ウェブアプリケーションのロジックに依存する攻撃の自動生成による脆弱性検査に関する研究

2015 年度

高　松　勇　輔

学位論文　博士（工学）

ウェブアプリケーションのロジックに依存する攻撃の自動生成による脆弱性検査に関する研究

2015年度

慶應義塾大学大学院理工学研究科

高松 勇輔

ウェブアプリケーションのロジックに依存する攻撃の自動生成による脆弱性検査に関する研究

高松 勇輔

論文要旨

ウェブアプリケーションの脆弱性は深刻な不正攻撃につながっており，不正攻撃によってクレジットカード番号などの個人情報が流出した事例などが多く報告されている．さらに，77% 以上のウェブアプリケーションに少なくとも 1 つの脆弱性があると報告されており，開発段階においてウェブアプリケーションの脆弱性を検査する必要がある．実際に 85% の企業がウェブアプリケーションのリリース前に脆弱性の検査を行っている．それにもかかわらずウェブアプリケーションに脆弱性が残っている主な要因は次の 3 点である．1 点目は，開発者が脆弱性や攻撃に関する知識を持たないことである．2 点目は，脆弱性や攻撃についての知識があっても防御手法の回避法について知識がないことである．3 点目は，開発者に脆弱性や攻撃，防御手法，その回避手法などについての十分な知識はあるものの防御手法の実装に不具合が残ることである．

本論文では，ウェブアプリケーションのロジックに依存する攻撃を自動的に実行することで脆弱性を検査する手法を提案する．ウェブアプリケーションのロジックとは，ウェブアプリケーションが持つ機能 (例: ログイン機能やメッセージ送信機能) を動作させるために必要な手順や情報のことである．提案機構は，こうしたロジックに関する情報を利用してウェブアプリケーションの機能を動作させることで，ロジックに依存する攻撃を実行する．提案機構と同様にウェブアプリケーションへの攻撃を実行することで脆弱性を検査する既存手法があるものの，ウェブアプリケーションのロジックに依存する攻撃を実行することができない．なぜなら，このような攻撃の実行に必要となるロジックに関する情報を収集できないためである．提案機構はウェブアプリケーションの開発段階における利用を想定することで，開発者からこのロジックに関する情報を獲得する．例えば，ウェブアプリケーションのロジックに依存する Cross-Site Request Forgery (CSRF) という攻撃を実行するためにウェブアプリケーションにログインする必要があり，開発者からログインのロジックに関する情報を獲得する．提案機構は，このロジック

に関するいくつかの情報を自動的に収集するためにテストフェーズにおいて開発者が行うウェブアプリケーションの動作確認を監視する．これは，ロジックに関する情報を要求することで開発者にかかる負担を軽減するためである．提案機構がウェブアプリケーションのロジックに依存する攻撃を実行することで，これまで攻撃を用いて自動的に検査できなかった脆弱性が検査できるようになる．提案機構が自動的に検査を行うことで，脆弱性や攻撃，防御手法，その回避手法などの知識がない開発者でもこの脆弱性を検査することができる．さらに，提案機構は実際に攻撃を実行することで防御手法の不具合も検査することができる．

本手法の有用性を示すために，既存手法では実行できない CSRF と session fixation，visual clickjacking という攻撃に本手法を適用する．提案機構はこれらの攻撃を実行するために開発者からロジックに関する情報を獲得する．CSRF を実行するためにログインのロジックと検査対象の機能のロジックに関する情報を獲得し，session fixation を実行するためにログインのロジックとログアウトのロジックに関する情報を獲得し，visual clickjacking を実行するに検査対象の機能のロジックに関する情報を獲得する．提案機構が実行するこれらの攻撃で脆弱性を検査できることを確認するために，さまざまなサイトで利用されているオープンソースのウェブアプリケーションを対象に脆弱性検査を行った．その結果，提案機構が検査した機能に残る脆弱性については全て検出することができた．実際に提案機構は，5 つのウェブアプリケーションに残った 11 個の CSRF の脆弱性と 6 個の session fixation の脆弱性を検出し，4 つのウェブアプリケーションに残った 26 個の visual clickjacking の脆弱性を検出した．以上の結果から，本手法は脆弱性を検査するためにウェブアプリケーションのロジックに依存する攻撃を自動的に実行できることがわかった．

A Study on Vulnerability Detection of Web Applications by Automatic Generation of Logic-Aware Attacks

Yusuke Takamatsu

Abstract

Web application vulnerabilities have become an attractive target for attackers. The vulnerabilities could allow the attackers to steal personal information (e.g., credit card number) and force users to perform unintended financial transactions (e.g., money transfer). Three security vendors reported that more than 77 percent of web applications had at least one vulnerability. To eliminate the vulnerabilities, developers should check for them in the web applications during the development phase. WhiteHat Security reported that 85 percent of organizations perform security testing of their web applications. Nevertheless, many web applications are vulnerable in the wild. There are three causes that make the web applications vulnerable. First, the developers implement no countermeasure against attacks in the web applications because they do not have knowledge on the attacks and the vulnerabilities. Second, incomplete defenses are implemented because the developers do not have knowledge on evasion techniques of the defenses. Third, the web applications remain vulnerable due to implementation mistakes in the defenses.

This dissertation presents automatic detection of vulnerabilities in web applications by performing logic-aware attacks. The logic here means a procedure to operate functions in the web applications (e.g., login procedure). Our technique performs the logic-aware attacks by operating the functions with the logic. Although existing techniques perform pseudo attacks, they cannot do the logic-aware attacks. This is because the existing techniques cannot obtain information on the logic. Our technique obtains the information on the logic from web application developers because our technique is designed to be used in the development phase. For example, to perform Cross-Site Request Forgery (CSRF) as the logic-aware attack, our technique obtains the logic to log in the target applications. Our system automatically collects some pieces of information on the logic while the developers confirm the target applications work well. This is because it releases the developers from the burden of providing the information on the logic. Our technique automatically detects vulnerabilities which the existing techniques cannot do because it can perform the logic-aware attacks. The developers can detect

the vulnerabilities with our technique even if they are not familiar with the attacks, the vulnerabilities, the defenses, and the evasion techniques of the defenses. Our technique also detects the incomplete defenses due to implementation mistakes because it carries out the attacks.

To demonstrate the usefulness of our technique, it has been applied to CSRF, session fixation, and visual clickjacking which the existing techniques cannot perform. Our technique obtains the information on the logic from the developers to perform these attacks. To perform CSRF, session fixation, and visual clickjacking, our technique obtains the logic to log in and to log out, and the logic to manipulate target functions (e.g., a function to post messages). In experiments our technique detected CSRF vulnerabilities, session fixation vulnerabilities, and visual clickjacking vulnerabilities in real-world web applications. Our experimental results demonstrate that our technique can detect 11 CSRF vulnerabilities and 6 session fixation vulnerabilities in 5 real-world web applications, and 26 visual clickjacking vulnerabilities in 4 real-world web applications. These results also show that our technique can perform logic-aware attacks to check for vulnerabilities.

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景

攻撃者がウェブアプリケーションに残った脆弱性を突いた攻撃を行うことで，ウェブアプリケーションのユーザにさまざまな被害が生じる．例えば，クレジットカード番号や住所などの個人情報の漏洩や送金などの金銭的な処理の強要，アカウントを乗っ取られることによるなりすましなどの被害がある．稼働中のウェブアプリケーションの 77% 以上に少なくとも 1 つの脆弱性があると報告されている [1,2,3].

実際にこの脆弱性を突いた攻撃によって，ウェブアプリケーションを運営する企業やユーザがさまざまな被害を被っている．2014 年には，42 万以上のウェブアプリケーションが一つのハッカーグループに 12 億件のログイン情報を盗まれたという報告がある [4]．他にも，著名な企業が運営するウェブアプリケーションも例外なく被害を受けている．例えば，Yahoo! は 45 万人以上のユーザ名とパスワードが記述されたファイルを漏洩した [5]．これらのユーザ名とパスワードは，Yahoo! のものだけでなく，Gmail や Hotmail，MSN，Live.com などのものも含まれている．また，Linkedin は 650 万人以上のログイン情報を漏洩し，約 100 万ドルの被害を被った [6]．さらに，その後セキュリティを向上させるために 2〜300 万ドルの費用を負担した．他にも Sony Pictures は 100 万人以上の個人情報を漏洩した [7]. この個人情報には，パスワードだけでなくメールアドレスや住所，誕生日などが含まれる．さらに，Sony Pictures を攻撃したハッカーグループは，データベース内のパスワードを含むすべての管理者情報にも不正にアクセスできたことを明らかにしている．上記に示した被害以外にもさまざまなウェブアプリケーションがこのような攻撃によって被害を被っている [8, 9, 10].

開発者はこのような攻撃に対する防御手法をウェブアプリケーションに実装する必要がある．しかし，攻撃毎に防御手法があるために，さまざまな防御手法を実装することが開発者に求められる．例えば，SQL インジェクションを防ぐために SQL クエリを改竄する入力値を検出する [11]．Cross-Site Scripting (XSS) を防

ぐために攻撃者によるページへのスクリプトの挿入を検出する [12, 13, 14, 15, 16]. Cross-Site Request Forgery (CSRF) を防ぐために攻撃者によって強要されたリクエストを識別する [17, 18, 19]．Session fixation を防ぐためにログイン時に攻撃者によって強要されたセッション ID を変更する [20]．同様に他の攻撃についても防御手法が存在している．

ウェブアプリケーションの開発段階において実装された防御手法が攻撃を防ぐことを検査する必要がある．なぜなら，上記に示したように攻撃毎にさまざまな防御手法があり，すべての防御手法を完璧に実装することは容易ではないためである．さらに，近年ウェブアプリケーションは大規模化・複雑化しているために防御手法の実装を忘れる可能性もある．実際に 85% の企業がウェブアプリケーションのリリース前に脆弱性検査を行っていると報告されている [1].

脆弱性検査が行われているにもかかわらず，稼働中のウェブアプリケーションに脆弱性が残ってしまっている．このようにウェブアプリケーションに脆弱性が残っている主な要因は次の 3 点である．1 点目は，開発者が脆弱性や攻撃に関する知識を持たないために防御手法を実装しないことである．2 点目は，開発者が脆弱性や攻撃についての知識を持っていても防御手法の回避法についての知識を持たないために不完全な防御手法を実装してしまうことである．3 点目は，開発者が脆弱性や攻撃，防御手法，その回避手法に関する十分な知識を持っているものの防御手法の実装に不具合が残ることである．

1 点目の要因において，防御手法を実装するために開発者は脆弱性や攻撃に関する知識が求められる．しかし，攻撃にもさまざまな種類があり，新しい攻撃も発見され続けているために，開発者が全種類の攻撃に精通していることは難しい．OWASP は，脆弱性を突いた攻撃を 63 種類に分類している (実際には 66 種類に分類しているが 4 種類は同じ攻撃であったために 1 つにまとめている) [21]．また，Clickjacking [22] や mXSS [23]，NoSQL インジェクション [24]，Session Puzzles [25] などの新しい攻撃も発見されている．

ここで，2 点目の要因で具体例を示すために SQL インジェクションについて説明する．SQL インジェクションは，ウェブアプリケーションが生成する SQL クエリを改竄する文字列を挿入することでデータベースに不正にアクセスする攻撃である．この攻撃を用いることでデータベース内にある情報の漏洩や改竄などを引き起こすことができる．例えば，ウェブアプリケーションがユーザからの入力を利用した次のような SQL クエリを発行すると仮定する．

```
SELECT * FROM users WHERE name = '{ユーザからの入力}';
```

攻撃者がユーザからの入力として “ ’ or ‘1’ = ‘1’ ” のような文字列を与えることで，users テーブルの全ての情報を入手することができる．これは，“ ’ or ‘1’ = ‘1’ ” の最初のクオートが SQL クエリの name の値を終わらせ，or 以降の文字列によって WHERE 節が常に真となるためである．

2 点目の要因において，適切な防御手法を実装するために開発者は防御手法を回避する手法についての知識が求められる．しかし，防御手法を回避するためのさまざまな手法があり，開発者が全ての手法に精通していることは難しい．

例えば，文字コードを悪用することで，XSS や SQL インジェクションを防ぐ入力検査を回避できる [26]．PHP が提供している addslashes() 関数を SQL インジェクションを防ぐために利用する場合，回避手法を用いることでこの関数は回避できてしまう．この関数はクオートなどの特殊文字の前にバックスラッシュ “\” を追加する．よってこの関数は，SQL インジェクションを引き起こす入力として説明した “ ’ or ‘1’ = ‘1’ ” を “ \’ or \‘1\’ = \‘1\’ ” に変更し，この攻撃を防ぐことができる．しかし，文字コードに Shift_JIS が指定されており，攻撃者によって “ \x95 ’ OR A = A ” が値として入力された場合，addslashes() 関数は回避されてしまう．addslashes() 関数はこの文字列を “ 表 ’ OR A = A ” に変更するために文字列内にクオートが残り，SQL インジェクションが成功してしまう．

他にもブラウザのセキュリティポリシーや XSS フィルタなどを悪用することで，clickjacking を防ぐ frame busting を回避できる [27]．また，防御手法の実装 (例: 簡単な文字列検索) を悪用することで，この防御手法を回避できる [14]．

3 点目の要因において，開発者には防御手法の実装に不具合を残さないことが求められるものの，常に完璧な防御手法を実装することは難しい．実際に，Joomla 3.x (コンテンツマネージメントシステムとして幅広く使われている) は，visual clickjacking を防ぐために X-Frame-Options という防御手法を採用している．しかし，綴りの間違いによってこの X-Frame-Options は動作しないことがわかっている [28]．Joomla の behavior.php において X-Frame-Options が X-Frames-Options (‘Frame’ の後の ‘s’ が必要ない) と実装されており，SAMEORIGIN が SAME-ORIGIN (‘-’ が必要ない) と実装されている．

さらに，フレームワークによって防御手法が提供されていても，その防御手法の適用を忘れることがある．実際に Ruby on Rails が防御手法を提供しているにもかかわらず適用されていないために，実験対象のウェブアプリケーションに脆弱性が残っていた [29]．

1.2 脆弱性検査の自動化

本研究では，ウェブアプリケーションの開発段階において脆弱性を検査する手法を自動化する．脆弱性を検査するために，実際にウェブアプリケーションに攻撃を行い，攻撃に対する挙動を解析する．脆弱性検査が自動化されることによって，開発者は脆弱性や攻撃に関する知識がなくても，脆弱性を検査することができる．さらに，回避手法を用いて攻撃することで不完全な防御手法を検査でき，防御手法の実装に残った不具合も検査できる．以降の項では，脆弱性検査の現状について説明し，現状を考慮した上で本研究の目的を示す．

1.2.1 脆弱性検査の現状

1.1 節でも説明したように，ウェブアプリケーションの開発時において脆弱性検査が行われている．開発者はウェブアプリケーションの脆弱性を検査するために手動での検査を行ったり，自動的に脆弱性を検査する手法や脆弱性スキャナを利用している．手動での検査では，開発者がソースコードをチェックしたり，実行環境においてウェブアプリケーションに攻撃を実行する．しかし，全ての開発者が手動での検査を用いて網羅的に脆弱性を検査することは難しい．なぜなら，開発者には脆弱性や攻撃，防御手法などのさまざまな知識が求められるためである．

より網羅的に脆弱性を検査するために，自動的に脆弱性を検査する既存手法や自動的に検査を行う脆弱性スキャナが利用されている．これらの既存手法は脆弱性の有無を検査するためのものであり，ソースコードの修正や修正箇所の特定は行わない．自動化された脆弱性検査は静的検査と動的検査に大別することができる．

静的検査 [29,30,31] は，利用者から獲得したウェブアプリケーションのソースコードを解析することで脆弱性を検査する．このソースコードの解析において実行可能なパスを検査できるために，静的検査には高いコードカバレッジを実現できるというメリットがある．

しかし，静的検査のデメリットとして次の 2 点が挙げられる．1 点目は，ソースコードを実行していないために実行環境に依存する攻撃に対する検査が困難である．例えば，ブラウザによる文字コードの解釈に依存する攻撃 [32] があり，実行環境でなければ脆弱性の検査が困難である．2 点目は，検査システムの適用範囲がプログラミング言語やフレームワークに依存してしまう．実際に Pixy [30] は，PHP で実装されたソースコードしか検査できず，Rubyx [29] や Doupé らが提案し

た手法 [31] は Ruby on Rails 上に実装されたウェブアプリケーションしか検査できない．

それに対して，動的検査は稼働中のウェブアプリケーションの挙動を解析することによって実行環境に依存する脆弱性を検査する．さらに，稼働中の挙動を解析しているために，プログラミング言語やフレームワークに依存することなくウェブアプリケーションに適用することができる．この動的検査は，レスポンス解析とペネトレーションテストに分類することができる．

レスポンス解析 [33, 34, 35, 36, 37, 38, 39] は，攻撃を行わずに無害なリクエストに対するウェブアプリケーションのレスポンスを解析する．例えば，ユーザを識別するためのセッション ID が第三者によって推測可能である脆弱性を検査する場合，検査システムはウェブアプリケーションにアクセスすることでセッション ID (例: SID=01) を獲得する．そして，セッション ID の長さが 2 桁であることから攻撃者が推測可能であるためにこのセッション ID は脆弱であると判断する．

しかし，レスポンス解析には，攻撃を行わないために検査できる脆弱性が限られるというデメリットがある．例えば，SQL インジェクションの脆弱性を検査する場合，無害なリクエストを送信してもウェブアプリケーションが SQL インジェクションを防ぐために入力確認を行っているか判断できない．

ペネトレーションテスト [40, 41, 42, 43, 44, 45, 46] は，ウェブアプリケーションに対して擬似的な攻撃を行い，攻撃に対するレスポンスを解析することで脆弱性の有無を判断する. 例えば，SQL インジェクションの脆弱性を検査する場合，検査システムはリクエストパラメータに SQL インジェクションを引き起こす文字列を埋め込み，そのリクエストを送信する．このリクエストに対して獲得したレスポンスに SQL の構文エラーが表示されている場合，リクエストに埋め込んだ文字列が SQL の構文を改竄できたことを意味するために脆弱であると判断する．

しかし，既存のペネトレーションテストは，ウェブアプリケーションのロジックに依存する攻撃を実行することができない．ウェブアプリケーションのロジックとは，ウェブアプリケーションが持つ機能 (例: ログイン機能やメッセージ送信機能) を動作させるために必要な手順や情報のことである．このようなロジックに依存する攻撃を実行するためには，ウェブアプリケーションのロジックに関する情報を獲得して，ウェブアプリケーションが持つ機能を正確に動作させる必要がある．しかし，このロジックはウェブアプリケーション毎に異なるために自動的に推測することが難しい．

1.2.2　本研究の目的

本研究の目的は，脆弱性を検査するためにウェブアプリケーションのロジックに依存する攻撃を自動的に実行することである．このような攻撃を実行するために，提案機構は開発者からウェブアプリケーションのロジックに関する情報を獲得する．提案機構は，開発者から獲得した情報をもとにウェブアプリケーションが持つ検査対象の機能を正確に動作させることで攻撃を実行する．

提案機構はウェブアプリケーションの開発段階における利用を想定することで，開発者からウェブアプリケーションのロジックに関する情報を獲得する．例えば，ウェブアプリケーションにログインするための手順やログインするために必要なユーザ名とパスワード，セッション ID を特定するためにセッション ID の名前などの情報を獲得する．開発者であればこれらのロジックに関する情報を与えることは難しくない．なぜなら，これらの情報は開発者が開発したウェブアプリケーションについての情報だからである．

これらの情報を要求することで開発者にかかる負担を軽減するために，テストフェーズにおいて開発者がウェブアプリケーションの動作確認を行う間に提案機構は情報を自動的に収集する．開発者が動作確認を行うために発行したリクエストやレスポンス，ページに対して発行したマウスイベントなどのあらゆる情報を収集する．さらに，提案機構はこれらの情報を解析することで攻撃に必要な情報を獲得する．

このように開発者からロジックに関する情報を獲得することで，提案機構はウェブアプリケーションのロジックに依存するさまざまな攻撃を自動的に実行することができる．提案機構は，このような攻撃を実行するために攻撃の手順に従ってウェブアプリケーションを操作する．この攻撃の手順は提案機構の開発者によって用意されるために，この開発者が攻撃の手順を用意することでさまざまな攻撃を実行することができる．

1.3　本研究の貢献

本研究における貢献は，ペネトレーションテストによって検査できる脆弱性の範囲が拡大することである．その結果，これまで既存のペネトレーションテストを用いて検査できなかった脆弱性を多くの開発者が検査できるようになり，ウェブアプリケーションのセキュリティを向上させることができる．本手法は，既存

のペネトレーションテストが検査できないウェブアプリケーションのロジックに依存するさまざまな攻撃を実行することができる．

本手法がウェブアプリケーションのロジックに依存する攻撃を自動的に実行できることを示すために，CSRF [47] と session fixation [48]，visual clickjacking [22] の 3 種類の攻撃に本手法を適用する．提案機構はこれらの攻撃を実行するために開発者からロジックに関する情報を獲得する. CSRF を実行するために検査対象の機能のロジックとログインのロジックに関する情報を獲得し，session fixation を実行するためにログインのロジックとログアウトのロジックに関する情報を獲得する．さらに，visual clickjacking を実行するために検査対象の機能のロジックに関する情報を獲得する．

提案機構が実行するこれらの攻撃でウェブアプリケーションの脆弱性を検査できることを確認する．そのために提案機構をさまざまなサイトで利用されているオープンソースのウェブアプリケーションに適用した．その結果，いくつかの false positive や検査できない機能はあったものの，提案機構が検査を行った機能の脆弱性について全て検出することができた．実際に提案機構は，5 つのオープンソースのウェブアプリケーションに残った 11 個の CSRF の脆弱性と 6 個の session fixation の脆弱性を検出し，4 つのオープンソースのウェブアプリケーションに残った 26 個の visual clickjacking の脆弱性を検出した．

1.4 本論文の構成

本論文は，次章より次のように構成される．第 2 章で本研究の関連研究をまとめることで，既存手法ではウェブアプリケーションのロジックに依存する攻撃に対する脆弱性を制限なく検査できないことを示す．第 3 章では，このようなロジックに依存する攻撃を自動的に実行するために開発者から情報を獲得するアプローチについて説明する．第 4 章と第 5 章では，本手法の有用性を示すために CSRF と session fixation，visual clickjacking を実行することで脆弱性を検査する手法とその手法の有用性について説明する．第 6 章で本論文をまとめる．

第2章　関連研究

本章では，本研究に関連する研究についてまとめる．1.2.1 に示したように，網羅的に脆弱性を検査するために検査を自動化する既存手法や自動的に検査を行う脆弱性スキャナが利用されている．これらの自動化された脆弱性検査は静的検査と動的検査に大別することができ，これらの既存手法についてまとめる．そして既存手法では，ウェブアプリケーションのロジックに依存する攻撃に対する脆弱性を制限なく検査できないことを示す．

2.1　静的検査

静的検査 [29, 30, 31] は，脆弱性を検査するためにウェブアプリケーションを実行せずに，利用者から獲得したソースコードを解析する．静的検査には，高いコードカバレッジで脆弱性を検査できるメリットがある．なぜなら，ウェブアプリケーションのソースコードを解析することで実行可能なパスを網羅的に検査しているためである．

しかし，静的検査には 2 点のデメリットがあり，1 点目はソースコードを実行しないために実行環境に依存する攻撃に対する脆弱性の検査が困難な点である．実際に実行環境に依存する攻撃として Internet Explorer (IE) による文字コードの解釈を利用した攻撃 [32] がある．攻撃者がページに埋め込んだ文字列を IE に UTF-7 の文字コードで解釈させることで，この文字列は XSS を引き起こすスクリプトとして処理される．このようなブラウザの特徴を悪用した攻撃に対する脆弱性をソースコードの解析によって検査することは難しい．

2 点目は静的検査の適用範囲が対象とするプログラミング言語やフレームワークに依存してしまう点である．Pixy [30] は，XSS の脆弱性を検査するためにデータフロー解析を用いて，外部から入力された文字列がサニタイズされてから出力されることを確認する．しかし，この手法は PHP が提供しているサニタイズを行う関数を利用しているかを確認するために，PHP で実装されたソースコードしか

解析することができず，開発者が実装したサニタイズを行う関数を検査することができない．

Rubyx [29] は，CSRF の脆弱性を検査するために，ウェブアプリケーションのソースコードに対してシンボリック実行を行い，Ruby-on-Rails が提供する防御手法が実装されていることを確認する．この手法は，Ruby-on-Rails を使ったウェブアプリケーションにしか適用できず，開発者が実装した防御手法を検査することができない．

Doupé らが提案する手法 [31] は，Execution After Redirect (EAR) の脆弱性を検査するために，ソースコードから生成した制御フローグラフを解析することで，リダイレクト後に意図しない処理が実行されるかを確認する．しかし，この手法を用いて確認を行えるのは，Ruby on Rails 上に実装されたウェブアプリケーションに対してだけである．なぜなら，この確認は Ruby on Rails が提供する特定の関数を利用しているかを調べているためである．

Rocchetto らが提案する手法 [49] は，ウェブアプリケーションのモデルチェックにより CSRF の脆弱性を検査することで，プログラミング言語やフレームワークに依存することなく検査を行うことができる．しかし，モデルチェック後に脆弱性のないモデルに従ってウェブアプリケーションを実装したとしても，実装時の不具合によって脆弱性が残る可能性があるために，実装後に脆弱性の検査を行う必要がある．

このように既存の静的検査では，検査できる脆弱性が限られ，幅広くウェブアプリケーションに適用することができない．Rubyx [29] や Rocchetto らが提案する手法 [49] はウェブアプリケーションのロジックに依存する攻撃に対する脆弱性を検査できるものの，特定のフレームワークを利用しているウェブアプリケーションにしか適用できず，実装後に脆弱性の検査を行う必要がある．

2.2 動的検査

動的検査は脆弱性を検査するために稼働中のウェブアプリケーションの挙動を解析する．ウェブアプリケーションの挙動を解析しているために，実行環境に依存する脆弱性を検査でき，プログラミング言語やフレームワークに依存することなく幅広いウェブアプリケーションに適用することができる．この動的検査は，レスポンス解析とペネトレーションテストに分類することができ，以降の項でそれ

ぞれの既存手法についてまとめる．

2.2.1 レスポンス解析

レスポンス解析 [33, 34, 35, 36, 37, 38, 39] では，攻撃を行わずに無害なリクエストに対するウェブアプリケーションのレスポンスを解析する．しかし，レスポンス解析は，攻撃を行わないために検査できる脆弱性が限られてしまい，レスポンスから特定の防御手法が実装されていることしか確認することができない．

既存のスキャナ [34, 36, 37, 39] は CSRF を防ぐトークンによる手法がウェブアプリケーションに実装されているかを検査する．この防御手法はレスポンス内のフォームにトークンを含ませるもので，これらのスキャナはフォーム内にトークンが含まれることを確認する．

また，Kumar らの手法 [33] と既存のスキャナ [35,36,37,38,39] は，session fixation を防ぐ手法が実装されているかを検査する．この防御手法はログイン時にセッション ID を変更するもので，これらの手法はセッション ID の変更を確認するためにログイン前後に発行されるセッション ID を比較する．

これらの既存手法は，ウェブアプリケーションのロジックに依存する攻撃に対する脆弱性を検査する．しかし，特定の防御手法の有無を確認しているために他の防御手法が実装されていた場合，正確に検査することができない．さらに，攻撃によって検査を行わないために，特定の防御手法に残った不具合を検査することができない．

2.2.2 ペネトレーションテスト

ペネトレーションテスト [40, 41, 42, 43, 44, 45, 46] は，ウェブアプリケーションに対して擬似的な攻撃を行い，このウェブアプリケーションが発行するレスポンスを解析することで攻撃が成功したかを確認する．ペネトレーションテストは攻撃を行うことによって，特定の防御手法に依存することなく検査することができ，防御手法の不具合も検査することができる．

しかし，既存のペネトレーションテストはウェブアプリケーションのロジックに依存する攻撃を実行できないために，この攻撃に対する脆弱性を検査することができない．ウェブアプリケーションのロジックとは，ウェブアプリケーションが持つ機能 (例: ログイン機能やメッセージ送信機能) を動作させるために必要な

手順や情報のことである．このロジックに依存する攻撃は，検査対象の機能を正確に動作させることで初めて成功する．

既存のペネトレーションテストや既存のスキャナは，攻撃箇所を特定するためにウェブアプリケーションの情報 (ページ内のリンクやフォームの action 属性，タグの source 属性など) を提供するクローラを利用する．このクローラは，ウェブアプリケーションに関する情報を提供するために，ウェブアプリケーションを自動的に探索する．例えば，PAPAS [45] は，クローラから獲得したリクエストやレスポンスを利用してリクエスト内のパラメータを特定し，そのパラメータに攻撃用の変数を挿入することでウェブアプリケーションに攻撃を実行する．

しかし，クローラはウェブアプリケーションのロジックに関する情報を提供することができない．なぜなら，クローラはウェブアプリケーションのロジックや機能の目的 (例: e-コマースの場合，商品を購入すること) を理解せずに探索しているためである．

そこで，検査対象とするウェブアプリケーションを制限することによって，対象となったウェブアプリケーションに共通するロジックを利用して攻撃を実行することができる．しかし，この手法では検査できるウェブアプリケーションが限られるとともに，ロジックの異なるウェブアプリケーション毎にこの手法を適用する必要がある．

例えば，SSOScan [46] は，Facebook の Single Sign-On (SSO) を利用するウェブアプリケーションを検査対象とすることで，これらのウェブアプリケーションに共通する SSO のロジックを獲得し，このロジックに依存する攻撃を実行する．例えば，Facebook の SSO では，“Log in with Facebook” ボタンをクリックすることで，Facebook のログインページが現れ，ログインするためにメールまたは電話番号とパスワードが要求される．しかし，この手法では，他の SSO (Twitter や Google, LinkedIn など) を利用しているウェブアプリケーションを検査できず，SSO のロジック毎にシステムを生成する必要がある．

他にもウェブアプリケーションのロジックに依存する攻撃の一部の工程を自動化する手法 [50, 51, 52] があり，残りの工程はこの手法を利用する開発者が行う．これらの手法は，開発者が手動で攻撃を実行する時に攻撃者として用意する必要があるページを自動的に生成する．この手法によって開発者はこのようなページを生成するために知識が必要なくなり，ページの生成に時間をかける必要もなくなる．しかし，これらの手法が自動化していない工程については開発者が行わなければならず，攻撃や防御手法を回避する手法についての知識は求められる．

Rforge [50] は，開発者が手動で CSRF を実行するために使用するページを生成する．このページは，被害者に攻撃者の意図するリクエストの送信を強要するために，Rforge は開発者に攻撃者が強要するリクエストを要求し，獲得したリクエストからページを生成する．このようなリクエストを与えるために CSRF についての知識が必要になるので，全ての開発者が正確に攻撃者が強要するリクエストを与えることは難しい．

CJTools [51] と BeEF のプラグイン [52] は，visual clickjacking を実行するために使用するページの生成をサポートする．これらの既存ツールは，2 種類に分類される visual clickjacking の内の 1 種類にしか対応していない．さらに，防御手法を回避する手法が 7 種類あるにもかかわらず，これらの手法に幅広く対応していない．CJTool は，防御手法を回避する手法をサポートしておらず，BeEF のプラグインは，1 つの回避手法しかサポートしていない．

本研究では，このウェブアプリケーションのロジックを開発者から獲得することによって，対象とするウェブアプリーションを制限することなく脆弱性を検査する．本研究と同様に，開発者からウェブアプリケーションの情報を獲得することで攻撃を実行する既存手法 [40,41,42,43,44] がある．しかし，これらの既存手法は，ウェブアプリケーションのロジックに依存した攻撃を実行することができない．なぜなら，これらの既存手法が情報を獲得するのは，ウェブアプリケーションのロジックに依存する攻撃を実行するためではなく，より無駄なく網羅的に攻撃を実行するためだからである．

例えば，Sania [40] は SQL インジェクションを成功させる効率的な文字列を生成するために，正しい文字列の入力に対してウェブアプリケーションが発行する SQL クエリを獲得する．このような SQL クエリを獲得するために，Sania は開発者にウェブアプリケーションに対して正しい文字列の入力を要求する．この正しい文字列と SQL クエリを利用することで，入力された文字列がどのように SQL クエリに利用されるかがわかる．この情報を利用することで，攻撃を成功させるためにより効率的な文字列を生成することができ，無駄な攻撃を避けることができる．実際に Sania は既存のスキャナに比べて，少ない false positive でより多くの SQL インジェクションの脆弱性を検出することができる．

他にも，開発者からウェブアプリケーションのユースケースを獲得することで，XSS の脆弱性を検査する既存のスキャナの検査精度やクローラのページの網羅率を向上させる手法 [41] がある．例えば，複数の入力を必要とするフォームのユースケースを利用して，このフォームに対して複数の入力を行い正確に検査する．

Notamper [42] は，開発者に入力欄への正しい値の入力を要求し，この値からサービスを悪用する不正な値を生成する．Detoxss [44] は，開発者とウェブアプリケーション間で送受信されるリクエストとレスポンスを解析することで，XSS を検査する既存手法では生成できない複雑な XSS のためのスクリプトを生成する．Flax [43] は，開発者によるブラウザ上でのウェブアプリケーションの操作情報を獲得することで，クライアントサイドでの入力確認を検査するための文字列を生成する．

これらの既存手法は，XSS や SQL インジェクションの脆弱性のような入力確認の不備による脆弱性を対象としており，開発者から入力に関する情報を獲得している．既存手法が収集する入力に関する情報では，ウェブアプリケーションのロジックに依存する攻撃を実行することはできない．

また，既存のスキャナや既存手法の検査精度を向上させるために，ウェブアプリケーションのロジックを抽出するための手法 [53,54] がある．これらの手法と本研究の手法を組み合わせることで，開発者がウェブアプリケーションに関する情報を与えるためにかかる負担を軽減することができる．しかし，既存手法は開発者と同等の精度で情報を得ることができないために，本研究は開発者から獲得した情報を利用してペネトレーションテストを行う．

2.3 まとめ

本章で示したように，既存手法ではウェブアプリケーションのロジックに依存する攻撃に対する脆弱性を網羅的に検査することができない．静的検査では，検査できる脆弱性が限られ，検査範囲がプログラミング言語やフレームワークに依存してしまうために幅広いウェブアプリケーションの脆弱性を検査することができない．レスポンス解析では，ウェブアプリケーションに特定の防御手法の有無しか確認できないから，特定の防御手法以外の防御手法が実装されていた場合，正確に検査できない．

ペネトレーションテストでは，ウェブアプリケーションのロジックを獲得できないためにこのロジックに依存した攻撃を実行することができない．また開発者から獲得したウェブアプリケーションに関する情報を利用してペネトレーションテストを行う既存手法はあるものの，入力確認の不備による脆弱性を対象としており，獲得する情報は入力に関する情報であるためにこのような攻撃を実行することができない．

第3章　ウェブアプリケーションのロジックに依存する攻撃の実行

本論文では，脆弱性を検査するためにウェブアプリケーションのロジックに依存する攻撃を実行する手法を提案する．このロジックに依存する攻撃を実行するために，本手法はウェブアプリケーションのロジックに関する情報を獲得する．2.2.2に示したように，既存のペネトレーションテストはこのロジックに関する情報を獲得できないために，このような攻撃を実行することができない．本手法によってこれまで既存手法が実行できなかった攻撃による脆弱性検査が可能となり，開発者がペネトレーションテストを用いて自動的に検査できる脆弱性の網羅範囲が拡大する．

本章では，本手法がウェブアプリケーションのロジックに依存する攻撃を実行するためにとるアプローチを示し，具体的な概要について説明する．そして，最後に本手法を用いてこのような攻撃を実行できることを示すために，本手法を適用した攻撃を紹介する．

3.1　アプローチ

提案機構は，既存のペネトレーションテストでは実行できないウェブアプリケーションのロジックに依存する攻撃を実行する．このウェブアプリケーションのロジックに依存する攻撃は，攻撃の過程においてウェブアプリケーションが持つ機能を動作させなければいけない攻撃である．このような攻撃を実行するために，攻撃者はウェブアプリケーションの機能を操作して攻撃に必要な情報を獲得するなどの準備を行い，被害者は攻撃の準備が整った状態でウェブアプリケーションの機能を操作する必要がある．

ウェブアプリケーションが持つ機能を動作させるためにウェブアプリケーションのロジックが必要である．ウェブアプリケーションのロジックとは，ウェブアプリケーションの機能を動作させるための手順や情報である．実際の攻撃時にお

いて攻撃者や被害者は，ウェブアプリケーションのロジックを理解して操作しているために，ウェブアプリケーションを操作できないことによって攻撃が失敗することはない．攻撃が失敗する時は，被害者が攻撃に気付いたり，防御手法によって攻撃が妨げられる時である．

しかし，これらのロジックを自動的にウェブアプリケーションから獲得することは難しい．なぜなら，自動的に脆弱性を検査する機構はウェブアプリケーションのロジックや機能の目的を正確に理解することができないためである．よって何らかの手段を用いて，ウェブアプリケーションのロジックに関する情報を獲得する必要がある．

提案機構は，開発段階において利用されることを想定することで，提案機構の利用者である開発者からウェブアプリケーションのロジックに関する情報を獲得する．例えば，ウェブアプリケーションが持つ機能を操作する手順やユーザを識別するためのセッション ID の名前などを獲得する．これらの情報は，攻撃に関する情報でないために攻撃に関する知識を持たない開発者であってもこれらの情報を与えることができる．そして，検査対象のウェブアプリケーションの開発者であればこれらの情報を与えることは難しくない．

また，提案機構は開発者が情報を提供するための負荷を軽減するために，テストフェーズにおいて開発者がウェブアプリケーションの動作確認を行う間にいくつかの情報を自動的に獲得する．開発者が動作確認時にページのボタンをクリックするために発行したイベントやクリックによって発行したリクエストやレスポンスなどのあらゆる情報を獲得する．この獲得した情報を解析することで，ウェブアプリケーションのロジックに関する情報を獲得する．

開発者には，提案機構を用いるためにウェブアプリケーションのロジックに関する情報を提供するという負荷がかかるものの，攻撃や防御手法の回避手法などに関する詳細な知識がない開発者であっても，脆弱性や不完全な防御手法を検査することができる．なぜなら，提案機構は開発者から獲得した情報をもとにウェブアプリケーションのロジックに依存する攻撃を自動的に実行し，この攻撃に対するウェブアプリケーションの挙動を解析することによって脆弱性を検査するためである．

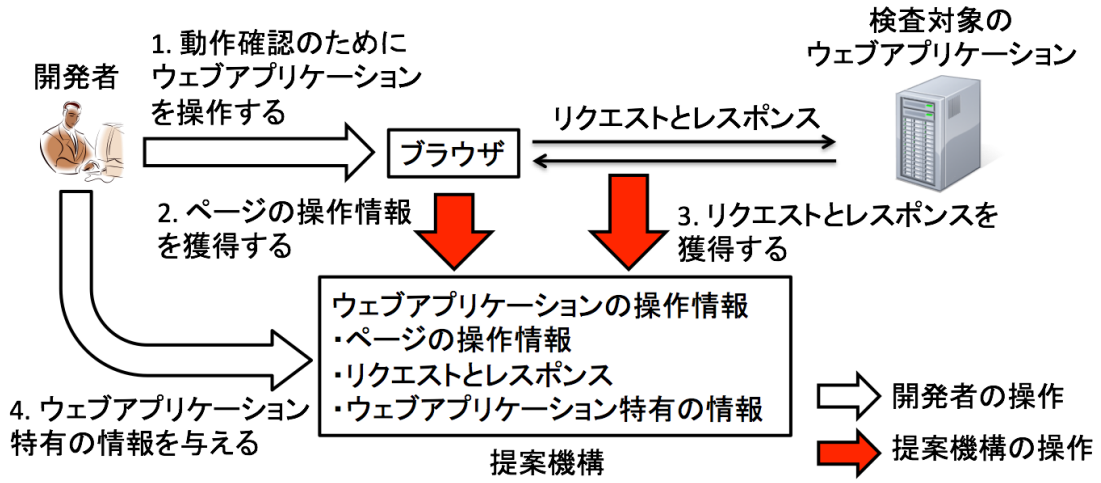

図 3.1: 情報収集フェーズ

開発者は動作確認のためにウェブアプリケーションを操作する (ステップ 1)．提案機構はブラウザ上で発行されるマウスイベントなどのページの操作情報や操作時に発行されるリクエストとレスポンスを獲得する (ステップ 2 と 3)．開発者からウェブアプリケーション特有の情報を獲得する (ステップ 4)．

3.2 提案機構の概要

提案機構は情報収集フェーズと検査フェーズの二つのフェーズによって構成される．図 3.1 に示すように情報収集フェーズにおいて，提案機構は攻撃を実行するためにウェブアプリケーションのロジックに関する情報を開発者から獲得する．3.1 節に示したように提案機構は，開発者がテストフェーズにおいて動作確認のために，ウェブアプリケーションを操作している (ステップ 1) 間にこれらの情報を獲得する．

提案機構は，ブラウザに表示された検査対象のページを操作する手順を獲得するために，開発者が発行したクリックなどのマウスイベントや入力欄に入力した文字列などの情報を獲得する (ステップ 2)．そして，検査対象の機能を動作させるリクエストと正しく機能が動作したときのレスポンスを獲得するために，検査対象の機能の操作時に発行されたリクエストとレスポンスを全て獲得する (ステップ 3)．

最後に，獲得したリクエストとレスポンスを解析しても正確に特定できないウェブアプリケーション特有の情報を開発者に与えてもらう (ステップ 4)．これらの情報は，ウェブアプリケーション毎に異なる情報であるために正確に特定すること

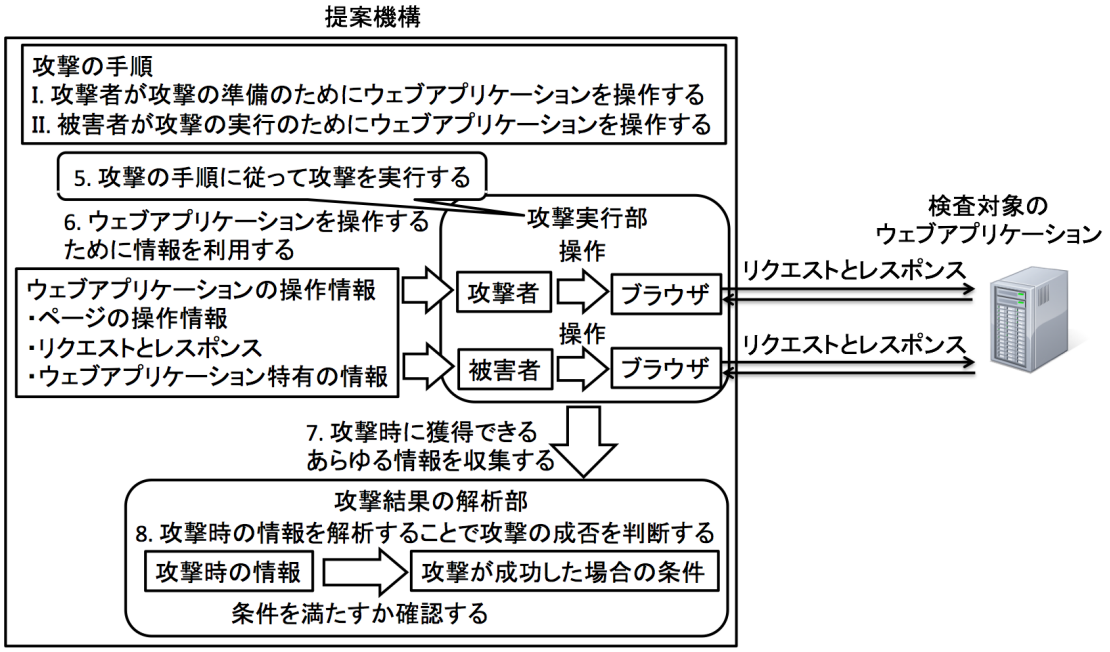

図 3.2: 検査フェーズ

提案機構が攻撃の手順に従ってウェブアプリケーションに攻撃を実行する (ステップ 5)．提案機構は，ウェブアプリケーションを操作するために情報収集フェーズに獲得した操作情報を利用する (ステップ 6)．攻撃時に獲得できる情報を獲得する (ステップ 7)．ステップ 7 において獲得した情報が攻撃が成功した場合の条件を満たすことを確認することで，その攻撃の成否を判断する (ステップ 8)．

が難しい．例えば，ウェブアプリケーションがユーザを識別するために利用するセッション ID の名前やウェブアプリケーションにログインするためのユーザ名とパスワードなどである．

図 3.2 に示すように攻撃フェーズにおいて，提案機構は攻撃の手順に従ってウェブアプリケーションに対して攻撃を実行する (ステップ 5)．この攻撃の手順とは，ウェブアプリケーションのロジックに依存する攻撃を実行するために，攻撃者や被害者がウェブアプリケーションを操作する手順や攻撃時における制限，攻撃者が提供するページの生成方法などをモデル化したものである．提案機構は，攻撃の手順に従ってウェブアプリケーションを操作する時に，情報収集フェーズに獲得したウェブアプリケーションの操作情報を利用する (ステップ 6)．そして，攻撃の実行時に獲得できるあらゆる情報を獲得する (ステップ 7)．例えば，ウェブアプリケーションが発行したレスポンスやブラウザ上でのページの挙動などの情報を獲得する．提案機構は，ステップ 7 において獲得した情報を解析することで，攻

撃の成否を判断する (ステップ 8)．攻撃の成否を判定するために，攻撃が成功した場合の条件を満たすことを確認する．

提案機構の開発者が，検査フェーズにおける攻撃の手順と攻撃が成功した場合の条件を用意することで，提案手法はウェブアプリケーションのロジックに依存するさまざまな攻撃を実行することができる．提案機構は，ウェブアプリケーションのロジックに関する情報を獲得しているために，どのような攻撃の手順を与えられても実行することができる．さらに，提案機構は攻撃時のあらゆる情報を獲得することができるために攻撃が成功した場合の条件があれば，攻撃の成否を判断することができる．

3.3 本手法を適用する攻撃

提案機構の有用性を示すために，提案機構がウェブアプリケーションのロジックに依存する攻撃を実行できることを示す．前節で説明したように，提案機構はウェブアプリケーションのロジックに依存したさまざまな攻撃を実行することができる．本論文において，ウェブアプリケーションのロジックに依存する攻撃として CSRF と session fixation，visual clickjacking を選択した．

CSRF は，被害者に攻撃者の意図するリクエストを強制的に送信させることで，被害者の権限でウェブアプリケーションの持つ機能を動作させる攻撃である [47]．CSRF の実行時に被害者はウェブアプリケーションにログインし，攻撃者は被害者に攻撃対象となる機能を動作させるためのリクエストを強制的に送信させる．この CSRF を実行するために，提案機構はログインのロジックと攻撃対象 (検査対象) となる機能のロジックに関する情報を獲得する必要がある．

Session fixation は，被害者に攻撃者が用意したセッション ID の利用を強要することで，攻撃者は被害者のセッション ID を獲得して被害者になりすます攻撃である [48]．このセッション ID は，ウェブアプリケーションがユーザを識別するための識別子で，攻撃者が被害者のセッション ID を利用することで被害者になりすますことができる．Session fixation の実行時に攻撃者はウェブアプリケーションからセッション ID を獲得するためにログインし，強要されたセッション ID と被害者を関連付けさせるために，被害者はこのセッション ID を使ってウェブアプリケーションにログインする．この session fixation を実行するために，提案機構はログインのロジックに関する情報を獲得する必要がある．

Visual clickjacking は，被害者にページ上のエレメントを正しく認識させないことで意図しないエレメントをクリックさせる攻撃である [22]．Visual clickjacking の実行時に被害者は，攻撃者が提供するページに埋め込まれた攻撃対象のページを操作する．この visual clickjacking を実行するために，提案機構は攻撃対象 (検査対象) となる機能のロジックに関する情報を獲得する必要がある．

これらの攻撃は現実的な脅威であり，実際に被害がもたらされている．さらに，稼働中のウェブアプリケーションには，これらの攻撃に対する脆弱性が残ってしまっている．

CSRF の脆弱性によって電子通貨である Bitcoin [55] の価値が，それまで $30 程度を推移していたにもかかわらず，$10 まで暴落してしまった [56]．これは，Bitcoin 交換アプリケーションである Mt.Gox [57] に CSRF の脆弱性があったためである．この脆弱性によって被害者は Mt.Gox で強制的に Bitcoin を交換させられる．また WhiteHat Security は，稼働中のウェブアプリケーションの 26% に CSRF の脆弱性があり，14% に Session fixation の脆弱性があると報告している [1]. さらにこれらの脆弱性は，普及度とリスクによって順位付けされる OWASP TOP 10 に選ばれている [58].

Sophos は，visual clickjacking によって多くの Facebook ユーザがワームに感染したことを報告している [59]．この visual clickjacking により，ある Facebook ユーザは攻撃者のページを自分のフレンドに推薦させられてしまう．これを繰り返すことで，多くの Facebook ユーザが攻撃者のページを推薦させられてしまった．

他にも Adobe Flash Player の設定マネージャに visual clickjacking の脆弱性があった [60]．被害者は，visual clickjacking によって設定マネージャのボタンをクリックさせられることで，被害者のカメラやマイクを乗っ取られてしまう．2012 年に行われた調査 [61] によると，Alexa トップ 10 の 30% と銀行のサイトトップ 20 の 70%，5 つの有名なオープンソースのウェブアプリケーションの 80 % が visual clickjacking に対する防御手法を実装していないことがわかった．

以降の 4 章と 5 章で，本手法を用いて実行するセッション管理の脆弱性を突いた攻撃 (CSRF と session fixation) と visual clickjacking で脆弱性を検査する手法とその手法の有用性をそれぞれ説明する．

第4章　セッション管理の脆弱性を突いた攻撃の実行

本章では，本手法を用いることでウェブアプリケーションのロジックに依存する Cross-Site Request Forgery (CSRF) と session fixation を実行し，これらの脆弱性を検査できることを示す．まず，CSRF と session fixation を防御する手法と実行する手法を説明するためにそれぞれの攻撃について説明する．そして，既存の防御手法は実装後に動作確認のために検査を行う必要があることを示した後に，これらの攻撃を実行することで脆弱性を検査する提案機構について説明する，前章で示したように，提案機構はこれらの攻撃を実行するために開発者からログインのロジックやログアウトのロジック，検査対象の機能のロジックに関する情報を獲得する．CSRF や session fixation の実行に必要な開発者から獲得する情報や攻撃の手順が異なるために，4.3 節と 4.4 節で CSRF と session fixation を実行する手法をそれぞれ説明する．最後に，提案機構が実環境で利用されているオープンソースのウェブアプリケーションの脆弱性を検査できることを示す．

4.1　セッション管理の脆弱性を突いた攻撃

この節では，セッション管理の脆弱性を突いた CSRF や session fixation を説明するために，ウェブアプリケーションが行っているセッション管理について説明した後に，それぞれの攻撃について説明する．

4.1.1　セッション管理

現在，多くのウェブアプリケーションがより便利なサービスを提供するためにセッション管理を行っている．このセッション管理は，HTTP や HTTPS のようなステートレスプロトコルにおいてユーザの識別や動作の捕捉を行うために利用されている．セッション管理においてセッション (ログイン状態やページ遷移の状態，

入力されたデータの情報など) とユーザを関連づけるためにセッション ID を利用する．ウェブアプリケーションは，ユニークなセッション ID をそれぞれのユーザに割り当てて，ユーザにこのセッション ID を付加したリクエストを送信させることでユーザを識別する．

ウェブアプリケーションは，ユーザが送信するリクエストにこのセッション ID が付加されるようにレスポンスにセッション ID を埋め込む．ユーザが送信するリクエストにセッション ID を付加させるために，ウェブアプリケーションはレスポンスの Set-Cookie かフォームの hidden フィールド，URI のいずれかにセッション ID を埋め込む．

また，セッション ID の名前や値の生成方法にルールはなく，ウェブアプリケーションの開発者が自由に決定することができる．実際に，オープンソースのウェブアプリケーションである phpBB や phpNuke，Mambo のセッション ID の名前としてそれぞれ “phpbbmysql_sid” や “user”，“1e850c4b3e85eb7663d66c7ddff906c4” が利用されており，値についてもランダムな値やユーザ名とパスワードを暗号化した値などが利用されている．

4.1.2 Cross-Site Request Forgery (CSRF)

CSRF は，被害者に攻撃者が意図するリクエストを送信させることで，被害者の権限でウェブアプリケーションが持つ機能 (例: 商品を購入する機能や管理者権限を変更する機能) を動作させる攻撃である．この攻撃は，ウェブアプリケーションがサードパーティのページから送信されたリクエスト (以降，クロスサイトリクエストと呼ぶ) が，被害者の意図したリクエストか強要されたリクエストかを識別できないことを悪用する．被害者の権限でウェブアプリケーションが持つ機能を動作させることで，被害者に攻撃者の口座へ送金させたり，特定の株式を購入させることができる．

図 4.1 を用いて，CSRF の概要を説明する．まず，被害者はログインページを要求し，被害者のユーザ名とパスワードが埋め込まれたログインするためのリクエスト (以降，ログインリクエストと呼ぶ) を送信することでウェブアプリケーションにログインする (ステップ 1 と 2)．このとき，ウェブアプリケーションはセッション管理のために被害者に対してセッション ID を発行する．

次に，攻撃者はソーシャルエンジニアリングなどを利用することで被害者を攻撃者のページに誘導する (ステップ 3)．この攻撃者のページは，被害者のブラウザ

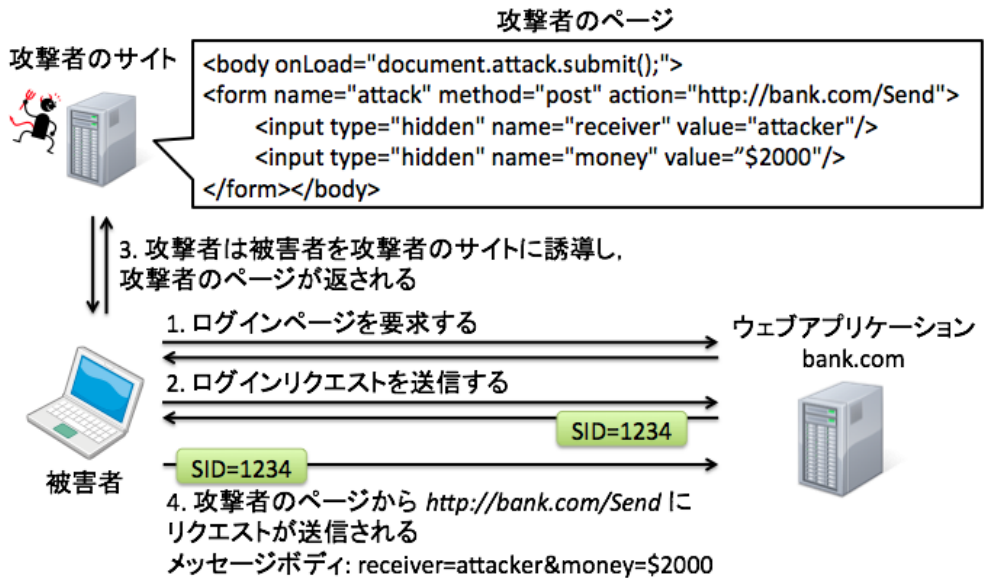

図 4.1: Cross-Site Request Forgery (CSRF) の例

被害者がウェブアプリケーションのログインページを要求する (ステップ 1)．ログインリクエストを送信し，ウェブアプリケーションからセッション ID を獲得する (ステップ 2)．被害者は攻撃者によって攻撃者のサイトに誘導され，攻撃者のページが返される (ステップ 3)．この攻撃者のページがウェブアプリケーションにリクエストを送信する (ステップ 4)．

にウェブアプリケーションへ攻撃者の意図するリクエストを送信させる (ステップ 4)．図 4.1 において，攻撃者の意図するリクエストはウェブアプリケーションが持つ送金機能に攻撃者の口座へ現金を送金させる．このとき，被害者のブラウザは自動的にこのリクエストにセッション ID を付加する．最後に，ウェブアプリケーションはセッション ID からこのリクエストが被害者からのものであると判断し，送金機能は被害者の権限で攻撃者の口座に現金を送金する．

4.1.3 Session fixation

Session fixation は，攻撃者が用意したセッション ID を被害者に使用させて，被害者のセッション ID を獲得する攻撃である．この攻撃は，ウェブアプリケーションがあるユーザに対して発行したセッション ID を他のユーザが利用できることを悪用する．攻撃者に強要されたセッション ID を使って被害者がログインすることでこのセッション ID と被害者が関連づけられ，攻撃者は被害者のセッション ID を入手することができる．攻撃者は，入手した被害者のセッション ID を使って被

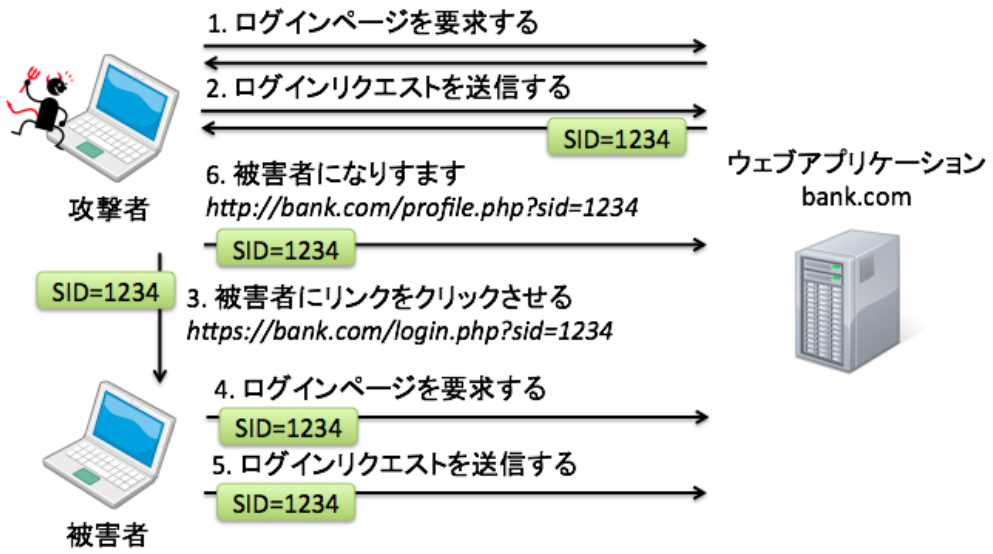

図 4.2: Session fixation の例

攻撃者がログインページを要求する (ステップ 1)．ログインリクエストを送信し，セッション ID を獲得する (ステップ 2)．攻撃者は被害者にセッション ID を強要するために，被害者にリンクをクリックさせる (ステップ 3)．被害者は，強要されたセッション ID を使用してログインページを要求し，ログインリクエストを送信する (ステップ 4 と 5)．攻撃者は強要したセッション ID を利用して被害者になりすます (ステップ 6)．

害者になりすますことができる．被害者になりすますことで，攻撃者は被害者の個人情報を入手したり，被害者として買い物を行うことができる．

図 4.2 を用いて，session fixation の概要を説明する．被害者に強要するセッション ID を獲得するために，攻撃者はログインページを要求し，攻撃者のユーザ名とパスワードが埋め込まれたログインリクエストを送信するすることでウェブアプリケーションにログインする (ステップ 1 と 2)．被害者が攻撃者のセッション ID で攻撃者のセッションを利用できることを防ぐために，セッション ID を獲得した後に攻撃者はログアウトする．攻撃者は，さまざまな手法 [62] を利用してこのセッション ID を被害者に強要する．例えば，被害者にセッション ID を埋め込んだリンクをクリックさせる (ステップ 3)．

被害者は，ログインページを要求するために強要されたセッション ID とリクエストを送信し (ステップ 4)，被害者のユーザ名とパスワードが埋め込まれたログインリクエストを送信することでウェブアプリケーションにログインする (ステップ 5)．攻撃者は，被害者に強要したセッション ID とリクエストをウェブアプリケーションに送信することで (ステップ 6)，ウェブアプリケーションはセッション ID

からこのリクエストが被害者からのものであると判断するために，攻撃者は被害者の権限でウェブアプリケーションを操作することができる．

4.2 攻撃を防ぐための既存手法

CSRF や session fixation を防ぐための既存手法として，サーバサイドでの防御手法とクライアントサイドでの防御手法，サーバとクライアントによる防御手法が提案されている．以降の項では，サーバサイドでの防御手法は攻撃を正確に防ぐかを検査する必要であり，クライアントサイドでの防御手法とサーバとクライアントによる防御手法は，クライアントによる防御手法の導入が必須となるために必ず攻撃を防ぐことができないことを説明する．

4.2.1 サーバサイドでの防御手法

CSRF を防ぐために，ウェブアプリケーションは被害者が意図的に送信したリクエストと攻撃者によって強要されたリクエストを識別する．これらのリクエストを識別するために，トークンを利用する手法やセッション ID を利用する手法，リファラヘッダを利用する手法がある．

トークンを利用する手法において，ウェブアプリケーションがリクエストにトークンの有無を確認することで攻撃者によって強要されたリクエストを識別する．リクエスト内にトークンを含ませるために，ウェブアプリケーションは HTML ページ内にトークンを埋め込む．このトークンは，リクエストと一緒にウェブアプリケーションに送信されるためにリクエストに正しいトークンが含まれるときのみ，ウェブアプリケーションはそのリクエストを処理する．攻撃者はこのトークンを用意できないために，攻撃者のページから発行されるリクエストにはトークンが含まれず，そのリクエストはウェブアプリケーションによって処理されない．

このトークンについてもセッション ID と同様にトークンの名前や値の生成方法にルールはなく，ウェブアプリケーションの開発者が自由に決定することができる．

開発者はこのトークンによる防御手法をウェブアプリケーションに実装し，機能毎に防御手法を適用する必要があるために，実装や適用時に不具合が残る可能性がある．このように不具合が残ることを防ぐために Noforge [17] は，プロキシとしてトークンによる防御手法を提供する．開発者は Noforge を導入することで

ウェブアプリケーションを改変することなく CSRF を防ぐことができる．しかし，ウェブアプリケーションのロジックにかかわらず，トークンが含まれないクロスサイトリクエストを全て処理しないためにサードパーティと協力してサービスを提供する場合に Noforge を利用できない．

Pelizzi らは，Noforge を拡張することで特定のクロスサイトリクエストを許可する手法 [19] を提案している．この手法は，クロスサイトリクエストを送信したオリジンを確認し，特定のオリジンから送られてきたリクエストを処理する．しかし，開発者がこの特定のオリジンを指定する際に間違いを犯す可能性があるために検査を行う必要がある．

セッション ID を利用する手法において，ウェブアプリケーションがレスポンスの URI やフォームの hidden フィールドにセッション ID を埋め込むことでトークンを用いた手法と同様に攻撃者によって強要されたリクエストを識別する．ウェブアプリケーションはリクエストのパラメータやボディにセッション ID が含まれるか確認し，被害者のセッション ID がリクエストに含まれる場合のみ，そのリクエストを処理する．攻撃者は被害者のセッション ID を用意できずに，攻撃者のページから発行されるリクエストのパラメータやボディにセッション ID が含まれないためにこのリクエストは処理されない．

攻撃者が session fixation や XSS を利用して被害者のセッション ID を盗むことができた場合，攻撃者によって強要されるリクエストにセッション ID が含まれるために CSRF を防ぐことができない．しかし，セッション ID は攻撃者によって盗まれてはいけないため，開発者はウェブアプリケーションに session fixation や XSS の脆弱性がないことを検査する必要がある．

リファラヘッダを利用する手法において，ウェブアプリケーションはリクエストを発行したページの URI を示す HTTP リファラヘッダを確認することで，攻撃者によって強要されたリクエストを識別する．ウェブアプリケーションはリファラヘッダを参照して，正しいドメインやページから発行されたリクエストのみを処理する．攻撃者のページから発行されたリクエストのリファラヘッダは攻撃者のドメインや攻撃者のページを示すために，ウェブアプリケーションはそのリクエストを処理しない．ウェブアプリケーションは正しい URI のリストを利用して，リクエストのリファラヘッダを確認する．開発者によってこのリストが正しく指定されていることやリストに従って正確にリクエストを処理していることを検査する必要がある．

しかし，サードパーティへの情報漏洩を防ぐためにブラウザの設定でリクエス

トにリファラヘッダを付加しないことが可能である．例えば，このリファラヘッダにはリクエストパラメータが付加されるために，セッション ID が URI で伝播されている場合に，セッション ID を漏洩してしまう．設定によってリファラヘッダが付加されない場合に，ウェブアプリケーションはリクエストを識別できないために CSRF を防ぐことができない．

そこで，Barth らの手法 [18] では，ウェブアプリケーションがリクエストを送信したオリジンを特定するための新しいヘッダを提案している．この手法は，オリジンのみが格納された新しいヘッダをサーバに送信することで，リクエストパラメータなどの情報の漏洩を防ぎつつ CSRF を防ぐことができる．Pelizzi らの手法 [19] と同様に開発者によって正しいオリジンのリストが指定されるために，検査を行う必要がある．

Session fixation を防ぐために，ウェブアプリケーションはログイン時に新しいセッション ID を発行する．Session fixation は，あるユーザに対してウェブアプリケーションが発行したセッション ID を他のユーザが利用できることで起こる．図 4.2 のステップ 5 において，被害者が攻撃者に強要されたセッション ID と関連づけられたために，ステップ 6 においてウェブアプリケーションは攻撃者からのリクエストを被害者からのリクエストとして識別してしまう．それに対して，ログインの度に新しいセッション ID が発行されることで，ステップ 2 で攻撃者が獲得したセッション ID とステップ 5 で被害者が獲得したセッション ID は異なる値になる．

開発者は，フレームワークやサーバが提供しているセッション管理機能を利用しているために，この機能がログインの度に新しいセッション ID を発行しているかわからない．この機能の利用者である開発者や機能の提供者が，ログインの度に新しいセッション ID を発行しているかを検査する必要がある．

Johns らの手法 [20] は，プロキシとしてユーザのログイン時に新しいセッション ID を発行する機能を提供する．この手法では，ユーザに第二のセッション ID を発行し，ユーザのログイン時に新しい第二のセッション ID を発行する．攻撃者が被害者のログイン前にオリジナルのセッション ID と第二のセッション ID を強要しても，ログイン時に発行された新しい第二のセッション ID を獲得することができない．この手法の適用時における不具合によって正しく動作しないことがあるために，検査を行う必要がある．

4.2.2 クライアントサイドでの防御手法

クライアントサイドでの防御手法 [63, 64, 65, 66, 67] では，攻撃時に得られる情報が多いためにそれらの情報を活用することで攻撃を防ぐ．これは，CSRF におけるリクエストの強要や session fixation におけるセッション ID の強要が被害者のブラウザ上で実行されるためである．

しかし，ウェブアプリケーションの開発者は，クライアントサイドでの防御手法に依存せずに，サーバサイドでの防御手法で CSRF や session fixation を防ぐべきである．クライアントサイドでの既存手法は，ウェブアプリケーションのロジックを考慮できないために false positive や negative を起こす可能性があり，ウェブアプリケーションの利便性を損なってしまう．さらに，ユーザがブラウザにこれらの防御手法を導入しなければ攻撃を防ぐことができないために，攻撃が成功する可能性が残ってしまう．

RequestRodeo [63] は，クライアントサイドでプロキシとして動作することでトークンによる防御手法を実現して CSRF を防ぐ．しかし，全てのクロスサイトリクエストを禁止するために，サードパーティと協力してサービスを提供しているウェブアプリケーションに悪影響を及ぼす．

BEAP [64] は，発行されたリクエストが攻撃者に強要されたリクエストであるか識別することで CSRF を防ぐ．リクエストを識別するために，リクエストを攻撃者に強要されたリクエストとユーザの意図するリクエストに分類する．しかし，無害なクロスサイトリクエストであっても分類に当てはまれば全て禁止されてしまう．

クロスサイトリクエストを利用するウェブアプリケーションに与える影響を軽減するための手法 [65, 66] がある．Ryck らは，過去の通信ログを調査することでクロスサイトリクエストが強要されたものか識別する手法 [65] を提案している．クロスサイトリクエストが発行される時に，このリクエストをやりとりするサイト間で過去にリダイレクトか POST メッセージが発行されたか確認し，発行されていない場合にこのリクエストが強要されたと判断する．しかし，過去にこれらのリクエストが発行されていなければ，無害なクロスサイトリクエストでも強要されたと判断する．

Shahriar らは，クロスサイトリクエスト発行時にセッション ID を付加せずに送信したクロスサイトリクエストに対するレスポンスを確認することで，強要されたリクエストを識別する手法 [66] を提案している．例えば，img タグを利用したリク

エストに対して画像 (コンテンツタイプ: image) が得られるにも関わらず，HTML (コンテンツタイプ: text/html) が得られる場合，このリクエストは強要されたと判断する．これは，CSRF の実行時に攻撃者は img タグを利用してリクエストの発行を強要するためである．しかし，開発者がコンテンツタイプを正しく設定していない場合に false positive が生じる．

Serene [67] は，session fixation を防ぐために Cookie 内のセッション ID を追跡することで攻撃者に強要されたセッション ID を検知し，このセッション ID を破棄する．Serene は，ヒューリスティックな手法を用いてセッション ID を特定しているために false positive や negative を起こす．

4.2.3 サーバとクライアントによる防御手法

サーバとクライアントによる防御手法である SOMA [68] は，サイト間で相互に認証を行うことでリクエストの強要を防ぐ．クロスサイトリクエストを送信する際に，ブラウザがリクエストの送信を要求するサイトとリクエストを受信するサイトに確認を取ることで，リクエストの送信を決定する．SOMA は，サーバが設定をすることで必ず攻撃を防ぐことができるわけではなく，クライアントにも SOMA を導入する必要がある．しかし，ユーザに CSRF などの攻撃に関する知識がなければ導入は難しい．

4.3 CSRF の実行による検査手法

提案機構は，脆弱性を検査するためにウェブアプリケーションに対して自動的に CSRF を実行する．4.1.2 で示したように，この攻撃はウェブアプリケーションが被害者の意図したリクエストと強要されたリクエストを識別できないことを悪用する．提案機構は，CSRF を実行するために攻撃者によって強要されるリクエストを生成し，生成したリクエストを被害者としてウェブアプリケーションに送信する．そして，提案機構は攻撃時に送信したリクエストに対するウェブアプリケーションの挙動を解析することで，実行した CSRF が成功したかを確認する．

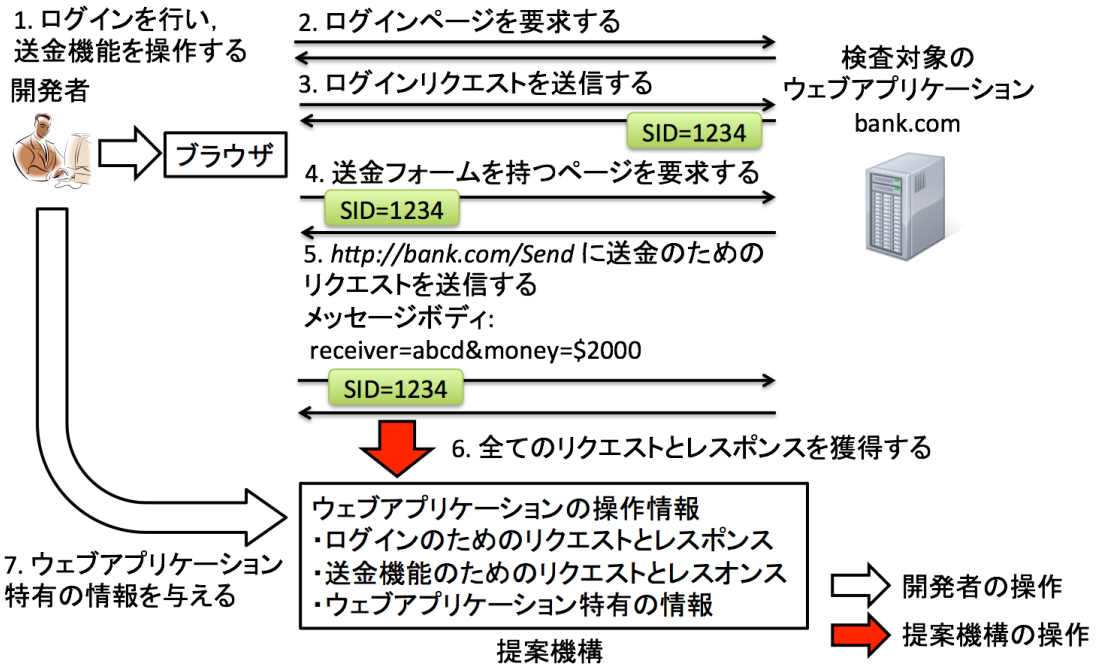

図 4.3: CSRF の脆弱性を検査するための情報収集フェーズ

開発者は，動作確認のためにログインを行い，送金機能を操作する (ステップ 1)．ブラウザは，開発者のログイン操作に従ってログインページを要求し，ログインリクエストを送信する (ステップ 2 と 3)．送金機能の操作に従って送金フォームを持つページを要求し，送金のためのリクエストを送信する (ステップ 4 と 5)．提案機構は，ブラウザとウェブアプリケーション間で送受信されるリクエストとレスポンスを獲得し，開発者からウェブアプリケーション特有の情報を獲得する (ステップ 6 と 7)．

4.3.1 概要

提案機構は，CSRF を実行するために情報収集フェーズと検査フェーズの 2 つのフェーズを実行する．情報収集フェーズにおいて，提案機構は CSRF を実行するためにウェブアプリケーションのロジックに関する情報を獲得する．このロジックに関する情報とは，ウェブアプリケーションの機能を動作させるための手順や情報のことである．CSRF において，この情報はログインのためのリクエストとレスポンスと検査対象の機能 (例の場合，送金機能) のためのリクエストとレスポンス，ウェブアプリケーション特有の情報である．

提案機構は，開発者が動作確認のためにウェブアプリケーションが持つ機能を操作している間に，ウェブアプリケーションのロジックに関する情報を獲得する．図 4.3 は，提案機構が図 4.1 で用いたウェブアプリケーションの情報を獲得する例を示している．この例のように，開発者はテストフェーズにおいて動作確認のため

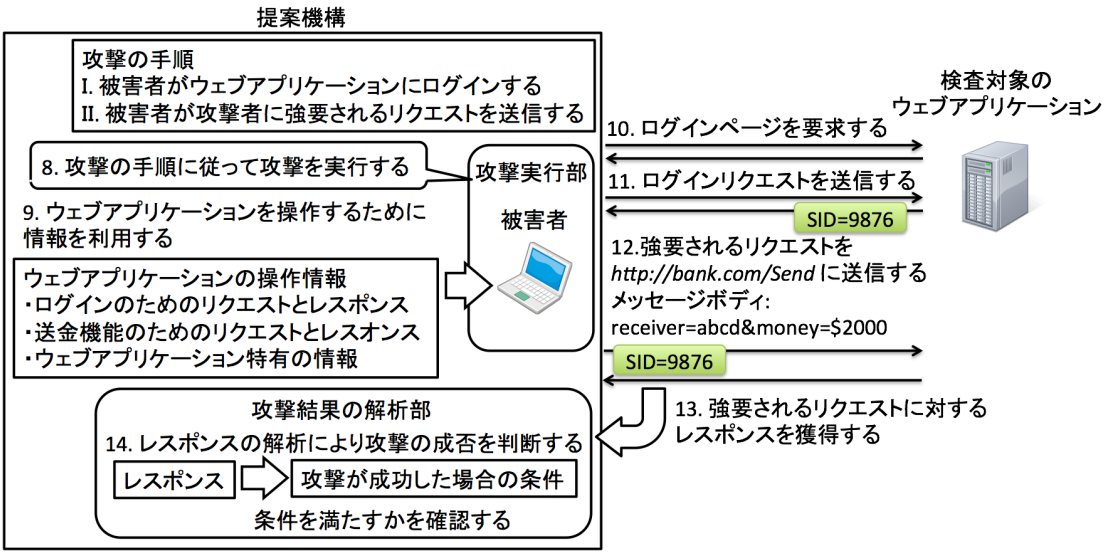

図 4.4: CSRF の脆弱性を検査するための検査フェーズ

提案機構は，攻撃の手順に従って CSRF を実行する (ステップ 8)．攻撃の実行に伴いウェブアプリケーションを操作する必要があるために，情報収集フェーズで獲得した情報を利用する (ステップ 9)．提案機構は被害者としてログインページを要求し，ログインリクエストを送信することでセッション ID を獲得する (ステップ 10 と 11)．被害者は獲得したセッション ID と強要されるリクエストを送信する (ステップ 12)．提案機構はステップ 12 で獲得したレスポンスを解析することで，CSRF が成功したか確認する (ステップ 13 と 14)．

に検査対象のウェブアプリケーションにログインし，送金機能を操作する (ステップ 1)．そのとき，ブラウザは開発者のログイン操作に従ってログインページを要求するリクエストを送信し，開発者のユーザ名とパスワードが埋め込まれたログインリクエストを送信する (ステップ 2 と 3)．次に，ブラウザは送金機能の操作に従って送金フォームを持つページを要求するリクエストを送信し，“http://bank.com/Send” に送金のためのリクエストを送信する (ステップ 4 と 5)．

提案機構は，プロキシとして動作することでブラウザとウェブアプリケーション間で送受信されるすべてのリクエストとレスポンスを獲得する (ステップ 6)．そして，開発者からウェブアプリケーション特有の情報を獲得する (ステップ 7)．開発者から獲得する情報の詳細は，4.3.2 で説明する．

検査フェーズにおいて，提案機構はウェブアプリケーションに CSRF を実行し，攻撃結果を解析することで脆弱性の有無を判断する．図 4.4 は，提案機構が図 4.1 で用いたウェブアプリケーションの CSRF の脆弱性を検査する例である．提案機構は，攻撃の手順に従って検査対象のウェブアプリケーションに CSRF を実行す

る (ステップ 8)．このとき提案機構はウェブアプリケーションにログインし，送金機能を動作させるリクエストを発行する必要があるために，情報収集フェーズで獲得した操作情報を利用する (ステップ 9)．提案機構は，この操作情報をウェブアプリケーション特有の情報で解析することによって操作に必要な情報を獲得する．操作に必要な情報と解析方法の詳細は 4.3.3 で説明する．

提案機構は，ログインのロジックに従ってログインするためにログインページを要求するリクエストを送信し，被害者のユーザ名とパスワードを埋め込んだログインリクエストを送信する (ステップ 10 と 11)．このログインリクエストに対してウェブアプリケーションが発行したセッション ID を獲得する．そして，送金機能のロジックに従って被害者が強要されるリクエストと獲得したセッション ID を検査対象のウェブアプリケーションに送信する (ステップ 12)．このリクエストは情報収集フェーズで獲得した送金機能のためのリクエストから生成する．この生成方法については 4.3.4 で説明する．攻撃結果を解析するために被害者が強要されるリクエストに対するレスポンスを獲得する (ステップ 13)．

そして，提案機構はステップ 13 において獲得したレスポンスを解析することで，実行した CSRF が成功したかを確認する (ステップ 14)．攻撃が成功したかを確認するために，獲得したレスポンスが条件を満たしているかを確認する．この条件とは，攻撃が成功した場合の条件であり，4.3.5 で詳細について説明する．

4.3.2 開発者が手動で与える情報

検査フェーズの手順を実行するために，提案機構は情報収集フェーズにおいて開発者に次の 5 つのウェブアプリケーション特有の情報を要求する．この情報は，セッション ID の名前と被害者のユーザ名とパスワード，検査対象の機能の URI，トークンの名前 (CSRF を防ぐためにトークンの防御手法を実装している場合)，完了メッセージである．この完了メッセージは，検査対象の機能が正常に動作したときにページに現れるメッセージである．例えば，送金機能を動作させた場合にページあらわれる完了メッセージは “The money transfer has been completed” である．

セッション ID の名前は，ステップ 11 においてレスポンス内のセッション ID を特定するために利用する．被害者のユーザ名とパスワードは，ステップ 11 において被害者としてウェブアプリケーションにログインするために利用する．検査対象の機能の URI は，開発者が検査対象の機能を操作した時のリクエストから検査対象の機能を動作させるためのリクエストを特定するために利用する．このリ

表 4.1: CSRF の脆弱性検査における開発者による入力例

ウェブアプリケーション特有の情報	入力例
セッション ID の名前	phpbbmysql_sid
被害者のユーザ名とパスワード	victim & victim_pwd
トークンの名前	null
検査対象の機能の URI	http://localhost/phpBB/ privmsg.php?mode=post
完了メッセージ	Your message has been sent

クエストからステップ 12 で送信される被害者に強要するリクエストを生成する．トークンの名前は，被害者が強要されるリクエストの生成時にリクエストに正しいトークンの値が埋め込まれないようにするために利用する．このトークンの名前を利用したリクエストの生成方法については 4.3.4 で説明する．完了メッセージは，ステップ 14 においてレスポンスが攻撃が成功した場合の条件を満たすことを確認するために利用する．表 4.1 にウェブアプリケーション特有の情報として開発者が与える入力の実例を示す．この入力例は，phpBB 2.0.12 のメッセージ送信機能を検査するために提案機構に与える入力である．

提案機構の利用者である開発者であれば，脆弱性についての知識を持っていなくてもこれらの情報を提案機構に与えることは難しくない．5.5 節の実験において，提案機構を用いてオープンソースのウェブアプリケーションの脆弱性を検査するために開発者でない著者がこれらの情報を与えたものの，情報を与えることは難しくなかった．

4.3.3 開発者から獲得した操作情報の解析

提案機構は，開発者から獲得した情報を解析することで CSRF を実行するために必要な情報を自動的に抽出する．自動的に抽出する情報を以下に列挙する．

- セッション ID を伝播する方法
- ログインリクエスト
- ログインリクエスト内にユーザ名とパスワードを埋め込む箇所
- ログインのためのリクエスト

セッション ID を伝播する方法は，ステップ 11 においてセッション ID を獲得し，ステップ 12 においてセッション ID をリクエストに埋め込むために特定する．ログインのためのリクエストは，ステップ 10 と 11 において検査対象のウェブアプリケーションにログインするために特定する．ログインリクエストとそのリクエスト内にユーザ名とパスワードを埋め込む箇所は，ステップ 11 においてログインリクエストに被害者のユーザ名とパスワードを埋め込むために特定する．

これらの情報を抽出するために，提案機構はステップ 6 で獲得した HTTP リクエストとレスポンスをウェブアプリケーション特有の情報で解析する．セッション ID の伝播方法を特定するために，ステップ 6 で獲得したレスポンスを開発者から獲得したセッション ID の名前で解析する．レスポンスの Cookie と URI，hidden field にセッション ID の名前があるかを確認する．HTTP レスポンスの Cookie にセッション ID の名前を発見した場合，セッション ID が Cookie によって伝播されると判断する．Cookie にセッション ID の名前を発見しなかった場合，URI か hidden field がセッション ID の伝播方法になるために URI と hidden field を確認する．例えば，表 4.1 に示したように開発者によって指定されたセッション ID の名前が “phpbbmysql_sid” であると仮定する．レスポンスの Cookie と URI，hidden field に “phpbbmysql_sid” があるかを確認する．HTTP レスポンスの Cookie “Set-cookie: phpbbmysql_sid=6da54ea....; path=/” に “phpbbmysql_sid” を発見した場合，セッション ID が Cookie によって伝播されると判断する．

ログインリクエストを特定するために，提案機構はステップ 6 で獲得した HTTP リクエストとレスポンスを解析する．ログインリクエストを特定する上で，ログインリクエストを発行するフォームの input 要素にある type 属性が “password” であると仮定する．なぜなら，ログインフォームのパスワードを入力する欄の type 属性は，他者にパスワードを盗み見られないように “password” になっていることが一般的だからである．

まず，提案機構はあるリクエスト A が GET メソッドか POST メソッドであるかを確認する．なぜなら，多くのウェブアプリケーションがログインリクエストを POST メソッドで送信しているためである．そのリクエスト A が POST メソッドであれば，提案機構はリクエスト A を発行したフォームを特定する．また，リクエスト A が GET メソッドであれば，提案機構はリクエスト A をログインリクエストでないと判断する．フォームを特定するために，リクエスト A のメッセージボディ内の全ての変数とリクエスト A を発行したページ内にあるフォームの input 要素が持つ name 属性を比較する．これらの変数と name 属性が一致する時，リク

エスト A は一致したフォームから送られたと判断する．この一致したフォームの input 要素が持つ type 属性が “password” のとき，提案機構はこのリクエスト A がログインリクエストであると判断する．

提案機構がログインリクエストを特定する一例を示す．開発者が次のログインフォームを利用してログインリクエストを発行し，

```
<form method="POST" action="./login_check.php">
    User name: <input type="text" name="username">
    Password: <input type="password" name="pwd">
    <input type="submit" value="Submit">
</form>
```

次のリクエストのメッセージボディが生成されたことを想定する．

```
username=tester&pwd=tester_pwd
```

このリクエストを受け取ると，提案機構はこのリクエストがどのフォームから送信されたものかを調査する．リクエストのメッセージボディ内にある全ての変数名を取り出し，それぞれのフォームの値と比較する．この場合，提案機構はメッセージボディ内の変数名 “username” と “pwd” を取り出し，上記のフォームと比較する．それぞれの変数名と値が一致することから提案機構はこのリクエストが上記のログインフォームから発行されたと判断する．このフォームに type 属性が “password” である input 要素があるために，このリクエストがログインリクエストであると判断する．

ログインリクエストに関する情報を利用して，ログインリクエストにユーザ名とパスワードを埋め込む箇所を特定する．ログインフォームの中にユーザ名の入力欄とパスワードの入力欄が設置されており，ユーザ名の入力欄が上 (または左) に設置されていると仮定する．まず，提案機構はパスワードを埋め込む箇所を特定するためにログインフォームの input 要素を確認する．input 要素 A の type 属性が “password” である name 属性を獲得する．獲得した name 属性と一致するログインリクエストのメッセージボディ内にある変数名を特定する．そして，この変数名の値がパスワードを埋め込む箇所であると判断する．

次に，提案機構はログインリクエスト内のユーザ名を埋め込む箇所を特定する．ユーザ名の入力欄はパスワードの入力欄の上 (または左) に設定されているために，type 属性が “password” の input 要素 A より前に設置された input 要素 B を確認す

る．input 要素 B の name 属性と一致するログインリクエストのメッセージボディ内の変数名を特定し，この変数名の値がユーザ名を埋め込む箇所であると判断する．

提案機構がログインリクエストにユーザ名とパスワードを埋め込む箇所を特定する一例を示す．上記の例と同様に開発者が次のログインフォームを利用してログインリクエストを発行し，

```
<form method="POST" action="./login_check.php">
    User name: <input type="text" name="username">
    Password: <input type="password" name="pwd">
    <input type="submit" value="Submit">
</form>
```

次のログインリクエストのメッセージボディが生成されたことを想定する．

```
username=tester&pwd=tester_pwd
```

まず，このログインフォームの input 要素の type 属性が “password” である name 属性 “pwd” を特定する．この “pwd” と一致するログインリクエストのメッセージボディ内の変数名が持つ値 “tester_pwd” がパスワードを埋め込む箇所である．そして，type 属性が “password” の input 要素より先に設置された input 要素を確認する．input 要素の name 属性 “username” と一致するログインリクエストのメッセージボディ内の変数名が持つ値 “tester” がユーザ名を埋め込む箇所であると判断する．

提案機構が，ログインリクエストを利用してログインするためのリクエストを特定する．提案機構は，開発者が最初にリクエストを発行した時から開発者がログインリクエストに対するレスポンスを得た時までをログインのリクエストと判断する．さらに，ログイン後から開発者が最後のレスポンスを獲得する時までが検査対象の機能のためのリクエストであると判断する．

4.3.4 被害者が強要されるリクエストの生成

提案機構は，ステップ 12 において被害者が強要されるリクエストを送信するためにこのリクエストを生成する必要がある．このリクエストは攻撃者によって強要されるから，攻撃者が得られる情報のみを利用して生成する．つまり，ウェブアプリケーションにトークンによる防御手法が実装されている場合，攻撃者は被

害者が獲得できるトークンの値を獲得できないため強要されるリクエストに被害者のトークンの値を埋め込んではいけない．

しかし，提案機構はランダムな文字列で生成した偽のトークンの値が埋め込まれた強要されるリクエストを生成する必要がある．なぜなら，検査対象のウェブアプリケーションが，リクエストにトークンが埋め込まれていることを確認するものの，トークンの値を確認していない場合，攻撃者によって用意された偽のトークンの値が埋め込まれたリクエストを被害者に強要することで CSRF が成功するためである．

さらに，4.1.2 にも示したようにユーザはブラウザでリファラヘッダを送信しない設定にできるために，リファラヘッダを持たない強要されるリクエストを生成する必要がある．このリクエストは，リファラヘッダを利用して CSRF を防いでいるウェブアプリケーションの脆弱性を発見するために生成する．

提案機構は，開発者から獲得したリクエストと検査対象の機能の URI を利用して，被害者に強要する 2 種類または 4 種類のリクエストを生成する．開発者が指定した検査対象の機能の URI を利用して検査対象の機能のためのリクエストから検査対象の機能を動作させるリクエストを特定する．そして，特定したリクエストから次の 4 種類の被害者が強要されるリクエストを生成する．

1. トークンとリファラヘッダを持たないリクエスト
2. トークンを持ち，リファラヘッダを持たないリクエスト
3. トークンを持たないものの，リファラヘッダを持つリクエスト
4. トークンとリファラヘッダを持つリクエスト

プライバシーのためにユーザがブラウザでリファラヘッダを送信しない設定をした環境を実現するために 1. と 2. のリクエストを生成する．検査対象のウェブアプリケーションにトークンによる防御手法が実装されている時に 2. と 4. のリクエストを生成する．提案機構は，開発者から獲得したトークンの名前で検査対象の機能のためのリクエストに埋め込まれていたトークンを特定する．そのトークンの値の長さと同じ長さの偽のトークンの値を生成し，2. と 4. のリクエストに埋め込む．

4.3.5 レスポンスの解析

提案機構は，攻撃が成功したかを確認するために，ある条件が満たされていることを確認する．この条件は，CSRF において被害者が強要されるリクエストによって，ウェブアプリケーションの機能が動作することである．この機能が動作したことを確認するために 2 つの手法が考えられる．

手法 1: 提案機構は，強要されるリクエストを送信することで獲得したレスポンス内のメッセージボディを確認する．このメッセージボディが情報収集フェーズに獲得したメッセージボディと同じであれば，強要されるリクエストが処理されたと判断する．しかし，この手法では，時間やユーザなどによってレスポンスが変化する場合に，CSRF が成功したにもかかわらず失敗したと判断してしまう。

手法 2: 提案機構は，攻撃時にウェブアプリケーションが発行したレスポンスのメッセージボディ内に開発者から獲得した完了メッセージが含まれるかを確認する．4.3.2 で示したように，この完了メッセージは検査対象の機能が正常に動作したときにページに現れるメッセージである．強要されるリクエストに対して得られたレスポンスがこのメッセージを持っていた場合，検査対象の機能を正常に動作させたためにその CSRF は成功したと判断する．

提案機構は，手法 1 は特定のページにおいて false positive や false negative を起こすために手法 2 を採用している．

4.4 Session fixation の実行による検査手法

提案機構は，session fixation の脆弱性を検査するためにウェブアプリケーションに自動的に攻撃を実行する．4.1.3 で示したように session fixation は，ウェブアプリケーションがあるユーザに対して発行したセッション ID を他のユーザが利用できることを悪用する．Session fixation を実行するにあたって提案機構は，攻撃者としてウェブアプリケーションからセッション ID を獲得し，被害者としてこのセッション ID を利用してウェブアプリケーションにログインする．そして提案機構は，被害者がこのセッション ID を利用できることを確認することで，実行した session fixation が成功したことを確認する．

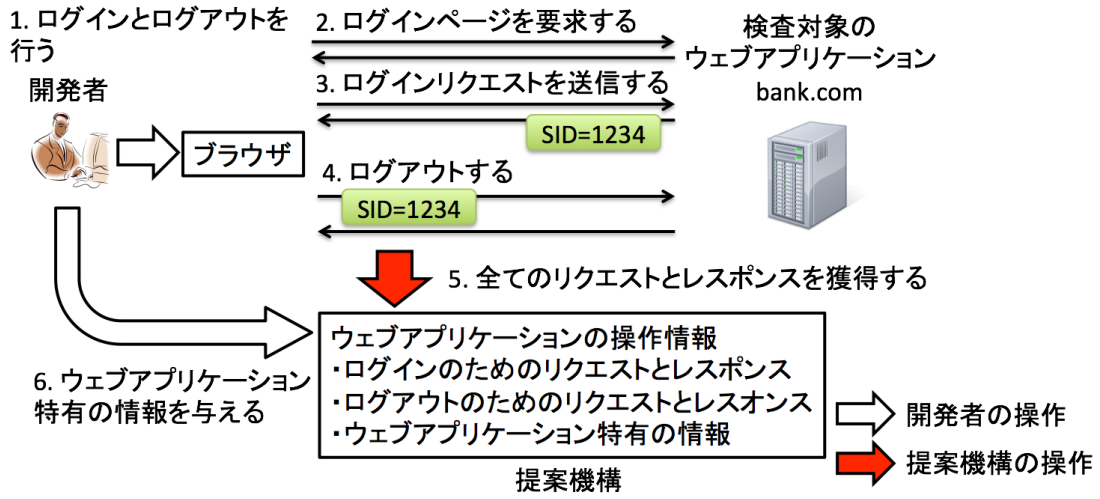

図 4.5: Session fixation の脆弱性を検査するための情報収集フェーズ

開発者は動作確認のためにログインとログアウトを行う (ステップ 1)．ブラウザは，開発者のログイン操作に従ってログインページを要求し，ログインリクエストを送信する (ステップ 2 と 3)．ブラウザはログアウトの操作に従ってログアウトのためのリクエストを送信する (ステップ 4)．提案機構は，送受信されるリクエストとレスポンスを獲得し，開発者からウェブアプリケーション特有の情報を獲得する (ステップ 5 と 6)．

4.4.1 概要

提案機構は，session fixation を実行するために情報収集フェーズと検査フェーズの 2 つのフェーズを実行する．情報収集フェーズにおいて，提案機構は session fixation を実行するためにウェブアプリケーションのロジックに依存する情報を獲得する．CSRF において，このロジックに関する情報はログインのためのリクエストとレスポンスとログアウトのためのリクエストとレスポンス，ウェブアプリケーション特有の情報である．

提案機構は，開発者が動作確認のためにログインとログアウトの操作を行っている間に情報を獲得する．図 4.5 は，提案機構が図 4.2 のウェブアプリケーションの情報を獲得する例を示している．この例のように，開発者はテストフェーズにおいて動作確認のために検査対象のウェブアプリケーションにログインして，ログアウトする (ステップ 1)．そのとき，ブラウザは開発者のログイン操作に従ってログインページを要求するリクエストを発行し，開発者のユーザ名とパスワードが埋め込まれたログインリクエストを送信する (ステップ 2 と 3)．次に，ブラウザは，ログアウトするためのリクエストを送信する (ステップ 4)．CSRF の場合と同

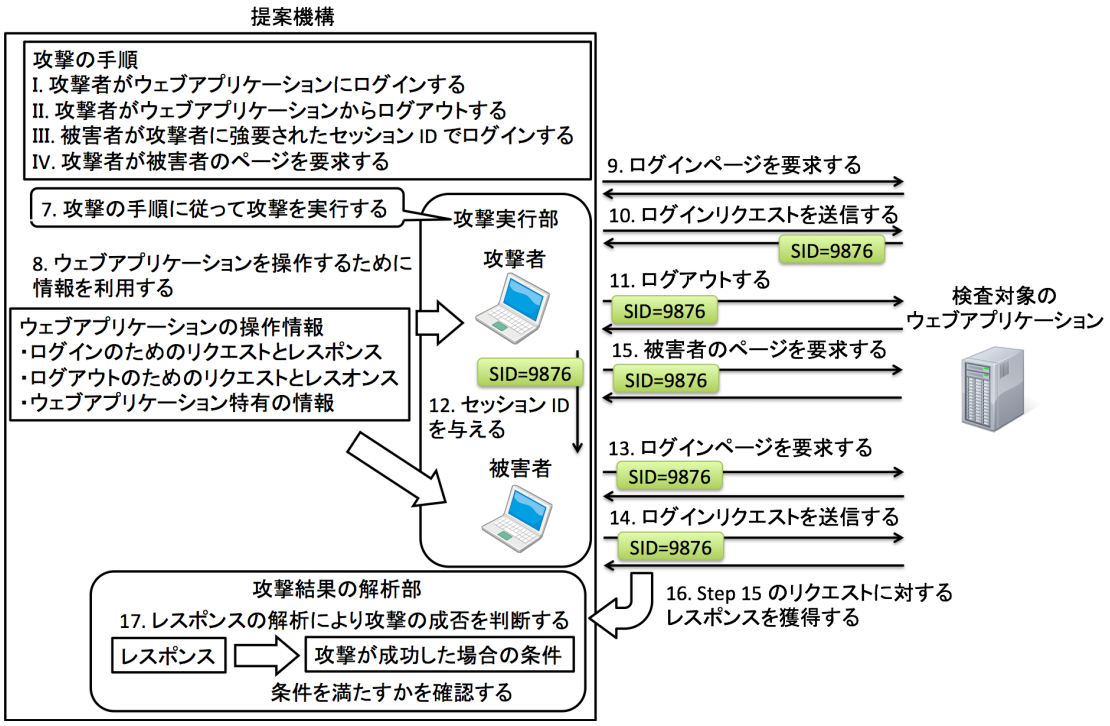

図 4.6: Session fixation の脆弱性を検査するための検査フェーズ

提案機構は，攻撃の手順に従って session fixation を実行する (ステップ 7)．攻撃を実行するために，情報収集フェーズで獲得した情報を利用する (ステップ 8)．提案機構は攻撃者としてログインすることでセッション ID を獲得し，ログアウトする (ステップ 9 と 10，11)．被害者としてステップ 10 で獲得したセッション ID でログインする (ステップ 12 と 13，14)．攻撃者としてステップ 10 で獲得したセッション ID で被害者のページにアクセスする (ステップ 15)．提案機構はステップ 15 で獲得したレスポンスを解析することで，session fixation が成功したか確認する (ステップ 16 と 17)．

様に，提案機構はブラウザとウェブアプリケーションで送受信されるリクエストとレスポンスを獲得し，開発者からウェブアプリケーション特有の情報を獲得する (ステップ 5 と 6)．開発者から獲得する情報の詳細は，4.4.2 で説明する．

検査フェーズにおいて session fixation を実行し，攻撃結果を解析することで脆弱性の有無を判断する．図 4.6 は，提案機構が図 4.2 のウェブアプリケーションの session fixation の脆弱性を検査する例である．提案機構は，攻撃の手順に従って検査対象のウェブアプリケーションに session fixation を実行する (ステップ 7)．このとき提案機構はウェブアプリケーションにログインとログアウトする必要があるために，情報収集フェーズで獲得した操作情報を利用する (ステップ 8)．CSRF の場合と同様に，提案機構はログインとログアウトに必要な情報を獲得するためにこの操作情報をウェブアプリケーション特有の情報で解析する．操作に必要な情

表 4.2: Session fixation の脆弱性検査における開発者による入力例

ウェブアプリケーション特有の情報	入力例
セッション ID の名前	phpbbmysql_sid
攻撃者のユーザ名とパスワード	attacker & attacker_pwd
被害者のユーザ名とパスワード	victim & victim_pwd
被害者のページに現れる文字列	Log out [victim]

報と解析方法は 4.4.3 で説明する．

提案機構は，ログインのロジックに関する情報に従って攻撃者としてログインページを要求するリクエストを送信し，攻撃者のユーザ名とパスワードを埋め込んだログインリクエストを送信し，そのウェブアプリケーションが発行したセッション ID を獲得する (ステップ 9 と 10)．そして，ログアウトのロジックに関する情報に従ってログアウトのためのリクエストを送信する (ステップ 11)．次に，提案機構は，ログインのロジックに関する情報に従って被害者としてステップ 10 で獲得したセッション ID とログインページを要求するリクエストを送信し，被害者のユーザ名とパスワードを埋め込んだログインリクエストを送信する (ステップ 12 と 13，14)．最後に，提案機構はステップ 10 で獲得したセッション ID を使って攻撃者として被害者のページを要求する (ステップ 15)．攻撃結果を解析するために，ステップ 15 で送信したリクエストに対するレスポンスを獲得する (ステップ 16)．

そして提案機構はステップ 16 において獲得したレスポンスを解析することで実行した session fixation が成功したかを確認する (ステップ 17)．攻撃が成功したかを確認するために，獲得したレスポンスが条件を満たしているかを確認する．この条件とは，攻撃が成功した場合の条件であり，4.4.5 で詳細について説明する．

4.4.2 開発者が手動で与える情報

検査フェーズの手順を実行するために，提案機構は情報収集フェーズにおいて開発者に 4 つのウェブアプリケーション特有の情報を要求する．この情報は，セッション ID の名前と攻撃者のユーザ名とパスワード，被害者のユーザ名とパスワード，被害者のページに現れる文字列である．この文字列は，被害者が自身のページにアクセスしたときにページに現れる “victim (被害者のユーザ名) さん” などの文字列である．

セッション ID の名前は，ステップ 10 においてレスポンス内のセッション ID を特定するために必要である．攻撃者のユーザ名とパスワードと被害者のユーザ名とパスワードは，ステップ 10 と 14 において攻撃者と被害者としてウェブアプリケーションにログインするために必要である．被害者のページに現れる文字列は，ステップ 17 において攻撃者が被害者のページにアクセスできることを確認するために必要である．表 4.2 にウェブアプリケーション特有の情報として開発者が与える入力の例を示す．

CSRF の場合と同様に開発者であれば，これらの情報を提案機構に与えることは難しくない．5.5 節の実験において，開発者でない著者でもこれらの情報を与えることができた．

4.4.3 開発者から獲得した操作情報の解析

提案機構は，開発者から獲得した情報を解析することで session fixation を実行するために必要な情報を自動的に抽出する．このとき自動的に抽出する情報を以下に列挙する．

- セッション ID を伝播する方法
- ログインリクエスト
- ログインリクエスト内にユーザ名とパスワードを埋め込む箇所
- ログインとログアウトのためのリクエスト

セッション ID の伝播する方法は，ステップ 10 においてセッション ID を獲得し，ステップ 11 と 13，14，15 においてセッション ID をリクエストに埋め込むために特定する．ログインとログアウトのためのリクエストは，ステップ 9 と 10，11，13，14 において検査対象のウェブアプリケーションにログインとログアウトするために特定する．ログインリクエストとそのリクエスト内にユーザ名とパスワードを埋め込む箇所は，ステップ 10 と 14 においてログインリクエストに被害者のユーザ名とパスワードと被害者のユーザ名とパスワードを埋め込むために特定する．これらの情報は，CSRF と同様の方法で自動的に抽出している．

4.4.4　検査の効率化

提案機構はウェブアプリケーションに session fixation を行う前に，検査を効率化するために開発者のログインの前後でセッション ID が変更されたことを確認する．もしセッション ID が変更された場合，提案機構は検査対象のウェブアプリケーションに脆弱性がないと判断する．これは，ログインの前後でセッション ID を変更することは有効な防御手法だからである．

提案機構は，情報収集フェーズにおいて開発者がログインを行う間にログインの前後でセッション ID の値が変更されていることを確認する．セッション ID の値を確認するために，開発者から獲得したセッション ID の名前と開発者がログイン時に発行するリクエストやレスポンスを解析する．開発者がログインする時にセッション ID の値が変更された場合，この段階で提案機構は session fixation の脆弱性がないと判断する．セッション ID の値が変更されなかった場合，提案機構は session fixation を行い脆弱性を検査する．なぜなら，ウェブアプリケーションにログインの前後でセッション ID を変更する以外の防御手法が実装されているかを確認するためである．

4.4.5　レスポンスの解析

提案機構は，攻撃が成功したことを確認するために条件が満たされていることを確認する．この条件とは，session fixation において攻撃者が被害者のページにアクセスできたことである．この条件を確認するために，提案機構はステップ 15 で被害者のページにアクセスできることを確認する．

提案機構は，CSRF と同様にステップ 15 において，ウェブアプリケーションが発行したレスポンスのボディ内に，開発者から獲得した被害者のページに現れる文字列が含まれることを確認する．このレスポンスが被害者のページに現れる文字列を持っていた場合，攻撃者が被害者のページにアクセスできているので，その Session fixation は成功したと判断する．

4.5　リクエストの生成

提案機構は，攻撃時に攻撃者と被害者が検査対象のウェブアプリケーションに送信するリクエストを生成するために，情報収集フェーズにおいて開発者から獲

得したリクエストを利用する．リクエストを利用する上で，提案機構はリクエスト内のユーザや時間による変化を考慮する必要がある．例えば，タイムスタンプを利用して有効なフォームからリクエストが送信されていることを確認するウェブアプリーションがあるとする．このウェブアプリケーションはフォームの生成時にタイムスタンプをフォームに埋め込む，ユーザがこのフォームからリクエストを送信することでこのリクエストには必ずタイムスタンプが含まれる．このタイムスタンプはフォームの生成時の時間であるために，リクエスト毎にタイムスタンプを修正する必要がある．

提案機構は，攻撃時に検査対象のウェブアプリケーションによって発行されたレスポンスを全て監視する．提案機構は監視した情報をもとに開発者から獲得したリクエストを変更する．例えば，開発者から獲得したリクエスト内にあるセッション ID は開発者に発行されたものだから，提案機構はリクエスト内にある開発者のセッション ID を攻撃者や被害者のセッション ID に変更する．

4.6 実装

提案機構を Amberate [69] の脆弱性検査用プラグインとして実装する．Amberate は，ウェブアプリケーションの脆弱性を検査するためのフレームワークで，さまざまな脆弱性を検査するためのプラグインを実装するために API を提供している．現在 Amberate は，SQL インジェクションや XSS，JavaScript hijacking，ディレクトリトラバーサルを検査するためのプラグインを提供している．Amberate は，実際に経済産業省が運営しているオープンガバメント [70] などのサイトを検査するために利用されている．

Amberate は，検査対象のウェブアプリケーションに攻撃を行い，攻撃が成功したことを確認するためにレスポンスを解析する．そのなかで Amberate のプラグインは次の 2 つのタスクを実行する．1) 攻撃を実行するためにリクエストやメッセージなどを生成する．2) 脆弱性を検査するために攻撃によって得られたレスポンスを解析する．例えば，CSRF のためのプラグインは CSRF を実行するために被害者に強要されるリクエストを生成し，CSRF の脆弱性を検査するために生成したリクエストに対するレスポンスを解析する．

リクエストやメッセージを生成したり，攻撃結果を解析するために，Amberate はウェブアプリケーションと開発者の間で送受信されるリクエストとレスポンスを獲

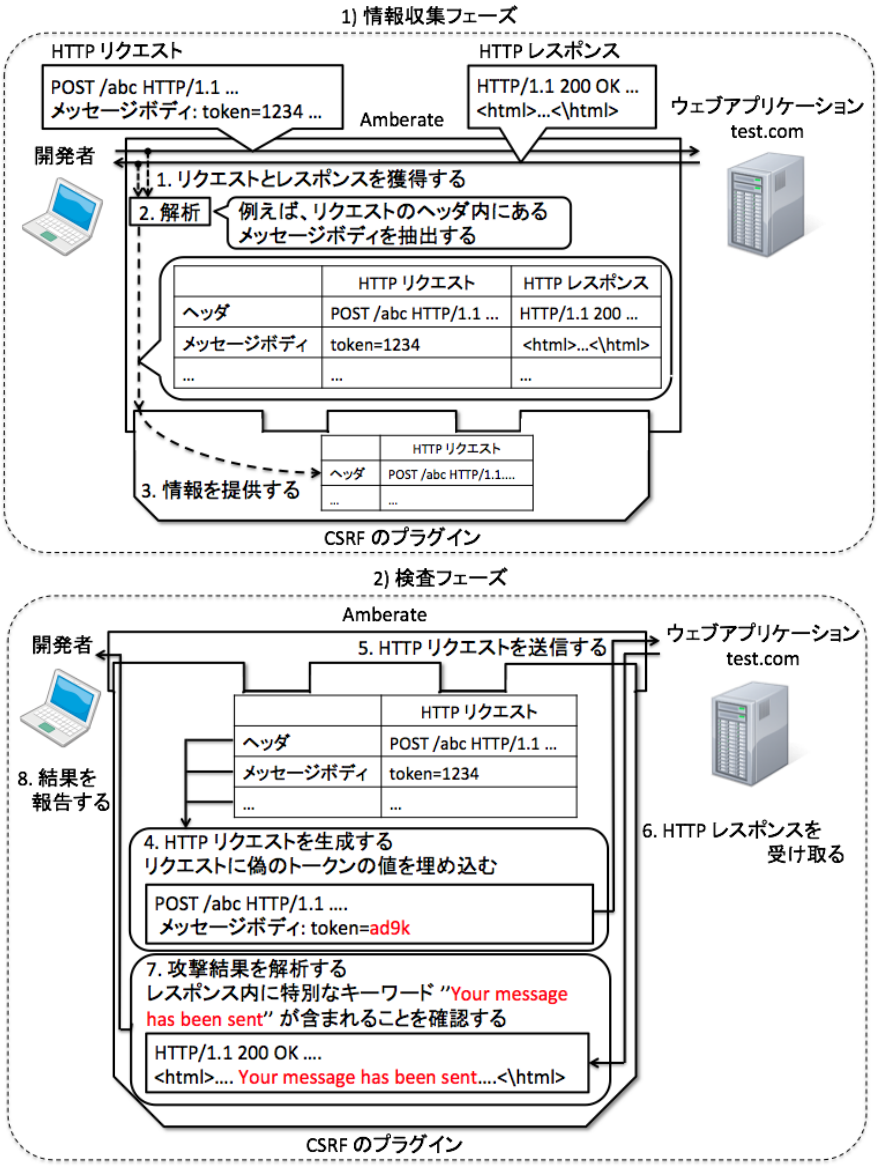

図 4.7: Amberate の概要

情報収集フェーズにおいて，Amberate は送受信されるリクエストやレスポンスを獲得し，このリクエストとレスポンスを解析する (ステップ 1 と 2)．Amberate は獲得した情報をプラグインに与える (ステップ 3)．検査フェーズにおいて，CSRF のプラグインは，強要されるリクエストを生成し，そのリクエストを送信する (ステップ 4 と 5)．このリクエストに対するレスポンスを獲得し (ステップ 6)，このレスポンスのボディに特別なキーワードがあるかを確認する (ステップ 7)．CSRF の脆弱性の有無を開発者に結果として報告する (ステップ 8)．

得したり，ウェブアプリケーション特有の情報を開発者に要求する．例えば，CSRF のためのプラグインがレスポンス内のセッション ID を特定するために，Amberate は開発者にセッション ID の名前を要求する．

図 4.7 に示すように，Amberate は 1) 情報収集フェーズと 2) 検査フェーズで構成されている．Amberate の利用者である開発者は，情報収集フェーズにおいてブラウザでウェブアプリケーションにアクセスする．例えば，開発者がウェブアプリケーションへのログインやメッセージの送信を実行する．Amberate は，HTTP プロキシとして動作し，ブラウザとウェブアプリケーション間で送受信されるリクエストやレスポンスを獲得する (ステップ 1).

Amberate は，ステップ 1 で獲得したリクエストとレスポンスを解析する (ステップ 2)．例えば，Amberate はリクエストのヘッダ内にあるフィールド名が “Cookie” である値を Cookie として抽出する．また，レスポンスのメッセージボディをパースすることで，プラグインによるレスポンスの解析をサポートする．Amberate はリクエストやレスポンスを含むこれらの情報を全てのプラグインに与える (ステップ 3).

検査フェーズにおいて，それぞれのプラグインが Amberate から獲得した情報を利用して攻撃を生成し，攻撃をウェブアプリケーションに送信する．送信したリクエストに対するレスポンスを解析することでウェブアプリケーションの脆弱性を判断する．

例えば，CSRF のためのプラグインは，リクエストのメッセージボディ内に偽のトークンの値を含む強要されるリクエストを生成し，そのリクエストをウェブアプリケーションに送信する (ステップ 4 と 5)．このリクエストを生成するために，Amberate から獲得したリクエストやセッション ID の名前，トークンの名前を利用する．次に、このプラグインは送信したリクエストに対するレスポンスを獲得し (ステップ 6)，このレスポンスのボディに特別なキーワードがあることを確認する (ステップ 7)．そして，検査対象のウェブアプリケーションにおける CSRF の脆弱性の有無を開発者に結果として報告する (ステップ 8).

4.7 実験

提案機構が実行した CSRF や session fixation で脆弱性を検査できることを確認するために，実際にさまざまなサイトで利用されている 5 つのオープンソースのウェブアプリケーションに提案機構を適用した．[71,72,73] などの脆弱性リポジトリを調査し，CSRF の脆弱性や session fixation の脆弱性を持ったオープンソースのウェブアプリケーションと脆弱性を持たないもので実験を行った．利用した 5 つのウェ

ブアプリケーションは，Mambo [74] や Joomla [75]，phpBB [76]，phpNuke [77]，osCommerce [78] である．さらに，これらのウェブアプリケーションに対して，実際に CSRF の脆弱性と session fixation の脆弱性の有無を手動での攻撃とソースコードの解析によって確認した．

検査対象のウェブアプリケーションは MAMP マシン (Mac OS X version 10.6.8 と Apache 2.2.11，MySQL 5.1.39，PHP 5.3.0) 上で動作する．MAMP マシンと Amberate は 4 GB メモリで 2.8 GHz Intel Core 2 Duo と Mac OS X version 10.6.8 上で動作する．

4.7.1 CSRF の検査結果

表 4.3 は，提案機構が CSRF の脆弱性を検査した結果をまとめたものである．表 4.3 の “ウェブアプリケーション” は検査対象のウェブアプリケーションの名前とバージョンを示している．“ユーザ” は検査対象の機能を利用するユーザを示しており，“検査対象の機能” は提案機構が検査を行った機能を示している．“脆弱性” は手動での攻撃とソースコードの解析で得た検査結果を示しており，“提案機構” は提案機構による検査結果を示している．表 4.3 から提案機構は，検査できない 3 つの機能以外の脆弱性を正確に検査できることがわかる．

提案機構は，phpNuke 7.0 と 8.2.4 が持つユーザ情報の変更を行う機能を検査できない．これは，検査対象の機能を利用してユーザ情報を変更した後のページに特別なキーワードがないためである．ユーザ情報が変更された時，phpNuke は処理の完了を示すページに進まずにスタートページに戻る．このような機能の検査時に開発者がスタートページに現れるキーワードを与えた場合，提案機構は CSRF の成否にかかわらずに攻撃が成功したと判断してしまい，false positive が発生する可能性がある．

このような機能を検査するために，機能の操作成功時に影響がでるページを開発者から獲得し，攻撃後にそのページを確認することで検査を行うことができる．しかし，このような機能はまれであると考えており，ある操作を行った時にその操作が成功したか失敗したかをユーザに示すことが一般的であるためにこのような実装を行っていない．

提案機構は phpBB 3.0.9 のユーザ情報の変更を行う機能を検査できない．これは，管理者として phpBB 3.0.9 にログインするために 2 度のユーザ名とパスワードの入力を要求されるためである．現在の提案機構の実装は，ユーザ名とパスワー

表 4.3: CSRF の検査結果

ウェブアプリケーション	ユーザ	検査対象の機能	脆弱性	提案機構
phpBB 2.0.12	ユーザ	メッセージの送信	vul.	vul.
		メッセージの削除	vul.	vul.
	管理者	管理者権限の変更	No vul.	No vul.
		ユーザ情報の変更	No vul.	No vul.
phpBB 3.0.9	ユーザ	メッセージの送信	No vul.	No vul.
		メッセージの削除	No vul.	No vul.
	管理者	ユーザ情報の変更	No vul.	unable
phpNuke 7.0	ユーザ	メッセージの追加	vul.	vul.
		メッセージの削除	vul.	vul.
	管理者	ユーザ情報の変更	vul.	unable
		ユーザの登録	vul.	vul.
phpNuke 8.2.4	ユーザ	メッセージの追加	vul.	vul.
		メッセージの削除	vul.	vul.
	管理者	ユーザ情報の変更	vul.	unable
		ユーザの登録	vul.	vul.
Mambo 4.6.2	ユーザ	ユーザ情報の変更	No vul.	No vul.
	管理者	ユーザの登録	vul.	vul.
Joomla 1.0.9	ユーザ	ユーザ情報の変更	No vul.	No vul.
	管理者	ユーザの登録	vul.	vul.
osCommerce 2.2-MS1	ユーザ	カートに商品の追加	vul.	vul.
		商品の購入	No vul.	No vul.
	管理者	-	-	-

“vul.” は CSRF の脆弱性があることを示しており，“No vul.” は脆弱性がないことを示す．“-” は，提案機構で検査を行わなかったことを示す．これは，検査対象のウェブアプリケーションが管理者のためのログイン機能を持たないためである．“unable” は，提案機構で検査を行えないことを示す．

ドを 1 回入力することを想定しており，このようなページを検査できない．しかし，2 度目のログインを行うための情報は開発者から獲得できているので，実装を変更することでこの問題は解決できると考えている．

表 4.4: Session fixation の検査結果

ウェブアプリケーション	ユーザ	脆弱性	提案機構
Mambo 4.6.2	ユーザ	vul.	vul.
	管理者	vul.	vul.
Joomla 1.0.9	ユーザ	vul.	vul.
	管理者	vul.	vul.
phpBB 2.0.12	ユーザ	vul.	vul.
	管理者	-	-
phpBB 3.0.9	ユーザ	No vul.	No vul.
	管理者	No vul.	unable
phpNuke 7.0	ユーザ	No vul.	No vul.
	管理者	No vul.	vul.
phpNuke 8.2.4	ユーザ	No vul.	No vul.
	管理者	No vul.	vul.
osCommerce 2.2-MS1	ユーザ	vul.	vul.
	管理者	-	-

“vul.” は Session fixation の脆弱性があることを示しており，“No vul.” は session fixation の脆弱性がないことを示す．“-” は，提案機構で検査を行わなかったことを示す．これは，検査対象のウェブアプリケーションが管理者のためのログイン機能を持たないためである．“unable” は，提案機構で検査を行えないことを示す．

4.7.2 Session fixation の検査結果

表 4.4 は，提案機構が session fixation の脆弱性を検査した結果をまとめたものである．表 4.4 の “ウェブアプリケーション” は検査対象のウェブアプリケーションの名前とバージョンを示している．“ユーザ” はログイン機能を利用するユーザを示している．“脆弱性” は手動での攻撃とソースコードの解析で得た検査結果を示しており，“提案機構” は提案機構による検査結果を示している．表 4.4 から提案機構は 3 つのログイン機能以外の脆弱性を正確に検査できることがわかる．

提案機構は，phpNuke 7.0 と 8.2.4 のログイン機能に脆弱性がないにもかかわらず脆弱性があると判断した．これは，phpNuke の管理者ページが管理者毎に差違がなく，管理者がログアウトしてもセッション ID を無効化しないためである．phpNuke は，開発者がログインする度に開発者のユーザ名とパスワード，タイムスタンプを暗号化したセッション ID を発行しているために session fixation の脆弱

性はない．このようなセッション ID を利用しているためか，ログアウトしても有効期限がすぎるまでこのセッション ID を利用できる．

提案機構が phpNuke の脆弱性を検査する時に false positive を起こすステップを説明する．まず，開発者がブラウザで管理者として phpNuke にログインし，ログアウトする (ステップ 1)．開発者は，セッション ID の名前と攻撃者のユーザ名とパスワード，被害者のユーザ名とパスワード，被害者のページに現れる特別なキーワードの 4 つの情報を提案機構に与える．phpNuke の管理者ページは管理者毎に差違がないために，この特別なキーワードは攻撃者のページと被害者のページに現れる文字列である．

提案機構は開発者から獲得した情報を用いて session fixation を行う．提案機構は攻撃者として phpNuke にログインし，phpNuke からセッション ID を獲得する (ステップ 2)．提案機構がログアウトするとき，phpNuke は攻撃者のセッション ID を無効化しない (ステップ 3)．提案機構は被害者として phpNuke に攻撃者のセッション ID を用いてログインをし，phpNuke から新しいセッション ID を獲得する (ステップ 4)．提案機構は攻撃者として被害者に強要したセッション ID を用いて管理者のページを要求する (ステップ 5)．提案機構は，開発者から獲得した特別なキーワードを用いて，ステップ 5 で獲得したレスポンスを解析することで攻撃が成功したと判断する (ステップ 6)．なぜなら，特別なキーワードがレスポンス内に現れるためである．

phpNuke に session fixation の脆弱性はないものの，セッション管理の方法に問題がある．このセッション管理の方法は，管理者のログアウト時にセッション ID を無効化しないというものであり，開発者はこのセッション管理の方法を改善するべきである．なぜなら，攻撃者が管理者からセッション ID を盗むことで，管理者がログアウトした後でもこのセッション ID を利用して管理者としてウェブアプリケーションを操作できてしまうためである．

提案機構が phpBB 3.0.9 の管理者のログイン機能を検査できない理由は CSRF の脆弱性を検査できない理由と同じで，phpBB 3.0.9 が管理者としてのログインするために 2 度のユーザ名とパスワードの入力を要求するためである．

検査対象としたウェブアプリケーションのソースコードを解析した結果，Mambo と Joomla，php-Nuke はセッション ID と IP アドレスを使ってユーザの識別を行っていることがわかった．このような識別方法を利用することで，session fixation を防ぐことができる．Session fixation を行うために攻撃者が被害者にセッション ID を強要することで被害者のセッション ID を獲得したとしても，攻撃者と被害者

の IP アドレスは異なるためにウェブアプリケーションは被害者と攻撃者を識別できる．

しかし，セッション ID と IP アドレスを用いてユーザを識別する手法は session fixation に対して完璧な防御手法ではない．これは，イントラネット内から同じ IP アドレスでユーザがウェブアプリケーションにアクセスしたとき，イントラネット内で生じた session fixation は成功してしまうためである．Kolsek はこの手法を緩和手法として紹介している [48]．提案機構は，この防御手法のみが実装されているウェブアプリケーションに session fixation の脆弱性があると判断する．

4.8 まとめ

ウェブアプリケーションのロジックに依存する CSRF と session fixation を自動的に実行することによって脆弱性を検査する手法を提案した．提案機構は攻撃を実行するために，開発者から獲得したウェブアプリケーションのロジックに関する情報を利用する．開発者には情報を与えるという負荷がかかるものの，提案手法を用いることで開発者は攻撃や防御手法を回避する手法に関する知識なしに脆弱性を検査できる．さらに，ウェブアプリケーションに実装される既存の防御手法は攻撃を正確に防ぐか検査する必要があり，これらの防御手法を検査するために提案機構を利用することができる．

提案機構が CSRF や session fixation の脆弱性を検査できることを確認するために，実際にサイトで利用されている 5 つのオープンソースのウェブアプリケーションに提案機構を適用した．その結果，提案機構が検査した機能に残る脆弱性については全て検出することができ，5 つのオープンソースのウェブアプリケーションに残った 11 個の CSRF の脆弱性と 6 個の session fixation の脆弱性を発見した．また，提案機構が特定の機能を検査できなかった理由や false positive を起こした理由について議論した．

第5章　Visual clickjacking の実行

本章では，本手法を用いてウェブアプリケーションのロジックに依存する visual clickjacking を実行し，脆弱性を検査できることを示す．まず，visual clickjacking を防御する手法と実行する手法を説明するために攻撃について説明する．そして，防御手法の実装に不具合が残りやすく，防御手法を回避する手法もあるために検査を行う必要があることを示す．その後に，visual clickjacking を実行することで脆弱性を検査する Clickjuggler を提案する．3 章で示したように，Clickjuggler は攻撃を実行するために開発者から獲得した検査対象の機能のロジックに関する情報をもとにウェブアプリケーションを動作させる．さらに，Clickjuggler は，防御手法を回避する攻撃を実行することで回避手法に対する防御手法の脆弱性を検査することができる．最後に，Clickjuggler が実環境で利用されているオープンソースのウェブアプリケーションの脆弱性を正確に検査できることを示す．

5.1　Visual clickjacking

Visual clickjacking は，被害者に意図しないウェブページ上にあるエレメントのクリックを強要する [22]．この攻撃は，被害者にページ上のエレメントを十分に認識させない．例えば，攻撃者は被害者が認識できないボタンやリンク，フォームなどを用意することで，被害者にそれらのボタンを操作させる．被害者に特定のボタンの操作を強要することで，意図しない商品の購入や攻撃者のアクセス権や被害者のパスワードの変更などを行わせることができる．

Visual clickjacking の例を用いて，攻撃の概要を説明する．この例では，攻撃者があるショッピングサイト (shopping.com) で異常に高額な商品を出品し，被害者にこの高額な商品を購入させることが目的である．この商品を購入させるためには，被害者にショッピングサイトのページ (ターゲットページと呼ぶ) にある “Buy” ボタンをクリックさせる必要がある．図 5.1 は，この clickjacking の例を表している．このショッピングサイトにある clickjacking の脆弱性を利用するために攻撃者

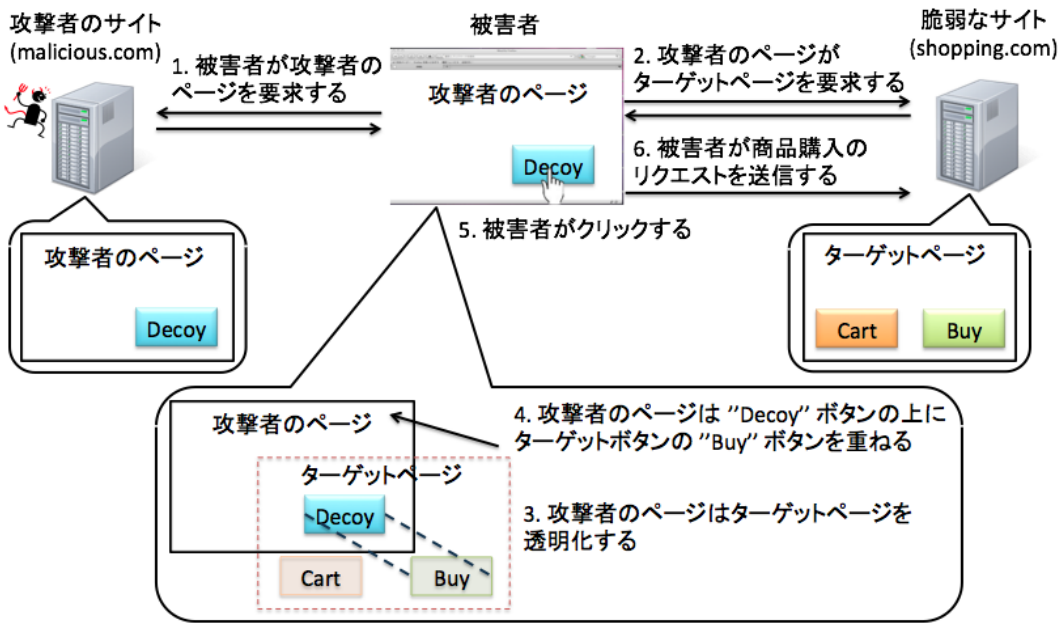

図 5.1: Clickjacking の例

被害者が攻撃者のページにアクセスする (ステップ 1)．攻撃者のページがターゲットページを要求する (ステップ 2)．攻撃者のページは，被害者に “Buy” ボタンのクリックを強要するためにターゲットページを透明化し (ステップ 3)，攻撃者のページにある偽のボタンの上にターゲットページにある “Buy” ボタンを配置する (ステップ 4)．被害者は偽のボタンをクリックし，脆弱なサイトにリクエストを送信する (ステップ 5 と 6)．

が用意したウェブサイト (malicious.com) を攻撃者のサイトと呼ぶ．そして，ソーシャルエンジニアリングなどを利用して，攻撃者は被害者をこの攻撃者のサイトに誘導する (ステップ 1)．

攻撃者のサイトにあるページ (攻撃者のページと呼ぶ) は，脆弱なサイト (shopping.com) のターゲットページにある “Buy” ボタンを被害者に操作させるために設計されている．攻撃者のページは被害者にターゲットページを操作させるためにターゲットページを獲得し，そのページを攻撃者のページに埋め込む (ステップ 2)．そして，被害者にターゲットページの存在を認識させないために攻撃者のページはターゲットページを透明化する (ステップ 3)．次に，被害者にターゲットページ内の “Buy” ボタンをクリックさせるために，攻撃者のページに偽のボタンを表示し，この偽のボタンの上にターゲットページの “Buy” ボタンを重ねて配置する (ステップ 4)．

被害者は，攻撃者のページに埋め込まれたターゲットページにある “Buy” ボタンの操作を強要される．被害者が攻撃者のページにある偽のボタンをクリックす

るとき，ターゲットページの “Buy” ボタンがクリックされる (ステップ 5)．なぜなら，ステップ 4 において透明化された “Buy” ボタンが偽ボタンの上に配置されているためである．最後に，被害者はショッピングサイト (shopping.com) で意図しない高額な商品を購入するためのリクエストを送信し (ステップ 6)，攻撃者はこの異常に高額な商品の代金を獲得する．

既存の clickjacking は，被害者を騙すために視覚を利用する手法 (visual clickjacking) と状態を利用する手法 (switchover clickjacking) がある [79]．Visual clickjacking は clickjacking や cursorjacking として広く認知されているために，本論文では visual clickjacking を対象とする．

(1) *Visual clickjacking*: 攻撃者は被害者にブラウザ上のオブジェクトを正しく認識させない．Visual clickjacking は，被害者を騙すための手法に基づいて basic clickjacking と cursorjacking に分類される [79]．Basic clickjacking では，攻撃者が脆弱なページのエレメントを操作することで被害者を騙す．上記に示した例のように，ページ上に透明なページを覆い被せる手法がある．被害者は攻撃者のページを操作していると思っているが，ターゲットページのボタンを操作させられる．

Cursorjacking では，攻撃者がカーソルのフィードバックを操作する事で被害者を騙す．被害者はカーソルフィードバックによってマウスイベントを発行する位置を特定するために，攻撃者は本物のカーソルを隠して偽のカーソルを表示することでカーソルフィードバックを偽る．被害者はカーソルの正しい位置を認識できないために，ターゲットページの意図しないボタンをクリックさせられてしまう．

例えば，攻撃者は被害者を騙すために本物のカーソルの 200 ピクセル右に偽のカーソルを表示し，本物のカーソルを隠す．攻撃者のページは，本物のカーソルを隠すためにカーソルのタイプを制御する CSS カーソルプロパティを利用する．被害者にターゲットボタンをクリックさせるために，ターゲットボタンの 200 ピクセル右に偽のボタンを配置する．被害者は偽のボタンをクリックしていると思っているが，実際はターゲットボタンをクリックさせられてしまう．

(2) *Switchover clickjacking*: 攻撃者は，被害者がクリックしようとするオブジェクトを理解するために十分な時間を与えない．例えば，攻撃者のページは，被害者が偽のボタンをクリックしようとした時に偽のボタンの上にターゲットページのボタンを配置する．被害者は，この状態の変化に反応することができずにターゲットページのボタンをクリックしてしまう．

5.2 Visual clickjacking を防ぐための既存手法

Visual clickjacking を防ぐためにウェブアプリケーションが持つコンテンツの使用を制限する手法とクライアントサイドで攻撃を検出しユーザに警告することで攻撃を防御する手法が提案されている．しかし，コンテンツを制御する手法では，開発者が制御法を正確に指定していることを確認するために検査する必要があり，攻撃を検出する手法では，攻撃として検出されたクリックの処理をユーザが選択するために，攻撃に関する知識のないユーザでは攻撃を防げない可能性があることを説明する．

5.2.1 コンテンツの利用を制限する手法

コンテンツの利用を制限する手法としてよく知られた 1) frame busting と 2) X-Frame-Options を紹介する．これらの防御手法は防御対象のページが他のページに埋め込まれることを防ぐために，開発者はサードパーティのページに防御対象のページが埋め込まれることを許可しながらそのページ内の特定のボタンを無効にできない．このようなページを設計するための手法として element-customized defense を紹介する．

Frame Busting

Frame busting [80] は，visual clickjacking を防ぐために防御対象のページが他のページに埋め込まれることを制限する．なぜなら visual clickjacking において，被害者を騙すために脆弱なページは攻撃者のページに埋め込まれるためである．防御対象のページを他のページに埋め込ませないために，frame busting のコードは防御対象のページが他のページに埋め込まれたときに，ブラウザを防御対象のページにリダイレクトする．防御対象のページにリダイレクトすることで，被害者は防御対象のページを正しく認識することができる．この frame busting を実行するコード (通常は JavaScript で実装される) は，防御対象のページに実装される．

しかし，frame busting のコードを正確に実装することが難しいためにこのコードの脆弱性を検査する必要がある．Frame busting のコードを正確に実装するために開発者には frame busting の回避手法 [27, 80, 81] についての詳細な知識が求められる．Frame busting の正確な実装が困難であることを示すために，5.2.3 で 7 種類の

回避手法を紹介する．Rydstedt らはこれらの回避手法を防ぐための frame busting の実装手法 [27] を示しているものの，実装時に不具合が残る可能性があり同様に検査が必要である．

さらに，Tang らは frame busting を回避する手法があるためにブラウザに frame busting の代わりに X-Frame-Options を実行させる手法 [82] を提案している．レスポンスを監視し，frame busting においてリダイレクトをするためによく利用される “top.location = URI” がレスポンス内に含まれる時に X-Frame-Options を実行する．X-Frame-Options は frame busting を回避する手法に依存せずに防御対象のページを他のページに表示しない．X-Frame-Options の詳細は次項で紹介する．しかしリダイレクトを行うために “top.location = URI” を利用しているものの，Visual clickjacking を防ぐためにリダイレクトを行わない場合に，この手法は false positive を起こす．

Frame busting のポリシーは一つのページ上にある全てのボタンやリンク，フォームに対して適用されるため，ページをサードパーティのページに埋め込みたい場合に利用できない．5.2.1 で示すように，一つのページ上にあるエレメント毎に異なるポリシーを設定したい場合がある．

X-Frame-Options

X-Frame-Options HTTP ヘッダ [83] は，他のページの frame や iframe 内に防御対象のページを表示することを制限する．ブラウザはウェブアプリケーションによって X-Frame-Options HTTP ヘッダがセットされたページを他のページに埋め込まれないために，visual clickjacking を防ぐことができる．X-Frame-Options には，1) DENY，2) SAMEORIGIN や 3) ALLOW-FROM の 3 種類の値を設定できる．DENY は，iframe 内にページを表示することを禁止する．SAMEORIGIN は，生成元が同じページの iframe 内にのみページを表示することを許可する．ALLOW-FROM は，あらかじめ指定された生成元のページの iframe 内にページを表示することを許可する．Frame busting と同様に X-Frame-Options のポリシーはページ毎に適用されるために，X-Frame-Options は一つのページ内の異なるエレメント毎にポリシーを設定したい場合に利用できない．

X-Frame-Options の実装においても些細なミスによって不具合が生じる．Joomla 3.x (コンテンツマネージメントシステムとして幅広く使われている) は，visual clickjacking を防ぐために X-Frame-Options を採用しているものの， 1.1 章で示し

たように，実装のミスによってこの X-Frame-Options は正しく動作しない．

ウェブアプリケーションのフレームワークがサポートしている X-Frame-Options を利用することで攻撃を防ぐことができるものの，X-Frame-Options の正当性を検査する必要がある．なぜなら，開発者はこれらのフレームワークが提供する X-Frame-Options を利用した場合であっても，ポリシーの設定時にミスを犯す可能性があるためである．例えば，Ruby on Rails [84] や django [85] は X-Frame-Options をサポートしているものの，開発者はページ毎にポリシーを設定する必要があるためにミスを犯す可能性がある．

Element-customized defenses

上記に示したように frame busting や X-Frame-Options は，サードパーティのページに防御対象のページを埋め込みたい場合に利用できない．図 5.1 の “Buy” ボタンと “Cart” ボタンを持つ脆弱なページをサードパーティのページに埋め込みたいと仮定する．そして，このページがサードパーティのページに埋め込まれている時，“Buy” ボタンはクリックできない状態にする一方で，“Cart” ボタンは重要な処理を実行しないためにクリックできるようにしたい．このページに対して Frame busting や X-Frame-Options を適用することができない．

このようなページを実現するために，エレメント毎にカスタマイズされた防御手法を実装する必要がある．開発者はエレメント毎に攻撃を防ぐためのスクリプトコードを実装する．例えば，防御対象のページがサードパーティのページに埋め込まれる時，このコードは “Buy” ボタンのような重要なボタンを表示しない．また，他の手法として重要なボタンがクリックされる時，そのリクエストに Cookie を付加しない手法がある．この手法によってユーザを識別するセッション ID が含まれる Cookie が送信されないために，そのリクエストは処理されない．

Element-customized defenses は frame busting や X-Frame-Options よりも正確に実装することが難しいために実装に残る脆弱性を検査する必要がある．なぜなら，ページ上のそれぞれのエレメント毎にサードパーティのページに埋め込まれたときの挙動を制御するコードが必要であるために，開発者は実装時にミスを犯すまたはコードの実装を忘れるためである．また，既存のフレームワークを拡張することで element-customized defenses をサポートすることができる．しかし，フレームワークが提供する防御手法を利用する場合でも，それぞれのボタン毎にポリシーを設定することが求められる．よって，ポリシーの設定の正当性を確認するため

に，ウェブアプリケーションに visual clickjacking を実行して検査する必要である．

InContext [79] や CSP [16]，Nepomnyashy の手法 [86]，App isolation [87]，BetterAuth [88] を利用して，サーバがブラウザ上でのページやコンテンツの振る舞いを指定する場合においても．脆弱性の検査を行う必要がある．InContext は，ページ上の各エレメントの振る舞いを指定する．CSP と Nepomnyashy の手法は，各ページやコンテンツの振る舞いを指定する．App isolation は，ブラウザから送信できるリクエストを指定する．BetterAuth は，セッション ID の付加を許可するサードパティを指定する．これらの手法を採用した場合においても，開発者が振る舞いを指定しているためにミスを犯す可能性がある．

5.2.2 Visual clickjacking を検出する手法

Visual clickjacking を防ぐために攻撃を検出する手法 [89,90,91,92] が提案されている．これらの手法はクライアントサイドで動作することで攻撃時におけるページの視覚的な情報やユーザが獲得したレスポンスを獲得することができるために，これらの情報を利用して攻撃と思われる兆候を検出する．

ClearClick [89] と Clicksafe [90]，CSCP [91] は，basic clickjacking を防ぐためのブラウザの拡張機能である．ClearClick は，basic clickjacking によるボタンの偽装を検出するためにクリックされたボタンのビットマップとこのボタンのみを表示した時のビットマップを比較する．これらのビットマップが異なるなら，ClearClick は basic clickjacking によってこのボタンが偽装されていると判断し，ユーザに警告する．ユーザはクリックによって発行されるリクエストを送信するかしないかを選択する．しかし，この選択は clickjacking に関する知識のないユーザにとって簡単ではない．

そこで，Clicksafe はリクエストを送信するかしないかをより安全に選択させるために過去に他のユーザが行った選択の割合を提供する．この割合は ClearClick のユーザが過去に行った選択のフィードバックに依存している．しかし，ClearClick はこの割合が正しいことを保証できず，悪意あるユーザが正しくない選択をフィードバックとして与えることでユーザに間違った選択を行わせることができる．

CSCP は，隠された Facebook ウィジェットを検出する既存手法を採用し，ユーザに警告を与える．CSCP は，Facebook 以外のウェブアプリケーションに対する basic clickjacking を検出できない．

ProClick [92] は，プロキシを用いて basic clickjacking を検出する防御手法である．Basic clickjacking の兆候を識別するために，ProClick はユーザによって作成されたポリシーに従いリクエストのパラメータとレスポンスを調査する．ProClick のユーザは，ポリシーを設定するために frame busting の回避手法についての詳細な知識が必要である．

また，Balduzzi らの手法 [93] は，世の中にある basic clickjacking を行うためのページを自動的に検出する．この手法は自動的に世の中にあるページを探索し，ページ内にあるボタンをクリックする．そして，クリックに対するページの挙動を解析することで，そのページが攻撃者のページかどうかを決定する．この手法は，攻撃者のページを発見するためのものであり，防御手法の実装の正当性を確認することはできない．

5.2.3 Frame busting の回避手法

Frame busting は，リダイレクトやコード自体を無効化することで回避することができる．この項では，Rydstedt らが紹介している frame busting を回避する 7 種類の手法 [27] について説明する．

double framing

Frame busting のコードがブラウザを防御対象のページにリダイレクトするために parent.location を利用する場合，この frame busting によるリダイレクトは double framing によって無効化される．double framing は，リダイレクトを無効化するために二つの攻撃者のページが frame を利用してターゲットページを入れ子にする．二つの攻撃者のページでターゲットページを入れ子にすることで，frame busting のコード内の parent.location によるリダイレクトは descendant policy によってセキュリティ違反になり，このリダイレクトはブラウザによって無効化される．Descendant policy において，フレームは子孫のフレームのみを異なる URI に誘導することができる．つまり，子孫のフレーム (防御対象のページ内にある frame busting のコード) が親フレームをリダイレクトすることはできない．

onBeforeUnload Event

リダイレクトが取り消された時に frame busting のコードが防御対象のページを表示したままなら，この frame busting は回避されてしまう．攻撃者は，frame busting によるリダイレクトを取り消すために onBeforeUnload ハンドラを登録した攻撃者のページを用意する．この onBeforeUnload ハンドラは，リダイレクトによって攻撃者のページがアンロードされた時に呼び出され，リダイレクトを取り消すかをユーザに尋ねる．ユーザが取り消すことを選んだ場合，ブラウザによってリダイレクトが取り消されて防御対象のページは攻撃者のページに埋め込まれたままとなる．

No-Content Flushing

ヘッドタグまたはボディタグ内の先頭に frame busting のコードが設置されていない場合，この frame busting は回避されてしまう．frame busting のコードがボディタグ内の後尾に設置されている場合，ページがレンダリングされた後に frame busting のコードが評価され，このコードがブラウザをリダイレクトする．リダイレクトを無効化するために，攻撃者のページに onBeforeUnload ハンドラを登録する．上記の通り，このハンドラは frame busting のコードによってリダイレクトがあったときに呼び出され，HTTP/1.1 204 NoContent を返信するサーバにリクエストを送信する．ブラウザは 204 No Content を受け取るとリクエストパイプラインをフラッシュするために，frame busting のコードによるリダイレクトが取り消される．

manipulating Referrer

Frame busting のコードは，document.referrer を確認することであらかじめ指定されたサードパーティのページに防御対象のページを埋め込むことができる．document.referrer は，防御対象のページを要求したページの URI が入っており，この document.referrer に指定されたサードパーティのページの URI が入っていた場合，frame busting のコードはリダイレクトを行わずに，防御対象のページが埋め込まれることを許可する．

もしこの確認が簡単な文字列検索を利用して行われているなら，frame busting のコードは回避されてしまう．なぜなら，攻撃者が検索に当てはまる文字列を document.referrer に埋め込むことができるためである．例えば，防御対象のページ

が safe.com が生成元のページに埋め込まれることを許可する時，攻撃者は document.referrer に safe.com を埋め込んだ URI である attacker.com/page?s=safe.com を入れることができる．

Browser-dependent Approaches

以下の frame busting を回避する 3 つの手法はブラウザ特有の挙動を利用しているために脆弱性のない frame busting の実装をより困難にしている．さらに，このブラウザ特有の挙動はバージョンによっても異なり，同じ種類のブラウザであっても frame busting を回避できるバージョンとできないバージョンがある．

clobbering location variable: 攻撃者のページは，frame busting のコードを回避するために他のページのローカル変数へのアクセスによるセキュリティ違反を利用する．Frame busting のコードは，親ページが防御対象のページの埋め込みを許可されたページかを確認するためにグローバル変数である top.location (親ページのオリジン) にアクセスする．このような frame busting のコードを無効化するために，攻撃者のページはローカル変数として location 変数を再定義する．Frame busting のコードがローカル変数として再定義された top.location にアクセスしたとき，frame busting のコードはセキュリティ違反のために無効化される．この回避手法は IE7 や Safari 4.0.4 上で実行することができる．

restricting JavaScript: JavaScript で実装されている frame busting のコードは，JavaScript が無効化されることで動作しなくなる．Firefox や Chrome は iframe タグの sandbox 属性を指定することで iframe 内の JavaScript コードを無効化する．そのために iframe の中に frame busting のコードを含んだ防御対象のページが読み込んだ時，JavaScript コードである frame busting のコードは無効化される．また IE の security 属性に “restricted” を指定したり，IE8 や Firefox の designMode プロパティをオンにすることで iframe 内の frame busting のコードを無効化することができる．

XSS filter: Frame busting のコードを無効化するために，攻撃者のページは IE8 や Chrome の XSS フィルタを悪用する．XSS フィルタはリクエストとそのリクエストに対するレスポンスの両方に含まれるスクリプトコードを無効化する．よって攻撃者が frame busting のコードを抜き取り，防御対象のページをリクエストするための URI に埋め込むことで，防御対象のページ内の frame busting のコードは XSS フィルタによって無効化される．

5.3 Visual clickjacking の実行による検査手法

前節で示したように，visual clickjacking に対する防御手法を正確に実装するためにいくつかの注意点がある．Clickjuggler は，防御手法の実装の正当性や脆弱性を検査するために，ウェブアプリケーションのロジックに依存する visual clickjacking を自動的に実行する．Visual clickjacking を実行するために開発者から獲得した検査対象のボタンに関するロジックを利用して攻撃者のページを生成し，被害者としてこのページに埋め込まれた検査対象のボタンを操作する．そして，Clickjuggler によって実行された攻撃が成功したかを確認することで脆弱性を検査する．

Clickjuggler は，幅広く visual clickjacking や frame busting を回避する手法に対応している．現在の Clickjuggler は，basic clickjacking と cursorjacking，frame busting の 6 つの回避手法に対する脆弱性を検査することができる．一方で，既存ツールである CJTool と BeEF のプラグインは攻撃者のページの生成をサポートするものの，cursorjacking を対象としておらず，frame busting の 1 つの回避手法にしか対応していない．

5.3.1 Clickjuggler の概要

Clickjuggler は visual clickjacking を実行するために情報収集フェーズと検査フェーズの 2 つのフェーズを実行する．情報収集フェーズにおいて，Clickjuggler は被害者として検査対象のボタンを操作するために，このボタンのロジックに関する情報を収集する．例えば，この情報は検査対象のページ (検査対象のボタンを持つページ) の URI や検査対象のボタンの位置座標や幅，高さ，検査対象のボタンを操作するためのイベントである．

Clickjuggler は，ウェブアプリケーションのテストフェーズにおいて開発者が検査対象のボタンを操作している間にこれらの情報を収集する．図 5.2 に示すように，テストフェーズにおいて開発者は検査対象のボタンの動作を確認するために検査対象のページを開き，検査対象のボタンを操作する (ステップ 1)．このとき，ブラウザは検査対象のページを要求し，開発者による検査対象のボタンの操作に対するリクエストを発行する (ステップ 2 と 3)．Clickjuggler は，これらの操作を通して検査対象のボタンのロジックに関する情報を獲得する (ステップ 4)．そして，開発者からウェブアプリケーション特有の情報を獲得する (ステップ 5)． 5.3.3 で開発者から獲得する情報の詳細を示す．

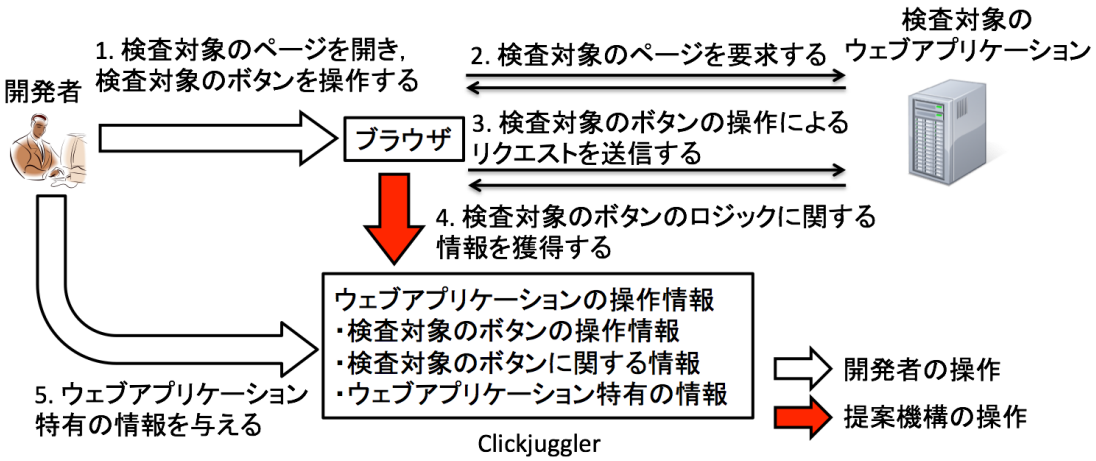

図 5.2: 情報収集フェーズ

開発者は動作確認のために検査対象のページを開き，このページにある検査対象のボタンを操作する (ステップ 1)．ブラウザは，これらの操作に対して検査対象のページを要求し，リクエストを送信する (ステップ 2 と 3)．Clickjuggler は，開発者の操作から検査対象のボタンのロジックに関する情報を獲得する (ステップ 4)．開発者は，Clickjuggler に検査対象のウェブアプリケーション特有の情報を与える (ステップ 5)．

検査フェーズにおいて，Clickjuggler はウェブアプリケーションに visual clickjacking を実行し，攻撃結果を解析することで脆弱性の有無を判断する．図 5.3 に示すように，攻撃の手順に従って検査対象のウェブアプリケーションに visual clickjacking を実行する (ステップ 6)．攻撃を実行するために Clickjuggler は開発者から獲得した検査対象のボタンに関する情報を利用して攻撃者のページを生成する (ステップ 7)．攻撃者のページは，情報収集フェーズに獲得した URI を利用して，検査対象のページを攻撃者のページに埋め込む (ステップ 8)． 5.3.4 で攻撃者のページの生成方法の詳細を示す．

そして，Clickjuggler は被害者として攻撃者のページに埋め込まれた検査対象のボタンを操作する．Clickjuggler はこのボタンを操作するために情報収集フェーズに獲得した情報を利用する (ステップ 9)．この情報を利用して攻撃者のページにマウスイベントを発行することで，攻撃者のページに埋め込まれた検査対象のボタンを操作する (ステップ 10)．このときブラウザは，Clickjuggler による操作に対するリクエストを送信する (ステップ 11)．Clickjuggler は，このリクエストに対するレスポンスと攻撃時における検査対象のボタンに関する情報を獲得する (ステップ 12)．

Clickjuggler はステップ 12 において獲得したレスポンスと検査対象のボタンに

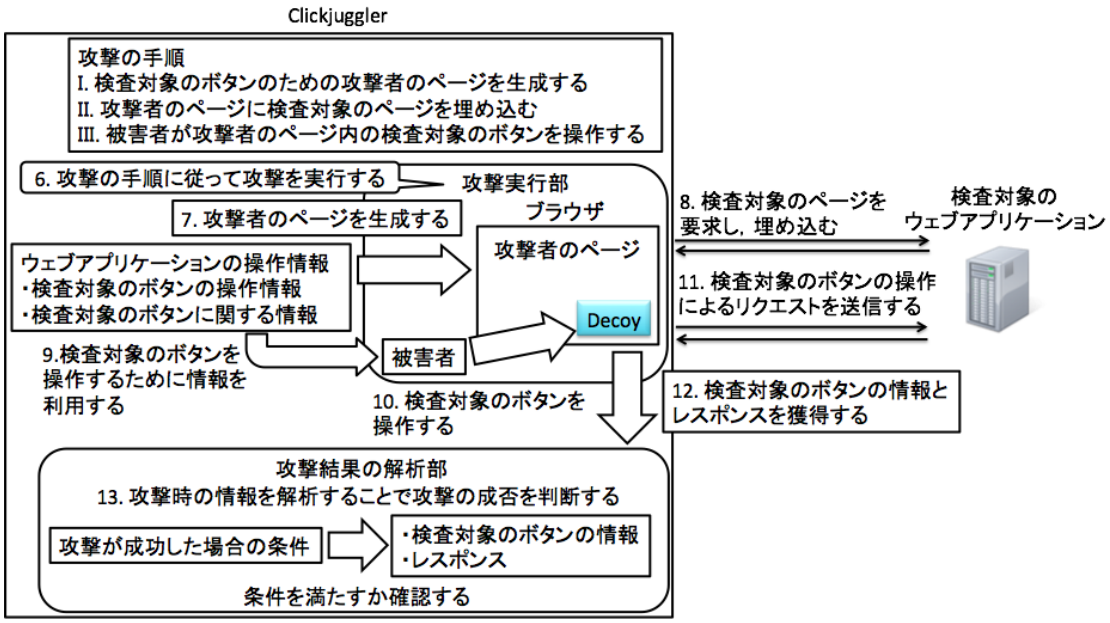

図 5.3: 検査フェーズ

Clickjuggler は，攻撃の手順に従って visual clickjacking を実行する (ステップ 6)．攻撃を実行するために，情報収集フェーズで獲得した情報から攻撃者のページを生成する (ステップ 7)．この攻撃者のページは，検査対象のページを要求し，このページを攻撃者のページに埋め込む (ステップ 8)．Clickjuggler は情報収集フェーズで獲得した情報をもとに攻撃者のページに埋め込まれた検査対象のボタンを操作する (ステップ 9 と 10)．このとき，ブラウザは検査対象のボタンの操作に対するリクエストを送信する (ステップ 11)．Clickjuggler は，ステップ 11 におけるレスポンスと攻撃時における検査対象のボタンに関する情報を獲得し (ステップ 12)，これらの情報を解析することで実行した攻撃が成功したかを確認する (ステップ 13)．

関する情報を解析することで実行した visual clickjacking が成功したかを確認する (ステップ 13)．攻撃が成功したかを判断するために，獲得したレスポンスと検査対象のボタンに関する情報が攻撃が成功した場合の条件を満たすかを確認する．攻撃時における検査対象のボタンに関する情報と成功した場合の条件の詳細は，5.3.5 で説明する．

5.3.2　開発者に要求する操作

Clickjuggler は，2 つのフェーズを実行するために開発者に 2 つの操作を要求する．1 つ目は，情報収集フェーズにおいてウェブアプリケーション特有の情報として特別なキーワードを Clickjuggler に与える．このキーワードは検査対象のボタンがクリックされることで得られるページに現れる文字列であり，5.3.5 に示すよう

にこのキーワードを利用して攻撃結果を解析する．例えば，検査対象のボタンが商品を購入するためのボタンである場合，“Thank you for purchase” のようなキーワードを与える．

2 つ目は，検査フェーズにおいて確認ダイアログの操作を要求する．なぜなら，5.4.3 節に示すように実装の都合により Clickjuggler が確認ダイアログを操作できないためである．この 2 つの操作はウェブアプリケーションの開発者であれば困難ではない．5.5 節の実験において，ウェブアプリケーションの開発者でない著者でも 2 つの操作を行うことができた．

開発者はこれらの操作を要求されるものの，手動での visual clickjacking の脆弱性検査は Clickjuggler によって自動化される．開発者は，攻撃者のページを生成するために検査対象のボタンについての情報を収集する必要がなく，frame busting の回避手法に関する知識が必要ない．また，数種類の攻撃者のページを生成することや実際の攻撃を行うこと，攻撃毎に成否を確認することも必要ない．

5.3.3 検査対象のボタンのロジックに関する情報

Clickjuggler は，開発者が検査対象のボタンを操作している間に検査対象のボタンのロジックに関する情報を獲得する．図 5.4 は Clickjuggler が収集する情報の種類を示しており，この図で検査対象となっているウェブアプリケーションは図 5.1 と同じものである．開発者が検査対象のボタンの動作確認を行うために，検査対象のページを開いた時に Clickjuggler は開かれたページの URI を収集する．図 5.4 において開発者が購入ページを開いた時に Clickjuggler は URI “http://shopping.com/purchase” を保存する (ステップ 1 と 2).

次に，開発者は検査対象のページを開いた後に検査対象のボタンやフォームの動作確認のためにボタンをクリックしたり，フォームの入力欄にデータを入力する．Clickjuggler は，クリックされたボタンの位置座標や幅，高さ，クリックされたボタンを操作するために発行されたイベントを保存する．図 5.4 において，開発者は “Buy” ボタンをクリックする (ステップ 3)．Clickjuggler は “Buy” ボタンの位置座標や幅，高さを保存する (ステップ 4)．攻撃時において検査対象のボタンを操作するために，開発者がどのオブジェクトにどのような操作が行われたかも保存する．例えば，チェックボックスの項目がクリックされた時，そのオブジェクトに行われた操作は保存される．

さらに Clickjuggler は開発者によってクリックされたボタンやクリックされた

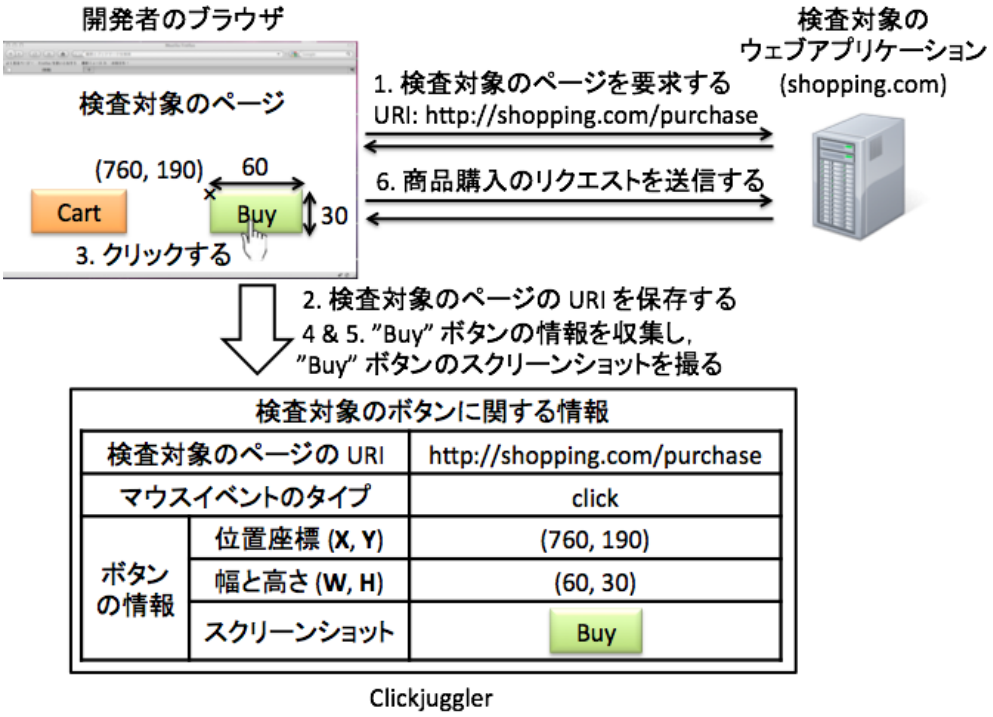

図 5.4: 開発者から獲得する情報

開発者は検査対象のページを要求し (ステップ 1)，Clickjuggler は開発者が要求したページの URI を保存する (ステップ 2)．開発者は検査対象のボタンをクリックし (ステップ 3)，Clickjuggler はクリックしたボタンについての情報とそのボタンのスクリーンショットを保存する (ステップ 4 と 5)．開発者のクリックによりリクエストが発行される (ステップ 6)．

フォームのスクリーンショットを撮る．これらのスクリーンショットは攻撃が成功したかを確認するために使用する．図 5.4 において，Clickjuggler は “Buy” ボタンのスクリーンショットを撮る (ステップ 5)．

5.3.4 攻撃者のページの生成

Clickjuggler は visual clickjacking を実行するために数種類の攻撃者のページを生成する．このページは，Clickjuggler の開発者によってあらかじめ用意された HTML のテンプレートに基づいて生成される．これらのテンプレートは，検査対象のボタン毎に位置座標などの情報が異なるために検査対象のボタンについての情報を持たず，被害者を騙す方法または frame busting を回避するための方法が実装されている．Clickjuggler はこのテンプレートに情報収集フェーズで獲得した検査対象のボタンについての情報を適用することで，さまざまな攻撃者のページを

生成する．

Clickjuggler は basic clickjacking と cursorjacking のための 2 つのテンプレートと frame busting を回避するための 6 つのテンプレートをあらかじめ用意している．Basic clickjacking と cursorjacking のテンプレートから 5.1 節で紹介した basic clickjacking と cursorjacking を実行する攻撃者のページを生成する．6 つのテンプレートから 5.2.3 で紹介した回避手法の内の 1 つを実行する攻撃者のページを生成する．

これらのテンプレートは basic clickjacking の脆弱性や cursorjacking の脆弱性，不完全な防御手法の実装を見つけ出すことができる．あるページに visual clickjacking に対する防御手法が実装されていなければ，basic clickjacking と cursorjacking のテンプレートによって脆弱性が発見される．あるページに不完全な frame busting が実装されていたならば，6 つのテンプレートの内の少なくとも 1 つのテンプレートによって脆弱性が発見される．あるページに不完全な X-Frame-Options または不完全な element-customized defenses が実装されていたならば，basic clickjacking と cursorjacking のためのテンプレートによって脆弱性が発見される．

また，新たな visual clickjacking の手法や frame busting を回避する手法が出現しても，この visual clickjacking のために新しいテンプレートを用意することで Clickjuggler は検査できるようになる．なぜなら，visual clickjacking 毎の違いは被害者を騙す方法または frame busting を回避する方法であり，これらの手法がテンプレートに実装されているためである．

テンプレートから生成される攻撃者のページは検査対象のページと異なるオリジンである必要がある．これは，いくつかの防御手法 (例: X-Frame-Options) がウェブページのオリジンに依存しているためである．そこで，Clickjuggler はテンプレートをローカルファイルとして保存することで，このテンプレートから生成される攻撃者のページのオリジンは検査対象のページのオリジンと異なる．

5.3.5 攻撃結果の解析方法

実行した visual clickjacking が成功したかを判断するために，Clickjuggler はテンプレート毎に攻撃が成功した場合の条件を満たしているかを確認する． 5.1 節で紹介したように basic clickjacking は検査対象のページ上にある検査対象のエレメントを操作しているのに対して，cursorjacking はカーソルのフィードバックを操作しているために basic clickjacking が成功した場合の条件と cursorjacking の条

件は異なる．以下において basic clickjacking の条件と cursorjacking の条件をそれぞれ説明する．

Basic clickjacking の条件

Basic clickjacking が成功したかを判断するために，Clickjuggler は二つの条件が満たされていることを確認する．1 つ目の条件は，攻撃者のページに埋め込まれたときに検査対象のボタンが透明化されていることである．この条件を確認するために，Clickjuggler は攻撃者のページに埋め込まれていない検査対象のボタンと埋め込まれている検査対象のボタンを比較する．そのために，Clickjuggler は情報収集フェーズにおいて開発者が検査対象のボタンを操作しているときの検査対象のボタンのスクリーンショットと検査フェーズにおいて攻撃者のページに埋め込まれたときの検査対象のボタンのスクリーンショットを獲得する．

2 つ目の条件は，偽のボタンが表示されている箇所にクリックイベントが発行された時に検査対象のボタンがクリックされることである．検査対象のボタンがクリックされたことを確認するために，3 つの手法が考えられる．

手法 1: Clickjuggler は，ボタンがクリックされた後にリクエストが送信されたサイトを確認する．リクエストが送信されたサイトが検査対象のページのサイトであれば，Clickjuggler は検査対象のボタンがクリックされたと判断する．この手法では，検査対象のボタンに 5.2.1 で示した Cookie を送信しない防御手法が実装されていた場合に，検査対象のボタンに脆弱性があると間違った判断を下す．

手法 2: Clickjuggler は，ボタンがクリックされたことで獲得したレスポンス内のメッセージボディを確認する．このメッセージボディが情報収集フェーズに獲得したメッセージボディと同じであれば，Clickjuggler は検査対象のボタンがクリックされたと判断する．これは，検査対象のボタンをクリックすることで，サイトが情報収集フェーズに獲得したメッセージボディを返すためである．しかし，この手法は常にレスポンスのメッセージボディが変化する場合に正しく動作しない．

手法 3: 手法 2 と同様に，Clickjuggler はボタンがクリックされたことで獲得したレスポンス内のメッセージボディを確認する．メッセージボディが変化する場合に対応するために，Clickjuggler は 5.3.2 で示した特別なキーワードを利用してメッセージボディを識別する．この特別なキーワードは検査対象のボタンがクリックされた時にページに現れる文字列であり，このキーワードがメッセージボディ内にあるときに検査対象のボタンがクリックされたと判断する．5.3.2 で示したよう

に，開発者は情報収集フェーズに検査対象のボタンがクリックされた時にページに現れるこの特別なキーワードを指定する．

Clickjuggler は，手法 1 と手法 2 が特定の防御手法やページに対して false positive や negative を起こすために手法 3 を採用している．

Cursorjacking の条件

cursorjacking が成功したことを判断するために，Clickjuggler は 2 つの条件を確認する．1 つ目の条件は，被害者に本物のカーソルの正確な位置を認識させないために偽のカーソルが表示されていることである．この条件を確認するために，Clickjuggler は偽のカーソルが移動する前と後の攻撃者のページ内にある偽のボタンのスクリーンショットを比較する．このスクリーンショットが異なる時、偽のカーソルが表示されている．

2 つ目の条件は，攻撃者のページが被害者を視覚的に騙せる状況で検査対象のボタンがある位置にクリックイベントが発行された時に検査対象のボタンがクリックされることである．この条件を確認するために，Clickjuggler は basic clickjacking と同様の方法を利用して検査対象のボタンがクリックされたことを確認する．

Clickjuggler は，本物のカーソルを認識させないために本物のカーソルが隠されていることを確認しない．これは，もし本物のカーソルが表示されていたとしても，被害者が偽のカーソルに注目していれば偽のカーソルで偽のボタンをクリックしようとして cursorjacking が成功するためである．

また，本物のカーソルの正しい位置をユーザに注目させるための手法 [79] があるものの，cursorjacking を確実に防ぐことが難しい．これは，注目させたとしてもその後に偽のカーソルで偽のボタンをクリックするかどうかは被害者に委ねられるために cursorjacking が成功する可能性がある [79] ためである．Clickjuggler は，このような手法が実装された検査対象のボタンに脆弱性があると判断する．

5.4 実装

Clickjuggler のプロトタイプは 2 種類のバージョンの Firefox (20.0.1 と 3.6.8) に追加するプラグインとして実装する．これは，Firefox (20.0.1) が X-Frame-Options HTTP ヘッダをサポートしており，Firefox (3.6.8) がサポートしていないためである．ブラウザのプラグインであれば，ページ上にあるエレメントについての情報

表 5.1: Web API interfaces

Web API interface	Summary [94]
window.content	Returns a reference to the content element in the current window
document.URL	Returns the string URL of the HTML document
Element.getBoundingClientRect	Returns a text rectangle object that encloses a group of text rectangles
document.createEvent	Creates an event of the type specified
event.initMouseEvent	Initializes the value of a mouse event once it's been created
EventTarget.dispatchEvent	Dispatches an Event at the specified EventTarget, invoking the affected EventListeners in the appropriate order
EventTarget.addEventListener	Registers the specified listener on the EventTarget it's called on

やエレメントに発行されたイベント，スクリーンショットなどを獲得することができる．例えば，情報収集フェーズにおいて開発者が操作した検査対象のボタンについての情報や検査対象のボタンを操作するために発行したイベントを獲得することができる．また，プラグインは攻撃フェーズにおいて visual clickjacking を実行するためにブラウザを介して攻撃者のページに埋め込まれた検査対象のページにマウスイベントを発行することができる．

Clickjuggler の実装において Firefox の API を利用したものの，この実装に特殊な API を利用していないために Chrome や IE，Safari のようなブラウザのプラグインとしても実現できると考えている．表 5.1 に Clickjuggler の実装に利用した API を示す．

5.3.4 で示したように，攻撃者のページを生成するために Clickjuggler は basic clickjacking と cursorjacking のための 2 つのテンプレートと frame busting を回避するための 6 つのテンプレートを利用する．あらかじめ用意したテンプレートを以下に示す．

1. Basic clickjacking のテンプレート
2. Cursorjacking のテンプレート
3. double framing のテンプレート

```
1 <IFRAME style="opacity:0;" src="URI">
2 </IFRAME>
3 <BUTTON style="left:X; top:Y;
4              width:WIDTH; height:HEIGHT;
5              position:absolute; z-index:-1;">Decoy
6 </BUTTON>
```

図 5.5: Basic clickjacking のテンプレート

4. onBeforeUnload event のテンプレート

5. No-Content flushing のテンプレート

6. manipulating Referrer のテンプレート

7. restricting JavaScript with sandbox のテンプレート

8. restricting JavaScript with designMode のテンプレート

このように Clickjuggler は，security 属性を利用した restricting Javascript や clobbering location variable，XSS フィルタの 3 つの回避手法以外の回避手法をサポートしていない．なぜなら，これらの 3 つの回避手法は Chrome や IE，Safari 上で動作する回避手法だからである．以降の項では，basic clickjacking と cursorjacking のための 2 つのテンプレートと回避手法のための 6 つのテンプレートを紹介し，Chrome や IE，Safari 上で動作する回避手法について議論する．

5.4.1 Basic clickjacking のテンプレート

図 5.5 に示す basic clickjacking のテンプレートから生成される攻撃者のページは透明化した検査対象のボタンを偽のボタンの上に設置する．このテンプレートの iframe タグは URI から検査対象のページを読み込むために利用される (1 と 2 行目)．読み込まれた検査対象のページを透明化するために，iframe タグの style 属性に opacity を設定している．

button タグは，偽のボタンを座標 (X, Y) に 幅 WIDTH と 高さ HEIGHT で設置するために利用される (3 行目から 6 行目)．このテンプレートから攻撃者のページを作成するために，Clickjuggler は情報収集フェーズで獲得した情報を URI や X,

```
1   <BODY style="cursor: none;">
2   <IMG src="cursor.png"/>
3   <SCRIPT>
4     var move = function(e){
5       a = coordinate x of the real cursor + 200;
6       b = coordinate y of the real cursor;
7       // 座標 (a, b) に偽のマウスを移動させる
8     };
9     // 継続的に mousemove イベントをキャッチする
10    document.body
11          .addEventListener('mousemove', move, true);
12  </SCRIPT>
13  <BUTTON style="left:X+200; top:Y;
14                 width:WIDTH; height:HEIGHT;
15                 position:absolute; z-index:-1;">Decoy
16  <BUTTON>
17  <IFRAME src="URI"></IFRAME>
18  <DIV style="..."></DIV>...<DIV style="..."></DIV>
19  </BODY>
```

図 5.6: Cursorjacking のテンプレート

Y，HEIGHT，WIDTH に適用する．このように適用することで，偽のボタンが検査対象のページにある検査対象のボタンの位置に表示される．

5.4.2 Cursorjacking のテンプレート

図 5.6 に示す cursorjacking のテンプレートから生成された攻撃者のページは，本物のカーソルを隠し，偽のカーソルと偽のボタン，検査対象のボタンを表示する．本物のカーソルを隠すために，body タグの style 属性に cursor:none を設定する (1 行目)．img タグは，ページ上に表示する偽のカーソルのビットマップを読み込むために利用される (2 行目)．script タグ内の JavaScript コードは，読み込んだ偽のカーソルのビットマップを特定の座標 (本物のカーソルの x 座標 $+200$, 本物のカーソルの y 座標) に移動させる (3 行目から 12 行目)．

button タグは，偽のボタンを座標 (X + 200, Y) に 幅 WIDTH と高さ HEIGHT で設置するために利用される (13 行目から 16 行目)．iframe タグは URI から検査対

象のページを読み込むために利用される (17 行目)．div タグは，被害者に検査対象のページを認識させないために検査対象のボタン以外の検査対象のページを覆い隠す (18 行目)．Basic clickjacking のテンプレートと同様に，Clickjuggler は情報収集フェーズに獲得した情報を URI や X，Y，HEIGHT，WIDTH に適用する．

5.4.3 回避手法のテンプレート

それぞれのテンプレートは，回避手法を利用した攻撃者のページを生成するために basic clickjacking のテンプレートを拡張する．double framing のテンプレートは，検査対象のページを 2 つの攻撃者のページで入れ子にするために二つのテンプレートで構成される．1 つ目のテンプレートはフレーム内に 2 つ目のテンプレートを読み込み，2 つ目のテンプレートは検査対象のページを読み込む．

onBeforeUnload event のテンプレートは，frame busting のコードによって行われるリダイレクトを取り消す．リダイレクトを取り消すために，攻撃者のページは onBeforeUnload ハンドラを登録する．このハンドラは，確認ダイアログを用いてリダイレクトを取り消すかをユーザに尋ねる．よって，basic clickjacking のためのテンプレートに以下のコードを追加する．

```
window.onbeforeunload=function(){
                        return "Do you want to exit?";}
```

Clickjuggler は，検査フェーズにおいて確認ダイアログを操作することでリダイレクトの取り消しを開発者に要求する．JavaScript によって実装された Clickjuggler は確認ダイアログを操作できないために，リダイレクトの取り消しを行えない．

図 5.7 に示すように，No-Content flushing のテンプレートは frame busting のコードによるリダイレクトを取り消す．リダイレクトを取り消すために，攻撃者のページは onBeforeUnload ハンドラを登録する．このハンドラは，onBeforeUnload event のテンプレートのようにユーザに操作を求めずにリダイレクトを取り消す．まず，frame busting のコードによるリダイレクトを検知する (3 行目から 5 行目)．そして，このテンプレートはリダイレクトが実行されるかどうかを継続的に監視し，リダイレクトが実行される時に HTTP/1.1 204 No Content を返すサーバにリクエストを発行する (7 行目から 13 行目)．

manipulating Referrer のテンプレートは document.referrer を操作する．Clickjuggler は，frame busting のコードが document.referrer を調べるために，簡単な文字列検索を利用しているか確認する．Frame busting のコードを確認するために，Click-

```
1  var prevent_bust = 0;
2  // リダイレクトをキャッチするためのイベントハンドラ
3  window.onbeforeunload = function(){
4    prevent_bust++
5  };
6  // リダイレクトが起こったかどうかを監視する
7  setInterval(function(){
8    if(prevent_bust > 0){
9    prevent_bust -= 2;
10    // ``No Content'' を獲得する
11    window.top.location = 'http://no-content-204.com';
12   }
13 }, 1);
```

図 5.7: No-Content Flushing のテンプレート

juggler は document.referrer に割り当てられるこのテンプレートの URI に検査対象のページの URI を埋め込む．これは，開発者が自身のサイトにあるページを同じサイトのページに埋め込むことを許可すると仮定しているためである．

restricting JavaScript with sandbox のテンプレートは iframe 内の JavaScript のコードの実行を禁止する．攻撃者のページは JavaScript コードとして実装された frame busting の実行を防ぐために iframe タグに sandbox 属性を指定する．よって，basic clickjacking のためのテンプレートの iframe タグに sandbox 属性を追加する．さらに，iframe 内にあるフォームからリクエストを送信することを許可するために sandbox 属性の値として “allow-forms” を設定する．

```
<IFRAME style="opacity:0;"
                sandbox=”allow-forms” src="URI"></IFRAME>
```

restricting JavaScript with designMode のテンプレートは iframe 内の JavaScript のコードの実行を禁止する．攻撃者のページは iframe タグの designMode プロパティの値を “on” に設定することで，iframe 内の frame busting のコードを無効化する．以下のコードを basic clickjacking のためのテンプレートに追加することで iframe の designMode プロパティを “on” に設定する．

```
<IFRAME style="opacity:0;" id="tgt" src="URI"></IFRAME>
document.getElementById("tgt").
                        contentDocument.designMode="on";
```

5.4.4 Chrome，IE，Safari における回避手法

Chrome や IE，Safari のプラグインとして Clickjuggler を実装することで，Clickjuggler は clobbering location variable と restricting JavaScript with security attribute の 2 種類の回避手法を用いて visual clickjacking の脆弱性を検査できると考えている．Clickjuggler は，2 種類のテンプレートからこの 2 種類の回避手法を利用する攻撃者のページを作成することができる．

これらのテンプレートは，上記に示した回避手法のテンプレートと同様に basic clickjacking のためのテンプレートを拡張する．clobbering location variable のテンプレートは location 変数を操作する．攻撃者のページは，セキュリティ違反により frame busting のコードを無効化するためにローカル変数として location 変数を再定義する．よって，basic clickjacking のためのテンプレートに以下のコードを追加する．

```
<SCRIPT> var location="clobbered" </SCRIPT>
```

restricting JavaScript with security attribute のテンプレートは，iframe 内の JavaScript コードの実行を禁止する．攻撃者のページは，iframe タグの security 属性の値として “restricted” を設定することで iframe 内の frame busting のコードを無効化する．よって，basic clickjacking のためのテンプレートの iframe タグに security 属性を追加する．

```
<IFRAME style="opacity:0;"
                    security=”restricted” src="URI"></IFRAME>
```

Chrome と IE のプラグインとして実装された Clickjuggler は XSS フィルタによる回避手法を利用した visual clickjacking を実行することで脆弱性を検査することは難しい．なぜなら，Clickjuggler の現在の実装ではテンプレートから XSS フィルタを利用した攻撃者のページを生成することができないためである．この理由については 5.5.3 で議論する．

5.5 実験

Clickjuggler の有用性を示すために，Clickjuggler の検査結果の正当性とパフォーマンスを評価する．検査結果の正当性を評価するために，次項で Clickjuggler が実際にさまざまなサイトで利用されているオープンソースのウェブアプリケーション

に残った visual clickjacking の脆弱性を正確に検査できることを確認する．パフォーマンスを評価するために，5.5.2 で Clickjuggler と既存ツールの検査時間を比較した．さらに，5.5.3 で Clickjuggler の制約について議論する．

検査対象のウェブアプリケーションは MAMP マシン (Mac OS X version 10.8.2 と Apache 2.2.23，MySQL 5.5.29，PHP 5.2.17) 上で動作する．MAMP マシンと 2 種類のバージョンの Firefox (20.0.1 と 3.6.8) は 16 GB メモリで 2.7 GHz Intel Core i7 と Mac OS X version 10.8.2 上で動作する．Clickjuggler のプロトタイプを Firefox 20.0.1 と 3.6.8 にインストールする．

5.5.1 検査結果の正当性

Joomla [75] や Roundcube [95]，MediaWiki [96]，WordPress [97] の実際にサイトで利用されている 4 つのウェブアプリケーションに Clickjuggler を適用することで評価する．これらのウェブアプリケーションは広く利用されていて，例えば，Joomla と Roundcube，WordPress はそれぞれ 3500 万回以上と 200 万回以上，7000 万回以上ダウンロードされている．visual clickjacking の脆弱性を検査するために，2 種類のバージョンの Firefox (20.0.1 と 3.6.8) を利用する．5.4 節に示したように，Firefox (20.0.1) は X-Frame-Options HTTP ヘッダをサポートしており，Firefox (3.6.8) はサポートしていないためである．また，cursorjacking を検査するために，検査対象のボタンとして 4 つのウェブアプリケーションのリンクやボタンを選択した．これは，cursorjacking のテンプレートがリンクやボタンを対象としているためである．

表 5.2 は Clickjuggler の検査結果をまとめたものである．表 5.2 の “ウェブアプリケーション” は検査対象のウェブアプリケーションの名前とバージョンを示している．“ユーザ” は “検査対象のボタン” を普段利用しているユーザを示しており，“検査対象のボタン” は実験において検査対象としたリンクやボタン，フォームを示している．

この表 5.2 の “脆弱性” と “Clickjuggler の検査結果” から Clickjuggler は 4 つのオープンソースのウェブアプリケーションの脆弱性を正確に検査できることがわかる．“脆弱性” は検査対象のリンクやボタン，フォームが脆弱であるかを示している．この visual clickjacking の脆弱性を発見するために検査対象のリンクやボタン，フォームを手動で調査した．そして，[72] や [98] のような脆弱性リポジトリや wiki [99]，リリースノート [100] を使って検査対象のウェブアプリケーション

表 5.2: Clickjuggler の検査結果

ウェブアプリケーション	ユーザ	検査対象のボタン	脆弱性	Clickjuggler の検査結果
Joomla 1.6.1	ユーザ	リンク	vul.	vul.
		プロフィールの編集	vul.	vul.
	管理者	リンク	vul.	vul.
		記事の編集	vul.	vul.
		ユーザ情報の編集	vul.	vul.
Joomla 2.5.7	ユーザ	リンク	vul.	vul.
		プロフィールの編集	vul.	vul.
	管理者	リンク	vul.	vul.
		記事の編集	vul.	vul.
		ユーザ情報の編集	vul.	vul.
Joomla 3.0.2	ユーザ	リンク	vul.	vul.
		記事の作成	vul.	vul.
		プロフィールの編集	No vul.	No vul.
	管理者	リンク	vul.	vul.
		記事の編集	vul.	vul.
		ユーザ情報の編集	vul.	vul.
Roundcube 0.4.1	ユーザ	リンク	vul.	vul.
		設定の編集	vul.	vul.
Roundcube 0.7.0	ユーザ	リンク	No vul.	No vul.
		設定の編集	No vul.	No vul.
Roundcube 0.7.0 (Firefox 3.6.8)	ユーザ	リンク	No vul.	No vul.
		設定の編集	No vul.	No vul.
MediaWiki 1.16.0	ユーザ	リンク	vul.	vul.
		記事の編集	vul.	vul.
	管理者	リンク	vul.	vul.
		記事の編集	vul.	vul.
		ユーザ情報の編集	vul.	vul.
WordPress 3.1.2	ユーザ	リンク	vul.	vul.
		コメントの投稿	vul.	vul.
	管理者	リンク	vul.	vul.
		ユーザの追加	vul.	vul.

“vul.” は visual clickjacking の脆弱性があることを示しており，“No vul.” は visual clickjacking の脆弱性がないことを示している．

に残った visual clickjacking の脆弱性を探して，手動での調査結果の確認を行った．“Clickjuggler の検査結果” は，Clickjuggler から得られた検査結果を示している．こ

表 5.3: Clickjuggler の各テンプレートの攻撃結果

ウェブアプリケーション	ユーザ	検査対象のボタン	脆弱性	テンプレートの攻撃結果*							
				basic	1	2	3	4	5	6	cursor
Joomla 1.6.1	ユーザ	リンク	vul.	X	X	X	X	X	X	X	X
		プロフィールの編集	vul.	X	X	X	X	X	X	X	-
	管理者	リンク	vul.	X	X	X	X	X	X	X	X
		記事の編集	vul.	X	X	X	X	X	✓	✓	-
		ユーザ情報の編集	vul.	X	X	X	X	X	✓	✓	-
Joomla 2.5.7	ユーザ	リンク	vul.	X	X	X	X	X	X	X	X
		プロフィールの編集	vul.	X	X	X	X	X	X	X	-
	管理者	リンク	vul.	X	X	X	X	X	X	X	X
		記事の編集	vul.	X	X	X	X	X	✓	✓	-
		ユーザ情報の編集	vul.	X	X	X	X	X	✓	✓	-
Joomla 3.0.2	ユーザ	リンク	vul.	X	X	X	X	X	X	X	X
		記事の作成	vul.	X	X	X	X	X	✓	✓	-
		プロフィールの編集	No vul.	✓	✓	✓	✓	✓	✓	✓	-
	管理者	リンク	vul.	X	X	X	X	X	X	X	X
		記事の編集	vul.	X	X	X	X	X	✓	✓	-
		ユーザ情報の編集	vul.	X	X	X	X	X	✓	✓	-
Roundcube 0.4.1	ユーザ	リンク	vul.	X	X	X	X	X	✓	✓	X
		設定の編集	vul.	X	X	X	X	X	✓	✓	-
Roundcube 0.7.0	ユーザ	リンク	No vul.	✓	✓	✓	✓	✓	✓	✓	✓
		設定の編集	No vul.	✓	✓	✓	✓	✓	✓	✓	-
Roundcube 0.7.0 (Firefox 3.6.8)	ユーザ	リンク	No vul.	✓	✓	✓	✓	✓	✓	✓	✓
		設定の編集	No vul.	✓	✓	✓	✓	✓	✓	✓	-
MediaWiki 1.16.0	ユーザ	リンク	vul.	X	X	X	X	X	X	X	X
		記事の編集	vul.	X	X	X	X	X	✓	✓	-
	管理者	リンク	vul.	X	X	X	X	X	X	X	X
		記事の編集	vul.	X	X	X	X	X	✓	✓	-
		ユーザ情報の編集	vul.	X	X	X	X	X	✓	✓	-
WordPress 3.1.2	ユーザ	リンク	vul.	X	X	X	X	X	X	X	X
		コメントの投稿	vul.	X	X	X	X	X	X	X	-
	管理者	リンク	vul.	X	X	X	X	X	X	X	X
		ユーザの追加	vul.	X	X	X	X	X	✓	✓	-

テンプレートの攻撃結果 * において，“basic”: basic clickjacking のテンプレート，“1”: double framing，“2”: onBeforeUnload event，“3”: No-Content flushing，“4”: manipulating Referrer，“5”: restricting JavaScript with sandbox，“6”: restricting JavaScript with designMode，“cursor”: cursorjacking のテンプレートをそれぞれ示している．“-” は，cursorjacking のテンプレートが検査を行わなかったことを示している．“vul.” と “No vul.” は表 5.2 と同じものを示している．“✓” はテンプレートによる攻撃が失敗したことを示しており，“X” は攻撃が成功したことを示している．

の “脆弱性” と “Clickjuggler の検査結果” が全て一致しているために，Clickjuggler が false positive や negative なしに visual clickjacking の脆弱性を検査できたことがわかる．

さらに，表 5.3 は Clickjuggler の各テンプレートによる攻撃結果をまとめたものである．表 5.3 の “ウェブアプリケーション” と “ユーザ”，“検査対象のボタン”，“脆弱性” は，表 5.2 と同じものを示している．“テンプレートの攻撃結果” 内の “basic” と “1” から “6”，“cursor” は Clickjuggler の各テンプレートによる攻撃結果を示している．全ての脆弱な検査対象のボタン (“脆弱性” の列の vul.) は，少なくとも 1 つのテンプレートによる攻撃が成功している．全ての脆弱でない検査対象のボタン (“脆弱性” の列の No vul.) は，全てのテンプレートによる攻撃が失敗している．

この結果から Clickjuggler は 5.2 節で紹介した visual clickjacking に対する防御手法を検査できることがわかる．表 5.2 に示したように Clickjuggler は，Roundcube 0.7.0 に正確に実装された X-Frame-Options と element-customized defense に脆弱性はないと判断した．Clickjuggler は，表 5.3 からわかるように Firefox 20.0.1 上において Roundcube 0.7.0 の X-Frame-Options がテンプレートによる攻撃を全て防いだことから脆弱でないと判断した．

X-Frame-Options に加えて Roundcube 0.7.0 に実装されている element-customized defense は，X-Frame-Options をサポートしていない Firefox 3.6.8 上においても脆弱でないと判断された．これは，表 5.3 からわかるようにこの element-customized defense のコードがテンプレートによる攻撃を全て防いだためである．element-customized defense のコードは，Roundcube 0.7.0 のページが他のページに埋め込まれる時，Roundcube 0.7.0 のページを埋め込んだページのオリジンを調査する．そして，このページのオリジンが Roundcube 0.7.0 のページのオリジンと異なる場合，このコードは Roundcube 0.7.0 のページ内の全てのフォームエレメントを無効化する．

表 5.2 に示すように Clickjuggler は，Joomla 3.0.2 のプロフィール編集のためのボタンが basic clickjacking と frame busting の回避手法に対して脆弱ではないと判断した．なぜなら，このボタンには frame busting のコードが実装されており，このコードを回避することができないためである．このコードには，5.2.1 で示した回避手法を防ぐための手法 [27] が実装されていた．このコードは，ページのヘッダタグ内に実装されており，JavaScript が無効化されても，ページをブラウザ上に表示しない．さらに，parent.location や document.referrer のような変数をコード内で使用していない．

表 5.4: Clickjuggler と CJTool，BeEF plug-in の検査時間

ウェブアプリケーション	Clickjuggler			CJTool			BeEF plug-in		
	準備 (s)	テスト (s)	合計 (s)	準備 (s)	テスト (s)	合計 (s)	準備 (s)	テスト (s)	合計 (s)
Joomla 1.6.1	9.78	10.7	20.5	11.5	15.9	27.4	11.5	26.3	37.8
Joomla 2.5.7	9.44	10.7	20.1	11.5	15.9	27.4	11.5	25.4	36.9
Joomla 3.0.2	9.41	10.7	20.1	10.5	15.4	25.9	11.3	24.7	36.0
Roundcube 0.4.1	12.6	14.6	27.2	19.7	28.9	48.6	19.7	36.2	55.9
Roundcube 0.7.0	12.8	14.4	27.2	19.7	21.2	40.9	19.7	28.0	47.7
Roundcube 0.7.0 (Firefox 3.6.8)	12.8	14.5	27.3	19.5	31.3	50.8	19.5	36.1	55.6
MediaWiki 1.16.0	8.14	10.7	18.8	10.8	15.0	25.8	10.8	24.3	35.1
WordPress 3.1.2	9.18	10.6	19.8	10.3	15.0	25.3	10.3	22.8	33.1

5.5.2 パフォーマンス

CJTool [51] と BeEF のプラグイン [52] を利用して検査時間を比較する．これらの既存ツールは，開発者による手動での visual clickjacking の実行において攻撃者のページの生成をサポートする．実験において，Clickjuggler とこれらのツールを使って 5.5.1 で利用した 4 つのウェブアプリケーションが持つリンクの basic clickjacking の脆弱性を検査する．これは，CJTool が basic clickjacking の脆弱性のみを対象としており，BeEF のプラグインはデータの入力を必要とするフォームを対象としていないためである．それぞれのツールがリンクを検査するために必要だった時間を 5 回測定し，その平均時間を計算した．

表 5.4 は Clickjuggler と 2 種類の既存ツールの検査時間を測定した結果をまとめたものである．表 5.4 の “ウェブアプリケーション” は，検査対象のウェブアプリケーションの名前とバージョンを示している．“合計” の時間は，“準備” の時間と “テスト” の時間の総和を示している．そして，“準備” の時間は，脆弱性を検査するために必要な情報を準備し，それぞれのツールを操作するための時間を示しており，“テスト” の時間は，攻撃者のページを作成し，この攻撃者のページを利用して攻撃を実行することでリンクの脆弱性を検査するための時間を示している．

表 5.4 は，全ての場合において Clickjuggler は既存ツールよりも短い時間で脆弱性を検査できることを示している．表 5.4 からわかるように “テスト” の時間が，Clickjuggler と既存ツールの “合計” 時間に差が生まれた要因となっている．なぜなら，Clickjuggler は攻撃者のページを生成するためのプロセスや visual clickjacking を行う過程を自動化しているためである．一方で，CJTool と BeEF は 2.2.2 で示した様にこれらの過程を自動化していない．

5.5.3 制約

Clickjuggler は，CSS を利用している防御手法を検査した時に false positive を起こす．なぜなら，マウスイベントに反応する挙動を CSS で実装したボタンにブラウザを介してマウスイベントを発行しても，このイベントが CSS の挙動を動作させないためである．開発者達は，JavaScript を利用して CSS の挙動を動作させられないことについて議論している [101]．防御手法を CSS で実装したページにあるボタンに Clickjuggler がマウスイベントを発行した時，この防御手法は動作しないためにこのボタンは脆弱であると判断される．これに対してブラウザを修正することで，Clickjuggler は CSS で実装された防御手法を検査できると考えている．なぜなら，ブラウザはユーザのマウスイベントに対して CSS の挙動を動作させているためである．しかし，我々が知る限り，CSS で実装された防御手法はないため Firefox を修正しなかった．

Clickjuggler は，Chrome や IE の XSS フィルタを利用した攻撃者のページを生成できない．XSS フィルタを利用するために，Clickjuggler は検査対象のページの URI に frame busting のコードの一部を埋め込む必要がある．しかし Clickjuggler の現在の実装では，検査対象のページ内にある frame busting のコードを識別することができない．なぜなら，それぞれの開発者が異なる frame busting のコードを実装することができてしまうためである．これに対して Clickjuggler を拡張することで frame busting のコードを獲得することができると考えている．例えば，開発者に frame busting のコードを要求することで獲得できる．しかし，Clickjuggler は Firefox のプラグインとして実装されているためにこの拡張を行わなかった．

5.5.4 まとめ

ウェブアプリケーションのロジックに依存する visual clickjacking を自動的に実行することで脆弱性を検査する Clickjuggler を示した．開発者から獲得した検査対象のボタンのロジックに関する情報を利用して，Clickjuggler は visual clickjacking を実行する．開発者は visual clickjacking に対する防御手法を注意して実装したとしても，回避されない実装を施すことは容易なことではないために防御手法の実装の正当性を検査する必要がある．よって Clickjuggler は開発者にかかる防御手法の実装の正当性を確認する負荷を軽減することができる．

Clickjuggler が visual clickjacking の脆弱性を検査できることを確認するために，

実際にサイトで利用されている 4 つのオープンソースのウェブアプリケーションに適用した．Clickjuggler は 4 つのウェブアプリケーションに残った 26 個の visual clickjacking の脆弱性を発見することができ，これらのウェブアプリケーションに対して false positive と negative を起こさなかった．

さらに，Clickjuggler は既存ツールである CJTool と BeEF のプラグインよりも短い時間で脆弱性を検査できることを確認するために，上記の 4 つのウェブアプリケーションの検査に必要な時間を比較した．その結果，Clickjuggler は CJTool と BeEF のプラグインよりも短い時間で検査が行えた．これは，CJTool と BeEF のプラグインは攻撃者のページの生成のみを自動化しているのに対して Clickjuggler は攻撃者のページの生成と攻撃過程も自動化しているためである．

第6章　結論

本論文では，ウェブアプリケーションのロジックに依存する攻撃を自動的に実行することで脆弱性を検査する手法を提案した．提案機構は，ウェブアプリケーションのロジックに関する情報を獲得することでこのロジックに依存する攻撃を実行する．このような情報を自動的に獲得することは難しいために，ウェブアプリケーションの開発段階において開発者から獲得する．本手法の有用性を示すために，ウェブアプリケーションのロジックに依存する CSRF と session fixation，visual clickjacking に本手法を適用し，脆弱性を検査できることを示した．

本章では，本研究の貢献を要約することで本論文をまとめて，最後に本論文で提案した手法の今後の展望について議論する．

6.1　本研究の貢献

本研究における貢献は，ウェブアプリケーションのロジックに関する情報を獲得する手法を実現し，このロジックに依存する攻撃を実行することによって既存のペネトレーションテストが検査できる脆弱性の範囲を拡大したことである．本研究により，これまで検査できなかった脆弱性を多くの開発者が検査できるようになり，ウェブアプリケーションのセキュリティが向上する．

提案手法は，ウェブアプリケーションのロジックに依存する攻撃である CSRF と session fixation，visual clickjacking を実行でき，この攻撃を用いて脆弱性を検査できることがわかった．提案機構による攻撃がこれらの脆弱性を検査できることを確認するために，さまざまなサイトで利用されているオープンソースのウェブアプリケーションを対象に脆弱性検査を行った．その結果，検査を行った機能に残った脆弱性を全て検出することができた．実際に提案機構は 5 つのオープンソースのウェブアプリケーションに残った 11 個の CSRF の脆弱性と 6 個の session fixation の脆弱性を検出した．また，Clickjuggler は 4 つのオープンソースのウェブアプリケーションに残ったそれぞれ 26 個の visual clickjacking の脆弱性を検出した．

6.2 今後の展望

提案機構はウェブアプリケーションの挙動から脆弱性を判断しているために脆弱性の有無を検査できるものの，ソースコードの修正箇所を特定することはできない．そこで，提案機構と静的解析を組み合わせることによって，脆弱性の検査を行うと共にソースコードの修正箇所を特定することができると考えられる．提案機構が獲得したウェブアプリケーションのロジックに関する情報でソースコードを解析することで，このロジックとソースコードの対応関係を獲得する．そして提案機構はウェブアプリケーションのロジックに依存する攻撃を行うことで脆弱性を検出し，対応関係を利用してソースコードの修正箇所を特定できる．

特定したソースコードの修正箇所を修正することで脆弱性はなくなるが，他の機能を考慮して修正しなければウェブアプリケーションに悪影響を与える可能性がある．このような悪影響を回避するために，ウェブアプリケーションのすべてのロジックからモデルを生成し機能の依存関係を特定することで，悪影響を与えない修正方法を指示できると考えている．

謝辞

本論文は著者が慶應義塾大学大学院理工学研究科開放環境科学専攻の後期博士課程に在籍中の研究成果をまとめたものです．本研究を遂行し，研究成果を本論文にまとめるにあたり，多くの方々からご指導とご協力を賜りました．お世話になりました全ての方々に心より感謝申し上げます．

まず，本論文の主査であり，著者の指導教員である慶應義塾大学理工学部情報工学科の河野健二准教授に深く感謝いたします．河野健二准教授には，著者が学部 4 年次から修士課程，博士課程の 6 年間という長きにわたりさまざまなことをご教授いただきました．河野健二准教授は，自分の考えや調査結果に基づいて研究を進めていく難しさだけでなく面白さを教えてくださいました．この研究活動の面白さを教えていただかなければ，著者が後期博士課程に進学するという選択をしなかったのではないかと思います．また，研究活動を行っていく上で基礎となるプレゼンテーションにおいて研究内容を聴衆に伝える方法や論文作成において読み手を納得させる文章を作成する方法などの技術を指導していただきました．これらの技術は多くの場面で求められるものであり，著者にとって大きな財産になると考えている．

次に，本論文の副査を担当していただいた慶應義塾大学理工学部情報工学科の高田眞吾准教授，遠山元道准教授，及び管理工学科の山口高平教授に深く感謝しております．副査のみなさまにはお忙しいところ，本論文を査読するために貴重なお時間を割いていただきました．副査のみなさまとの議論や有益なコメントによって，本論文の完成度を大きく高めることができました．

小菅祐史博士に感謝いたします．小菅祐史博士には，著者が学部 4 年から修士 1 年の 2 年間にわたり，研究の方向性や実装に関する議論，論文の執筆方法，発表資料の添削などさまざまな面からご指導をいただきました．小菅祐史博士から指導を受けることなしに，本研究を遂行することはなかったと考えています．

本研究の遂行にあたり使用した実験機材の一部，および国内学会における原著論文の発表にあたっては，慶應義塾先端科学技術研究センターの KLL 後期博士課

程研究助成金の支援をいただきました．また，国際会議での聴講は著者にとって大変刺激となり，研究活動へのモチベーションへと繋がりました．さらに，日本学生支援機構奨学金は，経済的な心配なく研究に集中できる環境を整える上で大きな支えとなりました．

最後に，博士課程への進学にあたり経済的な支援を惜しまずに，現在まで暖かく見守っていただきました両親と弟に心より感謝いたします．

参考文献

[1] WhiteHat Security. WhiteHat Website Security Statistics Report. https://www.whitehatsec.com/resource/stats.html (accessed Mar, 2015).

[2] Cenzic. Application Vulnerability Trends Report : 2014. https://www.trustwave.com/Resources/Library/Documents/Cenzic-Application-Vulnerability-Trends-2014/ (accessed Mar, 2015), 2014.

[3] Symantec. Internet Security Threat Report 2014. https://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf (accessed Mar, 2015), 2014.

[4] Hold Security. YOU HAVE BEEN HACKED! http://www.holdsecurity.com/news/ (accessed Mar, 2015), 2014.

[5] Jamie Yap. 450,000 user passwords leaked in Yahoo breach. http://www.zdnet.com/article/450000-user-passwords-leaked-in-yahoo-breach/ (accessed Mar, 2015), 2012.

[6] John Fontana. Breach clean-up cost LinkedIn nearly $1 million, another $2-3 million in upgrades. http://www.zdnet.com/article/breach-clean-up-cost-linkedin-nearly-1-million-another-2-3-million-in-upgrades/ (accessed Mar, 2015), 2012.

[7] Mathew J. Schwartz. Sony Hacked Again, 1 Million Passwords Exposed. http://www.darkreading.com/attacks-and-breaches/sony-hacked-again-1-million-passwords-exposed/d/d-id/1098113 (accessed Mar, 2015), 2011.

[8] Chris Wysopal. What Happens When Companies Don't Give Web App Security the Attention it Deserves. https://www.veracode.com/blog/2013/07/what-happens-when-companies-dont-give-web-app-security-the-attention-it-deserves (accessed Mar, 2015), 2013.

[9] Kelly Jackson Higgins. Adobe Hacker Says He Used SQL Injection To Grab Database Of 150,000 User Accounts. http://www.darkreading.com/attacks-breaches/adobe-hacker-says-he-used-sql-injection-to-grab-database-of-150000-user-accounts/d/d-id/1138677 (accessed Mar, 2015), 2012.

[10] Casey Newton. GhostShell claims breach of 1.6M accounts at FBI, NASA, and more. http://www.cnet.com/news/ghostshell-claims-breach-of-1-6m-accounts-at-fbi-nasa-and-more/ (accessed Mar, 2015), 2012.

[11] Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, and V. N. Venkatakrishnan. CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 12–24, 2007.

[12] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *Proceedings of the 14th Network and Distributed Systems Security Symposium (NDSS '07)*, 2007.

[13] Trevor Jim, Nikhil Swamy, and Michael Hicks. Defeating Script Injection Attacks with Browser-enforced Embedded Policies. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*, pp. 601–610, 2007.

[14] Martin Johns, Björn Engelmann, and Joachim Posegga. XSSDS: Server-Side Detection of Cross-Site Scripting Attacks. In *Proceedings of the 24th Annual Computer Security Applications Conference (ACSAC '08)*, pp. 335–344, 2008.

[15] Prithvi Bisht and V. N. Venkatakrishnan. XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks. In *Proceedings of the 5th GI SIG SIDAR Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '08)*, pp. 23–43, 2008.

[16] Sid Stamm, Brandon Sterne, and Gervase Markham. Reining in the Web with Content Security Policy. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*, pp. 921–930, 2010.

[17] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. Preventing Cross Site Request Forgery Attacks. In *Proceedings of the 2nd IEEE International Conference on Security and Privacy in Communication Networks (SecureComm '06)*, pp. 1–10, 2006.

[18] Adam Barth, Collin Jackson, and John C. Mitchell. Robust Defenses for Cross-site Request Forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08)*, pp. 75–88, 2008.

[19] Riccardo Pelizzi and R. Sekar. A Server- and Browser-transparent CSRF Defense for Web 2.0 Applications. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*, pp. 257–266, 2011.

[20] Martin Johns, Bastian Braun, Michael Schrank, and Joachim Posegga. Reliable Protection Against Session Fixation Attacks. In *Proceedings of the 26th ACM Symposium on Applied Computing (SAC '11)*, pp. 1531–1537, 2011.

[21] OWASP. Category:Attack. https://www.owasp.org/index.php/Category:Attack (accessed Mar, 2015).

[22] Robert Hansen and Jeremiah Grossman. Clickjacking. http://www.sectheory.com /clickjacking.htm (accessed Mar, 2015), 2008.

[23] Mario Heiderich, Jörg Schwenk, Tilman Frosch, Jonas Magazinius, and Edward Z. Yang. mXSS Attacks: Attacking Well-secured Web-applications by Using innerHTML Mutations. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*, pp. 777–788, 2013.

[24] Bryan Sullivan. Server-Side JavaScript Injection. https://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf (accessed Mar, 2015), 2011.

[25] Shay Chen. Session Puzzles - Indirect Application Attack Vectors . https://puzzlemall.googlecode.com/files/Session%20Puzzles%20-%20Indirect%20Application%20Attack%20Vectors%20-%20May%202011%20-%20Whitepaper.pdf (accessed Mar, 2015), 2011.

[26] Chris Shiflett. addslashes() Versus mysql_real_escape_string() . http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string (accessed Mar, 2015), 2006.

[27] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *Proceedings of the IEEE Oakland Web 2.0 Security and Privacy Workshop (W2SP '10)*, 2010.

[28] Tino Brackebusch. Typo in header makes header useless. http://joomlacode.org/gf/project/joomla/tracker/?action=TrackerItemEdit&tracker_item_id=30790 (accessed Mar, 2015).

[29] Avik Chaudhuri and Jeffrey S. Foster. Symbolic Security Analysis of Ruby-on-rails Web Applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*, pp. 585–594, 2010.

[30] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In *Proceedings of the 27th IEEE Symposium on Security and Privacy (S&P '06)*, pp. 258–263, 2006.

[31] Adam Doupé, Bryce Boe, Christopher Kruegel, and Giovanni Vigna. Fear the EAR: Discovering and Mitigating Execution After Redirect Vulnerabilities. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*, pp. 251–262, 2011.

[32] Yosuke Hasegawa. UTF-7 XSS Cheat Sheet. http://openmya.hacker.jp/hasegawa/security/utf7cs.html (accessed Mar, 2015), 2008.

[33] Rahul Kumar, Indraveni K, and Aakash Kumar Goel. Automated Session Fixation Vulnerability Detection in Web Applications Using the Set-Cookie HTTP Response Header in Cookies. In *Proceedings of the 7th ACM International Conference on Security of Information and Networks (SIN '14)*, pp. 351–354, 2014.

[34] Andrés Riancho. Web Application Attack and Audit Framework. http://w3af.org (accessed Mar, 2015).

[35] Lavakumar Kuppan. Iron Web application Advanced Security testing Platform. http://ironwasp.org/index.html (accessed Mar, 2015).

[36] The Open Web Application Security Project (OWASP). OWASP Zed Attack Proxy Project. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project (accessed Mar, 2015).

[37] Tasos Laskos. Arachni. www.arachni-scanner.com (accessed Mar, 2015).

[38] Nicolas Surribas. Wapiti. http://wapiti.sourceforge.net/ (accessed Mar, 2015).

[39] David Byrne. Grendel Scan. http://sourceforge.net/p/grendel/code/ci/master/tree/ (accessed Mar, 2015).

[40] Yuji Kosuga, Kenji Kono, Miyuki Hanaoka, Miho Hishiyama, and Yu Takahama. Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC '07)*, pp. 107–117, 2007.

[41] Sean Mcallister, Engin Kirda, and Christopher Kruegel. Leveraging User Interactions for In-Depth Testing of Web Applications. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID '08)*, pp. 191–210, 2008.

[42] Prithvi Bisht, Timothy Hinrichs, Nazari Skrupsky, Radoslaw Bobrowicz, and V. N. Venkatakrishnan. NoTamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*, pp. 607–618, 2010.

[43] Prateek Saxena, Steve Hanna, Pongsin Poosankam, and Dawn Song. FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications. In *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS '10)*, 2010.

[44] Yuji Kosuga. A Study on Dynamic Detection of Web Application Vulnerabilities. Ph.D. dissertation, School of Science for Open and Environmental Systems, University of Keio, 2011.

[45] Marco Balduzzi, Carmen Torrano Gimenez, Davide Balzarotti, and Engin Kirda. Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications. In *Proceedings of the 18th Network and Distributed System Security Symposium (NDSS '11)*, 2011.

[46] Yuchen Zhou and David Evans. SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. In *Proceedings of the 23rd USENIX Conference on Security Symposium (USENIX Security '14)*, pp. 495–510, 2014.

[47] Chris Shiflett. Security Corner: Cross-Site Request Forgeries. http://shiflett.org/articles/cross-site-request-forgeries (accessed Mar, 2015).

[48] Mitja Kolsek. Session Fixation Vulnerability in Web-based Applications. http://www.acrossecurity.com/papers/session_fixation.pdf (accessed Mar, 2015).

[49] Marco Rocchetto, Martín Ochoa, and Mohammad Torabi Dashti. Model-Based Detection of CSRF. In *Proceedings of the 29th IFIP International Information Security and Privacy Conference (SEC '14)*, pp. 30–43, 2014.

[50] Petko D. Petkov. Rforge. http://blog.websecurify.com/2012/10/easy-cross-site-request-forgery-exploitation-with-websecurify-suite.html (accessed Mar, 2015).

[51] Paul Stone. Clickjacking Tool. http://www.contextis.com/research/tools/clickjacking-tool/ (accessed Mar, 2015).

[52] Brigette Lundeen and Jim Alves-Foss. Practical clickjacking with BeEF. In *Proceedings of the 12th IEEE Conference on Technologies for Homeland Security (HST '12)*, pp. 614–619, 2012.

[53] Adam Doupé, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. Enemy of the State: A State-aware Black-box Web Vulnerability Scanner. In *Proceedings of the 21st USENIX Conference on Security Symposium (USENIX Security '12)*, pp. 523–538, 2012.

[54] Giancarlo Pellegrino and Davide Balzarotti. Toward black-box detection of logic flaws in web applications. In *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[55] Bitcoin. http://bitcoin.org/ (accessed Mar, 2015).

[56] Timothy B. Lee. Bitcoin prices plummet on hacked exchange. http://arstechnica.com/tech-policy/2011/06/bitcoin-price-plummets-on-compromised-exchange/ (accessed Mar, 2015), 2011.

[57] MT. GOX. https://mtgox.com/ (accessed Mar, 2015).

[58] The Open Web Application Security Project (OWASP). Category:OWASP Top Ten Project. https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013 (accessed Mar, 2015), 2013.

[59] Sophos. Viral clickjacking 'Like' worm hits Facebook users. http://nakedsecurity.sophos.com/2010/05/31/viral-clickjacking-like-worm-hits-facebook-users/ (accessed Mar, 2015).

[60] US-CERT. CVE-2008-4503: Adobe Flash Player Clickjacking Vulnerability. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4503 (accessed Mar, 2015), 2008.

[61] Dingjie Yang. Clickjacking: An Overlooked Web Security Hole. https://community.qualys.com/blogs/securitylabs/2012/11/29/clickjacking-an-overlooked-web-security-hole (accessed Mar, 2015).

[62] Michael Schrank, Bastian Braun, Martin Johns, and Joachim Posegga. Session fixation - the forgotten vulnerability. In *Proceedings of the 5th conference on Sicherheit, Schutz und Zuverlssigkeit (GI Sicherheit '10)*, 2010.

[63] Martin Johns and Justus Winter. RequestRodeo: Client Side Protection against Session Riding. In *Proceedings of the OWASP Europe 2006 Conference*, pp. 5–17, 2006.

[64] Ziqing Mao, Ninghui Li, and Ian Molloy. Defeating Cross-Site Request Forgery Attacks with Browser-Enforced Authenticity Protection. In *Proceedings of the 13th International Conference Financial Cryptography and Data Security (FC '09)*, pp. 238–255, 2009.

[65] Philippe De Ryck, Lieven Desmet, Wouter Joosen, and Frank Piessens. Automatic and Precise Client-side Protection Against CSRF Attacks. In *Proceedings of the 16th European Conference on Research in Computer Security (ESORICS '11)*, pp. 100–116, 2011.

[66] Hossain Shahriar and Mohammad Zulkernine. Client-Side Detection of Cross-Site Request Forgery Attacks. In *Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE '10)*, pp. 358–367, 2010.

[67] Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, Frank Piessens, and Wouter Joosen. Serene: Self-reliant Client-side Protection Against Session Fixation. In *Proceedings of the 12th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS'12)*, pp. 59–72, 2012.

[68] Terri Oda, Glenn Wurster, P. C. van Oorschot, and Anil Somayaji. SOMA: Mutual Approval for Included Content in Web Pages. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08)*, pp. 89–98, 2008.

[69] Yuji Kosuga and Kenji Kono. Amberate: A Framework for Automated Vulnerability Scanners for Web Applications. In *JSSST Transaction on Computer Software*, pp. 175–195, 2011.

[70] Open Government Lab. http://www.openlabs.go.jp/ (accessed Mar, 2015).

[71] SecurityFocus. SecurityFocus. http://www.securityfocus.com/ (accessed Mar, 2015).

[72] US-CSRT. National Vulnerability Database. http://web.nvd.nist.gov/ (accessed Mar, 2015).

[73] CVEdetails. http://www.cvedetails.com/ (accessed Mar, 2015).

[74] Mambo. http://www.mamboserver.com/ (accessed Mar, 2015).

[75] Joomla. http://www.joomla.org/ (accessed Mar, 2015).

[76] phpBB. http://www.phpbb.com/ (accessed Mar, 2015).

[77] phpNuke. http://phpnuke.org/ (accessed Mar, 2015).

[78] osCommerce. http://www.oscommerce.com/ (accessed Mar, 2015).

[79] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schechter, and Collin Jackson. Clickjacking: attacks and defenses. In *Proceedings of the 21st USENIX Conference on Security Symposium (USENIX Security '12)*, pp. 413–428, 2012.

[80] The Open Web Application Security Project (OWASP). Clickjacking Defense Cheat Sheet. https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet (accessed Mar, 2015).

[81] Sebastian Lekies, Mario Heiderich, Dennis Appelt, Thorsten Holz, and Martin Johns. On the Fragility and Limitations of Current Browser-provided Clickjacking Protection Schemes. In *Proceedings of the 6th USENIX Workshop on Offensive Technologies (WOOT '12)*, pp. 53–63, 2012.

[82] Shuo Tang, Nathan Dautenhahn, and Samuel T. King. Fortifying Web-based Applications Automatically. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*, pp. 615–626, 2011.

[83] Microsoft. IE8 Security Part VII: ClickJacking Defenses. http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx (accessed Mar, 2015).

[84] Ruby on Rails. Ruby on Rails Security Guide. http://guides.rubyonrails.org/security.html (accessed Mar, 2015).

[85] django. Clickjacking Protection. https://docs.djangoproject.com/en/1.6/ref/clickjacking/ (accessed Mar, 2015).

[86] Michael Nepomnyashy. Protecting applications against Clickjacking with F5 LTM. SANS Institute InfoSec Reading Room, 2013.

[87] Eric Yawei Chen, Jason Bau, Charles Reis, Adam Barth, and Collin Jackson. App Isolation: Get the Security of Multiple Browsers with Just One. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*, pp. 227–238, 2011.

[88] Martin Johns, Sebastian Lekies, Bastian Braun, and Benjamin Flesch. BetterAuth: Web Authentication Revisited. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12)*, pp. 169–178, 2012.

[89] Giorgio Maone. Hello ClearClick, Goodbye Clickjacking! In *Black Hat Europe*, 2012.

[90] Jawwad A. Shamsi, Sufian Hameed, Waleed Rahman, Farooq Zuberi, Kaiser Altaf, and Ammar Amjad. Clicksafe: Providing Security Against Clickjacking Attacks. In *Proceedings of the 15th IEEE International Symposium on High-Assurance Systems Engineering (HASE '14)*, pp. 206–210, 2014.

[91] Ubaid Ur Rehman, Waqas Ahmad Khan, Nazar Abbas Saqib, and Muhammad Kaleem. On Detection and Prevention of Clickjacking Attack for OSNs. In *Proceedings of the 11th IEEE International Conference on Frontiers of Information Technology (FIT '13)*, pp. 160–165, 2013.

[92] Hossain Shahriar, Vamshee Krishna Devendran, and Hisham Haddad. ProClick: A Framework for Testing Clickjacking Attacks in Web Applications. In *Proceedings of the 6th ACM International Conference on Security of Information and Networks (SIN '13)*, pp. 144–151, 2013.

[93] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. A Solution for the Automated Detection of Clickjacking Attacks. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS '10)*, pp. 135–144, 2010.

[94] Mozilla. Web API interfaces. https://developer.mozilla.org/en-US/docs/Web/API (accessed Mar, 2015).

[95] Roundcube. Roundcube. http://roundcube.net/ (accessed Mar, 2015).

[96] MediaWiki. Mediawiki. http://www.mediawiki.org/wiki/MediaWiki (accessed Mar, 2015).

[97] WordPress. Wordpress. http://wordpress.org/ (accessed Mar, 2015).

[98] SECLISTS. SECLISTS.ORG. http://seclists.org/ (accessed Mar, 2015).

[99] Roundcube. Roundcube(wiki). http://trac.roundcube.net/wiki/Changelog (accessed Mar, 2015).

[100] Joomla. Joomla 3.0.2 Released. http://www.joomla.org/announcements/release-news/5471-joomla-3-0-2-released.html (accessed Mar, 2015).

[101] stackoverflow. Trigger css hover with JS. http://stackoverflow.com/questions/4347116/ (accessed Mar, 2015).

論文目録

定期刊行誌掲載論文

- Yusuke Takamatsu, Kenji Kono: “Detection of Visual Clickjacking Vulnerabilities in Incomplete Defenses”, *IPSJ Transactions on Advanced Computing System (ACS50)*, To Appear.

- Yusuke Takamatsu, Yuji Kosuga, Kenji Kono: “Automatically Checking for Session Management Vulnerabilities in Web Application”, *IPSJ Transactions on Advanced Computing System (ACS41)*, Vol.6, No.1, pp.45-55, Jan. 2013.

国際会議論文

- *Yusuke Takamatsu, Kenji Kono: “Clickjuggler: Checking for incomplete defenses against clickjacking,” In *Proceedings of the IEEE Annual Conference on Privacy, Security and Trust (PST ’14)*, pp.224-231, Jul. 2014.

- *Yusuke Takamatsu, Yuji Kosuga, Kenji Kono: “Automated Detection of Session Management Vulnerabilities in Web Applications,” In *Proceedings of IEEE Annual Conference on Privacy, Security and Trust (PST ’12)*, pp.112-119, Jul. 2012.

- *Yusuke Takamatsu, Yuji Kosuga, Kenji Kono: “Automated Detection of Session Fixation Vulnerabilities,” In *Proceedings of ACM international conference on World Wide Web (WWW ’10)*, Poster Session, pp.1191-1192, Apr. 2010.

国内学会発表

- *Yusuke Takamatsu, Kenji Kono: “Detection of Visual Clickjacking Vulnerabilities in Incomplete Defenses,” 情報処理学会 コンピュータシステム・シンポジウム (ComSys 2014), pp.16-26, Nov. 2014.

- *高松 勇輔, 河野 健二: “Clickjacking 脆弱性の自動検査手法”, 情報処理学会 システムソフトウェアとオペレーティング・システム研究会, Vol.2013-OS-124, No.10, pp.1-7, Feb. 2013.
- *高松 勇輔, 小菅 祐史, 河野 健二: “セッション管理の脆弱性検査の自動化”, 情報処理学会 コンピュータセキュリティシンポジウム (CSS 2011), pp.113-118, Oct. 2011.