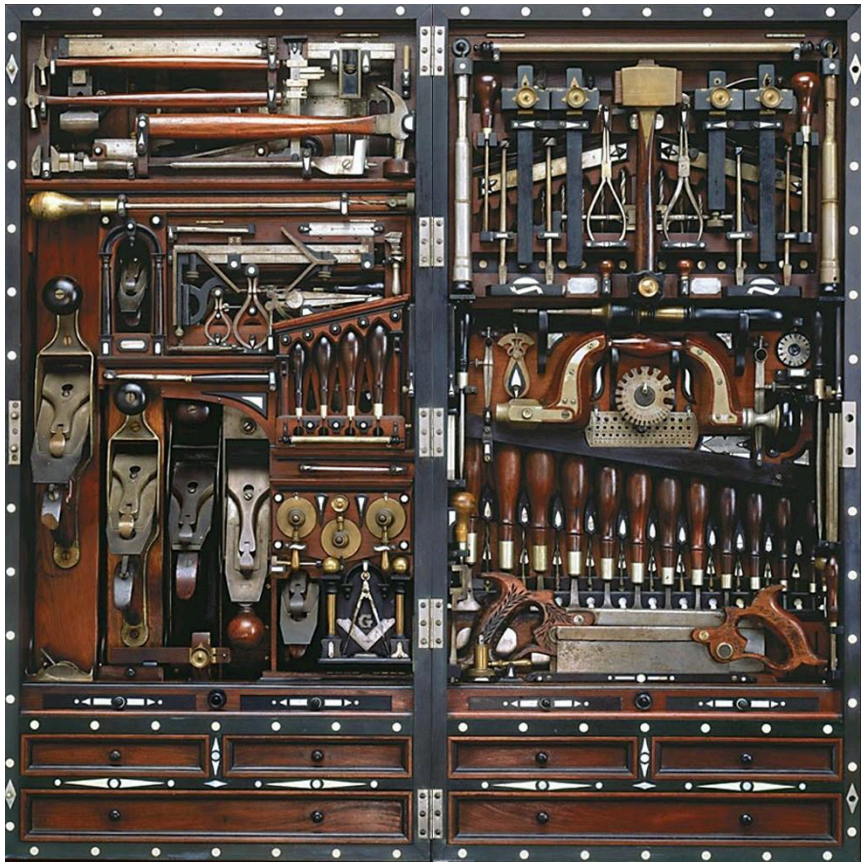

Program Analysis Tools

Edgar Barbosa

SyScan360

Beijing - 2015

Who am I?

Edgar Barbosa

Senior Security Researcher - COSEINC

Experience with Windows kernel rev eng and rootkits

My current focus is on applications of SMT solvers for Program Analysis

What to expect?

Program Analysis (and also Reverse Engineering) requires a lot of tools

We can do nothing without tools!

There will never be one tool able to solve all problems we can find on Program Analysis and Reverse Engineering

This talk will demonstrate how to use new tools and also less popular tools and libraries for program analysis

> because everybody already knows IDA Pro, GDB, OllyDBG :)

This presentations has few slides but is full of **DEMOs**

> lets hope they work!!! :)

The source code used for all demos will be released.

TOOLS

Tools

Every reverse engineer knows how important is to have access to good tools

Fortunately we have access to awesome tools, some of them opensource.

There are tools that everyone knows:

- IDA Pro
- Windbg
- GDB
- binutils

But there other tools very useful for program analysis which are not so famous.

This talk will show some of these tools

Symbolic execution

Concolic Execution uses Concrete execution + Symbolic Execution

Traces + SMT solvers

Binary instrumentation tools are the best options because they are faster than tracing with a common debugger

2 great tools:

- PIN
- DynamoRio

Intel SDE

From Intel

Uses Intel PIN and XED

SDE - Software Development Emulator

Allows you to run software that uses instructions not supported by your Intel CPU!

but can do much more than just emulation!!!

Supports Mac, Windows and Linux

32 and 64 bits

Command-line tool! Great for automation/scripts

https://software.intel.com/en-us/articles/intel-software-development-emulator

sde - emulation

```
-aes  [default 0]
       Emulate the AES instructions.
-avx  [default 0]
       Emulate the Intel AVX instructions. Default is depends upon host
       CPUID.
-avx2  [default 1]
       Emulate the Intel AVX2 instructions.
-bmi1  [default 1]
       Emulate the Intel BMI1 instruction.
-bmi2  [default 1]
       Emulate the Intel BMI2 instruction.
-clmul  [default 0]
       Emulate the PCLMULQDQ instruction.
-f16c  [default 1]
       Emulate the Intel AVX F16C instructions.
-fma  [default 1]
       Emulate the Intel FMA instructions.
-fsgs_abort  [default 0]
       Abort on executions of {RD,WR}{FS,GS}BASE
-lzcnt  [default 1]
       Emulate the LZCNT instruction.
-movbe  [default 1]
       Emulate MOVBE
-popcnt  [default 0]
       Emulate the SSE4.2 popcnt instruction.
-rdrand  [default 1]
       Emulate the Intel RDRAND instruction.
-sse3  [default 0]
       Emulate the SSE3 instructions.
-sse41  [default 0]
       Emulate the SSE4.1 instructions.
-sse42a  [default 0]
```

Intel SDE

sde -mix -- app (block stats)

sde -icount (instruction count)

sde -footprint -- app

traces:

- -itrace_execute (basic trace)
- -debugtrace (detailed trace)
  - dt_call
  - dt_instruction
  - dt_memory

-debugtrace

detailed trace includes:

every memory access (R/W and size)

disasm

```
TID0: INS 0xb7feb9e3                        BASE     jz 0xb7feb9e9
TID0: INS 0xb7feb9e9                        BASE     mov dword ptr [ebp-0x1c], edx
TID0: Write *(UINT32*)0xbfffedbc = 0xffffffff
TID0: Read 0x4008000 = *(UINT32*)0xbfffeda8
TID0: INS 0xb7feb9ec                        BASE     mov edx, dword ptr [ebp-0x30]        | edx = 0x4008000
TID0: INS 0xb7feb9ef                        BASE     mov dword ptr [ebp-0x20], eax
TID0: Write *(UINT32*)0xbfffedb8 = 0xffffc000
TID0: Read 0xffffc000 = *(UINT32*)0xbfffedb8
TID0: INS 0xb7feb9f2                        BASE     mov eax, dword ptr [ebp-0x20]        | eax = 0xffffc000
TID0: Read 0 = *(UINT32*)0xbfffedac
TID0: INS 0xb7feb9f5                        BASE     mov ecx, dword ptr [ebp-0x2c]        | ecx = 0
```

DynamoRio

Great alternative to PIN

Incredibly fast

DEMO

- memtrace
  - drrun -c ~/src/DynamoRIO/samples/bin32/libmemtrace.so -- ls
- instruction calls
  - drrun -c ~/src/DynamoRIO/samples/bin32/libinstrcalls.so -- ls
- conditional branch execution info
  - libcbr.so
  - libcbrtrace.so (log information)
- control flow transfer between modules

drcov

code coverage tool included with the DynamoRIO distribution

./drrun -t drcov -dump_text -- app args

by default it generates a binary format file with coverage information

drcov

```
DRCOV VERSION: 2
DRCOV FLAVOR: drcov-32
Module Table: 11
 0, 114688, /bin/ls
 1, 24576, /lib/i386-linux-gnu/libattr.so.1.1.0
 2, 253952, /lib/i386-linux-gnu/libpcre.so.3.13.1
 3, 20480, /lib/i386-linux-gnu/libdl-2.19.so
 4, 1761280, /lib/i386-linux-gnu/libc-2.19.so
 5, 36864, /lib/i386-linux-gnu/libacl.so.1.1.0
 6, 143360, /lib/i386-linux-gnu/libselinux.so.1
 7, 32768, /home/edgarmb/src/DynamoRIO/tools/lib32/release/libdrcov.so
 8, 69632, /home/edgarmb/src/DynamoRIO/lib32/release/libdrpreload.so
 9, 1638400, /home/edgarmb/src/DynamoRIO/lib32/release/libdynamorio.so.5.0
 10, 139264, /lib/i386-linux-gnu/ld-2.19.so
BB Table: 2484 bbs
<88>^D^@^@^N^@^H^@7í^@^@
^@
^@<80>í^@^@^H^@
^@dî^@^@^B^@
^@0î^@^@
^@
^@:î^@^@^R^@
^@Lî^@^@^X^@
^@Ði^@^@^Z^@
^@¨<87>^A^@^D^@
^@êi^@^@^Z^@
^@^Dí^@^@^G^@
```

drcov

- text format
- -dump_text
- file format
  - Version
  - Module Table
  - Basic Blocks

drcov - dump_text

```
DRCOV VERSION: 2
DRCOV FLAVOR: drcov-32
Module Table: 11
 0, 114688, /bin/ls
 1, 24576, /lib/i386-linux-gnu/libattr.so.1.1.0
 2, 253952, /lib/i386-linux-gnu/libpcre.so.3.13.1
 3, 20480, /lib/i386-linux-gnu/libdl-2.19.so
 4, 1761280, /lib/i386-linux-gnu/libc-2.19.so
 5, 36864, /lib/i386-linux-gnu/libacl.so.1.1.0
 6, 143360, /lib/i386-linux-gnu/libselinux.so.1
 7, 32768, /home/edgarmb/src/DynamoRIO/tools/lib32/release/libdrcov.so
 8, 69632, /home/edgarmb/src/DynamoRIO/lib32/release/libdrpreload.so
 9, 1638400, /home/edgarmb/src/DynamoRIO/lib32/release/libdynamorio.so.5.0
 10, 139264, /lib/i386-linux-gnu/ld-2.19.so
BB Table: 2486 bbs
module id, start, size:
module[  8]: 0x00000488,  14
module[ 10]: 0x0000ed37,  10
module[ 10]: 0x0000ed80,   8
module[ 10]: 0x0000ee64,   2
module[ 10]: 0x0000ee30,  10
module[ 10]: 0x0000ee3a,  18
module[ 10]: 0x0000ee4c,  24
module[ 10]: 0x0000ecd0,  26
module[ 10]: 0x000187a8,   4
module[ 10]: 0x0000ecea,  26
module[ 10]: 0x0000ed04,   7
module[ 10]: 0x0000ed0b,  13
module[ 10]: 0x0000ed18,   4
module[ 10]: 0x0000ed1c,  27
```

./crasher demo

classical example of the Microsoft SAGE paper

Last year I demonstrated how to solve it using SMT

Now we will solve it using search. The fitness function is the coverage (new basic blocks discovered).

Same principle of AFL and BCCF fuzzers (thanks @joxean)

Just a simple toy. Good to test new heuristics/fitness functions.

real-world

crasher is a toy example

we like real-world useful applications

lets test other tools

- readelf
- nm
- exif

DEMO

Translation

After access to the trace, we need to translate the instructions to SMT

We have 2 options:

- direct translation (hard)
- intermediate languages

The number of instructions of Intel Architecture is huge

There are good tools/libraries we can use to know the precise semantics of an x86/64 instruction

- amoco
- openreil

amoco

Created by Axel Tillequin

Python package for static analysis of binaries

https://github.com/bdcht/amoco

DEMO

openreil

translator and tools for REIL (Reverse Engineering Intermediate Language)

REIL was created by zynamics (Google) - binnavi

openreil was created by Dmytro Oleksiuk (@d_olex)

DEMOS

symbolic emulator

PySymEmu

Symbolic Emulator

https://github.com/feliam/pysymemu

Symbolic execution of ELF32/64 files

DEMO

Synthesis

Another way to translate instructions

Brute-force (Inspired by superoptimizers)

Generate code able to satisfy I/O examples

Search problem

DEMO

String solvers

Last year I discussed about applications of SMT solvers to program analysis

This time I will show a new application for a previously unsupported data type: strings!

https://github.com/z3str/Z3-str

https://people.csail.mit.edu/akiezun/hampi/

http://webblaze.cs.berkeley.edu/2010/kaluza/

We can express constraints about strings

You may need to learn a small DSL to use some tools

DEMO

Conclusion

Conclusion

There is a lot of new and cool tools being developed

Golden age for reverse engineering (imho)

They are open source! (Intel SDE is an exception)

Remember that automation is good but can’t solve everything!

Contribute!

Source

Links

The source code of all demos/tools will be released at https://github.com/edgarmb/syscan360-2015 in the next few days.

THANK YOU