Ghidra To The Next Level

Ding Zhanzhao (anciety)

Portal Lab, StarCross Tech.
About Me

- Ding Zhanzhao | Anciety @StarCross PortalLab
- r3kapig CTF player (pwn/re) 2017-2021
  - Team Leader since 2019
  - DEFCON Final Qualified 2017-2021
  - 0CTF 1st
  - WCTF 3rd
  - ...
- Open Source Dev
- Rev/Bin analysis tool enthusiast
Content

• How current ghidra works
• BinCraft
  • Dedication
  • Things Done
  • Things About to Be Done
  • Lessons Learned
What are needed in reversing?

F5
What are needed in reversing?

• Static Analysis
  • Read assembly: disassembly
  • Read C-like Code: decompilation
    • And those that improve decompilation

• Dynamic Analysis
  • Emulation
  • Debugging
What we have now?

• IDA: the ruler
• Ghidra: open sourced by NSA since 2019
• Binary Ninja: still improving
• Rev.ng?
• Radare2?
About Ghidra

- Open Sourced in 2019
  - written in Java + Native Lang (C++)
- (almost) Feature Complete
  - Decompiler √
  - Graphs √
  - Debugger √
  - extensible: Scripting/Plugin √
- Great Architecture: sleigh DSL, etc.
  - Extensible lifting (translate binary => IR + assembly)
- Only One in Open Source World that competes with IDA
About Ghidra

• Problems?
• Historical (Code, Experience)
About Ghidra

Use C99 types in decompiler #2880

XVilka wants to merge 1 commit into NationalSecurityAgency:master from XVilka:fix-types

Conversation 2  Commits 1  Checks 0  Files changed 1

XVilka commented on 30 Mar

I suppose this part was written when C/C++ compilers were not as good as today. These days, even MSVC supports C99 standard, thus lets reduce the code churn and easier future porting to new platforms.

👍 1
About Ghidra
BinCraft - Dedication

• Ghidra is nice already
• But we want more!
  • Modern experience
  • Modern features (apart from officially supported)
  • Modern Code (gradually)
BinCraft - Dedication (cont.)

• vs. Official Ghidra: improve while we use
  • better understanding of what WE need

• Why not just contribute to official?
  • Flavor Differs: decompiled code style, UI...
  • Tune the tool OUR way
  • Provide one more choice!
BinCraft - what

- A collection of ghidra related projects
- ghidracraft -- forked ghidra
- sleighcraft -- Rust API to sleigh processor
- pcodecraft -- API to abstract ghidra decompiler
ghidracraft - what done

• Decompilation minor improvements (style tuning)
• Pcode Patch feature
• (Partially) Modernize UI style
About Decompilation

• Complex Code
• Complex Algorithm
• Complex Architecture
• C++
• DIFFICULT TO TUNE
Ghidra Decompiler

Ghidra (Java)

Launch process

Interact

Decompiler Plugin

Decompiler (C++) Executable
Ghidra Decompiler C++

Global Context
Architecture
Protocol
Actions
Funcdata
Intra-function data
Ghidra Decompiler C++

- IR——Pcode
- Variables: Varnode
  Address Space: register, ram (memory), unique (temp), const
  offset
  size
  E.g: (register, 0x0, 0x8) -> rax; (register, 0x0, 0x4) -> eax;
- Operations
  Arith: INT_ADD, INT_SUB......
  Control flow: BRANCH, CBRANCH......
  Special: MULTIEQUAL (phi-node), INDIRECT......
Ghidra Decompiler C++

1. Determine entry
2. Gen pcode
3. Gen CFG
4. Check sub func
5. Adjust pcode
6. Gen SSA
7. dce
8. Local type inf.
9. Expr rewrite
10. Adjust CFG
11. Control flow structure
13. Exit SSA, merge var
14. Determine expr/temp var
15. Merge var (phase 2)
16. Add casting
17. Create func proto
18. Select var name
19. Final control flow structure
20. Gen C token
Problems in Decompilation

- Magic functions: CONCAT, ZEXT...
- Lengthy default var names
- Analysis failure
- Other minor issues
  - display format convert...
Magic Functions

- Reason: Pcode not translatable
- Format: CONCAT44 -> concat 4 bytes and 4 bytes into 8 bytes
- Better way? type casting
- Add rewrite rule

```c
__ptr = (void *)CONCAT44(uStack156, local_a0);
```

```c
__ptr = (void *)((uVar1 << 0x20) + local_a0);
```
Default Var names

```c
undefined4 main(undefined4 param_1,undefined4 param_2)
{
    size_t Var1;
    int Var2;
    int iStack64;
    uint uStack60;
    byte abStack54 [30];
    undefined4 uStack24;
    undefined4 uStack20;
    undefined4 uStack16;

    uStack16 = 0;
    uStack20 = param_1;
    uStack24 = param_2;
    printf("plz input flag: ");
    __isoc99_scanf(&DAT_080488b1,abStack54);
}
```

```c
undefined4 main(undefined4 p1,undefined4 p2)
{
    size_t v1;
    int v2;
    int s4;
    uint s5;
    byte s3 [30];
    undefined4 s6;
    undefined4 s7;
    undefined4 s8;

    s8 = 0;
    s7 = p1;
    s6 = p2;
    printf("plz input flag: ");
    __isoc99_scanf(&DAT_080488b1,s3);
}
```
Analysis Failure

- DEFCON quals 2021 rad
- Stack Var failure
- Function Params Failure
Analysis Failure

- And rsp, X prevents stack var analysis
- Rsp+X => X constant propagated.
- If not, fails!
- Imperfect Solution: ignore
Analysis Failure

- Empty Params
- Rust use empty params when optimized
- Solution: ignore param info if Rust and optimized
- (merged)
Analysis Failure

- String display issue
Pcode Patch

- pcode patch: support scripting and **manual**!
Pcode Patch

- Vs IDA: microcode API
  - Not a “pass” (works on phase)
  - Support manual
Pcode Patch

• Use case
  • quickly modify program semantic (don’t care about padding)
  • Deobfuscation (without carry about byte length)
  • Total custom decompilation
    • Provides Pcode the IR
    • Gives you the decompiled code
Pcode Patch

- Manipulating Program Semantic Manually
Pcode Patch

- Use pcode patch against LLVM: x86, arm, aarch, mips, PPC
Pcode Patch

- Custom Decompilation: patch the IR, get the decompilation
  - Create a new segment
  - Patch your IR there
  - Get your decompilation and profit!

- Circumstances
  - get a weird instruction set, want the decompilation
  - Want to skip the sleigh engine (no binary!), get only the decompilation
Pcode Patch
Modernized UI

- Fix some classical stylings
- Add configurable color-setting
Modernized UI
What to be done

• GraalVM integration
Graal VM

• Oracle Open source Compiler framework
  • jitted language implementation (even LLVM)
  • general instrumentation framework
• To us?
  • jitted pcode emulation (vs. current impl)
  • instrumentation (and more! tainted analysis? symbolic execution?)
Graal VM

- Interpreter => jitted code
  - By Partial Evaluation
- What do we get?
  - Write pcode interpreter => jitted!
  - Oh, wait, instrumentation works?
Graal VM
Binary Analysis with GraalVM!

- Binary
- Pcode interpreter
- GrCUDA
- Ruby
- Python
- GraalVM LLVM
- Bitcode interpreter
- Truffle Language Implementation Framework
- GraalVM Compiler
- Java HotSpot VM
Graal VM

• Vs LLVM impl(sulong)
• Both have unstructured control flow
• Both low-level
• Follow!
  • Manuel Rigger, Matthias Grimmer, Hanspeter Mössenböck Sulong – Execution of LLVM-Based Languages on the JVM
  • Manuel Rigger, Matthias Grimmer, Christian Wimmer, Thomas Würthinger, Hanspeter Mössenböck Bringing low-level languages to the JVM: efficient execution of LLVM IR on Truffle
Graal VM impl. stages.

- Stage 0: refactor current emulation to allow reuse
- Stage 1: interpreter in Truffle
- Stage 2: tune the interpreter to allow jitting
  - Follow sulong (LLVM) implementation
- Stage 3: instrumentation framework
  - Encapsulation
  - tainted analysis
  - symbolic execution
Lessons Learned

• Ghidra is a huge beast
  • Problem: huge, sometimes doc missing
  • Advantage: complete
  • Needed: time and community
Thanks For Listening

Star us!

https://github.com/StarCrossPortal/bincraft
https://github.com/StarCrossPortal/ghidracraft