社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－CSEC－41（17）
2008／5／23

Webアプリケーション脆弱性発見のための ソースコード診断ツールの開発

小黒 博昭　　市原 尚久　　道坂 修
株式会社NTTデータ 技術開発本部 SIアーキテクチャ開発センタ

あらまし：Webアプリケーションのセキュリティ上の脆弱性を発見するための手法の1つにソースコード診断ツールの適用がある．しかし，従来のソースコード診断ツールは，Webアプリケーションのセキュリティに特化したものは少なく，ほとんどはパターンマッチングまたは構文解析を応用した手法を利用しており，高精度で検出可能なセキュリティ上の脆弱性はSQLインジェクションのような主要な脆弱性に限定されるという問題があった．特に，レースコンディション脆弱性については，ブラックボックス試験では検出が困難であり，その高精度な検出は重要な課題であった．我々はJava言語で実装されたWebアプリケーションのセキュリティ上の脆弱性を検出するツールを開発した．特に，レースコンディション脆弱性の検出に対しては，バイトコード解析と独自のオペランドスタックを導入することにより，精度の高い検出を実現した．

Development of a Source Code Diagnosis Tool for Detection of Web Application Vulnerability

Hiroaki Oguro　　Naohisa Ichihara　　Osamu Dousaka
SI Architecture Center, Research and Development Headquarters, NTT DATA Corporation

Abstract: The application of source code diagnosis tools is one of the techniques to detect Web application vulnerability with respect to the security. However, there are few conventional tools specialized to the security of Web applications. Since most of the tools use the technique of either a pattern matching or a syntax analysis, there is a problem that vulnerability which the tools can detect with high precision is limited to major vulnerability such as SQL injection. Especially, detecting race condition vulnerability with high precision is important because it is difficult to detect it in black-box tests. We have developed a tool which detects vulnerability with respect to the security of Web application implemented with Java language. As for the detection of the race condition vulnerability, we have realized it with high precision by introducing a bytecode analysis and an original operand stack.

1　はじめに

近年，Webアプリケーションのセキュリティ上の脆弱性に起因したインシデントの発生が後を絶たない．Webサイトを狙ったSQLインジェクション攻撃は，最近再び増加傾向を見せており，Webサイトの改ざんや非公開情報の漏洩などの深刻な被害が多数報告されている．

Webアプリケーションのセキュリティ上の脆弱性を検出することを支援するツールとして，動作中のWebアプリケーションに擬似的な攻撃を仕掛けて脆弱性を発見するブラックボックス試験ツールと，ソースコードを検査するホワイトボックス試験ツールがある．システム開発における手戻りの防止という観点からは，バグが早期に発見されることが望ましく，製造工程においてホワイトボックス試験ツール（ソースコード診断ツール）を適用することの有用性が注目されている．

我々は，Java[1]言語で実装されたWebアプリケーションのセキュリティ脆弱性を網羅的に検出するツール（以下，本ツールと呼ぶ）のプロトタイプを開発した．特に，レースコンディション脆弱性の検出に対しては，バイトコード解析を用いて共有オブジェクトの状態（ロックまたはアンロック）を判定する機構を独自に導入することにより，精度の高い検出を実現した．

本稿では，本ツールの設計およびプロトタイプ実装に関する概要と，主にレースコンディション脆弱性の診断を中心とした評価結果について報告する．

本稿の以後の構成は以下の通りである．2章で，

[1] Java および Java 関連の商標は，米国およびその他の国における米国 Sun Microsystems, Inc. の商標または登録商標です．

Web アプリケーションのセキュリティ上の脆弱性を診断するツールの要件を定義する．3 章でその設計と実装を説明する．4 章では，本ツールに対して実際に診断を行った結果を示し，その評価を行う．5 章で，まとめと今後の課題を述べる．

2 診断ツールの要件

本節では，Web アプリケーションのセキュリティ上の脆弱性を診断するツールの要件を定義する．

2.1 診断対象の Web アプリケーション

Web アプリケーションのセキュリティ上の脆弱性を診断する際，クライアント側アプリケーションのソースコードのみでは脆弱性の有無を判断できず，アプリケーションサーバの設定ファイル等の情報が必要となることがある．従って，本ツールにおいて診断対象とする Web アプリケーションを，アプリケーションサーバ Tomcat[1]，WebLogic[2]，および WebSphere[3] 上で動作する J2EE 準拠の Java アプリケーションおよびそのソースコードを含むものと定義する．

2.2 診断対象の脆弱性

診断対象とする脆弱性については，文献[4]を参考に決定するものとし，少なくとも以下の全ての脆弱性に関連する検査項目が存在することとする．

- セッション管理
- 認証処理
- クロスサイトスクリプティング
- コマンドインジェクション
- SQL インジェクション
- 入力チェック
- パスワード管理
- レースコンディション

2.3 確度

本ツールは IT システムに存在しうるセキュリティ上の脆弱性を発見することを目的として利用されるため，診断ミスが与える影響は重大である．そのため，False Negative[4]は極力避けなければならない．一方，そのために False Positive[5] の誤検出が多少増えることについては容認されるべきと考えられる．しかし，False Positive があまりにも多い場合，それらの中から本当の脆弱性を探すことに手間を要するだけでなく，項目の多さに起因する見過ごしといったヒューマンエラーを引き起こす可能性がある．それらの問題を防止するため，検出項目に対して，それが脆弱性であることの確度を表すパラメータが付与されることが望ましい．

本ツールでは，全ての検出項目に対し 3 段階の確度を設定する．確度 1 は，脆弱性ではないかもしれないが念のために警告しておく程度の検出であり，False Positive である可能性が高い．確度 3 は脆弱性と断定できる検出，またはその可能性が高い検出に付与される．

2.4 利用者に対する受容性

本ツールが利用者に抵抗なく受け入れられるようにするためには，利用シナリオを意識する必要がある．本ツールにおいて想定される 2 つの利用シナリオを以下に示す．

A) プログラマが，コーディング作業中に本ツールをデバッグツールのように頻繁に利用する．
B) セキュリティ診断サービス提供者が，顧客のソースコードを診断し，結果を報告書にまとめて顧客へ提出する．

上記 A) が円滑に行われるためには，開発環境と診断ツールとの親和性が重要であると考えられる．上記 B) の場合は，診断結果の総合的な分析や，報告書の作成等を支援する機能が要求されると考えられる．

本ツールでは，将来的には上記両方のシナリオを満足することを目標とするが，初期リリース版においては上記 A)の意識を優先し，開発者に受け入れられるツールを目指すこととする．

2.5 拡張性

既存の診断ルールを更新する際や，新規の検出ロジック（脆弱性を検出するためのベースとなる解析方法）を追加する際は，ツールの追加開発規模が極力小さくなることが望ましい．

本ツールでは，診断ルールはツール外部の設定ファイルとして実現し，診断ルールを更新する際は設定ファイルを更新するだけで済むようにする．さらに，本ツールの機能構成は，新規の検出ロジックの追加が容易となるようなアーキテクチャとなっていることとする．

3 設計と実装

本節では，本ツールの全体的な機能構成，および構成される 3 つの機能の概要を述べる．

3.1 全体的な機能構成

本ツールの全体的な機能構成を図 1 に示す．本ツールは，オープンソースの Java 統合開発環境 Eclipse のプラグインとして実装され，以下の 3 つの機能から構成される．

- GUI 機能
- セキュリティ診断機能
- 拡張セキュリティ診断機能

以降，3.2節から3.4節において，各機能の概要を説明する．

---

[1] Tomcat は，Apache Software Foundation の商標または登録商標です．
[2] WebLogic は，BEA Systems, Inc. の登録商標です．
[3] WebSphere は，米国における米国 International Business Machines Corp. の登録商標です．
[4] 脆弱性の部分を誤って脆弱性ではないと判断すること．
[5] 脆弱性ではない部分を誤って脆弱性として検出すること．

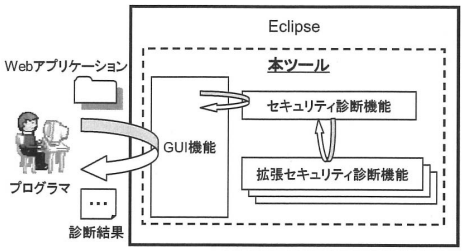

図 1：本ツールの機能構成

3.2 GUI 機能

本ツールは Eclipse のプラグインとして実装されるため，全ての GUI は Eclipse が提供する GUI をベースとし，Eclipse と一体感のあるツールとなるように設計した．本ツールの GUI 機能は以下の 3 つのサブ機能から構成される．

- **設定** GUI
  本ツールの設定を管理する GUI を提供する．Eclipse の各種設定を行うプリファレンスストアを利用する．
- **操作** GUI
  セキュリティ診断を実行するウィザードの起動，診断対象となる Web アプリケーションの選択，診断の進捗を表示するプログレスモニタの呼び出し，および診断結果を CSV 形式でクリップボードへコピーする操作の選択等の GUI を提供する．
- **結果出力** GUI
  セキュリティ診断の結果をツリー形式で表示し，診断項目のタイトルをダブルクリックすると診断概要や推奨対応等の情報を表示し，脆弱性検出項目をダブルクリックするとエディタ上の該当行を表示する GUI を提供する．

3.3 セキュリティ診断機能

セキュリティ診断機能は，キーワード検出やクラス検出といった比較的単純な複数の検出ロジックに基づいた検出条件を組み合わせて脆弱性を診断する．SQL インジェクションやクロスサイトスクリプティング等の脆弱性の診断は本機能で行われる．1 つの診断項目に対して，複数の検出条件の組み合わせが可能である．組み合わせの定義は，3.3.3節で示すセキュリティ診断チェックリストで与える．

以下の各小節で，キーワード検出ロジック，クラス検出ロジック，およびセキュリティ診断チェックリストの概要を述べる．

3.3.1 キーワード検出ロジック

キーワード検出ロジックは，ソースコードファイル中から指定されたキーワードを行内の開始列と終了列の粒度で検出する．例えば，SQL インジェクション脆弱性の診断では，SQL 文を実行するメソッド executeQuery がキーワードに指定されることで，1 つの検出条件が構成される．

表 1：セキュリティ診断チェックリストの XML 構造

タグ名	概要	内部で定義可能なタグ
securityCheckList	診断チェックリスト全体のタグ	securityCheck
securityCheck	診断項目を示すタグ（3階層まで診断項目の定義が可能）	必須タグ＝id,name 定義可能タグ＝ summary, measures, danger, keywordList, logic
id	診断項目のID	-
name	診断項目の名称	-
summary	診断項目の概要	-
measures	診断項目の推奨対応	-
danger	診断項目の脅威	-
keywordList	診断項目の検出ロジックの定義	-
logic	検出に用いる検出ロジックモジュールの名称	必須タグ＝ logicName,moduleName, pathType, extension, parameter 定義可能タグ＝ logic,relation, accuracy
logicName	検出ロジックの名称	-
moduleName	検出ロジックのクラス名称	-
pathType	検出ロジックに渡されるパスの種類	-
extension	対象となるファイルの拡張子（カンマ区切りで複数指定可能）	-
parameter	検出ロジックに用いるパラメータ	-
relation	上位の検出ロジックとの関係 0：最上位の検出ロジック 1：上位検出ロジックとOR関係 2：上位検出ロジックとAND関係 3：上位検出ロジックとNOT関係	未定義時は0
accuracy	確度	未定義時は上位ロジックのaccuracyを適用

3.3.2 クラス検出ロジック

クラス検出ロジックは，クラスファイルを解析し，検出対象のパッケージ，クラス（インターフェース），およびメソッドの各バイトコード位置をリスト化し，ソースコード上の行数と対応付けて出力する．クラスファイルの解析にはバイトコード解析ライブラリの 1 つである BCEL [1] を利用する．

3.3.3 セキュリティ診断チェックリスト

セキュリティ診断チェックリストは，本ツールの診断ルールを規定する外部 XML ファイルである．セキュリティ診断チェックリストの XML 構造を表 1 に示す．

セキュリティ診断チェックリストでは，単一の検出ロジックでセキュリティ脆弱性を検出できない場合を想定し，複数の検出条件を指定可能とすることで複雑なセキュリティ脆弱性の検出に対応できるようにしている．複数の検出条件の指定とは，2 つの検出条件を組み合わせる際に各々の検出ロジックの関係を定義し，その関係によって診断結果の取捨選択を行うことを意

味する．2 つの検出ロジックにはそれぞれ上位検出ロジックおよび下位検出ロジックが決定され，その関係を表すために本ツール独自で定義した記号 OR，AND，および NOT を用いる．

以下に OR，AND，および NOT の関係の定義を示す．上位検出ロジックを L_A，下位検出ロジックを L_B で表し，それぞれの結果であるソースコード位置情報を R_A および R_B とする．

- **OR 関係**
  上位ロジックの検出結果に下位ロジックの検出結果を追加する．
  （例）L_A OR L_B は，R_A および R_B の 2 つのソースコード位置情報を出力する．
- **AND 関係**
  上位ロジックと下位ロジックの検出結果であるソースコード位置情報に重複がある場合，上位ロジックの検出結果を出力する．重複がない場合は出力しない．
  （例）L_A AND L_B は，R_A と R_B に重複がある場合は R_A を出力する．重複がない場合は出力しない．
- **NOT 関係**
  上位ロジックと下位ロジックの検出結果であるソースコード位置情報に重複がある場合は出力しない．重複がない場合は上位ロジックの検出結果を出力する．
  （例）L_A NOT L_B は，R_A と R_B に重複がある場合は出力しない．重複がない場合は R_A を出力する．

ここで，ソースコード位置情報の重複とは，それぞれの検出結果であるソースコード位置情報（ファイル名，開始行，終了行，開始列，および終了列）に重なる部分が存在することを意味する．例えば，R_A がファイル test.java の 18 行目から 20 行目であり，R_B が test.java の 20 行目から 25 行目であるとき，R_A と R_B は test.java の 20 行目で重複していると判断する．

3.4 拡張セキュリティ診断機能

拡張セキュリティ診断機能は，3.3 節で示したセキュリティ診断機能のみでは検出が本質的に困難であるような脆弱性を対象として，本ツールを拡張開発する際に，独自の検出ロジックを容易に追加できるようにした枠組みの機能である．例えば，レースコンディション脆弱性の診断は本機能の 1 つとして実現される．

3.4.1 レースコンディション

レースコンディション（競合状態）とは，共有資源に対し同時に複数のアクセスが行われた際，排他制御が正しく機能していない状態を指す．レースコンディションが起きると，システムに予期せぬ動作を引き起こすことがある．

Web アプリケーションは，不特定多数の利用者からアクセスされるだけでなく，取り扱うデータに個人情報等の貴重なデータが含まれることが多いため，レースコンディションが原因で非公開情報が漏洩するといったインシデントに発展する可能性がある．例えば，図 2 に示すように，利用者 A と利用者 B が同時に Web サーバへログインを試みるとき，レースコンディションの対策が行われていない場合は，利用者 A の画面に利用者 B の情報が表示されるといったことが起こりうる．

Java 言語には，排他制御を実現する仕組みとして synchronized 修飾子がある．排他制御をかけたいメソッドまたはブロックに synchronized を記述すればレースコンディションは発生しない．しかし，過剰な（不要な）synchronized の記述はアプリケーションの性能を低下させる．一方，本来必要な部分への synchronized の記述漏れは競合状態発生の可能性を残す．従って，プログラマには必要最小限の synchronized の記述が要求される．

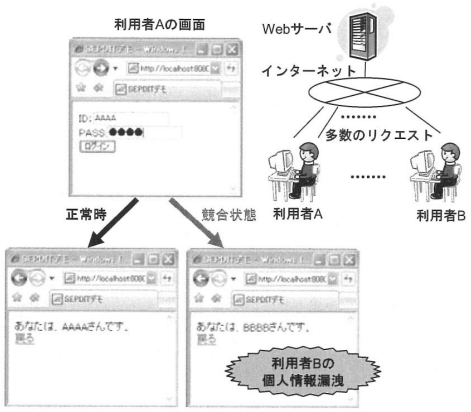

図 2：レースコンディションの発生例

3.4.2 レースコンディション検出ロジック

本ツールのレースコンディション検出ロジックで採用したレースコンディション判定のアイディアを図 3 に示す．図 3 において，2 つのスレッドが 1 つのアンロック状態の共有変数に対して同時に読み書きする機会があることと，レースコンディションが発生する可能生があることは等価である（これを命題 A とする）．このとき明らかに，あるスレッドにおいてアンロック状態の共有変数を読み書きする処理フローが存在する（これを命題 B とする）．従って A→B が成立する．この逆 B→A は一般には成り立たない．反例は，マルチスレッドの状態にならないようなリクエスト系列である場合，すなわち同時アクセスが起こらない場合である．しかし，Web アプリケーションのサービス特性を考慮すると，このようなリクエスト系列がサービス期間を通して継続することは非現実的である．従って，このような場合は起こらないという前提の下で，B→A が成立し，このとき A と B が等価であると見なすことできる．本ツールでは，B の成否を検査することにより，A の成否の情報を得る．

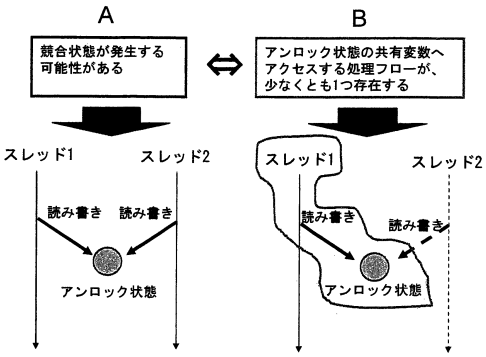

図 3：レースコンディション判定のアイディア

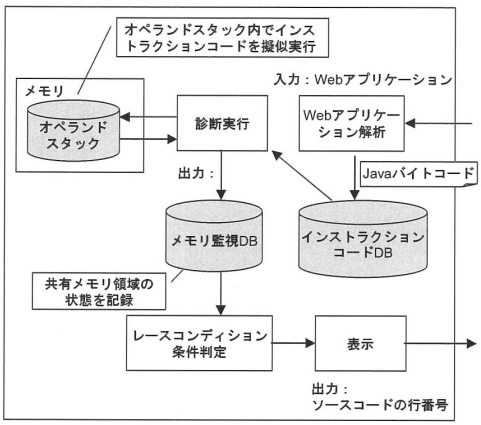

図 4：レースコンディション検出ロジックの概要

上記のアイディアを基にしたレースコンディション検出ロジックの概要を図 4 に示す．レースコンディション検出ロジックは，Web アプリケーションを入力し，そのクラスファイルに対してバイトコード解析を行い，利用者からのリクエストを契機として開始する処理フローを全て抽出する．それらの処理フローを独自に用意したオペランドスタック（これは軽量な Java VM としての役割を持つ）上でシミュレートし，共有オブジェクトに対するインストラクションコード，バイトコード位置，およびその時の共有オブジェクトの状態（ロックまたはアンロック）をレコードとしてメモリ監視 DB へ追記保存していく．その記録内において，共有オブジェクトの状態がアンロックであり，かつ，読み書き系のインストラクションコード（getstatic，putstatic，getfield，putfield のいずれか）が実行されている場合をレースコンディションが発生しうると判断する．このとき，該当する共有オブジェクトのバイトコード位置からソースコード上の行番号を求め，GUI 機能へ返却する事により，Eclipse のエディタ上の該当部分の行頭に警告マーカーが表示される．

4 評価

本節では，本ツールを用いて実際にセキュリティ上の脆弱性を持つ Web アプリケーションのソースコードを診断し，脆弱性として検出された結果と，脆弱性が修正されたソースコードを診断した結果の比較を中心として評価を行う．

4.1 レースコンディション脆弱性の検出

実際にレースコンディションを発生させる試験プログラムを作成し，その動作を負荷テストツール JMeter[2]を用いて測定した結果を表 2 に示す．

表 2：試験対象プログラムの動作実験

クライアント	リクエスト数	synchronized 付加前		synchronized 付加後	
		競合状態発生数	競合状態発生率(%)	競合状態発生数	競合状態発生率(%)
A	1015	2	0.197	0	0.000
B	1014	1	0.099	0	0.000
C	1012	2	0.198	0	0.000
D	1011	5	0.495	0	0.000
E	1011	3	0.297	0	0.000
F	1011	3	0.297	0	0.000
合計	6074	16	0.263	0	0.000

試験プログラムは，6 人の利用者（A から F）を想定しており，各利用者が順番にログインを試みることを高速に繰り返す．synchronized の記述がないため，レースコンディションが発生しているが，その発生率は平均 0.26%弱であり極めて低い．再現性もないことから，ブラックボックス試験での発見が難しいことがよくわかる結果が得られた．

試験プログラムに対し，本ツールを用いて診断を行った結果の画面を図 5 に示す．その結果，レースコンディションを発生させる共有変数 user に対する synchronized 修飾子の記述漏れが指摘された．

次に，共有変数 user が現れる範囲に synchronized ブロックを付加し，再び本ツールで診断を行ったところ，該当する警告が表示されなくなったことを確認した．このとき JMeter を用いて再び試験プログラムの動作を測定したところ，表 2 に示すように，確かにレースコンディションは発生しないことが確認された．

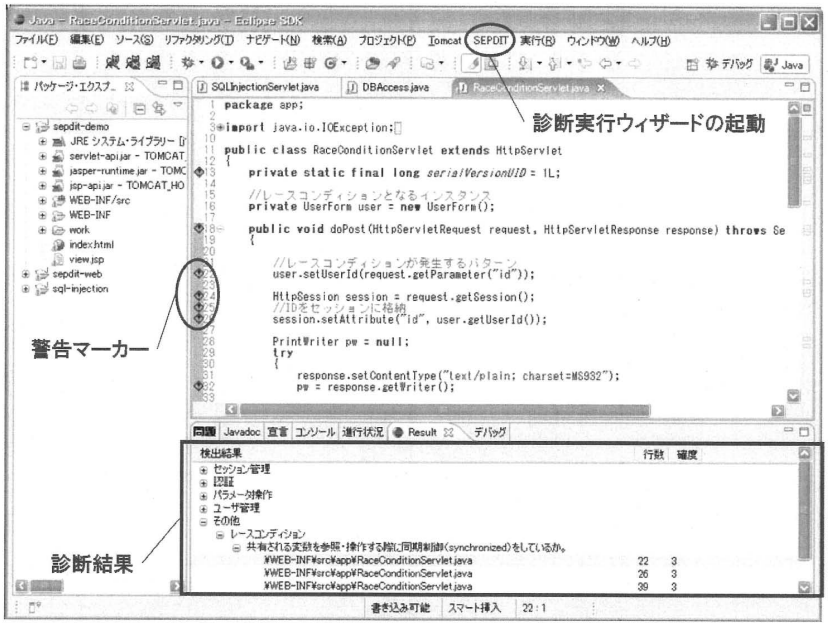

図 5：本ツールの診断実行画面（レースコンディション脆弱性の検出）

4.2 SQL インジェクション脆弱性の検出

実際に SQL インジェクションを発生させる試験プログラムを作成し，本ツールで診断を行った．その結果，SQL 文をデータベースに送信するメソッド executeQuery の部分に警告が表示されることを確認した．さらに，SQL 文の組み立て時に文字列をチェックする対策を施したプログラムに対して診断を行ったところ，該当箇所の警告が表示されなくなることを確認した．

また，診断結果に表示される診断項目のタイトルをダブルクリックすることにより，図 6 に示す詳細情報が表示されることを確認した．

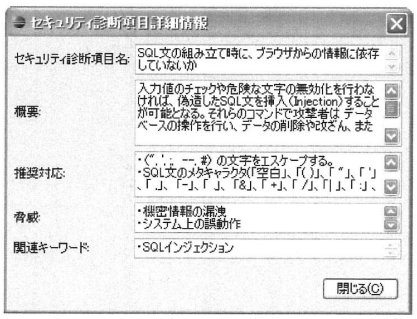

図 6：診断項目詳細情報の表示
（SQL インジェクションに関する診断項目）

5 まとめ

本項では，Java 言語で実装された Web アプリケーションのセキュリティ脆弱性を網羅的に検出するツールの開発状況を報告した．本ツールでは，レースコンディション脆弱性の検出に対して，バイトコード解析を用いた精度の高い検出を実現した．

今後の課題として，診断対象とする脆弱性のカバー範囲を拡大することと，検出の確度を高めるための診断ルールおよび検出ロジックの改良が挙げられる．さらに，ツールの維持管理面を考慮すると，セキュリティ診断チェックリストをネットワーク経由で更新する機能や，セキュリティ診断チェックリストの編集を支援するツール等も必要になると考えられる．

参考文献

[1] Apache Software Foundation: Byte Code Engineering Library (BCEL). http://jakarta.apache.org/bcel/

[2] Apache Software Foundation: Apache JMeter. http://jakarta.apache.org/jmeter/

[3] 大浜伸之，宮崎博: データフロー解析に基づくソースコード脆弱性検証手法，コンピュータセキュリティシンポジウム 2007 (CSS2007) 論文集, pp.423-428, 2007.

[4] 情報処理推進機構（IPA）：セキュア・プログラミング講座, 2007. http://www.ipa.go.jp/security/awareness/vendor/programmingv2/index.html