The **Extended HTML Form attack** revisited

By Sandro Gauci, EnableSecurity

Date:23 June 2008

T +44 08458677214 sandro@enablesecurity.com http://enablesecurity.com/

A brief history

In early 2002 I had issued a paper and an advisory detailing an innovative way of performing Cross Site Scripting attacks by making use of various protocols which are not based on HTTP (Hyper Text Transfer Protocol). The paper was titled "Extended HTML Form Attack" and was released under my pseudonym "Obscure" on EyeonSecurity.org. At the time both Internet Explorer and Opera were found to be vulnerable and I had worked with the vendors concerned so that they could fix these issues in a timely fashion. EyeonSecurity hosted some sample code as did other 3rd party security sites to demonstrate this problem.

Summary of the attack

The original paper can still be downloaded from EyeonSecurity[1] and contains more information about the attack. The following is a short description of the attack and how it works.

HTML forms (i.e. <form>) are one of the features in HTTP that allows users to send data to HTTP servers. An often overlooked feature is that due to the nature of HTTP, the web browser has no way of identifying between an HTTP server and one that is not an HTTP server. Therefore web browsers may send this data to any open port, regardless of whether the open port belongs to an HTTP server or not. Apart from that, many web browsers will simply render any data that is returned from the server. One thing to keep in mind is that HTML forms can be hosted on one website (attacker's website) and send data to an open port on a victim server.

When an attacker can control what is returned by the server, the victim becomes vulnerable to security issues such as Cross Site Scripting. In the case of HTTP servers, this is a well known issue and therefore modern web servers do not exhibit this behavior by default. However this is not the case with other kinds of servers such as SMTP (Simple Mail Transfer Protocol) or FTP (File Transfer Protocol) servers, often these servers will echo back error messages containing user input. When this user input can be controlled by the attacker, bad things can happen.

The following is a hypothetical attack scenario:

1. Victim browses to a malicious site while logged in to an Auction Site
2. The victim's web browser sends a POST request as instructed by the malicious site to the auction site's IMAP (Internet Message Access Protocol) server. The POST request contains malicious Javascript code.
3. The IMAP server returns errors containing the malicious Javascript code
4. The Javascript code is rendered on the victim's browser and runs under the Auction Site's security context. The reason why this code runs within the Auction Site's security context is that it conforms to the same origin policy.
5. The attacker obtains the session cookie and can now login as the victim.

So how would an attacker go about exploiting this vulnerability? Requirements:

- A service running on a blocked port not blocked by the web browser
- Victim making use of a vulnerable web browser (non-Mozilla web browser at the time of this writing).
- The service needs to have the following properties
    1. A forward DNS record belonging to the target's domain. Eg. If the attacker wants to steal example.org's cookie, then mail.example.org would be a good target
    2. The service needs to echo back content supplied by the victim's web browser

[1] http://eyeonsecurity.org/papers/extendedformattack.html

Example

An attacker prepares an HTML page with the content as seen in **Figure 1**.

Let's go step by step of how the attack could work in practice.

- A victim making use of Microsoft Internet Explorer points his browser to a malicious website attacker.com. Attacker.com contains the prepared HTML page.
- The victim's web browser renders the HTML page and executes Javascript code which posts the form automatically.
- The web browser sends an HTTP POST request to the IMAPS server on port 993
- The IMAPS server does not like what it gets, since the HTTP request does not contain valid IMAP commands. It replies with error messages containing data supplied by the HTTP request. This is illustrated by **Figure 2**.
- The victim web browser receives the error messages from the IMAPS server and tries to render them as HTML (**Figure 3**).

```
<form method="POST"
enctype="multipart/form-data" action="https://victim-imaps-server:993/" name="demo">
<textarea name="cmd" rows="4" cols="70">
&lt;script&gt;evil script&lt/script&gt;
a002 logout
</textarea>
<br/><input type="submit" value="DoIT!">
</form>
<script>document.demo.submit()</script>
```

Figure 1

```
POST / HTTP/1.1
Host: victim-imaps-server:993
Content-Type: multipart/form-data; boundary=----------NnEwpnt99sDGh2y6HfQ2g9

------------NnEwpnt99sDGh2y6HfQ2g9
Content-Disposition: form-data; name="cmd"

<script>evil script</script>
a002 logout

------------NnEwpnt99sDGh2y6HfQ2g9--
* OK IMAP4rev1 server ready (3.5.27)
POST BAD Unknown command
Host: BAD Unknown command
Content-Type: BAD Unknown command
...
<script>evil script</script> BAD Unknown command
* BYE IMAP4rev1 Server logging out
a002 OK LOGOUT completed
```

Figure 2

Does it still work?

The short answer is yes. Most popular web browsers seem to block certain well known ports. However, there are many cases where an attacker can make use of ports which are not on the blacklist. Internet Explorer and Opera do not block as many ports are Safari or Firefox (Mozilla). During my testing I found that ports such as 993 (standard port for port Secure IMAP) are not blocked on these web browsers. Therefore an attacker can make use of IMAPS servers by redirecting the victim to an HTTPS URL. For example: https://imap.victimservice.org:993/. The screenshot at figure 3 shows a proof of concept example.

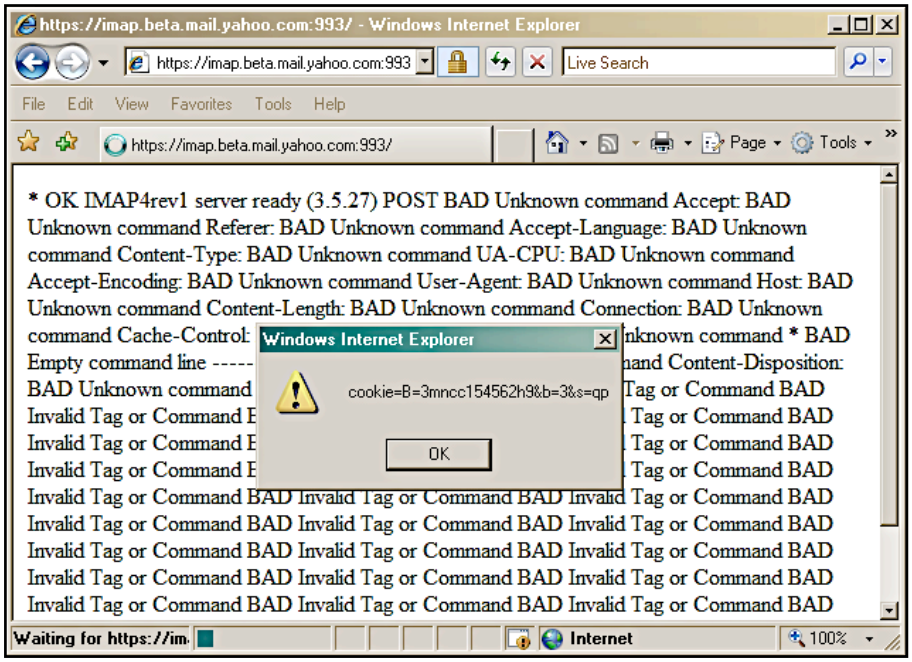

Figure 3

The below table shows how different browsers exhibit different behavior when they connect to a non-HTTP Server. During my testing, Mozilla was the only browser not rendering the content as HTML and therefore it appears to be immune to the attack. Opera 9.50 also showed some inconsistent results, sometimes rendering the HTML content returned from an IMAPS server, other times showing the content as plain text.

Web Browser / Version	Restricts Ports	Renders HTML from non-HTTP Servers
Mozilla-based browsers (Firefox etc)	**Well known ports**	**No**
Opera 9.27 / 9.50	**Some ports**	**Yes (but not consistent)**
Internet Explorer 6 / 7 / 8 on XP / Vista	**Some ports**	**Yes**
Safari 1.3.2 / 3.1.1 on OS X / Windows	**Well known ports**	**Yes**

A complete list of blocked ports for http and https URIs is included in the Appendixes.

Suggestions to the Web browser vendors

The primary reason why various web browsers are vulnerable to this attack is that they do their best to render any response as HTML. Therefore, to solve this security flaw web browsers have to change this behavior. One solution would be to check the HTTP response headers and make sure that they are indeed HTTP/1.0 or 1.1 compliant. This would identify IMAP or FTP servers responding with malicious HTML. When the HTTP response headers are not found, the web browser should not render the content as HTML.

About EnableSecurity

EnableSecurity is dedicated to providing high quality Information Security Consultancy, Research and Development. EnableSecurity is focused on analysis of security challenges and providing solutions to such threats. EnableSecurity works on developing custom targeted security solutions, as well as working with existing off the shelf security tools to provide the best results for their customers. More information about EnableSecurity can be found at enablesecurity.com.

Disclaimer

The information within this document may change without notice. Use of this information constitutes acceptance for use in an AS IS condition. There are NO warranties with regard to this information. In no event shall the author be liable for any consequences whatsoever arising out of or in connection with the use or spread of this information. Any use of this information lies within the realm of the user's responsibility.

Redistribution

Appendixes

Restricted ports per browser

Port	Internet Explorer	Firefox	Opera	Safari
1	FALSE	TRUE	TRUE	TRUE
7	FALSE	TRUE	TRUE	TRUE
9	FALSE	TRUE	TRUE	TRUE
11	FALSE	TRUE	TRUE	TRUE
13	FALSE	TRUE	TRUE	TRUE
15	FALSE	TRUE	TRUE	TRUE
17	FALSE	TRUE	TRUE	TRUE
19	TRUE	TRUE	TRUE	TRUE
20	FALSE	TRUE	TRUE	TRUE
21	TRUE	TRUE	TRUE	TRUE
22	FALSE	TRUE	TRUE	TRUE
23	FALSE	TRUE	TRUE	TRUE
25	TRUE	TRUE	TRUE	TRUE
37	FALSE	TRUE	TRUE	TRUE
42	FALSE	TRUE	TRUE	TRUE
43	FALSE	TRUE	TRUE	TRUE
53	FALSE	TRUE	TRUE	TRUE
70	FALSE	FALSE	TRUE	FALSE
77	FALSE	TRUE	TRUE	TRUE
79	FALSE	TRUE	TRUE	TRUE
87	FALSE	TRUE	FALSE	TRUE
95	FALSE	TRUE	FALSE	TRUE
101	FALSE	TRUE	FALSE	TRUE
102	FALSE	TRUE	FALSE	TRUE
103	FALSE	TRUE	FALSE	TRUE
104	FALSE	TRUE	FALSE	TRUE
109	FALSE	TRUE	TRUE	TRUE
110	TRUE	TRUE	TRUE	TRUE
111	FALSE	TRUE	TRUE	TRUE
113	FALSE	TRUE	TRUE	TRUE
115	FALSE	TRUE	TRUE	TRUE

Port	Internet Explorer	Firefox	Opera	Safari
117	FALSE	TRUE	TRUE	TRUE
119	TRUE	TRUE	TRUE	TRUE
123	FALSE	TRUE	FALSE	TRUE
135	FALSE	TRUE	TRUE	TRUE
139	FALSE	TRUE	FALSE	TRUE
143	TRUE	TRUE	TRUE	TRUE
179	FALSE	TRUE	FALSE	TRUE
210	FALSE	FALSE	TRUE	FALSE
389	FALSE	TRUE	TRUE	TRUE
465	FALSE	TRUE	FALSE	TRUE
512	FALSE	TRUE	TRUE	TRUE
513	FALSE	TRUE	TRUE	TRUE
514	FALSE	TRUE	TRUE	TRUE
515	FALSE	TRUE	TRUE	TRUE
526	FALSE	TRUE	TRUE	TRUE
530	FALSE	TRUE	TRUE	TRUE
531	FALSE	TRUE	TRUE	TRUE
532	FALSE	TRUE	TRUE	TRUE
540	FALSE	TRUE	TRUE	TRUE
556	FALSE	TRUE	TRUE	TRUE
563	FALSE	TRUE	TRUE	TRUE
587	FALSE	TRUE	FALSE	TRUE
601	FALSE	TRUE	TRUE	TRUE
636	FALSE	TRUE	FALSE	TRUE
993	FALSE	TRUE	FALSE	TRUE
995	FALSE	TRUE	FALSE	TRUE
2049	FALSE	TRUE	FALSE	TRUE
4045	FALSE	TRUE	FALSE	TRUE
6000	FALSE	TRUE	FALSE	TRUE