Vol.2022-IOT-58 No.5
2022/7/12

ブラウザの拡張機能を用いた脆弱なOAuth2.0実装の検知

国広真吾[1,a)]　鄭 俊俊[2,b)]　猪俣 敦夫[3,c)]　上原 哲太郎[2,d)]

概要：OAuth2.0 を用いてユーザ認証の統合を行う Web アプリケーションが広く普及している．OAuth2.0 にはクロスサイトリクエストフォージェリ (以下 CSRF) 攻撃等に対する脆弱性が存在しており，開発者が Web アプリケーションに OAuth2.0 を実装する際に，URL に state パラメータを付与する等の対策をすることが必要とされている．しかし，CSRF 攻撃等に脆弱であるまま OAuth2.0 を実装している Web アプリケーションが複数確認されている．本研究では，CSRF 攻撃等に脆弱な OAuth2.0 の実装をしている Web アプリケーションを検知し，ユーザへ知らせる事で CSRF 攻撃等の被害を未然に防ぐ事を目的とし，ブラウザの拡張機能を用いて検知する手法を提案した．結果，ブラウザの拡張機能を用いることで，CSRF 攻撃への対策が不十分なまま OAuth2.0 実装をしている Web アプリケーションを検知することが可能であった．

キーワード：Web アプリケーション，OAuth2.0，CSRF，ブラウザ拡張

Detection of vulnerable OAuth2.0 implementations by browser extensions

KUNIHIRO Shingo[1,a)]　ZHENG Junjun[2,b)]　INOMATA Atsuo[3,c)]　UEHARA Tetsutarou[2,d)]

Abstract: OAuth2.0 is widely used in Web applications that realize integrated user authentication. But OAuth 2.0 is vulnerable to cross-site request forgery (CSRF) attacks and developers are asked to take adequate countermeasures such as adding a state parameter to the redirect URL in their web applications. It has been confirmed that some OAuth 2.0 implementations are vulnerable to CSRF attacks. In this study, we proposed a detection method using the browser extension with the aim of preventing damage from CSRF attacks by detecting web applications of which OAuth 2.0 implementation is vulnerable to the CSRF attack and notify users of the vulnerability. As a result, it was possible to detect web applications that implement OAuth2.0 without sufficient countermeasures against CSRF attacks by using the browser extension.

1. はじめに

ユーザが Web アプリケーションを利用する際に，ユーザ認証を行う場面は多い．複数の Web アプリケーションで ID とパスワードを用いた認証を行うと，パスワード管理が煩雑になるだけではなく，使用頻度の低い Web アプリケーションでの ID とパスワードが漏えいした際に気づきにくい等のセキュリティ上の問題が指摘されている．そこで，複数の Web アプリケーションにアクセスする際に単一の ID とパスワードで認証を行う統合認証が提案されている．統合認証には，一度のユーザ認証で複数の Web アプリケーションにおけるログインを可能にするシングルサインオン (SSO) がある．SSO には，OAuth や OpenID connect, SAML(Security Assertion Markup Language) 等の技術が用いられている．中でも OAuth は SNS を用いた SSO を実装する際に広く用いられている．しかし OAuth はクロスサイトリクエストフォージェリ攻撃に対して脆弱であることが指摘されている [1]．そのため，RFC6819[2] では実装者が state パラメータを付与することで認可リク

[1] 立命館大学 大学院 情報理工学研究科
[2] 立命館大学 情報理工学部
[3] 立命館大学総合科学技術研究機構・大阪大学
[a)] kunihiro@cysec.cs.ritsumei.ac.jp
[b)] jzheng@cysec.cs.ritsumei.ac.jp
[c)] inomata.atsuo.cysec@osaka-u.ac.jp
[d)] t-uehara@fc.ritsumei.ac.jp

エストとリダイレクトレスポンスを紐づけることや，クライアントの開発者とエンドユーザに信頼出来ない URL にアクセスすべきではないと周知させること等の対策が提案されている．しかし，菊田らの調査 [4][5] では，OAuth2.0 が CSRF 攻撃に対して脆弱なまま実装されている Web アプリケーションが複数確認されている．そこで本研究では，ブラウザの拡張機能を用いて脆弱な OAuth2.0 の実装を検知する手法を提案する．本研究の目的は，ユーザが Web アプリケーションを利用する際に，その Web アプリケーションで OAuth2.0 を CSRF 攻撃への対策をして実装しているかを判断し，ユーザに知らせることである．

2. 準備

2.1 統合認証

統合認証とは，複数のシステムにアクセスするための ID やパスワードを統一する仕組みである．統合認証を実現する方法の 1 つとして，1 つの ID とパスワードを用いていずれかのシステムにログインすることで，連携した他のサービスにログイン出来るようにする方法をシングルサインオン (SSO) という．Web アプリケーションにおいては SSO は，OAuth や OpenID connect，SAML(Security Assertion Markup Language) 等の技術によって実現されている．統合認証を用いることで，ユーザが記憶しておくべき ID とパスワードが 1 つで済むため負担が少なく，ユーザは 1 つのパスワードだけを記憶しておけば良いので複雑で強固なパスワードを設定できるメリットがある．しかし，ID とパスワードが漏えいした場合全てのサービスへログインが可能になるリスクがある．このリスクはパスワードの他にデバイス認証や生体認証を併せて用いる多要素認証を用いることで軽減が可能である．

2.2 OAuth2.0

OAuth は，ユーザが Web アプリケーションを利用する際にクライアントへ ID やパスワードを共有することなく，クライアントがユーザのリソースへアクセスすることを認可することができる認可フレームワークであり，SSO を実現するために広く使われている．ここでユーザとは，Web アプリケーションの利用者を指し，ここでは Web ブラウザを用いて Web アプリケーションを利用することを前提とする．またここでいうクライアントとは，OAuth 認証を実装した Web アプリケーションを指す．リソースとは図 1 のように，認証を行う際に要求される情報である．OAuth は，RFC5849[3] で規定された OAuth1.0 と RFC6749[1] で規定された OAuth2.0 があるが，RFC6749 で OAuth1.0 は廃止され OAuth2.0 を利用することが推奨されている．

2.2.1 OAuth2.0 を用いた認証の流れ

OAuth2.0 では，ユーザがクライアントにユーザのリソースへアクセスすることを認可したことを証明するランダムな文字列で構成されたアクセストークンを用いてクライアントのアクセス許可を管理する．OAuth2.0 におけるアクセス権限の移譲はユーザ，Web アプリケーションのクライアント，認可サーバ，リソースサーバの間で行われる．認可サーバは認証の結果に基づいて認可を行う．リソースサーバはアクセス権を管理しているサーバである．そのフローを図 2 に示す．

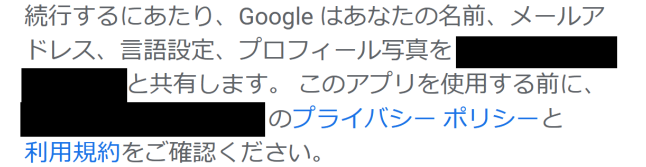

図 1 リソースの例

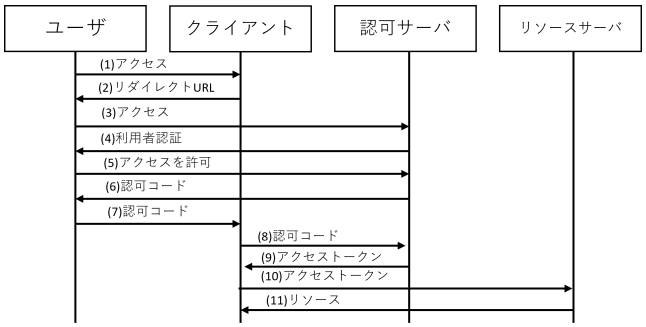

図 2 OAuth におけるアクセス権限移譲

(1) ユーザが認証を必要とするページ（クライアント）にアクセスする．
(2) クライアントがユーザに認証ページへのリダイレクト URL を送信する．
(3) ユーザがリダイレクト URL へアクセスする．
(4) 認可サーバがユーザの利用者認証をする．
(5) ユーザはクライアントがリソースにアクセスする事を許可する．
(6) 認可サーバは認可コードを生成し，クライアントへアクセスするためのリダイレクト URL に含ませてユーザに送信する．認可コードとは，クライアントのリソースへのアクセスをユーザが許可したことを証明するランダムな文字列である．(ここではセッション A と呼ぶ)
(7) ユーザが URL にアクセスをすることで，認可コードがクライアントへ渡される．
(8) クライアントは認可サーバに認可コードを送信し，アクセストークンを要求する．アクセストークンとは，リソースサーバにアクセスするためのランダムな文字列である．(ここではセッション B と呼ぶ)
(9) 認可サーバはランダムな文字列であるアクセストークンを発行し，クライアントに送信する．

(10)クライアントはリソースサーバにアクセストークンを送信する．
(11)リソースサーバはクライアントにユーザのリソースを送信する．

2.2.2 アクセストークンの発行

アクセストークンとは，クライアントがリソースサーバにアクセスする際に用いられるトークンである．認可サーバがアクセストークンを発行するフローは認可コードをリクエストする Authotization Code Grant とアクセストークンをリクエストする Implicit Grant の二つが存在し，クライアントはどちらのフローを選択するのかを response_type パラメータの値によって指定する．RFC6749[1] では，response_type の値は code，token，または登録された値のいずれかを指定することが必須とされている．response_type の値が code である場合は Authotization Code Grant のフローが選択され，token である場合は Implici Grant のフローが選択される．しかし，Implicit Grant のフローでは，URL に直接アクセストークンを付与しているのでアクセストークンの漏えいやアクセストークンの再利用といった脆弱性が存在しており推奨されていない [2].

2.2.3 認可コード

認可コードは，ユーザがクライアントにユーザへリソースへのアクセスを許可したことを証明するランダムな文字列である．クライアントは認可コードが漏えいするリスクを軽減するために認可コードの有効期限を設定しなければならず，その有効期限は最大でも 10 分とすることを推奨されている．また，同じ認可コードを 2 回以上使用してはならず，同じ認可コードが 2 回以上使用された場合は認可サーバがリクエストを拒否し，その認可コードを基に発行されたこれまでのすべてのトークンを無効化するべきであるとされている．また，RFC6749[1] では，『攻撃者が生成されたトークン (とエンドユーザにより扱われないその他のクレデンシャル) を推測する可能性は 2^(-128) 以下にしなければならず (MUST)，2^(-160) 以下にすべきである (SHOULD)』とされている．

3. OAuth2.0 における CSRF 脆弱性

3.1 OAuth2.0 における CSRF 攻撃の流れ

OAuth2.0 には CSRF 攻撃に対する脆弱性が存在している [1][2]．ここではユーザが利用しようとしている Service Provider(以下 SP) を SP1, SP1 と同じ Identify Provider(以下 IdP) を共有しており，OAuth 認証に用いる SP を SP2 とする．ユーザは SP2 のアカウントを所有しており，OAuth2.0 認証の実装された SP1 のサービスを利用し IdP で認証を行う際に攻撃者の SP2 のリソースが結びつけられることで CSRF 攻撃は行われる．OAuth2.0 における CSRF 攻撃のフローを図 3 に示す．

(1) 攻撃者は攻撃に使用するための SP2 のアカウントを作成する．
(2) 攻撃者は SP1 に対して，攻撃用に作成した SP2 のアカウントを用いた認証を開始し，図 2 の (1)〜(5) のやり取りを行う．
(3) IdP は攻撃者へ攻撃者の SP2 のアカウントに基づき発行された，SP1 のアカウントに攻撃者の SP2 のリソースへのアクセスを認可した認可コードの付与されたリダイレクト URL(以下 Trap URL) を送信する．
(4) 攻撃者は SP1 との通信を中断する．
(5) 攻撃者は SP1 になりすましてユーザに Trap URL を送信する．
(6) ユーザは攻撃者から受け取った Trap URL にアクセスし，SP1 との通信を続行する．
(7) SP1 はユーザから受け取った SP1 のアカウントに攻撃者の SP2 のリソースへのアクセスを認可した認可コードを元に IdP へアクセストークンを要求する．
(8) IdP は攻撃者のリソースに紐づくアクセストークンを SP1 へ送信する．
(9) SP1 は攻撃者のリソースに紐づくアクセストークンを SP2 のリソースサーバへ送信する．
(10)ユーザの SP1 でのアカウントに攻撃者の SP2 のアカウントのリソースが紐づき，攻撃者は SP2 のリソースを通じて SP1 から情報を得たりすることが可能になる．

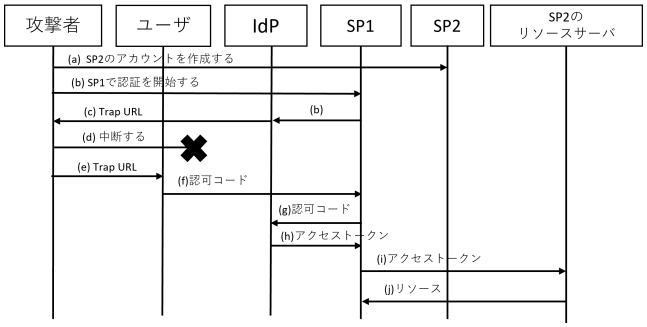

図 3 CSRF 脆弱性を利用した攻撃の手順

SP1 が state パラメータを用いる実装である場合，図 4 のように CSRF 攻撃を防ぐことができる．state パラメータが付与された実装では，攻撃者が SP1 で認証を開始する際に，攻撃者のセッションに紐づく state パラメータが TrapURL に付与される．TrapURL に付与された state パラメータの値は，ユーザのセッションに紐づく state パラメータの値と異なるため SP1 はユーザからのアクセスを拒否する．

3.2 OAuth2.0 における CSRF 攻撃の被害例

OAuth2.0 における CSRF 攻撃を受けると，ユーザは SP1 に攻撃者の SP2 のリソースが紐づけられた状態で SP1 にアクセスした状態となる．SP1 と SP2 のサービスの内容によってはユーザが攻撃者から更なる被害を受けること

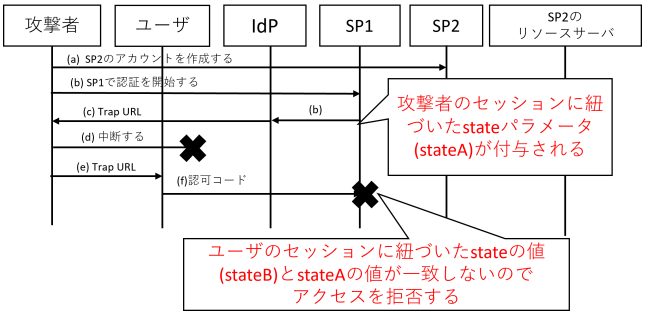

図 4 state パラメータによる CSRF 攻撃対策

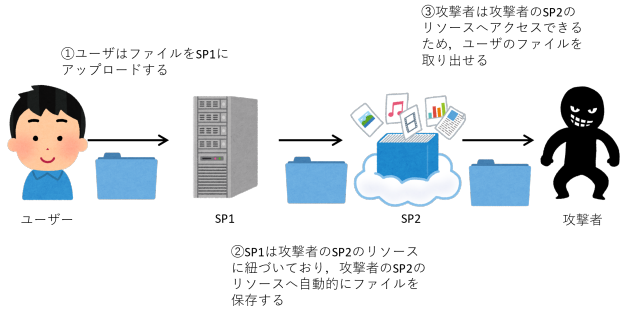

図 5 被害例

https://example.com?**state=FbwCyQF5AJ**? scope=email

図 6 state パラメータの例

が考えられる．例として，以下のような場合を考える．

SP1 画像や動画等のファイルを管理する Web アプリケーションで，アップロードされたファイルを自動的に外部のオンラインストレージサービスのリソースに保存する．

SP2 画像や動画などのファイルを保存するオンラインストレージサービス．

この場合，ユーザが SP1 にアップロードしたファイルは，SP2 の攻撃者のリソースに保存される．攻撃者は自身の SP2 のアカウントにログインすることで，図 5 のようにリソースに保存されたユーザのファイルをとりだすことができる．

3.2.1 state パラメータ

OAuth2.0 では CSRF 脆弱性への対策としてリダイレクト URL に state パラメータを付与する方法が推奨されている．state パラメータは，認証を開始したセッション（2.2.1 節のセッション A）と認可コードを要求するセッション（2.2.1 節のセッション B）が同一のユーザによって行われている一連のセッションであるのかを判定するための文字列であり，図 6 のようにリダイレクト URL のクエリパラメータに付与される．state パラメータが付与されていると図 7 のように CSRF 攻撃を防ぐことができる．state パラメータが付与されている実装では，state パラメータの値は認証を開始したセッションに紐づけられている．その

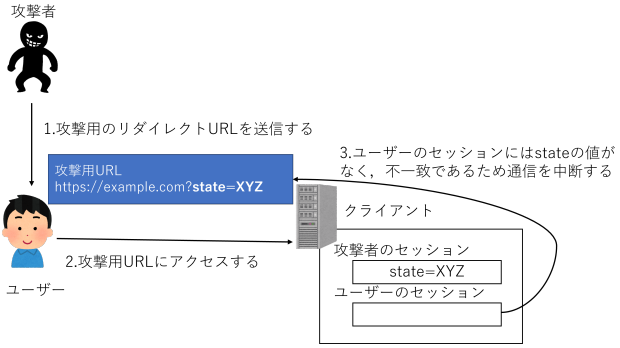

図 7 state パラメータによる検証

ため，攻撃者が用意した URL には攻撃者のセッションに紐づく state パラメータが付与されている．しかし，ユーザはクライアントでの認証を開始していないためユーザのセッションに紐づいた state パラメータは無く，攻撃者の用意した URL に付与されている state パラメータと値が一致しないためアクセスを拒否する．

4. ブラウザの拡張機能を用いた OAuth2.0 実装の判定方法

本研究では，ブラウザの拡張機能を用いることで，Web アプリケーションで OAuth2.0 が CSRF 攻撃への対策をして実装されているかを判定することを目標とする．本実装では，菊田らの調査 [4][5] で OAuth2.0 の実装が多数確認され，脆弱な実装も複数存確認された Google と Facebook を対象とする．Twitter は TwitterOAuth[10] という独自の認可フレームワークが用いられているため実装の対策外とした．また，提案手法内でのリダイレクト URL はユーザが OAuth 認証を行う際にクライアントから送られてくる認可サーバへの URL のことを示す．

4.1 state パラメータの実装状況の判定

state パラメータはリダイレクト URL に付与されているため，ユーザが通信している Web アプリケーションからのリダイレクト URL を取得し，そのリダイレクト URL に state パラメータが付与されているかを確認することで，脆弱性に対する対策がなされているのかを判断する．取得したリダイレクト URL に state パラメータが付与されていない場合，その Web アプリケーションは CSRF 攻撃へ脆弱に対して脆弱な実装であるとする．取得したリダイレクト URL に state パラメータが付与されていた場合，その state パラメータに数字，小文字，大文字が含まれているかを判定し，state パラメータに含まれている文字の種類と文字数からリダイレクト URL に付与されている state パラメータの値が推測される確率を算出することで，認可コードの場合と同様に十分に小さな確率でしか推測し得ないかどうかを判定する．state パラメータがリダイレクト URL に付与されているかの判定は，ユーザが SNS の認証画面

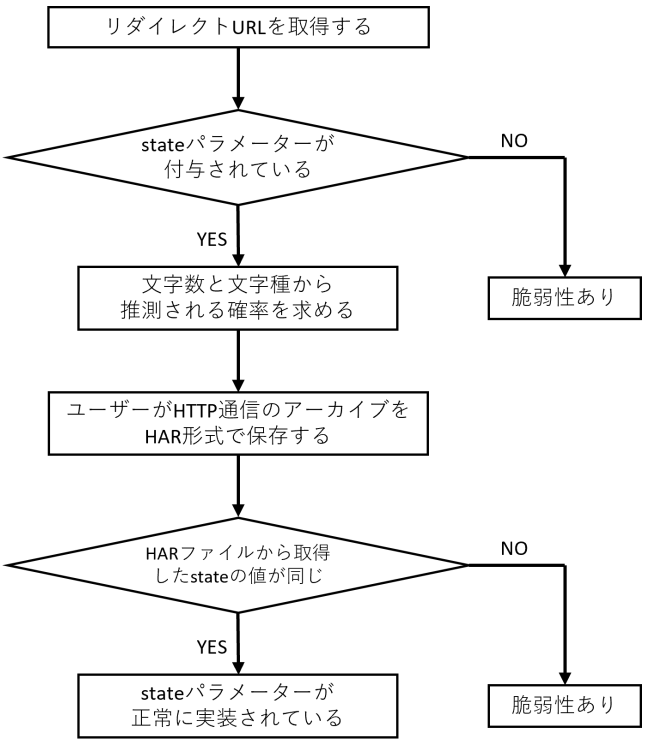

図 8 state パラメータの確認フロー

に遷移した際に行い，ユーザが認証を行う前に判定することができる．また，state パラメータは付与されているがクライアントが state パラメータの値の判定を行っていないため，途中で state パラメータの値が変更されていても認証が行われている場合が考えられる．そのため，ユーザがクライアントへアクセスしてからしてからクライアントへリソースへのアクセス権限の委譲を終えるまでの HTTP 通信を取得し，state パラメータの値が同一であるかを検証する必要がある．ブラウザでは開発者モードを利用することで，ユーザの HTTP 通信のやり取りを HAR ファイルで保存することができる．そのため，ユーザは開発者モードで HTTP 通信のやり取りを HAR 形式で保存し，拡張機能に実装した HTML ファイルで読み込むことで使用されている state パラメータの値が同一であるかの判定を行う．state パラメータを確認するフローを図 8 に示す．

4.2 アクセストークンの実装状況の判定

アクセストークンの発行フローはリダイレクト URL に付与されている response_type の値によって決定される．response_type の値が token である場合は URL に直接アクセストークンが付与されてしまい，アクセストークンが漏えいする可能性がある [6] ので脆弱性のある実装であるとする．リダイレクト URL に付与されている response_type の値 code である場合は，認可コードを利用したアクセストークンの発行フローであるため，その後の HTTP 通信に付与されている認可コードが十分に推測されにくい値であるかをさらに判定する．HTTP 通信で用いられた認可コードの値は，state パラメータが全て同じ値であるのかを判定するために保存したものと同じ HAR ファイルから取得する．菊田らの調査 [4] によると，確認された認可コードの値は Google では『4/jg』，Facebook では『AQ』から始まるという特徴があった．そこで，本実装では取得した認可コードの頭文字がそれぞれの特徴に一致していた場合は認可コードが適切に発行されていると判定した．しかし，現在確認された Google の認可コードでは『4/』のに文字のみ共通しておりそれ以降はランダムな値であったので Google を用いた OAuth 認証で用いる認可コードは先頭 2 文字が『4/』であれば適切なフローで生成された値であるとする．Facebook の認可コードの特徴は現在も変化がなかったので先頭 2 文字が『AQ』であれば適切なフローで生成された値であるとする．RFC6749[1] では，『攻撃者が生成されたトークン (とエンドユーザにより扱われないその他のクレデンシャル) を推測する可能性は 2^(-128) 以下にしなければならず (MUST)，2^(-160) 以下にすべきである (SHOULD)』とされているので，認可コードに使用されている文字種と文字数が十分に推測されにくいための条件も満たしているかを判定する．認可コードに使用されている文字種毎に，認可コードが推測される確率が基準の値以下であるために必要な文字数の条件は表 1 のようになる．この時，認可コードの共通する頭文字は推測される可能性が高いので判定する際の文字数に含めないものとする．取得した認可コードの値が推測される確率を 2^(-128) 以下にするために必要な文字数より少ない場合脆弱性のある実装とする．また，推測される確率が 2^(-128) 以上 2^(-160) である場合は改善の余地のある実装とする．実際に使用している文字種より検知した文字種が少ない可能性があるため，脆弱性が確認された際は認可コードを再取得しもう一度検証を行う．それに加えて，認証を終えるまでの通信内で付与されている認可コードの値が途中で変更されていた場合 CSRF 攻撃を受けている可能性があるため，認可コードを全て取得し，同じ値であるかを判定する必要がある．認可コードに関する検証は権限の委譲が完了してから行う．認可コードを確認するフローを図 9 に示す．

表 1 RFC6749 の条件を満たす文字種と文字数

	2^(-128) (必須)	2^(-160) (推測)
10 種	39 文字	49 文字
26 種	28 文字	35 文字
36 種	25 文字	31 文字
52 種	23 文字	29 文字
62 種	22 文字	27 文字

5. 評価

5.1 結果

Google や Facebook を用いた OAuth 認証を実装しているサイトを対象に，提案した手法を用いて OAuth2.0 に存

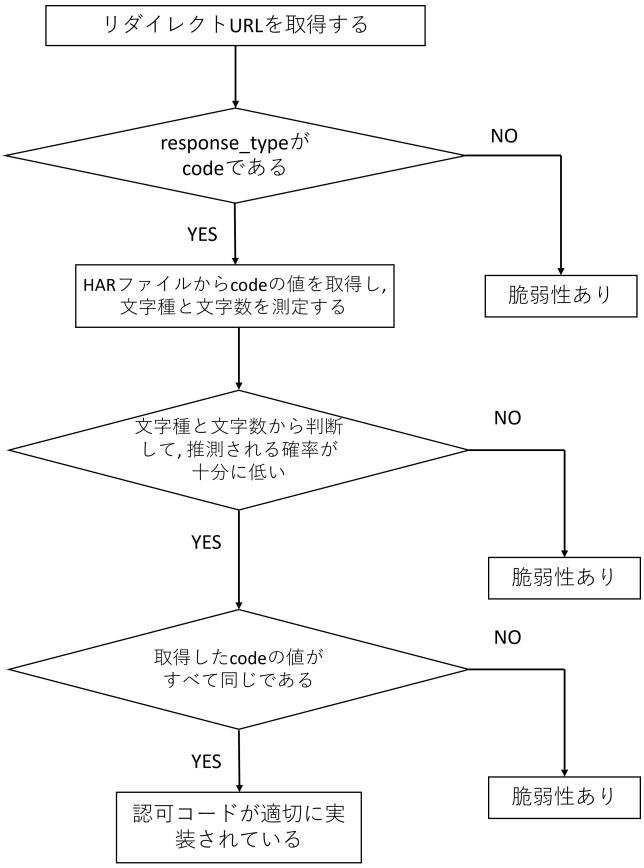

図 9 認可コードの確認フロー

在する脆弱性への対策がなされた実装であるかを判断できるかを評価した．検証には Firefox を用いた．提案手法では，拡張機能を用いてリダイレクト URL に付与されている state パラメータを取得することで，CSRF 脆弱性に対する対策がなされているかを可視化することを目的とした．また，拡張機能のみでは取得出来なかった HTTP 通信のアーカイブは，ユーザが開発者モードから HAR ファイルで保存することにより取得する．その HAR ファイルを用いて，state パラメータが常に同じ値であるか，認可コードが付与されていて十分に複雑な値であるかを判定することを目的とした．

5.1.1 state パラメータが付与されているかの判定

提案手法では，拡張機能のみを用いてリダイレクト URL を取得し state パラメータが付与されているかを判定することで CSRF 攻撃に対して脆弱な実装ではないかを検知することを目標とした．提案手法を用いることで，Google では図 10，Facebook では図 11 のようにユーザが認可サーバで認証を終える前に，リダイレクト URL に付与されている state パラメータを取得する事ができた． ユーザが認証を開始してからリソースがクライアントへ送信されるまでの HTTP 通信で使用される state パラメータの値は，HTTP 通信の内容から取り出す必要があるが，これをブラウザの拡張機能のみで実装することはできなかった．しかし，ユーザがブラウザの開発者モードから手動で HAR

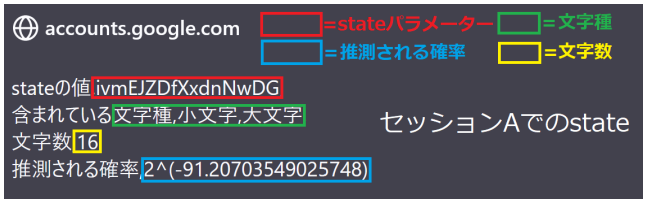

図 10 Google での state パラメータの判定

図 11 Facebook での state パラメータの判定

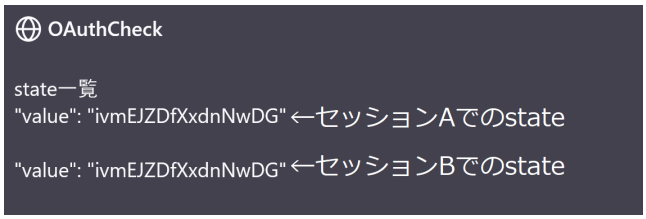

図 12 HAR ファイルを用いた state パラメータの判定-Google

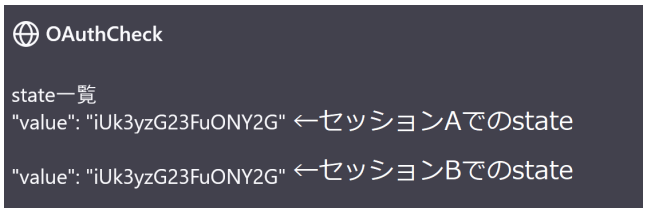

図 13 HAR ファイルを用いた state パラメータの判定-Facebook

図 14 state パラメータが付与されていない OAuth2.0 実装の検知

ファイルを保存し，その HAR ファイルを拡張機能内で読み込むことで全ての state パラメータを取り出すことができた．提案手法を用いることで，ユーザが認証を終えた後に，Google では図 12，Facebook では図 13 のように全ての state パラメータを取得する事ができた．

また，state パラメータが付与されていない場合は図 14 のように検知することができた．以上の結果より，拡張機能のみを用いることでリダイレクト URL に state パラメータが付与されているかを判断する目標が達成された．これにより，ユーザが認証を行う前に CSRF 脆弱性が存在するかを知ることができたので，CSRF 攻撃による被害を未然に防ぐ事ができる．また，HAR ファイルから state パラメータの値を取得し全てが同じ値であるかを判定する事が

図 15 アクセストークンを用いている場合

図 16 認可コードを用いている場合

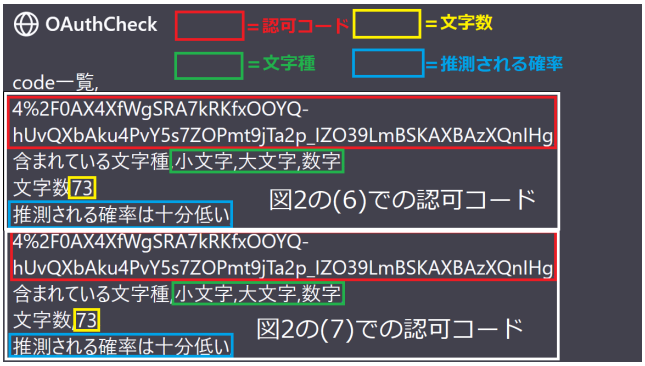

図 17 認可コードの取得と判定-Google

達成できたため，認証中に state パラメータが変更されていた場合もユーザがアクセストークンを無効化することで CSRF 攻撃による被害を抑える事ができる．

5.1.2 認可コードが予測困難であるかの判定

ブラウザの拡張機能を用いることで，ユーザが認証を開始する前に，URL に直接アクセストークンが付与されている場合は図 15，アクセストークンの代わりに認可コードを用いている場合は図 16 のように判断する事ができた．

認可コードの値はブラウザの拡張機能のみで取得することはできなかったが，HAR ファイルを読み取ることで認可サーバから受け取る認可コードとクライアントへ送信する認可コードの値を Google では図 17，Facebook では図 18 のように取得する事ができた． 以上の結果より，提案手法を用いることで HAR ファイルから URL でアクセストークンの代わりに認可コードが付与されているかを判断し，HTTP 通信でアクセストークンの代わりに認可コードが付与されていた場合は第三者に値を推測される確率が十分に低いかを判定することが出来た．ユーザが利用している Web アプリケーションでの実装に脆弱性がある場合は，ユーザがアクセストークンを無効化することでリソースの漏洩や書き換えなどの被害を未然に防ぐことができる．

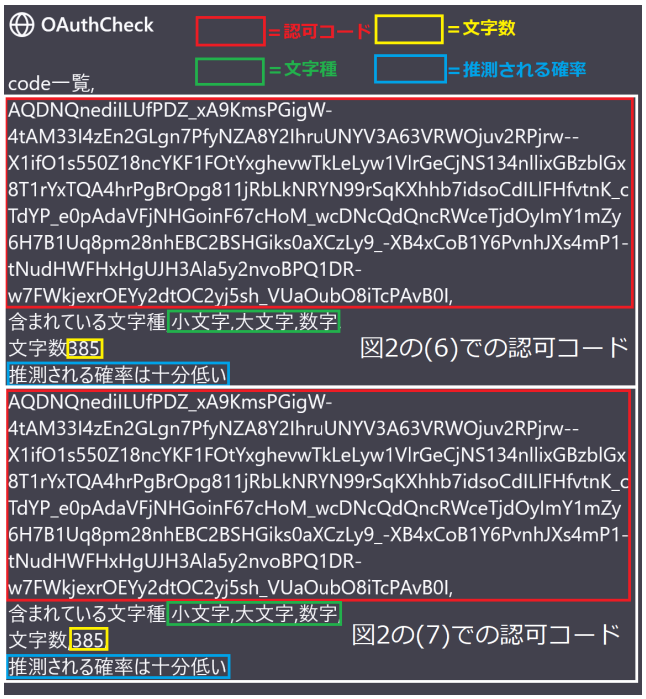

図 18 認可コードの取得と判定-Facebook

5.2 考察

本手法では，state パラメータや認可コードの値が適切であるかを判定するためにはユーザが HTTP 通信のアーカイブを HAR ファイルで保存する必要があり，拡張機能のみで判定することはできなかった．一方で，ブラウザ拡張を用いることで，OAuth 認証による認証を実装しているが，リダイレクト URL に state パラメータや認可コードが付与されていない CSRF 攻撃等に脆弱な Web アプリケーションを検知する事ができたため，ユーザが利用しようとしている Web アプリケーションが OAuth 認証における CSRF 攻撃に対して脆弱であるかどうかを知ることが出来，被害を未然に防ぐことができると考えられる．

6. まとめ

本研究では，まず Web アプリケーションにおける SSO 認証として OAuth2.0 が採用されており，OAuth2.0 の実装状況によっては CSRF 攻撃等に脆弱となることを述べた．また，CSRF 攻撃等に脆弱なまま OAuth2.0 が実装されている Web アプリケーションが複数存在することを述べ，脆弱性のある OAuth 実装をユーザへ事前に知らせる事で未然に防ぐ必要がある事を示した．そこで，複数の Web アプリケーションで確認されている，URL に state パラメータのない実装，リダイレクト URL でアクセストークンの代わりに認可コードが付与されていない実装をブラウザの拡張機能を用いて検知しユーザへ知らせることを目的として，ブラウザの拡張機能の開発と検証を行った．その結果，ブラウザ拡張を用いることでリダイレクト URL に state パラメータや認可コードが付与されていない明ら

かな脆弱性が存在する Web アプリケーションを検知する事ができた．これにより，ユーザは悪意のある URL にアクセスすることを避けることができ，CSRF 攻撃の被害に遭うことを未然に防ぐことができる．

しかし，認証が終了するまでに付与されている全ての state パラメータや認可コードの値が適切であるかを判定するためにはユーザがブラウザの開発者モードから HTTP 通信のアーカイブを HAR ファイルで保存する必要があり，拡張機能だけで判定することは実現できなかった．また，クライアントが state パラメータの値を検証しているかを判断することが出来ないため，今回の実装では CSRF 攻撃に対して脆弱性のある実装ではないと判定されるが，実際には CSRF 攻撃に対して脆弱である Web アプリケーションが存在してしまう．したがって，クライアントが，認証の途中で異なる値の state パラメータの付与された通信が行われた際に通信を中断しているのかを追加検証することで精度を上げることができる．また，ブラウザの拡張機能を用いた方法では PC ブラウザ以外で脆弱な OAuth2.0 の実装を検知することができないため，スマートフォンなど，ブラウザの拡張機能が利用できない環境でも検知する方法を開発することが今後の課題である．

参考文献

[1] The OAuth 2.0 Authorization Framework https://tools.ietf.org/html/rfc6749
[2] OAuth 2.0 Threat Model and Security Considerations https://datatracker.ietf.org/doc/html/rfc6819
[3] The OAuth 1.0 Protocol https://datatracker.ietf.org/doc/html/rfc5849
[4] 菊田翼, 真木康太郎, 細谷竜平, 八代哲, 齋藤孝道. (2019). OAuth/OpenID Connect 実装におけるセキュリティ状況の調査. 第 81 回全国大会講演論文集, 2019(1), 487-488.
[5] 上瀧悠輔, 菊田翼, 小芝力汰, 齋藤孝道. (2020). 米国の OAuth/OpenID Connect 実装におけるセキュリティ状況の調査および日米比較. 第 82 回全国大会講演論文集, 2020(1), 423-424.
[6] Zhou, Y.,Evans, D. (2014). SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. In 23rd USENIX Security Symposium (USENIX Security 14) (pp. 495-510).
[7] アドオン-Mozilla—MDN. 入手先 ⟨https://developer.mozilla.org/ja/docs/Mozilla/Add-ons⟩
[8] OAuth2.0 を使用した Google API へのアクセス Google Identity Platform Google Developers https://developers.google.com/identity/protocols/oauth2
[9] ウェブ-Facebook ログイン https://developers.facebook.com/docs/facebook-login/web
[10] TwitterOAuth https://twitteroauth.com/