公衆無線 LAN を用いた HTTPS 通信に対する MITM 攻撃の新手法の開発とその実装

著者
神戸大学工学部電気電子工学科
学籍番号 1914320T
木村圭一朗

指導教員
森井 昌克 教授
白石 善明 准教授

2022 年 4 月 3 日

概要

昨今のネットワークの著しい普及により，屋外で利用できるネットワークの需要が拡大している． その需要に応える通信インフラとして公衆無線 LAN があり，公共のあらゆる場でその整備が進んでいる． しかし，セキュリティ面での懸念は以前として残っている． 利用した公衆無線 LAN が通信の傍受を目的とした悪性のものである可能性は，その懸念の一例である．屋外で手軽にインターネットを利用できる反面，ユーザーたちはその公衆無線 LAN の提供元を確認することはほとんどないだろう． 近年は通信内容が暗号化された HTTPS 通信が主流になり，中間者攻撃も簡単に実行できないが，公衆無線 LAN を利用した通信の場合はそれが可能になる場合がある． 本稿では，主にその手法と具体的な検証結果について記す．

目次

1 序論

2 前提知識・用語

本稿を進めるにあたり，必要であると思われる前提知識及び用語を以下に記す.

2.1 通信関連について

HTTP

Hypertext Transfer Protocol の略であり，主にブラウザとサーバ間で Web 情報のやりとりを目的としたアプリケーション層プロトコルを指す.

HTTPS

HTTP over SSL の略であり，TLS/SSL という仕組みを用いて HTTP 通信を暗号化したものを指す． 暗号化の主な目的としては，アカウントのログイン情報やクレジットカード番号を筆頭とする機密性・重要性の高い情報を他人に読み取られないようにする事である.

中間者攻撃（man-in-the-middle attack）

通信を行う二者の通信に対して，第三者がプロキシサーバーの様に中継を行う事で，通信内容の傍受・改竄を行う攻撃手法を指す． 国内外を通じてインターネットの普及に伴う公衆無線 LAN の設置が進んでいるが，これらの中に悪意を持ったアクセスポイントがあれば，そこを起点として通信内容が盗み見られる可能性がある． 特に，アカウントへのログイン情報やクレジットカードに加え，セッショントークンなどが漏洩すると，個人情報の漏洩のみならず通信のコンテンツの改竄も可能になる． そのような機密情報の漏洩を防止する通信プロトコルが HTTPS であるが，HTTPS を導入することで完全に中間者攻撃が防止できるわけではない． HTTPS が保護するのはあくまで「通信経路」のみで，サーバやブラウザのセキュリティ性を担保するものではない点に注意する必要がある [1].

2.2 無線 LAN 関連について

SSID

Service Set Identifier の略で，無線 LAN の識別子を指す． これは一意である必要は無く，その名前は変更可能である為，例えばデフォルトで「Free-WiFi-0123」の SSID を「Kobe-Univ-Free-WiFi」のように変更することが可能である． つまり，SSID を偽造することで公式な公衆無線 LAN の AP になりすます事が可能になる.

Captive Portal

無線及び有線ネットワークに接続したユーザーが，ネットワークへのアクセスを許可される前に表示される認証ページを指す． ユーザーは，アクセス許可に必要な情報の入力或いは利用規約への同意を行い，その認証を行ってもらう事でインターネットへのアクセスが可能になる． Captive Portal を実装しているものの中には，認証終了後に公衆無線 LAN を提供している会社のホームページに自動的に飛ばすものがある.

2.3 その他

HTML

HyperText Markup Language の略で，WWW(World Wide Web) におけるブラウザとサーバ間のデータの送受信に用いられるデータフォーマットを指す． ブラウザに表示する文字のみならず，その大きさ，色，位置を指定することが可能である． また，HTML にはハイパーテキストという機能があり，画面上に表示される文字列や画像に対してリンクを貼り付ける事が可能で，そこから別のページ及びサイトに遷移することができる [2]． この様なリンクのことをハイパーリンクと言い，画面の見た目から直接的にどのようなリンクが埋め込まれているかは確認ができない． つまり，例えば「公式ホームページへ」というテキストに対してリンクが埋め込まれていたとしても，必ずそのリンクが公式ホームページへのもの，或いは公式が提供してるサイトへのものとは限らない． これは，ブラウザの開発者ツールや，ハイパーテキスト上でマウスを一定時間以上置く事でどのようなリンクが埋め込まれているかを確認するこができるが，一般的な利用者で逐一その確認を行う者は極めて少ないと考えられる．その実状は，公式のリンクと偽って偽のサイトへの遷移を促し，機密情報を盗もうとするフィッシングの被害が今もなお絶えない点から伺える（1, [3]）.

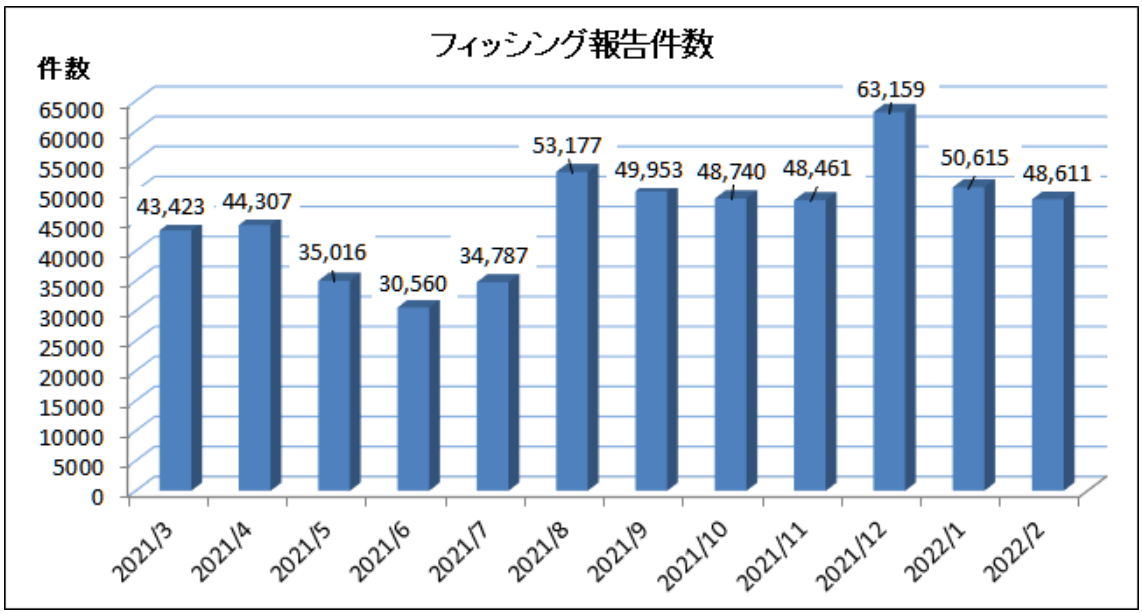


図 1　2021 年 3 月から 2022 年 2 月までのフィッシング報告件数 引用：フィッシング対策協議会

3 目標

本研究の目標を以下に示す.

偽の SSID を用いた公衆無線 LAN からクライアントと正規サーバとの通信に割り込み，ブラウザに警告を出さずに通信の傍受・改竄を行う事.

4 前提条件と仮説

4.1 前提条件

今回，MITM 攻撃を仕掛けるにあたり以下の前提条件を設ける．

利用者は

- 偽 SSID を有する無線 LAN アクセスポイント（以下，AP）を使用する．
- HTTPS 通信で任意のサイトを閲覧する．
- 通信先を確認しない．

まず，偽 SSID を有する AP を利用するという前提条件について説明する． 攻撃者が利用者の通信の内容を盗み見る為には，利用者を攻撃者が用意した AP へ接続させる必要がある． しかし，通常の公衆の AP の SSID は提供している施設や会社の名前に関連したものが多く，利用者も提供元の名前に関連した SSID を有する AP に接続を試みる． 従って，攻撃者が用意した AP の SSID も，利用者が施設や会社が提供した WiFi であると誤認させる様な SSID を用いるという前提が必要である．

次に，利用者は HTTPS 通信でサイトを利用するという条件について，これは現在の Web サイト，特に個人情報などのセンシティブな情報を扱うような Web サイトは常時 HTTPS 通信を用いている点にある． 特に，中間者攻撃への対抗策として HSTP(HTTP Strict Transport Security) という，サーバーがブラウザに対して HTTPS 通信を強制する様な仕組みも実装されているものもある． 従って，通信内容に対して暗号化が施されていない HTTP 通信を前提としての検証は現実的出ない為，HTTPS 通信を想定する必要がある．

最後に，通信先を確認しないという前提についてであるが，これは一般的な利用者の想定を意味する． 通常，ネットワークについての知識が深い利用者や，セキュリティリテラシーの高い利用者は，閲覧しているサイトのドメインなどを確認して，正規サーバと通信しているかを確認する事があるが，一般的な利用者はそのような確認は行わない． 本研究は公衆無線 LAN を用いる一般的な利用者を想定している為，通信先の確認は想定しない．

4.2 仮説

上記の前提条件をもとに，次の仮説を立てる．

> ユーザーが攻撃者が用意した偽の SSID を有する公衆無線 LAN を用い，Captive Portal の仕様を用いて認証させた後，表示された偽の検索エンジンからネットを利用した場合， ユーザーの通信が HTTPS 通信でも，攻撃者がその通信の盗聴・改竄が可能になる．

5 検証内容

5.1 検証対象

MITM 攻撃が可能か否かの検証にあたり，昨今急激な普及が見られるインターネット通販サイトを対象とした．その中でも国内 EC モールの売上ランキング上位 2 つを占める「楽天」と「Amazon」を対象とした．また，これ以降は，攻撃者が用意した悪意ある AP を「悪性 AP」，通信の仲介に用いる攻撃者サーバを「悪性サーバ」をして説明する．

5.2 検証フロー

主な検証フローは次の通り．各フェースの具体的な説明は次のサブセクションに記述している．

1. CaptivePortal を検知させて，予め攻撃者が用意しておいた Captive Portal サイトに誘導し，認証を行わせる．
2. 認証後，偽の検索エンジンの画面へ遷移させ，被害者が検索したワードを悪性サーバで取得する．
3. 攻撃者は取得したワードをバックエンドで検索し，その検索結果を取得する．
4. 取得した HTML ファイル内にあるハイパーリンクを書き換え，そのコピーを被害者に提示する．
5. 以後，クリックした URL を悪性サーバで取得する．
6. 攻撃者は取得した URL を利用して正規サーバにアクセスする．
7. 正規サーバから返却された HTML ファイル内のハイパーリンクを書き換えたものを被害者に提示する．

5.3 実装

悪性サーバの実装は Golang で実装した． 付録に該当の GitHub リポジトリのリンクを載せている． ここでは，検証フローの内容について具体的な実装内容を示す．

5.3.1 検証フロー 1,2 について

前提知識・用語のセクションでも記述したが，Captive Portal とは，インターネットを利用する際にそのユーザーの認証や利用規約に同意させる為に，一時的に外部との通信を制限して認証画面に飛ばす仕組み，或いはその認証ページを指す． 例えばカフェや飛行機，新幹線で提供されている公衆無線 LAN を利用した際などは，認証ページに飛ばされるという経験は決して珍しいものではない． また，要求された認証が終えると，インターネットへのアクセスが許可され，予め用意された別のサイトに飛ばされる事もある． 今回はこの仕組みを利用する．

5.3.2 検証フロー 3 について

ここでは，最も利用されている検索エンジン Google を用いた． Google での検索結果を取得する為には，検索クエリを作成する必要がある． 基本的に任意のクエリ (query) に対して https://google.com/search?q=query という URL が Google の検索 URL となっているが，クエリに

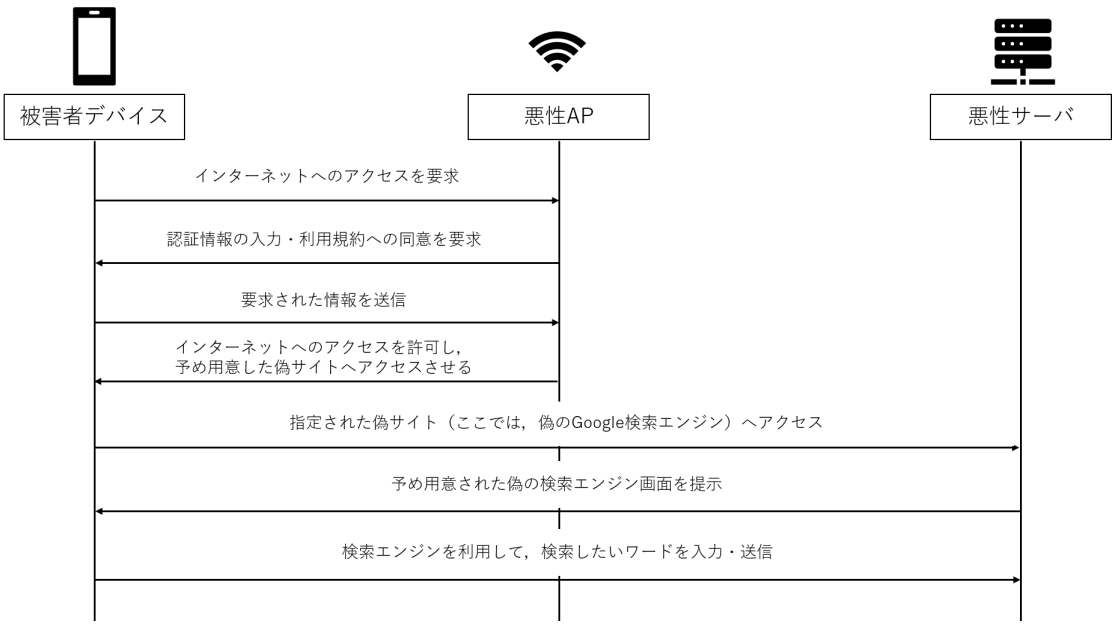


図 2　偽の AP と Captive Portal の仕組みを仕様して，偽の検索エンジンに飛ばすまでの挙動

空文字列が存在する場合は，その空文字列を + に置換する必要があることに留意しなければならない．例えば，「神戸大学　工学部」と検索する場合には，その URL は https://google.com/search?q=神戸大学 + 工学部となる．この URL を叩くことで，正規サーバからの応答を取得・自動生成を行う (3).

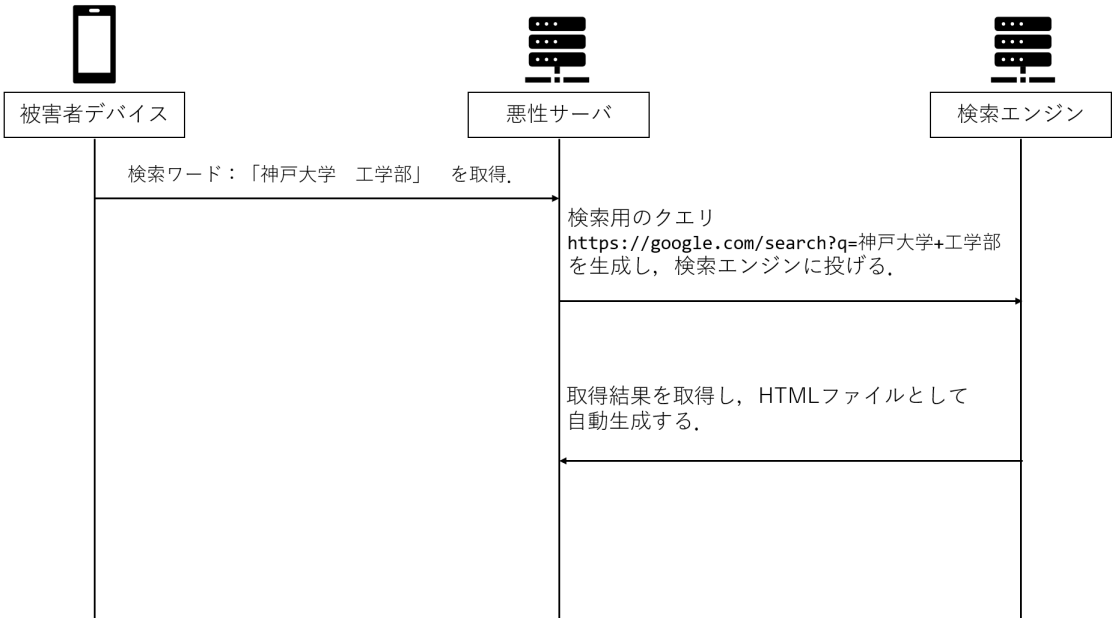


図 3　検索ワードを取得してから，検索結果を取得するまでの流れ

5.3.3 検証フロー 4,5 について

取得した HTML ファイル内にあるハイパーリンクがそのままであれば，悪性サーバではなく正規サーバとの通信に切り替わってしまう． 従って，既存の URL を悪性サーバへ通信するように書き換え，且つ既存の URL を正確に抽出する必要がある． これを実現する為には，既存のハイパーリンクをルールに則って書き換える必要がある． 具体的に，https://example.com という URL に対して処理を行うことを考える． サーバには予め，URL を受け取る為のエンドポイントを設置する．今回の場合は「/templates」というエンドポイントに対して，「url」というパラメータを受け取るものとする． このエンドポイントに対して適切に URL を飛ばすために，サーバ側で予め/templates?url=https://example.com のように書き換える． その結果，被害者に提示した HTML ファイル内にあるハイパーリンクをクリックすると悪性サーバに飛び，且つサーバ側で遷移しようとしたページのリンクを取得できる．

5.3.4 検証フロー 6,7 について

フローの 4 と 5 で得た正規 URL を用いて，バックエンドで正規サーバとの通信及び HTML ファイルの取得を行う． 通常の通信であれば HTML ファイルの取得のみでよいが，個人アカウントへのログインを行う際は，その ID とパスワードを取得し且つ得られた情報と実際に登録されている情報との整合性の確認を行わなければならない． 登録情報の整合性に関しては，取得した個人情報を攻撃者が手動で入力・確認を行わず，バックエンドでブラウザのインスタンスを生成して入力・確認を行う． この処理に関して，Chromedp という Golang で実装されているパッケージを用いた． 具体的な実装については付録から GitHub レポジトリを参照して頂きたい．ここでは，処理の流れを以下に簡単に示す．

1. Chrome インスタンスを生成する．
2. 指定 URL を叩き，JS Path を指定して該当部分に取得した ID やパスワードを入力する．
3. 個人情報の入力が完了すれば，同じくログインボタンの JS Path を指定してログイン処理を行う．
4. 正規サーバに情報を整合させ，返却された HTML ファイル内のハイパーリンクを，4 と 5 と同じ要領で書き換えて被害者に返却する．

JS Path の指定に関しては，予め正規サイトのログイン画面を見てから確認・指定をする必要がある．

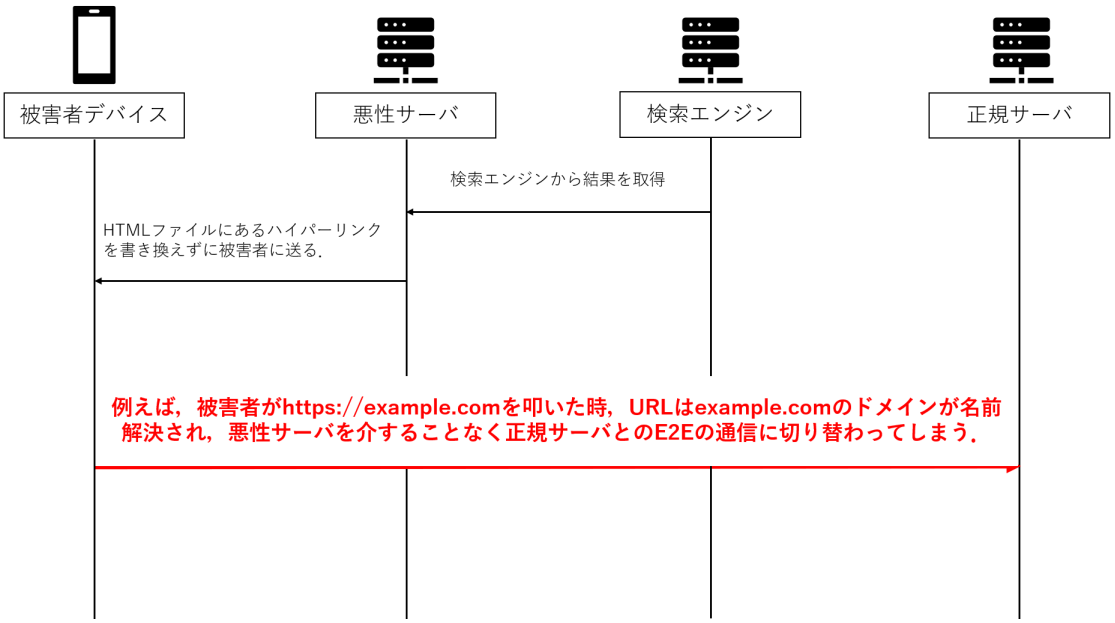


図 4　HTML ファイル内のハイパーリンクを書き換えなかった場合の挙動

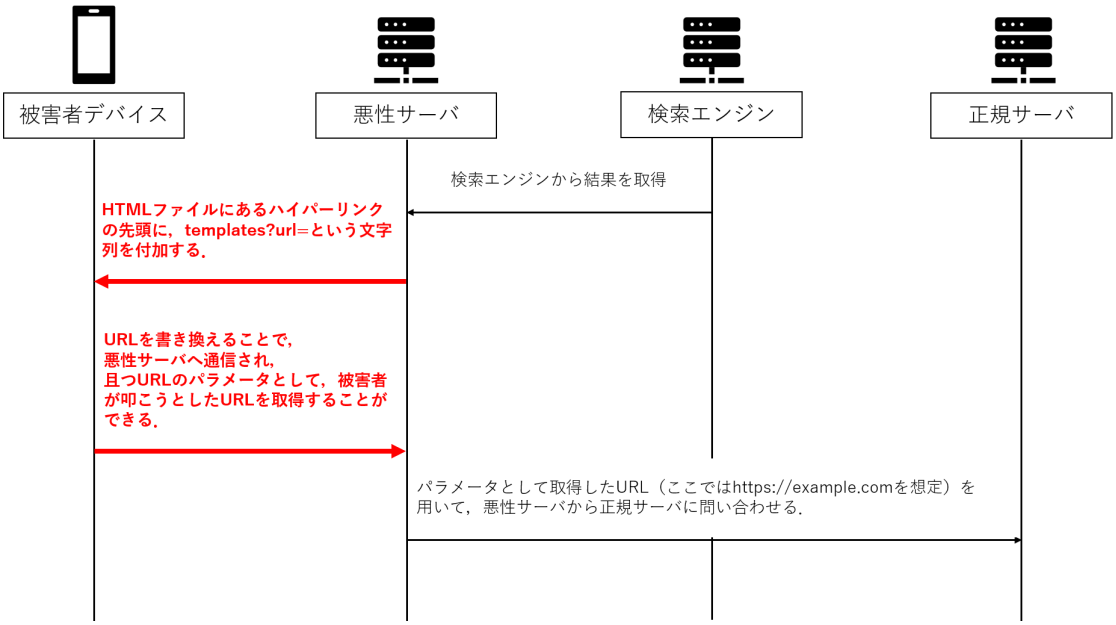
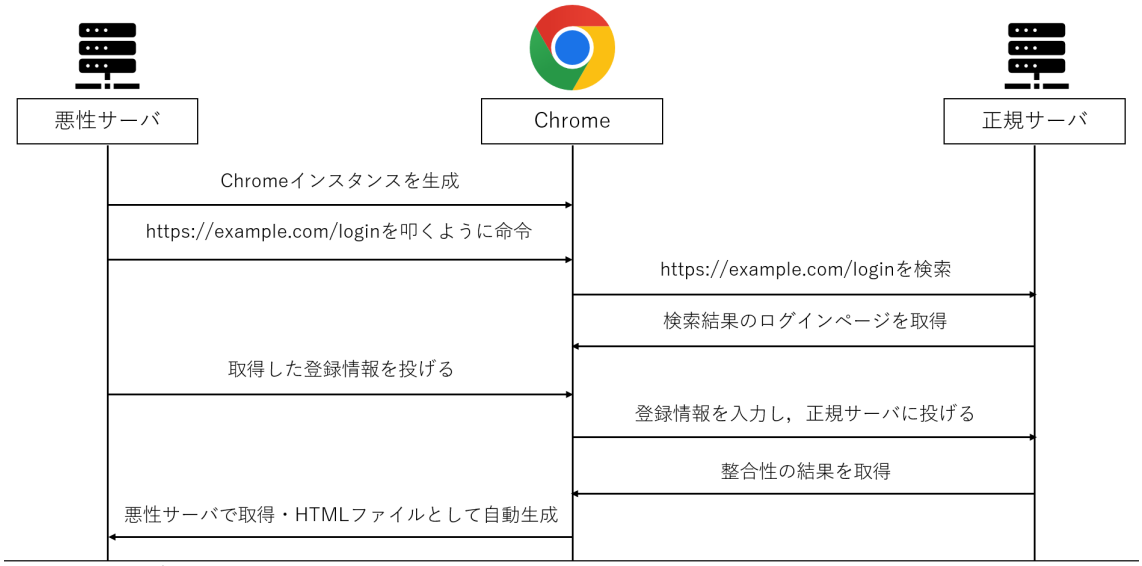


図 5　HTML ファイル内のハイパーリンクを書き換えた場合の挙動



Google Chrome アイコン：
https://www.google.com/url?sa=i&url=https%3A%2F%2Fja.wikipedia.org%2Fwiki%2FGoogle_Chrome&psig=AOvVaw2kssD2Ly7wPm81a55qTbUj&ust=1648885193495000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCPCTmpiu8vYCFQAAAAAdAAAAABAD

図 6　登録情報の整合性の確認を行う流れ

6　検証結果

検証結果は以下のようになった.

6.1　楽天の場合

6.1.1　動作結果

6.1.2　問題点

楽天の場合の主な問題点として，ログイン情報の入力から実際にログインが完了されるまで，ある程度の時間を要する事が挙げられる.

6.2　Amazon の場合

6.2.1　動作結果

6.2.2　問題点

Amazon の場合，場合によれば CAPTCHA に引っかかりログイン画面を突破できない場合があるという問題点が挙げられる． CAPTCHA とは，自動化されたボットによる Web サイトの操作を防止するもので，例えば，人間は認識可能でボットには認識不可能な歪みを含んだ文字列を提示し，その文字列を入力させる事で人間かボットかを認識するようなものが挙げられる（7)． 今回の実装では，Cpatcha を回避或いは CAPTCHA の文字列を被害者に入力させて突破させる事はできなかった.

図 7　CAPTCHA の例：AWS コンソールにログインする際に実際に出る CAPTCHA の一例

7　考察

今回の結果から，

8　参考文献・サイト

参考文献

[1] オライリージャパン，渋川きよし 著，Real World HTTP 一歴史とコードに学ぶインターネットとウェブ技術 第 2 班，14.4 中間者攻撃（MITM 攻撃）

[2] Ohmsha，井上直哉・村山公保・竹下隆史・荒井透・苅田幸雄 共著，マスタリング TCP/IP 入門編 第 6 班，8.5.4 HTML(HyperText Markup Language)

[3] フィッシング対策協議会，2022 年 02 フィッシング報告状況，https://www.antiphishing.jp/report/monthly/202202.html

9　付録

9.1　実装コード

実装コードの GitHub リポジトリ　　https://github.com/ES3-Kobe-U/go-malproxy