EAI SecureComm
Security and Privacy in Communication Networks

BinSimDB: Benchmark Dataset Construction for *Fine-Grained* Binary Code Similarity Analysis

Fei Zuo[1], Cody Tompkins[1], Qiang Zeng[2], Lannan Luo[2],
Yung Ryn Choe[3], Junghwan Rhee[1]

[1]University of Central Oklahoma
[2]George Mason University
[3]Sandia National Laboratories

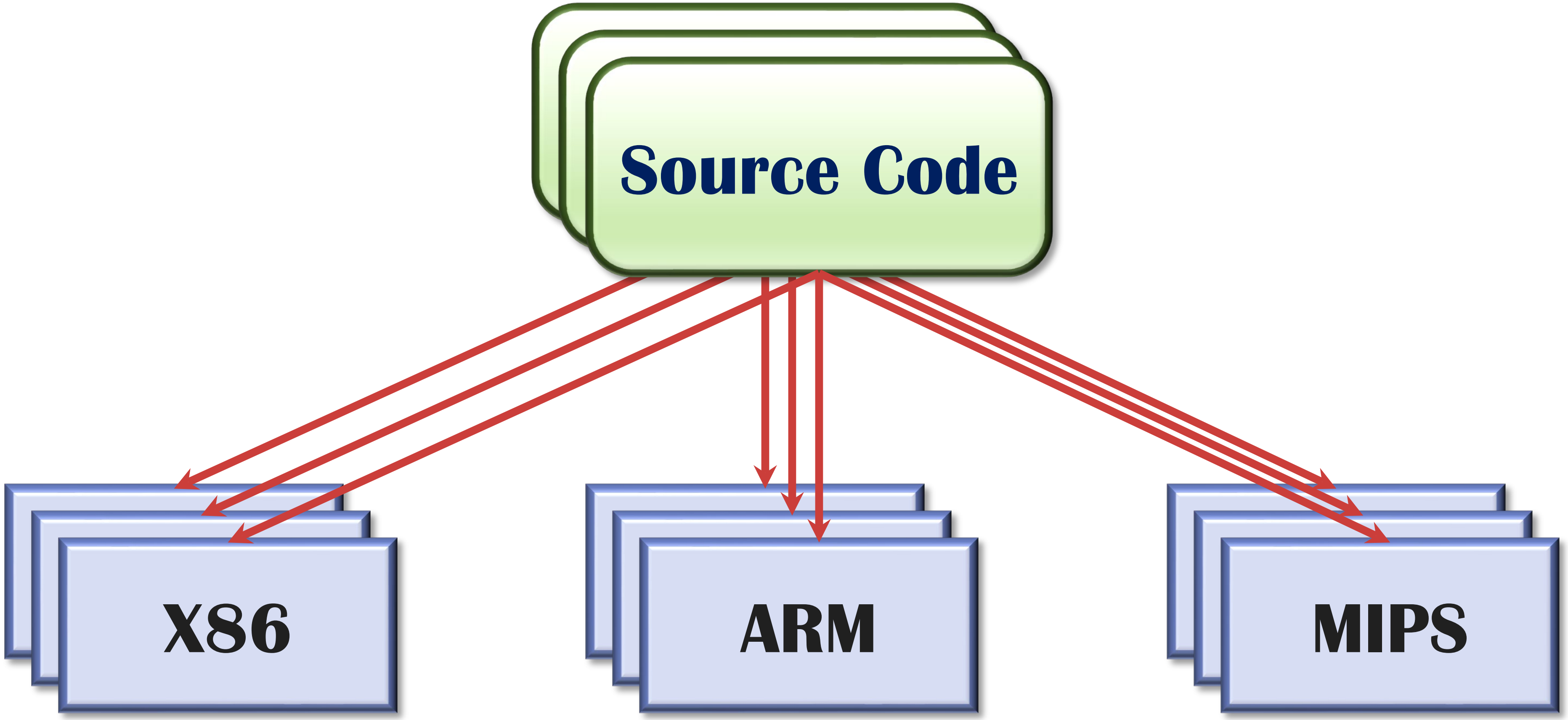
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525

Outline

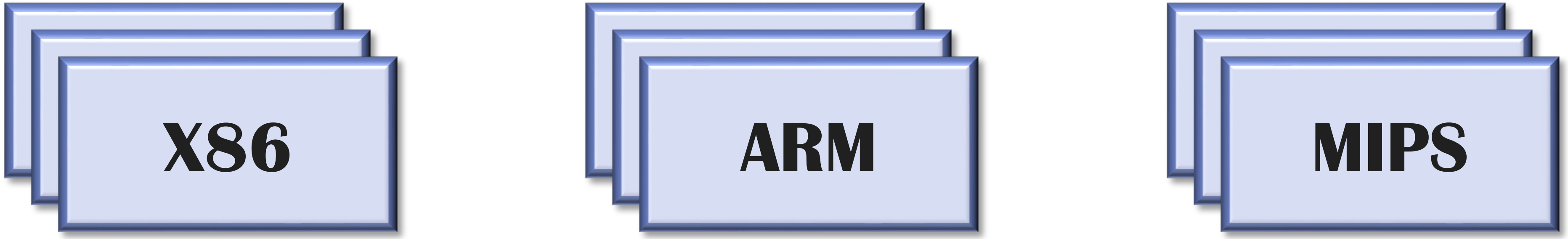
- Background
- Related work
- Methodology
- Evaluation
- Case study
- Take-away message

Cross-Architecture
Binary Code Similarity Analysis (BCSA)

From source code to binary code



From source code to binary code



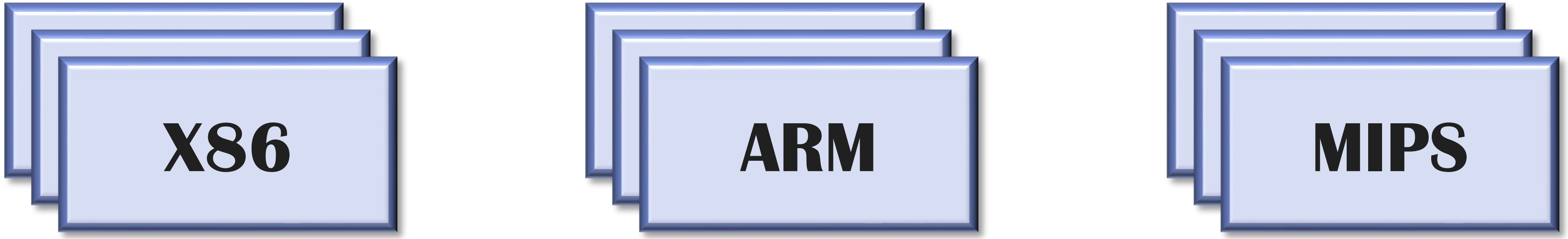
X86

ARM

MIPS

Source code is unavailable

- Proprietary software
- IoT firmware
- Malware



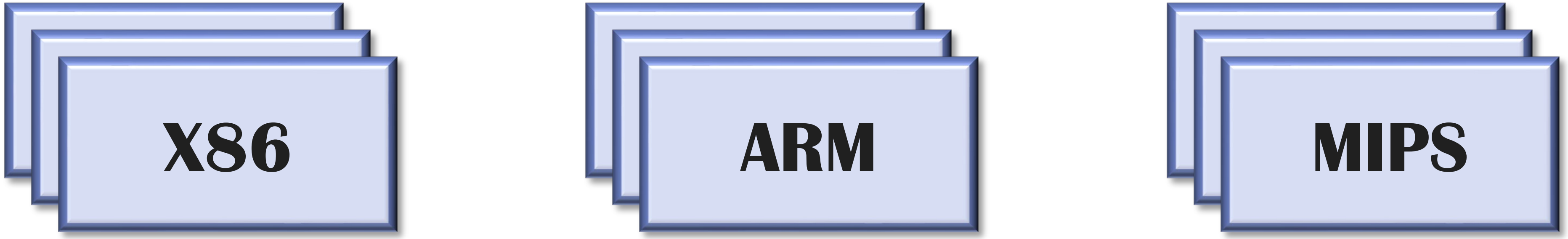
X86

ARM

MIPS

Cross-architecture binary similarity analysis

- Plagiarism detection
- Malware family identification
- Vulnerability discovery



X86

ARM

MIPS

Current Status and Challenges

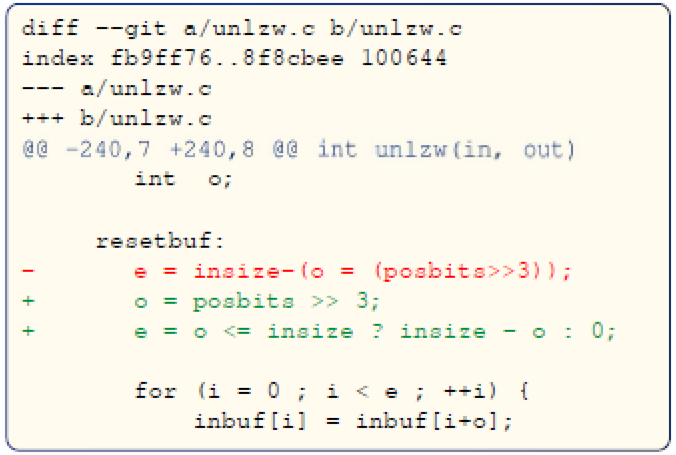
- Deep-learning-based methods have shown promise, where dataset matters
- However, well-labelled, high-quality datasets are precious or even scare in cross-architecture BCSA
- Eg. After investigating 43 papers, Kim et al. [TSE'22] found "*only two of them opened their entire dataset*".

Current Status and Challenges

- Dataset Construction
  - Kim et al, [TSE'22]
  - Marcelli et al, [USENIX Security'22]
  - Song et al, [IEEE IoT Journal'22]
- Mainly focusing on function-level equivalent binary pairs
- Cannot support fine-grained analysis

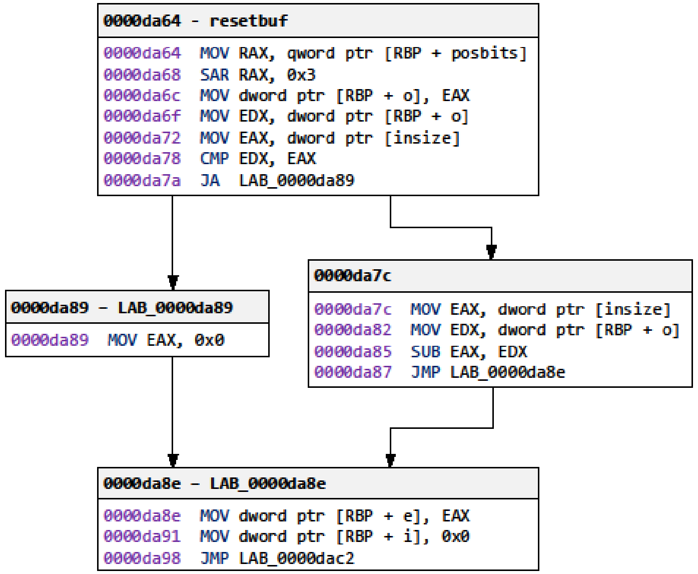
Current Status and Challenges

- Fine-grained analysis is useful for detecting subtle discrepancy
- Eg., There are 75 basic blocks in the unlzw function, while code changes caused by the patch only take up approximately 5.3% of the entire function.



```
diff --git a/unlzw.c b/unlzw.c
index fb9ff76..8f8cbee 100644
--- a/unlzw.c
+++ b/unlzw.c
@@ -240,7 +240,8 @@ int unlzw(in, out)
         int  o;

     resetbuf:
-        e = insize-(o = (posbits>>3));
+        o = posbits >> 3;
+        e = o <= insize ? insize - o : 0;

         for (i = 0 ; i < e ; ++i) {
             inbuf[i] = inbuf[i+o];
```

The patch in gzip for CVE-2010-0001



```
0000da64 - resetbuf
0000da64  MOV RAX, qword ptr [RBP + posbits]
0000da68  SAR RAX, 0x3
0000da6c  MOV dword ptr [RBP + o], EAX
0000da6f  MOV EDX, dword ptr [RBP + o]
0000da72  MOV EAX, dword ptr [insize]
0000da78  CMP EDX, EAX
0000da7a  JA  LAB_0000da89
```

```
0000da89 - LAB_0000da89
0000da89  MOV EAX, 0x0
```

```
0000da7c
0000da7c  MOV EAX, dword ptr [insize]
0000da82  MOV EDX, dword ptr [RBP + o]
0000da85  SUB EAX, EDX
0000da87  JMP LAB_0000da8e
```

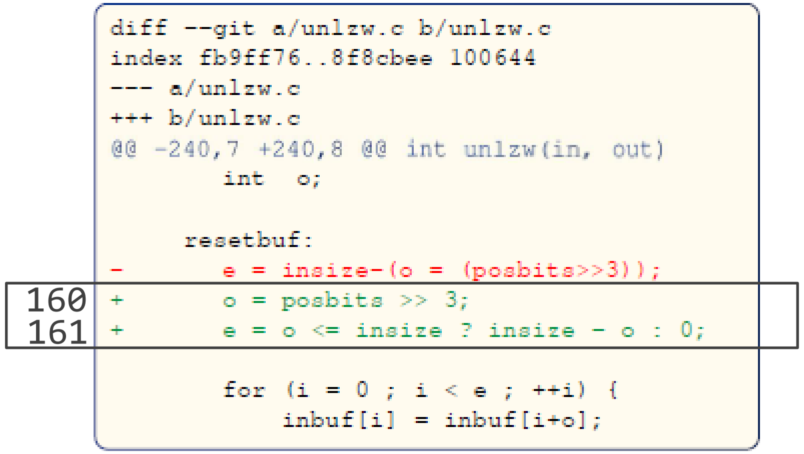
```
0000da8e - LAB_0000da8e
0000da8e  MOV dword ptr [RBP + e], EAX
0000da91  MOV dword ptr [RBP + i], 0x0
0000da98  JMP LAB_0000dac2
```

Current Status and Challenges

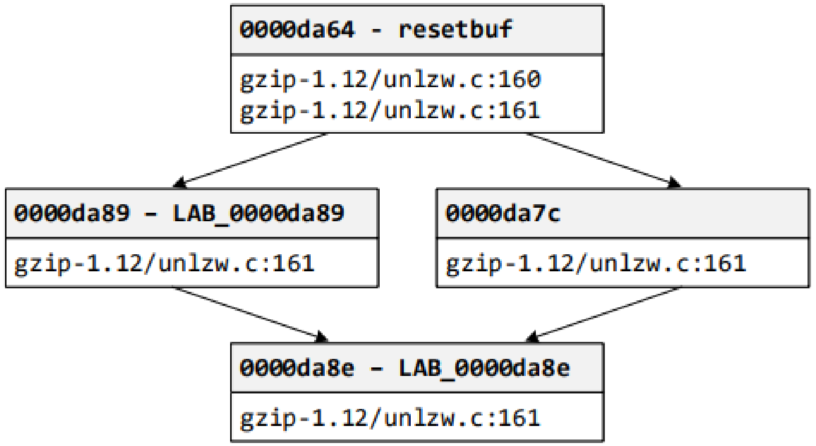
- Unlike functions, which have names
- Constructing fine-grained equivalent binary code snippet pairs (e.g. basic block-level) is non-trivial
- Prior attempt [NDSS'19] relies on the annotations generated by the LLVM, leaving a gap in practical application

Current Status and Challenges

- More importantly, prior work cannot well address the following two challenges:
  1. A single line of source code may correspond to multiple basic blocks, causing confusion when pairing

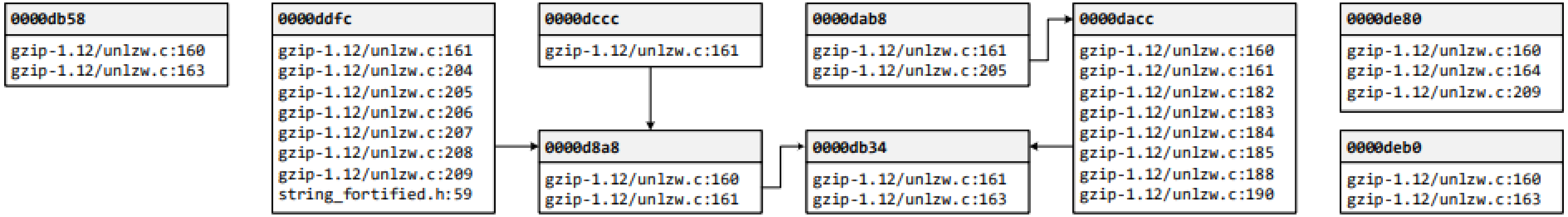


```
    diff --git a/unlzw.c b/unlzw.c
    index fb9ff76..8f8cbee 100644
    --- a/unlzw.c
    +++ b/unlzw.c
    @@ -240,7 +240,8 @@ int unlzw(in, out)
                int   o;

            resetbuf:
    -           e = insize-(o = (posbits>>3));
160 +           o = posbits >> 3;
161 +           e = o <= insize ? insize - o : 0;

                for (i = 0 ; i < e ; ++i) {
                    inbuf[i] = inbuf[i+o];
```



Current Status and Challenges

- More importantly, prior work cannot well address the following two challenges:
  1. A single line of source code may correspond to multiple basic blocks, causing confusion when pairing
  2. Due to compiler optimization behavior, basic blocks can be merged or reorganized when using different opt levels



0000db58
gzip-1.12/unlzw.c:160
gzip-1.12/unlzw.c:163

0000ddfc
gzip-1.12/unlzw.c:161
gzip-1.12/unlzw.c:204
gzip-1.12/unlzw.c:205
gzip-1.12/unlzw.c:206
gzip-1.12/unlzw.c:207
gzip-1.12/unlzw.c:208
gzip-1.12/unlzw.c:209
string_fortified.h:59

0000dccc
gzip-1.12/unlzw.c:161

0000d8a8
gzip-1.12/unlzw.c:160
gzip-1.12/unlzw.c:161

0000dab8
gzip-1.12/unlzw.c:161
gzip-1.12/unlzw.c:205

0000db34
gzip-1.12/unlzw.c:161
gzip-1.12/unlzw.c:163

0000dacc
gzip-1.12/unlzw.c:160
gzip-1.12/unlzw.c:161
gzip-1.12/unlzw.c:182
gzip-1.12/unlzw.c:183
gzip-1.12/unlzw.c:184
gzip-1.12/unlzw.c:185
gzip-1.12/unlzw.c:188
gzip-1.12/unlzw.c:190

0000de80
gzip-1.12/unlzw.c:160
gzip-1.12/unlzw.c:164
gzip-1.12/unlzw.c:209

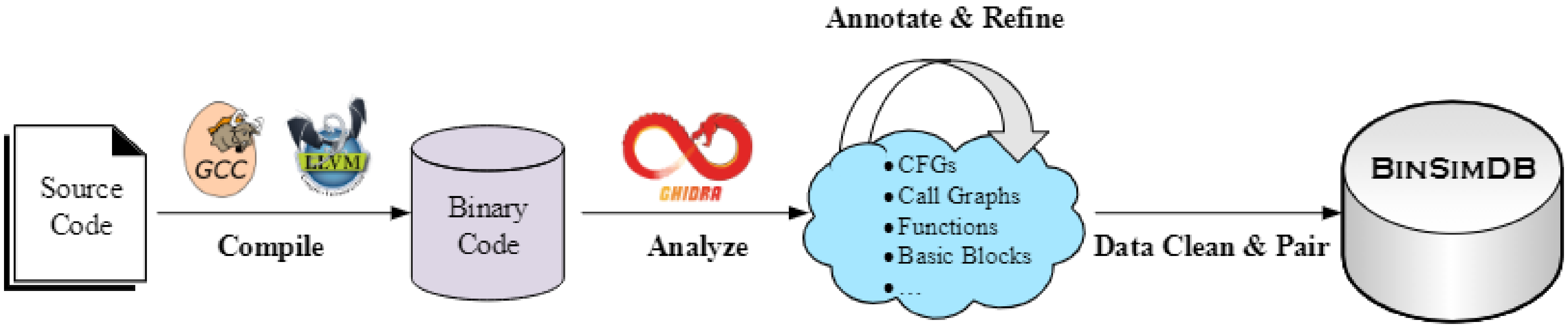
0000deb0
gzip-1.12/unlzw.c:160
gzip-1.12/unlzw.c:163

Methodology

- Pipeline
- BMerge Algorithm
- BPair Algorithm
- Transformer-based Similarity Detector

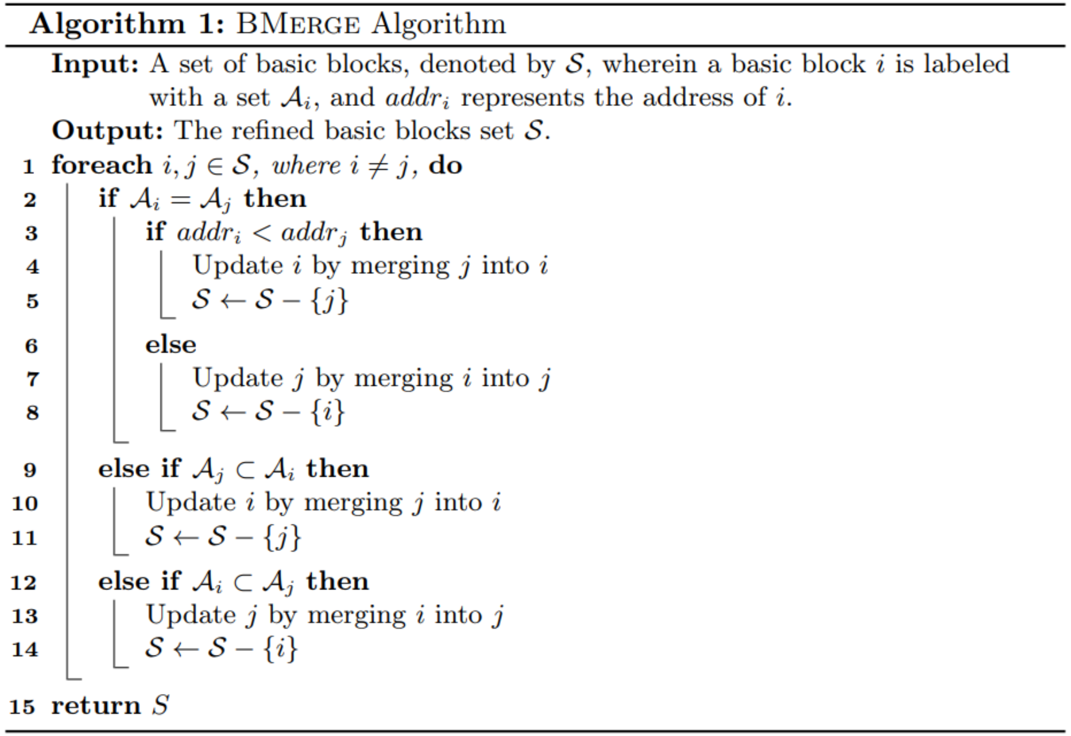
Methodology

- Pipeline



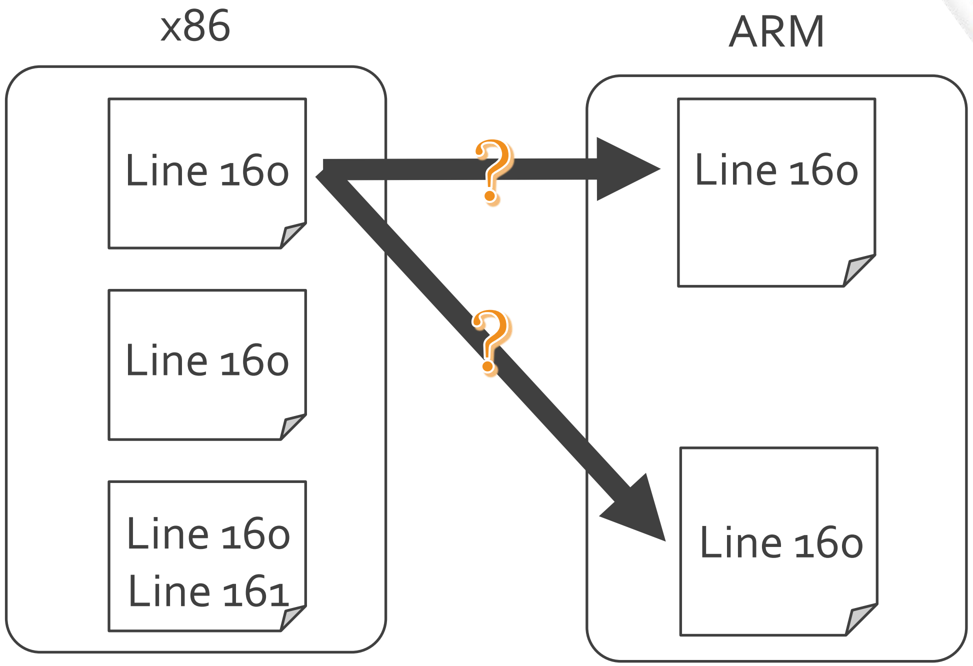
Methodology

- BMerge Algorithm



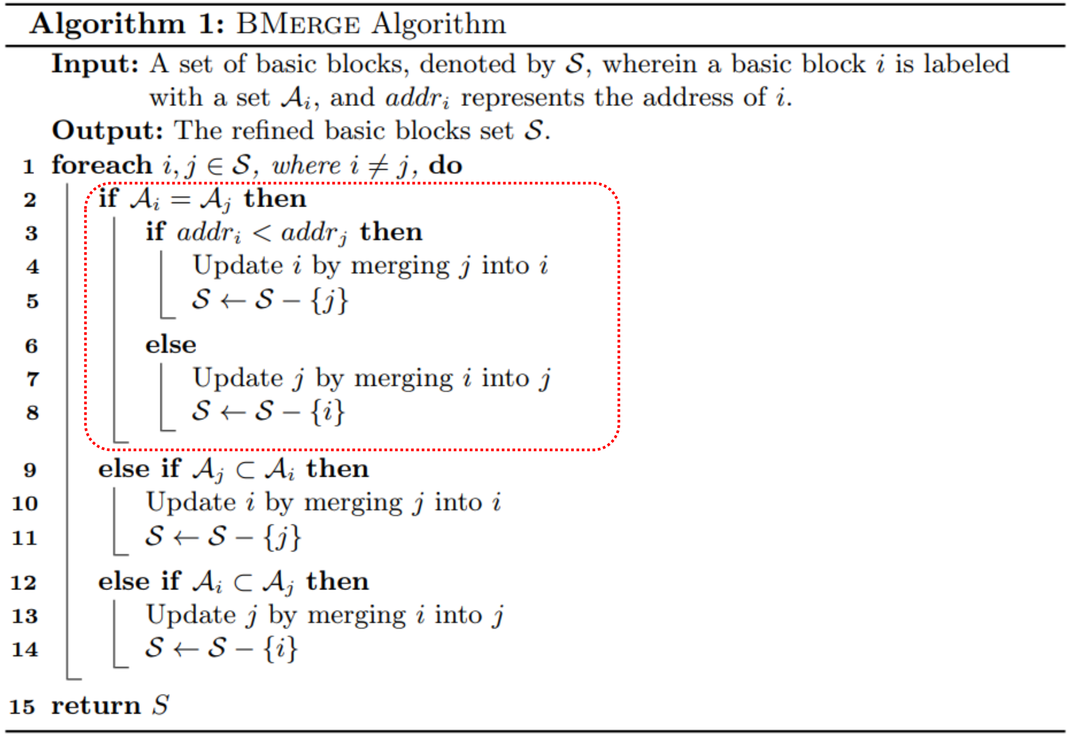
Algorithm 1: BMERGE Algorithm

```
Input: A set of basic blocks, denoted by $\mathcal{S}$, wherein a basic block $i$ is labeled
       with a set $\mathcal{A}_i$, and $addr_i$ represents the address of $i$.
Output: The refined basic blocks set $\mathcal{S}$.
foreach $i, j \in \mathcal{S}$, where $i \neq j$, do
    if $\mathcal{A}_i = \mathcal{A}_j$ then
        if $addr_i < addr_j$ then
            Update $i$ by merging $j$ into $i$
            $\mathcal{S} \leftarrow \mathcal{S} - \{j\}$
        else
            Update $j$ by merging $i$ into $j$
            $\mathcal{S} \leftarrow \mathcal{S} - \{i\}$
    else if $\mathcal{A}_j \subset \mathcal{A}_i$ then
        Update $i$ by merging $j$ into $i$
        $\mathcal{S} \leftarrow \mathcal{S} - \{j\}$
    else if $\mathcal{A}_i \subset \mathcal{A}_j$ then
        Update $j$ by merging $i$ into $j$
        $\mathcal{S} \leftarrow \mathcal{S} - \{i\}$
return $S$
```



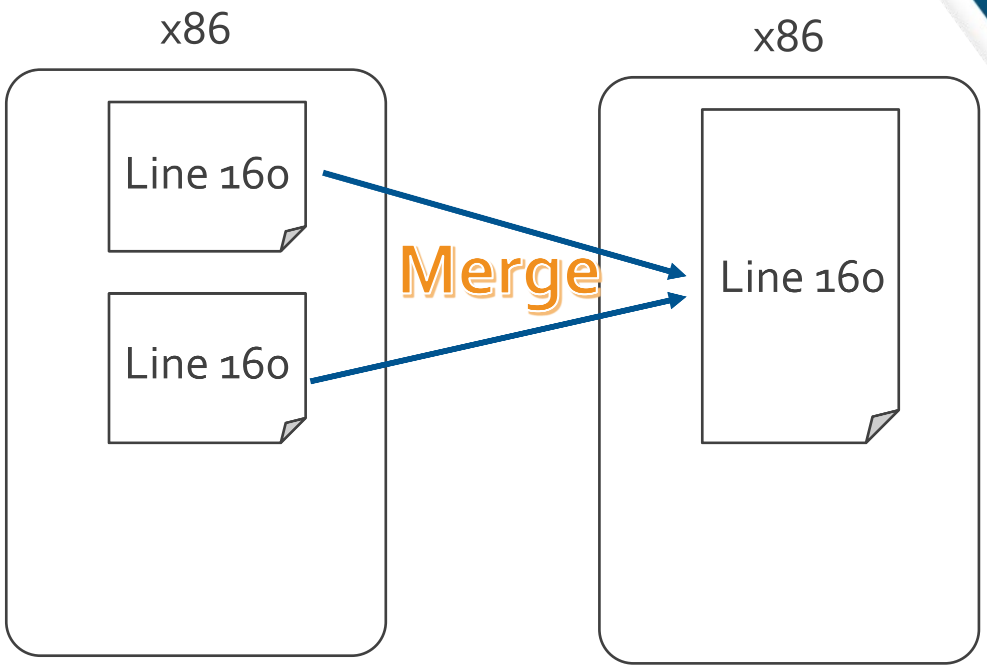
Methodology

- BMerge Algorithm



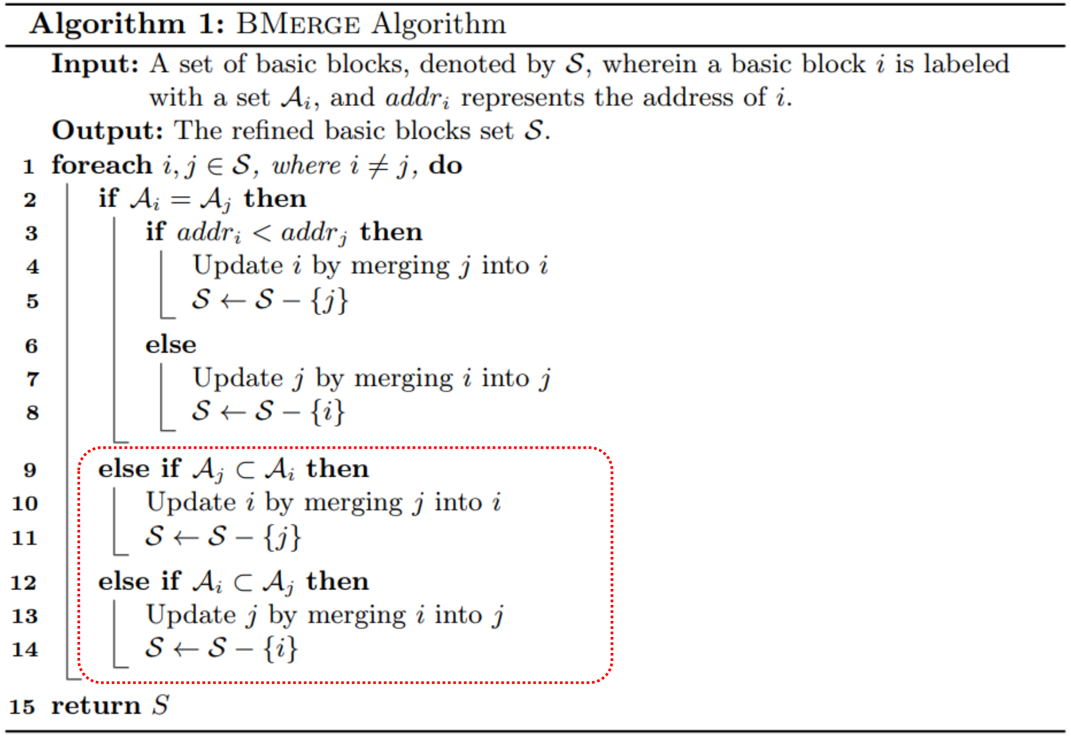
Algorithm 1: BMERGE Algorithm

```
Input: A set of basic blocks, denoted by $\mathcal{S}$, wherein a basic block $i$ is labeled
       with a set $\mathcal{A}_i$, and $addr_i$ represents the address of $i$.
Output: The refined basic blocks set $\mathcal{S}$.
foreach $i, j \in \mathcal{S}$, where $i \neq j$, do
    if $\mathcal{A}_i = \mathcal{A}_j$ then
        if $addr_i < addr_j$ then
            Update $i$ by merging $j$ into $i$
            $\mathcal{S} \leftarrow \mathcal{S} - \{j\}$
        else
            Update $j$ by merging $i$ into $j$
            $\mathcal{S} \leftarrow \mathcal{S} - \{i\}$
    else if $\mathcal{A}_j \subset \mathcal{A}_i$ then
        Update $i$ by merging $j$ into $i$
        $\mathcal{S} \leftarrow \mathcal{S} - \{j\}$
    else if $\mathcal{A}_i \subset \mathcal{A}_j$ then
        Update $j$ by merging $i$ into $j$
        $\mathcal{S} \leftarrow \mathcal{S} - \{i\}$
return $S$
```



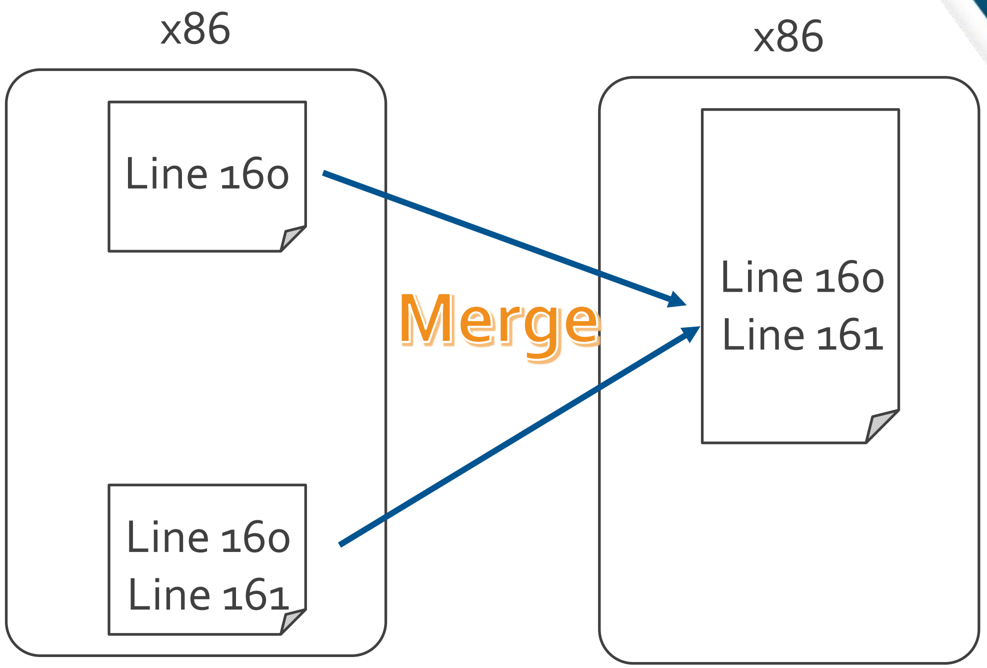
Methodology

- BMerge Algorithm



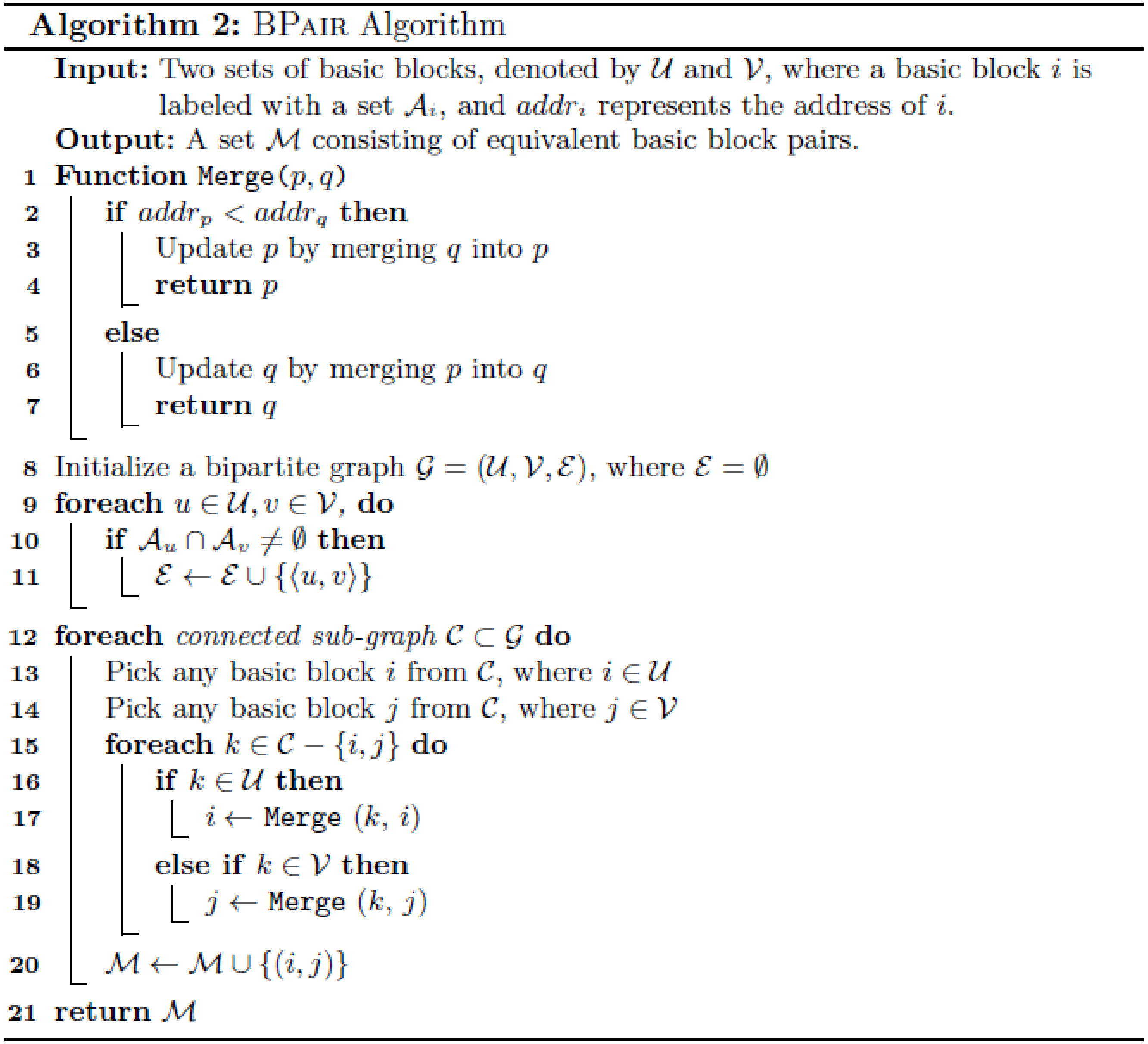
Algorithm 1: BMERGE Algorithm

```
Input: A set of basic blocks, denoted by $\mathcal{S}$, wherein a basic block $i$ is labeled
       with a set $\mathcal{A}_i$, and $addr_i$ represents the address of $i$.
Output: The refined basic blocks set $\mathcal{S}$.
foreach $i, j \in \mathcal{S}$, where $i \neq j$, do
    if $\mathcal{A}_i = \mathcal{A}_j$ then
        if $addr_i < addr_j$ then
            Update $i$ by merging $j$ into $i$
            $\mathcal{S} \leftarrow \mathcal{S} - \{j\}$
        else
            Update $j$ by merging $i$ into $j$
            $\mathcal{S} \leftarrow \mathcal{S} - \{i\}$
    else if $\mathcal{A}_j \subset \mathcal{A}_i$ then
        Update $i$ by merging $j$ into $i$
        $\mathcal{S} \leftarrow \mathcal{S} - \{j\}$
    else if $\mathcal{A}_i \subset \mathcal{A}_j$ then
        Update $j$ by merging $i$ into $j$
        $\mathcal{S} \leftarrow \mathcal{S} - \{i\}$
return $S$
```



Methodology

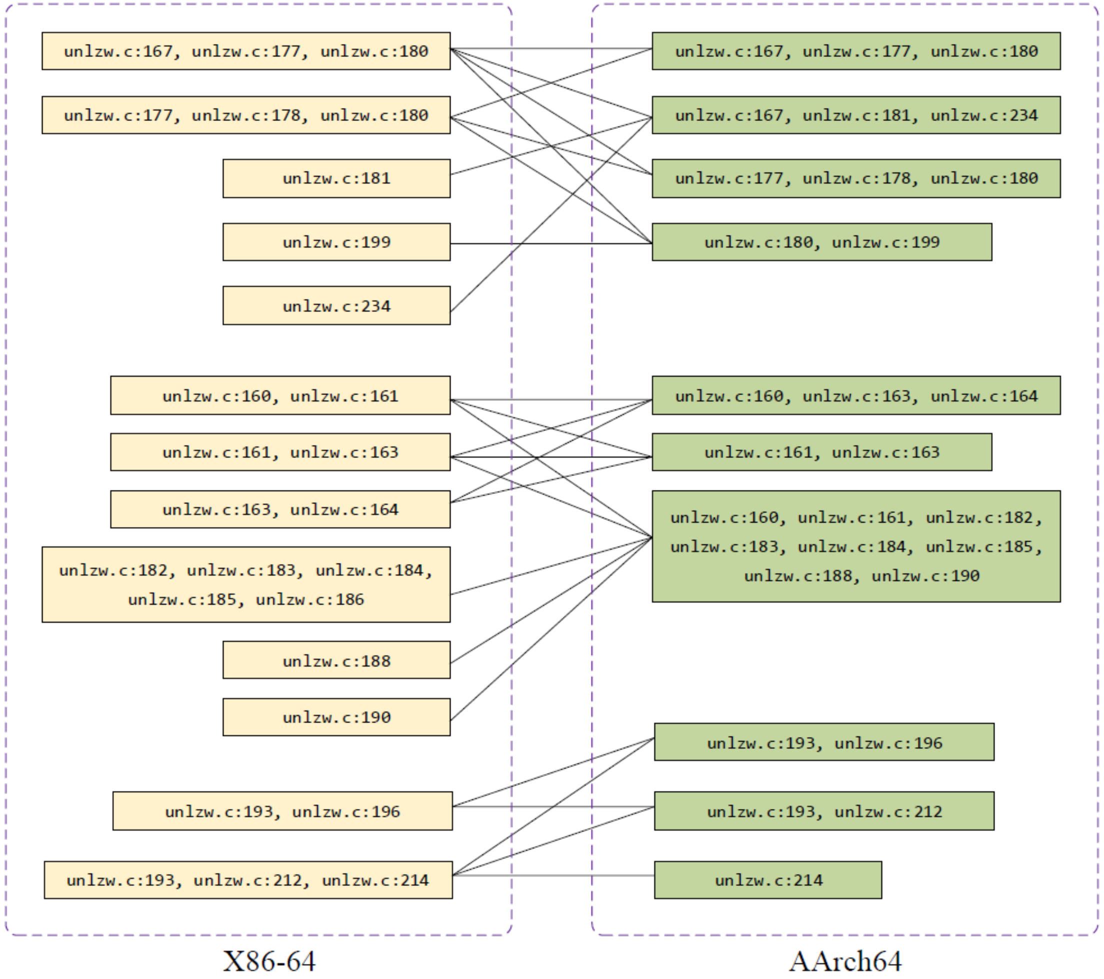
- BPair Algorithm
- Intuition: to transform the problem of matching equivalent basic blocks into a graph problem.

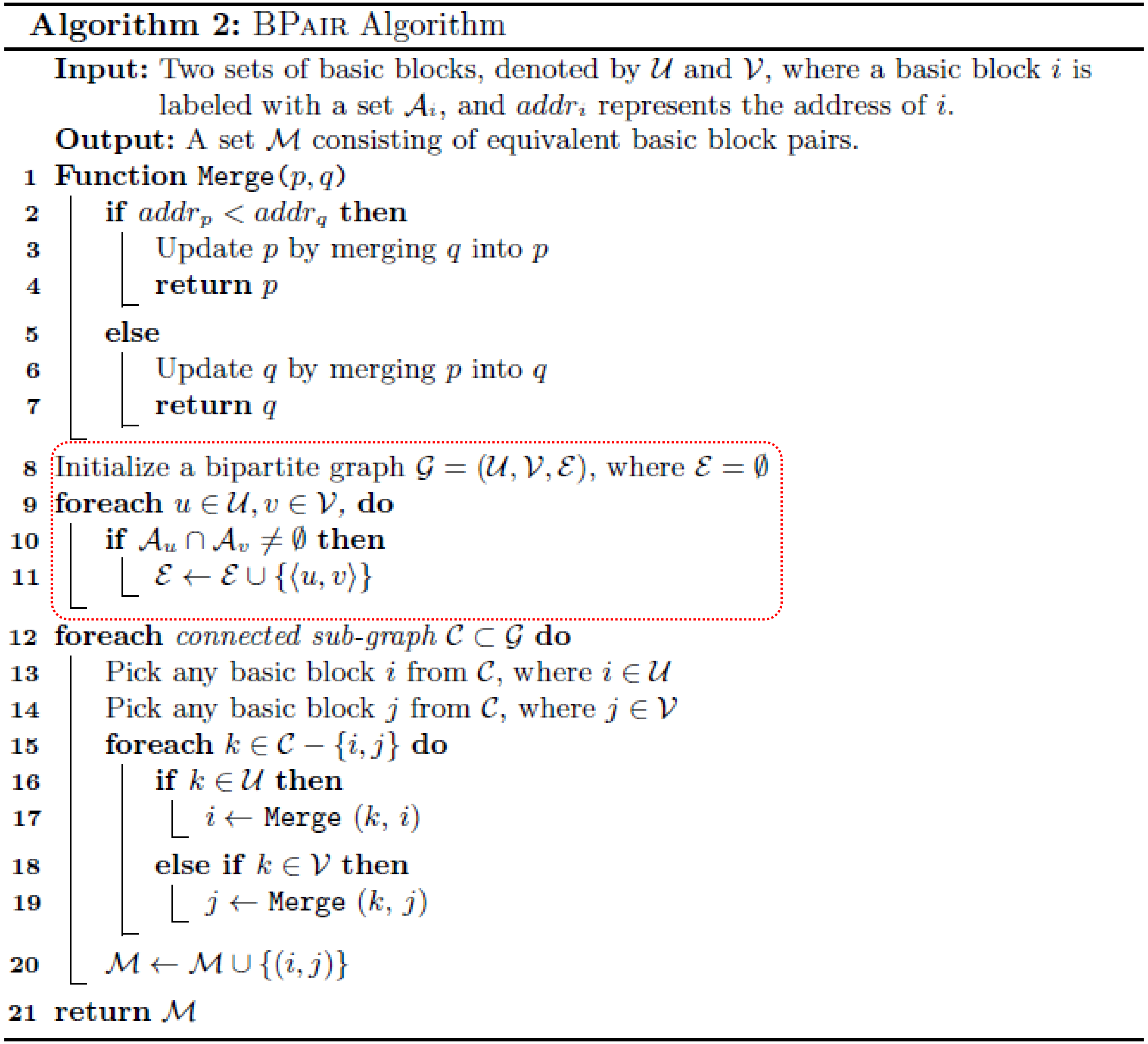


Algorithm 2: BPair Algorithm

```
Input: Two sets of basic blocks, denoted by $\mathcal{U}$ and $\mathcal{V}$, where a basic block $i$ is
       labeled with a set $\mathcal{A}_i$, and $addr_i$ represents the address of $i$.
Output: A set $\mathcal{M}$ consisting of equivalent basic block pairs.
Function Merge$(p, q)$
    if $addr_p < addr_q$ then
        Update $p$ by merging $q$ into $p$
        return $p$
    else
        Update $q$ by merging $p$ into $q$
        return $q$
Initialize a bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$, where $\mathcal{E} = \emptyset$
foreach $u \in \mathcal{U}, v \in \mathcal{V}$, do
    if $\mathcal{A}_u \cap \mathcal{A}_v \neq \emptyset$ then
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle u, v \rangle\}$
foreach connected sub-graph $\mathcal{C} \subset \mathcal{G}$ do
    Pick any basic block $i$ from $\mathcal{C}$, where $i \in \mathcal{U}$
    Pick any basic block $j$ from $\mathcal{C}$, where $j \in \mathcal{V}$
    foreach $k \in \mathcal{C} - \{i, j\}$ do
        if $k \in \mathcal{U}$ then
            $i \leftarrow$ Merge $(k, i)$
        else if $k \in \mathcal{V}$ then
            $j \leftarrow$ Merge $(k, j)$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i, j)\}$
return $\mathcal{M}$
```

Methodology

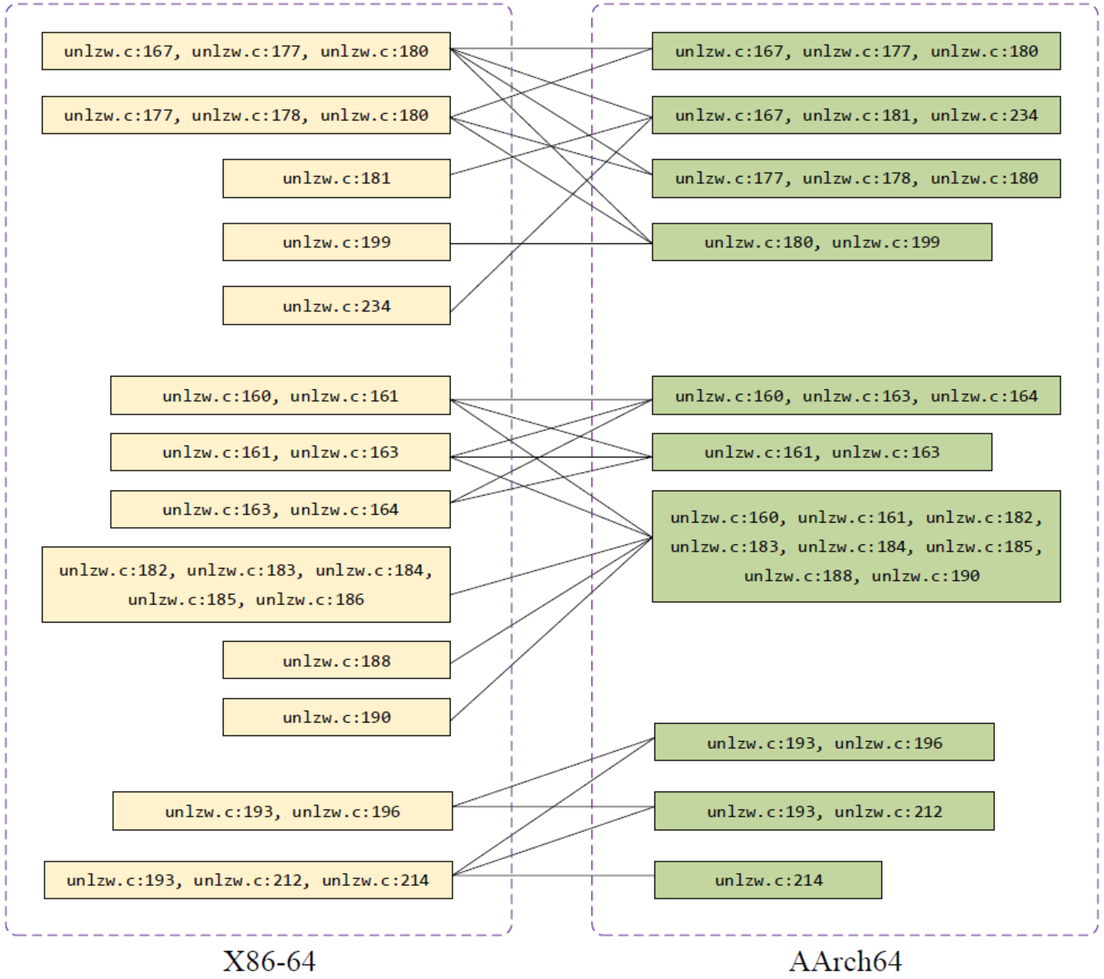

unlzw.c:167, unlzw.c:177, unlzw.c:180
unlzw.c:177, unlzw.c:178, unlzw.c:180
unlzw.c:181
unlzw.c:199
unlzw.c:234
unlzw.c:160, unlzw.c:161
unlzw.c:161, unlzw.c:163
unlzw.c:163, unlzw.c:164
unlzw.c:182, unlzw.c:183, unlzw.c:184, unlzw.c:185, unlzw.c:186
unlzw.c:188
unlzw.c:190
unlzw.c:193, unlzw.c:196
unlzw.c:193, unlzw.c:212, unlzw.c:214
unlzw.c:167, unlzw.c:177, unlzw.c:180
unlzw.c:167, unlzw.c:181, unlzw.c:234
unlzw.c:177, unlzw.c:178, unlzw.c:180
unlzw.c:180, unlzw.c:199
unlzw.c:160, unlzw.c:163, unlzw.c:164
unlzw.c:161, unlzw.c:163
unlzw.c:160, unlzw.c:161, unlzw.c:182, unlzw.c:183, unlzw.c:184, unlzw.c:185, unlzw.c:188, unlzw.c:190
unlzw.c:193, unlzw.c:196
unlzw.c:193, unlzw.c:212
unlzw.c:214
X86-64
AArch64

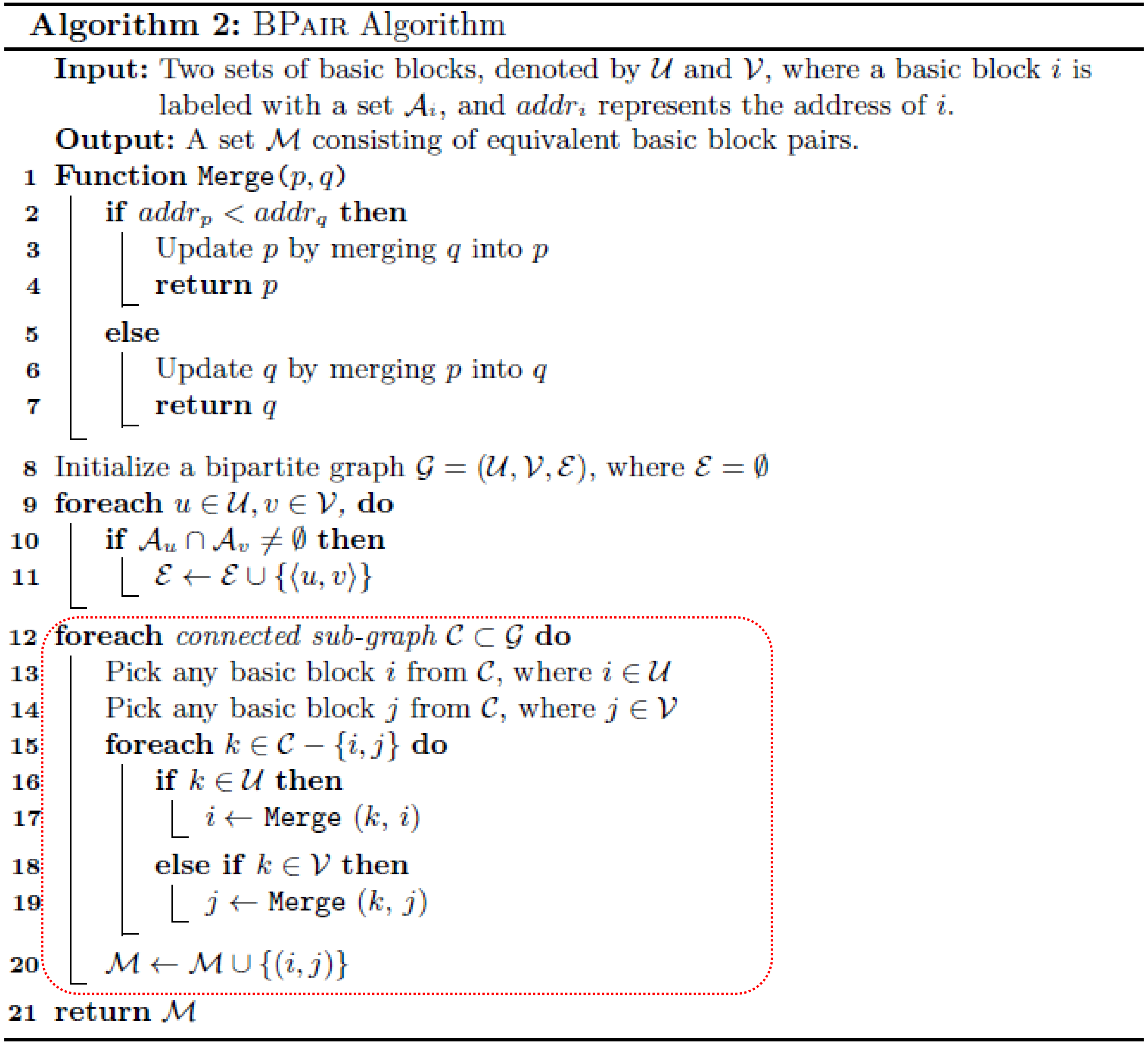


Algorithm 2: BPair Algorithm

```
Input: Two sets of basic blocks, denoted by $\mathcal{U}$ and $\mathcal{V}$, where a basic block $i$ is
       labeled with a set $\mathcal{A}_i$, and $addr_i$ represents the address of $i$.
Output: A set $\mathcal{M}$ consisting of equivalent basic block pairs.
Function Merge($p, q$)
    if $addr_p < addr_q$ then
        Update $p$ by merging $q$ into $p$
        return $p$
    else
        Update $q$ by merging $p$ into $q$
        return $q$
Initialize a bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$, where $\mathcal{E} = \emptyset$
foreach $u \in \mathcal{U}, v \in \mathcal{V}$, do
    if $\mathcal{A}_u \cap \mathcal{A}_v \neq \emptyset$ then
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle u, v \rangle\}$
foreach connected sub-graph $\mathcal{C} \subset \mathcal{G}$ do
    Pick any basic block $i$ from $\mathcal{C}$, where $i \in \mathcal{U}$
    Pick any basic block $j$ from $\mathcal{C}$, where $j \in \mathcal{V}$
    foreach $k \in \mathcal{C} - \{i, j\}$ do
        if $k \in \mathcal{U}$ then
            $i \leftarrow$ Merge ($k$, $i$)
        else if $k \in \mathcal{V}$ then
            $j \leftarrow$ Merge ($k$, $j$)
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i, j)\}$
return $\mathcal{M}$
```

Methodology


unlzw.c:167, unlzw.c:177, unlzw.c:180
unlzw.c:177, unlzw.c:178, unlzw.c:180
unlzw.c:181
unlzw.c:199
unlzw.c:234
unlzw.c:160, unlzw.c:161
unlzw.c:161, unlzw.c:163
unlzw.c:163, unlzw.c:164
unlzw.c:182, unlzw.c:183, unlzw.c:184, unlzw.c:185, unlzw.c:186
unlzw.c:188
unlzw.c:190
unlzw.c:193, unlzw.c:196
unlzw.c:193, unlzw.c:212, unlzw.c:214
unlzw.c:167, unlzw.c:177, unlzw.c:180
unlzw.c:167, unlzw.c:181, unlzw.c:234
unlzw.c:177, unlzw.c:178, unlzw.c:180
unlzw.c:180, unlzw.c:199
unlzw.c:160, unlzw.c:163, unlzw.c:164
unlzw.c:161, unlzw.c:163
unlzw.c:160, unlzw.c:161, unlzw.c:182, unlzw.c:183, unlzw.c:184, unlzw.c:185, unlzw.c:188, unlzw.c:190
unlzw.c:193, unlzw.c:196
unlzw.c:193, unlzw.c:212
unlzw.c:214
X86-64
AArch64



Algorithm 2: BPair Algorithm

Input: Two sets of basic blocks, denoted by $\mathcal{U}$ and $\mathcal{V}$, where a basic block i is labeled with a set $\mathcal{A}_i$, and $addr_i$ represents the address of i.

Output: A set $\mathcal{M}$ consisting of equivalent basic block pairs.

```
Function Merge($p, q$)
    if $addr_p < addr_q$ then
        Update $p$ by merging $q$ into $p$
        return $p$
    else
        Update $q$ by merging $p$ into $q$
        return $q$
Initialize a bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$, where $\mathcal{E} = \emptyset$
foreach $u \in \mathcal{U}, v \in \mathcal{V}$, do
    if $\mathcal{A}_u \cap \mathcal{A}_v \neq \emptyset$ then
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle u, v \rangle\}$
foreach connected sub-graph $\mathcal{C} \subset \mathcal{G}$ do
    Pick any basic block $i$ from $\mathcal{C}$, where $i \in \mathcal{U}$
    Pick any basic block $j$ from $\mathcal{C}$, where $j \in \mathcal{V}$
    foreach $k \in \mathcal{C} - \{i, j\}$ do
        if $k \in \mathcal{U}$ then
            $i \leftarrow$ Merge ($k$, $i$)
        else if $k \in \mathcal{V}$ then
            $j \leftarrow$ Merge ($k$, $j$)
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i, j)\}$
return $\mathcal{M}$
```

Methodology

- Transformer-based Similarity Detector

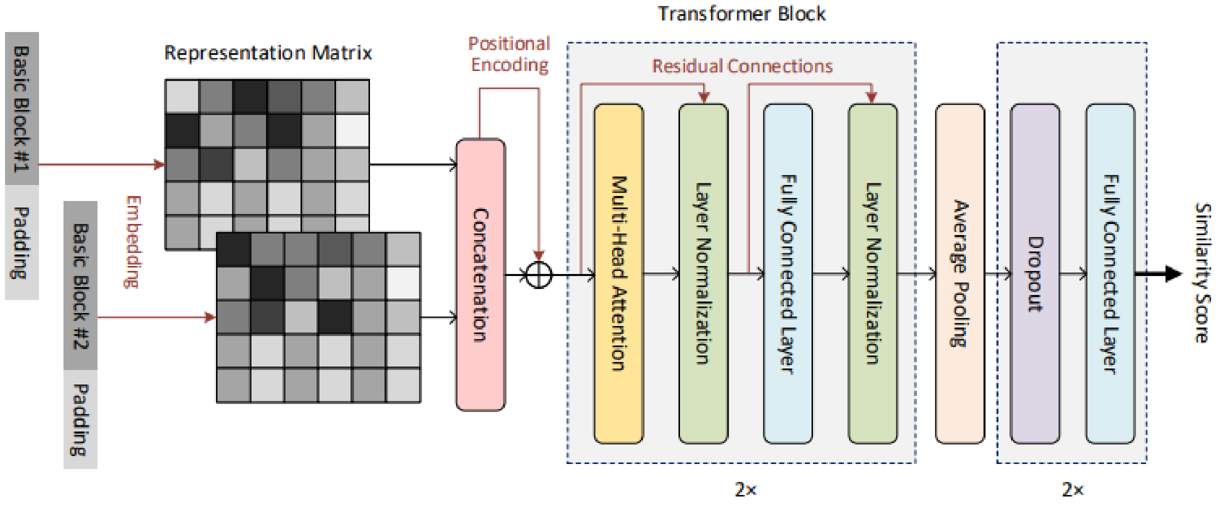


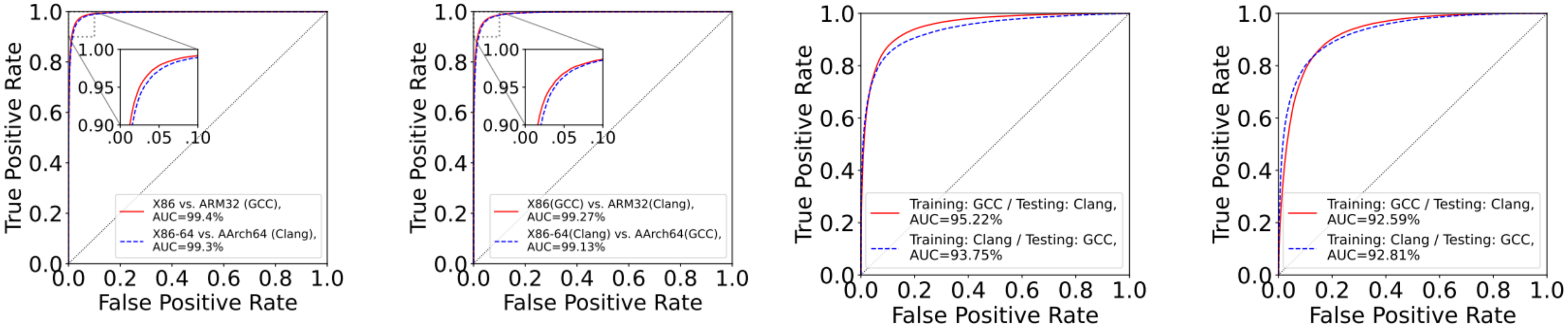
Fig. : Architecture of the proposed binary code similarity detector.

Dataset Composition

- Source code of 30 binaries from 8 GNU software projects, i.e., `binutils, datamash, findutils, grep, gzip, macchanger, tar, and which`
- Involves 980,251 functions across 32 distinct combinations of compilers, optimization levels, and target platform
  - Four different ISAs: x86, x86-64, ARM32, and AArch64
  - Two representative compilers, i.e., GCC and Clang
  - Four optimization levels: O0, O1, O2, O3
- Eventually, consisting of 4,426,258 equivalent assembly pairs

Application in Similarity Detection

- Cross-ISAs & opt-levels evaluation
- Cross-ISAs, opt-levels & compilers evaluation
- Transferability evaluation



(a) Cross-ISAs & opt-levels

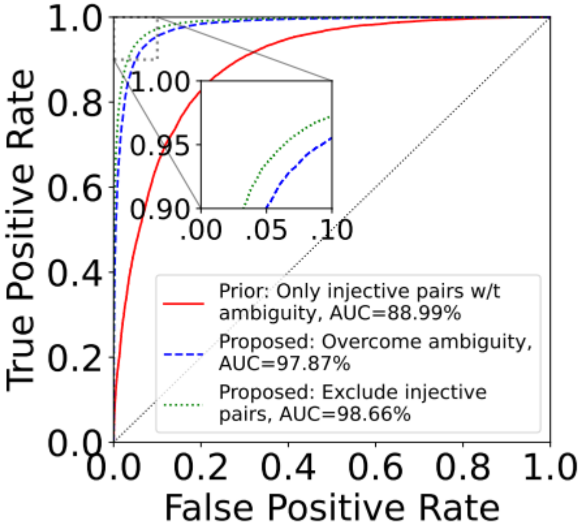
(b) Cross-ISAs, opt-levels & compilers

(c) x86 vs. ARM32

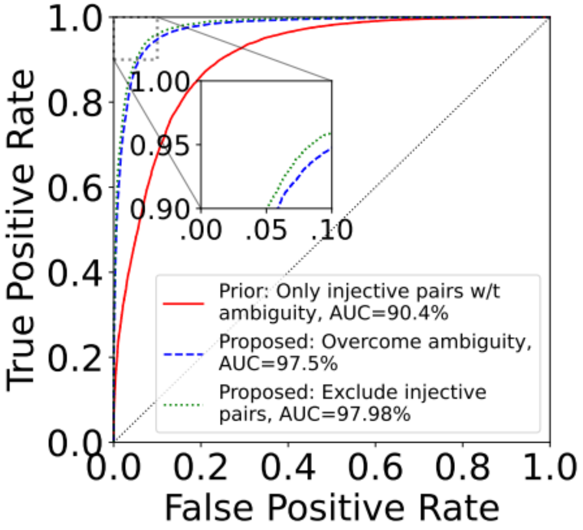
(d) x86-64 vs. AArch64

Comparison Study

- Based on the same set of source code, we apply [NDSS'19] and the proposed method to construct equivalent binary pairs
- More equivalent and complicated matching pairs can be found by the proposed method



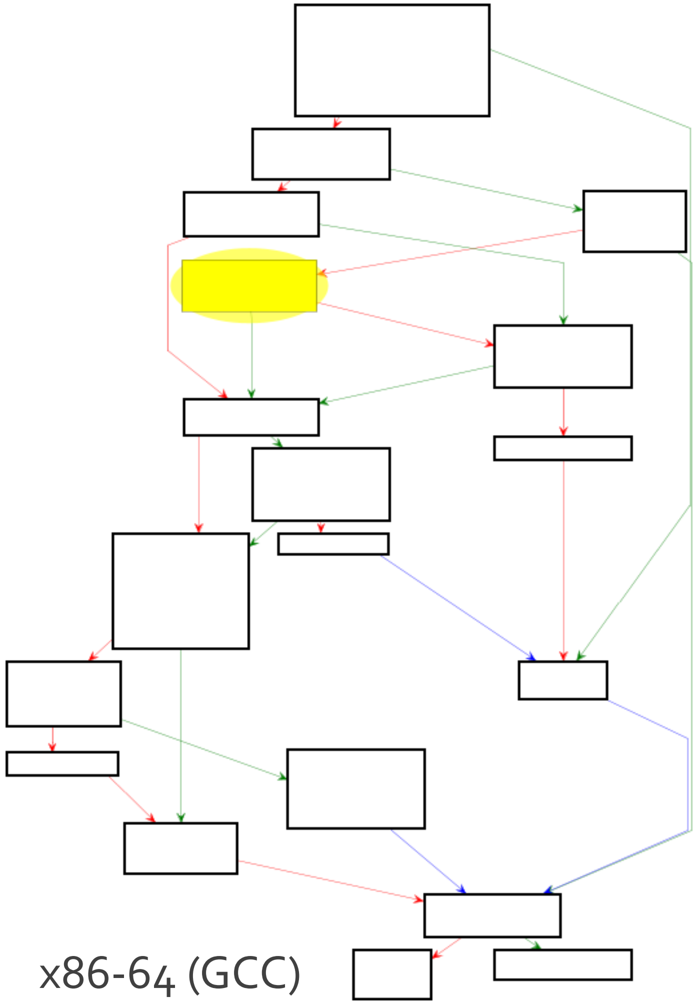
(a) x86(GCC) vs. ARM32(GCC)



(b) x86-64(Clang) vs. AArch64(GCC)

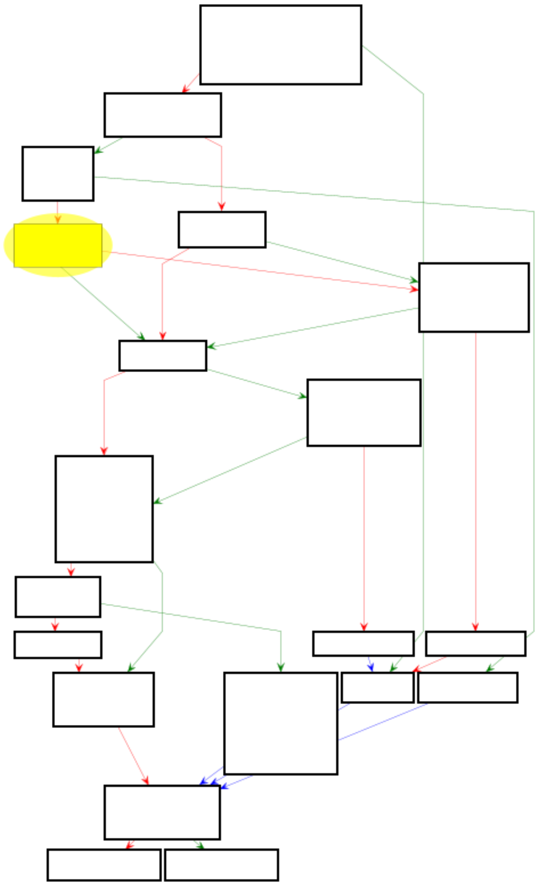
Case Study

- Patch identification
  - CVE-2019-5482 is a heap buffer overflow vulnerability in the TFTP protocol handler of libcurl.
  - The affected versions range from 7.19.4 to 7.65.3.

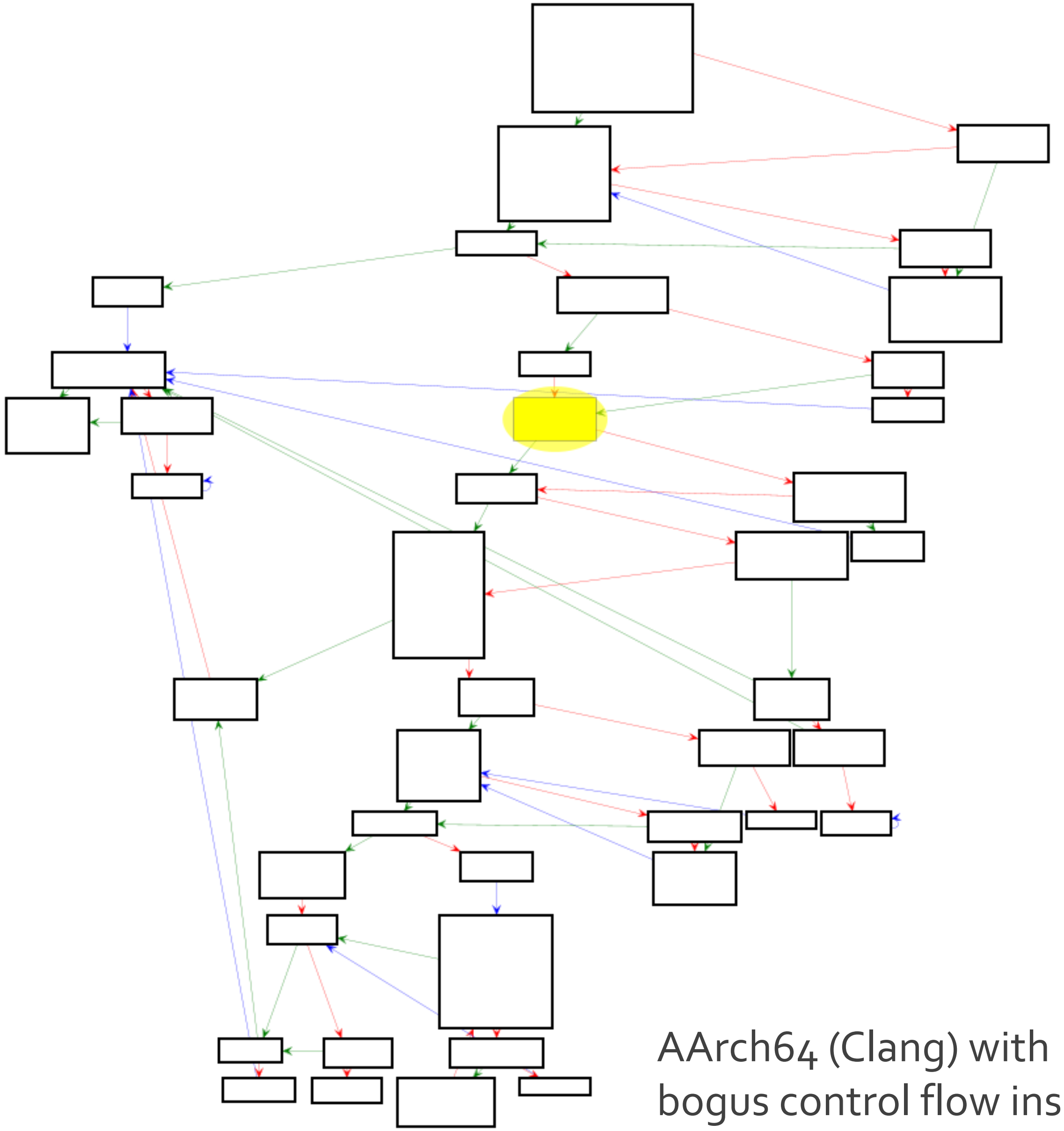


x86-64 (GCC)

Case Study



AArch64 (Clang)



AArch64 (Clang) with bogus control flow insertion

Take-away message

- We construct BinSimDB to facilitate *fine-grained* BCSA research
- The dataset and script are available at

  https://uco-cyber.github.io/research/#binsimdb

Thank you