Instruction Set Randomization using Intel PIN Tool

Muhammad Ali Akbar

UNI: maa2206

MS CS, Columbia, NY

AIM

- To port the Instruction Set Randomization (ISR) PIN tool from Linux to Windows (Windows 7)

STEP: CONFIGURING VISUAL STUDIO & PIN

- include dirs:
  - ..\..\..\..\..\source\include\;..\..\..\..\..\source\include\gen;..\..\..\..\..\extras\components\include\;..\..\..\..\..\extras\xed2-ia32\include;

- lib dirs:
  - ..\..\..\..\..\ia32\lib-ext;..\..\..\..\..\extras\xed2-ia32\lib;..\..\..\..\..\ia32\lib;

- preprocessor defs:
  - ISRUPIN_EXPORTS;TARGET_IA32;HOST_IA32;TARGET_WINDOWS;USING_XED

- Additional Compiler opts:
  - /MT /EHs- /EHa- /wd4530 /DTARGET_WINDOWS /DBIGARRAY_MULTIPLIER=1 /DUSING_XED /D_CRT_SECURE_NO_DEPRECATE /D_SECURE_SCL=0 /nologo /DTARGET_IA32 /DHOST_IA32

- Additional linker opts:
  - /DLL /EXPORT:main /NODEFAULTLIB /NOLOGO /ENTRY:Ptrace_DllMainCRTStartup@12 ntdll-32.lib libxed.lib pin.lib pinvm.lib libcmt.lib libcpmt.lib /DEBUG

Step: Writing Test Tools with PIN

- Several test programs including:
  - Instruction Counting
  - Trace Disassembly
  - … and more

STEP: SETTING UP REQUIREMENTS FOR ISR TOOL

- Installed and Configured Sqlite library and headers
- Add new headers for missing unix types and byteswapping (stdint.h, byteorder.h)
- Setting up general UNIX memory functions like mmap and munmap using VirtalAlloc and VirtualFree
- Started the ISR PIN tool from scratch and started adding parts of code incrementally.

STEP: MEMORY PROTECTOR

- The userspace/kernel memory division in Windows is different.
  - Taking the case of 32 bit process, with normal settings (IMAGE_FILE_LARGE_ADDRESS_AWARE not set)
    - Total virtual address space = 4 GB
    - Userspace = 2 GB
    - Kernel space = 2 GB
- I tried out two approaches to enumerate the allocated memory space for process
  - Proactive approach
  - Reactive approach

Step: Memory Protector (Proactive)

- At Image load time.
- Track system calls for memory management (mmap, munmap, sbrk etc)
  - Challenge: Linux has fixed system call numbers, Windows changes the system call numbers in every minor update (for ASLR).
  - Solution: Using PIN, replaced the memory management functions with wrapper functions:
    - VirtualAlloc, VirtualAllocEx, VirtualFree, VirtualFreeEx, HeapCreate, HeapDestroy
- Not found anywhere? Walk through the shared libraries
  - Linux API for walking through shared libraries not available
  - Using EnumProcessModules & GetModuleInformation
- Didn’t work ☹

STEP: MEMORY PROTECTOR (REACTIVE)

- When an invalid address is found, Using VirtualQueryEx, walk through all allocated memory regions, find the address and mark the image boundaries as allowed.
- Works ☺
- Challenge: How to separate PIN tool memory from process memory
- Idea (not implemented): Use a black list for PIN tool memory, and don’t allow that portion to get marked.

STEP: GETTING THINGS TO WORK WITHOUT ACTUAL ISR

- Ported the code and got it to work without actual ISR.
- Worked successfully on small programs as well as large programs.
- Testing out on Apache httpd web server revealed that child processes are not followed properly (only on Windows 7).
- Luckily, the latest version of PIN issued in September fixed this. Reconfiguring with the new PIN release, all test cases passed (remember: no ISR yet).

Step: Randomizing Executable

- Got Cygwin, set it up for development
- Got binutils-2.21 and compiled it from source
- With very minor modifications for compile errors, the previous code for randomizing ELF executables using objcopy worked for PE executables too.

STEP: TESTING (WITH ISR)

- Simple Hello World.
  - Works ☺
- Added memory allocation etc.
  - Works ☺
- With windows API like CreateProcess
  - Works ☺
- Using Apache httpd server
  - Failed ☹
- Using MySQL server mysqld
  - Failed ☹

DEBUGGING

- Tools used
- Text Logging from tool using PIN
- Interactive Disassember (IDA Pro)
- Diff tools (using VI, Windiff, VisualDiff etc)
- Process Explorer
- Sqlite Clients

TESTING

- Tried to build programs that crashed from source with static linking instead of dynamic linking.
  - No success

- Digging deeper: disassembly of trace instructions using PIN tool for both randomized and unrandomized binaries
  - Finding the point of divergence in the program flow.
  - Comparing this with the disassembly in IDA to identify the cause for this divergence.

DEBUGGING/TESTING: JUMP TABLES

- It turns out that in PE:
  - ‘text’ section contains code as well as data
  - This data is basically the jump tables for switch cases.
  - If we encrypt the whole text section, the jump table is encrypted too.
  - One such jump table was causing the crash of the randomized binary

- To fix this:
  - Identify the code/data boundaries and decrypt them beforehand.

Patching: Jump Tables

- Tried about a dozen different disassembly libraries
- The most promising was libopdis with its control flow disassembly.
- The idea was to find all the code flow and patch only the code in the text section.
- It didn’t find the complete control flow properly.

- After trying different options, switched to IDA scripting. The benefit of IDA is that the user can interact with the disassembler and correct its mistakes interactively.

Patching: IDA Scripting

- IDC Script
  - Find the text section.
  - List all function boundaries.

- C++ program
  - Parses the PE headers
  - Reads the patch file
  - Translates virtual addresses in patch file to offset in the raw binary using PE headers
  - Patches (decrypts) the data blocks in text section in the randomized binary

PATCHING: IDA SCRIPTING

- Didn’t work ☹
  - There is a lot of code that is not part of functions.
    - Interrupt handlers, stubs for dynamic libraries, etc

- Needed something more sophisticated. Found out that IDA has a powerful Python API available.

PATCHING: IDA SCRIPTING

- Python Script for IDA:
  - Find the text section
  - Traverse the entire virtual space of the text section
    - For each address, identify if it is part of data or code
  - Print the data blocks

- Identifying starting address of data block:
  - A data reference
    - Must be data reference from text section
  - No flow/non-flow code reference
  - Not beginning of a function
- Identifying ending address of data block:
  - Any flow/nonflow code reference

Testing

- Changed the C++ program for patching to take data block boundaries instead of function boundaries.

- Patched and successfully ran the randomized Apache httpd web server with ISR tool. ☺

To Do:

- Fix memory protector problem
- Find a way to fix data references from within text section which is not actually data
- Clean up, comment and package presentable code.