

Web アプリケーションの開発時における Session 管理の脆弱性の自動検査手法

高松 勇輔^{†1} 小菅 祐史^{†1} 河野 健二^{†1,†2}

ログインや商品購入などの機能を持つ Web アプリケーションが利便性のためにセッションを採用している。セッションはユーザの状態（ログイン状態やページ遷移の状態などの情報）などの情報である。それぞれのユーザは Web アプリケーションで管理しているセッションと関連づけられている。攻撃者はユーザのセッションを乗っ取ることでユーザの識別を乗っ取ることができるために、セッションは攻撃者の標的になっている。セッション管理の脆弱性を取り除くためには、実運用前にこれらの脆弱性に対する検査を Web アプリケーションに行うことが重要である。しかしこうした検査を行うためには開発者には脆弱性についての十分な知識が必要であるため、開発者がこれらの脆弱性を検査することは難しい。本論文では Web アプリケーションにセッション管理の脆弱性が存在するかを自動的に検査するシステムを提案する。提案システムはこれらの脆弱性を突く攻撃を自動的に生成する。そして対象の Web アプリケーションに対して生成した攻撃を実行し、攻撃が成功したか調査することで脆弱性を検査する。Session Fixation と Cross-Site Request Forgery の脆弱性を検査するプロトタイプを実装し、提案システムがオープンソースの Web アプリケーションの脆弱性を検出できることを示す。

Automated Detection of Session Management Vulnerabilities

YUSUKE TAKAMATSU,^{†1} YUJI KOSUGA^{†1}
and KENJI KONO^{†1,†2}

Many web applications employ some kind of session management in order for user-friendly functions like a login function and a purchase function. A session is a piece of information of user's activity (e.g. login state), and each user is related to a session. Naturally, sessions are very appealing targets for attackers since the attackers can hijack a user's identity by hijacking this user's session. Since session management vulnerabilities can be eliminated by carefully implementing a session management function, it is important to thoroughly test a web application for these vulnerabilities before shipping. However, it is difficult for ordinary developers to test web applications for these vulnerabilities because

it needs the detail knowledge about them. In this paper, we propose a technique that automatically tests web applications for session management vulnerabilities. It automatically creates attacks that exploit possible vulnerabilities in the session management of the target web application. Then, it executes these attacks to the web application and detects vulnerabilities by checking if the attacks success. We implemented a prototype that tests web applications against two representative attacks: Session Fixation and Cross-Site Request Forgery. Our experiments demonstrate that our technique can detect the vulnerabilities of some real web applications to the above mentioned vulnerabilities.

1. はじめに

ログインや商品購入などの機能を持つ Web アプリケーションはセッションを採用している。セッションは、各ユーザの情報を表す情報であり、ログイン状態やページ遷移の状態等を含んでいる。それぞれのユーザはセッション ID によって Web アプリケーション側で保存されるセッションと関連づけられている。セッションを用いるとユーザのログイン状態やページ遷移の状態をもとに提供するサービスを変更したり、ユーザ毎にショッピングカートを持つことができる。

セッションを利用することで Web アプリケーションの利便性が向上する一方で、セッションを悪用した攻撃が問題となっている。各ユーザはセッションと関連づけられているため、攻撃者がセッションを乗っ取ることができると、そのユーザになりすますことができるためである。そのような攻撃の例として、Session Fixation [1] や Cross-Site Request Forgery (CSRF) [2] がある。

実運用中の Web アプリケーションにはセッション管理の脆弱性を持つものが多く報告されている。WhiteHat Security [3] によれば、14 %の Web アプリケーションに Session Fixation 脆弱性が、21 %の Web アプリケーションに CSRF 脆弱性があると報告されている。Session Fixation は攻撃者が用意したセッションを被害者に使用させる攻撃である。CSRF は、攻撃者が用意した悪意ある攻撃が被害者によって Web アプリケーションへ送信させるように仕向ける攻撃である。

^{†1} 慶應義塾大学
Keio University

^{†2} 科学技術振興機構 戦略的創造研究推進事業
JST CREST

セッションを悪用した攻撃は、セッション管理に工夫を施すことで防ぐことができる。例えば、Session Fixation の対策手法としてログインの前後でユーザのセッション ID を変更する手法がある。また、CSRF の対策手法としてトークンを利用して攻撃を防ぐ手法がある。したがって、Web アプリケーションの開発時にセッション管理の脆弱性検査を行い、上記のようなセッション管理の工夫が施されていることを確かめることで、セッションを悪用する攻撃を防ぐことができる。

しかし、各開発者がセッション管理の脆弱性を手動で検査することは難しい。手動で脆弱性を検査するためには、テストを行う開発者がセッション管理の脆弱性を突く攻撃を熟知している必要があるためである。よって、セッション管理の脆弱性を自動で検査するツールが有用であると考えられる。

本研究では対象の Web アプリケーションにセッション管理の脆弱性が存在するかを自動的に検査するシステムを提案する。このシステムは模擬的な攻撃者と模擬的な被害者を用意して実際に Web アプリケーションに攻撃を仕掛け、その攻撃結果を解析して脆弱性を検査する。ここで、Web アプリケーションに自動で攻撃を仕掛けて検査を行うためには、その Web アプリケーションが持ちうる脆弱性を突く攻撃を自動で生成する必要がある。そこで本システムは、Session Fixation 等の各攻撃毎に、その攻撃の手順と必要とする情報を表す攻撃テンプレートを用意しておき、そのテンプレートに対して必要な情報を付加することで攻撃を自動生成する。攻撃の自動生成に必要な情報の多くは、開発者が Web アプリケーションをテスト実行した際の HTTP リクエストとレスポンスを監視することで自動抽出する。そのため、開発者は一部の簡単な情報を入力するだけでよい。ここで、本システムは Web アプリケーションの開発段階に用いることを前提としているため、開発者が情報を入力することは可能である。

提案システムを脆弱性検査用のフレームワークである Amberate [4] のプラグインとして実装した。Amberate は開発者と Web アプリケーションのやりとりを監視し、HTTP リクエストとレスポンスをプラグインに渡す。本研究では、Session Fixation と CSRF 用のプラグインをそれぞれ実装した。

提案システムの有効性を示すために、セッション管理の脆弱性を持つオープンソースの Web アプリケーションで実験を行った。実験では、対象の Web アプリケーションに提案システムを適用して、セッション管理の脆弱性の有無を検査した。そして対象の Web アプリケーションに手動で攻撃を行い、提案システムで検査した結果が正しいかどうか調査した。その結果、オープンソースの Web アプリケーションに提案システムで行った検査結果と手

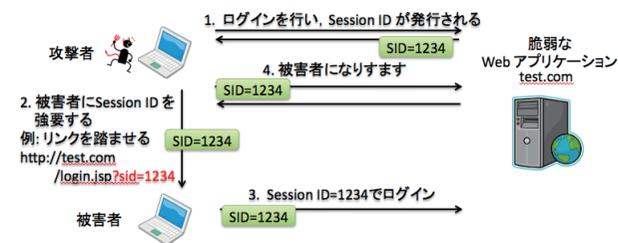


図 1 Session Fixation

動で行った検査結果が同じ結果であった。

本論文の構成は以下の通りである。2 章ではセッション ID やセッション管理の脆弱性や既存の対策手法について述べる。6 章では既存の関連研究について述べる。3 章では提案システムについて説明する。4 章では提案システムの実装について説明する。5 章では提案システムの有用性を確認した実験と結果について述べる。7 章では今後の課題について説明する。最後に 8 章で本論文をまとめる。

2. セッション管理の脆弱性

セッションは、各ユーザの情報を表す情報であり、ログイン状態やページ遷移の状態等を含んでいる。Web アプリケーションはセッションをそれぞれのユーザごとに保持し、複数のページでこのセッションを共有する。それぞれのユーザはセッション ID によって Web アプリケーション側で保存されるセッションと関連づけられる。Web アプリケーションは複数のユーザから特定のユーザを識別するために、異なるセッション ID をユーザごとに割り当てる。Web アプリケーションがセッション ID をブラウザに向けて発行し、ブラウザが送るリクエストにそのセッション ID が含まれる。

セッションを利用することで Web アプリケーションの利便性が向上する一方で、セッションを悪用した攻撃が問題となっている。その攻撃の例として、Session Fixation [1] や Cross-Site Request Forgery (CSRF) [2] が挙げられる。

2.1 セッション管理の脆弱性を利用した攻撃

2.1.1 Session Fixation

Session Fixation は、攻撃者が用意したセッションを被害者に利用させ、同一のセッションを攻撃者も利用することで、被害者のセッションを乗っ取る攻撃である [1]。この攻撃により攻撃者は、被害者になりすますことができる。例えば、攻撃によってクレジットカード

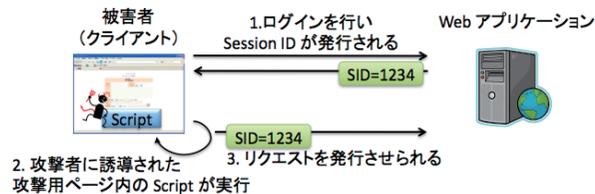


図 2 Cross Site Request Forgery

番号の盗難、意図しない商品購入などの金銭的な被害、またコミュニティからの退会、他のユーザへの犯罪・迷惑メール等のさまざまな被害が生じる。

図 1 とともに、Session Fixation の攻撃手順を示す。最初に攻撃者が Web アプリケーションにログインする。そして攻撃者は Web アプリケーションのレスポンスからセッション ID (=1234) を獲得する。攻撃者は獲得したセッション ID (=1234) を被害者に強要する罠リンクを作成し、フィッシングやメールなどでその罠リンクに被害者を誘導する。次に被害者は強要された攻撃用セッション ID (=1234) を使用してログインする。最後に攻撃者が被害者になりすますために、リクエストに攻撃用セッション ID (=1234) を含み送信する。その結果、Web アプリケーションは攻撃者を被害者とみなすので、攻撃者は被害者のセッションを利用して被害者になりすますことができる。

2.1.2 Cross-Site Request Forgery (CSRF)

CSRF は、攻撃者が用意した悪意あるリクエストが、被害者によって Web アプリケーションへ送られるように仕向ける攻撃である。この攻撃により攻撃者は、被害者に意図しない操作を行わせることができる。例えば、攻撃によって攻撃者の指定した株や商品をむりやり購入させるなどの金銭的な被害が生じる。

図 2 とともに、CSRF の攻撃手順を示す。最初に被害者は Web アプリケーションにログインする。次に攻撃者はフィッシングやメールなどで被害者を攻撃用ページに誘導する。そして被害者のブラウザ上で攻撃用ページ内のスクリプトが実行され、被害者は Web アプリケーションにリクエストを送信させられる。ブラウザはこのリクエストにセッション ID (=1234) を含み送信する。そして Web アプリケーションは、送られてきたリクエストをセッション ID (=1234) から被害者のリクエストとして処理する。CSRF は、Web アプリケーションがセッション ID のみでセッション管理を行っている時、攻撃者がむりやり送信させたリクエストを被害者のリクエストを識別することができないために生じる。

2.2 対策手法

2.2.1 Session Fixation の対策

Session Fixation の対策手法として、ログインの前後でユーザのセッション ID を変更する手法がある。被害者がログインした後に Web アプリケーションが新たなセッション ID で被害者を管理する。攻撃者が知っているセッション ID は、ログイン前のセッション ID であるため、ログイン後の被害者のセッションを乗っ取ることはできない。

2.2.2 CSRF の対策

CSRF の対策手法として、トークンを利用して攻撃を防ぐ手法がある。トークンとは、送られてきたリクエストが正規のリクエストかどうかを識別するための識別子である。Web アプリケーションが、ユーザが重要な処理を行う場合 (ログインや商品購入) のリクエストにトークンが含まれるようにレスポンスの URL などにトークンを埋め込む。こうすることで、Web アプリケーションと正規の通信を行っているリクエストには、トークンが含まれるようになる。したがって、Web アプリケーションはリクエストにトークンが含まれているかを確認することで、そのリクエストが強要されたリクエストか正規のリクエストかを識別することができる。

さらにセッション ID の伝播法に URL Rewriting か Hidden Field を利用すると Web アプリケーションに CSRF 脆弱性が無くなる。これは Cookie の場合、攻撃者は被害者にリクエストを発行させると被害者のブラウザがこのリクエストに被害者のセッション ID を付加して送信する。一方で、URL Rewriting か Hidden Field の場合、攻撃者はリクエストの中に被害者のセッション ID を含んで被害者にリクエストを発行させなければいけない。しかし攻撃者は被害者のセッション ID を推測できないために、強要させたいリクエストの中に被害者のセッション ID を含むことができず、攻撃は成功しない。

2.3 Web アプリケーション開発時の脆弱性検査

Web アプリケーションの開発時にセッション管理の脆弱性検査を行うことで、セッションを悪用する攻撃を未然に防ぐことができる。前節で述べたように、セッションを悪用する攻撃は、それらを意識してセッション管理を実装することで防ぐことができる。ところが、実装ミスによるバグによって対策が不十分であったり、開発者の知識不足によってこれらの対策が取られていない場合が考えられる。そこで、開発時にテストを行い、セッションを悪用した攻撃を防げることを確認する必要がある。しかし、各開発者がセッション管理の脆弱性を手動で検査することは難しい。なぜなら脆弱性を検査するためには Web アプリケーションに攻撃をするしかなく、攻撃を行うためには開発者はその攻撃について十分な知識を

必要があるためである。しかし全ての開発者が攻撃について十分な知識を持つことは難しい。そこで、セッション管理の脆弱性を自動で検査するシステムが有用であると考えられる。

3. 提案手法

本研究は対象の Web アプリケーションにセッション管理の脆弱性が存在するかを自動で検査するシステムを提案する。前述したように、Web アプリケーションが適切なセッション管理を行えば、セッション管理の脆弱性を突いた攻撃を防ぐことができる。Web アプリケーションの開発時に、提案システムを利用してデバッグを行うことで、Web アプリケーションの安全性を高めることができる。提案システムは自動的にテストを行うため、Session Fixation や CSRF などの攻撃に関する詳細な知識が無い開発者でもテストを行うことができる。そのため、効率よく Web アプリケーションの安全性を向上させることができる。

提案システムは、診断対象の Web アプリケーションに対し、セッション管理の脆弱性を突く攻撃を実際に行う。そうすることで、その攻撃に対する対策が取られているかどうかを検査する。適切な対策が取られていない場合、提案システムが行った攻撃が成功するため、その攻撃に関して脆弱性が存在すると判断できる。

自動的にセッション管理の脆弱性を突く攻撃を行って検査を行うためには、これらの攻撃を自動的に生成する必要がある。本システムは、各攻撃の手順を表す攻撃テンプレートを予め用意しておき、そのテンプレートに適切な情報を付加することで攻撃を自動生成する。

3.1 攻撃の自動生成

本システムでは、予め各攻撃を汎用的に表す攻撃テンプレートを用意しておき、Web アプリケーション固有の情報を攻撃テンプレートに適用することで攻撃を自動生成する。

3.1.1 攻撃テンプレート

攻撃を生成するために、攻撃の概要を示した攻撃テンプレートを用意する。2章で説明したそれぞれの攻撃の手順をもとに Session Fixation と CSRF 用のテンプレートのを以下に示す。

Session Fixation 用のテンプレート

1. 攻撃者がログインし、攻撃用のセッション ID を獲得する
2. 被害者が攻撃用のセッション ID でログインする
3. 攻撃者が攻撃用のセッション ID を使用して、被害者専用のページを要求する

CSRF 用のテンプレート

1. 被害者がログインする

2. 攻撃者が被害者にむりやり送信させる強要リクエスト（開発者が検査したいページのリクエスト）を作成する
3. 被害者が強要リクエストを送信する

3.1.2 攻撃生成に必要な情報

攻撃の生成に必要な情報を与える方法として、システムの利用者が全て手動で与える方法が考えられる。本システムはアプリケーション開発時のテスト工程で利用することを想定しているため、アプリケーションの中身を把握している開発者が手動で情報を与えることが可能である。しかし、全ての情報を手動で与えるためには、その攻撃に関する深い知識が必要となる。各攻撃の手順等を把握していないと、どのような情報を必要とするかがわからないためである。その結果、システムの利便性が損なわれてしまう。

そこで本システムは、攻撃の生成に必要な情報を 1) Web アプリケーションの挙動から自動抽出できるもの、2) 開発者が静的に与える必要があるものの 2 つに大別し、多くの情報を自動抽出する。

Web アプリケーションのセッション ID 伝播方法やログインリクエストは自動抽出することが可能である。具体的な方法は、次項で述べる。その他の情報は、開発者が手動で与える必要がある。具体的には、以下のような情報を開発者が入力する必要がある。

- セッション ID の名前: どの方法を利用してセッション ID を伝播しているか特定し、セッション ID の値を特定するために利用する。
- 攻撃者と被害者のユーザ名とパスワード: 攻撃者と被害者が Web アプリケーションにログインするために利用する。
- 攻撃成功時のみ現れる特有の文字列: 攻撃後のページを解析するために利用する。CSRF のテンプレートにのみ必要な情報について以下に示す。
- トークンの名前 (脆弱性対策にトークンを利用している場合に限り): トークンの値を特定するために利用する。
- 検査したい機能の URL: 開発者にクライアントとして検査したい機能を使用してもらう中でどのページを検査するか決定するために利用する。

3.1.3 情報の自動抽出

セッション ID の伝播方法やログインリクエストなどの情報は、HTTP リクエスト&レスポンスを監視した情報と開発者から静的に獲得した情報を利用することで自動抽出できる。具体例として Session Fixation と CSRF に必要な情報として、セッション ID がある。これについては、攻撃を行うために攻撃者と被害者のセッション ID を管理する必要があり、

Web アプリケーションがセッション ID を変更した場合に、その変更をリクエストに反映させる必要があるためである。セッション ID を搬送する方法は、Cookie、URLRewriting、Hidden Field の 3 種類がある。Cookie はレスポンスヘッダの Cookie、URLRewriting は HTML ドキュメント内の URL、Hidden Field は HTML ドキュメント内のフォームにある Hidden Field を調べることでセッション ID をどの方式で伝播しているかわかる。つまりクライアント（開発者）が操作したときに発行された HTTP リクエスト・レスポンスを開発者に与えてもらったセッション ID の名前で解析することで、対象の Web アプリケーションがどの方式で伝播を行っているかわかる。

また Session Fixation と CSRF に必要な情報として、ログインを行うためのリクエストの特定がある。これについては、攻撃時にそのリクエスト内のユーザ名とパスワードを攻撃者や被害者のユーザ名やパスワードに変更して Web アプリケーションにログインを行うためである。まずリクエストが GET か POST かどうかを識別する。ログインするためにはユーザ名とパスワードを入力するので、POST でリクエストが送信される。そして POST の場合、リクエストを発行した HTML ドキュメント内のどの入力フォームから送られてきたものか判断する。そのために、リクエストのボディ内にあるパラメータと HTML ドキュメント内の入力フォームのパラメータを比較し、全てのパラメータが一致した場合、そのフォームからリクエストが発行されたものと判断する。その入力フォーム内にあるパラメータの type の一つが password の場合、その HTML ドキュメントがログインページであると判断する。type が password の場合、その入力フォームにはパスワードが入力されるからである。そしてログインページであると判断したページが発行したリクエストをログインするためのリクエストであると判断する。

3.1.4 攻撃の生成

開発者から静的に獲得した情報や Web アプリケーションの挙動の情報をテンプレートに反映させて攻撃を生成する。

まず Session Fixation の場合の生成方法について説明する。Session Fixation 用のテンプレートの 1 と 2 を実行するためにはログインの手順が、テンプレートの 3 にはユーザ専用のページの要求方法が必要である。そこで、開発者から得た機能の使用方法をテンプレートに反映させる。Session Fixation の場合は、開発者にはログイン機能を操作してもらう。この操作情報からログインの手順とユーザ専用のページの要求方法を獲得することができ、これらをテンプレートに反映することで攻撃手順を作成する。

次に、CSRF の場合の生成方法について説明する。CSRF 用のテンプレートの 1 を実行

するためにはログインの手順が、テンプレートの 2 にはリクエストの要求方法とどの機能を検査したいのかという情報が必要である。CSRF の場合は、開発者にはログイン機能と検査したい機能を操作してもらう。検査したい機能には、例えば商品購入機能などがある。商品購入機能の場合に開発者は、ログインを行い、商品をカートに入れ、決算ページで商品を購入するという操作を行う。この操作情報からログインの手順と決算を行うリクエストの要求方法を獲得することができ、これらをテンプレートに反映させることで攻撃手順を作成する。

3.2 攻撃の実行

模擬的な攻撃者と模擬的な被害者が、対象の Web アプリケーションに生成した攻撃手順に従ってリクエストを送信する。その際に、攻撃者と被害者が Web アプリケーションに送信するリクエストは開発者が発行したリクエストを利用する。しかしユーザや時間によってリクエストの中身が異なるので、開発者から獲得したリクエストをそのまま送信することができない。よって開発者から獲得したリクエストを攻撃者や被害者用のリクエストに変更する必要がある。例えば、開発者から獲得したリクエストには開発者に割り振られたセッション ID がリクエストに入っている。そのリクエスト内のセッション ID を攻撃者や被害者用に Web アプリケーションから割り振られたセッション ID に変更する必要がある。以上のことから攻撃時に攻撃者と被害者に Web アプリケーションから発行されるレスポンスを常に監視し、その情報をもとに必要に応じて開発者から獲得したリクエストを攻撃者や被害者用のリクエストに変更する。

3.3 攻撃結果の検証

攻撃結果として得られたレスポンスを調べて脆弱性の有無を検査する。特定のレスポンス内に開発者が指定した文字列が含まれる場合、セッション管理の脆弱性があると判定し、指定した文字列が含まれない場合、セッション管理の脆弱性がないと判定する。この指定した文字列は攻撃成功時のみページ内に現れる文字列である。例えば、「Thank you, victim (被害者名)」、「Welcome, victim (被害者名)」などである。このようにして脆弱性の有無を検査する。

4. 実 装

提案システムを Amberate [4] の脆弱性検査用プラグインとして実装し、Session Fixation 用と CSRF 用のプラグインをそれぞれ実装した。開発言語は Java で、各々のプラグインについて約 1500 行実装した。Amberate は脆弱性の自動検査を行うフレームワークで、開発

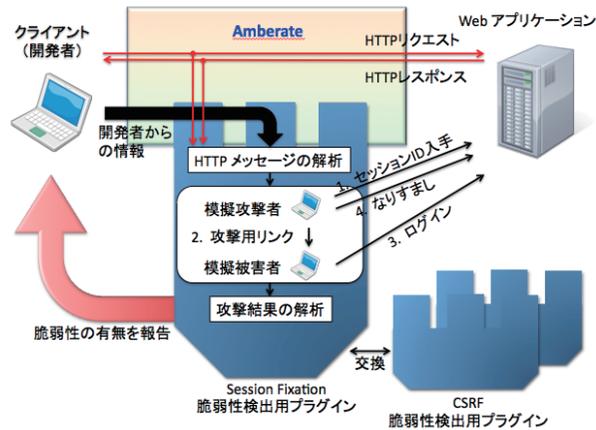


図3 実装の全体像

者の Web アプリケーション操作を監視し、得られた情報をプラグインに渡す。提案システムでは開発者や攻撃時の HTTP リクエストとレスポンスの監視という機能を実装する必要があるために、Amberate の HTTP リクエストとレスポンスを監視する機能を利用した。図3に示すように、プラグインである提案システムは Amberate からの情報と Amberate の機能、開発者からの情報を利用し、Web アプリケーションのセッション管理の脆弱性の有無を検査する。

5. 実験

5.1 マイクロベンチマーク

5.1.1 実験方法

提案手法の有効性を示すために、意図的に脆弱性を含めた自作の Web アプリケーションを用いて実験を行い、正しく脆弱性を検出できるかを検証した。自作の Web アプリケーションには、7種類のセッション管理方法を実装した。また、脆弱性検査の対象とする機能として、ログイン機能と商品購入の機能を実装した。Web アプリケーションの実装は、PHPで行った。

自作 Web アプリケーションが行うセッション管理のパターンとして、Session Fixation脆弱性や CSRF脆弱性を持つパターン、またそれぞれの脆弱性を持たないパターンを想定した。具体的には、自作の Web アプリケーションにセッション管理の方法を7パターン実

装した。

- (1) Session Fixation 対策を行わない：セッション ID を持っていないユーザーのみセッション ID を発行する。
- (2) Session Fixation 対策を行う：対策として、ログインの前後で必ずセッション ID を変更する。
- (3) 独自の Session Fixation 対策を行う：ログイン後のみセッション ID を発行する。ユーザーがセッション ID を持っている状態でログインを行う場合、Web アプリケーションはログイン後に必ず新しいセッション ID を発行する。
- (4) 不完全な Session Fixation 対策を行う：ログイン後のみセッション ID を発行する。ユーザーがセッション ID を持っている状態でログインを行う場合、Web アプリケーションはログイン後に新しいセッション ID を発行せず、ユーザーが持っているセッション ID を使用する。
- (5) CSRF 対策を行わない：商品の購入ページにおいてセッション ID のみでユーザーの識別を行う。
- (6) CSRF 対策を行う：商品の購入ページにおいてセッション ID とトークンでユーザーの識別を行う。このトークンが発行されるリクエストに含まれるように商品の購入ページにトークンを埋め込む。
- (7) 不完全な CSRF 対策を行う：商品の購入ページにおいてセッション ID とトークンの有無でユーザーの識別を行う。このトークンが発行されるリクエストに含まれるように商品の購入ページにトークンを埋め込む。ただし、Web アプリケーションはトークンの値の正当性は検証せず、トークンを持っていれば正しいリクエストとして処理する。

さらに1から4の Session Fixation 脆弱性に関するセッション管理について、Web アプリケーションにセッション ID の伝播法として URL Rewriting と Cookie, Hidden Field をそれぞれ施し、実験を行った。また5から7の CSRF 脆弱性に関するセッション管理について、2章で説明した理由からセッション ID の伝播法として Cookie のみを施し、実験を行った。

手動で1から4のセッション管理を施した Web アプリケーションに Session Fixation 攻撃を行い、5から7のセッション管理を施した Web アプリケーションに CSRF 攻撃を行った。結果として、1と4のセッション管理を行う Web アプリケーションに Session Fixation 脆弱性が発見され、2と3のセッション管理を行うものに Session Fixation 脆弱性は発見

表 1 提案システムによる脆弱性検査と手動による脆弱性検査の結果

セッション管理の パターン	URL Rewriting		Cookie		Hidden Field	
	手動	提案システム	手動	提案システム	手動	提案システム
1	脆弱性あり	脆弱性あり	脆弱性あり	脆弱性あり	脆弱性あり	脆弱性あり
2	脆弱性なし	脆弱性なし	脆弱性なし	脆弱性なし	脆弱性なし	脆弱性なし
3	脆弱性なし	脆弱性なし	脆弱性なし	脆弱性なし	脆弱性なし	脆弱性なし
4	脆弱性あり	脆弱性あり	脆弱性あり	脆弱性あり	脆弱性あり	脆弱性あり
5	-	-	脆弱性あり	脆弱性あり	-	-
6	-	-	脆弱性なし	脆弱性なし	-	-
7	-	-	脆弱性あり	脆弱性あり	-	-

されなかった。さらに5と7のセッション管理を行う Web アプリケーションに CSRF 脆弱性が発見され、6のセッション管理を行うものに CSRF 脆弱性は発見されなかった。

5.1.2 実験結果

それぞれのセッション管理のパターンを施した Web アプリケーションにおいて、手動で Session Fixation と CSRF を行うことによる脆弱性検査の結果と、提案システムによる脆弱性検査の結果を表1に示す。

表1より、全てのパターンで手動による脆弱性検査の結果と提案システムによる脆弱性検査の結果は一致した。以上のことから提案手法は自作の Web アプリケーションに対して脆弱性を検査できたといえる。

5.2 マクロベンチマーク

5.2.1 実験方法

さらに提案システムの有効性を示すために、マクロベンチマークとしてオープンソースの Web アプリケーションを用いて実験を行った。Web アプリケーションの脆弱性を報告しているサイト [5] [6] で Session Fixation 脆弱性や CSRF 脆弱性を持ったオープンソースの Web アプリケーションと脆弱性を持たないものを調査した。これらのオープンソースの Web アプリケーションを利用して、提案システムの脆弱性検査の実験を行った。Web アプリケーションの実装は、全て PHP で行われている。

5.2.2 実験結果

オープンソースの Web アプリケーションにおいて、手動で Session Fixation と CSRF を行うことによる脆弱性検査の結果と、提案システムによる脆弱性検査の結果を表2に示す。

表2より、全ての Web アプリケーションで手動による脆弱性検査の結果と提案システムによる脆弱性検査の結果は一致した。以上のことから提案手法はオープンソースの Web ア

表 2 提案システムによる脆弱性検査と手動による脆弱性検査の結果

脆弱性	オープンソースの Web アプリケーション		検査対象とする機能	手動	提案システム
	脆弱性	脆弱性			
Session Fixation	Mambo 4.6.2		ログイン機能	脆弱性あり	脆弱性あり
	Joomla 1.0.9		ログイン機能	脆弱性あり	脆弱性あり
	phpBB 2.0.12		ログイン機能	脆弱性あり	脆弱性あり
	phpNuke 7.0		ログイン機能	脆弱性なし	脆弱性なし
CSRF	phpBB 2.0.12		メール送信機能	脆弱性あり	脆弱性あり
			メール削除機能	脆弱性あり	脆弱性あり
	phpNuke 7.0		メール送信機能	脆弱性あり	脆弱性あり
			メール削除機能	脆弱性あり	脆弱性あり

プリケーションに対して脆弱性を検査できたといえる。

6. 関連研究

本章では、対象の Web アプリケーションに攻撃を仕掛けて脆弱性を検査する既存の研究やセッション管理の脆弱性を利用した攻撃を防御する研究について説明するとともに、本研究との相違点を示す。

Sania [7] は対象の Web アプリケーションに SQL インジェクション脆弱性の有無を自動的に検査するシステムである。SQL インジェクションは Web アプリケーションを通して SQL クエリに特定の文字列を挿入することでデータベースへの無制限のアクセスを行う攻撃である。Sania は対象の Web アプリケーションの HTTP リクエストや SQL クエリから SQL インジェクション脆弱性を自動的に識別し、脆弱性を利用した攻撃を生成する。そして攻撃を実行し、Web アプリケーションの脆弱性を検査する。開発者は一般クライアントとして Web アプリケーションに対して無害な操作を行う。Sania は、開発者が送信したこの無害なリクエストと Sania が送信した攻撃リクエストから発行される SQL クエリを比較して攻撃が成功したかを解析する。Sania は開発中の Web アプリケーションにデバッグとして使用される。この研究は SQL インジェクション攻撃を対象としているので本研究とは異なり、本研究はセッション管理の攻撃を対象としている。

Secubat [8] は対象の Web アプリケーションに SQL インジェクションと Cross Site Scripting (XSS) 脆弱性の有無を自動的に検査するシステムである。XSS は攻撃者が Web ページの脆弱性を利用し、悪意のあるスクリプトをフォームから Web ページ内に埋め込み、Web ページの訪問者のブラウザ上でそのスクリプトを実行させる攻撃である。Secubat は、ク

ロールによりさまざまな Web アプリケーションのページを集める。そしてページにフォームがあった場合、フォームに SQL インジェクションや XSS を引き起こす値を入力し、攻撃を実行する。Web アプリケーションからのレスポンスを解析し脆弱性を検査する。Secubat は Web ページの収集から脆弱性の解析までを自動で行う。Secubat は稼働中の Web アプリケーションに使用される。この研究は SQL インジェクションや XSS を対象とし、また稼働中の Web アプリケーションを対象としている点で本研究とは異なる。本研究はセッション管理の脆弱性を対象とし、開発時の Web アプリケーションを対象としている。

Preventing CSRF Attacks [9] は、既存の対策手法のトークンを利用した CSRF を防御するシステムである。CSRF を防ぐためのトークンを管理するメカニズムを Web アプリケーションから分離して実装を行い、多くの Web アプリケーションに適応することができるようにする。トークンを管理するメカニズムを Web アプリケーションとクライアントの間にプロキシとして設置する。このプロキシがトークンの管理を行い、Web アプリケーションがサービス提供時にプロキシが HTTP リクエストとレスポンスに自動的にトークンの処理を行い、CSRF の攻撃を防御する。この研究は攻撃を防御する事が目的である点が本研究とは異なる。本研究はセッション管理の脆弱性の検査を目的としている。

7. 今後の課題

現状、本システムでは検査を行えない Web アプリケーションがある。まず、全てのページで URL が同じ Web アプリケーションに対して提案システムは CSRF の検査を行うことができない。これは攻撃対象とするページの URL を識別できないためであり、このような Web アプリケーションの場合でも、CSRF の検査を行うことができるようにする。

さらに現状では、提案システムの攻撃時に攻撃者と被害者の IP アドレスが同じ状態で攻撃を行っているため、攻撃環境をより再現するために攻撃者と被害者の IP アドレスを変えて攻撃を行うことができるようにする。この他にも、オープンソースの Web アプリケーションで実験を行うにつれて様々な課題が出てくると考えられるので、それらに対応させていくことでより多くの Web アプリケーションで脆弱性の検査を行えるようにする。

8. おわりに

稼働中の Web アプリケーションにはセッション管理の脆弱性を持つ Web アプリケーションが存在する。本研究は実際に Web アプリケーションに攻撃を仕掛け、Web アプリケーションの挙動を調べることで、自動的にセッション管理の脆弱性を検査する手法を提

案した。本研究の利点としては、対象の Web アプリケーションにセッション管理の脆弱性があるか、Web アプリケーションに施したセッション管理の脆弱性に対する対策が攻撃を防御するかを検査することを自動で行えることである。提案手法では、Session Fixation や CSRF を行うために攻撃手順を生成し、攻撃者と被害者に攻撃手順通りにリクエストを送信させ、攻撃結果の解析を行いセッション管理の脆弱性を検査する。提案手法を実装し、異なるセッション管理を施した自作の Web アプリケーションとオープンソースの Web アプリケーションを用いて実験を行った結果、手動でセッション管理の脆弱性の検査を行った場合と提案システムでセッション管理の脆弱性の検査を行った場合で全ての検査結果が一致した。これにより提案システムで正しくセッション管理の脆弱性の検査が行えることがわかった。

参考文献

- 1) Kolsek, J.: Session fixation vulnerability in web-based applications. <http://www.acrossecurity.com/papers.html>.
- 2) Shiflett, C.: Security Corner: Cross-Site Request Forgeries. <http://shiflett.org/articles/cross-site-request-forgeries>.
- 3) WhiteHatSecurity: WhiteHat Website Security Statistics Report. <http://www.whitehatsec.com/home/resource/stats.html>.
- 4) Kosuga, Y.: Amberate. <http://www.sslab.ics.keio.ac.jp/yuji/amberate/index.html>.
- 5) SecurityFocus: Security Focus. <http://www.securityfocus.com/>.
- 6) NationalVulnerabilityDatabase: National Vulnerability Database. <http://web.nvd.nist.gov/>.
- 7) Kosuga, Y., Kono, K., Hanaoka, M., Hishiyama, M. and Takahama, Y.: Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection, *Annual, Computer Security Applications Conference (ACSAC '07)*, pp.107–117 (2007).
- 8) Kals, S., Kirda, E., Kruegel, C. and Jovanovic, N.: SecuBat: a web vulnerability scanner, *Proceedings of the 15th international conference on World Wide Web (WWW '06)*, pp.247–256 (2006).
- 9) Jovanovic, N., Kirda, E. and Kruegel, C.: Preventing Cross Site Request Forgery Attacks, *Securecomm and Workshops, 2006(Securecomm '06)*, pp.1–10 (2006).