# UNC1860 and the Temple of Oats: Iran’s Hidden Hand in Middle Eastern Networks

cloud.google.com/blog/topics/threat-intelligence/unc1860-iran-middle-eastern-networks

Mandiant

Written by: Stav Shulman, Matan Mimran, Sarah Bock, Mark Lechtik

## Executive Summary

UNC1860 is a persistent and opportunistic Iranian state-sponsored threat actor that is likely affiliated with Iran’s Ministry of Intelligence and Security (MOIS). A key feature of UNC1860 is its collection of specialized tooling and passive backdoors that Mandiant believes supports several objectives, including its role as a probable initial access provider and its ability to gain persistent access to high-priority networks, such as those in the government and telecommunications space throughout the Middle East.

UNC1860’s tradecraft and targeting parallels with Shrouded Snooper, Scarred Manticore, and Storm-0861, Iran-based threat actors publicly reported to have targeted the telecommunications and government sectors in the Middle East. These groups have also reportedly provided initial access for destructive and disruptive operations that targeted Israel in late October 2023 with BABYWIPER and Albania in 2022 using ROADSWEEP. Mandiant cannot independently corroborate that UNC1860 was involved in providing initial access for these operations. However, we identified specialized UNC1860 tooling including GUI-operated malware controllers, which are likely designed to facilitate hand-off operations, further supporting the initial access role played by UNC1860.

UNC1860 additionally maintains an arsenal of utilities and collection of “main-stage” passive backdoors designed to gain strong footholds into victim networks and establish persistent, long-term access. Among these main-stage backdoors includes a Windows kernel mode driver repurposed from a legitimate Iranian anti-virus software filter driver, reflecting the group’s reverse engineering capabilities of Windows kernel components and detection evasion capabilities. These capabilities demonstrate that UNC1860 is a formidable threat actor that likely supports various objectives ranging from espionage to network attack operations. As tensions continue to ebb and flow in the Middle East, we believe this actor’s adeptness in gaining initial access to target environments represents a valuable asset for the Iranian cyber ecosystem that can be exploited to answer evolving objectives as needs shift.

## Teamwork Makes the Dream Work: UNC1860’s Role as an Initial Access Provider

Mandiant identified two custom, GUI-operated malware controllers tracked as TEMPLEPLAY and VIROGREEN that we assess were used to provide a team outside of UNC1860 remote access to victim networks. This tooling, coupled with public reporting and evidence suggesting that the group collaborates with MOIS-affiliated groups such as APT34, strengthens the assessment that UNC1860 acts as an initial access agent.

## Using Sustained Access to Support Initial Access Operations

In 2020, Mandiant responded to an engagement in which UNC1860 used the victim's network as a staging area to conduct additional scanning and exploitation operations against unrelated entities. The actor was observed scanning IP addresses predominantly located in Saudi Arabia in an attempt to identify exposed vulnerabilities. UNC1860 also used a command-line tool to validate credentials of accounts and email addresses across multiple domains belonging to Qatari and Saudi Arabian entities, and later targeted VPN servers of entities in the region.

## UNC1860 Overlaps with APT34

Mandiant responded to several engagements in 2019 and 2020 in which organizations compromised by suspected APT34 actors were previously compromised by UNC1860. Similarly, organizations previously compromised by suspected APT34 actors were later compromised by UNC1860, suggesting the group may play a role in assisting with lateral movement. Mandiant additionally identified recent indications of operational pivoting to Iraq-based targets by both APT34-related clusters and UNC1860.

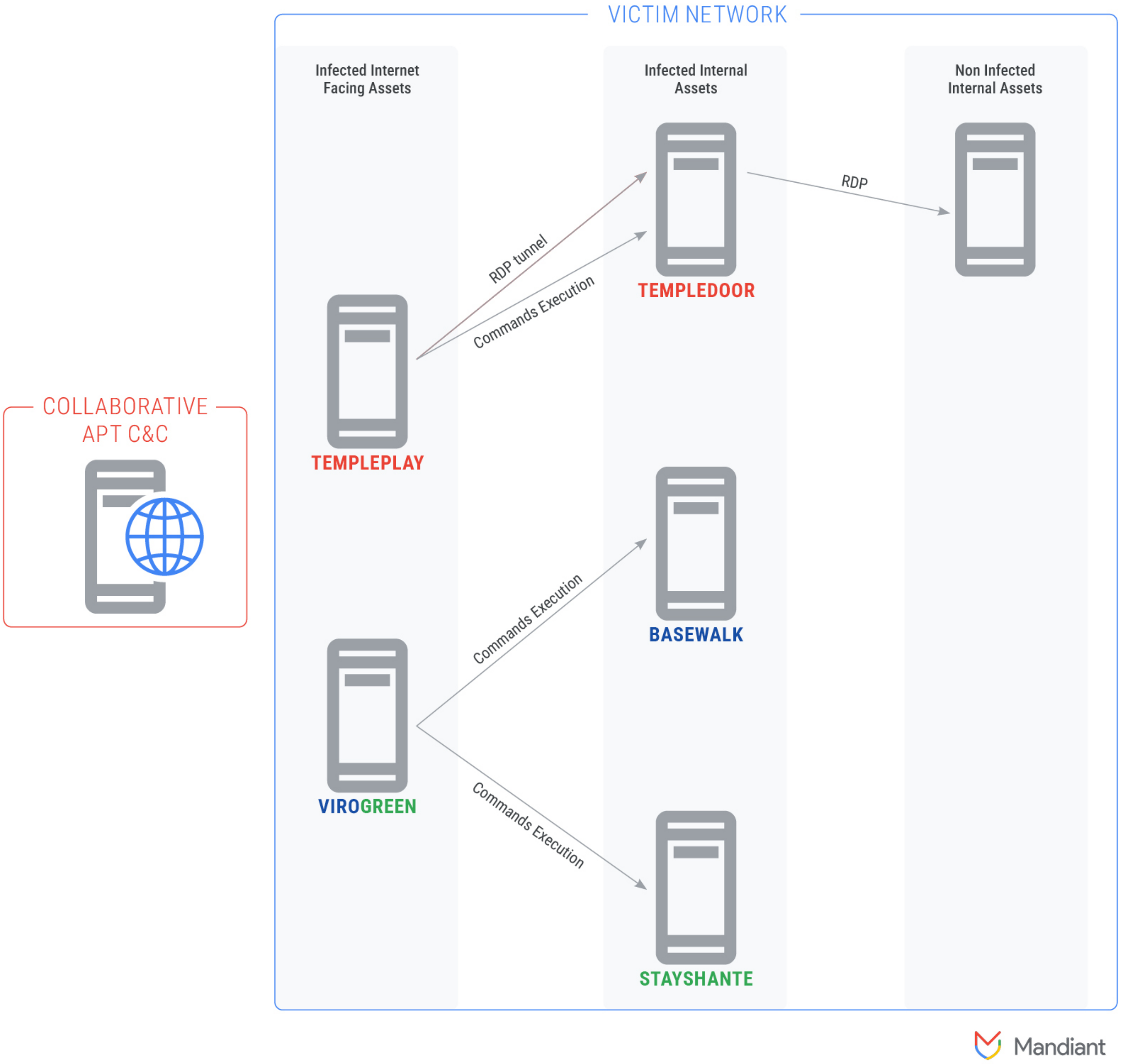
## Web Shell and Droppers

UNC1860 web shells and droppers, such as STAYSHANTE and SASHEYAWAY, deployed and placed on compromised servers by the group after gaining initial access have the potential to be used in hand-off operations based on their functionality. In March 2024, the Israeli National Cyber Directorate was alerted to wiper activity targeting Israeli entities across various sectors in Israel, including managed service providers, local governments, and academia; technical indicators included the unique STAYSHANTE web shell and the SASHEYAWAY dropper we attribute to UNC1860.

- STAYSHANTE is typically installed using names masquerading as Windows server file names or dependencies, and is controlled by the VIROGREEN custom framework described as follows.
- SASHEYAWAY has a low detection rate that allows for the smooth execution of full passive backdoors, such as TEMPLEDOOR, FACEFACE, and SPARKLOAD, embedded within it.

## Custom, GUI-Operated Malware Controllers

UNC1860 GUI-operated malware controllers TEMPLEPLAY and VIROGREEN could provide third-party actors who have no previous knowledge of the target environment the ability to remotely access infected networks via RDP and to control previously installed malware on victim networks with ease. These controllers additionally could provide third-party operators an interface that walks operators through how to deploy custom payloads and perform other operations such as conducting internal scanning and exploitation within the target network.


VICTIM NETWORK
Infected Internet Facing Assets
Infected Internal Assets
Non Infected Internal Assets
COLLABORATIVE APT C&C
TEMPLEPLAY
RDP tunnel
Commands Execution
TEMPLEDOOR
RDP
VIROGREEN
Commands Execution
BASEWALK
Commands Execution
STAYSHANTE
Mandiant

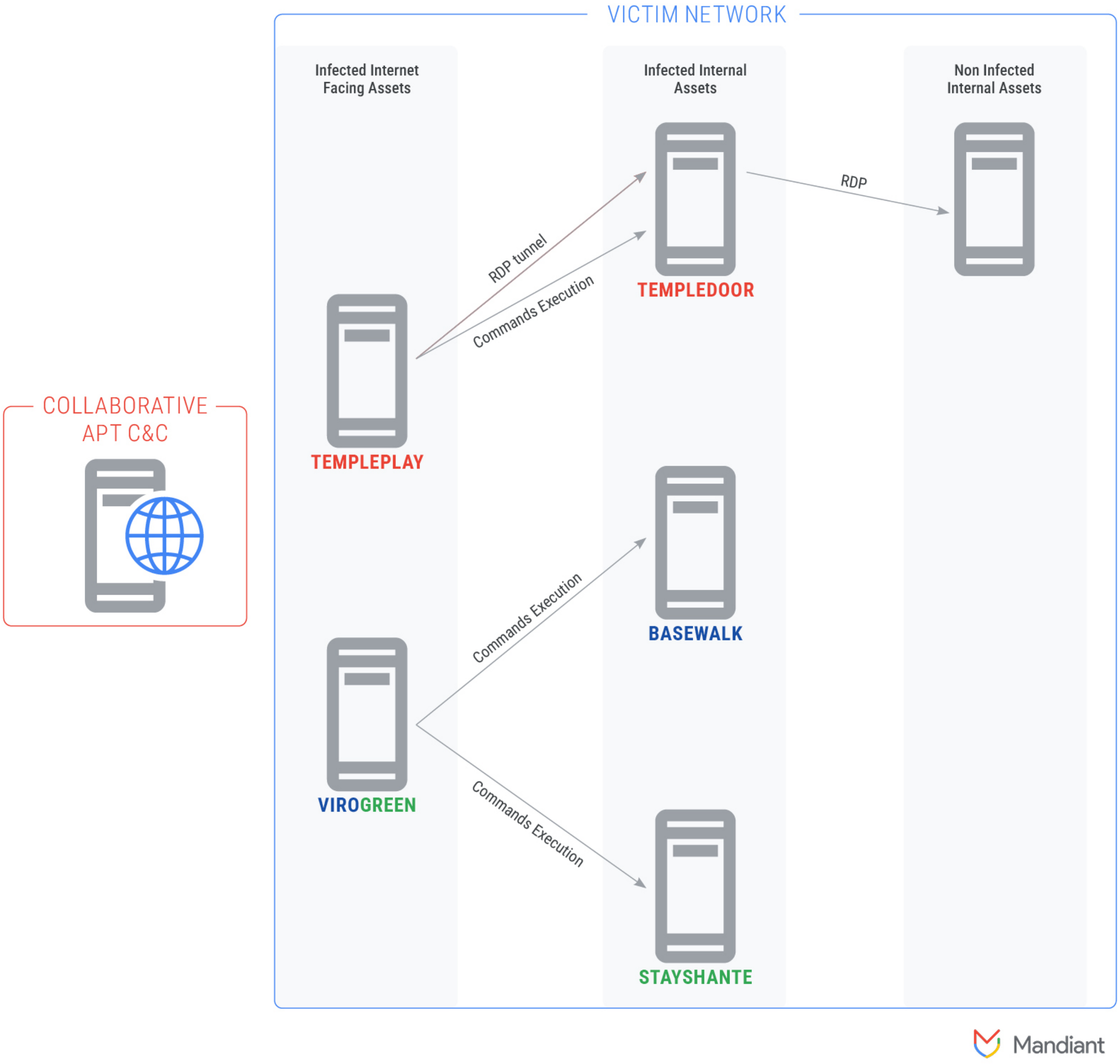
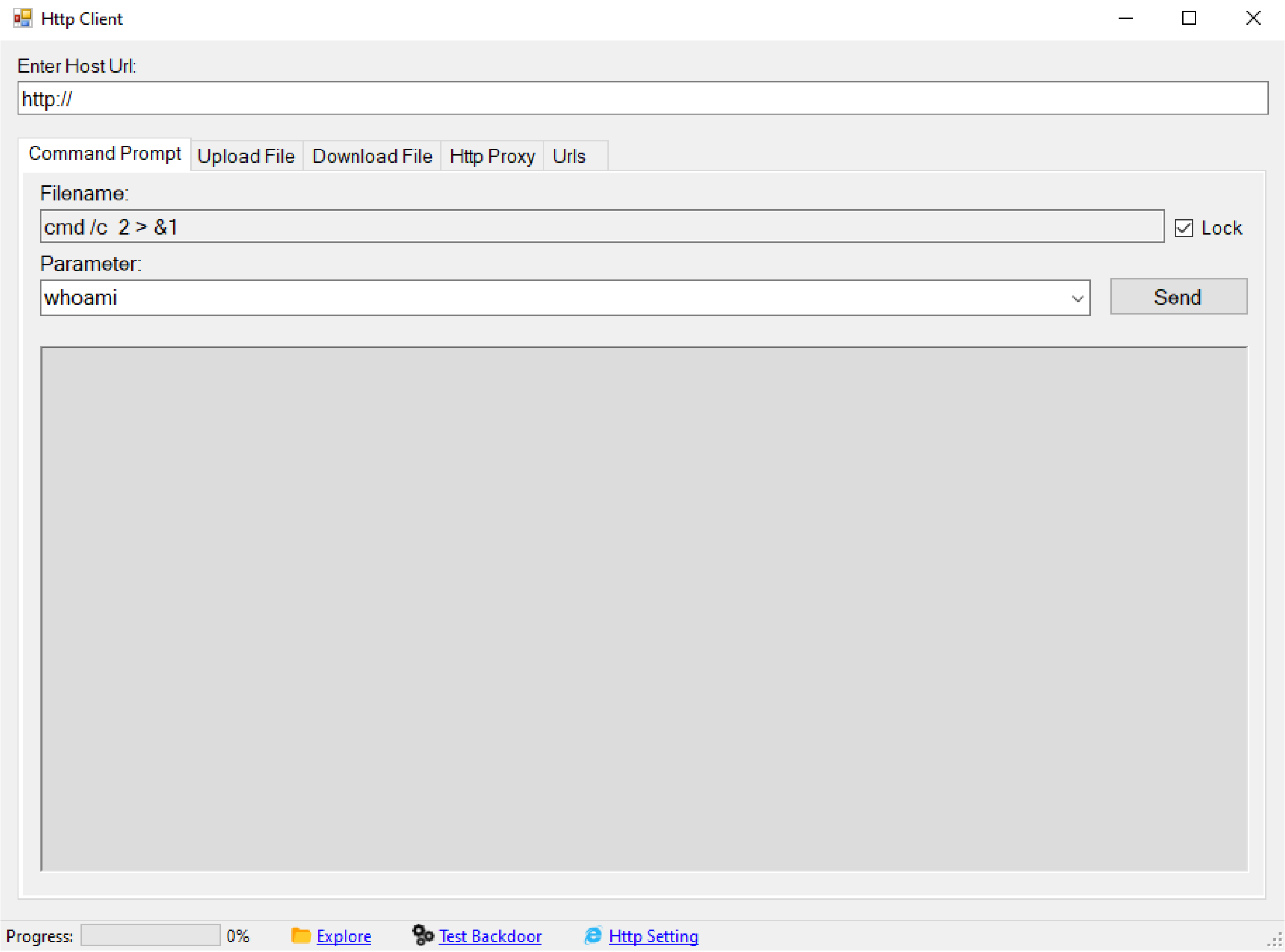


Figure 1: Illustration of collaborator actor's command and control (C2 or C&C) used to utilize existing UNC1860 implant infrastructure in compromised network

## TEMPLEPLAY Controller

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

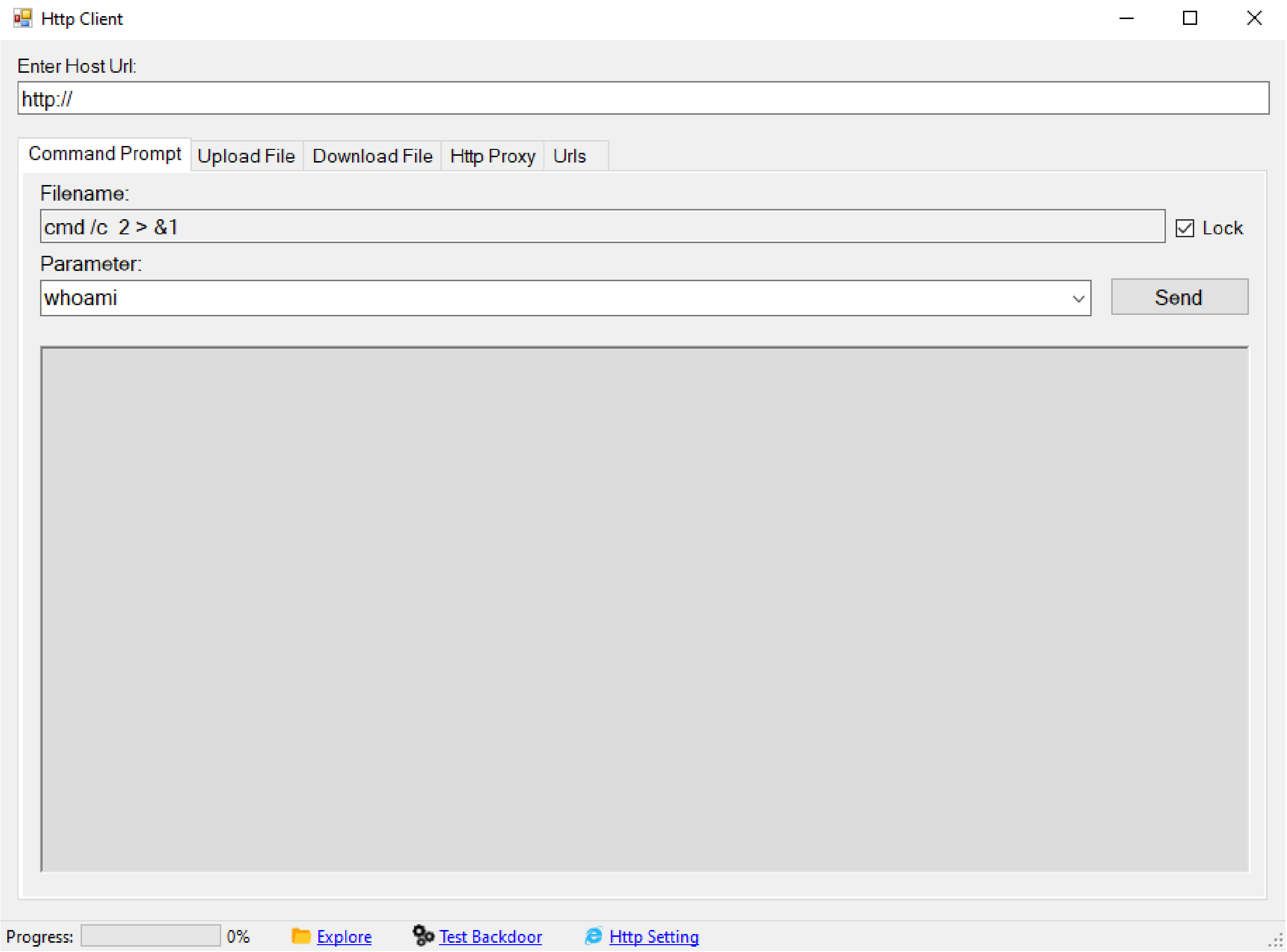


Enter Host Url: http://

Filename: cmd /c 2 > &1

☑ Lock

Parameter: whoami



Enter Host Url: http://

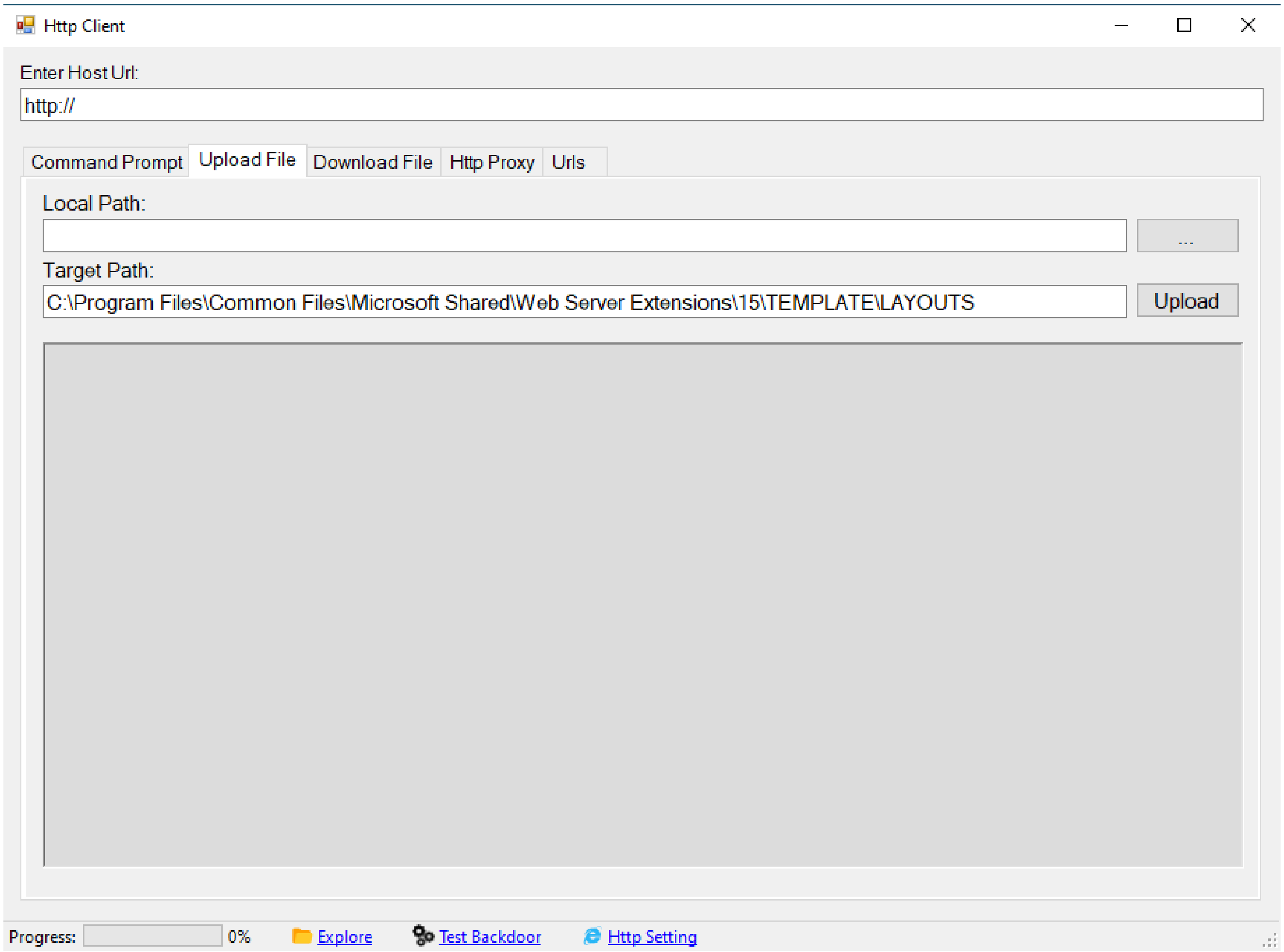
Filename: cmd /c 2 > &1

☑ Lock

Parameter: whoami

Figure 2: TEMPLEPLAY GUI, Command Prompt Tab

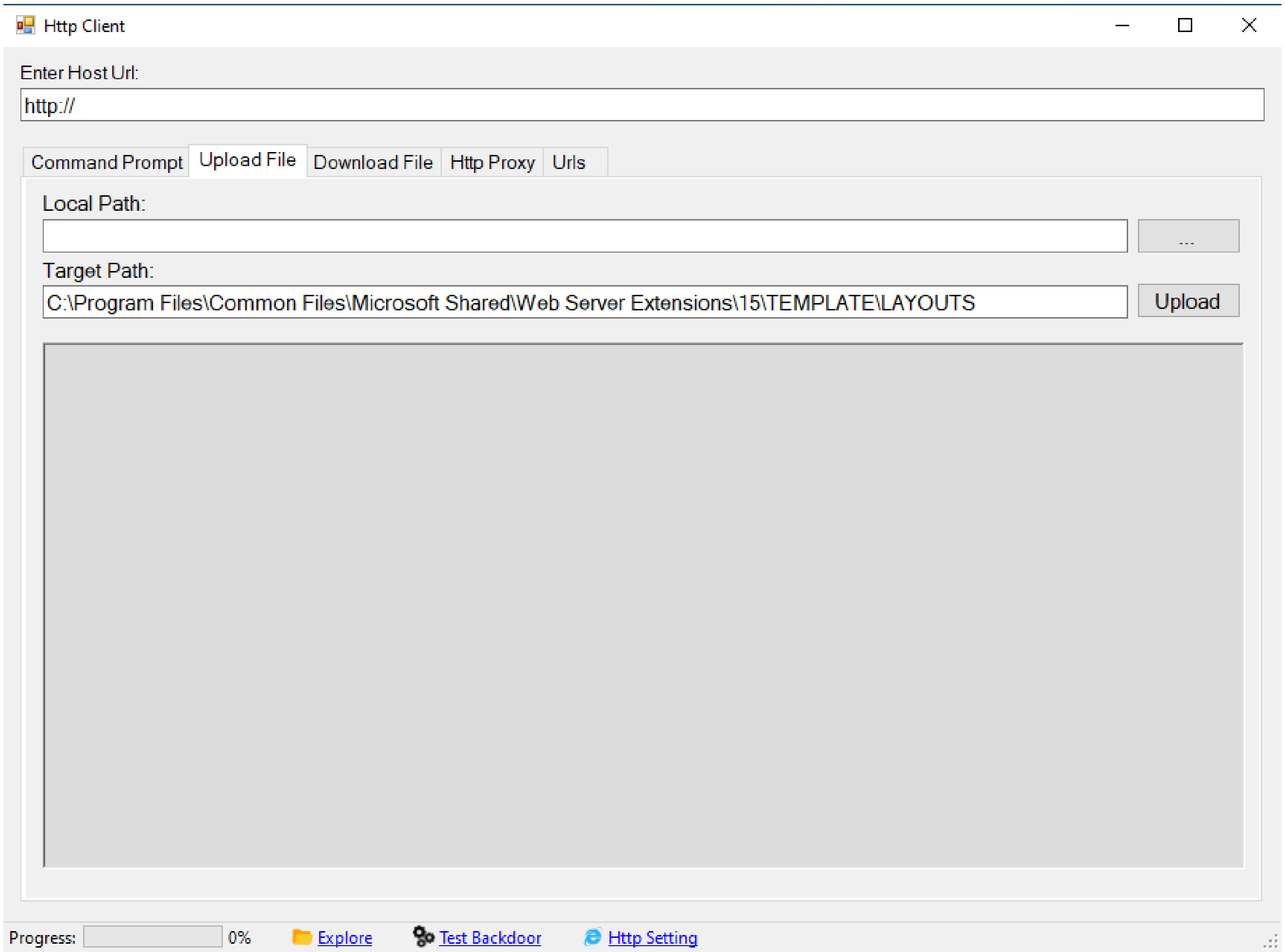
The Upload File Tab (Figure 3) sends a file from a local path to a target path on the remote machine using a POST request. The default target path is `C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15\TEMPLATE\LAYOUTS`.



Enter Host Url: http://

Local Path:

Target Path: C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15\TEMPLATE\LAYOUTS



Enter Host Url: http://

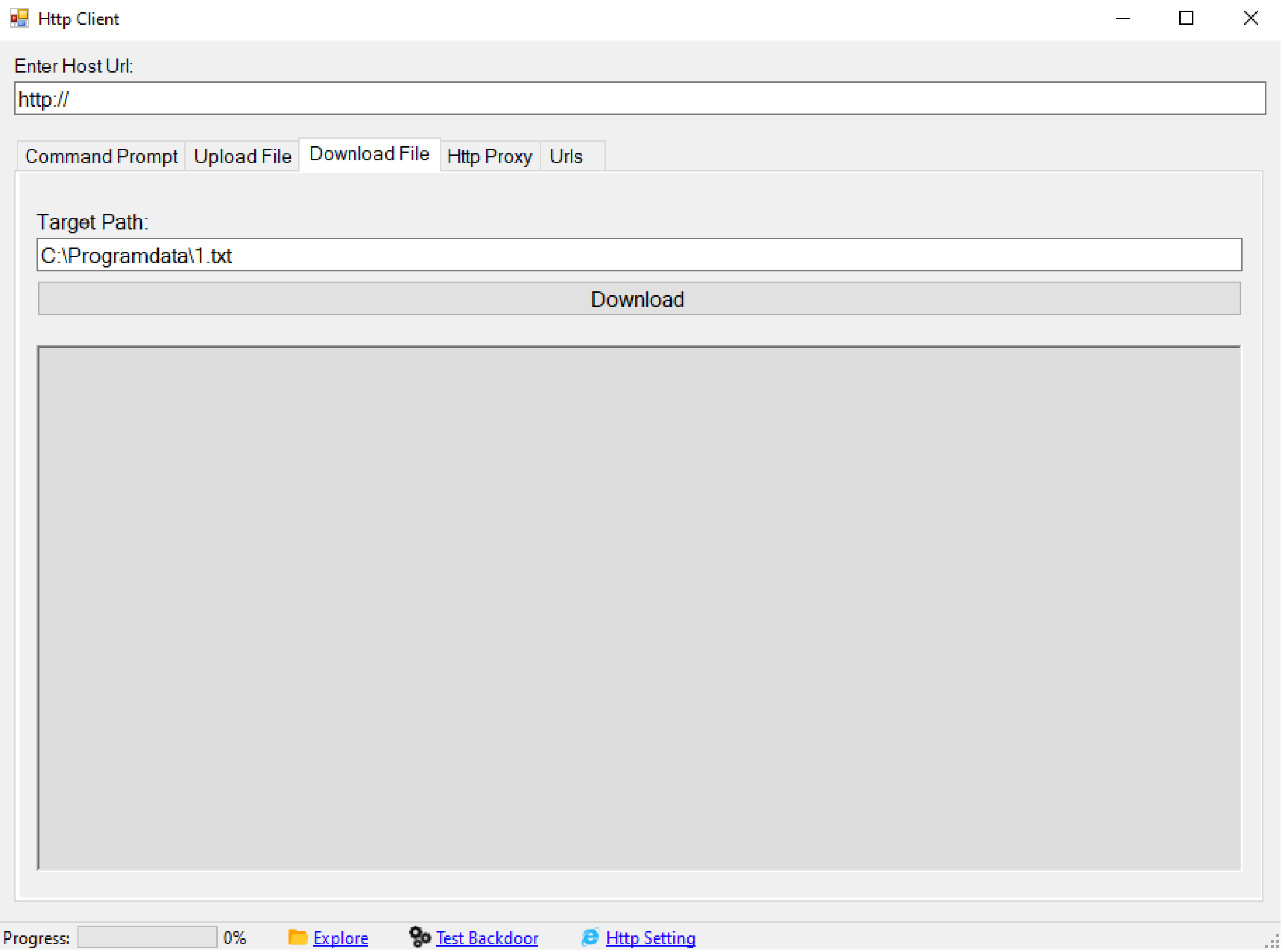
Local Path:

Target Path: C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15\TEMPLATE\LAYOUTS

Progress: 0%

Figure 3: Upload File Tab

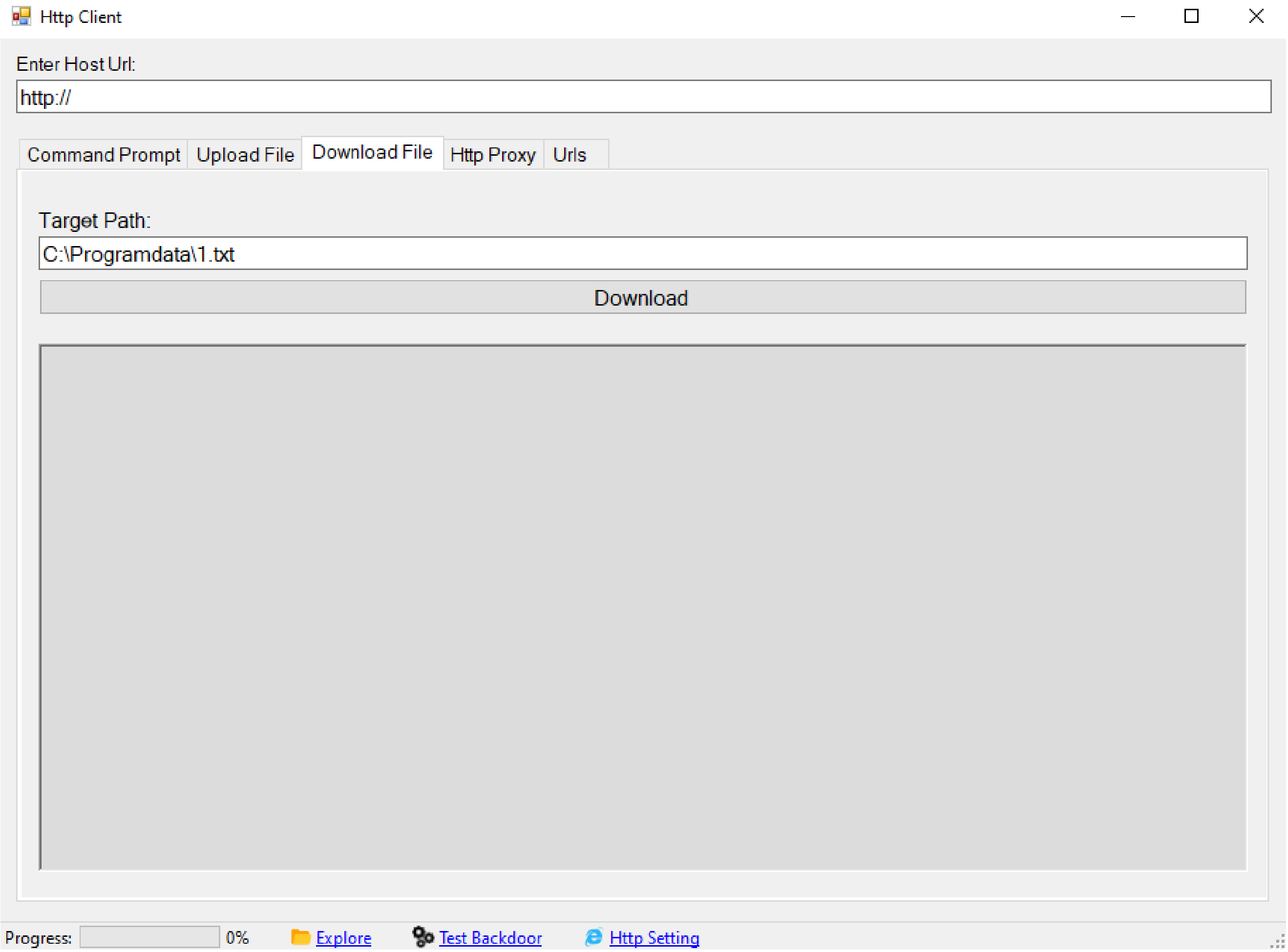
The Download File Tab (Figure 4) is used to obtain a file from a given path on the infected machine. The default path on the infected machine is C:\Programdata\1.txt.



Enter Host Url: http://

Target Path: C:\Programdata\1.txt

Progress: 0%



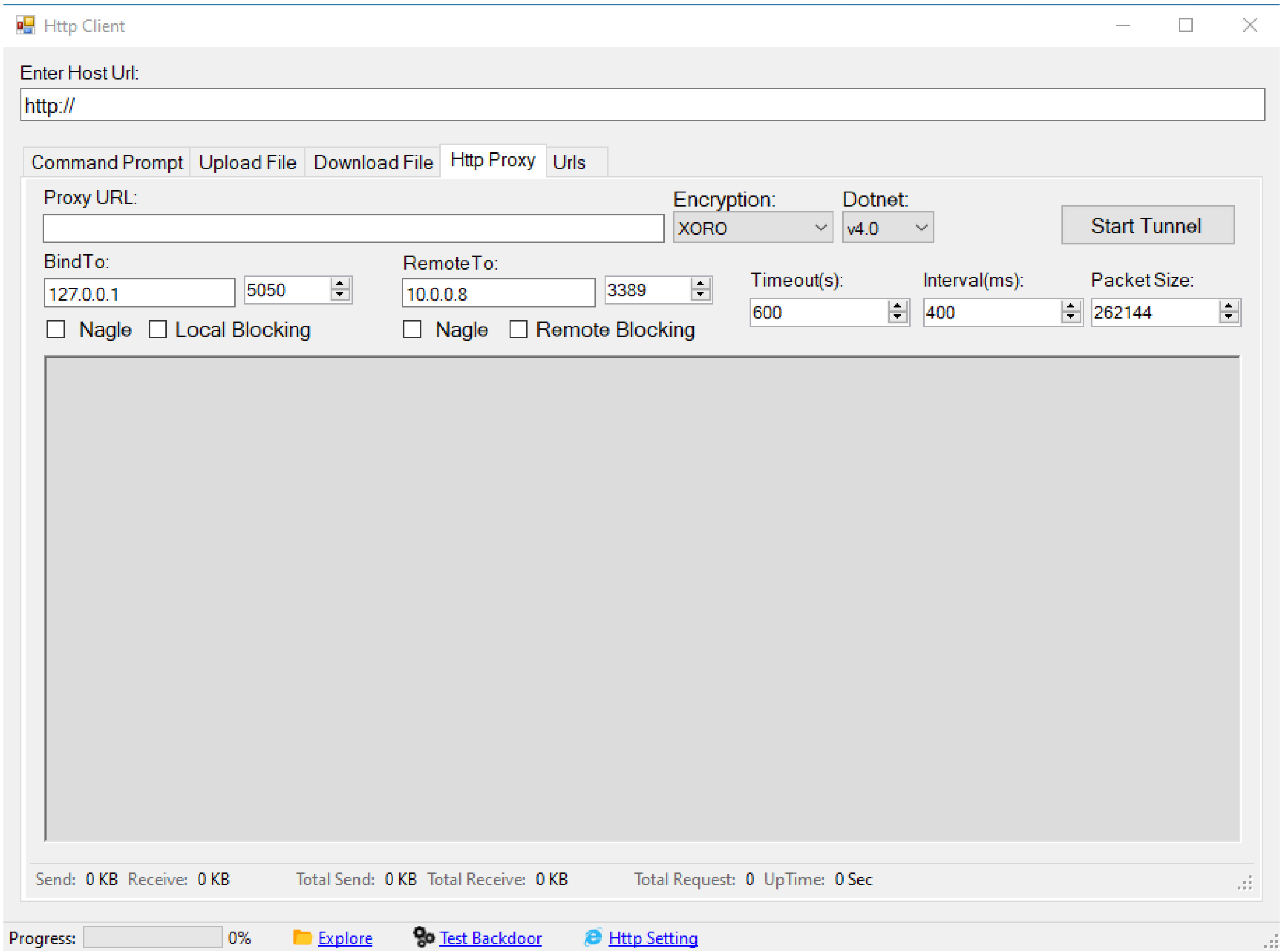
Enter Host Url: http://

Target Path: C:\Programdata\1.txt

Progress: 0%

Figure 4: Download File Tab

The Http Proxy Tab (Figure 5) allows a remote machine infected with TEMPLEDOOR to be used 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.



Enter Host Url: http://

Proxy URL:

Encryption: XORO

Dotnet: v4.0

BindTo: 127.0.0.1 5050

RemoteTo: 10.0.0.8 3389

Timeout(s): 600

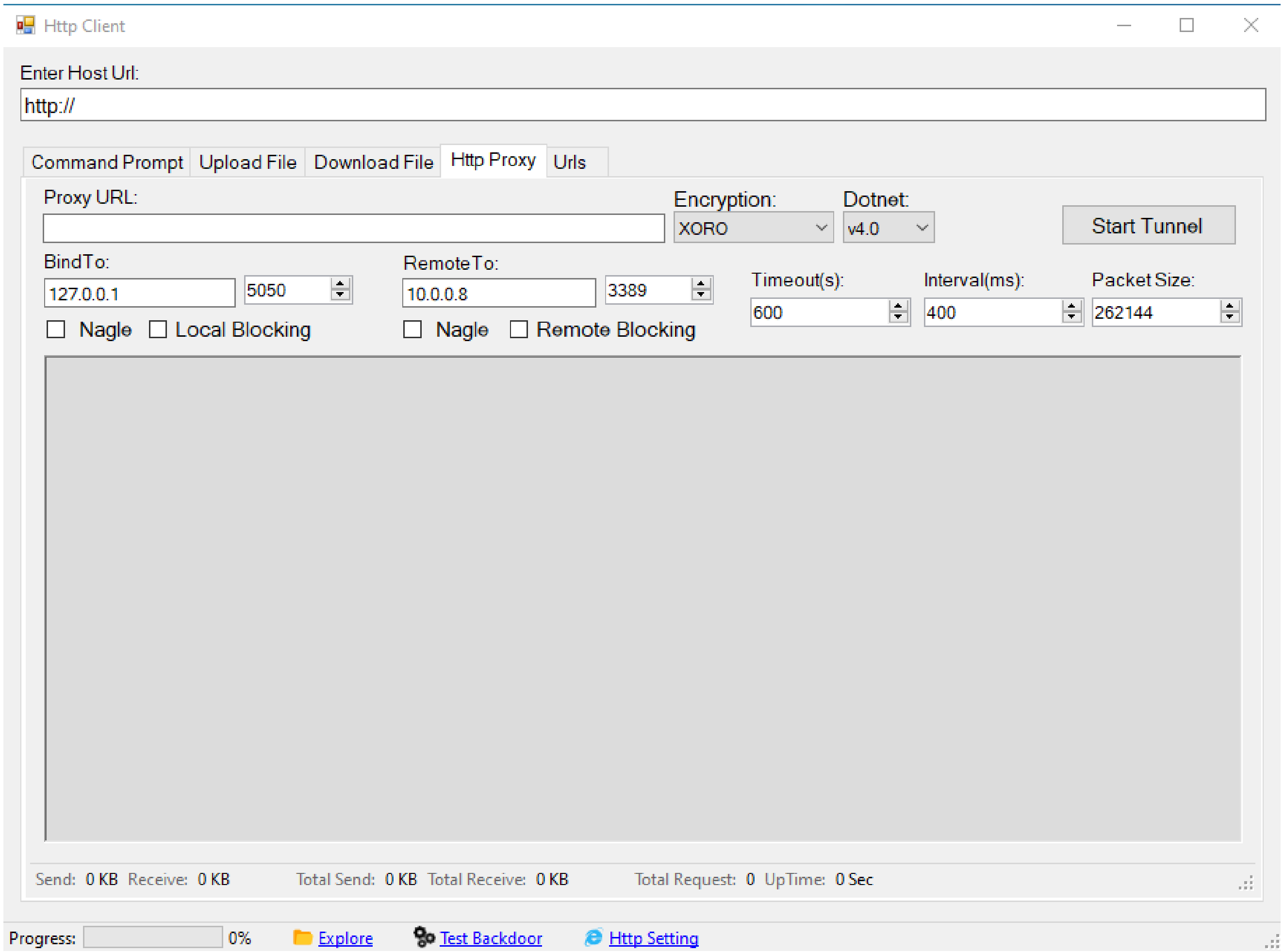
Interval(ms): 400

Packet Size: 262144

□ Nagle □ Local Blocking □ Nagle □ Remote Blocking

Send: 0 KB Receive: 0 KB Total Send: 0 KB Total Receive: 0 KB Total Request: 0 UpTime: 0 Sec

Progress: 0%



Enter Host Url: http://

Proxy URL:

Encryption: XORO

Dotnet: v4.0

BindTo: 127.0.0.1 5050

□ Nagle □ Local Blocking

RemoteTo: 10.0.0.8 3389

□ Nagle □ Remote Blocking

Timeout(s): 600

Interval(ms): 400

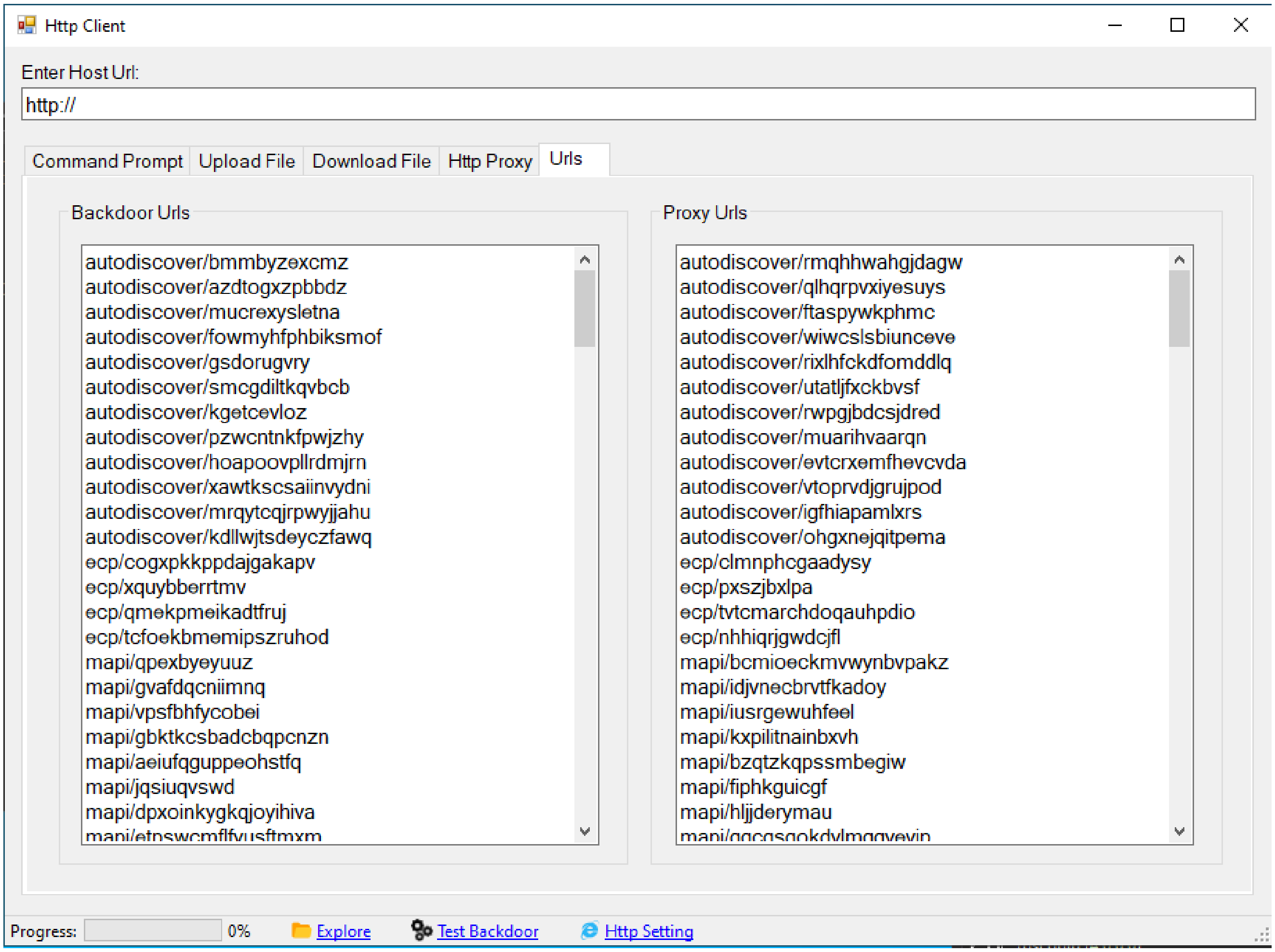
Packet Size: 262144

Send: 0 KB Receive: 0 KB Total Send: 0 KB Total Receive: 0 KB Total Request: 0 UpTime: 0 Sec

Progress: 0%

Figure 5: HTTP Proxy Tab

The URLs Tab (Figure 6) includes URL endpoints that are used when connecting to the infected machine. An endpoint string is 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`).



Enter Host Url: http://

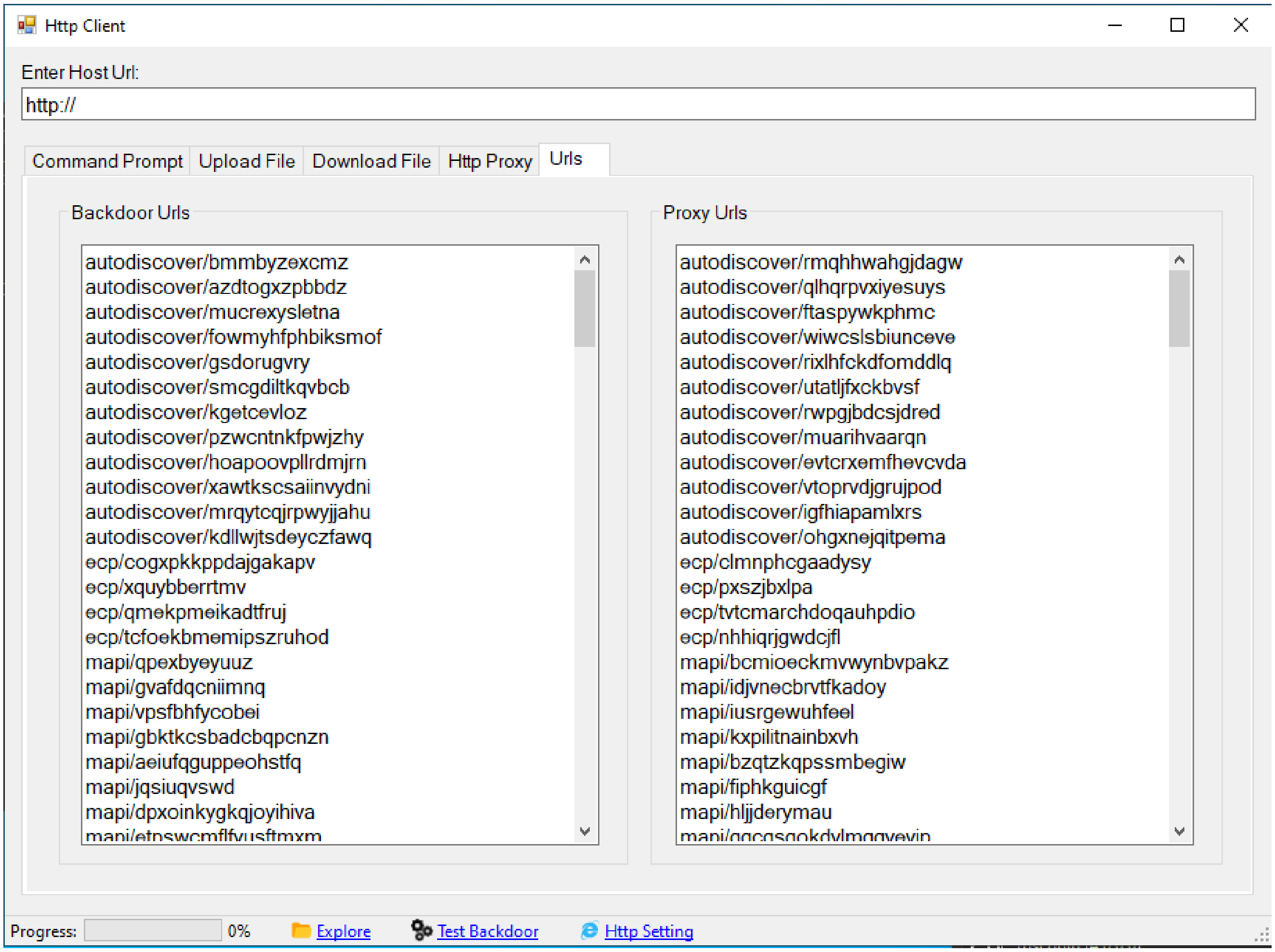
Backdoor Urls

```
autodiscover/bmmbyzexcmz
autodiscover/azdtogxzpbbdz
autodiscover/mucrexysletna
autodiscover/fowmyhfphbiksmof
autodiscover/gsdorugvry
autodiscover/smcgdiltkqvbcb
autodiscover/kgetcevloz
autodiscover/pzwcntnkfpwjzhy
autodiscover/hoapoovpllrdmjrn
autodiscover/xawtkscsaiinvydni
autodiscover/mrqytcqjrpwyjjahu
autodiscover/kdllwjtsdeyczfawq
ecp/cogxpkkppdajgakapv
ecp/xquybberrtmv
ecp/qmekpmeikadtfruj
ecp/tcfoekbmemipszruhod
mapi/qpexbyeyuuz
mapi/gvafdqcniimnq
mapi/vpsfbhfycobei
mapi/gbktkcsbadcbqpcnzn
mapi/aeiufqguppeohstfq
mapi/jqsiuqvswd
mapi/dpxoinkygkqjoyihiva
mapi/etpswcmflfvusftmxm
```

Proxy Urls

```
autodiscover/rmqhhwahgjdagw
autodiscover/qlhqrpvxiyesuys
autodiscover/ftaspywkphmc
autodiscover/wiwcslsbiunceve
autodiscover/rixlhfckdfomddlq
autodiscover/utatljfxckbvsf
autodiscover/rwpgjbdcsjdred
autodiscover/muarihvaarqn
autodiscover/evtcrxemfhevcvda
autodiscover/vtoprvdjgrujpod
autodiscover/igfhiapamlxrs
autodiscover/ohgxnejqitpema
ecp/clmnphcgaadysy
ecp/pxszjbxlpa
ecp/tvtcmarchdoqauhpdio
ecp/nhhiqrjgwdcjfl
mapi/bcmioeckmvwynbvpakz
mapi/idjvnecbrvtfkadoy
mapi/iusrgewuhfeel
mapi/kxpilitnainbxvh
mapi/bzqtzkqpssmbegiw
mapi/fiphkguicgf
mapi/hljjderymau
mapi/gqcqsqokdvlmqqvevin
```

Progress: 0%



Enter Host Url: http://

Backdoor Urls

```
autodiscover/bmmbyzexcmz
autodiscover/azdtogxzpbbdz
autodiscover/mucrexysletna
autodiscover/fowmyhfphbiksmof
autodiscover/gsdorugvry
autodiscover/smcgdiltkqvbcb
autodiscover/kgetcevloz
autodiscover/pzwcntnkfpwjzhy
autodiscover/hoapoovpllrdmjrn
autodiscover/xawtkscsaiinvydni
autodiscover/mrqytcqjrpwyjjahu
autodiscover/kdllwjtsdeyczfawq
ecp/cogxpkkppdajgakapv
ecp/xquybberrtmv
ecp/qmekpmeikadtfruj
ecp/tcfoekbmemipszruhod
mapi/qpexbyeyuuz
mapi/gvafdqcniimnq
mapi/vpsfbhfycobei
mapi/gbktkcsbadcbqpcnzn
mapi/aeiufqguppeohstfq
mapi/jqsiuqvswd
mapi/dpxoinkygkqjoyihiva
mapi/etnswcmflfvusftmxm
```

Proxy Urls

```
autodiscover/rmqhhwahgjdagw
autodiscover/qlhqrpvxiyesuys
autodiscover/ftaspywkphmc
autodiscover/wiwcslsbiunceve
autodiscover/rixlhfckdfomddlq
autodiscover/utatljfxckbvsf
autodiscover/rwpgjbdcsjdred
autodiscover/muarihvaarqn
autodiscover/evtcrxemfhevcvda
autodiscover/vtoprvdjgrujpod
autodiscover/igfhiapamlxrs
autodiscover/ohgxnejqitpema
ecp/clmnphcgaadysy
ecp/pxszjbxlpa
ecp/tvtcmarchdoqauhpdio
ecp/nhhiqrjgwdcjfl
mapi/bcmioeckmvwynbvpakz
mapi/idjvnecbrvtfkadoy
mapi/iusrgewuhfeel
mapi/kxpilitnainbxvh
mapi/bzqtzkqpssmbegiw
mapi/fiphkguicgf
mapi/hljjderymau
mapi/qqcqsqokdvlmqqvevin
```

Progress: 0%

Figure 6: URLs Tab

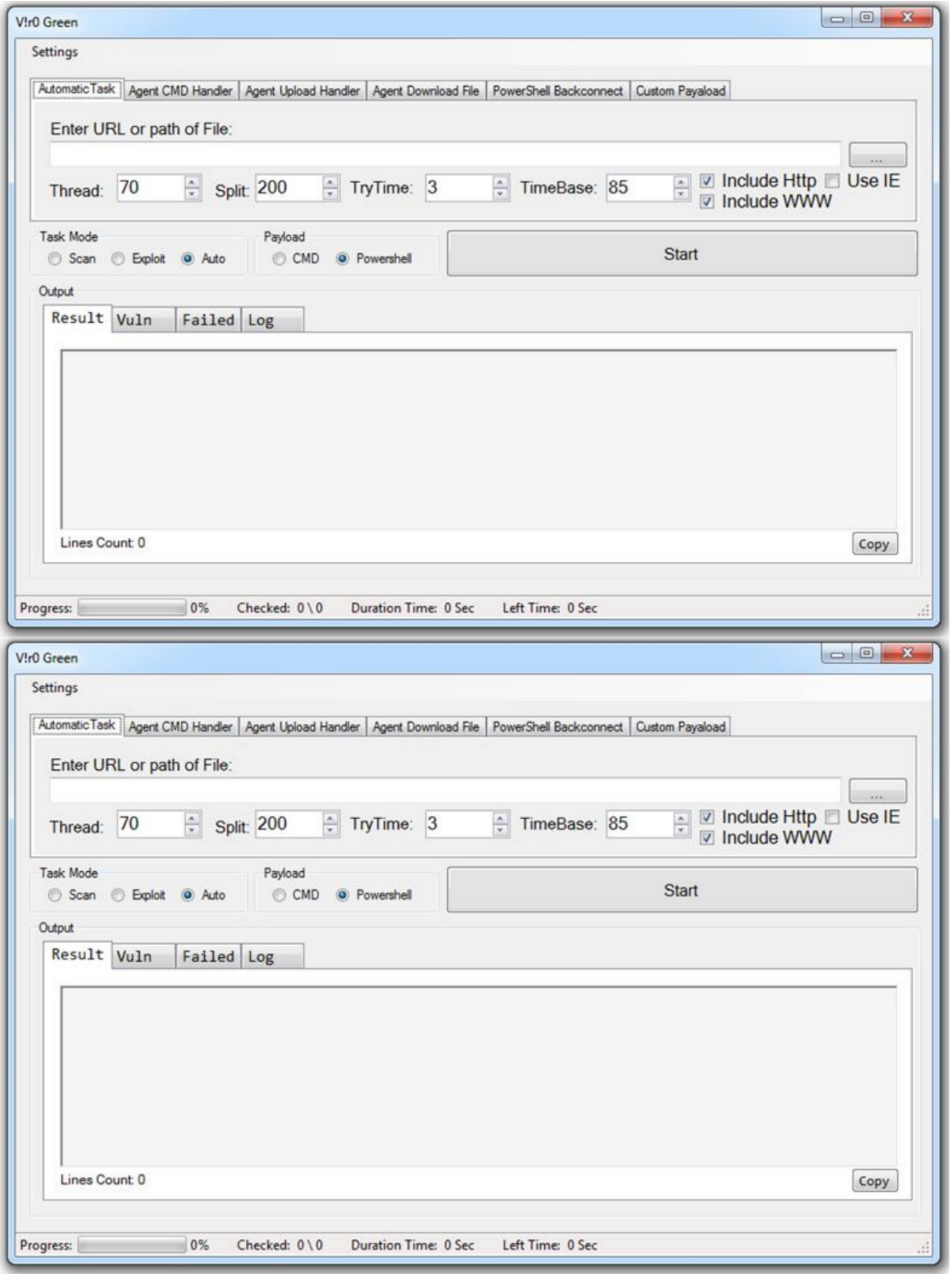
The TEMPLEPLAY GUI also includes a Test Backdoor link, which creates a GET request with the string `wOxhuoSBgpGcnLQZxipa` as the relative URI and checks for the string `UsEPTIkCRUwarKZfRnyjcG13DFA` in the response. This corresponds to an echo \ ping mechanism that was seen in use in the TEMPLEDOOR samples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

Additional links include the Explore link that opens a new Explorer window in the host where the controller runs, and 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.

## VIROGREEN Controller

VIROGREEN is a custom framework used to exploit vulnerable SharePoint servers with CVE-2019-0604 (Figure 7). The framework provides post-exploitation capabilities including scanning for and exploiting CVE-2019-0604; controlling post-exploitation payloads, backdoors (including the STAYSHANTE web shell and the BASEWALK backdoor) and tasking; controlling a compatible agent regardless of how the agent has been implanted; and executing commands and uploading/downloading files.

Additional details on TEMPLEPLAY and VIROGREEN can be found in the Technical Annex.



Enter URL or path of File:

Thread: 70 Split: 200 TryTime: 3 TimeBase: 85 ☑ Include Http □ Use IE ☑ Include WWW

Task Mode ○ Scan ○ Exploit ● Auto

Payload ○ CMD ● Powershell

Lines Count: 0

Progress: 0% Checked: 0\0 Duration Time: 0 Sec Left Time: 0 Sec

Enter URL or path of File:

Thread: 70 Split: 200 TryTime: 3 TimeBase: 85 ☑ Include Http □ Use IE ☑ Include WWW

Task Mode ○ Scan ○ Exploit ● Auto

Payload ○ CMD ● Powershell

Lines Count: 0

Progress: 0% Checked: 0\0 Duration Time: 0 Sec Left Time: 0 Sec

Figure 7: VIROGREEN GUI

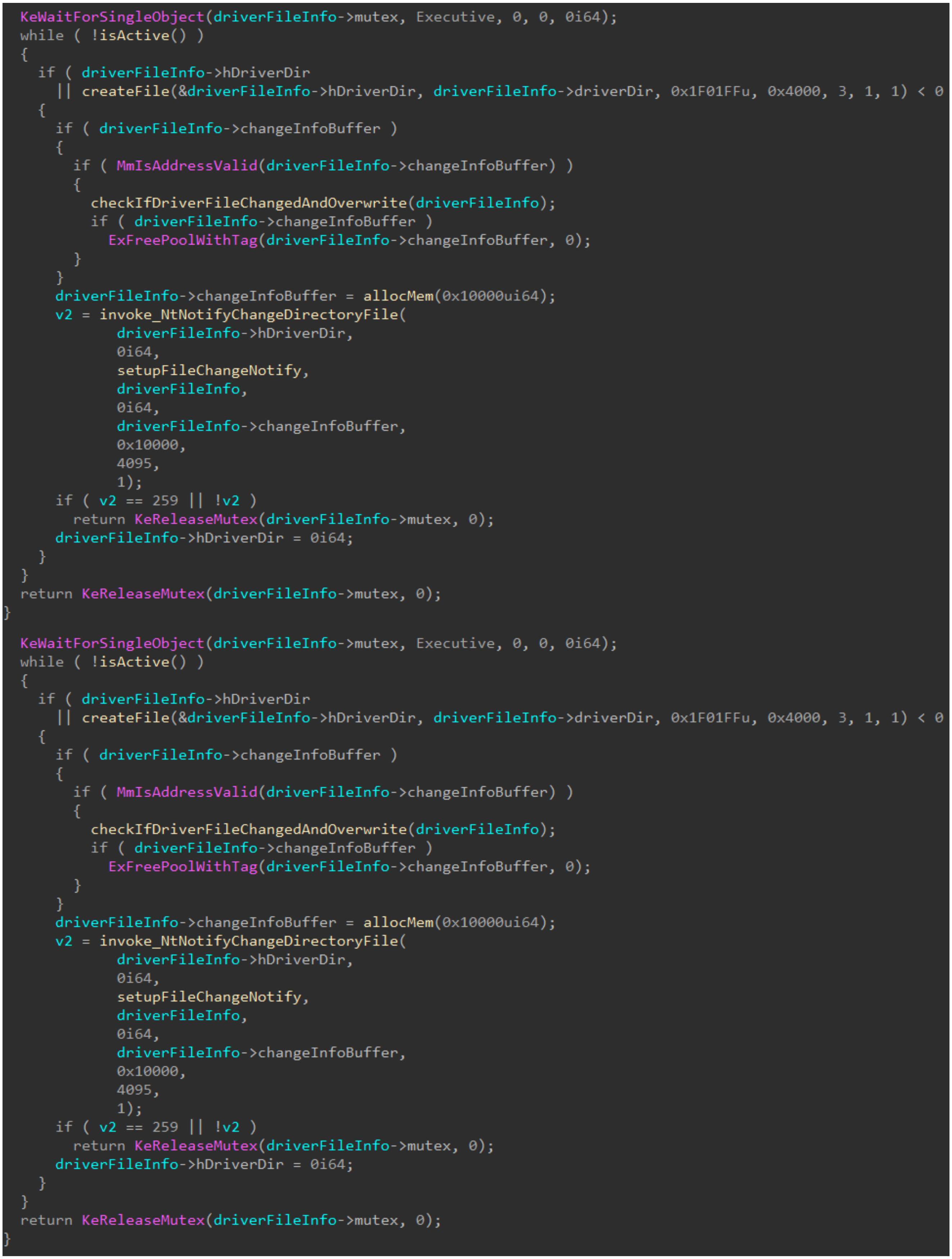
## UNC1860 Malware: Gaining Persistent Access

UNC1860 gains initial access to victim environments in an opportunistic manner via the exploitation of vulnerable internet-facing servers leading to web shell deployment. After obtaining an initial foothold, the group typically deploys additional utilities and a selective suite of passive implants that are designed to be stealthier than common backdoors. These provide a higher degree of operational security by removing the dependency for classic C2 infrastructure, making detection more difficult for network defenders. Cisco and Check Point have provided extensive analysis on UNC1860's passive implants that correspond to OATBOAT, a loader that loads and executes shellcode payloads; Fortinet additionally provided analysis regarding the Windows kernel driver, WINTAPIX, which has similar code to a malicious driver we track as TOFUDRV (Figure 8 and Figure 9).

A key feature of UNC1860 includes its maintenance of this diverse collection of passive/listener-based utilities that support the group's initial access and lateral movement goals. We believe the group additionally maintains a smaller collection of “main-stage” backdoors that have greater capabilities than the usual web shells and small .NET utilities that may be deployed for select high-priority victims in the telecommunications sector. These implants demonstrate the group's keen understanding of the Windows operating system (OS) and network detection solutions, reverse engineering capabilities of Windows kernel components, and detection evasion capabilities.

- Passive implants do not initiate outbound traffic from the victim network to a C2 server. Further, the inbound traffic containing commands or payloads can arrive from any volatile source (e.g., VPN nodes within the target country, from another victim, or even internally from another part of the victim network). This makes network monitoring more difficult. Web shells and passive implants leverage HTTPS-encrypted traffic so commands/payloads cannot be extracted from captured network traffic.
- Both passive implants TOFUDRV and TOFULOAD leverage undocumented Input/Output Control commands for communication, which requires knowledge of the OS and can lower the chances of this traffic being detected by endpoint detection and response (EDR) solutions.
- Loading drivers is a "high risk / high reward" situation as loading them without creating a critical error screen requires extensive knowledge both of the OS internals and victim environments; however, using them promises lower detection rates and possibilities akin to filtering drivers, which act as middlemen allowing for the inspection, modification, or blocking of network traffic before it reaches the device or application, as well as assets like file system objects and registry entries.

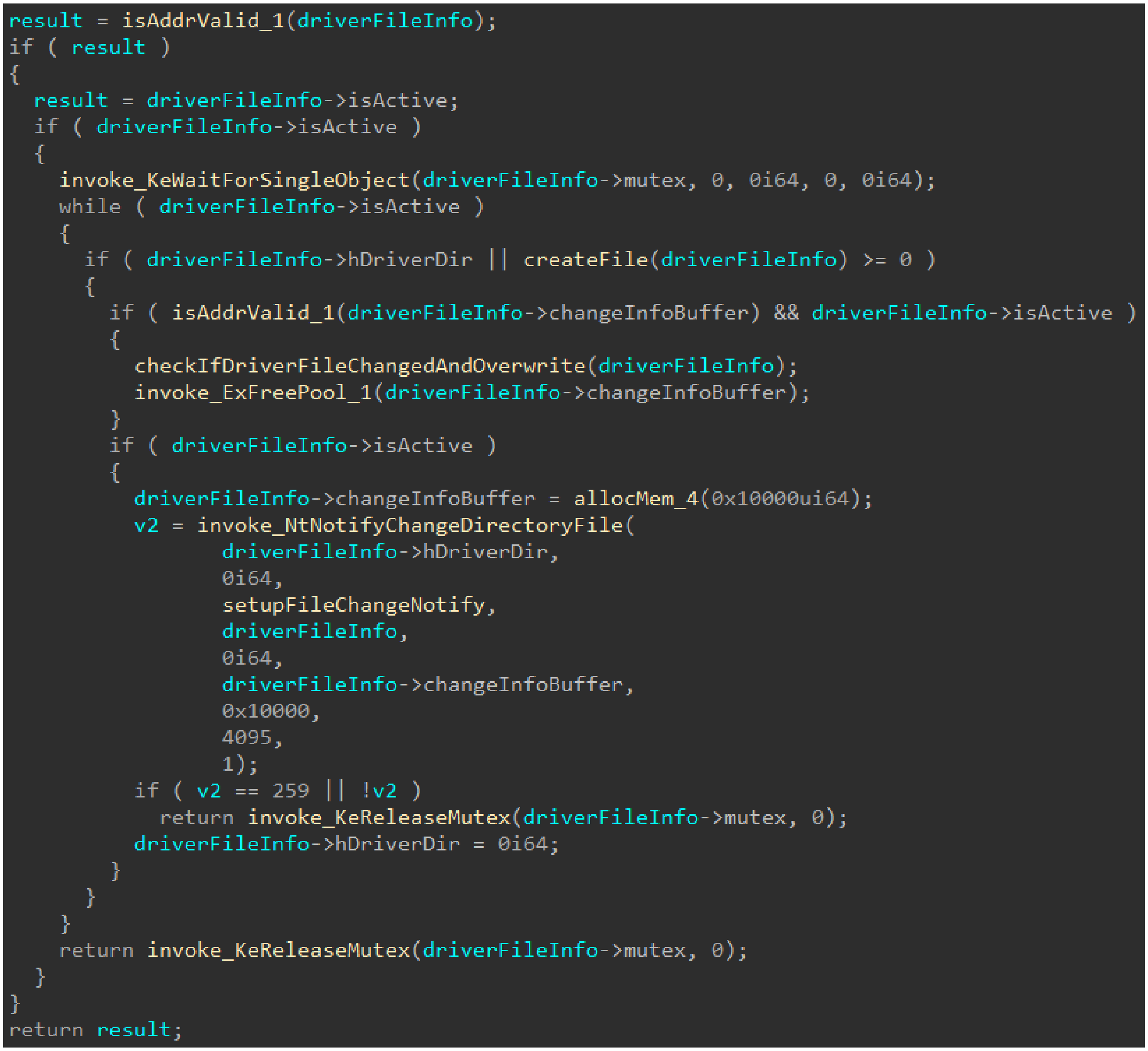
- The passive backdoor TEMPLEDROP repurposed an Iranian AV software Windows file system filter driver named Sheed AV (MD5: 0c93cac9854831da5f761ee98bb40c37) for the purpose of protecting some of the files it deploys as well as its own file from modification.
- A .NET-based utility for defense evasion tracked as TEMPLELOCK was observed being implemented in both foothold utilities such as ROTPIPE and more complex passive implants such as TEMPLEDROP. TEMPLELOCK is capable of terminating threats associated with the Windows Event Log service and restarting the service's operation on demand.



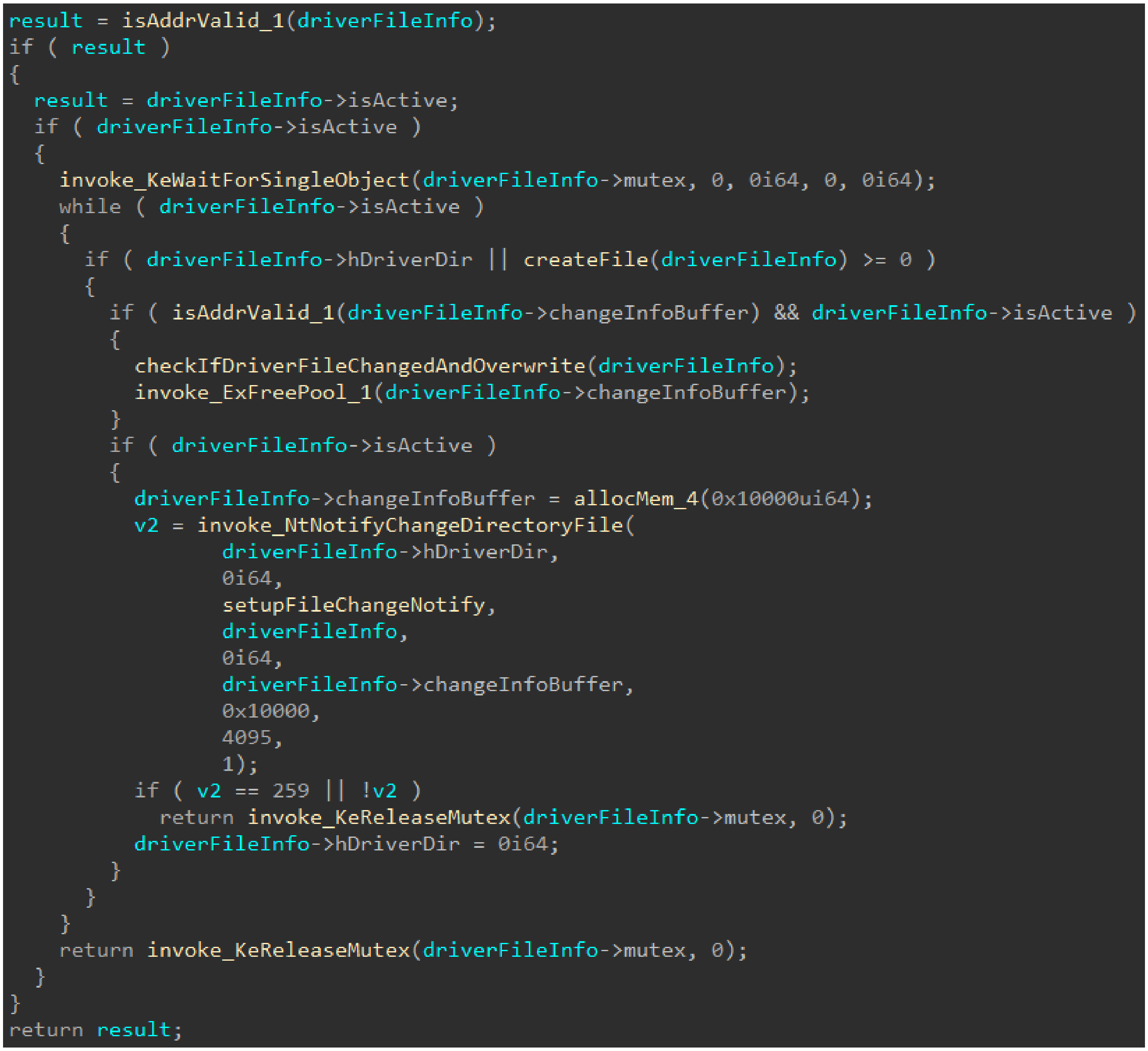
```
  KeWaitForSingleObject(driverFileInfo->mutex, Executive, 0, 0, 0i64);
  while ( !isActive() )
  {
    if ( driverFileInfo->hDriverDir
      || createFile(&driverFileInfo->hDriverDir, driverFileInfo->driverDir, 0x1F01FFu, 0x4000, 3, 1, 1) < 0
    {
      if ( driverFileInfo->changeInfoBuffer )
      {
        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);
}
```

```
  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 8: 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;
```



```
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 9: Driver file protection logic in TOFUDRV (MD5: b4b1e285b9f666ae7304a456da01545e)

## UNC1860 Unique Artifacts Suggest Consistent Development Support

In addition to the previous observations, we identified the following recurring artifacts related to the group’s independent implementation of Base64 encoding/decoding and XOR encryption/decryption in .NET code, despite these functions being available in build-in .NET code.

The intent of the independent implementation of these functions is not entirely clear. Nevertheless, it is highly likely that using such custom libraries bypasses common detections by EDRs and other security tools—detections designed to identify usage combinations of functions commonly seen in malware. Additionally, using these custom libraries may allow better compatibility if any of the built-in functions change in a specific version of a .NET control to ensure the group's tooling is always compatible with its encryption and encoding schemes and/or to better help evade detection.

- We observed the same encoding method using the Base64 algorithm to encode and decode data sent between controllers and proxy servers. In several cases, we identified the reuse of a seemingly misspelled Base64 DLL using the name "bsae64" in both foothold utilities deployed via SASHEYAWAY and passive implants including TEMPLEDOOR.

- We observed the same rolling encryption module, XORO (MD5: `57cd8e220465aa8030755d4009d0117c),` dropped by the TANKSHELL utility; TUNNELBOI network tunneller capable of establishing a connection with a remote host, managing web shells on the network, and creating RDP connections; and the TEMPLEPLAY controller.

### Foothold Utilities and Backdoors and Malware Use for Longer Term Persistence

Mandiant is tracking multiple foothold utilities and backdoors used in UNC1860 initial access operations. These generally use custom obfuscation methods that can lower detection rates and make analysis more difficult by renaming strings and function names.  Additionally, we are tracking numerous code families that we consider to be UNC1860 "main-stage" implants that further increase the group's persistence in victim environments.

Please see the Technical Annex for more information.

## Additional Protection Information for Google Cloud Customers

For Google SecOps Enterprise+ customers, SecOps rules have been released to the Emerging Threats rule pack, and IOCs listed in this blog post are available for prioritization with Applied Threat Intelligence.

## Indicators of Compromise (IOCs)

A Google Threat Intelligence Collection featuring IOCs related to the activity described in this post is now available for registered users.

### MD5 Hashes

| | |
|---|---|
| `1176381da7dea356f3377a59a6f0e799` | `41f4732ed369f2224a422752860b0bc5` |

| | |
|---|---|
| 4029bc4a06638bb9ac4b8528523b72f6 | 126bc1c30fba27f8bf67dce4892b1e8c |
| 0c9ff0db00f04fd4c6a9160bffd85a1d | a7693e399602eb79db537c5022dd1e01 |
| d9719f6738dbfaa21be7f184512fe074 | 17b27e6aa0ab6501f11bb4d2e0f829ff |
| 4dd6250eb2d368f500949952eb013964 | 69fd67c115349abb4a313230a1692642 |
| 7f5f5f290910d256e6b012f898c88bf3 | c90ec587e3333dabb647ebc182673460 |
| efe8043e1b4214640c5f7b5ddf737653 | a90236e4962620949b720f647a91f101 |
| b26d54b7da7b2bf600104f69da4ea00f | d87ca3f830b8b53fde358bb64900f6af |
| c50ae2c4b76f0d5724ec240568c78c4f | 57cd8e220465aa8030755d4009d0117c |
| 4b2c78bb2c439998cff0cc097a14b942 | 4abcf21b63781a53bbc1aa17bd8d2cbc |
| a3ea0d13848a104c28d035a9d518acc2 | bd6464f12bb6f7f02b6ffebb363d8e5f |
| f89be788e4adf665acf1a8ef8fcaa133 | f292e61774c267c3787fdfcace50ea7b |
| c11a4e4a2d484513f79bd127a0387b0c | 14e54ff4805840e656efb8cd38de4751 |
| 3d5d05f230ae702c04098de512d93d48 | a038975255d3dda636d86ccd307f7838 |
| 31f2369d2e38c78f5b3f2035dba07c08 | c21eefc65cda49f17ddd1d243a7bffb5 |
| c8fa0ce3ae6a13af640607ea606c55f9 | 2cece71e107d12ffd74b2fb24bf339a6 |
| fa1c6f7a5e02374b9d33de2578cb3399 | 1e896f026246872b2feb4f8e3e093815 |
| 57c916da83cc634af22bde0ad44d0db3 | 07db3058e32fe5f36823dc7092cd7d5b |
| 3dd829fb27353622eff34be1eabb8f18 | 1e6679cd25d1bb127a0bec665adcf21e |
| 2e803d28809be2a0216f25126efde37b | 2398a83f10329a107801d3d23d06f7cb |

| | |
|---|---|
| 73fb0fe5cd96a14a4f85639223aec6a8 | 85427a8a47c4162b48d8dfb37440665d |
| a500561c0b374816972094c2aa90da2a | a65ee1a82975ee4c8d4e70219e1bfff5 |
| ce537dd649a391e52c27a3f88a0a8912 | e67687b4443f58d2b0a465e3af3caffe |
| b34883fb1630db43e06a38cebfa0bce2 | 46804472541ed61cc904cd14be18fe1d |
| 4de802f7e61cb8c820a02e042b58b215 | 929b12bc9f9e5f8e854de1d46ebf40d9 |
| f0dfb7bf01c0412891da8fa2702f4c7b | b219672bcd60ce9a81b900217b3b5864 |
| fc90907e70f18c7f6a6b9d9599b6f97c | d1e45afbfd3424612b4a4218cc7357ef |
| da0085a97c38ead734885e5cced1847f | 490590bfdeeedf44b3ae306409bb0d03 |
| e86e885e6c96ac72482741d8696c17fb | ca3f0d25f7da0e8cde8e1f367451c77a |
| 7b2fa099d51fa3885766f6d60d768748 | 6626dbe74acd15d06ff6900071ef240c |

## YARA Rules

```
rule M_Autopatt_DropperMemonly_WINTAPIX_1 {
    meta:
        author = Mandiant
        description = "wintapix malware family"
        created = "06/26/2023"
        modified = "06/26/2023"
        version = "1.0"

    strings:
        $p00_0 = {84ec5ff5f84863f6e9[4]66458b65??4981c5[4]4d0faccf}
        $p00_1 = {0f16c00f11014c03c14883c1??
4883e1??4c2bc14d8bc849c1e9??74}
    condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and
        (
            ($p00_0 in (660000..690000) and $p00_1 in (9700..20000))
        )
}
```

```
import "pe"
rule M_WINTAPIX_StringDecodingMethod_1 {
    meta:
        author = Mandiant
        hash1 = "286bd9c2670215d3cb4790aac4552f22"
        hash2 = "4dd6250eb2d368f500949952eb013964"
        desc = "Detects the byte pattern of a string decoding
method found in the WINTAPIX driver image"
    strings:
        $a1 = { 48 89 54 24 10 48 89 4C 24 08 48 83 EC 18 C7
04 24 00 00 00 00 48 63 04 24 48 8B 4C 24 ?? 0F BE 04 01
48 8B 4C 24 ?? 0F B6 49 ?? 33 C1 48 63 0C 24 48 8B 54 24
?? 88 04 0A 8B 04 24 FF C0 89 04 24 8B 04 24 FF C8 48 98
48 8B 4C 24 ?? 0F B6 04 01 85 C0 75 }
    condition:
        uint16(0) == 0x5A4D and
        filesize < 1MB and
        pe.subsystem == pe.SUBSYSTEM_NATIVE and
        all of them
}

import "pe"
rule M_WINTAPIX_PaddedStrings_1 {
    meta:
        author = Mandiant
        hash1 = "286bd9c2670215d3cb4790aac4552f22"
        hash2 = "4dd6250eb2d368f500949952eb013964"
        desc = "Detects unique strings found in the WINTAPIX
driver image"
    strings:
        $a1 = { CC CC CC CC CC CC CC 4E 74 44 65 6C 61 79
45 78 65 63 75 74 69 6F 6E 00 }
        $a2 = { CC CC CC CC CC 5C 00 }
        $a3 = "InitSafeBootMode" ascii fullword
    condition:
        uint16(0) == 0x5A4D and
        pe.subsystem == pe.SUBSYSTEM_NATIVE and
        filesize < 1MB and
        (
            (
                all of them and
                #a2 == 2
            ) or
            pe.imphash() == "8d070a93a45ed8ba6dba6bfbe0d084e7"
        )
}
```

```
import "dotnet"
rule M_UNC1860_TEMPLEDOOR_Strings_1 {
    meta:
        author = Mandiant
        date = "28/02/2024"
        hash1 = "caffdb648a0a68cd36694f0f0c7699d7"
        desc = "Detects the TEMPLEDOOR family based on
unique strings"
        comment = "Triggers on TUNNELBOI sample
c517519097bff386dc1784d98ad93f9d"
    strings:
        $url = "{0}://+:{1}/{2}/" wide fullword
        $a1 = "+CjxoZWFkPgo8bWV0YSBodHRwLWVxdWl2
PSJDb250ZW50LVR5cGUiIGNvbnRlbnQ9InRleHQvaHRtb
DsgY2hhcnNldD1pc28tODg1OS0xIi8" wide
        $b1 = "Jet" wide fullword
        $b2 = " Ver" wide fullword
        $b3 = "CmD" wide fullword
        $c1 = "Command" wide fullword
        $c2 = "Upload" wide fullword
        $c3 = "Download" wide fullword
        $c4 = "Load" wide fullword
        $c5 = "Rundll" wide fullword
        $c6 = "ERROR" wide fullword

    condition:
        int16(0) == 0x5a4d and
        uint32(uint32(0x3C)) == 0x00004550 and
        dotnet.is_dotnet and
        $url and
        (
            $a1 or
            2 of ($b*) or
            5 of ($c*)
        )
}
```

```
import "dotnet"
rule M_UNC1860_TEMPLEDOOR_BytePatterns_1 {
    meta:
        author = Mandiant
        date = "28/02/2024"
        hash1 = "caffdb648a0a68cd36694f0f0c7699d7"
        desc = "Detects the TEMPLEDOOR family based
on unique byte patterns"
        comment = "Triggers on TUNNELBOI sample
c517519097bff386dc1784d98ad93f9d and on WINPAY
sample b219672bcd60ce9a81b900217b3b5864"
    strings:
        $encode_msil = { 7E ?? ?? 00 04 1F 41 1F 61 6F ??
?? 00 0A D2 0A 02 2C 07 02 8E 16 FE 03 2B 01 16 2C 69
16 0B 2B 0F 02 07 02 07 91 06 61 19 58 D2 9C 07 17 58
0B 07 02 8E 69 FE 04 2D E9 02 28 ?? ?? 00 0A } // Packet
encoding method MSIL
        $encryption_key = { 54 62 2d 0c 03 45 49 15 2b 43
59 4a 4e 0c 40 }
    condition:
        int16(0) == 0x5a4d and
        uint32(uint32(0x3C)) == 0x00004550 and
        dotnet.is_dotnet and
        any of them
}
```

```
rule M_OBFUSLAY_UNC1860_1 {
    meta:
        desc = "Detects the UNC1860 OBFUSLAY malware by its
string decryption method"
        rs1 = "b66919a18322aa4ce2ad47d149b7fe38063cd3cfa2
e4062cd1a01ad6b3e47651"
    strings:
        $a1 = {
            FE 09 00 00
            6F ?? 00 00 0A
            FE 0E 00 00
            FE 0C 00 00
            20 02 00 00 00
            5B
            8D ?? 00 00 01
            FE 0E 01 00
            20 00 00 00 00
            FE 0E 04 00
            38 39 00 00 00
            FE 0C 01 00
            FE 0C 04 00
            20 02 00 00 00
            5B
            FE 09 00 00
            FE 0C 04 00
            20 02 00 00 00
            6F ?? 00 00 0A
            20 10 00 00 00
            28 ?? 00 00 0A
            9C
            FE 0C 04 00
            20 02 00 00 00
            58
            FE 0E 04 00
            FE 0C 04 00
            FE 0C 00 00
            3F BA FF FF FF
            FE 0C 01 00
        }
    condition:
        uint16(0) == 0x5A4D and
        all of them
}
```

```
rule M_APT_CRYPTOSLAY_UNC1860_1 {
    meta:
        desc = "Detects the UNC1860 CRYPTOSLAY malware by its
string decryption method"
        rs1 = "3F2FD2DFD27BF3CAFCBF0946E308832E11A1D9C1
D98FB04AC848E023E6720F53"
        rs2 = "5c1a42e9baaec115df337d2f4a9dcce8d73f29375921
827e367fcba8499cdfa2"
    strings:
        $a1 = {
            FE 09 00 00
            6F ?? 00 00 0A
            FE 0E 00 00
            FE 0C 00 00
            20 02 00 00 00
            5B
            8D ?? 00 00 01
            FE 0E 01 00
            20 00 00 00 00
            FE 0E 04 00
            38 39 00 00 00
            FE 0C 01 00
            FE 0C 04 00
            20 02 00 00 00
            5B
            FE 09 00 00
            FE 0C 04 00
            20 02 00 00 00
            6F ?? 00 00 0A
            20 10 00 00 00
            28 ?? 00 00 0A
            9C
            FE 0C 04 00
            20 02 00 00 00
            58
            FE 0E 04 00
            FE 0C 04 00
            FE 0C 00 00
            3F BA FF FF FF
            28 ?? 00 00 0A
        }
        $a2 = {
            FE 09 00 00
            6F ?? 00 00 0A
            FE 0E 00 00
            FE 0C 00 00
            20 02 00 00 00
            5B
            8D ?? 00 00 01
            FE 0E 01 00
            20 00 00 00 00
            FE 0E 06 00
            38 39 00 00 00
            FE 0C 01 00
            FE 0C 06 00
            20 02 00 00 00
            5B
            FE 09 00 00
            FE 0C 06 00
            20 02 00 00 00
            6F ?? 00 00 0A
            20 10 00 00 00
            28 ?? 00 00 0A
            9C
            FE 0C 06 00
            20 02 00 00 00
            58
            FE 0E 06 00
            FE 0C 06 00
            FE 0C 00 00
            FE 04
            FE 0E 07 00
            FE 0C 07 00
            3A B0 FF FF FF
        }
    condition:
        uint16(0) == 0x5A4D and
        any of them
}

rule M_Autopatt_DropperMemonly_OATBOAT_1 {
    meta:
        author = "autopatt"
        description = "oatboat malware family"
        created = "02/09/2024"
        modified = "02/09/2024"
        version = "1.0"

    strings:
        $p00_0 = {48897c24??55488bec4883ec??488bf9c745[5]33d
bc745[5]488d4d}
        $p00_1 = {443ac975??48ffc64883c3??493bf372??498b42??4885c075}
    condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and
        (
            ($p00_0 in (250..6500) and $p00_1 in (0..6000))
        )
}
```

```
rule SASHEYAWAY_Strings_1 {
    meta:
        desc = "Strings observed in the webshell loader"
        rs1 = "2538767f13218503bccf31fccb74e753199
4b69a36a3780b53ba5020d938af20"
    strings:
        $ = "FromBase64String"
        $ = "Page Language=\"C#\""
        $ = "private static System.Reflection.Assembly"
        $ = "Page_Load"
        $ = "System.Reflection.MethodInfo"
        $ = "Activator.CreateInstance"
        $ = "Invoke"
    condition:
        all of them
}

rule M_Hunting_Backdoor_TOFULOAD_1 {
  meta:
    author = Mandiant
    date_created = "2023-08-15"
    date_modified = "2023-08-15"
    description = "This is a hunting rule to look for TOFULOAD
backdoor used by UNC1860"
    md5 = "d1ce3117060e85247145c82005dda985"
  strings:
    $s1 = {66 77 88 99 48 8D [2] C7 [2] 52 74 6C 52}
              // 0x99887766; LEA ??, ??; MOV ??, 'RltR';
    $s2 = {B8 E1 83 0F 3E F7 [1] C1 [1] 03 0F [2] 6B [1] 21}
      // MOV ??, 0x3E0F83E1; MUL ??, ??; SHR ??, 03; MOVZX ??, ??;
IMUL ??, ??, 21;
    $s3 = {FF [1] 40 [2] 43 32 [2] 41 88 [3] 44 8B [1] 4D [2] 7C} //
INC ??; MOV ??, ??; XOR ??, ??; MOV ??, ??; MOV ??, ??; CMP ??, ??; JL
  condition:
    filesize < 50KB and
    any of them
}
```

```
import "dotnet"
rule M_UNC1860_TEMPLEDROP_Strings_2 {
    meta:
        author = Mandiant
        date = "28/02/2024"
        hash1 = "6d3041b89484c273376e5189e190d235"
        desc = "Detects the TEMPELDROP family based on unique strings"
        comment = "Triggers on TEMPLEDOOR controller sample c517519
097bff386dc1784d98ad93f9d"
    strings:
        $a1 = "Nothing changed :D" wide fullword
        $a2 = "Access: KO" wide fullword
        $a3 = "Eventlog stoped." wide fullword
        $b1 = "The Microsoft Exchange Self Protection Driver." wide fullword
        $b2 = "The Microsoft Exchange Filter Driver." wide fullword
        $c1 = "Create RegKey: " wide
        $c2 = "Create Service: " wide
        $c3 = "Test Event lock: " wide
        $c4 = "Test http listner: " wide
        $c5 = "Test IO Changes: " wide
        $c6 = "Test 'Event lock': " wide
        $d1 = "no active http port to listen." wide
        $d2 = "Prefixes.Add Error , " wide
        $d3 = "' driver service created and started." wide
        $d4 = "' service started." wide
        $d5 = "Unhandled exception on create reg key " wide
        $d6 = "Failed to change file 'CreationTime'." wide

    condition:
        int16(0) == 0x5a4d and
        uint32(uint32(0x3C)) == 0x00004550 and
        dotnet.is_dotnet and
        (
            1 of ($a*) or
            1 of ($b*) or
            2 of ($c*) or
            2 of ($d*)
        )
}
```

```
rule M_Autopatt_Backdoor_TOFUDRV_1 {
    meta:
        author = Mandiant
        description = "tofudrv malware family"
        created = "11/29/2023"
        modified = "11/29/2023"
        version = "1.0"

    strings:
        $p00_0 = {eb??33c083f8??0f85[4]488b4c24??e8[4]eb??c74424[5]eb}
        $p00_1 =
{f3aa41b8[4]33d2488d4c24??e8[4]488b8424[4]48898424[4]48638424[4]48898424}
    condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and
        (
            ($p00_0 in (34000..45000) and $p00_1 in (28000..39000))
        )
}

import "pe"
rule M_TOFUDRV_Strings_1 {
    meta:
        author = Mandiant
        hash = "b4b1e285b9f666ae7304a456da01545e"
        desc = "Detects cleartext strings that appear in the TOFUDRV image"
    strings:
        $a1 = "\\systemroot\\system32\\drivers" ascii fullword
        $a2 = "\\SafeBoot\\Minimal\\" ascii fullword
        $a3 = "\\REGISTRY\\MACHINE\\SYSTEM\\CurrentControlSet\\Control"
ascii fullword
        $a4 = "\\SafeBoot\\Network\\" ascii fullword
        $a5 =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
ascii fullword
        $a6 = "Found" ascii fullword
    condition:
        uint16(0) == 0x5A4D and
        filesize < 500KB and
        pe.subsystem == pe.SUBSYSTEM_NATIVE and
        (
            3 of them or
            pe.imphash() == "ff6f16b00c9f36b32cd60fecd4dfc8e9"
        )
}
```

```
import "pe"
rule M_TOFUDRV_RtlSubtreeStackStrings_1 {
    meta:
        author = Mandiant
        hash = "b4b1e285b9f666ae7304a456da01545e"
        desc = "Detects a stack string byte pattern in a function intended
to resolve the memory image base of ntoskrnl.exe in TOFUDRV"
    strings:
        // "RtlSubtreePredecessor"
        $a1 = { C6 44 24 ?? 52 C6 44 24 ?? 74 C6 44 24 ?? 6C C6 44 24 ??
53 C6 44 24 ?? 75 C6 44 24 ?? 62 C6 44 24 ?? 74 C6 44 24 ?? 72 C6 44
24 ?? 65 C6 44 24 ?? 65 C6 44 24 ?? 50 C6 44 24 ?? 72 C6 44 24 ?? 65
C6 44 24 ?? 64 C6 44 24 ?? 65 }
        // "RtlSubtreeSuccessor"
        $a2 = { C6 84 24 ?? 00 00 00 6C C6 84 24 ?? 00 00 00 53 C6 84 24
?? 00 00 00 75 C6 84 24 ?? 00 00 00 62 C6 84 24 ?? 00 00 00 74 C6 84
24 ?? 00 00 00 72 C6 84 24 ?? 00 00 00 65 C6 84 24 ?? 00 00 00 65 C6
84 24 ?? 00 00 00 53 C6 84 24 ?? 00 00 00 75 }
        $KeGetPcr = { 65 48 8B 04 25 18 00 00 00 48 89 44 24 }
    condition:
        uint16(0) == 0x5A4D and
        filesize < 500KB and
        pe.subsystem == pe.SUBSYSTEM_NATIVE and
        $KeGetPcr and
        any of ($a*)
}

rule M_Dropper_MSIL_TEMPLESHOT_1 {
    meta:
        author = Mandiant
        date_created = "2020-05-22"
        date_modified = "2020-05-22"
        md5 = "6d3041b89484c273376e5189e190d235"
        rev = 2
    strings:
        $ss1 = "--install" fullword wide
        $ss2 = "' directory created." fullword wide
        $ss3 = "' file created." fullword wide
        $ss4 = "' service created." fullword wide
        $ss5 = "Nothing changed :D" fullword wide
        $ss6 = "\x00ProtectDriver\x00"
        $ss7 = "\x00WriteAllBytes\x00"
        $ss8 = "\x00CopyTime\x00"
        $ss9 = "T\x00V\x00q\x00Q\x00"
    condition:
        (
            uint16(0) == 0x5A4D and
            uint32(uint32(0x3C)) == 0x00004550
        ) and
        all of them
}
```

```
rule M_Backdoor_MSIL_TEMPLESHOT_2 {
    meta:
        author = Mandiant
        date_created = "2020-05-22"
        date_modified = "2020-05-22"
        md5 = "a991bdbf1e36d7818d7a340a35a4ea26"
        rev = 2
    strings:
        $sb1 = { 02 7B [2] 00 04 [0-8] FE 03 [0-8] 39 [4-8] 02 7B [2] 00 04
[5] 0? 02 7B [2] 00 04 [5-12] 0C }
        $sb2 = { 7B [2] 00 04 [0-16] 13 ?? 11 [1-8] 17 59 45 04 00 00 00 02
[4-64] 2B ?? 02 [1-2] 7B [2] 00 04 73 [2] 00 06 28 [2] 00 06 0A 2B ?? 02
[1-2] 7B [2] 00 04 73 [2] 00 06 28 [2] 00 06 [0-4] 0A 2B }
        $ss1 = "\x00set_UseShellExecute\x00"
        $ss2 = "\x00HttpListenerRequest\x00"
        $ss3 = "\x00HttpListenerResponse\x00"
        $ss4 = "\x00HttpListener\x00"
    condition:
        (
            uint16(0) == 0x5A4D and
            uint32(uint32(0x3C)) == 0x00004550
        ) and
        all of them
}

rule M_Backdoor_MSIL_TEMPLESHOT_1 {
    meta:
        author = Mandiant
        date_created = "2020-05-22"
        date_modified = "2020-05-22"
        md5 = "952482949f495fb66e493e441229ae4b"
        rev = 2
    strings:
        $sb1 = { 06 17 7D [4] 06 20 36 01 00 C0 7D [4] DE 00 07
15 3B [4] 07 28 [4-12] 0D [8-64] 11 06 [4-12] 13 07 11 07 39 [4-32]
20 FF FF 1F 00 12 09 [0-12] 11 09 12 0A [4-12] 12 0A 11 07 }
        $ss1 = "\x00GetProcessById\x00"
        $ss2 = "\x00NtOpenThread\x00"
        $ss3 = "\x00NtQueryInformationThread\x00"
        $ss4 = "\x00ReadProcessMemory\x00"
        $ss5 = "\x00NtTerminateProcess\x00"
        $ss6 = "\x00set_UseShellExecute\x00"
        $ss7 = "\x00DESCryptoServiceProvider\x00"
        $ss8 = "\x00GetExecutingAssembly\x00"
    condition:
        (
            uint16(0) == 0x5A4D and
            uint32(uint32(0x3C)) == 0x00004550
        ) and
        all of them
}
```

Posted in

Threat Intelligence