VIDEO intypedia007en

LESSON 7: WEB APPLICATION SECURITY - INTRODUCTION TO SQL INJECTION TECHNIQUES

AUTHOR: Chema Alonso

Informática 64. Microsoft MVP Enterprise Security

BOB

Hello and welcome to Intypedia. You have probably read or heard recently the news about an attacker being able to steal clients' data from a company through the company's website and you may have wondered how they did it. In this lesson we will try to explain briefly how this is possible. Join us!

SCENE 1. ARCHITECTURE OF A WEB APPLICATION. SECURITY INCIDENTS

ALICE

Many Internet users may wonder how is it possible for an attacker to access the sensitive information of a company, when it isn't published on the Internet through the company's website. For example, gaining access or even modifying the values of a company's database. The technical answer is much simpler than one would think.

BOB

Indeed. Even if the database server storing the company's information is behind a number of firewalls or other protective measures, there is a communication channel between the web server and the database server that is what the attacker uses to reach the stored data. Alice, could you describe to us the basic architecture of a web application that uses a database?

ALICE

Certainly. Although a web application can be as complex as a system would require, typical applications have the following structure. Here you can see that there are three distinct levels according to their roles.

The first one is the user interface manager, that is, the part in charge of human-computer interaction, which in a typical web application would be the web browser.

The second level is where the application logic is located. It will run on web servers and application servers that using different technologies will generate knowledge and process information for a particular purpose.

Finally, we would have the data storage, that is, the information repository where the company's knowledge is stored. This repository can be implemented by an LDAP tree, a relational database, data storage in XML or a simple data file.

There is a connection between each of the adjacent levels. The user, through the information sent by the web browser, interacts with the web server and, in turn, the application logic interacts with the data storage to read and write information on the system.

Of course, each interaction has its own set of protocols and languages. This way, between the interface level and the application logic level, protocols such as http or https can be used to send information. While specific network protocols are used for the queries sent between the web server and the data repositories in their own languages.

For example, if the repository is an LDAP tree, the application logic will send LDAP queries with LDAP search filters. In the case of a relational database, queries in SQL language would be used over protocols such as Tabular Data Stream or Oracle Net. For a repository in XML format we would use XPath queries.

BOB

How interesting! Thus, the answer to how an attacker can recover information from a computer network with a web application seems pretty clear. Chances are there is a security breach in the application's code.

Currently, according to the Open Web Application Security Project (OWASP), the 10 most likely attack vectors for these applications are: A1-Injection, A2-Cross-Site Scripting (XSS), A3-Broken Authentication and Session Management, A4-Insecure Direct Object References, A5-Cross Site Request Forgery (CSRF), A6-Security Misconfiguration, A7-Insecure Cryptographic Storage, A8-Failure to Restrict URL Access, A9-Insufficient Transport Layer Protection, A10-Invalidated Redirects and Forwards.

These attacks attempt to exploit security flaws in web applications (in client or server) with a specific purpose, like for example extracting information from a database. We will have enough time in the future to take a closer look at each one of them, but now we are going to explain the basics of one of the most famous and effective attacks: the SQL code injection.

SCENE 2. CODE INJECTION IN WEB APPLICATIONS

ALICE

One of the key security challenges that web applications face is protecting themselves against injection attacks. Bob, would you mind telling us about the breach exploited by attackers in these types of vulnerabilities?

BOB

Of course, Alice, it is simple to explain. The vulnerability exploited by the attacker is that the web application does not correctly validate the data that is being entered by the user in the interface, which is then used in queries to the data repository. Let's take a look at an example.

Let's suppose that a web application has a user authentication system using a form on which you must enter a username and password. If the programmer doesn't validate properly the information that is entered, but instead, concatenates the data directly building a query that is sent to the database server, the attacker can interact directly with the database. They can use SQL queries, for example and, depending on the environment, remove, modify or delete information.

ALICE

As you can see, the attacker hasn't entered a valid password, but has introduced a condition using the quote character. When this is concatenated in the SQL query that is sent to the database, the result is always compliant, so the application logic always receives an authenticated user from the application.

One may think that the vulnerability exploit consists only of using the quote character effectively, but it's not only that. The vulnerability is, in fact, that the data isn't filtered correctly. If instead of a login form with text fields, the attacker were to exploit the call to a program with a numeric parameter, it would be as easy as introducing a space between the number and the command they want to inject to be able to execute commands on the relational database.

For example, something as simple as http://www.server.com/app.aps?id=1; shutdown – could be an attack to stop the database engine using SQL Injection. And it wouldn't be necessary to enter any quotation marks.

BOB

That's right and this is not just for databases. The attack can be performed successfully on systems with any type of data repository. You would only have to change the types of commands you want to inject and, then, almost any command allowed by the engine could be executed.

ALICE

Of course, the attacker finds certain limitations like the privileges that the web application account has inside the data repository engine or the type of information shown by the application after executing the commands. That is why researchers have developed very refined techniques for extracting the information stored by the system, even without ever seeing the data directly on the website.

BOB

Yes, when extracting data from an engine, there's a distinction between inbound and outbound attacks. The former are those in which the web application itself, handled properly, shows the attacker the extracted information. On the contrary, outbound attacks extract information by generating error messages or triggering events that allow them to extract the data. There are proofs of concept in which data is extracted by generating DNS queries to a controlled server where the extracted data is kept in a log.

ALICE

Furthermore, the attacker doesn't need to view the data drawn from a database through the web application, because of the so-called blind attacks. In these attacks, the data is inferred from the behaviour of a web application after injecting a certain command, as Bob will explain to us now.

SCENE 3. BLIND INJECTION TECHNIQUES

BOB

As you said, Alice, when an attacker can inject any code in the queries that are sent to the databases, they can force changes in the application's behaviour based on the different injections. If an attacker is able to find a constant behaviour when injecting code with True values in all the conditions of the generated query—meaning that all conditions are met—or a constant behaviour when injecting code with a False value—meaning that the condition isn't met—then the attacker can construct a binary logic to extract all the information in the database.

ALICE

For a better understanding of this, let's imagine a simple environment. Suppose a developer has built a web page that displays a news article. The article is stored in a table inside a relational database to which the developer has access using a simple query like:

Select * from articles where id='+article_id+'.

The developer uses the value article_id to select the article and this is picked up by a GET parameter from the client, something like: www.server.com/show_articles.php?article_id=1

Supposing that the construction of the SQL query with the article_id value is done by concatenating a string without filtering the value that is sent for article_id, then we would have a SQL Injection vulnerability.

BOB

With this vulnerability, an attacker could study the application's behaviour in extreme situations with True and False conditions, in order to establish a binary logic.

This way, they could inject something like:

http://www.server.com/show_articles.php?article_id=1 and 1=1

and

http://www.server.com/show_articles.php?article_id=1 and 1=0.

In the first case, the resulting SQL query doesn't change the application's functionality, but in the second case, the SQL query doesn't return any results. This would generate two SQL queries such as:

- Select fields from table where conditions and 1=1
- Select field from table where conditions and 1=0

If the application displays different behaviours to these injections, such as showing an article in the first case and an error message in the second case, then you could extract the information using a blind attack.

ALICE

With these behaviours, the attacker would know that if an article is shown after an injection, this would mean that the condition entered gives a True result. But if an error message is shown, that would mean that the entered condition gives a False result. Therefore, to extract the information, the attacker could make injections of the following type:

http://www.server.com/show_articles.php?article_id=1 and (100=(select top 1 ascii (substring(login,1,1)) from users))

This would be asking if the ASCII value of the first letter of the first user is equal to 100, i.e, if the user's name begins with d lowercase. Using this type of queries it's possible to dump the entire database without even seeing the information. Obviously, it is possible to "guess" the names of the tables. This technique generates a large number of requests to the database that could be recorded in a log or raise the awareness of an intrusion detection system.

BOB

These blind techniques have evolved by analyzing the behaviour of the applications and looking for different behaviours in all the characteristics of the result pages. Even the time delays have been used to recognize True and False values.

SCENE 4. AVOIDING INJECTION TECHNIQUES. MITIGATION

ALICE

As we have already seen, if the application doesn't filter correctly the parameters that come from the user to generate the query, the attacker can find ways to extract the information from the data repository using queries of different complexity.

BOB

This requires a special emphasis on filtering the data that comes from the client. In development environments there are advanced components for creating secure data mining queries, so it is advisable to use these types of components to create queries instead of doing it manually with the concatenation of text strings and variables.

ALICE

In addition, some of these integrated development environments come with static code analysers that are able to detect such vulnerabilities through a comprehensive source code analysis.

BOB

Well, that's all for today. At the Intypedia website you will find additional documents for this lesson. Goodbye!

ALICE

See you next time!

Script adapted to the Intypedia format from the document delivered by Chema Alonso

Madrid, Spain. May 2011

http://www.intypedia.com

http://twitter.com/intypedia