

Module Extraction and DLL Hijacking Detection via Single or Multiple Memory Dumps

Pedro Fernández-Álvarez, **Ricardo J. Rodríguez***

© All wrongs reversed – under CC-BY-NC-SA 4.0 license



Universidad
Zaragoza

Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

March 23, 2023

10th Annual DFRWS Europe Conference
Bonn, Germany



*Corresponding author: rjrodriguez@unizar.es

Outline

- 1 Introduction
- 2 Background
- 3 Modex and Intermodex
- 4 Experiments
- 5 DLL Hijacking Detection
- 6 Conclusions and Future Work

Outline

- 1** Introduction
- 2 Background
- 3 Modex and Intermodex
- 4 Experiments
- 5 DLL Hijacking Detection
- 6 Conclusions and Future Work

Introduction

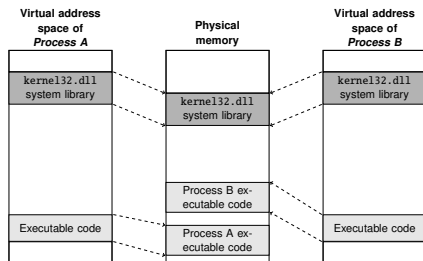
Motivation

Dynamic-Link Library (DLL)

- **Shared library containing functions and data that others can use**
- Helps promote code modularization, code reuse, and efficient memory usage, among other benefits
- It is a **module** (in Windows, *a module is an executable file or DLL*)

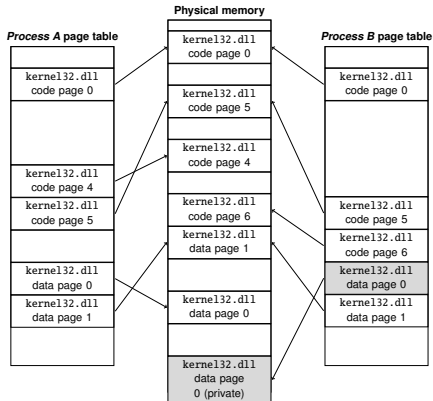
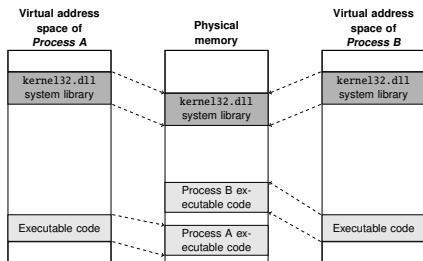
Introduction

Motivation



Introduction

Motivation

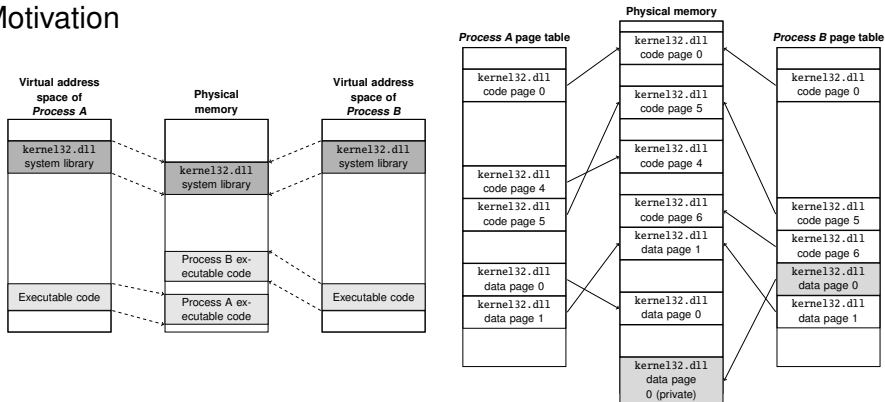


■ Only pages accessed by the process are mapped into virtual memory

- A page is a contiguous block of virtual memory of fixed length (typically, 4KiB)

Introduction

Motivation



■ Only pages accessed by the process are mapped into virtual memory

- A page is a contiguous block of virtual memory of fixed length (typically, 4KiB)

Limitation of extraction tools: process-level view

Current tools for extracting modules from memory dumps only dumps the pages mapped into a single process address space

Introductions

Summary of our contributions

- **New tools to get as much content as possible from a given module**
 - From a single memory dump (*intradump extraction*): Modex
 - From multiple memory dump (*interdump extraction*): Intermodex
- **Both tools are released under GNU/GPLv3 license at GitHub**
 - Python3-based tools
 - Modex is a Volatility 3 plugin, while Intermodex is a standalone tool that relies on Modex
 - **Create. Share. Build community** ♥
- Relevant to analyze:
 - **Whether a DLL module is malicious or not**
 - **Detection of DLL hijacking attacks**

Introductions

Summary of our contributions

- **New tools to get as much content as possible from a given module**
 - From a single memory dump (*intradump extraction*): Modex
 - From multiple memory dump (*interdump extraction*): Intermodex
- **Both tools are released under GNU/GPLv3 license at GitHub**
 - Python3-based tools
 - Modex is a Volatility 3 plugin, while Intermodex is a standalone tool that relies on Modex
 - **Create. Share. Build community** ♥
- Relevant to analyze:
 - **Whether a DLL module is malicious or not**
 - **Detection of DLL hijacking attacks**

Some remark...

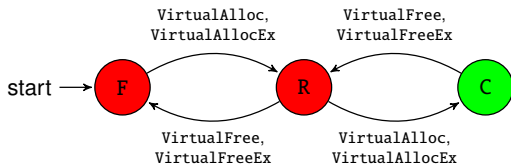
We focus on module extraction of Windows 64-bit DLLs, but our tools are also valid for extracting modules of Windows 64-bit executable files!

Outline

- 1 Introduction
- 2 Background**
- 3 Modex and Intermodex
- 4 Experiments
- 5 DLL Hijacking Detection
- 6 Conclusions and Future Work

Background

On Windows virtual memory management



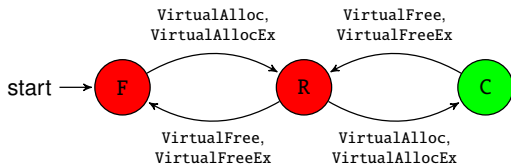
- States of a page: *free, reserved, committed*

Page Table Entries (PTE)

- Relationship between virtual memory and physical memory

Background

On Windows virtual memory management



- States of a page: *free, reserved, committed*

Page Table Entries (PTE)

- Relationship between virtual memory and physical memory

Shared vs. private pages ↔ prototype vs. real/process PTE

- Shared pages are stored only once in physical memory

- **Prototype PTE**: enables shared memory support in Windows

- Copy-on-write mechanism

- Prevents modifications to shared pages from being visible to processes sharing them

Background

On Windows virtual memory management

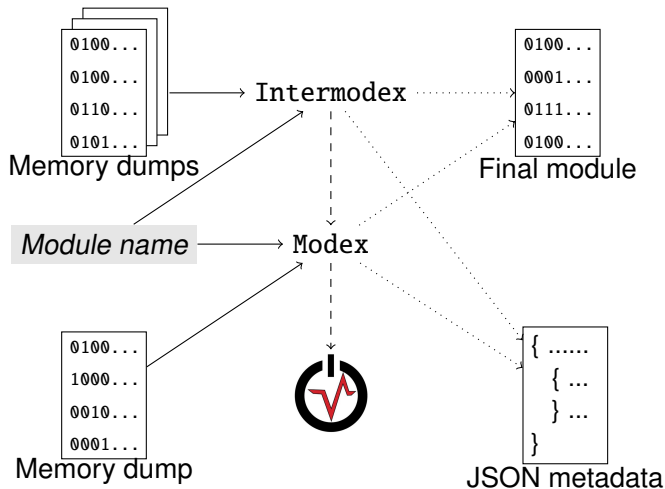
Page frame number database (PFN DB)

- Windows kernel data structure
- **Describes each page stored in physical memory** (PFN DB entry)
- Fields of interest:
 - *PteAddress*: contains the virtual address of the PTE
 - *PrototypePTE*: determines whether it is a prototype PTE or not

Outline

- 1 Introduction
- 2 Background
- 3 Modex and Intermodex**
- 4 Experiments
- 5 DLL Hijacking Detection
- 6 Conclusions and Future Work

Modex and Intermodex



Modex and Intermodex

Implementation details of Modex

- **Volatility 3 plugin**

- **Input:** a memory dump and the module name to be extracted

- **Output:** final (combined) module, JSON file, and an execution log file

- **Workflow:**

- 1 Walks through all the processes in the memory dump and checks which one loaded the given module as an argument
- 2 It dumps this module, saving it as an *intermediate .dmp* file
- 3 Those intermediate *.dmp* files are combined in a single *.dmp* file

Modex and Intermodex

Implementation details of Modex

■ Volatility 3 plugin

- **Input:** a memory dump and the module name to be extracted
- **Output:** final (combined) module, JSON file, and an execution log file

■ **Workflow:**

- 1 Walks through all the processes in the memory dump and checks which one loaded the given module as an argument
- 2 It dumps this module, saving it as an *intermediate .dmp* file
- 3 Those intermediate *.dmp* files are combined in a single *.dmp* file

■ **Third-party dependencies:**

- DllList plugin: to dump modules from processes
- SimplePteEnumerator plugin: to check the *PrototypePTE* flag in PFN DB entries

■ **Current limitation:** only works for 64-bit modules

Modex and Intermodex

Modex: rules for reconstructing modules

Which page do we choose for the combined module?

Modex and Intermodex

Modex: rules for reconstructing modules

Which page do we choose for the combined module?

At a given offset:

- **No page retrieved** → the page is filled with zeroes
- **Only one page is retrieved** → this page is put into the final module

Modex and Intermodex

Modex: rules for reconstructing modules

Which page do we choose for the combined module?

At a given offset:

- **No page retrieved** → the page is filled with zeroes
- **Only one page is retrieved** → this page is put into the final module
- **Multiple pages are retrieved:**
 - *All pages are shared*: we choose one of them at random
 - *Some pages are private, some are shared*: we discard the private pages and consider only the shared pages, choosing one of them at random

Modex and Intermodex

Modex: rules for reconstructing modules

Which page do we choose for the combined module?

At a given offset:

- **No page retrieved** → the page is filled with zeroes
- **Only one page is retrieved** → this page is put into the final module
- **Multiple pages are retrieved:**
 - *All pages are shared*: we choose one of them at random
 - *Some pages are private, some are shared*: we discard the private pages and consider only the shared pages, choosing one of them at random
 - *All pages are private*: we choose the page that most closely resembles the shared page
 - Similarity score (with TLSH) between every two pages to reflect their similarity
 - Page with the lowest value is chosen as the page to include in the final module (recall that the score trend of TLSH is descending)

Modex and Intermodex

Implementation details of Intermodex

- **Python 3-based tool**
- **Input:** a directory containing multiple memory dumps and the module name
- **Output:** final (combined) module, JSON file, and an execution log file
- It relies on Modex, as Volatility cannot handle multiple dumps at once
- Module reconstruction follows the same rules as Modex
- **Rules for combining modules:**
 - R1** Loaded at the same base address
 - R2** With the same path
 - R3** With the same size
- **Performs also a derelocation process on the extracted module**

Outline

- 1 Introduction
- 2 Background
- 3 Modex and Intermodex
- 4 Experiments**
- 5 DLL Hijacking Detection
- 6 Conclusions and Future Work

Experiments

Methodology

- **VM Windows 10 64-bit** (Pro edition, version 21H2) with 8 GiB of RAM
- **Four applications installed** (most used and most popular):
 - Web browser (Google Chrome)
 - Word processor (Microsoft Word)
 - PDF reader (Adobe Acrobat Reader DC)
 - Spreadsheet processor (Microsoft Excel)
- **Simulation of user behavior in steps** (power on, web browsing, view PDFs, creation of Word and Excel documents)
- Each application is used for 5 minutes. **Two experimental scenarios:**
 - 1 Applications are not closed after using them
 - 2 Applications are closed after using them

Experiments

Methodology

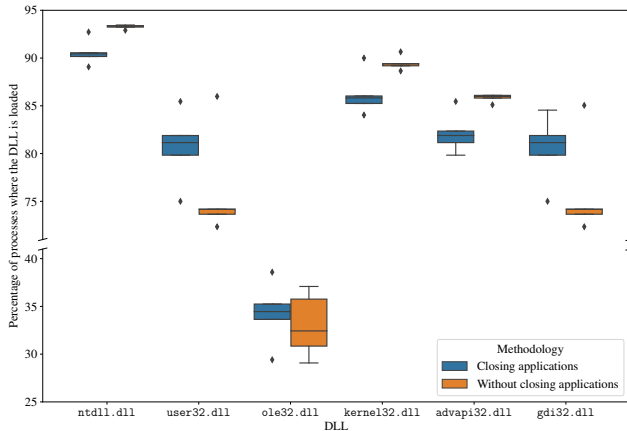
- **Memory dumps collected after each user step** (10 in total)
- **Subset of DLLs loaded by all the applications**
 - ntdll.dll, user32.dll, ole32.dll, kernel32.dll, advapi32.dll, and gdi32.dll

For each DLL and scenario:

- Modex on the first memory dump
- Intermodex on the first and second memory dumps
- Intermodex on the first, second, and third memory dumps
- ... (*until we consider all five memory dumps*)

Experiments

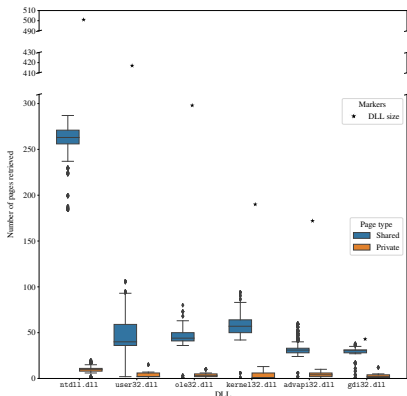
Use of selected DLLs in multiple processes



All of them (except ole32.dll) are loaded by a large no. processes

Experiments

Intradump extraction

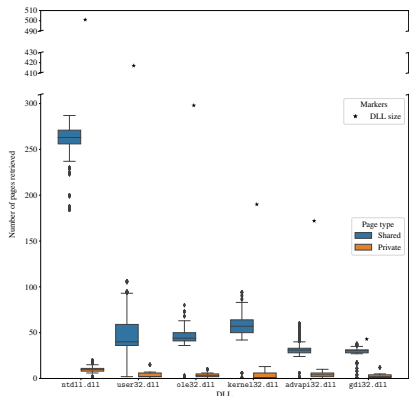


(a) Without combining pages

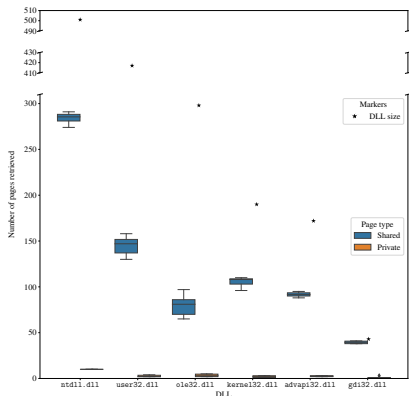
- Only the mapped pages on the address space of a process can be retrieved
- **No. shared pages is greater than the no. private pages for all modules**

Experiments

Intradump extraction



(a) Without combining pages

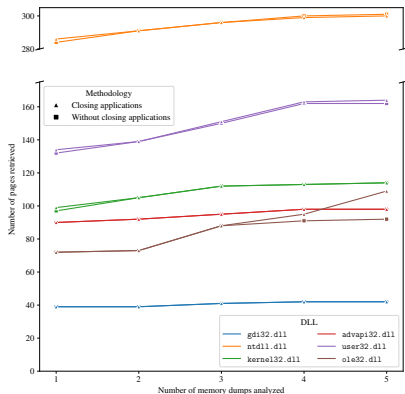


(b) Combining pages

- **Combined module contains more pages in all cases**
- **No. private pages decreases when modules are combined (as shared pages take precedence over private pages by our implementation)**

Experiments

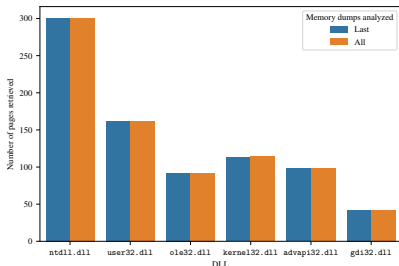
Interdump extraction



- **Results in both scenarios are very similar**, with slightly variations
 - Behavior of ole32.dll can be caused by many factors
- **More complete modules are obtained by combining pages and memory dumps**

Experiments

No. pages considering the last or all memory dumps



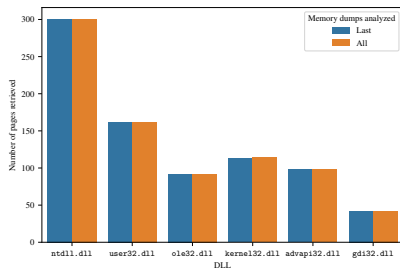
(a) Without closing applications

■ In the first scenario, practically no difference

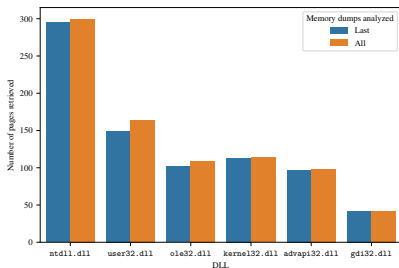
- The same content was in memory in both situations because the apps were not closed

Experiments

No. pages considering the last or all memory dumps



(a) Without closing applications



(b) Closing applications

■ In the first scenario, practically no difference

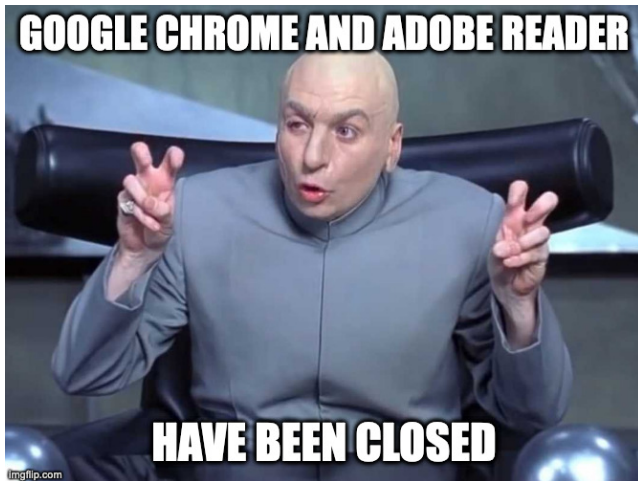
- The same content was in memory in both situations because the apps were not closed

■ In the second scenario, more memory dumps is clearly beneficial

- No. pages retrieved is slightly higher considering all vs. just the last memory dump
- We expected these differences to be larger... 😞

Experiments

No. pages considering the last or all memory dumps



Experiments

Interesting findings and limitations

- **Some pages marked as shared and owned by a DLL, loaded at the same base address in multiple processes, but with different content**
 - Found in DLLs other than the ones used for experimentation
 - We manually verified the differences in **these pages correspond to memory addresses stored within those pages**
 - **This happens very rarely.** We treat it as an anomaly and implement functionality to analyze it. When found, we choose the most repeated shared page



Experiments

Interesting findings and limitations

- **Some pages marked as shared and owned by a DLL, loaded at the same base address in multiple processes, but with different content**
 - Found in DLLs other than the ones used for experimentation
 - We manually verified the differences in **these pages correspond to memory addresses stored within those pages**
 - **This happens very rarely.** We treat it as an anomaly and implement functionality to analyze it. When found, we choose the most repeated shared page



Limitations

- **Base addresses of the modules must be the same to be combined**
 - A page-granularity level derelocation process is required to normalize the page content before combining the dumped modules (future work)

Outline

- 1 Introduction
- 2 Background
- 3 Modex and Intermodex
- 4 Experiments
- 5 DLL Hijacking Detection**
- 6 Conclusions and Future Work

DLL Hijacking Detection

Background on hijacking execution flow attacks

■ Different purposes:

- Persistence
- Escalating privileges
- Hiding malicious actions behind a legitimate process

■ DLL search order hijacking:

- Adversaries take advantage of the Windows DLL search order to make a particular program load a malicious DLL
- The malicious DLL must have the same filename as the legitimate one, and also the same exported function names
- These functions must work as the originals, so that the program can run as usual
- **DLL proxying: acts as a proxy between the program and the legitimate DLL**
 - A well-known malware that uses this technique is Stuxnet

DLL Hijacking Detection

- --detect flag
 - No module is extracted in this case
 - The JSON file provided as output contains information about the detection of DLL hijacking techniques
 - Modex indicates the affected processes, while Intermodex also indicates the affected memory dumps
- **Module path and size in all processes that contains it are compared**
 - Actual path and size are those that are most common for all the modules found
 - We assume that the processes targeted by DLL hijacking techniques are a minority
- **DLL hijacking detected when at least one path is different from the most common path or at least one size is different from the most common size**
 - **Disadvantage:** it will not detect the attack when the paths and sizes of the malicious DLL and the legitimate DLL match

DLL Hijacking Detection

- --detect flag
 - No module is extracted in this case
 - The JSON file provided as output contains information about the detection of DLL hijacking techniques
 - Modex indicates the affected processes, while Intermodex also indicates the affected memory dumps
- **Module path and size in all processes that contains it are compared**
 - Actual path and size are those that are most common for all the modules found
 - We assume that the processes targeted by DLL hijacking techniques are a minority
- **DLL hijacking detected when at least one path is different from the most common path or at least one size is different from the most common size**
 - **Disadvantage:** it will not detect the attack when the paths and sizes of the malicious DLL and the legitimate DLL match
- **Limitations**
 - **Our tools need a DLL name.** As future work, we will integrate this feature directly in Modex and Intermodex
 - **We focus exclusively on 64-bit processes**
 - **If the hijacked DLL is loaded only in a single process, there would be no other processes to compare against**

DLL Hijacking Detection

- PoC performing DLL proxying on cryptbase.dll
- VLC media player as victim application

```
{  
  "memory_dump_location": "file:///tmp/MemoryDumps/  
    InfectedDump.elf",  
  "mapped_modules": [  
    ...  
  ],  
  "dll_hijacking_detection_result": true,  
  "suspicious_processes": [  
    3208  
  ]  
}
```

Code 1: DLL hijacking detection of our PoC with Modex.

```
{  
  "dll_hijacking_detection_result": true,  
  "suspicious_processes": {  
    "file:///tmp/MemoryDumps/InfectedDump.elf": [  
      3208  
    ]  
  }  
}
```

Code 2: DLL hijacking detection of our PoC with Intermodex.

Outline

- 1 Introduction
- 2 Background
- 3 Modex and Intermodex
- 4 Experiments
- 5 DLL Hijacking Detection
- 6 Conclusions and Future Work**

Conclusions and Future Work

- **Two tools to extract a module as complete as possible from memory**
 - Modex: a Volatility 3 plugin that combines the pages of the same module that are mapped in different processes from a single Windows memory dump (*intradump* extraction)
 - Intermodex: it does the same, but with multiple memory dumps (*interdump* extraction)
- **Available under the GNU/GPLv3 license at [GitHub](#)**
- **Functionality to detect DLL hijacking attacks**

Conclusions and Future Work

- **Two tools to extract a module as complete as possible from memory**
 - Modex: a Volatility 3 plugin that combines the pages of the same module that are mapped in different processes from a single Windows memory dump (*intradump* extraction)
 - Intermodex: it does the same, but with multiple memory dumps (*interdump* extraction)
- **Available under the GNU/GPLv3 license at GitHub**
- **Functionality to detect DLL hijacking attacks**

Future work

- *How to combine the same modules with different base addresses?*
 - Theoretically, simply apply derelocation on the intermediate .dmp files. In practice...
- *Extend our tools to detect other DLL injection techniques*
- *Can we extract more content from packed malware modules using interdump extraction?*

Module Extraction and DLL Hijacking Detection via Single or Multiple Memory Dumps

Pedro Fernández-Álvarez, **Ricardo J. Rodríguez***

© All wrongs reversed – under CC-BY-NC-SA 4.0 license



Universidad
Zaragoza

Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

March 23, 2023

10th Annual DFRWS Europe Conference
Bonn, Germany



*Corresponding author: rjrodriguez@unizar.es