# 平成２２年度　成果報告書

「静的及び動的解析の組み合わせによるWebアプリケーションのセキュリティ診断システムに関する研究開発」

目　次

## １　研究開発課題の背景

ここ数年の動向を見ている限りインターネットを利用するサービスの大半は何らかのアプリケーションとしての仕組みを保有し、小規模な Web サービスにおいてもオープンソースのブログシステムや CMS(Content Management System)を背後に持つものが多くなっている。Web アプリケーションの利用が一般化するにつれ、アプリケーションの欠陥が原因とみられるセキュリティインシデントの発生も大幅に増加しつつある。

特に個人情報の奪取を狙った攻撃や Web サイトが持つ脆弱性を利用した二次的被害も拡大傾向にある。個人情報やカード情報等の漏洩事件が後を絶たず、漏洩件数が数十万件に及ぶ事も珍しくはない。情報集約、蓄積の仕組みが発達することにより、セキュリティに関わるリスクは比例して大きくなる。当然ながらリスク相応の管理がなされている環境では大きな問題は発生しづらいと言える。大きな被害が報告されている事件、事故が続発している現状では根本的な問題解決がどのユーザーレベルにおいても期待されていることは言うまでも無い。

このように一瞬にして大きな事件、事故を引き起こす可能性がある Web アプリケーションのセキュリティは大きな問題となっており、診断技術も年々進歩している。従来の診断技術は、既知の脆弱性を総当たりで試し、問題の存在有無を判定する擬似攻撃診断が主であった。近年ソースコードを解析することで問題指摘が可能な技術も大幅に進歩している。擬似攻撃診断はブラックボックス試験であり、Web アプリケーションのセキュリティ診断においては、アプリケーションの表層から不正な変数を流し込むなどして攻撃を仕掛け、その反応を解析し問題の有無判定を行う。よって問題が有る、無いといった判定は可能であるが、当該問題がどこでどのように発生しているのかは情報が得られない。結果として問題の特定は開発者側で改めてソースコードを確認することで判断せざるを得ない状況であった。

静的解析によるセキュリティ診断技術は上記擬似攻撃診断に比べて多くの優位点がある。擬似攻撃診断はその特性上アプリケーションが稼働状態にならなければ診断が実施出来ないが、ソースコードを解析する場合、アプリケーションが動く前の状態でも診断は実施可能となる。セキュリティに限らずソフトウェアの問題全般において、早期発見が大幅なコスト削減につながることは事実であり、問題の検知、改修に関わるコスト効率の面では擬似攻撃診断に比べると有用である。また前述の問題箇所を特定することも、ソースコードレベルで詳細に報告することが可能であり、早期に容易に修正まで可能な技術としては利用者の立場からの優位性は高い。

しかしながらプログラムはデータを伴って実行されるものであり、静的解析では純粋にソースコードのみで解析、問題の判定を行っている為、システムやソフトウェアの種類によっては診断の品質が大きく変動する。よって常に一定品質での診断精度を求める場合、相応のノウハウを必要とする。現在販売されている診断ソフトウェアに単純にソースを流し込むだけでは、相当数の誤検知が発生することとなる。その結果を判断する為に相当の時間を要しては本末転倒であり、誰もが簡単に利用出来るという品質には至っていないと思われる。

本研究開発は、これまでに当社が蓄積してきた擬似攻撃診断の技術をベースに、当該技術の弱い部分をソースコードの解析により補完することを主目標として実施した。前述の通りセキュリティ診断の主技術として応用されてきた擬似攻撃診断は、当該試験方式がブラックボックスであるが故に問題がどのように発生しているのかが特定出来ず、これにより問題の改修段階では全ての問題をソースレベルで再確認しなければならない。本研究開発の成果物としては、擬似攻撃診断実施時に問題が検知された場合に、当該問題が発生した制御フローを特定する事が可能で、またその応用として診断が実施された際のソースコード網羅率を評価可能にする技術及びこれら技術の応用製品があげられる。擬似攻撃診断

とソースコード解析技術を組み合わせることで、擬似攻撃診断を補完する技術構成を可能とし、従来とは異なるレベルでの結果報告及び問題指摘を可能とするものである。

平成 21 年度までの研究成果から両技術の相互補完的役割が評価検証され、同技術の学術雑誌及び国際会議による報告により一定の成果を達成した。

平成 22 年度の研究開発として平成 21 年度までの研究成果をベースに、特に問題検知後の対策、対応に必要となる問題特定をどのように行うかという視点で研究開発を実施した。開発コストの削減から期間減縮が強いられているソフトウェア開発現場において、問題が検知できるという診断と問題が特定できるという診断では、結果には大きな違いがある。

問題を発見することと、改修することは現状として異なる作業であり、改修に要するコスト、期間が十分ではない結果、当該問題が放置されるケースもあり得る。このような最悪のケースを回避する為にはより低コスト且つ容易に修正まで至る仕組みが必要とされている。

## ２　研究開発の全体計画

### ２－１　研究開発課題の概要

ア．網羅性向上

主要な診断方式である擬似攻撃診断は、ソースコードの網羅性が極端に低くなることがある。これは内部構造等に一切関与せず、予め用意されたパターンに従って診断を行っていることに原因がある。ソースコードにおける、特に分岐条件の網羅性を確保できないことが原因であり、ソースコード全体の網羅性を確保するには、ソースコードの解析をベースにした静的解析を組み入れる必要がある。本研究開発では Web アプリケーションのセキュリティに特化した問題検知が可能なソースコード診断の手法に関して研究開発を行う。一般的に静的解析では 70%以上のソースコード網羅率を達成でき、より広い範囲での問題検知を可能にする。

イ．誤検知の減少

一般的に静的解析では多くの誤検知が検出される。診断実施時に誤検知が大量発生すると生産効率を大幅に落とす結果となる。本提案技術においては、静的解析であるソースコード診断と動的解析である擬似攻撃診断を組み合わせることにより、ソースコード診断実施時に発生する誤検知を抑制させる手法についての研究を行う。

ウ．問題箇所の特定、絞り込み

ソースコード診断の段階で発見された問題は、発生箇所を絞り込むことが可能であるが、ブラックボックステストである擬似攻撃診断では、内部状態を把握できないことから、問題の発生要因を分析することは難しい。本研究開発では、擬似攻撃診断実施時に内部状態の把握を可能にする手法の研究を行い、問題の検知しか行えなかった擬似攻撃診断の実施時にも問題箇所の特定を行える手法の確立を目指す。

エ．修正支援

本研究開発では、最終的に静的解析及び動的解析の何れにおいても、問題発生時の実行トレースとデータの流れを追跡することが可能となる。これにより、問題箇所を如何に修正するかという点についても研究を行い、修正支援に関わる技術の研究及び開発を行う。

### ２－２　研究開発の最終目標（平成２２年９月末）

1.　ソースコード診断ツールの研究開発

（1）診断実施時のソースコード網羅率が 80%～90%を達成すること
本件研究開発ではソースコード網羅率が低くなりやすい擬似攻撃診断実施前に、予めソースコード診断を実施することで、擬似攻撃診断のパターン生成が内部構造を把握した状態で実行できる。これにより擬似攻撃診断実施時のソースコード網羅性を向上させることが可能となる。一般的なソースコード診断では 70%～90%程度が網羅性の平均であり、特に Web アプリケーションの入出力に特化した静的解析では、80%～90%程度の網羅率を確保できると考える。
（2）ソースコード診断における現状 50%の誤検知率を 30%以下に抑えること
ソースコード診断実施時に発生し易い誤検知を分析し、誤検知の発生頻度が高い問題に対しては、擬似攻撃診断実施時に検証することで、ソースコード診断実施時の誤検知を減少させる。

2. 擬似攻撃診断と実行時内部トラッキングによる問題検知ツールの研究開発
（1）擬似攻撃診断によって検知された問題に対して制御フローを追跡可能なこと
本研究では擬似攻撃診断実施前に、ソースコードの各制御点に、実行時の追跡を可能とするデータ出力を行う為の関数ラッピングを行う。これにより、擬似攻撃診断を仕掛けた際の、各テストパターンに対してどのような制御フローが発生したのかを追跡することが可能となる。
（2）擬似攻撃診断によって検知された問題に対してデータフローを追跡可能なこと
本研究では擬似攻撃診断実施前に、ソースコード内の主要 API に対して動的に用意する検証用 API へと置換を行い、プログラム実行時にコールされた各 API への値出力を実施することで、各テストパターンに対してどのようなデータのながれが生じたのかを追跡することが可能となる。
3. 問題箇所の特定及び修正支援ツールの開発
（1）問題発生時の制御フロー、データフローを視覚化すること
本研究ではソースコード診断または擬似攻撃診断で検知された問題に対して、制御フロー及びデータフローの分析が可能となる。問題発生時の制御フローを視覚化することで問題箇所の特定を支援する。
（2）修正支援機能を提供すること
本研究では各種問題毎に一般的な修正案を示し、さらに上記制御フロー及びデータフローの結果を用いることで、どのような修正を実施すれば該当の問題を抑制できるのかといった修正案を示し、静的診断、動的診断の診断サイクルを繰り返す中で、問題の修正確認を行える。

## ２－３　研究開発の年度別計画

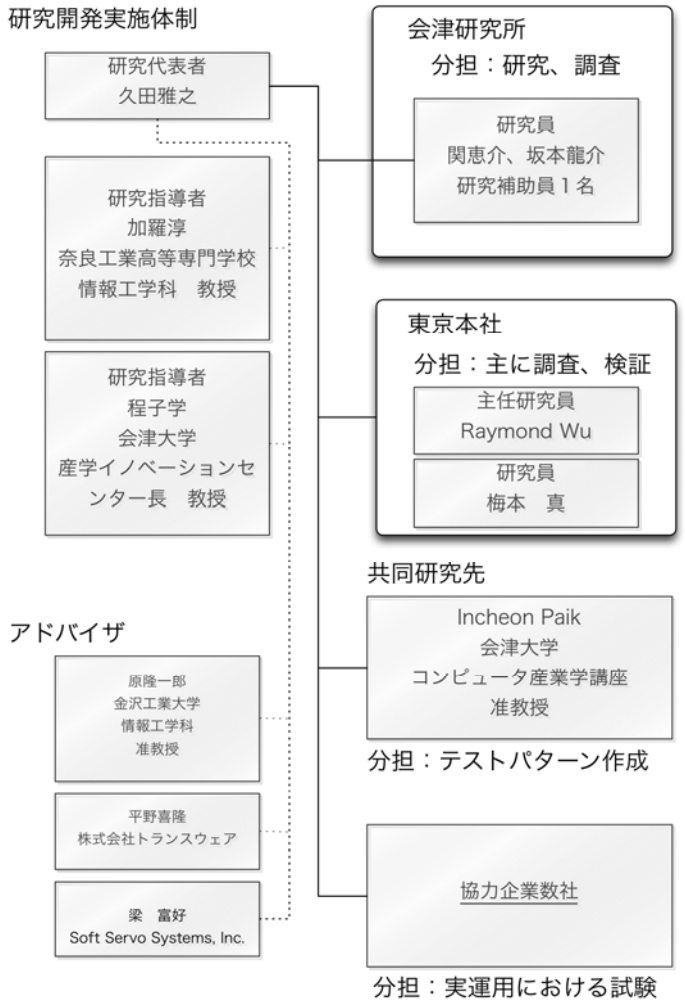
金額は非公表

| 研究開発項目 | 20 年度 | 21 年度 | 22 年度 | 計 | 備　考 |
|---|---|---|---|---|---|
| 静的及び動的解析の組み合わせによる Web アプリケーションのセキュリティ診断システムに関する研究開発 | | | | | |
| １　ソースコードの診断技術に関わる研究 | － → | － | | － | 20 年度 10 月から 9 カ月 |
| ２　ソースコード診断と擬似攻撃診断の相互連携手法の研究 | | － | | － | 21 年度 6 月から 6 カ月 |
| ３　擬似攻撃診断と実行時内部トラッキングによる問題検知手法の研究 | | － → | － | － | 21 年度 9 月から 1 年間 |
| ４　問題箇所の特定及び修正支援に関わる研究 | | | － | － | 22 年度 4 月から 6 カ月 |
| 間接経費額（税込み） | － | － | － | － | |
| 合　計 | － | － | － | － | |

注）１　経費は研究開発項目毎に消費税を含めた額で計上。また、間接経費は直接経費の３０％を上限として計上（消費税を含む。）。
　　２　備考欄に再委託先機関名を記載
　　３　年度の欄は研究開発期間の当初年度から記載。

# 3　研究開発体制

## 3－1　研究開発実施体制




１．脆弱性発生時の原因特定
２．擬似攻撃診断とソースコート診断の総合連携

１．各脆弱性の修正に関わる情報収集、調査
２．各脆弱性の知識ベース構築
３．修正支援に関するウィザードデータ作成

１．オープンソース等公開アプリケーションへのツール適用検証を実施

１．社内アプリケーションへのツール適用検証を実施

## ４　研究開発実施状況

### ４－１　サブテーマ１：ソースコードの診断技術に関わる研究

#### ４－１－１　脆弱性検知を目的とした静的解析手法

本研究開発においては、特に小、中規模のアプリケーション開発に用いられる PHP 言語を対象にプログラムの解析方法について研究開発を実施した。

本研究開発の実施にあたり、PHP 言語を対象とした静的解析ツールについて調査した。学術的な文献があり、且つオープンソースで提供されている Pixy を参考及び比較対象に決定した。動的解析として一般的に用いられる擬似攻撃診断と静的解析をベースにしたソースコード診断について、診断精度（特に誤検知率）及びソースコード網羅性の 2 点について比較した。Pixy は制御フローの解析及びデータフローの解析を実施することで、当該フローにおいて不正データが混入した際の処理について、その存在有無を解析する。同アプリケーションは主要な脆弱性の一つであるクロスサイトスクリプティングが検知可能である。Pixy の動作上の制約により大規模なアプリケーションの診断は実施出来なかったが、当社サンプルアプリケーションを対象とした診断で誤検知の発生状況を検証した。

データベースからの入力で当該データが確実に汚染されないことが保証されているケースでは、不正データのチェックや処理機能は必要とされない。今回確認した誤検知は全てチェック機能を必要としない制御フローにおいて、プログラム実行時に当該制御フローにおける不正データ混入に対して処理機能が不足している場合に報告された。これは全ての入力データを汚染されたと仮定して各制御フローの解析を実施していることに起因する誤検知であり、現状では誤検知の発生は避けられない。

商用擬似攻撃診断ツールを使用してクロスサイトスクリプティングの診断を実施した場合も誤検知と思われる結果が生じた（脆弱性有りと判定されているが混入スクリプトは実際には実行されなかった)。クロスサイトスクリプティングはスクリプト言語（一般的に脆弱性として発生しやすい例としては JavaScript 言語）で書かれた任意のプログラム混入によって発生する。今回の結果は当該テスト実施時に混入スクリプトが HTML の構成上は発動しなかったという結果であった。しかしながら HTML の規約上、HTML 内の特殊文字については HTML エンコードが推奨されていることを考えると、当該エンコード処理の不備が脆弱性を引き起こす可能性につながる事から、最終的に本結果は誤検知ではないと判断した。

検証では当社で用意したサンプルアプリケーションを対象にクロスサイトスクリプティングの診断を実施したが、結果的に擬似攻撃診断では誤検知は認められず、ソースコード診断においては誤検知が過半数を超える結果となった。診断対象のアプリケーションによって結果は異なるが、ソースコード診断において誤検知が多いという傾向は確認された。

次に診断実施時のソースコード網羅性について評価した。擬似攻撃診断を実施する際には結果取得までに

1. 事前巡回処理
2. テストケース生成
3. 診断実施（スキャン及びレスポンス解析）
4. 結果報告

の４行程がある。

擬似攻撃診断の診断精度を表す指標としては、当該診断がソースコード全体のどの程度を網羅したのかというソースコード網羅性があげられる。診断実施時のソースコード網羅性が仮に３０％であれば、残りの７０％は診断がなされておらず、当該箇所にセキュリティ上の問題が含まれる可能性を否定できず、結果的に診断の品質は低いと言える。故に擬似攻撃診断における精度向上を目的としたソースコード網羅性について検証することにした。

前述の Pixy を用いて当社サンプルアプリケーションを解析した結果、全ソースコードがパースされ、１００％網羅されていることを確認した。Pixy の制約上は大規模なソースに

対しての診断が困難であり、上記２アプリケーションに対してPixyで診断した場合の網羅性評価は実施出来なかったが、原理的には１００％網羅を達成できたと思われる。

オープンソースで開発されているBlogやCMS(WordPress, Simple PHP Blog)のWebアプリケーションを使って、擬似攻撃診断に必要となる巡回時ソースコード網羅性（本検証では分岐網羅を評価）を検証した。網羅性の計算には本研究開発におけるサブテーマ２で使用した当社独自の網羅性検証方式を利用し、プロトタイプ段階のツールを利用して網羅性を計算した。

2種類のアプリケーションに対する巡回時網羅性を表1に示す。Simple PHP Blogに対して通常アクセス可能な範囲で３回の巡回を実施したところ、これら巡回時網羅性の平均は30.75%であった。また WordPress を同じ手順で巡回した際に得られた網羅性の平均は19.60%であった。

以上、静的解析、動的解析のソースコード網羅性について検証した結果として、擬似攻撃診断の網羅性が当初想定していた以上に低く、その差が大きく開いた。

| アプリケーション名（バージョン） | 巡回時網羅率（１回目） | 巡回時網羅率（２回目） | 巡回時網羅率（３回目） | 巡回時網羅率（平均） |
|---|---|---|---|---|
| Simple PHP Blog (v0.5.1) | 28.85% | 31.69% | 31.70% | 30.75% |
| WordPress (v2.8.5) | 20.68% | 18.23% | 19.91% | 19.60% |

表１:巡回時網羅率

上記の網羅性検証において、網羅されなかった箇所をそれぞれ解析した結果、ある特定の変数値が入力されることによって当該箇所が実行されるケースが多くみられた。事前巡回の一般的なプロセスとしては、自動巡回及び手動巡回共に各ページ内に存在するリンク構造を元に遷移していくこととなり、当該ページを構成するHTMLやJavaScript内に含まれるリンク構造ではアプリケーション内部の全ての変数を網羅できておらず、アプリケーション内部に潜在的にある変数を網羅できないことが網羅性低下の一因であることを確認した。

巡回時に取得出来なかった各入力変数をソースコードの静的解析により抽出し、テスト生成のプロセスに反映させる事は網羅性の向上に繋がると考え、関連モジュールについて研究開発を行った。今回の実装では第一段階として、前記Blog及びCMSを対象とし、これらアプリケーションがPHPで構成されていることから、PHPで書かれたアプリケーションの入力変数を自動抽出するモジュールを開発することとなった。なおPHPの言語仕様では入力変数は_GET及び_POSTによってソースコード内で参照される為、当該モジュールはソースコードを解析しこれら入力変数が参照される箇所を検出、使用されている変数名を抽出する動作となる。

本モジュールの動作としては、予め指定した予約語について変数名（_GET及び_POSTを対象とした）を自動で取り出し記録する。_GET及び_POSTは予約後で有る為、各ソースコードに対して、文字列前方一致で当該予約語の使用を検索する方法を使用し、実装を完了した（最終的には抽象構文木上で抽出)。当該モジュールはサブテーマ２の事前処理として利用された。

サブテーマ３では、擬似攻撃診断実施時に発見された脆弱性が発現した制御フローを特定、明示出来ることを目標としており、当該目的達成の為に必要となるWebアプリケーション実行時の各制御フローを記録する為の方法について研究開発を実施した。

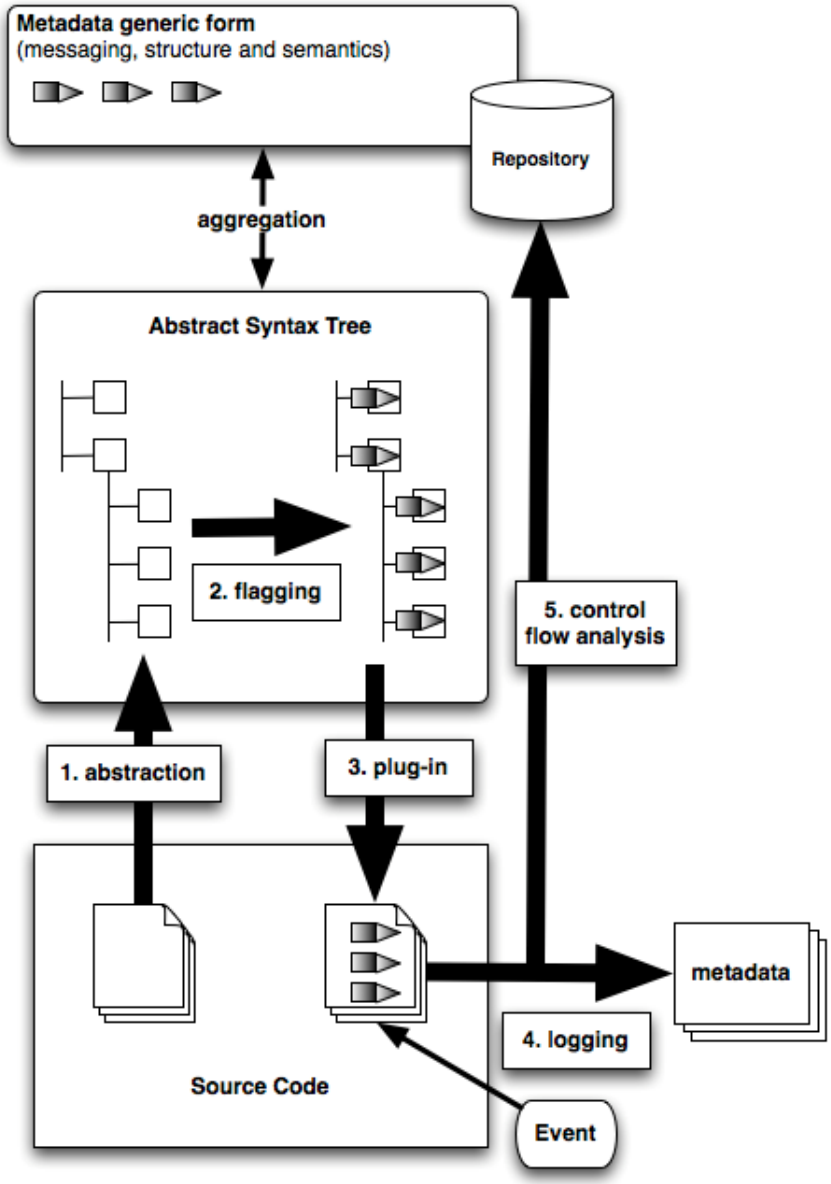



**図 1**
制御フロー特定を目的としたソースコード改変技術

図 1 は本研究開発の概要を示すもので、基本的な考え方としては診断対象となるアプリケーションのソースコードから、診断用途のソースコードを生成し、当該生成コードを用いてプログラムを実行すると内部状態が記録され、結果として制御フローを特定できるというものである。

1. 診断対象のソースコードは、字句解析及び構文解析により抽象構文木へと変換される（abstraction）
2. 抽象構文木は解析され、if 文や switch 文など制御を決める命令付近に記録を目的とした特殊なコードを埋め込む（flagging）
3. 変更された抽象構文木は、ソースコードに書き戻され（plug-in）実行される。
4. 実行時にプログラムに対して何らかのイベントが発生した際には、各制御文においてメタデータが記録される（logging）
5. これらメタデータの記録から、当該イベントに対する実行フローは解析により特定される（control analysis）

本研究開発では、実装期間の短縮の為に抽象構文木の生成には Eclipse に含まれる PDT を利用した。

以上の様にソースコードの改変技術をベースとした制御フローの特定技術を確立させ、擬似攻撃診断実施時の制御フロー特定及び網羅性の評価を実現する技術確立に成功した。

なお、PDT と同様の JDT による抽象構文木生成、抽象構文木の書き換え及びソースコードの書き戻しも試験し、動作確認を行った。本ツールは Java ベースのアプリケーション(Servlet, Struts を用いたアプリケーション)にも対応を完了している。

### ４－１－２　擬似攻撃実施時の網羅性調査

前記ログ命令の自動追加モジュールを利用することで、ソースコード内の各分岐点に通過条件を記録するためのコード改変が可能となった。各分岐点の通過可否を判定することで、プログラム実行時の分岐網羅性を評価することが可能となる。Web アプリケーションが実行された際のメタ情報は個別識別子により管理され、実行時の識別子は当該アプリケーションが返したレスポンス内部に埋め込まれる。以下は HTML のサンプルであり、HTML の閉じタグ直前に埋められた数字が上記識別子となる。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>NST(デモサイト)</title>
</head>
<body>
...省略...
</body>
<!--_NST_SCA_CTRL_72,35,79,92,77,90,75,89,71,73,80,-->
</html>
```

識別子を順次参照していくことで、この HTML が生成された際の制御フローは以下の通り特定される。

```
72 | _NST_SCA_CTRL_ | footer.php | TopStatement | 0 | 0
35 | _NST_SCA_CTRL_ | content/welcome.php | TopStatement | 0 | 0
79 | _NST_SCA_CTRL_ | index.php | block | 29 | 0
92 | _NST_SCA_CTRL_ | menu/public_left_menu.php | TopStatement | 0 | 0
77 | _NST_SCA_CTRL_ | index.php | block | 20 | 0
90 | _NST_SCA_CTRL_ | main_navi/public_main_navi.php | TopStatement | 0 | 0
75 | _NST_SCA_CTRL_ | index.php | block | 8 | 0
89 | _NST_SCA_CTRL_ | lib.php | TopStatement | 0 | 0
71 | _NST_SCA_CTRL_ | do.php | TopStatement | 0 | 0
73 | _NST_SCA_CTRL_ | header.php | TopStatement | 0 | 0
80 | _NST_SCA_CTRL_ | index.php | TopStatement | 0 | 0
```

オープンソースを中心に上記コード改変技術を用いて、分岐網羅性の評価を実施したところ、当初想定していた以上に分岐網羅性の確保が難しいことがわかった。一般的なクローリングアルゴリズムを用いて自動巡回した場合は最大で約３０％、人が無意識に巡回した場合で最大で約５０％、相当意識して巡回作業を実施しても最大で約７０％程度の網羅性しか確保できない結果となった。なお大規模なアプリケーションほど網羅性は低く、網羅性が３０％に達しないケースも多かった。

高網羅の条件としては前記の通りソースコードの分岐条件に使われている変数が、ページ内に存在するリンクの URL にて変数として参照されていることが必要である。該当の変数が呼ばれていない場合は、常に一定の分岐条件を満たす事から網羅性が低くなることがわかった。本結果はサブテーマ２における網羅性向上のアプローチを導き出した。

平成２２年度の研究開発として、巡回実施時に網羅性を確認できるよう変更を加え、意識的に網羅性を向上させられるよう改善した。これにより平均的に７０％超の網羅性を確認でき、巡回時網羅性８０％を達成した例もあった。以下はオープンソースを中心に巡回時の網羅性を計測したサンプルである。

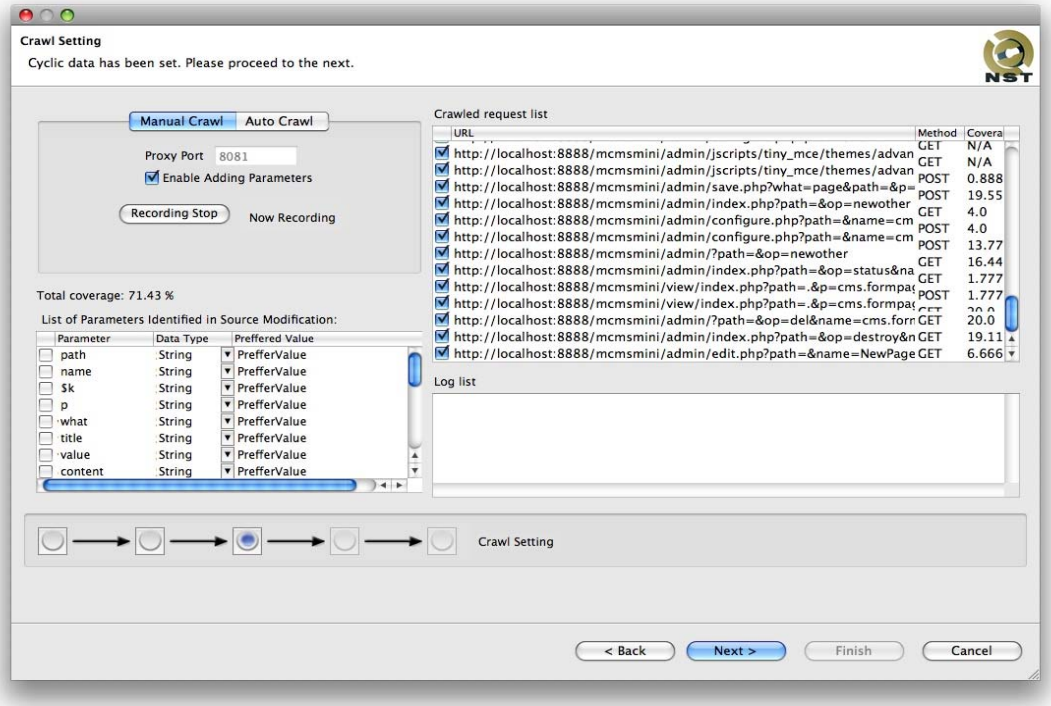



**図 2**
CMSMini 巡回時網羅性: 71.43%

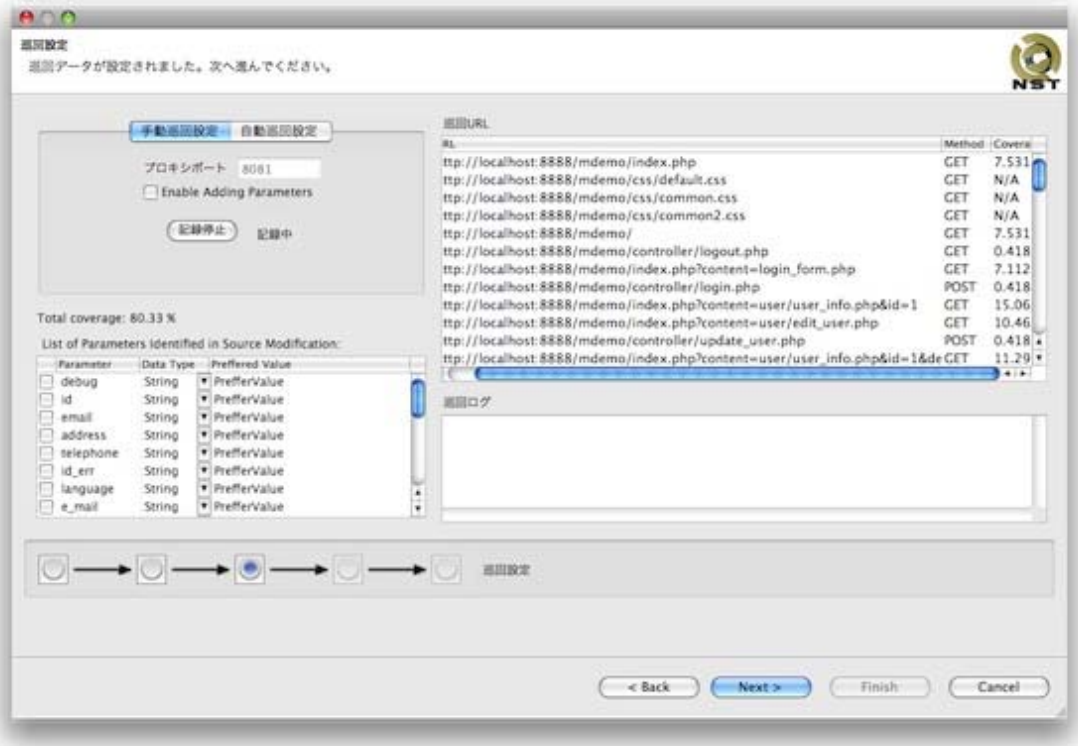



**図 3**
当社デモサイト巡回時網羅性: 80.33%

### ４－１－３　ソースコードの抽象化及び脆弱性検知の多言語対応

PDT 及び JDT が生成する抽象構文木を統一して解析可能なユニバーサルアダプタの設計及び実装を行った（図２）。言語仕様の差異は抽象構文木を用いることで吸収され、プログラムの変更手続きは構文木上の操作に変換される。また言語単位で抽象構文木の定義情報が異なる場合には、各ノードの対応関係を別に定義しておくことで、前記コード解析エンジンの多言語対応が容易になった。本研究開発で作成したソースコード解析エンジンはPHP言語及び Java 言語に対応している。また平成２２年度の研究開発として新規に機能追加した脆弱性発現箇所のコード解析にも抽象構文木を利用している。

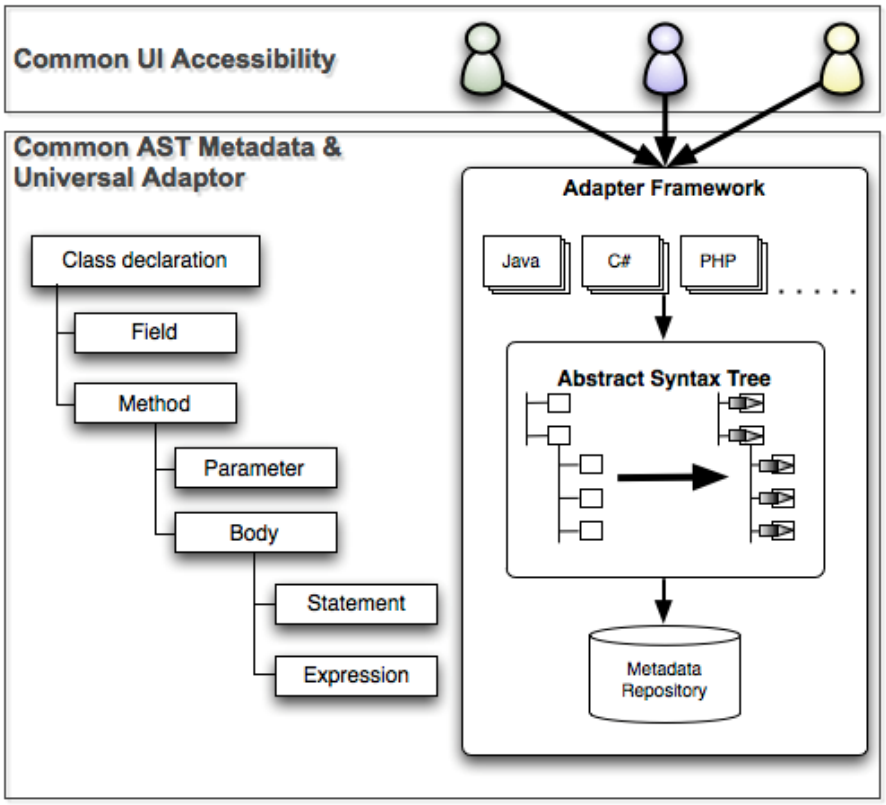



図　４
ソースコード改変の多言語対応

### ４－２　サブテーマ２：ソースコード診断と擬似攻撃診断の相互連携手法の研究

### ４－２－１　擬似攻撃診断実施時の網羅性向上

主要な Web アプリケーションサーバーの仕様を調査し、それぞれの動作を検証した結果、リクエストに含まれる変数項目は事前にパースされ、コード実行時に参照があった場合に値を返す仕組みが取られており、コード内で参照されていない変数を URL に付加しても特に影響が無い事がわかった。PHP 及び Java の主要な環境で検証を実施し、問題が発生しないことを確認した。

本結果を踏まえ、事前にソースコード解析を実施し、入力可能な変数項目を抽出しておくことで、巡回時に不足している入力変数を自動追加し網羅性を向上させる為の方法を考案し、下記テストパターン生成の手法確立に至った。

### ４－２－２　擬似攻撃診断実施時の効果的テストパターン生成

前記、変数抽出モジュールを利用して、擬似攻撃診断実施時の網羅性向上を目的とした連携手法について研究を実施した。

擬似攻撃診断は、実際に攻撃を仕掛けてその反応を調べる診断方式である。診断時には

事前巡回を実施し、巡回時に利用された URL や変数項目に対して既知の攻撃パターンを総当たりで埋め込んでテストケースを生成する。サーバーへ実際に各テストケース（リクエスト）を送信し、そのレスポンスを解析、問題の有無を判定する。

前述の調査結果から、発生したリクエストに含まれる変数不足が網羅性の低下原因として多く、ソースコード内で参照される変数項目を網羅することで分岐網羅性の向上が期待されることから、結果的に変数追加は擬似攻撃診断で使用されるテストケースを効果的に生成する作用を持つと考えられる。検証の為に Web ページ内からは参照されない変数をソースコード内部で定義し試験を実施した。当該検証アプリケーションは弊社内で試験用に作成したものであり、デバッグ用途の変数が内部に５つ存在、これら変数により新しく分岐が発生するよう構成されている。

当該検証アプリケーションにおける分岐点は９３カ所であり、巡回時の分岐通過点は４６カ所で巡回時分岐網羅率は約５０％であった。これら巡回データに対してソースコードから抽出された前記５つの変数を追加することで、通過点は５６カ所となり分岐網羅率は約６０％まで向上した。結果的に追加された変数により１０カ所の分岐が新たに発生したこととなる。

また WordPress 管理画面のログイン関連ページに対して本手法を適用したところ、当該 URL の分岐網羅率は 6.54%であったが、内部から抽出された変数の追加により分岐網羅率は 12.77%まで向上した。これにより一般的に使用されているアプリケーションに対しても本手法の有効性が確認された。

今回の手法適用による分岐網羅率の向上はアプリケーションにより大きく異なる。しかしながら Web ページ内で参照されない変数により制御が変わるアプリケーションでは、確実に分岐網羅性を向上させることが可能となる。

## ４－３　サブテーマ３：擬似攻撃診断と実行時内部トラッキングによる問題検知手法の研究

### ４－３－１　トラッキング方法の確立・検証

擬似攻撃診断実施時の制御フローを特定する技術について研究を実施した。PHP 言語及び Java 言語で書かれたプログラムに対して、各種制御命令の成立条件をプログラム実行時に記録出来るコード改変を安定的に行う技術を確立した（図１）。

対象のメタデータとして、プログラムコードの ID、行番号、制御命令の種類、コードインデントの深度を記録し、擬似攻撃診断の各テストケースに対してそれぞれどのような制御を取ったのか出力が可能となった（4－ 1－ 2にて例示)。また各制御文の正否を特定できることから、分岐網羅性の評価もでき、改変コードに対する事前巡回及び擬似攻撃診断の実施時には分岐網羅性を評価することが可能となった。なお、本研究における分岐網羅性評価は全て本成果を用いて取得している。

また制御フローに加え、プログラム実行時のデータ取得についても研究を実施した。Web アプリケーションの脆弱性において、その大半はデータの入出力に関連して発生する。またＳＱＬインジェクション等のバックエンドに対して発生する脆弱性については、問題が表層部分に帰らないことが多く、擬似攻撃診断による検知では限界がある。

本研究開発では、ＳＱＬインジェクションをこれまでとは違った形で検知できるよう、テストパタン入力時のＳＱＬクエリを取得可能な機能を追加した。これにより不正なデータを入力した場合に、当該データが正常にエスケープされずにデータベースへ到達する現象（結果的にＳＱＬインジェクションを発生させる）を正確に検知できる。主な機能としてはソースコードの改変により当該クエリを記録、取得可能な仕組みがある。以下は診断対象となるアプリケーションのソースコードにおけるＳＱＬクエリを発行している箇所で、診断用途の改変コードでは代入によるクエリデータの取得と当該データの記録関数がコールされている。

〇診断対象のオリジナルソースコード：

```
$res = mysql_query("SELECT * FROM users WHERE id='". $user_id ."'",$this->conn) or die(mysql_error());
```

○自動改変されたソースコード：

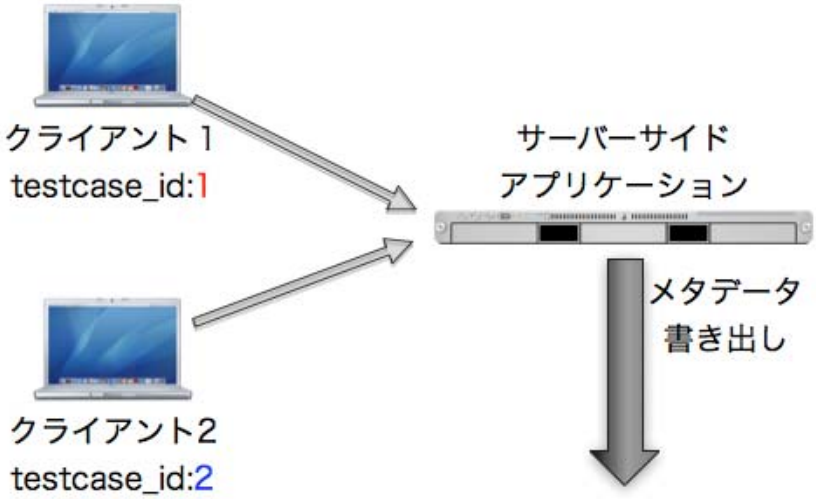
```
$sql = "SELECT * FROM users WHERE id='". $user_id ."'";
$res = mysql_query($sql,$this->conn) or die(mysql_error());
Inserter::insert_query($sql);
```

上記サンプルコードでは、改変によりＳＱＬの入出力に関わる mysql_query 関数の第一引数が別変数に代入され、埋め込まれた Inserter クラスの insert_query 関数をコールすることにより、ＳＱＬデータが記録される仕組みとなっている。

### ４－３－２　トラッキングデータの解析法

トラッキングデータはプログラム実行時に自動的に記録されるが、擬似攻撃診断実施時には複数のスレッドが同時にテストケースを送信することが一般的であり、各テストケースと記録されたトラッキングデータを効率的に対応づける仕組みが必要となる。

本研究開発ツールでは、PHP に対する診断実施時には各テストケース送信時にリクエストヘッダにテストケースの識別番号を埋め込み、コード改変により埋め込まれたロギング命令が該当の識別番号をログとしてメタデータと共に記録することで、上記複数スレッドによるテスト送信に対応した仕様となっている(図５)。




```
72|_NST_SCA_CTRL_| footer.php | TopStatement | 0 | 1
35|_NST_SCA_CTRL_| content/welcome.php | TopStatement | 0 | 1
79|_NST_SCA_CTRL_| index.php | block | 29 | 1
72|_NST_SCA_CTRL_| footer.php | TopStatement | 0 | 2
35|_NST_SCA_CTRL_| content/welcome.php | TopStatement | 0 | 2
79|_NST_SCA_CTRL_| index.php | block | 29 | 2
92|_NST_SCA_CTRL_| menu/public_left_menu.php | TopStatement | 0 | 1
92|_NST_SCA_CTRL_| menu/public_left_menu.php | TopStatement | 0 | 2
77|_NST_SCA_CTRL_| index.php | block | 20 | 1
77|_NST_SCA_CTRL_| index.php | block | 20 | 2
90|_NST_SCA_CTRL_| main_navi/public_main_navi.php | TopStatement | 0 | 1
75|_NST_SCA_CTRL_| index.php | block | 8 | 1
89|_NST_SCA_CTRL_| lib.php | TopStatement | 0 | 1
90|_NST_SCA_CTRL_| main_navi/public_main_navi.php | TopStatement | 0 | 2
75|_NST_SCA_CTRL_| index.php | block | 8 | 2
89|_NST_SCA_CTRL_| lib.php | TopStatement | 0 | 2
71|_NST_SCA_CTRL_| do.php | TopStatement | 0 | 1
73|_NST_SCA_CTRL_| header.php | TopStatement | 0 | 1
80|_NST_SCA_CTRL_| index.php | TopStatement | 0 | 1
71|_NST_SCA_CTRL_| do.php | TopStatement | 0 | 2
73|_NST_SCA_CTRL_| header.php | TopStatement | 0 | 2
80|_NST_SCA_CTRL_| index.php | TopStatement | 0 | 2
```

図　５
複数スレッドからのテスト送信とメタデータの記録識別

### ４－３－３　脆弱性の原因特定及び擬似攻撃診断とソースコード解析の相互連携

従来の擬似攻撃診断では問題の内在判定に止まり、問題の発生要因に関わる情報は提供されない。問題が発見された場合に、当該問題がどのようにして発生したのかが表示されることで、問題発生時の原因特定が飛躍的に容易になる。４－３－１及び４－３－２の基礎技術により、擬似攻撃診断実施時の制御フローを取得出来るようになった。当社が独自に製品開発している擬似攻撃診断ツールに制御フローが表示可能な機能を追加した。

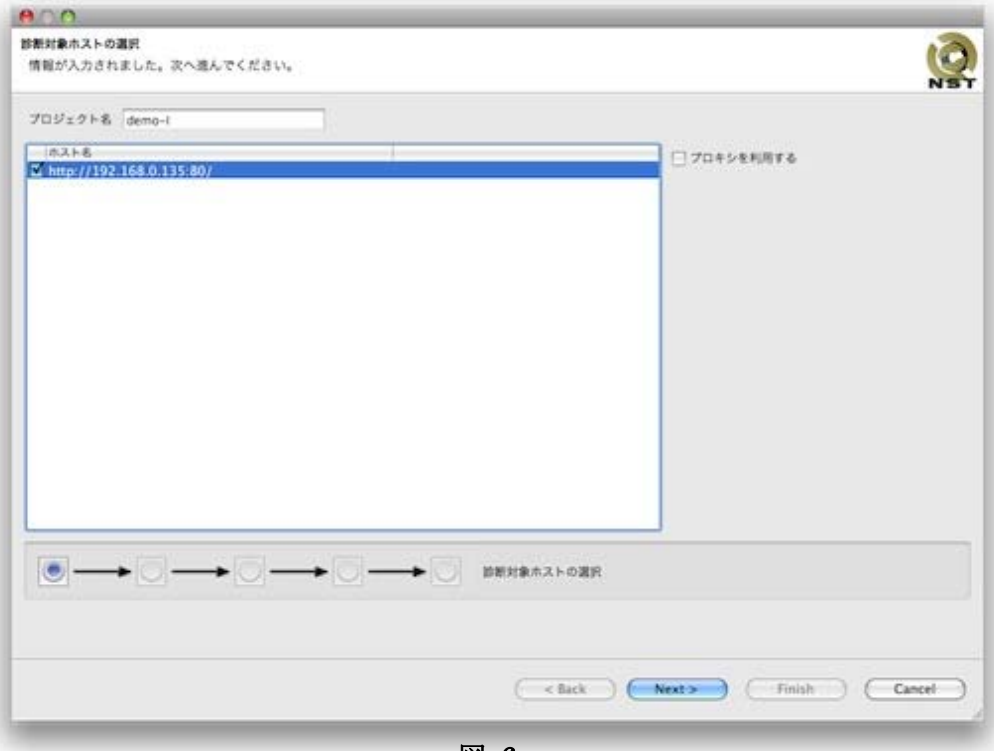



**図 6**
診断対象ホストの選択

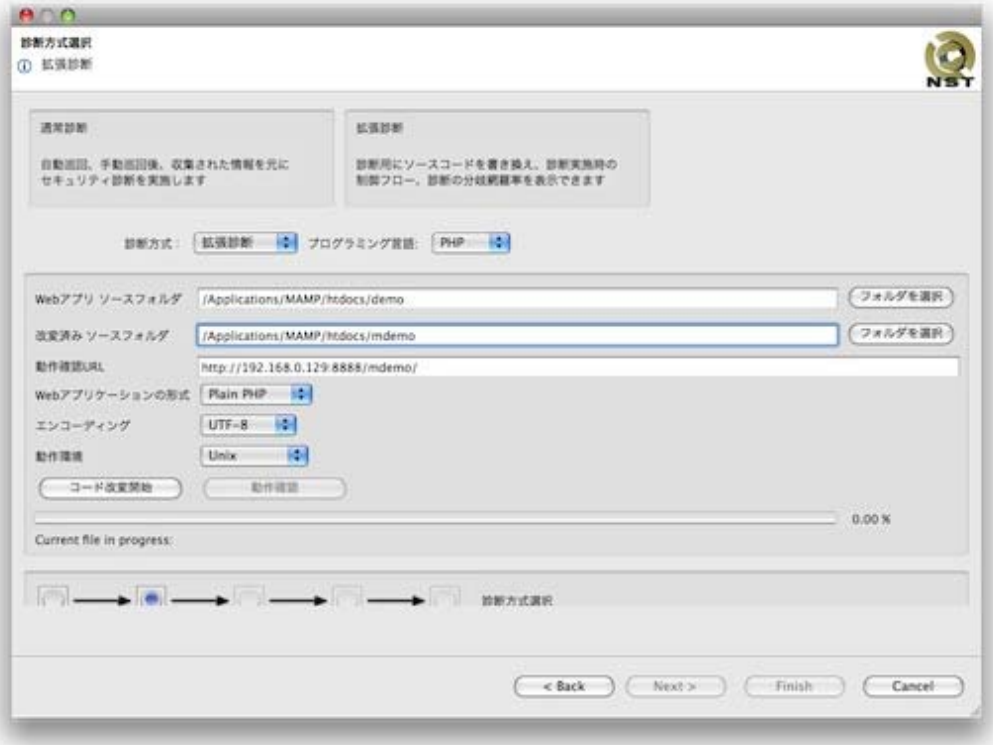



**図 7**
ソースコードの改変及び設置確認

図６は診断対象となるホストを選択する画面で、事前登録されたホストに対して診断が実施される。図７は本研究開発によって追加された機能であり、ソースコードの改変を実施する画面となる。診断対象のアプリケーションソースコードが含まれるフォルダを選択し、ユーザーにより指定されたフォルダに改変されたソースコードが生成される。改変されたソースコードをアプリケーションサーバーに設置、確認後、巡回作業開始となる。

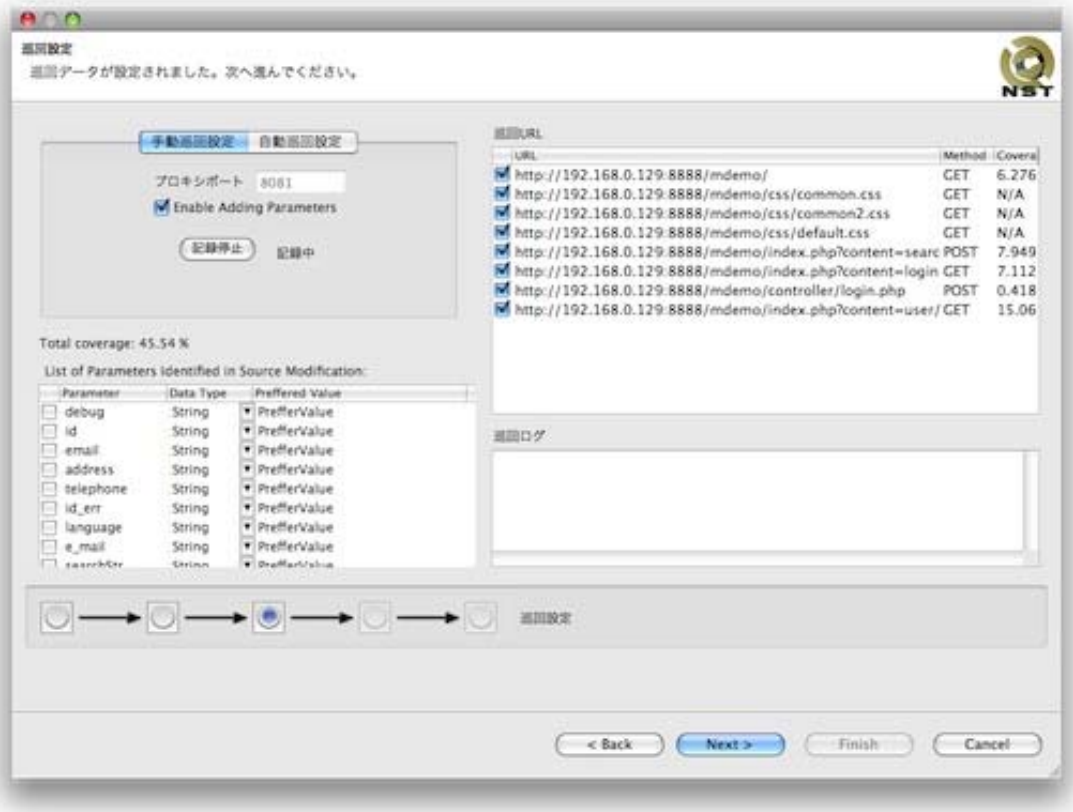



図 ８
ソースコードの改変及び設置確認

図８は手動巡回を実施している画面となり、巡回時の網羅性（画面左中央：４５．５４％）が順次更新されていく。網羅性は各リクエスト単位で集計され（画面右上）、全制御点に対して通過した割合が表示される。４－ ２－ １及び４－ ２－ ２で説明した変数の抽出（画面左下）も実現されている。抽出された変数は巡回データに付与され、一般的な診断と比して網羅性が向上する。これら機能は平成 22 年 9 月現在において販売されている擬似攻撃診断ツールには存在せず、同機能は他製品と比べ非常に優位性が高い。なお巡回後のテスト生成及び診断実施は一般的なツールと同じである為に割愛する。

図９は結果表示のサンプルであるが、特徴として下部に表示されているＳＱＬ文が上げられる。このＳＱＬ文は４－ ３－ １で説明した置換及び記録関数の呼び出しにより実現されているもので、本機能も当社製品固有の機能である。現時点ではデータのトレースに止まるが、将来的にはＳＱＬインジェクション等のバックエンドに対する脆弱性に対して、より高品質な診断実施を可能にする技術である。

図１０は４－ ３－ １及び４－ ３－ ２で説明されている手法により取得されたテスト実施次の制御フローである。各テストケース単位でプログラムがどのように実行されたかが確認でき、問題の特定に至るまでの時間を大幅に短縮出来る。

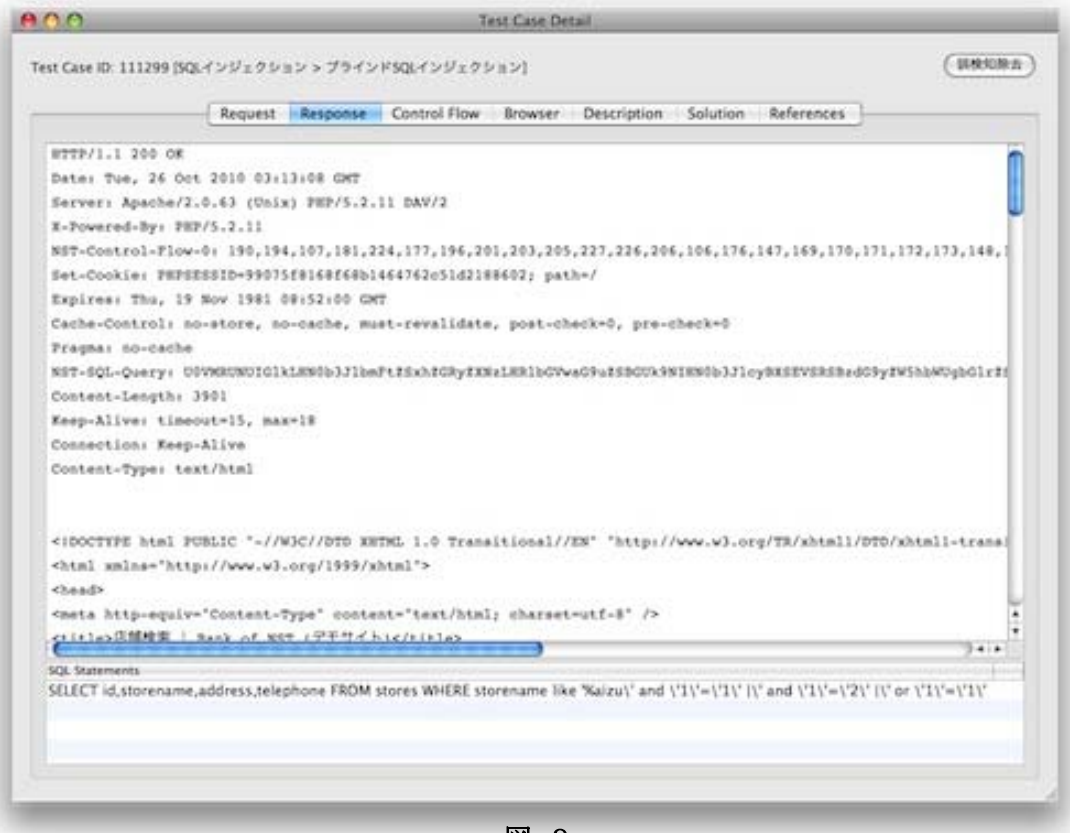



図 9
テスト結果の表示

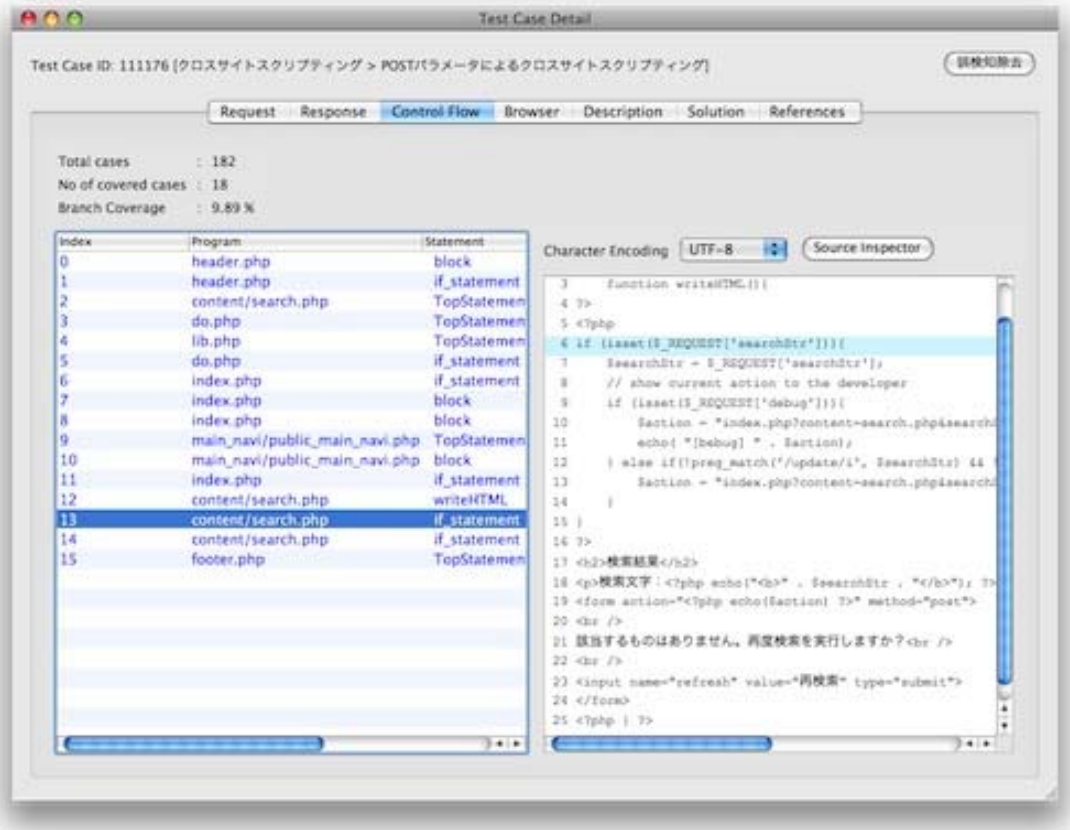



図 10
診断実施時の制御フロー表示

## ４－４　サブテーマ４：問題箇所の特定及び修正支援ツールに関わる研究

### ４－４－１　各脆弱性の修正に関わる情報収集、調査、知識ベースの構築

WebCheckup ではWeb Application Consortium により標準化されたカテゴリをベースに、２８のカテゴリ及び６２０のサブカテゴリについて診断機能を備えている。各サブカテゴリには解説文が入っており、脆弱性の詳細やリファレンス情報を有している。今回の研究開発では、擬似攻撃診断の結果から問題特定に至る手順を容易にする為の機能を追加することから、特に重要な脆弱性について、より詳しい解説の準備と当該問題を修正する為の基本情報を収集した。修正支援機能の対象とする重要な脆弱性の選定は当社が公開している診断統計レポート（Beige Book）を基準に脆弱性として発現頻度の多い、フィルタ処理の不足による脆弱性を対象とした。なお診断統計レポートは２００８年４月１日から２００９年3月３１日に当社が診断事業として実施した Web アプリケーション診断において集計したものであり、フィルタ処理の不足による脆弱性（主な脆弱性としてはクロスサイトスクリプティングとＳＱＬインジェクション）が全体に占める割合は４１．８４％である。


図 １１
脆弱性の詳細解説及び対策方法

図１１は解説のサンプルであり４－ ４－ ２の機能により、ユーザーはソースコードの該当箇所を確認しながら、脆弱性の詳細情報と共に対策方法について確認できる。

### ４－４－２　修正支援に関わるウィザードデータ作成

WebCheckup の拡張診断は、診断対象のアプリケーションに対するソースコード改変により、脆弱性検知時の制御フローを特定可能にする。クロスサイトスクリプティング及びＳＱＬインジェクションの脆弱性は、データの入出力チェックに関わる問題であり、これら問題が発生するようなソースコードのパターンは事前定義可能である。問題が検知された制御フローにおいて当該パターンを存否確認することで修正支援の為の付加情報を提供可能にした。

下記擬似コードは PHP のソースコードを PDT ライブラリにより抽象構文木に変換した上で、問題が発見された際の制御フローにおいて、当該フローの抽象構文木を走査することで、特定パターンの存否を確認しているものである。

```
ArrayAccess: something[] {

    if ( something == $GET, $POST, or $REQUEST ) {
        // something is an AST node, so find the ancestors
        // if any of the ancestor is an echo stmt get it
        EchoStatment = getParentEchoStmt(something);
        if( null != EchoStatment ) {
            // this is a vulnerable stmt: echo with request param
            // like echo $GET['name'];
            new Breach(lineNumber, CRITICAL, XSS);
        }
    }
}

EchoStatement: echo("".variable.functionCall(params)."") {

    // echo has list of expressions like function calls, values, variables
    for (Expression expression : echoStmt.getExpressions() ) {
        if( expression instanceof ParanthesisExpression ){
            // detect a variable: find child nodes which are
            // types of Variable
            if(variableDetected(expression)){
                new Breach(lineNumber, MEDIUM, XSS);
            }
        }
    }
}

InfixExpression: something."".$a {

    // infix stmt has left and right, side can be another infix
    // Scalar is a string
    if(infix.getSide() instanceof Scalar){
        // checks whether this scalar has any sql keywords in it
        isSQLPresent = isSQLStatement(scalar);
    }
    // check whether infix has any variables in it
    if(infix.getSide instanceof Variable){
        isVariablePresent = true;
    }
    if( isVariablePresent && isSQLPresent ) {
        new Breach(lineNumber, CRITICAL, SQLI);
    }
}

FunctionInvocation: method_call(params) {

    // checks whether this is mysql query invoke
    if (method_call.getFunctionName == mysql_query ) {
        // collect sql stmts and inserts query to list of queries
        // at runtime this will get added to headers when the function is
called
        collectQueries(method_call);
        new Breach(lineNumber, LOW, SQLI);
    }
}
```

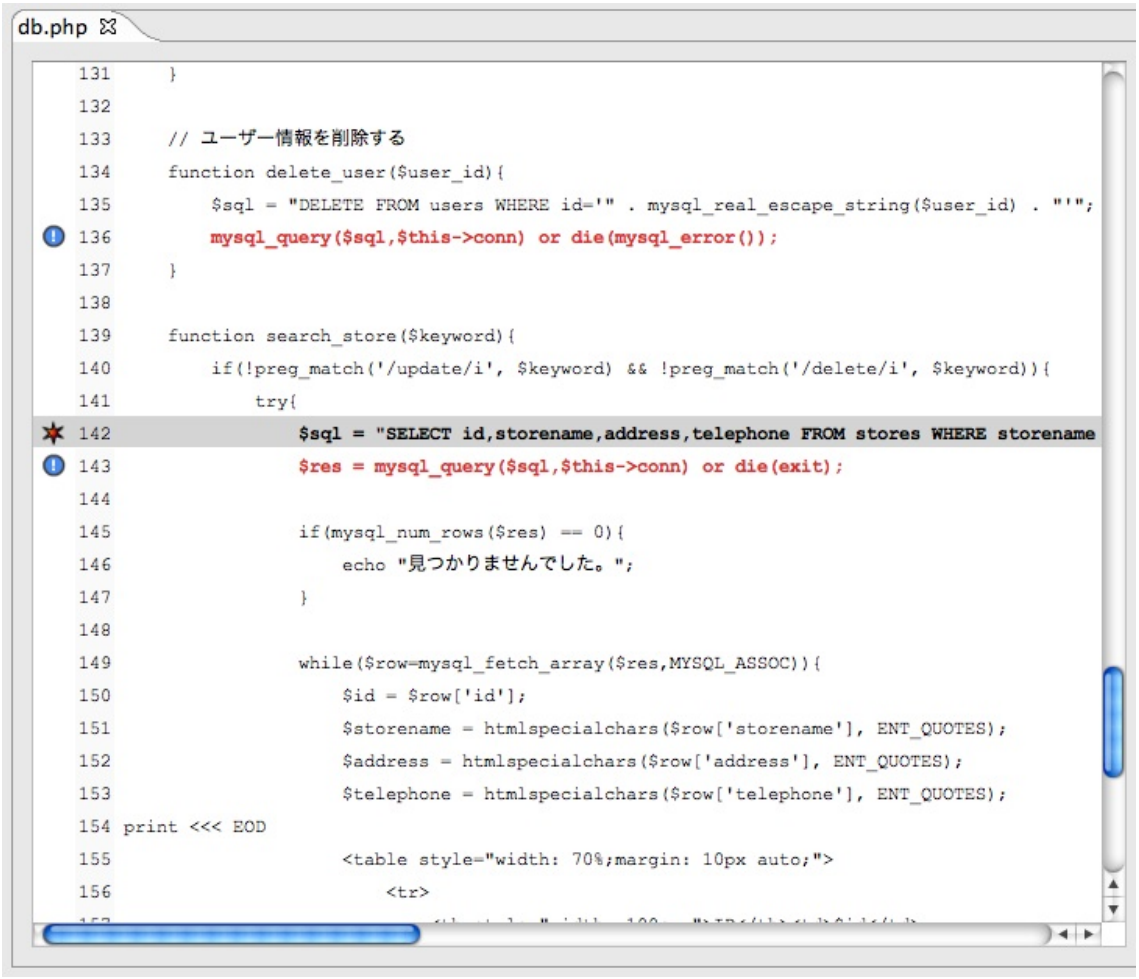



図 １２
ＳＱＬインジェクションの検知及び原因箇所の表示

図１２は脆弱性に関連するコードパタンを警告しているサンプルであり、行番号左に表示されているアイコンは問題の発生有無を表示している。１３６行目と１４３行目のアイコンは当該ラインにデータベースコールが存在することを示している。ここではＳＱＬインジェクションが検知されたことから、データベースにアクセスする関数の有無が解析され表示された結果となる。１４３行目の関数コールでは第一引数がその前行で代入されているため、１４２行目には最も警告度合いの高いアイコンが表示されている。

ユーザーは問題が発生した際の制御フローを順次追跡し、図１２と同様の警告を確認していくことで問題箇所を迅速に特定することが可能となった。

## ４－５　総括

本研究開発においては大きく下記６つの成果が得られた。

１．擬似攻撃診断を実施する際の網羅性評価方法の確立
２．抽象構文木を用いたソースコードの自動改変
３．巡回時網羅性の取得
４．問題検知時の制御フロー取得
５．問題発生箇所を特定する為の支援機能
６．問題を修正する為の支援機能

擬似攻撃診断はブラックボックステストであり、入力データに対する出力データの解析を実施する。よってセキュリティ診断を実施した場合に、当該アプリケーションのソースコードに対して、どの程度の範囲で（ソースコード網羅性）試験が実施されたかを知る事は出来ない。試験実施された結果として問題が検知されなかったとしても、ソースコード

全体が網羅されていなければ試験実施されなかった範囲に問題が存在する可能性がある。
本研究開発では、ソースコードの一部を自動で改変し、プログラム実行時の各制御点において通過したかどうかの情報を記録し、制御フローの特定を可能にした。これにより診断を実施する際の事前巡回時にどの制御点を通過したかが取得可能となり、結果的にソースコード全体に対する分岐網羅性を評価することが可能となった。ソースコードの改変は、言語依存性を排除するために抽象構文木を生成し、構文木上での改変を実現した。また構文木上で改変を加えることで、ソースコードを破壊することなく改変を加えることを可能にした。
各制御点の通過可否が判断出来ることで、各テストケースにおけるプログラム実行時の制御フローを特定することが可能となり、ブラックボックス試験として実施されてきた擬似攻撃診断の結果として、どこがどのように実行され問題が発生したのかが確認できるようになった。また当該制御フローにおいて主要な攻撃手法でもある、クロスサイトスクリプティング及びＳＱＬインジェクションについて、これら脆弱性の発現要因となるコードパタンを解析することで、問題を特定するための支援情報を提供し、また当該問題の修正方法を提示することで、ユーザーは問題発見後の対応をより迅速に、容易に実施できるようになった。これらの基本技術は当社独自のものであり、本研究開発に関する特許も 2 件出願された。当社製品における中核技術として今後活用される技術となる。

最後に研究開発の最終目標についてその達成状況は以下の通りである。

１．ソースコード診断ツールの研究開発

（１）診断実施時のソースコード網羅率が８０％～９０％を達成すること
４－ １－ ２で述べたとおり研究開発により得られた成果物を利用して、８０．３３％の網羅性を確認しており達成された。当社が実施した実験の範囲では、大半のアプリケーションにおいて、単純に診断を実施した際には５０％以下の網羅性しか得られない事が多かった。本結果を反映させた成果物を用いることで７０％台の巡回網羅を達成できるようになった。
（２）ソースコード診断における現状５０％の誤検知率を３０％以下に抑えること
今回の研究開発対象である脆弱性診断ツールは、擬似攻撃診断をベースにソースコードの診断実施をしており、その原理的特性から擬似攻撃診断の誤検知率に準ずる。当社が研究開発したソースコード解析技術は、擬似攻撃診断の実施を前提としたコード解析であり、原理的にコード解析及び改変の範囲では誤検知は発生しない為、本目標も達成されている。

２．擬似攻撃診断と実行時内部トラッキングによる問題検知ツールの研究開発

（１）擬似攻診断によって検知された問題に対して制御フローを追跡可能な事
４－ ３で述べたとおりサブテーマ３により達成された。
（２）擬似攻撃診断によって検知された問題に対してデータフローを追跡可能な事
４－ ３－ １で述べたとおりＳＱＬクエリとして渡されるデータの取得及び記録を実現し、バックエンドの診断技術確立に至った。ツール自体はコード実行時の各点でデータ出力を行っていないが、当該技術は本目標を満たすものである。

３．問題箇所の特定及び修正支援ツールの開発

（１）問題発生時の制御フロー、データフローを視覚化すること
４－ ４サブテーマ４で述べたとおり達成された。

（２）修正支援機能を提供すること
　４－ ４－ ２により達成された。
以上の通り、提案時に掲げた目標値については全て問題無く達成された事を報告する。

## ５　参考資料

### ５－１　研究発表・講演等一覧

**学術雑誌**

雑誌名：International Journal of Electronic Security and Digital Forensics
発表者：Raymond Wu and Masayuki Hisada
発表タイトル：Static and dynamic analysis for web security in industry applications
発表月日：平成 22 年 5 月

雑誌名：Journal of System and Information Technology
発表者：Raymond Wu and Masayuki Hisada
発表タイトル：The Architecture and Industry Application of Web Security in Static and Dynamic Analysis
発表月日：平成 22 年 5 月

雑誌名：Journal of Object Technology
発表者：Raymond Wu and Masayuki Hisada
発表タイトル：SOA Web Security and Applications
発表月日：平成 22 年 3 月

雑誌名：International Journal of Computer Science and Network Security
発表者：Raymond Wu and Masayuki Hisada
発表タイトル：The Architectural Review of Web Security in Static and Dynamic Analysis
発表月日：平成 21 年 9 月

**国際会議**

会議名：International Conference on Security and Management
発表者：Raymond Wu, Masayuki Hisada, Atsushi Kara and Takafumi Hayashi
発表タイトル：Event Capture Approaches for Vulnerability Detection
発表月日：平成 22 年 7 月

会議名：International Conference on Internet Computing
発表者：Raymond Wu, Masayuki Hisada, Atsushi Kara and Takafumi Hayashi
発表タイトル：Metadata Driven Approaches for Web Security of New-age Computing
発表月日：平成 22 年 7 月

会議名：5th International Conference on Global Security, Safety, and Sustainability, ICGS3’09
発表者：Raymond Wu and Masayuki Hisada
発表タイトル：Static and Dynamic Analysis for Web Security in Generic Format
発表月日：平成 21 年 9 月

会議名：The 2009 International Conference on Internet Computing, ICOMP’09
発表者：Raymond Wu and Masayuki Hisada
発表タイトル：Static Analysis for Web Security in Abstract Syntax Format
発表月日：平成21年7月

国際ワークショップ
ワークショップ名：International Workshop on Wireless and Network Security
発表者：Raymond Wu, Keisuke Seki, Ryusuke Sakamoto and Masayuki Hisada
発表タイトル：Knowledge-base Semantic Gap Analysis for Vulnerability Detection
発表月日：平成22年6月

## ５－２　産業財産権

### ５－２－１　特許出願数

国内出願　２件、　国際出願　１件

### ５－２－２　公開特許一覧

該当なし

### ５－２－３　登録特許一覧

該当なし