

「Javaアプリケーション脆弱性事例調査資料」について

- この資料は、Javaプログラマである皆様に、脆弱性を身近な問題として感じてもらい、セキュアコーディングの重要性を認識していただくことを目指して作成しています。
- 「Javaセキュアコーディングスタンダード CERT/Oracle版」と合わせて、セキュアコーディングに関する理解を深めるためにご利用ください。

JPCERTコーディネーションセンター
セキュアコーディングプロジェクト
secure-coding@jpcert.or.jp

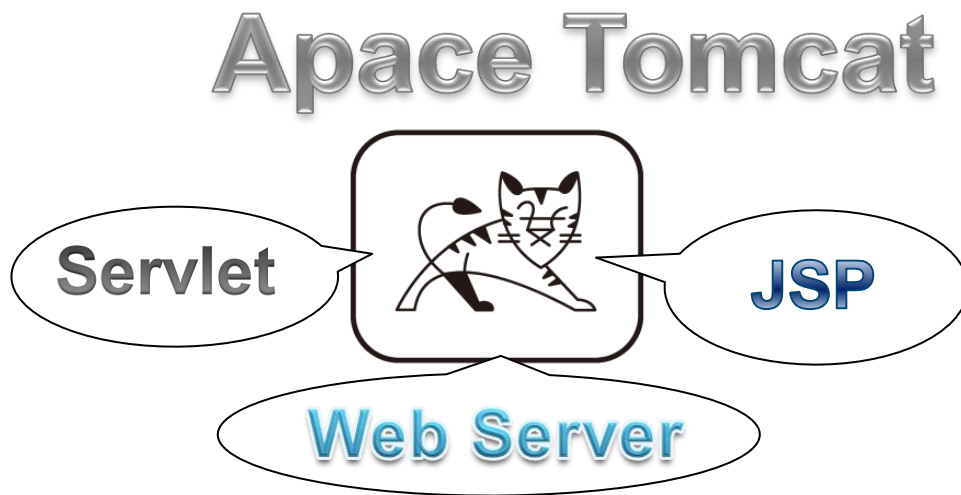
Apache Tomcat における クロスサイトリクエストフォージェリ (CSRF)保護メカニズム回避の脆弱性

CVE-2012-4431

JVNDB-2012-005750

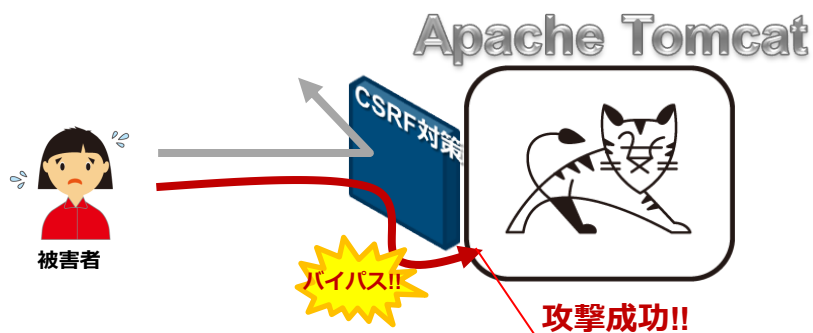
Apache Tomcatとは

- Java Servlet や JavaServer Pages (JSP) を実行するためのサーブレットコンテナ(サーブレットエンジン)
- CSRF対策のために、トークンを使ったリクエストフォームの検証機能が実装されている



脆弱性の概要

- Apache Tomcatには、クロスサイトリクエストフォージェリ対策をバイパスできる脆弱性が存在する
- 脆弱性を悪用されることで、被害者が意図しない操作を実行させられる可能性がある
- Apache Tomcat上で展開されるWebアプリケーションの機能を不正に実行させることが可能となる



通常処理フロー

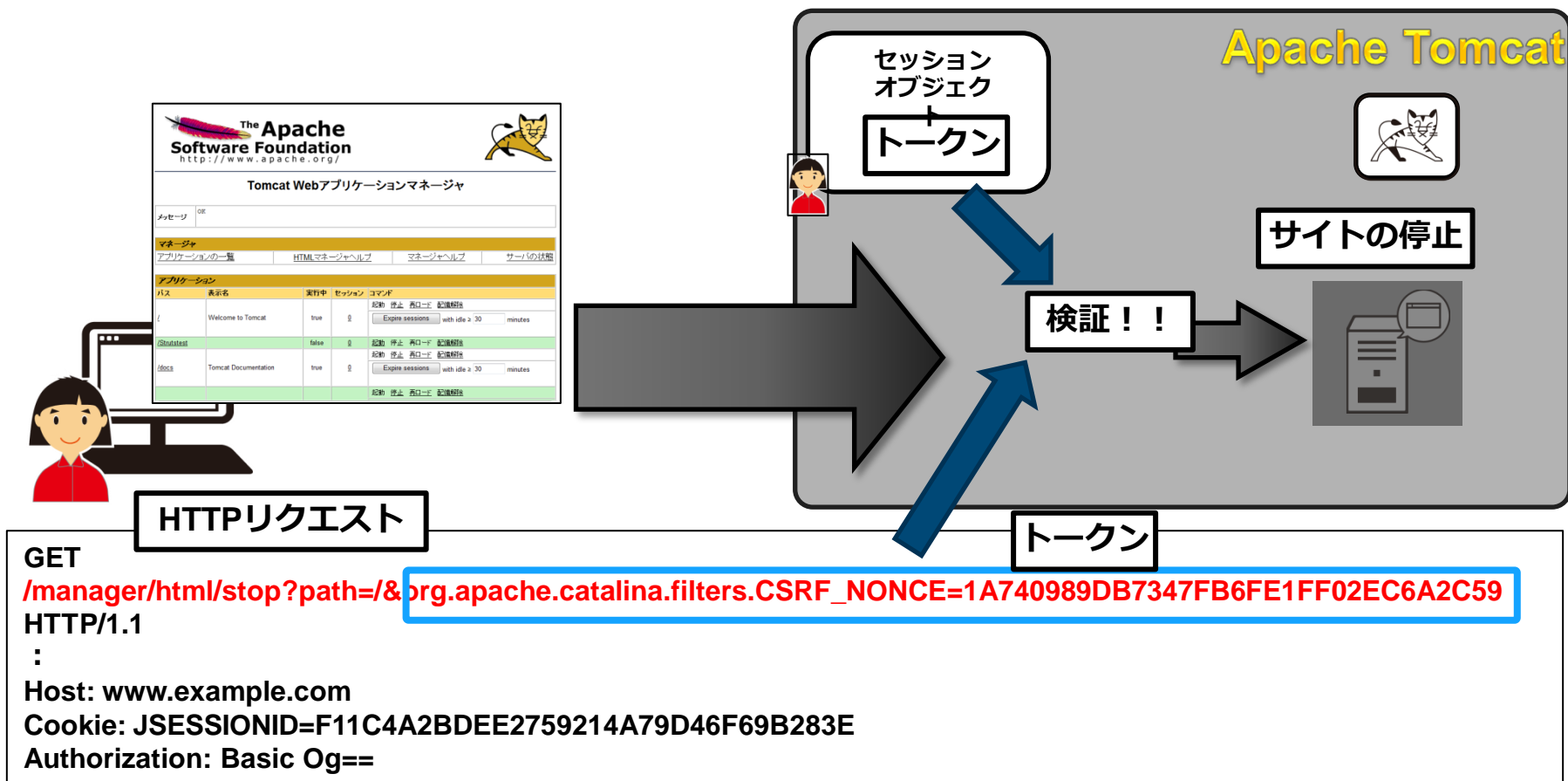
Tomcat Webアプリケーションマネージャのサイト停止機能における処理フローを解説する。

サイト停止機能を実行する際のApache Tomcatの処理フロー

- ① Tomcat Managerにクライアント(サイト管理者)がBasic認証でログインする。
- ② 管理者画面にアクセス時に、アプリケーションはトークンを発行しForm要素に埋め込む。さらにセッション変数にトークンを格納する
- ③ クライアントがサイト停止機能を実行しリクエストが送信される。
- ④ アプリケーションは送信されてきたトークンとセッション変数に格納されているトークンが同一かを検証する
- ⑤ アプリケーションがリクエストを受信し、処理を実行する。
- ⑥ 結果を含むレスポンスがクライアント (サイト管理者) へ送信される。

サイト停止機能実行時

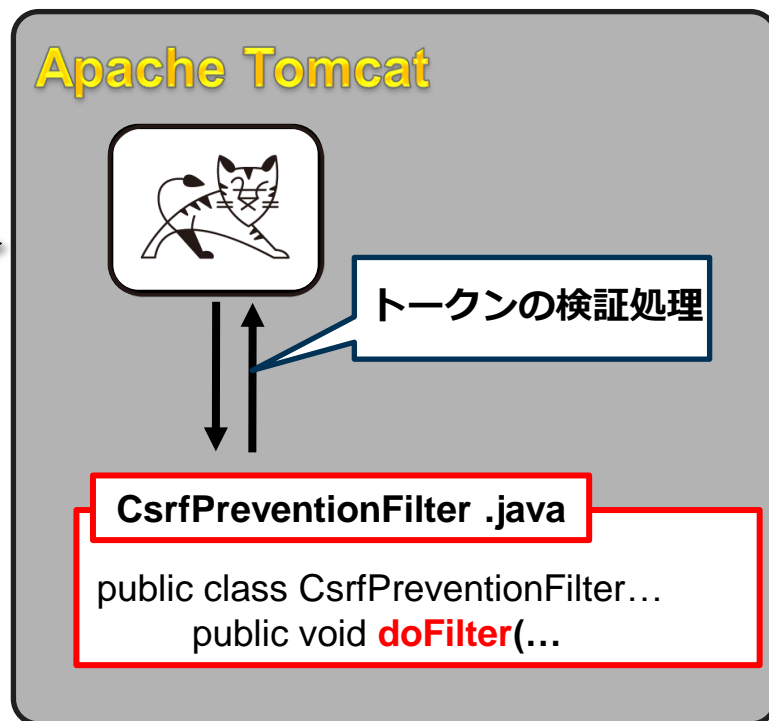
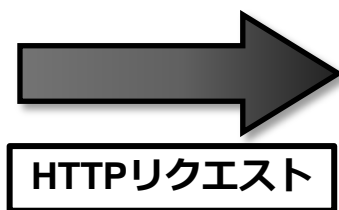
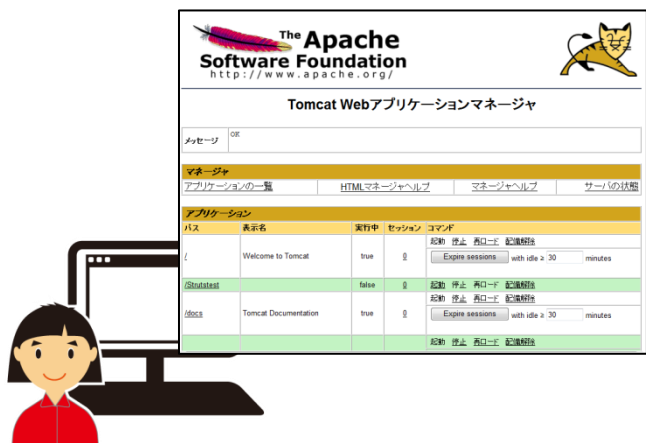
アプリケーションはリクエストを受信後、CSRFトークンを検証を実施し、正規のトークンが送信されてきたときのみ機能を実行する。



CSRFトークンは付与され、検証もされている。

doFilterメソッドによるトークンの検証

トークンの検証はCsrfPreventionFilterクラスの doFilter メソッドで実行される。



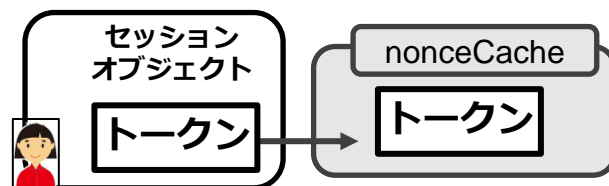
doFilterメソッドによるトークンの検証: セッションからトークンの取得

CsrfPreventionFilter.java

```
public class CsrfPreventionFilter extends FilterBase {
    :
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
    :
    HttpServletRequest req = (HttpServletRequest) request;
    :
    @SuppressWarnings("unchecked")
    LruCache<String> nonceCache =
        (LruCache<String>) req.getSession(true).getAttribute(
            Constants.CSRF_NONCE_SESSION_ATTR_NAME);
    :
    if (!skipNonceCheck) {
        String previousNonce =
            req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM); ... (B)

        if (nonceCache != null && !nonceCache.contains(previousNonce)) { ... (C)
            res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
            return;
        }
    }
}
```

リクエストのセッションからセッション変数を取得し、変数nonceCacheに格納する。



文字列 "org.apache.catalina.filters.CSRF_NONCE"

doFilterメソッドによるトークンの検証: リクエストからトークンの取得

CsrfPreventionFilter.java

```
public class CsrfPreventionFilter extends FilterBase {
    :
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
    :
    HttpServletRequest req = (HttpServletRequest) request;
    :
    @SuppressWarnings("unchecked")
    LruCache<String> nonceCache =
        (LruCache<String>) req.getSession(true).getAttribute(
            Constants.CSRF_NONCE_SESSION_ATTR_NAME);

    if (!skipNonceCheck) {
        String previousNonce =
            req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM);

        if (nonceCache != null && !nonceCache.contains(previousNonce)) {
            res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
            return;
        }
    }
}
```

リクエストのパラメータから変数を取得し、
変数previousNonceに格納する。

HTTPリクエスト

```
GET
 /manager/html/stop?path=/&org.apache.catalina.filters.CSRF_NONCE=1A740989DB7347FB6FE1FF02EC6A2C59 HTTP/1.1
 :
 Host: www.example.com
 Cookie: JSESSIONID=F11C4A2BDEE2759214A79D46F69B283E
 Authorization: Basic Og==
```

previousNonce

トークン

文字列 "org.apache.catalina.filters.CSRF_NONCE"

doFilterメソッドによるトークンの取得: トークンの比較

CsrfPreventionFilter.java

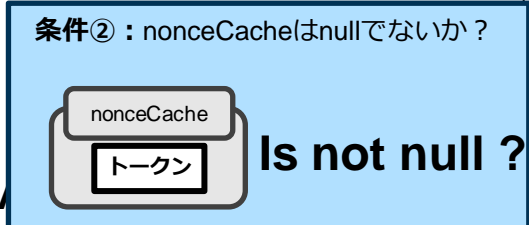
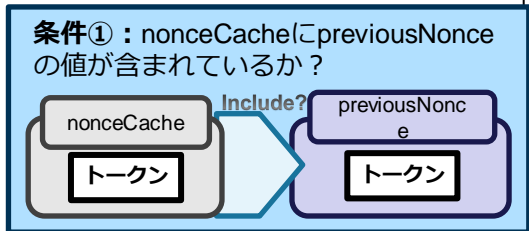
```
public class CsrfPreventionFilter extends Filter {
    :
    public void doFilter(ServletRequest request,
        throws IOException, ServletException {
    :
    HttpServletRequest req = (HttpServletRequest) request;
    :
    @SuppressWarnings("unchecked")
    LruCache<String> nonceCache =
        (LruCache<String>) req.getSession(true).getAttribute(
            Constants.CSRF_NONCE_SESSION_ATTR_NAME);

    if (!skipNonceCheck) {
        String previousNonce =
            req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM);

        if (nonceCache != null && !nonceCache.contains(previousNonce)) {
            res.sendError(
                HttpServletResponse.SC_FORBIDDEN); //エラー処理
            return;
        }
    }
    : //正常処理
}
```

トークンの検証処理を実行する。
下記の2つの条件が両方とも成り立つ場合は
エラーとして処理される。

- ① 変数nonceCacheの値に変数previousNonceが含まれていない。
- ② 変数nonceCacheがnullでない。



if (nonceCache != null && !nonceCache.contains(previousNonce)) {
res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
return;
}

条件①、②がともに成立しないときに正常処理となる。

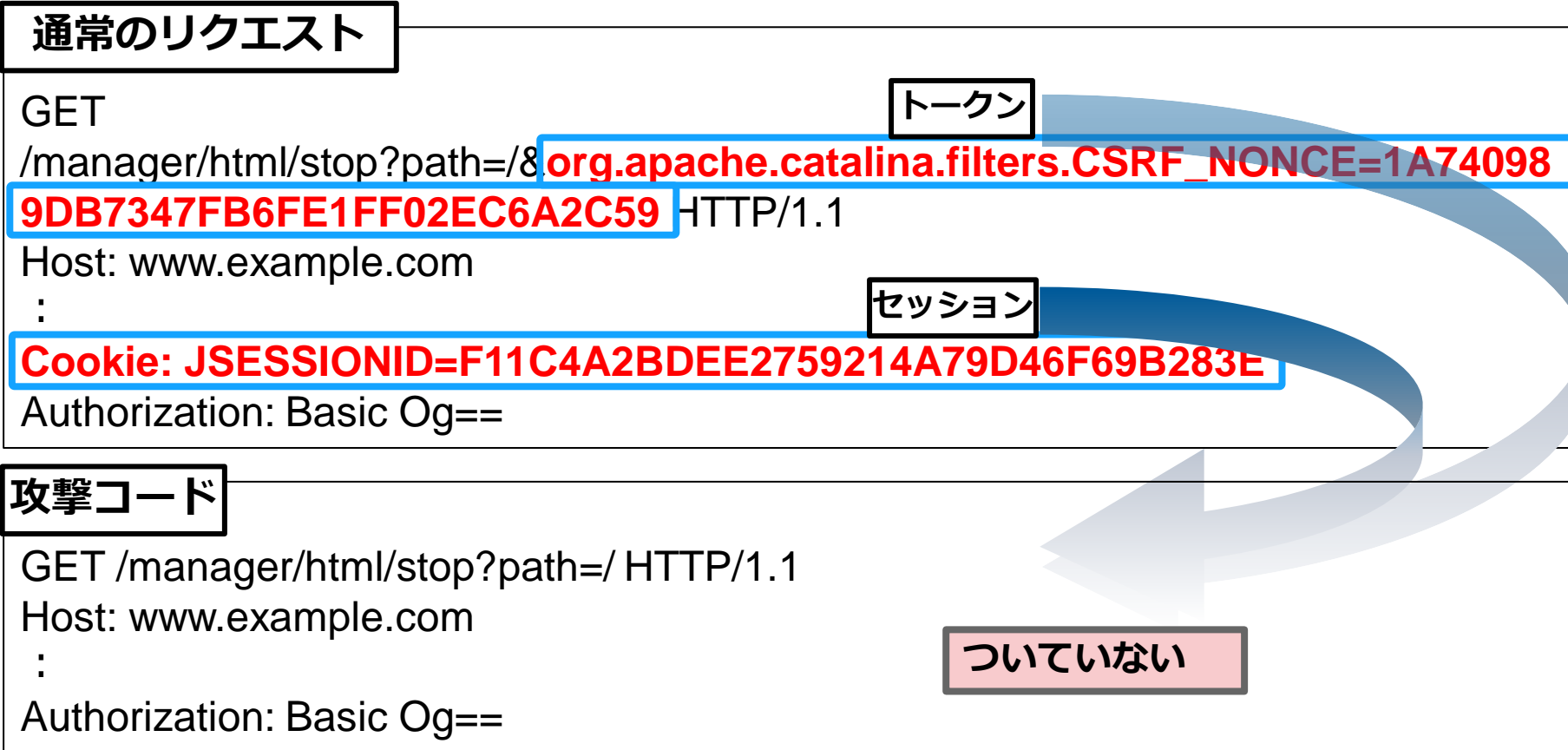
攻撃コード (正常なリクエストとの比較)

通常のリクエスト

```
GET /manager/html/stop?path=/&org.apache.catalina.filters.CSRF_NONCE=1A74098  
9DB7347FB6FE1FF02EC6A2C59 HTTP/1.1  
Host: www.example.com  
:  
Cookie: JSESSIONID=F11C4A2BDEE2759214A79D46F69B283E  
Authorization: Basic Og==
```

トークン

セッション

A diagram illustrating the difference between a normal request and an attack code. The top section, '通常のリクエスト' (Normal Request), shows a GET request with a path containing a CSRF nonce and a session ID. The nonce and session ID are highlighted in red and blue boxes, with labels 'トークン' (Token) and 'セッション' (Session) pointing to them. The bottom section, '攻撃コード' (Attack Code), shows the same GET request but with the nonce and session ID removed. A pink box labeled 'ついていない' (Not attached) points to the missing fields. A large blue arrow points from the normal request to the attack code, and a large grey arrow points from the attack code back to the normal request.

攻撃コード

```
GET /manager/html/stop?path=/ HTTP/1.1  
Host: www.example.com  
:  
Authorization: Basic Og==
```

ついていない

■ 攻撃コードのポイント

リクエストに含まれるトークン (GETパラメータの
ord.apache.catalina.filters.CSRF_NONCE) とセッション(Cookieヘッダ)が削除さ
れている

攻撃コード実行時の処理フロー

サイト停止機能を実行する際のApache Tomcatの処理フロー

- ① Tomcat Managerにクライアント(サイト管理者)がBasic認証でログインする。
- ② 管理者画面にアクセス時に、アプリケーションはトークンを発行しForm要素に埋め込む。さらにセッション変数にトークンを格納する
- ③ クライアントがサイト停止機能を実行しリクエストが送信される。
- ④ アプリケーションは送信されてきたトークンとセッション変数に格納されているトークンが同一かを検証する
- ⑤ アプリケーションがリクエストを受信し、処理を実行する。
- ⑥ 結果を含むレスポンスがクライアント(サイト管理者)へ送信される。

④の検証処理が不十分だった!!

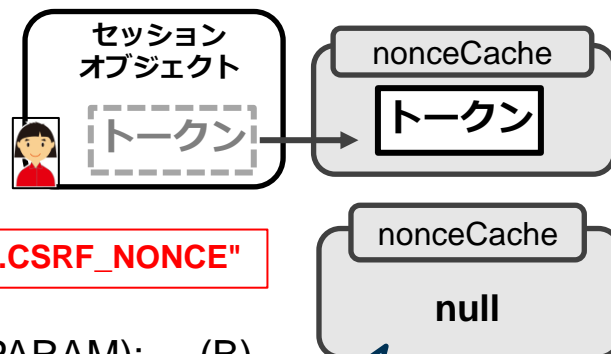
④ 送信されてきたトークンとセッション変数のトークンの検証

CsrfPreventionFilter.java

```
public class CsrfPreventionFilter extends FilterBase {
    :
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        :
        HttpServletRequest req = (HttpServletRequest) request;
        :
        @SuppressWarnings("unchecked")
        LruCache<String> nonceCache =
            (LruCache<String>) req.getSession(true).getAttribute(
                Constants.CSRF_NONCE_SESSION_ATTR_NAME);
        :
        if (!skipNonceCheck) {
            String previousNonce =
                req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM); ... (B)

            if (nonceCache != null && !nonceCache.contains(previousNonce)) { ... (C)
                res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
                return;
            }
        }
    }
}
```

リクエストのセッションからセッション変数を取得し、変数 nonceCache に格納する。



文字列 "org.apache.catalina.filters.CSRF_NONCE"

攻撃コードではセッションが削除されており、セッションオブジェクトが存在しないため、nullが格納される。

④ 送信されてきたトークンとセッション変数のトークンの検証

CsrfPreventionFilter.java

```
public class CsrfPreventionFilter extends FilterBase {
    :
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        :
        HttpServletRequest req = (HttpServletRequest) request;
        :
        @SuppressWarnings("unchecked")
        LruCache<String> nonceCache =
            (LruCache<String>) req.getSession(true).getAttribute(
                Constants.CSRF_NONCE_SESSION_ATTR_NAME);

        if (!skipNonceCheck) {
            String previousNonce =
                req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM);

            if (nonceCache != null && !nonceCache.contains(previousNonce)) { ...(C)
                res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
                return;
            }
        }
    }
}
```

リクエストの**パラメータ**から変数を取
得し、変数previousNonceに格納する。

HTTPリクエスト

```
GET /manager/html/stop?path=/ HTTP/1.1
Host: www.example.com
:
Authorization: Basic Og==
```

previousNonce

null

攻撃コードにはトーク
ンが含まれていないの
でnullが格納される。

文字列 "org.apache.catalina.filters.CSRF_NONCE"

④ 送信されてきたトークンとセッション変数のトークンの検証

CsrfPreventionFilter.java

```
public class CsrfPreventionFilter extends FilterBase {
    :
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        :
        HttpServletRequest req = (HttpServletRequest) request;
        :
        @SuppressWarnings("unchecked")
        LruCache<String> nonceCache =
            (LruCache<String>) req.getSession(true).getAttribute(
                Constants.CSRF_NONCE_SESSION_ATTR_NAME);

        if (!skipNonceCheck) {
            String previousNonce =
                req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM);

            if (nonceCache != null && !nonceCache.contains(previousNonce)) {
                res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
                return;
            }
        }
        : //正常処理
    }
```

トークンの検証処理を実行する。
下記の2つの条件が両方とも成り立つ場合は
エラーとして処理される。

- ① 変数nonceCacheの値に変数previousNonceが含まれていない。
- ② 変数nonceCacheがnullでない。

```
if (nonceCache != null && !nonceCache.contains(previousNonce)) {
    res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
    return;
}
```

④ 送信されてきたトークンとセッション変数のトークンの検証

CsrfPreventionFilter.java

①は正常なトークンの検証処理のため問題ないが、問題は②の条件。

「変数nonceCacheがnull」 = 「セッションがそのものが存在しない」

であり、セッション自体を削除する(Cookieヘッダを削除すること)で変数nonceCacheがnullとなり、このトークンの検証処理をバイパスすることができる!!

```
@SuppressWarnings("unchecked")
LruCache<String> nonceCache =
    (LruCache<String>) req.getSession(true).getAttribute(
        Constants.CSRF_NONCE_SESSION_ATTR_NAME);

if (!skipNonceCheck) {
    String previousNonce =
        req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM);

    if (nonceCache != null && !nonceCache.contains(previousNonce)) {
        res.sendError(HttpServletResponse.SC_FORBIDDEN); //エラー処理
        return;
    }
}
: //正常処理
}
```

response response,FilterChain chain)

トークンの検証処理を実行する。
下記の2つの条件が両方とも成り立つ場合は
エラーとして処理される。

- ① 変数nonceCacheの値に変数previousNonceが含まれていない。
- ② 変数nonceCacheがnullでない。

セッションを削除したリクエストの処理

通常、セッションを削除(Cookieヘッダを削除)してしまうと、Webアプリケーションはユーザーを認識できなくなり、認証エラーが発生する。



しかし、Apache TomcatのTomcat WebアプリケーションマネージャはBasic認証を使用しており、認証にセッション(Cookieヘッダ)を使用していない。

※Basic認証ではAuthorizationヘッダを使用する



CSRF対策であるトークン検証処理にてセッション(Cookieヘッダ)が無い場合を想定していなかったため、認証エラーが発生せず機能が実行されてしまう!!

●今回のアプリケーションにおける具体的な問題点

セッションが存在しないケース(Cookie以外によるセッション管理)を想定していなかった。

セッションが存在しない場合は、CSRF対策をパスしてしまっていた。

●問題点に対してどうすべきだったか。

アプリケーションの仕様(今回の場合はセッションの管理方式)を確認し、適切なCSRF対策を選択する必要があった。

修正版コード

脆弱性はバージョン7.0.32、6.0.36にて修正が適用されている

サイト停止機能を実行する際のApache Tomcatの処理フロー

- ① Tomcat Managerにクライアント（サイト管理者）がBasic認証でログインする。
- ② 管理者画面にアクセス時に、アプリケーションはトークンを発行しForm要素に埋め込む。さらにセッション変数にトークンを格納する
- ③ クライアントがサイト停止機能を実行しリクエストが送信される。
- ④ アプリケーションは送信されてきたトークンとセッション変数に格納されているトークンが同一かを検証する
- ⑤ アプリケーションがリクエストを受信し、処理を実行する。
- ⑥ 結果を含むレスポンスがクライアント（サイト管理者）へ送信される。

④の処理におけるコードが修正されている

修正版コード

CsrfPreventionFilter.java

```
public class CsrfPreventionFilter extends FilterBase {
    :
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest) request;
    HttpSession session = req.getSession(false);

    @SuppressWarnings("unchecked")
        LruCache<String> nonceCache = (session == null) ? null
            : (LruCache<String>) session.getAttribute(
                Constants.CSRF_NONCE_SESSION_ATTR_NAME);

    if (!skipNonceCheck) {
        String previousNonce =
            req.getParameter(Constants.CSRF_NONCE_REQUEST_PARAM);

        if (nonceCache == null || previousNonce == null ||
            !nonceCache.contains(previousNonce)) {
            res.sendError(denyStatus);
            return;
        }
    }
    : //正常処理
}
```

セッションがnullである
場合、またはリクエ
ストにトークンが含まれ
ていない場合はエラー
として処理する

参考文献

■ OWASP CSRFGuard Project

https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project



■ IPA ISEC セキュア・プログラミング講座： Webアプリケーション編

第4章 セッション対策：リクエスト強要（CSRF）対策

<https://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/301.html>



■ 安全なウェブサイトの作り方、IPA

<https://www.ipa.go.jp/security/vuln/websecurity.html>



著作権・引用や二次利用について

- 本資料の著作権はJPCERT/CCに帰属します。
- 本資料あるいはその一部を引用・転載・再配布する際は、引用元名、資料名および URL の明示をお願いします。

記載例

引用元：一般社団法人JPCERTコーディネーションセンター
Java アプリケーション脆弱性事例解説資料
Apache Tomcat における CSRF 保護メカニズム回避の脆弱性
https://www.jpccert.or.jp/securecoding/2012/No.09_Apache_Tomcat.pdf

- 本資料を引用・転載・再配布をする際は、引用先文書、時期、内容等の情報を、JPCERT コーディネーションセンター広報(office@jpccert.or.jp)までメールにてお知らせください。なお、この連絡により取得した個人情報は、別途定めるJPCERT コーディネーションセンターの「プライバシーポリシー」に則って取り扱います。

本資料の利用方法等に関するお問い合わせ

JPCERTコーディネーションセンター
広報担当
E-mail : office@jpccert.or.jp

本資料の技術的な内容に関するお問い合わせ

JPCERTコーディネーションセンター
セキュアコーディング担当
E-mail : secure-coding@jpccert.or.jp