Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

効果的な攻撃テストによるWebアプリケーションの脆弱性検出手法

小菅祐史[†1]　河野健二[†1,†2]

Web アプリケーションの脆弱性の多くは設計時や開発時に排除することができる．しかしそれらは開発者によって手作業で行われるため，間違いや見落としが発生しやすい．また，既存の脆弱性検出ツールは，実行時間が長い上，誤検知を多く発生してしまう．本論文では，Web アプリケーションのセキュリティに関する検証をより正確かつ効率的に行うことができるフレームワーク Amberate を提案する．Amberate はそれぞれの Web アプリケーションに適した攻撃を動的に自動生成し，攻撃テストを実行する．そして攻撃後の反応を自動検証することで，脆弱性や設定ミスなどを自動検出する．Amberate に組み込み可能な SQL インジェクション攻撃，クロスサイト・スクリプティング，JavaScript Hijacking に対する脆弱性を検出するプラグインの開発を行い，評価実験で各プラグインの脆弱性検出手法の有効性を示した．

An Effective Audit Testing for Detecting Vulnerabilities in Web Applications

Yuji Kosuga[†1] and Kenji Kono[†1,†2]

Most of security-related configurations and source-code instructions for Web applications are done by developers' hands, which can be tedious and error-prone. Existing vulnerability scanners are helpful to validate the correctness of their security, but they may take a long time to perform a penetration test, and even worse, generate a lot of false positives/negatives. Amberate is designed to automatically verify the security of Web applications, more readily and accurately. It can automatically generate attacks suited for each Web application, and verify the security by analyzing its reaction after sending them. In evaluation experiment to discover vulnerabilities against SQL Injection, Cross-Site Scripting, and JavaScript Hijacking, we found our solution was efficient in comparison with popular web application scanners.

1. はじめに

Web 技術の向上により，Web アプリケーションが広く利用されるようになった．その一方，Web アプリケーションには多くの脆弱性が存在している．米国のセキュリティ調査会社 Cenzic[3] によると，2008 年第 2 四半期のコンピュータ関連の脆弱性において，全体の 73%が Web 関連の脆弱性であると報告している．これらの脆弱性を狙った攻撃が後を絶たない．

脆弱性の多くは設計時や開発時に排除することができる．しかし，Web アプリケーションのセキュリティに関する設定やプログラムの記述は難しく，その多くは開発者によって手作業で記述されている．また，攻撃に対する動作チェックが十分に行われていない．セキュリティに関する設定やプログラムの記述に加え，これらの正しさの検証も手作業で行われるため，間違いや見落としが発生しやすい．そのため Web アプリケーションの開発者は，脆弱性検出ツールを使用して Web アプリケーションの出荷前に脆弱性を根絶させようと試みる．しかし既存のツールでは，テストの実行回数が多い上，誤検知を多く発生してしまう．

さらに近年では，Web2.0 と呼ばれる技術の登場によって，Web アプリケーションのメカニズムはさらに複雑化し，それに伴ってセキュリティ問題も高度化している．そのため，セキュリティの確保はより一層困難になっているのが現状である．

Web アプリケーションにおけるセキュリティ確保を徹底しなくてはならない．そのためには，人手に頼らず，また，従来の脆弱性検出ツールより正解に，かつ簡単に検証を行うことができる脆弱性検出ツールが必要である．

そこで，動的にコンテンツを生成する Web アプリケーションに対し，セキュリティの自動検証をより正確かつ効率的に行うフレームワーク Amberate を提案する．Amberate はそれぞれの Web アプリケーションに適した攻撃を自動生成し，攻撃テストを実行するためシステムである．さらに，攻撃後の反応を自動検証することで，脆弱性や設定ミスなどの自動検出を行う．また，様々な攻撃手法が存在している現状から，より多くの攻撃に対してテストを行うことができるように，脆弱性検出用プラグインを読み込むことによって適宜必要なテストを行う．

Amberate の開発において，脆弱性検出用プラグインの開発も行った．今回対象としたのは，SQL インジェクション攻撃，クロスサイト・スクリプティング，JavaScript Hijacking

†1 慶應大学
Keio University
†2 科学技術振興機構 CREST
CREST, Japan Science and Technology Agency

Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

図 1 Amberate の動作概要

に対する脆弱性を検出するプラグインである．それぞれの攻撃生成や脆弱性検出の手法は，既存の手法では検出できなかった脆弱性を効率よく検知することを目的とした，新規性のある検知手法を基に開発を行った．また，評価実験ではそれらの手法の有効性を確認した．

以下，2 章では Amberate ついて述べる．3 章では SQL インジェクション攻撃の脆弱性検出用プラグインについて，4 章ではクロスサイト・スクリプティングの脆弱性検出用プラグインについて，5 章では JavaScript Hijacking の脆弱性検出用プラグインについて述べる．6 章で実験結果を述べ，7 章で関連研究を述べる．最後に 8 章で本論文をまとめる．

2. Amberate

Amberate は，開発者が Web アプリケーションの出荷前に行う脆弱性検出作業をサポートするツールとして設計しており，既存のツールより，短いテスト時間・回数で多くの脆弱性を検出することを目指している．

2.1 Amberate のアプローチ

今日の Web アプリケーションの多くは，クライアントの入力に応じて動的にコンテンツを生成する．そこで Amberate は，Web アプリケーションへの入力（HTTP リクエストなど）と，入力から生成される出力（HTTP レスポンスなど）を取得し，解析を行う．この入出力の解析によって，Web アプリケーションが入力に対してどのように振る舞うかを知ることができる．ただし，検出を試みる脆弱性によって取得する入出力は異なる．例えば SQL インジェクション攻撃に対する脆弱性を検出する際には，出力として SQL クエリを取得・解析する．このように取得する入出力は，対象とする攻撃によって異なるので，各脆弱性検出手法を述べる際に示す．

取得した入出力を基に，次に挙げる 3 つの処理を行う．また，概要図を図 1 に示した．ただし Amberate のフレームワーク自体は，下記の処理の具体的な処理を行わず，各脆弱性検出用プラグインに処理を委ねる．

(1) 攻撃ターゲットの特定
クライアントの入力や Cookie などの HTTP リクエストに現れるパラメータのうち，攻撃が可能なパラメータ（**攻撃ターゲット**）を正確に特定する．

(2) 攻撃の生成
出力に現れる攻撃ターゲットの構文や文脈に従って，それぞれの Web アプリケーションに適切で強力な攻撃文字列（**攻撃コード**）を生成する．

(3) 脆弱性の検出
攻撃コードを含む HTTP リクエスト（**攻撃リクエスト**）を Web アプリケーションに送り，そのときの出力を検証することによって，脆弱性の検出を高い精度で行う．

また，攻撃によっては攻撃テストを実行せず，HTTP レスポンスの内容を検査することによって脆弱性を検出することができる．そのような攻撃に対しては，無害な入力によって生成された HTTP リクエストと，そのとき発行された出力（HTTP レスポンスなど）を解析することによって脆弱性検出を行う．

以上に示したように，Amberate は攻撃を自動生成し，Web アプリケーションに向けて送信するという仕組みを持つ．つまり，使い方によっては危険なツールになってしまう．そこで，脆弱性検出ツールとしてローカル環境で動作するものとして実装した．つまり，図 1 で示した模擬クライアントは Web アプリケーションの開発者やそれに準ずる正当なデバッガーなどに相当する．

Amberate のフレームワーク自体は，テスト実行用の動作環境なので，脆弱性検出能力は各検出用プラグインの能力に依存する．そこでそれぞれの脆弱性検出用プラグインの検出手法について，以下に述べる．

3. SQL インジェクション攻撃に対する脆弱性検出用プラグイン

本章では，SQL インジェクション攻撃の攻撃手法と一般的な対策手法，既存の脆弱性検出ツールの手法について述べる．その後，本脆弱性検出用プラグインの手法について述べる．

3.1 SQL インジェクション攻撃と対策

SQL インジェクション攻撃とは，データベースと連動して動的にコンテンツを配信する Web アプリケーションに不正に SQL コマンドを実行させる攻撃である．この攻撃によって，Web アプリケーションは情報漏洩や改竄などの被害に遭う危険性がある．例えば，データベースが `users` テーブルに `name` カラムと `password` カラムを持ち，Web アプリケーションはユーザのログイン時に次のプログラムを使用して，SQL クエリを生成すると仮定する．

```
sql = “SELECT * FROM users WHERE name = ’”
    + request.getParameter(name)
```

Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

```
    + "' AND password = '"
    + request.getParameter(password) + "'";
```

ここで，攻撃者が password に "' or '1'='1" を挿入すると，次に示す SQL クエリが生成される．

```
SELECT * FROM users WHERE
    name = 'xxx' AND password = '' or '1'='1'
```

この攻撃ではシングル・クオートが構文を不正に変化させるため，WHERE 節が "or '1'='1'" により常に真と評価される．その結果，攻撃者は users テーブルの全ての情報を取り出すことができてしまう．このように SQL クエリの構文を不正に変化させるような文字列を挿入する攻撃を SQL インジェクション攻撃と言う．Cenzic[3] によると，Web 関連の脆弱性の 34%が SQL インジェクション攻撃に対する脆弱性であると報告している．

SQL インジェクション攻撃を防ぐためには，クライアントから与えられる文字列の正当性を検証する必要がある．この検証処理を**サニタイズ**と呼ぶ．サニタイズを適切に行うことによって，SQL クエリの構文を変化させるような危険な文字を無害化することができる．例えば，シングルクオートにバックスラッシュを付加することによって，データベースでシングルクオートを単なる 1 文字と認識させる方法がある．

サニタイズを適切に行うことによって，SQL インジェクション攻撃を防ぐことができる．しかし，サニタイズの処理を行うコードの挿入やサニタイズ忘れの検査は，開発者によって手作業で行われる．そのため間違いや見落としが発生しやすい．

3.2 既存の脆弱性検出手法

SQL インジェクション攻撃の脆弱性を自動検出する既存のツールは，HTTP リクエストに現れる全ての入力パラメータや Cookie を攻撃ターゲットと認識する．そして攻撃ターゲットにあらかじめ定義された全ての攻撃コードを順次挿入していき，攻撃リクエストの送信後に Web アプリケーションが発行するレスポンスの内容にデータベースでのエラーが含まれると攻撃成功と判断する．この手法では，不適切な攻撃ターゲットにも攻撃コードを挿入したテストを実行する上，あらかじめ定義された全ての攻撃コードを実行する．そのため，多くの時間を要する．また，データベースが返したエラーを Web アプリケーションが隠蔽すると脆弱性検出は不可能となる．

3.3 提 案 手 法

本手法では HTTP リクエストから生成される SQL クエリの取得・解析を行う．図 2 に SQL インジェクション攻撃に対する脆弱性検出用プラグインの動作概要を示した．図では Web アプリケーションをアプリケーション部分とデータベース部分に分けて示している．ここでデータベースをテスト用ダミーデータベースと示しているが，これは攻撃テストではデータベースの内容を変化させてしまう可能性があるため，テスト用に用意したダミーのデータベースを使用することを示している．Amberate は HTTP プロキシとして働くことで，HTTP リクエストを取得する．また，プラグインが SQL プロキシを起動させ，SQL パケットを取得する．そして，これらの取得したパケットを解析し，次の 3 つの処理を行う．

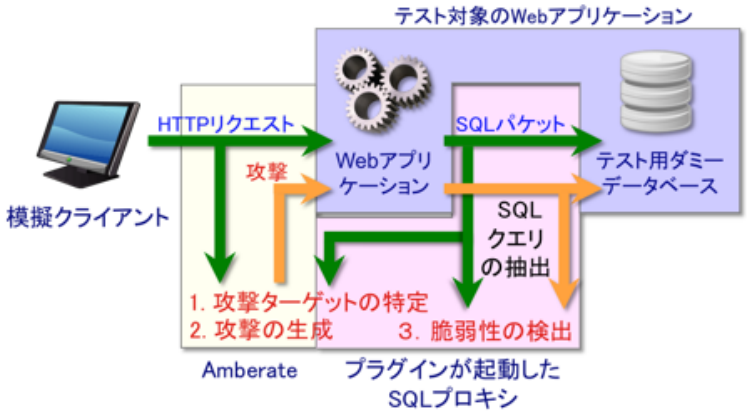

図 2 SQL インジェクション攻撃に対する脆弱性検出用プラグインの動作概要

(1) 攻撃ターゲットの特定
クライアントの入力や Cookie などの HTTP リクエストに現れるパラメータのうち，SQL クエリに現れるパラメータを攻撃ターゲットとする．例えば，Web アプリケーションが 'name=xxx&action=yyy' のように 2 つのパラメータを持つ HTTP リクエストを受け取り，次の SQL クエリを発行すると仮定する．

```
select * from users where username='xxx'
```

このとき 'xxx' は SQL クエリに現れるため攻撃ターゲットであるが，'yyy' は攻撃ターゲットではない．

(2) 攻撃の生成
SQL クエリの構文を解析し，攻撃ターゲットの文脈を得る．この文脈は，汎用 SQL 文法を基に 95 種類に分類した．そして攻撃ターゲットの文脈に従って，動的に攻撃コードを生成する．例えば，String 型として現れる以下の攻撃ターゲットには '')) or 1=1--' を挿入する．

```
SELECT * FROM users WHERE
            (id=999 AND (name='ø'))
```

(ø: 攻撃ターゲット)

Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

この例のように，文脈情報を得ることによって適切な数の括弧を持つ攻撃コードを生成することができる．また，Select 文の item として現れる以下の攻撃ターゲットには‘ from yyy--’を挿入する．

SELECT id, pass, ø FROM users WHERE name='xxx' (ø: 攻撃ターゲット)

また，攻撃には一度に 1 つの攻撃ターゲットを攻撃する単一攻撃と，一度に複数の攻撃ターゲットを攻撃する組合せ攻撃がある．攻撃コードは，攻撃コード生成規則というルールに基づいて動的に生成する．攻撃コード生成規則には，生成される攻撃コードの構造が定義されている．攻撃コード生成規則は，SQL インジェクション攻撃への攻撃コードをすべて網羅することができるように，以下に示すように 4 つの成分で構成される．

(文字列 1，クオートの有無，括弧を使用の有無，文字列 2)

ただし，クオートを使用する場合はシングルクオートとダブルクオートの使用に関する正当性検査を行う．また，括弧を使用する場合は，括弧数の計算も含む．この 4 つの成分をつなぎ合わせることによって，攻撃コードを生成する．例えば String 型への攻撃コード生成規則は次のように表現できる．

```
(λ | ϵ,
 true,
 true,
 or '1'='1 | or "1"="1 | or 1=1-- | or 1=1;-- | or 1=1/*)
```

ただし，(λ) はユーザの入力，(ϵ) は空白文字を示している．攻撃コード生成規則は XML で記述されており，新しいルールの追加を容易に行うことができる．

(3) 脆弱性の検出

無害なリクエストから生成された SQL クエリと，攻撃リクエストから生成された SQL クエリの構文の比較を行う．Web アプリケーションが適切な対策をとっていると，攻撃から生成された SQL クエリの構文は，安全なリクエストから生成された SQL クエリと同じ構文になる．しかし，適切に対策が行われていないと，構文は異なる．この手法は SQLGuard[2] で提案された手法で，本プラグインでもこの手法を用いて脆弱性の検知を行う．

以上に示した手法で攻撃テストを行うのだが，脆弱性を“全自動”で検出を行うには限界がある．その理由は，今や誰でも Web アプリケーションを作成することができるため，様々な仕様の Web アプリケーションが存在しており，攻撃テストに有効な情報を自動認識できない場合があるからである．例えば，Web ページにおいてパスワード入力欄とその確認用の入力欄など，複数の入力欄に同じ値を入力しなくてはいけない場合がある．このような場合，Amberate のような脆弱性検出ツールでは，外部から Web アプリケーションの情報を得ようとするため，どの入力欄に同じ値を入れるべきか自動認識することはできない．また，入力として受理できる文字列に，データベース側で長さ制限を設けている場合がある．しかし，この長さ制限についても同様の理由で自動認識することはできない．このような情報を認識できないと，正確な脆弱性検出を行うことができない場合がある．例えば，攻撃テストにおいて，不適切な攻撃を送り，Web アプリケーションにリジェクトされると，エラーページなどに誘導される．この誘導後のページが予期していなかった新しい SQL クエリを発行すると，Amberate はその SQL クエリを，攻撃が成功した結果生成された SQL クエリであると判断してしまい，誤検知となる．

そこで本プラグインでは，個々の Web アプリケーション固有の仕様を認識することができる仕組みを用意した．認識することができる情報は以下の 4 つである．

- 同じ値をもつパラメータの情報
  パスワード入力欄やその確認用の入力欄などのように，同じ値を持つ複数の入力欄に関する情報である．この情報によって指定した入力欄が攻撃ターゲットとなるとき，複数の入力欄に同じ攻撃コードを用いたテストを行うことができる．
- 攻撃コードの長さに関する情報
  データベースで受理することのできる文字列の長さに制限がある場合，この情報に従って必要以上に長い文字数の攻撃コードの生成を避けることができる．
- SQL クエリの構文の変化を許可する情報
  Web アプリケーションによっては，SQL クエリの構文の変化を許可する場合がある．そこでこの情報を付加することによって，構文の変化を許可するような柔軟性を持ったテストを行うことができる．
- テスト不要なパラメータの情報
  必要に応じてテストを実行することができるように，テスト不要なパラメータの情報を与えることができる．また，攻撃ターゲットの特定にミスが生じた場合でも，この機能を用いることによって修正を加えることができる．

Web アプリケーション固有の情報は，Amberate のユーザに入力してもらう．Amberate はローカル環境で動作することによって，脆弱性検出ツールとして正当なユーザに使用されることを想定している．そのため Amberate のユーザとは，Web アプリケーションの開発者となり，Web アプリケーション固有の情報を入力することができる．このように Web アプリケーション固有の情報を認識することによって，より適切に攻撃ターゲットを特定し，より強力な攻撃コードを生成し，さらにより正確な脆弱性検出を行うことができる．

Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

4. クロスサイト・スクリプティングに対する脆弱性検出用プラグイン

本章では，クロスサイト・スクリプティングの攻撃手法と一般的な対策手法，既存の脆弱性検出ツールの手法について述べる．その後，本脆弱性検出用プラグインの手法について述べる．

4.1 クロスサイト・スクリプティングと対策

クロスサイト・スクリプティングとは，悪意あるスクリプトをクライアントのブラウザで実行させることによって，そのクライアントの個人情報を盗み取ることができる攻撃である．例えば，攻撃者が脆弱性のある Web アプリケーションに攻撃スクリプトを保存させ，クライアントのリクエストに対するレスポンスがこの攻撃スクリプトを含む場合，クライアントのブラウザで読み込まれる際に，Cookie などの個人情報を外部に送信することができてしまう．Cenzic[3] によると，Web 関連の脆弱性の 23%がクロスサイト・スクリプティングに対する脆弱性であると報告している．

クロスサイト・スクリプティングは，以下に示す 3 種類に分類することができる．

(1) DOM-based XSS
リクエストが Web アプリケーションに送られることなく，JavaScript などのプログラムによってクライアント側のみで実行可能な攻撃．

(2) Non-Persistent XSS
攻撃用の URL やページを経由して攻撃コードを挿入する攻撃．

(3) Persistent XSS
攻撃コードが Web アプリケーションに保存され，クライアントへのレスポンスがこの保存された攻撃コードを含む場合，ブラウザで読み込まれる際に実行される攻撃．

また，上記の攻撃にはブラウザの文字列自動認識機能を用いた攻撃手法も存在する．

クロスサイト・スクリプティングへの対策としては，SQL インジェクション攻撃に対する対策と同じようにサニタイズが有効である．クロスサイト・スクリプティングの場合は，ブラウザでスクリプトとして意味を持つ文字列が攻撃者によって挿入された場合でも，それらを安全な文字列に置換することによって攻撃を防ぐことができる．例えば，HTML タグを形成する比較演算子 “<” や “>” を HTML 上で単なる文字として扱われるように “<” や “>” に置換する方法がある．

4.2 既存の脆弱性検出手法

既存の手法では，全ての入力パラメータや Cookie を攻撃ターゲットとして認識する．そして攻撃ターゲットにあらかじめ定義された攻撃コード全てを順次挿入し，攻撃後に Web アプリケーションが発行するレスポンスの内容に，挿入した攻撃コードが存在するか，文字列検索を行う．検索の結果，攻撃コードがサニタイズされていない状態で存在すると攻撃成功と判断する．

この手法では，実際に攻撃ターゲットとして相応しくない箇所にも攻撃を実行する上，あらかじめ定義された全ての攻撃コードを実行するので，多くの時間を要する．また，脆弱性検出手法も，HTML タグを単なる文字列として出力することができる “<PLAINTEXT>” や “<XMP>” タグ内やタグの属性値として攻撃コードが現れた場合でも攻撃成功と判断してしまうため，誤検知となる．さらに，単純な文字列検索では少しでも文字列が変化しているだけでも脆弱性の検出ができない．

4.3 提 案 手 法

本手法では，攻撃リクエスト送信後，攻撃コードが実際に実行可能な状態で存在するか，HTTP レスポンス内の HTML と JavaScript の構文解析を行うことによって検証する．以下に 4.1 章で示したクロスサイト・スクリプティングの 3 つの手法に対する脆弱性検出手法を順に示す．

(1) DOM-based XSS 対策
攻撃コードを挿入し，JavaScript インタプリタを用いてレスポンス内のプログラムを実行する．攻撃コードが実行されると脆弱性と判断する．ただし，今回は実装は行っていない．

(2) Non-Persistent XSS 対策
クライントの入力や Cookie などのパラメータが HTTP レスポンス内に現れるとき，それらを攻撃ターゲットとする．攻撃コードは HTTP レスポンス内の攻撃ターゲットの構文と文脈に応じて動的に生成する．攻撃後の HTTP レスポンス内の HTML と JavaScript の構文解析によって，攻撃コードが実際に実行されるか検証を行う．

(3) Persistent XSS 対策
無害な入力を記憶し，後に訪れるレスポンスページでこの入力が現れるとき，この入力を攻撃ターゲットとする．攻撃ターゲットに攻撃コードを挿入し，攻撃リクエストを Web アプリケーションに送った後，ターゲットのレスポンスページに訪れたときと同じ経路となるようにページ遷移をする．入力が現れる予定のページにたどり着くと，攻撃コードが実行可能な状態であるか HTTP レスポンス内の HTML と JavaScript の構文解析を行う．

また，ブラウザによる文字コード自動認識機能を悪用した攻撃に対しては，HTML に文字コードが正しく記述されているか，HTML を解析することによって検証を行う．

Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

5. JavaScript Hijacking に対する脆弱性検出用プラグイン

本章では，JavaScript Hijacking の攻撃手法と一般的な対策手法，既存の脆弱性検出ツールの手法について述べる．その後，本脆弱性検出用プラグインの手法について述べる．

5.1 JavaScript Hijacking と対策

JavaScript Hijacking とは，Web アプリケーションが JSON[5] 形式のデータを用いて Ajax 通信を行うとき，この通信内容を奪い取ることができる攻撃である．この攻撃は JavaScript のオブジェクトのオーバーライドを悪用した攻撃である．Firefox ver.2 以前のブラウザで動作する．

JavaScript Hijacking の脆弱性が存在する原因の 1 つとして，JSON 形式のデータに，テキストデータをスクリプトとして返す `eval()` 関数を用いることが挙げられる．これに対し，JSON データの先頭に “`while(1);`” など，スクリプトとして認識された場合，後方に存在するデータの取得を不可能にする文字列（**安全化文字列**）を挿入する対策が知られている．このとき正当な Web アプリケーションはテキストデータとして受け取った Ajax 通信の内容から安全化文字列を読み飛ばすことによって正しくデータを扱うことができる．

5.2 既存の脆弱性検出手法

これまでに，Google の ratproxy[14] が JavaScript Hijacking に対する脆弱性を検出する手法を実装している．その手法は，あらかじめ定義した安全化文字列が JSON データの先頭に存在するか，単純な文字列検索を行うものである．しかし ratproxy の手法では，定義されていない文字列の検出を行うことができないため，誤検知を発生する場合がある．

5.3 提 案 手 法

本プラグインでは，JSON データに正しく安全化文字列が記述されているか，JSON と JavaScript の構文解析を用いて検証を行う．この解析によって，単純な文字列検索では検出することができなかった安全化文字列の検出も行うことができるため，既存の手法で存在していた誤検知を防ぐことができる．

また本手法では，脆弱性を検出すると攻撃スクリプトを自動生成し，攻撃者を模したサーバにロードする．このとき，Amberate のユーザにこの攻撃者を模したサーバにアクセスしてもらい，実際にどのような情報を取得できるかをユーザに指摘する．ただし，JavaScript Hijacking によって取得できる情報が，公開情報であるのか，機密情報なのかは各 Web アプリケーションに依存する．そのため，この機能を利用して取得された情報を Web アプリケーションの開発者が確認することで重大な脆弱性であるかどうかを判断できるように設計した．

表 1 SQL インジェクション攻撃用プラグインの実験結果

Web アプリケーション名	Amberate			Paros		
	攻撃数	正検知数	誤検知数	攻撃数	正検知数	誤検知数
A	214	21	0	362	5	2
B	706	26	0	4802	0	7
C	1080	44	5	5477	0	20
D	276	8	2	1698	0	20
E	498	16	0	1210	0	6
F	290	9	0	1924	0	11
合計	3064	124	7	15473	5	66

6. 実　　験

Amberate の脆弱性検出能力は，各脆弱性検出用プラグインの手法に依存している．そのため，各プラグインが対象としている攻撃に対し，評価実験を行った．

6.1 SQL インジェクション攻撃に対する脆弱性検出用プラグインの実験

実験では 6 つの Web アプリケーションに対して攻撃テストを行った．これらの Web アプリケーションは，実際にフリーで公開されており，テストではそれらをダウンロードし，ローカル環境でテストを行った．また，これらのアプリケーションが実際に改変され，使用されていることを確認した．ただし，評価実験において未知の脆弱性を検出したため，公的機関に脆弱性の届け出を行っている．本論文執筆時点では公表許可が未だ降りていないため，アプリケーション名は公表しない．

また，実験において提案手法と従来の手法の比較を行うため，Paros[4] との比較を行った．Paros を用いた理由は，Amberate と同じように未知の脆弱性を検出でき，脆弱性検出能力は高く，評価が高いからである．尚，Paros を選出するにあたって，Security-Hacks.com[12] で紹介されている 15 件のツールと，Insecure.Org[8] で紹介されている 10 件のツール，計 25 件のツールの調査を行った．Paros は Insecure.Org の Web アプリケーション脆弱性スキャナランキングで 2 位であった．Paros は HTTP リクエストと HTTP レスポンスを取得・解析し、攻撃テストを行う。この手法は、Paros 以外の他のツールでも用いている手法である。

実験結果を表 1 に示した．表中の Amberate と Paros における攻撃数とは，実際にテスト対象の Web アプリケーションに攻撃を送った回数である．正検知数とは，検出した脆弱性の数であり，テスト後に実際に攻撃が成功するか手作業で確認を行った．誤検知数とは，攻撃が成功しないことを確認した数である．表 1 の結果からは，Amberate の方が Paros より，少ない攻撃数で，多くの脆弱性を検出し，誤検知が少ないことがわかった．

Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

表 2 SQL インジェクション攻撃に対する実験結果の脆弱性種別による分類

脆弱性種別	Amberate			Paros		
	攻撃数	正検知数	誤検知数	攻撃数	正検知数	誤検知数
単一攻撃	2844	13	0	15473	5	66
組み合わせ攻撃	220	111	7	0	0	0
合計	3064	124	7	15473	5	66

表 3 クロスサイト・スクリプティング用プラグインの実験結果

Web アプリケーション名	Amberate			Paros		
	攻撃回数	攻撃成功数	誤検知数	攻撃回数	攻撃成功数	誤検知数
WebGoat	347	13	0	156	11	2
HacmeBank	48	2	0	112	0	2
合計	393	15	0	268	11	4

テスト結果を単一攻撃と組み合わせ攻撃に分類し，表 2 に示した．その結果からは正検知数の多くは組み合わせ攻撃であることがわかった．Amberate は組み合わせ攻撃を実行できるため，これらの脆弱性を検出することができたが，Paros は組み合わせ攻撃を実行しないため，検出できない．Paros の手法では SQL クエリの知識がないため，正確な攻撃ターゲットの情報が得られない．このとき組み合わせ攻撃に対する脆弱性を検出するためには，莫大な数の組み合わせの攻撃を実行する必要がある．この状況を避けるため，組み合わせ攻撃に対する脆弱性検出を試みないと考えられる．

6.2 クロスサイト・スクリプティングに対する脆弱性検出用プラグインの実験

実験では Web セキュリティ学習用としてフリーで公開されている WebGoat[11] と HacmeBank[10] を対象にテストを行った．比較には，SQL インジェクション攻撃に対する脆弱性検出用プラグインの評価実験と同様に Paros を用いた．実験結果を表 3 に示した．

この結果から，Amberate の方が Paros より多くの脆弱性を検出し，誤検知が少ないことがわかった．しかし攻撃回数は多い．この原因は Paros では検出することができないような脆弱性を Amberate は検出しようとするためである．例えば，既知のブラウザのバグに対する安全性の検証を行うための攻撃テストを実行する．実際にテストでは，<xss> という実在しないタグが HTML の要素として認識されてしまうか判断する．このような実在しないタグの検証を行う理由は，実際に <script/xss> というタグが script タグとして認識されてしまうことによる攻撃が存在するからである．このような脆弱性を検出するため，Amberate は Paros より多くの攻撃を実行する．

クロスサイト・スクリプティングの脆弱性を攻撃種類ごとに分類し，表 4 に示した．この結果から，テストに使用した Web セキュリティ学習用アプリケーションの DOM-based 以外のすべてのステージをクリアしていることがわかった．また Paros では対応していない Persistent 攻撃に Amberate は対応している．Paros の誤検知を分析すると，Paros は単純な文字列検索によって，攻撃文字列が実行されない領域に現れるときでも，攻撃成功と判断してしまう．Amberate ではそれらを正しく認識することができる．また，DOM-based に関しては，Amberate は対策手法を今後実装する予定である．

表 4 クロスサイト・スクリプティングに対する実験結果の脆弱性種別による分類

脆弱性種別	Amberate			Paros		
	攻撃数	正検知数	誤検知数	攻撃数	正検知数	誤検知数
Non-Persistent	335	12	0	132	11	2
Persistent	58	3	0	136	0	2
DOM-based	N/A			N/A		
合計	393	15	0	268	11	4

表 5 JavaScript Hijacking に対する実験結果

安全化文字列	Amberate	Paros
while(1);	√	√
while (1);	√	√
while (1):	√	×
while(0==0);	√	×

6.3 JavaScript Hijacking に対する脆弱性検出用プラグインの実験

JavaScript Hijacking は安全化文字列を挿入することによって防ぐことができるため，本実験では安全化文字列を正しく認識することができるか調べた．実験では，Google の ratproxy[14] と検出能力の比較を行った．ratproxy は，単純な文字列検索を行って，安全化文字列が Ajax 通信の結果得られたデータに含まれているか検査を行う．表 5 に実験結果を示した．表中のチェックマークは，安全化文字列を正しく認識できたことを示し，×マークはその逆である．

結果からは，Amberate の方が正確な検証を行っていることがわかる．これは ratproxy 手法だと，あらかじめ定義されていない安全化文字列の検出を行うことができないためである．それに対し Amberate は，Ajax 通信内容の構文解析を正しく行うことによって，while と (1) の間にいくつスペースが入ろうと関係なく，正しく検証を行うことができる．

7. 関 連 研 究

WAVES[6] や SecuBat[7] は攻撃リクエストに対する HTTP レスポンスを解析し，脆弱性

Vol.2009-ARC-183 No.11
Vol.2009-OS-111 No.11
2009/4/22

の検出を行う．動的解析を行う点では Amberate に近い手法だが，評価実験での比較に用いた Paros と同様に HTTP 通信の情報のみ用いて攻撃テストを行うため，SQL インジェクション攻撃の組み合わせ攻撃やクロスサイト・スクリプティングの Persistent XSS のような近年の巧妙な攻撃手法に対する脆弱性は検出することができない．

QED[9] はモデル検査を行って SQL インジェクション攻撃とクロスサイト・スクリプティングに対する脆弱性を検出する．セッション情報を保持することによって，Persistent XSS の検出を行うことができる．しかしモデル検査は静的解析を用いて行うため，動作環境は検査対象の Web アプリケーションの実装言語に依存する．それに対し Amberate は動的解析を行うため，実装言語に依存しない．

SQLCheck[13] は静的解析と動的解析を組み合わせた手法である．まず静的解析時に正しい SQL クエリのモデルを定義する．そして動的解析時に実際に生成された SQL クエリとモデルを比較し，一致する場合のみ，データベースへの発行を許可する．XSS-GUARD[1] はクロスサイト・スクリプティングをターゲットにサーバサイドで危険なレスポンスの生成を監視する．これらの手法は Web アプリケーション運用時に使用されるため，システムにオーバーヘッドを与えてしまう．また，誤検知は業務や運用システムに悪影響を及ぼす恐れがある．それに対し，Amberate は Web アプリケーション開発時に使用されることを想定しているため，システム運用時に与える悪影響がない．

8. ま と め

Web アプリケーションには多くの脆弱性が存在している．しかし，既存の手法では脆弱性を正確かつ効率的に検出するには不十分である．本論文では，Web アプリケーションの脆弱性の自動検出を行うフレームワーク Amberate を提案した．Amberate はそれぞれの Web アプリケーションに適した攻撃を動的に自動生成し，攻撃テストを実行する．そして攻撃後の反応を自動検証することで，脆弱性や設定ミスなどを自動検出する．Amberate に組み込み可能な SQL インジェクション攻撃，クロスサイト・スクリプティング，JavaScript Hijacking に対する脆弱性を検出するプラグインの開発を行い，評価実験で各プラグインの脆弱性検出手法の有効性を示した．

謝　　辞

Amberate の開発は，独立行政法人 情報処理推進機構 人材発掘・育成事業 2008 年度上期未踏ユースの支援を受けた．また，本研究の一部は，日本学術振興会の支援による．

参 考 文 献

1) Bisht, P. and Venkatakrishnan, V.N.: XSS-GUARD: Precise Dynamic Detection of Cross-Site Scripting Attacks, *Poceedings of the 5th GI International Conference on Detection of Intrusions & Malware, and Vulnerability Assesment (DIMVA '08)*, pp.23–43 (2008).
2) Buehrer, G., Weide, B.W. and Sivilotti, P. A.G.: Using Parse Tree Validation to Prevent SQL Injection Attacks, *Proceedings of the 5th International Workshop on Software Engineering and Middleware (SEM '05)*, pp.106–113 (2005).
3) Cenzic, Inc.: Q2 2008 Trends Report on Web Security, http://www.cenzic.com/downloads/Cenzic_AppSecTrends_Q2_2008.pdf.
4) Chinotec Technologies Company: Paros, http://www.parosproxy.org/.
5) Crockford, D.: JavaScript Object Notation (JSON), RFC 4627.
6) Huang, Y.-W., Huang, S.-K., Lin, T.-P. and Tsai, C.-H.: Web Application Security Assessment by Fault Injection and Behavior Monitoring, *Proceedings of the 12th international conference on World Wide Web (WWW '03)*, pp.148–159 (2003).
7) Kals, S., Kirda, E., Kruegel, C. and Jovanovic, N.: SecuBat: A Web Vulnerability Scanner, *Proceedings of the 15th international conference on World Wide Web (WWW '06)*, pp.247–256 (2006).
8) Lyon, G.: Top 10 Web Vulnerability Scanners, http://sectools.org/web-scanners.html (2006).
9) Martin, M. and Lam, M.S.: Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking, *Proceedings of the 17th Conference on USENIX Security Symposium (USENIX Security '08)*, pp.31–43 (2008).
10) McAfee, Inc: Hacme Bank v2.0, http://www.foundstone.com/us/resources/proddesc/hacmebank.htm.
11) OWASP: WebGoat Project, http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project.
12) Security-Hacks.com: Top 15 free SQL Injection Scanners, http://www.security-hacks.com/2007/05/18/top-15-free-sql-injection-scanners (2007).
13) Su, Z. and Wassermann, G.: The Essence of Command Injection Attacks in Web Applications, *Proceedings of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL '06)*, pp.372–382 (2006).
14) Zalewski, M.: ratproxy, http://code.google.com/p/ratproxy/.