Introduction to the OWASP Top 10 – 2021

Risk A3: Injection

Key Concepts

Key Concept 1: Protocol & Protocol Encapsulation

- Protocol: Set of rules for exchanging information
- Protocol Encapsulation: Wrapping one set of rules into another

Key Concept 2: New malicious commands added to application (hence the term "injection")

- Injected SQL queries will run under the context of the application account allowing read and/or write access to application data and more

Additional Information

A protocol is a set of rules for exchanging information. Protocol Encapsulation is when one set of rules for exchanging information is wrapped into another set. As an example of protocol encapsulation, consider web applications that insert untrusted data into database queries via string building. This allows attackers to execute arbitrary queries against backend databases.

Injection is being able to execute new malicious commands within a web application. For example, it is a problem, if we as users can inject SQL commands of our own in the database of a web application. The reason this is a problem is because with injection we can manipulate information we get from or put into the database.

Definition

Users of the application can operate outside of their defined permissions.

- This changes the execution flow typically leading to:
  - Changes to the execution flow of your web app or API
  - Stealing data from databases and other data sources
  - Running malicious operating system commands
  - Abuse authentication systems
  - Bypass access control

- Many forms of injection depending on the protocol
  - SQL Injection

- Operating System (O/S) Command Injection
- LDAP Injection
- Object Query Injection

Additional Information

The definition of A3 of 2021 – Injection – is an instance where an attacker can supply untrusted data to a web application or API, where the data will be processed by a protocol as a command or a query. Being able to supply untrusted data that is processed as a command or query can lead to changes to the execution flow of your web app or API. Injection is as bad as it gets when it comes to security vulnerabilities.

Depending on the protocol for which we are changing the execution flow, there are many forms of injection Some of the most common ones are SQL and LDAP Injection. There is also Operating System Injection also known as Command Injection and Object Query Injection against queries on objects.

Example

Bad Example

```
SqlCommand objCommand = new SqlCommand (
 "SELECT id,name FROM user_table
 WHERE  username = ' " & Request("NameTextBox.Text") & " ' AND  password = ' "
 &  Request("PasswordTextBox.Text")  & " ' ");
```

Good Example

```
SqlCommand objCommand = new SqlCommand(
  "SELECT id,name FROM user_table WHERE
   Name = @Name AND Password = @Password", objConnection);
objCommand.Parameters.Add("@Name", NameTextBox.Text);
objCommand.Parameters.Add("@Password", PasswordTextBox.Text);
```

Additional Information

In pseudocode, as a bad example of what not to do, we see the building of a SQL command as a concatenated string. This is SQL injection. The user defines both username and password, so an attacker can craft a special query and run other commands. This is not what the developer wanted when they wrote the code.

As a good example, we see the same SQL command, but parameterized. Using a parameterized queries prevents the user input from leading to SQL injection.

Challenges

- It is hard to get protocols to talk to one another correctly
- Limits what data gets passed, seen as an extra step
- Injection vulnerabilities are caused by insufficient user input validation
- Data the user controls is not validated for correctness
- SQL can be used to circumvent authentication, access control and other defensive layers for data theft

Additional Information

We now understand how injection attacks happen, let us look at why they are common.

Building web applications involves many layers and many protocols. It is hard enough getting these protocols to talk to one another correctly. As a result, often limiting what data gets passed where is seen as an additional step, which is sometimes forgotten.

The primary reason for injection vulnerabilities is insufficient user input validation. That means data the user controls while interacting with our web application or API is not validated for correctness.

If you are expecting an e-mail address as part of your form to subscribe to your newsletter, you need to validate that input and check that only valid email addresses can be submitted. Injection attacks happen because it is hard enough to get all the protocols to work together. When an injection attack succeeds, it is because there is not sufficient input validation.

Best Protection Strategies

VECUS

Validate untrusted data
Encode data where necessary
Configure databases using least privilege principle
Use safe APIs for protocol queries
Sanitize data when parameterization is not available

Additional Information

Above you have some of the best protection strategies against A3 of the Top 10 – Injection.