

Parasoft Jtest 8.3 技術資料

セキュアコーディングのすすめ 2-

前回の「今月のピックアップ」において OWASP TOP 10 や MITRE 2006 の脆弱性の報告件数を基に Web アプリケーションの脆弱性は以前と変わらず増加傾向にある、という事を書きました。

その記事の中にもありましたが、OWASP Top 10 2007 と 2004 を比較すると、クロスサイトスクリプティングとインジェクション系の脆弱性が増加傾向にあります。

表 1 OWASP Top10 / 2007 と 2004 の比較

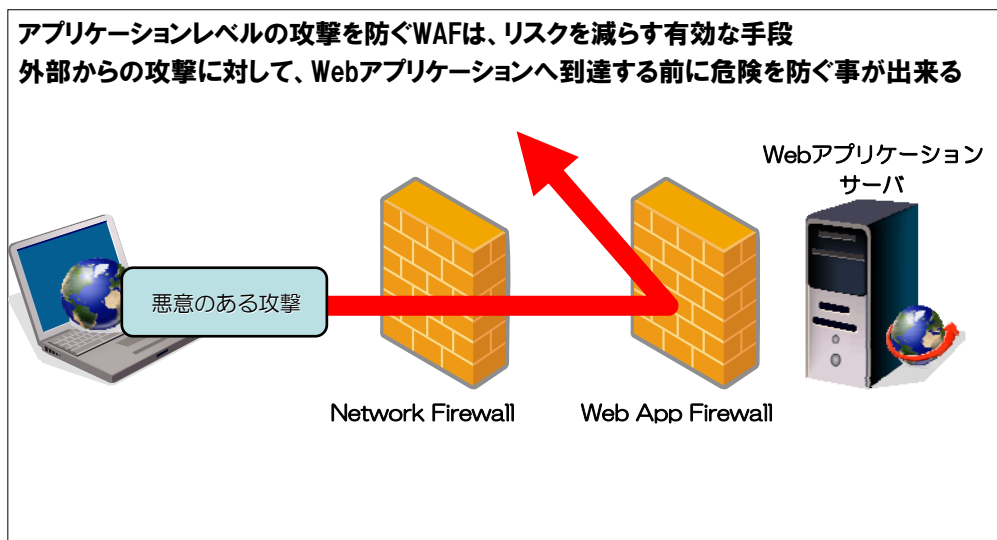
| OWASP Top 10 2007 | OWASP Top 10 2004 | MITRE 2006 |
|---|--|------------|
| A1. Cross Site Scripting (XSS) | A4. Cross Site Scripting (XSS) | 1 |
| A2. Injection Flaws | A6. Injection Flaws | 2 |
| A3. Malicious File Execution (NEW) | | 3 |
| A4. Insecure Direct Object Reference | A2. Broken Access Control (split in 2007 T10) | 5 |
| A5. Cross Site Request Forgery (CSRF) (NEW) | | 36 |
| A6. Information Leakage and Improper Error Handling | A7. Improper Error Handling | 6 |
| A7. Broken Authentication and Session Management | A3. Broken Authentication and Session Management | 14 |
| A8. Insecure Cryptographic Storage | A8. Insecure Storage | 8 |
| A9. Insecure Communications (NEW) | Discussed under A10. Insecure Configuration Management | 8 |
| A10. Failure to Restrict URL Access | A2. Broken Access Control (split in 2007 T10) | 14 |

これらの Web アプリケーションの脆弱性は、Web アプリケーションが一般的に利用されるようになった当初から脆弱性として認識されている問題で、特に目新しいというわけではありません。古くから脆弱性として認識があるにも関わらず、いまだに報告される脆弱性として上位を占めており、さらに増加傾向にあるようです。また、近年では、Cross Site Request Forgery (CSRF) といった比較的目新しい脆弱性も確認されています。

では、こういった Web アプリケーションの脆弱性に対してどのような対応を行う必要があるのでしょうか。OS やアプリケーションサーバのバグによるセキュリティホールを突かれ、攻撃をされる場合は事前に対策を行うことはなかなか難しいかも知れません。

しかし、アプリケーションレベルでの脆弱性を防止する手段は、いくつかの方法が考えられます。例えば、WAF や IDS などが導入といった事も有効な手段です。これらのシステムは開発者がセキュリティ対策へ掛ける負担を軽減し、アプリケーションの安全性を高め、脆弱性を防御する事ができます。

図 1 WAF (Web Application Firewall)



もちろん、これらのシステムが導入されたからといって、100%脆弱性を防御できるという訳ではありません。WAF やIDS などを使ってアプリケーションへ到達する前にリスクを防いでも、アプリケーションそのものが脆弱性を持ったままでは意味がありません。これらのシステムに検知されないような攻撃もあるかも知れませんが、そもそも予算の関係などで導入できない場合もあるかも知れません。

つまり、**アプリケーション自体もセキュアなものである必要があります。**

アプリケーションの脆弱性は、コーディングの段階で（ソースコードレベルで）実装方法に気をつける事によって多くのリスクを減らす事ができます。しかし、実際のプロジェクトの中ではセキュリティを意識したコーディングを行う事がなかなか難しいという話をよく聞きます。

その理由としては以下の様な事が考えられます。

- ・ **開発者の技術レベルにバラつきがある**
プロジェクトに参加している開発者全員がセキュアコーディングに関する知識が有り、技術レベルが高いというのは稀で、多くの場合は開発者ごとに技術レベルにバラつきがあると思います。技術レベルの高い開発者がチーム内にセキュアコーディングのノウハウを浸透させるというのは大変な作業であり、ソースコードレビューやテストなどに多くの時間を割く事になります。
- ・ **セキュリティに関してまでテストしている余裕がない**
納期のあるシステム開発の中で顧客の要求を満たす機能を実装しながら、単体テストや機能テストを行う必要があります。さらにはパフォーマンスに関するテストも考慮しなければなりません。そんな状況で、セキュリティに関するテストも行うとなるとそこにかける工数に余裕がない、という場合もあります。また、テストに漏れがある場合も考えられます。

それではどうすればセキュアなコーディングが実現できるのでしょうか。
これらの問題に対処してセキュアなコーディングを行うためにはツールの利用が効果的です。

ツールを使う事で実装したコードがセキュアなものかを検証できます。ツールによってソースの検証を自動化する事により、上記に挙げた問題である技術レベルのバラつきや、テスト漏れ、考慮漏れといった部分を減らし、一定のレベルまで引き上げることができるようになります。

また、一番の問題である工数もツールによる自動化を行うことで大幅に削減することが可能です。ツールによる検証もまた、セキュリティ上の脆弱性が 100%防御できるというわけではありませんが、脆弱性を防御する上で守らなければいけない社内ルールの準拠の検証や、脆弱性の検証を行うことにより、品質を確保できます。

このような検証を行うために Parasoft Jtest は有効なツールであると言えます。

Jtest8.3 では以前は有償であった静的解析におけるセキュリティルールが無償で付属する様になりました。それによって、脆弱性対策としてソースコードに対して以下の検証を行う事ができます。

- ・ バグ探偵を使って実行パスから実際に発生する様な脆弱性の検出
- ・ 静的解析から脆弱性に繋がりがやすい、バグとなりやすい実装の検出

また、アプリケーションの品質を確保する上で欠かすことのできない単体テストのテストケースを自動生成することで、高い堅牢性も確保する事ができます。

Jtest を使ったセキュアコーディング

前回の「今月のピックアップ」ではフローベースの静的解析(バグ探偵)をご紹介しましたが、Jtest が持っているパターン マッチングによる静的解析(コーディングスタンダードルール)の検出例を見てみましょう。

Jtest8.3 ではセキュリティに関する静的解析のルールとしては以下の様なカテゴリのルールが用意されています。(License Required と書かれているものが 8.3 から無償となったルール)

- ・ **セキュリティ (10 ルール)**
アプリケーションにセキュリティの脆弱性をもたらす可能性があるコーディング構造を特定するコア ルール
- ・ **セキュリティ (License Required) (23 ルール)**
セキュリティの脆弱性をアプリケーションにもたらす可能性があるコード構造を特定する上級ルール
- ・ **セキュリティポリシー ルール (License Required) (13 ルール)**
セキュリティ ポリシーに従ってカスタマイズできるルール
- ・ **Web セキュリティ (License Required) (12 ルール)**
Web コンポーネントに関連するセキュリティの脆弱性を特定するルール

セキュリティに特化しているわけではありませんが、Web アプリケーションに対しては以下の様なルールも有効に活用することができます。

- ・ **サーブレット (9 ルール)**
サーブレット、JavaServer ファイル、および JDBC 接続を持つ Web アプリケーション開発のベスト プラクティスを推進するためのルール
- ・ **Struts フレームワーク (6 ルール)**
Struts フレームワークに特化したルール

また、Web アプリケーション用のルールというわけではありませんが様々な問題に対応するルールも用意されています。

- ・ スレッドと同期化 (28 ルール)
スレッドと同期化に関連するルール
- ・ 発生するかもしれないバグ (103 ルール)
コード中のバグの可能性をチェックするルール

など等、800 以上のルールを搭載しています。

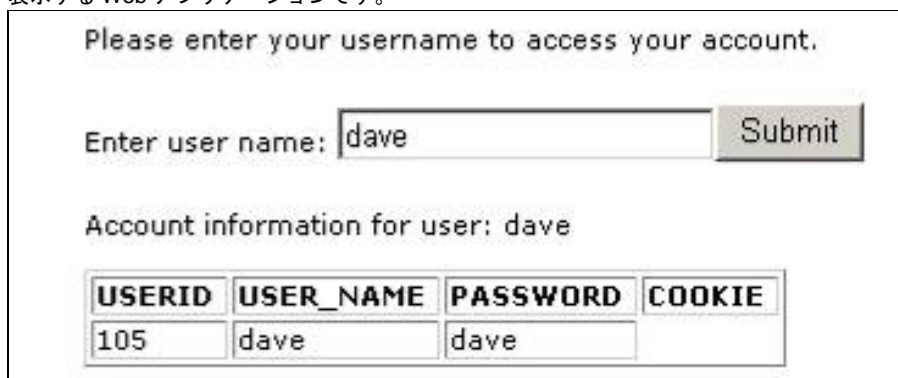
WebGoat Example プロジェクト

今回も WebGoat プロジェクトを使って Jtest の静的解析における脆弱性の検出例を見ていくことにします。(WebGoat は OWASP が提供しているセキュリティ学習用の Web アプリケーションであり、Jtest8.3 ではサンプルプロジェクトとして簡単に取り込む事ができます。)

スレッドの安全性の例

アプリケーションの説明

スレッドの安全性に関するサンプルを動かしてみます。
このサンプルは入力されたユーザ名をキーにデータベースを検索し、パスワードなどのユーザ情報を表示する Web アプリケーションです。



| USERID | USER_NAME | PASSWORD | COOKIE |
|--------|-----------|----------|--------|
| 105 | dave | dave | |

図 2 WebGoat Concurrency - Thread Safety Problems

脆弱性の確認

このアプリケーションにはスレッドの安全性に関して脆弱性があるので、それを確認するために同時に 2 ユーザが同時にログインを試みます。

2人で別のマシンからテストを実行するか、ひとつの端末でブラウザを 2つ立ち上げて、ひとつは Enter user name に jeff を、もうひとつは dave を入れて同時に Submit ボタンを押下します。(ひとつの端末から 2つブラウザを立ち上げている場合はなるべく間隔を開けずに Submit ボタンを押下する)

Please enter your username to access your account.

Enter user name:

Account information for user: dave

| USERID | USER_NAME | PASSWORD | COOKIE |
|--------|-----------|----------|--------|
| 105 | dave | dave | |

図 3 ブラウザ 1

*** Congratulations. You have successfully completed this lesson**

Enter user name:

Account information for user: jeff

| USERID | USER_NAME | PASSWORD | COOKIE |
|--------|-----------|----------|--------|
| 105 | dave | dave | |

図 4 ブラウザ 2

成功すると図 4 ブラウザ 2 の様に赤字で成功の旨を伝えるメッセージが表示されます。タイミングが合わないと成功しないので、赤字でメッセージが表示されない場合は何回か試してみてください。

削除: 図 4 ブラウザ 2

図 3 ブラウザ 1 では dave というユーザ名でログインし、dave のユーザ情報が表示されています。図 4 ブラウザ 2 では jeff というユーザ名でログインし、jeff のユーザ情報が表示されるはずなのですが、dave のユーザ情報が表示されてしまいました。

削除: 図 3 ブラウザ 1

削除: 図 4 ブラウザ 2

Jtest を使用する

このアプリケーションはどこに脆弱性があったのでしょうか。

このアプリケーションに対して Jtest の静的解析を実行してみたいと思います。ここで問題になるソースに対して Security Assessment テストコンフィギュレーションを使用してテストを行います。

[対象ファイル]

WebGoat-5.1 > Java Source > org.owasp.webgoat.lessons > ThreadSafetyProblem.java

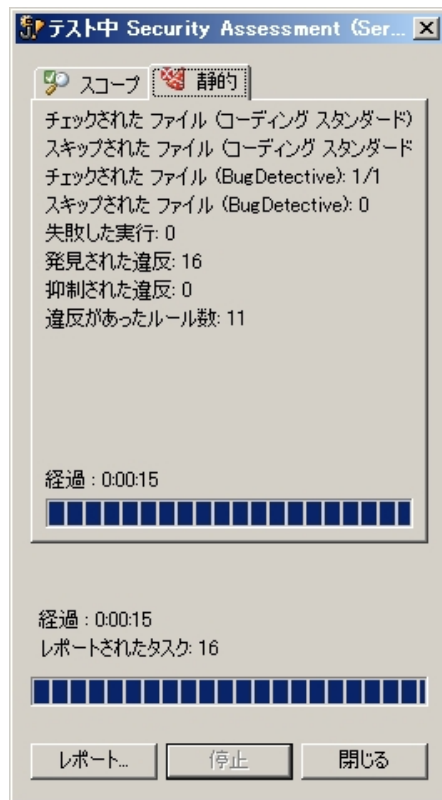


図 5 Security Assessment テストコンフィギュレーションによるテスト

結果を確認する

いくつか違反が検出されますが、その中でもスレッドと同期化の違反に注目します。

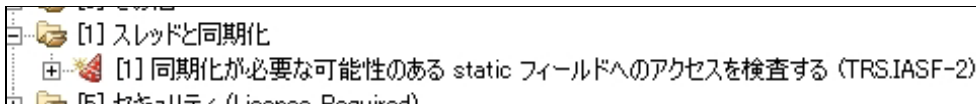


図 6 Jtest スレッドと同期化の違反

さらにドリルダウンすると、違反箇所が表示されます。

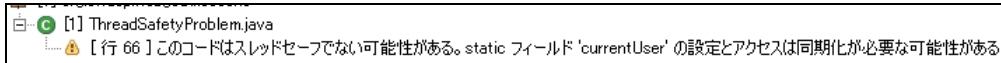


図 7 Jtest 違反の発生箇所

図 7 Jtest 違反の発生箇所をみると、このコードがスレッドセーフではなく、static フィールドに同期化が必要な可能性がある、という旨のメッセージが表示されています。

削除: 図 7 Jtest 違反の発生箇所

ソースファイルの該当箇所を確認します。

```
ThreadSafetyProblem.java x
42     new IMG("images/logos/aspect.jpg").setAlt("Aspect Security").setBorder(
43         0).setHspace(0).setVspace(0);
44
45     private final static String USER_NAME = "username";
46
47     private static String currentUser;
48
49     private String originalUser;
50
51     /**
52     * Description of the Method
53     *
54     * @param s
55     *     Description of the Parameter
56     * @return Description of the Return Value
57     */
58     protected Element createContent(WebSession s) {
59         ElementContainer ec = new ElementContainer();
60
61         try {
62             Connection connection = DatabaseUtilities.getConnection(s);
63
64             ec.addElement(new StringElement("Enter user name: "));
65             ec.addElement(new Input(Input.TEXT, USER_NAME, ""));
66             currentUser = s.getParser().getRawParameter(USER_NAME, "");
67             originalUser = currentUser;
68
69             // Store the user name
70             String user1 = new String(currentUser);
71
```

図 8 検出箇所のソース

図 8 検出箇所のソースを確認すると、赤い四角で囲まれている部分が検出箇所です。クラスフィールドである currentUser に値が設定されている箇所です。違反が検出されています。さらに上の青い四角で囲まれている箇所が currentUser の宣言箇所です。currentUser は static フィールドですが、final で宣言されていない、ということが今回の指摘です。

削除: 図 8 検出箇所のソース

つまり、static 変数のインスタンスはひとつしかないため、すべてのアクセスを同期化するのでない限り、final ではない static 変数は同時アクセスがあった場合に予期しない変数の書き換えが発生する可能性があります、ということを指摘しています。

Web アプリケーションのようなマルチスレッド環境では、スレッドが完了する前に別のスレッドが値を変更してしまう場合があるということです。そのため、脆弱性の確認を行ったときに 2 人のユーザが同時にアクセスすると、タイミングによってはスレッド上にひとつしかない currentUser のインスタンスにアクセスしたため、別のユーザ情報が表示されてしまったわけです。

この脆弱性は、今回の例のように誤った情報を第三者に公開してしまったり、計算結果が合わないなどの問題が発生します。そして、特定のタイミングでしか問題が発生しない場合が多いため、調査や追跡が困難な問題が発生させます。同期化の問題はインスタンスと Web アプリケーションのスレッドの関係を理解していない様な未熟な開発者に良く見られます。

いかがでしょうか。このように、Jtest を使ってソースをチェック自動化することで一定レベルの品質を保つことができるということをご理解頂けたでしょうか。今回紹介した内容は、Jtest チュートリアルにも記載されています。

Jtest8.3 のセキュリティ機能に関しては無償の Jtest セキュアコーディングセミナーを定期的に開催しております。興味がある場合はホームページにて日程を確認の上、お申し込みください。

また、無償のセキュアコードチェック評価支援も準備しておりますので、同様に興味がある場合はぜひお問い合わせください。

【お問い合わせ先】

TechMatrix

テクマトリックス(株)

システムエンジニアリング事業部

ソフトウェアエンジニアリング営業部

ソフトウェアエンジニアリング営業課

TEL 03-5792-8606 FAX 03-5792-8706

E-MAIL parasoft-info@techmatrix.co.jp

URL <http://www.techmatrix.co.jp/products/quality/jtest/>