Basic Static Analysis Lab Solution and Guide
level32.exe

SHORT ANSWERS

What compiler/packer was used?

"*MS Visual C++ 8*"

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

The malware contains a resource named `X86` which appears to be binary data.

How can you extract the embedded binary?

The resource can be extracted using "*Resource Editor"* in "*CFF Explorer*". The binary is XOR-encoded with the key `0x80`. It can be decoded using *CyberChef*.

List any potential host-based indicators of this malware.

`level1_payload.exe`, "`C:\helloworld_\FLARELABS\branches\MACC_Training\Materials\Basic Static and Dynamic\Labs\level1\source\Level32Lab\Debug\level32.pdb`". The resource name of `X86` can be used to identify this sample.

List any potential network-based indicators of this malware.

A few of the initial NBIs are `evil.mandiant.com` as well as the unique URI `/level1.mdt`. Additionally, what appears to be the User-Agent string "`Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;Trident/5.0)`" is also unique.

What might this program do?

`level32.exe` appears to be a launcher for an embedded encoded payload. The payload may be written to disk as `level1_payload.exe`. The payload likely connects to a Command and Control (C2) server and sends information about the infected host. It is possible the malware then downloads an additional payload, but more analysis is needed to speculate further.

DETAILED ANALYSIS

What compiler/packer was used?

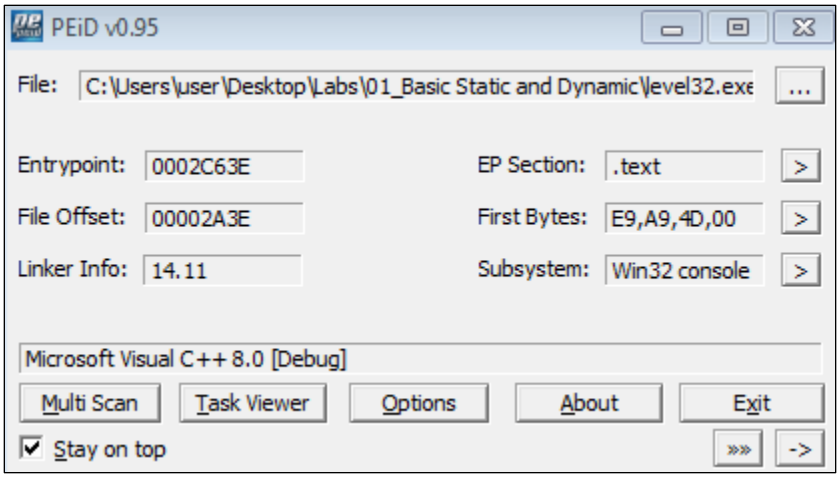
Open the file in "*CFF Explorer*". Observe "*File Info"* which indicates "*Microsoft Visual C++ 8.0*".

level32.exe

Property	Value
File Name	C:\Users\user\Desktop\Labs\01_Basic Static and Dynamic\level32.exe
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 8.0 (Debug)

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

Open *PEiD* and *DIE* to check for packing. No packing is detected. The sample appears to have been compiled using "*Visual Studio*".



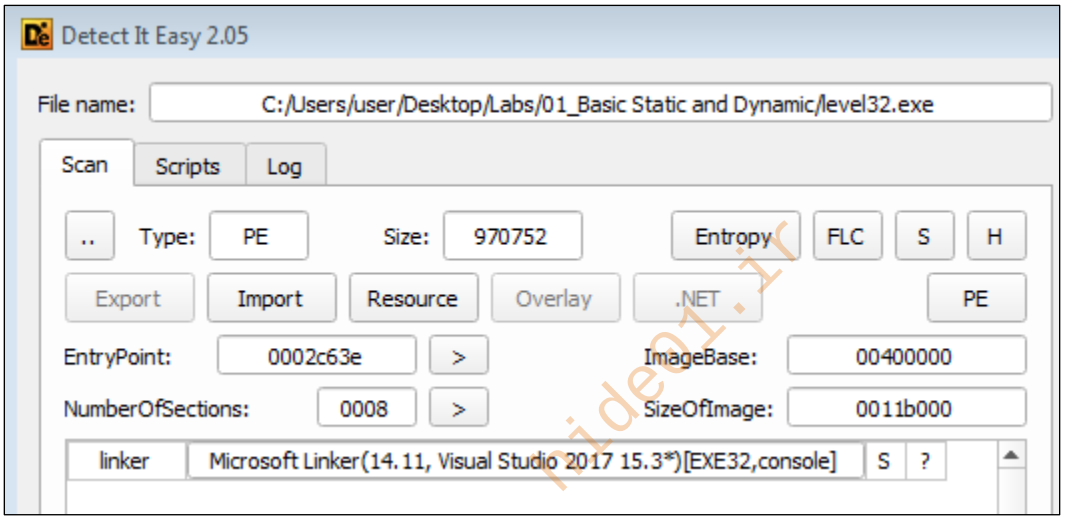


Figure 2: PEiD and DIE indicate no packing

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 (`0x81C00`). The total size is `970752` bytes (`0xED000`) – so by dividing the `.rsrc` size by the total (`0x81C00/0xED000`), it is confirmed that the `.rsrc` section comprises 55% of the entire unpacked binary.

Name	Virtual Size	Virtual Address	Raw Size
Byte[8]	Dword	Dword	Dword
.textbss	00028458	00001000	00000000
.text	0005A6D6	0002A000	0005A800
.rdata	0000B910	00085000	0000BA00
.data	000023C4	00091000	00000E00
.idata	00000CF6	00094000	00000E00
.00cfg	00000104	00095000	00000200
.rsrc	00081BD0	00096000	00081C00
.reloc	00002FED	00118000	00003000

Figure 3: Section Headers shows large .rsrc section

Navigate to "*Resource Editor*", expand the `RCData` directory, and observe the resource `X86`. At first glance it appears to be random binary data.

How can you extract the embedded binary?

There appear to be many repeating bytes `0x80`. This could be key leakage – since any value XORed with zero is itself, a single repeating byte can suggest a single-byte XOR key, in this case `0x80`. Save the resource to disk so the theory can be tested.

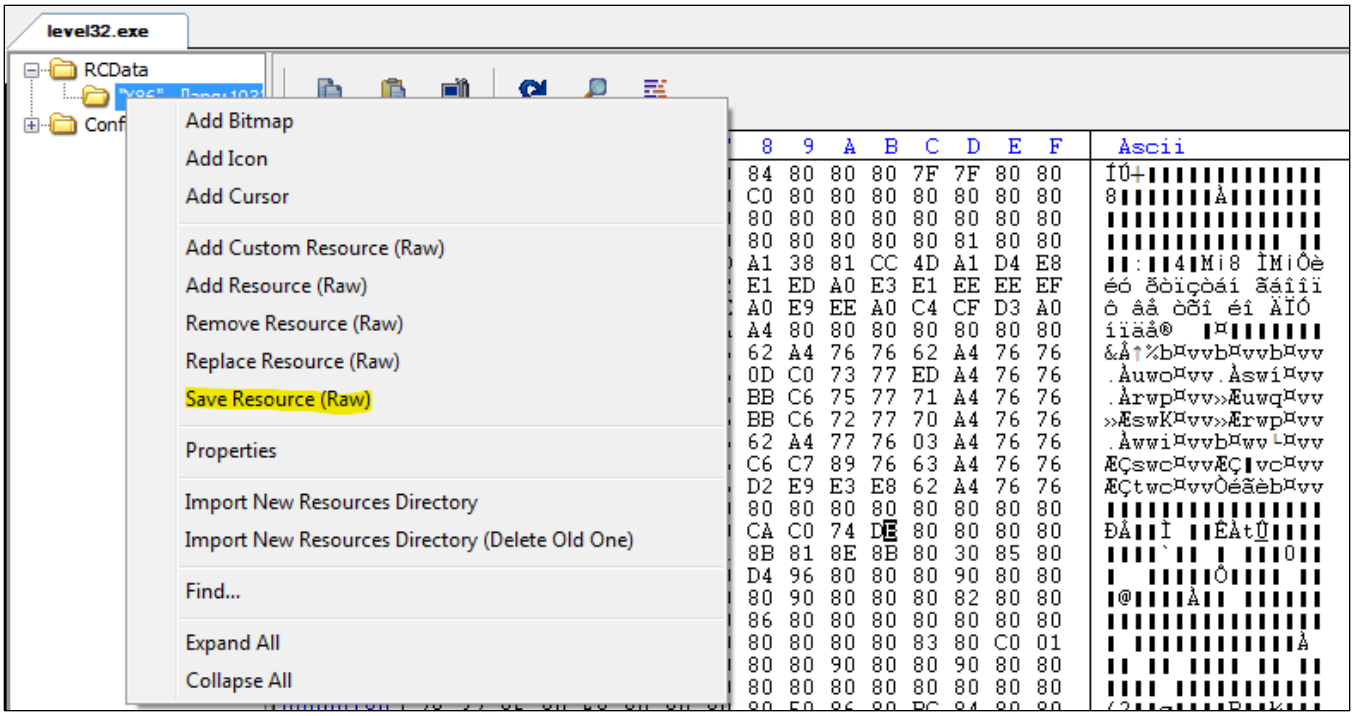


Figure 4: Use "Resource Editor" to save resource binary to disk

Use *CyberChef* to decode the file. Remember to use the local instance of *CyberChef* so the data is not shared with the public internet. Drag and drop the file into the *Input* pane within the *CyberChef* view. Select the operation *XOR* and enter the key `80`. Select *BAKE!*. Confirm the decoding is successful by observing PE artifact strings in the *Output* window.

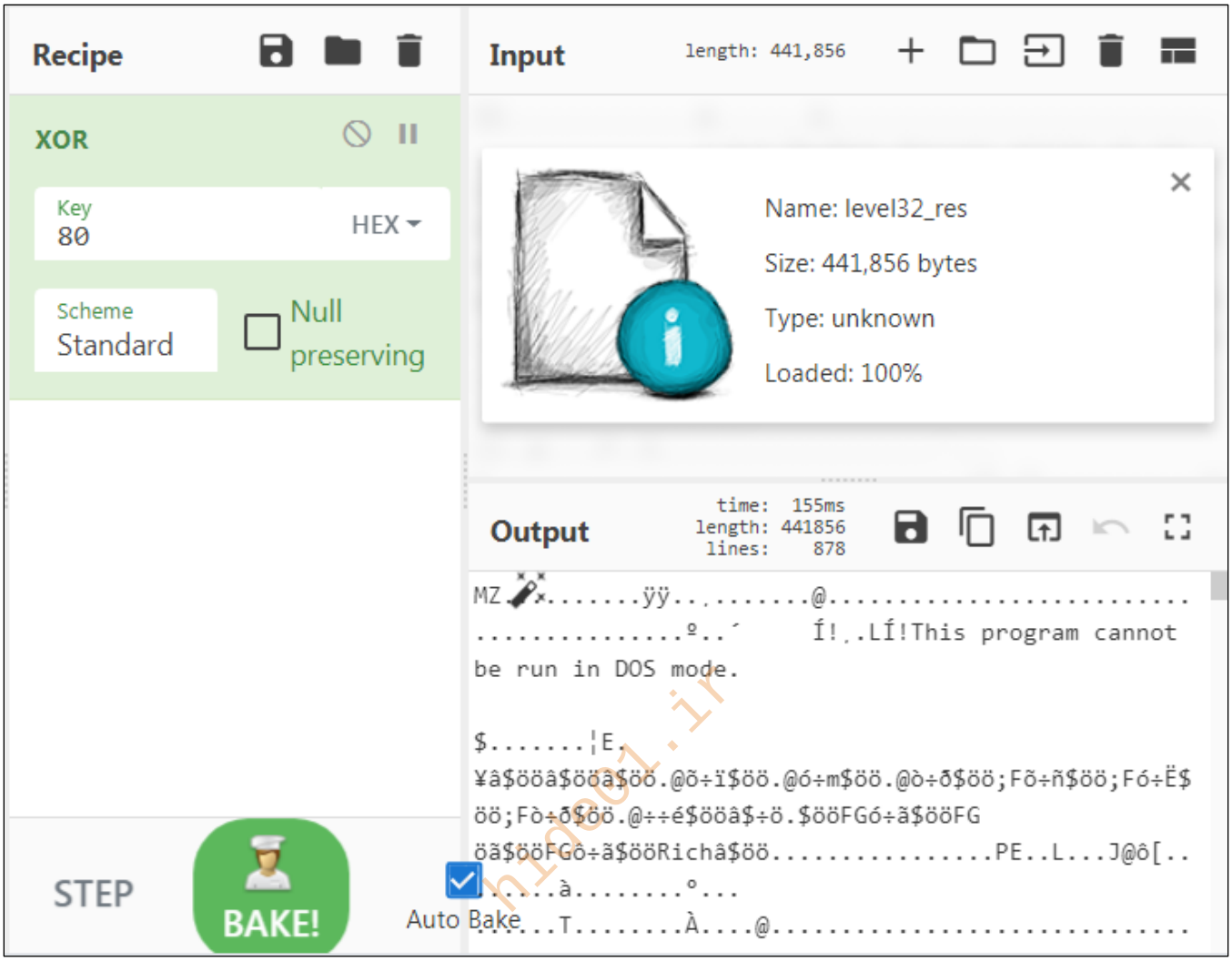


Figure 5: Using CyberChef to XOR-decode embedded binary

Open the repaired file in "*CFF Explorer*" and confirm that the file is a valid PE. If the file is a valid PE, "*CFF Explorer*" will display the headers as well as file metadata such as "*File Type"*, "*File Info"*, and "*PE Size"*. If not valid, the "*PE Size"* entry will display "`Not a Portable Executable`".

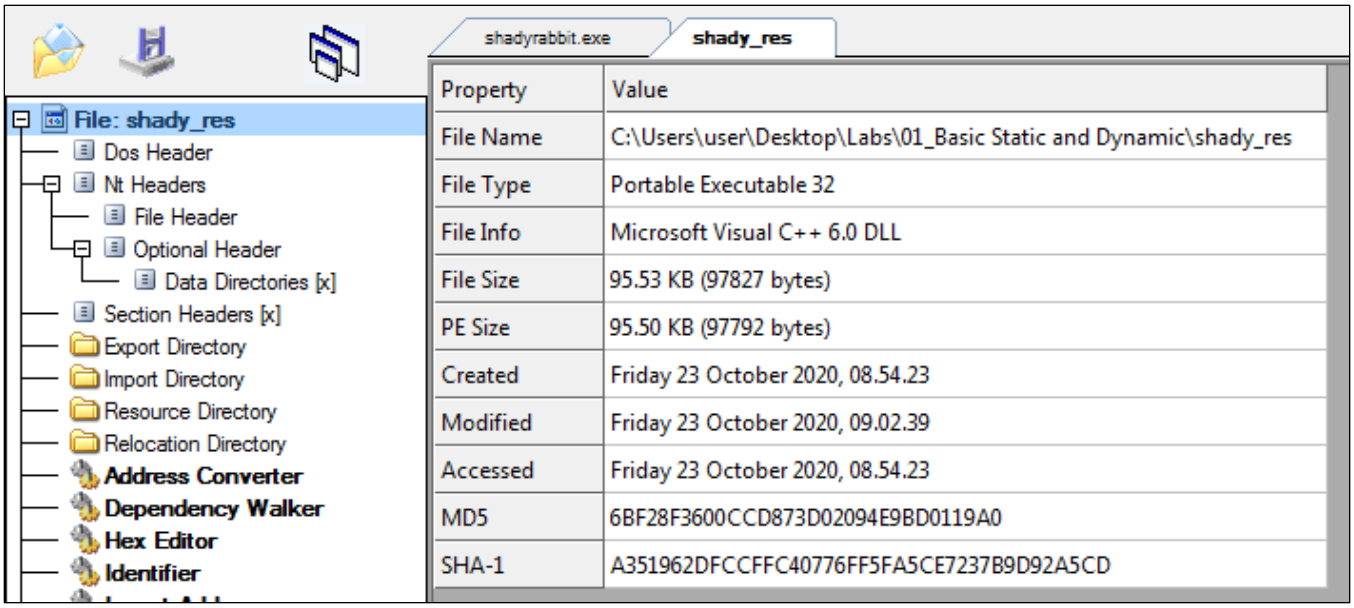


Figure 6: "CFF Explorer" indicates PE is valid ("PE Size is populated", and headers are displayed)

List any potential host-based indicators of this malware

Run *FLOSS* on `level32.exe` ("`floss` *`input_filename`* `>` *`output_filename`*"). Most of the strings are common, including PE artifacts, imports, C++ runtime, and statically linked library strings. Learning which strings are common is a practice that develops with practice. *StringSifter* can be helpful ("`floss -q` *`input_filename`* `| rank_strings >` *`output_filename`*") if the output is overwhelming. In this case, the only unique and/or relevant strings are:

```
XOR X86 failed!
explorer.exe
level1_payload.exe
C:\helloworld_\FLARELABS\branches\MACC_Training\Materials\Basic Static and
Dynamic\Labs\level1\source\Level32Lab\Debug\level32.pdb
```

Now run *FLOSS* on the extracted payload. There are many strings – the most relevant are displayed here.

```
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;Trident/5.0)
evil.mandiant.com
/level1.mdt
POST
%s %d core %llu MB
host=
net=
WinNT 3.51
WinNT 4.0
Workstation
Server Standard
Server Enterprise
Windows
2000
XP Professional x64
Home Server
```

```
Windows Server 2003 R2
Server 2003
Vista
Server 2008
Server 2008 R2
user=
Content-Type: application/x-www-form-urlencoded
```

There are three potential host-based indicators - `explorer.exe`, `level1_payload.exe, and C:\helloworld_\FLARELABS\branches\MACC_Training\Materials\Basic Static and Dynamic\Labs\level1\source\Level32Lab\Debug\level32.pdb`. There is not enough information at this point to understand how `explorer.exe` is used since it is a common Windows process. `level1_payload.exe`, however, is relatively unique. It is possible the payload, once decoded, can be written to this filename. The `.pdb` path represents a program database file that may have been created when the malware was compiled. Microsoft compilers can store debugging information in this file. We do not have the file, but the path is a unique indicator.

List any potential network-based indicators of this malware

Looking at the strings listed previously, it seems the malware may connect to a C2 server at `evil.mandiant.com` and request the file `level1.mdt` via HTTP `POST` request. It may use the HTTP User-Agent "`Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;Trident/5.0)`". Speculating further, it seems possible the malware may take a system survey, including information about the operating system version, hostname, network, and username, and send the information within the HTTP query string. The User-Agent string is particularly interesting because it includes a possible error/formatting inconsistency. The final semicolon is not followed by a space like the previous instances. This is likely a typo by the malware author which renders this User-Agent to be unique, making it a useful indicator of compromise.

What might this program do?

Run *capa* on the decoded payload.

```
C:\Users\user\Desktop\Labs\01_Basic Static and Dynamic>capa level32_res_decoded
2557 functions [00:27, 92.95 functions/s]
+-------------------------+-------------------------------------------------------------------------+
| md5                     | f252bb2daba1e5c57ca7e54f0beb10dd                                        |
| path                    | level32_res_decoded                                                     |
+-------------------------+-------------------------------------------------------------------------+

+-------------------------+-------------------------------------------------------------------------+
| ATT&CK Tactic           | ATT&CK Technique                                                        |
+-------------------------+-------------------------------------------------------------------------+
| DEFENSE EVASION         | Virtualization/Sandbox Evasion::System Checks [T1497.001]               |
| DISCOVERY               | File and Directory Discovery [T1083]                                    |
|                         | System Information Discovery [T1082]                                   |
| EXECUTION               | Shared Modules [T1129]                                                  |
+-------------------------+-------------------------------------------------------------------------+

+-------------------------------------------------+-----------------------------------------------+
| CAPABILITY                                      | NAMESPACE                                     |
+-------------------------------------------------+-----------------------------------------------+
| execute anti-VM instructions (6 matches)        | anti-analysis/anti-vm/vm-detection            |
| send data                                       | communication                                 |
| connect to HTTP server                          | communication/http/client                     |
| create HTTP request                             | communication/http/client                     |
| send HTTP request                               | communication/http/client                     |
| initialize Winsock library                      | communication/socket                          |
| contain a resource (.rsrc) section              | executable/pe/section/rsrc                    |
| accept command line arguments                   | host-interaction/cli                          |
| query environment variable (2 matches)          | host-interaction/environment-variable         |
| set environment variable (3 matches)            | host-interaction/environment-variable         |
| enumerate files via kernel32 functions          | host-interaction/file-system/files/list       |
| write file (8 matches)                          | host-interaction/file-system/write            |
| get memory capacity                             | host-interaction/hardware/memory              |
| print debug messages (3 matches)                | host-interaction/log/debug/write-event        |
| resolve DNS                                     | host-interaction/network/dns/resolve          |
| get hostname (2 matches)                        | host-interaction/os/hostname                  |
| get system information (2 matches)              | host-interaction/os/info                      |
| get OS version                                  | host-interaction/os/version                   |
| terminate process (3 matches)                   | host-interaction/process/terminate            |
| link function at runtime                        | linking/runtime-linking                       |
| parse PE header (9 matches)                     | load-code/pe                                  |
+-------------------------------------------------+-----------------------------------------------+
```


Figure 7: capa output on decoded payload

`level32.exe` appears to be a launcher for an embedded encoded payload. The payload may be written to disk as `level1_payload.exe`. The payload likely connects to a C2 server and sends information about the infected host. It is possible the malware then downloads an additional payload, but more analysis is needed to speculate further.