Web 2.0におけるセキュリティー
― セキュアなWeb 2.0環境構築のために ―

急速に変化し続ける市場やビジネス・モデルに柔軟に対応できるIT（情報技術）基盤やアプリケーションへの要請の答えの一つとして、Ajax（Asynchronous JavaScript + XML）やマッシュアップに代表されるWeb 2.0技術を用いた新しいWebアプリケーションやサービスが、現在注目を集めています。必要な機能を、既存のコンポーネントやAPI（Application Program Interface）を結合（マッシュアップ）することで、短い開発時間で、使いやすいユーザー・インターフェースを持ったWebアプリケーションを提供するWeb 2.0技術は、個人ユーザーのみならず企業にとっても、とても魅力的なものとなっています。

しかし、ここで重要になってくるのが、セキュリティーの問題です。Web 2.0技術は「簡単で使いやすい」が身上ですが、それは同時にセキュリティー上の脅威に対する脆弱（ぜいじゃく）性の問題をはらんでいます。本解説では、Web 2.0技術を用いて構築されたWebアプリケーションにおけるセキュリティー上の問題とその対策、IBMのこの分野での取り組みなどについて紹介します。

Article 3

Web 2.0 Security
–Towards establishing a secure Web 2.0 environment–

Web 2.0 Technology including Ajax and Mashup is a new trend for Web Applications. Where rapid adaptation is required in fast moving markets and business models, the concept of Web 2.0 supporting the composition (mashup) of components or APIs is now gaining momentum at enterprises.

Security plays an important role in realizing an enterprise-level quality Web 2.0 environment. In this article, we introduce some security problems of Web 2.0 applications and their countermeasures, and describe IBM's activities in this area.

❶ はじめに

Web 2.0技術の特長は、Ajaxやマッシュアップなど柔軟で生産性の高いプログラミング手法や使い勝手の良いユーザー・インターフェースにあります。必要な機能を、既存のコンポーネントやAPIを結合（マッシュアップ）することで短期間に構築する状況依存型アプリケーション（Situational Applications）や、Wiki、ブログ、SNSといったソーシャル・コンピューティング系のアプリケーションが、このWeb 2.0技術を用いて構築されるようになってきました。

しかし、Web 2.0技術が、エンタープライズ・レベルの使用に耐えるためには、信頼性やセキュリティーといった基盤技術が必要不可欠です。一方で、Web 2.0の良さを損なわないためには、開発者やユーザーに対して、セキュリティーに関する細かな知識や作業なしに、安全性の高いWeb 2.0アプリケーションを開発し、使ってもらうための仕組みが必要となってきます。

Web 2.0技術の特徴と、典型的なアプリケーション、さらに、それに付随するセキュリティー上の問題をまとめたものが**図1**です。

SNS（Social Networking Service）、ブログ、Wikiといった、ユーザーがデータを作成し、ほかのユーザーとそれを共有するというスタイルのアプリケーション（Social Computing Software）が増えてきています。それとともに、Webアプリケーションに対する攻撃も増加し、大きな問題となっています。例えば、クロス・サイト・スクリプティング（Cross Site Scripting、CSSまたは

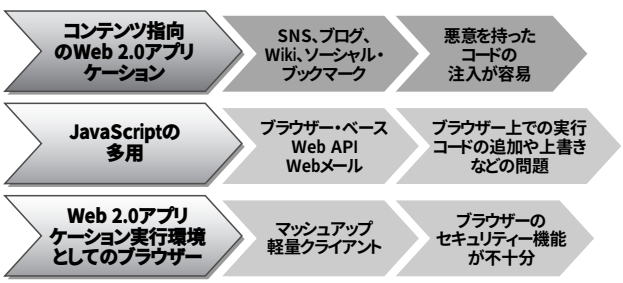


図1. Web 2.0環境におけるセキュリティー上の問題

XSSと略される）や、クロス・サイト・リクエスト・フォージェリー（Cross Site Request Forgery、CSRFまたはXSRFと略される）といった攻撃手法がよく知られており、さまざまなバリエーションが発見されています。

動的なHTML（Dynamic HTML、DHTML）文書の多くはWebブラウザー上で解釈実行されるスクリプト言語であるJavaScript言語で書かれたコードやライブラリーを含んでいます。JavaScriptを用いることで、従来の静的なHTML文書では容易になし得なかったようなユーザー・インターフェースが実現されますが、その柔軟さは、しばしばセキュリティー上の脅威となります。また、Web 2.0アプリケーションの多くは、Webブラウザーをクライアント環境として用います。Webブラウザーは、HTML文書の閲覧環境から、複数のコンポーネントやWeb APIを組み合わせたマッシュアップの実行環境として進化を遂げようとしています。しかし、セキュリティーの観点からは、現行のWebブラウザーのセキュリティー・モデルはあまりにシンプルです。

このような背景を踏まえながら、本解説では、Web 2.0アプリケーションに関連するセキュリティー上の脅威や、現時点で取り得る対策について述べます。また、IBMの外部コミュニティーへの貢献や、基礎研究部門で行われているWeb 2.0セキュリティー・プロジェクトについて簡単に紹介します。

❷ Web 2.0環境におけるセキュリティー上の脅威

それでは、Web 2.0アプリケーションに対する典型的な攻撃手法を幾つか紹介します。

2.1. クロス・サイト・スクリプティング（XSS）

XSSは、動的にHTML文書を生成するWebアプリケーションにおいて、攻撃意図を持ったスクリプト・コードを外部から与えることで、HTML文書内に挿入させ、そのコードを実行させる攻撃手法です。従来のWebアプリケーションでもこの問題は生じますが、複数ユーザーからの入力が頻繁に発生するWeb 2.0アプリケーションでは、特に深刻な問題として知られています。

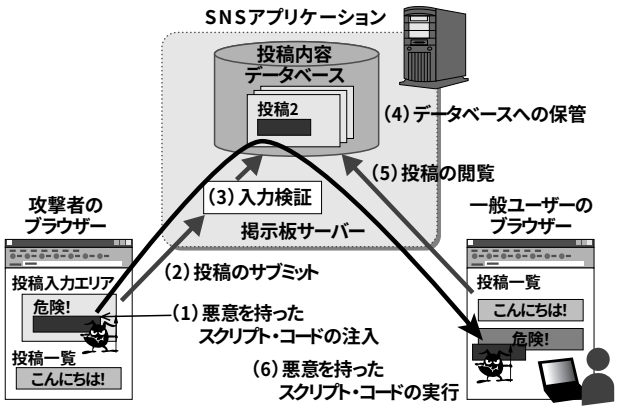


図2. クロス・サイト・スクリプティングの例

それでは、XSSが具体的にどのようなものか、SNSアプリケーションを例に説明します（**図2**）。このアプリケーションでは、ユーザーからの投稿を入力する画面と、これまでの投稿を閲覧する機能があるとします。処理の流れは以下になります。

(1) 攻撃者は、投稿入力画面から、文章とともに悪意を持ったスクリプト・コードを入力します。
(2) 入力した内容はSNSサーバーに送られます。
(3) このようなアプリケーションの多くは、入力された内容の中に、スクリプトが注入されていないかを検証する機能があります。しかし攻撃者は、検証メカニズムの不備を突いたり、文字列に細工したりすることで、しばしばこの処理を巧妙にすり抜けることができます。
(4) 検証をパスした悪意を持ったスクリプトがサーバー上のデータベースに格納されます。
(5) その後で、一般ユーザーが、SNSサーバーに対してこれまでの投稿の閲覧リクエストを送ります。
(6) 悪意を持ったスクリプトを含む投稿がWebブラウザーにロードされ、実行されます。

ここで注意したいのは、XSSでは、悪意を持ったスクリプトが、信頼されたサーバー（この場合はSNSサーバー）から送られてくることです。実行されるスクリプト・コードそのものは表示されないので、一般ユーザーは、信頼するWebサイトを閲覧している過程で、何も知らないうちに悪意を持ったスクリプトの実行を許してしまいます。現行の技術で、どのようにこれを防ぐかは、後章で説明します。

2.2. クロス・サイト・リクエスト・フォージェリー(CSRF)

皆さんは、知らないうちにショッピング・サイトで商品を購入したことになっていて多額の請求書が届いたという経験をしたことはありませんか? CSRFは比較的最近になって注目を浴びている新しいタイプの攻撃です。XSSと異なり、CSRFでは、攻撃のために悪意のあるスクリプトをWebサイトに挿入する必要がありません。

CSRFはWebアプリケーションの認証クッキーの持つ脆弱(ぜいじゃく)性を悪用します。一般的に認証を必要とするWebサイトは、ユーザーがログインした後に認証クッキーをWebブラウザーに返します。Webブラウザーは一定の有効期間中この認証クッキーを保存しているので、毎回パスワードを入力しなくても、サーバー側で認証済みのユーザーかどうかを判別することができます。

しかしこの方式の欠点は、ユーザーが意識していなくても、WebブラウザーがHTTP(Hypertext Transfer Protocol)リクエストを発行するたびに自動的に認証クッキーが送付されてしまうことです。例えば、ユーザーがオンライン・ショップのサイトにログインし、そのクッキーの有効期限が切れる前に何らかのきっかけで悪意のあるWebサイトを表示したとしましょう。そのサイトに含まれるスクリプトから、オンライン・ショップに対して不正なコマンド(例えば購入リクエスト)を発行することが可能になります。この場合、サーバー側ではログインしているユーザーからの正規のコマンドとの見分けが付きません。**図3**にCSRFによる攻撃例を示します。

CSRFを防ぐための方法は幾つかあります。

(1)サーバーでHTTPのReferrerヘッダーをチェックする。しかし、プライバシー保護のためにReferrerヘッダーを送らないようなユーザー設定のあるWebブラウザーもあるため、常にこの方法でチェックできるとは限りません。

(2)HTMLフォームのhiddenフィールドなどを利用し、第2の認証トークンをHTMLページ内から直接送ってサーバー側でチェックするようにする。

(3)購入リクエストなど重要な場面で、もう一度ユーザーにパスワードを入力させる。

CSRFはWebでコマンドを受け付けるインターフェースを持つほとんどのアプリケーションに適用可能な攻撃です。またCSRFによって、ファイアーウォールの外からイントラネット内のサーバーに対しての攻撃が可能になるため、適切な対策を取ることは今後ますます重要になっていくと思われます。

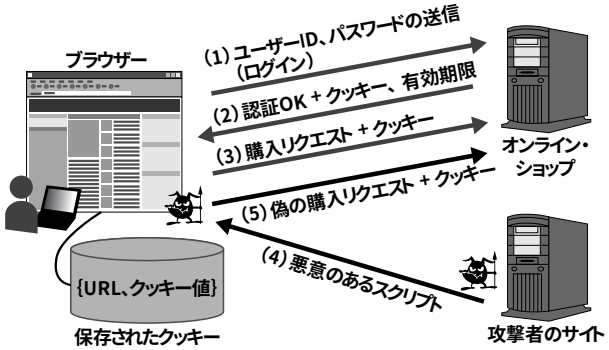


図3. クロス・サイト・リクエスト・フォージェリーの例

2.3. コンテンツの信頼性

Web 2.0の特徴の一つは、個人が生成する情報とその集合(集合知)が大きな価値を生み出すことです。従来は、企業・団体や権威付けられた一握りの個人によって発信されていた情報をしのぐような質と量の情報が、(草の根の)個人によって作られるようになりました。WikiPediaが良い例です。

しかし一方で、個人が書いた誹謗(ひぼう)中傷などのネガティブな内容が、大きな問題を生んでしまうという問題もあります。2ちゃんねるなどの巨大匿名掲示板では、心ない個人が書き込んだ情報を、不特定多数のユーザーが支持し、コメントを付加していくことで、特定個人に対する大きな脅威となるケースが幾つも見られます。

企業における掲示板やSNSの使用においては、多くの場合、ログイン時にユーザー認証が行われるため、誹謗(ひぼう)中傷などの書き込みは大きな問題とはならないと思いますが、機密情報や個人情報を不注意からつい書き込んでしまうといった危険性があります。これらの危険性に対し、言語処理技術を用いて、文書内容をスキャンし、不適切な語を除去するための研究開発が、IBM基礎研究所でも行われています。

2.4. その他の問題

Web 2.0アプリケーションは非同期通信を行うため、従来のWebアプリケーションとは異なる振る舞い

を取る点に注意が必要です。例えば従来のWebアプリケーションではユーザーがフォームにデータを入力した後、明示的に「送信」ボタンを押すまではデータは送られませんでした。しかしWeb 2.0アプリケーションの場合は、ユーザーが1文字ずつ入力するごとにそのキーボード・イベントを拾ってサーバーに送ることが可能になります。最近、コンピューターに詳しくないユーザーがWebブラウザーを使ってショッピングを楽しむことが増えてきました。このようなユーザーは、過去の経験から「送信ボタンを押すまではデータは送られることはない」と思っていることが多いのですが、Web 2.0アプリケーションを用いる際にはこのような思い込みは危険です。ユーザーのプライバシー保護にもっと注意を払う必要があります。

Web 2.0アプリケーションでは、XMLデータより軽量なJSON（JavaScript Object Notation）[1]形式でデータをやり取りすることがありますが、このJSONの実装にも注意が必要です。JSONはJavaScript言語のサブセットであるため、多くのAjaxライブラリーではeval()関数*1を用いてJSON文字列をJavaScriptオブジェクトに変換しています。これは手軽な方法ですが、もしJSON文字列だと思っているものの中に悪意のあるスクリプトが含まれていると実行されてしまいます。また、JSONでやり取りされる情報に守秘性の高い情報が含まれる場合、前述のCSRFを応用してサーバー側での認証を回避して情報を盗む攻撃も報告されています（詳細は[2]の記事をご覧ください）。こういった攻撃を防ぐには、JSONを取得するHTTPリクエストを正しく認証し、CSRFによる他サーバーからのリクエストを許さないようにしておく必要があります。

*1 eval()はJavaScriptの組み込み関数で、与えられた任意の文字列をJavaScriptとして解釈し、実行します。

❸ セキュアなWeb 2.0アプリケーション構築のために何をすべきか

これまで、Web 2.0アプリケーションが直面するさまざまなセキュリティー上の脅威（とその対策の一部）を紹介しました。それでは、どのようにしてこれらの脅威から、わたしたちのシステムを守ればいいのでしょうか?

残念ながら、これさえあれば100%安全だという魔法の呪文はありません。現実的には、複数の対策を組み合わせて用いることで全体的なセキュリティー強度を上げていくことが必要です。特にWeb 2.0アプリケーションでは、サーバー側とクライアント側の両方にロジックがあり、その一部はサーバー／クライアント間でやり取りされることがあるため、エンド・ツー・エンドでの対策が必要になります。主に開発者向けの注意点を幾つか以下に紹介します。アプリケーションの使用者側でもこれらの知識は役に立つでしょう。

(1)外部（ユーザー）からの入力の検証を確実に行う

XSS型の攻撃を防ぐためには、クライアントからの入力文字列をチェックし、意図しないスクリプト・コードが注入されていないかを検証する必要があります。幾つかのプログラミング言語では、有用な関数が用意されています。検証の際には、取り除きたい文字列パターンをあらかじめ登録しておくブラックリスト方式と、登録されたパターンに合致するものだけを受理するホワイトリスト方式があります。一般には後者の方が、セキュリティー強度が高いとされています。

(2)実行コードを動的に生成・実行することを避ける

不用意なシステムでは、ユーザーが入力した文字列の一部から構成されるプログラム・コード（例えばJavaScriptコード）をそのまま実行しているものがあります。十分な入力検証を行うことなしに、このような処理を行うべきではありません。

(3)インタラクションが必要でないコンポーネントには<iframe>タグを用いる。

例えば、ショッピング・サイトのようなマッシュアップ・アプリケーションでは、カタログの表示や購入処理の画面の横に広告サイトが表示されていることがあります。この広告サイトが攻撃され、悪意を持ったコードが注入されていると、そのコードが購入画面上のクレジット・カード番号などの情報にアクセスすることが可能になります。お互いにデータをやり取りする必要がないコンポーネントを定義するには<frame>ではなく<iframe>タグを用いることで、異なるサーバーからダウンロードされたコンテンツを含むフレーム間のアクセスを禁止することができます。

(4)セキュリティー上の脆弱性（ぜいじゃく）をチェックするツールを

用いる

Web 2.0アプリケーションは、攻撃者にとって主要な対象の一つとなっており、新しいタイプの攻撃が日々生まれています。これに対処するには、信頼の高いセキュリティー・ツールやサービスを用いるのが有効です。次章では、この分野におけるIBMの取り組みを簡単に紹介します。

❹ IBMの取り組み

IBMは、お客様のWebアプリケーションをセキュアにするためのさまざまな製品やサービスを提供しています。例えば、Tivoli®や、IBM製品／サービスとして新たに加わったISSやWatchfireの製品およびサービスを通じて、さまざまなビジネス規模やシナリオに沿ったソリューションを提供しています。例えば、ISSでは不正侵入防御・異常検出・アンチスパム対策などのセキュリティー機能を持ったアプライアンス製品群（Proventia®）を提供しています［3］。また、Watchfire AppScan®は、Webアプリケーションの脆弱（ぜいじゃく）性検査を自動化し、安全なWebアプリケーションの利用を支援します［4］。

また、筆者らが所属するIBM東京基礎研究所では、海外の基礎研究部門や開発部門と協業しながら、Web 2.0環境におけるエンド・ツー・エンドのセキュリティーに関する研究開発を行っています。

技術の普及のためには、外部コミュニティーや標準化活動への貢献も非常に重要です。Ajaxを中心にしたWeb 2.0技術の普及を目指す業界コンソーシアムであるOpenAjax Alliance［5］に、IBMは積極的に参加しています。筆者らは、Ajax Security技術に関するSecurity部会（Task Force）に参加し、ユースケースの収集やコンポーネント・モデルの標準化などに貢献しています。

［参考文献］

［1］RFC 4627： The application/json Media Type for JavaScript Object Notation（JSON）, D. Crockford（2006. 6）.
URL：http://www.ietf.org/rfc/rfc4627.txt

［2］Ajaxアプリケーションに対するセキュリティーの脅威を克服する －マッシュアップ・アプリケーションをセキュアにするためのヒントとベスト・プラクティス, http://www-06.ibm.com/jp/developerworks/xml/library/x-ajaxsecurity.shtml（2007.6.19）.

［3］Internet Security Systems 製品&サービス, http://www.isskk.co.jp/productservice.html

［4］Watchfire AppScanの概要, http://www.watchfire.com/jp/products/appscan/

［5］OpenAjax Appliance, http://www.openajax.org/

日本アイ・ビー・エム株式会社
東京基礎研究所
サービス指向コンピューティング担当

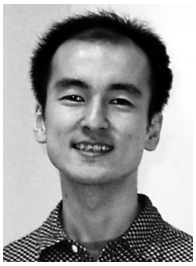
浦本 直彦 Naohiko Uramoto

［プロフィール］
1990年に入社以来、東京基礎研究所にて、機械翻訳、テキスト・マイニング、XMLやWebサービス関連の研究開発に従事。現在は、SOAやWeb 2.0におけるセキュリティーやパフォーマンスのプロジェクトをリードしている。博士（工学）。2000～2005年、国立情報学研究所客員助教授兼務。著作に、XML and Java - Developing Web Applications（Addison Wesley、共著）などがある。

日本アイ・ビー・エム株式会社
東京基礎研究所 セキュリティ&プライバシー
主任研究員

吉濱 佐知子 Sachiko Yoshihama

［プロフィール］
2001年よりIBM T.J. Watson研究所勤務、2003年より現職。トラステッド・コンピューティング、情報フロー制御、Webアプリケーション・セキュリティーなどの情報セキュリティー分野の研究に従事している。



日本アイ・ビー・エム株式会社
東京基礎研究所 サービス志向コンピューティング
主任研究員

牧野 聡 Satoshi Makino

［プロフィール］
2001年に入社以来、Web関連の技術（Webサービス、Ajaxなど）のセキュリティーや処理速度の向上に関する研究・開発に一貫して取り組むかたわら、技術書の翻訳も手掛ける。主な訳書に「入門XML第2版」「Ajaxデザインパターン」（ともにオライリー・ジャパン）。