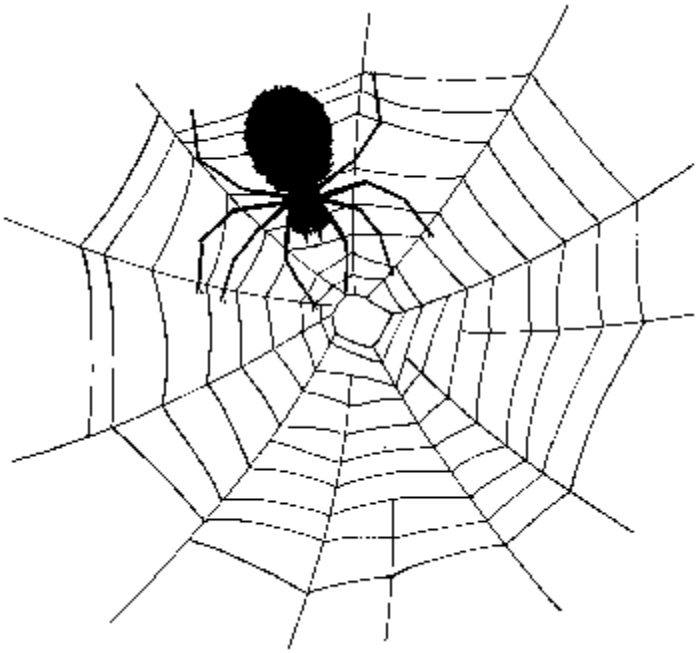
CYBER ATTACKS EXPLAINED: WEB EXPLOITATION



Websites are no longer merely about having an "Internet presence" today, but are also used for commercial transactions and to transfer sensitive data. Such wide usage helps crackers gain more knowledge on vulnerabilities and exploitation techniques than before. Various security studies show that attacking websites to gain fame or money is definitely on the rise. This article explains various Web vulnerabilities and the attacks that exploit them. We will also learn a few techniques that could be incorporated by systems administrators to protect a firm's Web infrastructure.

Before discussing how Web servers are cracked, let us first look into the various components that form a complete Web portal. To begin, a Web server is a service that typically listens on Port 80. The client software, usually a browser, connects to the port and sends HTTP queries. The Web service responds by providing the

requested content, such as HTML, JavaScript, etc. In some cases, the service could be configured to run on ports other than the default one, as a small step towards security. Web servers also may host services such as FTP or NNTP, which run on their own separate default port. Figure 1 shows how Web services map into the OSI layers. The HTTP protocol works at Layer 7, whereas HTTPS (Secure Socket Layer) works at Layer 6.

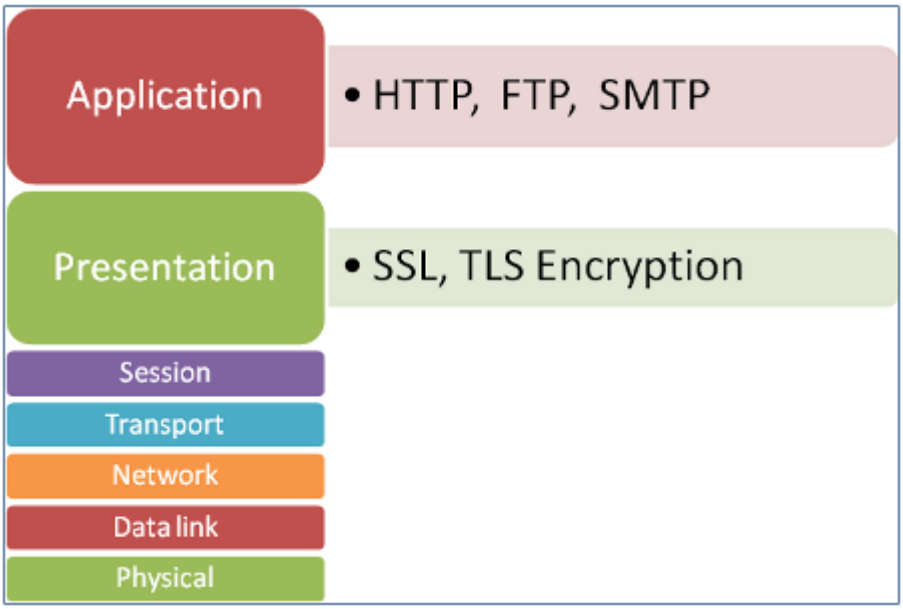


Figure 1: Web services and OSI layers

Modern Web applications often do not just deliver content in the form of simple Web pages. Business logic and data warehousing components such as a database server, application servers and middleware software are also used to generate and provide business-specific data to the website users. These components are usually installed and run on a separate set of servers, and may or may not share storage space with each other. Advanced Web application code may internally call Web services hosted on different servers, and the resultant page is delivered to the client. Web programmers also use cookies to maintain sessions and to store session-specific information in the client browser.

Web hijacking

It is fairly easy to crack a website. A novice may attempt to steal data from a website, whereas a pro may cause serious damage by either defacing the site, or using the Web server to spread a virus. Unlike most other attacks, the techniques used in Web attacks range from Layer 2 to Layer 7 attacks, thus making the Web server susceptible to a wider variety of possible hacking attempts. Since the firewall port must be opened for the Web service (by default, port 80), it cannot help in preventing Layer 7 attacks, which makes the detection of Web attacks difficult. Refer to Figure 2, which shows the typical components used to form Web portal infrastructure.

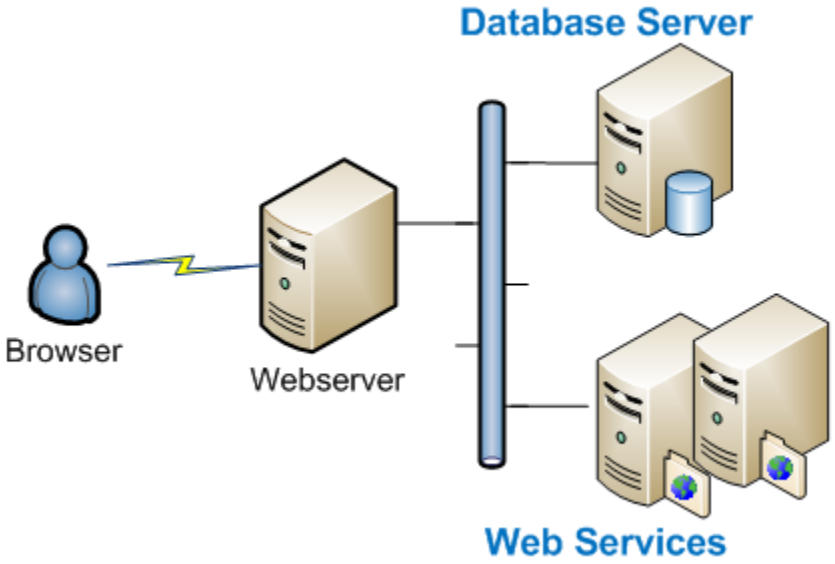


Figure 2: Web portal infrastructure

From the security perspective, each of these components does exhibit some vulnerability, which if exploited, can result in hacking of Web content. Let us now discuss some common yet dangerous attacks in detail.

DoS and sniffing

Since the website is hosted on an IP address open to the Internet, a denial of service attack can easily take the Web server down. Similarly, packet sniffing can easily be used to capture plain-text user IDs and passwords on the wire, if encryption or other security measures are not put in place during Web designing. Almost all Layer 2 and 3 attacks such as packet flooding, SYN flooding, etc, are possible on a website IP and the port on which it is hosted.

HTTP DoS attack

Unlike a network-layer-based denial of service attack, an HTTP DoS attack works at Layer 7. In this type of attack, the website is programmatically crawled to get a list of pages to be accessed, during which the attacker also makes note of the time required by the server to process each page. Those pages that require higher processing time are selected, and multiple HTTP requests are sent to the Web server, each requesting one of the selected pages.

To cater to each request, the Web server starts consuming resources. Upon reaching its resource limits, it eventually gives up and stops responding. Attackers are known to use simple scripts to create a flood of HTTP GET requests to achieve this attack. If the website contains only simple static HTML pages, this attack does not work very well. However, if dynamic pages pull data from a backend database server, this attack can wreak considerable damage.

While it may or may not result in data theft, it certainly shuts the website down, creating a bad user experience and damage to the reputation. Intelligent techniques

must be deployed to detect and stop such attacks, which we will learn about shortly.

Access control exploitation

Usually, in the case of Web portals, a user is given an ID and a password to log in and perform certain functions. Portal administrators are also given their own credentials for maintenance and data management. If the Web services and applications are not designed to be secure from the coding perspective, crackers can exploit those to gain elevated privileges.

For example, if a Web server is not patched with the latest security fixes, which may result in remote code execution, attackers can possibly write a script to exploit that vulnerability and gain access to the server and control it remotely. In some cases, this can happen because the best coding and security practices are not followed, leaving holes in the security configuration, and making the Web solution susceptible to attacks.

Forms input invalidation

Many websites use forms that are filled with information by website users and submitted to the server. The server then validates the input and saves it to the database. The validation process is sometimes delegated to the client browser, or to the database server. If these validations are either not strong enough, or not properly programmed, they can leave security holes that can be exploited by attackers.

For example, if a field such as the PAN number is mandatory, and if the validation for duplicate entries is not done properly, the attacker can programmatically submit

forms with dummy PAN numbers, thus flooding the database with bogus entries. This can eventually help the attacker in planting a denial of service (DoS) attack, by simply querying the page asking for entries that do not exist.

Code exploitation

While this is a bit similar to the previous vulnerability, there is some difference in the way crackers exploit it. Often, programmers make assumptions while setting limits for various user inputs. Typical examples are that the user name should never exceed 50 characters, or that numeric values will always be positive, etc.

These assumptions are dangerous from the security standpoint, because crackers can exploit them. For example, by filling the name field with 100 characters, and thus putting a stress on the datasets, or by providing negative integers in the numeric fields to create incorrect calculation results.

All the attacks mentioned above are used by novice attackers, and following good programming practices can help stop them. Let’s now take a look at technically advanced attacks, which are also common today.

Cookie poisoning

As explained earlier, cookies are small information snippets residing in the browser (on the client machine’s hard drive), and are used to store user session-specific information. It’s the cookie that remembers our shopping cart contents, our preferences and the previous log-in information, in order to provide a rich Web experience.

While it is not very easy to tamper with a cookie, an pro attacker can gain control of it and manipulate its content. Poisoning is achieved via a Trojan or a virus, which sits in the background and keeps forging cookies to gather a user's personal information and send it to the attacker.

Besides this, the virus can also alter the contents of cookies to cause serious problems, such as submitting shopping cart contents in such a way as to deliver the purchased items to a dummy address accessible to the hacker, or to let the browser connect to advertisement servers, which helps the attacker gain money, etc. If session information is stored in the cookie, pro attackers can gain access to it and steal the session, causing a man-in-the-middle attack.

Session hijacking

A Web server talks to multiple browsers at the same time, to take requests in and to deliver the requested content. While each connection is made, the Web server needs to have a way to maintain the uniqueness for each connection. It uses session tokens for this — dynamically generated text strings, which are factors of an IP address, the date, time, etc.

Attackers can steal this token either by guessing programmatically or sniffing on the network, or by performing a client-side script attack on a victim computer. Once stolen, this token can be used to create a fake Web request and steal the victim user's session and information.

URL query-string tampering

Websites that pull data from a database server and show it on the Web page are often found to use query-strings in the main URL. For example, if the website

URL is http://www.a.com/, it may use http://www.a.com/showdata?field1=10&field2=15 to pass field1 and field2 as parameters, with their respective values, to the database — and the resultant output is provided to the browser in the form of a Web page. Having this query-string format exposed so easily, it is possible for users to edit and alter field values beyond the expected limits, or fill them with junk characters. It can further result in users gaining access to information they are not supposed to get. In the worst case, if the field values are userid and password, a brute-force dictionary attack can be used to gain system-level access, merely over HTTP.

Cross-site scripting

This is the most common vulnerability in Web technology, attracting XSS (cross-site scripting) attacks on all major and famous websites. It has been found that a large number of websites are vulnerable to this attack, even today. This vulnerability is a result of improper programming practices and the unavailability of appropriate security measures in a Web infrastructure.
As we know, a client browser maintains its own security in terms of not allowing website contents and the website cookies to be accessed by anyone, except by the users themselves. In this case, the vulnerabilities in a Web application let crackers inject client-side code into the page accessed by users. This code is typically written using JavaScript.

To understand this, consider a page that takes the user name as input, and writes back on the screen “Welcome username”. Let us then suppose the input box is filled up with JavaScript instead, such as what follows:

<script> alert ('You are in trouble') </script>
Here, the Web page may end up executing the script tags, showing the dialogue box message “You are in trouble”. This can be further exploited by the attacker, by

simply poisoning the cookie, stealing the session and injecting this code into the victim user's browser. Upon doing so, the JavaScript code would run in the victim's browser, and create damage to any extent possible.

SQL injection

As we saw earlier, Web portals use database servers in the backend, whereby the Web page connects to the database, queries for data, and presents the fetched data in a Web format to the browser. SQL injection attacks can occur if the input on the client side is not filtered appropriately before it is sent to the database in a query form. This can result in the possibility of manipulating SQL statements, in order to perform invalid operations on the database.

A common example for this attack would be of an SQL server, which is accessed by a Web application, wherein the SQL statements are not filtered by middleware or validation code components. This can lead to the attacker being able to craft and execute his own SQL statements on the backend database server, which could be simple `SELECT` statements to fetch and steal data, or could be as serious as dropping an entire data table. In other cases, the data can be corrupted by populating record sets with malicious and fake content.
Despite the increasing awareness about cyber security, SQL injection attacks are still possible on many websites.

While it is impossible to cover all the possible attacks in this article, let us take a look at a couple of lesser-known attacks, which are increasingly being used to exploit websites.

Slow HTTP attack

While this one is similar to the denial of service attack, the technique is a bit different. It exploits the fact that each HTTP request must be listened to by the Web server. Every Web request starts with a field named content-length, which tells the server how many bytes to expect, and terminates with a carriage-return and line-feed (CRLF) character combination.

The HTTP request is initiated by the attacker with a large value for content-length, and instead of sending the CRLF to conclude the request, it is simply delayed by sending very small amounts of data to the Web server. This makes the Web server wait for more data that is yet to come, to complete the request. This consumes Web server resources.

If the request is delayed to a point that is just less than the session timeout setting on the server, multiple such slow requests can completely consume resources and create a denial of service attack. This can be achieved merely by creating slow and delayed requests from one single browser, which makes it dangerous from the security perspective.

Cryptographic exploitation

Secure websites use SSL certificate-based technology to encrypt data flowing over the network. This leads to an illusion that everything is safe, which is unfortunately not the case. Many shopping-cart applications forget to further encrypt the cookie contents, and leave those in plain text. Though the data over the wire is protected by SSL, running a client-side script to intercept the cookie and read its contents can potentially result in data or session theft.

As for SSL, modern attackers use tools to detect and break into weaker cipher algorithms, thus rendering SSL protection useless, though this is not very common.

Protecting FOSS systems

Apache running on CentOS/Red Hat, Ubuntu and Debian has gained immense popularity among serious FOSS Web infrastructures and solutions. The very first step is to harden the Apache Web service itself; there are numerous guides and examples on the Internet regarding that — for each Linux distro, along with examples.

Disabling ports other than the Web service port, and stopping and disabling unnecessary services is highly recommended. Deploying a well-configured firewall or intrusion-detection device is essential. As mentioned earlier, a simple firewall is not sufficient; hence, a content-filtering firewall equipped to detect Web layer attacks is required.

Securing Web portals is not limited to the Web server, but also extends to components such as database servers, Web services, etc. From the network security stand-point, allowing IP connections to the database only from front-end Web servers is a good idea. Running rootkit detectors, anti-virus tools and log analysers must be a routine job, to prevent hacking attempts.

For advanced security between the middleware and Web server, a stronger authentication mechanism should be in place too. Cookies should be encrypted and SSL deployed, with stronger cipher algorithms.

From the coding perspective, as we learnt earlier, it is essential to use secure programming techniques, and also to follow the best security practices, such as code reviews and penetration testing. Additional processes such as input code validation, server and database-side validation, is recommended too.

Web exploitation is a common way of attacking websites. Due to its easy availability and programmability, FOSS infrastructure is also susceptible to such attacks — and hence, network administrators must understand techniques to protect their infrastructure from information loss or theft.

Source : http://www.opensourceforu.com/2012/03/cyber-attacks-explained-web-exploitation/