Web アプリケーションの脆弱性

トップ 10

2004 Update

2004 年 1 月 27 日

レビュー

「ほぼ週単位で新しい脆弱性がアナウンスされている。最新状態を維持するのに企業の多くが辟易しているかもしれない。しかし、脆弱性とその防御方法をリスト形式にしたコンセンサスが負担を減らしてくれる」

「Open Web Application Security プロジェクトは、Web アプリケーションとデータベースのセキュリティの脆弱性について、最も危険な 10 項目のリストを作成し、脆弱性に対処する最も効果的な方法を示した。
アプリケーションの脆弱性は軽視されがちだが、ネットワークの問題と同様に重要である。企業がすべてこの共通の脆弱性を排除したとしても、作業が終わることはないだろう。しかし、企業そしてインターネットは著しく安全になるだろう」

■ ***J. Howard Beales, III, Director of the Federal Trade Commission's Bureau of Consumer Protection, before the Information Technology Association of America's Internet Policy Committee, Friday, December 12, 2003***

「設定ミス、怠慢、欠陥のあるソフトウェアがインターネットで災いをもたらしている。脆弱性分野で重要なものに WWW 接続がある。WWW サービスはオープンさが身上であり、何でも受け付け、有用なリソースへのインタフェースとして活躍している。従って、これらのサービスが安全であることは重要である。しかし、あまりに多くの潜在的な脆弱性ゆえに、どこから防御を施し始めれば良いのか、判断するのがあまりにも困難になっている。OWASP の Top 10 は、カスタム WWW アプリケーションに存在する最も重要かつ起こりそうな脆弱性についてコンセンサスを得た。組織はこれを使用して力を集中すべきだ。これを信頼することで、アプリケーションを安全にするのに最も大きな影響が出る部分に取り組むことになる」

■ ***Eugene H. Spafford, Professor of Computer Sciences, Purdue University and Executive Director of the Purdue University Center for Education and Research in Information Assurance and Security (CERIAS)***

「この『10 個のとても欠陥がある』リストは、巨大な氷山の一角を鋭く削り取った。実情は情けない状況だ。システムと Web アプリケーションの大半は、セキュリティの原則やソフトウェアエンジニアリング、運用上の考慮点、そして実際のところ常識をも気に留めずに書かれている」

■ ***Dr. Peter G. Neumann, Principal Scientist, SRI International Computer Science Lab, Moderator of the ACM Risks Forum, Author of "Computer-Related Risks"***

「消費者とベンダー両者にとって、このリストは大きな成果だ。この成果によってベンダーは鍛えられ、これまで Web アプリケーションで何度となく繰り返されてきた、同じような間違いを防ぐだろう。そして消費者も、Web アプリケーションに求められる最低基準をベンダーに尋ねる手段と同時に、どのベンダーが期待に応えられるか否かを見分けられる手段も手にした」

■ ***Steven M. Christey, Principal Information Security Engineer and CVE Editor, Mitre***

「政府や営利組織が直面している最も深刻で見過ごされがちだったリスクに、OWASP の Top Ten はスポットライトを当てた。これらのリスクの根本的な原因は欠陥のあるソフトウェアではなく、セキュリティに対してほとんど何も注意を払っていないソフトウェア開発プロセスにある。組織にセキュリティに明るい文化を作り上げる最も効果的な第一ステップは、すぐさま Top Ten を Web アプリケーションセキュリティの最低基準として採用することだ」

■ ***Jeffrey R. Williams, Aspect Security CEO and OWASP Top Ten Project Leader***

「技術系企業がうまいことを言ったとしても、Web セキュリティに特効薬はない。この複雑で骨の折れる問題を解決するには、偉大な人間と素晴らしい知識、優れた教育、秀でた経営プロセスが揃った上で、素晴らしい技術を利用することだ。OWASP の Top Ten は体系立った熟慮の末に出来たリストだ。自身のセキュリティ方針(もしくサービスプロバイダの)の理解をまずここから始め、除々に自分たちの貴重なリソースを割く計画を立てることだ」

■ ***Mark Curphey, Founder of OWASP and Director of Application Security Consulting, Foundstone Strategic Security.***

「Web ページから事務部門の多量の計算まで、組織のほとんどすべてはアプリケーションコードを保持している。たくさんの人間がコードを書き、それを皆で利用している。しかし、プログラミング技術が 50 年余りの時を経た今でも、アプリケーションにあいかわらず欠陥が見つかる。さらに困ったことに、同種の欠陥が繰り返し発生している。自分たちの経験のみならず、親世代からも学ぶことを怠ったことで、潜在的な攻撃に対する脆弱性が数え切れない程生じている。アプリケーションへの攻撃が増加するのも無理はない。

OWASP は、アプリケーションコードのとても危険な欠陥を 10 個載せたリストを作り上げた。これによって、よくある弱点に焦点を当て、その弱点にどう対処するのか、開発者とユーザ双方に寄与した。ここで提示されたこの考え抜かれた指標を採用することは、ソフトウェアを開発する組織やプログラマ、そして利用者の義務である」

■ ***Dr. Charles P. Pfleeger, CISSP, Master Security Architect, Cable & Wireless, Author of "Security in Computing"***

「セキュリティは新しい投資収益だ。企業は取引相手に対して信頼できる Web アプリケーションと Web サービスを提供できなければいけない。取引相手は、セキュアな技術と証書のようなこれまでの金融的保証である証印、その双方を要求している。しかしそれで終わりではない。不完全なセキュリティは、企業のデータとアプリケーションを危険な状態に置き、企業存続をも危うくする。顧客を失望させることとシステムを破壊や汚染から復旧しようとすることとは別問題だ」

■ ***Robert A. Parisi, Jr., Senior VP and Chief Underwriting Officer, AIG eBusiness Risk Solutions***

「Web 開発者は、好ましくないインターネット環境からどの程度ビジネスアプリケーションと顧客データがを保護するべきかを知る必要がある。自分たちが書いたコードが直接安全を左右するからだ。OWASP の Top Ten リストは、コードを防御し、Web アプリケーションを悩ますセキュリティ上の落とし穴を回避する手段を理解する、素晴らしい方法だ」

■ ***Chris Wysopal, Director of Research & Development, @stake, Inc.***

「ユーザのプライバシーを保護する安全な Web アプリケーションの提供。医療関連産業は切にそれを必要としている。医療関連組織が Web アプリケーション製品とソリューションのセキュリティを評価するのに、OWASP の Top Ten が役に立つ。欠陥を抱えたアプリケーションを使っていれば、どんな医療関連組織も HIPAA 法を満たすのは困難だろう」

■ ***Lisa Gallagher, Senior VP, Information and Technology Accreditations, URAC***

「ますます多くの企業が Web アプリケーション開発を海外にアウトソーシングするようになってきている。従って、安全なコーディングの実施を実行優先リストのトップにすぐにでも上げなければならない。OWASP の Top Ten プロジェクトは、アプリケーションの脆弱性をきちんと明確にした。そして経営層には、Web アプリケーションのセキュリティポリシーを設定するのに役立つ願っても無い情報となった」

■ ***Stuart McClure, President and CTO of Foundstone Inc.***

「アプリケーションのセキュリティを検討する場合、たいていはそれを専門分野として扱い、コーディングが終わり、その他すべてのビジネス上の必要条件を満たした後に適用される。Oracle は、アプリケーションを安全にする作業が当初からセキュリティを組み込んでこそ成功率が高くなる、と理解している。開発者はセキュリティの脆弱性の基礎と脆弱性を回避する手段を理解しなければいけない。またリリース担当マネージャーは、製品を出荷する前にセキュリティの必要条件を満たしているかどうかを入念にチェックしなければいけない。OWASP の Top Ten リストは、開発者にセキュリティを自覚させ、アプリケーション開発のセキュリティ基準を用意する。企業にとって素晴らしい第一歩である」

■ ***John Heimann, Director of Security at Oracle Corp.***

「Top Ten リストが『ネットワークセキュリティ上の最大のリスク』と思っている人間が多数だった。これらのリストが指摘しているのは、ネットワーク上で動くであろうサードパティのソフトウェアに存在する欠陥についてだけである。このリストにある情報を適用すれば、開発するソフトウェアが何処かのリストで触れられることは無くなるだろう」

■ ***John Viega, Chief Scientist, Secure Software Inc., co-author of "Building Secure Software"***

目 次

はじめに

Open Web Application Security Project(OWASP)は、組織が Web アプリケーションや Web サービスのセキュリティを理解し改善する手助けに全力を注いでいます。このリストは、法人や政府関連機関が最も深刻な脆弱性に関心を向けるように作成されました。Web アプリケーションのセキュリティは、企業が競ってコンテンツを作成し、Web 使ってサービスを提供するにつれ、論争の的になっいます。と同時に、アプリケーション開発者が作った共通の弱点にハッカーが注目してきています。

Web アプリケーション立ち上げると、HTTP リクエストを世界中から呼び込みます。攻撃はこのリクエストに紛れ込み、ファイアーウォールやフィルタ、強化したプラットフォーム、侵入検知システムを警告無しで通り抜けます。攻撃が正規の HTTP リクエストだからです。SSL を使った「安全な」Web サイトでさえ、暗号化された経路を通じて精査されることなく、リクエストを受け取ってしまいます。**つまり、Web アプリケーション・コードはセキュリティ境界線の一部になのです**。Web アプリケーションの数、規模、複雑さが増える程、境界線が広がって行きます。

ここで述べるセキュリティ上の問題は目新しいものではありません。実際数十年も前から知られているものもあります。様々な事情から、代表的なソフトウェア開発プロジェクトはいまだにこの過ちを犯し続け、顧客のセキュリティはおろか、インターネット全体のセキュリティまでも危うくしています。この問題を直す「特効薬」はありません。評価や防御の技術は進歩してきていますが、現状では良くてもそのほんの一部にしか対処できていません。このドキュメントで扱う課題に取り組むには、開発スタイルを見直し、開発者を教育し、最新の開発プロセスの導入した上で、適切な技術を使用する必要があります。

OWASP の Top Ten は、すぐにも改善が必要な脆弱性のリストです。攻撃者はこれらの欠陥を盛んに突きますので、脆弱性がないか、既存のコードをチェックしてください。開発プロジェクトでは、必要とするドキュメントや設計・構築において、これらの脆弱性に注意を払ってください。また、アプリケーションに脆弱性が入り込んでいないか必ずテストしましょう。プロジェクトマネージャーは、アプリケーションのセキュリティを実現するために、開発者の教育、アプリケーションのセキュリティポリシー策定、セキュリティ機構の設計・開発、侵入テスト、セキュリティコードのレビューに時間と予算を割いてください。

企業が **OWASP の Top Ten を最低基準として採用し**このリストに参画するとともに、脆弱性がない Web アプリケーションの構築に注力するように働きかけています。

このリストは、ANS/FBI Top Twenty List という、とてもうまくいった書式と同じ書式を採用しました。より分かりやすく、使いやすくするのが目的です。SANS リストは広く利用されているネットワークやインフラ設備に特に存在する欠点にターゲットを当てています。ただし Web サイトは一つ一つが別々なので、OWASP の Top Ten では Web アプリケーションでよく発生する特定の脆弱性のタイプもしくはカテゴリにまとめました。これらのカテゴリは OASIS Web Application Secuurity(WAS) XML プロジェクトが規格化しています。

OWASP の専門家たちは、長年に渡ってアプリケーションのセキュリティについて、政府、金融業、製薬業、製造業で経験を積みながら、ツールやテクノロジーを開発してきました。その彼らの知識を集約した結果がこのリストです。このドキュメントは Web アプリケーションの重大な脆弱性知っていただくために書かれました。これら脆弱性についてより詳しく記述があり、それを排除する方法を詳しく手引きにして載せている書籍やガイドラインがたくさんあります。そんなガイドラインの一つに OWASP Guidance があります。http://www.owasp.orgで利用できます。

OWASP の Top Ten は、刻々と変化するドキュメントです。セキュリティ上の欠陥を修正するのに役立つ情報へのポインタや解説があります。リストや解説を更新し、より重大な脅威やより現状に即した使いやすい方法を提供します。もちろんこの先皆さんからの情報を歓迎します。このドキュメントはコミュニティー同の総意によるものです。皆さんが攻撃者と戦ったり、脆弱性を排除したりした経験が、他のまだ未経験な人々を助けることになります。「OWASP Top Ten Comments」というサブジェクトで電子メールを topten@owasp.orgまで送ってください。【訳注：翻訳ミスは、高橋 聡hisai@din.or.jpまでお願いします】

OWASP は、調査と準備に対する **Aspect Security** の絶大なる 協力に たいへん感謝します。

http://www.aspectsecurity.com

背景

Web アプリケーションの脆弱性の「トップ」を決定するという難題は、事実上不可能な作業です。「Web アプリケーションのセキュリティ」という言葉が意味するところが厳密には何なのかということでさえ、広く合意がとれていません。焦点を当てるべきは、カスタム Web アプリケーションのコードを書く開発者に影響を与えるセキュリティ課題だ、という主張もあります。一方、もっと広範にライブラリやサーバ設定、アプリケーション層プロトコルといった、アプリケーション層全体をカバーすべきだ、という主張もあります。我々は組織が直面している重大なリスクに取り組むという目的から、Web アプリケーションのセキュリティをかなり広く解釈することにしました。しかし、ネットワークやインフラ面のセキュリティ課題については触れないままにします。

もう一つ課題を難しくしているのは、脆弱性が組織の Web サイトそれぞれに固有である点です。個々の組織の Web アプリケーションに存在する特定の脆弱性が、表面化することはほとんどありません。その存在がサイトを見ている多数の知るところとなると、うまくいけば即行で修正されるからです。そのような訳で、Web アプリケーションの脆弱性でもトップクラスのものに焦点を当てることにしました。

このドキュメントの初版では、Web アプリケーションの広範囲に渡る問題を分類し、重要なカテゴリに分けることにしました。脆弱性を分類する様々な方法を研究し、体系だったカテゴリを作成しました。脆弱性のカテゴリとして優れた条件は、欠陥に直接結びつくかどうか、同様な対策を施せるか、典型的な Web アプリケーション・アーキテクチャで頻繁に発生するのか、です。この版では対策を洗練させています。対策は OASIS WAS 技術委員会で継続して審議されました。あわせてこの委員会では、セキュリティの調査者が XML フォーマットで問題の特徴を表現できるシソーラスを用意する予定です。

多数の候補の中から TopTen を選ぶのはそれ自体困難です。Web アプリケーションの問題について信頼に足る統計データがあるわけではありません。将来的には Web アプリケーションのコードにある欠陥の頻度統計を集めるつもりです。その測定基準に基づいて、TopTen の優先順位を決めるのに活用する予定です。しかし様々な理由から、この種の測定方法は近い将来には実現しないでしょう。

Top Ten に脆弱性のカテゴリが存在するはずということに対し、「正しい」答えはないと認識しています。組織に対するリスクは、欠陥のどれかが起こる可能性と事業に対する影響度に基づいて、各組織で検討しなければいけません。さし当りこのリストでは、様々な分野の組織に対して、とりわけリスクが高い問題をまとめて提起します。Top Ten には特に順序はありません。どれが最もリスキーなのかを決定するのは、ほとんど不可能だからです。

OWASP の Top Ten プロジェクトは、Web アプリケーションセキュリティの重大な欠陥について広く皆さんが情報を利用できるよう、常に努めています。OWASP のメーリングリストでの議論に基づき、ドキュメントを年一回更新するつもりです。フィードバックは topten@owasp.orgまでお願いします。

更新部分は？

2003 年 1 月の TopTen のリリース以降、OWASP は進歩しました。この更新ではあらゆる議論を取り込み、考え方や評価を再考し、過去 12 か月に渡る OWASP コミュニティでの議論を盛り込みました。結局、細かい修正を全体的に行い、大幅な修正はほんのわずかとなりました。

- **WAS と XML の連携** – 2003 年に立ち上がった新しいプロジェクトの一つに、OASISの Web Application Security Technical Committee(WAS TC)があります。WAS TC の目的は、Web セキュリティ関連の脆弱性の分類体系を構築し、初期兆候やその影響、それによって発生するリスクの評価について、指標となるモデルを作成することです。そして、評価と防御ツールいずれもが利用可能な Web セキュリティの状態を表現した XML 手法を構築することも目的としています。OWASP の TopTen プロジェクトでは、WAS TC をリファレンスとして利用して、トップテンを再分析し、Web セキュリティの脆弱性の分類に標準化した手法を持ち込みました。WAS シソーラスは、Web セキュリティの脆弱性を論ずるための標準言語を定義していて、このドキュメントではその用語を採用しています。
- **サービス妨害の増加** – トップレベルで唯一変更されたカテゴリは、「A9 サービス妨害」のリストへの追加です。私たちの調査によって、このタイプの攻撃に影響されやすい組織が広く存在することがわかっています。サービス妨害攻撃の可能性とその攻撃が成功した結末を踏まえて、このトップテンに入れるだけの正当な理由があると判断しました。この新しい項目を取り込むために、昨年の「A9 リモート管理の欠陥」を「A2 不完全なアクセス制御」のカテゴリにまとめました。「A9 リモート管理の欠陥」が「A2 不完全なアクセス制御」の特殊なケースだからです。私たちはこれが適切だと考えます。およそ A2 にあるタイプの欠陥は A9 と同じで、必要な対処方法も同じだからです。

下記の表で、新旧の TopTen と WAS TC シソーラスとの関係を示します。

新 Top Ten 2004	旧 Top Ten 2003	新 WAS シソーラス
A1 許可されていない入力	A1 許可されていないパラメタ	入力の検証
A2 不完全なアクセス制御	A2 不完全なアクセス制御 (A9 リモート管理の欠陥)	アクセス制御
A3 認証とセッション管理が不完全	A3 アカウントとセッション管理が不完全	認証とセッション管理
A4 クロスサイトスクリプティング(XSS)の欠陥	A4 クロスサイトスクリプティング(XSS)の欠陥	入力の検証->クロスサイトスクリプティング
A5 バッファオーバーフロー	A5 バッファオーバーフロー	バッファオーバーフロー
A6 挿入(injection)の欠陥	A6 コマンド挿入の欠陥	入力の検証->挿入
A7 不適切なエラー処理	A7 エラー処理問題	エラー処理
A8 安全ではない保存	A8 暗号の利用が安全ではない	データ保護
A9 サービス妨害	N/A	可用性
A10 設定管理が安全ではない	A10 Web サーバとアプリケーションサーバの設定ミス	アプリケーション設定管理 インフラストラクチャー設定管理

トップテンのリスト

Web アプリケーションのセキュリティの脆弱性で、最も重大なものについて下記に概要を挙げます。それぞれについては、追って各セクションで詳細に説明します。

Web アプリケーションの脆弱性トップ		
A1	**許可されていない入力**	Web リクエストからの情報は、Web アプリケーションが使用する前に検証されていません。攻撃者はこの欠陥を使って、Web アプリケーション経由でバックエンドのコンポーネントを攻撃します。
A2	**不完全なアクセス制御**	認証済みユーザに何を認めるのか、制限がきちんと施されていません。攻撃者はこの欠陥を突いて、他のユーザのアカウントにアクセスし、秘密のファイルを見たり、承認されていない機能を使ったりします。
A3	**認証とセッション管理が不完全**	アカウント証明やセッショントークンがきちんと保護されていません。攻撃者はパスワードや鍵、セッションクッキー、その他トークンを汚染し、認証制限を無効にし、他のユーザの身分証明を推測します。
A4	**クロスサイトスクリプティング(XSS)の欠陥**	エンドユーザのブラウザに攻撃を送り込むしくみとして、Web アプリケーションが利用されます。攻撃が成功すると、エンドユーザのセッショントークンが暴かれ、ローカルマシンを攻撃したり、コンテンツを偽ってユーザをだましたりします。
A5	**バッファオーバーフロー**	入力をしっかり検証していない言語における Web アプリケーションのコンポーネントには、クラッシュさせられ、場合によってはプロセス制御を乗っ取られるものがあります。これらのコンポーネントに該当するのは、CGI やライブラリ、ドライバ、Web アプリケーションサーバです。
A6	**挿入(injection)の欠陥**	Web アプリケーションは、外部システムやローカルの OS にアクセスする時にパラメタを渡します。攻撃者が不正なコマンドをパラメタに埋め込めば、外部システムはこのコマンドを Web アプリケーションに代わって実行してしまうかもしれません。
A7	**不適切なエラー処理**	通常運用時のエラー状態を適切に処理していません。攻撃者がエラーを起こして Web アプリケーションが処理しないと、システムの詳細な情報を手に入れ、サービスを妨害し、セキュリティ機構を破壊し、サーバをクラッシュさせます。
A8	**安全ではない保存**	Web アプリケーションは、暗号機能を使用して情報や証明書を保護しています。これらの機能とそれを実装しているコードが適切であることを示すのは困難で、度々防御の弱点となります。
A9	**サービス妨害**	攻撃者は、他の正規ユーザをアクセス不可にしたり、アプリケーションを利用できなくしたりするところまで Web アプリケーションのリソースを消費できます。またユーザをアカウントから閉め出したり、アプリケーションを全く使えなくしたりさえできます。
A10	**設定管理が安全ではない**	強固なサーバ設定基準は、安全な Web アプリケーションにとって決定的な要素です。これらのサーバには、セキュリティに関連しかつ安全を脅かす設定オプションがたくさんあります。

A1 許可されていない入力

A1.1 解説

Web アプリケーションは HTTP リクエスト(場合によってはファイル)を入力として使用し、どのように応答するのかを決めています。攻撃者は HTTP リクエストのどの部分も細工できます。HTTP リクエストには URL、クエリー文字列、ヘッダー、クッキー、フォームフィールド、隠しフィールドがあり、それを利用してサイトのセキュリティ機構を通り抜けようとします。入力を操って行われる攻撃の呼び名には、次のものが挙げられます。強制的なブラウジング(force browsing)、コマンド挿入(command inspection)、クロスサイトスクリプティング、バッファオーバーフロー、書式文字列(format string)攻撃、SQL 挿入(injection)、クッキー汚染(cookie poisoning)、隠しフィールド操作(hidden field manipulation)です。このタイプの攻撃は後ほどこのドキュメントで説明します。

- A4 – クロスサイトスクリプティングの欠陥は、他ユーザのブラウザ上で実行されるスクリプトを含む入力として論じます
- A5 – バッファオーバーフローは、プログラムの実行空間を上書きするように設計された入力として論じます
- A6 – 挿入の欠陥は、実行コマンドを含むように改変される入力として論じます

不正な入力をフィルタリングすることで、防御しようとするサイトも存在します。この方法の問題は、情報を符号化するのにあまりにも色々な方法がある点です。符号化形式は暗号化形式と同じではありません。復号が簡単だからです。にもかかわらず、開発者はすべてのパラメタを使用前に最も単純な形式に復号するのをついつい忘れます。パラメタは、有効になる前に最も単純な形に変換しなければいけません。さもなければ、不正な入力は見つかること無くフィルタを通り過ぎてしまいます。符号化を単純にする過程を「正規化(canonicalization)」と呼びます。HTTP 入力の大部分は複数の書式で表現できますので、このドキュメントで説明する脆弱性を狙った攻撃を混乱させるのに、この技術が使えます。フィルタリングはとても難しくなります。

Web アプリケーションが入力を検証する際に、クライアント側のしくみだけを利用するものがあまりに多すぎます。クライアント側の検証機構は簡単にすり抜けられるので、不正なパラメタに対して Web アプリケーションが無防備になってしまいます。攻撃者は telnet のような単純なツールを使って自分で HTTP リクエストを作れます。攻撃者は、開発者がクライアント側で何をしようと気にする事は何もありません。パフォーマンスや使いやすさという面では、クライアント側の検証は良い考えです。しかしセキュリティ面では良いことは何もありません。パラメタ改竄攻撃を防ぐには、サーバ側のチェックが必須です。サーバ側のチェックが働けば、クライアント側のチェックも包括し、正規ユーザが使い易くなり、サーバへの無効なトラフィックも減らせます。

パラメタを「ファジー」にしたり、不正に変えたりするツールの数とともに、これらの攻撃は増加しています。総当り攻撃(brute forcing)も増加しています。入力を検証無しで使用する影響を過小評価してはいけません。開発者が入力を使用前に検証するだけで、攻撃の大多数が困難もしくは不可能になります。Web アプリケーションが強固で集約したしくみを使って HTTP リクエスト(その他のいかなる情報源)からの入力をすべて検証しない限り、不正な入力による脆弱性は残るでしょう。

A1.2 影響を受ける環境

Web サーバとアプリケーションサーバ、Web アプリケーション環境は、パラメタ改竄から影響を受けやすくなっています。

A1.3 例と参考文献

- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Data Validation http://www.owasp.org/documentation/guide/
- modsecurity project (Apache module for HTTP validation) http://www.modsecurity.org
- How to Build an HTTP Request Validation Engine (J2EE validation with Stinger) http://www.owasp.org/columns/jeffwilliams/jeffwilliams2
- Have Your Cake and Eat it Too (.NET validation) http://www.owasp.org/columns/jpoteet/jpoteet2

A1.4 脆弱性の判定方法

Web アプリケーションが使用する HTTP リクエストを入念に検証しないと、そのどの部分も「汚染された」パラメタと呼ばれます。汚染されたパラメタの利用を簡単に発見するには、きめ細かいコードレビューを行い、HTTP リクエストから得た情報を呼び出す箇所をすべてを捜してください。例えば J2EE アプリケーションでは HttpServerRequest クラスのメソッドが該当します。コードを追って、変数がどこで取得しているのか見つけてください。変数を使用前にチェックしないと、多分問題になります。Perl では「taint(-T)」オプションの使用を検討しましょう。

OWASP の WebScarab のようなツールを使っても、汚染されたパラメタを見つけられます。HTTP リクエストに規定外の値を入れ、Web アプリケーションの応答を見ることで、汚染されたパラメタが何処で使用されているのか特定できます。

A1.5 防御方法

パラメタ改竄を防ぐには、使用前にすべてのパラメタを必ず検証するのが最良の方法です。ライブラリもしくはコンポーネントを集約することが、最も効果的なようです。コードが実行されると、チェックがすべて一箇所で行われるはずだからです。どの入力を許可するのかを厳密に規定したフォーマットを使って、各パラメタをチェックしてください。不正な入力をフィルタリングする「禁止」方式や署名に基づいた方式は効果が疑わしく、メンテナンスが難しいでしょう。

パラメタは「許可」方式で検証してください。つまり、

- データ型(文字列、整数、実数等)
- 許可する文字セット
- 最大最小長
- null を許すか否か
- パラメタの要不要
- 重複を許すか否か
- 数の範囲
- 正しいの値の規定(一覧)
- 正しいパタンの規定(正規表現)

Web アプリケーション・ファイアーウォールという新しい種類のセキュリティデバイスを使うと、パラメタ検証サービスが利用できます。しかし有効に活用するには、サイトにとって適切なパラメタをそれぞれ厳密に定義・設定しなければいけません。このデバイスは、HTTP リクエストに由来する全タイプの入力を確実に防御します。入力には URL フォーム、クッキー、クエリー文字列、隠しフィールド、その他パラメタがあります。

OWASP Filters プロジェクトでは、複数言語で再利用可能なコンポーネントを作成し、様々な形式のパラメタ改竄を防止するのに役立っています。Stinger HTTP request validation engine(stinger.sourceforge.net)も J2EE 環境用に OWASP が開発しました。

A2 不完全なアクセス制御

A2.1 解説

アクセス制御は承認(authorization)とも呼ばれます。Web アプリケーションがコンテンツや機能に対して、あるユーザにはアクセスを認め、その他のユーザには認めないしくみです。これらのチェックは認証(authentication)後に実行され、「承認済み」ユーザが認められている行為を制御します。アクセス制御は単純な問題という印象を与えますが、正しく実装するのは思った以上に困難です。Web アプリケーションのアクセス制御モデルは、サイトが提供するコンテンツや機能と密接に結びついています。おまけにユーザは、様々な資格や権限を持つ多数のグループや役割に分かれているかもしれません。

開発者は、信頼性が高いアクセス制御機構を実装する難しさを軽視しがちです。実装案の多くは熟慮した上で立てられたものではなく、Web サイトとともにそれなりに改善されました。こういった場合、コードのあちこちにアクセス制御ルールが入り込みます。サイトオープンが間近になると、アドホックなルールの固まりがそこかしこに渡り、ほとんど理解不可能な状態になります。

欠点を抱えたアクセス制御システムの多くで、欠陥を発見し悪用することは難しくありません。許可すべきではない機能もしくはコンテンツを含むリクエストを巧妙に作り込むこと、これが必要なすべてである場合がほとんどです。欠陥が発見されると、その欠陥を持ったアクセス制御システムは壊滅的になるでしょう。承認されていないコンテンツが見られるのに加えて、攻撃者はコンテンツを改竄・削除をしたり、承認されていない機能を実行したり、場合によってはサイト管理を乗っ取っとるかもしれません。

アクセス制御の特徴的な問題の一つに、管理者がサイトをインターネット越しに管理できるインタフェースがあります。この機能によってサイト管理者は、ユーザやデータ、サイトのコンテンツを効率良く管理できます。多くの場合、サイトは様々な管理機能をサポートし、事細かにサイト管理ができるようになっています。この強力さに目が付けられ、度々これらのインタフェースは内外から攻撃の最初の標的になっています。

A2.2 影響を受ける環境

既知の Web サーバとアプリケーションサーバ、Web アプリケーション環境が少なくともこの問題のいくつかを被る恐れがあります。サイトが完全に静的なものであっても、きちんと設定していなければ秘密ファイルにハッカーがアクセス可能になるか、サイトが荒らされるか、他に被害を被るでしょう。

A2.3 例と参考文献

- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Access Control: http://www.owasp.org/guide/
- Access Control (aka Authorization) in Your J2EE Application http://www.owasp.org/columns/jeffwilliams/jeffwilliams3
- http://www.infosecuritymag.com/2002/jun/insecurity.shtml

A2.4 脆弱性の判定方法

現実にアクセス制御はすべてのサイトで必須です。従って、アクセス制御ポリシーはきちんとドキュメント化してください。また設計書はこのポリシーを実現するように取り組みましょう。このドキュメントが存在しなければ、そのサイトはやられたも同然です。

アクセス制御ポリシーを実装するコードはチェックが必要です。そのようなコードは構造化・モジュール化し、集約した方が良い場合がほとんどです。詳細なコードレビューでは、アクセス制御の実装の正しさを検証するようにしてください。さらに、アクセス制御システムに問題があるか否かの判断には、侵入テストがかなり役立ちます。

皆さんの Web サイトがどのように運用されているのか調べてください。どのように Web ページが変更されるのか、どこでテストされるのか、どのように本番サーバに転送されるのかを調べましょう。管理者がリモートで変更をかけられるなら、通信経路がどのように保護されているかを理解してください。各インタフェースを慎重にレビューし、必ず承認された管理者のみがアクセスできるよ

うにしましょう。また、インタフェース経由で異なるタイプやグループのデータをアクセスできるなら、同様に必ず承認済みデータだけがアクセスできるようにしてください。そのようなインタフェースが外部コマンドを利用しているなら、コマンドの使用方法をレビューし、このドキュメントで説明しているコマンド挿入の欠陥から何も影響されないようにしましょう。

A2.5 防御方法

最も大切なステップは、アプリケーションのアクセス制御の要件を熟考し、Web アプリケーションのセキュリティポリシーに反映することがです。アクセス制御ルールを決定するのに、アクセス制御の真理表を利用してください。セキュリティポリシーをドキュメント化しないと、そのサイトで何が安全なのか定義がないことになります。ドキュメント化しておくべきポリシーとは、どのタイプのユーザがシステムにアクセスできるのか、そのタイプの各ユーザがどんな機能とコンテンツにアクセスできるのか、です。バイパスする方法が絶対ないように、アクセス制御のしくみを詳細にテストしてください。このテストには、様々なアカウントと承認されていないコンテンツや機能への広範囲に渡るアクセスが必須です。

アクセス制御の具体的な問題は下記の通りです。

- 安全でない識別子-大部分の Web サイトは、ユーザや役割、コンテンツ、オブジェクト、機能などを照会する際に、ID や鍵もしくはインデックスを何らかの形で利用しています。攻撃者がこれら識別子を推測し、与えられた値が現在のユーザの承認済と検証されなければ、攻撃者はアクセス制御システムを好きに操り、アクセス可能なものを好き勝手に見られます。Web アプリケーションは防御目的で、識別子のどのような秘密性にも依存してはいけません。
- アクセス制御のチェックをすり抜けた強制的なブラウジング-ユーザがサイトの「奥」に潜む URL にアクセスを認められる前に、サイトの多くはチェックをパスする必要があります。セキュリティチェックがあるページを飛び越して、ユーザがチェックを回避できてはいけません。
- パスの乗り越え(Path Traversal)-相対パス情報(例えば「../../target_dir/target_file」)をリクエスト情報の一部に組込む攻撃です。通常誰も直接はアクセスできないか、直接リクエストしても拒否されるファイルに対して、攻撃者がアクセスを試みます。この手の攻撃は、結局はファイルにアクセスすることになる他の入力(例えば、システムコールやシェルコマンド)と同様、URL に挿入できます。
- ファイルのパーミッション-Web サーバとアプリケーションサーバの多くは、プラットフォームが備えているファイルシステムが規定したアクセス制御リストに依存しています。バックエンドサーバにすべてのデータがあるにしても、Web サーバとアプリケーションサーバのローカルには、公開してはいけないファイルが常に存在しています。それは、設定ファイルやデフォルトファイル、Web サーバとアプリケーションサーバのほとんどにインストールされているスクリプトです。目的があってわざわざ Web ユーザへ公開しているファイルは、OS のパーミッション機構で読み込み可と設定してください。ディレクトリの大半を読み込み不可とし、実行可能ファイルはあったとしても、最低限に抑えましょう。
- クライアント側のキャッシュ-ユーザの多くが、図書館や学校、空港、その他公共のアクセスポイントにある共用のコンピュータから Web アプリケーションにアクセスしています。ブラウザは Web ページを頻繁にキャッシュします。攻撃者はキャッシュにアクセスし、本来そのサイトではアクセスできない所にアクセスしてしまいます。開発者は複数のしくみを使ってください。HTTP ヘッダーやメタタグを使い、ページにある秘密情報がユーザのブラウザに決してキャッシュされないようにしてください。

アプリケーション層でセキュリティをかけるコンポーネントがあり、アクセス制御を厳密に適用するのに役立ちます。また一方パラメタ改竄に関して効果を上げるためには、サイトにとって何が適切なアクセスのリクエストなのかを厳密に定義して、コンポーネントを設定しなければいけません。そのようなコンポーネントを使う場合、それが用意するアクセス制御がサイトで定められたセキュリティポリシーに対しどんな支援をするのか、正確かつ慎重に理解しなければいけません。そして、アクセス制御ポリシーのどこがそのコンポーネントの範疇ではないのか、そのため自分のアプリケーションのコードがどこを扱わなければいけないのかも、同様に理解してください。

管理機能に最も望まれるのは、可能なら管理者に決してサイト表口からのアクセスを認めないことです。インタフェースを有効にすることで、外部から攻撃可能なインターフェースを持つリスクを、組織の大半は許すべきではありません。リモート管理によるアクセスがどうしても必要なら、サイト表口を開けることなく実現可能です。VPN 技術をを使えば、管理者はバックエンドの保護された接続を経由して、外部から企業(もしくはサイト)の内部ネットワークへアクセスできます。

A3 認証とセッション管理が不完全

A3.1 解説

認証とセッション管理には、ユーザ認証とアクティブなセッション管理を扱うすべての要素が含まれます。認証はこの工程の欠くことのできない要素ですが、強固な認証機構でさえ欠陥のある身分証明の管理機能（パスワード変更やパスワード記憶、アカウント更新、その他関連機能）によって穴を開けられます。理由は「通りすがり(walk by)」攻撃が Web アプリケーションの多くに対して行われているからです。たとえユーザが正しいセッション ID を持っていても、アカウント管理機能で再認証を求めてください。

Web のユーザ認証は、普通ユーザ ID とパスワードを使用します。より強固な認証方式には、暗号化トークンやバイオメトリックスを用いたソフトウェア・ハードウェアがあり、製品として利用できます。しかしそのようなしくみは、Web アプリケーションの大部分にとってコスト上見合いません。アカウント管理とセッション管理に欠陥がたくさんあると、ユーザやシステム管理者のアカウントを危険にさらします。身分証明をサイトのあらゆる面から適切に保護するには、認証やセッション管理の設計が複雑になります。開発チームはこれを軽視しがちです。

Web アプリケーションはセッションを確立し、各ユーザから来る一連のリクエストを監視しなければいけません。HTTP にはこの機能がないので、Web アプリケーション自身で行わざるを得ません。Web アプリケーション環境には、セッション機能が用意されているのが普通ですが、開発者の多くは自分でセッショントークンを作りたがります。ただどちらのケースでも、セッショントークンがしっかり保護れていないと、攻撃者はアクティブなセッションを乗っ取っとり、ユーザ識別を推測してしまいます。強固なセッショントークと有効期間中そのトークンを保護するシステムを作成するしくみは、開発者の多くにとっていかんともしがたかったようです。

認証証明やセッション識別すべてが常時 SSL で保護されず、クロスサイトスクリプティングのような他の欠陥による漏洩も防御できていないなら、攻撃者はユーザのセッションを乗っ取っとり、識別を推測できてしまいます。

A3.2 影響を受ける環境

既知の Web サーバやアプリケーションサーバ、Web アプリケーション環境は、不完全な認証やセッション管理の問題に影響されやすくなっています。

A3.3 例と参考文献

- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 6: Authentication and Chapter 7: Session Management: http://www.owasp.org/guide/
- White paper on the Session Fixation Vulnerability in Web-based Applications: http://www.acros.si/papers/session_fixation.pdf
- White paper on Password Recovery for Web-based Applications -- http://fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf

A3.4 脆弱性の判定方法

認証やセッション管理の問題の原因を突き止めるには、コードレビューと侵入テストが向いています。認証機構をあらゆる面から慎重にレビューし、動きがない間(例えばディスク上にある)も何かをしている最中(例えばログイン中)も、常にユーザの身分証明が保護されているようにしてください。ユーザの身分証明を変更するセッション管理機構を残らずレビューして、必ず承認されたユーザだけが変更できるようにしましょう。セッション管理機構をレビューして、セッションの識別子が常に保護され、うっかりもしくは悪意を持った漏洩が最小限になるようにしてください。

A3.5 防御方法

オーダーメードもしくは市販の認証・セッション管理機構を慎重かつ適切に利用すれば、この分野の問題はかなり減少するはずです。まず最初に、ユーザの身分証明管理を安全にする観点からサイトのポリシーを定義し、ドキュメント化します。実装が一貫してこのポリシーを確実に実施することが、安全で強固な認証とセッション管理機構の鍵になります。重要な部分として下記が挙げられます。

- パスワードの強固さ-パスワードには最小文字数と複雑さという規定が必要です。アルファベットや数字、アルファベット外の文字を使用することが、ユーザのパスワードに最低限必要な複雑さです(例えば少なくとも各々一つは使用する)。ユーザには定期的にパスワードを変更させる必要があります。以前のパスワードを再利用させないようにしてください。
- パスワードの利用-一定時間内でユーザがログインできる回数を決め、制限をかけてください。また繰り返し失敗しているログイン動作をログに取りましょう。ログインが失敗している間のパスワードは記録しないでください。というのは、記録してしまうと、ログにアクセスできる誰もにユーザのパスワードが公開されてしまうかもしれないからです。ログインが失敗したなら、ユーザ名やパスワードが間違っていても、システムはユーザ名やパスワードを表示してはいけません。最後にログインが成功した日付や時間、その時間から自分のアカウントにアクセスして失敗した回数をユーザはわかっているはずです。
- パスワード変更の制御-ユーザにパスワードの変更を認めているなら、状況の如何にかかわらずいつでもパスワードを単独で変更できるようにしてください。パスワードを変更する時には(すべてのアカウント情報と同様に)、新旧のパスワードを常にユーザに対して要求するようにしましょう。ユーザが忘れてしまったパスワードを電子メールで送るなら、ユーザが電子メールのアドレスを変更する度に、システムがユーザに再認証を求めてください。さもないと、攻撃者が一時的にセッションにアクセスして(例えば、ユーザがログインしている間にそのコンピュータに近寄って)、電子メールのアドレスを変更し、「忘れてしまった」パスワードを攻撃者に送るよう仕向けられます。
- パスワードの保存-漏洩に備えて、すべてのパスワードはどこに保存されていてもハッシュするか、暗号化しておかなければいけません。ハッシュ方式は非可逆なので選ばれています。パスワードがプレインテキストである必要があるなら、暗号を使用してください。例えばパスワードを使って他のシステムにログインするケースのように。パスワードはソースが何であれ、決して埋め込まないでください。復号鍵はしっかり保護し、横取りされてパスワードファイルを復号するのに間違っても使用されないようにしましょう。
- 経路中の身分証明を保護する-SSL のような手段を使ってログイン手続きを完全に暗号化することが、唯一効果的な技術です。転送前にクライアント側でパスワードをハッシュするような単純な転送方法では、ほとんど防御になりません。ハッシュ方式は簡単に盗聴され、実際のプレインテキストのパスワードが分からなくても再送可能だからです。
- セッション ID の保護-ユーザの全セッションは原則 SSL で保護してください。保護しておけば、セッション ID(例えば、セッション・クッキー)はネットワーク上で横取りできません。横取りされると、セッション ID の漏洩というとても大きいリスクを背負うことになります。パフォーマンスやその他の理由で SSL が利用できないなら、セッション ID 自体を他の方法で保護しなければいけません。まず、決して URL には入れないようにしましょう。ブラウザにキャッシュされて、referrer ヘッダに入れられるか、「知り合い」に間違って転送されてしまうかもしれません。セッション ID は簡単に推測できないように、長くて複雑な乱数にしてください。またセッション中は、セッション ID が頻繁に変更できるようにして、セッション ID の有効期間を短くするようにしましょう。SSL や認証、その他大きな変更が生じた時点で、セッション ID は変更しなければいけません。ユーザが選んだセッション ID は、決して受け取らないようにしてください。
- アカウントリスト-ユーザがサイトにあるアカウント名のリストにアクセスできないように、システムを設計してください。ユーザのリストを表示しなければならないなら、実際のアカウントリストを表示するかわりに、仮名(スクリーンネーム)の形式にしましょう。仮名では、ログインやユーザアカウントを狙ったその他のハックはできません。
- ブラウザのキャッシュ-GET の一部として認証やセッションデータを絶対設定しないでください。代わりに常に POST を使用しましょう。認証ページにはキャッシュさせないタグをあれこれ使い、ユーザのブラウザでバックボタンを押してログインページまで誰も遡れないようにしてください。また以前入力した身分証明を再入力させるようにしましょう。ブラウザの多くは現状自動補完を無効にする機能をサポートし、自動補完キャッシュに身分証明を保存しないようにできます。
- 信頼関係-サイトの構成は、可能な限りコンポーネント間で無条件に信頼させ合わないようにしてください。各コンポネントは相互作用しあう他のコンポネントに対して、特にそうしない理由がなければ(例えば、パフォーマンスや便利なしくみがないといった理由で)自身で認証するようにしましょう。信頼関係が必要なら、手続きと構成で強固なしくみを設け、サイトの構成が時間とともに変わっていっても、その信頼関係が決して悪用されないようにしてください。

A4 クロスサイトスクリプティング(XSS)の欠陥

A4.1 解説

クロスサイトスクリプティング(XSS と言うこともある)の脆弱性は、攻撃者が Web アプリケーションを利用して、スクリプトの形をとった不正なコードを別のエンドユーザに送り込むことで生じます。これらの欠陥は非常に広がっていて、ユーザからの入力を受け、その入力を検証しないまま出力している Web アプリケーションのどの部分でも発生します。

攻撃者はクロスサイトスクリプティングを使って、不正なスクリプトをそうとは知らないユーザに送ります。エンドユーザのブラウザには、スクリプトを信頼すべきではない、と知る手立てがありません。そうしてスクリプトが実行されることになります。なぜなら、ブラウザはそのスクリプトが信頼できるソースから来たと判断するので、ブラウザが保持しそのサイトで利用するクッキーやセッショントークン、その他秘密情報に不正なスクリプトがアクセスできてしまうからです。これらのスクリプトは、HTML ページのコンテンツを書き換えることさえ可能です。

XSS 攻撃は通常 2 つのカテゴリに分けられます。それは蓄積(stored)と折り返し(reflected)です。蓄積攻撃は挿入されたコードが長期間に渡ってターゲットサーバ上のデータベースやメッセージフォーラム、訪問者のログ、コメントフィールド等に留まります。犠牲者が保存情報を要求した時に、サーバから不正なコードを取ってきます。折り返し攻撃は、挿入されたコードが Web サーバから反射されます。エラーメッセージや検索結果、その他の反応といったリクエストの一部としてサーバに送られた一部もしくはすべての入力です。折り返し攻撃は、別経路で犠牲者に送られます。例えば電子メールや他の Web サーバを通じてです。ユーザが不正なリンクをクリックしたり、巧妙にフォームに誘導されたりした場合、挿入されたコードが脆弱な Web サーバに送られ、それがユーザのブラウザに対して攻撃を返します。ブラウザはそのコードが「信頼した」サーバからのものなので実行してしまいます。

XSS 攻撃は、蓄積攻撃であれ折り返し攻撃であれ、結果は同じです。違いといえば、運ばれてくるものがどのようにサーバに到着するかです。「読み込みだけ」や「ブローシュアウェア」なサイトは、折り返しタイプの深刻な XSS 攻撃には影響されない、とは考えないでください。いらいらして頭にくる程度のものから完璧にアカウントさらすものまで、XSS はエンドユーザに様々な問題を起こします。最もシビアな XSS 攻撃は、ユーザのセッションクッキー漏洩に関するものです。この攻撃で攻撃者はユーザセッションを乗っ取り、そのアカウントに成り代わります。他にもエンドユーザのファイル漏洩やトロイの木馬、ユーザを他のページやサイトにリダイレクトする、コンテンツの表示を変えてしまう等といった有害な攻撃があります。XSS の脆弱性によって、攻撃者はプレスリリースやニュース内容を変えて、企業の株価や消費者の信用を落とすことも可能です。製薬関連のサイトで XSS の脆弱性があると、攻撃者が薬の投与情報を変えて、過量摂取させられます。

攻撃者は色々な手段を使って不正なタグを符号化します。例えば、Unicode を使ってリクエストがそれほど疑わしくないようにユーザに見せかけます。この攻撃は多種多様です。<>記号がを全く必要としないものさえあります。そのような訳で、これらのスクリプトを「除去」する試みはうまく行っていないようです。除去する代わりに入力に何が求められているのかを正しく規定し、厳密に検証してください。XSS 攻撃は組み込み JavaScript の形でもたらされるのが一般的です。しかし組み込みの動的なコンテンツなら、どれもが危険をはらんでいます。ActiveX(OLE)や VBscript、Shockwave、Flash 等がそれに該当します。

Web サーバとアプリケーションサーバどちらにも XSS 問題が潜在します。Web サーバとアプリケーションサーバの大部分は、単純な Web ページを生成して様々なエラーを表示します。例えば 404「page not found」、500「internal server error」です。ユーザが URL にアクセスしようとリクエストし、ページが何か情報を表示したら、折り返しタイプの XSS 攻撃に弱いかもしれません。

極めて高い割合でサイトに XSS の脆弱性があります。Web アプリケーションをだまし、不正なスクリプトを中継させる方法は色々とあります。リクエストから不正部分を除去しようとする開発者は、ありそうな攻撃や符号化を見過ごしがちです。攻撃者にとってこれらの欠点を見つけるのは、大した問題ではありません。必要なのはブラウザとちょっとした時間です。ハッカーが XSS 攻撃を巧妙に作り、ターゲットサイトに挿入するツールと同様、欠陥を見つけるのに便利なフリーのツールがいくつもあります。

A4.2 影響を受ける環境

Web サーバとアプリケーションサーバすべてと Web アプリケーション環境は、クロスサイトスクリプティングに影響されやすくなっています。

A4.3 例と参考文献

- The Cross Site Scripting FAQ: http://www.cgisecurity.com/articles/xss-faq.shtml
- CERT Advisory on Malicious HTML Tags: http://www.cert.org/advisories/CA-2000-02.html
- CERT “Understanding Malicious Content Mitigation” http://www.cert.org/tech_tips/malicious_code_mitigation.html
- Cross-Site Scripting Security Exposure Executive Summary: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/topics/ExSumCS.asp
- Understanding the cause and effect of CSS Vulnerabilities: http://www.technicalinfo.net/papers/CSS.html
- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Data Validation http://www.owasp.org/documentation/guide/
- How to Build an HTTP Request Validation Engine (J2EE validation with Stinger) http://www.owasp.org/columns/jeffwilliams/jeffwilliams2
- Have Your Cake and Eat it Too (.NET validation) http://www.owasp.org/columns/jpoteet/jpoteet2

A4.4 脆弱性の判定方法

Web アプリケーションの XSS の欠陥を特定し、それを除去するのが難しい場合があります。コードのセキュリティレビューを行い、HTML 出力に結びつくと思われる HTTP リクエストからの入力箇所すべてを探し出すことが、欠陥を見つける最良の方法です。不正な JavaScript を送り込むのに様々な HTML タグが利用されています。これらの欠陥をスキャンするのに Nessus や Nikto といったツールが役立ちます。しかし、それはうわべをきれいにするだけです。Web サイトに脆弱な部分が一つでもあるなら、高い確率で他の問題も存在します。

A4.5 防御方法

XSS 攻撃から Web アプリケーションを守る最適な方法は、すべてのヘッダー、クッキー、クエリ文字列、フォームフィールド、隠しフィールド(つまりすべてのパラメタ)に対して、何を許可するのかを厳密に定義した上で、確実に検証をかけることです。動的なコンテンツを検証で確認しようとしてはいけません。削除したり、フィルターをかけたり、無害化(sanitize)してもいけません。動的なコンテンツの種類があまりにも多く、符号化する方法もあまりに多数なので、そのようなコンテンツにはフィルターをかけられません。何を許可するかを規定した「肯定による(positive)」セキュリティポリシにしてください。「否定(negative)による」ものや攻撃の形跡に基づくポリシは、維持が困難なだけでなく、不完全になりがちです。

ユーザからの出力を符号化しても、XSS の脆弱性を無効化できます。実行形式でユーザへ送られる挿入スクリプトを防げます。生成された出力に含まれる下記の文字を適切な HTML エンティティに変換することで、javascript を使った攻撃からアプリケーションをかなり保護できるようになります。

これから	これへ
<	<
>	>
(	(
)	)
#	#
&	&

OWASP Fileters プロジェクトでは、XSS 攻撃の挿入を含むパラメタ改竄の防御が楽になるように、いくつかの言語で再利用可能なコンポーネントを作成しています。また OWASP では CodeSeeker もリリースしています。このアプリケーションは、アプリケーションレベルのファイアーウォールです。OWASP WebGoat トレーニング・プログラムには、クロスサイトスクリプティングとデータ符号化に関する講座があります。

A5 バッファオーバーフロー

A5.1 解説

攻撃者はバッファオーバーフローを利用して、Web アプリケーションの実行スタックを壊します。Web アプリケーションへ巧妙に作製した入力を渡し、任意のコードを実行させます-事実上マシンを乗っ取ります。バッファオーバーフローは簡単に発見できず、できたとしても概して悪用は困難です。にもかかわらず、攻撃者は膨大な数の製品やコンポーネントのバッファオーバーフローを見つけてきました。類似タイプの有名な攻撃に書式文字列攻撃(format string attack)があります。

バッファオーバーフローの欠陥は、静的及び動的なコンテンツを公開しているサイトの Web サーバとアプリケーションサーバ製品双方、もしくは Web アプリケーション自身に存在している場合があります。広く利用されているサーバ製品に見つかったバッファオーバーフローは広く知られるところとなり、利用者に相当なリスクを負わせているようです。Web アプリケーションが例えば画像を扱うグラフィック・ライブラリのようなライブラリを使用すると、そのことでバッファオーバーフロー攻撃の可能性が出てきます。

カスタム Web アプリケーションコードにバッファオーバーフローが存在する場合もありますので、Web アプリケーションが遭遇するよりもセキュリティが不十分かもしれません。カスタム Web アプリケーションに存在するバッファオーバーフローの欠陥は、もっと見つけ難いでしょう。ハッカーが特定のアプリケーションに存在するような欠陥を見つけ、悪用するケースは普通ほとんどないからです。カスタムアプリケーションで発見されても、欠陥が悪用される可能性(アプリケーションがクラッシュする以外)はごくわずかです。ハッカーがアプリケーションのソースコードと詳細なエラーメッセージを普通は利用できないからです。

A5.2 影響を受ける環境

既知のほとんどの Web サーバとアプリケーションサーバ、Web アプリケーション環境がバッファオーバーフローの影響を受けやすくなっています。Java と J2EE 環境は例外で、これらの攻撃から影響を受けません(JVM 自身のオーバーフローを除き)。

A5.3 例と参考文献

- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Data Validation http://www.owasp.org/documentation/guide/
- Aleph One, “Smashing the Stack for Fun and Profit”, http://www.phrack.com/show.php?p=49&a=14
- Mark Donaldson, “Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention”, http://rr.sans.org/code/inside_buffer.php

A5.4 脆弱性の判定方法

サーバ製品やライブラリなら、使用している製品の最新のバグレポートに追従してください。カスタムアプリケーションソフトウェアなら、ユーザから HTTP リクエスト経由で来る入力を受け取るコードすべてが任意で大きな入力を確実に扱えるのか、必ずレビューをしましょう。

A5.5 防御方法

Web サーバとアプリケーションサーバ製品、その他インターネット関連システムの最新のバグレポートに追随してください。最新パッチを製品に適用しましょう。サーバ製品とカスタム Web アプリケーションにあるバッファオーバーフローを探し出す、世間でよく利用されるスキャナーを複数使って、定期的に Web サイトをスキャンしてください。

カスタムアプリケーションコードに対しては、HTTP 経由でユーザから来る入力を受け取るコードすべてをレビューする必要があります。またそのような入力に対しては、大きさが適切なのか必ずチェックするようにしてください。そのような攻撃の影響が無い環境でも実行しましょう。受け取れない程大きな入力が、サービス妨害や他の運用上の問題を引き起こすかもしれないからです。

A6 挿入(injection)の欠陥

A6.1 解説

挿入の欠陥によって、攻撃者が Web アプリケーション経由で他のシステムに不正なコードを中継します。これらの攻撃は、システムコール経由でオペレーティングシステムを呼び出したり、シェルコマンド経由で外部プログラムを使用したりします。SQL 経由(つまり SQL 挿入)でバックエンドのデータベースを呼び出すものもあります。Perl や Python、その他の言語で書かれた多数のスクリプトが、ずさんな設計の Web アプリケーションに挿入され、実行されるかもしれません。Web アプリケーションがインタープリタ系を利用している限り、挿入攻撃を受ける恐れがあります。

Web アプリケーションの多くは、オペレーティングシステムと外部プログラムを使ってその機能を実行しています。sendmail はおそらく最もよく実行されている外部プログラムですが、さらにその他のプログラムも利用されています。外部リクエストの一部として HTTP リクエストから来た情報を Web アプリケーションが通す場合、入念に整形しなければいけません。さもないと、攻撃者はスペシャル(メタ)キャラクタを入れて、不正なコマンドやコマンド修飾子を情報に組み込みます。そうなると、Web アプリケーションはチェック無しに外部システムへ情報を渡し、実行してしまいます。

SQL 挿入はそこここに広まっていて、危険なタイプの挿入攻撃です。SQL 挿入の欠陥を悪用するには、Web サーバがデータベースに渡せるパラメタを攻撃者が発見しなければいけません。巧妙に組み込まれた不正な SQL コマンドによって、攻撃者は Web アプリケーションを欺き、不正なクエリーをデータベースに転送します。この攻撃を試してみるのは簡単で、欠陥をスキャンするツールも色々とあります。結果は悲惨で、攻撃者がデータベースの内容を取得したり、改変したり、破壊したりします。

挿入攻撃は発見、悪用とも簡単ですが、捉えどころがありません。結局全体で危険な状況となります。つまりシステムの些細な部分から全体に渡って、クラックされるか破壊されます。いずれにしても外部呼出しはあちらこちらにあるので、Web アプリケーションは高い確率でコマンド挿入の欠陥を抱えているはずです。

A6.2 影響を受ける環境

システムコールやシェルコマンド、SQL リクエストといった外部コマンドの実行を許可している Web アプリケーション環境が皆該当します。コマンド挿入に対して外部呼出しをしてしまう可能性は、どのように呼び出しが作成され、コンポーネントが呼び出されるのかによります。しかし Web アプリケーションをきちんととコーディングしなければ、外部呼出しのほぼ全てが攻撃対象となります。

A6.3 例と参考文献

- 「例」現ユーザのファイルを検索し、別ユーザのファイルにアクセスする通常のシステムコールによって生じる動作を、不正なパラメタが変更するかもしれません(例えばファイル名リクエストの一部に「../」文字列を入れたパス変更)。追加コマンドがシェルスクリプトに渡るパラメタの末尾に付いて、意図したコマンドとともに追加されたシェルスクリプトが実行されます(例えば「; rm –r *」)。SQL クエリーは「制約(constraints)」を WHERE 句に追加することで(例えば「OR 1=1」)、承認されていないデータにアクセスもしくは変更できます。
- OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Data Validation http://www.owasp.org/documentation/guide/
- How to Build an HTTP Request Validation Engine (J2EE validation with Stinger) http://www.owasp.org/columns/jeffwilliams/jeffwilliams2
- Have Your Cake and Eat it Too (.NET validation) http://www.owasp.org/columns/jpoteet/jpoteet2
- White Paper on SQL Injection: http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

A6.4 脆弱性の判定方法

コマンド挿入攻撃にやられやすいか否かを判断する最良の方法は、ソースコード中で外部リソース(例えば、system、exec、fork、Runtime.exec、SQL クエリーもしくはインタープリタへリクエストをしている構文全て)を呼び出している部分全てを調査することで

す。言語の多くは、外部コマンドを実行する複数の方法を用意している点に注意してください。開発者はコードをレビューし、これらの呼び出しに繋がる可能性がある HTTP リクエストからの入力部分すべてを調査しましょう。慎重にこれらの呼び出しを調査し、下記に示すステップに従って確実に防御してください。

A6.5 防御方法

挿入を防ぐ最も簡単な方法は、外部インタプリタへのアクセスを可能な限り避けることです。シェルコマンドとシステムコールの多くには、同様な機能を実現する言語固有のライブラリが用意されています。そのライブラリを使用しても、オペレーティングシステムのシェル・インタプリタは使いません。これでシェルコマンドに関わる大部分の問題を回避できます。

バックエンド・データベースの呼び出しのように、これらの呼び出しをまだ使わざるを得ないなら、データを慎重に検証して、決して不正なコードを入れないようにしてください。全てのパラメタは、実行形式扱いではなくデータ扱いにし、規則に従ってリクエストを構成しましょう。ストアド・プロシージャーや事前に用意された命令文を使用すると、かなり防御が可能になり、入力を確実にデータとして扱えます。これらの手段は外部呼出しに存在するリスクを低くすることはできますが、完全には排除できません。常に入力を検証し、アプリケーションが前提としている入力に必ず合わせて ください。

コマンド挿入に対するもう一つの強力な防御は、機能するのに必要な権限に限定し、Web アプリケーションを動かすことです。従って、Web サーバを root で動かしたり、データベースに DBADMIN でアクセスしたりしないでください。さもないと攻撃者は、管理者権限を横取りし、Web アプリケーションを手中に収めてしまいます。J2EE 環境の中には、Java サンドボックスを利用できるものがあり、これでシステムコマンドの実行を防げます。

外部コマンドを使わざるを得ないなら、コマンドに渡されたどのユーザ情報も厳密にチェックしましょう。呼び出しの間に発生するどんなエラーやタイムアウト、障害も、そのしくみで扱えるようにしてください。

呼び出しからのリターンコードやエラーコードの出力すべてが、実際に発生することになっているプロセスなのか、必ずチェックしてください。少なくともそうすれば、何か良くないことが発生したのが分かります。さもないと、攻撃されても気付かないでしょう。

OWASP Filters プロジェクトでは、挿入の様々な形態を防ぐのが楽になるように、再利用可能なコンポーネントをいくつかの言語で作成しています。また OWASP は CodeSeeker もリリースしています。このアプリケーションは、アプリケーションレベルのファイアーウォールです。

A7 不適切なエラー処理

A7.1 解説

エラー処理が適切でないと、Web サイトに様々なセキュリティ上の問題が発生するかもしれません。最もよく起こる問題は、スタックトレースやデータベースのダンプ、エラーコードのような詳細な内部エラーメッセージがユーザ(ハッカー)に対して表示されてしまう場合です。これらのメッセージは、決して漏らしてはならない実装の詳細を漏らしてしまいます。ハッカーはこのような詳細情報を鍵にして、サイトにある潜在的な欠陥を見つけてしまいます。またそのようなメッセージは一般ユーザにとっては邪魔です。

Web アプリケーションは通常動作している間にも頻繁にエラーを上げてきます。メモリ不足やヌル・ポインタ例外、システムコールの失敗、データベース利用不可、ネットワークのタイムアウト、その他たくさんの状態がエラーとして報告されます。これらのエラーは、十分に考え抜いた上で扱わなければいけません。つまり、ユーザに対しては良く分かるエラーメッセージであり、サイトの管理者に対しての診断情報であり、攻撃者に対しては何も役に立たない情報にするということです。

エラーメッセージが詳細の多くを語っていなくても、メッセージにある矛盾によって、サイトの動作や裏に隠れているどんな情報が何なのか、といった重要な鍵をさらしてしまうかもしれません。例えば、存在していないファイルにユーザがアクセスすると「file not found」と表示されます。ユーザに権限がないファイルにアクセスすると、「access denied」と表示されます。ファイルが存在していること自体ユーザは知らないはずなのに、その不整合によってアクセスできないファイルやサイトのディレクトリ構成の存在の有無がすぐにユーザにばれてしまいます。

エラー処理が適切でないことでセキュリティ上頻繁に問題となるものの一つに、フェイルオープン・セキュリティチェックがあります。セキュリティ機構は皆、特別に許可されていなければアクセスを拒否し、拒否されている間はアクセスを認めないでください。これがフェイルオープンエラーが発生する共通の理由です。他のエラーには、システムをクラッシュさせたり、リソースを著しく消費させたりして、事実上正規のユーザがサービスを受けられなかったり、サービスを受ける機会が減ったりするものがあります。

しっかりしたセキュリティを実施しつつ、ありそうな入力をどれも扱えるものが、エラー処理のしくみとして優れています。エラーメッセージはシンプルに作成し、それをログすることによって、エラーメッセージがサイトのものなのか、ハッキングの結果なのかを検査できます。エラーを処理するには、ユーザが入力したものだけに焦点を当てるのではなく、システムコールやデータベースのクエリー、その他内部機能のような内部コンポーネントが生成するものにも注意してください。

A7.2 影響を受ける環境

Web サーバ、アプリケーションサーバすべてと Web アプリケーション環境が、エラー処理問題の影響を受けやすくなっています。

A7.3 例と参考文献

- OWASP discussion on generation of error codes: http://www.owasp.org/documentation/guide/

A7.4 脆弱性の判定方法

通常は単純なテストによって、様々な入力エラーに対しサイトがどのように応答するのかわかります。さらにテストを行うには内部でエラーを起こして、サイトがどのように動作するのかを見る必要があります。

詳細なコードレビューを行って、エラー処理ロジックを調査するのも有効な手段の一つです。エラー処理はサイト全体に渡って統一しましょう。一つ一つが整合のとれた設計体系の一部になるようにしてください。コードレビューによって、システムが様々なタイプのエラーをどのように扱うつもりなのかを明らかにします。体系だったエラー処理機構がなかったり、複数の機構が存在していたりすると、恐らく問題ありと考えてよいでしょう。

A7.5 防御方法

エラー処理の扱い方について、具体的なポリシーをドキュメント化しましょう。処理するエラーのタイプとそのタイプごとに、ユーザに何の情報が伝えられ、どんな情報をロギングするのか定義してください。開発者は皆そのポリシーを理解し、コードが必ず準拠するようにしましょう。

実装するのに当たっては、起こりうるエラーすべてを安全に処理するようにしっかりと構築してください。エラーが発生した場合は、不必要に内部情報を漏らすことなく、ユーザに役立ち、意図した通りの情報を返しましょう。ある種のエラーはサイトに存在する実装上の欠陥や攻撃の企てを検知するのに役立つので、ログを取るようにしてください。

Web アプリケーション側で何らかの侵入検知機能を備えているサイトはまれです。しかし、Web アプリケーションが失敗の繰り返しを記録し、警告を発することは十分にありえます。Web アプリケーション攻撃の大多数は決して検知されません。それは、そもそも検知する機能を備えているサイトがまれだからです。どうやら Web アプリケーションのセキュリティに対する攻撃はひどく軽視されているようです。

OWASP Filters プロジェクトでは、アプリケーションが動的に作成するページをフィルタリングすることによって、ユーザの Web ページでエラーコードの漏洩を防ぐのが楽になるよう、再利用可能なコンポーネントをいくつかの言語で作成しています。

A8 安全ではない保存

A8.1 解説

Web アプリケーションのほとんどは、秘密情報をデータベースもしくはファイルシステムのどこかに保存する必要があります。その情報はパスワードであったり、クレジットカード番号であったり、アカウントリストであったり、機密情報であったりします。秘密情報の保護には、暗号技術が良く利用されています。暗号は実装して利用するのが比較的簡単ですが、それでも開発者は Web アプリケーションに組み込む際にちょくちょく間違いを犯します。開発者が暗号で得られる保護をあいかわらず軽視していて、サイトを別の側面から安全にすることにそれほど注意を払っていません。良く間違いが起こる場所は限られています。その場所は次の通りです。

- 重要なデータの暗号化が失敗する
- 鍵や証明書、パスワードを安全に保存しない
- メモリにある秘密を適切に保存しない
- いい加減な乱数源を利用する
- いい加減なアルゴリズムを選ぶ
- 新規に暗号アルゴリズムを開発しようとする
- 暗号鍵交換のサポートとその他必須の管理手段を忘れる

これらの弱点は Web サイトのセキュリティに致命的な影響を及ぼします。普通は、サイトで最も秘密に値するものを保護するのに暗号が利用されますが、ある弱点によって全く信頼できなくなるかもしれません。

A8.2 影響を受ける環境h

たいていの Web アプリケーション環境は、何らかの形で暗号をサポートしています。サポートが受けられないケースはまれで、オプションでサードパーティ製品が色々と存在します。暗号を利用して保存情報もしくはやり取りする情報を保護している Web サイトだけが、攻撃の影響を受けます。このセクションでは SSL の使用については触れません。SSL は「A10 設定管理が安全ではない」で触れます。このセクションでは、アプリケーション層のデータを扱うプログラムにおける暗号についてのみ扱います。

A8.3 例と参考文献

- OWASP Guide to Building Secure Web Applications and Web Services http://www.owasp.org/documentation/guide/
- Bruce Schneier, “Applied Cryptography”, 2nd edition, John Wiley & Sons, 1995

A8.4 脆弱性の判定方法

ソースコードを見ずに暗号の欠陥を探し出すのは骨が折れる作業です。しかし、トークンやセッション ID、クッキー、その他の証明書を調べれば、ランダムか否かはっきりします。Web サイトがどのように暗号機能を利用しているのかを明らかにするには、従来の暗号解読方法がすべて生かせます。

間違いなく最も簡単な方法は、コードをレビューしてどのように暗号機能が実装されているのかを見ることです。暗号モジュールの構成や品質、実装を慎重にレビューしましょう。レビューワーには、暗号の利用と欠陥全般についての深い知識が必要です。鍵やパスワード、その他の秘密が、どのように保存、保護され、ロードされてからどのように処理され、メモリからどのようにクリアされるのかについてもレビューしましょう。

A8.5 防御方法

暗号の欠陥を防ぐのに最も簡単な方法は、暗号の利用を最小限にして、本当に必要な情報に限定することです。例えば、クレジットカードの番号を暗号化して保存するよりも、単に番号の再入力が必要になるようにします。また暗号化パスワードを保存しないで、SHA-1 のような一方向関数を使って、パスワードをハッシュするようにしてください。

暗号技術を使わなければいけないなら、オープンに精査された、何も脆弱性が存在しないライブラリを必ず使用してください。使用している暗号機能を分離して、コードを慎重にレビューしましょう。鍵や証明書、パスワードのような秘密は安全に保存してください。攻撃者が手間取るように、重要な秘密を少なくとも 2 か所に分け、処理時に組み合わせましょう。設定ファイルや外部のサーバ、コード自身の中が場所の候補として考えられます。

A9 サービス妨害

A9.1 解説

Web アプリケーションはサービス妨害攻撃を特に受けやすいアプリケーションです。SYN floods のようなネットワーク上のサービス妨害攻撃は別問題なので、このドキュメントでは扱わないことをご理解ください。

Web アプリケーションは、攻撃と通常のトラフィックを見分けられません。区別を困難にしている要素は色々とありますが、数ある理由の中で最も大きいのは、IP アドレスが相手を特定するのに役立たないという点です。HTTP リクエストがどこから来ているのかを知る信頼に足る方法がないため、不正なトラフィックをフィルターするのが大変困難です。分散攻撃に対して、アプリケーションがどのように本当の攻撃と複数のユーザが皆同時にリロードする行為(そのサイトに一時的な問題がある場合に発生するかもしれません)を区別できるのでしょうか。「スラッシュドット効果」もありますし。

大半の Web サーバは定常運用時で数百ユーザを扱えます。単独の攻撃者は一つのホストからアプリケーションを無力にするのに十分足るトラフィックを発生できます。ロードバランシングでこれらの攻撃は困難にはなりますが、不可能になるわけではありません。セッションが特定のサーバと張られている場合は特にそうです。アプリケーションのセッションデータを可能な限り小さくし、新しいセッションの開始を多少なりとも困難にするのは適切な判断です。

攻撃者が必要とされるリソースすべてを使い果たすと、正規ユーザがシステムを利用できなくなります。制限を受けるリソースには、帯域やデータベース接続、ディスク、CPU、メモリ、スレッド、アプリケーション固有のリソースがあります。これらのリソースは、リソースをターゲットにした攻撃によって食い尽くされます。例えば認証無しで掲示板へのリクエストをユーザが流せるサイトは、HTTP リクエスト毎に多量のデータベース・クエリーを発生させるかもしれません。攻撃者は多量のクエリーを簡単に送りつけることでデータベース接続数を一杯にし、正規のユーザがサービスを受ける余地を無くしてしまいます。

特定のユーザに関連したシステムリソースをターゲットにした他の攻撃もあります。例えば、システムがアカウントを無効にしない限り、攻撃者は不正な証明書を送ることで正規ユーザを締め出せるかもしれません。もしくは、攻撃者があるユーザの新しいパスワードをリクエストして、強制的に電子メールのアカウントにアクセスし、アクセスを奪ってしまうことも可能です。さらに、システムがある単独ユーザのリソースを使用不可にすると、攻撃者がそれを利用し、他のユーザがリソースを利用できなくさせるかもしれません。

Web アプリケーションの中には、アプリケーションをすぐにオフラインにしてしまう攻撃に弱いものがあります。エラーを適切に扱えないアプリケーションは、Web システム自体を停止させてしまいます。これらの攻撃の影響は計り知れません。というのも、アプリケーションを利用しているその他すべてのユーザが、即座に利用できなくなるからです。

この種の攻撃は実に様々です。大部分はほんの数行の Perl コードで、低性能のコンピュータから実行できます。これらの攻撃を完璧に防ぐことはできませんが、攻撃の成功を困難にすることはできます。

A9.2 影響を受ける環境

Web サーバ、アプリケーションサーバすべてと Web アプリケーション環境が、サービス妨害攻撃の影響を受けやすくなっています。

A9.3 例と参考文献

- OWASP Guide to Building Secure Web Applications and Web Services http://www.owasp.org/documentation/guide/

A9.4 脆弱性の判定方法

サービス妨害攻撃における一番の問題は、脆弱か否かを判断する点です。負荷テストツール、例えば JMeter は Web トラフィックを生成し、サイトが高負荷時にどのように振舞うのか確実にテストできます。もちろん、アプリケーションが一秒間に何リクエストさばけるのかをテストするのも重要です。一つの IP アドレスからのテストは有効です。サイトにダメージを与えるのに、一人の攻撃者がどのくらいの量のリクエストを送り込まなければいけないかわかるからです。

サービス妨害を引き起こすのに、どのリソースが利用されるのかを見極めるには、各リソースを分析し、使い果たす手段があるかどうかを見つけてください。認証されていないユーザが何をできるのかについて、特に焦点を当てて調査しましょう。しかしユーザ全てが信頼できないなら、認証済みユーザについても同様に調査してください。

A9.5 防御方法

サービス妨害攻撃を防ぐのは困難で、完璧に保護する方法はありません。

一般的には、どのユーザにも本当に必要となる分のリソースを割り当てるようにしてください。認証済みのユーザに対しては、クォータを設定しておくことで、特定のユーザがシステムにかけられる負荷量に制限をかけられます。ユーザのセッションを同期させることで、一ユーザ当たりの処理リクエストを一つだけにすることも、検討しても良いでしょう。また、現在処理中のユーザから別のリクエストが来たなら、それを落とすことも検討に値します。

認証されていないユーザに対しては、不必要にデータベースやその他間違いが起こると大変なことになるリソースにアクセスさせないようにしましょう。認証されていないユーザが、代償が大きい操作を実行できないようにフローを設計してください。コンテンツを生成したりデータベースから引っ張ってくる代わりに、認証されていないユーザが受け取ったコンテンツをキャッシュすることを検討しても良いでしょう。

エラー処理方法をチェックして、エラーがアプリケーション操作全般に全く影響を及ぼさないようにしてください。

A10　設定管理が安全ではない

A10.1 解説

Web アプリケーションのセキュリティにおいて中心的な役割を果たしているのが、Web サーバとアプリケーションサーバの設定です。これらのサーバはコンテンツを提供し、アプリケーションを実行して、コンテンツを生成する役割を担っています。さらにアプリケーションサーバが、Web アプリケーションが利用可能なサービスをたくさん用意する場合がよくあります。例えば、データ保存やディレクトリサービス、電子メール、メッセージング等々です。サーバの設定管理をしっかりしないと、広い範囲でセキュリティ上の問題へと発展します。

Web 開発グループとサイトを運用しているグループが独立しているケースがよくあります。実際問題、アプリケーションを書く人間と運用環境に責任を持つ人間の間には、かなりのギャップがあります。Web アプリケーションのセキュリティ問題はこのギャップに関係があります。必ずプロジェクトの両サイドのメンバーでサイトのアプリケーションのセキュリティを守る必要があります。

サイトのセキュリティを脅かすサーバ設定上の問題は種々様々です。

- サーバのソフトウェアにパッチを当てないことによる欠陥
- ディレクトリ内が見られたり、ディレクトリ間を行き来したりする攻撃を認めてしまうサーバ・ソフトウェアの欠陥や設定ミス
- 不必要なデフォルトファイル、バックアップファイル、サンプルファイル。スクリプトやアプリケーション、設定ファイル、Web ページがこれに該当
- ファイルやディレクトリのパーミッションが適切でない
- 不要なサービスが有効になっている。コンテンツ管理やリモート管理がこれに該当
- アカウントとパスワードがデフォルトのまま
- 利用可能もしくはアクセス可能な管理機能やデバッグ機能
- エラーメッセージの情報が多すぎる(詳細はエラー処理セクションで)
- SSL 証明書と暗号設定の間違い
- 自己署名認証を使用して、認証と Man-In-The-Middle 攻撃を防ぐ
- デフォルトの認証を使用
- 外部システムを使った不適切な認証

簡単に入手できるセキュリティ・スキャンツールを使えば、これらの問題のいくつかを発見できます。発見すれば悪用するのは簡単で、Web サイト全体を危険にさらすことになります。攻撃が成功すると、データベースやネットワークといったバックエンドシステムも危険にさらされるかもしれません。安全なサイトであるには、安全なソフトウェアとともに安全な設定が必要になります。

A10.2 影響を受ける環境

Web サーバ、アプリケーションサーバのすべてと Web アプリケーション環境は設定ミスを受けやすくなっています。

A10.3 例と参考文献

- OWASP Guide to Building Secure Web Applications and Web Services http://www.owasp.org/documentation/guide/
- Web Server Security Best Practices: http://www.pcmag.com/article2/0,4149,11525,00.asp
- Securing Public Web Servers (from CERT): http://www.cert.org/security-improvement/modules/m11.html

A10.4 脆弱性の判定方法

Web サーバとアプリケーションサーバを連携させて強固にしようとしていなければ、おそらく脆弱な状況でしょう。サーバ製品で秀でて安全なのはあったとしてもまれです。自分のプラットフォームの安全な設定をドキュメント化し、頻繁に更新ください。設定ガイドのレビューを定期的に実行し、必ず最新の状態で一貫性を保ちましょう。実際に設置してあるシステムとも比較してください。

加えて、外部から Web サーバやアプリケーションサーバの既知の脆弱性をスキャンするスキャンニングツールがたくさん手に入ります。例えば Nessus と Nikto です。これらのツールを少なくとも月一回は定期的に実行し、すみやかに問題を発見してください。このツールは内部からも外部からも実行しましょう。外部からのスキャンは、サーバが設置してあるネットワークの外部にあるホストから実行してください。内部からのスキャンはターゲットのサーバと同じネットワークから実行しましょう。

A10.5 防御方法

まず第一に、特定の Web サーバとアプリケーションサーバ用にしっかりしたガイドラインを作成してください。アプリケーションが稼動してるホストすべてと開発環境にもこの設定を同様に適用しましょう。設定については、まずベンダーや OWASP、CERT、SANS などのセキュリティ団体から入手できる既存のガイダンスからはじめるようにしてください。そうしてから、自分たちの要求にあったものにして行きましょう。ガイドラインの作成に当たっては、下記のトピックを入れるようにしてください。

- セキュリティ機構すべてを設定
- 不要なサービスすべてを無効に
- 役割、パーミッション、アカウントの設定。デフォルト、アカウントの無効化やそのパスワードの変更
- ロギングと警告

ガイドラインが一度できれば、それを使ってサーバを設定・維持してください。設定するサーバが多数あるなら、設定を半自動もしくは全自動にすることを検討しましょう。既存の設定ツールを使うか、自分で開発してください。たくさんのツールがすでに存在しています。Ghost のようなレプリケーションツールを使って、すでに存在する強固なサーバのイメージを取ってきてもかまいません。そのイメージを新しいサーバにレプリケートしてください。ただご利用の環境でこの方法が動作するとは限りません。

サーバ設定を安全に維持するには、常に目配りが必要です。ある個人やチームにサーバ設定を最新にする仕事をしっかり割り当ててください。メンテナンス作業は下記のようになります。

- セキュリティの脆弱性として公開されている最新版を追いかける
- セキュリティパッチは最新のものを当てる
- セキュリティ設定ガイドラインを更新する
- 内部、外部両者の視点から、定期的に脆弱性をスキャンする
- 定期的にサーバのセキュリティ設定をガイドラインに沿って内部レビューする
- セキュリティ全体の状況について記述した、経営層に対する定期的な報告

結 論

OWASP はこのリストを整理し、Web アプリケーションの弱点についての関心を高めました。これらの脆弱性がインターネットにビジネス戦略を展開しているサービス提供機関や企業に深刻なリスクを与えると、OWASP の専門家たちは結論付けました。Web アプリケーションのセキュリティ問題は、ネットワークのセキュリティ問題と同様に深刻ですが、これまでほとんど注意を払われていませんでした。攻撃者は Web アプリケーションのセキュリティ問題に目を付けはじめ、ツールやテクニックを積極的に磨き、穴を見つけ出しては乗っ取っています。

この Top Ten リストははじまりに過ぎません。これらの欠陥が Web アプリケーションのセキュリティ上の重大なリスクを代表していると我々は確信しています。しかしリストで検討したセキュリティ上危険な分野は、他にもたくさん存在します。やはり Web アプリケーションを設置している組織に重大なリスクをもたらします。この分野に含まれる欠陥は下記の通りです。

- 必要がない不正なコード
- スレッドセーフになっていないコンカレントなプログラミング
- 承認されていない情報収集
- アカウンタビリティの問題と不十分なロギング
- データの不一致
- 破壊されたキャッシュや保存、再利用

この Top Ten リストへのフィードバックを歓迎します。OWASP のメーリングリストに参加して、Web アプリケーションのセキュリティを向上させるお手伝いをしていただけませんか。まずは http://www.owasp.org にアクセスしてください。