

The Science behind Cross-site Scripting

By Dave Hartley, security consultant at Activity

Recently hundreds of UK Government, school, and University websites have been hacked in order to launch cross-site scripting (XSS) attacks against their visitors.

These attacks occur when a web application is used to send malicious code, generally in the form of a browser side script, to a different end user. The failure of a web-based application to validate user-supplied input before storing it in a data store or returning it to the client system could be enough to let in an XSS attack.

Vulnerabilities

XSS vulnerabilities are used to attack the users rather than the systems or infrastructure components. The malicious script can access any cookies, session tokens, or other sensitive information retained by the user's browser and used with the site. In addition XSS vulnerabilities can be used to perpetrate phishing attacks against users of the application through pop-up boxes and spoofed authentication pages. If session credentials were retrieved, the attacker would then be able to access the application as the targeted user. Other damaging attacks include the disclosure of end user files, or the installation of Trojan horse programs. XSS attacks can be categorised into two general categories - stored and reflected.

Stored or reflected?

Stored attacks involve an attacker injecting code that is permanently stored on the target server, such as in a database, in a message forum, visitor log, or comment field, etc. The application retrieves the malicious script from the server when it requests the stored information on behalf of a user.

Reflected attacks involve injecting code that is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered via another route, such as e-mail or on some other web server. When a user is socially engineered into clicking on a malicious link or submitting a specially crafted form, the injected code is sent to the vulnerable web server, which reflects the attack back to the user's browser.

The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server.

Payloads

XSS payloads can also be embedded into websites and applications using an SQL injection (SQLi). SQLi is an attack in which SQL code is inserted or appended into application/user input parameters that are later passed to backend SQL Server for parsing and execution. Any procedure that constructs SQL statements could potentially be vulnerable as the diverse nature of SQL and the methods available for constructing it provide a wealth of coding options. The primary form of SQLi consists of direct insertion of code into parameters that are concatenated with SQL commands and executed.

It appears that many of the affected websites all use the same website software, however many more websites and applications could be exploited in similar ways. Recently tens of thousands of websites were compromised by the means of an automated SQLi attack. A tool was used to search for potentially vulnerable applications on the Internet and when a vulnerable site is found the tool automatically exploits them. The exploit payload, when delivered, executes an iterative SQL loop that locates every user created table in the remote database and then appends every text column within the table with a malicious client side script. As most database driven web applications use data in the database to dynamically construct web content, eventually the script is presented to a user of the compromised website or application. The tag instructs any browser that loads an infected web page to execute a malicious script that is hosted on a remote server. The purpose of this is to infect as many hosts with malware as possible. It is a very effective attack. A number of prominent websites such as those operated by government agencies, the United Nations and major corporations were compromised and infected by this mass attack. It is difficult to ascertain exactly how many client computers and visitors to these sites were in turn infected or compromised, especially as the payload that is delivered is customisable by the individual launching the attack.

Prevention

If you are a web developer or have developed your own web application the best way to protect a web application from XSS and SQLi attacks is to ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e. all parameters) against a rigorous specification of what should be allowed (white listing). The validation should not attempt to identify active content and remove, filter, or sanitise it (black listing), as there are too many types of active content and too many ways of encoding it to get around filters for such content. Several web application development frameworks such as .NET

provide methods that can transform XSS payloads so that they are not executed; these methods replace characters that have special meaning in HTML, to HTML variables that represent those characters. In order to prevent the existence and exploitation of SQLi vulnerabilities developers should use parameterised stored procedures to prevent variables from being interpreted as SQL commands. Stored procedures encapsulate reusable database procedures that are called with typed parameters. This provides several security advantages, by parameterising input parameters and type-enforcing them, user input is effectively filtered. Using stored procedures alone does not eradicate the vulnerability; if user input is not parameterised or filtered, an attacker could still inject statements.