Ivan Poliakov
I. H. Olishevskiy, research supervisor
Dnipro University of Technology, Dnipro (Ukraine)

WEB PORTAL SECURITY SYSTEMS

We regularly hear about websites becoming unavailable due to denial of service attacks, or the display of altered (and often corrupted) information on their pages. In other cases, millions of passwords, email addresses, and credit card details have become public, exposing website users to personal embarrassment or financial risk.

Effective website security requires design effort across the whole of the website: in your web application, the configuration of the web server, your policies for creating and renewing passwords, and the client-side code. While all that sounds very ominous, the good news is that if you're using a server-side web framework, it will almost certainly enable "by default" robust and well-thought-out defense mechanisms against a number of the more common attacks. Other attacks can be mitigated through your web server configuration, for example by enabling HTTPS. Finally, there are publicly available vulnerability scanner tools that can help you find out if you've made any obvious mistakes.

The rest of this article gives you more details about a few common threats and some of the simple steps you can take to protect your site.

There are several major site security threats, including Cross-Site Scripting, SQL injection, Cross-Site Request Forgery, etc.

Cross-Site Document Forgery (CSRF)

CSRF attacks allow penetration using another user's credentials without the user's or Australia's knowledge.

This type of attack is best explained at sunrise. An attacker who knows that a particular site allows logged-in users to send money to a specified account using an HTTP POST request that includes the account name and the amount of money. The scammer creates a form that includes his bank details and the amount of money as hidden fields, and emails it to other site users (with a "Submit" button disguised as a get-rich-quick site posted on the site).

If the user clicks data, an HTTP POST request will be sent to the server, discovering a large number of client-side cookies that are associated with the site (adding cookies on the site is a common behavior request). The server checks the cookies and uses them to determine if the user is logged in and has permission to reach the top.

One way to prevent this type of attack is to request a POST request server containing a user-generated secret for specific sites. The secret will be provided by the server when submitting the web form available for transfers. This approach does not allow him to create his own form, because he must know the secret that provides the resources of natural resources. Even if he learns the secret and creates a formula for a particular user, he will no longer be able to attack the same formula for each user.Cross-Site Scripting (XSS)

XSS is a term used to describe a class of attacks that allow an attacker to inject client-side scripts through the website into the browsers of other users. Because the injected code comes to the browser from the site, the code is trusted and can do things like send the user's site authorization cookie to the attacker. When the attacker has the cookie, they can log into a site as though they were the user and do anything the user can, such as access their credit card details, see contact details, or change passwords.

SQL injection

SQL injection vulnerabilities enable malicious users to execute arbitrary SQL code on a database, allowing data to be accessed, modified, or deleted irrespective of the user's permissions. A successful injection attack might spoof identities, create new identities with administration rights, access all data on the server, or destroy/modify the data to make it unusable.

SQL injection types include Error-based SQL injection, SQL injection based on boolean errors, and Time-based SQL injection.

This vulnerability is present if user input that is passed to an underlying SQL statement can change the meaning of the statement.

Broken Authentication

Problems that might occur during broken authentication don't necessarily stem from the same root cause. Rolling your own authentication code is not recommended, as it is hard to get right. There are myriad possible pitfalls, and here are a few:

1. The URL might contain the session ID and leak it in the referer header.
2. Passwords might not be encrypted in storage and/or transit.
3. Session IDs might be predictable, making it a little too easy to gain unauthorized access.
4. Session fixation might be possible.
5. Session hijacking could occur if timeouts are not implemented correctly, or if using HTTP (no SSL security), etc.

Prevention: The most straightforward way to avoid the web security vulnerabilities related to broken authentication is to implement a framework. If you roll your own code, be extremely paranoid and educate yourself on the potential issues that could arise.

Insecure Direct Object References

This is a classic case of trusting user input and paying the price by inheriting a resultant security vulnerability. A direct object reference means that an internal object (e.g., a file or a database key) is exposed to the user, leaving us vulnerable to attack. The attacker can provide this reference, and if authorization is either not enforced or broken, the attacker gets in.

For example, the code has a download.php module that reads and lets the user download files, using a CGI parameter to specify the file name (e.g., download.php?file=something.txt). If the developer omitted authorization from the code, the attacker can now use it to download system files accessible to the user running PHP (e.g., the application code or random server data like backups).

Another example of insecure direct object reference vulnerability is a password reset function that relies on user input to determine their identity. After clicking the valid URL, an attacker could modify the username field in the URL to say something like “admin.”

Incidentally, I have seen both of these examples often “in the wild.”

Prevention: Perform user authorization properly and consistently, and whitelist the choices. More often than not, the vulnerability can be avoided altogether by storing data internally and not relying on data being passed from the client via CGI parameters. Session variables in most frameworks are well suited to this purpose.

Nowadays, with the development of technology, more and more threats appear. The web space is a particularly vulnerable place. Having made the slightest mistakes in protection, irreparable damage may occur to the company and its clients. Therefore, website security is constantly improving and new methods are emerging to protect against the cybercriminal world.