Basic Static Analysis Lab Solution and Guide
shadyrabbit.exe

SHORT ANSWERS

Is the sample packed? How can you tell?

The sample is packed. See DETAILED ANALYSIS section for details.

Is there anything interesting or unique about the structure of this PE?

The unpacked binary has a modified PE in the resource section, `XIN`.

List any potential host-based indicators of this malware.

From strings, "`Global\SiShen %d`" could be a possible Mutex or Event. Strings such as "`%SystemRoot%\System32\svchost.exe -k netsvcs`" and references to `system\currentcontrolset\services` suggest the malware may install a service.

List any potential network-based indicators of this malware.

"`Mozilla/4.0 (compatible)`" is likely an HTTP User-Agent

Repeat your static analysis on the embedded binary – what indicators can you extract from this PE?

The dropped DLL seems to have some interesting strings that could be imports such as `ServiceMain`, `solo`, and `soloInstall`.

What might this program (shadyrabbit) do?

The program appears to be a dropper which contains an embedded payload. The payload may be installed as a service. The payload may be a backdoor which communicates with a Command and Control (C2) server via HTTP. See DETAILED ANALYSIS for more details.

DETAILED ANALYSIS

Is the sample packed? How can you tell?

Open the file in "*CFF Explorer*". Observe "*File Info*" which indicates *UPX*.

shadyrabbit.exe

Property	Value
File Name	C:\Users\user\Desktop\Labs\01_Basic Static and Dynamic\shadyrabbit.exe
File Type	Portable Executable 32
File Info	UPX 2.90 [LZMA] (Delphi stub) -> Markus Oberhumer, Laszlo Molnar & John Reise

Figure 1: "CFF Explorer" "File Info" indicates UPX

For additional verification, observe section names which also indicate *UPX*. Observe the `UPX0` section has "*Raw Size*" `0` and "*Virtual Size*" `0x11000` which suggests data can be decompressed and/or decoded at runtime.

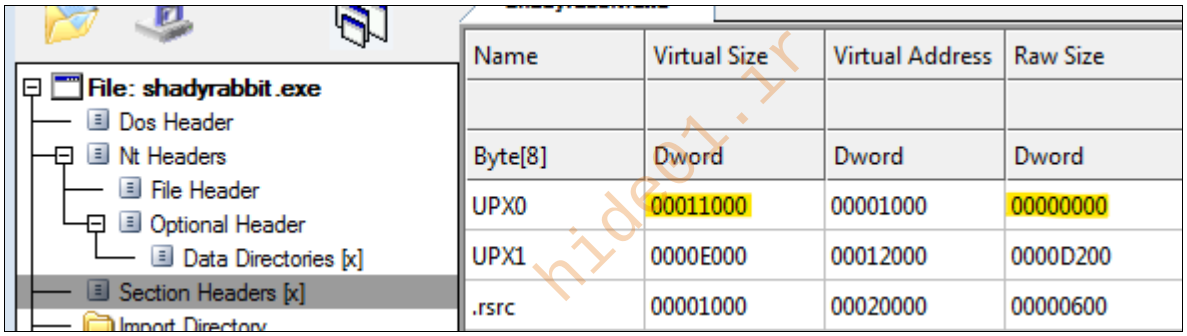

Figure 2: Section Headers – section names consistent with UPX

For even further verification, observe the "*Import Directory*". There are not many imports, and there is only a single import for six of the seven imported DLLs. This is a common packing technique used to load the libraries without exposing the actual functions. The functions imported within `kernel32.dll` are related to dynamic run-time linking and memory allocation which further indicates packing.

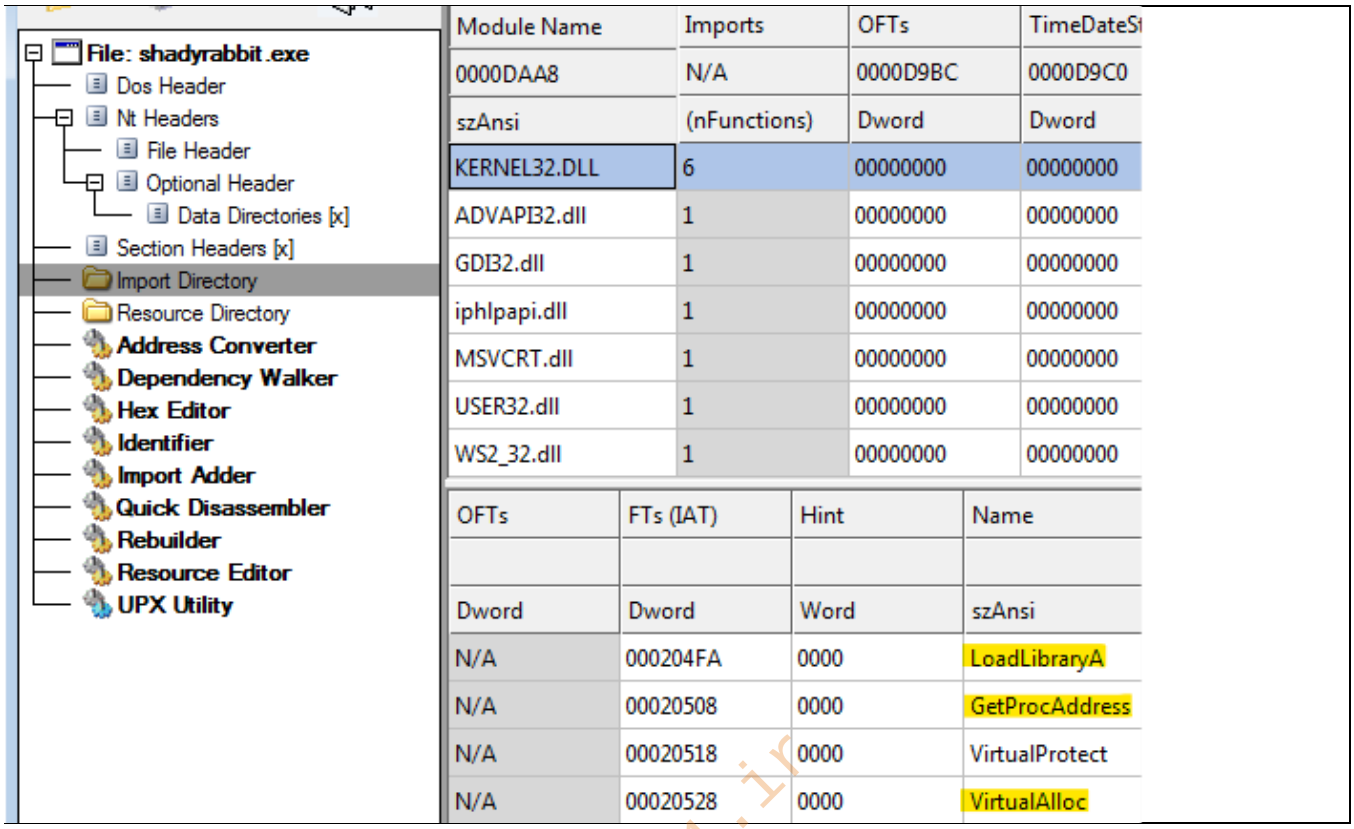

Figure 3: Imports indicate packing

There are more tools to detect packing, such as *PEiD* and *DIE*, but *UPX* packing is evident. Unpacking can be performed within "*CFF Explorer*" or by using the command line *UPX* utility. In "*CFF Explorer*" navigate to "*UPX Utility*", select *Unpack*, and save the file. Alternatively, use the command "`upx -d input_filename -o output_filename`".

```
C:\Users\user\Desktop\Labs\01_Basic Static and Dynamic>upx -d shadyrabbit.exe -o shayrabbit_unpacked.exe_
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96w        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    118961 <-     56497   47.49%    win32/pe     shayrabbit_unpacked.exe_

Unpacked 1 file.
```

Figure 4: Unpacking with UPX via command line

Is there anything interesting or unique about the structure of this PE?

Observe the "*Section Headers*" in "*CFF Explorer*". The `.rsrc` (resource) section is disproportionally large (`0x19000`). The total unpacked size is `118784` bytes (`0x1D000`) – so by dividing the `.rsrc` size by the total (`0x1D000/0x19000`), it is confirmed that the `.rsrc` section comprises 86% of the entire unpacked binary!

Name	Virtual Size	Virtual Address	Raw Size
Byte[8]	Dword	Dword	Dword
.text	00000DFA	00001000	00001000
.rdata	00000802	00002000	00001000
.data	000006E8	00003000	00001000
.rsrc	000181DC	00004000	00019000

Figure 5: Section headers - .rsrc section is disproportionally large

Navigate to "*Resource Editor*", expand the `XIN` directory, and observe the resource `101`. It looks like a PE file but it is missing the signature `MZ` at the beginning of the file. Extract the file by right clicking on the resource and selecting "*Save Resource (Raw)*".

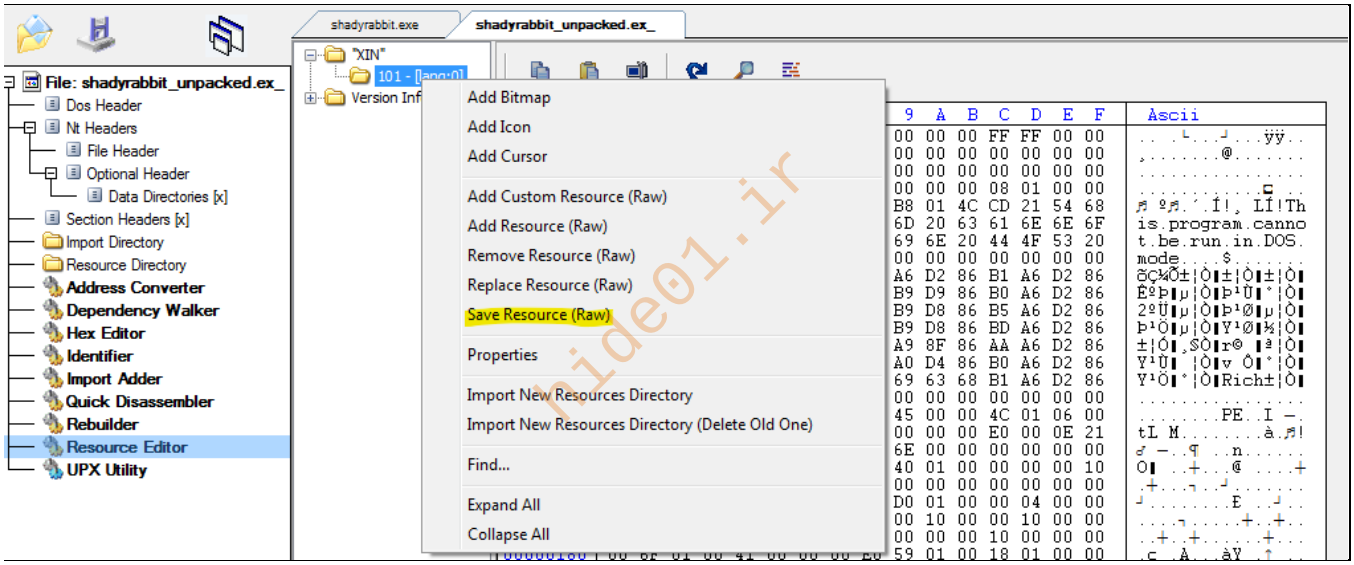

Figure 6: "Resource Editor" - save resource binary to disk

Repair the resource by adding the missing signature. Open the file in "*010 Editor*" and type `MZ` in the ASCII section, or "`4D 5A`" in the hex area, then save the file.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
0010h: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
0020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030h: 00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00  ................
0040h: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..°..´.Í!¸.LÍ!Th
0050h: 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
```

Figure 7: "010 Editor" - Repair missing MZ signature

Open the repaired file in "*CFF Explorer*". If the file is a valid PE, "*CFF Explorer*" should be able to parse it properly. Examine the "*File Type*", "*File Info*" and "*PE Size*" fields to confirm.

shadyrabbit_res.exe_

Property	Value
File Name	C:\Users\user\Desktop\Labs\01_Basic Static and Dynamic\shadyrabbi...
File Type	Unknown format
File Info	Unknown format
File Size	95.53 KB (97827 bytes)
PE Size	Not a Portable Executable.

shadyrabbit_res_fixed.exe_

Property	Value
File Name	C:\Users\user\Desktop\Labs\01_Basic Static and Dynamic\shadyrabbi...
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 6.0 DLL
File Size	95.53 KB (97827 bytes)
PE Size	95.50 KB (97792 bytes)

Figure 8: "CFF Explorer" – Top image is invalid PE, bottom image is repaired/valid PE

List any potential host-based indicators of this malware

Run *FLOSS* on the unpacked shadyrabbit binary ("`floss` *`input_filename`* `>` *`output_filename`*"). The output includes the strings from the dropper and from the embedded payload. Some abridged samples of strings analysis:

PE file format artifacts. These are usually common and not useful (`.Solo` is potentially unique)

```
!This program cannot be run in DOS mode.
Rich
.text
`.Solo
`.rdata
@.data
.rsrc
@.reloc
```

Incidental byte sequences misinterpreted as strings

```
^][Y
.PQV
^][Y
_^][d
$SUVW
```

```
_^]2
_^]2
```

Imports. When they appear together towards the beginning of the file, they are likely part of the import table and can be analyzed with "*CFF Explorer*" and ignored here.

```
CreateEventA
CloseHandle
TerminateThread
GetProcAddress
LoadLibraryA
WaitForSingleObject
SetEvent
```

C++ runtime artifacts. These are common.

```
??2@YAPAXI@Z
??3@YAXPAX@Z
__CxxFrameHandler
memmove
ceil
_ftol
```

Some relevant strings. This list is not exhaustive; there are many potentially relevant strings. Searching online can be helpful if a string appears interesting and more context is required.


```
solo
WinSta0\Default
%s\SHELL\OPEN\COMMAND
system\cURRENTcONTROLsET\sERVICES\%s
soloInstall
DeleteFileA
epyT
system\cURRENTcONTROLsET\sERVICES\
Game Over!Good luck!
Http/1.1 403 Forbidden
<body><h1>403 Forbidden</h1></body>
HTTP/1.0 200 OK
Eternal Update
aPPLICATIONS\IEXPLORE.EXE\SHELL\OPEN\COMMAND
System
ytiruceS
noitacilppA
Host
Hotkey
.DEFAULT\Keyboard Layout\Toggle
SYSTEM\CurrentControlSet\Control\Terminal Server\Wds\rdpwd\Tds\tcp
SYSTEM\CurrentControlSet\Control\Terminal Server\WinStations\RDP-Tcp
PortNumber
SYSTEM\CurrentControlSet\Control\Terminal Server\RDPTcp
fDenyTSConnections
SYSTEM\CurrentControlSet\Services\TermService
Start
SYSTEM\CurrentControlSet\Services\TermDD
```

```
TSEnabled
SYSTEM\CurrentControlSet\Control\Terminal Server
ShutdownWithoutLogon
SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
EnableAdminTSRemote
Installer
SOFTWARE\Policies\Microsoft\Windows
Enabled
netcache
SOFTWARE\Microsoft\Windows\CurrentVersion
\\.\PHYSICALDRIVE0
CONNECT
POST
HEAD
GET
http://
%s:\Documents and Settings\Local Server
winlogon.exe
taskkill /f /t /im LiveUpdate36O.exe
mouse_event
\CMD.EXE
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
roup1
Mozilla/4.0 (compatible)
https://
~MHz
HARDWARE\DESCRIPTION\System\CentralProcessor\0
tfg4h98dfh
Global\SiShen %d
winsta0
CVideoCap
```


Potential host-based indicators include registry and file paths (highlighted in yellow) and a possible mutex or event (highlighted in pink).

List any potential network-based indicators of this malware

Strings highlighted in green suggest network capability - specifically the HTTP protocol. "`Mozilla/4.0 (compatible)`" may represent an HTTP User-Agent.

Repeat your static analysis on the embedded binary – what indicators can you extract from this PE?

Run *FLOSS* on the extracted payload. The output includes the strings previously analyzed. Remember that the embedded resource/payload was large compared to the rest of the file, so it can be deduced that the strings analysis performed so far is mostly related to the payload, and the indicators from the previous questions are actually part of the embedded binary.

What might this program (shadyrabbit) do?

Examine the payload in "*CFF Explorer*". The "*File Info*" field indicates the file is a DLL.

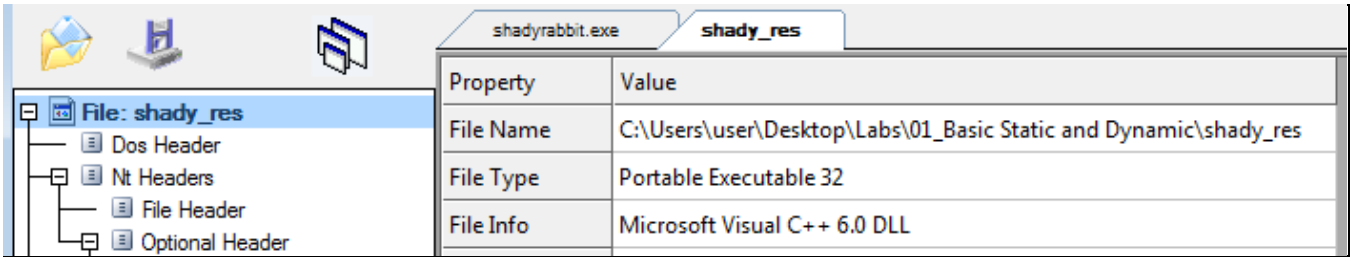

Figure 9: "File Info" indicates DLL

This can be confirmed by navigating to "*File Header*" – *Characteristics* and reviewing the flag "*File is a DLL*"

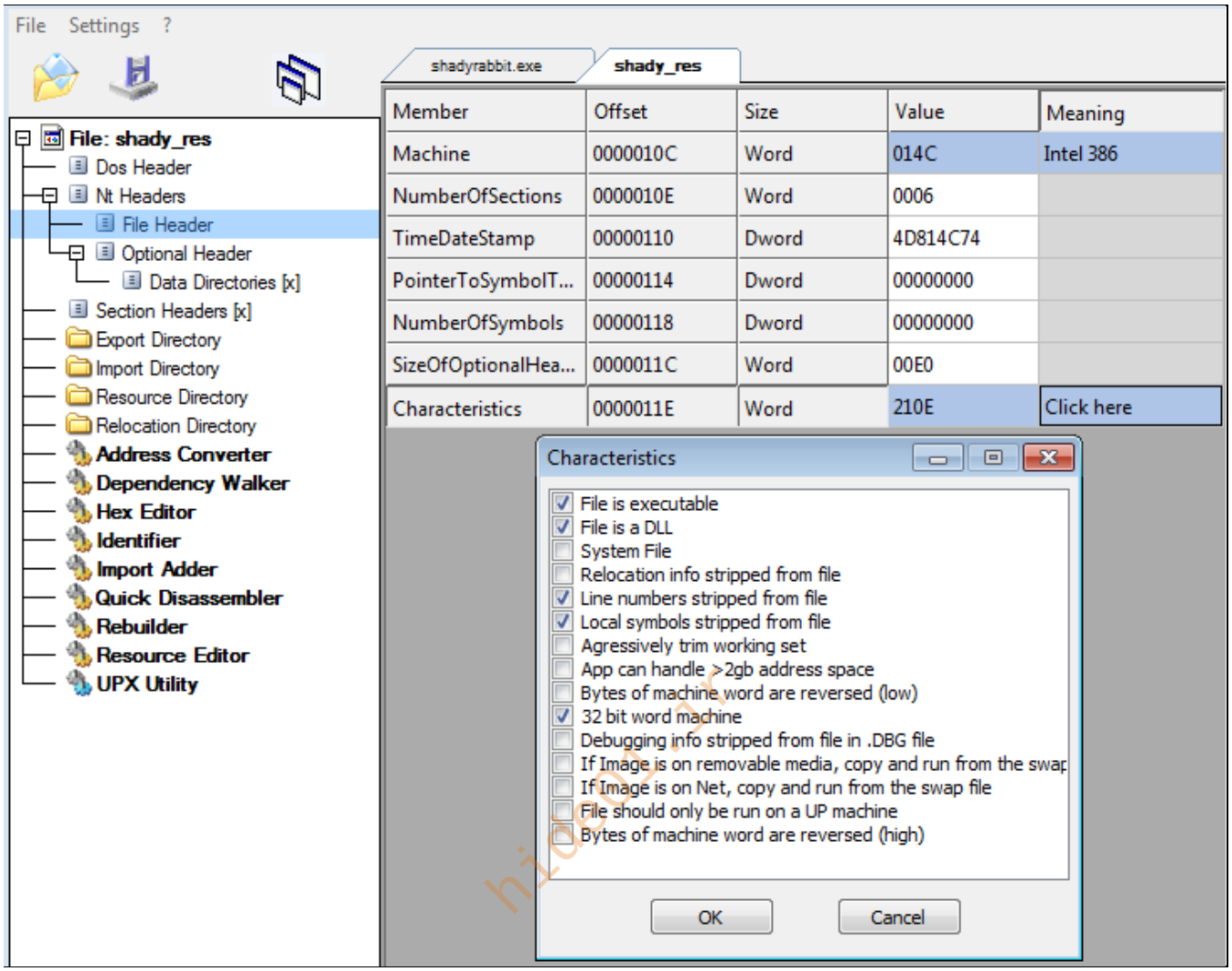

Figure 10: "File Header" - Characteristics flag indicates DLL

Now examine the "*Export Table*" – there is a single export named `solo`.

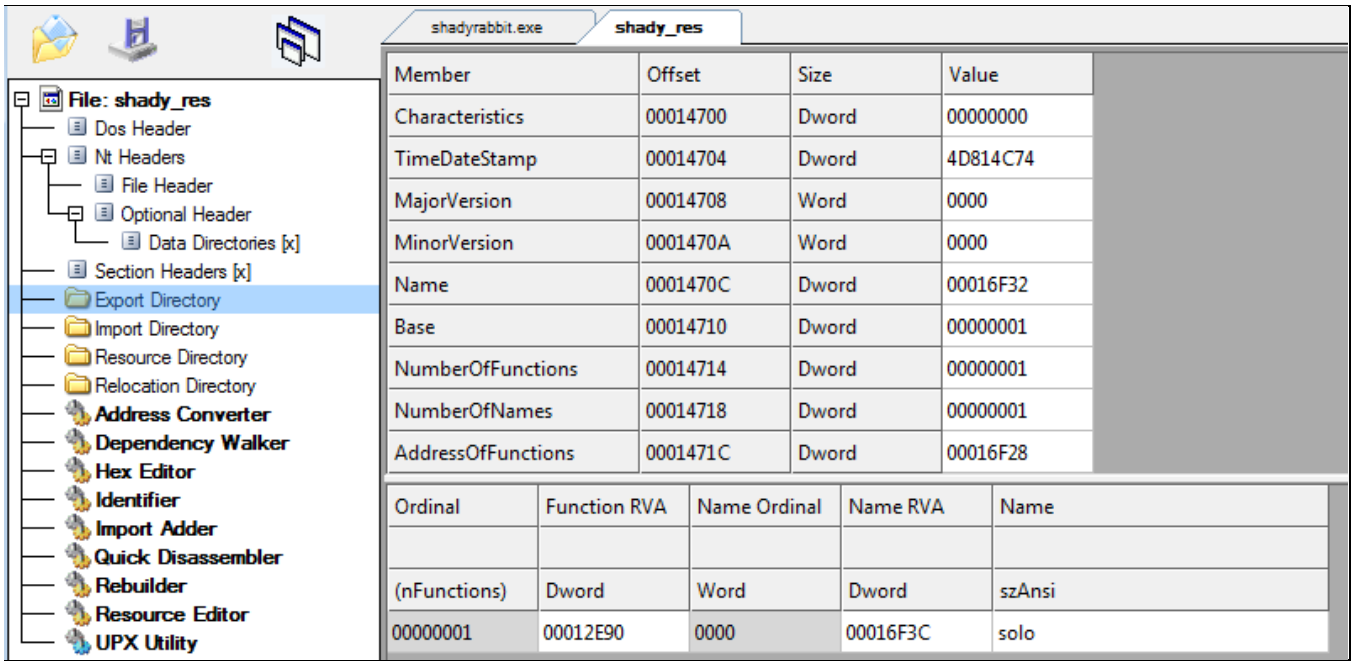

Figure 11: "Export Directory" - solo export

Run *capa* on the payload (`capa <filename>`).

CAPABILITY	NAMESPACE
check for time delay via GetTickCount (3 matches)	anti-analysis/anti-debugging/debugger-detection
contain obfuscated stackstrings	anti-analysis/obfuscation/string/stackstring
log keystrokes	collection/keylog
receive data	communication
send data (3 matches)	communication
create pipe	communication/named-pipe/create
create two anonymous pipes	communication/named-pipe/create
read pipe	communication/named-pipe/read
get socket status (4 matches)	communication/socket
initialize Winsock library (2 matches)	communication/socket
set socket configuration (2 matches)	communication/socket
receive data on socket	communication/socket/receive
send data on socket (3 matches)	communication/socket/send
connect TCP socket	communication/socket/tcp
create TCP socket (3 matches)	communication/socket/tcp
create UDP socket (5 matches)	communication/socket/udp/send
act as TCP client	communication/tcp/client
reference Base64 string	data-manipulation/encoding/base64
encode data using XOR (3 matches)	data-manipulation/encoding/xor
contain a resource (.rsrc) section	executable/pe/section/rsrc
open clipboard (2 matches)	host-interaction/clipboard
read clipboard data	host-interaction/clipboard
replace clipboard data	host-interaction/clipboard
write clipboard data	host-interaction/clipboard
interact with driver via control codes (2 matches)	host-interaction/driver
get common file path (2 matches)	host-interaction/file-system
create directory (2 matches)	host-interaction/file-system/create
delete directory	host-interaction/file-system/delete
delete file (3 matches)	host-interaction/file-system/delete
enumerate files via kernel32 functions (3 matches)	host-interaction/file-system/files/list
get file size	host-interaction/file-system/meta
read file (3 matches)	host-interaction/file-system/read
write file (4 matches)	host-interaction/file-system/write
enumerate graphical windows	host-interaction/gui/window/find
get CPU information	host-interaction/hardware/cpu
get memory capacity	host-interaction/hardware/memory
get disk information	host-interaction/hardware/storage
access the Windows event log	host-interaction/log/winevt/access
create mutex	host-interaction/mutex
resolve DNS (10 matches)	host-interaction/network/dns/resolve
get hostname	host-interaction/os/hostname
get system information	host-interaction/os/info
get OS version (2 matches)	host-interaction/os/version
create a process with modified I/O handles and window	host-interaction/process/create
create process (2 matches)	host-interaction/process/create
allocate RWX memory	host-interaction/process/inject
enumerate processes (2 matches)	host-interaction/process/list
modify access privileges (3 matches)	host-interaction/process/modify
enumerate process modules	host-interaction/process/modules/list
terminate process (3 matches)	host-interaction/process/terminate
create registry key (3 matches)	host-interaction/registry/create
set registry value (3 matches)	host-interaction/registry/create
delete registry key (3 matches)	host-interaction/registry/delete
open registry key (9 matches)	host-interaction/registry/open
query registry entry (4 matches)	host-interaction/registry/query
query registry value (4 matches)	host-interaction/registry/query
query service status	host-interaction/service
delete service	host-interaction/service/delete
stop service	host-interaction/service/stop
create thread (5 matches)	host-interaction/thread/create
terminate thread	host-interaction/thread/terminate
overwrite Master Boot Record (MBR)	impact/wipe-disk/wipe-mbr
link function at runtime (48 matches)	linking/runtime-linking
linked against ZLIB	linking/static/zlib
parse PE header (5 matches)	load-code/pe

Figure 12: capa output reveals many capabilities

There are many different capabilities. You can run *capa* in verbose mode (`capa` *`filename`* `-vv >` *`output_filename`*) to learn more about each capability. capa displays each rule that produced each capability output. One example from verbose output which maps imports to capabilities:

```
write clipboard data
namespace  host-interaction/clipboard
author     michael.hunhoff@fireeye.com
scope      function
examples   6F99A2C8944CB02FF28C6F9CED59B161:0x403180
function @ 0x10008830
  and:
    optional:
      match: open clipboard @ 0x10008830
        and:
          api: user32.OpenClipboard @ 0x10008832
          optional:
            api: user32.CloseClipboard @ 0x10008890
    api: user32.SetClipboardData @ 0x10008882
```

Figure 13: capa verbose mode example includes rule details

In summary, `shadyrabbit.exe` appears to be a dropper that writes a DLL to disk. `shadyrabbit.exe` includes service-related imports and strings that suggest it may install the DLL as a service. The payload appears to be a fully-featured backdoor that communicates with an unknown Command and Control (C2) server via HTTP.