# EECS 499

Introduction to Malware Analysis

# UNPACKING FREECELL

project 3

NOVEMBER 2017

Neil Orans
University of Michigan | College of Engineering
norans@umich.edu

# Contents

# Basic Executable Analysis

This section contains information obtained from a variety of tools and services that helped me form a preliminary hypothesis about the nature of this binary.

**MD5:** **E95998D23233476FCC61A1FECA6D02A2**
**SHA1:** **A5847314E89A0B335F583803D29D4BF753EB9174**
**Created:** **Tuesday, November 10, 2015 4:07:28 AM**
**Size:** **82944**

**VirusTotal** Most of the engine hits seemingly came from AV engines that equate a packed binary with malware (names reported like *Gen.Packer.PESpin* and *Pascked.Win32.MNSP.Gen*).

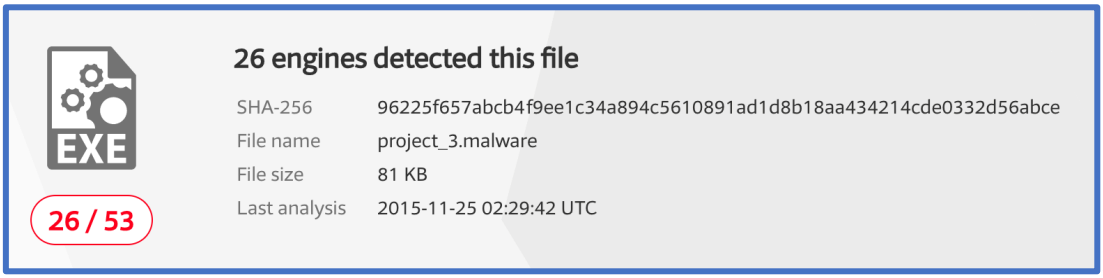


**Figure 1** VirusTotal report for the original project_3.malware binary.

**PEiD** The binary is packed with a version of PESpin ranging from 0.3 to 1.x. Not much could be found about PESpin online except for the programs website and a few manual unpacking tutorials.

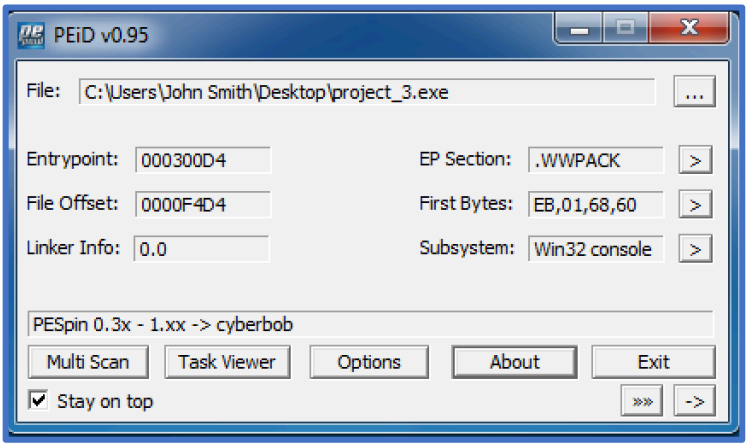


**Figure 2** PEiD scan of the original project_3.malware binary.

**ProcMon** Since the imports for this PE are not evident through static analysis, we must use dynamic analysis tools like Process Monitor to see what libraries the binary needs.

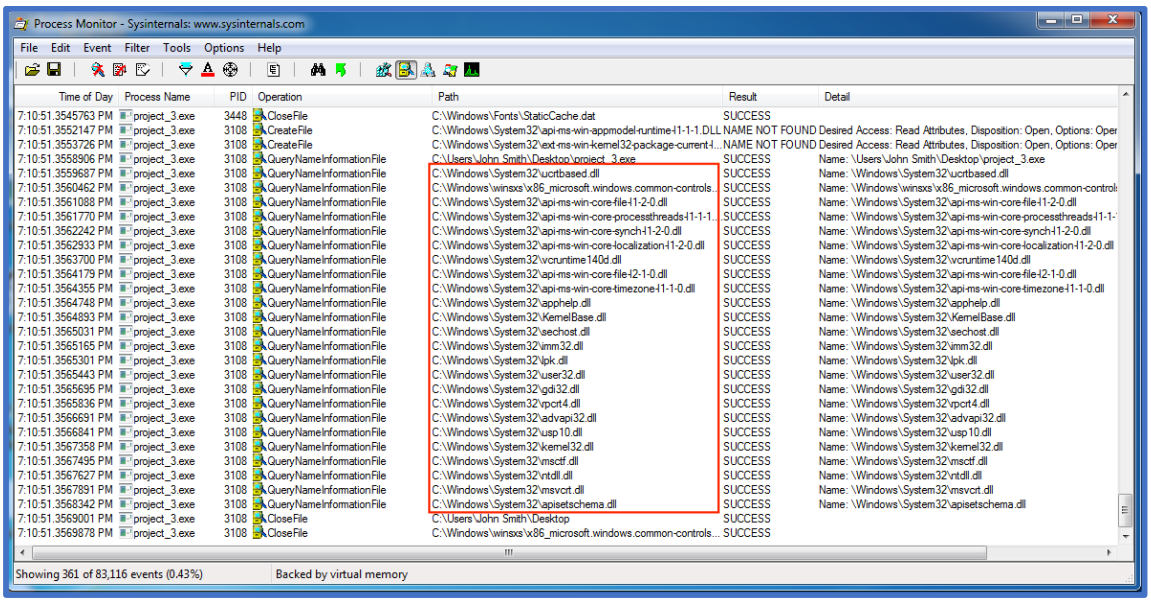


**Figure 3** Procmon showing the DLL imports by project_3.malware

When run, the executable simply shows a message box warning the user they need to insert a CD to continue.

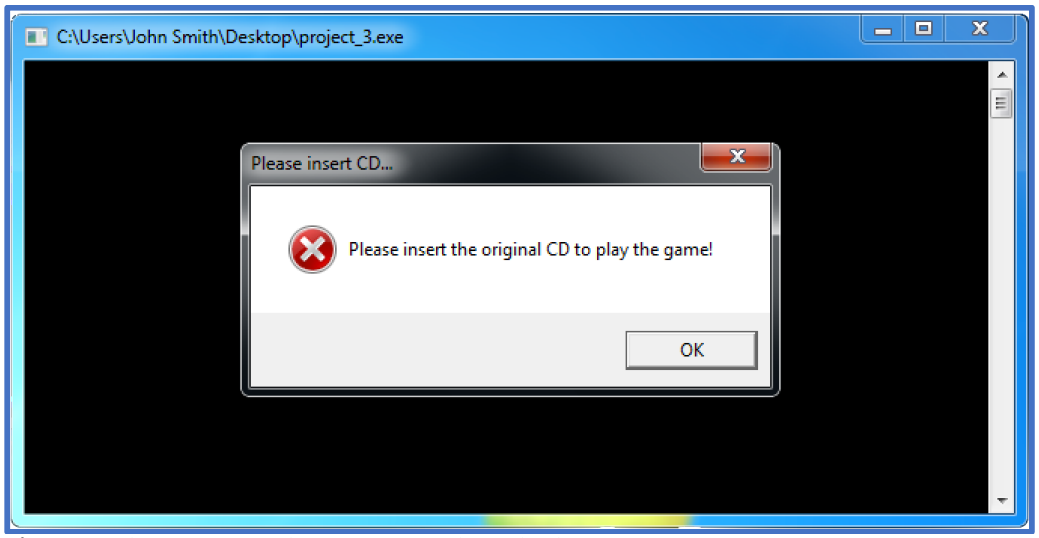


**Figure 4** The message that appears when the binary runs.

**OllyDbg** Observing the executable in OllyDbg is difficult because the program transfers control to threads using *ntdll.ZwContinue*, which switches thread contexts.


**Figure 5** A call to ntdll.ZwContinue.

Looking back at ProcMon shows us the malware is actually creating a second process.

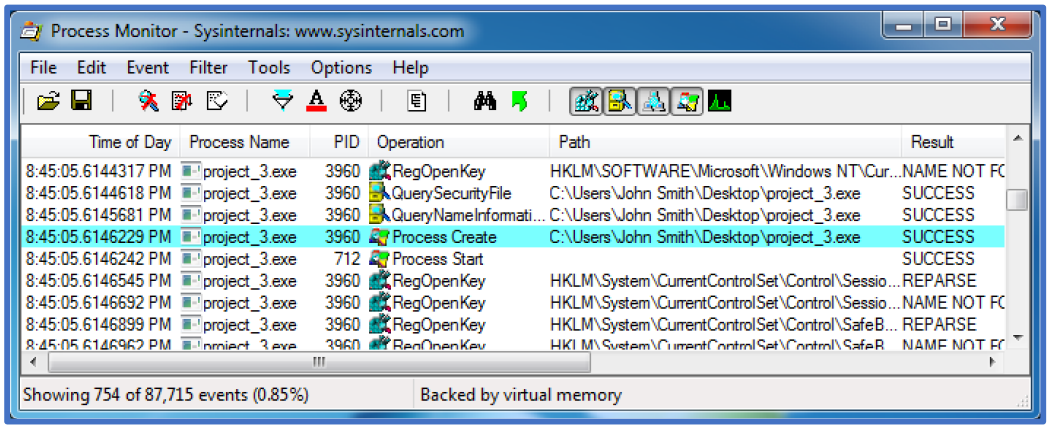


**Figure 6** The second process creation.

**ProcExp** If we observe that child process in Process Explorer we can see the strings loaded in memory, which seem to indicate the program is a game called “FreeCell”.

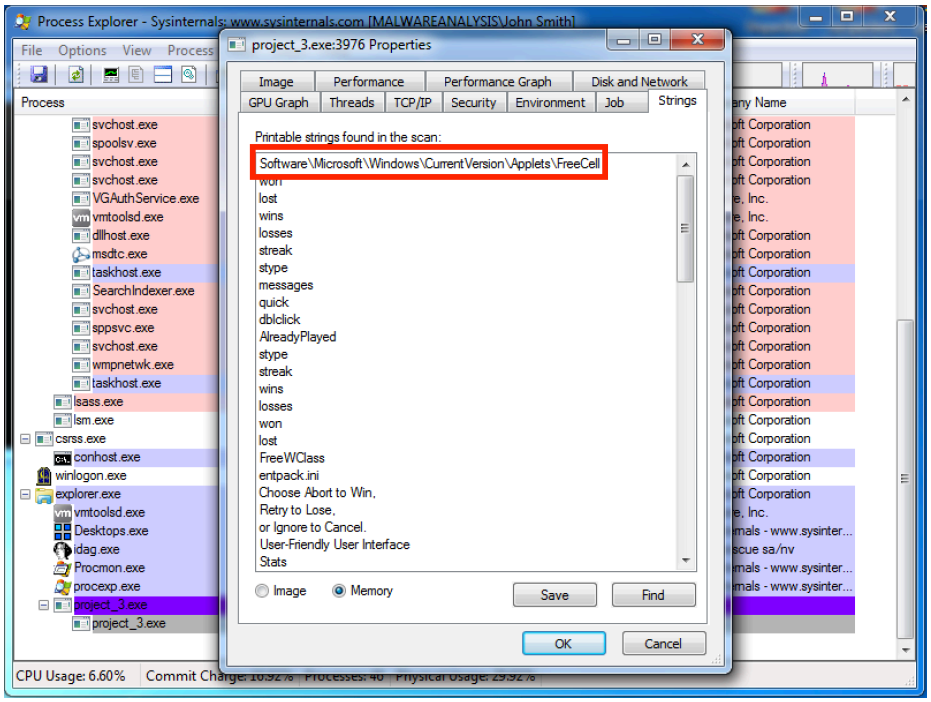


**Figure 7** The unpacked strings in memory.

**Conclusion** It seems as if this binary is a packed version of a game called “FreeCell.” The game (supposedly) cannot be played unless the user has a CD. In order to patch out this CD check, we first need to unpack the executable.

# Unpacking

## Stage 1 (PESpin)

Because this binary seems to deploy advanced anti-debug and anti-analysis techniques, I decided to use the OllyDbg plugin ScyllaHide, which has numerous anti-anti-debug features (see references section for GitHub page). Particularly useful for this binary was the RunPE dump feature.

RunPE is another packer that tends to modify malware so that the malware duplicates itself in a suspended process shortly after launch. The malware then injects code into the suspended process and resumes it. The ScyllaHide plugin hooks this process creation and dumps the new process into an executable instead of resuming it. PESpin uses a similar technique, so the ScyllaHide feature works for PESpin as well.

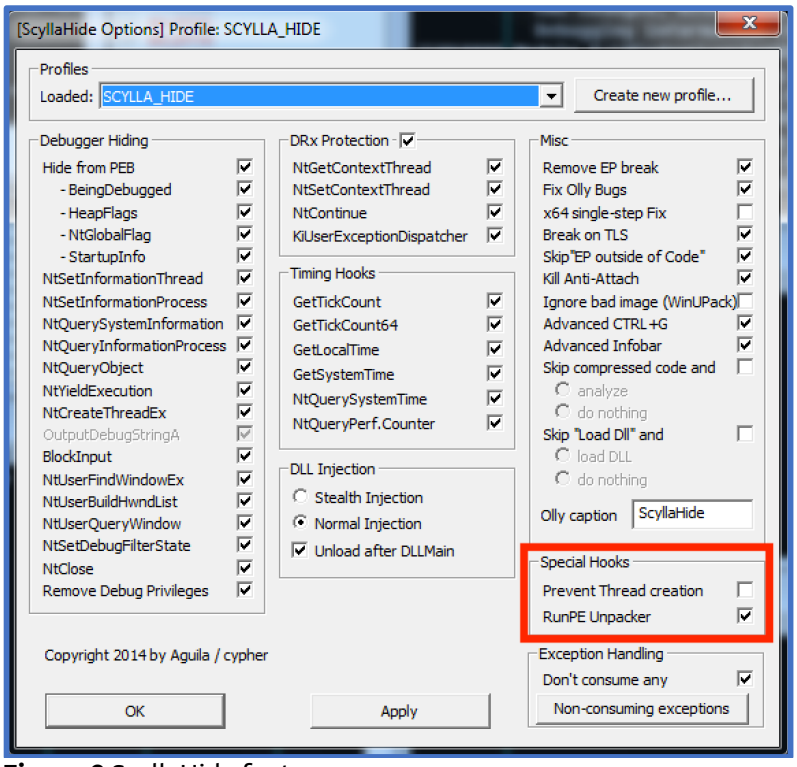


**Figure 8** ScyllaHide features.

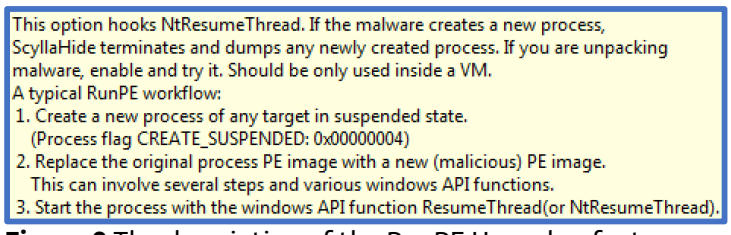


**Figure 9** The description of the RunPE Unpacker feature.

Running the executable in OllyDbg after enabling this feature creates the unpacked executable on the Desktop called “Unpacked.” This specific executable successfully loaded the icon from its resources section.


**Figure 10** FreeCell icon on Desktop.

If we load the unpacked executable into Dependency Walker, we see that the only imports are *LoadLibraryA* and *GetProcAddress*.

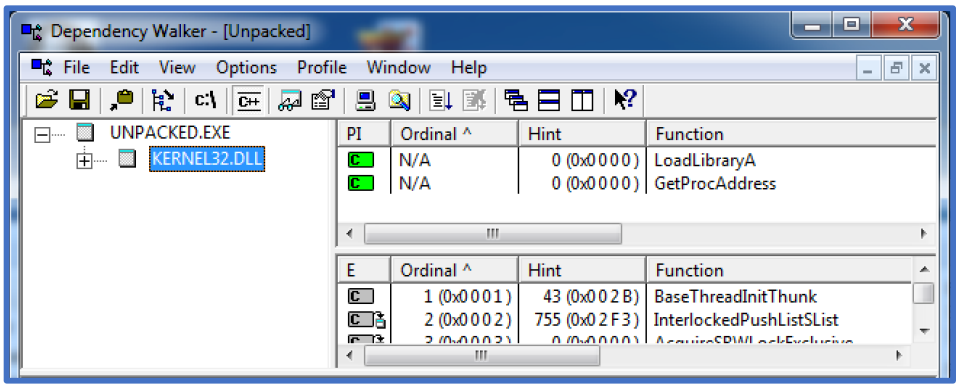


**Figure 11** Imports of executable produced by RunPE dump.

This is a strong indicator that the executable is still packed, which likely means the creator ran the binary through more than one packer. PEiD recognizes that the second packer used is YodaProt (Yoda’s Protector).

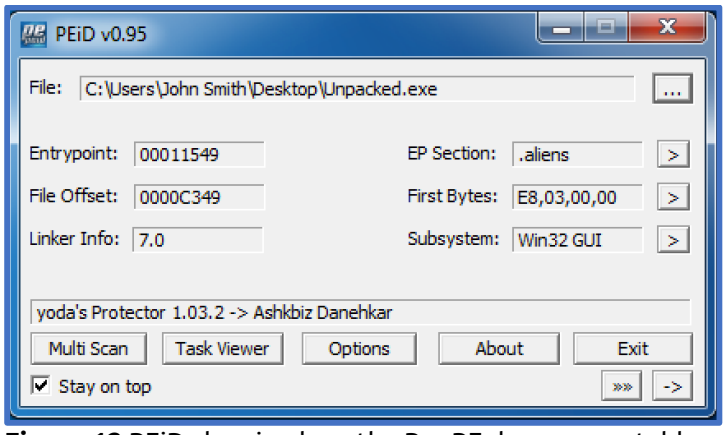


**Figure 12** PEiD showing how the RunPE dump executable still is packed.

## Stage 2 (YodaProt)

I couldn't find any automatic unpackers for YodaProt online, so manual unpacking is necessary. I ran the program in OllyDbg until the message box appeared asking for the CD. I then used OllyDbg's search feature to search memory for the string "Please insert," knowing that it had to be present in memory for the *MessageBox* function to work.

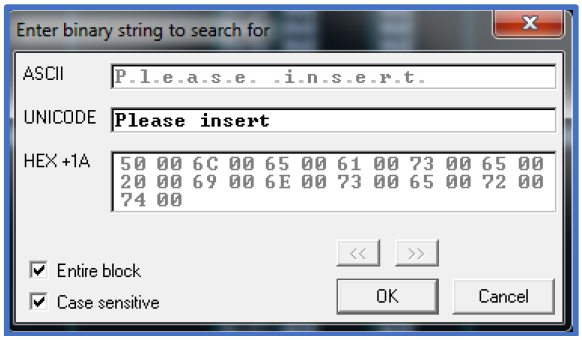


**Figure 13** Searching for the MessageBox text in OllyDbg.

Sure enough, I found the string in the .data section of the executable. Shortly below that I found the MessageBox function which contains the CD "check."

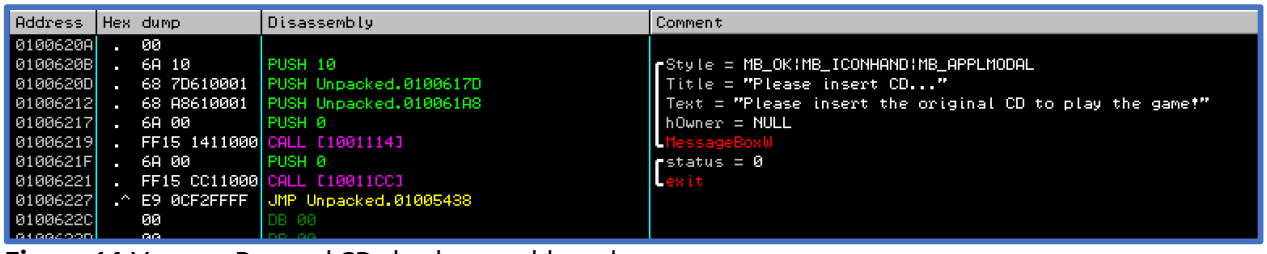


**Figure 14** MessageBox and CD check assembly code.

If we set an execution breakpoint at this line in OllyDbg, it will not register because the section is initially marked as data before the unpacking routine. However, if we mark it as a memory breakpoint, OllyDbg can stop execution at this point, and we can dump the process from memory using OllyDump.

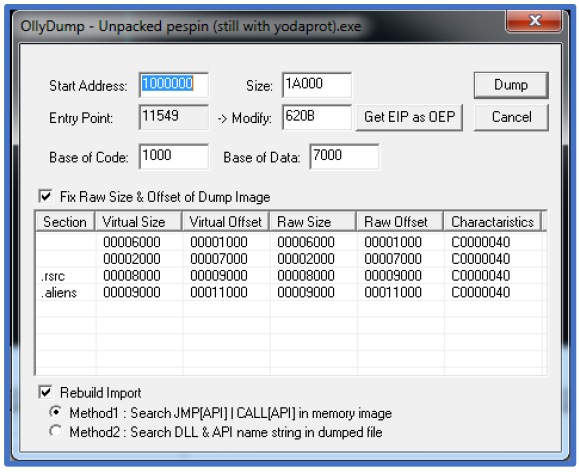


**Figure 15** OllyDump settings to dump the executable from the OEP.

We now have the fully unpacked program. We can check this by scanning it with PEiD to see if it detects a packer.

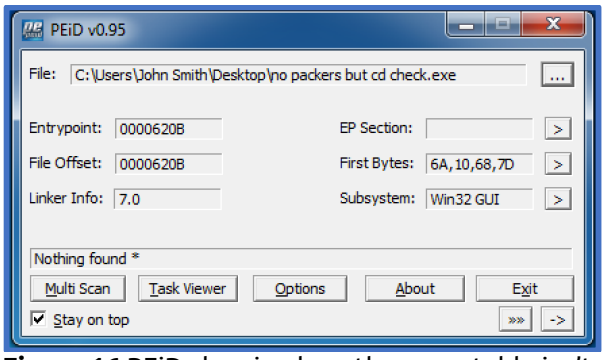


**Figure 16** PEiD showing how the executable isn't packed anymore.

The "check" isn't much of a check at all. In fact, all it does is deliver the message to the user, then immediately call *exit(0)*.

# Patching

## Patching out the CD Check

The CD “check” isn’t much of a check at all. In fact, all it does is deliver the message to the user, then immediately call *exit(0)*. If we simply NOP out the exit(0) call, the program will proceed with its normal flow and the game window will pop up.

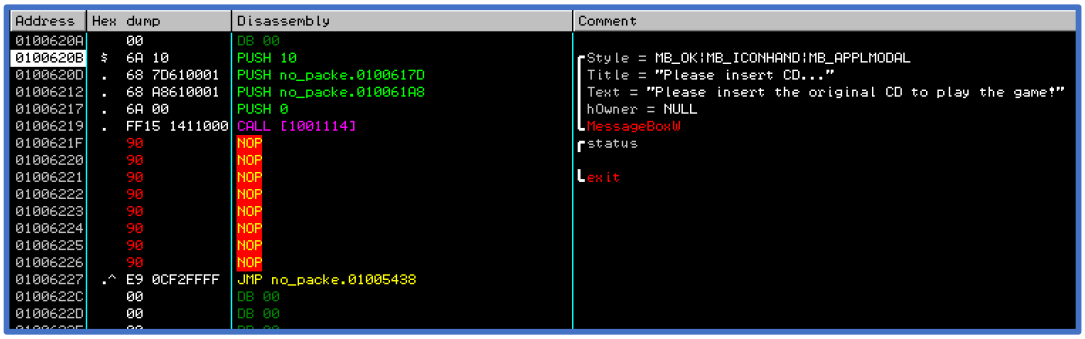


**Figure 17** The CD check exit(0) call is patched out with NOPs.

Additionally, I decided to change the arguments so that *MessageBox* function doesn’t ask the user for a CD, but rather informs them they are playing a patched version of the game.

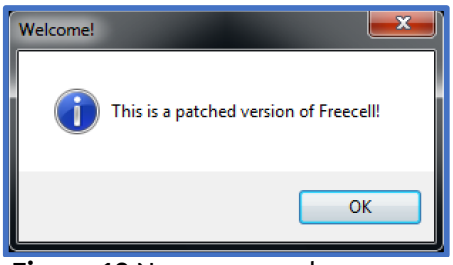


**Figure 18** New message box.

Once the user clicks “OK,” the FreeCell game pops up. According to Wikipedia, “FreeCell is a solitaire-based card game played with a 52-card standard deck.”


**Figure 19** FreeCell game.

## Adding an Easter Egg

There is an option to select a random seed for each new game of FreeCell, presumably to change the initial card layout.

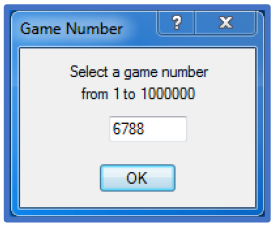


**Figure 20** Random seed selection.

I decided to add an easter egg in this section of the game. If the user enters 499 as the random seed, they will receive a special notification. (499 is the EECS department code for an undergraduate, senior, independent study).

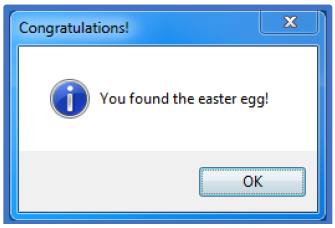


**Figure 21** Easter egg message.

In order to achieve this, I had to patch the cmp (compare) instruction that checks if the random seed value is less than 10001, which is the max random seed. I patched this instruction to jump to a section of the executable filled with zeroes. The jump destination first checks if the value entered (saved at 0x100834C) is 0x1F3 (499 in decimal). If it is, it calls the MessageBox function. If the value is anything but 499, it compares it to 10001 to replace the instruction it patched, and then returns to the instruction right below the patch to resume normal flow.

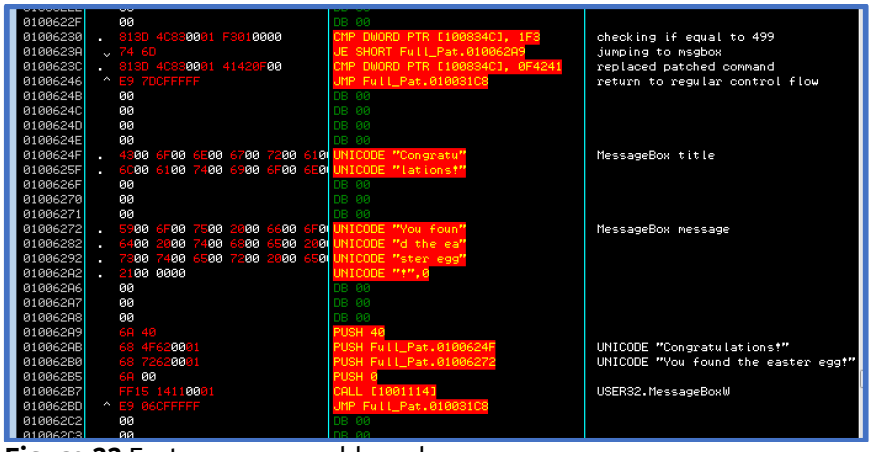


**Figure 22** Easter egg assembly code.

# Deliverables

## Automation Scripts

### *auto_oep.txt – OllyScript script*

The first script that a user can use to help automate the process of unpacking this binary is an OllyScript script. OllyScript is a "plugin meant to let you automate OllyDbg by writing scripts in an assembly-like language."

The following script ensures the user has downloaded ScyllaHide (the anti-anti-debug plugin), and then sets a breakpoint at OEP for the user. After running this script, all the user has to do is run OllyDump to produce the fully-unpacked executable.

```
var breakpoint

MSGYN "Is ScyllaHide installed?"
cmp $RESULT,1
jne no_scylla
jmp start:

start:
rtu                        // Runs till SEH takes over, which means code section is unpacked
mov breakpoint, 01001000 // Store OEP to hardware breakpoint local variable
bprm breakpoint, 6000
run
cmt eip, "~~~~~~ THIS IS OEP ~~~~~~"
msg "EIP is now at the OEP! Dump the file using OllyDump's default settings (method 1 for IAT
rebuild)"
ret

no_scylla:
msg "Please download it from https://github.com/x64dbg/ScyllaHide"
ret
```

## *patcher.py – Python 3 script*

The second script I wrote is a Python script that takes in a single command line argument, a PE filename. It then reads in this PE, finds the CD check, patches it out, and replaces the message box text informing the user it has been patched out.

```
#!/usr/bin/python

import sys

# REQUIRES:  'replacement' is new bytes
#            'bytes' is the entire byte stream (whole file)
#            'position' is location of original
def patch(replacement, bytes, position):
        first_part = bytes[:position]
        second_part = replacement + bytes[(position+len(replacement)):]
        return first_part + second_part

original =
bytearray(b'\x50\x00\x6C\x00\x65\x00\x61\x00\x73\x00\x65\x00\x20\x00\x69\x00\x6E\x00\x73\x00
\x65\x00\x72\x00\x74\x00\x20')

replacement =
bytearray(b'\x57\x00\x65\x00\x6C\x00\x63\x00\x6F\x00\x6D\x00\x65\x00\x21\x00\x00\x00\x73\x00
\x65\x00\x72\x00\x74\x00\x20\x00\x43\x00\x44\x00\x2E\x00\x2E\x00\x2E\x00\x00\x00\x00\x00\x00
\x54\x00\x68\x00\x69\x00\x73\x00\x20\x00\x69\x00\x73\x00\x20\x00\x61\x00\x20\x00\x70\x00\x61
\x00\x74\x00\x63\x00\x68\x00\x65\x00\x64\x00\x20\x00\x76\x00\x65\x00\x72\x00\x73\x00\x69\x00
\x6F\x00\x6E\x00\x20\x00\x6F\x00\x66\x00\x20\x00\x46\x00\x72\x00\x65\x00\x65\x00\x63\x00\x65
\x00\x6C\x00\x6C\x00\x21\x00\x00\x00\x68\x00\x65\x00\x20\x00\x67\x00\x61\x00\x6D\x00\x65\x00
\x21\x00\x00\x00\x00\x00\x00\x6A\x40\x68\x7D\x61\x00\x01\x68\xA8\x61\x00\x01\x6A\x00\xFF\x15
\x14\x11\x00\x01\x6A\x00\x90\x90\x90\x90\x90\x90')

bytes = 0
with open(sys.argv[1], "rb") as cd_check_exe:
        bytes = bytearray(cd_check_exe.read())
        bytes = patch(replacement, bytes, bytes.find(original))

with open("patched_cd_check.exe", "wb") as output_exe:
        output_exe.write(bytes)
```

## Instructions

The following instructions will allow someone in possession of the original, packed binary to retrieve the fully-unpacked, CD-patched binary.

1. Download and install OllyDbg, and the ScyllaHide, OllyScript, and OllyDump plugins (the latter two are probably installed by default).
2. Open the PE in OllyDbg and check “RunPE dump” in the ScyllaHide options.
3. Run the executable.
4. Reopen OllyDbg with the executable produced by ScyllaHide (should be titled “Unpacked” and placed on the Desktop).
5. Run the OllyScript auto_oep.txt.
6. Open OllyDump and dump the process from memory. Ensure method 1 is selected as the import rebuilding method.
7. Run patcher.py on the executable produced in step 6.

# References

https://github.com/nihilus/ScyllaHide
https://sourceforge.net/projects/odbgscript/files/English%20Version/
http://www.pespin.com/ (Chrome warns this site is malicious)

**Electrical Engineering and Computer Science**
**2017**