WordPressプラグインで発生するCSRF脆弱性とSelf-XSS脆弱性の組み合わせによるXSS脆弱性

末吉　大輝[1]　酒井　隼[1]　齋藤　真幸[1]　齊藤　泰一[1]

概要：WordPress とは世界中で利用されているオープンソースのコンテンツ管理システム (CMS) である．WordPress はプラグインという機能によって機能を追加できる．被害者自身に悪意のあるスクリプトを注入させる Self-XSS という攻撃手法がある．この攻撃手法は悪意のあるスクリプトを被害者自身が書き込まなければならないため大きな被害には発展しにくい．しかし，Self-XSS を発生させるリクエストが偽造可能である場合に，攻撃者が任意のスクリプトを注入することが可能になる．本稿では，CSRF が可能なプラグインに対する Self-XSS 脆弱性との組み合わせによる XSS 脆弱性の事例とそれを引き起こす WordPress プラグインについて述べる．

キーワード：WordPress, CSRF, XSS, Self-XSS

1. はじめに

WordPress とは，主にブログサイトを管理するためのコンテンツマネジマントシステム (Content Management System,CMS) である [1]．PHP，MySQL を利用しオープンソースソフトウェアとして開発されている．サイトの見た目を変える”テーマ”や，機能拡張を行える”プラグイン”が豊富で，拡張性と自由度の高さから CMS の中でも圧倒的なシェアを誇っている．ユーザが web サイトに関する専門知識を持っていなくても，サードパーティ製プラグインやテーマを入手しブログに適用するだけで，自分に合った利便性の高いブログサイトを作り上げることが出来る．

プラグインは便利なものであるが，プラグインに不具合が存在する場合，結果として WordPress のサイトシステム全体に影響を及ぼす例が数多くある．WordPress 本体の重要な処理を行う HTTP リクエストには乱数のトークンが含まれ，管理者がコンテンツの更新や設定変更を行う際に，サーバは受信したリクエストが正規のリクエストであるかをトークンを用いた検証を行うことで Cross-Site Request Forgery(CSRF) を防いでいる．もしプラグインが，コンテンツの更新や設定変更時に送られてくる HTTP リクエストに対してトークンやパスワードを用いた検証を行わず，Referer のチェックも行っていない場合，そのプラグインが利用されているサイトに CSRF を利用した攻撃が可能になる．また，Self-Cross-Site Scripting(Self-XSS) 脆弱性を抱えているが放置されているプラグインも多く存在する．プラグインが持つ脆弱性が Self-XSS のみであれば，管理者が自らのサイト上でスクリプトを実行することしかできないため脅威にはならない．しかし，CSRF，ソーシャルエンジニアリング，XSSJacking[2] といった他の攻撃手法と組み合わせることによって Self-XSS は脅威になる．本稿では，WordPress プラグインの脆弱性事例で Self-XSS と CSRF と組み合わせた攻撃手法が利用可能であるか調査し，実際に Self-XSS と CSRF を組み合わせた脆弱性が報告されているプラグイン”HMS Testimonials”[3] を例に取り上げ脆弱性の発生原因を議論する．HMS Testimonials プラグインは，testimonial の作成，ブログ記事への投稿などを可能にするプラグインである．HMS Testimonials プラグインを利用している管理者を悪質な Web サイトにアクセスさせ，サイト管理者の意図しないリクエストを送信させることで，攻撃者の用意したスクリプトが Web サイトに格納・実行されてしまう CSRF と Self-XSS を組み合わせた脆弱性があることを確認する．

なお，本稿で議論するソフトウェアのバージョンは WordPress 4.9.8, HMS Testimonials プラグイン 2.0.9 である．

[1] 東京電機大学
Tokyo Denki University
5 Senju, Asahi-cho, Adachi-ku, Tokyo 120-8551, JAPAN

2. WordPress のセキュリティ

2.1 CSRF 対策

2.1.1 CSRF とは

Cross-Site Request Forgery(CSRF) とは，被害者がサイト A にログインした状態で攻撃者の罠サイトを閲覧することにより，被害者の権限で攻撃者の指定した処理が実行される脆弱性である．本稿では，Web サイトの管理を行う管理者の権限で実行されてしまう CSRF について述べる．

IPA では CSRF の根本的対策として以下を紹介している [4].

(1) 処理を実行するページを POST メソッドでアクセスするようにし，その hidden パラメータに秘密情報が挿入されるよう，前のページを自動生成して，実行ページではその値が正しい場合のみ処理を実行する．
(2) 処理を実行する直前のページで再度パスワードの入力を求め，実行ページでは，再度入力されたパスワードが正しい場合のみ処理を実行する．
(3) Referer が正しいリンク元かを確認し，正しい場合のみ処理を実行する．

WordPress では 1 および 3 の対策機能を提供している．

2.1.2 WordPress の秘密情報による対策

WordPress での CSRF 対策の一つとして，nonce という英数字列からなるランダムなハッシュ値の秘密情報を利用する方法がある [5]．WordPress の nonce は wp_create_nonce 関数によって生成される．

wp_nonce_url 関数を利用することで，nonce を含む URL を生成できる．

```
<?php echo wp_nonce_url(admin_url()); ?>
```

wp_nonce_field 関数を利用することで，nonce と referer を form に埋め込むための hidden フィールドを生成できる．

```
<?php echo wp_nonce_field('my-action'); ?>
```

上記を実行すると以下のフォームに変換される

```
<input type="hidden" id="_wpnonce"
name="_wpnonce" value="e134aacef3" />
<input type="hidden" name="_wp_http_referer"
value="/wp-admin/admin.php?page=myplugin" />
```

_wpnonce の value は wp_create_nonce 関数に応じて生成される．_wp_http_referer は呼び出し元のリクエスト URI を元に生成される．

クライアントから送信された nonce は wp_verify_nonce 関数，check_admin_referer 関数，check_ajax_referer 関数を利用することで，サーバサイドで有効な nonce であるか検証する．

```
if(isset($_POST['_wpnonce']) &&
   wp_verify_nonce($_POST['_wpnonce'])){
   // do something.
}
```

```
check_admin_referer('my-action','_wpnonce');
```

```
check_ajax_referer('my-action','_wpnonce');
```

2.1.3 WordPress の referer による対策

check_admin_referer 関数の引数を指定せず (または第一引数に-1 を指定) に利用すると，referer の有効性を検証する機能になる．この関数は，referer が管理画面からである場合を有効な referer としている．

```
check_admin_referer();
```

ただし，この方法は WordPress バージョン 3.2 以降非推奨となっており，現在は後方互換性のために残された機能である．

2.2 XSS とは

Cross-Site Scripting(XSS) とは，正常なサイトを閲覧したユーザに攻撃者が挿入したスクリプトを実行させる攻撃手法である．Web アプリケーションが，スクリプトを含んだ入力をサーバ側でそのまま利用することによって発生する．XSS が可能であると，被害者の Cookie 情報を窃取しなりすましを行ったり，サイトの改ざん，Drive-by Download によって訪問者にマルウェアをダウンロードさせるなど様々な攻撃の起点になり得る．

2.2.1 Self-XSS

Self-XSS は，XSS の中でも被害者自身がスクリプトを挿入することで，スクリプトが動作するものを指す．Self-XSS は被害者自身がスクリプトを挿入しなければ動作しないため，大きな被害には発展しにくい．図 1 は，Self-XSS 発生手順の図である．Self-XSS 可能であれば，そのスクリプトは自身のブラウザ上で動作する．

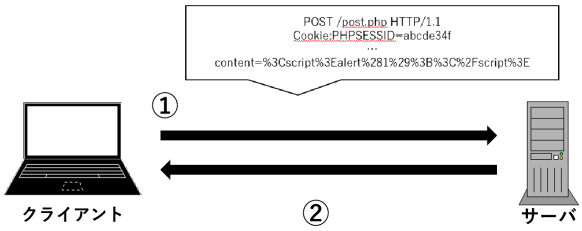

図 1　Self-XSS 模式図

(1) ユーザは正規の手順でサーバにリクエストを送信するが，送信するリクエストの中にスクリプトを含める
(2) サーバがリクエストに従って処理を行い，レスポンスを返す

スクリプトはサーバがユーザからの入力を無害化せずにデータベースに格納し，後で出力するタイミングで動作する．またはレスポンスに入力値をそのまま含めて返すことによって反射的に動作する．

3. CSRF と Self-XSS を組み合わせた XSS 攻撃

Self-XSS を可能とするリクエストが偽造可能であるとき，偽造したリクエストを送信することで XSS に発展する場合がある．図 2 にその攻撃手順を示す．

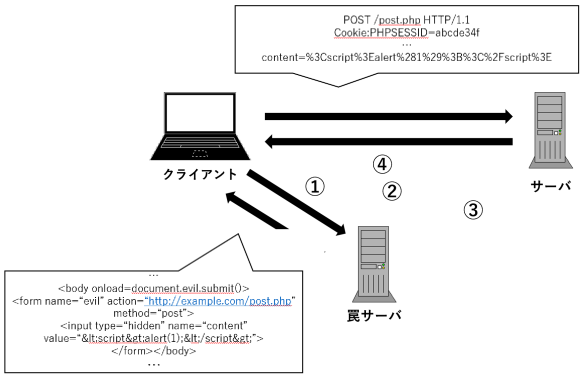

図 2　CSRF と Self-XSS による XSS 攻撃手順

(1) 被害者は攻撃者が用意した悪意のある Web サーバにアクセスする
(2) 罠サーバは悪意のあるスクリプトを含んだレスポンスを返す
(3) 罠 Web サイト内に用意されていた偽のリクエストを指定したサーバに送信
(4) サーバが偽リクエストの処理を行い，レスポンスを返す

罠サーバから返されるスクリプトには，指定したサーバへ，攻撃者の用意したリクエストを送信する内容が含まれている．それによって被害者のブラウザは，リクエストをサーバへ送信する．セッション ID を Cookie に保存する Web サイトである場合，セッション ID を含めた Cookie を偽リクエストに自動的に付加して送ってしまうため，そのセッションの権限で，処理が実行されてしまう．偽造されたリクエストが Self-XSS 可能である場合，Self-XSS を引き起こすリクエストを送信させることで，攻撃者の用意したスクリプトを格納・実行させることができてしまう．

3.1 修正済みプラグインに対する CSRF と Self-XSS 組み合わせ事例調査

2018 年 8 月 19 日現在 JVN に掲載されている，WordPress プラグインにおける CSRF 脆弱性は 137 件である [6]．これらの修正前のプラグインに対して，その CSRF 脆弱性を Self-XSS と組み合わせて XSS 可能であったかどうかを調査した．

公開されている実証コードで Self-XSS を利用しているもの，または実際に脆弱性のあるバージョンを入手し，Self-XSS を利用可能であるか検証を行った．公開中止によってプラグインを入手できず，実証コードによって Self-XSS を利用可能か検証できないものについては不明として扱った．

その結果，Self-XSS 可能であるものは 72 件，Self-XSS 不可能であるものは 42 件，不明であるものは 23 件であった．不明であるものを除くと，CSRF 脆弱性を持っていた約 63 %のプラグインが CSRF で偽造したリクエストによって Self-XSS を悪用可能であった．しかし，現在それらは CSRF 脆弱性が修正されていると同時に Self-XSS 脆弱性も修正されたため，本稿で議論しているような XSS 脆弱性は発生しない．

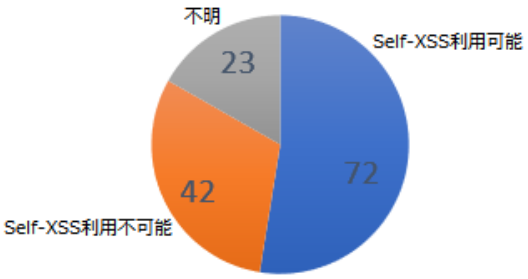

図 3　修正済みプラグインに対する CSRF と Self-XSS の組み合わせ可否事例

次章では，JVN に掲載されている内の 1 つである HMS Testimonials プラグインを例として説明する．

4. HMS Testimonials プラグインの例 (CVE-2013-4240)

”HMS Testimonials プラグイン”は，testimonial を作成し，ブログのページや投稿に testimonial を表示させるためのプラグインである [3]．

Add New から testimonial の項目を入力し，testimonial を作成する．Groups，Custom Field などの機能で，特定のページに testimonial を整理して表示したり，testimonial の項目を自由にカスタマイズすることができる．

このプラグインは，バージョン 2.0.11 未満に CSRF 脆弱性が存在する．本稿ではバージョン 2.0.9 で検証を行った．

4.1 Self-XSS

このプラグインでは，testimonial の作成リクエストに，Self-XSS 可能な個所が存在する．name，testimonial，url パラメータにスクリプトを注入可能である．注入したスクリプトはデータベースに格納され，削除されるまで永続的に残り続ける．格納されたスクリプトは testimonial の確認ページや，testimonial を投稿したページで動作する．

QueryString

Name	Value
page	hms-testimonials-addnew

Body

Name	Value
name	<script>alert(1);</script>
image	
testimonial_date	
url	http://<script>alert(3);</script>
testimonial	<script>alert(2);</script>
display	1
save	Save Testimonial

図 4　Self-XSS を起こすリクエストパラメータ

4.2 CSRF

前述のリクエストは nonce や referer の検証がされていない．そのために CSRF が可能になってしまっている．

図 5 は，CSRF と Self-XSS を利用してスクリプトを注入する実証コードである [7]．管理者としてログイン済みの被害者が下記のフォームリクエストを送信してしまうと，Self-XSS を通して攻撃者の用意したスクリプトがデータベースに格納される．

```
<form method="post" action="http://example.com/wp-admin/admin.php?page=hms-testimonials-addnew";>
    <input type="hidden" name="name" value="&lt;script>alert('xss')&lt;/script>">
    <input type="hidden" name="image" value="0">
    <input type="hidden" name="testimonial_date" value="08/08/2013">
    <input type="hidden" name="url" value="&lt;script>alert(String.fromCharCode(88,83,83))&lt;/script>">
    <input type="hidden" name="testimonial" value="&lt;script>alert('xss')&lt;/script>">
    <input type="hidden" name="display" value="1">
    <input type="submit" name="save" value="Save Testimonial">
</form>
```

図 5　CSRF の実証コード

testimonial を作成する際に送信するリクエストに，CSRF トークンが使用されていないため，testimonial を作成するリクエストを悪用されてしまう．結果として，リクエスト中の Self-XSS 可能なパラメータにスクリプトを挿入され，データベースに格納・実行されてしまう．

4.3 プラグインの対策

修正された HMS Testimonials 2.0.11 は，wp_nonce_field 関数と check_admin_referer 関数によって nonce の設置と検証を行う処理が追加された．

```
<form method="post" action="<?php echo admin_url('admin.php?page=hms-testimonials-addnew'); ?>" accept-charset="UTF-8">
<div id="poststuff">
    <div id="post-body" class="metabox-holder columns-2">
        <div id="post-body-content">
```

図 6　CSRF 修正前 (wp_nonce_field)

```
<form method="post" action="<?php echo admin_url('admin.php?page=hms-testimonials-addnew'); ?>" accept-charset="UTF-8">
<?php wp_nonce_field( action: 'hms-testimonials-new'); ?>
<div id="poststuff">
    <div id="post-body" class="metabox-holder columns-2">
        <div id="post-body-content">
```

図 7　CSRF 修正後 (wp_nonce_field)

```
if (isset($_POST) && (count($_POST)>0)) {
    if (!isset($_POST['name']) || trim($_POST['name']) == '')
        $errors[] = 'Please enter a name for this testimonial.';

    if (!isset($_POST['testimonial']) || (trim($_POST['testimonial'])==''))
        $errors[] = 'You forgot to enter the testimonial.';
```

図 8　CSRF 修正前 (check_admin_referer)

```
if (isset($_POST) && (count($_POST)>0)) {
    check_admin_referer( action: 'hms-testimonials-new');

    if (!isset($_POST['name']) || trim($_POST['name']) == '')
        $errors[] = 'Please enter a name for this testimonial.';

    if (!isset($_POST['testimonial']) || (trim($_POST['testimonial'])==''))
        $errors[] = 'You forgot to enter the testimonial.';
```

図 9　CSRF 修正後 (check_admin_referer)

加えて，Self-XSS が可能であった変数に対して HTML フィルタライブラリである HTML Purifier[8] を用いて，ホワイトリスト方式でタグを無害化させる処理も追加された．

```
$_POST = stripslashes_deep($_POST);

$display_order = $this->wpdb->get_var( query: "SELECT `display_order` FROM `"
.$this->wpdb->prefix."hms_testimonials` ORDER BY `display_order` DESC LIMIT 1");

$this->wpdb->insert( table: $this->wpdb->prefix."hms_testimonials",
    array(
        'blog_id' => $this->blog_id, 'user_id' => $this->current_user->ID, 'name' => trim($_POST['name']),
        'testimonial' => trim($_POST['testimonial']), 'display' => $display, 'display_order' => ($display_order+1),
        'url' => $url, 'testimonial_date' => $testimonial_date, 'created_at' => date( format: 'Y-m-d h:i:s'),
        'image' => (($image_url != '') ? (int)$_POST['image'] : 0)));
```

図 10　Self-XSS 修正前

```php
$_POST = stripslashes_deep($_POST);

$display_order = $this->wpdb->get_var( query: "SELECT `display_order` FROM `"
.$this->wpdb->prefix."hms_testimonials` ORDER BY `display_order` DESC LIMIT 1");

/**
 * Purify the testimonial field
 **/
if (!class_exists( class_name: HTMLPurifier)) {
    require_once HMS_TESTIMONIALS . 'HTMLPurifier/HTMLPurifier.auto.php';
}

$config = HTMLPurifier_Config::createDefault();

/**
 * Just in case some users can't use the cache, kill it for all
 **/
$config->set('Core', 'DefinitionCache', null);
$config->set('URI', 'AllowedSchemes', array('http' => true, 'https' => true, 'mailto' => true,
'ftp' => true, 'nntp' => true, 'news' => true));
$purifier = new HTMLPurifier($config);

$testimonial = $purifier->purify(trim($_POST['testimonial']));
$name = $purifier->purify(trim($_POST['name']));


$this->wpdb->insert( table: $this->wpdb->prefix."hms_testimonials",
    array(
        'blog_id' => $this->blog_id, 'user_id' => $this->current_user->ID, 'name' => $name,
        'testimonial' => $testimonial, 'display' => $display, 'display_order' => ($display_order+1),
        'url' => strip_tags($url), 'testimonial_date' => $testimonial_date, 'created_at' => date( format: 'Y-m-d h:i:s'),
        'image' => (($image_url != '') ? (int)$_POST['image'] : 0)));
```

図 11　Self-XSS 修正後

図 10 は修正前のコードである．POST パラメータ $_POST['name'] や$_POST['testimonial']，および URL が格納されている$url を無害化せずにデータベースに格納している．図 11 は修正後のコードである．POST パラメータを HTML Purifier によって無害化し，$url は strip_tags 関数を使ってタグを除去している．これらによって入力がデータベースに格納される前に，script タグ等を削除して Self-XSS を防いでいる．

5. CSRF と Self-XSS の組み合わせによる XSS を許す WordPress プラグインの条件

例に示したような XSS を発生させるには，以下の動作条件を満たす必要がある．

(1) HTTP リクエストを偽造可能である
(2) Self-XSS 脆弱性が存在する
(3) 偽造した HTTP リクエストから Self-XSS 脆弱性を動作させられる

条件 1 を満たすには CSRF のような脆弱性が存在することが必要である．HMS Testimonials プラグインは nonce や referer の検証を行っていなかったため，CSRF によって HTTP リクエストが偽造可能であった．

条件 2 を満たすにはプラグインに何らかの理由によって Self-XSS 脆弱性が存在することが必要である．プラグインの利用権限が管理者のみであるためにサニタイズの必要がないという開発時の思考，スクリプトの挿入を仕様として許容している場合，または意図せず Self-XSS を残していた場合に，Self-XSS 可能な状態が発生する．HMS Testimonials は name パラメータ等にスクリプトを挿入可能であり，Self-XSS 脆弱性が存在した．

最後に，条件 1,2 を複合した条件 3 が成立することで，CSRF と Self-XSS を組み合わせることができる．HMS Testimonials は testimonial の作成リクエストが偽造可能であり，その中に含まれる name パラメータを通して Self-XSS が可能であった．これらの条件が重なって CSRF と Self-XSS の組み合わせが可能になっていた．

6. まとめ

HMS Testimonials プラグインは管理者が Self-XSS 可能なプラグインであった．また，そのリクエストに対して nonce や referer の検証を行っていなかったため，CSRF 可能でもあった．攻撃者は CSRF を利用して攻撃者の指定したスクリプトを挿入し，格納型 XSS のようにスクリプトを設置できた．この脆弱性は Self-XSS が可能であり，攻撃者が CSRF 脆弱性によって Self-XSS を利用することによって，XSS 攻撃が可能になる事例であった．

このような脆弱性は，CSRF に対策しさえすれば，とりあえず Self-XSS の第三者悪用を防ぐことができる．しかし，調査した範囲では，脆弱性修正時に Self-XSS を残しているプラグインは見つからなかった．CSRF と Self-XSS の組み合わせによる XSS 脆弱性を持っていたプラグインは，全て CSRF のみならず Self-XSS 脆弱性までも含めて修正していた．HMS Testimonials の例であれば，CSRF の対策をしてしまえば，Self-XSS は大きな脅威にはならないため，対策しなくてはならない必要性は無かった．しかし，開発者にとって Self-XSS が可能であることは意図したものではなく，バグの一種であったため修正されたのではないだろうか．

Self-XSS 脆弱性自体は大きな脅威ではないが，他の脆弱性と組み合わせて利用される可能性は常に潜在している．WordPress プラグインは非常に自由度が高く，HTML の自動生成といったことを，PHP の技術知識無しで可能にしてくれるものもある．しかしその反面，機能の複雑性から Self-XSS 脆弱性を抱えているが放置されているプラグインも多く存在する．例えば Javascript の挿入を仕様として許可している場合や，意図せず Self-XSS 可能な状態になっているプラグインが該当する．とりわけ，意図せず Self-XSS 可能な状態のプラグインは多く存在している．不必要にスクリプトを許可してしまう構造はセキュリティ上好ましくない．Self-XSS 悪用による被害を最小限に抑えるためには，開発者が Self-XSS に対する理解を深め，必要があるのではないだろうか．

7. 参考文献

参考文献

[1] Blog Tool, Publishing Platform, and CMS - WordPress, 入手先 ⟨https://wordpress.org/⟩
[2] New Attack ”XSSJacking” Combines Clickjacking, Pastejacking, and Self-XSS 入手先 ⟨https://www.bleepingcomputer.com/news/security/new-attack-xssjacking-combines-clickjacking-pastejacking-and-self-xss/⟩
[3] HMS Testimonials — WordPress.org，入手先 ⟨https://wordpress.org/plugins/hms-

testimonials/#developers⟩ (2018.08.19).
[4] IPA：安全なウェブサイトの作り方第 7 版，入手先 ⟨https://www.ipa.go.jp/files/000017316.pdf⟩ (2018.08.06).
[5] WordPress Codex: WordPress Nonces，入手先 ⟨https://codex.wordpress.org/WordPress_Nonces⟩ (2018.08.06).
[6] JVN iPedia - 脆弱性対策情報データベース，入手先 ⟨https://jvndb.jvn.jp/search/index.php?mode=_vulnerability_search_IA_VulnSearch&keyword=wordpress+%83v%83%89%83O%83C%83%93&cwe=352⟩ (2018.08.19).
[7] Full Disclosure: [RCA-201308-01] HMS Testimonials 2.0.10 WP plugin - Multiple vulnerabilities，入手先 ⟨http://seclists.org/fulldisclosure/2013/Aug/96⟩ (2018.08.19).
[8] HTML Purifier - Filter your HTML the standards-compliant way!，入手先 ⟨http://htmlpurifier.org/⟩ (2018.08.19).