64-bit Imports Rebuilding and Unpacking

Sebastien Doucet  <sdoucet@ncircle.com>
Who am I?

- Security Research Engineer at nCircle in Toronto
- My accent is from Montreal
- Used to be involved in the online reverse-engineering community
  - Moderator at reverse-engineering.net forum
  - Co-founder of video.reverse-engineering.net (defunct)
  - Most evil moderator at crackmes.de
  - Member of ARTeam
Warning

- Used to be a 60-minutes presentation
- Had to take away all the jokes
- Only dry technical stuff left
- Focus is on the unpacking process instead of the tool
- For more information or the full deck of slides, come see me at the nCircle booth
Looking at the Bigger Picture

• More 64-bit malware is appearing every day
• Next generation of Windows will probably be 64-bit only like Windows Server 2008 R2
• Programming languages limited to a specific architecture like C and C++ increase the difficulty in dealing with cross-architecture tasks like unpacking
• Platform-independent languages like Python simplify things but still require you to know what you are doing when dealing with the PE format and headers
• If I had to redo it today, I would do it in C#
Overview

• Part 1: What is unpacking?
• Part 2: What is ImpREC? (original tool)
• Part 3: What is CHimpREC? (my tool)
• Part 4: Inner workings of an imports rebuilder
• Part 5: Live 64-bit unpacking session
Part 1: What is unpacking?

• Packers are designed to protect the content of an executable binary or library
  – Commercial software (game copy protection)
  – Malware
• Most packers use encryption or compression
• Original assembly code not accessible
• Make static analysis impossible or at least, very hard
• Called a “shell” in Chinese
How simple packers work

Program
IAT
Import Directory
Unpacker Stub
Unpacker IAT
Import Directory

Entry Point
General unpacking theory
Part 2: What is ImpREC?

• 32-bit only imports rebuilder
• Saves a lot of time over the manual rebuilding method
• Wasn’t designed to deal with “features” of Vista
• No documentation or source code was available
• It did its job really well though
Part 2: What is ImpREC?
Why do this project?

- ImpREC was getting older.
- There were no public 64-bit imports rebuilder freely available on the internet at the time.
- Some functionality was missing (process dumper)
- I was curious.
  I am a reverser.
  It’s what I do.
- Got tired of waiting for somebody else to do it.
  So I made it by myself.
Part 3: What is CHimpREC?

- 32 and 64-bit imports rebuilder
- Improved version of ImpREC
- Fixes many existing bugs
- Introduces new features
- Made especially for WoW64 compatibility
- Allows for an all-in-one version
- Chinese version available (unexpectedly)
- Done entirely through black box reverse-engineering
- Done by making up equivalent operations that yield exactly the same results in all possible scenarios
Part 3: What is CHimpREC?
What is CHimpREC? (Chinese)
Limitations of ImpREC

XP or Vista w/o ASLR
Limitations of ImpREC

Vista ASLR
Limitations of ImpREC

Vista ASLR
## Limitations of ImpREC

### Vista ASLR

<table>
<thead>
<tr>
<th>GDI32</th>
<th>KERNEL32</th>
<th>USER32</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdi32.dll</td>
<td>77f10000 77f11000 R . . D .</td>
<td>kernel32.dll</td>
</tr>
<tr>
<td>gdi32.dll</td>
<td>77f11000 77f30000 R . X D .</td>
<td>kernel32.dll</td>
</tr>
<tr>
<td>gdi32.dll</td>
<td>77f53000 77f54000 R W . . D .</td>
<td>kernel32.dll</td>
</tr>
<tr>
<td>gdi32.dll</td>
<td>77f54000 77f57000 R . . D .</td>
<td>kernel32.dll</td>
</tr>
</tbody>
</table>

### Additional Libraries

- **KERNEL32**
- **USER32**
Limitations of ImpREC

Buffer Overflow Vulnerabilities
Limitations of ImpREC

Buffer Overflow Vulnerabilities

![Image of a Windows error message indicating that ImportREC_Wow64.exe has stopped working. The error message explains that a problem caused the program to stop working correctly and that Windows will close the program and notify the user if a solution is available. The interface includes options such as Debug, Close program, and various import-related functionalities.]
Part 4: Inner workings of an import rebuilder

- API families: Toolhelp32 vs. PSAPI
- How planning efficiently can save time
- 5-steps of the process:
  - Dump
  - IAT AutoSearch
  - Get Imports (Unforwarding)
  - Show Invalid
  - Fix Dump
Toolhelp32 vs. PSAPI

- Toolhelp32 APIs
  - CreateToolhelp32Snapshot
  - Process32First
  - Process32Next
  - Module32First
  - Module32Next
  - ToolHelp32ReadProcessMemory
Toolhelp32 vs. PSAPI

• PSAPI APIs
  – EnumProcesses
  – EnumProcessModules
  – EnumProcessModulesEx
  – GetModuleInformation
  – GetModuleBaseName
  – GetModuleFileNameEx
## Toolhelp32 vs. PSAPI

### Windows Version Compatibility

<table>
<thead>
<tr>
<th></th>
<th>95</th>
<th>98</th>
<th>Me</th>
<th>NT4</th>
<th>2000</th>
<th>2003</th>
<th>XP</th>
<th>Vista</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateToolhelp32Snapshot</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EnumProcessModules</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EnumProcessModulesEx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
How planning efficiently can save time

• 2 Single-Architecture versions (x86 OR x64)
  – To each his own
  – APIs: CreateToolhelp32Snapshot
  – Best OS compatibility range
  – Allows for common project source and headers
  – Coded in 32-bit then ported to 64-bit

• Cross-Architecture All-in-one version (x86 AND x64)
  – Made from a different x64 project
  – Requires 64-bit OS
  – EnumProcessModules & Ex
  – Runs on Vista x64 and Windows 7 only (not XP)
Step 1: Dump

- Copying the memory area of a process to a file
- When the process has reached its Original Entry Point
- Each section is dumped individually
- Each section RawSize must be realigned from FileAlignment to SectionAlignment
- RawAddress matches VirtualAddress
- All sections are made writable by adding the flag:
  - IMAGE_SCN_MEM_WRITE
- VirtualProtectEx to change the process memory to:
  - PAGE_EXECUTE_READWRITE
Step 1: Dump

<table>
<thead>
<tr>
<th>Name</th>
<th>Virtual Size</th>
<th>Virtual Address</th>
<th>Raw Size</th>
<th>Raw Address</th>
<th>Reloc Address</th>
<th>Linenumbers</th>
<th>Relocations N...</th>
<th>Linenumbers ...</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte[8]</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Word</td>
<td>Word</td>
<td>Dword</td>
</tr>
<tr>
<td>UPX0</td>
<td>00005000</td>
<td>00001000</td>
<td>00000000</td>
<td>00000400</td>
<td>00000000</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>E0000080</td>
</tr>
<tr>
<td>UPX1</td>
<td>00001000</td>
<td>00006000</td>
<td>00000600</td>
<td>00000400</td>
<td>00000000</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>E0000040</td>
</tr>
<tr>
<td>.rsrC</td>
<td>00001000</td>
<td>00007000</td>
<td>00002000</td>
<td>00000A00</td>
<td>00000000</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>C0000040</td>
</tr>
</tbody>
</table>
Step 2: IAT AutoSearch

- Binary search looking for indirect call opcodes:
  - 8B0D MOV ECX,[ADDRESS]
  - 8B15 MOV EDX,[ADDRESS]
  - 8B1D MOV EBX,[ADDRESS]
  - 8B25 MOV ESP,[ADDRESS]
  - 8B2D MOV EBP,[ADDRESS]
  - 8B35 MOV ESI,[ADDRESS]
  - 8B3D MOV EDI,[ADDRESS]
  - A1 MOV EAX,[ADDRESS]
Step 2: IAT AutoSearch

- Binary search looking for direct call opcodes:
  - FF15 CALL [ADDRESS]
  - FF25 JMP [ADDRESS]
  - FF35 PUSH [ADDRESS]
- Binary search ignores relative calls in 32-bit
- Starting from ImageBase or EntryPoint
- Found call must lead to a valid import
- Search up for the beginning of the IAT
- Search down for the end of the IAT
- Just like trying to identify a weird object in the dark
Step 2: IAT AutoSearch

```
public start
start proc near
  6A 00  push  0      ; lpModuleName
  E8 61 00 00 00  call GetModuleHandleA
  A3 04 30 40 00  mov  hInstance, eax
  6A 00  push  0      ; dwInitParam
  68 29 10 40 00  push offset DialogFunc ; lpDialogFunc
  6A 00  push  0      ; hWndParent
  6A 65  push  65h    ; lpTemplateName
  FF 35 04 30 40 00  push hInstance    ; hInstance
  E8 54 00 00 00  call DialogBoxParamA
  6A 00  push  0      ; uExitCode
  E8 59 00 00 00  call ExitProcess
start endp
```

```
.text:00401088
.text:00401088
.text:00401088
.text:00401088
.text:00401088
.text:00401088
.text:00401088
.text:00401088
 FF 25 00 20 40 00  jmp ds:__imp_GetModuleHandleA
 GetModuleHandleA endp
```
Step 2: IAT AutoSearch

.idata:00402000; Imports from kernel32.dll
.idata:00402000;
.idata:00402000; ==============================================================
.idata:00402000; _idata
.idata:00402000; HMODULE __stdcall GetModuleHandleA(LPCSTR lpModuleName)
.idata:00402000; extern __imp_GetModuleHandleA:dword
.idata:00402000; ; DATA XREF: GetModuleHandleA@r
.idata:00402004; void __stdcall ExitProcess(UINT uExitCode)
.idata:00402004; extern __imp_ExitProcess:dword ; DATA XREF: ExitProcess@r
.idata:00402008
.idata:0040200C; Imports from user32.dll
.idata:0040200C;
.idata:0040200C; BOOL __stdcall EndDialog(HWND hWnd, INT_PTR nResult)
.idata:0040200C; extern __imp_EndDialog:dword ; DATA XREF: EndDialog@r
.idata:00402010; INT_PTR __stdcall DialogBoxParamA(HINSTANCE hInstance, LPCSTR lpTemplateName, HWND hWndParent,
.idata:00402010; extern __imp_DialogBoxParamA:dword
.idata:00402010; ; DATA XREF: DialogBoxParamA@r
.idata:00402010
.rdata:00402014; ==============================================================

© nCircle 2010 All rights reserved.
Step 3: Get Imports

• Identify the elements of the IAT in the specified range
• Exactly the contrary of GetProcAddress
• Using custom-made reusable functions:
  – GetProcModuleName
  – GetProcName
  – GetProcOrdinal
  – GetProcNameAndOrdinal
  – GetProcInfo
  – Unforward
Step 3: Get Imports (Unforwarding)

- The Entry Point of the function is not code but a string
- Imports are forwarded for compatibility between all the different versions of Windows
- If an import can be unforwarded, it doesn’t mean that it really was forwarded
- There are many false-positives
- Must analyze the context with some fuzzy logic
- Could be called guessing too
Step 3: Get Imports (Unforwarding)
Step 3: Get Imports (Unforwarding)

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Function RVA</th>
<th>Name Ordinal</th>
<th>Name RVA</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>0000290C</td>
<td>00004106</td>
<td>00003608</td>
<td>0000500E</td>
</tr>
<tr>
<td>(nFunctions)</td>
<td>Dword</td>
<td>Word</td>
<td>Dword</td>
<td>szAnsi</td>
</tr>
<tr>
<td>00000096</td>
<td>0001751E</td>
<td>0095</td>
<td>00014BF0</td>
<td>DefWindowProcA</td>
</tr>
<tr>
<td>00000097</td>
<td>00017539</td>
<td>0096</td>
<td>00014BFF</td>
<td>DefWindowProcW</td>
</tr>
</tbody>
</table>

```
user32.dll:76460800 user32_dll segment byte public 'CONST' use32
user32.dll:76460800 assume cs:user32_dll
user32.dll:76460800 ;org 76460800h
user32.dll:764779e8 aNtdll_ntdll_d 1 db 'NTDLL.NtdllDialogWndProc_A',0
user32.dll:76477503 aNtdll_ntdll_dia db 'NTDLL.NtdllDialogWndProc_W',0
user32.dll:7647751E aNtdll_ntdll_def db 'NTDLL.NtdllDefWindowProc_A',0
user32.dll:76477539 aNtdll_ntdll_d_0 db 'NTDLL.NtdllDefWindowProc_W',0

.idat:0096d790 ; LRESULT __stdcall DefWindowProcW(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
.idat:0096d790 4d 30 84 77 DefWindowProcW dd offset ntdll_NtdllDefWindowProc_W
```
Step 3: Get Imports (Unforwarding)

False-positives
Step 3: Get Imports (Unforwarding)

Forwarding by ordinal
Step 4: Show Invalid

- Display unidentified IAT entries
- Text search through the interface
- Check all imports one by one for validity
- Simplest step to implement
Step 5: Fix dump

- Recreate the Import Directory to satisfy the loader
- Restore the original IAT
- Assemble structures that point to each other
  - IMAGE_IMPORT_DESCRIPTOR
  - IMAGE_IMPORT_BY_NAME
- Like gears in a clock
Changes from PE to PE32+ format

• All registers extended to QWORDs
  – EAX -> RAX
  – ESP -> RSP

• New registers
  – R8X-R15X

• All DLLs used must be 64-bit
• BaseOfData has disappeared
• New calling convention for APIs
Changes in the imports rebuilding process

- IAT elements are QWORDs
- Pointer to Original First Thunk is a QWORD
- ImageBase is a QWORD
- Exception Handlers are now stored as structures in the new PE32+ Exception Directory
Part 5: Live 64-bit unpacking session

- Tools used:
  - IDA Pro Advanced 64
  - CHimpREC-64

- Example: MPRESS 1.07
  - Simple UPX-like packer
Do you have any questions?
For more information or the full deck of slides, come see me at the nCircle booth.