BINALIGNER: Aligning Binary Code for Cross-Compilation Environment Diffing

Yiran Zhu[1], Tong Tang[1], Jie Wan[1], Ziqi Yang*[1,2], Zhenguang Liu*[1,2], and Lorenzo Cavallaro[3]

1 The State Key Laboratory of Blockchain and Data Security, Zhejiang University
2 Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security
3 University College London
{zhuyiran, tong.tang, wanjie, yangziqi, liuzhenguang}@zju.edu.cn, l.cavallaro@ucl.ac.uk
* Corresponding author

NDSS 2026, San Diego, February 2026

Background – Binary Diffing

Definition

align code portions corresponding to the same source code snippets between two binaries

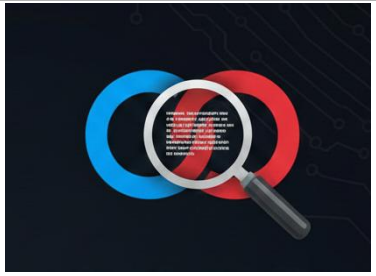
Background – Binary Diffing

Definition

align code portions corresponding to the same source code snippets between two binaries

Application

vulnerability&patch analysis



code plagiarism detection

malware analysis

Background – Binary Diffing

Definition

align code portions corresponding to the same source code snippets between two binaries

Application

vulnerability&patch analysis

code plagiarism detection

malware analysis

Distinction

Binary Code Similarity Detection → Binary Diffing:
how similar two functions are → which parts correspond to each other

Background – Challenge and Existing Methods

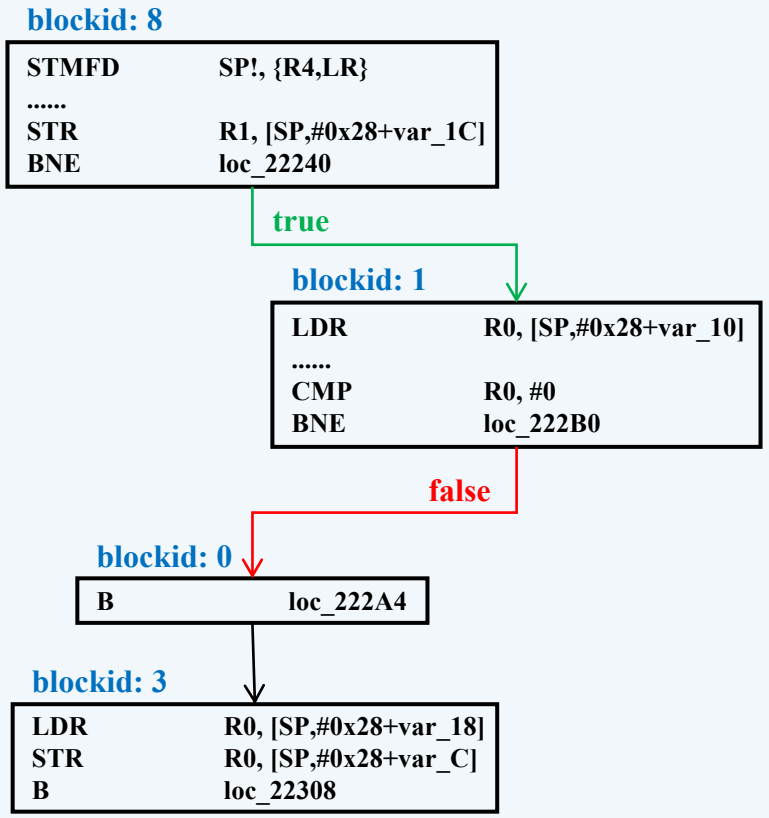
Challenge

boundaries of basic blocks are not stable across different compilation environments

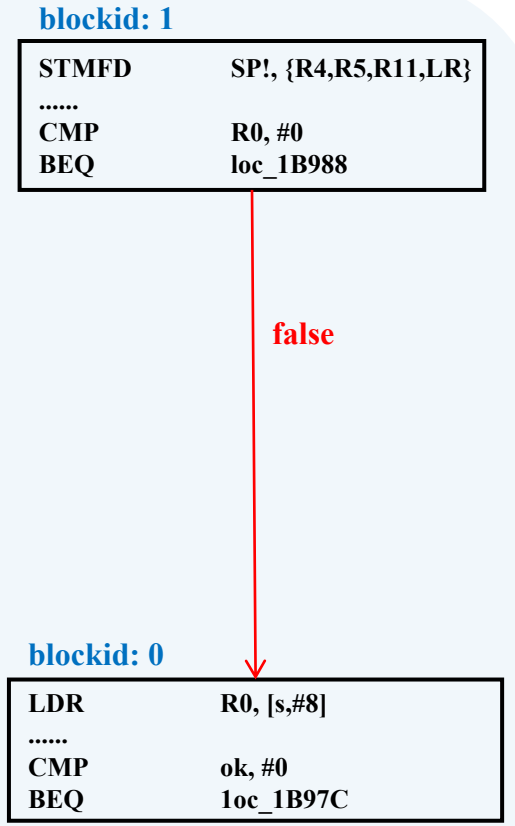
Existing Methods

- Traditional approaches: break easily
- ML-based approaches: node-level matching, rely on instruction sets

Partial CFGs of Function ssl3_check_finished



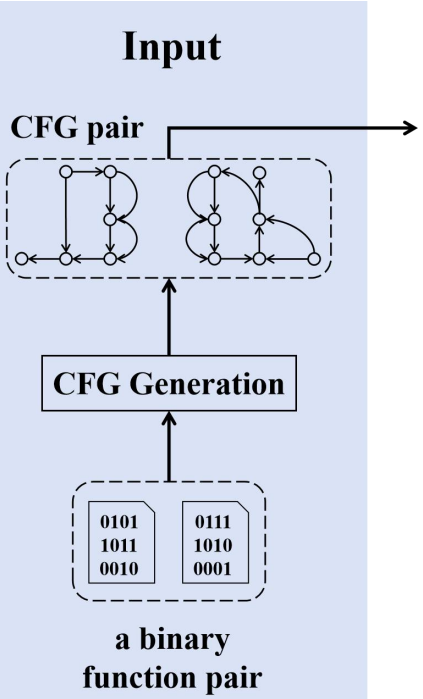
(a) OpenSSL 1.0.1f, clang-6.0, arm_32, O0



(b) OpenSSL 1.0.1f, clang-6.0, arm_32, O3

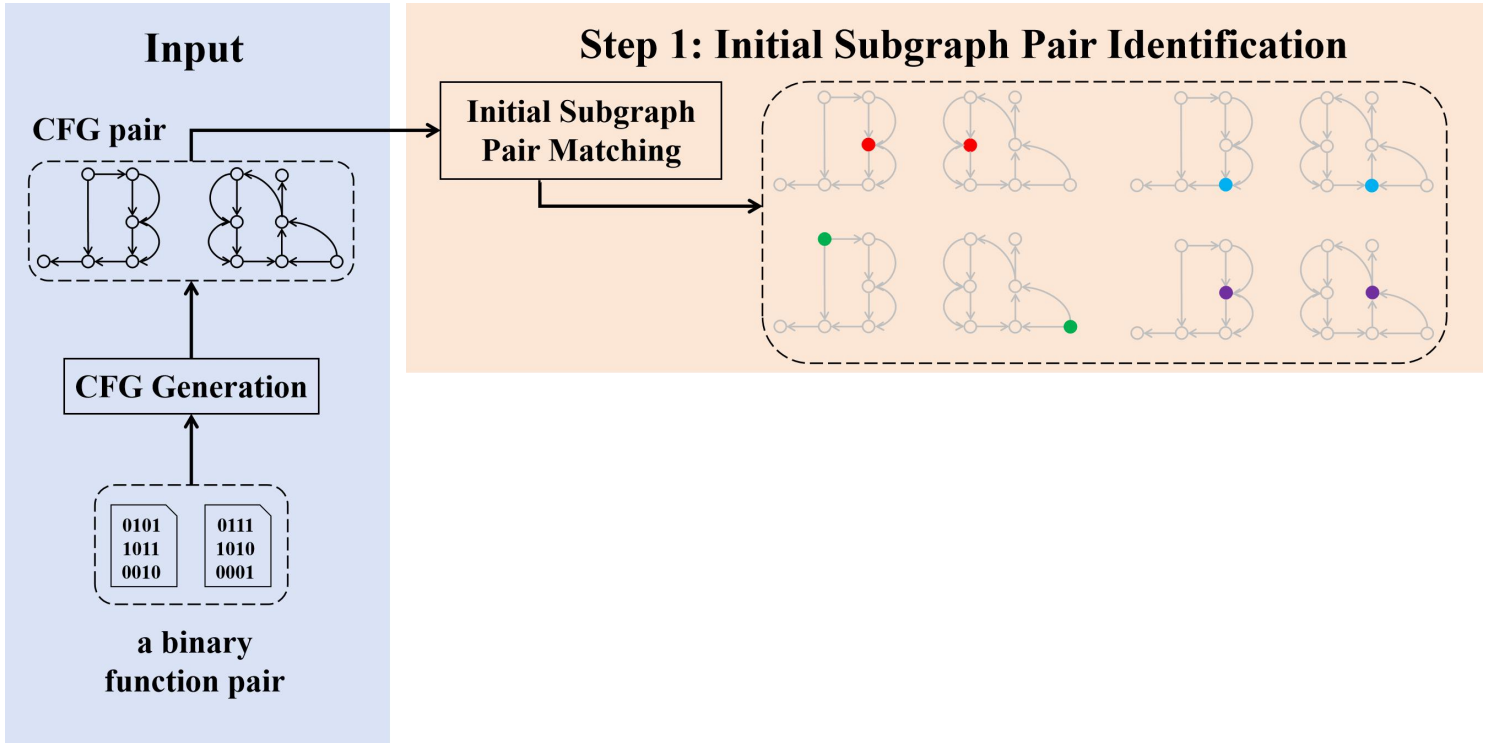
Approach – Overview

BINALIGNER: A Three-Step Pipeline for Subgraph Alignment



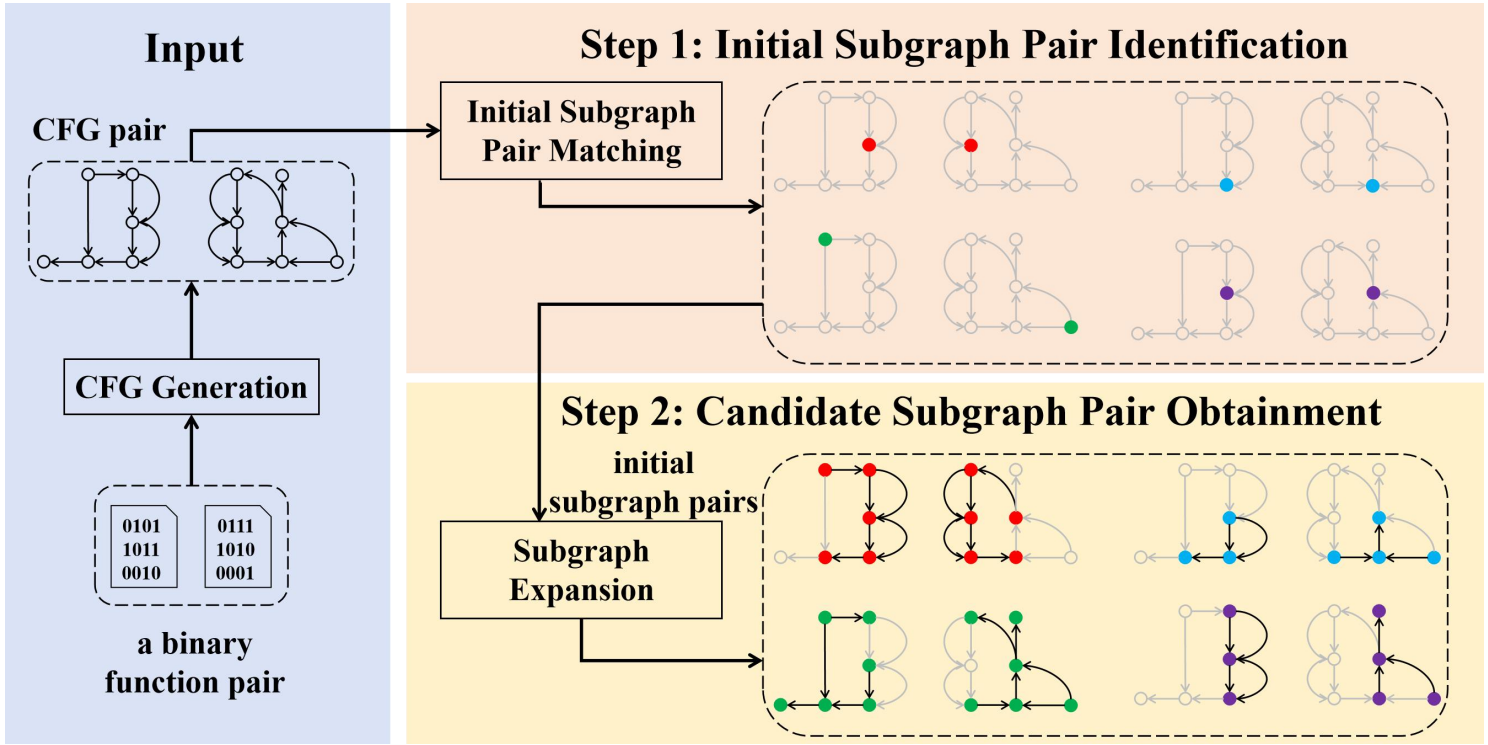
Approach – Overview

BINALIGNER: A Three-Step Pipeline for Subgraph Alignment



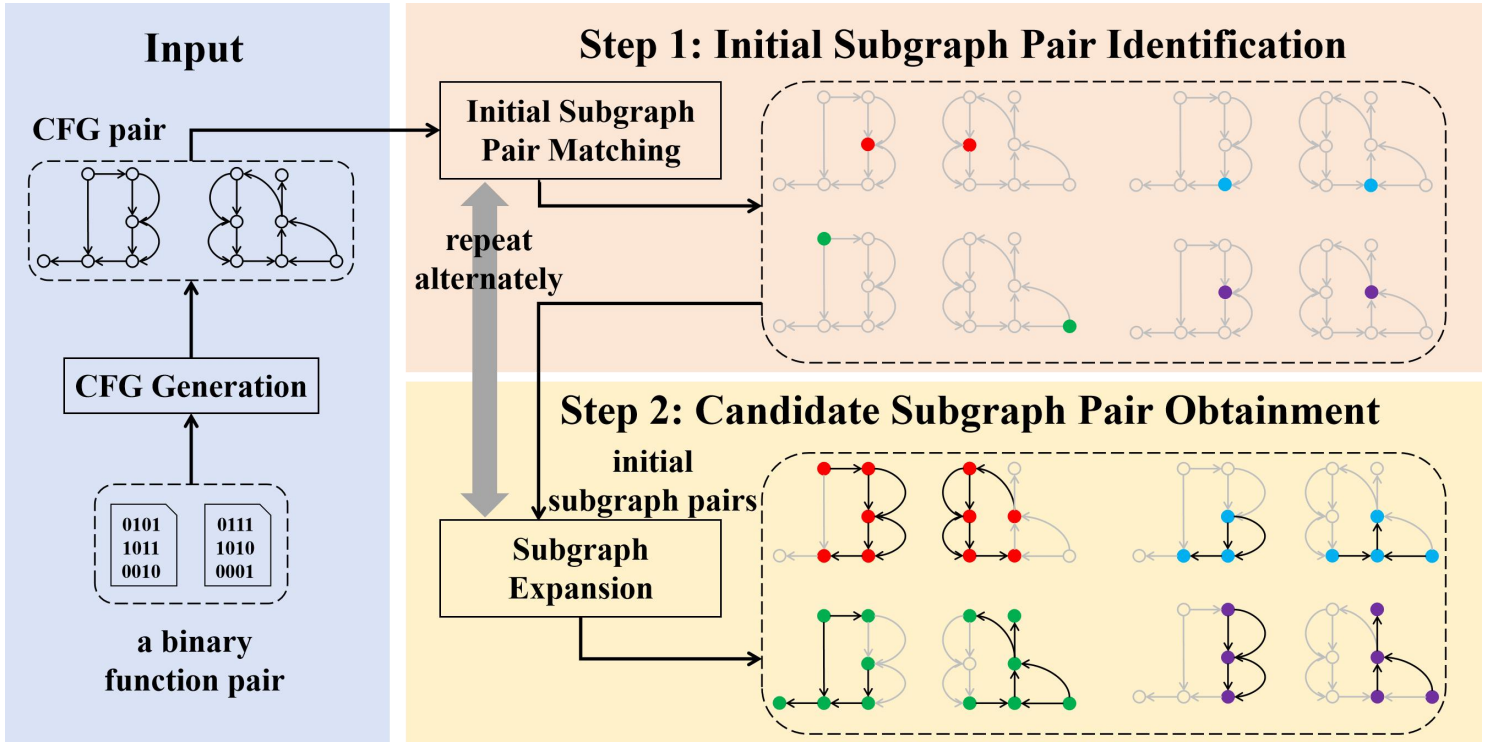
Approach – Overview

BINALIGNER: A Three-Step Pipeline for Subgraph Alignment



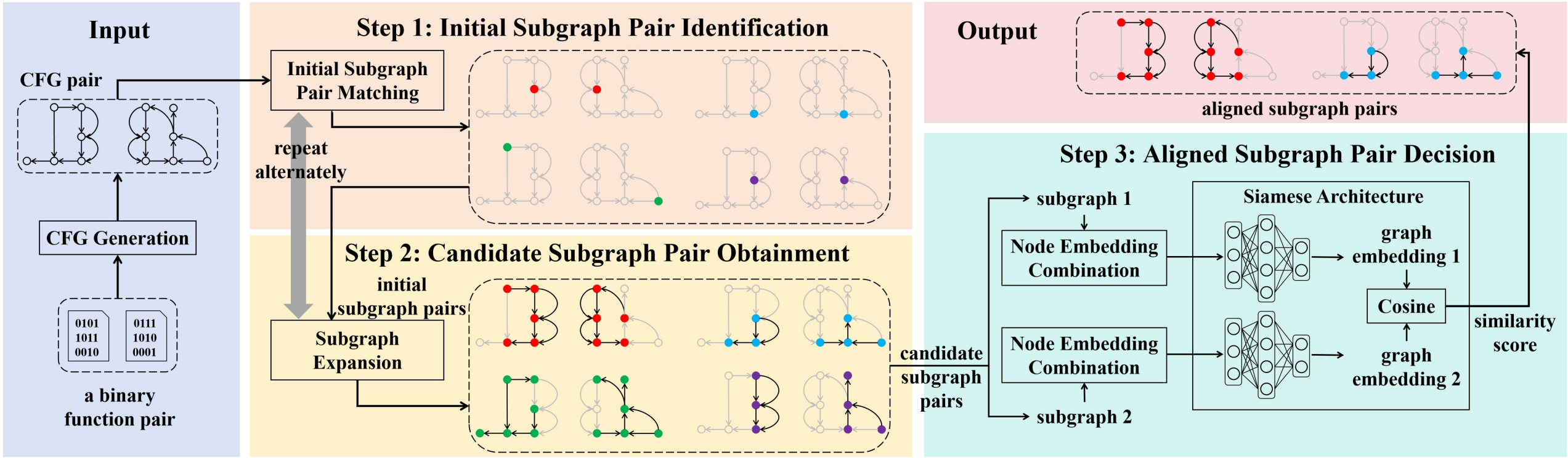
Approach – Overview

BINALIGNER: A Three-Step Pipeline for Subgraph Alignment



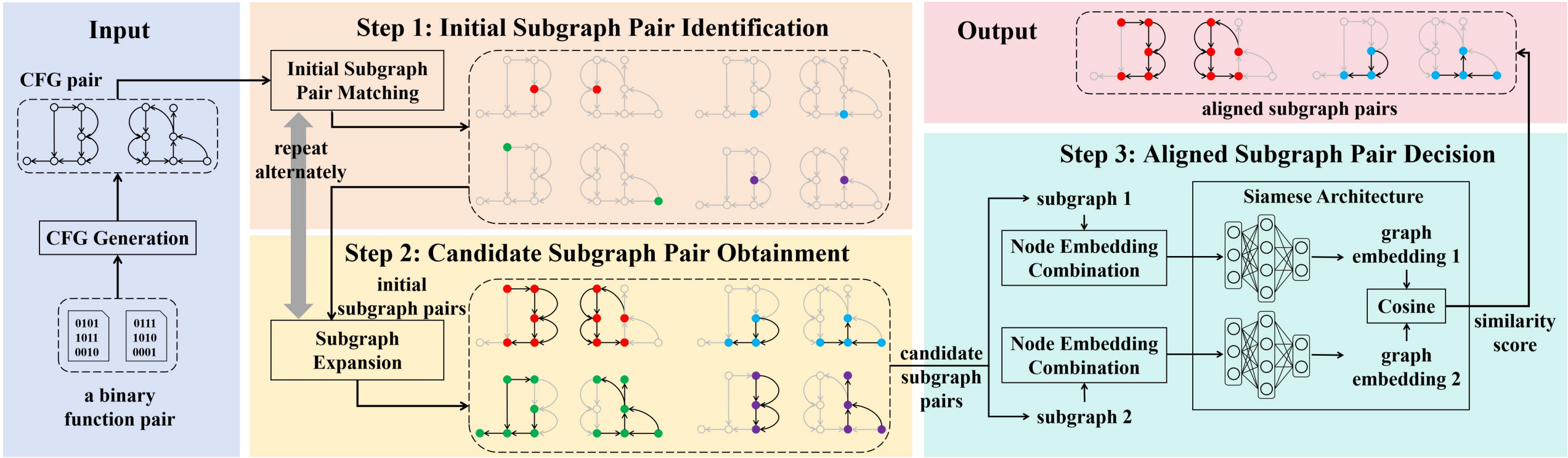
Approach – Overview

BINALIGNER: A Three-Step Pipeline for Subgraph Alignment



Approach – Overview

BINALIGNER: A Three-Step Pipeline for Subgraph Alignment



Two novelties:

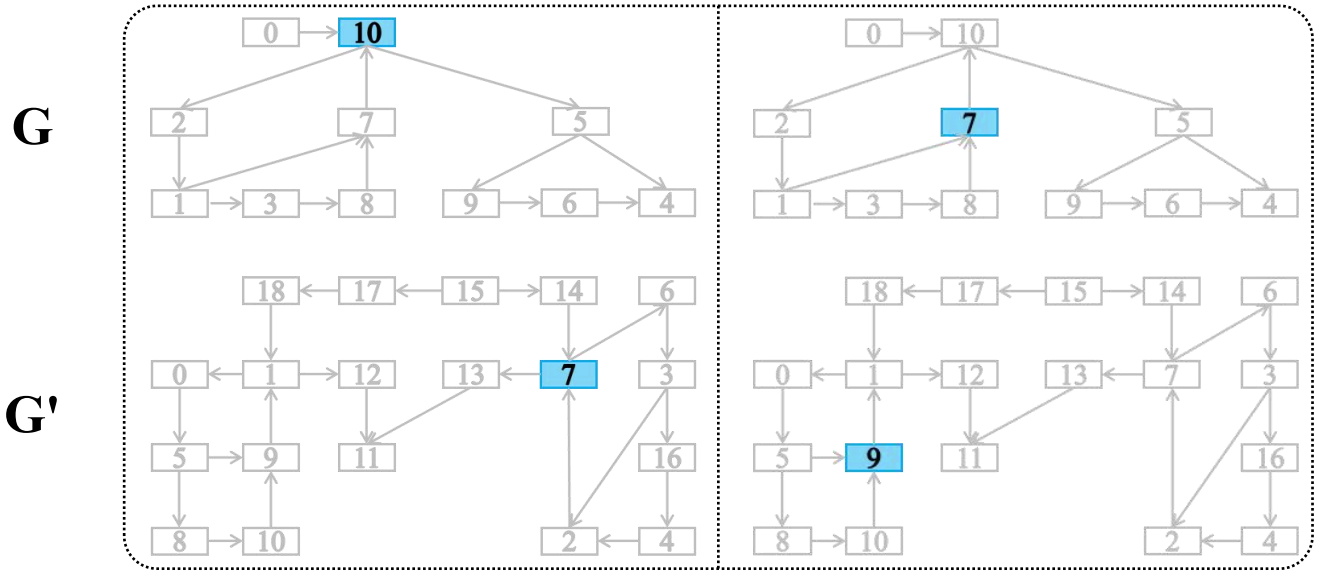
- graph-level matching instead of node-level matching
- architecture-independent node features that enable flexible cross-environment diffing

Approach – Candidate Subgraph Pair Obtainment

Step 2: Conditional Relaxation Strategies for Robust Subgraph Expansion

Chain Change <<< handle <<< $St.^{1st}$

Branch Change <<< mitigate <<< $St.^{2nd}$



Two candidate subgraph pairs expanded using two conditional relaxation strategies $St.^{1st}$ and $St.^{2nd}$

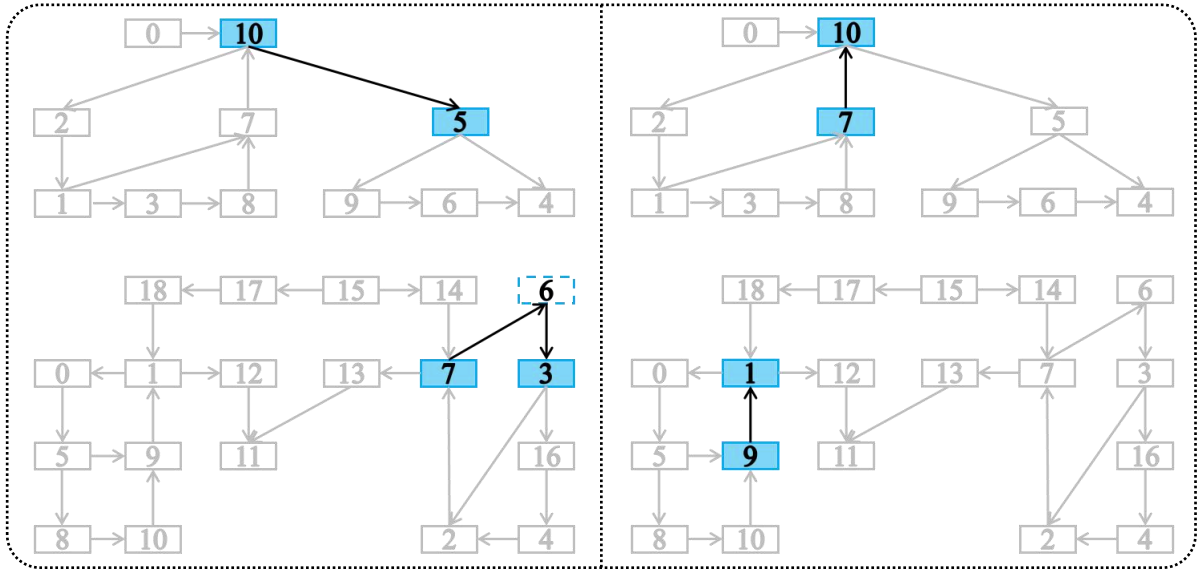
Approach – Candidate Subgraph Pair Obtainment

Step 2: Conditional Relaxation Strategies for Robust Subgraph Expansion

Chain Change <<< handle <<< $St.^{1st}$

Branch Change <<< mitigate <<< $St.^{2nd}$

In(5) = **In**(3')=1
Out(5) = **Out**(3')=2



In(10) = **In**(1')=2
Out(10) = **Out**(1')=2

Two candidate subgraph pairs expanded using two conditional relaxation strategies $St.^{1st}$ and $St.^{2nd}$

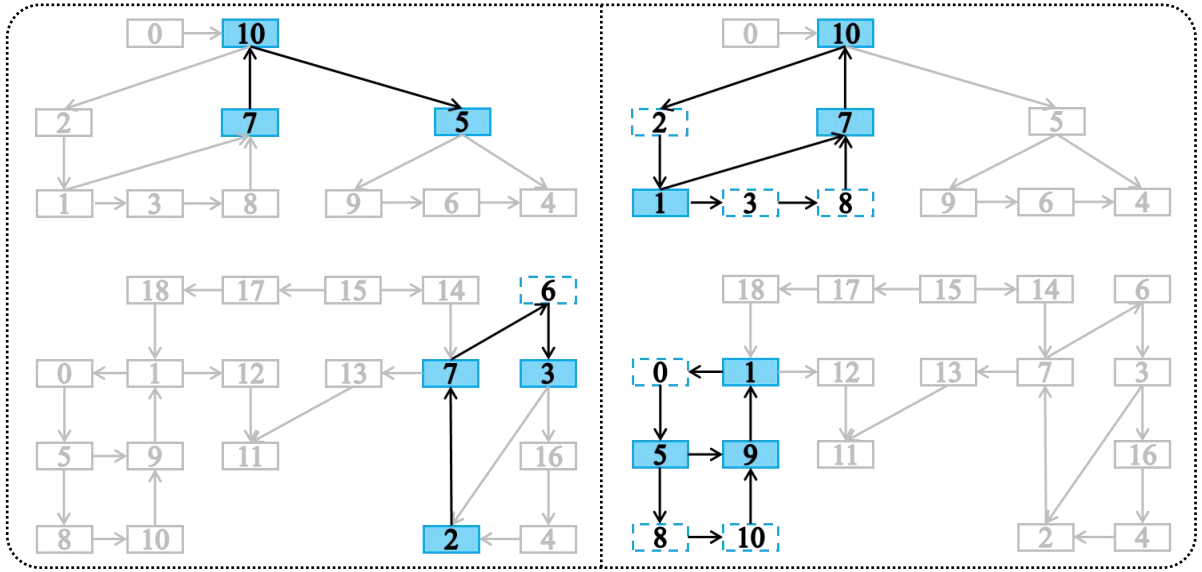
Approach – Candidate Subgraph Pair Obtainment

Step 2: Conditional Relaxation Strategies for Robust Subgraph Expansion

Chain Change << handle << *St.*^{1st}

Branch Change << mitigate << *St.*^{2nd}

In(7) = **In**(2')=2
Out(7) = **Out**(2')=1



In(1) = **In**(5')=1
Out(1) = **Out**(5')=2

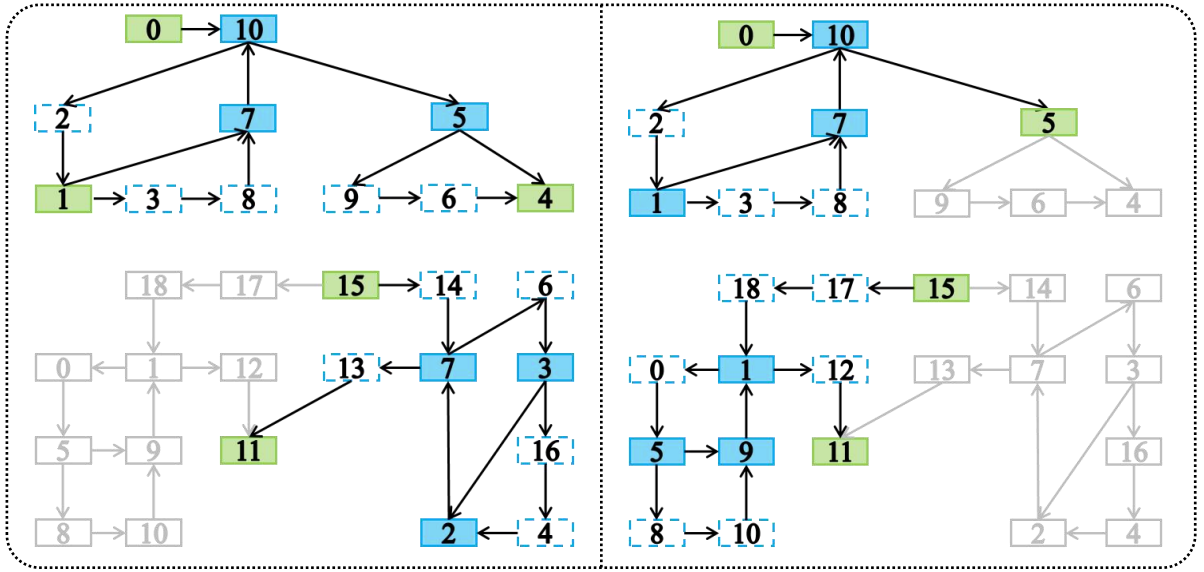
Two candidate subgraph pairs expanded using two conditional relaxation strategies *St.*^{1st} and *St.*^{2nd}

Approach – Candidate Subgraph Pair Obtainment

Step 2: Conditional Relaxation Strategies for Robust Subgraph Expansion

Chain Change << handle << $St.^{1st}$

Branch Change << mitigate << $St.^{2nd}$



Two candidate subgraph pairs expanded using two conditional relaxation strategies $St.^{1st}$ and $St.^{2nd}$

Evaluation – Experimental Setup

Datasets

Binary Files:

- the GNU dataset: Coreutils, Diffutils, and Findutils
- the OpenSSL dataset: libssl.so, libcrypto.so

Compilation Environments:

- Compilers: GCC (5.4/8.2), Clang (3.8/6.0/7.0)
- Optimization level: O0-O3
- Architecture: x86, ARM, and MIPS(for training only)
- Code versions: Multiple versions spanning 7-10 years

Baselines

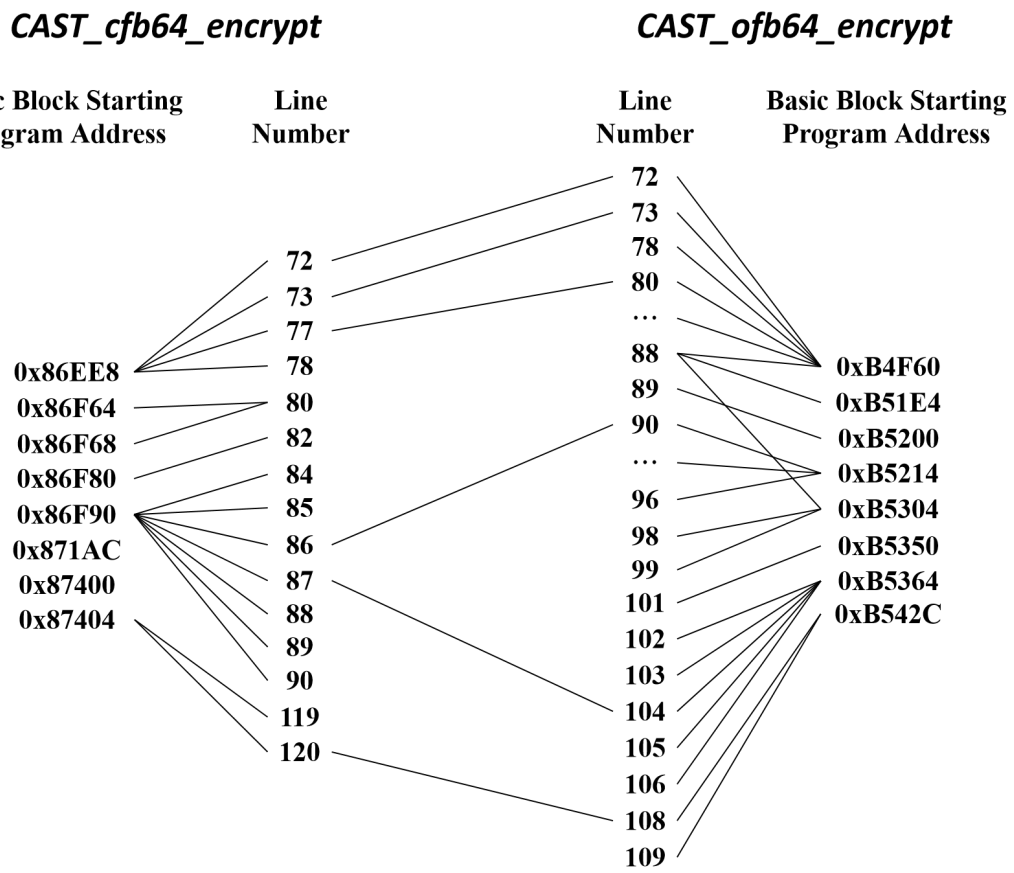
- InnerEye (IE)
- DeepBinDiff (DBD)
- DeepBinDiff-k-hop with InnerEye-embedding (DBD^{IE})
- SigmaDiff-DGMC with Gemini-embedding (SD^{G})

Evaluation – Experimental Setup

Ground Truth

The sets of nodes in subgraphs corresponding to the same source code snippets of two functions.

For example:
0x87404 ↔ 120 ↔ 108 ↔ 0xB542C



Evaluation – Four Cross-compilation Environment Scenarios

BINALIGNER performs optimally in most scenarios, especially in the cross-architecture scenario.

Cross-architecture Diffing

BINALIGNER **outperforms** the state-of-the-art methods. The highest R of baselines is 90.9% and the highest P is 73.7%, while all metrics of BINALIGNER are above 93%.

	Approach	Coreutils 8.30	Diffutils 3.6	Findutils 4.6
R	IE	0.081	0.09	0.074
	DBD^{IE}	0.098	0.121	0.1
	SD^{G}	0.34	0.366	0.371
	BA	**0.961**	**0.935**	**0.989**
	BA-0	0.941	0.897	0.953
	BA-1	0.936	0.92	0.971
	BA-GMN	0.929	0.932	0.967
P	IE	0.44	0.47	0.352
	DBD^{IE}	0.322	0.326	0.204
	SD^{G}	0.71	0.737	0.713
	BA	**0.991**	**0.977**	**1.0**
	BA-0	0.982	0.977	0.993
	BA-1	0.985	0.97	0.993
	BA-GMN	0.954	0.977	0.98
$F1$	IE	0.137	0.151	0.122
	DBD^{IE}	0.15	0.176	0.134
	SD^{G}	0.46	0.489	0.488
	BA	**0.976**	**0.956**	**0.994**
	BA-0	0.961	0.935	0.973
	BA-1	0.96	0.944	0.982
	BA-GMN	0.941	0.954	0.973

Evaluation – Complex Scenarios

BINALIGNER excels in complex scenarios (compilation environment evolution, across different similarity degrees).

Compilation Environment Evolution

	IE	DBD^{IE}	SD^{G}	BA
R	0.181	0.005	0.128	**0.688**
P	0.742	0.017	0.526	**0.921**
$F1$	0.291	0.008	0.206	**0.788**

Model train: Coreutils 8.30 (GCC-5.4, O0)
Model test: Coreutils 9.1 (GCC-8.2, O3)

BINALIGNER maintains a massive lead, with F1 49.7% to 78% higher than baselines. This demonstrates the **strongest resilience to environment changes**.

Evaluation – Case Study

BINALIGNER pinpoints real-world vulnerabilities and patches.

Practical Application

Approach	Vulnerability			Patch		
	X(A)	X(A+O)	X(A+O+C)	X(A)	X(A+O)	X(A+O+C)
IE	0.0	0.0	0.0	0.0	0.0	0.0
DBD^{IE}	**1.0**	**1.0**	0.5	0.5	0.667	0.375
SD^{G}	**1.0**	0.0	0.0	0.75	0.0	0.417
BA	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

X(A): (x86-64, O0, GCC-8.2.0) vs (ARM-64, O0, GCC-8.2.0)
X(A+O): (x86-64, O0, GCC-8.2.0) vs (ARM-64, O3, GCC-8.2.0)
X(A+O+C): (x86-64, O0, GCC-8.2.0) vs (ARM-64, O3, Clang-6.0)

BINALIGNER **achieves the highest recall** in pinpointing the vulnerable and patched code.

Evaluation – Case Study

BINALIGNER pinpoints real-world vulnerabilities and patches.

Robustness Verification

Library	Ver.	CVE	Vul./Pat.	Function	IE	DBD^{IE}	SD^{G}	BA
Binutils	2.4	CVE-2025-5245	Vul.	debug_type_samep				✓
		CVE-2025-5244	Vul.	bfd_elf_gc_sections	✓		✓	✓
		CVE-2025-1176	Vul.	_bfd_elf_gc_mark_rsec	✓		✓	✓
Tar	1.34	CVE-2023-39804	Vul.	xheader_decode	✓		✓	✓
		CVE-2023-39804	Vul.	xattr_decoder	✓		✓	✓
Libmicrohttpd	0.9.75	CVE-2023-27371	Vul.	MHD_create_post_processor	✓			✓
Inetutils	2.4	CVE-2022-39028	Pat.	telrcv	✓			✓
		CVE-2021-40491	Pat.	initconn			✓	✓
Recutils	1.9	CVE-2021-46022, CVE-2021-46019	Pat.	rec_parse_comment	✓			✓

Model train: Coreutils, Diffutils, Findutils (GCC-5.4, O0)
Model test: Binutils, Tar, Libmicrohttpd, Inetutils, Recutils (Clang-13, O3)

BINALIGNER successfully **identified all nine** vulnerabilities and patches.

Conclusion – Limitations and Future Work

BINALIGNER enables accurate and flexible cross-environment binary diffing.

Key Novelties

- graph-level matching with conditional relaxation strategies
- architecture-agnostic graph embeddings based on node statistical features

Main Results

- BINALIGNER significantly outperforms the state-of-the-art methods in most scenarios. Especially in the cross-architecture scenario and multiple combinations of cross-compilation environment scenarios.
- Two case studies using real-world vulnerabilities and patches further demonstrate the utility of BINALIGNER.

Conclusion – Limitations and Future Work

BINALIGNER enables accurate and flexible cross-environment binary diffing.

Semantic Features

richer semantic analysis
via IR lifting and NLP

Conclusion – Limitations and Future Work

BINALIGNER enables accurate and flexible cross-environment binary diffing.

Attack Robustness
stronger robustness to
complex obfuscations

Semantic Features
richer semantic analysis
via IR lifting and NLP

Conclusion – Limitations and Future Work

BINALIGNER enables accurate and flexible cross-environment binary diffing.

Attack Robustness

stronger robustness to complex obfuscations

Disassembler Dependency

support for more disassemblers

Semantic Features

richer semantic analysis via IR lifting and NLP

Conclusion – Limitations and Future Work

BINALIGNER enables accurate and flexible cross-environment binary diffing.

Attack Robustness
stronger robustness to complex obfuscations

Disassembler Dependency
support for more disassemblers

Semantic Features
richer semantic analysis via IR lifting and NLP

Indirect Jumps
better handling of indirect jumps