3.Web アプリケーションにおける脆弱性

（独）産業技術総合研究所
情報セキュリティ研究センター
高木 浩光　takagi.hiromitsu@aist.go.jp

インターネットを利用した通信販売や金融取引，行政機関への電子申請などのシステムが **Web アプリケーション**として構築されることが多くなってきた．Web アプリケーションにはさまざまな種類の脆弱性があり，社会で実運用に供されているものにも数多く存在すると考えられている．本稿では，そのような状況が生じている背景，およびその対策が近年特に重要とされるに至った理由について整理した後，主にログイン機能に関する脆弱性についてその類型を示す．

Web アプリケーションの現状

◆ Web アプリケーションとは

Web アプリケーションは，入力フォームや送信ボタンなどを配置した HTML ページを Web ブラウザ上に表示させることでユーザのデータ入力を促し，ブラウザから送信されてくる入力データを処理して，動的に生成した HTML により結果をブラウザに表示させる．

Web ブラウザから Web サーバ内のプログラムを起動する仕組みを指す用語としては **CGI（Common Gateway Interface）**があるが，Web アプリケーションは，個々の CGI プログラムを指すよりも，CGI プログラムの呼び出しを複数回にわたって連携させて動作するシステムの全体を指す言葉として用いられている（単一の CGI プログラム呼び出しによる構成のものも含む）．

情報システムを Web アプリケーションとして構築することには次のような利点がある．(1) ユーザのコンピュータには汎用の Web ブラウザさえインストールされていればよいため，システムを導入する際の手間（ユーザないし管理者の手間）を小さくできる．(2) システムを更新するにはサーバ上のプログラムを変更するだけでよいため，更新の手間を小さくできる．(3) ユーザインタフェースに統一感を持たせることができるため，ユーザが利用しやすい．(4) ユーザのコンピュータの OS として複数のものをサポートする場合，開発コストを小さくできる．

通信販売サイトなど，初めてサイトを訪れた消費者にすぐさま利用してもらうことが重要となるシステムでは，Web アプリケーションでの構築が必須となっているが，それだけでなく，上記の（1）と（2）などの理由から，社内向けの業務システムの構築にも Web アプリケーションが用いられることが多くなってきている．

◆脆弱性を巡る動向の変化

Web アプリケーションの脆弱性は古くて新しい問題といえる．新しい点としては，(1) 被害発生時の損害波及範囲が拡大してきたこと，(2) 製品の修正パッチを待つだけでは対策にならなくなってきたこと，(3) 脆弱性を生みやすい実装手段が用いられることが多くなってきたことなどがある．

損害波及範囲の拡大

Web サイトの脆弱性は WWW が普及し始めたころから問題とされてきたが，当初は，脆弱性を突かれて侵入されたときの被害が，サーバ管理者のみにあると考えられることが少なくなかった．しかし近年では，Web アプリケーションが一般消費者の個人情報などを預かる機能を持つことが多くなったことから，被害が消費者にも及ぶとして，対策の必要性は無視できないものと考えられるようになった．

（独）情報処理推進機構が 2004 年 7 月より受付を開始した「脆弱性関連情報に関する届出」の制度でも，その根拠となった経済産業省告示[1]において，取り扱い対象とする脆弱性の範囲が「その脆弱性に起因する被害が不特定多数の者に影響を及ぼし得るもの」と限定されている．このことからも，消費者に被害をもたらす脆弱性

は特に対策が重要視されていることがうかがえる．

また，顧客情報を扱う社内用の業務システムが Web アプリケーションで構築されている場合も，そこに脆弱性があると，社内ネットワークに接続できる社員などに無権限で顧客情報を持ち出されるといった被害も考えられ，対策の重要性はインターネットに開放されたシステムだけにとどまらない．

脆弱性の個別化

Web サイトのセキュリティ対策は，かつては，サーバ製品の既知の脆弱性に対して修正パッチを適用することが主として考えられていた．あるいは，広く普及している CGI プログラム（掲示板プログラムなど）に脆弱性がある場合にも，修正版をインストールすることで問題は解決すると考えられていた．

しかし，特定のサイト用にカスタマイズして構築した Web アプリケーションに，それ固有の脆弱性がある場合には，修正パッチのリリースを待つという管理体制では脆弱性を排除できない．自力で自社のプログラムに脆弱性がないかを検査する必要がある．

昨今では，サイトごとに個別に Web アプリケーションが設計，実装されることが多くなってきているため，こうした検査の必要性が高まっている．

脆弱性の原因の変化

Web アプリケーションは，ログイン機能を独自に実装したものと，していないものに分類できる．かつては，ログイン機能を持たないか，ログイン機能を持つものでも，**Basic 認証（Basic Access Authentication）**によるものがほとんどであった．Basic 認証は，仕様が RFC 2617 で規定されており，Web ブラウザと HTTP サーバにあらかじめ実装されている機能であるので，それらに脆弱性が見つかっても，修正パッチを適用することで対策できた．

ところが昨今では，Basic 認証を使用せずに，HTML の入力フォーム中にユーザ名とパスワードを入力させるログイン方式を採用することがほとんどである．この場合，各画面において，入力されたパスワードが正しいことの確認が必要となるが，その実装方法には，cookie を使う方法や hidden タイプの input 要素を用いる方法，セッション ID を用いる方法やユーザ ID をそのまま使う方法など，さまざまなものがあり，脆弱性について関心の低い開発者が実装を担当すると，その場の思いつきで独自の方式を作り込むことがあり，その結果，パスワード入力をスキップして途中の画面に飛び込むことができてしまうといった脆弱性が生じている．

こうしたログイン機能の実装方法は規格化されておらず，その安全性は Web アプリケーション実装者の技量に委ねられているのが現状であり，脆弱なシステムが作り込まれやすい状況が生じている．

Web アプリケーションの脆弱性類型

◆ログイン機能に関する脆弱性

Web アプリケーションの脆弱性は，ログイン機能のない CGI プログラムにも古くから問題とされてきたものと，ログイン機能を独自に実装した Web アプリケーションならではの脆弱性とに分類できる．本稿では主に後者について整理する．以下，ログイン機能の実現に必要な**セッション追跡**について述べ，その実装方法別に脆弱性を分類する．

◆セッション追跡の一般的な実装

HTTP は，クライアントからの 1 つのアクセス要求に対してサーバが 1 つの応答を返して終了するという，単純なステートレスプロトコルを基本としているため，1 つのアクセスでは 1 つの処理しか実現できない．一方，Web アプリケーションは，複数の画面から構成されるものであるため，複数のアクセス要求を束ねて一連の操作としてみなす処理が必要となる．ログイン機能を持つ Web アプリケーションでは，サーバは，アクセス要求がどのユーザからのものであるかを識別する必要があり，同一のユーザからの連続したアクセスであることを識別することで，一連の操作の処理を実現する．

Web アプリケーションにおけるログインからログアウトまでの一連の操作を，**ログインセッション**（以下単に**セッション**）と呼ぶとき，ログインユーザからの一連の操作を識別するための処理は，**セッション追跡（session tracking）**または **セッション管理（session management）**と呼ばれている．

Basic 認証では，ログイン中のユーザがアクセスしたときに，そのユーザ名とパスワードの組（それらを base64 エンコードした文字列）を，HTTP の要求ヘッダに Authorization: フィールドとして含めるよう設計されている．Web ブラウザは，ユーザによって入力されたそれらを，それ以降のアクセスにおいて自動的に要求に含める動作をする．サーバは，要求に含まれているユーザ名に対し，パスワードが正しいかを確認して，応答を正常に返すかを判断するよう構成される．

すなわち，Web アプリケーションのログイン機能を Basic 認証で実現すると，画面ごとに毎回パスワードの送信と確認（認証）が行われることになる．ログインユーザからの一連のアクセスの識別は，Basic 認証で送信されるユーザ名で識別されるので，セッション追跡の

ための特別な仕掛けを Web アプリケーションに付加する必要はない．

これに対し，Basic 認証を使わないでログイン機能を実現する場合は，パスワードは最初の画面（ユーザがパスワードを入力して送信ボタンを押した直後の画面）でしか送信されないため，何らかの方法で，2 つ目以降の画面でユーザを識別するためのセッション追跡処理が必要となる．

最も一般的なセッション追跡の実装方法は，最初の画面でログインが成功した時点で，「受付番号」を発行し，これを cookie としてブラウザに記憶させる方法である．

cookie とは，サーバが応答ヘッダに「Set-Cookie: *name*=*value* の形式で指定すると，それを受信した Web ブラウザが，*name* で指定された名前の cookie として *value* の値を記憶し，それ以降，同じドメインのサーバにアクセスする際に自動的に要求ヘッダに「Cookie: *name*=*value*」の形式で記憶している値を送信するという仕組みである．そのため，「受付番号」を cookie に格納させれば，Web アプリケーションは，アクセスごとにそれがどの受付番号のユーザからのものであるかを識別できる．

cookie を使用しないセッション追跡の実装方法もある．ログインが成功した直後の画面の HTML ページ中に，「<input type="hidden" name="*name*" value="*value*">」の形式で，画面上には表示されない入力フォーム（以下 **hidden パラメタ**と呼ぶ）を埋め込む方法である．この画面で，次の画面に進むための送信ボタンが押されると，HTTP の GET 要求ないし POST 要求のパラメタとして，*name*=*value* の組がサーバに送信されることになる．*value* に「受付番号」を格納しておけば，ログイン直後の次の画面でもどの受付番号のユーザからのものかを識別できる．続く画面でも同様の形式で埋め込んでおくことで，連続した画面へのアクセスをセッション追跡できる．

要するに，アクセス要求中に含まれるいずれかのデータに，ユーザ名や受付番号を含ませれば，それをセッション追跡に利用できるのであり，cookie を送信する Cookie: フィールドや，Basic 認証の Authorization: フィールドもアクセス要求のパラメタの 1 つと見なせば，いずれかのパラメタがセッション追跡用に使われているはずということになる．そのようなパラメタを**セッション追跡パラメタ**と呼ぶことにする．

◆セッション追跡パラメタの値が推測可能な脆弱性

GET 要求のパラメタにユーザ名

セッション追跡パラメタにユーザ名のみを格納した Web アプリケーションを稀に見かけることがある．たとえば，ログイン後の画面の URL が次のようになっていたことがある（「takagi」はこの Web アプリケーションにおける筆者のユーザ名）．

http://example.com/foo.cgi?user=takagi

ここで，このページが POST 要求ではなく，かつ，cookie が発行されていないならば，この画面に対する要求のパラメタは「user=takagi」のみであることになるから，「takagi」の部分を他のユーザ名に変更してアクセスすれば，他のユーザになりすましてこの画面を閲覧できてしまうはずだと推定できる．

Web 開発技術者に限らず多くの人にとって，ユーザ名が URL に現れていればそこを変更してみたいという衝動[☆1]にかられることはよくあると思われるので，さすがにこうした欠陥を有する Web アプリケーションが，Web サイト運営者に気付かれずに放置されることは少ないはずであるが，筆者が目撃した事例では，画面が <frameset> で構成されており，サブフレームの URL がこのようになっていたため，Web ブラウザのアドレスバーを見ているだけではこのことに気付かないようになっていた．Web では，画面が <frameset> で構成されていようが，1 つ 1 つのフレームは独立した Web ページにすぎないので，直接上記の URL にアクセスされれば，パスワード入力なしに任意のユーザでアクセスされてしまうことになる．

<frameset> が使われていない場合でも，携帯電話用の Web アプリケーションにおいて同様の脆弱性が存在した事例がある．携帯電話ではアドレスバーが存在しないため，こうした脆弱性があってもサイト運営者が気付かない恐れがある．

POST 要求のパラメタにユーザ名

前節の事例では GET 要求が使われていたため，URL を確認するだけで脆弱ではないかと疑うことが容易にできた．これが POST 要求として実装されると，URL にユーザ名は現れなくなるが，セッション追跡パラメタにユーザ名だけが格納されているならば，脆弱であることに変わりない．

POST 要求の hidden パラメタにユーザ名だけを格納するセッション追跡の実装は，それなりに多く見かける．筆者が目撃したところでは，ある情報系学会の年次大会

☆1　故意に他人のユーザ名を指定してなりすましアクセスすることは，不正アクセス行為の禁止等に関する法律の第三条違反となると考えられるので，管理者の承諾を得て行う場合を除き，論理的にそれが可能であるはずと推定するにとどめておくべきであろう．

の登壇者個人情報登録サイトなどに存在した．

個人情報登録用 Web アプリケーションの典型的な構成では，新規登録と登録情報変更の機能が用意される．登録情報変更機能を使う際には，ユーザ名とパスワードを入力してログインし，ログインすると現在登録中のデータが表示され──（A），「変更」ボタンを押すと編集画面に進む──（B）といった構成である．このとき，筆者が目撃したサイトでは，ログイン直後の画面（A）を表示するにはパスワードの入力が必要であるものの，（B）の画面は，ユーザ名を hidden パラメタに与えるだけで取得できてしまう（と推定される）脆弱性を有していた．

cookie にユーザ名

GET や POST のパラメタが存在しない Web アプリケーションの場合，一見，なりすましアクセスはできないかのように感じられるかもしれない．しかし，何らかのパラメタでセッション追跡をしているはずであり，Basic 認証が使われていないならば，cookie が使われているはずである．

cookie は通常，ユーザの目に触れないところでやりとりされるためか，Web 開発技術者でさえその仕組みを知らないまま使用していることがあるようである．cookie にユーザ名だけを格納してセッション追跡を実装した Web アプリケーションがしばしば存在する．

筆者は，2001 年から 2002 年にかけて，著名な事業者の大規模サイトにおいてそうした事例を少なくとも 5 件目撃した [2)]．延べ 4 百万〜5 百万人分ほどと推定される個人情報が，パスワードなしに誰でも閲覧可能な状態にあったと推定される事例であった．

cookie は，HTTP のアクセス要求のヘッダに「Cookie: *name=value*」形式のフィールドを記述してサーバに送信するという単純な仕組みであるから，誰でもいつでもどんな値でも送信できる．たとえば，telnet コマンドで直接 HTTP の要求を送信し，cookie 付きでアクセスすることもできるし，Web ブラウザに，手作業で JavaScript を実行させて，任意の値の cookie をセットすることもできる．

推測可能な値の他の例

前節までの事例では，いずれもセッション追跡用パラメタにユーザ名を使用しているため，それが推測可能な値となっていた．他の変種としては，セッション追跡用のパラメタ値に，ユーザ名に対応付けた会員番号を採用していた事例や，ユーザ名の MD5 ハッシュ値を採用していた事例，ユーザ名を稚拙な独自暗号で暗号化していた事例などがある．

サイトにユーザ登録した正規の利用者が，自分のアカウントでログインしたときに，自分の Web ブラウザが送信するパラメタを観察することは自由である．5 桁前後の整数値のパラメタが存在して，それ以外にセッション追跡用らしきパラメタが存在せず，ログインを複数回繰り返してもその値が変化しないならば，それはそのサイトにおける自分の会員番号だと推察できる．

もし連続してもう 1 つのアカウントを登録することができるなら，そのパラメタがどのように変化するかを確認できる．筆者が目撃した事例では，複数のアカウントを取得したとき，連番の会員番号が発行されていた事例があった．これは，このパラメタの値を 1 番ずつずらすことで，すべてのユーザになりすましてアクセスできてしまうと推定されるものであった．

パラメタ値が文字列である場合，それが MD5 値らしき形式をしているならば，自分のユーザ名の MD5 値と比較してみることができる．一致するならば，他人のユーザ名の MD5 値を求めればそれでなりすましアクセスができてしまうと推定できる．

セッション追跡パラメタを暗号化したユーザ名とした場合であっても，その暗号が破られてしまえば，それは Web アプリケーションの脆弱性となる．筆者が目撃した事例では，シーザー暗号をひとひねりした程度の独自暗号や，単文字換字暗号を使用したものが存在した．

正しい実装方法

こうした脆弱性を生じさせないためには，セッション追跡パラメタが十分に推測困難な値となっている必要がある．最も単純な方法は，パラメタにパスワードも含めることである．アクセスごとにパスワードが通信路を流れることに抵抗感があってそれを採用しないという開発者の声を耳にすることがあるが，Basic 認証がそうしているのと同程度の強度となる．通信路上での盗聴を問題とするのであれば，TLS/SSL（以下「SSL」とする）により HTTP 通信全体を暗号化することが考えられる．

しかし，セッション追跡パラメタを cookie としている場合，cookie にパスワードを格納していると，それが別の脆弱性（Web ブラウザの脆弱性や，Web アプリケーションのクロスサイトスクリプティング脆弱性など）が原因で漏えいする心配がある．この心配を払拭するには，（1）パスワードのハッシュ値とユーザ名の組をパラメタ値とする，（2）ユーザ名を HMAC（Keyed-Hashing for Message Authentication）演算した値をパラメタ値とする，（3）両者を組み合わせる，といった実装方法が考えられる．

ただし，これらの値を採用していても，その値が何らかの原因で漏えいすれば，その値を使ってなりすましア

クセスはできてしまう．ユーザが同じパスワードを使用している他のサービスへの影響を防止するという効果しかない．

最も適切で一般に普及している実装方法は，先に述べた「受付番号」を使用する方法である．ログインごとにランダムに生成した受付番号を発行し，サーバ内で，受付番号からユーザ名を検索する記憶表を用意しておくことで，セッション追跡を実現する．受付番号はそのセッション限りのものであるため，一般に**セッション ID** と呼ばれる．

もしセッション ID が漏えいすると，ログイン状態を乗っ取られる「セッションハイジャック」攻撃を許すことになる．その点で，この実装方法でも cookie が漏えいした場合の完全な対策にはならないが，ユーザがログアウト状態のときに cookie が漏えいしてもセッションハイジャックされることはないという点で，前述の方法よりも強度が高いといえる．

◆強度の低いセッション ID の脆弱性

セッション ID が漏えいしなくても，セッション ID が予測できるような値となっているようでは，セッションハイジャック攻撃を許してしまうことになる．

たとえば，セッション ID を時刻を元に生成している場合は注意が必要である．時刻から ID を生成するアルゴリズムが解読されてしまうと，最近ログインしたユーザのセッションをハイジャックする攻撃を許してしまう．

開発者は，独自の発案による方法で乱数を生成しようとせずに，よく知られた安全な乱数生成系を用いるべきである．最近では，Web アプリケーションサーバなどと呼ばれるミドルウェアが普及しつつある．そうした製品は，セッション ID 発行やセッション追跡をミドルウェアレベルで提供しているので，これを用いるのがよい．

Web アプリケーションサーバに脆弱性が見つかることもある．筆者が目撃した事例では，日本のある銀行で，セッション追跡用に使われていた cookie のセッション ID が，ログインを連続して繰り返すと，一部の文字がわずかに変化するだけという現象を確認できた．これは，ある Web アプリケーションサーバ製品の脆弱性が原因であり，それより数カ月前に指摘され，修正パッチがリリースされていたものであった．

◆ クロスサイトスクリプティング脆弱性

cookie をセッション追跡用パラメタとしている場合，**クロスサイトスクリプティング（Cross-Site Scripting）脆弱性** に注意が必要である．Web アプリケーションにこの脆弱性があると，そのサイトで発行した cookie の値が攻撃者に漏えいする可能性があり，セッションハイジャック攻撃を許すことになる．

筆者らは 2001 年に行った調査[3] で，国内の大手サイト 8 カ所において個人情報が漏えいする可能性があり，うち 3 サイトではクレジットカード番号も盗まれ得る状態にあったことを指摘した．また，プライバシーマークおよびオンラインマークの取得事業者から無作為に抽出した 50 サイトと，銀行 22 サイトのうち，約 8 割にこの脆弱性が残存していたことを明らかにした．

この脆弱性は，米国 CERT/CC が 2000 年に発表した勧告 CERT Advisory CA-2000-02[4] において指摘されていたが，cookie 漏えいによるセッションハイジャックの危険性については触れられておらず，その深刻さが周知されていなかった．後に，cookie を用いたセッション追跡を行う Web アプリケーションが増加したこともあって，その危険性は知られるようになった．

◆ Secure 属性のない cookie の脆弱性

個人情報の送受信を伴う Web アプリケーションでは，SSL により HTTP アクセス全体を暗号化して保護するのが一般的となってきている．この場合，通信路でパケット盗聴が行われる可能性の存在を前提として，保護する画面が閲覧されないことが要件となる．

このとき，セッション追跡パラメタが暗号化されない通信（http: のページ）に含まれるならば，パケット盗聴によってセッションハイジャック攻撃を許してしまうことになる．

通信販売サイトなどにおいて，個人情報やパスワードが送受信される画面は https: のページとなっているが，ショッピングカートなどの画面は http: のページとなっていることがしばしばある．これは，ショッピングカートの内容を盗聴されることはユーザの許容範囲内だと判断して設計したものと考えられ，それ自体は妥当である．ここで，一度ログインすればショッピングカートにも，個人情報を変更するための画面にも，再度のパスワード入力なしに行き来できるよう設計されているなら，両者の画面は 1 つのセッションとして追跡されているはずである．

そのセッション追跡パラメタが盗聴によって漏えいするならば，ショッピングカート画面をハイジャックできるだけでなく，登録済み個人情報を閲覧する画面もハイジャックできてしまう．

筆者らは 2003 年に行った調査で，cookie を用いたセッション追跡方式を採用している国内の 22 サイトにおいて，こうした脆弱性のあるサイトが 20 カ所に及ぶことを明らかにした[5]．

この脆弱性を排除するには，cookie 発行時のオプションとして選択できる「secure」属性を指定すればよいの

であるが，このことが Web アプリケーション開発者にほとんど知られていなかった．

すべての cookie に secure 属性を指定すると，http: と https: の画面の混在したセッションを追跡できなくなるが，secure 属性を指定するものと指定しないものの 2 個の cookie に独立したセッション ID を記憶させることで，この問題を解決できる [5]．

この問題は，cookie に限らず hidden パラメタによるセッション追跡方法を採用している場合であっても同様に生じ得るものであるが，cookie による実装のサイトに事例が多い．

◆ ユーザ識別の脆弱性

セッション追跡が安全に実装されていても，画面ごとのアクセス制御に不備があることもある．

たとえば，セッション追跡をセッション ID で正しく行っているサイトにおいて，他のパラメタにユーザ名が含まれているとき，ログイン中の正規のユーザであれば，そのパラメタを別のユーザに変更してアクセスすると，そのユーザの画面が表示されてしまうという脆弱性があり得る．

2000 年に実際に事故が起きて報道されるに至ったゲーム販売会社の事例では，パラメタに伝票番号のようなものが含まれており，伝票番号を書き換えると，別のユーザの伝票であっても表示されてしまうという脆弱性があったようである．

また，ユーザごとに閲覧できる画面を制限している Web アプリケーションにおいて，閲覧権限の確認処理に不備があって制限できていないという脆弱性や，設定ミスによって制限がかけられていなかったといった事故も脆弱性の 1 つといえる．

◆ログイン機能以外の脆弱性

ログイン機能に関するもの以外にも，Web アプリケーションの脆弱性にはさまざまなものがある．それらの多くは古典的な脆弱性として古くから知られているので，ここではいくつかを簡単に列挙するにとどめる．

- パス名パラメタ未チェックの脆弱性
  パラメタをサーバコンピュータ内でのパス名（ファイル名）として解釈する CGI プログラムにおいて，絶対パス名が指定されるなどの攻撃によって，サイト運営者の意図に反して，任意のファイルを操作されてしまう脆弱性．
- ディレクトリ・トラバーサル脆弱性
  パス名パラメタをチェックして，絶対パス名などの指定を拒否する意図で設計された CGI プログラムにおいて，「../」や「..¥」，「...¥」など文字列が指定される攻撃によって，任意のディレクトリのファイルを指定されてしまう脆弱性．
- OS コマンドインジェクション脆弱性
  OS コマンドを起動する処理を含む CGI プログラムが，OS コマンドの引数に CGI パラメタの値を文字列連結で渡す場合において，パラメタへの記号の挿入などにより，任意のコマンドの実行を許してしまう脆弱性．
- SQL インジェクション脆弱性
  SQL クエリによるデータベース操作の処理を含む CGI プログラムが，SQL 文を，CGI パラメタの値を文字列連結で含ませて構成する場合において，パラメタへの記号の挿入などにより，任意の SQL 文の実行を許してしまう脆弱性．

特に SQL インジェクション脆弱性については，Web 開発者にまだ十分に知られておらず，稼働中の多くの Web アプリケーションに存在するのではないかと心配する声をしばしば耳にする．

研究動向

脆弱性を発見する研究は，アプリケーションレベルのものについてはあまり広く行われていない様子がある．特定のアプリケーションに特定の脆弱性を発見したとしても，それ自体を発表することは学術研究の場にはあまり馴染まない．アプリケーションは人工物であるので，特定の現象をただ発見したというだけでは普遍性がない．

Web アプリケーションの脆弱性は，学術研究の場よりも，セキュリティ検査を実施するサービス事業者などの現場の技術者らにより執筆された論文が，Web やメーリングリスト等で公開されて多くの技術者に読まれるというかたちで，論点が整理されているのが現状である．

脆弱性の発見を従来型の学術研究に発展させるには，次のような要素が必要であろう．

- その脆弱性が新たに発見された種類の原因によるものであり，他のアプリケーションにも一様に存在するなど，一定の不変性を持つ．
- その脆弱性の存在を確認する方法が統計的な手段を必要とするなど，高度な分析を必要とする．
- その脆弱性があってもサービスを安全に保つ，防止システムの研究開発につなげられるもの．
- その脆弱性の存在を自動的に発見する，検査システムの研究開発につなげられるもの．
- その脆弱性が実社会においてどの程度の割合で存在するかなど，実態調査に関するもの．

- その脆弱性を生じさせにくい開発方法論など，ソフトウェア工学の議論につなげられるもの．

バッファオーバーフロー脆弱性など，古くから知られている脆弱性については，脆弱性があっても安全を保つ言語処理系の研究や，脆弱性をコンパイル時や実行時に検出するシステムの研究がこれまでにも盛んに行われてきた．Web アプリケーションの個別の脆弱性については，最近になって，SQL インジェクションの攻撃を防止するミドルウェアの研究開発や，クロスサイトスクリプティング脆弱性を発見するツールの研究開発などが行われるようになってきている．

残された課題

そもそも，脆弱性の生じやすい実装方法で Web アプリケーションが開発されていることが根本的な問題であり，SSL のクライアント認証機能を用いる（ログイン機能に関する個別の脆弱性は生じにくくなる）とか，Web アプリケーションではなく，規格化されたログインセッションプロトコルで構築するといった実装方法を選択するべきだといえる．しかしながら，冒頭で述べたように，開発の柔軟性や経済効率の都合から Web アプリケーションが選択されているのであり，この状況はしばらくは変わらないと考えられる．

したがって，システムの発注段階から脆弱性の存在を想定した契約を交わすなどして，適正な費用をかけて安全なシステムを構築することが望まれる．そのためには，脆弱性の実態が Web サイト運営者に理解されることが必要であろう．

ソフトウェア製品の脆弱性については，脆弱性が発見されると，具体的に製品名を指してその事実が発表されるのが一般的となっている．これは，危険を回避するためにその製品に修正パッチを適用する必要があり，当該製品のユーザに脆弱性の存在を知らせることが求められているためであろう．それに対し Web アプリケーションの脆弱性については，特定の Web サイトに脆弱性が発見されたとき，その事実はその Web サイトの運営者だけに知らされることが多い．

これは，Web サイト運営者が脆弱性を排除する改修を施せばその時点でその危険は回避されるため，ユーザに周知する必然性が薄いためと考えられる．この特性から，Web アプリケーションの脆弱性の問題は社会に認知されないままとなりやすく，Web 開発者や Web サイト運営者たちが対策の必要性を知らされないという状況が生じやすかった．

この問題は，（独）情報処理推進機構が 2004 年 7 月より受付を開始した「脆弱性関連情報に関する届出」の制度によって，一定の解決が図られたといえる．届け出られた特定のサイトの脆弱性は，サイト名を挙げて公表されることはないが，脆弱性の種類別に統計情報として届出件数などが公表されている．これによって，どの種類の脆弱性がどの程度存在し，どの程度の期間で改修されているのかといった実態が周知されるようになった．

しかし，この届出窓口には報告されない種類の脆弱性もある．届け出られる脆弱性は，たいていの場合，Web サイト運営者の承諾を得ずに存在を知ったものであるため，不正アクセス行為の禁止等に関する法律に抵触しない方法でその存在を知る必要がある．同法が定める「利用の制限を免れる行為」を伴わずに脆弱性を発見しなくてはならない．

本稿で述べたように，ログイン機能に関する脆弱性のいくつかについては，正規のユーザが自分のアカウントで正規の手順でログインし，その際の通信内容を観察するといった手法によって，利用の制限を免れる実験を行わないで，脆弱性の存在を推定できるものもある．そうした脆弱性については，届出によって実態が明らかにされるであろう．それに対し，不正アクセス行為を伴わない限り発見されにくい脆弱性については，認知されないままとなりかねないという課題がある．

この制度では，確実に脆弱性が存在すると確認できていない段階であっても，その疑いがあるという程度で届け出ることもできる．事実かどうかの確認は，届出の受付機関と当該 Web サイトの運営者によってなされるので，適法な範囲内での報告の蓄積によって，そうした脆弱性についても，一部のものについては実態が徐々に明らかになるであろう．

参考文献

1）経済産業省：ソフトウェア等脆弱性関連情報取扱基準，平成 16 年経済産業省告示第 235 号（July 2004）．
2）産業技術総合研究所グリッド研究センターセキュアプログラミングチーム：秘密情報を含まない cookie に頼ったアクセス制御方式の脆弱性（Aug. 2002），http://securit.gtrc.aist.go.jp/SecurIT/advisory/rawcookie/
3）高木浩光，関口智嗣，大蒔和仁：クロスサイトスクリプティング攻撃に対する電子商取引サイトの脆弱さの実態とその対策，情報処理学会第 5 回コンピュータセキュリティシンポジウム CSS 2001（Nov. 2001）．
4）CERT/CC: CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests（Feb. 2000），http://www.cert.org/advisories/CA-2000-02.html
5）高木浩光，関口智嗣：Cookie 盗聴による Web アプリケーションハイジャックの危険性とその対策，産業技術総合研究所テクニカルレポート AIST03-J00017（July 2003），http://securit.gtrc.aist.go.jp/research/paper/AIST03-J00017/

（平成 17 年 5 月 19 日受付）