QuickBinDiff: Efficient Binary Code Diff Tool Based on Command Line Interface

Motivation

- Binary diff
  - Malware detection and analysis
  - Code plagirarism and infringement
  - Find bug or vulnerability
  - binary code similarity
  - binary diversifying

Motivation

- Compiler provenance
  - The study of tracking and analyzing the origins, transformations, and optimizations applied to source code throughout the compilation process
  - For example, given a binary, it identifies provenance information such as the type of compiler used or the optimization levels applied
- Recent studies
  - Rely on expensive deep learning techniques
  - Inexplicable and difficult to understand
  - Hinders the accumulation of knowledge

Motivation

- Assumption
  - A certain researcher aims to identify optimization levels using only the differences introduced by optimization paths as features
  - For exmaple, when GCC is set to optimization level 1 or higher, it applies the omit-frame-pointer optimization pass
  - Therefore, if one understands the patterns introduced by this optimization pass, they can identify whether the given binary is compiled with O0 (no optimization) or higher (O1 and above)
  - To distinguish between a binary with this optimization pass applied and one without it, one needs to identify specific differences

Problems

- The absence of a free, open source, CLI-based diff tools focused on assembly code diff
  - GUI-based diff tools require a paid license
    - DarunGrim
      - IDA Pro required
    - Diaphora
      - IDA Pro required
    - BinDiff
      - IDA Pro required to utilize all features
      - IDA free, Binary Ninja, Ghidra required to utilize limited features

Problems

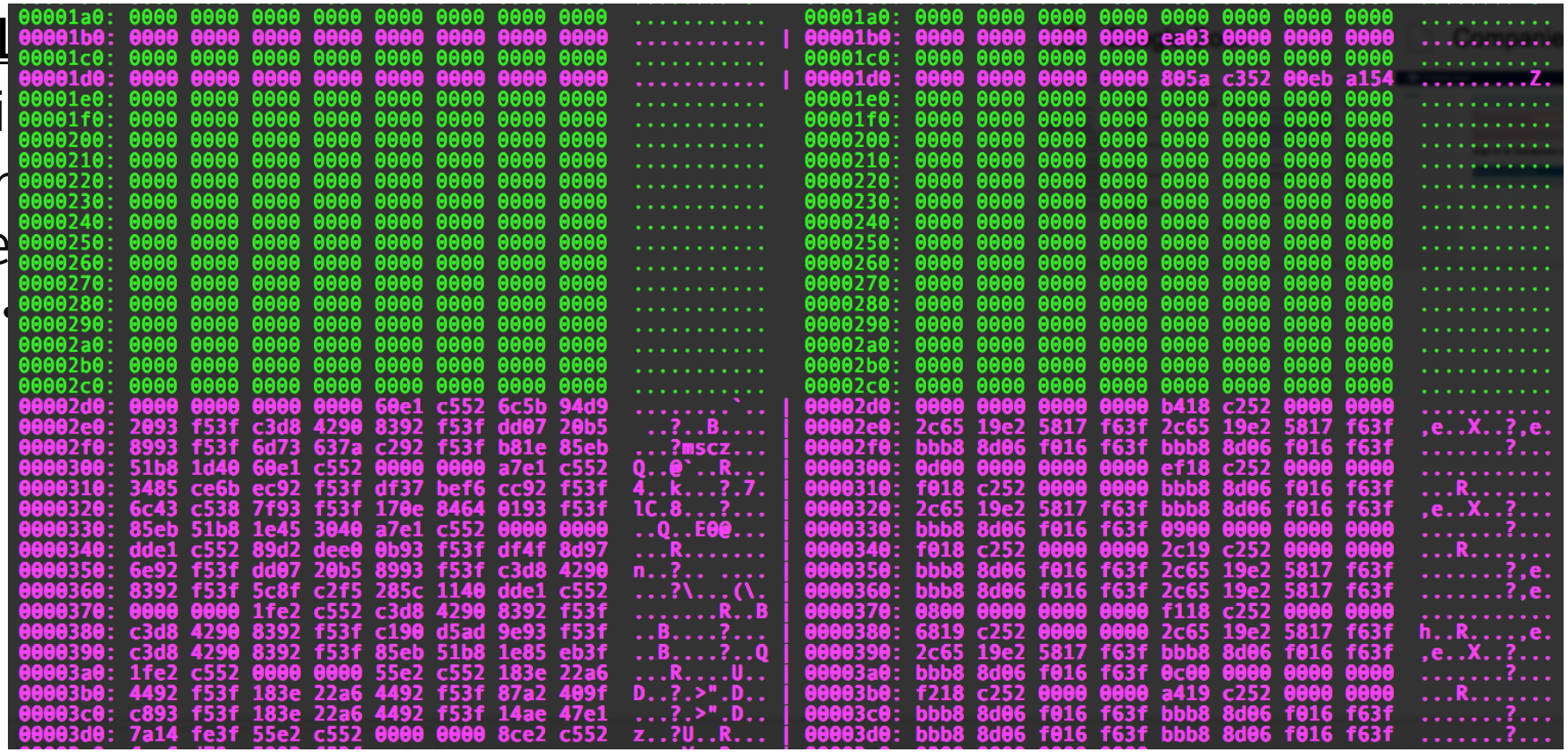
- The absence of a free, open source, CLI-based diff tools focused on assembly code diff
  - GUI-based diff tools is slow and impossible to automation
    - Assuming the license has been purchased
    - For example, to check if the target binary group has specific instructions pattern,
      - Open GUI-based diff tool
      - Load two binaries
      - Wait for analysis
      - Manually verify and record the findings

Problems

- The absence of a free, open-source, CLI-based diff tools focused on assembly code diff
  - CLI-based diff tools focus on **bytecode diff**
  - Significant loss of information and difficult to understand.
  - The only CLI-based assembly code diff tool (elf_diff) requires source code
    - compilation needed including debugging information

Problems

- The absence of a free, open-source, CLI-based diff tools focused on assembly code diff
  - CL
  - Si
  - Th
    re



Problems

	CLI	Source code required	License required	Open sourced
DarunGrim	X	X	O	O
Diaphora	X	X	O	O
BinDiff	X	X	△	O
Elf_diff	O	O	X	O
QuickBinDiff	O	X	X	O

QuickBinDiff

- Low-quality CLI tools ← **QuickBinDiff** → High-quality GUI tools

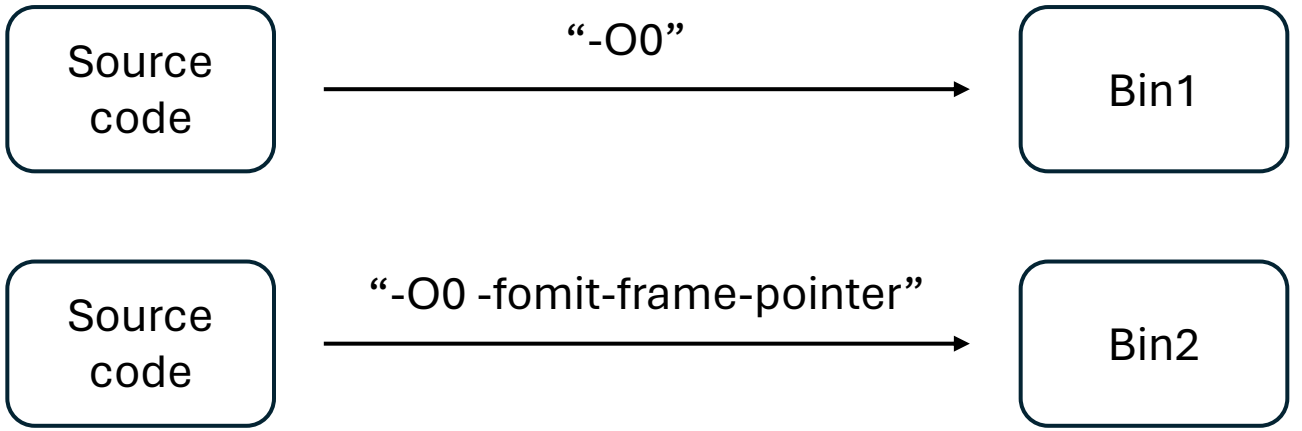
- Why not CLI tools?
  - Only suppurt bytecode diff or need source code to recompile
- Why not GUI tools?
  - Requires a license, slow, and not automatable
  - "Even though we have IDA Pro, we often use gdb or objdump”

QuickBinDiff

- The first CLI-based binary code diff tool
  - Free and open source
- Based on a text diff algorithm (myers diff)
- Supporting function-level (based on basic block similarity)
- opcode/operand-level diff for readability
- Support cross-platform, various file foramt and multi architecture
  - Platform: Linux, Windows, OS X
  - File format: PE, ELF, Match-O
  - Architecture: X86, X64, ARM, MIPS

Example

- For example, researcher in the field of compiler provenance
  - needs to analyze the differences in binaries at different optimization levels.
  - needs to analyze what differences are introduced by optimization passes.



Example

- For example, researcher in the field of compiler provenance
  - needs to analyze what differences are introduced by optimization passes.

```
    <swap>                                          <swap>
    0x4013b7: endbr64                               0x4013e5: endbr64
[-] 0x4013bb: push RBP
[-] 0x4013bc: mov RBP, RSP
[-] 0x4013bf: mov qword ptr [RBP-0x18], RDI     [+] 0x4013e9: mov qword ptr [RSP-0x18], RDI
[-] 0x4013c3: mov qword ptr [RBP-0x20], RSI     [+] 0x4013ee: mov qword ptr [RSP-0x20], RSI
[-] 0x4013c7: mov RAX, qword ptr [RBP-0x18]     [+] 0x4013f3: mov RAX, qword ptr [RSP-0x18]
    0x4013cb: mov EAX, dword ptr [RAX]              0x4013f8: mov EAX, dword ptr [RAX]
[-] 0x4013cd: mov dword ptr [RBP-0x4], EAX       [+] 0x4013fa: mov dword ptr [RSP-0x4], EAX
[-] 0x4013d0: mov RAX, qword ptr [RBP-0x20]     [+] 0x4013fe: mov RAX, qword ptr [RSP-0x20]
    0x4013d4: mov EDX, dword ptr [RAX]              0x401403: mov EDX, dword ptr [RAX]
[-] 0x4013d6: mov RAX, qword ptr [RBP-0x18]     [+] 0x401405: mov RAX, qword ptr [RSP-0x18]
    0x4013da: mov dword ptr [RAX], EDX              0x40140a: mov dword ptr [RAX], EDX
[-] 0x4013dc: mov RAX, qword ptr [RBP-0x20]     [+] 0x40140c: mov RAX, qword ptr [RSP-0x20]
[-] 0x4013e0: mov EDX, dword ptr [RBP-0x4]      [+] 0x401411: mov EDX, dword ptr [RSP-0x4]
    0x4013e3: mov dword ptr [RAX], EDX              0x401415: mov dword ptr [RAX], EDX
    0x4013e5: nop                                   0x401417: nop
[-] 0x4013e6: pop RBP
    0x4013e7: ret                                   0x401418: ret
[-] 0x4013e8: nop dword ptr [RAX+RAX+0x0]       [+] 0x401419: nop dword ptr [RAX+0x0]
```

Example

- For example, researcher in the field of compiler provenance
  - needs to analyze what differences are introduced by optimization passes.

```
    <_start>                                          <_start>
    0x0010c0: endbr64                                 0x0010c0: endbr64
    0x0010c4: xor EBP, EBP                            0x0010c4: xor EBP, EBP
    0x0010c6: mov R9, RDX                             0x0010c6: mov R9, RDX
    0x0010c9: pop RSI                                 0x0010c9: pop RSI
    0x0010ca: mov RDX, RSP                            0x0010ca: mov RDX, RSP
    0x0010cd: and RSP, 0xfffffffffffffff0             0x0010cd: and RSP, 0xfffffffffffffff0
    0x0010d1: push RAX                                0x0010d1: push RAX
    0x0010d2: push RSP                                0x0010d2: push RSP
[-] 0x0010d3: lea R8, qword ptr [RIP+0x3a6]       [+] 0x0010d3: lea R8, qword ptr [RIP+0x3d6]
[-] 0x0010da: lea RCX, qword ptr [RIP+0x32f]      [+] 0x0010da: lea RCX, qword ptr [RIP+0x35f]
    0x0010e1: lea RDI, qword ptr [RIP+0xc1]           0x0010e1: lea RDI, qword ptr [RIP+0xc1]
    0x0010e8: call qword ptr [RIP+0x2ef2]             0x0010e8: call qword ptr [RIP+0x2ef2]
    0x0010ee: hlt                                     0x0010ee: hlt
```

Example

- For example, researcher in the field of compiler provenance
  - needs to analyze what differences are introduced by optimization passes.

```
    <_start>                                                    <_start>
    0x0010c0: endbr64                                           0x0010c0: endbr64
    0x0010c4: xor EBP, EBP                                      0x0010c4: xor EBP, EBP
    0x0010c6: mov R9, RDX                                       0x0010c6: mov R9, RDX
    0x0010c9: pop RSI                                           0x0010c9: pop RSI
    0x0010ca: mov RDX, RSP                                      0x0010ca: mov RDX, RSP
    0x0010cd: and RSP, 0xfffffffffffffff0                       0x0010cd: and RSP, 0xfffffffffffffff0
    0x0010d1: push RAX                                          0x0010d1: push RAX
    0x0010d2: push RSP                                          0x0010d2: push RSP
[-] 0x0010d3: lea R8, qword ptr [RIP+0x3a6] ; <__libc_csu_fini>  [+] 0x0010d3: lea R8, qword ptr [RIP+0x3d6] ; <__libc_csu_fini>
[-] 0x0010da: lea RCX, qword ptr [RIP+0x32f] ; <__libc_csu_init> [+] 0x0010da: lea RCX, qword ptr [RIP+0x35f] ; <__libc_csu_init>
    0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>            0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>
    0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main> 0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main>
    0x0010ee: hlt                                               0x0010ee: hlt
```

Example

- For example, researcher in the field of compiler provenance
  - needs to analyze what differences are introduced by optimization passes.

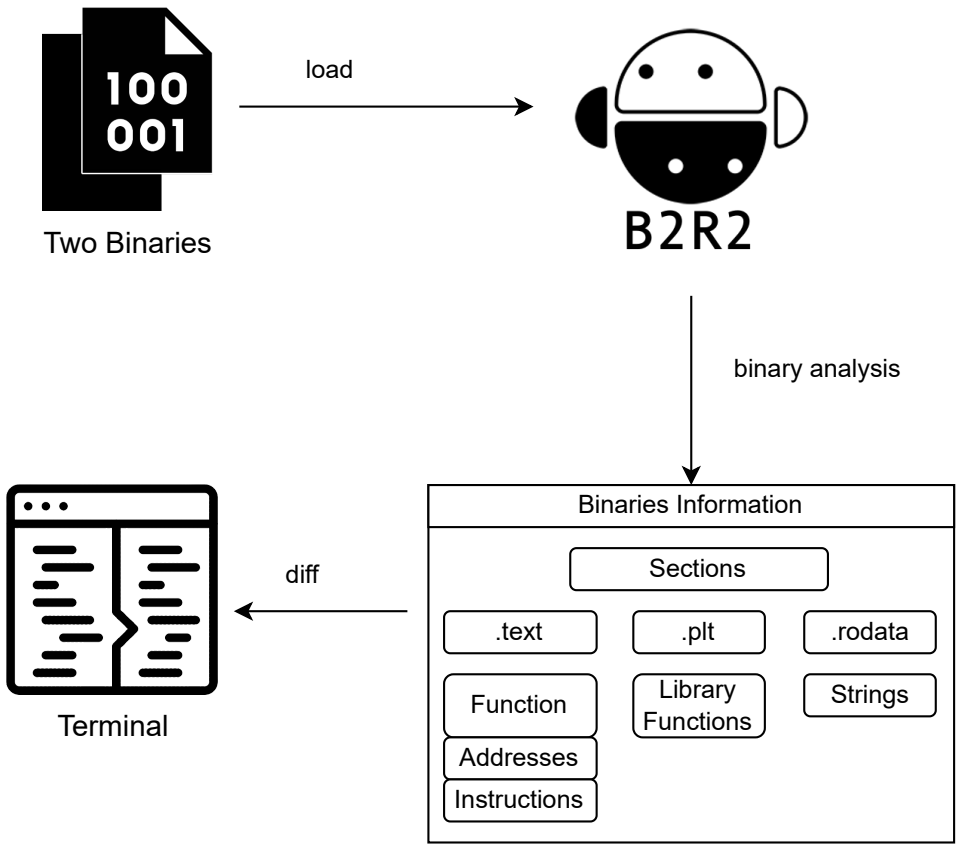
```
<_start>
0x0010c0: endbr64
0x0010c4: xor EBP, EBP
0x0010c6: mov R9, RDX
0x0010c9: pop RSI
0x0010ca: mov RDX, RSP
0x0010cd: and RSP, 0xfffffffffffffff0
0x0010d1: push RAX
0x0010d2: push RSP
0x0010d3: lea R8, qword ptr [RIP+0x3a6] ; <__libc_csu_fini>
0x0010da: lea RCX, qword ptr [RIP+0x32f] ; <__libc_csu_init>
0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>
0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main>
0x0010ee: hlt
```

```
<_start>
0x0010c0: endbr64
0x0010c4: xor EBP, EBP
0x0010c6: mov R9, RDX
0x0010c9: pop RSI
0x0010ca: mov RDX, RSP
0x0010cd: and RSP, 0xfffffffffffffff0
0x0010d1: push RAX
0x0010d2: push RSP
0x0010d3: lea R8, qword ptr [RIP+0x3d6] ; <__libc_csu_fini>
0x0010da: lea RCX, qword ptr [RIP+0x35f] ; <__libc_csu_init>
0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>
0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main>
0x0010ee: hlt
```

Demo

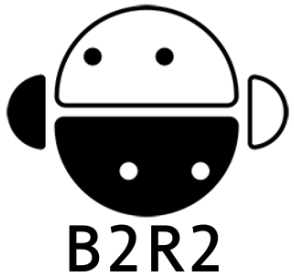
https://drive.google.com/file/d/1I-h1qa2-HIMRvWhbzwbn_fZjrBochyAo/view?usp=drive_link

Overview


100
001
Two Binaries
load
B2R2
binary analysis
Binaries Information
Sections
.text
.plt
.rodata
Function
Library Functions
Strings
Addresses
Instructions
diff
Terminal

B2R2 – Binary Analysis

- Binary Analyzer
  - Free, open-source, CLI-based
  - Cross platform (Linux, Windows, OS X)
  - Various file format (including ELF, PE, Mach-O)
  - Architecture-agnostic (including x86, x64, ARM, MIPS)
  - Providing various information necessary for diff



QuickBinDiff

- Support
  - Binary-level diff
  - Section-level diff
  - Function-level diff
  - Opcode/operand-level level diff
- Features to address several issues
  - **Fine-grained diff**
    - **Tokenize instructions**
  - **Function-level diff**
    - **basic block diff based on similarity**

Resolve “Address Shift” Problem

- Resolve function address
  - Function addresses can also vary due to “address shift”
  - They may point to semantically equivalent functions but with different syntax
    - Instruction-pointer register calculation
    - call/jmp <func>

```
    <_start>                                                    <_start>
    0x0010c0: endbr64                                           0x0010c0: endbr64
    0x0010c4: xor EBP, EBP                                      0x0010c4: xor EBP, EBP
    0x0010c6: mov R9, RDX                                       0x0010c6: mov R9, RDX
    0x0010c9: pop RSI                                           0x0010c9: pop RSI
    0x0010ca: mov RDX, RSP                                      0x0010ca: mov RDX, RSP
    0x0010cd: and RSP, 0xfffffffffffffff0                       0x0010cd: and RSP, 0xfffffffffffffff0
    0x0010d1: push RAX                                          0x0010d1: push RAX
    0x0010d2: push RSP                                          0x0010d2: push RSP
[-] 0x0010d3: lea R8, qword ptr [RIP+0x3a6] ; <__libc_csu_fini>  [+] 0x0010d3: lea R8, qword ptr [RIP+0x3d6] ; <__libc_csu_fini>
[-] 0x0010da: lea RCX, qword ptr [RIP+0x32f] ; <__libc_csu_init> [+] 0x0010da: lea RCX, qword ptr [RIP+0x35f] ; <__libc_csu_init>
    0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>            0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>
    0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main> 0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main>
    0x0010ee: hlt                                               0x0010ee: hlt
```

Resolve “Address Shift” Problem

- Resolve function address

```
    <_start>                                                    <_start>
    0x0010c0: endbr64                                           0x0010c0: endbr64
    0x0010c4: xor EBP, EBP                                      0x0010c4: xor EBP, EBP
    0x0010c6: mov R9, RDX                                       0x0010c6: mov R9, RDX
    0x0010c9: pop RSI                                           0x0010c9: pop RSI
    0x0010ca: mov RDX, RSP                                      0x0010ca: mov RDX, RSP
    0x0010cd: and RSP, 0xfffffffffffffff0                       0x0010cd: and RSP, 0xfffffffffffffff0
    0x0010d1: push RAX                                          0x0010d1: push RAX
    0x0010d2: push RSP                                          0x0010d2: push RSP
[-] 0x0010d3: lea R8, qword ptr [RIP+0x3a6] ; <__libc_csu_fini>  [+] 0x0010d3: lea R8, qword ptr [RIP+0x3d6] ; <__libc_csu_fini>
[-] 0x0010da: lea RCX, qword ptr [RIP+0x32f] ; <__libc_csu_init> [+] 0x0010da: lea RCX, qword ptr [RIP+0x35f] ; <__libc_csu_init>
    0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>            0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>
    0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main> 0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main>
    0x0010ee: hlt                                               0x0010ee: hlt
```

```
    <_start>                                                    <_start>
    0x0010c0: endbr64                                           0x0010c0: endbr64
    0x0010c4: xor EBP, EBP                                      0x0010c4: xor EBP, EBP
    0x0010c6: mov R9, RDX                                       0x0010c6: mov R9, RDX
    0x0010c9: pop RSI                                           0x0010c9: pop RSI
    0x0010ca: mov RDX, RSP                                      0x0010ca: mov RDX, RSP
    0x0010cd: and RSP, 0xfffffffffffffff0                       0x0010cd: and RSP, 0xfffffffffffffff0
    0x0010d1: push RAX                                          0x0010d1: push RAX
    0x0010d2: push RSP                                          0x0010d2: push RSP
    0x0010d3: lea R8, qword ptr [RIP+0x3a6] ; <__libc_csu_fini> 0x0010d3: lea R8, qword ptr [RIP+0x3d6] ; <__libc_csu_fini>
    0x0010da: lea RCX, qword ptr [RIP+0x32f] ; <__libc_csu_init> 0x0010da: lea RCX, qword ptr [RIP+0x35f] ; <__libc_csu_init>
    0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>            0x0010e1: lea RDI, qword ptr [RIP+0xc1] ; <main>
    0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main> 0x0010e8: call qword ptr [RIP+0x2ef2] ; <__libc_start_main>
    0x0010ee: hlt                                               0x0010ee: hlt
```

Align instructions

- More detailed differences than the instruction level are required

```
    <swap>                                          <swap>
    0x4013b7: endbr64                               0x4013e5: endbr64
[-] 0x4013bb: push RBP                          [+] 0x4013e9: mov qword ptr [RSP-0x18], RDI
[-] 0x4013bc: mov RBP, RSP                      [+] 0x4013ee: mov qword ptr [RSP-0x20], RSI
[-] 0x4013bf: mov qword ptr [RBP-0x18], RDI     [+] 0x4013f3: mov RAX, qword ptr [RSP-0x18]
[-] 0x4013c3: mov qword ptr [RBP-0x20], RSI
[-] 0x4013c7: mov RAX, qword ptr [RBP-0x18]
    0x4013cb: mov EAX, dword ptr [RAX]              0x4013f8: mov EAX, dword ptr [RAX]
[-] 0x4013cd: mov dword ptr [RBP-0x4], EAX      [+] 0x4013fa: mov dword ptr [RSP-0x4], EAX
[-] 0x4013d0: mov RAX, qword ptr [RBP-0x20]     [+] 0x4013fe: mov RAX, qword ptr [RSP-0x20]
    0x4013d4: mov EDX, dword ptr [RAX]              0x401403: mov EDX, dword ptr [RAX]
[-] 0x4013d6: mov RAX, qword ptr [RBP-0x18]     [+] 0x401405: mov RAX, qword ptr [RSP-0x18]
    0x4013da: mov dword ptr [RAX], EDX              0x40140a: mov dword ptr [RAX], EDX
[-] 0x4013dc: mov RAX, qword ptr [RBP-0x20]     [+] 0x40140c: mov RAX, qword ptr [RSP-0x20]
[-] 0x4013e0: mov EDX, dword ptr [RBP-0x4]      [+] 0x401411: mov EDX, dword ptr [RSP-0x4]
    0x4013e3: mov dword ptr [RAX], EDX              0x401415: mov dword ptr [RAX], EDX
    0x4013e5: nop                                   0x401417: nop
[-] 0x4013e6: pop RBP
    0x4013e7: ret                                   0x401418: ret
[-] 0x4013e8: nop dword ptr [RAX+RAX+0x0]       [+] 0x401419: nop dword ptr [RAX+0x0]
```

Align instructions

- The red boxes are similar, but the diff algorithm is not sensitive to similarity.

```
    <swap>                                          <swap>
    0x4013b7: endbr64                               0x4013e5: endbr64
[-] 0x4013bb: push RBP                          [+] 0x4013e9: mov qword ptr [RSP-0x18], RDI
[-] 0x4013bc: mov RBP, RSP                      [+] 0x4013ee: mov qword ptr [RSP-0x20], RSI
[-] 0x4013bf: mov qword ptr [RBP-0x18], RDI     [+] 0x4013f3: mov RAX, qword ptr [RSP-0x18]
[-] 0x4013c3: mov qword ptr [RBP-0x20], RSI
[-] 0x4013c7: mov RAX, qword ptr [RBP-0x18]
    0x4013cb: mov EAX, dword ptr [RAX]              0x4013f8: mov EAX, dword ptr [RAX]
[-] 0x4013cd: mov dword ptr [RBP-0x4], EAX      [+] 0x4013fa: mov dword ptr [RSP-0x4], EAX
[-] 0x4013d0: mov RAX, qword ptr [RBP-0x20]     [+] 0x4013fe: mov RAX, qword ptr [RSP-0x20]
    0x4013d4: mov EDX, dword ptr [RAX]              0x401403: mov EDX, dword ptr [RAX]
[-] 0x4013d6: mov RAX, qword ptr [RBP-0x18]     [+] 0x401405: mov RAX, qword ptr [RSP-0x18]
    0x4013da: mov dword ptr [RAX], EDX              0x40140a: mov dword ptr [RAX], EDX
[-] 0x4013dc: mov RAX, qword ptr [RBP-0x20]     [+] 0x40140c: mov RAX, qword ptr [RSP-0x20]
[-] 0x4013e0: mov EDX, dword ptr [RBP-0x4]      [+] 0x401411: mov EDX, dword ptr [RSP-0x4]
    0x4013e3: mov dword ptr [RAX], EDX              0x401415: mov dword ptr [RAX], EDX
    0x4013e5: nop                                   0x401417: nop
[-] 0x4013e6: pop RBP
    0x4013e7: ret                                   0x401418: ret
[-] 0x4013e8: nop dword ptr [RAX+RAX+0x0]       [+] 0x401419: nop dword ptr [RAX+0x0]
```

Align instructions

```
    <swap>                                          <swap>
    0x4013b7: endbr64                               0x4013e5: endbr64
[-] 0x4013bb: push RBP                          [+] 0x4013e9: mov qword ptr [RSP-0x18], RDI
[-] 0x4013bc: mov RBP, RSP                      [+] 0x4013ee: mov qword ptr [RSP-0x20], RSI
[-] 0x4013bf: mov qword ptr [RBP-0x18], RDI     [+] 0x4013f3: mov RAX, qword ptr [RSP-0x18]
[-] 0x4013c3: mov qword ptr [RBP-0x20], RSI
[-] 0x4013c7: mov RAX, qword ptr [RBP-0x18]
    0x4013cb: mov EAX, dword ptr [RAX]          0x4013f8: mov EAX, dword ptr [RAX]
```

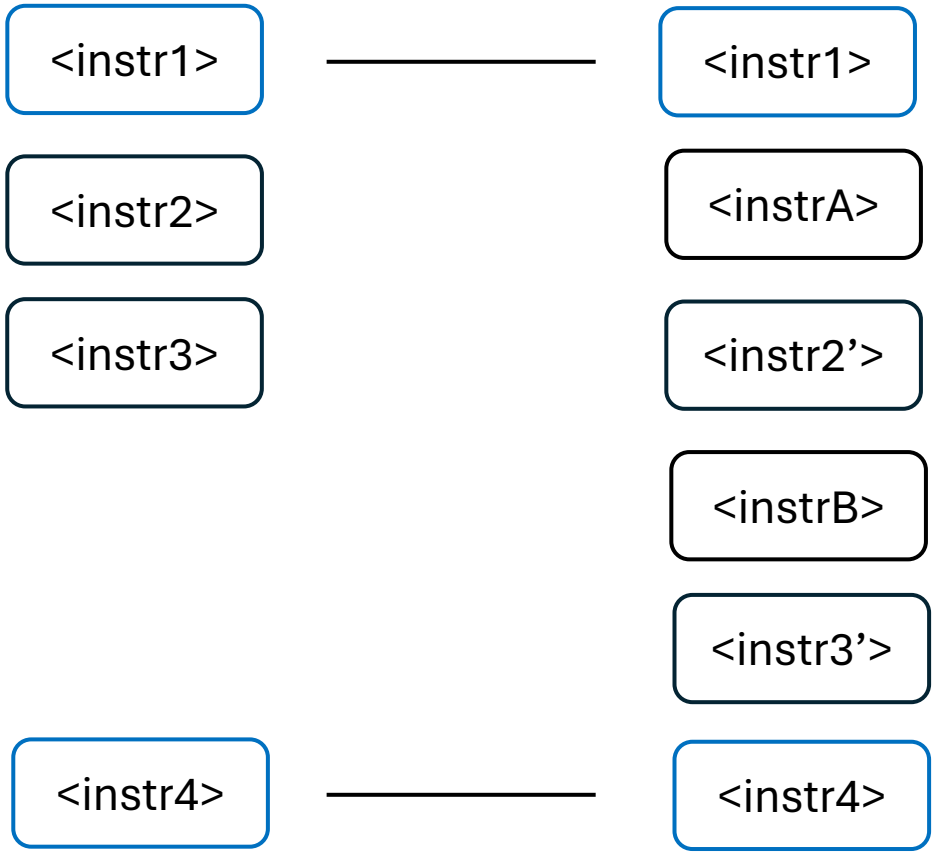
```
    <swap>                                          <swap>
    0x4013b7: endbr64                               0x4013e5: endbr64
[-] 0x4013bb: push RBP
[-] 0x4013bc: mov RBP, RSP
[-] 0x4013bf: mov qword ptr [RBP-0x18], RDI     [+] 0x4013e9: mov qword ptr [RSP-0x18], RDI
[-] 0x4013c3: mov qword ptr [RBP-0x20], RSI     [+] 0x4013ee: mov qword ptr [RSP-0x20], RSI
[-] 0x4013c7: mov RAX, qword ptr [RBP-0x18]     [+] 0x4013f3: mov RAX, qword ptr [RSP-0x18]
    0x4013cb: mov EAX, dword ptr [RAX]          0x4013f8: mov EAX, dword ptr [RAX]
```

Align instructions

- Opcode/operand-level diffs that are sensitive to similarity

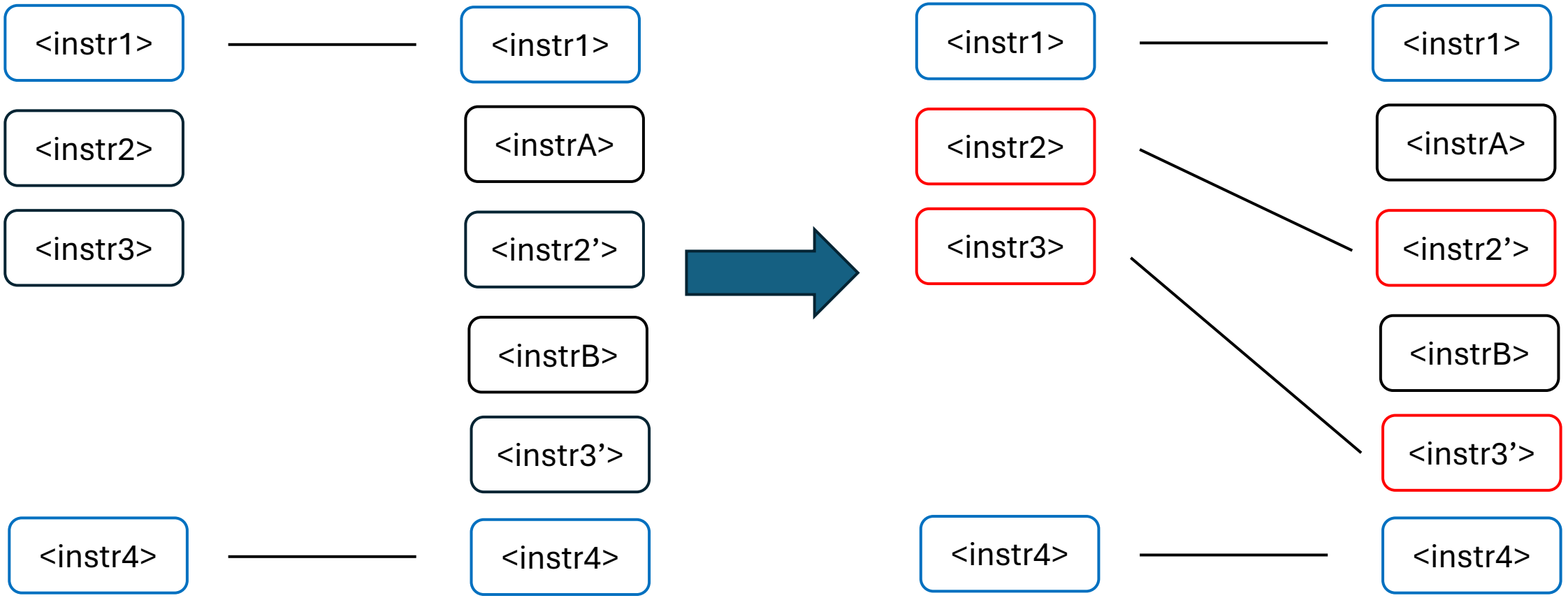
Align instructions

- Opcode/operand-level diffs that are sensitive to similarity



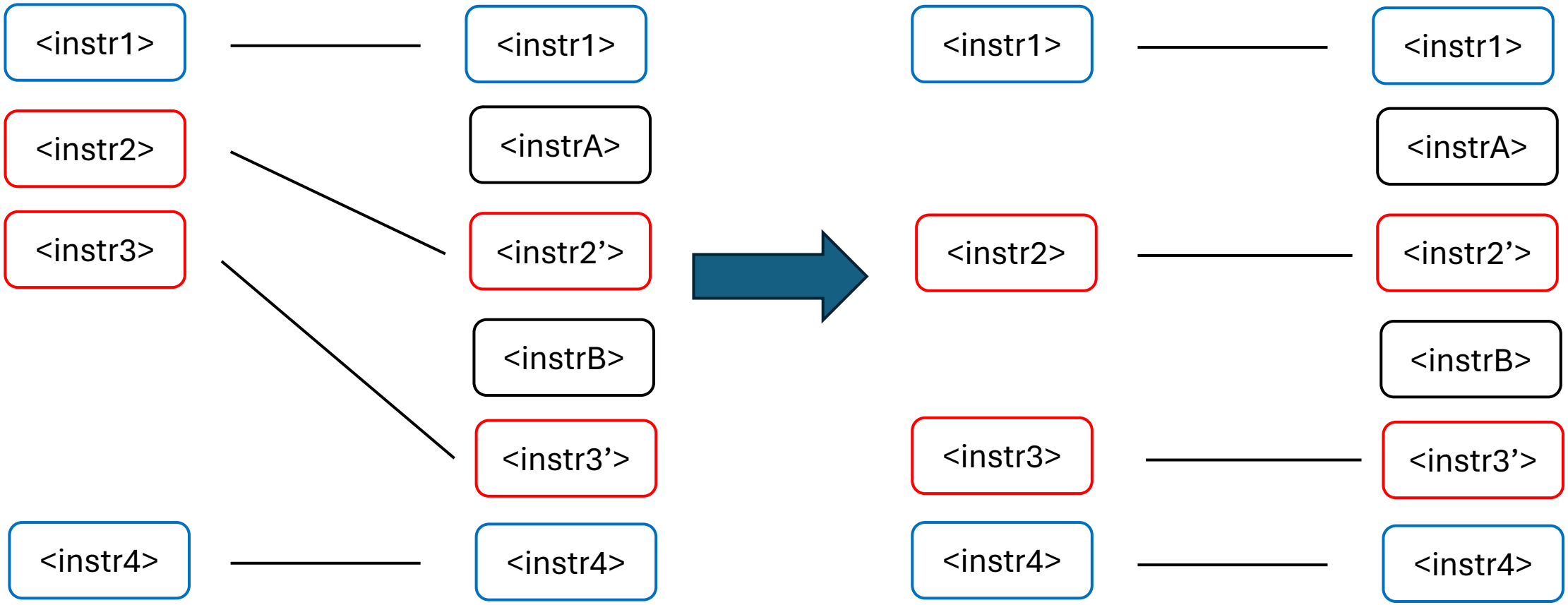
Align instructions

- Opcode/operand-level diffs that are sensitive to similarity



Align instructions

- Opcode/operand-level diffs that are sensitive to similarity



Fine-grained diff

```
    <swap>                                          <swap>
    0x4013b7: endbr64                               0x4013e5: endbr64
[-] 0x4013bb: push RBP
[-] 0x4013bc: mov RBP, RSP
[-] 0x4013bf: mov qword ptr [RBP-0x18], RDI     [+] 0x4013e9: mov qword ptr [RSP-0x18], RDI
[-] 0x4013c3: mov qword ptr [RBP-0x20], RSI     [+] 0x4013ee: mov qword ptr [RSP-0x20], RSI
[-] 0x4013c7: mov RAX, qword ptr [RBP-0x18]     [+] 0x4013f3: mov RAX, qword ptr [RSP-0x18]
    0x4013cb: mov EAX, dword ptr [RAX]              0x4013f8: mov EAX, dword ptr [RAX]
```

```
    <swap>                                          <swap>
    0x4013b7: endbr64                               0x4013e5: endbr64
[-] 0x4013bb: push RBP
[-] 0x4013bc: mov RBP, RSP
[-] 0x4013bf: mov qword ptr [RBP-0x18], RDI     [+] 0x4013e9: mov qword ptr [RSP-0x18], RDI
[-] 0x4013c3: mov qword ptr [RBP-0x20], RSI     [+] 0x4013ee: mov qword ptr [RSP-0x20], RSI
[-] 0x4013c7: mov RAX, qword ptr [RBP-0x18]     [+] 0x4013f3: mov RAX, qword ptr [RSP-0x18]
    0x4013cb: mov EAX, dword ptr [RAX]              0x4013f8: mov EAX, dword ptr [RAX]
```

Fine-grained diff

- Opcode/operand-level diffs that are sensitive to similarity
- Tokenize instructions and match using Jaccard similarity
  - `mov qword ptr [RBP-0x10], RSI`

```
Opcode: mov
Operand1: qword ptr [RBP-0x10]
  Memory Operand Type: qword ptr
  Memory Addressing Mode: [RBP-0x10]
    Opening Bracket: [
    Register: RBP
    Displacement: -0x10
    Closing Bracket: ]
Operand2: RSI
  Register: RSI
```

```
tokens = {"mov",
          "qword ptr", "[", "RBP", "-0x10", "]",
          "RSI"}
```

Future work

- Evaluation
  - RQ: How efficient is QuickBinDiff compared to CLI-based binary diff tools?
  - QuickBinDiff vs ELF_Diff

Summary

- Binary diff is a technology required in various fields.
  - The absence of a free, open-source, CLI-based diff tools focused on assembly code diff
  - Difficult to automatically handle a large quantity of binaries
- Fast, free, open sourced and CLI-based binary code diff tool
  - Support cross-platform, various file foramt and multi architecture