CL-006

SSL/TLS を回避する中間者攻撃の新たな提案とその脅威

～実装による評価と考察～

New Proposals and Threats of Man-in-the-Middle Attacks to Evade SSL/TLS

～Evaluation and Consideration by Implementation～

木村 圭一朗† 白石善明‡ 森井昌克‡
Keiichiro Kimura Yoshiaki Shiraishi Masakatu Morii

1. はじめに

スマートフォンの普及に伴う屋外の快適なネットワーク環境の需要が高まっており，その具体的なインフラとして，公衆無線 LAN が挙げられる．現在では，公衆無線 LAN は日常の様々な側面に溶け込んでおり，その一例として，新幹線や飛行機，カフェや宿泊施設など，日常生活に欠かせない施設が挙げられる．総務省は，災害時の情報収集や通信手段として，また観光情報の収集や教育現場での活用による利便性の向上を目指し，平成 30 年までに全国約 3 万箇所に公衆無線 LAN を設置する案をまとめた[1].

このように，公衆無線 LAN の普及が加速的に行われている中，そのセキュリティ対策も課題となっている．特に，公衆無線 LAN には，大衆利用を想定した特性上の多くの脅威が潜んでおり，IPA（独立行政法人情報処理推進機構）による「公衆無線 LAN 利用に係る脅威と対策～公衆無線 LAN を安全に利用するために～」の資料[2]には，主に 1. 盗聴 2. なりすまし 3. 悪意ある AP（AP：アクセスポイント）4. 不正目的でのインフラの利用，の 4 つの脅威が示されている．このように，公衆無線 LAN には様々な脅威が潜んでおり，その対策は現在も大きな課題となっている．

今回，この 4 つの脅威の中の「悪意ある AP」に着目し，「悪意ある AP」を起点とした中間者攻撃の手法ついて議論している．「悪意ある AP」には，そもそも通信内容の傍受・窃取や機密性の高い情報の盗聴を目的として設置されたものが多いため，その攻撃起点は様々であり，アクセスポイントの利用者が，一般的な利用者，つまりインターネットや通信・セキュリティに関する専門的知識を有さない者を想定した場合，利用者はその脅威を認識することは非常に困難である．本研究では，Captive Portal の仕組みを起点とし，攻撃者がクライアントとサーバ間の通信に割り込む形をつくることで，通信が SSL/TLS 通信でも，クライアントのブラウザに警告を出さずに通信内容及び機密情報の傍受・窃取を行う手法を考え，そのシステムの設計・実装を行い，実際に中間者攻撃に成功した． Captive Portal を用いる主な目的は，攻撃者が通信の中継形態をつくる為に，認証後に遷移させる偽の Web サイトに飛ばす事である．具体的には，予め設置した偽 AP に対して認証要求を設定し，認証後に偽の検索エンジンに誘導させる事で通信の中継機会をつくり，その個人情報の傍受・窃取を成功させ，その有効性を確認した.

2. 中間者攻撃とその現状

中間者攻撃とその対策は，これまでの研究で十分に検討されている．SSL/TLS 通信に対する中間者攻撃として，2009 年の Black Hat で Moxie Marlinspike 氏によって提案された SSL Strip 攻撃がある[3]．これは，クライアントと攻撃者間の通信を HTTPS 通信から HTTP 通信へ置換することで，クライアントの機密情報を傍受・窃取する手法である．この手法に対しては，Web サイトがブラウザに対して HTTP の代わりに HTTPS を用いて通信を行うように強制する HSTS(HTTP Strict-Transport-Security)を用いることでその実行を防ぐことが可能となった．Takashi Setozaki の研究[4]によれば，クライアントがアクセスしようとしているドメインがブラウザの HSTS リストに存在しない前提のもと，サーバから送られてきた HTML ファイル内の"https://"から始まるリンクに対して，そのドメイン名を HTTPS 接続する直前に HTTP 接続していたドメインにすり替えることで HSTS を回避する SSL Strip 攻撃の手法を提案した．しかし，前提条件にもあるように，対象のドメインが HSTS リストに存在した場合はこの手法による SSL Strip 攻撃は困難と考えられる．また，菊池浩明らの研究[5]では，偽造 DNS サーバを構築することで攻撃サーバを経由させるように通信させ，且つ任意のサーバに対して証明書を自動的に作成するプログラムを構築することで，動的に SSL 中間者攻撃を行う手法が提案されている．サーバの偽造証明書は，Open SSL を用いて正規サイトのサーバ証明書を偽装した自己署名証明書を作成することで実現させている．しかし，結果として生成された証明書は本来の証明書を完全に偽造することは不可能とされており，ブラウザによる証明書検証の際の警告出現の可能性は拭えているとは言えない.

3. Captive Portal を利用した中間者攻撃

3.1 Captive Portal

Captive Portal は攻撃起点として利用されることが多く，その攻撃手法に対する対策は，既に多くの研究で十分に議論されている．例えば，LUNODZO J. MWINUKA らは，正規 AP になりすまして中間者攻撃を行うような悪意ある AP の検出を行う Android アプリケーションの開発を行った[6]. その中で，Captive Portal を起点として認証情報の窃取を行うような悪意ある AP に対し，ランダムな認証情報の入力を行うことで対象の AP の真正性を確認する手法を提案している．また，Vineeta Jaina らは，ビーコンフレームから

† 神戸大学工学部電気電子工学科
Department of Electrical and Electronic Engineering, Faculty of Engineering, Kobe University
‡ 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University

フィンガープリントを生成し，その情報と SSID や信号強度との比較を行うことでワイアレス Evil Twins への接続を未然に防止する ET Guard というアプリケーションを開発した[7]．そこで想定されている攻撃シナリオとして，ユーザーが悪意あるインテントが組み込まれたアプリケーションをインストールしているという前提のもと，Evil Twins が提供する Captive Portal を起点として，その悪性あるインテントを起動させてポート開放などの危険性を生み出す手法が想定されている． ET Guard は，それらの AP に接続する前に検出を行うことでインテントの起動を未然に防いでいる．しかし，これらの想定された攻撃手法は，偽 AP が提供する Captive Portal の認証画面を起点として認証情報あるいは個人情報を窃取する方法であり，必ずしも中間者攻撃とは言えなかった．

2 章でも述べたように，既存の中間者攻撃の手法では，HSTS によって強制的に HTTPS 化されたり，自己署名証明書ではブラウザに警告が出現したりと，通信の傍受・窃取が困難と考えられる．また，Captive Portal を起点とした攻撃手法に関する既存研究では，認証情報の窃取を目的としたものが想定されており，認証終了後の遷移画面を起点とした攻撃手法については議論されていない．そこで我々は，認証終了後の遷移画面に偽の検索エンジンを用意して通信の中継機会をつくり，SSL/TLS 通信の影響を受けずに通信内容の傍受・窃取を行う手法を考え・その実装を行った．

3.2 中間者攻撃の攻撃手順

攻撃手順は次の通り．

ステップ１．認証・画面遷移フェーズ
　Captive Portal 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．そして，認証後は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．

ステップ２．検索結果取得フェーズ
　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．

ステップ３．リンク書き換えフェーズ
　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．以後，クリックした URL を悪性サーバで取得する．

ステップ４．サイトアクセス中継フェーズ
　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．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，被害者に提示する．

一連の基本的な動作は上記の各フェーズに従うが，個人アカウントへのログインなどの認証情報を要するフェーズでは，攻撃手順のステップ４で，取得した認証情報を窃取するだけでなく，その情報と実際に登録されている情報の整合性の確認を自動化して行う．

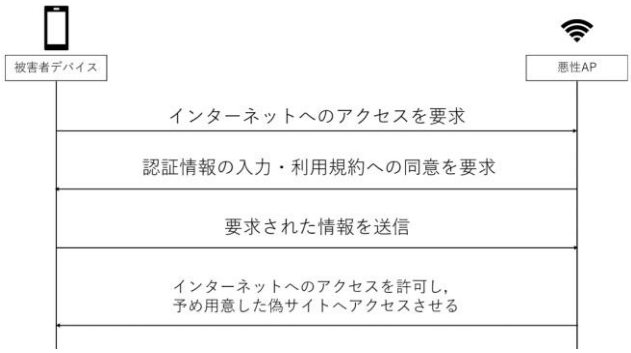

図 1： 偽 AP と Captive Portal の式身を利用して，偽検索エンジンに飛ばすまでの挙動

3.2.1 認証・画面遷移の方法

図 1 に，具体的な挙動を示す．まず，被害者は疑うことなく悪性 AP への接続を試みるとする．使用している AP は Captive Portal を実装している為，接続要求をした被害者デバイスに対して認証を行うように命じる．被害者は認証を行い，認証が終了すると被害者デバイスに対して，悪性サーバへのリダイレクトを命じる．続いて，図 2 のフェーズに移る．被害者デバイスは，命令通りに悪性サーバにリダイレクトする．今回は，悪性サーバは偽の検索エンジンを装い，被害者はそれを疑うことなく用いるものとする．偽の検索エンジンでは，入力されたワードを悪性サーバ側で取得するように実装されている．今回は，サーバ側に /google というエンドポイントを設置し，検索ワードを params という名前で取得するように実装した．

3.2.2 検索結果の取得方法

ここでは，最も利用されている Google の検索エンジンを用いた． 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，クエリに空文字列（全角と半角の両方含む）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　電気電子工学科」と検索する

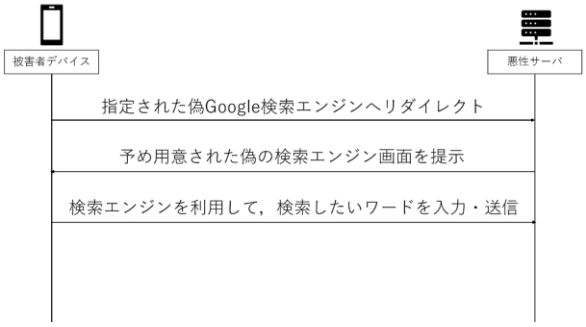

図 2：偽検索エンジンにリダイレクトし，検索を行うまでの流れ

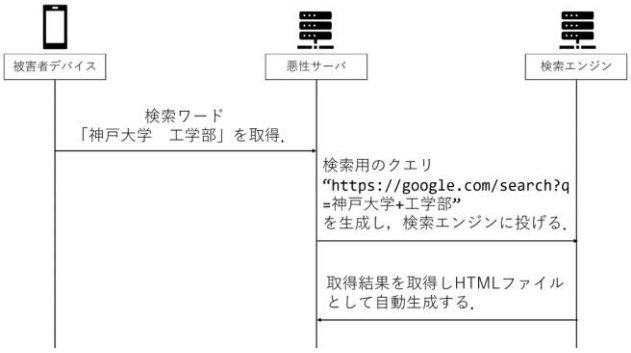

図 3：検索ワードを取得してから，検索結果を取得するまでの流れ

る場合には，その URL は”https://google.com/search?q=神戸大学+工学部+電気電子工学科”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(図 3)．自動生成に関しては，取得した HTML をサーバ内の指定ディレクトリ下に設置するだけではなく，サーバが読み込めるように，{{define KEYWORD}} + string(HTML Code)+ {{end}} として，HTML ファイルに記述・保存する必要がある．

3.2.3 リンクの書き換え方法

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（図 4）．
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サーバで取得・処理するものとする．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，

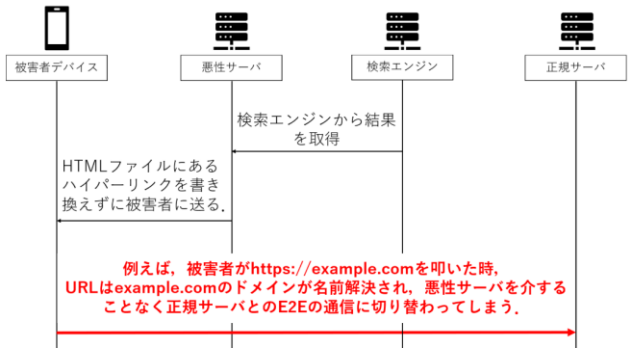

図 4：HTML ファイル内のハイパーリンクを書き換えなかった場合の挙動

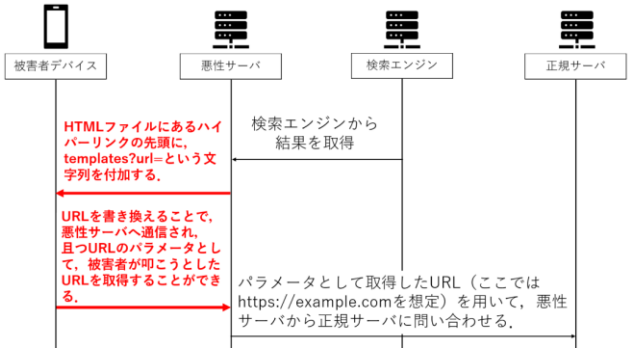

図 5：HTML ファイル内のハイパーリンクを書き換えた場合の挙動

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（図 5）．しかし，URL 内に特定の文字及び文字列が混在している場合には留意しなければならない．具体的には AND を意味する`&amp;` `&` `%26`，また EQUAL を表す`=`がある場合である．URL 内にこれらがある場合は，特定の文字列に置き換える必要がある．これは，URL Encode 及び Decode を用いても回避することは困難である．理由としては，今回の場合，該当するエンドポイントで取得するパラメータが params の 1 つである為である．従って，取得した HTML 内にあるハイパーリンクを探索し，それらの URL に対して，特定の文字・文字列を探索・特定の文字列に置換する操作を行う．

3.2.4 サイトアクセスの中継方法

ステップ 4 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．認証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（付録 2 参照）．具体的な実装については付録 1 の GitHub レポジトリを参照して頂きたい．ここでは，処理の流れを示す（図 6）．

1. Chrome インスタンスを生成する．
2. 指定 URL を持たせ，Chrome インスタンスに対してナビゲートするように命令する．
3. 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．
4. 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．
5. 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，攻撃過程の 4 と 5 と同じ要領で書き換えて被害者に返却する．

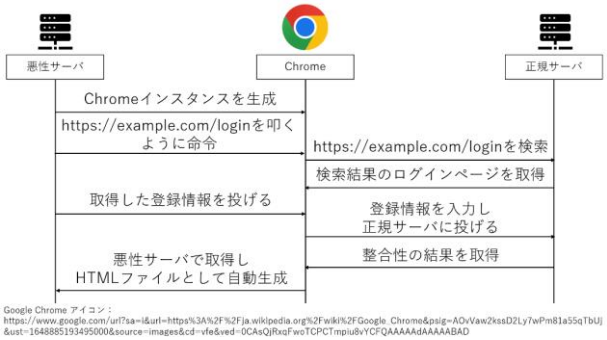

図 6：認証情報の整合性の確認を行う流れ

ここで，Chrome インスタンスとは，Chromdp が提供する，現行の Google Chrome とほぼ同様の機能を持った Context を指す．この Context を利用して，バックグラウンドで Google Chrome の検索エンジンを起動させる．そして，起動させた Chrome に対して認証画面の URL を叩き，その画面へ遷移する．そして，受け取った認証情報を，JS Path を指定して入力し，ログイン処理を行うという流れである．ここで，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．

4. 新たな中間者攻撃の問題点とその克服

我々は，3 章で述べた新たな攻撃手法について，攻撃システムを設計・実装し，実環境上でのシステムの攻撃可能性を検証した．本章では 3 章を受けて，その実装と問題点，そしてその考察を与える．

4.1 中間者攻撃システムの設計・実装

3.1.5 にも述べたが，システムは 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．システムは主に 6 つのモジュールで構成されている．各モジュールの機能と関係は以下の通りである（図 7）．

1. Router
   被害者からのリクエストをエンドポイントごとにルーティングし，Handler に渡す．
2. Handler
   リクエストからパラメータを抽出・整理し，Usecase に渡す．
3. Usecase
   Handler から渡されたパラメータを元に，具体的な処理を行う．また，正規サーバへのリクエストの送信及びレスポンスの取得を行う．
4. Template
   Usecase から自動生成されたファイルを管理する．Usecase は Handler に対して被害者に表示すべきファイルを指定し，その情報をもとに Handler は Template から返却すべきファイルを選択する．
5. Params
   Template に自動生成ファイルを置く前に，そのファイルから削除・置換すべき文字列を管理する．Usecase はこれらの情報をもとに，自動生成ファイルの加工を行う．
6. Public
   攻撃者が予め用意しておく画面の画像リソースなどを管理する．

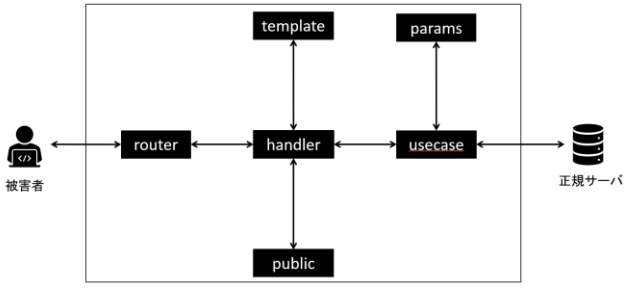

図 7：システムの設計

4.2 実装方法

実験機器として使用する AP は，Captive Portal を設定できるものであれば，危機に依存しない．今回は，"GL.iNet GL-AR300M(shadow) 無線 LAN vpn トラベルルーター Openwrt インストール 中継器ブリッジ 11n/g/b 高性能 300Mbps 128MB Nand フラッシュ OpenVPN/WireGuard クライアント/サーバー 日本語界面"を使用し，認証後に実装したシステムに飛ばすように設定した．偽検索エンジンは Google Chrome を想定した．被害者デバイスとして用いた機器は，PC は Microsoft Surface Laptop4 (OS Windows 10)，スマートフォンは Pixel 3XL(OS Android12)を利用した．攻撃対象は，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．通販サイトは，幅広い利用者が利用する点に加え，個人アカウントを持つものが多数である．従って，通信内容及び機密性の高い情報の傍受・窃取をおこなうターゲットに適していると考えた．

評価方法としては，利用者は攻撃者が用意した AP を利用し，且つ通信先の確認を行わないという前提のもと，攻撃者が通信の中継を行ったまま，実際に「楽天（rakuten.co.jp）」と「Amazon（amazon.co.jp）」の Web サイトに侵入し，サイトが閲覧できるか，またログイン情報を入力してログインできるかを確認して評価を行う．

図 8：GLiNet の認証画面

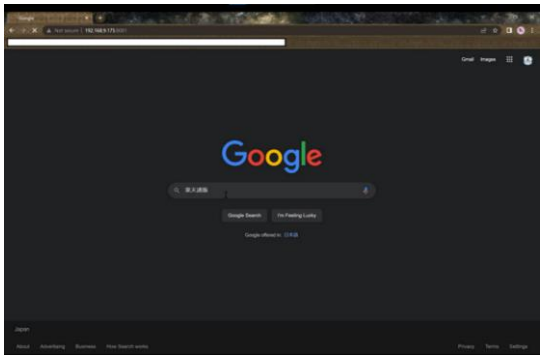

図 9：偽の Google 検索エンジン

4.3 実験結果

4.3.1 rakuten.co.jp に対する攻撃の評価と問題点

まず，攻撃者は用意した偽の AP を設置する．被害者が偽 AP に接続を要求し，認証画面へ遷移させられる（図 8），認証が終了すると偽の検索エンジンが表示される（図 9）．ここの検索欄に「楽天通販」と検索して，その検索結果を待つ．「楽天通販」と検索して得られた結果を表示する（図 10）．ここで，被害者は Google の検索結果が得られたものと認識し，表示された楽天通販の結果から該当のサイトへのリンクをクリックする．ここまでの動作は，図 に準じて処理されたものである．

得られた検索結果の画面から，楽天通販サイトに入る．特に目立った文字化けや画像切れは起こっていない事から，ここまでも被害者は．楽天通販サイトに遷移したと認識するものとする．ここで，個人アカウントに入る為に図 11 の右上の「ログイン」ボタンを押す．「ログイン」ボタンを押した結果は図 12 のようになる．このログイン画面は，攻撃が予め用意して攻撃者サーバにパラメータを送信させるように実装されたものである．ここに，実際に自分のアカウント情報を入力し，「ログイン」ボタンを押す．

入力されたパラメータを攻撃者サーバ内から処理をし，正規サーバから返却された結果を被害者に提示する．図 13 は，実際に個人アカウントに侵入した結果である．ここまでの動作は図 6 に準じて処理されており，目立った警告や画像切れ無く且つ正規サーバとの通信を中継する形で個人アカウントへのログインに成功した．

図 10：偽検索エンジンで「楽天通販」と検索した際の画面

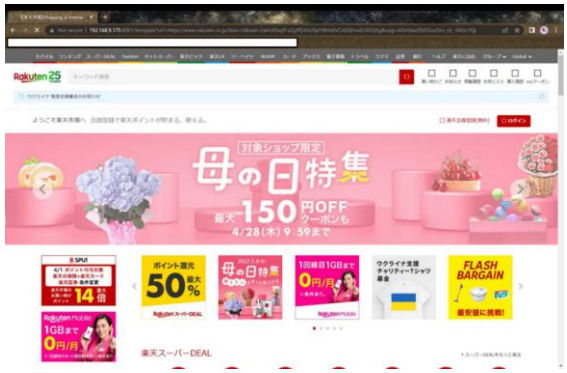

図 11：検索結果の画面から楽天の複製サイトへ侵入した際の画面

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．具体的な実行時間の結果を表 1 に示す．この原因は，主に二つ考えられる．

一つ目は，ログインのプロセスに複数の処理が絡んでいる点にある．具体なプロセスは，図 6 に示すプロセスに加え，結果の出力と悪性サーバに通信するようにするための URL の書き換えプロセスが含まれている．しかし，書き換えプロセスには実行時間をほとんど要さない為，実行時間の長さは Chromedp の一連の処理による部分が大きいと考えられる．具体的には，認証画面へのナビゲート，JS Path を参照したデータの入力，クリックに伴う画面遷移での実行時間及び待機時間が挙げられる．

二つ目は，Chromedp からログイン結果を取得しようとした場合，最低でも 13 秒程度待機しなければ，ログイン後の画面の HTML を確実に取得できない点にある．これは，実装段階で待機秒数を決定する時に判明した．待機時間に，13 秒未満の任意の秒数を指定した場合，ログイン前の画面が返却されるか，真っ白な画面が返却されるか，そのどれかであった．今回はログイン後の画面を確実に取得する為に，デフォルトで 13 秒待機を設定している為，実行時間が必然的に長くなってしまっている．

図 12：偽のログイン画面へ入り，登録情報を入力した際の画面

図 13：ログイン情報の整合性がとれ，実際に個人アカウントへログインした後の画面

以上の二つの問題点は，どちらも Chromedp に大きく依存しており，外部ライブラリに依存しない実装部分での実行時間短縮が困難であった．ネットワーク環境にもよるが，認証完了までの時間が長い場合，被害者の通信に対する意識を集中させる可能性は排除できない．

表 1：楽天と Amazon のログインにかかる時間比較

	楽天	Amazon
1 回目	24.73[s]	6.64[s]
2 回目	24.65[s]	6.98[s]
3 回目	24.43[s]	7.30[s]
4 回目	24.81[s]	7.35[s]
5 回目	24.43[s]	7.27[s]
6 回目	24.86[s]	7.55[s]
7 回目	24.37[s]	7.51[s]
8 回目	25.43[s]	7.45[s]
9 回目	24.38[s]	7.63[s]
1 0 回目	24.61[s]	7.31[s]

4.3.2 amazon.co.jp に対する攻撃の評価と問題点

まず，図 8 と同様に悪性の AP にアクセスし，認証を終了させた後，図 9 のように偽の Google 検索エンジンが表示され，Amazon 通販と入力して検索結果を取得し（図 14），そこから複製された Amazon の通販サイトに遷移した．続いて，ログインを行う為に画面右上にあるログインボタンを押して個人アカウントへの侵入に試みた（図 15）．

図 14：「Amazon 通販」と偽検索エンジンで検索した結果

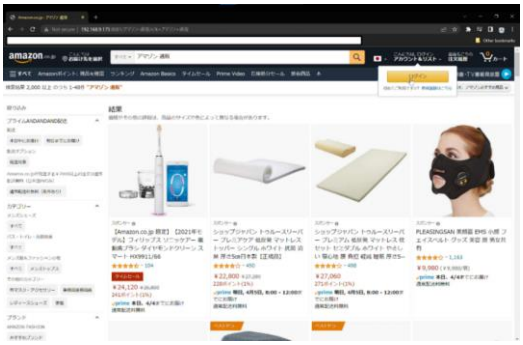

図 15：「Amazon 通販」と検索した結果から，Amazon の複製サイトへ遷移した際の画面

ログインボタンを押すと，ログイン画面に遷移した（図 16）．この画面は予め攻撃者側で作成したもので．HTML 内で定義されて入り入力フォームとその入力データの送信を悪性サーバ側で取得するように細工している．この入力フォームに，実際にアカウント情報を入力してログインボタンを押して，攻撃者サーバに情報を送信した．サーバ側で取得した情報は，図 5 の手順に従い処理を行った．そして，入力情報の整合性の確認が完了し，実際に個人アカウントへの侵入に成功した．図 16 には，実際に個人アカウントの名前が明記されていることが分かる．ここまでの動作は図 6 の動作に従って処理されており，楽天と同様に目立った画像切れや警告を出さずに，且つ正規サーバとの通信を攻撃者が中継する形で個人アカウントに侵入することができた．

問題点としては，Amazon の場合，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（図 18）．今回の実装では，CAPTCHA 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 16：ログイン画面へ遷移し，認証情報を入力した際の画面

図 17：ログイン情報の整合性がとれ，実際に個人アカウントへログインした後の画面

4.4 考察

今回，「楽天」と「Amazon」の二つの通販サイトに絞ってシステムの構築を行ったが，両者ともブラウザ上での警告を出さずに個人情報の傍受・窃取が可能であることが分かった．加えて，被害者のブラウザに対しても明確な警告の表示や画像切れが見られなかった為，今回実装したシステムが通信の中継としての役割を十分に担う事ができていると評価できる．

しかし，いくつかの問題点も残されている．前節の検証結果でも述べたものを改めて列挙すると，次のようになる．

1. （楽天の場合）登録情報の整合性の確認にやや時間を要する点．
2. （Amazon の場合）CAPTCHA に引っかかり，ログインできない場合がある点．

まず 1 についてであるが，これは 4.1.2 問題点にも記述したように，Chromedp を利用して個人情報の整合性の確認を行う際に非常に時間を要している点が主な原因と考えられる．具体的な処理プロセスは以下の 5 つの工程に分かれている．

（1）Chrome インスタンスの作成
（2）ブラウザを開き楽天のログイン画面へ遷移
（3）指定の JS Path を検索し個人情報を入力
（4）ログインとその認証
（5）得られた結果を HTML として抽出

この 5 工程の中で特に実行時間を要しているのが，ログイン後の結果を完全に取得しようとした際，（4）と（5）の工程で，その実行時間は計約 13 秒程度となっている．4.1.2 にも記述したが，検証の結果，ログイン後の正確な情報を得る為には最低 13 秒時間待機する必要があり，これが主なボトルネックとなっている．今後の課題としては，Chromedp に依存していない部分の実行時間の削減に加え，今回の処理が別の言語で実行できる場合，Golang の場合との実行時間の比較を行い，ボトルネックの解消の可能性を探ることである．特に，Chromedp を筆頭とした外部ライブラリへの依存を減らし，Golang の標準ライブラリで実装されている Context にセッション情報を保持させたままの実装を確立させることで，実行時間の削減方法を克服する．

図 18：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

続いて，2 についてであるが，CAPTCHA に引っかかる主な原因は，個人情報をプログラムで入力している点が原因である．ブラウザ情報を維持した Chrome インスタンスを CAPTCHA の回避を目的として再度別の処理で再利用する場合，Chrome インスタンスのリロードが必要となる．しかし，Chrome インスタンスのリロードはブラウザ画面のリロードと同様の挙動を示すため，その際に CHATCHA として表示された文字列も変わってしまう．従って，被害者が CAPTCHA の文字列を入力したとしても，結果的に整合性が取れない．認証情報を機械的に入力することでの CAPTCHA の発動の防止策の一例として，試験的にランダム時間ごとに認識を入力する実装を行ったが，それでも CAPTCHA を回避することはできなかった．認証の際に毎回 CAPTCHA が検出されることは無いが，実際に起動してしまった場合は，認証情報の窃取は可能であるが，その整合性の確認及び個人アカウントへのログインができなくなってしまう．この CAPTCHA の問題を解決するその他の具体的な実装は，現状できていないため，今後も克服方法を考えていく必要がある．

5. むすび

本稿では，Captive Portal の認証後に飛ばされるサイトを起点として，クライアントとサーバ間の通信に攻撃者を中継させ，通信内容及び機密情報の傍受・窃取を行う手法について提案し，その有効性を確認した．これまでの研究では，HSTS でないドメインを利用した手法や，動的に自己署名証明書を作成する手法で中間者攻撃を行うことが議論されてきたが，昨今のブラウザやサイトの進化により，HTST を強制するサイトが増加し，また自己署名証明書を利用したサイトに対してはユーザーに警告を出すなどの対策が講じられており，これらの手法での中間者攻撃の確立は困難であった．また，Captive Portal を起点とした攻撃手法も十分に検討されているが，主に認識情報の窃取を目的としたもので，通信の傍受・窃取の手法については十分な議論がなされていない．本研究の手法では，Captive Portal が認証後に指定の画面への遷移を行うことができる仕組みに着目し，遷移先を攻撃者サーバとすることで，容易に通信上での攻撃者の中継機会をつくった．この手法では，クライアントはセッションフェーズから正規サーバと直接やり取りをすることはない為，サーバ側が SSL/TLS 通信を強制するものに関係なく，通信の傍受・窃取を行う事が可能である．また，実験結果からブラウザ上に目立った警告や画像切れが出現しなかった為，被害者は AP 及び通信先の確認を行わない限り攻撃の検出は困難と言える．従って，本研究での提案手法を用いれば，これまでの中間者攻撃の手法よりも容易に攻撃が可能になり，被害者側は通信内容や機密性の高い情報を盗まれる危険性が非常に高く，現実的脅威のある攻撃であると言える．

謝辞

本研究は，総務省の「電波資源拡大のための研究開発(JPJ000254)」における委託研究「安全な無線通信サービスのための新世代暗号技術に関する研究開発」の成果の一部である．また，本研究の一部は JSPS 科研費 20K11810 の助成を受けたものである．

参考文献

[1] 総務省，2020 年に向け全国約 3 万箇所の Wi-Fi 整備を目指して，https://www.soumu.go.jp/main_content/000548781.pdf

[2] IPA，公衆無線 LAN 利用に係る脅威と対策 ～公衆無線 LAN を安全に利用するために～，https://www.ipa.go.jp/files/000051453.pdf

[3] Moxie Marlinespike: New Tricks For Defeating SSL In Practice

[4] Takashi Setozaki, Kazuto Matsuo: An Enhanced Sslstrip Attack aginst HTTPS with HSTS

[5] MATSUMOTO Hiroaki(Tokei University), KODAMA Yosuke(Data Management), KIKUCHI Hiroaki(Meiji University), ISHII Hiroshi(Tokai University): On the risk of SSL Man-in-the-Middle Attack with dynamic fake SSL server certificates and DNS servers

[6] LUNODZO J. MWINUKA, ABEL Z. AGGHEY, SHUBI F. KAIJAGE, (Senior Member, IEEE), AND JEMA D. NDIBWILE: FakeAP Detector: An Android-Based Client-Side Application for Detecting Wi-Fi Hotspot Spoofing

[7] Vineeta Jaina, Vijay Laxmi, Manoj Singh Gaur, Mohamed Mosbah: ETGuard: Detecting D2D attacks using wireless Evil Twins

付録

1. 実装コードの GitHub リポジトリ
   https://github.com/ES3-Kobe-U/go-malproxy
2. Chromedp の GitHub リポジトリ
   https://github.com/chromedp/chromedp}