

第1章 はじめに

インターネットの急速な普及に伴って、情報のアクセスが容易になり、我々の生活が豊かになった一方で、インターネットを悪用したサイバー犯罪が横行している。

サイバー犯罪は、サーバやアプリケーションの脆弱性を利用して、アクセス権限を要する情報を不正に入手したり、サーバに莫大な負荷をかけて、サービスに支障をきたしたりする行為のことである。サイバー犯罪の被害として、顧客の個人情報流出したり、サービスが長期間、利用できなくなることなどが挙げられる。これらの被害は、サービスを利用する顧客にとっても、サービスを運営する企業にとっても大きな損失となるため、サイバー犯罪の被害を未然に防ぐことはもちろん、被害に遭った際に犯人を特定するための対策を講じることも重要である。

サイバー犯罪の犯人を特定する情報として、多くの場合、IP アドレスが用いられる。サイバー犯罪の被害を受けたサーバのログから、サイバー犯罪に該当するアクセスの IP アドレスを捜査し、インターネットサービスプロバイダに照会を要請することで、犯人の身元を特定することができる。

しかし、技術の進歩により、近年では、IP アドレスのみによる個人の特定は困難になってきている。VPN や Tor を用いることで、インターネットサービスプロバイダから割り振られている本来の IP アドレスを秘匿化してインターネットに接続することができる。また、キャリアグレード NAT の普及により、通常より IP アドレスでの個人の特定が難しくなっていることも事実である。これらの技術についての詳細は、次章で後述する。

そこで、本論文では、サイバー犯罪が発生した際に、IP アドレス以外の方法で犯人を特定する手法について提案する。インターネット利用者が、サーバに送信する情報は複数挙げられるが、その中でも Cookie に着目した特定システムを提案する。Cookie は、Web サーバが Web ブラウザに対して、任意の文字列を記憶させるための仕組みである。サーバから Cookie を保存するように指示されたブラウザは、以降、その Web サイトに対して、記憶した Cookie を送信するようになる。

Cookie を利用することで、サービスのログイン状態を記憶したり、EC サイトにおいて、ショッピングカートの中身を記憶させたりすることができる。一方で、アクセスしてきたユーザに、ユニークな文字列を発行し、Cookie として保存させることで、過去にそのユーザがアクセスしてきたかどうかをわか

るため、個人を特定するために用いられることもある。このように、Cookie はユーザを特定する十分な情報となり得る。

第2章 研究背景

2.1 IP アドレスによる特定の難しさ

発信元の IP アドレスを特定することで、発信者の利用しているインターネットサービスプロバイダやキャリア、地域レベルの位置情報を取得できることが多い。サイバー犯罪が発生した際に、IP アドレスからインターネットサービスプロバイダを特定し、インターネットサービスプロバイダに照会を要請することで犯人を特定することができる。

しかし、技術の進歩により、IP アドレスによる個人の特定は困難であるケースも珍しくない。IP アドレスによる個人の特定が困難となる技術の代表例を以下に示す。

2.1.1 キャリアグレード NAT

キャリアグレード NAT とは、インターネットサービスプロバイダなどが、ネットワークアドレス変換 (NAT) を行う仕組みである。ネットワークアドレス変換とは、同一 LAN 内の各端末に割り振られたプライベート IP アドレスと、インターネットサービスプロバイダから割り振られたグローバル IP アドレスを変換するための仕組みである。IPv4 アドレスの枯渇問題により、インターネットに接続する各端末すべてにグローバル IP アドレスを割り振ることは困難である。そこで、各端末には、グローバル IP アドレスの代わりに、プライベート IP アドレスを割り振り、インターネットに接続する際に、プライベート IP アドレスをグローバル IP アドレスに変換することで、LAN 内の複数の端末を同じグローバル IP アドレスでインターネットに接続させることができる。ネットワークアドレス変換は、各家庭の LAN 内で行われるが、これをインターネットサービスプロバイダ単位で行ったものがキャリアグレード NAT である。

IPv4 アドレスの枯渇問題を解消する手段として、キャリアグレード NAT がしばしば利用されているが、同一のグローバル IP アドレスが、複数のインターネット利用者によって使用されるため、キャリアグレード NAT を使用している場合は、IP アドレスによる個人の特定が困難となる。インターネットサービスプロバイダでは、キャリアグレード NAT を利用した際の IP アドレスの変換テーブルを保持しているが、変換テーブルは、最新の約 2 ヶ月分

しか記録されていないため、保存期間を過ぎると、IP アドレスによる特定ができなくなる。

2.1.2 VPN

2.1.3 Tor

2.2 IP アドレスによる誤認逮捕

2.3 Cookie による個人の特定

IP アドレスによる個人の特定は困難であったり、誤認逮捕を招く可能性があるため、サイバー犯罪において犯人を特定する情報としては、十分とはいえない。そこで、本論文では、Cookie による個人の特定可能性に着目した。

Cookie とは、Web サーバが、Web ブラウザに対して、任意の情報 (文字列) を記憶させるための仕組みである。サーバから Cookie を保存するように指示されたブラウザは、以降、その Web サイトに対して、記憶した Cookie を送信するようになる。

Cookie を利用することで、サービスのログイン状態を記憶したり、EC サイトにおいて、ショッピングカートの中身を記憶させたりすることができる。一方で、アクセスしてきたユーザに、ユニークな文字列を発行し、Cookie として保存させることで、過去にそのユーザがアクセスしてきたかどうかを調べることができる。このように、Cookie はユーザを特定する十分な情報となり得る。

ブラウザがサーバに送信する情報として、使用している言語、ブラウザや OS の種類、バージョンなどが挙げられるが、これらは他のユーザと似たような情報となることが多い上、本来とは異なる情報を送信することも可能である。それに対して、Cookie は、サーバがブラウザに保存するように指示する値であるため、サーバがユニークな値を発行し、発行した値を記録しておくことができる。そのため、他のユーザと区別することができる上、ユーザが、サーバから指示された値とは異なる Cookie を送信しても、詐称したことがわかる。よって、Cookie は、ブラウザがサーバに送信する情報の中でも、個人を特定できる可能性が高いことがわかる。

2.4 SNS による個人の特定

近年、多くのインターネット利用者が SNS を利用している。Twitter 社が公開した”Q2 2017 Letter to Shareholders”によれば、2017 年における Twitter の月間利用者数は 3 億 2800 万人である。また、Facebook 社が公開した”Facebook Q2 2017 Results”によれば、2017 年における Facebook の月間利用者数は約

20 億人である。SNS が発展した現代においては、世界中の人々が、SNS を通じてインターネットに自分の情報を発信しているといえる。

Twitter や Facebook などの SNS には、多くの個人情報が保存されている。そのため、サイバー犯罪の犯人が利用している SNS のアカウントを特定し、SNS を運営する企業に協力を依頼することで、犯人の身元を特定することは十分に可能であると考えられる。

最近の多くの Web サイト、とりわけニュースサイトやブログサイトでは、自身もしくは自社のサイトを読者に広めてもらうために、SNS のシェアボタンを設置していることが多い。SNS のシェアボタンのリソース (画像やスクリプトなど) を読み込むために、ブラウザは SNS のサイトに対してリクエストを送信するが、その際に、ユーザがその SNS にログインした状態であれば、SNS のサーバにセッション情報を含む Cookie を送信しているはずである。

したがって、SNS のシェアボタンを設置したサイトでサイバー犯罪が発生した際、犯人がその SNS にログインした状態であれば、犯人の SNS のアカウントを特定することができる可能性が高いと考えられる。Cookie が個人を特定する情報となり得ることについては前述したが、SNS の Cookie はサイバー犯罪の犯人を特定する上で非常に重要な情報であるといえる。

2.5 犯人の不注意による逮捕事例

SNS による特定は、犯人が SNS にログインしていることが前提である。用心深い犯人であれば、サイバー犯罪を行う前に SNS からログアウトしているか、別のブラウザを使用しているであろう。

しかし、すべてのサイバー犯罪の犯人が、完全に証拠を残さずに犯罪を行うわけではない。実際に、Tor を利用して児童ポルノを投稿した犯人が逮捕された事例がある。Tor を使用することで発信元を隠蔽することができるが、逮捕された犯人は、Tor の利用中に、不注意により通常のブラウザを使用したために発信元を特定されてしまった。このように、犯人の不注意によって逮捕されるケースもある。

SNS において、利用を中断する度にログアウトするユーザは少数であると考えられる。ログインした状態でも通常のインターネットの使用に支障がないためである。よって、SNS にログインした状態で、その SNS のシェアボタンが設置されたサイトにアクセスすることは十分に考えられる。犯人の不注意により SNS のアカウントを特定することは、犯人の特定に大きく寄与するといえる。

第3章 研究方法

3.1 研究目的

第2章で述べたように，IP アドレスによる特定は困難である．そのため，サイバー犯罪の犯人を特定する際は，IP アドレスだけでなく，それ以外の情報を元に特定する手法が必要である．本論文では，Cookie が個人を特定できる可能性が高いことに着目した．また，近年において SNS を利用するユーザが多く存在し，ログアウトする機会が少ないことから，サイバー犯罪において，犯人の SNS のアカウントを発見することが，犯人の特定に大きく寄与するのではないかと考えられる．最近では，SNS のシェアボタンを設置する Web サイトが多く見受けられる．SNS のシェアボタンを読み込む際に送信された Cookie を利用することで，サイバー犯罪の犯人を特定する手法を提案することを目的とする．

3.2 提案手法

SNS のシェアボタンを読み込む際に送信された Cookie から，個人をどの程度特定できるか実験するために，本研究では，匿名掲示板サイトと SNS サイトを用意した．

第4章 実装

4.1 匿名掲示板

本研究で使用した匿名掲示板は、インターネット上にフリーソフトウェアとして公開されている KENT-WEB の LIGHT BOARD(以下, LIGHT BOARD)である。この掲示板は、アカウント登録が不要で利用できる掲示板サイトである。名前の入力が必要となっているが、実名を入力する必要はなく、投稿毎に別の名前を入力することも可能であるため、匿名掲示板として利用することができる。

本研究では、LIGHT BOARD のソースプログラムをダウンロードし、改変したものを、自身のサーバで公開し実験を行った。実験で使用した匿名掲示板サイトを図 4.1 に示す。

図 4.1: 匿名掲示板サイト

掲示板への書込みの際に入力する項目は複数あるが、必須項目は書込みを行った人の名前、書込みのタイトル、本文、画像認証の4項目である。画像認証の項目には、入力欄の右側に画像として表示された数字を入力する。こ

これはスパム投稿を防止するためのものである。先述の通り、名前は実名を入力する必要はないため、個人情報は一切入力せずに投稿できるため、利用者から見ると匿名掲示板のように利用することができる。必須項目を入力して投稿ボタンを押下すると投稿内容がサーバに送信され、書込み一覧に投稿内容が表示される。

4.1.1 シェアボタンの設置

LIGHT BOARD にはシェアボタンは設置されていない。本研究では、SNS のシェアボタンを利用して実験を行うため、シェアボタンを設置した。設置した SNS のシェアボタンは、Twitter, Facebook, Mastodon の 3 種類である。

Mastodon のシステムは、匿名掲示板サイトと同様に、本研究の実験用に用意したサーバ上に置かれている。そのため、Mastodon のシェアボタンを読み込む際に送信された Cookie は、保存、解析することができる。送信された Cookie を保存し、その Cookie から Mastodon のユーザを特定するプログラムを作成した。詳細は後述する。

4.1.2 同一ユーザ特定機能の実装

LIGHT BOARD は、書込み時に名前の入力が必要であるが、同一ユーザが複数の書込みを行う際に、それぞれ異なる名前をつけることが可能である。そのため、複数の書込みが同一人物によるものかどうかを判断することは難しい。そこで、本研究の実験で使用する匿名掲示板には、ユーザを追跡するための Cookie を付与する機能を実装した。

初めて匿名掲示板にアクセスしてきたユーザには、サーバ側で生成したランダムな文字列を Cookie として付与する。2 回目以降にアクセスしてきたユーザ (ブラウザ) は、初めてアクセスした際に付与された Cookie をサーバに送信するため、以前にアクセスしたことがあることがサーバ側でわかる。したがって、複数の書込みを行ったユーザは、書込みを行う際に送信している Cookie が同じであるため、同一人物による書込みであることがわかる。

ユーザを追跡するための Cookie を付与する機能に該当する箇所をソースコード 4.1 に示す。

ソースコード 4.1: 同一ユーザ特定機能

```
1 use String::Random;
2
3 sub track {
4   my $host = $ENV{REMOTE_HOST};
5   my $addr = $ENV{REMOTE_ADDR};
6   my $time = time;
7   my ($min,$hour,$mday,$mon,$year,$wday) = (localtime($time))[1..6];
8   my @wk = ('Sun','Mon','Tue','Wed','Thu','Fri','Sat');
9   my $date = sprintf("%04d/%02d/%02d(%s) %02d:%02d",
```



```

10     $year+1900,$mon+1,$mday,$wk[$wday],$hour,$min);
11 my $cookie = $ENV{HTTP_COOKIE};
12
13 my %cookie;
14 foreach (split(/;/, $cookie)) {
15     my ($key, $val) = split(/=/);
16     $key =~ s/\s//g;
17     $cookie{$key} = $val;
18 }
19
20 # もしトラッキング用のクッキーがセットされていればファイルに保存する
21 # 保存されていなければ新しく発行してセットさせる
22 if ($cookie{tracking_id}) {
23     open(DAT, ">> $cf{trackingfile}") or error("open err: $cf{
24         trackingfile}");
25     my $new = "$date<>$cookie{tracking_id}<>$host<>$addr";
26     print DAT "$new\n";
27     close(DAT);
28 }
29 else {
30     my $random = String::Random->new();
31     my $tracking_id = $random->randregex("[a-zA-Z0-9]{64}");
32     print "Set-Cookie: $cf{tracking_id}=$tracking_id\n";
33 }

```

サブルーチン track は、トラッキング (特定) 用の Cookie が送信されなければ新たに Cookie を発行し、送信されていればその Cookie をファイルに保存する。発行する Cookie は、String::Random を使用して生成した乱数を値とする。String::Random を使用して生成する乱数は、大文字、小文字を区別した 64 文字の英数字であるため、他のユーザと重複する可能性は極めて低いと考えられる。よって、匿名掲示板への書込みとトラッキング用の Cookie を照らし合わせ、複数の書込みに対して同じ Cookie が送信されている場合、同一人物による書込みであると判断できる。

4.2 SNS

本研究では、SNS のアカウントを特定する実験を行うためのシステムとして、Mastodon を利用した。本来のサイバー犯罪においては、Twitter や Facebook などの、世界的に利用されている SNS の運営と協力して犯人のアカウントを特定することを想定しているが、本研究の実験では、Twitter や Facebook のサーバ内の情報を閲覧することはできない。そこで、個人で運営することができる、Twitter に似た SNS である Mastodon を利用することで、シェアボタンを読み込む際に送信される Cookie から Mastodon のアカウントを特定する実験を行った。送信される Cookie からアカウントを特定することは、Twitter や Facebook においても技術的に可能であるため、実際のサイバー犯罪では Twitter や Facebook などの運営と協力することで、犯人のアカウントが特定できるといえる。

Mastodon

Mastodon とは、ドイツ人の Eugen Rochko 氏が、2016 年 2 月に開発を開始した、分散型 SNS のことである。Twitter に似た機能を持つ SNS として日本でも注目を集めている。オープンソースソフトウェアとして GitHub 上に公開されており、誰でも自由にプログラムを利用することができる。

Twitter や Facebook とは異なり、Mastodon はプログラムを改変することができ、また改変したプログラムをサーバ上に置いて運営することもできる。Mastodon は分散型 SNS であり、個人や団体、企業などが運営するため、複数のサーバが存在する。Mastodon が運営されている各々のサーバのことを、Mastodon のインスタンスと呼ぶ。GitHub からダウンロードした Mastodon のプログラムに、本研究の実験に必要な機能を実装して利用した。

4.2.1 シェアボタンの実装

匿名掲示板に設置するシェアボタンの実装を行った。/button という URL にアクセスすると、Mastodon のシェアボタンの画像だけが表示されるページを作成した。匿名掲示板では、HTML の iframe タグを利用し、Mastodon のシェアボタンのページを読み込んでいる。

ユーザ (ブラウザ) は、匿名掲示板にアクセスした際に、Mastodon のシェアボタンを読み込むために Mastodon のサーバにアクセスする。Mastodon のサーバにアクセスする際に、もしユーザが Mastodon にログインしている状態であれば、そのログイン状態を保持する Cookie を Mastodon のサーバに送信するため、匿名掲示板にアクセスしたユーザと Mastodon にログインしているユーザを結びつけることができる。

4.2.2 送信された Cookie を保存する機能の実装

匿名掲示板に設置されたシェアボタンを通して Mastodon のサーバに送信された Cookie をデータベースに保存する機能を実装した。Mastodon のシェアボタンが読み込まれた際に、送信元の情報を取得し、データベースに保存している。保存する送信元の情報を表 4.1 に示す。

表 4.1: 保存した送信元のデータ

データ	データ型	データ例
IP アドレス	string	133.19.169.6
Cookie	text	remember_user_token=ABCD1234; _session_id=EFGH5678; _mastodon_session=IJKL9012
リファラ	string	https://www.google.co.jp/
ユーザエージェント	string	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36

取得したデータは、IP アドレス、Cookie、リファラ、ユーザエージェントの 4 つである。データ型は、データベースに保存する際の型を意味する。データ例は、実際に保存されるデータの例を示している。

データベースには、Cookie の他に送信元の IP アドレスやリファラ、ユーザエージェントを保存している。本研究においては Cookie のみの保存で十分であるが、Cookie が取得できなかった場合や、より詳細な情報を得るために、Cookie 以外の情報も保存することにした。

リファラ

リファラとは、どのページからアクセスしたかを示す情報である。例えば、Google の検索結果からリンクをクリックしてサイト A にアクセスした場合、ブラウザがサイト A に対して送信するリファラは `https://www.google.co.jp/` である。リンクをクリックしてページを遷移した場合に限らず、iframe を使用して別のサイトを読み込んだ場合にもリファラは送信される。匿名掲示板の HTML 内に含まれる iframe タグから Mastodon のページを読み込んだ場合、Mastodon のサーバに送信されるリファラは、匿名掲示板の URL である。このように、リファラはどのページを経由してアクセスしてきたかがサーバ側にわかるため、Web サイトの解析ツールなどでしばしば用いられる。

ユーザエージェント

ユーザエージェントとは、ブラウザがサーバに対して送信する情報であり、使用しているブラウザや OS、またその種類などの情報が含まれている。例えば表 4.1 のデータ例の場合、アクセスしてきたユーザは、macOS 10.12.6

を使用しており、Google Chrome 62.0.3202.94 でアクセスしていることがわかる。使用しているブラウザごとに Web ページの表示を変えるなど、ユーザビリティの向上を目的に使用されることもあるが、サイバー犯罪においては、犯人の使用している OS やブラウザがわかるといった利点がある。ただし、ユーザエージェントは詐称（本来とは異なる情報をサーバ側に送信）することも可能であるため、必ずしもユーザエージェントと同じ OS やブラウザを使用しているとは限らない。

4.2.3 ブラウザフィンガープリンティングの実装

Cookie やリファラ、ユーザエージェントなどはブラウザからサーバに送信される情報であるが、JavaScript を使用することで、通常はサーバに送信されない情報を取得することができる。

JavaScript はブラウザ上で実行されるスクリプト言語であり、Web ページの遷移を行うことなく HTML の構成要素を書き換えたり、非同期でサーバと通信したりすることができる。近年ではほとんどの Web サイトに JavaScript が使用されている。また、JavaScript を使用して、ブラウザやデバイスの情報を取得することができる。このように、Cookie の代わりに JavaScript を利用して、ユーザが使用するデバイスを調べて同一ユーザを特定する仕組みをブラウザフィンガープリンティングという。

ブラウザフィンガープリンティングには、4.2.2 項で説明したリファラやユーザエージェントなども含まれる。本研究の実験では、通常はサーバに送信されない情報である画面サイズを、JavaScript を使用して取得することにした。JavaScript を使用しないと取得できない情報を収集することで、より精度の高い特定手法を提案できると考えられるためである。

サーバ側で、ブラウザから送信された画面サイズの情報を受け取るための API エンドポイントを作成した。API エンドポイントとは、API にアクセスする際の URL のことである。本研究では、`/api/v1/fingerprint` というエンドポイントを作成し、このエンドポイントに対して HTTP の POST リクエストが送信された場合に、送信された情報をデータベースに保存する機能を実装した。

JavaScript では、ユーザが使用するデバイスの画面サイズを取得し、サーバに送信するプログラムを実装した。該当する JavaScript のプログラムをソースコード 4.2 に示す。

ソースコード 4.2: 画面サイズの取得とサーバへの送信

```
1 var xhr = new XMLHttpRequest();
2 xhr.open('POST', '/api/v1/fingerprint', true);
3 xhr.setRequestHeader('content-type', 'application/x-www-form-
  urlencoded');
4 xhr.send('screen_size=' + screen.width + 'x' + screen.height);
```

JavaScript でサーバにリクエストを送信する際には、XMLHttpRequest() を使用する。新しく生成した XMLHttpRequest() のオブジェクトを変数 xhr に格納する。xhr.open() を使用して、HTTP メソッド (GET や POST など) やリクエストを送信する URL(エンドポイント) を指定する。必要であれば xhr.setRequestHeader() を使用してリクエストヘッダを設定することもできる。最後に xhr.send() を使用して、サーバに送信するデータを指定し、送信する。JavaScript でデバイスの画面サイズを取得するには screen.width(画面横幅) と screen.height(画面縦幅) を使用する。例えば screen.width で取得した値が 1440, screen.height で取得した値が 900 であった場合、'screen.size=1440x900' という文字列をサーバに送信する。この情報を受け取ったサーバは、データベースにこのデータを文字列として保存する。

4.3 実験用ツール

匿名掲示板と SNS の他に、本研究の実験に当たって必要となるツールの実装を行った。

4.3.1 UNIX タイム変換ツール

匿名掲示板サイトへの書込みは、ファイルに保存するように実装されている。投稿された書込みを保存したファイルの一部を以下に示す。

```
5<>2017/12/13(Wed) 18:01<>baz<><>piyo<>ううう<><>133.19.169.6<><>1513155668<>
4<>2017/12/13(Wed) 17:59<>bar<><>huga<>いいい<><>133.19.169.6<><>1513155569<>
3<>2017/12/13(Wed) 17:51<>foo<><>hoge<>あああ<><>133.19.169.6<><>1513155083<>
2<>2017/12/13(Wed) 17:51<>xxx<><>xxx<>test<><>133.19.169.6<><>1513155081<>
1<>2017/12/13(Wed) 17:50<>なまえ<><>無題<>テスト<><>133.19.169.6<><>1513155034<>
```

投稿された各々の書込みは行ごとに保存されており、書込み内容に関する情報を、<>で区切って管理されている。保存されている情報は、左から順に、書込み番号 (最初の書込みを 1 とした連番)、投稿時刻、名前、書込みのタイトル、書込みの本文、IP アドレス、UNIX タイムスタンプである。

本研究の実験では、匿名掲示板への書込み時刻と SNS のシェアボタンを読み込む際に送信された Cookie の保存時刻を照合して、書込みを行ったユーザの SNS のアカウントを特定している。書込み内容を管理するファイルには、書込み時刻が保存されているが、秒数までは記録されていない。実際に実験を行ったところ、秒単位でのアクセスが集中したため、書込み時刻からは、Cookie の保存時刻と照合することができなかった。そこで、書込み時刻の代わりに UNIX タイムスタンプを確認することで、Cookie の保存時刻との照合を行った。本研究では、UNIX タイムスタンプを、秒数を含めた時刻に

変換するプログラムを Perl で実装した．そのプログラムをソースコード 4.3 に示す．

ソースコード 4.3: UNIX タイムスタンプ変換プログラム

```
1 print "localtime: ";
2 my $num = <STDIN>;
3 my $time = $num;
4 my ($sec,$min,$hour,$mday,$mon,$year) = (localtime($time))[0..5];
5 my $date = sprintf("%04d-%02d-%02d %02d:%02d:%02d",$year+1900,
    $mon+1,$mday,$hour,$min,$sec);
6 print "timestamp: " . $date . "\n"
```

標準入力から UNIX タイムスタンプを受け取り，時刻に変換して出力するプログラムである．`localtime` 関数は引数として受け取った UNIX タイムスタンプを 9 つの要素の配列に変換する関数である [1]．ソースコード 4.3 では，`localtime` 関数を使用して，入力された UNIX タイムスタンプから，秒，分，時，日，月，年を取得し，それぞれ変数に格納している．`sprintf` 関数を使用して表示する時刻のフォーマットを指定した後，標準出力を行っている．

UNIX タイムスタンプ

UNIX タイムスタンプとは，1970 年 1 月 1 日 0 時 0 分 0 秒を 0 として，その時刻からの経過秒数を数値で表したものである．たとえば，UNIX タイムスタンプが 1513155034 であった場合，1970 年 1 月 1 日 0 時 0 分 0 秒からカウントして 1513155034 秒が経過した時点での時刻 (2017 年 12 月 13 日 17 時 50 分 34 秒) を表す．ファイルに保存された書込み時刻には秒数が記録されていないが，UNIX タイムスタンプから時刻に変換することで，書き込みがあった時刻の秒数を求めることができる．

4.3.2 Rails セッションデコーダ

本研究の実験に使用する SNS である Mastodon は，Ruby on Rails(以下，Rails) と呼ばれる Web アプリケーションフレームワークを用いて実装されている．Web アプリケーションフレームワークとは，Web アプリケーションにおける一般的な機能を提供する雛形のことである．一般的な Web アプリケーションでは，Web サイトのフォームから入力された情報をデータベースに保存したり，データベースから取得した内容を表示したり，データベースに存在するデータを更新・削除したりする機能がある．それらの機能を，Web アプリケーションの開発者が 1 から実装することは，開発コストがかかるだけでなく，セキュリティに十分な注意を払わなければ脆弱性を生み出す危険性がある．Web アプリケーションフレームワークを利用すれば，少ない開発コストで，それらの機能を持った Web アプリケーションを構築することができる．また，多くの Web アプリケーションフレームワークは，オープンソース

ソフトウェアとしてインターネット上に公開されているため、Web の脆弱性に対する対策が行われている。

ブラウザが Web サービスにログインしている情報は、通常、Cookie に保存されている。ログインに成立すると、サーバはブラウザに、ログイン状態を保持するセッション情報を Cookie として保存するように指示する。以降、セッション情報が含まれる Cookie をブラウザがサーバに送信することで、サーバは、アクセスしてきたユーザがログイン済みであると認識する。

したがって、SNS のシステムにおいて、ブラウザから送信された Cookie を調べることで、その Cookie を送信してきたユーザの SNS のアカウントを特定することができる。4.2 節で述べたように、本研究で使用する Mastodon に送信された Cookie はデータベースに保存される。ところが、Rails 4.0 以降では、セッション情報が暗号化されて Cookie に保存されている [2] ため、そのままでは Mastodon のアカウントを特定することができない。そこで、データベースに保存された Cookie の中に含まれるセッション情報を複合するプログラムを Node.js で実装した。そのプログラムをソースコード 4.4 に示す。

ソースコード 4.4: Rails セッションデコーダ

```
1 const rsd = require('rails-session-decoder');
2 const read = require('readline-sync');
3 const fs = require('fs');
4
5 var cookie = read.question('Input your cookie: ');
6 var key = read.question('Input your secret key: ');
7
8 console.log('');
9
10 var decoder = rsd(key);
11
12 decoder.decodeCookie(cookie, (err, data) => {
13   if (data) {
14     console.log(data);
15   }
16   else {
17     console.log(err);
18   }
19 });
```

暗号化されたセッション情報を復号するには、セッション情報を含む Cookie と Rails(Mastodon) システム内で管理されている秘密鍵が必要である。セッション情報を含む Cookie はデータベースに保存されているものを使用し、秘密鍵は Rails(Mastodon) の設定ファイルに書かれているものを使用する。

ソースコード 4.4 は、標準入力から、暗号化されたセッション情報を含む Cookie と、秘密鍵を受け取り、暗号化されたセッション情報の復号を行う。正しく復号できた場合は復号されたセッション情報を出力し、正しく復号できなかった場合はエラーを出力する。

暗号化されたセッション情報の復号には、rails-session-decoder という Node.js のパッケージ (ライブラリ) を利用した [3]。rails-session-decoder を読み取っ

たオブジェクトに対して、暗号化されたセッション情報を含む Cookie と秘密鍵を渡して decodeCookie メソッドを呼び出すと、暗号化されたセッション情報が復号される。ユーザ (ブラウザ) から送信された Cookie がセッション情報を含んでいれば、復号に成功する。

第5章 実験

5.1 実験手順

本研究では，SNS のシェアボタンを読み込む際に送信される Cookie から，SNS のアカウントを特定する実験を行った．本実験の目的は，Web サイトに設置されている SNS シェアボタンを利用した，SNS のアカウントの特定可能性を検証するためである．本実験では，Twitter や Facebook に代わる SNS として，Mastodon を利用した．


本実験を被験者として，立命館大学情報理工学部サイバーセキュリティ研究室に所属する学部生 15 名に協力を仰いだ．被験者 15 名には，本研究で用意した匿名掲示板と SNS サイトに対して，以下の事柄を実行してもらった．

1. Mastodon アカウントの作成
2. メールアドレスの確認と Mastodon へのログイン
3. 匿名掲示板へのアクセス
4. 匿名掲示板への書込み

5.1.1 Mastodon アカウントの作成

通常，Mastodon のインスタンスをインターネット上に公開した場合，誰でもそのインスタンスにアカウントを作成することができる．ただし，Mastodon v2.1.0 以降では，招待システムが導入され，招待用のリンクを知っている者のみがアカウントを登録することができる [4]．外部の第三者（本実験の被験者ではない者）が，本研究で用意した Mastodon インスタンスにアカウントを登録できる状態になっていた場合，本実験で得られるデータの収集の妨げになると判断したため，本研究の Mastodon インスタンスでは，招待リンクを知っている者のみがアカウントを登録できるように設定した．事前に招待用のリンクを被験者に配布した．

匿名掲示板へ書込みを行った被験者の Mastodon アカウントを特定するため，被験者には Mastodon のアカウントを作成してもらった．アカウント登録ページを図 5.1 に示す．



ユーザー名

メールアドレス

パスワード

パスワード（確認用）

登録する

登録すると [利用規約](#) と [プライバシーポリシー](#) に同意したことになります。

[ログイン](#) 確認メールを受信できませんか？

図 5.1: Mastodon アカウント登録ページ

一般的な Web サービスのアカウント登録ページである。被験者は、このフォームにユーザー名 (任意の文字列)、メールアドレス、パスワードを入力して、アカウントを作成する。

5.1.2 メールアドレスの確認と Mastodon へのログイン

アカウントを登録すると、登録時に入力したメールアドレス宛に確認メールが届く。届いたメールに記載されているリンク先にアクセスするとメールアドレスが確認される。登録したアカウントのメールアドレスの有効性が確認できるまで（確認メールのリンク先にアクセスするまで）は、アカウントにログインすることができない。

被験者は、メールアドレスの確認を行った後、ログインページにアクセスしてログインする。

5.1.3 匿名掲示板へのアクセス

被験者は、Mastodon にログインした状態で、匿名掲示板にアクセスする。匿名掲示板も Mastodon と同様に、外部から利用されることを防ぐため、アクセス用のパスワードを設置した。アクセス用のパスワードは被験者のみに共有し、第三者から勝手に書き込みが行われないようにした。パスワードを入力すると、4.1 節で示した図 4.1 のページが表示される。なお、図 4.1 のページには SNS(Mastodon) のシェアボタンが設置されているため、Mastodon のアカウントにログインした状態でこのページにアクセスした時点で、Mastodon インスタンスにログイン状態を保持する Cookie が送信されている。

5.1.4 匿名掲示板への書き込み

最後に、被験者は、図 4.1 のページから適当な書き込みを行う。書き込む際の名前や書き込み内容などは、各々の被験者に一任した。また、同一人物が異なる名前で複数の書き込みを行っても同一人物による書き込みを特定できることを証明するために、一部の被験者には異なる名前で複数の書き込みを行ってもらった。

5.1.5 匿名掲示板への投稿時の注意点

匿名掲示板への投稿時に以下の事柄に注意して投稿するよう被験者に説明した。

- Mastodon アカウントにログインした状態で投稿すること
- ログインした端末・ブラウザで投稿すること

Mastodon アカウントにログインした状態で書き込みを行わなければ、ログイン状態を保持する Cookie が送信されないため、アカウントを特定すること

ができない。そのため、被験者には、Mastodon アカウントを作成するだけでなく、作成したアカウントにログインした状態で書込みをするよう指示した。

また、Mastodon アカウントにログインした状態であっても、ログインしたものとは異なる端末・ブラウザを使用して書込みを行った場合も同様に、ログイン状態を保持する Cookie が送信されない。そのため、ログインしたものと同一端末・ブラウザを使用するよう指示した。また、ログインしたものと同一ブラウザを使用したとしても、ブラウザのシークレットモード (プライベートブラウジングモード) を使用して書込みを行わないよう指示した。シークレットモードでは、閲覧履歴や Cookie がリセットされた状態でブラウジングを行うようになるためである。ただし、シークレットモードでも一度アカウントにログインし直した場合はこの限りではない。

5.2 実験内容

匿名掲示板に投稿された書込みすべてに対して、書込みを行ったユーザの Mastodon のアカウントを特定した。アカウントを特定するまでの手順を以下に示す。

1. 書込みと Cookie の紐付け
2. Cookie に含まれるセッション情報の復号
3. アカウント名の検索

5.2.1 書込みと Cookie の紐付け

4.3.1 項で示したように、匿名掲示板に投稿された書込みの書込み時刻を UNIX タイムスタンプから算出する。書込みが投稿された直後に SNS のシェアボタンが読み込まれ Cookie が送信されるはずである。したがって、書込みの直後に送信された Cookie を調べればよい。

匿名掲示板へ書き込まれた時刻が 2017 年 12 月 13 日 17 時 51 分 21 秒であったとする。その直後に送信された Cookie を Mastodon のデータベースから検索する。その際に実行した SQL をソースコード 5.1 に示す。

ソースコード 5.1: 2017 年 12 月 13 日 17 時 50 分 34 秒にあった書込みに対応する Cookie を検索する SQL

```
1 SELECT * FROM buttons WHERE created_at BETWEEN
   '2017-12-13 08:51:00' AND '2017-12-13 08:52:00' ORDER BY
   created_at;
```

挿入された日時が、2017 年 12 月 13 日 17 時 51 分 00 秒から 2017 年 12 月 13 日 17 時 52 分 00 秒までの期間であるレコードを、buttons テーブルから検索

し、挿入された日時順に並び替えて表示する SQL である。ただし、Mastodon で使用するデータベースでは、時刻が日本標準時ではなくグリニッジ標準時として保存されてしまっていたため、実際の時刻より 9 時間前の時刻で検索している。

buttons テーブルには、Mastodon のシェアボタンを読み込み際に送信された Cookie が保存されている。保存されているデータは表 4.1 の通りである。Rails で扱うデータベースには、すべてのテーブルにレコードの作成日時と更新日時が自動で追加される。ソースコード 5.1 で使用している created_at はレコードの作成日時に該当する。検索結果は、指定した期間内のレコードの作成日時順に、該当するレコードを表示している。つまり、書込みがあった時刻に近い範囲の時刻で送信されている Cookie をすべて表示する SQL となっている。

表示された複数のレコードのうち、書込み時刻よりも後であり、最も書込み時刻に近いレコードを調べる。そのレコードに保存されている IP アドレスやユーザエージェントが、書込みのものと一致していれば、そのレコードが書込みに対応するものであると判断する。なお、匿名掲示板側の IP アドレスは匿名掲示板への書込みが保存されたファイル内に保存されているもの、ユーザエージェントは Web サーバのログに保存されているものを利用した。

5.2.2 Cookie に含まれるセッション情報の復号

書込み情報と一致したレコードから Cookie を調べる。例として、2017 年 12 月 13 日 17 時 51 分 21 秒に投稿された書込みと一致するレコードの Cookie は以下の通りである。

```
_mastodon_session=TzhBbE5SWVBjcXBmakgzQlZFclpqRExDMFB3WXJiSFdk  
emQ4VkI3TThaNUlENmJGeGp00VR4azJSVVVZUFVjQVlKL1pack1VTzRNdkpvcF  
h4c2o1czFPVkl1R3dpdHJpZ3AzboVneEhZUVp2d1ZEbDE0MnJEZ2srBEk1L2ZJ  
Q2ZPc1ZWU09DL0VDcVJhNG15aTZmWTZQMdhDc2pKS2VUdWdITmR1TWfQrRGZ0dW  
hoc0gxeFhaYU5GWTlMY1BvZ29uK1V2RE1KSnfFaU2Vvb28wS2dFVHBDS3ZXRXWhR  
S0FaVlhZQnM2NUJ4ZjNkTEdaNUVFek9pWGxyaUtZN01kWDBFTzBxa1FjTkdXU2  
5IaklqVlUzeUs2MGVjV1hoNm8xdHFWWkxWcnhjeDU1UDdJNXVkJEpINUVUUmJT  
ekdTskljadhyT2dGM092ZVprbGcrS0pLR0JUZjg1d1pSNG0zNmRDY21HVnB2WT  
hYMy9ScC8rWk8zWjBIenZ3bmJJTGJhRDRRbnJpQUt1QXNTRHFDTDNQQjUyVmF6  
UT09LS1JTG00Wlc4b0g5Vk9rZWJwUlpqUT1BPT0%3D--7c1f476c80f45229f4  
f5b5f18d260e266f8f5930; _session_id=IjA5ZGMxYmQ5ZDIzOTF1NzJjMz  
dlNDFmYzh1MGUyNWQxIg%3D%3D--f749083440458553e2dfe0b45f1a226e6d  
5407e5; remember_user_token=W1szXSwiJDJhJDEwJDBtekVCUXo0cFR4bT  
hCNVI5dDFhT2UiLCIxNTEzMTU0NzkwLjgxMDU4MTciXQ%3D%3D--0e2f0c3b4c  
71e16a9828e0d0424cfe7545ba92df
```

Cookie は、名前と値のペアをイコール記号で結んだ形式となっている。左辺が名前で、右辺が値である。複数のペアを一つの Cookie にまとめる場合はセミコロンで区切る。

上記 Cookie は、`_mastodon_session` と `_session_id` と `remember_user_token` の 3 つで構成されている。ログイン状態のユーザ (ブラウザ) が Mastodon に送信する Cookie はこの 3 つの名前を持つ値であり、このうち、ログイン状態を保持するセッション情報を持つ Cookie は `_mastodon_session` である。

`_mastodon_session` を、4.3.2 項で説明した Rails セッションデコーダを使用して復号すると、以下の情報が得られる。

```
{
  "session_id": "fb177bece158591f43126c67987159f1",
  "warden.user.user.key": [[3], "$2a$10$mzEBQz4pTxm8B5R9t1a0e"],
  "user_return_to": "/button?text=%e6%8e%b2%e7%a4%ba%e6%9d%bf%e3%81%8b%e3%82%89%e3%81%93%e3%82%93%e3%81%ab%e3%81%a1%e3%81%af%ef%bc%81",
  "_csrf_token": "WgC1EbiMg3YV1VgqJNwCNkgw6ldgRwS1Qkz3ckUm/SM="
}
```

得られた情報が、Mastodon のシェアボタンを読み込む際に送信された Cookie から復号されたセッション情報である。このセッション情報のうち、`warden.user.user.key` 中の 3 という数値が、Cookie を送信してきたユーザのユーザ ID である。

5.2.3 アカウント名の検索

得られたユーザ ID をもとに Mastodon のデータベースで検索を行う。ユーザ ID が 3 であるユーザを検索する SQL をソースコード 5.2 に示す。

ソースコード 5.2: ユーザ ID が 3 であるユーザを検索する SQL

```
1 SELECT * FROM users WHERE id = 3;
```

ユーザに関する情報は `users` テーブルに保存されている。ここには、アカウント登録時に入力したメールアドレスや暗号化されたパスワードなどのユーザ情報が保存されている。さらに、メールアドレスや暗号化されたパスワードとともにアカウント ID が保存されている。ユーザ ID が 3 であるユーザのアカウント ID は 3 であることがわかった。この結果をもとにアカウント ID が 3 であるアカウントを検索する SQL をソースコード 5.3 に示す。

ソースコード 5.3: アカウント ID が 3 であるアカウントを検索する SQL

```
1 SELECT * FROM accounts WHERE id = 3;
```

アカウントに関する情報は accounts テーブルに保存されている。accounts テーブルには、アカウント登録時のユーザ名が保存されている。5.2 に示す一連の作業を実行すると、結果として、匿名掲示板に書込みを投稿した者の Mastodon のユーザ名を特定することができる。ユーザ名を特定することで、そのアカウントに登録されているメールアドレスなどを特定することができる。実際のサイバー犯罪でこの手法を活用する場合、犯人の Twitter や Facebook などのアカウントを特定することで、本実験以上に犯人を特定する情報が得られることが期待できる。

5.3 実験結果

本実験の被験者 15 名に本実験に協力してもらったところ、計 32 件 (自身による実験概要の説明のための書込みを除く) の書込みがあった。32 件の書込みすべてに対して、5.2 節で説明した手順を実行してアカウントの特定を行った。その結果を表 5.1 に示す。

表 5.1: 実験結果

番号	IP アドレス	Cookie ID	アカウント ID	ユーザ名	書込み時刻	Cookie 送信時刻
2	xxx.xxx.xxx.xxx	33	特定不可	特定不可	2017-12-13 17:50:34	2017-12-13 08:50:37
3	133.19.169.6	35	3	yo	2017-12-13 17:51:21	2017-12-13 08:51:23
4	133.19.169.6	37	7	hoge	2017-12-13 17:51:23	2017-12-13 08:51:29
5	133.19.169.6	49	特定不可	特定不可	2017-12-13 17:59:29	2017-12-13 08:59:36
6	133.19.169.6	55	4	eatarl_02	2017-12-13 18:01:08	2017-12-13 09:01:11
7	xxx.xxx.xxx.xxx	57	2	mahiro	2017-12-13 18:01:23	2017-12-13 09:01:26
8	133.19.169.6	59	10	Yuri	2017-12-13 18:01:27	2017-12-13 09:01:30
9	xxx.xxx.xxx.xxx	62	15	tetsutalow	2017-12-13 18:01:35	2017-12-13 09:01:41
10	133.19.169.6	64	12	ryota	2017-12-13 18:01:46	2017-12-13 09:01:49
11	133.19.169.6	75	16	rider555	2017-12-13 18:02:44	2017-12-13 09:02:46
12	xxx.xxx.xxx.xxx	77	13	tomo	2017-12-13 18:03:01	2017-12-13 09:03:04
13	xxx.xxx.xxx.xxx	78	9	miyamo	2017-12-13 18:03:10	2017-12-13 09:03:14
14	xxx.xxx.xxx.xxx	81	8	kazkaz144	2017-12-13 18:03:15	2017-12-13 09:03:23
15	xxx.xxx.xxx.xxx	86	5	wangyuncong	2017-12-13 18:03:31	2017-12-13 09:03:44
16	xxx.xxx.xxx.xxx	93	15	tetsutalow	2017-12-13 18:04:06	2017-12-13 09:04:09
17	xxx.xxx.xxx.xxx	92	11	bbbbbb	2017-12-13 18:04:06	2017-12-13 09:04:08
18	133.19.169.6	104	特定不可	特定不可	2017-12-13 18:04:24	2017-12-13 09:04:42
19	133.19.169.6	100	4	eatarl_02	2017-12-13 18:04:27	2017-12-13 09:04:29
20	133.19.169.6	108	7	hoge	2017-12-13 18:05:11	2017-12-13 09:05:13
21	133.19.169.6	109	3	yo	2017-12-13 18:05:15	2017-12-13 09:05:17
22	133.19.169.6	116	12	ryota	2017-12-13 18:05:29	2017-12-13 09:05:31
23	133.19.169.6	120	特定不可	特定不可	2017-12-13 18:05:33	2017-12-13 09:05:35
24	xxx.xxx.xxx.xxx	122	2	mahiro	2017-12-13 18:05:44	2017-12-13 09:05:46
25	133.19.169.6	132	特定不可	特定不可	2017-12-13 18:06:47	2017-12-13 09:06:49
26	xxx.xxx.xxx.xxx	134	9	miyamo	2017-12-13 18:07:01	2017-12-13 09:07:04
27	xxx.xxx.xxx.xxx	136	9	miyamo	2017-12-13 18:07:54	2017-12-13 09:07:56
28	133.19.169.6	138	特定不可	特定不可	2017-12-13 18:08:14	2017-12-13 09:08:16
29	133.19.169.6	144	12	ryota	2017-12-13 18:09:10	2017-12-13 09:09:12
30	133.19.169.6	145	7	hoge	2017-12-13 18:09:11	2017-12-13 09:09:14
31	xxx.xxx.xxx.xxx	146	15	tetsutalow	2017-12-13 18:09:22	2017-12-13 09:09:25
32	133.19.169.6	169	7	hoge	2017-12-13 18:13:48	2017-12-13 09:13:50
34	xxx.xxx.xxx.xxx	214	12	ryota	2017-12-16 16:18:02	2017-12-16 07:18:05

参考文献

- [1] Japan Perl Association, Perl の組み込み関数 localtime の翻訳
<<http://perldoc.jp/func/localtime>> (最終閲覧日: 2018 年 1 月 13 日)
- [2] Rails のセッション管理方法について
<<http://shindolog.hatenablog.com/entry/2014/11/02/164118>>
(最終閲覧日: 2018 年 1 月 13 日)
- [3] rails-session-decoder <<https://www.npmjs.com/package/rails-session-decoder>>
(最終閲覧日: 2018 年 1 月 13 日)
- [4] Release v2.1.0 · tootsuite/mastodon
<<https://github.com/tootsuite/mastodon/releases/tag/v2.1.0>>
(最終閲覧日: 2018 年 1 月 14 日)