available commands are:
'turnOn', 'turnOff', 'toggle', 'setBrightness', 'setColor', 'setColorTemperature.'
- **Parameter Field**:** This field provides additional information required by certain commands. It is also a string, but its content varies depending on the command:
- For 'turnOn', 'turnOff', and 'toggle', the 'Parameter' field is empty (""), as these commands do not require additional parameters.
- For 'setBrightness', the 'Parameter' should be a string representing a numerical value ...

以下に, 検査対象の脆弱性を buffer overflow とした場合

```

<Request><Command>brightness</Command><Parameter>50</Parameter></Request>
<Request><Command>contrast</Command><Parameter>50</Parameter></Request>
<Request><Command>saturat</Command><Parameter>80</Parameter></Request>
<Request><Command>mirror</Command><Parameter>0</Parameter></Request>
<Request><Command>light frequency</Command><Parameter>2</Parameter></Request>

```

図 3: IP カメラの通信ログ。

に 3.3 節のプロンプトテンプレート 2, 3 に対して得られた LLM による回答の一部を示す。なお、初期シードは図 2 に示す通信ログから得たデータを用いた。

スマート電球：プロンプト 2 への LLM の回答

1. ****Exceeding Command Lengths****:
...
- For instance, sending a string like “setBright-
nessssssssssssssssssssssss” might cause the device to allocate
insufficient space for handling what is presumed to be “set-
Brightness”.
2. ****Parameter Overflow****:
...
- ‘setColor’ requires a format of “R:G:B”. Providing overly
lengthy components or an extraordinarily long string like
“255255:255255:255255” could cause a buffer overflow if the
device is not designed to handle lengths beyond expected
values.

スマート電球：プロンプト 3 への LLM の回答

```

### BEGIN_PAYLOAD
{“Command”: “setColor”, “Parameter”: ”255:255:255:
255:255:255:255:255:255:255:255:255:255...”

```

上記の出力結果は、値の後ろに “### END_PAYLOAD” がいないため、Step 4 にてデータ形式の補完が実行された。

検査対象脆弱性を null pointer exception とした場合も同様にして、上記のような提案手法によるミューテーションを繰り返した結果、スマート電球に実装された計 8 個の脆弱性をすべて検出した。脆弱性を検出したミューテーション値の一部を、表 2 の 2-5 行目に示す。また、検査の開始から各脆弱性を検出するまでに要した時間を表 3 の 3 行目に示す。なお、表 3 に示す時間はすべて、LLM の推論の時間を含んでいる。

4.3.2 IP カメラでの検証

IP カメラでは、Step 1 において図 3 に示す通信ログが取得された。

また、Step 2 で生成された IP カメラのデータ形式仕様を以下に示す。

IP カメラのデータ形式仕様

The data format for communication logs to IoT devices is structured in XML-like syntax, where each request is encapsulated within ‘<Request>’ tags.

Inside each ‘Request’, there are two main elements: ‘<Command>’ and ‘<Parameter>’.

The ‘<Command>’ element specifies the action or setting being adjusted, such as “brightness”, “contrast”, “saturation”, “mirror”, or “light frequency”.

The ‘<Parameter>’ element provides the specific value for the command, such as “50” for brightness and contrast, “80” for saturation, “0” for mirror, and “2” for light frequency.

Each parameter value is likely numeric and corresponds to the expected input range for the respective command on the device.

以下に、検査対象の脆弱性を null pointer exception とした場合に 3.3 節のプロンプトテンプレート 2, 3 に対して得られた LLM による回答の一部を示す。初期シードとして、図 3 に示すデータを用いた。

IP カメラ：プロンプト 2 への LLM の回答

1. ****Missing ‘<Command>’ or ‘<Parameter>’ Elements****:
- If a ‘<Request>’ element is missing either the ‘<Command>’ or ‘<Parameter>’ element, this absence could lead to a null pointer exception if the processing code assumes these elements are always present. ...
2. ****Empty or Null Content****:
- Even if both elements are present, they might contain empty or null values. For example, ‘<Command></Command>’ or ‘<Parameter></Parameter>’ would provide no valid data for processing. ...

IP カメラ：プロンプト 3 への LLM の回答

```

### BEGIN_PAYLOAD
<Request><Parameter>50</Parameter></Request>
### END_PAYLOAD
...

```

buffer overflow の検査も同様にして、提案手法によるミューテーションを繰り返した結果、IP カメラに実装された計 12 個の脆弱性をすべて検出した。脆弱性を検出したミューテーション値の一部を、表 2 の 6-9 行目に示す。検査の開始から、各脆弱性を検出するまでに要した時間を表 3 の 6 行目に示す。

4.4 比較実験

4.4.1 Snipuzz との比較

本項では、IoT デバイスのファザーのうち有力なものの

表 4: Snipuzz で適用されるミューテーションの種類と操作内容.

ミューテーションの種類	操作内容
<i>BitFlip</i>	スニペットのビットを反転する
<i>Empty</i>	スニペットを空にする
<i>Repeat</i>	スニペットを 2 から 10 のうちランダムな回数繰り返す
<i>Dictionary</i>	スニペットを操作上問題が発生しそうな単語 (on, null, -1 など) に置換する

り得た.

参考文献

- [1] statista, “Number of internet of things (iot) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033.” <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>.
- [2] I. Ahmad, M. S. Niazy, R. A. Ziar, and S. Khan, “Survey on iot: security threats and applications,” *Journal of Robotics and Control (JRC)*, vol. 2, no. 1, pp. 42–46, 2021.
- [3] psacertified, “Why certify iot security?.” <https://www.psacertified.org/what-is-psa-certified/why-choose-psa-certified/>.
- [4] B. Zhao, S. Ji, J. Xu, Y. Tian, Q. Wei, Q. Wang, C. Lyu, X. Zhang, C. Lin, J. Wu, *et al.*, “A large-scale empirical analysis of the vulnerabilities introduced by third-party components in iot firmware,” in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 442–454, 2022.
- [5] C. Zhang, Y. Wang, and L. Wang, “Firmware fuzzing: The state of the art,” in *Proceedings of the 12th Asia-Pacific Symposium on Internetworking*, pp. 110–115, 2020.
- [6] J. Li, B. Zhao, and C. Zhang, “Fuzzing: a survey,” *Cybersecurity*, vol. 1, pp. 1–13, 2018.
- [7] X. Feng, R. Sun, X. Zhu, M. Xue, S. Wen, D. Liu, S. Nepal, and Y. Xiang, “Snipuzz: Black-box fuzzing of iot firmware via message snippet inference,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 337–350, 2021.
- [8] “boofuzz: Network protocol fuzzing for humans.” <https://boofuzz.readthedocs.io/en/stable/>, 2017.
- [9] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han, “Large language models can self-improve,” *arXiv preprint arXiv:2210.11610*, 2022.
- [10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.
- [11] “Peach fuzzer.” <https://peachtech.gitlab.io/peach-fuzzer-community/>.
- [12] “american fuzzy lop.” <https://lcamtuf.coredump.cx/afl/>.
- [13] Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, “Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models,” in *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*, pp. 423–435, 2023.
- [14] C. S. Xia, M. Paltenghi, J. Le Tian, M. Pradel, and L. Zhang, “Fuzz4all: Universal fuzzing with large language models,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.
- [15] J. Wang, L. Yu, and X. Luo, “Llmif: Augmented large language model for fuzzing iot devices,” in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 881–896, IEEE, 2024.
- [16] G. Gao, S. Gan, X. Wang, and S. Zhu, “Llmuzz: Llm-based seed optimization for black-box device fuzzing,” in *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1209–1216, IEEE, 2024.
- [17] Z. Yang, H. Peng, Y. Jiang, X. Li, H. Du, S. Wang, and J. Liu, “Chathttpfuzz: large language model-assisted iot http fuzzing,” *International Journal of Machine Learning and Cybernetics*, pp. 1–22, 2025.
- [18] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, “Evaluating large language models trained on code,” *arXiv preprint arXiv:2107.03374*, 2021.
- [19] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W.-t. Yih, L. Zettlemoyer, and M. Lewis, “InCoder: A generative model for code infilling and synthesis,” *arXiv preprint arXiv:2204.05999*, 2022.
- [20] S. Farahani, *ZigBee wireless networks and transceivers*. newnes, 2011.
- [21] M. B. Yassein, W. Mardini, and A. Khalil, “Smart homes automation using z-wave protocol,” in *2016 International Conference on Engineering & MIS (ICEMIS)*, pp. 1–6, IEEE, 2016.
- [22] J. D. Day and H. Zimmermann, “The osi reference model,” *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.
- [23] “About wireshark.” <https://www.wireshark.org/about.html>, 2020.
- [24] A. Kore, T. Hinduja, A. Sawant, S. Indorkar, S. Wagh, and S. Rankhambe, “Burp suite extension for script based attacks for web applications,” in *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, pp. 651–657, IEEE, 2022.
- [25] “Philips smart led lights with wiz connected.” <https://www.usa.lighting.philips.com/consumer/led-lights/smart-wifi-led>.
- [26] “Switchbot color bulb.” <https://www.switch-bot.com/products/switchbot-color-bulb>.
- [27] “Indoor/outdoor 5mp h.265 wdr poe ir fixed turret network camera.” <https://www.trendnet.com/products/5mp-turret-camera/indoor-outdoor-5mp-h265-wdr-poe-ir-fixed-turret-network-camera-TV-IP1515PI>.
- [28] “Nest cam (outdoor or indoor, battery).” https://store.google.com/gb/product/nest_cam_battery?hl=en-GB.