Mandiant

UNC1860 Technical Annex

This is the technical annex for our blog post: UNC1860 and the Temple of Oats: Iran's Hidden Hand in Middle Eastern Networks.

A Google Threat Intelligence Collection featuring Indicators of Compromise related to the activity described in the blog post is available for registered users.

UNC1860 Malware

Foothold Utilities and Backdoors

Mandiant is tracking multiple foothold utilities and backdoors used in UNC1860 initial access operations (Table 1). These generally use custom obfuscation methods, tracked as OBFUSLAY and CRYPTOSLAY, which can lower detection rates and make analysis more difficult by renaming strings and function names.

Malware	Role	Description
SPARKLOAD	Backdoor	A .NET-based loader that decodes Base64-encoded and XOR-encrypted arguments and then writes the result to newly allocated memory space.
FACEFACE	Web shell	A web shell for a .NET application server capable of creating, deleting, and copying files and directories, command execution, and file downloads and uploads. We observed FACEFACE being deployed by SASHEYAWAY.
BASEWALK	Backdoor	A .NET DLL that handles POST requests. It supports writing to disk and shell execution capabilities and sending plaintext results via HTTP.
TOFUPIPE.DOTNET	Backdoor	A backdoor that establishes a new named pipe and awaits a payload to execute via the pipe.
ROTPIPE	Utility	A utility used to communicate over a named pipe, it can start or terminate a process, read files, and delete itself.
YASSERVER	Backdoor	A .NET-based server component of the YASSDOOR backdoor. It is capable of sending commands to infected hosts over named pipes and TCP sockets.

STAYSHANTE	Web shell	A password-protected web shell that supports the execution of .NET modules.
SASHEYAWAY	Dropper	An ASMX dropper written in C#. It contains a Base64-encoded payload that will be decoded during execution. The payload is a DotNet DLL. SASHEYAWAY iterates through the DLL's method for one named "ProcessRequest." If this specific method is found, SASHEYAWAY invokes it.
CHINACHOP	Web shell	A public web shell that may be written in Java, C#, or Jscript. It functions as a backdoor and may be deployed as part of an ASP.NET web application. Supported commands include file execution, file transfer, database command execution, database query, file management, and file enumeration.
HAZIZDOOR	Web shell	A password-protected web shell capable of executing shell commands, writing and reading files from the server, and editing the metadata of files.
TANKSHELL	Web shell	An ASPX web shell written in C# that functions as a tunneler. It expects to be provided with configuration details that are used to establish a connection with a remote host. These details include the IP address, port, encryption module, and a private key. Once a connection is established, TANKSHELL can be used to proxy data to or from the remote host.
INKWELL	Web shell	An ASPX web shell written in C# that functions as a backdoor. Supported backdoor commands include file execution, file transfer, and file timestamp manipulation. INKWELL requires a password to operate.
SEASHARPEE	Web shell	An ASPX web shell written in C# that functions as a backdoor. The backdoor supports command execution and multiple file-related commands that include transfer, execution, deletion, and timestamp modification. Some variants of SEASHARPEE can connect to and interact with a specified SQL server. SEASHARPEE requires a password to operate.
HEADTOE	Web shell	A password-protected web shell capable of launching arbitrary processes and uploadin arbitrary files.
TUNNELBOI	Tunneler	A network tunneller capable of establishing a connection with a remote host, managing web shells on the network, and creating RDP connections.
GETERQUEEN	Utility	A utility that receives a list of hosts and resolves them to their IP addresses.

Table 1: UNC1860 foothold utilities

Malware Use for Longer Term Persistence

We consider the code families in Table 2 to be UNC1860 "main-stage" implants that further increase the group's persistence in victim environments.

Malware	Role	Description
OATBOAT	In-Memory Dropper	A loader capable of decrypting an embedded shellcode and parameter and executing it.
TOFULOAD	Backdoor	A passive listener backdoor capable of receiving a payload and executing it.
TOFUPIPE	Backdoor	A backdoor that establishes a new named pipe and awaits a payload to execute via the pipe.
WINTAPIX	In-Memory Dropper	A persistent Windows kernel driver used for injecting an embedded shellcode.
TEMPLEDOOR	Backdoor	A .NET-based passive backdoor that relies on an HttpListener class to wait for incoming requests at predefined URL endpoints, and upon arrival of an HTTP request, it will attempt to parse it in order to find a supported command within it to execute.
TEMPLEDROP	Dropper	A .NET-based installer that is intended to stage a payload for persistent execution on a target host. The payload will be written as a file to a predefined directory on disk, registered as a service, and executed subsequently.
TOFUDRV	Backdoor	A persistent malicious driver that listens to incoming traffic on configurable URLs and can receive and execute an additional payload.

Table 2: UNC1860 passive implants and kernel drivers

Malware Controllers Analysis

TEMPLEPLAY

TEMPLEPLAY (MD5: c517519097bff386dc1784d98ad93f9d) is a .NET-based controller for the TEMPLEDOOR passive backdoor. It is internally named "Client Http" and consists of several tabs, each one facilitating control of a separate backdoor command.

The Command Prompt tab (Figure 1) sends a command line to execute on the target host. The default command is cmd /c 2 > &1 with parameter whoami.

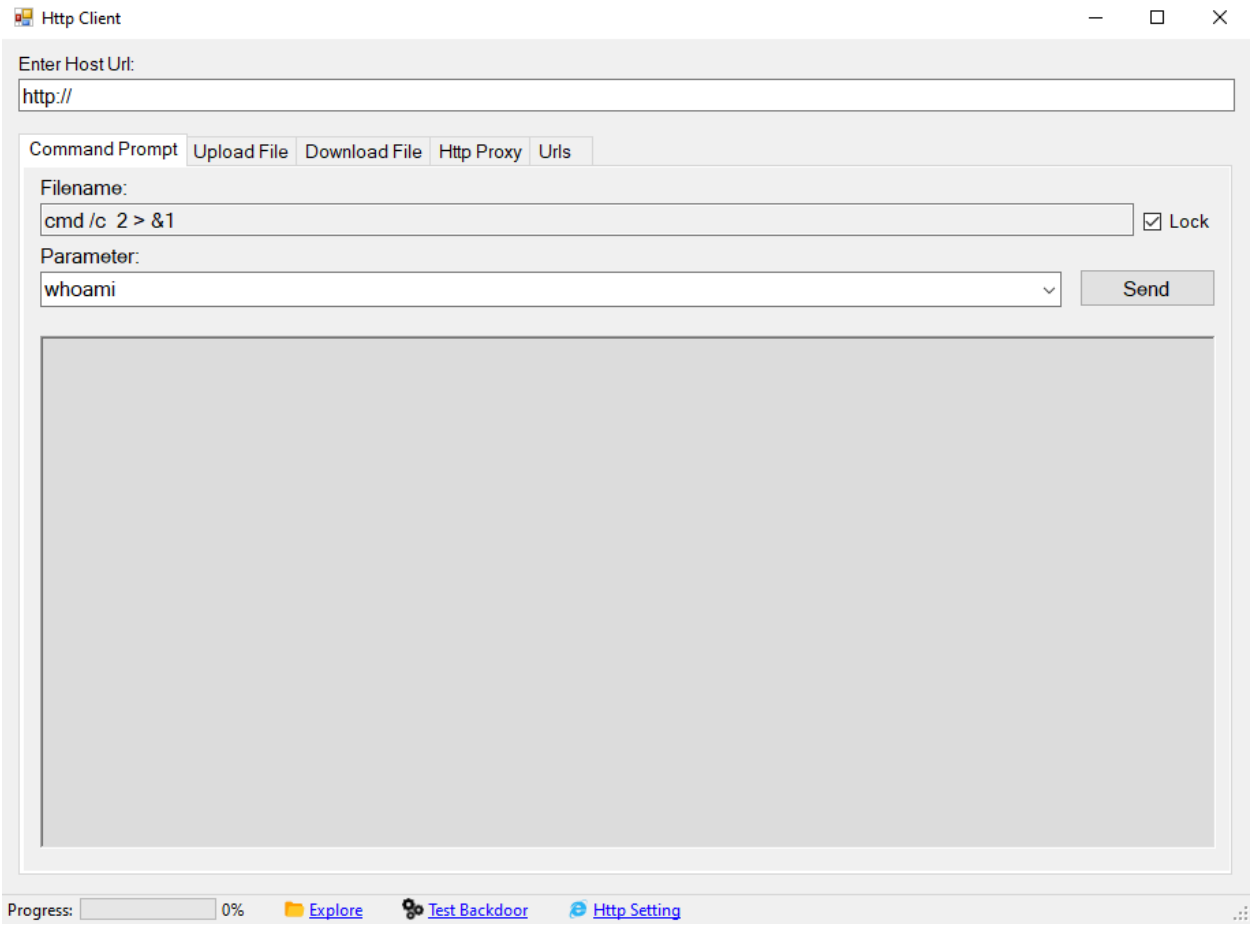


Figure 1: Command Prompt tab

The Upload File tab (Figure 2) sends a file from a local path to a target path on the remote machine using a POST request. The default target path is C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15\TEMPLATE\LAYOUTS.

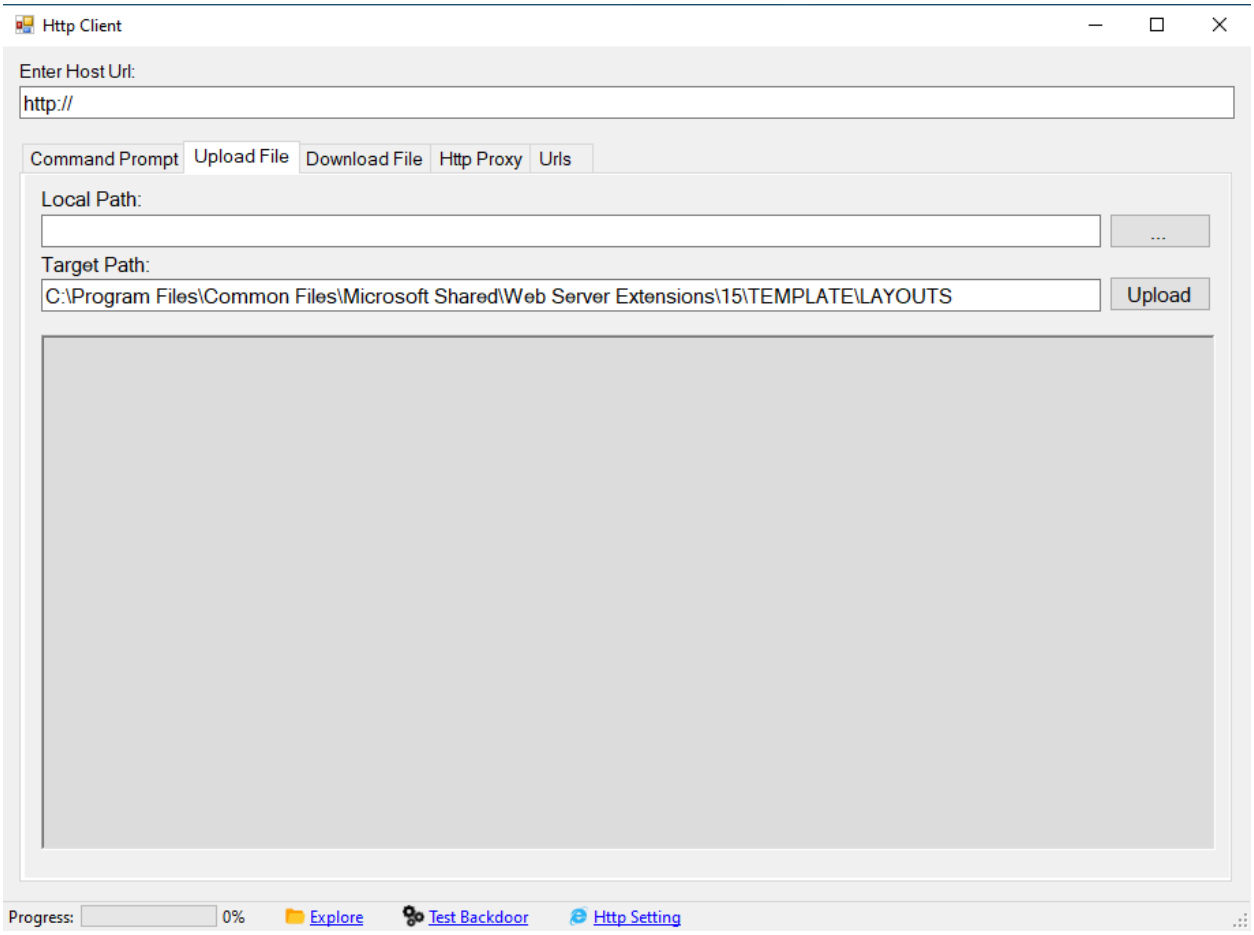


Figure 2: Upload File tab

The Download File tab (Figure 3) obtains a file from a given path on the infected machine. The default path on the infected machine is C:\Programdata\1.txt.

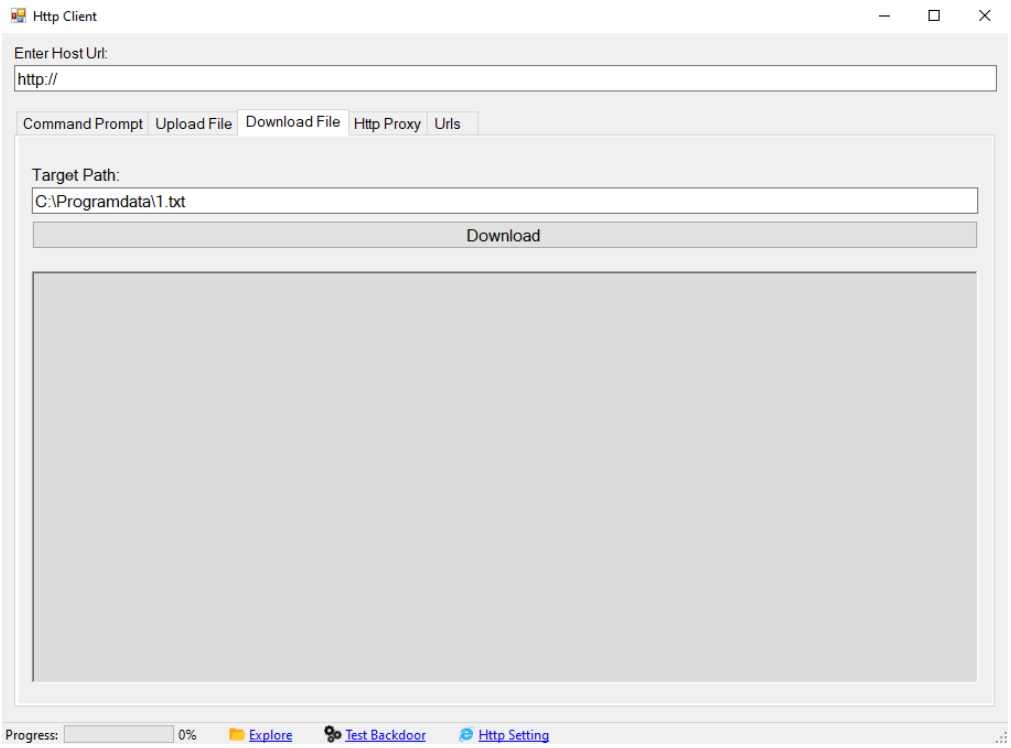


Figure 3: Download File tab

The Http Proxy tab (Figure 4) allows usage of a remote machine infected with TEMPLEDOOR as a middlebox that forwards data to a chosen target server. It appears that it is primarily intended to facilitate an RDP connection with the target server, most likely in cases where the latter is not accessible directly over the internet due to network boundaries (such as a NAT or a firewall), but may be accessible via the TEMPLEDOOR-infected machine. It relies on the following configuration parameters:

- **Proxy URL:** Address of the remote machine infected with TEMPLEDOOR that operates as the proxy server.
- **Encryption:** A .NET module that implements the encryption algorithm used to encrypt and decrypt the data sent between the controller and the proxy server. The default module used for this purpose is embedded within the controller and named "XORO" (MD5: 57cd8e220465aa8030755d4009d0117c). It exposes encrypt and decrypt methods that implement a plain XOR operation between the data and the key 0x0a18e2c5ddaa0f6574986414c64de5ce. The encrypted data is suffixed with the constant byte sequence 0xe0f1c6cf23930530cc270c8dd52e45e2. The TEMPLEDOOR instance (MD5: b219672bcd60ce9a81b900217b3b5864 contains another encryption DLL named "Base64" (MD5: ce537dd649a391e52c27a3f88a0a8912) that could be sent over to the controller during a handshake. As its name suggests, it merely implements the encoding and decoding of data using the Base64 algorithm.
- **Bind To:** IP and port of a local listener to which clients can connect.
    - Nagle: determines if an accepted socket (i.e., one that is formed as a result of a new client connecting to the listening socket) should use the Nagle algorithm to optimize the number of TCP packets sent over it.

    - Local Blocking: determines if an accepted socket should operate in blocking mode.
- **Remote To:** IP and port of a remote target machine to which data ought to be sent via the proxy server. The default IP and port are 10.0.0.8:3389, wherein 3389 is typically used for the RDP protocol.
    - **Nagle:** Determines if the socket formed between the proxy server and the target server ought to use the Nagle algorithm.
    - **Remote Blocking:** determines if the socket formed between the proxy server and the target server should operate in blocking mode.
- **Timeout(s):** Undetermined.
- **Interval(ms):** The time that the controller sleeps in between two data packages sent to the proxy server.
- **Packet Size:** The size of the TCP socket associated buffer on the proxy server that is used to receive data from the target server. The default size of it is 8192 bytes.

When the Start Tunnel button is pressed, the controller takes the following actions:

- Starts the mstsc.exe process, if not already running. This is the Windows utility used to form RDP connections.
- Creates a TCP socket and binds to the address and port specified in the Bind To configuration.
- Starts listening on the created socket.
- It is assessed that the operator manually configured the computer field in mstsc.exe (aka the RDP process) using the "Bind To" parameter (IP:port).
- When a client connects to the listening socket, a new thread is started to handle it. This thread performs the following actions:
    - Accepts the incoming socket.
    - Sets the socket's configuration according to the parameters specified in the Nagle and Local Blocking check boxes under the Bind To box.
    - Initiates a handshake with the proxy server by sending an HTTP GET request to the proxy's URL. The URL is built as a concatenation of the Proxy URL address and a randomly chosen endpoint string listed in the Proxy URLs list under the URLs tab.
    - The proxy server ought to respond with a 200 status code and may include an encoded .NET-based DLL in the body of the response. This DLL is responsible for the implementation of the encryption and decryption logic of the data that is sent between the controller and the TEMPLEDOOR proxy server. It ought to contain the namespace Encryption and expose the methods encrypt and decrypt.
    - In turn, the controller sends a byte array, which is a serialized version of an object named "RDPConfigPackage." This object contains the configuration of the connection formed between the TEMPLEDOOR proxy and the target server and includes the following fields:
        - **Type:** The type of the sent message to the server. In this case, the type is set to the value 67, which is the numeric value of the enum PackageType.RDPConfig.
        - **IP:** The IP address of the target server that the TEMPLEDOOR proxy ought to connect to.
        - **PORT:** The port on the target server that the TEMPLEDOOR proxy ought to connect to.

        - **Timeout:** Undetermined.
        - **Blocking:** The parameter set in the Remote Blocking field of the controller.
        - **PlaceStore:** Undetermined.
        - **EncryptionAssembly:** If no encryption DLL was received from the TEMPLEDOOR proxy server during the handshake, the controller will send its own encryption module.
        - **Nagle:** The parameter set in the Nagle field of the controller.
        - **PacketSize:** The parameter set in the Packet Size field of the controller.
    - Finally, the controller forwards the data received over the incoming socket to the proxy server as a sequence of HTTP POST requests.

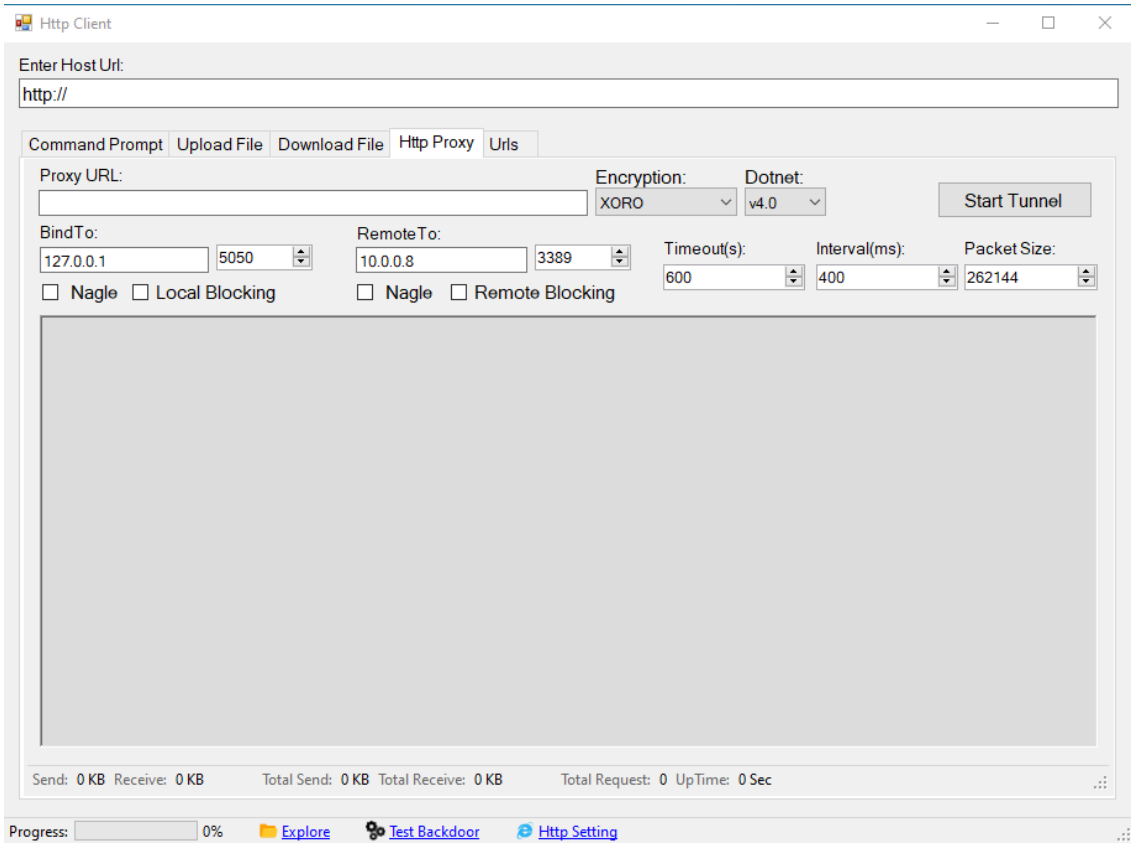


Figure 4: HTTP Proxy tab

The URLs tab (Figure 5) includes URL endpoints that will be used when connecting to the infected machine. An endpoint string would be chosen at random from the lists defined in this tab. These endpoints correspond to the ones that are defined in the TEMPLEDOOR sample (MD5: c57e59314aee7422e626520e495effe0).

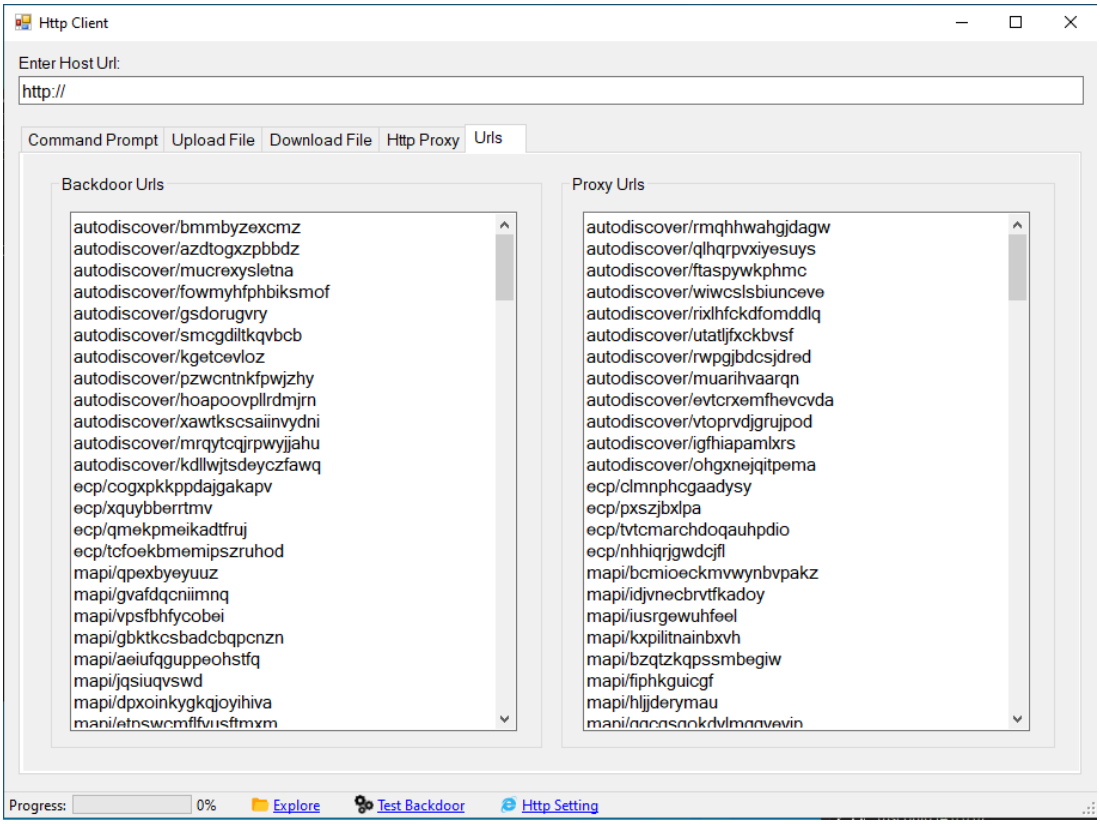


Figure 5: URLs tab

The Test Backdoor link creates a GET request with the string wOxhuoSBgpGcnLQZxipa as the relative URI and checks for the string UsEPTIkCRUwarKZfRnyjcG13DFA in the response. This corresponds to an echo \ ping mechanism that was seen being used in the TEMPLEDOOR samples (MD5: b219672bcd60ce9a81b900217b3b5864) and (MD5: c57e59314aee7422e626520e495effe0).

The Explore link opens a new Explorer window in the host in which the controller runs.

The Http Setting link points to a set of configuration parameters that pertain to the HTTP requests sent between the controller and the TEMPLEDOOR passive backdoor.

TEMPLEPLAY Dependencies

The TEMPLEPLAY instance depends on the following external .NET assemblies (i.e., those have to reside in the same directory with the controller and have the following names so that the latter operates properly):

- **HttpConnection.dll:** A library that facilitates the sending of HTTP requests to a server. It is not determined which sample of this DLL was used by the controller, but an instance of this library (MD5: 41ea0150b2265485cb41e8671eaf140e) was used by the VIROGREEN (MD5: 3dd397c4c1e2f9a0cf5190415f9cefe2) code family is entirely consistent with how this library ought to be implemented and used by TEMPLEPLAY.

- **Gadget Core.dll:** A library that implements the logic of the Http Proxy tab. It is not determined which sample of this DLL was used by the controller, but an instance of this library (MD5: b55371a333780b5fa3c07956ea2460ef) that was used by the TUNNELBOI (MD5: 5cf7cb0a19867365d22eb4cb5f643456) code family is entirely consistent with how this library ought to be implemented and used by TEMPLEPLAY.
- **System.Net.Primitive:** An instance of TEMPLEDOOR (MD5: c57e59314aee7422e626520e495effe0) that is embedded within a managed resource named "ManagedLoader" in TEMPLEPLAY.

TEMPLEPLAY Managed Resources

The following components were found as managed resources within the TEMPLEPLAY instance; however, they are not used in any way by the controller and are assessed to be test utilities for various component droppers and injectors:

- Installer (MD5: 487c7af570b4b1461502651c231f0225): a TEMPLEDROP installer. It is suspected to write a payload from a resource named "TEST" to the path %SYSTEM%\taskcomp.dll, but such a resource does not exist in the file.
- DLL_Injector (MD5: 96eb8126e3ed10ecdb00550675ddb801): A C++-based injection utility that is intended to drop a DLL to the path C:\Windows\System32\schedcore.dll from a resource named "TEST" and then inject it into a service named "Schedule." This will be followed by an attempt to load the DLL C:\Windows\System32\schedcomp.dll that is assumed to be present in the system. There is no resource named "TEST" in the utility, and it is undetermined what schedcomp.dll is.
- ManagedLoader (MD5: a64d9f9c0ccf458c4048ccbed2aeea14): A C++-based in-memory dropper for a .NET-based module. Like in the above cases, the dropped payload is suspected to reside in a resource named "TEST." Once loaded in memory, the dropper will invoke the method StartDec within the namespace System.Service.Program. The dropper's binary contains an instance of TEMPLEDOOR as an overlay (MD5: c57e59314aee7422e626520e495effe0); however, it is not being used by it in practice.

VIROGREEN

VIROGREEN (MD5: 9403c381e96f299f236733757433777e) is a custom framework UNC1860 uses to exploit vulnerable SharePoint servers with CVE-2019-0604. The framework is not just for exploitation, but it also provides post-exploitation capabilities including:

- Scanning for and exploiting CVE-2019-0604
- Controlling post-exploitation payloads, backdoors (including the STAYSHANTE web shell and BASEWALK backdoor), and tasking
- Executing commands and uploading/downloading files

Even though VIROGREEN was designed to exploit CVE-2019-0604, the exploitation and post-exploitation capabilities appear independent. VIROGREEN can control a compatible agent regardless of how the agent has been implanted, whether it was CVE-2019-0604 or not.

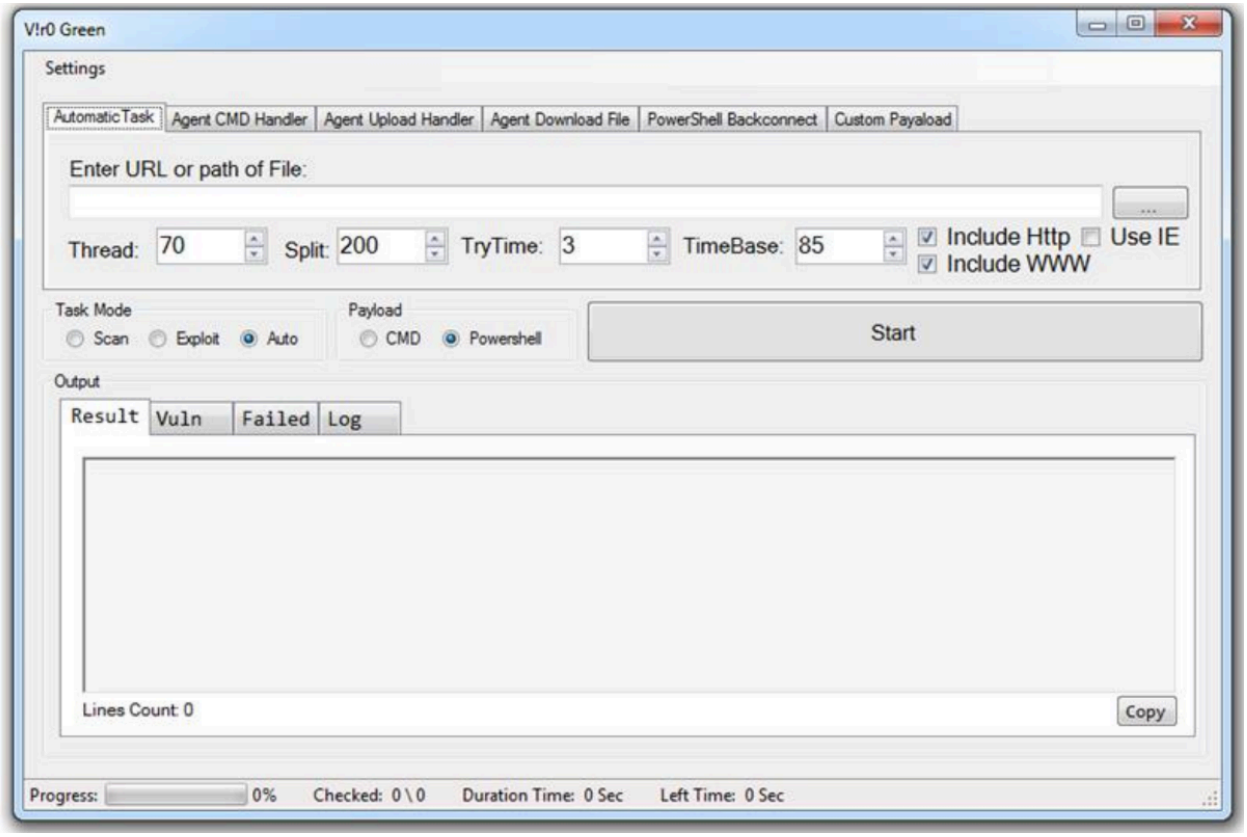


Figure 6: VIROGREEN GUI

Components

VIROGREEN consists of the following .NET assemblies:

Filename	Description
V!ro Green.exe	Main program
Viro Core.dll (VIROCORE)	Core capabilities: exploitation, payload and agent management, and tasking
Crypto.dll	Self-protection features: login form, password management, and self-deletion upon failed login attempts (max 4)
Restore Manager.dll	Saving/restoring process state to/from files
Thread Manager.dll	Multi-threading support
HTTPConnection.dll	Exploitation and post-exploitation HTTP configuration and request/response handling

Random Number.dll	Randomization of payloads, agents, User Agents, and encryption keys/seeds
Encryption.dll	Crypto support: AES keys generation, rudimentary custom crypto (XOR/ADD/REVERSE)
Gen3.5.dll and Gen3_5.dll	Generates and encodes the XML payloads used in CVE-2019-0604 exploitation
HTMLAgilityPack.dll	Publicly available HTML parser

Table 3: VIROGREEN components

Capabilities

VIROGREEN is capable of scanning and exploiting the SharePoint vulnerability CVE-2019-0604. The exploitation payloads can do one of the following actions:

- Injecting .NET agent module, which is capable of command execution and file upload/download
- Executing a PowerShell stager to connect back to a C2 IP and port (reverse shell)
- Executing stager commands, using either cmd.exe or powershell.exe, to drop web shell agents
- Executing any custom payload provided by the operator

.NET Module Agent

VIROGREEN allows the operator to inject a. NET module (the agent) in memory. The agent module provides command execution and file upload/download capabilities. The path to the DLL module and the expected parameters are passed in the exploitation payload. This capability has no persistence mechanism; it relies on CVE-2019-0604 to load the module.

During an engagement, the actor used this capability early in the intrusion lifecycle to establish a foothold. Later, the actor used web shell agents to maintain presence. In this engagement, the modules were loaded from disk from C:\ProgramData\w3handler.dll and C:\ProgramData\IISHelper.dll.

Reverse Shell

VIROGREEN allows the operator to run PowerShell code, which would cause the target to connect back to a specified C2.

Web Shell Agents

VIROGREEN supports a total of 17 (15 unique) web shell agents; the difference between them is the filename and password used to protect access to the web shells. The web shell expects the following elements in a HTTP POST request:

Element Name	Description of the Value
buffer	AES-encrypted, Base64-encoded .NET assembly that has a class named "C" and a method named "S"
key	Key and IV used to decrypt the buffer
n	First parameter to a tasking function
a	Second parameter to a tasking function

The web shell agent decodes and then decrypts the Base64-encoded data passed in buffer using the value of key as AES key and IV. The decrypted data should be a .NET module (tasking module) that has a class named "C," which contains a method named "S." The web shell reflectively loads the module and then invokes the method S with n and a as parameters.

The tasking modules support command execution and file upload/download. We have extracted 12 unique tasking modules from ViroGreen. For the three supported tasks, there are four different modules, the combination of the following:

- The .NET version (3.5 or 4.0)
- The tasking data (clear-text or encrypted)

Windows Kernel Drivers Analysis

TEMPLEDROP

TEMPLEDROP is a backdoor for Microsoft IIS servers consisting of a filter driver component, a protection driver component, and a .NET-based backdoor payload. Both driver components of the malware are persistent on the host by creation of service registry entries. The .NET backdoor payload contains a list of embedded URI paths that are used to construct URLs based on the sites hosted on the IIS server. These URLs are used to create listeners on the infected server, which will be used for receiving backdoor commands. The backdoor supports command execution, file download and upload, and DLL execution.

In order for TEMPLEDROP to work, it should be executed with Administrator permissions. It expects command-line arguments where the first argument can be one of the following:

- **--install**: Writes the payload and auxiliary components to the path %windir%\MSExchange.
- **--test:** Performs several access tests to verify if the installer and its deployed payload are capable of performing the operations required for their successful execution. Those include checks of whether the installer is run as an Administrator, whether it's capable of writing to the registry and disk, creating, starting and terminating a service, and starting an HTTP listener.

The second command-line argument named "--log" is optional. If specified, it will write a debug message to a file name log.txt in the current execution directory.

TEMPLEDROP most notably makes use of a Windows file system filter driver for the purpose of protecting some of the files it deploys as well as its own file from modification. The driver was originally used by an Iranian AV software named "Sheed AV" (MD5: 0c93cac9854831da5f761ee98bb40c37) seemingly to protect the AV's files. The driver gets registered as an upper filter for the CDROM ({4d36e965-e325-11ce-bfc1-08002be10318}), Disk drive ({4d36e967-e325-11ce-bfc1-08002be10318}), and Floppy disk ({4d36e980-e325-11ce-bfc1-08002be10318}) devices. Consequently, it is capable of intercepting operations against files and registry keys \ values that reside in preconfigured paths and restricts access to them if they are modified or deleted. As a precursor to the driver's deployment, TEMPLEDROP will run the following .reg file that sets all the keys and values pertaining to the driver, including a configuration string that specifies the paths to protect on disk and in the registry:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\msefd]
"Type"=dword:00000002
"Start"=dword:00000000
"ErrorControl"=dword:00000001
"Tag"=dword:00000002
"Group"="FSFilter Activity Monitor"
"DependOnService"=hex(7):46,00,6c,00,74,00,4d,00,67,00,72,00,00,00,00,00
"Enabled"=dword:00000001
"DebugFlags"=dword:00000000
"SupportedFeatures"=dword:00000003
"Config"="{\"ProtectedItems\":[{\"File\":{\"ProductName\":\"Antivirus\",\"De
stinationPath\":\"C:\\\\windows\\\\msexchange\\\\**\"}},{\"File\":{\"Product
Name\":\"SheedStore\",\"DestinationPath\":\"C:\\\\windows\\\\msexchange\\\\*
*\"}},{\"File\":{\"ProductName\":\"SheedStore\",\"DestinationPath\":\"C:\\\\
windows\\\\msexchange\"}},{\"File\":{\"ProductName\":\"Antivirus\",\"Destina
tionPath\":\"C:\\\\windows\\\\msexchange\"}},{\"Registry\":{\"ProductName\":
\"Antivirus\",\"Key\":\"HKEY_LOCAL_MACHINE\\\\SYSTEM\\\\CurrentControlSet\\\
\Services\\\\MSExchangeBackendManager**\",\"ValueName\":\"*\"}},{\"Registry\
":{\"ProductName\":\"SheedStore\",\"Key\":\"HKEY_LOCAL_MACHINE\\\\SYSTEM\\\\
CurrentControlSet\\\\Services\\\\MSExchangeBackendManager**\",\"ValueName\":
\"*\"}},{\"Registry\":{\"ProductName\":\"Antivirus\",\"Key\":\"HKEY_LOCAL_MA
CHINE\\\\SYSTEM\\\\CurrentControlSet\\\\Services\\\\msefd**\",\"ValueName\":
\"*\"}},{\"Registry\":{\"ProductName\":\"SheedStore\",\"Key\":\"HKEY_LOCAL_M
ACHINE\\\\SYSTEM\\\\CurrentControlSet\\\\Services\\\\msesp**\",\"ValueName\"
:\"*\"}},{\"Registry\":{\"ProductName\":\"SheedStore\",\"Key\":\"HKEY_LOCAL_
MACHINE\\\\SYSTEM\\\\CurrentControlSet\\\\Services\\\\msefd**\",\"ValueName\
":\"*\"}},{\"File\":{\"ProductName\":\"Antivirus\",\"DestinationPath\":\"C:\
\\\windows\\\\system32\\\\drivers\\\\msefd.sys\"}}]}"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\msefd\Instances]
```

```
"DefaultInstance"="msefd Instance"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\msefd\Instances\msefd
Instance]
"Altitude"="386421"
"Flags"=dword:00000000
```

Before starting the operation of the driver, TEMPLEDROP will deploy the following root certificate on the system using the command line certutil -addstore Root <%TEMP%>\<randname>.cer to ensure that the filter driver's signature will comply with the Windows DSE mechanism:

```
MIID1jCCAr6gAwIBAgIRAPTr50fMGHLl99R8FNHztXQwDQYJKoZIhvcNAQELBQAwgYAxKTAnBgNV
BAMMIFdvU2lnbiBUaW1lIFN0YW1waW5nIFNlcnZpY2VzIENBMRowGAYDVQQKDBFXb1NpZ24gQ0Eg
TGltaXRlZDEqMCgGA1UECwwhQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkgb2YgV29TaWduMQswCQYD
VQQGEwJDTjAeFw0xNDExMjMyMTIyNDdaFw0zOTExMjMyMTIyNDdaMIGAMSkwJwYDVQQDDCBXb1Np
Z24gVGltZSBTdGFtcGluZyBTZXJ2aWNlcyBDQTEaMBgGA1UECgwRV29TaWduIENBIExpbWl0ZWQx
KjAoBgNVBAsMIUNlcnRpZmljYXRpb24gQXV0aG9yaXR5IG9mIFdvU2lnbjELMAkGA1UEBhMCQ04w
ggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCIf5YQzQ/al8RdHPUenc3HlIh9gRl8Sr7q
W1p3Vtkst9hRiqBNkfAPPSJWsIvVrw48u9O2y1fzVfpv082FF3sKSa7LHQryfNUZdUcDCKqvbA4M
skwwWJQIRUhHHhKZ6crOwQUP5DMbwJoEabzgCD3BjXdNEabtpy/230qneVY46kBBJHPqxsvQg6oe
QAnJExrG3Pf1BARvamMIKNwCokuEVMIhs9+5NkqGv78c14dic855gtVhuhkAHavdm+5EIhfccE/h
uW+Nx6oF2cyEUTNVIv7mpZimABphKYEmMniI3Sp2mL+0M0LpQJWsnUXMKBkJ+E9QJc7mkQJxvC7Y
4hLzAgMBAAGjSTBHMA4GA1UdDwEB/wQEAwIHgDAWBgNVHSUBAf8EDDAKBggrBgEFBQcDCDAdBgNV
HQ4EFgQUcfFEQoCRBHw+iiqJw6gzuwWI6R0wDQYJKoZIhvcNAQELBQADggEBAAat4Lrjx0oW/a4R
55nnw8GzLCbXyn8qwSkVzRzwVQmYxcXElvnM3T7VMQoY9CnOrL40xGqSk0pPKjAdgfhe9D3hc+Kb
HqlREG9E53Inqaw0y5Q8myX8pvVEMFsPer1jBLwSpXokPWs6XFR1lYEfvBE25f9cbMNZm+Tvk/4D
jfluC9Q00J1VsA9X0MfddViIvcplF9Kp+OsNxfrxBWbaDdUaI7mnyShWbkBYu6fY+t/QWeqlQbm8
kjhULU6uubDaJD5I5jhwim1LjHAtrfLOUcAcy6ecrTd5NnAttfj6ggusrazVr5Mq9+4CEBF/hLQE
IWtrZ6HZ1UFfoUQU+/a8zxE=
```

Following are the certificate details:

```
Common Name: WoSign Time Stamping Services CA
Organization: WoSign CA Limited
Organization Unit: Certification Authority of WoSign
Country: CN
Valid From: November 23, 2014
Valid To: November 23, 2039
Issuer: WoSign Time Stamping Services CA, WoSign CA Limited
Key Size: 2048 bit
Serial Number: f4ebe747cc1872e5f7d47c14d1f3b574
```

In addition, TEMPLEDROP deploys a driver (MD5: 612462976f98c19cc3a29357bcc2fb8e) that is seemingly intended to protect the registry values COM1 and COM2 under the path HKLM\SYSTEM\ControlSet001\Services\msesp\Instances which ought to contain Base64-encoded

backup versions of the driver itself and the aforementioned file system filter driver. Other actions performed by the driver are undetermined, as it is heavily protected with VMProtect.

TEMPLEDROP invokes TEMPLEDOOR and TEMPLELOCK, discussed below.

TEMPLEDOOR

TEMPLEDOOR is a .NET-based passive backdoor. It relies on an HttpListener class to wait for incoming requests on predefined URL endpoints, and upon arrival of a request will attempt to parse it in order to find a command within it and execute it. The backdoor supports four commands that could be issued in the body of the request (e.g., upon receiving a POST request):

- **Command:** starts a new process, given an executable image path and arguments.
- **Upload:** Writes data received from the attacker to a file on disk, given the file and the data to write.
- **Download:** Sends a file back to the attacker, given the file's path.
- **Load:** Loads a file that was formerly written to disk as a .NET assembly using Reflection and invokes a method within it.

For each of the above command requests, the body should to be encoded properly for it to be processed and executed. The decoding of each request's body entails the following actions:

- Decoding from Base64
- XOR with the key 54 62 2d 0c 03 45 49 15 2b 43 59 4a 4e 0c 40
- Reversing the order of the bytes
- XOR decoding all of the bytes with the first byte in the reversed buffer

In addition, the HTTP listener may execute a command if it is issued as a query string in the request's URL (e.g., when issuing a GET request). In case it finds the strings ”Jet” or “Ver” as query parameters in the URL, it would either Base64 decode (for Jet requests) or hex decode (for Ver requests) the query's value and execute it as an argument of cmd.exe.

If the request fails to be processed at any point, namely if its not formatted correctly as specified above, the following bogus 404 page will be sent back to the request's initiator:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>404 - File or directory not found.</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica,
sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
```

```
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
#header{width:96%;margin:0 0 0 0;padding:6px 2% 6px
2%;font-family:"trebuchet MS", Verdana, sans-serif;color:#FFF;
background-color:#555555;}
#content{margin:0 0 0 2%;position:relative;}
.content-container{background:#FFF;width:96%;margin-top:8px;padding:10px;pos
ition:relative;}
-->
</style>
</head>
<body>
<div id="header"><h1>Server Error</h1></div>
<div id="content">
<div class="content-container"><fieldset>
<h2>404 - File or directory not found.</h2>
<h3>The resource you are looking for might have been removed, had its name
changed, or is temporarily unavailable.</h3>
</fieldset></div>
</div>
</body>
</html>
```

TEMPLELOCK

TEMPLELOCK is a .NET-based utility that is capable of terminating threads associated with the Windows Event Log service and restarting the service's operation on demand. It is embedded as a .NET class in TEMPLEDROP and is placed in a separate .NET assembly that is invoked to stop the Event Log service upon initiation of TEMPLEDOOR. A thread could be identified as one pertaining to the Event Log service by using the I_QueryTagInformation to get the associated service's name after reading the SubProcessTag field from the TEB structure of the thread in question and comparing it with the string eventlog. The Event Log service can get restarted by the utility by executing the cmd.exe command line sc start eventlog.

Main Components of WINTAPIX and TOFUDRV

- WINTAPIX, first reported by Fortinet, is a malicious Windows kernel driver for Microsoft IIS servers that contains an embedded DONUT packed shellcode that in turn invokes the .NET-based backdoor payload TEMPLEDOOR, which gains persistence on the host by creating service registry entries.
- TOFUDRV is a persistent malicious driver that listens to incoming traffic on configurable URLs and can receive and execute an additional payload.

WINTAPIX (MD5: `286bd9c2670215d3cb4790aac4552f22`)shares an underlying proprietary code base determined to be unique to UNC1860 with TOFUDRV (MD5: `b4b1e285b9f666ae7304a456da01545e`) :

- Kernel to user-mode process injection: Both are drivers capable of injecting code into a user-mode process, wherein all processes are enumerated so as to find one that meets a set of criteria:
    - It can't be one of several blacklisted processes. In the samples compared by Mandiant, the lists of those processes are hard-coded and are similar in both their names and order in which they appear:
        - TOFUDRV: winint.exe, csrss.exe, smss.exe, services.exe, winlogon.exe, vmtoolsd.exe, vmware, lsass.exe, LsaIso.exe, fontdrvhost.exe
            - Notably, LsoIso.exe and fontdrvhost.exe are used in more recent versions of windows that have Virtualization Based Security enabled. This indicates that different tooling may be used for different Windows versions, for example TOFUDRV would be used with a more recent version.
        - WINTAPIX : winint.exe, csrss.exe, smss.exe, services.exe, winlogon.exe, vmtoolsd.exe, vmware, lsass.exe.
    - It has to be a 32-bit process.
    - It has to be owned by the LocalSystem SID.
    - It can't be a protected process.
- **Registry key & value protection:** Both drivers attempt to protect registry keys and values that are used by them. The protection logic is unique and shared between both wherein all registry keys and values are enumerated recursively starting from given malware related registry keys and their information is saved in internal data structures. The drivers then use the ZwNotifyChangeKey API function to set-up a callback that would be invoked each time a change is made to the registry keys in question, and that callback is responsible for restoring all the previously saved data into the registry had any modification taken place. Consider the following code snippets from each of the compared samples that demonstrate this flow and the similarity in code that pertains to it:

```
hRegKey = 0i64;
keyInfo = 0i64;
len = 0;
memset(keyList, 0, sizeof(keyList));
memset(&valueList, 0, 0x10ui64);
if ( regPath )
{
  status = w_openKey(&hRegKey, regPath);
  if ( status >= 0 )
  {
    ZwQueryKey(hRegKey, KeyFullInformation, 0i64, 0, &len);
    keyInfo = allocMem(len);
    if ( keyInfo )
    {
      status = ZwQueryKey(hRegKey, KeyFullInformation, keyInfo, len, &len);
      if ( status >= 0 )
      {
        if ( keyInfo->SubKeys || keyInfo->Values )
        {
          status = collectProtectedRegInfo(regPath, keyList, &valueList);
          if ( status >= 0 )
          {
            memset(&regInfo, 0, sizeof(regInfo));
            Mutex = allocBuff(0x38ui64);
            KeInitializeMutex(Mutex, 0);
            regInfo.mutex = Mutex;
            regInfo.rootRegKey = allocAndCopyStr(regPath);
            regInfo.regKeyList = w_w_allocAndCopy(keyList, 0x10ui64);
            regInfo.regValueList = w_w_allocAndCopy(&valueList, 0x10ui64);
            c_regInfo = w_w_allocAndCopy(&regInfo, 0x30ui64);
            setupNotifyKeyChange(c_regInfo);
          }
        }
        else
        {
          status = 0xC0000034;
        }
      }
```

Figure 7: WINTAPIX code for registering a key change notification for protected registry keys

```
hRegKey = 0i64;
len = 0;
keyInfo = 0i64;
invoke_memset(&regKeyList, 0, 0x18ui64);
invoke_memset(&regValueList, 0, 0x10ui64);
if ( isAddrValid(rootRegKey) )
{
  if ( isAddrValid(regEntry) )
  {
    status = w_w_ZwOpenKey(&hRegKey, rootRegKey);
    if ( status >= 0 )
    {
      invoke_ZwQueryKey(hRegKey, KeyFullInformation, 0i64, 0, &len);
      keyInfo = allocMem(len);
      if ( keyInfo )
      {
        status = invoke_ZwQueryKey(hRegKey, KeyFullInformation, keyInfo, len, &len);
        if ( status >= 0 )
        {
          if ( keyInfo->SubKeys || keyInfo->Values )
          {
            _mm_lfence();
            status = cllectProtectedRegInfo(rootRegKey, &regKeyList, &regValueList);
            if ( status >= 0 )
            {
              _mm_lfence();
              invoke_memset(&c_regInfo, 0, sizeof(st_reg_info));
              mutex = allocAndClear(sizeof(st_reg_info));
              invoke_KeInitializeMutex_0(mutex, 0);
              c_regInfo.mutex = mutex;
              c_regInfo.rootRegKey = allocAndCopyStr(rootRegKey);
              c_regInfo.regKeyList = allocAndCopy_2(&regKeyList, 0x18ui64);
              c_regInfo.regValueList = allocAndCopy_2(&regValueList, 0x10ui64);
              c_regInfo.unkFlag = 1;
              regInfo = allocAndCopy_2(&c_regInfo, 0x38ui64);
              setupNotifyChangeKey(regInfo);
              regEntry->regInfo = regInfo;
            }
          }
          else
          {
            status = 0xC0000034;
          }
        }
```

Figure 8: TOFUDRV code for registering a key change notification for protected registry keys

As stated above, the information on protected registry keys and values is saved into proprietary data structures that are also identical in both code families:

```
struct st_reg_info
{
  PRKMUTEX mutex;
```

```
  HANDLE hRootRegKey;
  wchar_t *rootRegKey;
  st_reg_keys *regKeyList;
  st_reg_values *regValueList;
  st_reg_apc *apcRoutine;
};


struct st_reg_keys
{
  int numOfEntries;
  int regKeysArrLen;
  wchar_t regKeysArr[];
};

struct st_reg_values
{
  int numOfEntries;
  int regValuesArrayLen;
  st_reg_value_info regValuesArray[];
};

struct st_reg_value_info
{
  wchar_t *valuePath;
  int valueIndex;
  int valueType;
  wchar_t *valueName;
  int valueDataLen;
  BYTE *valueData;
};

struct st_reg_apc
{
  __int64 unk;
  __int64 unk2;
  PVOID apcRoutine;
  PVOID apcContext;
};
```

- **Driver file protection**: Similar to the protection of registry-based data, both malware families use a mechanism to protect the driver's image on disk. This is achieved by first retrieving and storing information about the driver's file and its contents in an internal data structure and then setting up a callback using `NtNotifyChangeDirectoryFile` that would restore this data upon any change in the file. The following code snippets demonstrate this logic and the code similarity that pertains to it between the malware families:

```
KeWaitForSingleObject(driverFileInfo->mutex, Executive, 0, 0, 0i64);
while ( !isActive() )
{
  if ( driverFileInfo->hDriverDir
    || createFile(&driverFileInfo->hDriverDir, driverFileInfo->driverDir, 0x1F01FFu, 0x4000, 3, 1, 1) < 0
  {
    if ( driverFileInfo->changeInfoBuffer )
    {
      if ( MmIsAddressValid(driverFileInfo->changeInfoBuffer) )
      {
        checkIfDriverFileChangedAndOverwrite(driverFileInfo);
        if ( driverFileInfo->changeInfoBuffer )
          ExFreePoolWithTag(driverFileInfo->changeInfoBuffer, 0);
      }
    }
    driverFileInfo->changeInfoBuffer = allocMem(0x10000ui64);
    v2 = invoke_NtNotifyChangeDirectoryFile(
           driverFileInfo->hDriverDir,
           0i64,
           setupFileChangeNotify,
           driverFileInfo,
           0i64,
           driverFileInfo->changeInfoBuffer,
           0x10000,
           4095,
           1);
    if ( v2 == 259 || !v2 )
      return KeReleaseMutex(driverFileInfo->mutex, 0);
    driverFileInfo->hDriverDir = 0i64;
  }
}
return KeReleaseMutex(driverFileInfo->mutex, 0);
}
```

Figure 9: Driver file protection logic in WINTAPIX (MD5: 286bd9c2670215d3cb4790aac4552f22)

```
result = isAddrValid_1(driverFileInfo);
if ( result )
{
  result = driverFileInfo->isActive;
  if ( driverFileInfo->isActive )
  {
    invoke_KeWaitForSingleObject(driverFileInfo->mutex, 0, 0i64, 0, 0i64);
    while ( driverFileInfo->isActive )
    {
      if ( driverFileInfo->hDriverDir || createFile(driverFileInfo) >= 0 )
      {
        if ( isAddrValid_1(driverFileInfo->changeInfoBuffer) && driverFileInfo->isActive )
        {
          checkIfDriverFileChangedAndOverwrite(driverFileInfo);
          invoke_ExFreePool_1(driverFileInfo->changeInfoBuffer);
        }
        if ( driverFileInfo->isActive )
        {
          driverFileInfo->changeInfoBuffer = allocMem_4(0x10000ui64);
          v2 = invoke_NtNotifyChangeDirectoryFile(
                 driverFileInfo->hDriverDir,
                 0i64,
                 setupFileChangeNotify,
                 driverFileInfo,
                 0i64,
                 driverFileInfo->changeInfoBuffer,
                 0x10000,
                 4095,
                 1);
          if ( v2 == 259 || !v2 )
            return invoke_KeReleaseMutex(driverFileInfo->mutex, 0);
          driverFileInfo->hDriverDir = 0i64;
        }
      }
    }
    return invoke_KeReleaseMutex(driverFileInfo->mutex, 0);
  }
}
return result;
```

Figure 10: Driver file protection logic in TOFUDRV (MD5: b4b1e285b9f666ae7304a456da01545e)

In this case there is also usage of a proprietary data structure for storing the driver's image information that is used by both families:

```
struct st_file_notify
{
  PRKMUTEX mutex;
  HANDLE hFile;
  HANDLE hDriverDir;
  wchar_t *driverName;
  wchar_t *driverDir;
  wchar_t *driverFullPath;
  BYTE *driverData;
```

```
  __int64 driverSize;
  BYTE *changeInfoBuffer;
};
```

- **Safe-mode check**: The business logic in both drivers is executed in separate system threads after checking that the driver is not run while the OS is in safe mode via the `InitSafeBootMode` variable:

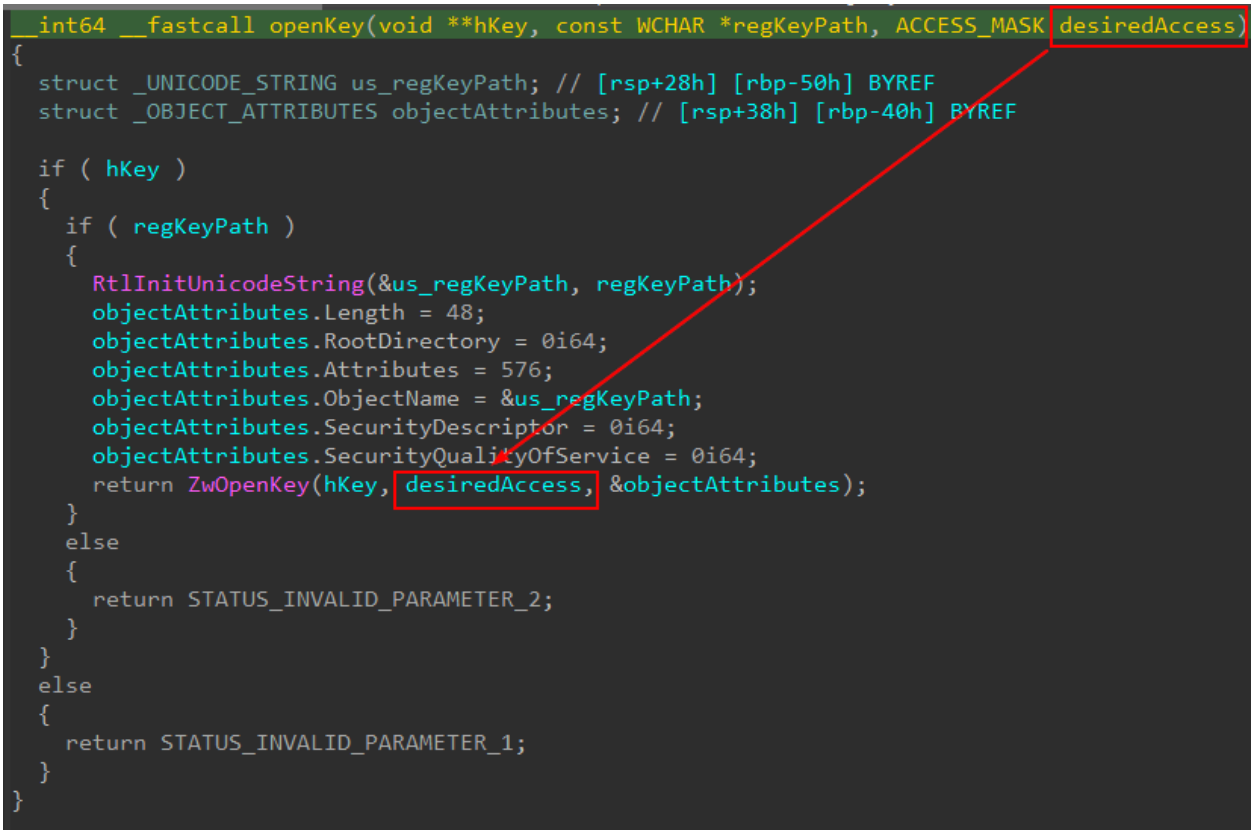
```
if ( !InitSafeBootMode )
  return startShellcodeApcInjectThread();
return status;
```

Figure 11: Safe mode check prior to main thread execution in WINTAPIX

```
if ( !InitSafeBootMode )
{
  _mm_lfence();
  hCommThread = 0i64;
  status = w_invoke_PsCreateSystemThread_0(communicationThread, stMain, &hCommThread);
  if ( status >= 0 )
    invoke_ZwClose_0(hCommThread);
}
```

Figure 12: Safe mode check prior to main thread execution in TOFUDRV

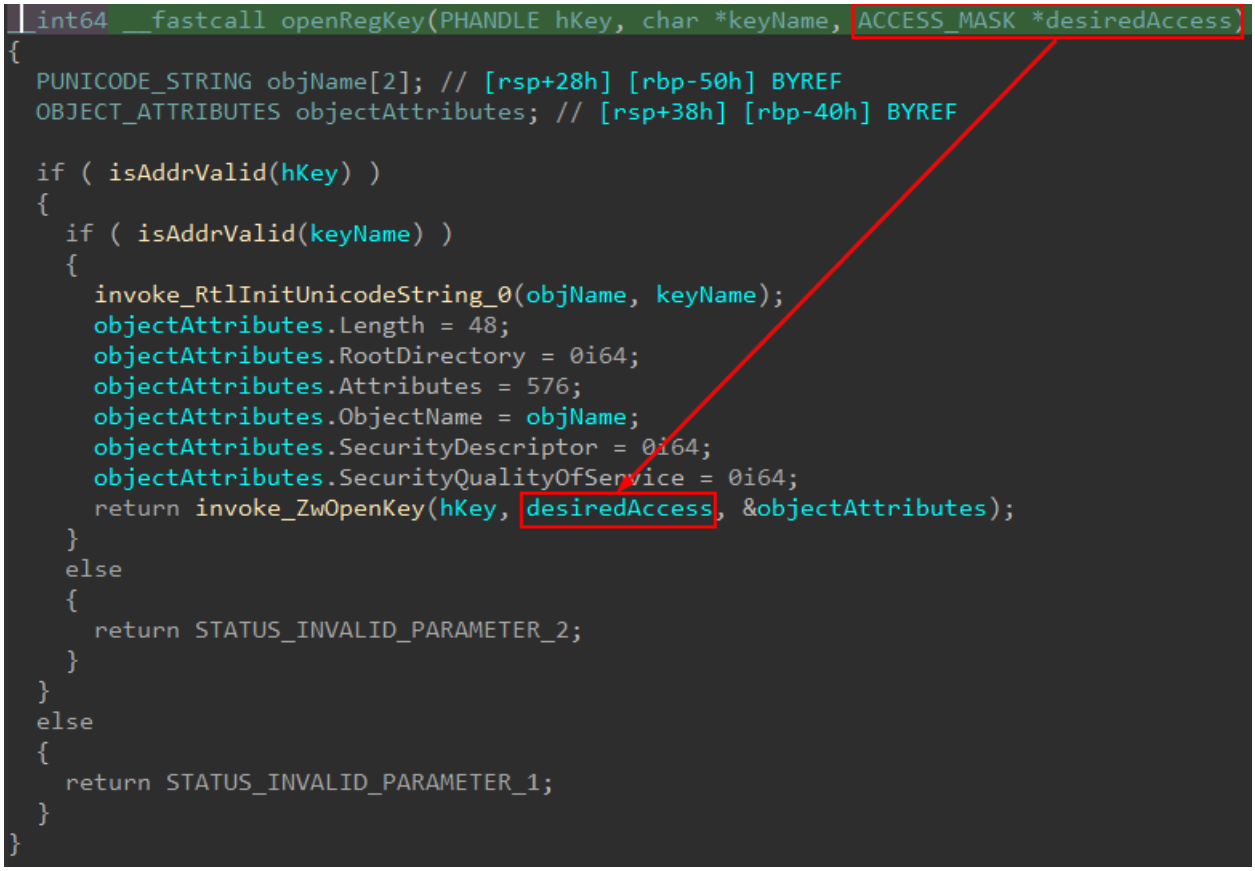
- **Unique wrapper function implementation**: both malware families use wrapper functions around Windows kernel APIs. Some of these disclose proprietary implementation choices that suggest they belong to a mutual code base. As an example, the wrapper around the `ZwOpenKey` function in the samples compared in this ticket places the `DesiredAccess` argument as the third one in the wrapper function, whereas its position in the API function itself is the second argument.



```
__int64 __fastcall openKey(void **hKey, const WCHAR *regKeyPath, ACCESS_MASK desiredAccess)
{
  struct _UNICODE_STRING us_regKeyPath; // [rsp+28h] [rbp-50h] BYREF
  struct _OBJECT_ATTRIBUTES objectAttributes; // [rsp+38h] [rbp-40h] BYREF

  if ( hKey )
  {
    if ( regKeyPath )
    {
      RtlInitUnicodeString(&us_regKeyPath, regKeyPath);
      objectAttributes.Length = 48;
      objectAttributes.RootDirectory = 0i64;
      objectAttributes.Attributes = 576;
      objectAttributes.ObjectName = &us_regKeyPath;
      objectAttributes.SecurityDescriptor = 0i64;
      objectAttributes.SecurityQualityOfService = 0i64;
      return ZwOpenKey(hKey, desiredAccess, &objectAttributes);
    }
    else
    {
      return STATUS_INVALID_PARAMETER_2;
    }
  }
  else
  {
    return STATUS_INVALID_PARAMETER_1;
  }
}
```

Figure 13: API wrapper function used in WINTAPIX



```
__int64 __fastcall openRegKey(PHANDLE hKey, char *keyName, ACCESS_MASK *desiredAccess)
{
  PUNICODE_STRING objName[2]; // [rsp+28h] [rbp-50h] BYREF
  OBJECT_ATTRIBUTES objectAttributes; // [rsp+38h] [rbp-40h] BYREF

  if ( isAddrValid(hKey) )
  {
    if ( isAddrValid(keyName) )
    {
      invoke_RtlInitUnicodeString_0(objName, keyName);
      objectAttributes.Length = 48;
      objectAttributes.RootDirectory = 0i64;
      objectAttributes.Attributes = 576;
      objectAttributes.ObjectName = objName;
      objectAttributes.SecurityDescriptor = 0i64;
      objectAttributes.SecurityQualityOfService = 0i64;
      return invoke_ZwOpenKey(hKey, desiredAccess, &objectAttributes);
    }
    else
    {
      return STATUS_INVALID_PARAMETER_2;
    }
  }
  else
  {
    return STATUS_INVALID_PARAMETER_1;
  }
}
```

Figure 14: API wrapper function used in TOFUDRV

TOFUDRV and TOFULOAD Code Comparison

TOFUDRV is implemented as a kernel-mode driver, while TOFULOAD is implemented as a user mode executable. Both are passive backdoors, listening for incoming HTTP requests on the infected hosts, destined to a designated URI path that is specified using UrlPrefix strings. In the case of TOFUDRV the UrlPrefix parameter is saved in the registry.

The low-level mechanism used to facilitate the passive listening is implemented through direct invocations of the very same IOCTLs to the HTTP.sys driver. These IOCTLs are not documented and there does not appear to be a public resource that underlies their usage, suggesting that a mutual proprietary code base was used when employing this technique as part of each code family.

Both handle each incoming request asynchronously in a separate thread. Both use the same encoding scheme for both ingress requests and egress responses, wherein the first byte is a randomly chosen XOR key used to encode the rest of the response, and the result is further encoded with Base64. The chosen XOR key for encoding egress responses is a randomly chosen byte within the range of 65 to 97.

Both anticipate shellcode in a similar structure within the request after it gets decoded:

```
struct st_received_shellcode {
    __int64 shellcode_size;
    BYTE shellcode[];
         __int64 shellcode_output;
         __int64 shellcode_output_len;
         __int64 magic_0x18;
         BYTE shellcode_arg[];
 };
```

In both cases, a successful shellcode execution will yield a `200` response, whereas any other faulty request will cause a `302` redirect to the path `/`.

In both cases the redirect response is sent with the unique string `Found` in its body, as illustrated below.

```
strcpy(s_Found, "Found");
s_slash = '/';
resp = (st_response *)invokeNtAllocateVirtualMemory(568, 4);
resp->val_0x10001 = 0x10001;
resp->responseCode = 302;
resp->responseBody = (__int64)s_Found;
for ( i = 0i64; s_Found[i]; ++i )
  ;
resp->bodyLen = i;
resp->flag = 1;
*(_QWORD *)&resp->s_slash = &s_slash;
result = deviceIoctl_0x12403F_UlSendHttpResponse(req->hHttp, req->stReqSub->UlSendHttpResponseIoctlBuffer, 0, resp);
if ( result >= 0 )
  return invokeNtFreeVirtualMemory(UNUSED_ARG(), resp);
return result;
```

Figure 15: Logic implementing a 302 redirect in TOFULOAD

```
resp = httpCreateResponseObject(302, "Found", 0i64, 0);
*&resp->s_slash = "/";
resp->flag = 1;
if ( httpIoctl_UlSendHttpResponseIoctl(req->hHttp, req->stReqSub->UlSendHttpResponseIoctlBuffer, 0, resp)
{
  _mm_lfence();
  ExFreePoolWithTag(resp, 0);
}
```

Figure 16: Logic implementing a 302 redirect in TOFUDRV

OATBOAT

OATBOAT is a small DLL used for loading and executing an embedded shellcode written in C++.

- The embedded shellcode is encrypted with XOR and should be executed with parameters.
- The parameters are also embedded in the DLL and are also encrypted with XOR.

OATBOAT was seen loading different shellcode payloads, predominantly TOFULOAD and TOFUPIPE.

TOFULOAD shellcode is executed with a set of parameters, and it opens two devices:

- Device\Http\Communication
- {{Device\Http\Requeue }}

Both devices are opened with the extended attribute "UlOpenPacket000."

TOFULOAD is leveraging these devices for setting up listeners for URLs over the server and later to craft and send back responses.

TOFULOAD can create requests to send back that are encrypted with a single-byte XOR that is randomly generated in a very specific range between 65–97 (correspond to printable characters) and are then encoded using Base64.

The list of listener URLs change between samples, following are some examples:

- http://+:80/Temporary_Listen_Addresses/
- http://+:80/lbsadmin/valve/
- http://+:80/lbsadmin/salon/
- http://+:80/lbsadmin/disorder/
- https://+:443/[REDACTED]/stable/
- https://+:443/[REDACTED]/dizzy/
- https://+:443/[REDACTED]/noodle/
- https://+:444/ews/exchanges/
- https://+:443/ews/exchanges/
- https://+:444/ews/exchange /

In 2023, an Israeli telecommunications entity was targeted with OATBOAT containing embedded shellcode payloads TOFULOAD and TOFUPIPE. As reported in open sources, multiple samples appeared to masquerade as an endpoint threat prevention solution and contained URI paths set as passive listeners, which included references to an Israeli telecommunications entity.

File MD5	Filename	Description
31f2369d2e38c78f5b3f2035dba07c08	CyveraConsole.exe	OATBOAT that contains an encrypted TOFUPIPE shellcode that can be executed with the embedded param: \\.\pipe\test-pipe
46804472541ed61cc904cd14be18fe1d	CyveraConsole.exe	OATBOAT that contains an encrypted shellcode of TOFULOAD. TOFULOAD is executed with URI paths to be set as passive listeners. The URI paths include references

		to an Israeli telecommunications entity and possibly their SOC
929b12bc9f9e5f8e854de1d46ebf40d9	CyveraConsole.exe	OATBOAT that contains an encrypted shellcode of TOFULOAD. For this sample, TOFULOAD is executed with URI paths to be set as passive listeners.

We also identified additional samples of OATBOAT, for example:

File MD5	Filename	Description
1176381da7dea356f3377a59a6f0e799	wlbsctrl.dll	OATBOAT that contains an encrypted TOFULOAD. The embedded XOR key is "BA" for this sample.
da0085a97c38ead734885e5cced1847f	wlbsctrl.dll	OATBOAT loading shellcode with a different set of parameters: \\.\pipe\test-pipe, leading to the TOFUPIPE shellcode
4abcf21b63781a53bbc1aa17bd8d2cbc	cct.exe	OATBOAT with TOFULOAD shellcode
57c916da83cc634af22bde0ad44d0db3	srvc.exe, systemre.exe	OATBOAT with TOFULOAD shellcode
85427a8a47c4162b48d8dfb37440665d	file.None.0xfffffa80237c4010.img	OATBOAT with TOFULOAD shellcode