Open Source Ghidra

The First Few Months

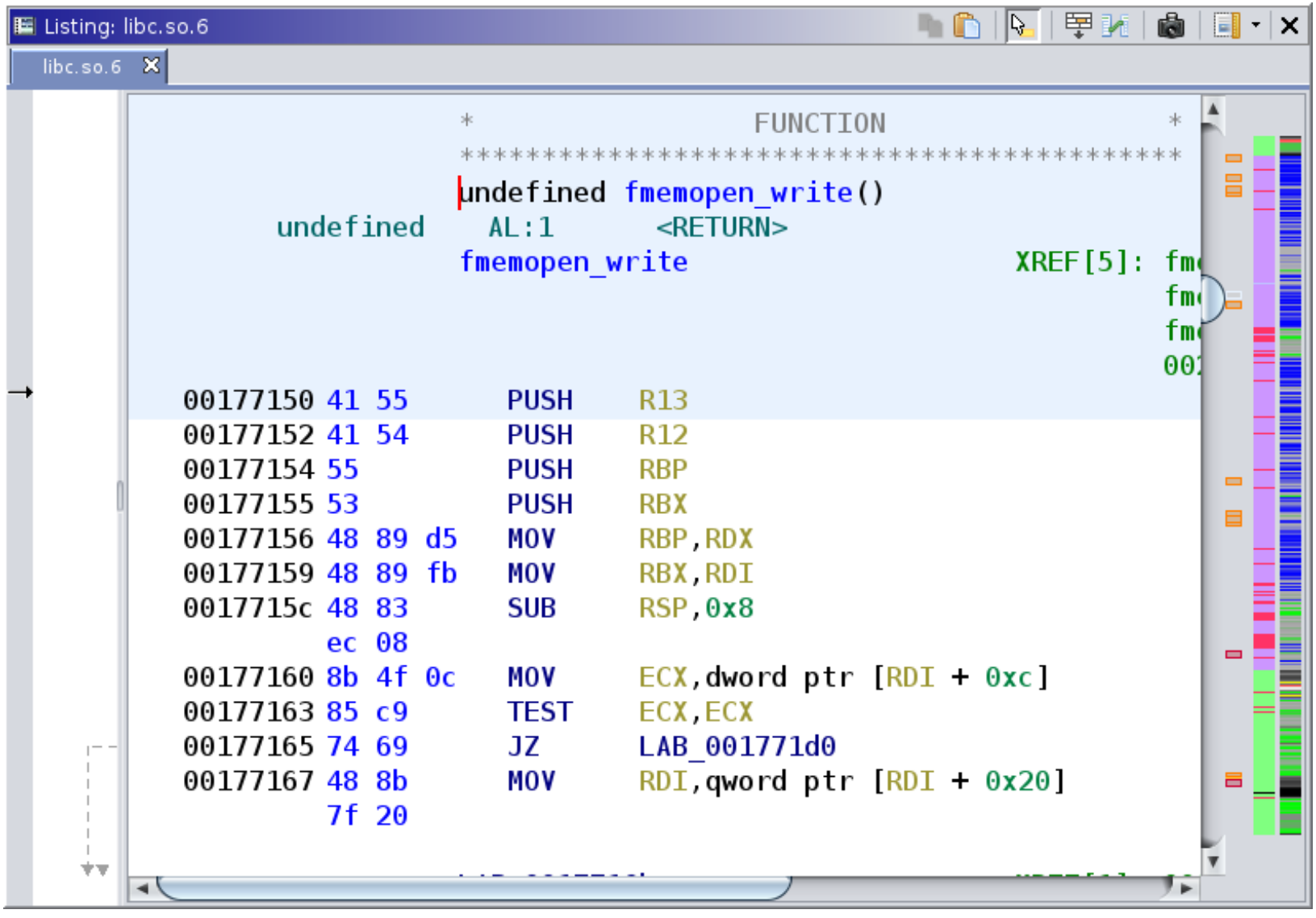
emteere
ghidracadabra

Recon MTL 2019

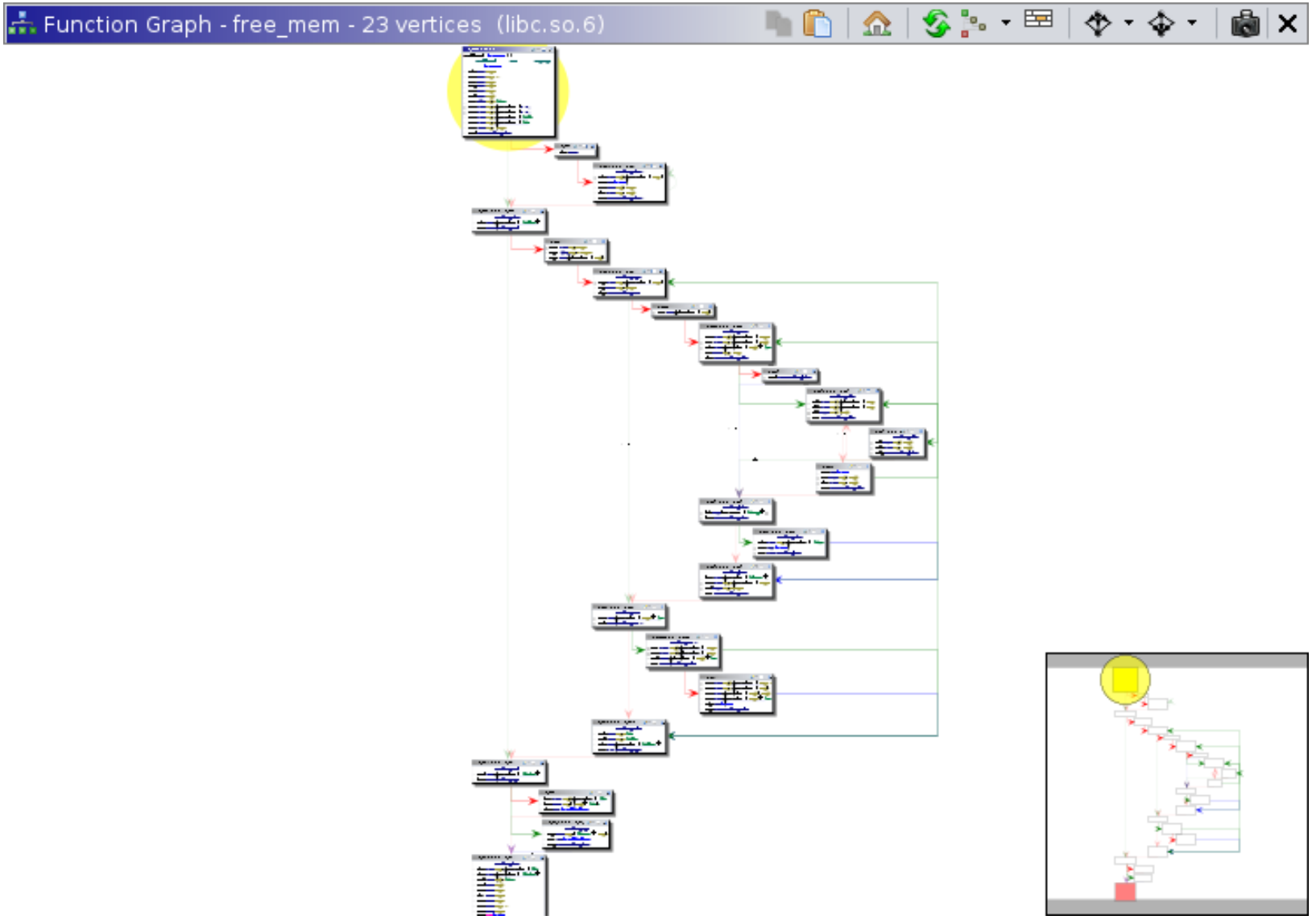
Outline

Ghidra Overview

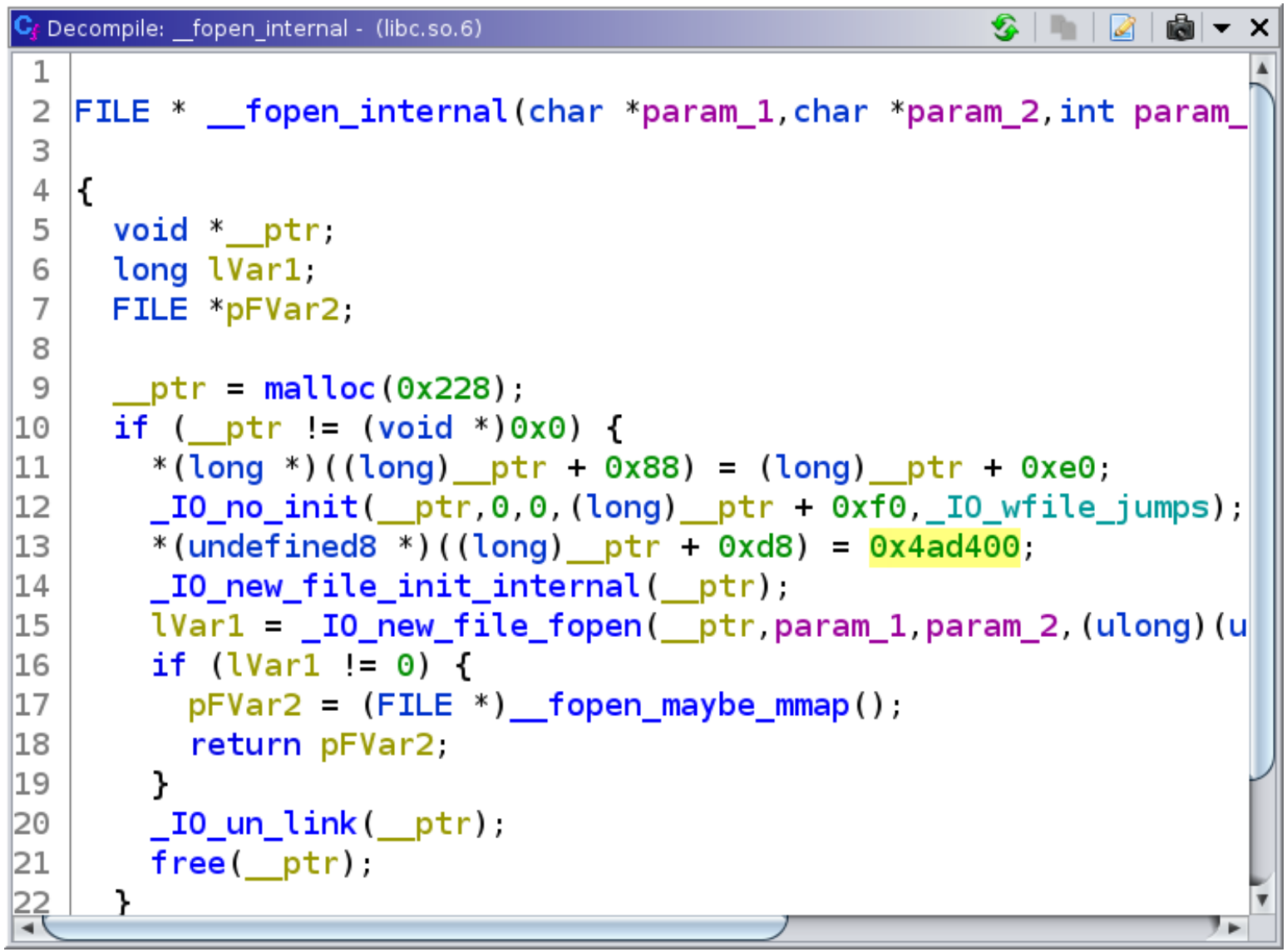
- Full-featured SRE framework created by NSA Research.
- In development for ~20 years.
- Primarily written in Java.
  - Some C/C++.
  - Can write scripts in Python.
- Designed for customizability and extensibility.
- Ghidra 9.0 publicly released March 2019.
- Source code released on Github April 2019.
- www.ghidra-sre.org
- https://github.com/NationalSecurityAgency/ghidra



Disassembler

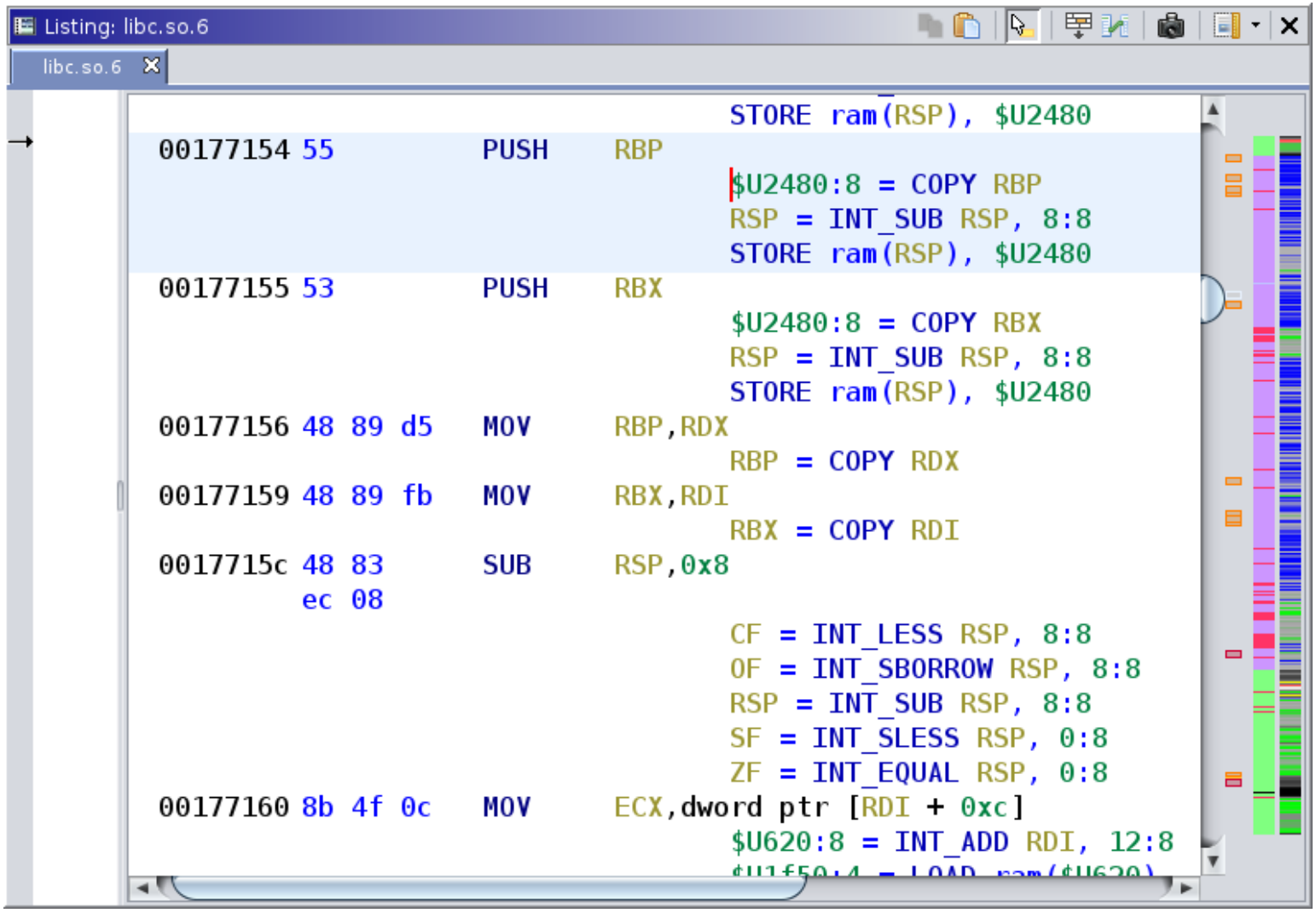


Function Graph

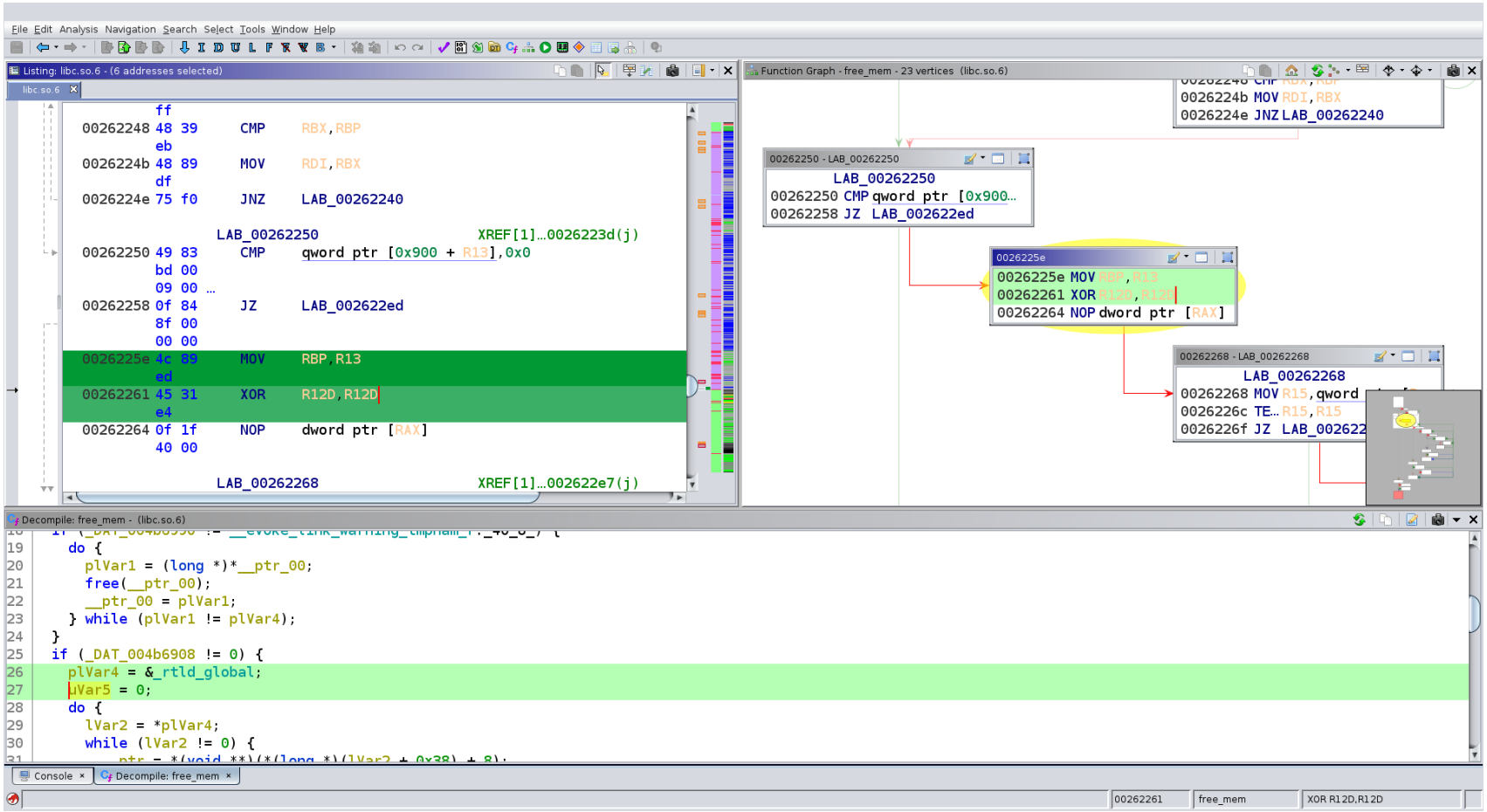

Decompile: __fopen_internal - (libc.so.6)
1
2 FILE * __fopen_internal(char *param_1,char *param_2,int param_
3
4 {
5 void *__ptr;
6 long lVar1;
7 FILE *pFVar2;
8
9 __ptr = malloc(0x228);
10 if (__ptr != (void *)0x0) {
11 *(long *)((long)__ptr + 0x88) = (long)__ptr + 0xe0;
12 _IO_no_init(__ptr,0,0,(long)__ptr + 0xf0,_IO_wfile_jumps);
13 *(undefined8 *)((long)__ptr + 0xd8) = 0x4ad400;
14 _IO_new_file_init_internal(__ptr);
15 lVar1 = _IO_new_file_fopen(__ptr,param_1,param_2,(ulong)(u
16 if (lVar1 != 0) {
17 pFVar2 = (FILE *)__fopen_maybe_mmap();
18 return pFVar2;
19 }
20 _IO_un_link(__ptr);
21 free(__ptr);
22 }

Decompiler

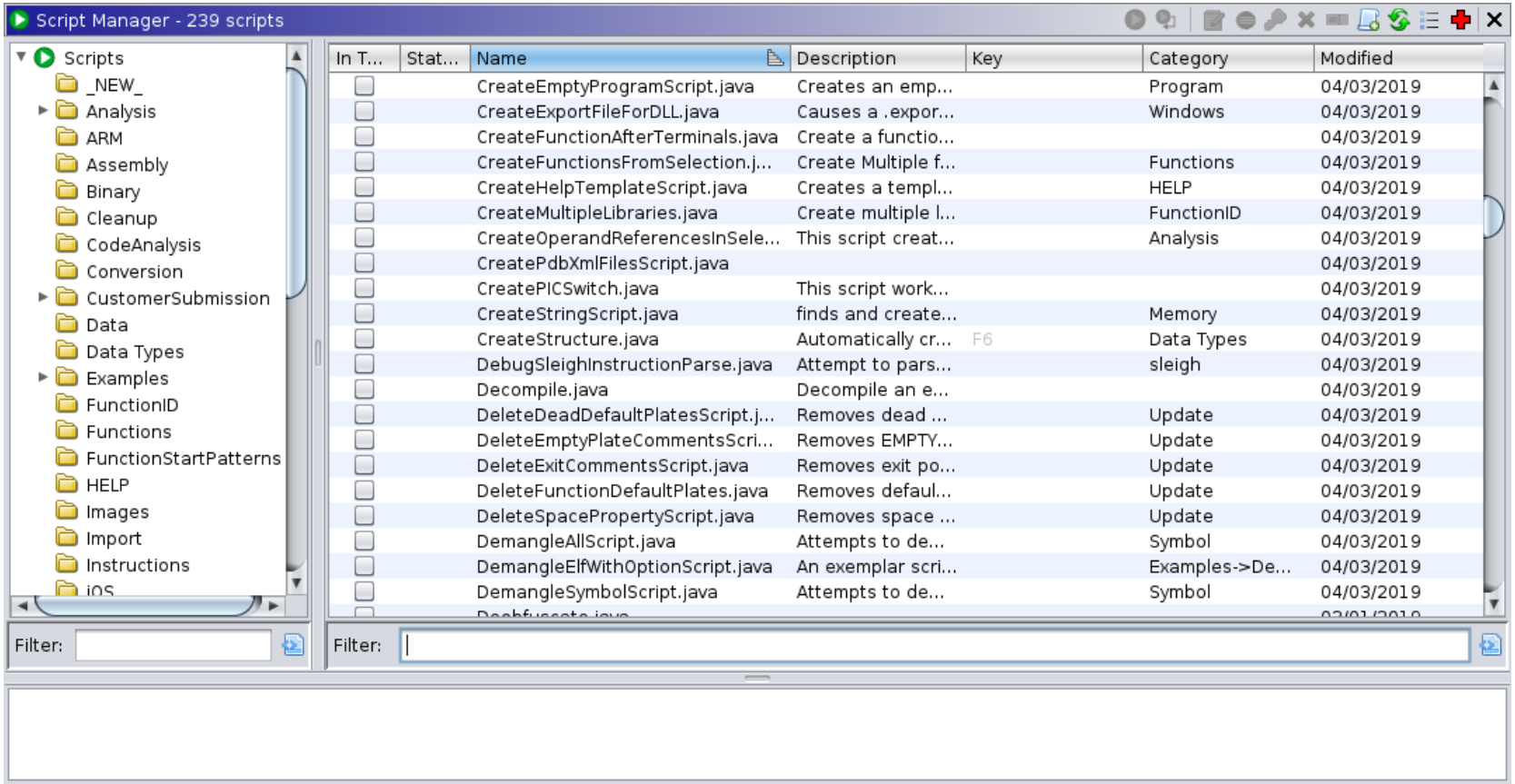

GHIDRA



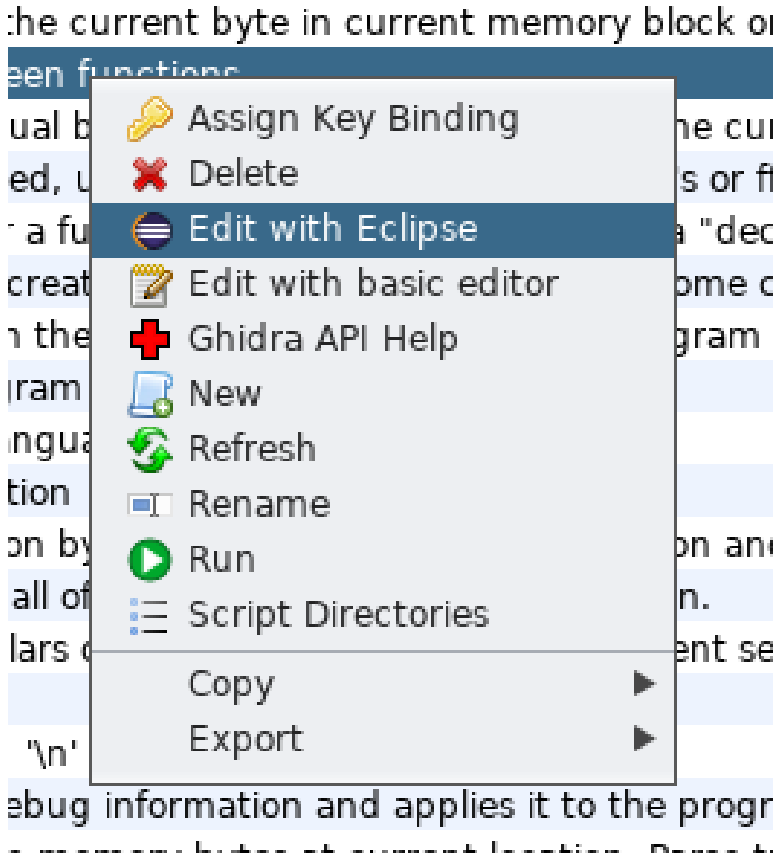
P-code: Ghidra's IR
Specified Using SLEIGH Language



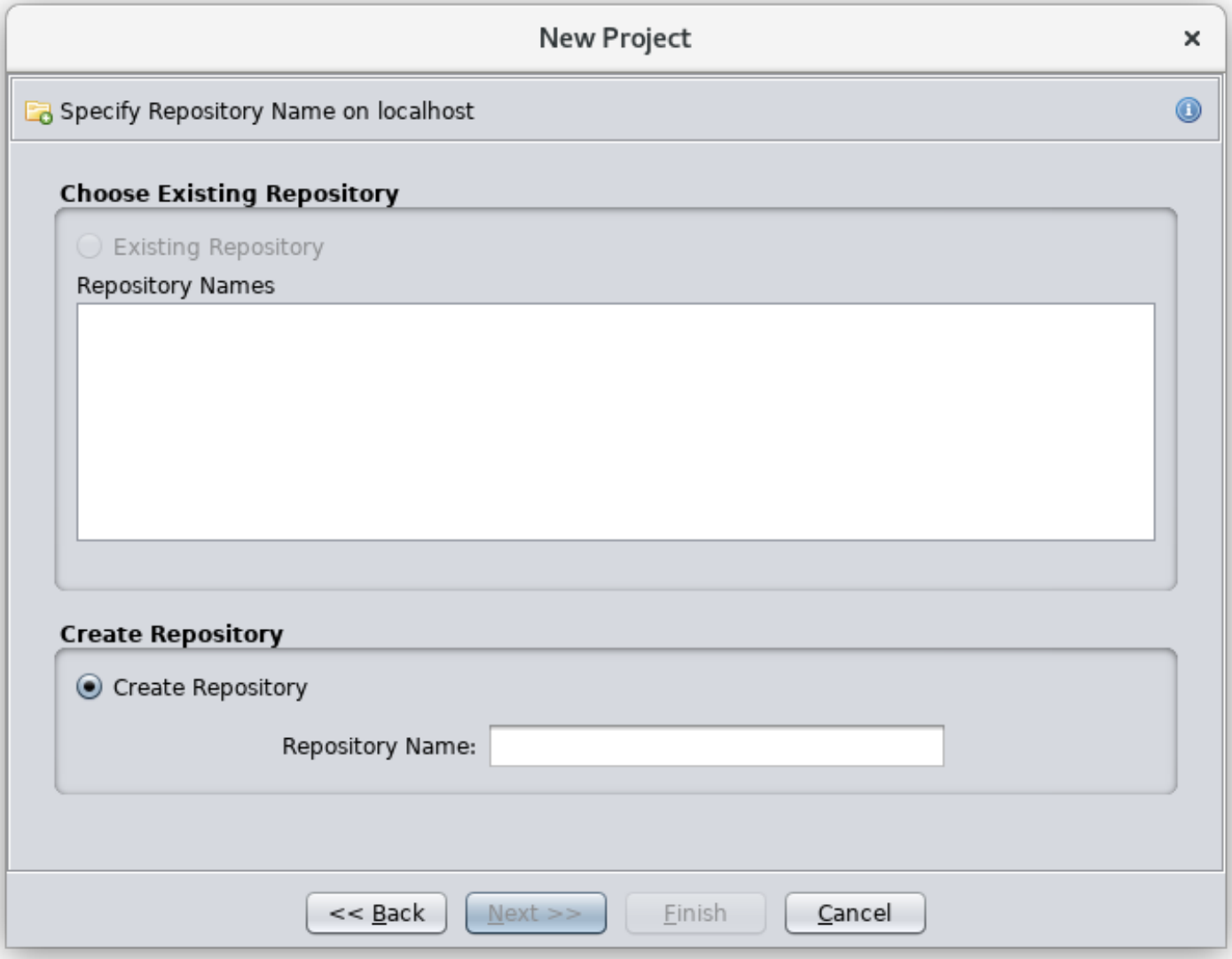
Connected Tools



Scripting in Java and Python



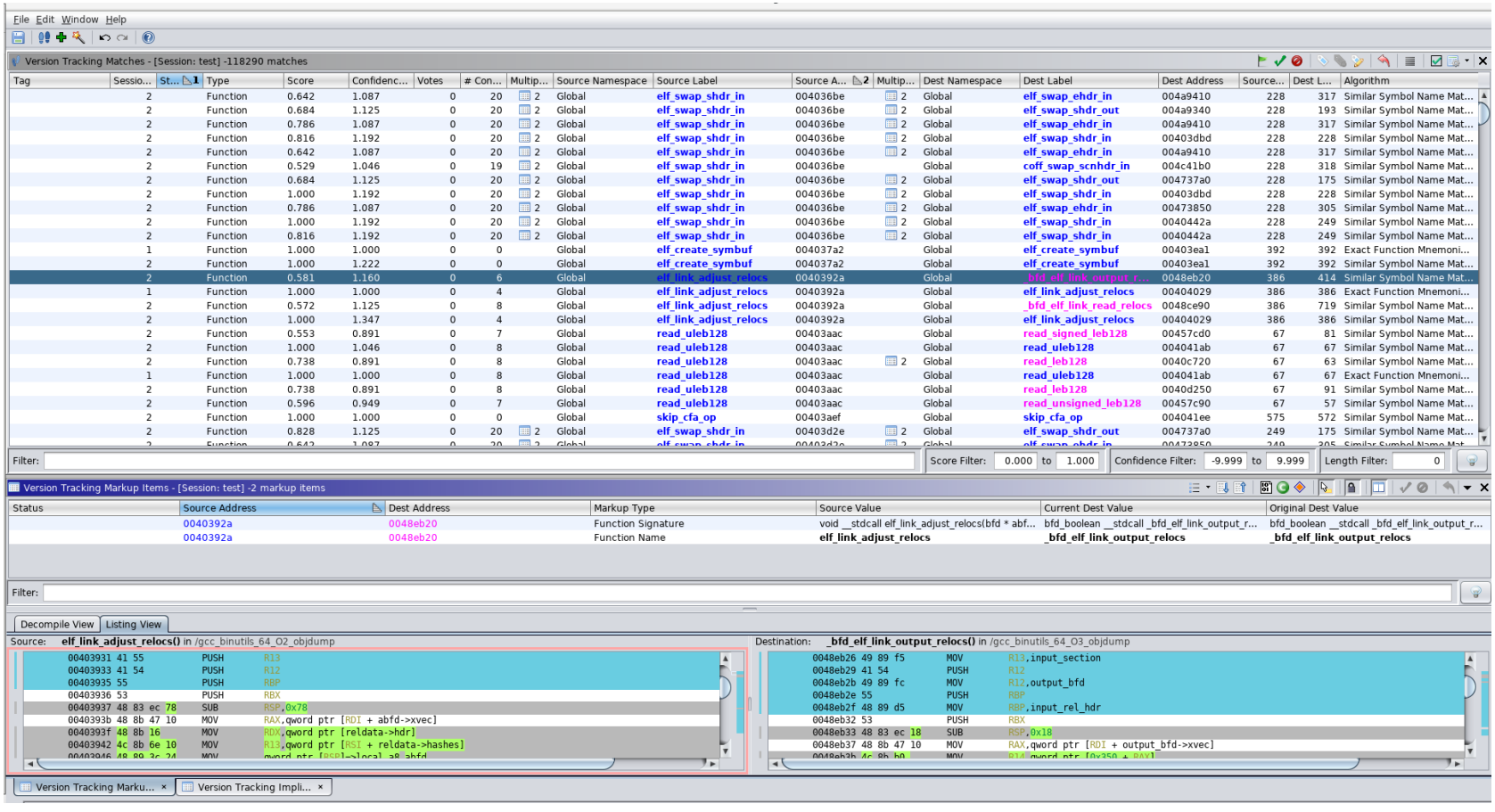
Eclipse Integration



Multi-user Server with Version Control

```
support]$ ./analyzeHeadless ghidra://localhost/repo
    -import /usr/bin/* -recursive
    -postScript MyScript.py
```

Batch Processing with the Headless Analyzer

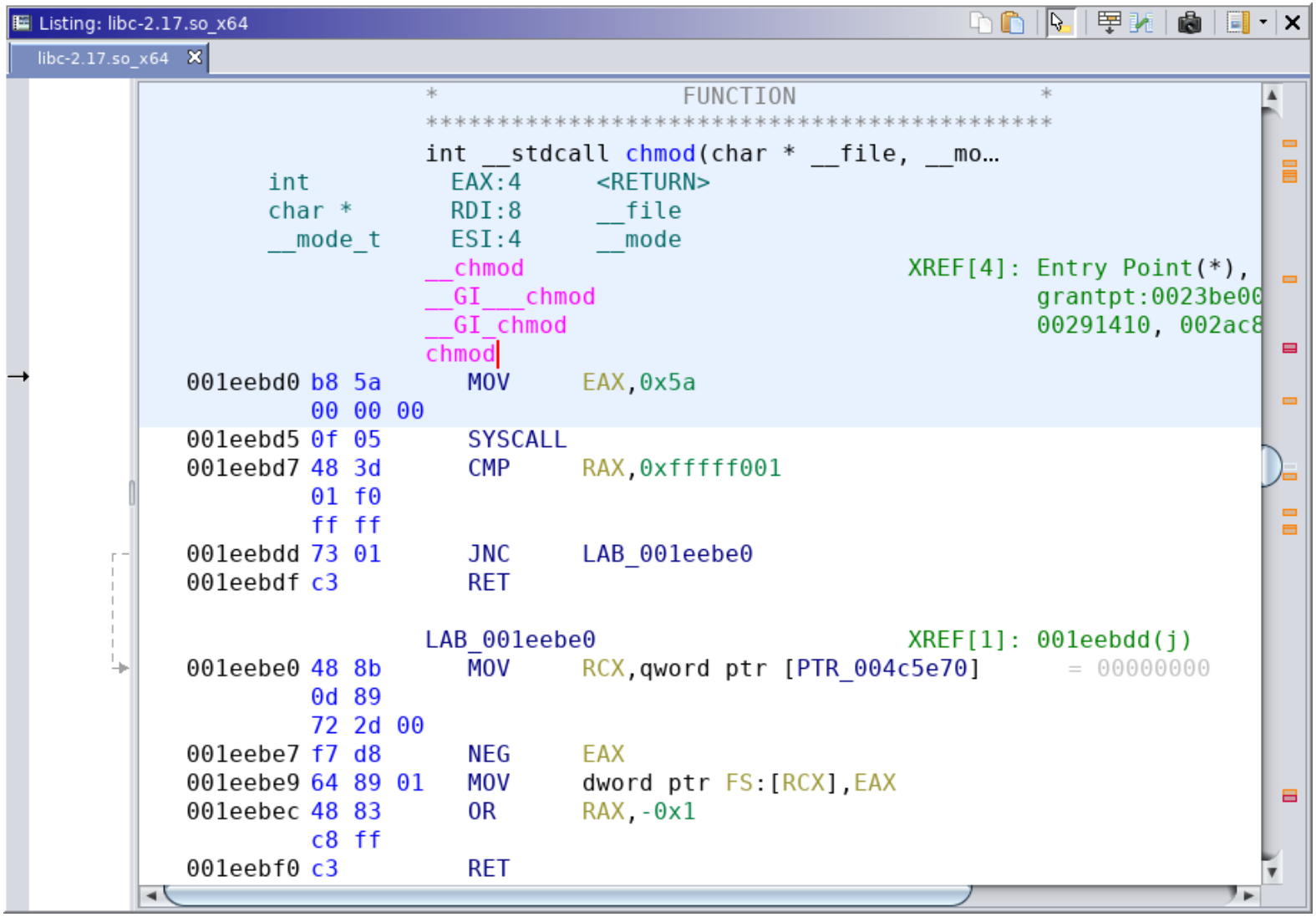


Version Tracking Tool

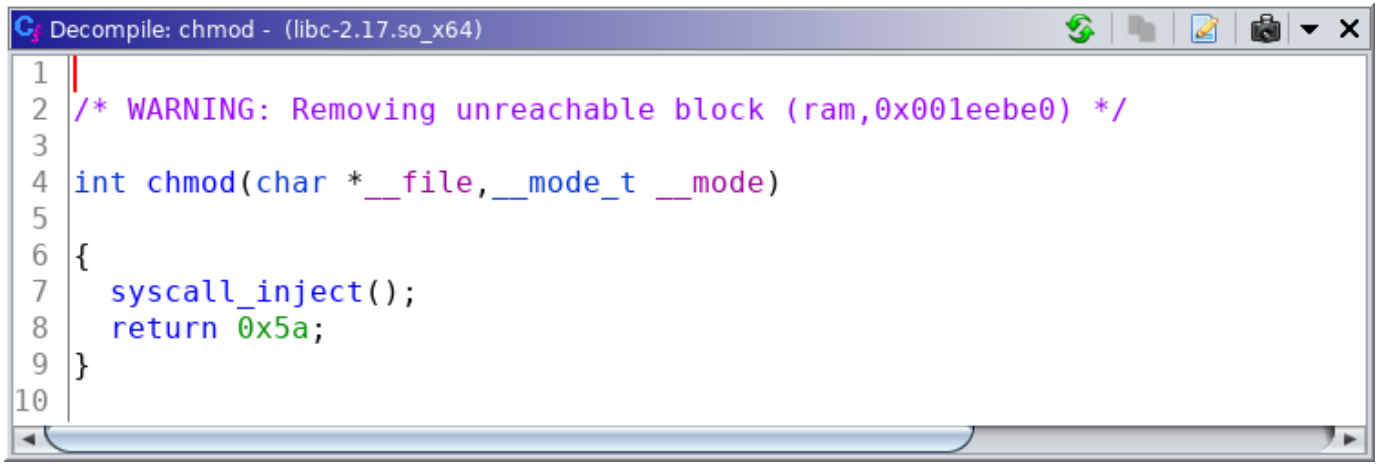
New Features for 9.1

Decompiling System Calls (syscalls)

- **System calls** are a way for a program to request a service from the operating system.
- Services include process control, file management, device management,...
- Typical implementation includes a native instruction and a register, which we'll call the **system call register**.
- When the instruction is executed, the value in the system call register determines which function is called.



x64 Linux syscall



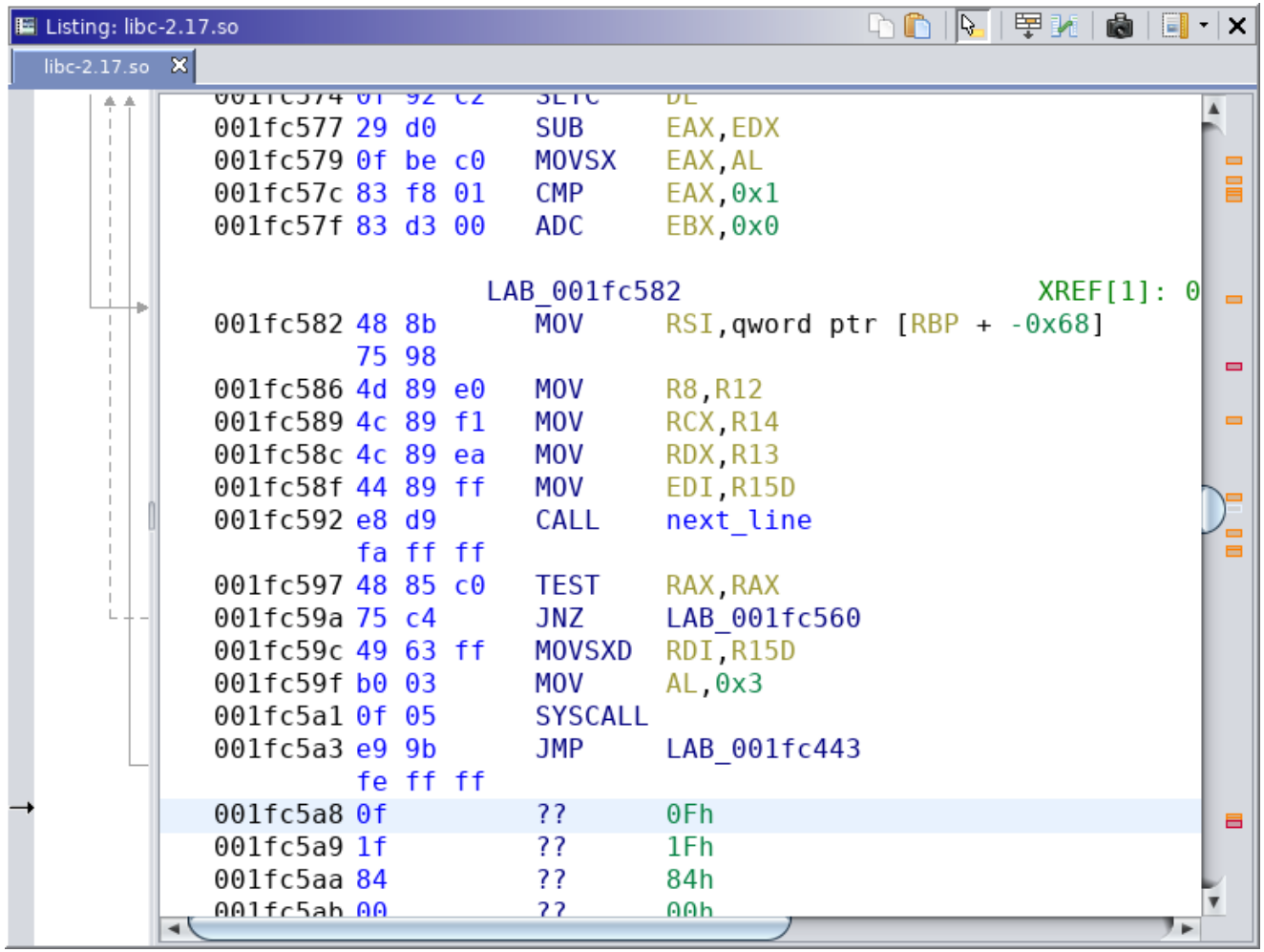
System Calls as User-defined Operations

- In this example, the `syscall` instruction implemented with a pcodeop/CALLOTHER
- Such operators certainly have their uses, but not very satisfying in this case.

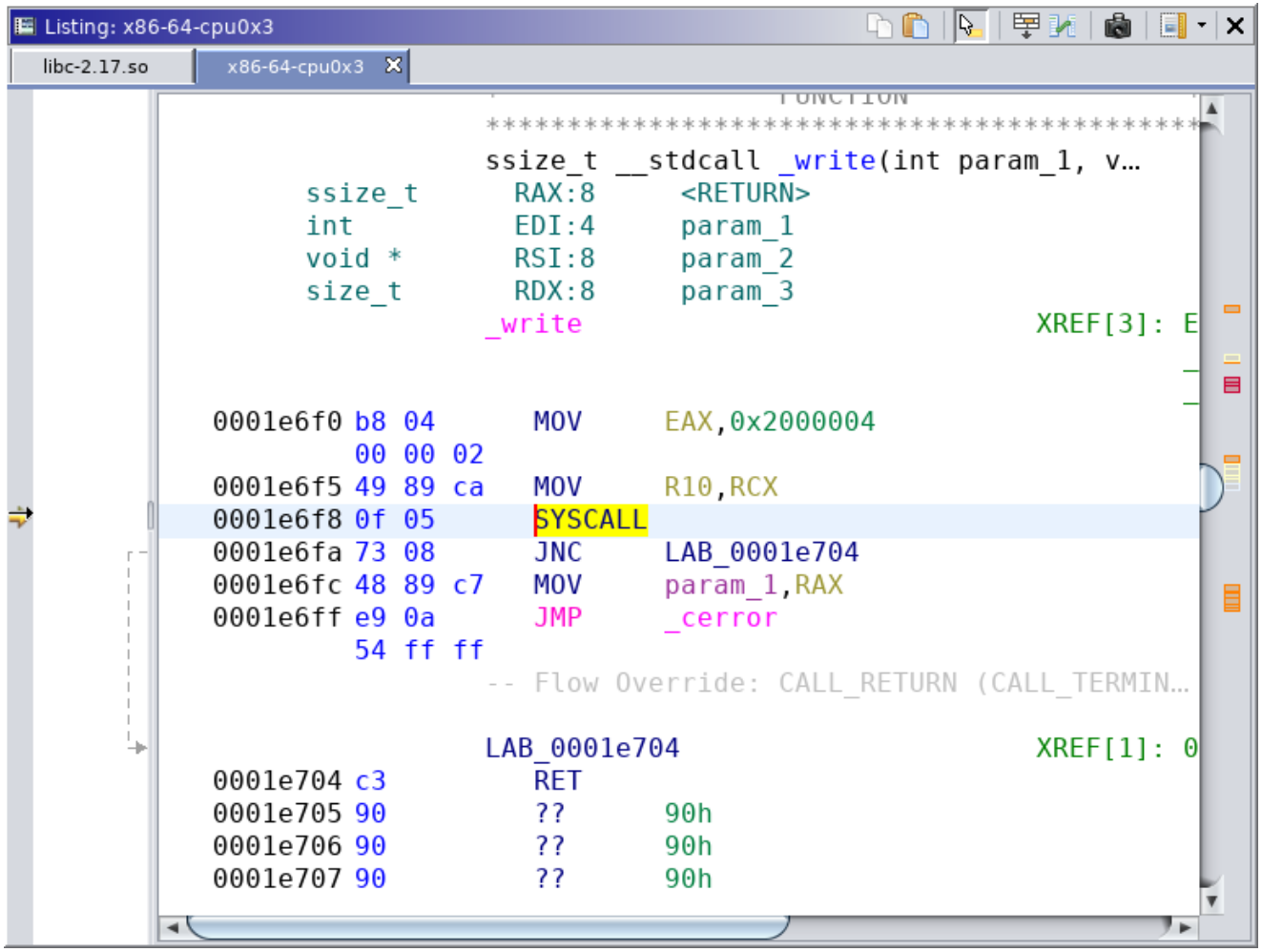
Desired Behavior

- We'd like to see the correct function call in the decompiler:
  - Correct name.
  - Correct signature.
  - Correct calling convention.
- We'd also like to get cross-references

- Need dataflow analysis to determine value in syscall register.



- Value in syscall register is not necessarily the syscall number defined in system header file.



Additional Issues

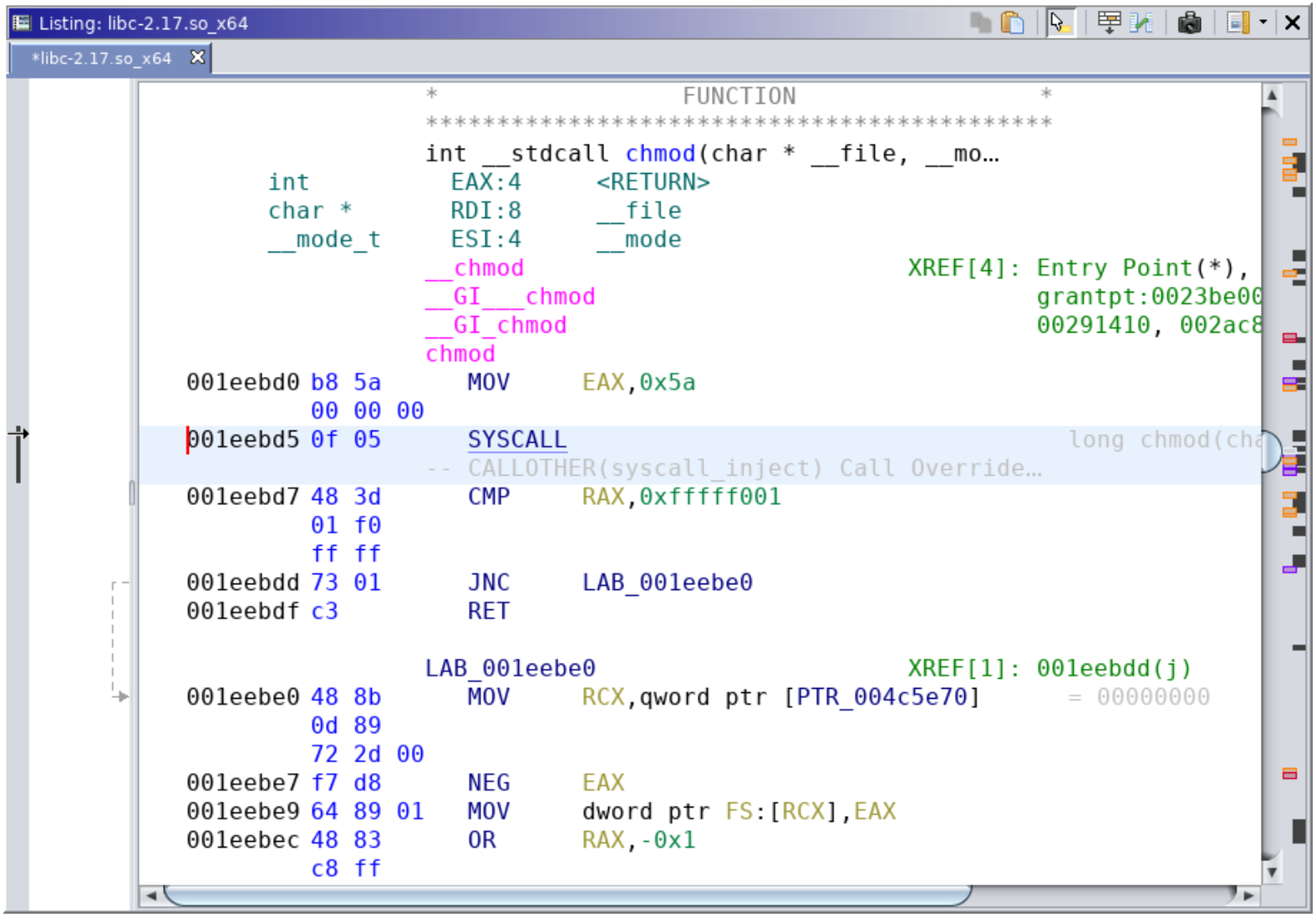
- The system call register can be an OS decision — not necessarily specified by ISA.
- System call numbers can change based on the OS version/service pack.
- System calls might have their own calling convention.
- There can be more than one native instruction used to make system calls (e.g., `syscall` and `int 2e`).
- Might not use a dedicated native system call instruction, e.g., system calls via `CALL GS:[0x10]`.

Where to Put Them?

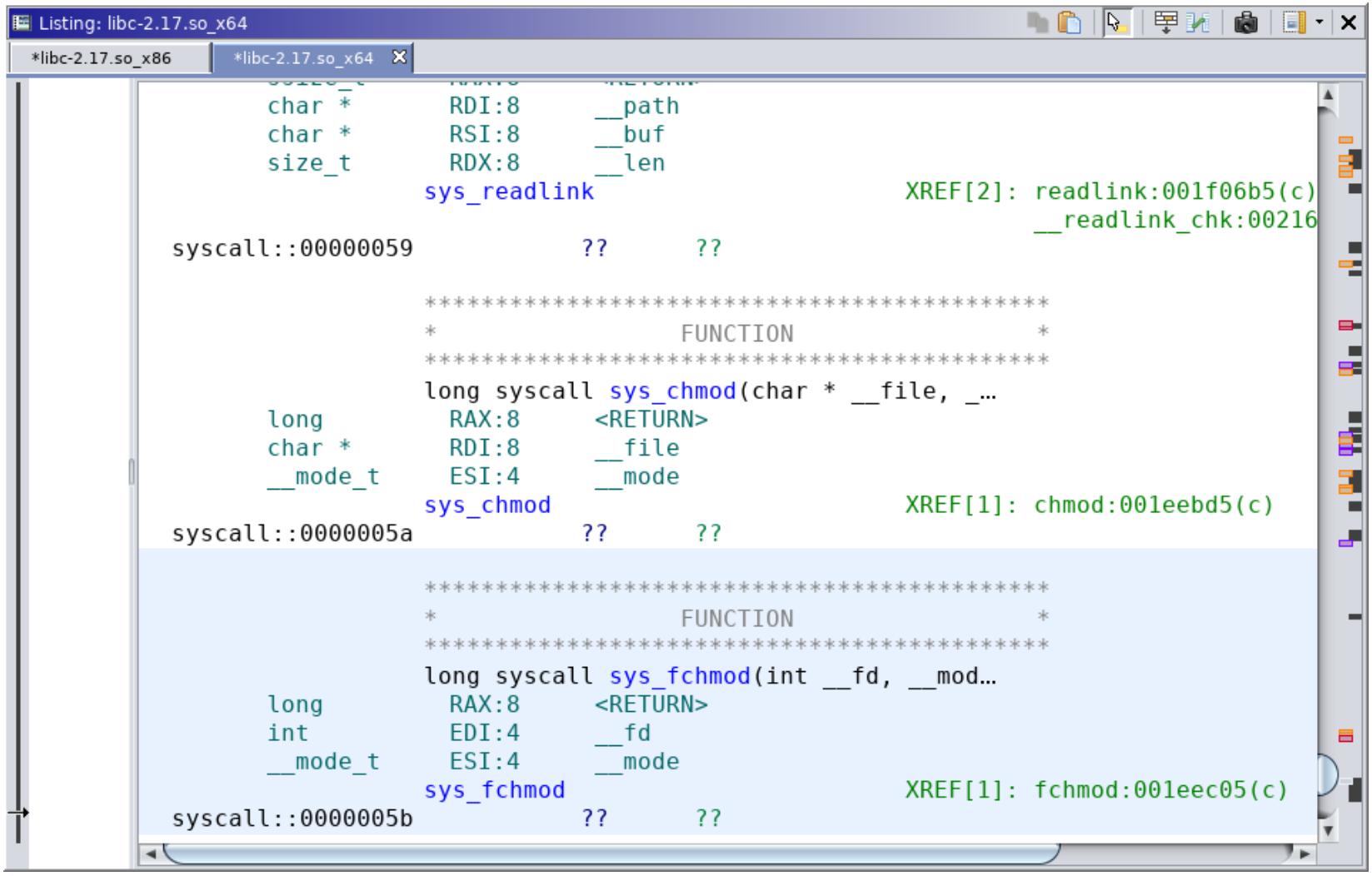
- In general, the code for system call targets is not in the program's address space.
- Where to put them in Ghidra?
- The OTHER space is used to store data from a binary that is not loaded into memory.
  - E.g., the `.comment` section of an ELF file.
- In 9.1, we've made the decompiler aware of the OTHER space.
- Recommendation for system calls:
  - System call target should be in overlay(s) of the OTHER space.
  - Use the system call number as the address in the overlay.

How to Get There?

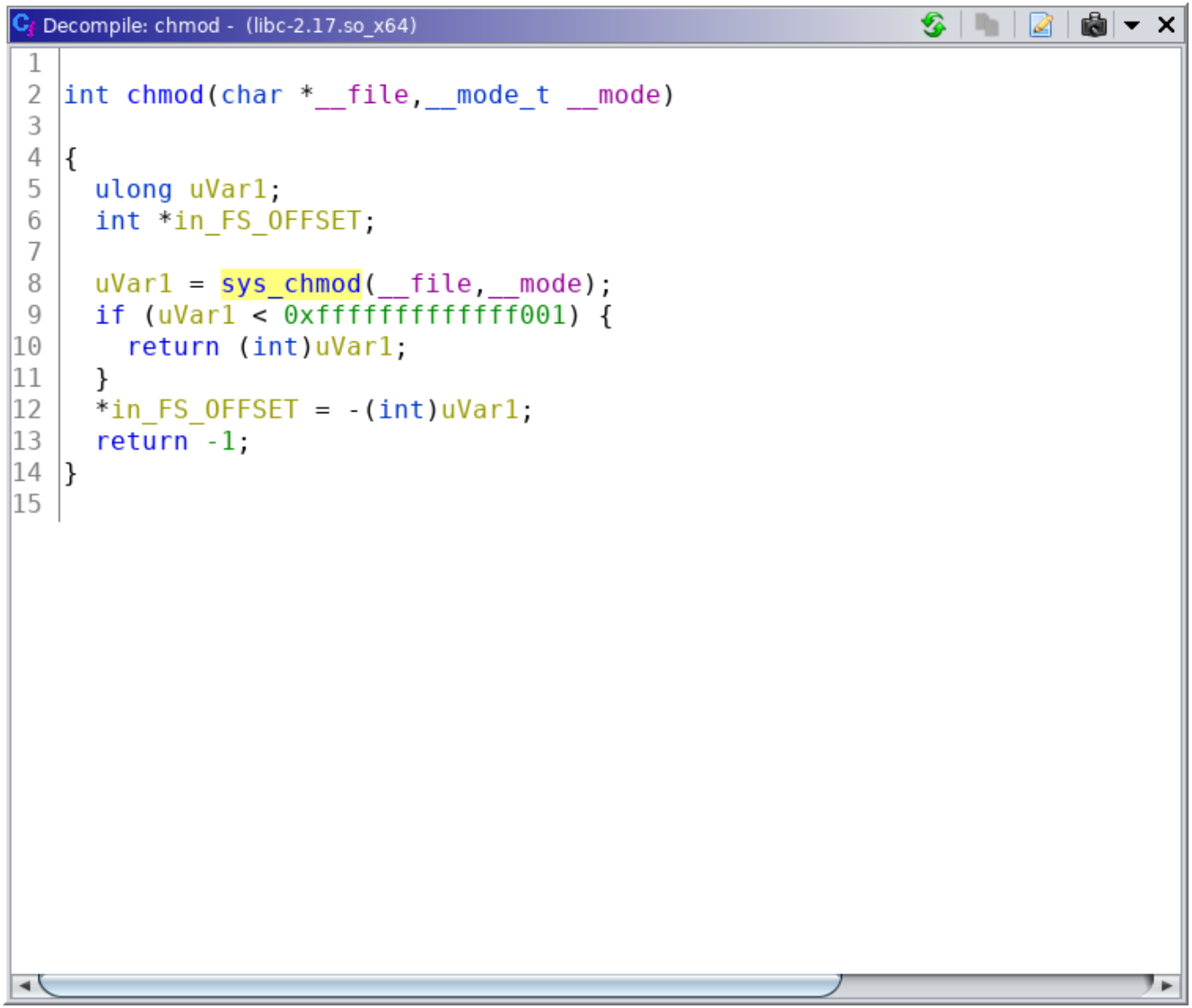
- OK, great, we have a place for system call targets.
- How do you get there?
- New feature: **Overriding References**.
- Basically, this allows you to intercept certain Pcode ops on their way to the decompiler and modify them.
  - Change `CALLOTHER` ops to `CALL` ops and set destination.
  - Change `CALLIND` to `CALL` ops and set destination.
  - (plus a few others)
- See `ResolveX86or64LinuxSystemCallsScript.java` for an example.



x64 Linux syscall with Overriding Reference



Functions in an Overlay of the OTHER Space



x64 Linux syscall Decompilation
Ghidra 9.1 (after running script)

Future Work

- We'd like an analyzer to be able to do this (mostly) automatically.
- Ghidra has a notion of per-processor configuration (`.pspec` files) and per-compiler configuration (`.cspec` files).
- System call data doesn't quite fit this model.
- Ideally all the system call related configuration would be in one place.
- Working on a notion of an OS/environment configuration.
- This will have other applications in Ghidra as well.

Sleigh Development Tools

- Sleigh
- SleighEditor
- Sleigh P-Code Tests
- Additional Techniques
- General Sleigh Development

Sleigh Processor Models

- Memory model
- Registers
- Display (printpiece)
- Decode patterns
- Semantics (Pcode)

```
define alignment=2;

define space ram       type=ram_space       size=2;
define space register  type=register_space  size=2;
define space rom       type=ram_space       size=3 wordsize=2 default;

#TOKENS
define token opbyte (8)
    imm8 = (0,7)
    oplo = (0,3)
    ophi = (4,7)
    Areg = (0,2)
    rn   = (0,2)
;

# Registers
define register offset=0xFF00 size=8 [ C OV Z X A ];
define register offset=0xFF00 size=4 [ d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13 d14 d15 ];

attach variables [rn] [d0 d1 d2 d3 d4 d5 d6 d7];

# macros
macro addflags(op1, op2) { C = carry(op1,op2); }
macro resultflags(op1, op2) { Z = op1 == 0; }

# Sub constructors - addressing modes
OP1: "#"imm8    is rn=2; imm8     { tmp:1 = imm8; export tmp; }
OP1: (imm8,X)   is rn=0 & X; imm8 { addr:2 = zext(imm8 + X); tmp:2 = *:2 addr; export *:1 tmp; }

Rel8:   relAddr  is imm8          [ relAddr=inst_next+imm8; ] { export *:1 relAddr; }

# Base constructors
:ADD A,rn  is ophi=2 & A & rn { addflags(A,rn); A = A + rn; resultflags(A); }
:JZ Rel8   is ophi=6 & oplo=0; Rel8 { if (A == 0) goto Rel8; }
```

Build it and the tools just work

Disassembly, Assembler(patch), Decompiler, Analysis...

Sleigh Processors

- Currently Included - evolving list

  X86 16/32/64, ARM/AARCH64, PowerPC 32/64/VLE, MIPS 16/32/64/micro 68xxx, Java / DEX bytecode, PA-RISC, PIC 12/16/17/18/24, Sparc 32/64 CR16C, Z80, 6502, 8051, MSP430, AVR8, AVR32, and variants.

- Full Processor Contributions

  Tricore, MCS-48

- Extensions, Improvements, and Bugs

  ARM, PPC, 68xxx, AVR, PIC-16F, PPC, 6502, golang

- Seen in Development

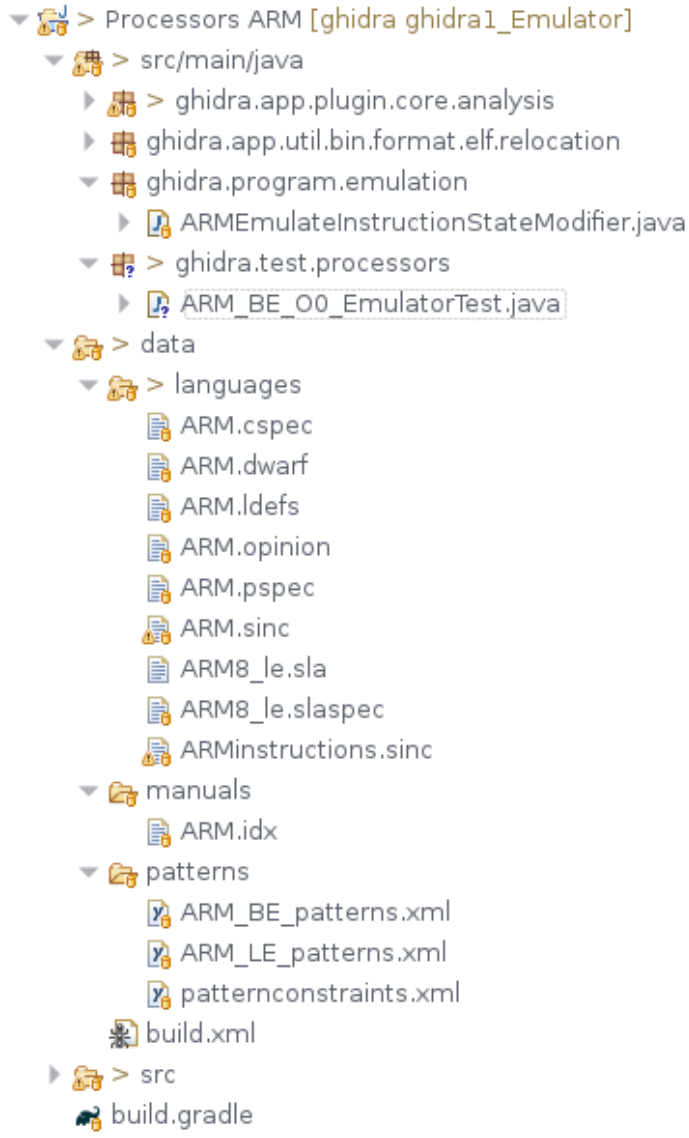
  SH-2, WebAssembly, Hexagon, Toshiba MeP-c4, Pic16F153xx, Arm4t-gba, NVIDIA Falcon, PowerPC 750CL/CXe, WDC-65816, RISC-V, TI TMS9900

Sleigh Files

- LDEF
- PSPEC
- CSPEC
- SLASPEC
- SLA

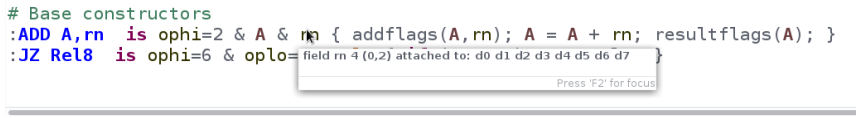
- Java Files
- Manual Index
- Pattern Files

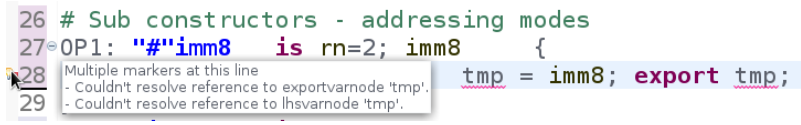
- Emulator^{new}
- Sleigh P-Code Tests^{new}

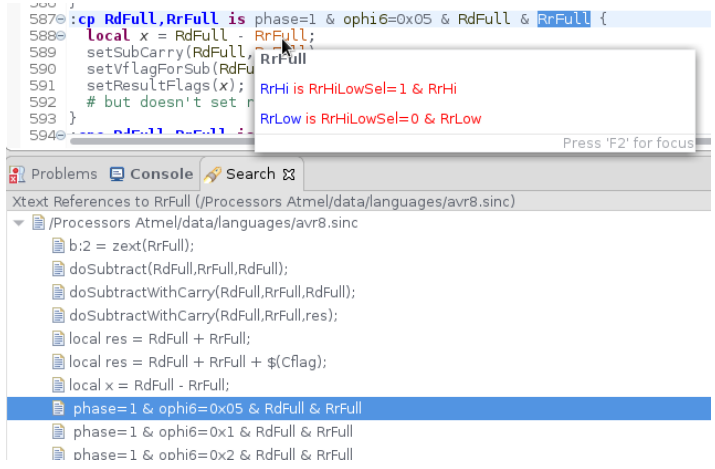


Sleigh Editor

- Syntax Coloring
- Hover
- Navigation
- Code Formatting
- Validation
- Quick Fixes
- Renaming
- Find References
- Content Assist
- Sleigh Compiler Error Navigation



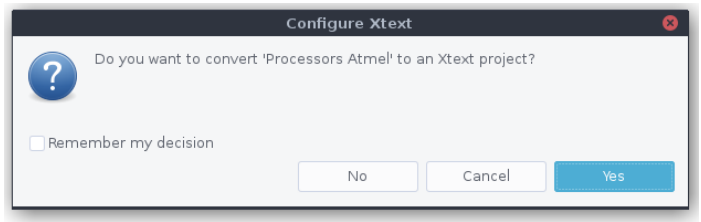




Xtext - DSL Framework for Eclipse

Eclipse IDE for Java and DSL Developers - 2019-03

Setting up Sleigh Editor - Xtext project

- Eclipse Help:Install New Software
  - Add Archive: GhidraSleighEditor.zip
- Convert GhidraScript to Xtext project
  - 

  - Allows for multi-file navigation
  - Good for casual browsing
  - Problem: all variables will be available (6502, PPC) quick-fixes will be slower

- Best: Import as new Java Project - Ghidra/Processors/6502
- Large Sleigh projects can be slow - AARCH64 - 85K LOC
- Use separate Eclipse

Sleigh Editor

Quick Demo

After Edit - ReloadSleigh Script

Only works for some changes

No Structural changes - register, memory, pcodeop,. . .

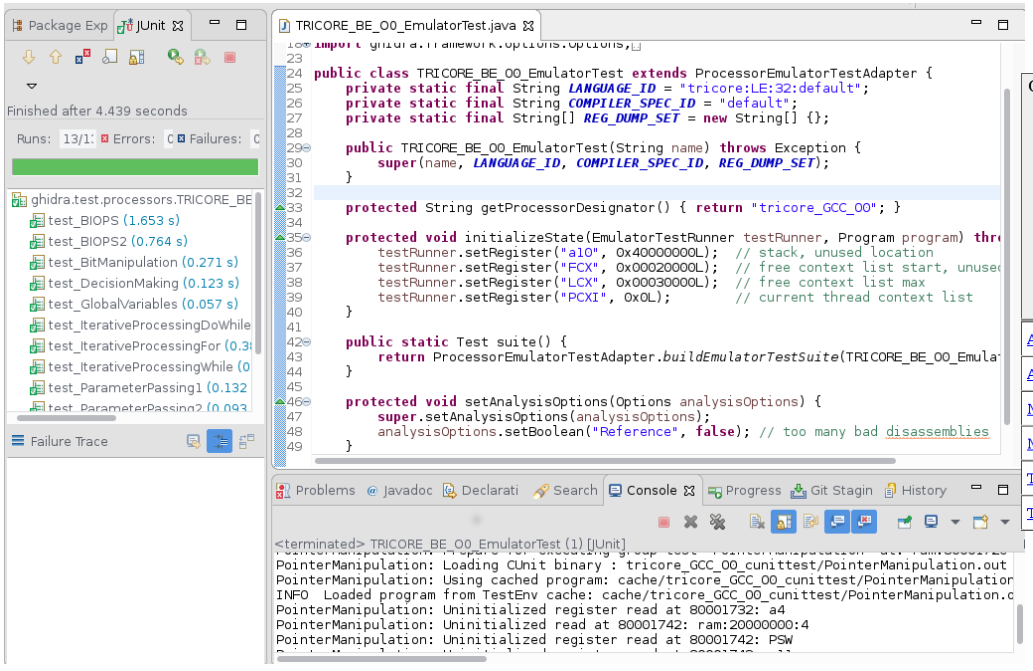
Sleigh Editor - Future Features

- Better project integration
- Code-Mining - auto-comment
- Navigation from Ghidra to SleighEditor in Eclipse
- Templates of common idioms
- More Hovers
- Conversion of number to different formats
- Syntax coloring in the printpiece
- Refactoring:
  Extract common patterns to sub-constructor
- Instruction Pattern Match Documentation

Sleigh P-Code Tests - Sleigh Testing Framework

- C code compiled for processor
- Small tests with known result
- General coverage of instructions emitted by C compilers
- Verifies core constructs - Addressing Modes, Registers
- Pcode Emulator to Execute and Verify
- Repeatable - regression testing
- Extendable - needs more cowbell
- Special case code - Assembly

Sleigh P-Code Tests - Tricore in Eclipse



All Passing

CUnit Test Results Pass/Fail/CallOther	DATE/TIME	SUMMARY	biopAddi1i1	biopAddi2i2	biopAddi4i4	biopAddu1u1	biopAddu2u2	
			BIOPS					
AARCH64_BE_O0_EmulatorTest	06/15/2019 13:21	1553/-/-	4/-/-	6/-/-	7/-/-	4/-/-	3/-/-	6/-
ARM_BE_O0_EmulatorTest	06/15/2019 13:22	1117/-/-	4/-/-	6/-/-	7/-/-	4/-/-	3/-/-	6/-
MIPS16_O0_EmulatorTest	06/15/2019 13:24	1117/-/-	4/-/-	6/-/-	7/-/-	4/-/-	3/-/-	6/-
MSP430X_O0_EmulatorTest	06/15/2019 13:26	1079/24/- ERR: 1	4/-/-	6/-/-	7/-/-	4/-/-	3/-/-	6/-
TRICORE_BE_O0_EmulatorTest	06/15/2019 13:38	1061/5/-	4/-/-	6/-/-	7/-/-	4/-/-	3/-/-	6/-
TRICORE_BE_O3_EmulatorTest	06/09/2019 20:09	1066/-/-	4/-/-	6/-/-	7/-/-	4/-/-	3/-/-	6/-

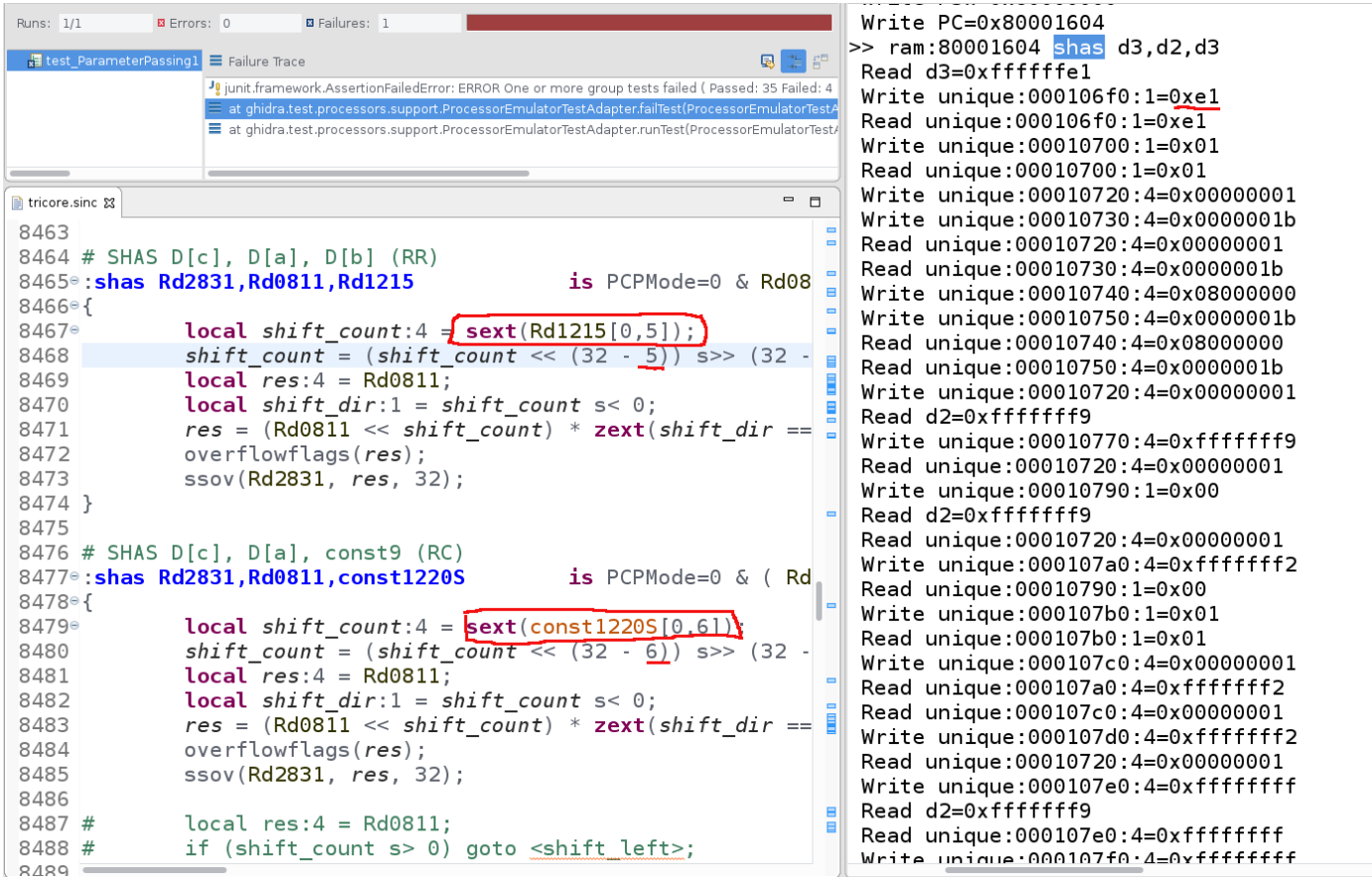
State of All Processors

Sleigh P-Code Tests - Example - Tricore

```
public class TRICOREEmulateInstructionStateModifier extends EmulateInstructionStateModifier
     Register FCX,PCXI,LCX,PSW,a10,a11,d8,a12,d12;

    public TRICOREEmulateInstructionStateModifier(Emulate emu) {
        super(emu);

            registerPcodeOpBehavior("saveCallerState", new tricore_SaveCallerState());
            registerPcodeOpBehavior("restoreCallerState", new tricore_RestoreCallerState())
            cacheRegisters(emu);
    }

    // Save Caller State, could be done in Pcode
    //
    private class tricore_SaveCallerState implements OpBehaviorOther {
        public void evaluate(Emulate emu, Varnode outputVarnode, Varnode[] inputs) {
            int numArgs = inputs.length - 1;
            if (numArgs != 3) throw new LowlevelError(this.getClass().getName() + ": requir

            MemoryState memoryState = emu.getMemoryState();

            BigInteger FCXvalue = memoryState.getBigInteger(FCX);
            // read the value at FCX, if get nothing, then assume just increment the FCX to

            long ea = FCXvalue.longValue();
            ea = ((ea & 0xffff0000) << 12) | ((ea & 0xffff) << 6);

            Address EA_addr = emu.getExecuteAddress().getNewAddress(ea);
            AddressSpace addressSpace = emu.getExecuteAddress().getAddressSpace();

            // new_FCX = M(EA, word);
            BigInteger new_FCXvalue = memoryState.getBigInteger(addressSpace, ea, 4, false)
            // if new_FCX == 0, or not-initialized, then just increment FCX again
            if (new_FCXvalue.equals(BigInteger.ZERO)) {
                new_FCXvalue = FCXvalue.add(BigInteger.ONE);
            }

            //  M(EA,16 * word) = {PCXI, PSW, A[10], A[11], D[8], D[9], D[10], D[11], A[12]
            byte[] outBytes = new byte[4*16];
            int index = 0;
            index += copyRegisterToArray(PCXI, PCXI.getBitLength()/8, memoryState, outBytes
            index += copyRegisterToArray(PSW, PSW.getBitLength()/8, memoryState, outBytes,
            index += copyRegisterToArray(a10, 2 * a10.getBitLength()/8, memoryState, outByt
            index += copyRegisterToArray(d8, 4 * d8.getBitLength()/8, memoryState, outBytes
            index += copyRegisterToArray(a12, 4 * a12.getBitLength()/8, memoryState, outByt
            index += copyRegisterToArray(d12, 4 * d12.getBitLength()/8, memoryState, outByt
```

- Contribution - mumbel
- Surprisingly well written
- Call Context Save/Restore
- TRICORE_O0_EmulatorTest
- EmulateInstructionStateModifier

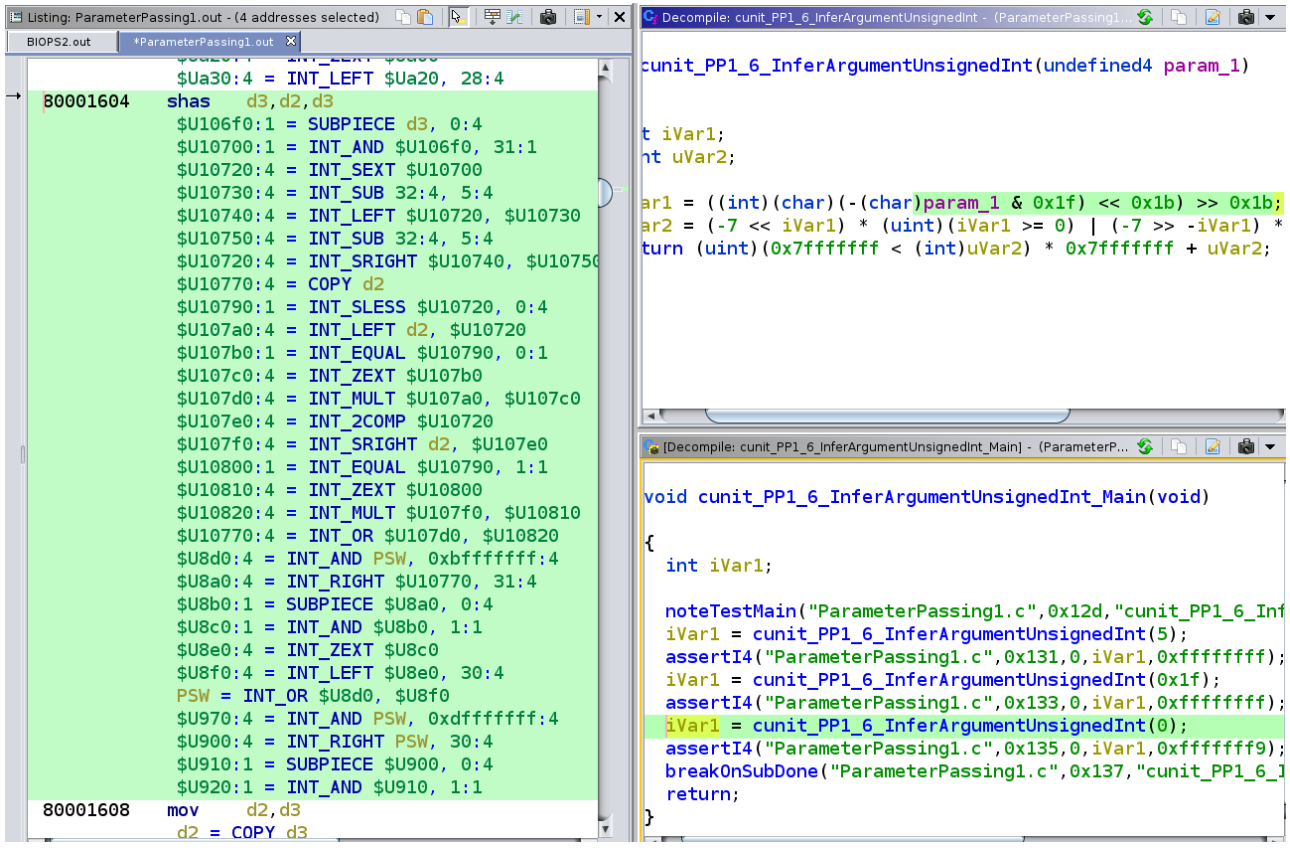
Sleigh P-Code Tests - Debugging Sleigh



Debug One Failing test - lots of output

Directory - test-output / cache, logs, results

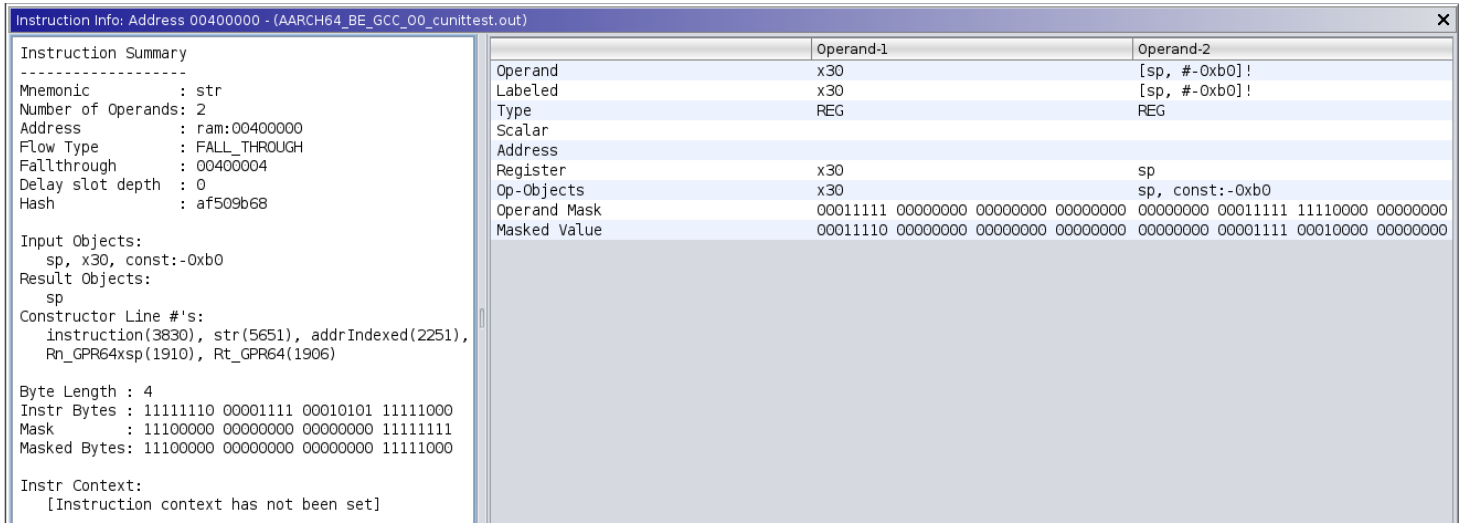
Sleigh P-Code Tests - Debugging tests



Match Unique to Read/Write

0x1f/0x1b should be 0x3f/0x1a, not extracting enough

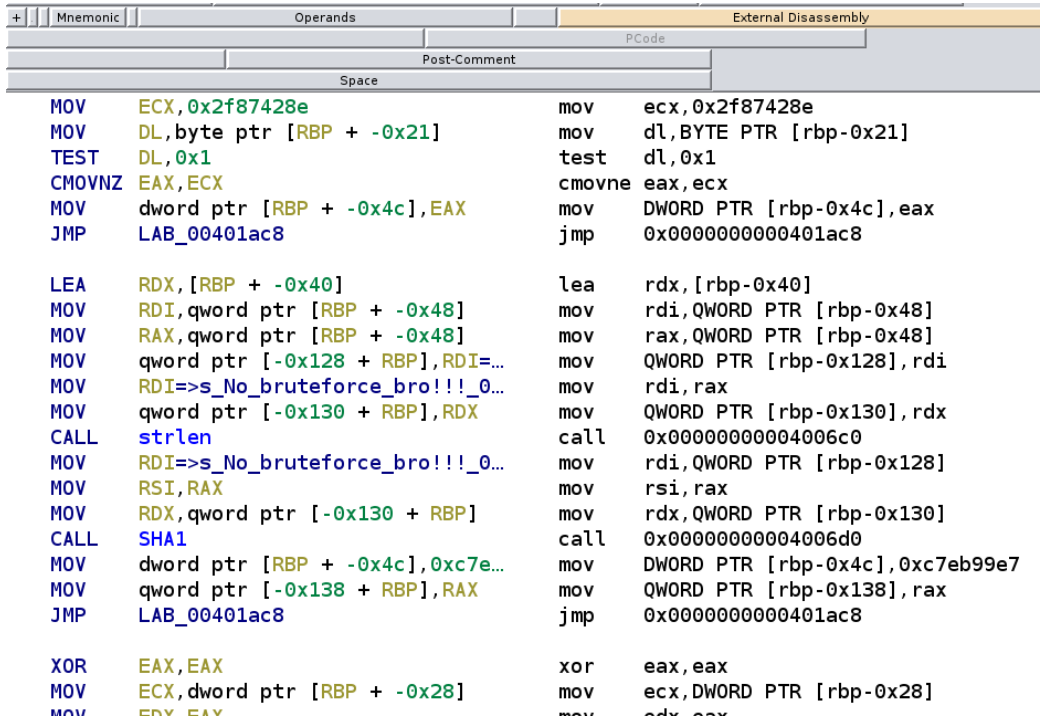
InstructionInfo - Locating problems



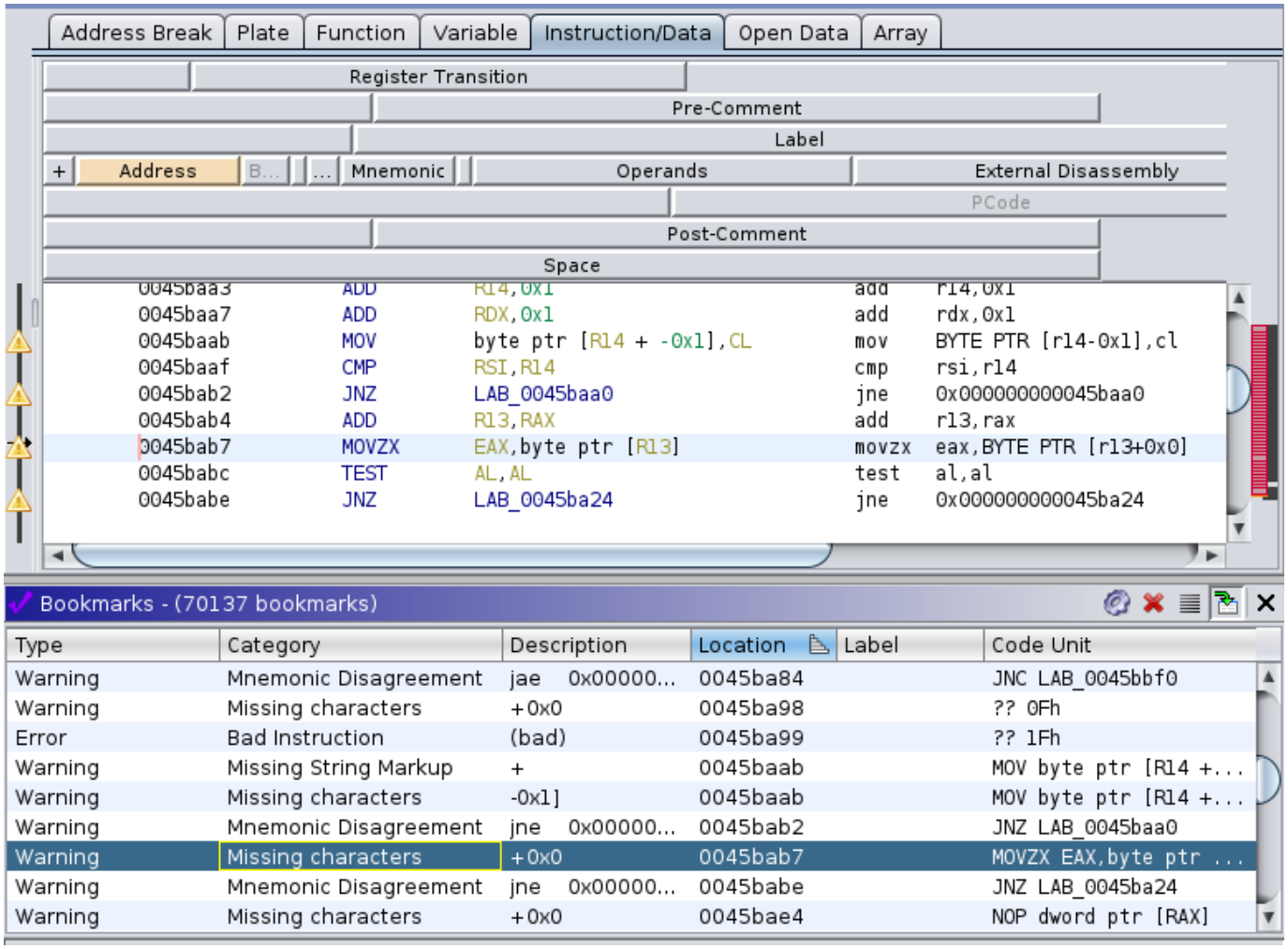

```
2249
2250 # pre indexed wback
2251 addrIndexed: "["^Rn_GPR64xsp, "#"^simm9^"]!"
2252 is size.ldstr & b_2729=7 & b_2425=0 & b_2121=0 & Rn_GPR64xsp & simm9 & opc.indexmode=3
2253 {
2254     Rn_GPR64xsp = Rn_GPR64xsp + simm9;
2255     export Rn_GPR64xsp;
2256 }
2257
```

External Disassembly Field

- *binutils* wrapper *gdis*
  - Acts as a server
- Other Disassemblers
  - dump/scrape
  - code composer studio
- Verify, Debug, Mine



Script - CompareSleighExternal



Script - DebugSleighInstructionParse

```
      : {line# 4522} XOR <spec_rm64>,<usimm8_64>
   decide on instruction bits: byte-offset=1, bitrange=(4,7), value=0x9, bytes=01001000.1000(1001)
      decendent constructors for decision node (complete tree dump ordered by line number):
      : {line# 3610} MOV <rm16>,<Reg16>
      : {line# 3612} MOV <rm32>,<Reg32>
      : {line# 3614} MOV <rm64>,<Reg64>
   decide on context bits: bitrange=(2,3), value=0x2, context=10(10)0000.00100010.00000000.00100000
      decendent constructors for decision node (complete tree dump ordered by line number):
      : {line# 3614} MOV <rm64>,<Reg64>
   check pattern[1 of 1] instruction: {line# 3614} MOV <rm64>,<Reg64>
      (  byte pattern: mask=11111111.00000000.00000000.00000000
                bytes[1-4]=10001001.11000111.01001000.10001001
               match-value=10001001.00000000.00000000.00000000 Matched
      ) -and- (
         context pattern: mask=00110000.00000001.00000000.00000000
                context(0..31)=10100000.00100010.00000000.00100000
                   match-value=00100000.00000000.00000000.00000000 Matched
                     vexMode(15,15) == 0x0 Match
                     opsize(2,3) == 0x2 Match
      ) Matched
      rm64: resolving...
         check pattern[1 of 2] rm64: {line# 1365} <Rmr64>
            byte pattern: mask=11000000.00000000.00000000.00000000
                   bytes[2-5]=11000111.01001000.10001001.10010101
                  match-value=11000000.00000000.00000000.00000000 Matched
            Rmr64: resolving...
               decide on context bits: bitrange=(13,13), value=0x0, context=10100000.00100(0)10.00000000.00100000
                  decendent constructors for decision node (complete tree dump ordered by line number):
                   Rmr64: {line# 910} <r64>
               check pattern[1 of 1] Rmr64: {line# 910} <r64>
                  context pattern: mask=00000100.00000000.00000000.00000000
                         context(8..39)=00100010.00000000.00100000.00000000
                            match-value=00000000.00000000.00000000.00000000 Matched
                              rexBprefix(13,13) == 0x0 Match
                  r64: resolving...
                     r64: register RDI (size:8)
      Reg64: resolving...
         decide on context bits: bitrange=(11,11), value=0x0, context=10100000.001(0)0010.00000000.00100000
            decendent constructors for decision node (complete tree dump ordered by line number):
             Reg64: {line# 893} <reg64>
         check pattern[1 of 1] Reg64: {line# 893} <reg64>
            context pattern: mask=00010000.00000000.00000000.00000000
                   context(8..39)=00100010.00000000.00100000.00000000
                      match-value=00000000.00000000.00000000.00000000 Matched
                        rexRprefix(11,11) == 0x0 Match
            reg64: resolving...
               reg64: register RAX (size:8)

Prototype parse successful: MOV RDI,RAX
Instruction length = 3 bytes
Instr Mask:  11111000.11111111.11000000
Instr Value: 01001000.10001001.11000000
Op-0 Mask:   00000000.00000000.00000111
Op-0 Value:  00000000.00000000.00000111
Op-1 Mask:   00000000.00000000.00111000
Op-1 Value:  00000000.00000000.00000000   (DebugSleighInstructionParse.java:54)
```

Developing a Sleigh Module - What's Good Enough?

- Disassembly
  - Decode, Display, Flow instructions
- References
  - Addressing modes
- Decompilation
  - All Data Flow, pseudoOp In/Out, Logic, Math
- Emulation - EmulateInstructionStateModifier
- Theorem Proving - Detailed effects
- Partial languages - OK
  - Use unimpl, BadInstruction(), pseudoOp
- Speed up the Process - Automate it
  - Scraping disassembly / PDF
  - Parse disassembly tables, XML descriptions

Developing a Sleigh Module - Now What?

- Tune for decompilation - calling convention
- Load format
  - ELF, .opinion for magic machineID
- Tune for emulation - Sleigh P-Code Tests
- Analyzers
  - Stock constant reference propagation can work well
  - Write specialized register propagation - Page register
- Pattern Files - recognize common patterns or key functions
- Variants - Pointer checking, Control Flow Guard
  - Decompiler Pcode UserOp injection
  - Use context, Define, variants with Slaspec
- FID Files - Static library pattern matching

Contacting Us

- The Ghidra team is on Github.
- @NSAGov on Twitter announces new releases.
- The Ghidra team is **not** on ~~Twitter~~, ~~reddit~~, ~~Slashdot~~, ~~VKontakte~~,...

Reporting Bugs

- Please report bugs!
- The perfect bug report includes:
  1. Source code.
  2. Relevant bytes from the binary.
  3. XML `Debug Function Decompilation` from decompiler.
  4. Stack trace if there is one.
- Often we need an entire function and surrounding instructions.
- Pictures work, but can limit triage.
- We reserve the right to ignore sketchy binaries :)

www.ghidra-sre.org Stats (June 25)

- 9.0.0: 302k downloads
- 9.0.1: 36k downloads
- 9.0.2: 100k downloads
- 9.0.4: 42k downloads
- Site views: 10.6M
- Video hits: 751k

Github Stats (June 25)

- 16145 stars
- 2019 forks
- 718 watching
- 608 issues, 272 open
- 111 pull requests, 35 open

References

- Xtext - itemis.com, https://www.eclipse.org/Xtext/
- mumbel - https://github.com/mumbel/ghidra/tree/tricore
- SleighEditor_README.html, build_README.txt

Questions?