セッション管理の脆弱性検査の自動化

高松 勇輔 †　　小菅 祐史 †　　河野 健二 †‡

† 慶應義塾大学大学院理工学研究科　　‡CREST/JST
{yusuke,yuji,kono}@sslab.ics.keio.ac.jp

あらまし　ログイン機能などを持つウェブアプリケーションはユーザ情報等の管理のためにセッションを利用している．しかしセッション管理に脆弱性があると，攻撃者はその脆弱性を利用してユーザのセッションを乗っ取ることができる．このような脆弱性を取り除くには，脆弱性検査を行う必要がある．しかし脆弱性検査を行うには攻撃手法に関する詳細な知識や多くの労力を必要とするため，全ての開発者が脆弱性検査を行うことは難しい．本論文ではセッション管理の脆弱性検査を自動化する手法を提案する．提案手法によって詳細な知識や多くの労力なく検査できる．実際に脆弱性検査を行った結果，6 つのオープンソースのウェブアプリケーションで検査を行うことができた．

Automated Testing of Session Management Vulnerabilities

Yusuke Takamatsu†　　Yuji Kosuga†　　Kenji Kono †‡

†Graduate School of Science and Technology, Keio University　　‡CREST/JST
{yusuke,yuji,kono}@sslab.ics.keio.ac.jp

Abstract　Many web applications employ session management to keep track of the visitor's activity. Attackers can hijack a user's session by exploiting session management vulnerabilities. For eliminating session management vulnerabilities, it is important to test web applications for these vulnerabilities. However, it is difficult for the developers to test web applications without the detailed knowledge about the vulnerabilities. We propose a technique that automatically tests web applications for session management vulnerabilities. By using this technique many developers can test web applications without manual testing. Our experiments demonstrate that our technique can detect the vulnerabilities of real web applications.

1　はじめに

ログインや商品購入などの機能を持つウェブアプリケーションはセッションを利用している．セッションとは，各ユーザのログイン状態やページ遷移の状態等の情報である．それぞれのユーザはセッション ID によってウェブ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ウェブ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，攻撃者が他のユーザのセッションを乗っ取ることで，そのユーザになりすますことができる．

セッションを悪用した攻撃として，Session Fixation [1] や Cross-Site Request Forgery（CSRF）

[2] がある．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リクエストを被害者のセッション ID を使ってウェブアプリケーションに送信する攻撃である．

これらの攻撃に対する対策はすでに知られているにも関わらず，多くのウェブアプリケーションにセッション管理の脆弱性が残っている．WhiteHat Security[3] によれば，14 %のウェブアプリケーションに Session Fixation 脆弱性が，24 %のウェブアプリケーションに CSRF 脆弱性がある．

脆弱性のないウェブアプリケーションを作成するためには，開発段階で脆弱性検査を十分に行わなければならない．その方法のひとつは開発者が実際に攻撃を行い，脆弱性の有無を検査する方法である．しかしそれには攻撃についての詳細な知識や経験が必要であり，多くの労力と時間も要する．

本論文ではセッション管理の脆弱性検査を自動化する機構を提案する．提案機構は，ウェブアプリケーション毎にセッション管理の脆弱性を突く攻撃を自動的に生成する．そして自動的にウェブ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．このように脆弱性検査を自動化することで，多くの開発者がウェブアプリケーションの開発段階に脆弱性検査を行えるようになる．

提案機構の有効性を示すために，6 つのオープンソースのウェブアプリケーションで実験を行った．実験では，提案機構を用いてセッション管理の脆弱性検査を行った．そして手動で攻撃とコード解析を行い，提案機構で検査した結果が正しいことを確認した．その結果，提案機構で行った検査結果が正しいことを確認した．

2 セッション管理の脆弱性

2.1 Session Fixation とその対策

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．

Session Fixation では，最初に攻撃者がウェブアプリケーションにログインする．そして攻撃者はウェブ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，ウェブアプリケーションは攻撃者を被害者とみなすため，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．

Session Fixation の対策には，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，ウェブアプリケーションは新たなセッション 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 CSRF とその対策

CSRF (Cross-Site Request Forgery) は，攻撃者が用意した悪意あるリクエストを，被害者がウェブアプリケーションに送るように仕向ける攻撃である．これにより攻撃者は被害者に意図した操作を行わせることができる．

CSRF では，最初に被害者がウェブ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．その結果，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，被害者は最初にログインしたウェブアプリケーションに，攻撃者が作成した悪意あるリクエストを送信する．被害者のブラウザは Cookie により自動的にセッション ID を付与して送信するため，送られてきたリクエストを被害者からのリクエストとして処理する．

CSRF の対策には，トークンを利用して攻撃を防ぐ手法がある．トークンは，送信さ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．ユーザが重要な処理 (ログインや商品購入) を行う場合，リクエストにトークンを埋め込んで送信する．このトークンはウェブアプリケーションがブラウザに返す URL 等に埋め込んでおく．リクエストに正しいトークンが含まれているかどうかを確認することで，ウェブアプリケーションはそのリクエストが正規のリクエストか攻撃者の偽造したリクエストかを識別することができる．

また，セッション ID の伝播に Cookie ではなく URL Rewriting か Hidden Field を利用すると CSRF を防止できる．URL Rewriting か Hidden Field でセッション ID を伝播する場合，攻撃者は被害者のセッション ID を含んだリクエストを偽造する必要ある．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，CSRF を実行することができない．

3　提案手法

このような対策手法が存在するにも関わらず，多くのウェブアプリケーションにセッション管理の脆弱性が残っている．セッション管理の脆弱性検査を行うには，実際に攻撃を仕掛ければよい．しかし実際に攻撃を行うにはブラウザによって隠匿されているセッション ID を獲得したり，攻撃用ページを作成しなければいけない．

そこでセッション管理の脆弱性検査を自動化する機構を提案する．提案機構は，診断対象のウェブアプリケーション毎にセッション管理の脆弱性を突く攻撃を自動的に生成する．そして提案機構は，ウェブアプリケーションに生成した攻撃を仕掛け，その攻撃結果を解析して自動的に脆弱性検査を行う．

3.1　攻撃の生成

Session Fixation　Session Fixation の脆弱性検査するためには，以下の一連の操作を行う必要がある．図 1 を用いて，説明する．1) 攻撃者がログインページにアクセスする．2) 攻撃者はログインするためのリクエスト内に存在するユーザ名とパスワードのリクエストパラメータの値に，攻撃者のユーザ名とパスワードを挿入し，ログインリクエストを送信する．3) 攻撃者

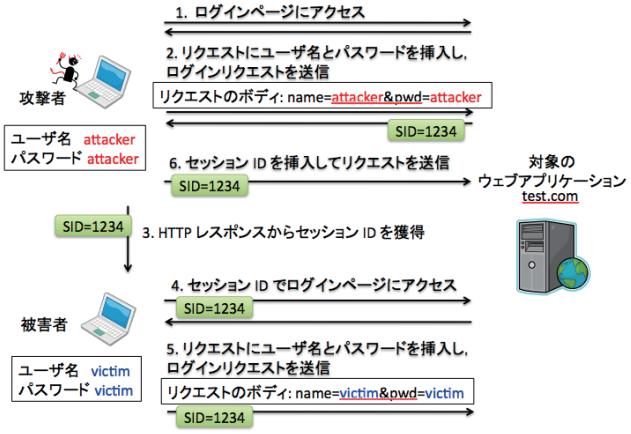

図 1: Session Fixation

は HTTP レスポンスからセッション ID を獲得する．4) 被害者は攻撃者が獲得したセッション ID をリクエストに挿入して，ログインページにアクセスする．5) 被害者は攻撃者と同様にログインするためのリクエスト内に被害者のユーザ名とパスワードを挿入し，ログインリクエストを送信する．6) 攻撃者は，被害者が使用したセッション ID をリクエストに挿入して，リクエストを送信する．

これらの操作に必要となる HTTP リクエストやログインの操作手順はウェブアプリケーションに固有であるため，提案機構が自動的に生成することは難しい．そこで脆弱性検査を行う前段階として提案機構の利用者がログインの操作を行い，検査対象のウェブページの URL やログインの操作手順等の情報を取得する．

利用者がこの操作を行うことによって，ブラウザとウェブアプリケーション間で送受信される HTTP リクエストやレスポンス，クッキー等の情報を取得することができる．そのため，リクエストヘッダやボディ内のリクエストパラメータ，セッション ID の伝播方法，ログインするためのリクエストの情報を取得できる．リクエストヘッダやボディ内のリクエストパラメータは，図 1 の 2) と 5) において，ユーザ名とパスワードをリクエストに挿入するために必要である．セッション ID の伝播方法は，図 1 の 3) において，セッション ID を獲得するために必要であり，図 1 の 4) と 5) と 6) において，セッション ID をリクエストに挿入するために必要である．ログインするためのリクエスト (ログ

インリクエスト) は，提案機構の利用者が行ったログイン操作の中でログインリクエストがどれかを特定し，図 1 の 2) と 5) において，特定したリクエストに攻撃者と被害者のユーザ名とパスワードを挿入するために必要である．

上記の必要な情報は獲得した HTTP メッセージから自動的に抽出する．リクエストヘッダやボディ内のリクエストパラメータの場合，提案機構は，リクエストから獲得する．セッション ID の伝播方法の場合，提案機構は，レスポンスヘッダの Cookie，レスポンス内の URL，レスポンス内のフォームにある Hidden Field に対してセッション ID の名前が存在するかを調べることでセッション ID がどの方式で伝播しているかわかる．ログインリクエストの場合，提案機構は，まず利用者が発行したリクエストが GET か POST かどうか識別する．そして POST の場合，リクエストを発行したレスポンス内のどの入力フォームから送られたかを判断する．そのために，リクエストのボディ内にあるリクエストパラメータとレスポンス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，その入力フォームからリクエストが発行されたと判断する．その入力フォーム内にあるパラメータの一つの type が password の場合，そのリクエストがログインリクエストであると判断する．

また提案機構は，セッション ID のパラメータの名前や攻撃者と被害者のユーザ名とパスワードを抽出できない．しかし開発者である利用者ならば容易にこれらの情報を得られるので，これらの情報については利用者が指定する．

CSRF　CSRF の脆弱性検査するためには，以下の一連の操作を行う必要がある．1) 被害者がログインページにアクセスする．2) 被害者はログインするためのリクエスト内に存在するユーザ名とパスワードのリクエストパラメータの値に，被害者のユーザ名とパスワードを挿入し，ログインリクエストを送信する．3) 被害者は HTTP レスポンスからセッション ID を獲得する．4) 被害者は，利用者が検査したい機能の URL をもとに作成した攻撃者によって強要されるリクエストに獲得したセッション ID を挿入して，リクエストを送信する．

これらの操作を行うためには，HTTP リクエストやログインを行うための操作手順，利用者が検査したい機能の操作手順，セッション ID の伝播方法，ログインするためのリクエストなどの情報が必要である．提案機構では，脆弱性検査を行う前段階として利用者がログインの操作を行った後で，検査したい機能の操作を行うことで，HTTP リクエストやログインを行うための操作手順，利用者が検査したい機能の操作手順の情報を獲得する．

そして Session Fixation の場合と同様に HTTP リクエストやレスポンス，クッキー等の情報を取得できる．そのため，リクエストヘッダやボディ内のリクエストパラメータ，セッション ID の伝播方法，ログインするためのリクエストの情報を取得できる．リクエストヘッダやボディ内のリクエストパラメータは，2) において，ユーザ名とパスワードをリクエストに挿入するために必要である．セッション ID の伝播方法は，3) において，セッション ID を獲得するために必要であり，4) において，セッション ID をリクエストに挿入するために必要である．ログインするためのリクエストは，2) において，そのリクエストに被害者のユーザ名とパスワードを挿入するために必要である．

そして Session Fixation の場合と同様の方法で上記の必要な情報を自動的に抽出する．

また提案機構は，セッション ID のパラメータの名前や被害者のユーザ名とパスワード,，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る），検査したい機能の URL を抽出できない．しかし開発者である利用者ならば容易にこれらの情報を得られるので，これらの情報については利用者が指定する．

3.2　攻撃結果の検証

提案機構は，攻撃結果として得られたレスポンスを調べて攻撃が成功したかを検証する．Session Fixation の場合，攻撃者が被害者のセッションを乗っ取れたかどうかを調べればよい．つまり得られたレスポンスが被害者のみが獲得できるレスポンスであれば攻撃は成功したことになる．

CSRF の場合，被害者が送信させられた偽造リクエストがウェブアプリケーションに実行されたかを調べればよい．つまり得られたレスポンスが偽造リクエストを実行した時のレスポンスであれば攻撃は成功したことになる．

よって Session Fixation の場合は，被害者のログイン時にのみページに現れる文字列が，CSRF の場合は，商品購入やメッセージ送信などが実行された時にのみページに現れる文字列がレスポンス内にあれば攻撃が成功したことになる．例えば，”Thank you, victim（被害者名）”，”Welcome, victim（被害者名）”などの文字列である．提案機構では，これらの文字列が得られたレスポンス内にあるときに攻撃が成功し，そのウェブアプリケーションに脆弱性があると判断する．これらの文字列について利用者が指定する．

3.3 実装

提案機構を Amberate[4] の脆弱性検査用プラグインとして実装した．Amberate はウェブアプリケーションの脆弱性自動検査を行うフレームワークで，ウェブアプリケーションの動作を監視し，得られた情報をプラグインに渡す．提案機構では HTTP リクエストとレスポンスを監視する必要があるため，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て実装した．

4 実験

提案手法の有用性を示すために，自作のウェブアプリケーションとオープンソースのウェブアプリケーションを用いて検証を行った．

4.1 マイクロベンチマーク

意図的に脆弱性を含めた自作のウェブアプリケーションを用いて実験を行い，正しく脆弱性を検査できるかを検証した．自作ウェブアプリケーションが行うセッション管理手法には，Session Fixation や CSRF の脆弱性を持つ 4 つの手法，また脆弱性を持たない 3 つの手法を実装した．

さらに 1 から 4 の Session Fixation に関するセッション管理について，セッション ID の伝播法として URL Rewriting と Cookie，Hidden Field をそれぞれ実装し，実験を行った．また 5 から 7 の CSRF 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 のみを実装し，実験を行った．

表 1: 提案機構による脆弱性検査の結果

対策の有無	脆弱性の有無		
	URLRewriting	Cookie	HiddenField
1.SessionFixation 対策なし	SessionFixation	SessionFixation	SessionFixation
2.SessionFixation 対策あり (セッション ID の変更)	脆弱性なし	脆弱性なし	脆弱性なし
3.SessionFixation 対策あり (セッション ID の発行方法 *)	脆弱性なし	脆弱性なし	脆弱性なし
4. 不完全な SessionFixation 対策あり (3 の対策にミスがある場合 **)	SessionFixation	SessionFixation	SessionFixation
5.CSRF 対策なし	-	CSRF	-
6.CSRF 対策あり (トークンでのユーザ識別)	-	脆弱性なし	-
7. 不完全な CSRF 対策あり (6 の対策にミスがある場合 ***)	-	CSRF	-

* ログイン後のみセッション ID を発行する．ログイン時，必ずセッション ID を発行する．

** ログイン後のみセッション ID を発行する．セッション ID を持つユーザのログイン時，そのセッション ID を継続して使用する．

*** リクエストにトークンが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規のリクエストとする．

表 2: 提案機構と手動による脆弱性検査の結果

アプリケーション名	検査対象の機能	脆弱性の有無	
		攻撃と解析の結果	提案機構
Mambo 4.6.2	ログイン機能	SessionFixation	SessionFixation
Joomla 1.0.9	ログイン機能	SessionFixation	SessionFixation
phpBB 2.0.12	ログイン機能	SessionFixation	SessionFixation
phpNuke 7.0	ログイン機能	脆弱性なし	脆弱性なし
phpBB 2.0.12	メール送信機能	CSRF	CSRF
	メール削除機能	CSRF	CSRF
phpNuke 7.0	メール送信機能	CSRF	CSRF
	メール削除機能	CSRF	CSRF

提案機構による脆弱性検査の結果を表 1 に示す．表 1 より，提案機構による脆弱性検査の結果が正しいことがわかる．以上のことから提案手法は自作のウェブ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．

4.2 マクロベンチマーク

オープンソースのウェブアプリケーションを用いた実験も行った．ウェブアプリケーションの脆弱性報告サイト [5] [6] などで Session Fixation や CSRF の脆弱性を持つと報告されているものを対象とした．ただし phpNuke7.0 は脆弱性の修正がなされたものである．

これらのウェブアプリケーションに対し，脆弱性の有無を手動での攻撃とソースコード解析

によって確認した．提案機構による脆弱性検査の結果を表 2 に示す．表 2 より，全てのウェブ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機構による脆弱性検査の結果は一致することが確認できた．

この調査の結果，Mambo, Joomla, phpBB, phpNuke がセッション ID と IP アドレスを利用してユーザ識別を行っていた．ウェブアプリケーションがセッション ID と IP アドレスを利用してユーザ識別を行うことで，Session Fixation を防ぐことができる．攻撃者と被害者のセッション ID が一致していても攻撃者と被害者の IP アドレスが異なる場合に，攻撃者と被害者を識別できるからである．

しかしこの方法は，Session Fixation の完全な対策ではなく，IP アドレスを利用した Session Fixation 対策は緩和措置であると言われている [1]．例えば，イントラネットなどで複数のユーザが同じ IP アドレスでウェブアプリケーションと通信を行う場合に，攻撃者と被害者の IP アドレスが一致するため，攻撃が成功する．

5 関連研究

Sania [7] は SQL インジェクション脆弱性の有無を自動的に検査する機構である．Sania は対象のウェブ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が疑われるケースを自動的に識別し，それを利用した攻撃を生成する．

Secubat [8] は 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機構である．Secubat は，ページ内の入力フォームに SQL インジェクションや XSS を引き起こす値を入力し，ウェブ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 はウェブページの収集から脆弱性の解析までを自動で行う．Secubat は SQL インジェクションや XSS を対象としいる点で本論文の提案機構とは異なる．

Preventing CSRF Attacks [9] は，CSRF を防ぐためのトークン管理メカニズムをウェブアプリケーションから分離する機構である．トークン管理を分離して実装することで，多くのウェブアプリケーションに適応できるようにしている．この機構によって容易に CSRF を対策を施せる．しかし設定ミスなどによって脆弱性が残こる可能性がある．よって本論文の提案機構で脆弱性検査を行う必要がある．

6 まとめ

セッション管理の脆弱性を持つウェブアプリケーションが多く存在する．本論文では実際にウェブアプリケーションに攻撃を仕掛けそ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．オープンソースの実用ウェブアプリケーションを用いて実験を行った結果，提案機構でセッション管理の脆弱性検査を行った場合と手動で脆弱性検査を行った場合で全ての検査結果が一致した．これにより提案機構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確認できた．

参考文献

[1] J. Kolsek. Session fixation vulnerability in web-based applications. http://www.acrossecurity.com/papers.html.

[2] Shiflett Chris. 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] 小菅 祐史, 河野 健二. Amberate: Web アプリケーションの脆弱性自動検出 フレームワーク. 日本ソフトウェア科学会 コンピュータソフトウェア, 2011.

[5] SecurityFocus. SecurityFocus. http://www. securityfocus.com/.

[6] NationalVulnerabilityDatabase. National Vulnerability Database. http://web.nvd.nist.gov/.

[7] Yuji Kosuga, Kenji Kono, Miyuki Hanaoka, Miho Hishiyama, and Yu Takahama. Sania: Syntactic and semantic analysis for automated testing against SQL injection. In Annual Computer Security Applications Conference (ACSAC ’07), pp. 107-117, 2007.

[8] Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic. Secubat: a web vulnerability scanner. In Proc of Int’l Conf on World Wide Web(WWW ’06), pp. 247-256, 2006.

[9] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. Preventing cross site request forgery attacks. In Proc of Securecomm(Securecomm ’06), pp. 1-10, 2006.