ISRuPIN-Win: Porting Instruction-Set Randomization Using Intel PIN Tool To Windows

Muhammad Ali Akbar (`UNI: maa2206`)
Supervision: Georgios Portokalidis and Angelos D. Keromytis
`maa2206@columbia.edu`

Computer Science Department,
Fu Foundation School of Engineering & Applied Sciences,
Columbia University, NY

1 Introduction

Instruction Set Randomization (ISR) is a technique used to counter code injection attacks. It works by randomizing the instructions understood by the system, making it difficult for an attacker to guess the right set of instructions to carry out an attack. Invalid instructions are readily trapped by the operating system, resulting in process termination.

Randomization using a 16-bit XOR key changes the instruction set as follows. A typical `xor eax, eax` instruction maps to the opcode `33 C0` on an x86 system. However, if we randomize it with the key `DE AD`, an attacker would need to inject `ED 6D`; changing the key to `CA FE` will change the correct opcode to `F9 3E`. If the key is regenerated randomly and the executable code is randomized periodically with the new key, it makes the task of a malware writer extremely difficult as (s)he has no way of guessing the right code for the instruction and change it with each key change without knowing the actual key.

In [1], a fast and reliable implementation of ISR is presented. With the help of Intel's dynamic instrumentation tool known as PIN [2], the proposed system randomizes an executable using randomly generated 16-bit XOR keys. It controls the execution of the executable and de-scrambles the instructions at runtime. Empirical experiments show a low overhead for the system as compared to the native execution. However, the system is limited to ELF executables and libraries for Linux.

Given the enormous market share that Microsoft Windows holds which is a huge incentive for commodity software development on Windows [3], it is imperative that this system needs to be ported to Windows as well. This work is a step towards this direction. We have ported the ISR system presented in [1] to native Windows environment. We use native Windows API to perform the same function as the Linux environment. Our system supports PE executable format, and has been tested on Windows 7.

2 Configuring & Using ISRuPIN-Win

This section describes the configuration and working of the components of the system. The system consists of five modules, which are described below:

2.1 objcopy: Randomizing the PE file

This is the modified version of objcopy. The original source file, the modified source file, the diff patch, and the binary executable are included.

Description: Objcopy takes a binary executable and creates a copy of it in the specified binary format. It supports a wide variety of formats including ELF and PE binaries. With the current patch, the objcopy can randomize the TEXT section of both PE and ELF format executables using the specified 16-bit XOR key.

Pre-requisites:

1. From binary: To run the included binary executable, you just need to install cygwin, and run this executable from cygwin's bash terminal.

2. From source: To recompile objcopy, you will need to get cygwin, install gcc and make. Then, get the binutils-2.21.1 source code, replace/patch the objcopy.c file and compile it.

Usage:

```
./objcopy --encrypt-xor-key [KEY] [input-file] [output-file]
```

For example,

```
./objcopy --encrypt-xor-key 0xDEAD input.exe encrypted-output.exe
```

2.2 ida: Generating patch (Separating interleaved DATA from CODE section)

This is a python script that uses IDA to generate patch file specifying the jumptables (DATA in the .text section).

Description: ELF executables have a clean separation between CODE and DATA. Unfortunately, the same is not true for PE executables. For example, the jump tables (DATA) is embedded right at the end of the function's code. Randomizing the whole TEXT section randomizes these jump addresses as well. This module is used to separate the DATA from CODE in the text section and create a patch file specifying the DATA blocks boundaries.

Pre-requisites: IDA Pro (Tested on v6).

Usage:

Open the original (non-encrypted) PE file with IDA. Wait till initial auto-analysis finishes. Run the python script from IDA (File → script file). In the output window, click on each result and verify that there is no false positive. If there is a false positive, you can fix it by telling IDA to convert it to code and then function. Run the script again, a patch file would have been generated (isr.func.patch).

2.3 apply_isr_patch: Patching Randomized PE File

This is the source code of the apply_isr_patch module.

Description: It reads the patch file, parses the input PE file, converts the virtual addresses in patch file to raw offset within PE file, and then patches the corresponding bytes in the output file.

Pre-requisites:

1. From binary: The compiled executable can be found here:
`apply_isr_patch\Debug\apply_isr_patch.exe`

2. From source: The visual studio project with the source code is attached. It has been created and compiled using Visual Studio 2008.

Usage:

```
apply_isr_patch.exe xor_encryption_key input_executable output_executable
patch_file
```

2.4 db: Updating ISRuPIN-Win Keys Database

This is the sqlite database for storing the encryption keys. The sqlite3 client, the database structure (sql) file, the database (db) file, and a script to insert keys in to the database are attached.

Pre-requisites: sqlite3.exe (included in the database folder).

Usage: Change the `varenckey` and `varencname` parameters to the required key and executable path in the insert_key_to_db.bat file, and run it to insert the key in the image.

You can browse the database using the sqlite3 client by:

```
sqlite3.exe image_keys.db
```

2.5 isrupin-win: Run the program using pintool

This is the source code of the isrupin (ISR pin tool for windows).

Description: This is the main PIN tool. It loads binary executable and the dlls. It reads the corresponding keys from the database. It sets up the instrumentation of the binary executable, and decodes the randomized binary instructions on the fly as they are fetched. It uses a memory protector to separate the tool and executable's memory.

Pre-requisites:

1. From binary: The compiled pintool can be found here:

```
isrupin\Debug\isrupin.dll
```

2. From source: The visual studio project with the source code is attached. It has been created and compiled using Visual Studio 2008.

Visual Studio Configuration: The project has been configured with relative library addresses. Get the pin-2.10-43611-msvc9-ia32_intel64-windows PIN package, and put the project folder (isrupin) at this location:

```
PIN_FOLDER\source\tools\MyWorkspace
```

Where PIN_FOLDER is the folder where you extracted the PIN package.

Usage:

```
[FULL_PATH_TO_PIN_FOLDER]\pin.bat -follow_execv -t [FULL_PATH_TO_PIN_FOLDER]\source\tools\MyWorkspace\isrupin\Debug\isrupin.dll
-unique_logfile -keydb [FULL_PATH_TO_DATABASE_FOLDER]\image_keys.db -- [FULL_PATH_TO_EXECUTABLE]
```

For example:

```
C:\pin-2.10-43611-msvc9-ia32_intel64-windows\pin.bat -follow_execv -t C:\pin-2.10-43611-msvc9-ia32_intel64-windows\source\tools
\MyWorkspace\isrupin\Debug\isrupin.dll -unique_logfile -keydb C:\database\image_keys.db -- C:\wamp32\bin\apache\Apache2.2.21\bin\httpd.exe
```

Formats supported: PE executables (32 bit)

Operating System: Tested on Windows 7

PIN version: pin-2.10-43611-msvc9-ia32_intel64-windows

3 Workflow

A workflow diagram that represents the steps required to randomize and run a PE using ISRuPIN-Win is given in Figure 1.

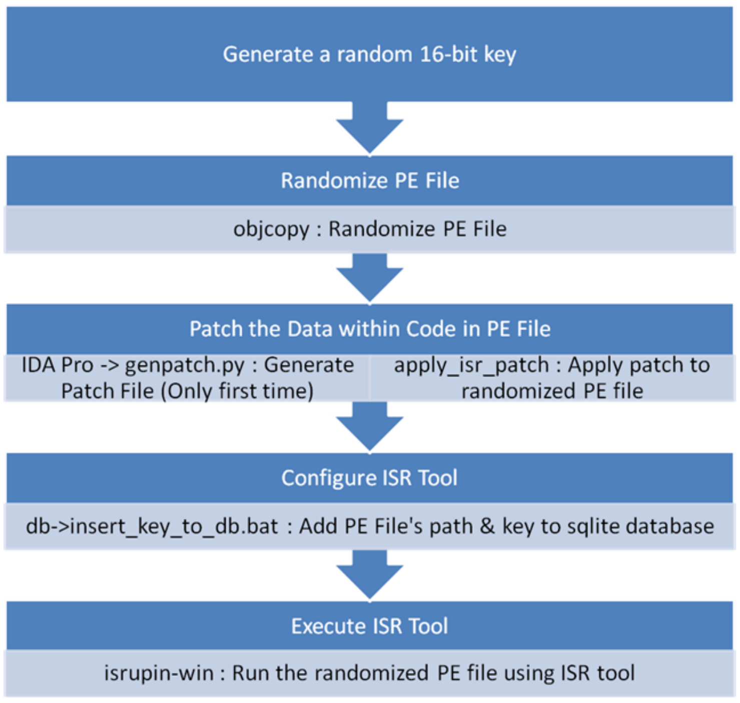
Generate a random 16-bit key
Randomize PE File
objcopy : Randomize PE File
Patch the Data within Code in PE File
IDA Pro -> genpatch.py : Generate Patch File (Only first time)
apply_isr_patch : Apply patch to randomized PE file
Configure ISR Tool
db->insert_key_to_db.bat : Add PE File's path & key to sqlite database
Execute ISR Tool
isrupin-win : Run the randomized PE file using ISR tool

Fig. 1. Workflow Diagram

The patch file generation (using IDA pro) needs to be done only once. This step can be skipped when re-encrypting the same binary with a different key. The patch file needs to be generated only if the binary changes. However, the randomized binary should be patched (using apply_isr_patch) every time the key is changed.

4 Challenges in Porting to Windows

Some of the important challenges we faced in porting ISRuPIN to windows are described below.

4.1 Switching to Windows API

For fast execution, it is imperative that ISRuPIN is run natively on Windows (without any emulation layer). For some cases, replacing Linux system calls with Windows API was straightforward. In other cases, we resorted to combining different APIs to perform the operations needed.

4.2 No fixed System call numbers

We want to track the memory usage of the executable in an attempt to separate its memory from the PIN tool's memory. Tracking specific system calls on Linux is easy, because the system call numbers are set in stone. Windows, on the other hand, treats system call numbers as another way to provide ASLR. System call numbers are changed in every minor Windows version. This is supposed to make it difficult for malware writers to make system calls using the system call table. This limits the tracking of memory management system calls. We had to resort to replacing the related API calls with wrapper functions using PIN API, and thus adding a level of indirection to be able to detect the required system calls. Note that this has to be done for each system call we want to monitor. We replaced this approach with a reactive one. Instead of monitoring memory management related system calls, we wait till we get a page fault, and then use Windows API to traverse the structure for allocated memory regions. This approach seems to work fast and efficient, as the traversal is done only on a hit, and the result is noted in an index of allowed memory regions for future use.

4.3 'Not So Clean' Separation of Code and Data

ELF executables have a clean separation between CODE and DATA. Unfortunately, the same is not true for PE executables. For example, the jump tables (DATA) is embedded right at the end of the function's code. Randomizing the whole TEXT section randomizes these jump addresses as well. This module is used to separate the DATA from CODE in the text section and create a patch file specifying the DATA blocks boundaries. We use Interactive Disassembler (IDA Pro) to disassemble the PE file and run a python script to identify the DATA blocks within TEXT section. Then, we parse the PE file using a custom C++ program, calculate offset in file corresponding to the virtual addresses, and patch those locations.

4.4 No equivalent of procfs

Windows doesn't have an equivalent of the virtual filesystem procfs that exports many kernel structures to the userspace. However, it does provide some decent API to walk through memory regions, shared modules etc, which almost makes up for the memory related information that procfs provides.

4.5 Difficult to replace known system dlls

Randomizing new dlls is easy as they can be placed in the executable's directory and will be loaded from there. However, Windows maintains a list of known dlls, and uses its own copy for known dlls even if they are not already loaded in memory. This makes it very difficult to replace the standard dlls with randomized ones without making this change for the whole operating system.

5 Testing

We tested the ISRuPIN-Win tool on several self written small programs. Testing was also done with programs that called Windows API and openssl library for encryption (both static and dynamic linking). The Apache web server (WAMP 32 bit windows distribution) was randomized and successfully tested with ISRuPIN-Win.

Some of the tools that we found helpful in debugging and testing were:

1. Text Logging from tool using PIN
2. Interactive Disassember (IDA Pro)

3. Diff tools (using VI, Windiff, VisualDiff etc)
4. Process Explorer

6 Conclusion

In this work, we have presented a port of ISR using PIN to Windows, described the workflow, discussed the challenges faced, and tested the implementation on real world production software.

References

1. Portokalidis, G., Keromytis, A.: Fast and practical instruction-set randomization for commodity systems. In: Proceedings of the 26th Annual Computer Security Applications Conference, ACM (2010) 41–48
2. Luk, C., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: ACM SIGPLAN Notices. Volume 40., ACM (2005) 190–200
3. Economides, N., Katsamakas, E.: Linux vs. windows: A comparison of application and platform innovation incentives for open source and proprietary software platforms. The Economics of Open Source Software Development, Elsevier Publishers (2006)