Blog Home (https://researchcenter.paloaltonetworks.com/) > Unit 42 (https://researchcenter.paloaltonetworks.com/unit42/) > OilRig uses RGDoor IIS Backdoor on Targets in the Middle East

OilRig uses RGDoor IIS Backdoor on Targets in the Middle East

By Robert Falcone (https://researchcenter.paloaltonetworks.com/author/robert-falcone/)

January 25, 2018 at 5:00 AM
Category: Unit 42 (https://researchcenter.paloaltonetworks.com/unit42/)
Tags: Middle East (https://researchcenter.paloaltonetworks.com/tag/middle-east/), OilRig (https://researchcenter.paloaltonetworks.com/tag/oilrig/), RGDoor (https://researchcenter.paloaltonetworks.com/tag/rgdoor/), TwoFace (https://researchcenter.paloaltonetworks.com/tag/twoface/)

👁 6,724 👍(3)
(https://twitter.com/home?status=https%3A%2F%2Fresearchcenter.paloaltonetworks.com%2F2018%2F01%2Funit42-oilrig-uses-rgdoor-iis-backdoor-targets-middle-east%2F+OilRig+uses+RGDoor+IIS+Backdoor+on+Targets+in+the+Middle+East)
(https://www.facebook.com/sharer/sharer.php?u=https%3A%2F%2Fresearchcenter.paloaltonetworks.com%2F2018%2F01%2Funit42-oilrig-uses-rgdoor-iis-backdoor-targets-middle-east%2F) (https://www.linkedin.com/shareArticle?mini=true&url=https%3A%2F%2Fresearchcenter.paloaltonetworks.com%2F2018%2F01%2Funit42-oilrig-uses-rgdoor-iis-backdoor-targets-middle-east%2F&title=OilRig+uses+RGDoor+IIS+Backdoor+on+Targets+in+the+Middle+East&summary=&source=)
(//www.reddit.com/submit)

Summary

While investigating files uploaded to a TwoFace (https://researchcenter.paloaltonetworks.com/tag/twoface/) webshell, Unit 42 discovered actors installing an Internet Information Services (IIS) backdoor that we call RGDoor. Our data suggests that actors have deployed the RGDoor backdoor on webservers belonging to eight Middle Eastern government organizations, as well as one financial and one educational institution.

We believe the actors deploy RGDoor as a secondary backdoor to regain access to a compromised webserver in the event a victim organization detects and removes the TwoFace shell. We do not have HTTP logs that show the actor interacting with RGDoor, so we do not know the activities the actors carry out using the backdoor. However, we were able to create a client application to interact with RGDoor for testing purposes, which allowed us to interact with the backdoor to see how it operates on an IIS server.

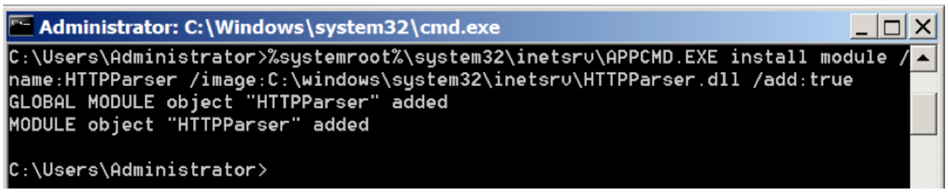
RGDoor

Unlike TwoFace, the actors did not develop RGDoor in C# to be interacted with at specific URLs hosted by the targeted IIS web server. Instead, the developer created RGDoor using C++, which results in a compiled dynamic link library (DLL). The DLL has an exported function named “RegisterModule”, which is important as it led us to believe that this DLL was used as a custom native-code HTTP module that the threat actor would load into IIS. Starting with IIS 7 (https://msdn.microsoft.com/en-us/library/bb757040.aspx), developers could create modules in C++ that the IIS webserver would load to extend IIS’ capabilities, such as carry out custom actions on requests. The fact that RGDoor is an IIS HTTP module suggests that there is no visual representation of the shell for actors to interact with, which also differs from TwoFace’s interface that actors can interact with by visiting the URL to the TwoFace ASPX file.

According to Microsoft’s documentation (https://technet.microsoft.com/en-us/library/cc771133(v=ws.10).aspx), native-code modules can be installed either in the IIS Manager GUI or via the command-line using the “appcmd” application. While we do not have logs to determine exactly how the actors install RGDoor, the actor can use the TwoFace webshell to install the RGDoor module via the command line. The following command could be used to install the HTTP module on an IIS server:

```
%systemroot%\system32\inetsrv\APPCMD.EXE install module /name:[module name] /image:[path to RGDoor DLL] /add:true
```

We confirmed the command above successfully installed the RGDoor backdoor in our test environment. Figure 1 shows the specific command we used to install RGDoor on our IIS server.

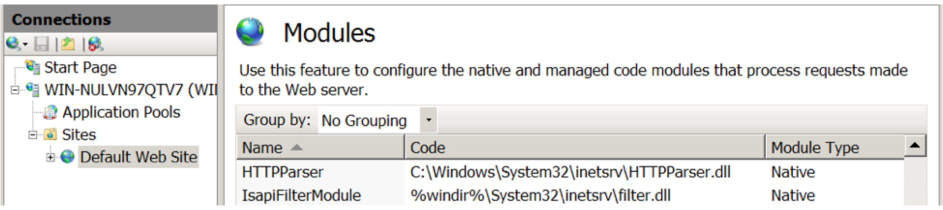


```
C:\Users\Administrator>%systemroot%\system32\inetsrv\APPCMD.EXE install module /
name:HTTPParser /image:C:\windows\system32\inetsrv\HTTPParser.dll /add:true
GLOBAL MODULE object "HTTPParser" added
MODULE object "HTTPParser" added

C:\Users\Administrator>
```

Figure 1 Command-line used to install RGDoor module into IIS

We confirmed RGDoor installed correctly into IIS by checking the HTTP Modules display in IIS Manager. Figure 2 shows the RGDoor DLL (HTTPParser.dll) was loaded into IIS using the module name HTTPParser.



Use this feature to configure the native and managed code modules that process requests made to the Web server.

Group by: No Grouping

Name	Code	Module Type
HTTPParser	C:\Windows\System32\inetsrv\HTTPParser.dll	Native
IsapiFilterModule	%windir%\System32\inetsrv\filter.dll	Native

Figure 2 Installed RGDoor module displayed in the HTTP Modules list in IIS Manger

Listening for Commands

We analyzed RGDoor samples and found that the “RegisterModule” function does very little other than calling the IHttpModuleRegistrationInfo::SetRequestNotifications method. According to MSDN (https://msdn.microsoft.com/en-us/library/ms693045(v=vs.90).aspx), the SetRequestNotifications function has the following prototype, which allows a developer to configure the module to handle GET requests (dwRequestNotifications) and/or POST requests (dwPostRequestNotifications):

```
SetRequestNotifications(

        IN IHttpModuleFactory * pModuleFactory,

        IN DWORD                dwRequestNotifications,

        IN DWORD                dwPostRequestNotifications

    )
```

In RGDoor, the code calls this function with arguments that ignore inbound HTTP GET requests, but act on all HTTP POST requests seen by the IIS server, even POST requests issued over HTTPS. RGDoor calls this function with the following arguments, the third of which (dwPostRequestNotifications) is set to “RQ_BEGIN_REQUEST”, which is an event triggered immediately after IIS receives the POST request:

```
SetRequestNotifications(pModuleFactory,0,RQ_BEGIN_REQUEST)
```

RGDoor is notified immediately when the IIS server receives an inbound HTTP POST request. RGDoor parses these POST requests, specifically looking for the HTTP “Cookie” field by accessing the HTTP header with the following function call:

```
pHttpContext->GetRequest()->GetHeader(“Cookie”,NULL)
```

After accessing the cookie field, RGDoor parses this field by looking for the string “RGSESSIONID=”, which is the basis for the name RGDoor. If present, the code uses the two-bytes immediately following the “RGSESSIONID=” string as a decryption key, specifically treating the two character bytes as a single hexadecimal byte. For instance, the two-byte key of “00” would represent the hexadecimal value of 0x00, where “FF” would represent 0xff, and so on. The key is followed by a Base64 encoded string that contains ciphertext. The following represents the structure of the cookie filed in inbound RGDoor requests:

```
RGSESSIONID=[two-bytes for key][Base64 encoded ciphertext]
```

RGDoor decodes the Base64 encoded string and then decrypts the decoded string using a custom algorithm. The custom algorithm iterates through the ciphertext using the ‘pxor’ instruction to XOR each byte of ciphertext with the single-byte hexadecimal value from the two-byte character key provided in the cookie field.

The code will parse the cleartext looking for one of three commands: “cmd$”, “upload$” and “download$”. The code treats the string immediately following the command as the command’s argument. Table 1 provides the arguments and further details on the three commands.

Command	Description

cmd$[command to execute]	Uses “popen” to run the specified command. It enters a loop that calls “fgets” to get the command results. The loop finishes when a call to “feof” designates the end of the command results, which are relayed back to the actor via the HTTP response.
upload$[path to file]	Gets the length of uploaded data by checking the “Content-Length” field within the HTTP request. Uses the IHttpRequest::GetRemainingEntityBytes and IHttpRequest::ReadEntityBody methods to obtain base64 encoded data within the body of the HTTP POST request. The code decrypts the decoded data using the XOR algorithm and writes the data to the specified file. It will respond to this request with either “write done \r\n” or “can’t open file: “.
download$[path to file]	Reads a specified file and encrypts it with the XOR algorithm, Base64 encodes the ciphertext and sends the data back to the actor via the HTTP response.

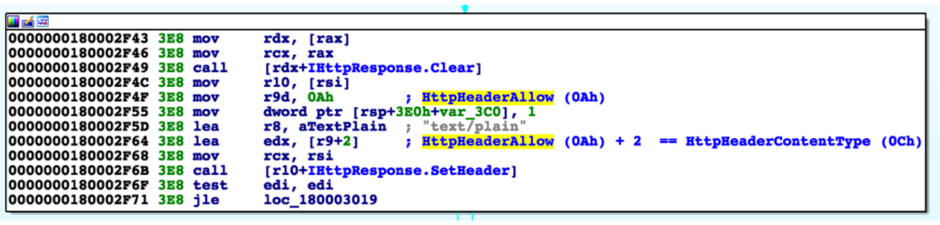
Table 1 Commands available within RGDoor

Responding to Commands

When responding to inbound requests, the code will clear the response that IIS would have responded to the HTTP POST request by calling the following functions:

`pHttpContext->GetResponse()->Clear()`

RGDoor then constructs its own HTTP response by first setting the “Content-Type” field within the HTTP header to “text/plain”. Figure 3 shows the code that uses the IHttpResponse::SetHeader method to set the “Content-Type” field within the HTTP response to “text/plain”, specifically by using a value of 0xC (0xA + 2) for the ulHeaderIndex within HTTP_HEADER_ID enumeration.



```
0000000180002F43 3E8 mov     rdx, [rax]
0000000180002F46 3E8 mov     rcx, rax
0000000180002F49 3E8 call    [rdx+IHttpResponse.Clear]
0000000180002F4C 3E8 mov     r10, [rsi]
0000000180002F4F 3E8 mov     r9d, 0Ah          ; HttpHeaderAllow (0Ah)
0000000180002F55 3E8 mov     dword ptr [rsp+3E0h+var_3C0], 1
0000000180002F5D 3E8 lea     r8, aTextPlain    ; "text/plain"
0000000180002F64 3E8 lea     edx, [r9+2]       ; HttpHeaderAllow (0Ah) + 2  == HttpHeaderContentType (0Ch)
0000000180002F68 3E8 mov     rcx, rsi
0000000180002F6B 3E8 call    [r10+IHttpResponse.SetHeader]
0000000180002F6F 3E8 test    edi, edi
0000000180002F71 3E8 jle     loc_180003019
```

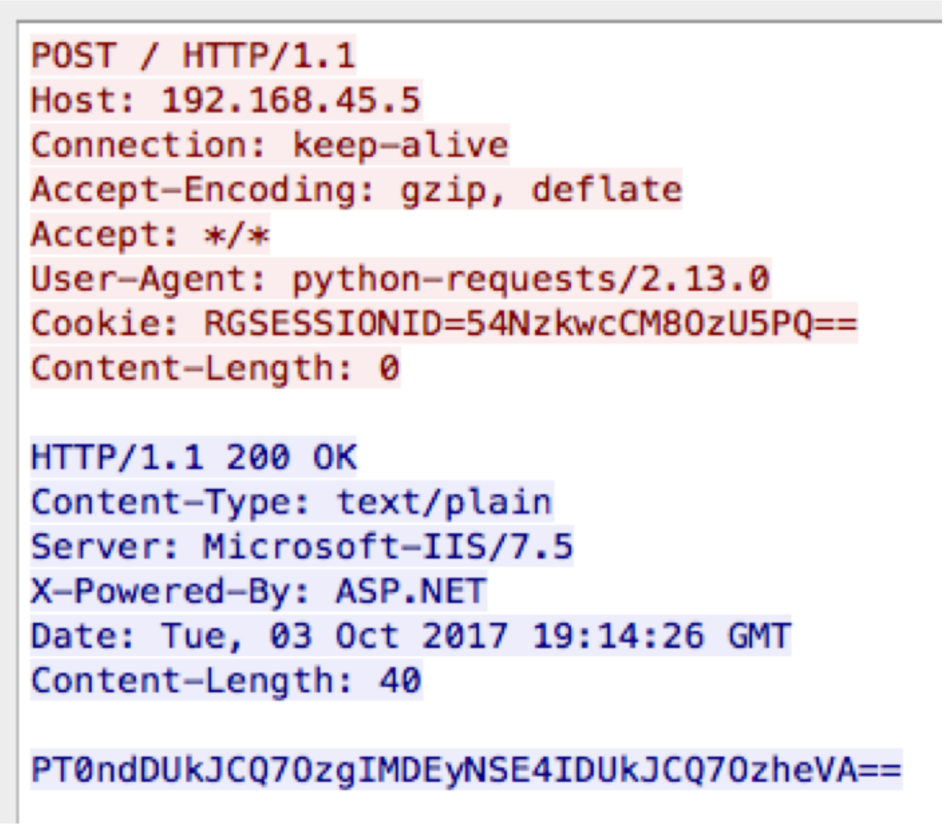
Figure 3 RGDoor code to set the HTTP Content-Type to “text/plain”

The sample then transmits the data back to the actor by creating a loop that calls the IHttpResponse::WriteEntityChunk method until all of the data is sent to the actor within HTTP responses. If the WriteEntityChunk method fails at any point during this loop, the code will respond to the actor with a HTTP 500 “Server Error” response by using the IHttpResponse::SetStatus method.

Interacting with RGDoor

We created a client application to interact with the RGDoor module, specifically to issue commands to the backdoor and to see how it operates on the IIS server. As mentioned previously in this blog, RGDoor has three available commands, specifically “cmd$”, “upload$” and “download$”. We used our client to issue one of each of these commands.

Figure 4 shows the HTTP request and response to our first interaction with the RGDoor module. The command issued in this request was “cmd$whoami”, which instructs RGDoor to run the ‘whoami’ command in command prompt. The key used to encrypt this command was “54” (0x54), which resulted in Base64 encoded string of “NzkwcCM8OzU5PQ==”. The HTTP response shows the RGDoor module returned the Base64 encoded string of “PT0ndDUkJCQ7OzgIMDEyNSE4IDUkJCQ7OzheVA==”, which when decrypted using the same “54” key is the result of the ‘whoami’ command, specifically the string “iis apppool\\defaultapppool\n\x00”.



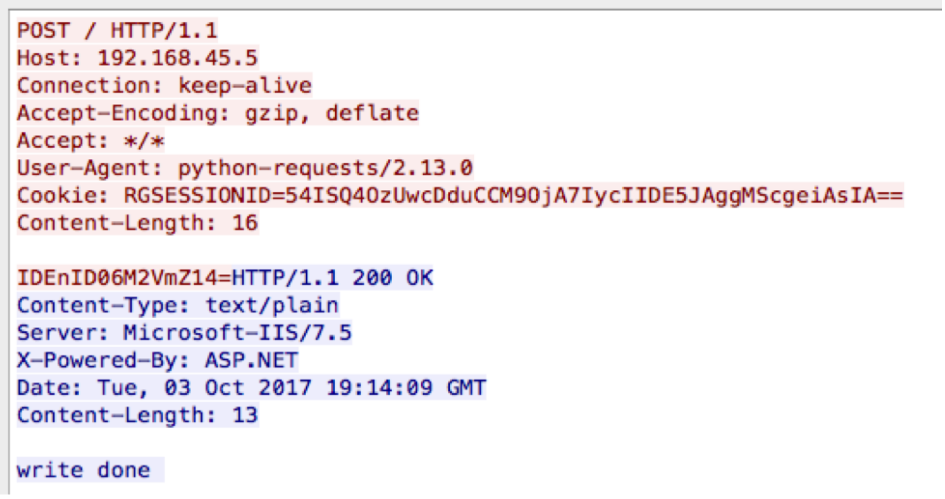
```
POST / HTTP/1.1
Host: 192.168.45.5
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.13.0
Cookie: RGSESSIONID=54NzkwcCM8OzU5PQ==
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/plain
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Tue, 03 Oct 2017 19:14:26 GMT
Content-Length: 40

PT0ndDUkJCQ7OzgIMDEyNSE4IDUkJCQ7OzheVA==
```

Figure 4 Issuing 'whoami' command and RGDoor's response

Figure 5 shows our testing of RGDoor's upload command, specifically "upload$c:\windows\temp\test.txt" that uploads a file from our local system to the webserver. The inbound request uses a key of "54" (0x54) that results in an encoded command string of "ISQ4OzUwcDduCCM9OjA7IycIIDE5JAggMScgeiAsIA=". The request also includes the string "IDEnID06M2VmZ14=" within the POST data, which is the string "testing123\n" from the file on our local system encrypted using the "54" key that will be written to the "test.txt" file on the server. As you can see from the HTTP response, RGDoor responded to this command with "write done", which is interesting as the response was in cleartext and not encrypted using the custom algorithm.



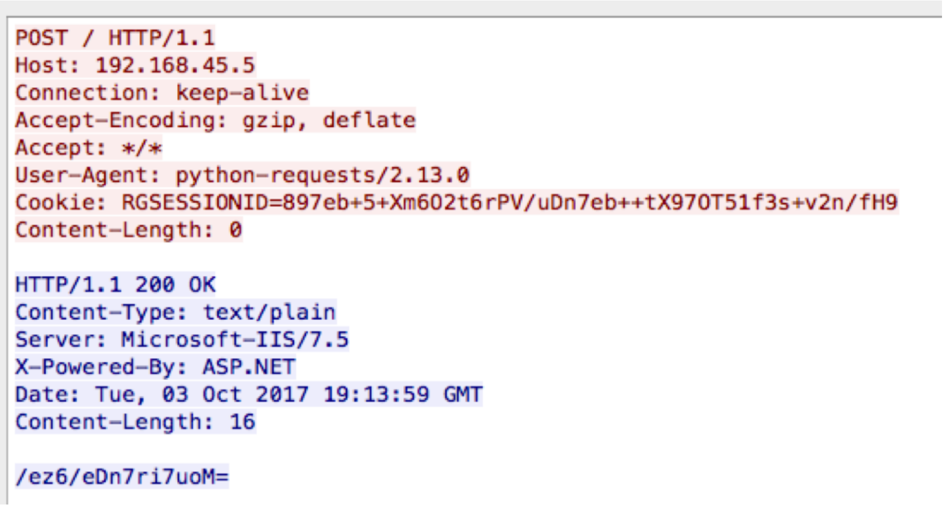
```
POST / HTTP/1.1
Host: 192.168.45.5
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.13.0
Cookie: RGSESSIONID=54ISQ4OzUwcDduCCM9OjA7IycIIDE5JAggMScgeiAsIA==
Content-Length: 16

IDEnID06M2VmZ14=HTTP/1.1 200 OK
Content-Type: text/plain
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Tue, 03 Oct 2017 19:14:09 GMT
Content-Length: 13

write done
```

Figure 5 Uploading a file to server via RGDoor Downloading a file from the server via RGDoor

Figure 6 shows our testing of the download command within RGDoor, specifically a command "download$c:\windows\temp\test.txt" that downloads the file uploaded in our previous test. We chose to use the key "89" (0x89) in this test to showcase RGDoor's ability to use any hexadecimal byte as a key, which resulted in an encoded command string of "7eb+5+Xm6O2t6rPV/uDn7eb++tX97OT51f3s+v2n/fH9". RGDoor responds to this command with the encoded string "/ez6/eDn7ri7uoM=", which when decrypted with the "89" key results in the string 'testing123\n', which is the contents of the "test.txt" file.



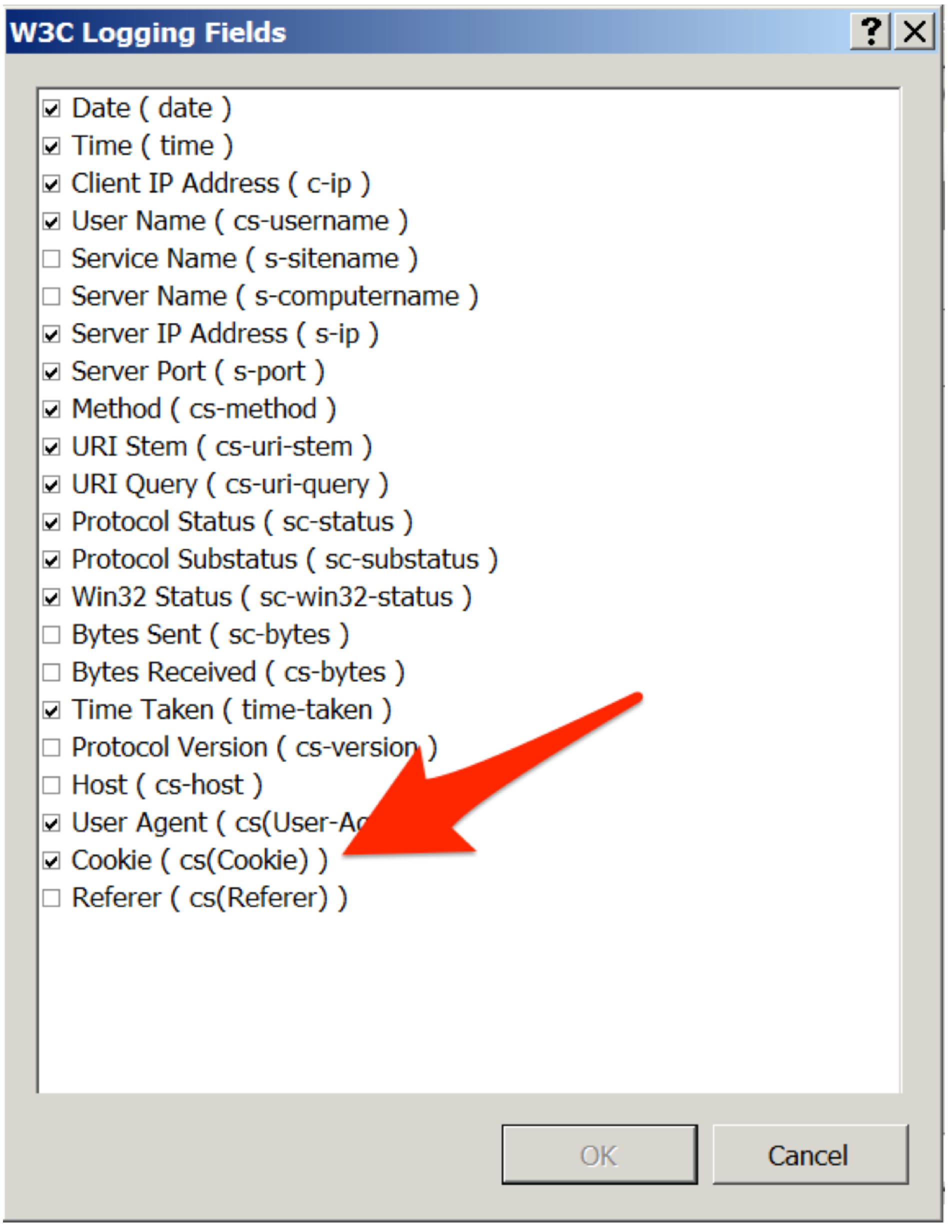
```
POST / HTTP/1.1
Host: 192.168.45.5
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.13.0
Cookie: RGSESSIONID=897eb+5+Xm6O2t6rPV/uDn7eb++tX97OT51f3s+v2n/fH9
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/plain
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Tue, 03 Oct 2017 19:13:59 GMT
Content-Length: 16

/ez6/eDn7ri7uoM=
```

Figure 6 Downloading a file from the server via RGDoor

To determine how the RGDoor client requests appears within the IIS request logs, we checked the logs of our default IIS installation in our test environment. By default, IIS does not log the values within Cookie fields of inbound HTTP requests, which would contain commands issued by actors to RGDoor. To see inbound RGDoor requests, an administrator must configure logging of Cookie fields in IIS, which can be selected in the W3C Logging Fields dialog in IIS Manager, as seen in Figure 7.



☑ Date (date)
☑ Time (time)
☑ Client IP Address (c-ip)
☑ User Name (cs-username)
☐ Service Name (s-sitename)
☐ Server Name (s-computername)
☑ Server IP Address (s-ip)
☑ Server Port (s-port)
☑ Method (cs-method)
☑ URI Stem (cs-uri-stem)
☑ URI Query (cs-uri-query)
☑ Protocol Status (sc-status)
☑ Protocol Substatus (sc-substatus)
☑ Win32 Status (sc-win32-status)
☐ Bytes Sent (sc-bytes)
☐ Bytes Received (cs-bytes)
☑ Time Taken (time-taken)
☐ Protocol Version (cs-version)
☐ Host (cs-host)
☑ User Agent (cs(User-Ag
☑ Cookie (cs(Cookie))
☐ Referer (cs(Referer))

Figure 7 W3C Logging Fields dialog in IIS Manager with Cookie fields enabled

The HTTP requests sent by the client to the RGDoor backdoor in the testing above will generate logs on the IIS server. Without the Cookie field logged, it is difficult to locate and analyze inbound requests related to RGDoor. Remember that RGDoor is an IIS module that checks all inbound POST requests for commands, so an actor does not need to use one particular URL to interact with RGDoor. However, the following shows the IIS log generated from the first testing request we made using our RGDoor client, which shows the “RGSESSIONID=” string, the “54” key and the Base64 encoded command within the Cookie field:

```
#Software: Microsoft Internet Information Services 7.5
#Version: 1.0
#Date: 2017-10-03 19:30:13
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) cs(Cookie) sc-status sc-substatus sc-win32-status time-taken
2017-10-03 19:30:13 192.168.45.5 POST / - 80 - 192.168.45.1 python-requests/2.13.0 RGSESSIONID=54NzkwcCM8OzU5PQ== 200 0 0 28
```

Conclusion

RGDoor is an IIS backdoor that actors used the TwoFace webshell to load onto an IIS web server. The RGDoor provides backdoor access to the compromised server, which we speculate the actors loaded in order to regain access to the server in the event the TwoFace webshell was removed. This backdoor has a rather limited set of commands, however, the three commands provide plenty of functionality for a competent backdoor, as they allow an actor to upload and download files to the sever, as well as run commands via command prompt. The use of RGDoor suggests that this group has contingency plans to regain access to a compromised network in the event their webshells are discovered and remediated.

Palo Alto Networks customers are protected from RGDoor by the following:

- All RGDoor samples have malicious verdicts in WildFire
- AutoFocus customers can investigate this activity with the RGDoor (https://autofocus.paloaltonetworks.com/#/tag/Unit42.RGDoor) tag
- IPS Signature RGDoor.Gen Command and Control Traffic (ID 11885) detects RGDoor network traffic

Indicators of Compromise

RGDoor SHA256

497e6965120a7ca6644da9b8291c65901e78d302139d221fcf0a3ec6c5cf9de3

a9c92b29ee05c1522715c7a2f9c543740b60e36373cb47b5620b1f3d8ad96bfa

RGDoor Filenames

HTTPParser.dll

TrafficHandler.dll

iishandler6.dll

Got something to say?

Leave a comment...

☐ **Notify me of followup comments via e-mail**

Name (required)

Email (required)

Website

SUBMIT

SUBSCRIBE TO NEWSLETTERS

Email SUBSCRIBE

COMPANY

Company (https://www.paloaltonetworks.com/company)

Careers (https://www.paloaltonetworks.com/company/careers)

Sitemap (https://www.paloaltonetworks.com/sitemap)

Report a Vulnerability (https://www.paloaltonetworks.com/security-disclosure)

LEGAL NOTICES

Privacy Policy (https://www.paloaltonetworks.com/legal-notices/privacy)

Terms of Use (https://www.paloaltonetworks.com/legal-notices/terms-of-use)

ACCOUNT

Manage Subscription (https://www.paloaltonetworks.com/company/subscriptions)

(https://www.linkedin.com/company/palo-alto-networks) (https://www.facebook.com/PaloAltoNetworks/)
(https://twitter.com/PaloAltoNtwks)

HANDS-ON DISRUPTION
ignite18
A Security Conference for the Future
Anaheim, CA | May 21–24
Register now

(https://ignite.paloaltonetworks.com/usa/?CampaignId=7010g000001IH8U&utm_content=Ignite18USA&utm_medium=390x90banner&utm_source=website)

© 2016 Palo Alto Networks, Inc. All rights reserved.

SALES > 866.320.4788 »

SEE A DEMO »

TAKE A TEST DRIVE (HTTP://CONNECT.PALOALTONETWORKS.COM/VIRTUAL-UTD)