Application Level HTTP Proxy

and

Common HTTP Vulnerabilities

Mehmet Nuri Tike Mustafa Turan

09.01.2004

Abstract

Web applications and browsers have inherent vulnerabilities due to rapid development of the Internet technology and HTTP protocol. Application Level HTTP Proxies are used widely to cover and prevent attacks caused by these vulnerabilities. In this paper Application Level HTTP Proxies are explained and then common web vulnerabilities are provided at some detail.

1. Introduction

The Internet gave our computers the capability of reaching the world but at the same time made our computers reachable by the world. Besides its benefits, the Internet introduced several security drawbacks. Today, there are several known vulnerabilities that threaten web services and users that benefit from these. Several technologies are developed against mal-intended attacks over the Web. One of the most common technologies used today is application level HTTP proxy.

In this paper, we will first define and explain Application level Proxies. Then common and popular web vulnerabilities will be shown and explained. While explaining the vulnerabilities, the possible aid of Application Level Proxies for preventing attacks through these vulnerabilities will be given in context. Finally a general working scheme for Application Level Reverse Proxy will be given since most attacks occur due to vulnerabilities at web applications on web servers.

2. Application Level Proxy

Application Level HTTP Proxy is a program that is intended to prevent common HTTP attacks on the web server side or on the client side. An Application Level Proxy is generally an intermediary server that separates outer world (the Internet) from the local network of an enterprise. All HTTP traffic to and from the outer Internet flows through this proxy, so providing single point of access and single point of control. The main advantage of application level proxies is their awareness application specific packets and data.

Application Level HTTP is a general term and actually covers two separate types: Application level forward proxy and application level reverse proxy. We will examine both and give main advantages of both.

2.1 Application Level Forward Proxy

A forward proxy acts as a gateway for a client's browser, sending HTTP requests on the client's behalf to the Internet [17]. The clients are connected to the internet from a single point through the proxy server. Additionally the outer internet does not even see the actual clients and it seems to them that there is a single client which is actually a proxy server. The following figure illustrates a common application level forward proxy.

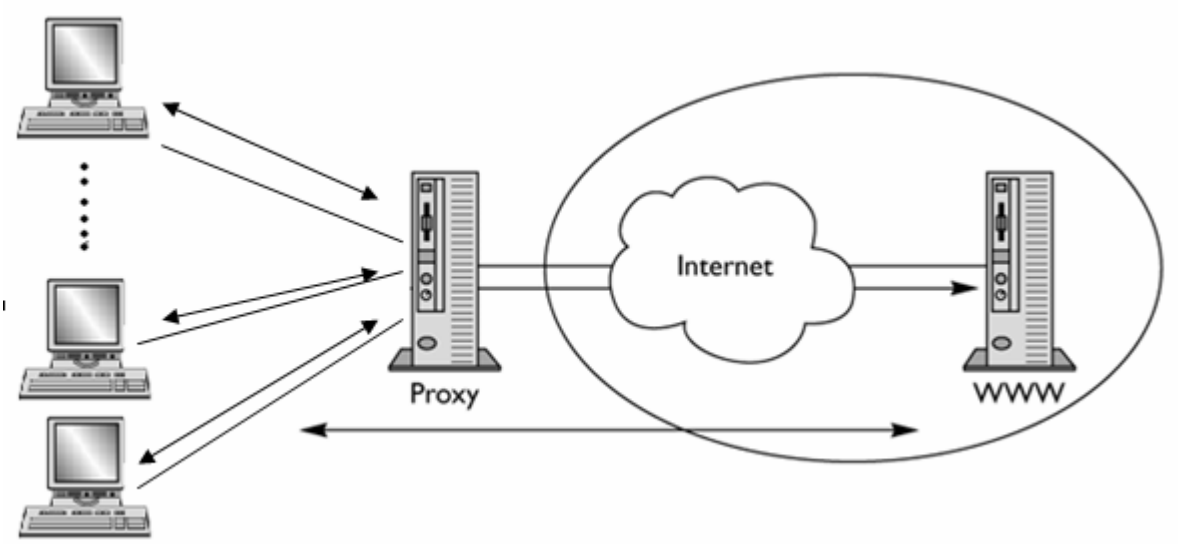

Figure 1. Application Level Forward Proxy for HTTP

The benefits of application level forward proxy are:

1. **Controlling and Filtering**: Most proxy servers have the capability to filter requests from users. Using this feature, an administrator may choose to limit access to Web sites that may not be appropriate for the workplace, such as www.espn.com. All application-layer proxy servers support this feature. It is a key component to ensuring productive use of the Internet connection. Application layer proxies can also filter based on the actual content being transferred, not just the location being transferred to or from.

2. **Conserving the Public IP Address Space**: RFC 1918 (Address Allocation for Private Internet) recommends using private IP addresses for local area networks. This helps to conserve the constantly shrinking pool of public IP addresses. Instead of assigning every node on a network a public IP address, an organization receives only enough public addresses to assign to those systems that will be accessible from the public Internet. The proxy is capable of receiving requests from private IP addresses on a private LAN, retrieving the requested information from the public Internet, and returning it to the original user.

3. **Hiding Internal Addresses**: The above explanation is also a useful technique for security reasons. Because the IP addresses on the LAN are not publicly accessible, no one on the LAN is capable of bypassing the proxy server. Likewise, it is impossible for a hacker to work around a proxy server that is acting as a firewall and access the systems on your LAN directly. Because the InterNIC has not assigned IP addresses to your network, routes to it do not exist in the Internet's routers.

4. **Caching**: As traffic on the Internet continues to increase, bandwidth becomes more and more valuable. Internet expenses and speed considerations may become significant. One way to overcome these is to configure a proxy server local to the LAN and enable caching. Caching is a feature available only in application-layer proxy servers.

5. **Logging**: Because Internet traffic passes through a proxy server, the proxy has the ability to log all requests that it processes. This information is useful in many ways. First, it may be logged to track usage information, which may be used for billing, bandwidth analysis, and needs analysis. It can also be used to pinpoint users who are consuming a large portion of the total bandwidth. Finally, administrators may use log file information to isolate Web sites that should be filtered.

2.2 Application Level Reverse Proxy

A reverse proxy is a gateway application that proxies on behalf of the back-end HTTP server or servers. It acts as the actual web server on the client's aspect. All client requests come to the reverse proxy. The reverse proxy server retrieves the Web resources through the firewall from the actual web servers and forwards them to the requestors. By being in contact with many back-end servers, a reverse proxy can select the appropriate server to forward the request, thus employs load-balancing. In addition, the data returned from the actual web server can be cached locally at the reverse proxy level to improve the access time of the web site. Having a firewall working in cooperation with a reverse proxy server greatly reduces the possibility of exposing the back-end data resources.

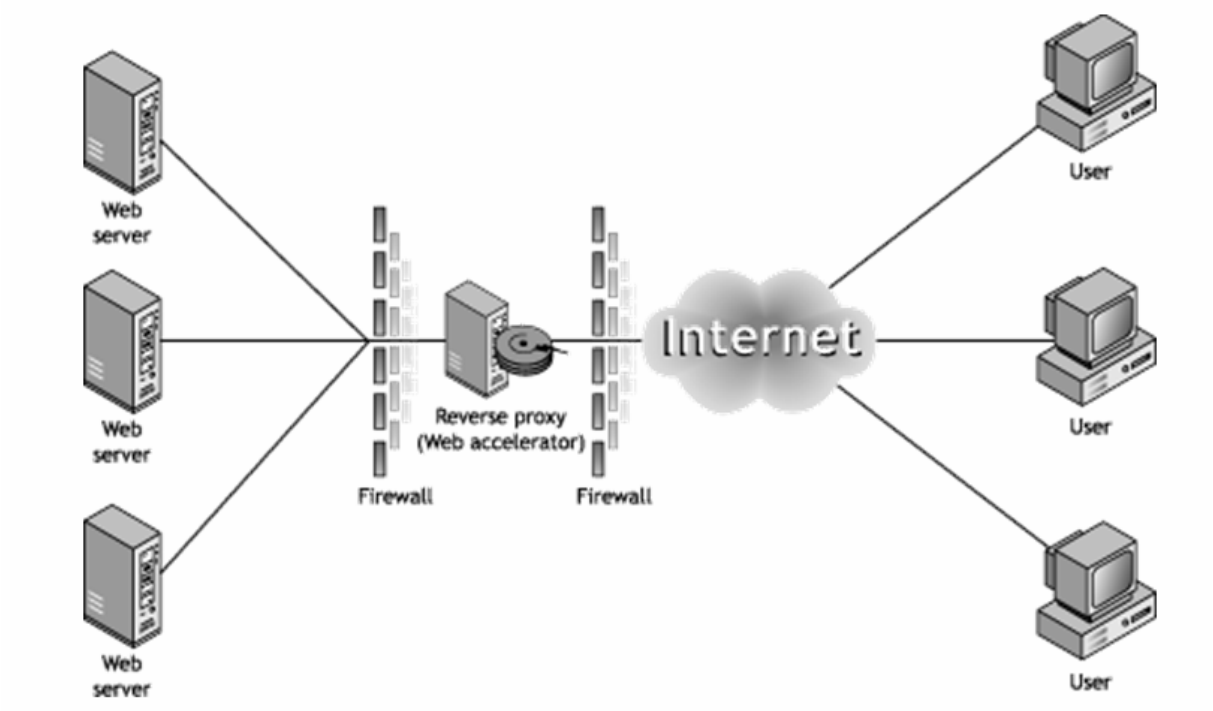

Figure 2. Application Level Reverse Proxy for HTTP

The benefits of application level reverse proxy are:

1. **Load Balancing**: An application level reverse proxy can spread load between several Web servers which it hides from the external Internet. The reverse proxy continually monitors the load on back-end servers and when new requests arrive, forwards them to the appropriate server thus distributing the load of web applications to several servers.

2. **Single Access Point**: One of the biggest benefits of having a reverse proxy configuration is that your clients have a single point of access to your content web servers. This obviously adds a second layer of security that allows you to track and contain an attack against your content servers. System administrators, have a single point of control over that can access the servers and what content is allowed to be accessed by the users. Also single point for logging and auditing activities on the servers.

3. **Hiding Back-end Servers**: A significant benefit of reverse proxy is that outsiders are not aware of the names of the web servers you are proxying. This allows you to easily replace web servers or make host name changes since the rules or "mappings" are handled by the reverse proxy. This does not affect outside clients. Additionally, since all traffic to the web servers pass through the reverse proxy and proxy server appears as the actual web server to the outsiders, the web servers can not be reached directly by outsiders. This provides extra security for the web servers and applications. The addresses of the servers are not open to the public, even may not be from the public address space. A reverse proxy may use NAT method to hide the back-end servers.

4. **Session Management**: The HTTP is a stateless protocol. That is, each request is processed independent of the others. However, a session management scheme may be applied on the reverse proxy by keeping information about the recent requestors of the web servers for with the reverse proxy proxies. So, several requests may be correlated and accepted based on the session info. The requests of the same session may be forwarded to the same back-end server. Also, all requests of the same session may be directly forwarded without if it is known to be reliable.

5. **Request Inspection**: A reverse proxy may inspect the incoming requests with application logic on them. Thus if any threat or malicious content detected or request is supposed to be from a malicious party, the reverse proxy may drop the requests without affecting the back-end servers. This is also true for responses from the servers. If a response is noticed to have malicious content (XSS scripting for example), then the respond may be cancelled to protect the web applications from intentional attacks.

3. Common Vulnerabilities in Web Applications

3.1 Invalidated Input

This is the broadest category of attacks that includes cross site scripting, buffer overflows, SQL injection, cookie poisoning and hidden form field manipulation. Each of these attacks will be described in more detail in this paper.

Web applications use HTTP requests to determine how to respond [1]. HTTP information is encoded in many different ways. Too often, web developers forget to decode all the information before using this information. Parameters have to be simplified to its simplest form before being validated. Attackers can change some part of these requests to bypass the site's security mechanisms. They do this by tampering some part of the HTTP request such as cookies, form fields, headers and fields. Sites must use some filtering mechanisms to protect themselves from malicious input.

Most of the web applications use client side input validation. This increases the web application performance by distributing most of the tasks to clients but is not necessary most of the time because client side validation is easily bypassed by simple tools as simple as telnet and Achilles proxy. Server side controls are necessary to defend against parameter manipulation attacks [1].

We can determine whether a web server is vulnerable to this type of attacks by ensuring that all the parameters are checked in the code. For example in a J2EE application, all parameters can be extracted from HttpServletRequest class.

The best way to prevent parameter tampering is to ensure that all of the parameters are validated before being used. A centralized component or library is likely to be the most effective way as all the checking should be in one place [1].

Parameters should be validated against a specification that defines data type, allowed character set, minimum and maximum length, whether parameter is required

or not, whether duplicates are allowed, numeric range, specific legal values and specific patterns(regular expressions)[1].

Application level proxies that are configured for specific parameter validation services can be another way to prevent these types of attacks. This must be done configuring the proxy for all the parameters with a strict definition of what is valid for the site. This includes properly protecting all types of input including headers, parameters, form fields and cookies.

3.2 Broken Authentication and Session Management

Authentication and session management includes all aspects of handling user authentication and managing active session [3]. Authentication is a critical aspect of this process but even solid authentication mechanisms can be undermined by flawed credential management functions.

Web applications typically use user id and password for user authentication. A wide array of account and session management flaws can result in compromise of system or administration accounts.

Web applications frequently use sessions to provide a user friendly environment to their users. To keep track of the stream of requests that are coming from each single user the session mechanism must be used. HTTP does not provide this capability because of its stateless nature unlike TCP/IP. Therefore, web applications use their own methods to keep track of sessions. The basic idea behind web session management is that the server generates a session identifier (SID) at some early time in user interaction, sends this ID to the user's browser and makes sure that this same identifier is sent back with each subsequent request [4]. If session tokens are not protected properly, an attacker can hijack the identity of a user and behave as if he is the actual user. Creating a scheme to create a strong session tokens and protect them against attacks proven elusive for many developers [3].

To determine whether a web application is vulnerable to this type of attacks, code review and penetration testing can be used. Application should make sure that all of

the authentication tokens are transmitted by SSL and the tokens are stored by secure enough techniques.

To protect the web application and user confidentiality careful and proper use of COTS authentication and session management components should be maintained [3]. Defining and documenting the web applications with respect to securely managing user credentials is a good first step. All account management functions should require reauthentication even if the user has a valid session id. Additionally, session management must be supported by timeout mechanisms of sessions and user should be enforced to reauthenticate after a specified time interval.

Session management is the most critical aspect of this topic. Protection of session ids can be ensured by using SSL from hijacking of session ids. When the performance is concern and use of SSL is not possible, other ways of protection must be used. First, they should never be included in the URL as they can be cached by the browser, sent in the referrer header, or accidentally forwarded to a 'friend' [3]. Session ids must be long and complicated numbers that can not be easily guessed. Session ids must be changed by specific time intervals and when switching to SSL, authenticating and other major transitions. Session ids that are chosen by users should never be accepted to protect against session fixation type of attacks. Authentication and session data should never be submitted as part of a GET request, POST should be used instead. Cookie based session management is proven to be the most secure way of handling sessions when compared to hidden fields and URL arguments[4].

3.3 Buffer Overflows

Attackers use buffer overflows to corrupt the execution stack of a web application [5]. Buffer overflows involve sending long and carefully crafted input to a web application [6]. An attacker can cause the web application to execute arbitrary code as it was part of the web application's code and take the control of the machine. The result is full server compromise or denial of service. Buffer overflows are not easy to discover and even discovered it is difficult to exploit [5].

Buffer overflows can be present in both the web server and application server products that serve the static and dynamic aspects of the site, or the web application itself. Buffer overflows found in widely used server products are likely to become widely known and can pose a significant risk to users of these products [5]. When web applications use some external libraries for using their functionalities (such as using Xine libraries for playing video), they open themselves to buffer overflows of the corresponding libraries.

The overall goal of a buffer overflow attack is to subvert the function of a privileged program so that the attacker can control and use the program for his own purposes. Typically, the attack is made on a root privileged program [7]. The attacker must arrange the program executable code to include the malicious piece of code and get the program to jump to that code in order to execute with suitable parameters. The attacker injects this code or use an existing vulnerable piece of code in the web application. Then the attacker makes the program to jump to that code by using activation records, long jump buffers and function pointers [7].

Generally the buffer overflow attacks focus on the usage of stack which is used for passing parameters to functions and for storing functions' variables. Before calling a function, the parameters and the return address is pushed onto the stack. If the attacker carefully overrides the return address with a selected address, then arbitrary code execution becomes possible [6].

If the programmer allocates a limited size buffer and the attacker knows this size, he can provide an input longer than the defined size, the input probably overflow the buffer and override other parts of the stack. This can be return address of the function and when the function finishes its execution it jumps to a specified location, which is most probably the attacker's malicious code [6].

For example the following HTTP request was used by the Code Red internet worm to exploit a buffer overflow vulnerability within the Microsoft IIS default.ida default page [6]:

```
GET
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6
858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c
3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a
```

Buffer overflow attacks are prevented by enforcing strict boundary checking on input received from users. If there is any problem the input must be truncated or blocked [6]. Type safe languages (such as Java) should be used when implementing the web application. Array bound checking and stack guarding can be used to provide better results. In addition the bug reports and updates should be carefully followed of the software that the web application uses directly or indirectly.

Almost all of the web applications are susceptible to buffer overflows, the notable exception being J2EE platforms which are immune to these attacks (except for buffer overflows in the Java Virtual Machine) [5].

3.4 Improper Error Handling

When detailed internal messages are displayed to the user such as memory dumps, stack traces and error codes, the malintended user can use this information to learn system's vulnerabilities. These messages reveal implementation details that should never be revealed [8].

During normal operation of the web application, it frequently generates error conditions. Out of memory, null pointer exceptions, system call failure, database unavailable, network timeout, and hundreds of other common conditions can cause errors to be generated. These errors must be properly handled with a well thought scheme that will provide meaningful error messages to the user or diagnostic messages to the site maintainers and no useful information to the attacker [8].

Even when error messages don't provide much of detail, inconsistencies in such messages can still reveal important clues on how a site works, and what information is present under the covers. For example, a message that mentions about the internal topological structure of the network or the type of database system that is used by the web site is valuable information for a hacker.

Good error handling mechanisms should be able to handle any set of inputs while enforcing proper security [8]. Simple error messages should be produced and logged so that their cause can be reviewed later. Error handling should not focus solely on input provided by the user but should also include any errors that can be generated by internal components such as database queries, system calls or any other internal functions.

Typically simple testing can determine how a site is vulnerable to different types of error generating inputs. Another valuable approach is the code review about the input and error handling mechanisms of the entire web site software.

A specific error handling policy must be grown up that covers all the aspects and all possible conditions that could likely exist in the normal execution of a web server. The policy should also include what information will be displayed to the user and what information will be hidden and logged. All developers must understand the policy and implement the software according to the policy that is grown up by the structural designers [8].

3.5 Insecure Configuration Management

Web application and application server configurations play a key role in the security of a web application [9]. These servers are responsible for serving content and invoking the functions or applications that generate content. Additionally, application servers provide other external services that web applications use such as directory management, messaging, data storage and more. Failure to manage the proper configurations of these servers can lead to a wide variety of security problems [9].

The key point when securing a web application configuration is the integrity of the application components and the web application environment.
There are a wide variety of security problems that can plague the security of a site [9]. These include:

- Unpatched security flaws in the server software
- Server software flaws or misconfigurations that permit directory listing and directory traversal attacks
- Unnecessary default, backup, or sample files, including scripts, applications, configuration files, and web pages
- Improper file and directory permissions
- Unnecessary services enabled, including content management and remote administration
- Default accounts with their default passwords
- Administrative or debugging functions that are enabled or accessible
- Overly informative error messages (more details in the error handling section)
- Misconfigured SSL certificates and encryption settings
- Use of self-signed certificates to achieve authentication and man-in-the-middle protection
- Use of default certificates
- Improper authentication with external systems[9]

Some of the security assessment tools can already detect most of these vulnerabilities. For example Qualys (www.qualys.com) provides a free online security assessment tool that can find and report some of the vulnerabilities that exist currently in the web application. Detection of these problems is very important for prevention of these security flaws because when they are detected by the attackers, it is easy to use these flaws and these can result in a total compromise of a web site. Successful attacks can also damage backend servers such as database servers and corporate networks.

To prevent these types of attacks, all of the security mechanisms, the configurations and configuration changes must be documented, all the unused services must be

turned off, roles permissions and accounts must be set up and all the alerts must be logged [9].

The web server position on the network topology is an important issue in configuring a web application. The web server machine(s) should be separated on a protected sub network to protect the other systems on the network from outside attacks [10]. In addition, firewall configuration is another important part of this topic.

The web server's object, device and file access controls must be strictly protected and limited. The web server must be configured to execute under a unique individual user and group identity. The protection needed for files and objects must be identified and ensured properly.

Finally the web site manager should maintain an authoritative copy of the web site content on a secure host.

3.6 Broken Access control

Access control defines how a web application grants access to content and functions to some individual users or users in user groups. These checks are performed after authentication, and govern what 'authorized' users are allowed to do. It may sound trivial but actually difficult to implement correctly. Difficulty of implementing a reliable access control mechanism is generally underestimated.

Access control rules are inserted in various locations all over the code. As the site nears deployment, the ad hoc collection of rules becomes so unwieldy that it is almost impossible to understand. It may finally have some flaws in it. Many of these flawed access control schemes are not difficult to discover and exploit. Frequently, all that is required is to craft a request for functions or content that should not be granted. In addition to viewing unauthorized content, an attacker might be able to change or delete content, perform unauthorized functions, or even take over site administration.

One example to access control problem is administrative interfaces that allow site administrators to manage a site over the Internet. Such features are frequently used to allow site administrators to efficiently manage users, data, and content on their site. In many instances, sites support a variety of administrative roles to allow finer granularity of site administration. Due to their power, these interfaces are frequently prime targets for attack by both outsiders and insiders.

Some Access Control Flaws and cautions are as follows:

1. Most web sites use some form of id, key, or index as a way to reference users, roles, content, objects, or functions (since HTTP is a stateless protocol). If an attacker can guess these id's or capture id's by session hijacking or session fixation, and the supplied values are not validated to ensure these are authorized for the current user, the attacker can exercise the access control scheme freely to see what they can access. To avoid such problems web applications should not rely on the secrecy of any id's for protection and single sign on schemes should not be applied.
2. Attackers may try to access files that are normally not directly accessible by anyone, or would otherwise be denied if requested directly. This attack can be done providing relative path information (e.g., "../../some_dir/some_file") as part of a request for information. Such attacks can be submitted in URLs as well as any other input that ultimately accesses a file (i.e., system calls and shell commands). To avoid such circumstances, web administrators should not use system default (web server or operating system) directories and should give specific names for directories and file names.
3. Many web and application servers rely on access control lists provided by the file system of the underlying platform. Even if almost all data is stored on backend servers, there are always files stored locally on the web and application server that should not be publicly accessible, particularly configuration files, default files, and scripts that are installed on most web and application servers. Only files that are specifically intended to be presented to web users should be marked as readable using the OS's permissions mechanism, most directories should not be readable, and very few files, if any, should be marked executable.

4. Browsers frequently cache web pages that can be accessed by attackers to gain access to otherwise inaccessible parts of sites. Developers should use multiple mechanisms, including HTTP headers and meta tags, to be sure that pages containing sensitive information are not cached by user's browsers.

For administrative functions, the primary recommendation is to never allow administrator access through the front door of your site if at all possible. Given the power of these interfaces, most organizations should not accept the risk of making these interfaces available to outside attack.

An application level reverse proxy can assist in the proper enforcement of some aspects of your access control scheme. Again, as for parameter validation, to be effective, the proxy must be configured with a strict definition of what access requests are valid for a site. When using such assistance, you must be careful to understand exactly what access control assistance the proxy can provide for you given your site's security policy, and what part of your access control policy that the component cannot deal with, and therefore must be properly dealt with in your own custom code.

3.7 Cross Site Scripting (XSS)

Cross Site Scripting is a popular attack type where servers that embed browser input into dynamically generated HTML pages can be manipulated into becoming a launch pad for running an attacker's malicious code. Embedding browser input into dynamically generated html pages is a well known scenario for most internet applications like chat rooms, bulletin boards, forums, etc. The key point in such applications is that user supplied input is sent back to the user browser which runs malicious content if supplied in the input and not protected or filtered well by the web server or web application. Static pages are immune to this attack since their content is strictly defined and can not be injected by attackers.

Often attackers will inject JavaScript, VBScript, ActiveX, HTML, or Flash into a vulnerable application to fool a user in order to gather data from them. This is mainly done by a script to call the "document.cookie" object in order to obtain the cookie data. The main reasons for an attacker to apply XSS on a victim are:

1. Account hijacking (obtaining session info form user cookies for impersonation and accessing the user's account on some web application)
2. changing of user settings (for example: changing browser's security settings for applying other kinds of attacks rather easily)
3. cookie theft/poisoning (obtaining account and usage info from cookies or inserting malicious content into cookies so that scripts and malicious content will be permanent and run each time the cookie is used by the web application)
4. New malicious uses are being found every day for XSS attacks.

There are different ways of performing a cross-site scripting attack. Among several, the following two are most known ones:

1. The attack can be applied without forms when applications take data from one person and use it to construct web pages for another user. Examples of this would be Internet based e-mail, chat rooms and web bulletin boards. The attacker writes the script he wants the victim to run and then uses the Web server to send it to the victim. Bu this method, several users may get affected from the same input.
2. Another way of XSS is using forms. There are two ways to submit forms on the Web, "GET" and "POST". An attacker uses the GET method by constructing a URL with malicious content or scripts from the content of a form that the browser then retrieves. The Web server interprets the request for that URL as a submission of the form. The user does not need to be aware that they have submitted a form. The attacker only needs for a user to follow the link that has been created. The user is not aware that they have done anything except retrieve a Web page. The victim Web server receives a form that it believes has been sent by the user. This attack can be done without the attacker using a Website. The URL can be sent in a mail message or posted to a news group.

Since the main reason for XSS attacks is for stealing or manipulating cookies, Microsoft came up with a solution and implemented it in IE 6.0 SP1, the httpOnly cookie attribute [18]. This attribute eliminates the ability of any client-side script from accessing a user's cookies. However a new kind of Cross Site Scripting attack

evolved recently named as Cross Site Tracing [19] which even bypasses the httpOnly cookie. Cross-site tracing bypasses the web application all together and takes advantage of the web server's HTTP TRACE method. By using the HTTP 1.1 TRACE method, the http functions of the browser (XMLHTTP for IE) and (XMLDOM for Mozilla) an attacker can write a basic script to access a user's cookies and a user's basic authentication information. TRACE is any other HTTP method like POST and get where the user request is supplied as a whole in the respond by the web server back to the user browser. This feature is mainly used for debugging web applications. The pages that use TRACE method bypass the httpOnly cookies and enables scripts to access these cookies. The main advice for web administrators is to turn off TRACE method in their servers.

Finally, what can be done to avoid Cross Site Scripting attacks? The main weight at this point is on the shoulders of web developers. Since, XSS is caused by supplying user input back; main caution must be taken at server side. Actually, there are some advices for the users also. Let's first start at the user side.

1. Only follow links from the main website you wish to view.
2. Do not trust links sent in e-mails, web bulletin boards, chat rooms or forums.
3. turn off scripting capabilities in the web browser (however this may cause reduction in functionality)
4. If the Web site allows, always logout before browsing elsewhere.
5. Check all fields of a form before submitting to ensure the information they have filled in is accurate.

And web developers should:

1. Never trust user input and always filter metacharacters. This will eliminate the majority of XSS attacks. Converting < and > to < and > is also suggested when it comes to script output.
2. Filtering < and > alone will not solve all cross site scripting attacks and it is suggested you also attempt to filter out (and) by translating them to (and), , and also # and & by translating them to &#35 (#) and &#38 (&).

The above advices to web developers can be implemented on a reverse proxy to keep safe from XSS attacks even if developers do not consider such attacks. So, while adding extra security beforehand at application level, it eases the job of web application development.

3.8 Injection Flaws

Code injection vulnerabilities allow for injecting arbitrary user-chosen code into a web request where server application feeds these inputs to external systems of the web application. These vulnerabilities arise from nonexistent or poorly designed input validation routines on the server-side. Through such vulnerabilities malicious users can run commands on the server operating system or cause unintended behavior on the database server. Cross Site Scripting vulnerabilities may be thought as an injection flow since script code is injected into form input which is not validated properly. However, XSS has the main purpose of having effect on the users while injection flows attend to have malicious affect on the server side. The main categories of injection flaws are Xpath injection, command injection and Sql injection in the order of priority [20].

XPath is a language for addressing parts of an XML document. An attacker can modify search strings to access unauthorized data in XML documents.

Command injection vulnerabilities are realized when web applications do some action on the local operating system using the input from a user input. For example some applications keep user information in files with the file names that are the same with user's id or name. If a malicious user supplies proper input as a name or id during registration, he can run malicious command on the operating system where he actually has not this authorization. Command injection attacks can have enormous affects since important information can be stolen or damaged by feeding simple commands. So, if some user input is concatenated with a system command string, extreme caution must be taken and should be avoided whenever possible.

The most popular injection flaws are with no argue Sql injection flaws since majority of web applications interfere with and insert user supplied data into database servers.

Web applications often use data read from a client to construct database queries. If the data is not properly processed prior to SQL query construction, malicious patterns that result in the execution of arbitrary SQL commands can be injected [21] [22] [23]. A basic scenario can be the following: a Web site includes a form with two edit boxes in its login.html to ask for a username and password. The form declares that the values of the two input fields should be submitted with the variables strUserName and strPassword to login.cgi, which includes the following code:

SQLQuery = “SELECT * FROM Users WHERE (UserName='” +strUserName + “') AND (Password='” + strPassword + “');”

If GetQueryResult(SQLQuery) = 0 Then bAuthenticated = false;

Else bAuthenticated = true;

If a user submits the username “Wayne” and the password “0308Wayne, ” the SQLQuery variable is interpreted as:

“SELECT * FROM Users WHERE (strUserName= 'Wayne') AND (Password='0308Wayne');

GetQueryResult() is used to execute SQLQuery and retrieve the number of matched records. Note that user inputs (stored in the strUserName and strPassword variables) are used directly in SQL command construction without preprocessing, thus making the code vulnerable to SQL injection attacks. If a malicious user enters the following string for both the UserName and Password fields:

X' OR 'A' = 'A

then the SQLQuery variable will be interpreted as:

“SELECT * FROM Users WHERE (strUserName='X' OR 'A' = 'A') AND (Password='X' OR 'A' = 'A');

Since the expression 'A' = 'A' will always be evaluated as TRUE, the WHERE clause will have no actual effect, and the SQL command will always be the equivalent of “SELECT * FROM Users”. Therefore, GetQueryResult() will always succeed, thus allowing the Web application’s authentication mechanism to be bypassed.

There are several sql injection techniques: Authorization bypass (example described above), using SELECT command, using INSERT command and using SQL Server stored procedures. Details and examples of these techniques are described in [24].

Avoiding sql injection attacks is not an easy issue. Web developers and administrators should consider following when dealing with database queries and administration in their development and maintenance:

1. All client-supplied data needs to be checked for any characters or strings that could be used maliciously. This should be done for all applications, not for those that use sql queries. Input syntax and allowable characters should be defined clearly and all input should be filtered according to this specification. If it is needed to include symbols or punctuation of any kind, make sure to convert them to HTML substitutes, such as &quote; or >.
2. Prefix and append a quote to all user input even if the data is numeric.
3. Limit the rights of the database user. Define authorization on all tables and queries and stored procedures. Do not let a user or an application deal with a data or operation that it is not authorized to do.

3.9 Denial of Service

Denial of Service (DoS) and its more powerful version, Distributed Denial of Service (DDoS), are among the most serious problems in web security. In these attacks, web servers are kept busy buy malicious traffic so that it can not perform its ordinary services to normal users. The main reason for the success of these attacks is that it is not easy for web applications to tell the difference between an attack and ordinary traffic because IP addresses are not useful as an identification credential.

In these types of attacks, attackers can consume limited resources to prevent legitimate users from using the system. These resources can be bandwidth, database connections, CPU, memory or application specific resources. For example, if an application supports some query service for unauthenticated users, a single user can consume a lot of resources by making huge numbers of requests. The results of this scenario with DDoS attack are much horrible. Several hosts try to consume the same resources at the same time.

Avoiding DoS or DDoS attacks are really difficult. Some research has shown that they may be avoided at some level by observing anomaly distribution of packet fields at attack packets [25]. The main approach is that, attack packets are produced automatically using some tools in these attacks. So these packets have very common similarities in their packet field distributions and field values. For example some field can be a constant or the result of a single function or completely random. Such a scheme can be constructed at an application level reverse proxy which drops packets that seem to have a common field pattern which implies that it is an attack packet.

Although it is not possible to have absolute protection against DoS and DDoS attacks, some precautions help great in minimizing the negative effects:

1. Resources allocated to any user should be kept at a limit. Authenticated users have quotas so that the load of a particular user on the system is limited. For example only one request from the same user can be handled at a time.
2. Unauthenticated users should be restricted not to have access to cost-expensive operations. For example, caching can be applied for unauthenticated users so that it will not be necessary to access the database each time.

3.10 Cookie Poisoning

HTTP is a stateless protocol. That means each request is processed independent of the other. However, there is a certain need to keep session information for web applications. Certain session information should be kept when an authenticated user keeps surfing on the same web site. There three ways to keep session information which are cookies, hidden form fields and URLs.

Cookies are the mostly used ones. Cookies are kept in files on the hard drive of the user. The semantics is that cookie is sent to the web application each time a request to it is done. Web server then behaves according to the values in the cookie and sends back to the user's browser to be saved back to hard drive. The problem with cookies arises because they may contain valuable information or attackers about the user. The cookies maintain information that allows the applications to authenticate

the user identity, speed up transactions, monitor behavior, and personalize content presented to the user based on identity and preferences. If an attacker can steal a cookie or write into it, he can impersonate a valid client. Cookies can also be changed so that a malicious script is run each time the user makes a request. For example, depending on the value on the cookie web sites put user name on the page. If a script is fed to the cookie instead of the real name, a malicious code will be executed on the user's browser, making the attack permanent.

Cookie poisoning attacks are generally applied with the help of Cross Site Scripting attacks. After a successful XSS attack an attacker can gather cookie information or manipulate it. So, to avoid cookie poisoning attack, the cautions for XSS attacks should be carefully considered.

4 How Reverse Proxy Works

Reverse proxy works for mainly three purposes, security features to handle transactions, caching and load balancing. We will describe the security issues here.

The reverse proxy takes the request, successfully authenticate the user, determine the physical proximity for addressing for global needs, masquerade as the final URL and translate the local URL back to the one of its backend servers [25]. Reverse proxy takes the request and controls the URL for all of the possible attack types step by step. If at any step the check fails, that it finds a potential attack pattern, it discards the request, writes necessary information to the logs and returns a reject page to the user. When the request passes all of the steps, then it is forwarded to the backend web servers after remapping the outside URL to a local one. For example, when a URL request comes it is subjected to header tests first. Then it is checked for form manipulation type of attacks, hidden field manipulation attacks, cookie poioning attacks and so on. At every step, the proxy makes some web application specific tests for request.

The reverse proxy is positioned in front of the web server. It behaves like the actual web server for the outside world. Generally it makes secure connections to the

outside world and insecure connections to the backend web servers. In some of the configurations it is guarded by two firewalls. One of them is between the outside internet and the reverse proxy and the other stays between the reverse proxy and the backend servers.

5 Conclusion

The Internet has provided a quick and cheap way for accessing information. It has taken an indispensable place in research, education and business. While benefiting from the Internet at these fields, security is an inevitable concern. Several vulnerabilities exist related with the HTTP protocol which is the prime protocol on the Internet for transferring data. Application level proxies provide great assistance in securing Internet applications, servers and clients. In this paper, most significant HTTP vulnerabilities are introduced and application level proxies are mentioned in their security and network shaping capabilities.

REFERENCES

[1] The OWASP Foundation, Unvalidated Input, 2004, http://www.owasp.org/documentation/topten/a1.html

[2] Checkpoint Software Technologies, A Practical Guide to Web Application Security, 2004, http://www.checkpoint.com/products/downloads/Web_Application_Security_Guide_wp.pdf

[3] The OWASP Foundation, Broken Authentication and Session Management, 2004, http://www.owasp.org/documentation/topten/a3.html

[4] Mitja Colsek, Session Fixation Vulnerability in Web Based Applications, December 2002, http://www.acros.si/papers/session_fixation.pdf

[5] The OWASP Foundation, Buffer Overflows, 2004, http://www.owasp.org/documentation/topten/a5.html

[6] Imperva Application Defense Center, Buffer Overflow Attack, 2004, http://www.imperva.com/application_defense_center/glossary/buffer_overflow.html

[7] Buffer overflows: attacks and defenses for the vulnerability of the decade *Cowan, C.; Wagle, P.; Pu, C.; Beattie, S.; Walpole, J.;* Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems] , 2003 Pages:227 - 237

[8] The OWASP Foundation, Improper Error Handling, 2004, http://www.owasp.org/documentation/topten/a7.html

[9] The OWASP Foundation, Insecure Configuration Management, 2004, http://www.owasp.org/documentation/topten/a10.html

[10] CERT Coordination Center, Isolate the Web server from public networks and your organization's internal networks, June 12, 2000, http://www.cert.org/security-improvement/practices/p075.html

[11] CERT Coordination Center, Configure the Web server with appropriate object, device, and file access controls, June 12, 2000, http://www.cert.org/security-improvement/practices/p076.html

[12] CERT Coordination Center, Identify and enable Web-server-specific logging mechanisms, May 3, 2001, http://www.cert.org/security-improvement/practices/p077.html

[13] CERT Coordination Center, Consider security implications before selecting programs, scripts, and plug-ins for your Web server, June 12, 2000, http://www.cert.org/security-improvement/practices/p078.html

[14] CERT Coordination Center, Configure the Web server to minimize the functionality of programs, scripts, and plug-ins, April 30, 2001, http://www.cert.org/security-improvement/practices/p079.html

[15] CERT Coordination Center, Configure the Web server to use authentication and encryption technologies, where required, June 12, 2000, http://www.cert.org/security-improvement/practices/p080.html

[16] CERT Coordination Center, Maintain the authoritative copy of your Web site content on a secure host, June 12, 2000, http://www.cert.org/security-improvement/practices/p081.html

[17] Stricek, Art, January 10,2002, A Reverse Proxy is a Proxy By Any Other Name, http://www.sans.org/rr/whitepapers/webservers/302.php

[18] Microsoft, "Mitigating Cross-site Scripting With HTTP-Only Cookies" URL: http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp?frame=true

[19] Cheryl Stephens, “Cross-Site Tracing – Protecting Businesses from a Simple Attack”, “GIAC Security Essentials (GSEC)”, June 1, 2003.

[20] Alvarez G.,Petrovi S., “A new taxonomy of web attacks suitable for efficient encoding”.

[21] Anley Chris. “Advanced SQL Injection In SQL Server Applications.” An NGSSoftware Insight Security Research (NISR) Publication, 2002.

[22] Cesar Cerrudo. “Manipulating Microsoft SQL Server Using SQL Injection.” Whitepaper, 2002.

[23] Finnigan, P., “SQL Injection and Oracle.” SecurityFocus, 2002. http://online.securityfocus.com/infocus/1644

[24] SPI Dynamics. “SQL Injection: Are Your Web Applications Vulnerable.” SPI Dynamics Whitepaper, 2002.

[25] Jong C. H., Shieh S. P., “Detecting Distributed DoS/Scanning by Anomaly Distribution of Packet Fields”

[26] Pfeifer, Nancy, “Creating a Secure Alternative for Remote Web Access”, October 14, 2002, SANSFIRE 2002 Boston Conference