Vol.2013-OS-124 No.10
2013/3/1

Clickjacking 脆弱性の自動検査手法

高松 勇輔[1,a)]　河野 健二[1,2]

概要：ウェブアプリケーションの脆弱性を突く新たな攻撃として，clickjacking という手法が知られるようになってきた．clickjacking とはユーザのクリックに対し，ユーザの意図とは異なる操作を強制する攻撃である．例えば，あるリンク上に別のサイトのリンクを透明に重ねて表示することで，被害者の意図とは異なるリンクをクリックさせ，意図しない商品購入を強制することなどができる．本研究では，ウェブアプリケーションの開発段階に clickjacking 脆弱性を自動的に検査する手法を提案する．Clickjacking を防止するためには，「表示方法を利用した手法」，「偽のカーソルを利用した手法」，「タイミングを利用した手法」といった clickjacking の手法についての詳細な知識とその対策手法に精通している必要がある．また，対策手法そのものが正しく実装されていることを確認するためには，実際に対象のウェブアプリケーションに攻撃を行う必要があり，そのためにはウェブアプリケーションが持つリンクやコンテンツごとに攻撃を作成する必要がある．さらに対策の不備を突いた攻撃手法も複数存在しているために，このような攻撃についての検査も必要となる．検査を自動化することによって，開発者は検査に必要となるこれらの負荷や労力を軽減することができる．実験では，提案手法を用いて clickjacking の脆弱性を埋め込んだウェブアプリケーションと実運用されているものの脆弱性を正確に検査できることを確認した．

キーワード：ウェブアプリケーションセキュリティ, 脆弱性, clickjacking

1. はじめに

ウェブアプリケーションの脆弱性を突く新たな攻撃として，clickjacking [1] という手法が知られるようになってきた．clickjacking とはユーザのクリックに対し，ユーザの意図とは異なる操作を強制する攻撃である．例えば，あるリンク上に別のサイトへのリンクを透明に重ねて表示することで，被害者の意図とは異なるリンクをクリックさせ，意図しない商品購入などの操作を強制することができる．また clickjacking には，「表示方法を利用した手法」，「偽のカーソルを利用した手法」，「タイミングを利用した手法」といった手法があり，これらの手法を利用して攻撃者は被害者の意図しない操作を行わせる [2]．実際に，2010 年にはセキュリティ企業の英 Sophos が，clickjacking 攻撃によって何十万人もの Facebook ユーザがウイルス感染被害を受けたと報告している [3]．

clickjacking 対策はすでに存在しているものの，正しく対策を行うことは難しい．例えば，よく知られた対策に frame busting という手法がある．この対策は，対策が施されたページが他のページに利用されることを防ぐことができる．しかしこの対策の不備を突いた clickjacking の攻撃も存在している [4]．さらに clickjacking には複数の攻撃手法があり，それぞれの攻撃手法に対して正確に対策を施さなければならない．

ウェブアプリケーションの開発時に clickjacking 脆弱性が残ってしまう二つの原因がある．一つ目は，開発者が clickjacking についての知識を十分に持っていないことがある．二つ目は，開発期間の都合により脆弱性のテストフェーズに十分な時間を割くことができないことがある．脆弱性の有無を確認するためのひとつの方法は，実際に検査対象のウェブアプリケーションに攻撃を行うことである．しかし clickjacking を行うためには，ウェブアプリケーションが持つそれぞれのリンクやコンテンツについて攻撃用のページを作成し，脆弱性の検査を行う必要がある．さらに clickjacking には 3 種類の手法があるために，このような攻撃手法の知識と攻撃手法を考慮した検査が必要となる．また対策の不備を突いた clickjacking 攻撃も存在しているために，このような攻撃についての知識と攻撃を考慮した検査も必要となる．

本論文は，開発段階のウェブアプリケーションに clickjacking 脆弱性があるかを自動的に検査するシステムを提案する．検査を自動化することによるメリットは次の通り

[1] Keio University
[2] JST CREST
[a)] yusuke@sslab.ics.keio.ac.jp

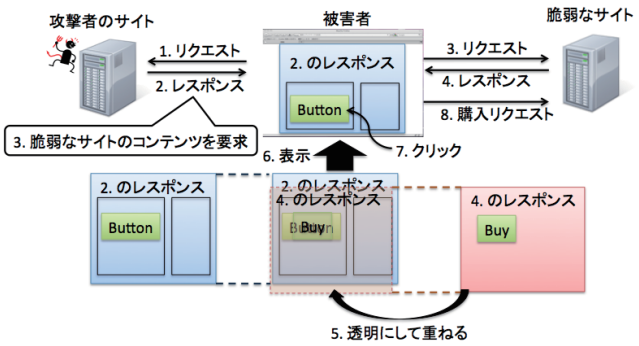

図 1 Clickjacking

である．一つ目は，たとえ開発者に clickjacking についての詳細な知識がなかったとしても検査が行えるようになる．さらにそれぞれのリンクやコンテンツについて攻撃用のページを作成するなどの環境構築の必要がなくなるため，検査にかかる時間を減らすことができるようになる．このように検査が自動化されることで，開発者は検査に必要となる攻撃についての詳細な知識と多くの時間と労力を軽減することができる．

提案機構をブラウザのアドオンとして実現する．clickjacking 対策にはブラウザとサーバの協力による防御手法も存在しているために，このような防御手法を検査するためにはブラウザの動作を確認する必要がある．また検査を自動化するためには，検査対象のリンクやコンテンツの情報を自動的に取得し，その情報を利用して攻撃用のページを生成し，攻撃を実行する必要がある．このような情報はブラウザから収集することができ，ブラウザから被害者のクリックを再現することができる．さらに提案機構がアドオンとして動作することで，開発者は提案機構を容易に導入することができる．

提案機構を firefox のアドオンとして実装し，clickjacking の脆弱性を埋め込んだ自作のウェブアプリケーションと対策を施した自作のものと実運用されているウェブアプリケーションの脆弱性を検査した．提案機構が全て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確認した．

本論文の構成は以下の通りである．2 章では clickjacking の攻撃手法や既存の対策手法について述べる．3 章では提案システムについて説明する．4 章では提案システムの有用性を確認した実験と結果について述べる．5 章では既存の関連研究について述べる．最後に 6 章で本論文をまとめる．

2. Clickjacking

2.1 Clickjacking の仕組み

clickjacking は被害者を騙すことによってクリックを実行させ，クリックによって攻撃者の意図する操作を被害者に強制する攻撃である．この攻撃により，個人情報の流出や改竄，意図しない商品購入などの被害が生じる．

図 1 を用いて，clickjacking の攻撃方法の一例を説明する．まず攻撃者は攻撃ページを用意し，そのページを公開する．攻撃ページは，ターゲットサイトのページ内のリンクを透明にして攻撃ページ内のリンクの上に重ねて表示されるように作成する．例えば，iframe タグによってターゲットサイトのページを獲得する．そして CSS の opacity プロパティを利用することで透明度を指定し，CSS で指定することで，攻撃ページのリンクにターゲットサイトのページのリンクを重ねることができる．次に被害者がこの攻撃ページを要求し，攻撃者のサイトがレスポンスとして攻撃ページを返信する (Steps 1 & 2)．被害者のブラウザは攻撃ページに従ってターゲットサイトのページを要求し，ターゲットサイトがレスポンスを返信する (Steps 3 & 4)．ブラウザは攻撃ページの CSS に従ってターゲットサイトのページを透明化し，攻撃ページに重ねて表示する (Steps 5 & 6)．この際，攻撃ページ内のリンクにターゲットサイトのページのリンクを重ねる．被害者は攻撃ページ内のリンクをクリックする (Step 7)．このとき，実際にクリックされるのは攻撃ページ内のリンクに透明化されて重ねられているターゲットサイトのページのリンクである．ブラウザからターゲットサイトにリクエストが発行される (Step 8)．最後にターゲットサイトでリクエストが処理される．

2.2 攻撃手法

clickjacking には，「表示方法を利用した手法」，「偽のカーソルを利用した手法」，「タイミングを利用した手法」の 3 つの手法があり，攻撃者はこれらの手法を利用して被害者にクリックを行わせる．

「表示方法を利用した手法」は，例で示したようにブラウザに表示されるページを偽ることによって被害者にクリックを行わせる．ターゲットサイトのページを透明化する方法以外にも，iframe で指定することでターゲットサイトのページ内のリンクのみを表示して攻撃ページに埋め込むことで，ターゲットサイトのリンクが攻撃ページの一部であると被害者に思わせる方法もある．

「偽のカーソルを利用した手法」は cursorjacking として知られており，偽のカーソルを利用して被害者を騙すことでクリックを行わせる [5]．例えば，ページ上に正しいカーソルとともに偽のカーソルを表示することでユーザを騙し，目的のリンクをクリックさせる．

「タイミングを利用した手法」は，ユーザがブラウザの表示の変化に反応できるまでの時間を利用して，目的のリンクをクリックさせる．例えば，ユーザに攻撃ページのリンクに対してダブルクリックを要求し，二回目のクリック時に攻撃ページのリンク上にターゲットサイトのリンクを表示させてユーザにクリックさせる．

2.3 対策手法

clickjacking 対策として，さまざまな対策が存在している．よく知られた対策として frame busting というサーバサイドでの手法がある．この対策は Javascript のコードとして実装され，対策が施されたページが他のページに利用されることを防ぐことができる．例えば，対策コードは，親フレーム (攻撃ページ) と子フレーム (対策が施されたページ) の URI を比較する．そして異なる場合に子フレーム (対策が施されたページ) を親フレームにする．対策が施されたページを親フレームにすることで，攻撃ページが対策が施されたページに置き換えられる．従って，攻撃ページは表示されず対策が施されたページだけがブラウザに表示される．

X-FRAME-OPTIONS [6] は，サーバとブラウザが協力してクロスサイト間でのコンテンツの利用制限を行う手法である．サーバは，HTTP ヘッダを利用してサーバの持つコンテンツを利用できるサイトをブラウザに指定する．ブラウザはサーバの指定に従って獲得したコンテンツをページ内に埋め込むかを決定する．最近の主要なブラウザにはこの機能が実装されている．

しかし clickjacking 対策を正確に行うことは難しい．これは，対策の不備を突いた clickjacking の攻撃が存在していたり，ミスが起こるためである．例えば，frame busting の不備を突いた攻撃として busting frame busting という攻撃が存在している [4]．この攻撃は Javascript のコードを利用して，対策の Javascript のコードを停止させる方法である．さらに X-FRAME-OPTIONS については，ページ毎に利用制限を行えるため，多くのページを保持しているようなウェブアプリケーションでは制限のミスが起こってしまう．

2.4 検査による負荷と労力

clickjacking の脆弱性がウェブアプリケーションに残ってしまう二つの原因がある．一つ目は，開発者が clickjacking についての知識を十分に持っていないことがある．二つ目は，開発期間の都合により脆弱性のテストフェーズに十分な時間を割くことができないことがある．脆弱性の有無を確認するためのひとつの方法は，実際に検査対象のウェブアプリケーションに攻撃を行うことである．しかし clickjacking を行うためには，ウェブアプリケーションが持つそれぞれのリンクやコンテンツについて攻撃用のページを作成する．そして開発者は攻撃用のページを公開し，被害者として攻撃を実行することで検査を行う必要がある．ウェブアプリケーションが持つリンクやコンテンツが多ければ多いほど作成しなければいけない攻撃用のページは増え，開発者が実行しなければいけない攻撃の回数も増える．さらに 2.2 節で説明したように clickjacking には，「表示方法を利用した手法」，「偽のカーソルを利用した手法」，「タイミングを利用した手法」といった攻撃手法があるためにこのような攻撃手法についても検査が必要となる．また対策の不備を突いた clickjacking 攻撃も存在しているために，このような攻撃についての検査も必要となる．このように開発者は．検査を行うために攻撃についての詳細な知識や多くの時間と労力が必要となる．

3. 提案

本論文で，開発段階のウェブアプリケーションに clickjacking の脆弱性があるかを自動的に検査するシステムを提案する．本論文は clickjacking の手法としてもっとも知られている「表示方法を利用した手法」を対象とする．検査を自動化することによって，検査に必要となるリンクやコンテンツ毎の攻撃用ページの生成や攻撃実行にかかる時間を削減することができる．さらに clickjacking の攻撃手法や対策の不備をついた攻撃を考慮した攻撃用ページを作成する必要がなくなるために，攻撃手法についての詳細な知識が必要なく検査を行えるようになる．

提案機構はブラウザのアドオンとして動作する．clickjacking 対策にはサーバとブラウザの協力による防御手法も存在しているために，このような防御手法を検査するためにはブラウザの動作を確認する必要がある．例えば，2.3 節で説明した X-FRAME-OPTIONS はブラウザがサーバによって送られた HTTP ヘッダに従ってコンテンツの利用を決定するために，ブラウザの動作を確認する必要がある．さらに検査を自動化するためには，検査対象のリンクやコンテンツの情報を自動的に取得し，その情報を利用して攻撃用のページを生成し，攻撃を実行する必要がある．ブラウザからこのような情報を収集することができ，さらにブラウザに被害者のクリックなどを再現する機能が用意されている．また提案機構がアドオンとして動作することで，開発者は提案機構を容易に導入し脆弱性検査を行う事ができる．

検査に必要な情報を獲得するために，開発者に提案機構を導入したブラウザで検査を行いたいリンクやコンテンツを利用してもらう必要がある．獲得する情報は，「検査したいリンクの位置情報」，「クリックしたリンクのスクリーンショット」，「クリックによるリクエストが正しく処理された場合のレスポンス」の 3 つの情報である．「検査したいリンクの位置情報」は攻撃用のページを生成するために利用する．「クリックしたリンクのスクリーンショット」と「クリックによるリクエストが正しく処理された場合のレスポンス」は攻撃結果を判定するために利用する．これらの情報を自動的に獲得することが難しい．これは，ウェブアプリケーションが持つ全てのリンクやコンテンツの情報を得ることが難しいなどの問題があるためである．例えば，ログイン後のリンクやコンテンツの情報を得るためには，自動的にログインする必要がある．しかしログインページを

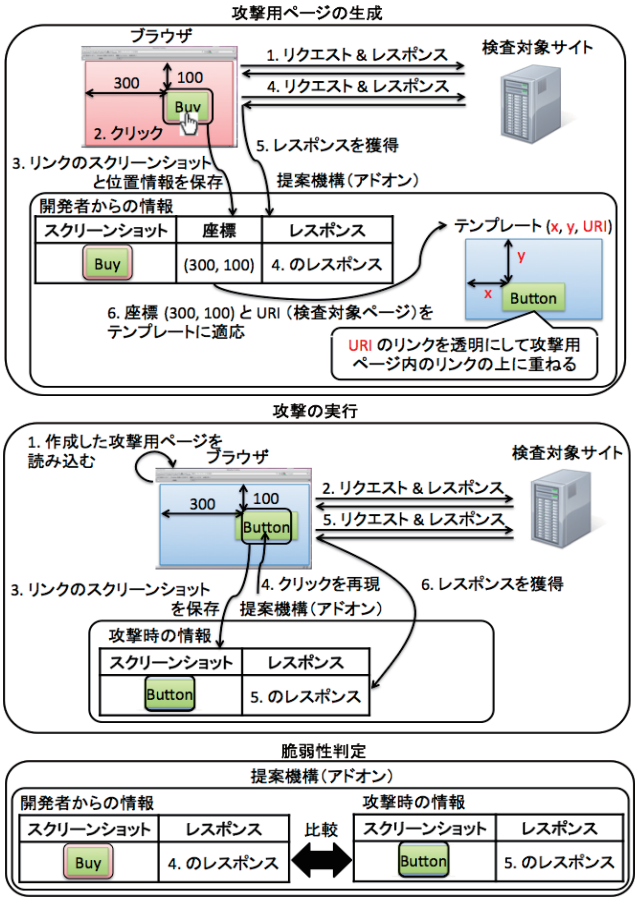

図 2 Clickjacking の自動検査手法

識別することやログインが成功したのかを判定することは難しいために，確実にログインすることは難しい．対して開発者が操作した情報を獲得することで確実に目的の情報を獲得できる．よって情報収集の失敗による false positive や false negative の発生を抑えることができる．また開発者にとって，開発中のウェブアプリケーションが持つリンクやコンテンツを利用することやログインを行うことは難しいことではない．

3.1 検査の自動化

検査を自動化するためには，clickjacking の攻撃と脆弱性の判定を自動的に行う必要がある．clickjacking の攻撃を自動的に行うために，**図 2** に示すように開発者が検査を行いたいリンクについての攻撃用のページを作成し，作成した攻撃用のページをブラウザに検査対象のページと異なる origin で表示する．そして攻撃用のページに対して被害者のクリックを再現することで攻撃を実行する．脆弱性の判定を自動的に行うために，攻撃結果を解析して clickjacking 脆弱性の有無を判定する．これらの機能を自動化する方法を説明する．

3.1.1 攻撃の自動化

攻撃用のページを作成するために，ページやコンテンツ毎の異なる位置にあるリンクに対して攻撃用のページを作成する必要がある．そこで，あらかじめ検査対象のリンクやコンテンツの位置情報を持たない攻撃用のページのテンプレート（HTML）を用意しておく．図 2 の攻撃用ページの生成にあるように，このテンプレートを読み込み，開発者から獲得した検査対象のリンクやコンテンツの位置情報をもとに攻撃用のページを生成する．このように攻撃用のページを生成することで，異なるサイトや同じページ内のリンクなどに対して攻撃用のページを生成することができる．

検査対象のページと攻撃用のページは異なる origin でならなければならない．攻撃者が，同じ origin つまり自分のサイトに clickjacking を仕掛けることがないためである．また X-FRAME-OPTIONS による対策では，「同じ origin にコンテンツを利用させる」や「特定の origin にのみコンテンツを利用させる」などの利用制限を行う事ができるために，異なる origin にする必要がある．異なる origin で攻撃用のページをブラウザに表示するためにテンプレートをローカルエリアに用意しておく．テンプレートをローカルエリアに置くことによって，ローカルファイルであるテンプレートから生成された攻撃用のページの origin は検査対象のページの origin と異なる．

作成した攻撃用のページを利用して攻撃を実行するために，被害者のクリックなどの動作を再現しなければならない．対策には frame busting のように javascript によって記述されているものがある．よって実際にブラウザに対策コードを実行させて，対策が施されたページのリンクをクリックして対策の動作を確認するためにクリックを再現する必要がある．ユーザのクリックを再現するために，ブラウザの機能を利用した．この機能を利用することによって，クリックだけでなく mouseover や mousemove などのマウス操作に関する動作も再現することができる．

3.1.2 脆弱性判定の自動化

対象のウェブアプリケーションの脆弱性を判定するために，clickjacking の攻撃が成功したかを自動的に判定する必要がある．攻撃時に被害者の視覚を騙すことができていて，攻撃対象のリンクを実際にクリックすることで生じるリクエストが正しく処理されている場合に攻撃が成功したと判断することができる．攻撃時に被害者の視覚を騙すことができているかを調べるために，攻撃対象のリンクがブラウザにどのように表示されているかを調べる．またクリックによって生じるリクエストが攻撃の目的を果たしたものかを調べるために，攻撃時に攻撃対象のリンクをクリックして発行されるレスポンスを調べる．

攻撃対象のリンクがブラウザにどのように表示されているかを調べるために，開発者から獲得した「クリックしたリンクのスクリーンショット」と攻撃時にブラウザに表示された攻撃対象のリンクのスクリーンショットを比較する．「クリックしたリンクのスクリーンショット」と攻撃

表 1 clickjacking 脆弱性の検査結果

ウェブアプリケーション		検査結果			
対策	脆弱性	攻撃パターン 1	攻撃パターン 2	攻撃パターン 3	総合結果
対策パターン 1	clickjacking	clickjacking	clickjacking	clickjacking	clickjacking
対策パターン 2	clickjacking	No vul.	clickjacking	clickjacking	clickjacking
対策パターン 3	clickjacking	No vul.	No vul.	clickjacking	clickjacking
対策パターン 4	No vul.	No vul.	No vul.	No vul.	No vul.
対策パターン 5	clickjacking	clickjacking	clickjacking	No vul.	clickjacking
X FRAME OPTIONS	No vul.	No vul.	No vul.	No vul.	No vul.

時における攻撃対象のリンクのスクリーンショットが異なるとき，被害者の視覚を騙すことができている．

クリックによって生じるリクエストが攻撃の目的を果たしたものかを調べるために，開発者から獲得した「クリックによるリクエストが正しく処理された場合のレスポンス」と攻撃時に攻撃対象のリンクをクリックした場合のレスポンスを比較する．「クリックによるリクエストが正しく処理された場合のレスポンス」と攻撃時に攻撃対象のリンクをクリックした場合のレスポンスが同じ時，クリックによって生じるリクエストが攻撃の目的を果たしている．

4. 実験

4.1 実験方法

提案機構を評価するために，様々な clickjacking 脆弱性のパターンを再現した自作のウェブアプリケーションと実運用中のウェブアプリケーションを使用して実験を行った．

以下の 5 パターンの対策を持った自作のウェブアプリケーションを作成した．(1)(2)(3)(5) の対策には clickjacking の脆弱性があり，(4) の対策には clickjacking の脆弱性がない．

(1) clickjacking の対策を行わない．
(2) clickjacking の対策である frame busting を行うものの，この対策の不備を突いた攻撃について考慮されていない．
(3) clickjacking の対策である frame busting を行い，この対策の不備を突いた攻撃について考慮されている．
(4) (3) の対策に加え，ページ内のリンクを全て javascript で記述する．
(5) 独自の対策

(4) の対策は，javascript を無効にされることで frame busting も無効化された場合のための対策である．clickjacking において，攻撃用のページが iframe の設定によって攻撃対象ページの javascript を無効にすることができる．(4) の対策のようにページ内のリンクを全て javascript で記述することで，javascript が無効されたときに攻撃対象ページのリンクは表示されない．よって被害者は表示されていないリンクをクリックすることができないので，clickjacking を防ぐことができる．

(5) の独自の対策は，ユーザがリンクをクリックするためには特定の動作を行ってからでなければリンクをクリックできない対策である．今回，リンクに 2 秒間カーソルを合わせ，その後リンクからカーソルを放さなければクリックできないという対策を施した．これは 2.2 節で説明した clickjacking の攻撃手法「タイミングを利用した手法」には有効であるものの，「表示方法を利用した手法」には十分な効果を発揮できない．「タイミングを利用した手法」はユーザがブラウザの表示の変化に反応できるまでの時間を利用して，目的のリンクをクリックさせる．しかしこの対策が施されていると，たとえ攻撃対象のリンクを表示したとしても特定の動作を行っていないのでクリックすることはできない．このとき，被害者は clickjacking に気づき，クリックを中止するので攻撃を防ぐことができる．対して，「表示方法を利用した手法」の場合，攻撃対象のリンクは透明な状態で表示されているので，被害者に特定の動作をさせる指示をして攻撃対象のリンクをクリックできる状態にできる．

実運用中のウェブアプリケーションとして google [7] のトップページを利用した．google のトップページには X FRAME OPTIONS の対策が施されている．HTTP ヘッダを解析したところ，same origin でのみ利用可能な設定になっていた．

この実験を行うために提案機構を firefox のアドオンとして実装した．また「表示方法を利用した手法」の攻撃パターンとして，以下の 3 つの攻撃パターンを用意した．

(1) 対策を考慮していない clickjacking
(2) frame busting の不備を突いた clickjacking
(3) iframe 内の javascript を動作させない clickjacking

それぞれのウェブアプリケーションについてこれらの攻撃パターンで検査を行った．

4.2 実験結果

表 1 に実験結果を示す．総合結果は，3 つの攻撃パターンの攻撃結果によって決定する．総合結果は，3 つの攻撃パターンの中で少なくとも 1 つの攻撃パターンで脆弱性ありと判断した場合，そのウェブアプリケーションに clickjacking 脆弱性があると判定する．表 1 より，総合結

果とウェブアプリケーションの clickjacking の脆弱性が一致していることがわかる．よって．提案機構は様々な clickjacking 脆弱性のパターンを再現した自作のウェブアプリケーションと実運用中のウェブアプリケーションに対して正確に脆弱性を検査できたといえる．

5. 関連研究

文献 [8] は，実環境で clickjacking を行っているページを自動的に検出する手法である．この機構は，自動的にページ内にある全てのリンクなどをクリックし，クリック時のページを解析し，解析結果からこのページが clickjacking を行っているかを検査する．例えば，クリックした座標に異なるページのリンクが重なって置かれている場合などに攻撃が行われていると判断する．この機構は，clickjacking を行っているページを自動的に見つけ出すことが目的で，ウェブアプリケーションの clickjacking 脆弱性を検査するための本研究とは目的が異なる．またこの機構を使って，clickjacking を行っているページが攻撃対象としているページに脆弱性があると判断することはできる．しかしこの機構では，攻撃対象になっていないページの脆弱性や他の攻撃パターンによる脆弱性を検査できない．

Sania [9] と Secubat [10] は，自動的に攻撃を行う事によって脆弱性の検査を行う手法である．Sania は，開発段階において，正常に動作しているときのリクエストと SQL クエリをもとに生成した攻撃を実行することで，SQL インジェクション脆弱性を検査する手法である．Secubat は，実運用されているウェブアプリケーションをクロールし，SQL インジェクションと XSS の脆弱性を検査する手法である．Sania と Secubat は，clickjaking 脆弱性を検査することができない．これは，clickjaking 脆弱性が入力確認の不備が原因の脆弱性ではないためである．

CSP [11] と InContext [2] はクライアントとサーバが協力して clickjacking を防御する手法である．CSP はクライアントのブラウザとサーバが協力して clickjacking や Cross-Site Scripting などの攻撃を防御する手法である．clickjacking の場合には，サーバが HTTP ヘッダを利用してサーバの持つコンテンツがブラウザ上でどのように動作するかを指定する．そしてブラウザはサーバの指定に従ってコンテンツを処理することで，攻撃者による無断でのコンテンツ利用を制限する．InContext は，クライアントのブラウザとサーバが協力して clickjacking を防御する手法である．サーバが防御したいリンクなどのコンテンツを指定し，ブラウザは指定されたコンテンツを防御する．例えば，指定されたリンクは表示されてから一定時間たたないとクリックできないといった防御手法を実行する．これらの防御手法はウェブアプリケーションの clickjacking 脆弱性を検査することはできない．クライアントサイドのブラウザが，レスポンスに CSP や InContext のための指定が含まれていなければ脆弱性があると判断できるかもしれない．しかし対策には frame busting などの javascript が実行されなければ脆弱性があるかわからない手法も存在しているために，多くの false positive が生じてしまう．

6. まとめ

本論文は，開発段階のウェブアプリケーションに clickjacking の脆弱性があるかを自動的に検査するシステムを提案した．検査を自動化することによって，開発者は検査に必要となるリンクやコンテンツ毎の攻撃用ページの生成や攻撃実行にかかる時間を削減することができ，攻撃手法についての詳細な知識も必要なく検査を行えるようになる．提案機構を firefox のアドオンとして実装し，clickjacking の脆弱性を埋め込んだ自作のウェブアプリケーションと対策を施した自作のものと実運用されているウェブアプリケーションの脆弱性を検査し，正確に検査できることを確認した．

今後の課題としては，オープンソースのウェブアプリケーションや実運用中のウェブアプリケーションで検査を行ってみる．また今回対象とした clickjacking の手法である「表示方法を利用した手法」以外の「偽のカーソルを利用した手法」と「タイミングを利用した手法」についても検査できるようにする．さらに「表示方法を利用した手法」についても実験で実装した 3 つの攻撃パターン以外の攻撃パターンを実装する．

参考文献

[1] Hansen, R. and Grossman, J.: Clickjacking, http://www.sectheory.com/clickjacking.htm (2008).
[2] Huang, L.-S., Moshchuk, A., Wang, H. J., Schechter, S. and Jackson, C.: Clickjacking: attacks and defenses, *Proceedings of the 21st USENIX conference on Security symposium*, pp. 22–22 (2012).
[3] Sophos: Viral clickjacking 'Like' worm hits Facebook users, http://nakedsecurity.sophos.com/2010/05/31/viral-clickjacking-like-worm-hits-facebook-users/ (2010).
[4] Rydstedt, G., Bursztein, E., Boneh, D. and Jackson, C.: Busting frame busting: a study of clickjacking vulnerabilities at popular sites, *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)* (2010).
[5] Kotowicz, K.: Cursorjacking again, http://blog.kotowicz.net/2012/01/cursorjacking-again.html (2012).
[6] Maone, G.: X-frame-options in firefox, http://hackademix.net/2009/01/29/x-frame-options-in-firefox/ (2009).
[7] google: http://www.google.com/.
[8] Balduzzi, M., Egele, M., Kirda, E., Balzarotti, D. and Kruegel, C.: A solution for the automated detection of clickjacking attacks, *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pp. 135–144 (2010).
[9] Kosuga, Y., Kono, K., Hanaoka, M., Hishiyama, M. and Takahama, Y.: Sania: Syntactic and Semantic Analy-

sis for Automated Testing against SQL Injection, *Proceedings of the Annual Computer Security Applications Conference (ACSAC '07)*, pp. 107–117 (2007).

[10] Kals, S., Kirda, E., Kruegel, C. and Jovanovic, N.: SecuBat: a web vulnerability scanner, *Proceedings of the International World Wide Web Conference (WWW '06)*, pp. 247–256 (2006).

[11] Stamm, S., Sterne, B. and Markham, G.: Reining in the web with content security policy, *Proceedings of the 19th international conference on World wide web*, pp. 921–930 (2010).