特定のフィッシング Web ページにおけるパスワードの自動入力を防ぐための セキュリティ対策の提案

研究年度　平成 30 年度
研究期間　平成 30 年度～平成 30 年度
研究代表者名 C.ソムチャイ

あらまし
最近、ユーザーの肉眼で見えないフィールドに機密情報を自動入力（オートフィル）する新しいフィッシングページが現れている。本研究の実験では、上記のフィッシングページの仕組を使ってブラウザのパスワードマネージャから保存されたパスワードを盗むことができると分かった。本問題は、Web ブラウザ開発者がまだ解決されていないようだ。本研究の目的は、これらのフィッシングページにおけるパスワードのオートフィルを防ぐためのセキュリティシステムを提案することである。提案されたシステムは、ユーザーが使用しているウェブブラウザのオートフィル機能をオフにする。ユーザーが閲覧したい Web ページの HTML ソースの分析を行って、システムは隠しフィールドの存在をユーザーに知らせる。それを見たユーザーはその Web ページにおけるオートフィルを許可するかどうかを判断する。

１．はじめに

ブラウザのパスワードマネージャは、サイバー犯罪者が狙っている重要な機密情報（例えば、電子メールアドレスやパスワードなど）を管理しているので、ハッカーの標的対象である。既存の研究[1]は、攻撃者が XSS（クロスサイトスクリプティング）攻撃の脆弱性を持つウェブページに JavaScript で書かれた追跡スクリプトを挿入する手法を実証した。ユーザーがそのような Web ページを閲覧すると、追跡スクリプトがその Web ページ上にユーザーの肉眼では見えない悪意のあるログインフォームを自動的に挿入する。それによって、パスワードマネージャはそのユーザーのログイン情報を自動的に入力してしまう。このようにして、追跡スクリプトは、目に見えないフォームのフィールドからユーザーのログイン情報を盗み取り、そのログイン情報のハッシュ値を攻撃者のサーバーに送信される。Web ブラウザのパスワードマネージャの最も便利な機能は情報セキュリティ問題につながる一つの手段になってしまう。

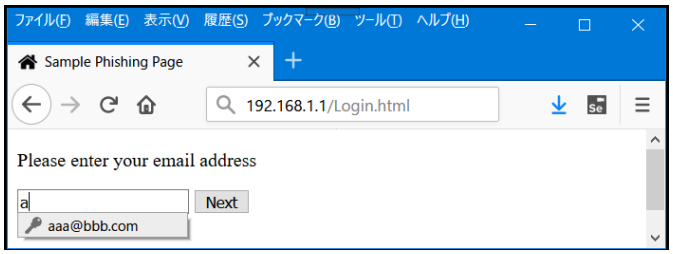
Silver ら[2]は、上記の問題を解決するために、フォームに自動入力する前に常に何らかのユーザーの操作を必要とする防御方法を提案した。そのユーザー操作の例としては、キーボードショートカット、ボタンのクリック、メニューからのエントリの選択、またはユーザー名フィールドへの入力などが挙げられる。彼らの提案した方法は、ユーザーの操作なしに複数のパスワードを抽出する攻撃を防ぐだろう。しかし、この防御方法は、パスワードの位

置の特別な定義でパスワードフィールドが見られない新しいフィッシングページへのパスワード自動入力を防ぐことはできない。ユーザーが送信ボタンをクリックした後、自動入力されたパスワードが密かに第三者に送信される。

本研究の目的は、これらのフィッシングページにおけるパスワードの不正の自動入力を防ぐためのブラウザの自動入力制御システムを提案することである。本システムは、ユーザーが使用しているウェブブラウザの自動記入機能を無効にする。ログイン Web ページの URL アドレスの入力を受け、本システムはログインページの HTML データを分析し、自動入力のターゲットとなるフィールドの名前または ID を表示する。このようにして、ユーザーは危険な隠しフィールドを認識し、ブラウザによるパスワードの自動入力を許可するかどうかを決定する。

２．パスワード盗難のシナリオ

本研究では、ブラウザのパスワードマネージャからパスワードを盗むための実験を行った。図 1（a）は、電子メールのテキストボックスと「次へ」のボタンのみを表示するサン



(a)

```
…
<form class="login100-form">
<p>Please enter your email address</p>
<input class="input100" type="text" name="username" placeholder="email" >
<button class="login100-form-btn">
Next
</button>
</div>
</div>
</div>
<p style="margin-left:-500px">
<input type="password" name="pass" placeholder="Password">
</p>
</form>
…
```

(b)

図 1（a）ユーザーがメールアドレスを入力するフィッシングページの例
（b）そのフィッシングページの一部の HTML ソース

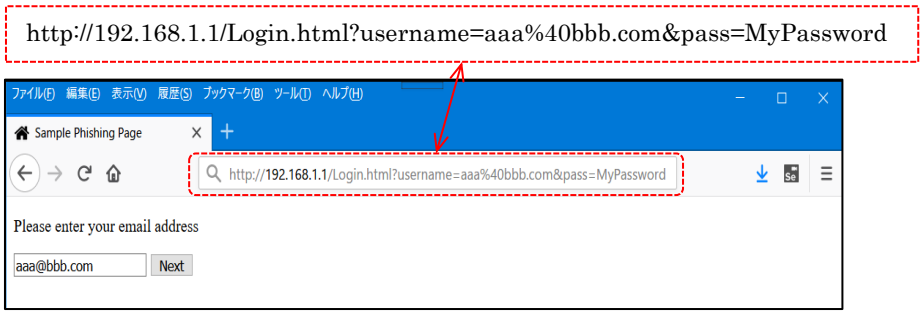


図 2.　図 1（a）のフィッシングページの「次へ」ボタンをクリックした後に
表示されたページの例

プルフィッシングページの例を示す。本フィッシングページは、ユーザーがすでにパスワードマネージャに自分のパスワードを保存した Web サイトに実装する。その実装を行える者は、Web サイトの運営する組織の内部犯罪者、または Web サイトのセキュリティを突破したハッカーである。図 1（b）はフィッシング Web ページの HTML ソースの一部を示している。パスワードのテキストボックスは 2 番目の入力タグで定義されている。ただし、<p style = "margin-left：-500px">タグによると、パスワードのテキストボックスは Web ブラウザの画面領域外にあるため、図 1（a）には表示されない。 <p style = "margin-top：-500px">タグを使用して、パスワードのテキストボックスを Web ブラウザの画面領域の中に隠すこともできる。このトリックは CSS（Cascading Style Sheets）を使用しても実行できる。図 2 に示すように、「次へ」ボタンをクリックした後、URL アドレスバーに「http://192.168.1.1/Login.html?username=aaa%40bbb.com&pass=MyPassword」と出力された。その中にユーザーのパスワードである MyPassword が含まれているので、そのパスワードを犯罪者のサーバーに送信されかねない。基本的には Web ブラウザのパスワードマネージャはパスワードとウェブサイトのドメイン名の組を保存している。それを知っている犯罪者は同じドメインにおける他の Web サーバーを設置して、そのサーバーにフィッシング Web ページを実装しておく。そのフィッシング Web ページに誘導されたユーザーが e-mail アドレスを入力したら、パスワードマネージャは隠れているパスワードフィールドにパスワードを自動入力する。

３．提案したシステムのアーキテクチャ

前章で説明した問題を解決するためのブラウザ自動入力制御システム（Browser Autofill Control System を提案する。以降は、本システムを BACS と呼ぶ。BACS は、ユーザーと Web ブラウザの間のインタフェースとして開発される。 BACS のアーキテクチャを図 3 に示す。ユーザーは BACS を介して Web ページにログインする。ユーザーが BACS を起動すると、ユーザーのコンピュータの Web ブラウザの自動入力機能を無効にする。BACS はユーザーが認証情報を入力したログインページの URL アドレスと HTML データのハッシ

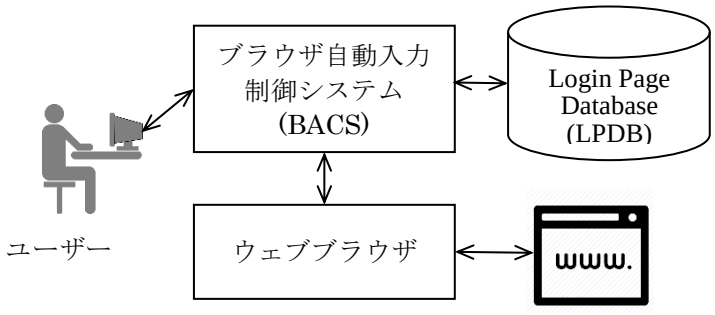


図 3.　提案したシステムのアーキテクチャ

ュ値を Login Page Database（以降 LPDB をいう）に保存する。ログインページが CSS ファイルで機能するように定義されている場合は、BACS が CSS ファイルのハッシュ値も同じレコードに挿入する。HTML と CSS データのハッシュ値を保存する理由は、これらのデータを格納し比較する負担を減らすためである。指定された URL アドレスのウェブページから算出したハッシュ値は、当該の LPDB レコードの HTML データのハッシュ値と同じなら、そのウェブページ内容が以前にログインした時のものと変わらないと判定する。BACS は、デジタル情報を保護するための最も安全なセキュアハッシュアルゴリズム 256（SHA256、略して）を用いてハッシュ値を算出する。SHA256 は、ログインページの HTML データのデータ文字列を 256 ビットの出力文字列に生成する。出力文字列は通常、元のデータよりはるかに小さい。SHA256 は、衝突に強いように設計されています[10]。つまり、異なるデータからに同じ 256 ビットのハッシュ値に作成される可能性が非常に低いことを意味する。

図 3 に示されているように、BACS はログインページに全ての入力テキストボックス（隠しテキストボックスを含む）の一覧を表示する。ユーザーはその一覧をみて、ブラウザのパスワードマネージャの自動入力を許可するかどうかを判断する。ユーザーが自動入力を許可する場合、BACS がブラウザの自動入力を有効にし、そのログインページを読むためにブラウザを再起動する。

４．パスワード自動入力を制御する手法

図 4 は、ウェブブラウザのパスワードマネージャの自動記入を許可すべきか否かを判定するフローチャートを示す。BACS を起動する時に、ユーザーは制御対象のブラウザを BACS に入力する。ダイアグラム 1 では、BACS はブラウザの自動入力機能を無効にする。ユーザーが閲覧したいログインページの URL アドレスが与えられると、BACS（ダイアグラム 2 を参照）はログインページと付属の CSS ファイルを読むようにブラウザに指示する。ダイアグラム 3 に示すように、BACS は Web ページ内のテキストボックスの input タグの存在を確認する。 Web ページに定義されたテキストボックスの入力タグが存在する場合、BACS は HTML データのハッシュ値を計算する（ダイアグラム 4 を参照）。 HTML デー

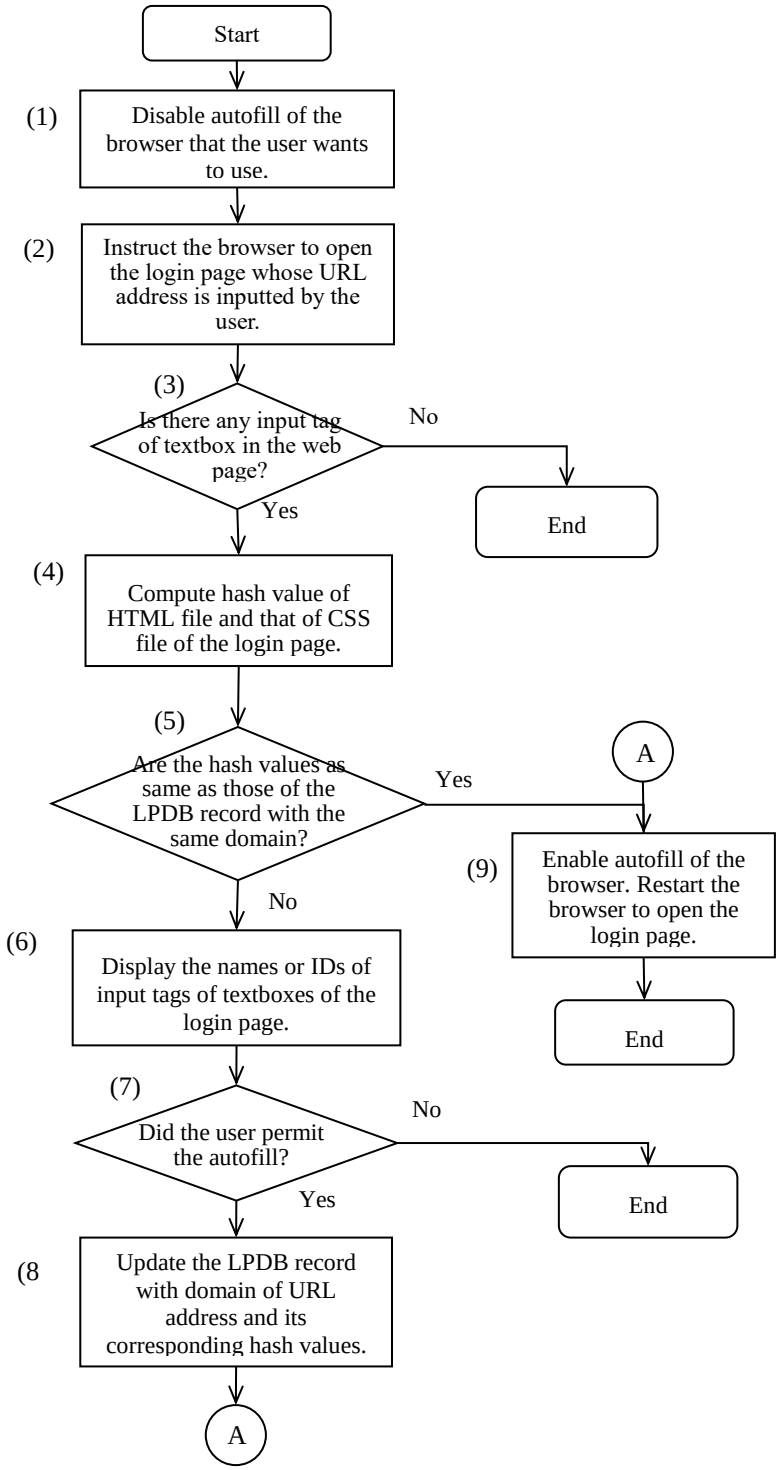


図 4. ログインページへの自動入力を許可するかどうかを判定するフローチャート

タに CSS ファイルが付いている場合、BACS は CSS ファイルのハッシュ値も計算する。

ダイアグラム 5 に示されるように、BACS は URL アドレスからドメイン名を抽出し、それをキーとして LPDB から当該レコードを読みだす。当該レコードが存在して尚且つその

レコードの HTML と CSS ハッシュ値がダイアグラム 4 で計算したものと一致するなら、ユーザーがすでにログインページの自動入力を許可していることを意味する。その際、ダイアグラム 9 で示されているように BACS は自動的にブラウザの自動入力を有効にする。その後、BACS はブラウザにログインページを再度読み込むように指示する。

ダイアグラム 6 において BACS は次の事項を確認する。指定されたドメイン名を持つ LPDB レコードが存在しない場合、または、そのようなレコードが存在するが、そのレコードのハッシュ値がダイアグラム 4 でのハッシュ値に一致しない場合は、BACS はログインページのテキストボックスの input タグの名前（または ID）を表示する。ダイアグラム 7 に示されるように、ユーザーは、BACS が表示したフィールド一覧をブラウザの画面のものに比較して隠しフィールドが存在するかどうかをチェックする。隠されたパスワードフィールドが存在しないことをユーザーが判断した場合、BACS はブラウザの自動入力を許可する。同じドメイン名を持つレコードが存在しない場合、BACS はログインページのドメイン名やそのハッシュ値からなる新しいレコードを LPDB に追加する。そうでなければ、BACS は、ダイアグラム 4 で計算されたハッシュ値で LPDB レコードのものを更新する（ダイアグラム 8 参照）。その後 BACS はダイアグラム 9 のプロセスに進む。

５．関連研究

パスワードマネージャに関する研究は、主に次の 3 つの方面に分けられる。一つ目は、Web アプリケーションごとにマスターシークレットに基づいた擬似ランダムで一意のパスワードを生成する手法に関する研究[3]。二つ目は、パスワードの安全な保管に関する研究[4]。三つ目は、フィッシングページ攻撃からユーザーを守る手法に関する研究[5]。Stock ら[6]はパスワードマネージャによって自動入力されたパスワードを盗む XSS 攻撃についての調査を行った。彼らは、その攻撃を対策する方法として、パスワードマネージャがダミーパスワードを使用してログインフォームのデータをリモートサーバに送信する直前にそのダミーパスワードを元のパスワードに置き換えることを提案した。 Blanchou ら[7]はパスワードマネージャのブラウザの拡張におけるいくつかの弱点を指摘し、オートフィルの危険性を示すフィッシング攻撃を検証した。彼らは、パスワード管理者がドメイン間のパスワード送信を防くべきだと指摘した。これらの研究とは異なった本研究では、攻撃者によって巧妙に作成されたフォームを使ってウェブページのパスワードフィールドを隠す問題を指摘し、その対策を提案した。

参考文献

[1] D. Silver, S. Jana, D. Boneh, E. Chen and C. Jackson, "Password Managers: Attacks and Defenses," in Proceeding of 23rd USENIX Security Symposium, pp. 449-464, 2014.

[2] B. Stock and M. Johns, "Protecting users against XSS-based password manager abuse, " In Proceedings of the 9th ACM symposium on Information, computer and communications security (ASIA CCS '14), pp. 183-194, 2014.

[3] S. Chiasson, P. C. Van Oorschot, and R. Biddle, "A usability study and critique of two password managers," In 15th USENIX Security Symposium (2006), pp. 1-16.

[4] R. Zhao, and C. Yue, "All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design," In Proceedings of the third ACM conference on Data and application security and privacy (2013), ACM, pp. 333-340.

[5] Z. E. Ye, S. Smith, and D. Anthony, "Trusted paths for browsers," in ACM Transactions on Information and System Security (TISSEC) 8, 2 (2005), pp.153-186.

[6] B. Stock and M. Johns, "Protecting Users Against XSS based Password Manager Abuse," In Proceedings of the 9th ACM symposium on Information, computer and communications security, pp. 183-194, 2014.

[7] M. Blanchou and P. Youn. Password managers: Exposing passwords everywhere, https://isecpartners.github.io/whitepapers/passwords/2013/11/05/Browser-Extension-Password-Managers.html.