Applications of the Reverse Engineering Language REIL

Hackers to Hackers Conference 2009, São Paulo

Sebastian Porst
zynamics GmbH
(sebastian.porst@zynamics.com)

Talk Overview

- Necessity of new RE methods
- Solutions we developed
- Applications of our solutions

About zynamics

- Small German company
- Unhappy with the state of Reverse Engineering
- Needed: New RE tools and methods
  - BinDiff, BinNavi, VxClass

About me

- Lead Developer of BinNavi
- Many years of RE experience
- Try to come up with new RE methods
- Talk about it at conferences

What we are doing

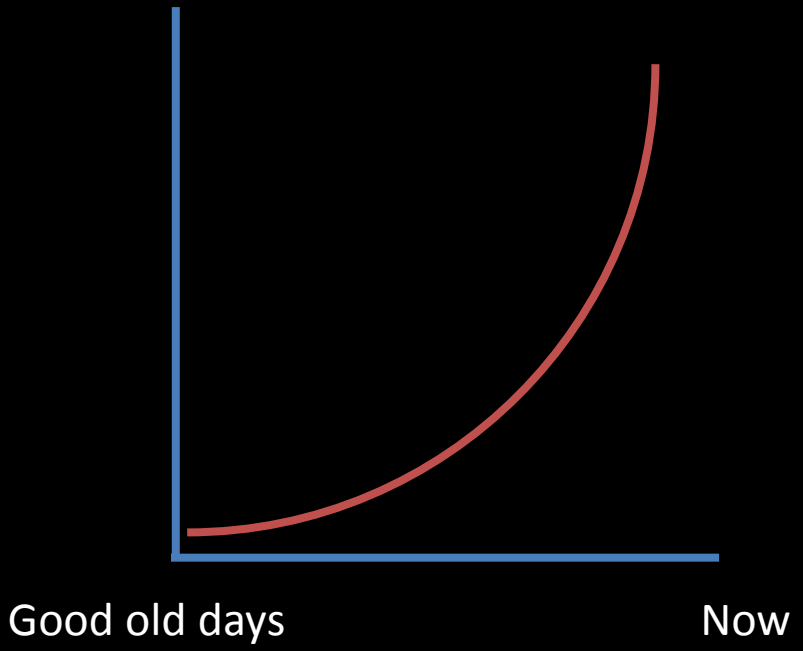
- Build Reverse Engineering tools
- Try to automize binary file analysis
- Help people find vulnerabilities

Why we are doing this

Software Complexity

Architectural Diversity

Microsoft Security Budget



How we are doing this

- Develop new RE methods
  - Platform-Independent
  - Easy to use
- Integrate them into our tools

REIL

- Reverse Engineering Intermediate Language
- Platform-Independent
- Designed for Reverse Engineering

Design Principles

- Very small instruction set
- Very regular operand structure
- Very simple operand types
- No side-effects


1005F9000 ldm 0x100123C, , t0 // 01005F90 mov esi, ds: [SendDlgItemMessageW]
1005F9001 str t0, , esi
1005F9600 sub esp, 4, qword t0 // 01005F96 push ebx
1005F9601 and qword t0, 0xFFFFFFFF, esp
1005F9602 stm ebx, , esp
1005F9700 sub esp, 4, qword t0 // 01005F97 push 30
1005F9701 and qword t0, 0xFFFFFFFF, esp
1005F9702 stm 0x1E, , esp
1005F9900 ldm esp, , t0 // 01005F99 pop edi
1005F9901 add esp, 4, qword t1
1005F9902 and qword t1, 0xFFFFFFFF, esp
1005F9903 str t0, , edi
1005F9A00 str 0x100A3E0, , ebx // 01005F9A mov ebx, 16819168
1005F9F00 sub esp, 4, qword t0 // 01005F9F push 0
1005F9F01 and qword t0, 0xFFFFFFFF, esp
1005F9F02 stm 0, , esp
1005FA100 sub esp, 4, qword t0 // 01005FA1 push 39
1005FA101 and qword t0, 0xFFFFFFFF, esp
1005FA102 stm 0x27, , esp
1005FA300 sub esp, 4, qword t0 // 01005FA3 push 197
1005FA301 and qword t0, 0xFFFFFFFF, esp
1005FA302 stm 0xC5, , esp
1005FA800 sub esp, 4, qword t0 // 01005FA8 push edi
1005FA801 and qword t0, 0xFFFFFFFF, esp
1005FA802 stm edi, , esp
1005FA900 add 8, ebp, qword t0 // 01005FA9 push ss: [ebp + hDlg]
1005FA901 and qword t0, 0xFFFFFFFF, t1
1005FA902 ldm t1, , t2
1005FA903 sub esp, 4, qword t3
1005FA904 and qword t3, 0xFFFFFFFF, esp
1005FA905 stm t2, , esp
1005FAC00 sub esp, 4, qword t0 // 01005FAC call esi
1005FAC01 and qword t0, 0xFFFFFFFF, esp
1005FAC02 stm 0x1005FAE, , esp
1005FAC03 jcc 1, , esi

REIL Usage

Convert native code to REIL

Run REIL algorithm

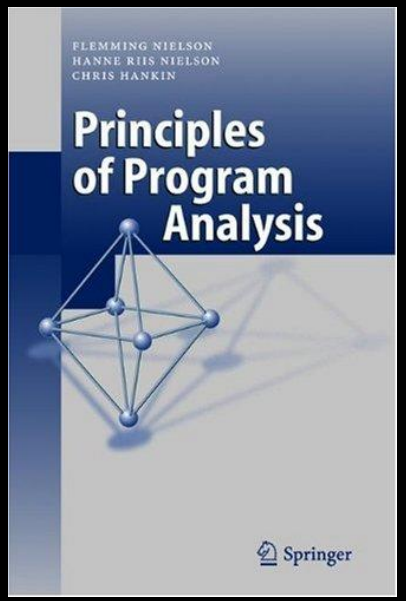
Port results back to original code

Advantages

- Easy to pick up and comprehend
- Reduces analysis complexity
- Write once; use everywhere

MonoREIL

- Monotone framework for REIL
- Simplifies analysis algorithm development
- Read the book



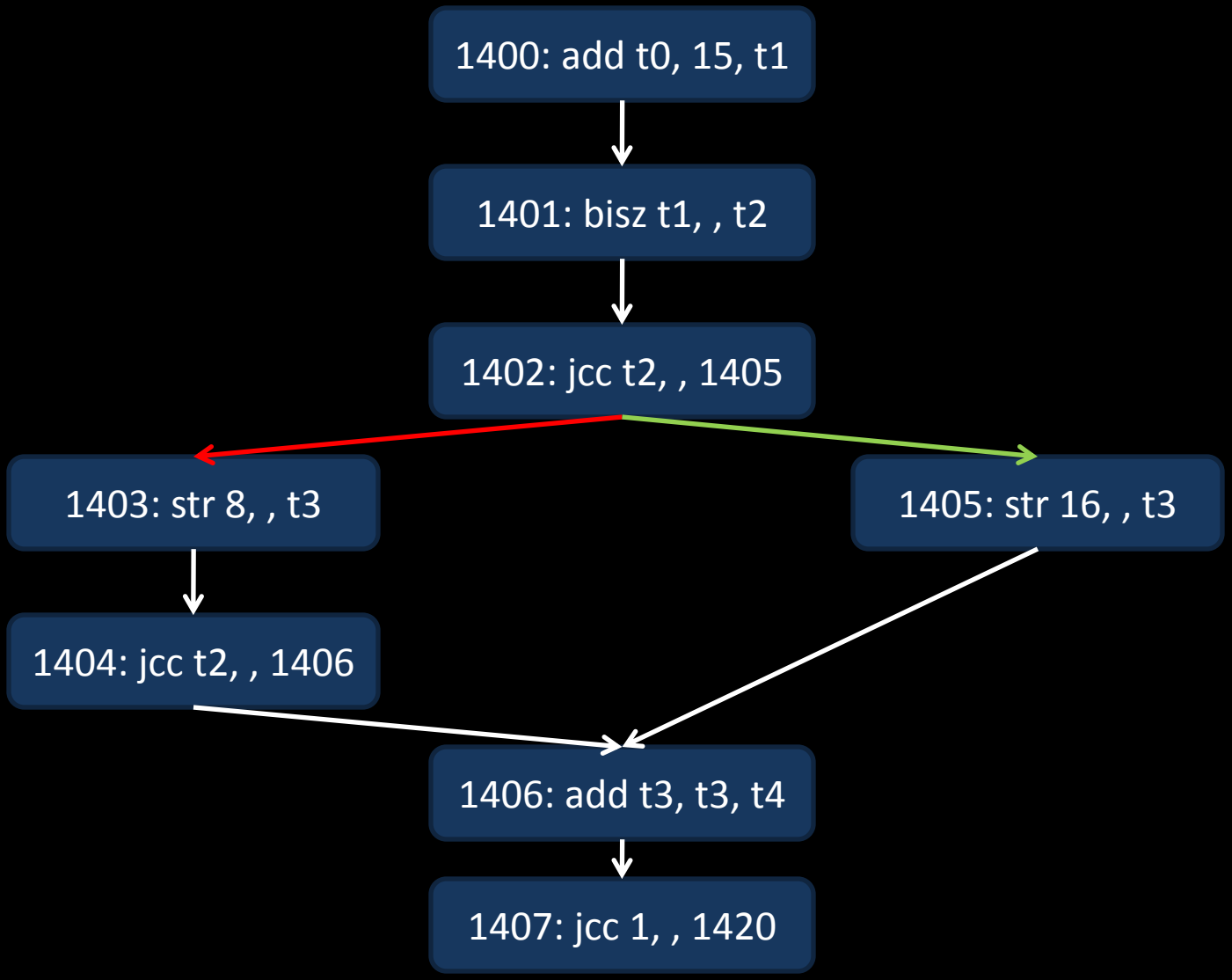
Advantages

- All algorithms have the same regular structure
- Simplifies algorithms
  - Trade-off: Runtime

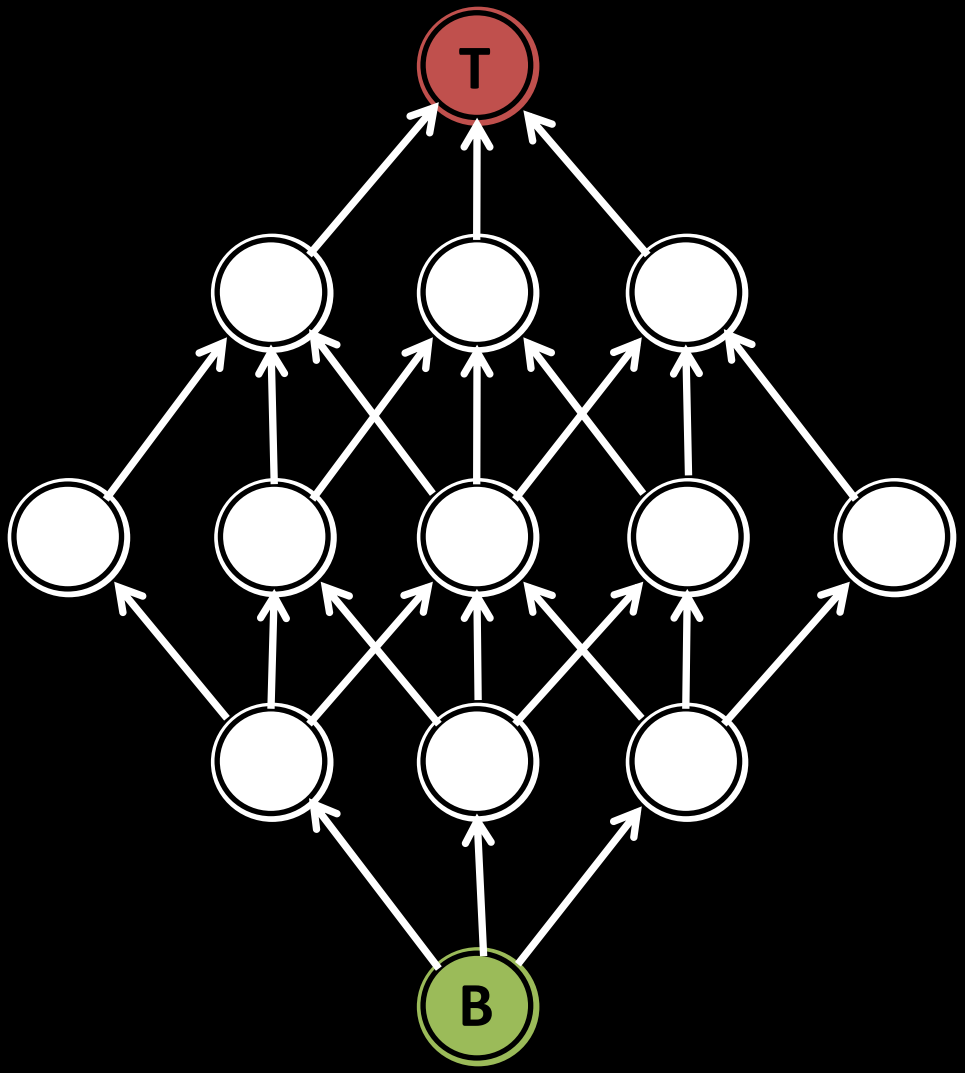
Core Concepts

- Instruction Graph
- Lattice
- Monotone Transformations

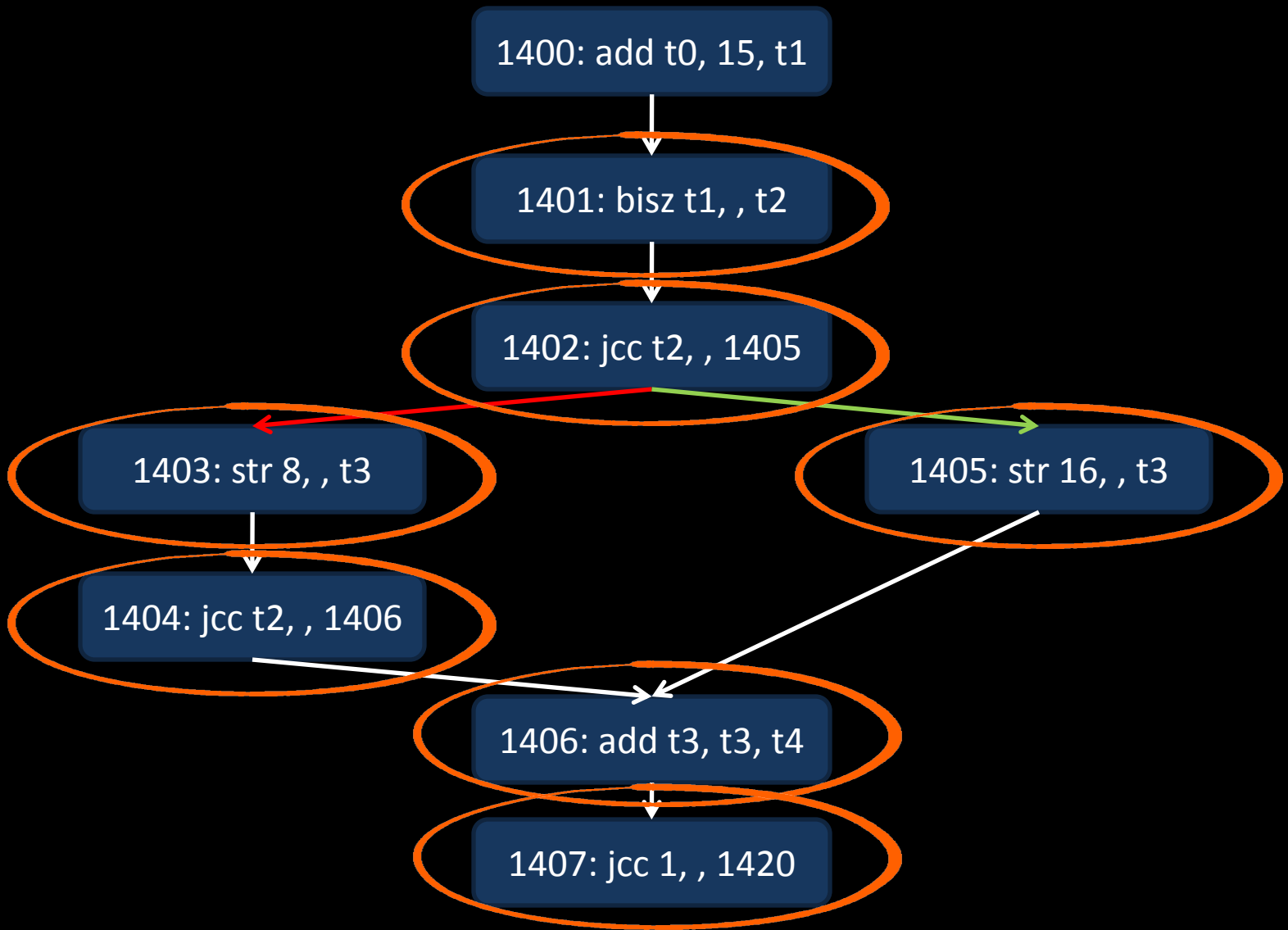
Instruction Graph



Lattice



Transformations



Applications

Register Tracking: Helps Reverse Engineers follow data flow through code
(Never officially presented)

Index Underflow Detection: Automatically find negative array accesses
(CanSecWest 2009, Vancouver)

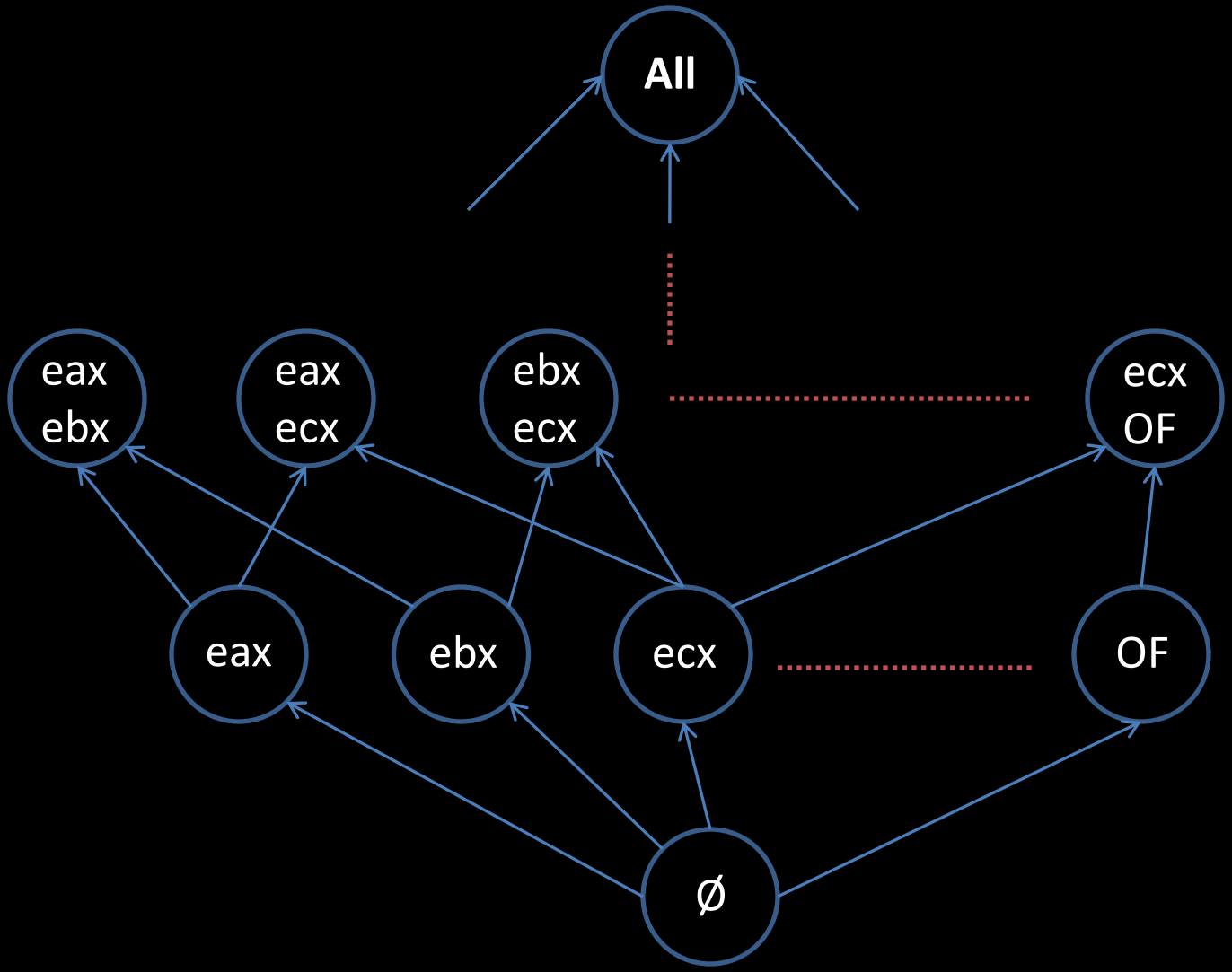
Automated Deobfuscation: Make obfuscated code more readable
(SOURCE Barcelona 2009, Barcelona)

ROP Gadget Generator: Automatically generates return-oriented shellcode
(Work in progress; scheduled for Q1/2010)

Register Tracking

- Follows interesting register values
- Keeps track of dependent values
- Transitive closure of the effects of a register on the program state

Lattice



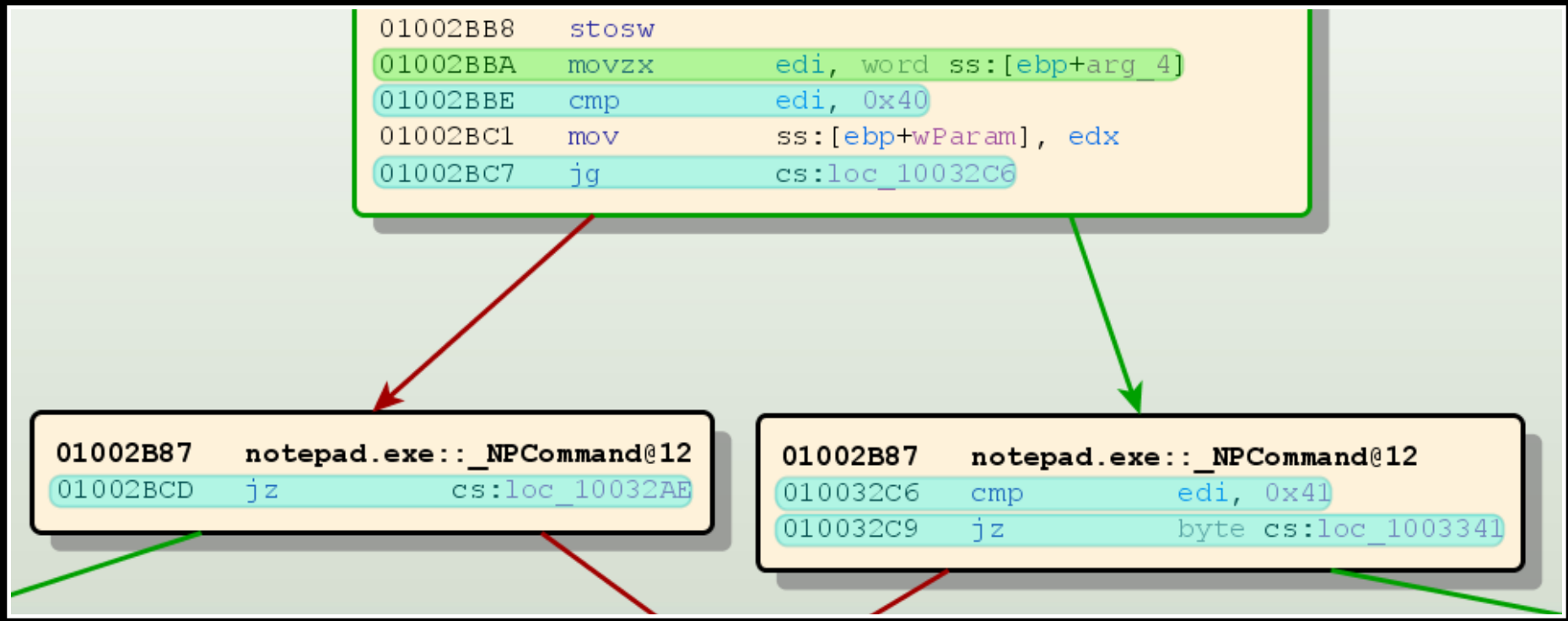
General Idea

- Start with the tracked register
- Follow the control flow
- Instruction uses register → Add modified registers to the tracked set
- Instruction clears register → Remove cleared register from the set

Example

```
{t0}
add  t0, 4, t1
{t0, t1}
bisz t2,  , CF
{t0, t1}
bisz t0,  , ZF
{t0, t1, ZF}
add  t2, 4, t1
{t0, ZF}
```

Result


01002BB8 stosw
01002BBA movzx edi, word ss:[ebp+arg_4]
01002BBE cmp edi, 0x40
01002BC1 mov ss:[ebp+wParam], edx
01002BC7 jg cs:loc_10032C6
01002B87 notepad.exe::_NPCommand@12
01002BCD jz cs:loc_10032AE
01002B87 notepad.exe::_NPCommand@12
010032C6 cmp edi, 0x41
010032C9 jz byte cs:loc_1003341

Use

- Fully integrated into BinNavi
- Makes it very simple to follow values
- Helps the reverse engineer to concentrate on what is important

Range Tracking

- Tracks potential ranges for register values
- Useful to detect buffer underflows like MS08-67
- Intervals are used to cut down on complexity

Lattice

- Complicated to show in a picture
- Keep track of register values and pointer dereferences as a list of ranges
- eax [0 .. 4] [0 .. 10]
  - Add a value between 0 and 10 to [eax], [eax + 1], [eax + 2], [eax + 3], or [eax + 4]

General Idea

- Track register values relative to their first use
- Follow the control flow
- Calculate maximum range of effects each instruction has on a register
- If the range gets negative for memory accesses, mark the location

Use

- Helps bug hunters to find potential vulnerabilities
- Automated and effective
- Not yet fully proven to work

Deobfuscation

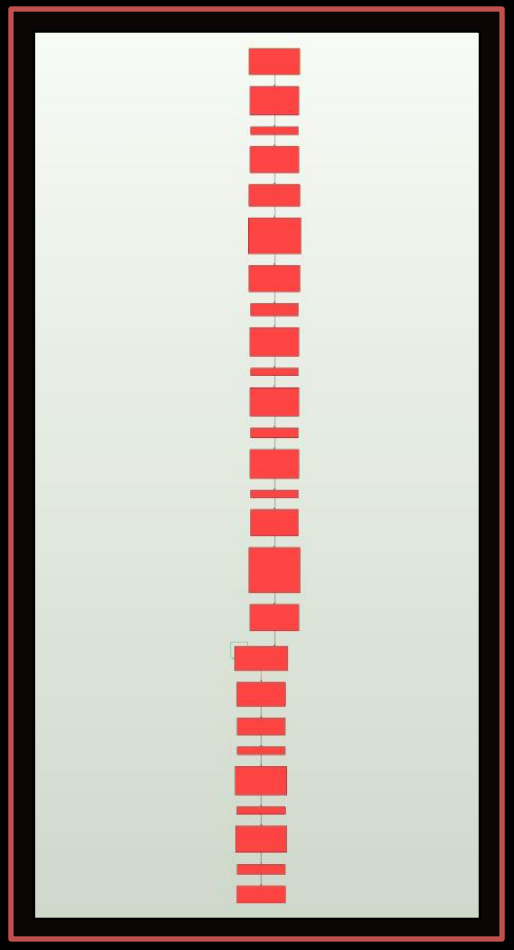
- Convert obfuscated code into something more readable
- Multi-process step with many lattices
  - Constant propagation
  - Dead code elimination
  - …

General Idea

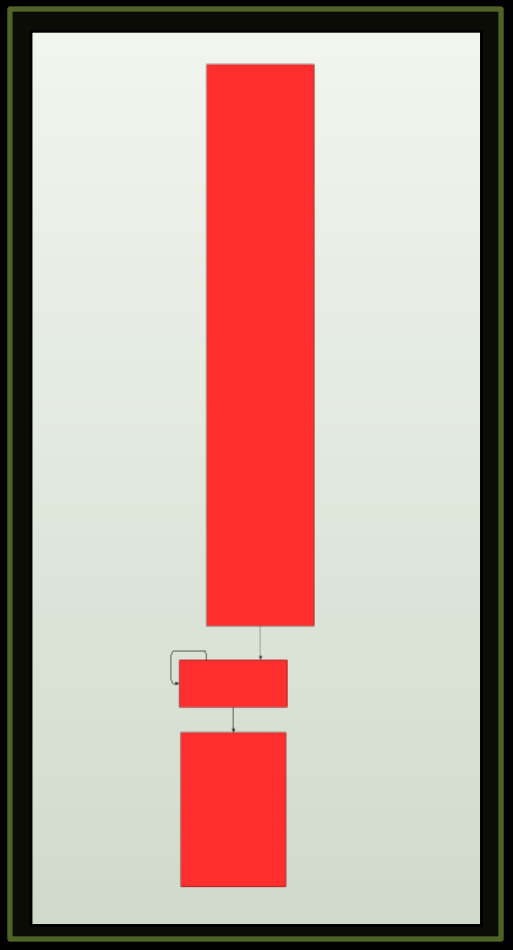
- Take a piece of code
- Apply the deobfuscation algorithms
- Repeat until no further deobfuscation is possible
- Result: Deobfuscated Code

Result

Before



After



Problems

- Turns out that deobfuscation is tricky for many reasons
- Further requirements:
  - Function that determines the readability of code
  - Backend that produces executable code from REIL

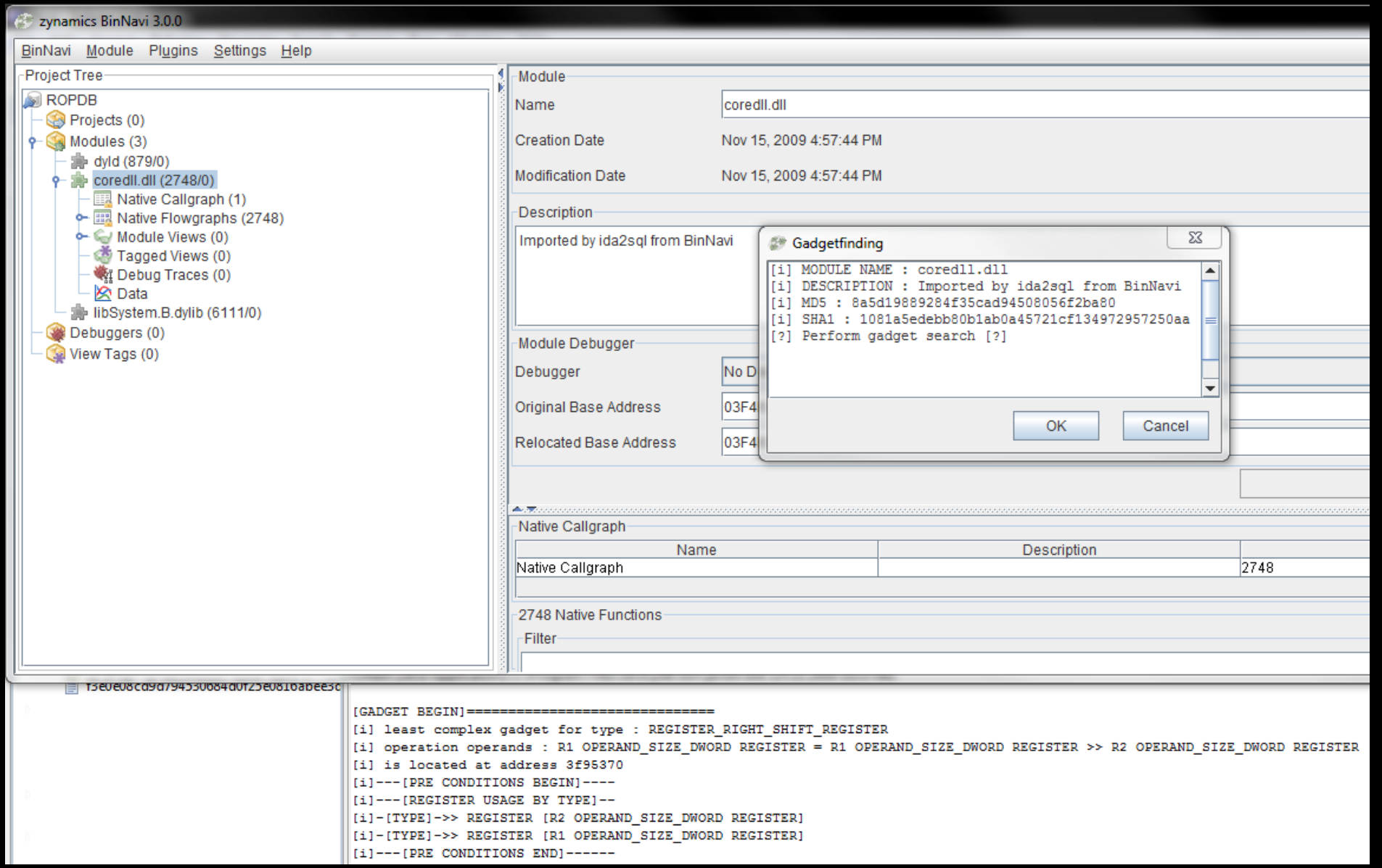
ROP Gadget Generator

- Return-oriented shellcode generator
- REIL-based but not MonoREIL-based
- Originally for Windows Mobile but platform-independent
- To be presented in 2010

General Idea

- Automated analysis of instruction sequences
- Automated extraction of useful instruction sequences
- Combines gadgets to shellcode
- Helps the development of return-oriented shellcode

Result

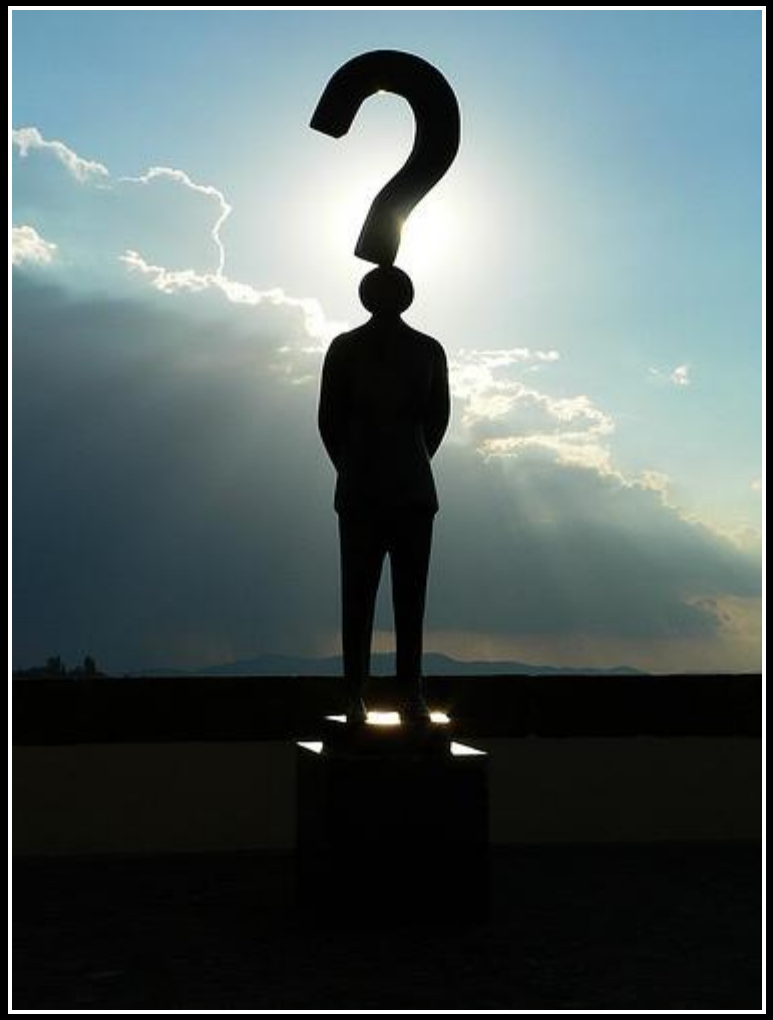

zynamics BinNavi 3.0.0
BinNavi Module Plugins Settings Help
Project Tree
ROPDB
Projects (0)
Modules (3)
dyld (879/0)
coredll.dll (2748/0)
Native Callgraph (1)
Native Flowgraphs (2748)
Module Views (0)
Tagged Views (0)
Debug Traces (0)
Data
libSystem.B.dylib (6111/0)
Debuggers (0)
View Tags (0)
Module
Name
coredll.dll
Creation Date
Nov 15, 2009 4:57:44 PM
Modification Date
Nov 15, 2009 4:57:44 PM
Description
Imported by ida2sql from BinNavi
Module Debugger
Debugger
Original Base Address
Relocated Base Address
Gadgetfinding
[i] MODULE NAME : coredll.dll
[i] DESCRIPTION : Imported by ida2sql from BinNavi
[i] MD5 : 8a5d19889284f35cad94508056f2ba80
[i] SHA1 : 1081a5edebb80b1ab0a45721cf134972957250aa
[?] Perform gadget search [?]
OK
Cancel
Native Callgraph
Name
Description
Native Callgraph
2748
2748 Native Functions
Filter
[GADGET BEGIN]==========
[i] least complex gadget for type : REGISTER_RIGHT_SHIFT_REGISTER
[i] operation operands : R1 OPERAND_SIZE_DWORD REGISTER = R1 OPERAND_SIZE_DWORD REGISTER >> R2 OPERAND_SIZE_DWORD REGISTER
[i] is located at address 3f95370
[i]---[PRE CONDITIONS BEGIN]----
[i]---[REGISTER USAGE BY TYPE]--
[i]-[TYPE]->> REGISTER [R2 OPERAND_SIZE_DWORD REGISTER]
[i]-[TYPE]->> REGISTER [R1 OPERAND_SIZE_DWORD REGISTER]
[i]---[PRE CONDITIONS END]------

Future Development

- BinAudit
  - Collection of algorithms for vulnerability research
- Type Reconstruction
  - Figuring out what higher level data types are stored in registers

Related Work

- ERESI Project
- BitBlaze
- Silvio Cesare



http://www.flickr.com/photos/marcobellucci/3534516458/