香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

Artifact Evaluated
NDSS
SYMPOSIUM
Available
Functional
Reproduced

Enhancing Semantic-Aware Binary Diffing with High-Confidence Dynamic Instruction Alignment

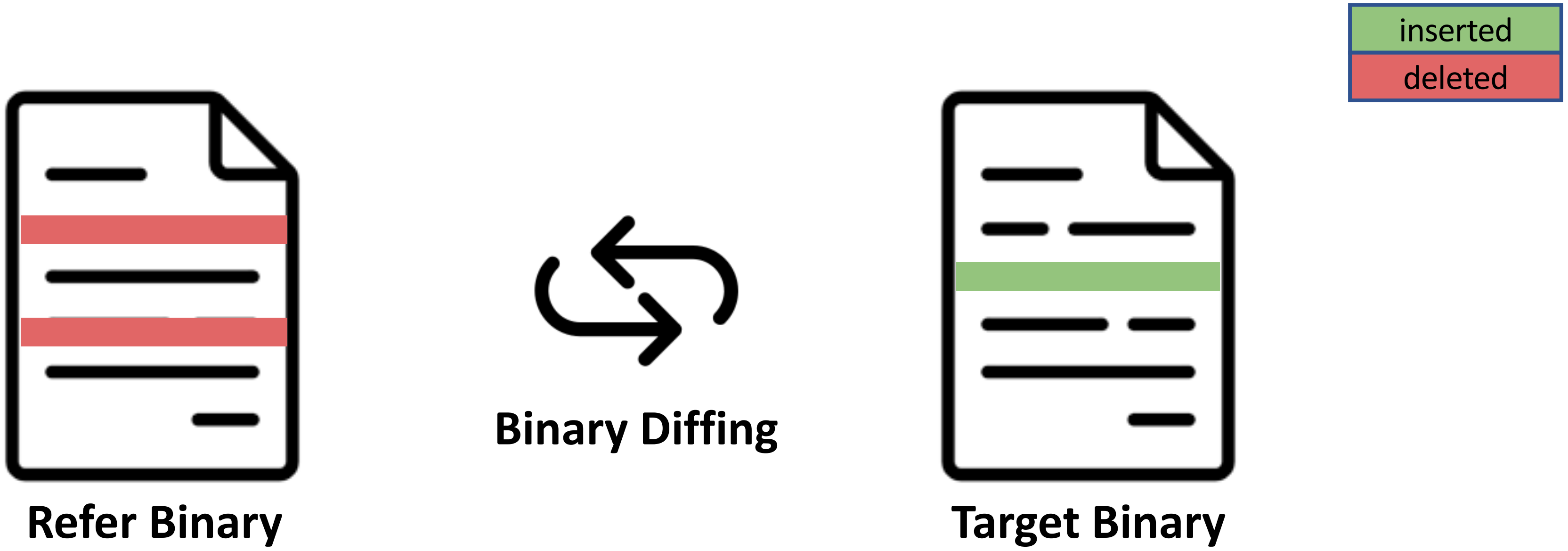
Chengfeng Ye, Anshunkang Zhou, Charles Zhang

The Hong Kong University of Science and Technology

Binary Diffing

Binary Diffing: locate similar parts and highlight differences between two binaries in a fine-grained manner

- Basic-Block Diffing
- Instruction/Pseudocode Diffing



🟩 inserted

🟥 deleted

Refer Binary

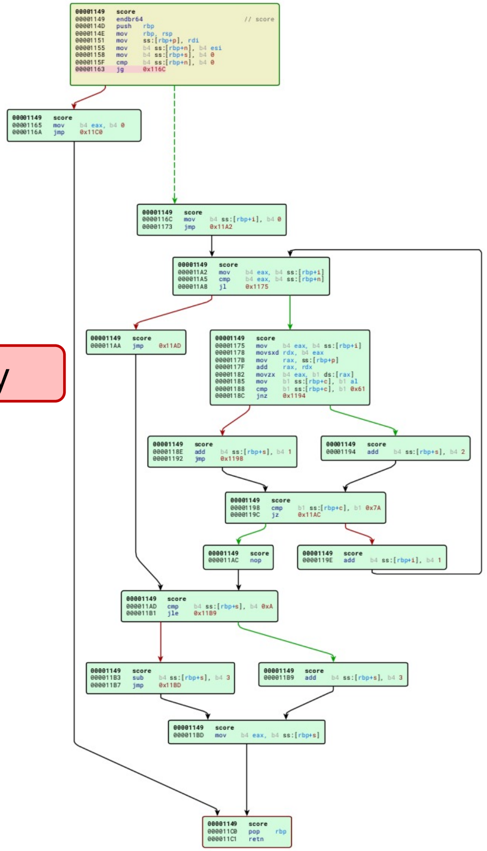
Binary Diffing

Target Binary

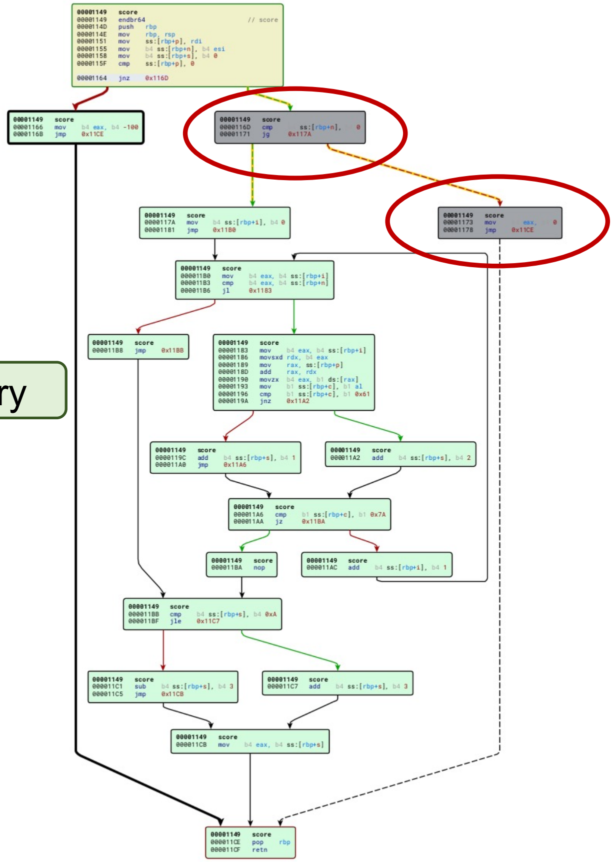
Binary Diffing Application

Security Application: locate and analyze N-day vulnerabilities

Buggy Binary



Patched Binary

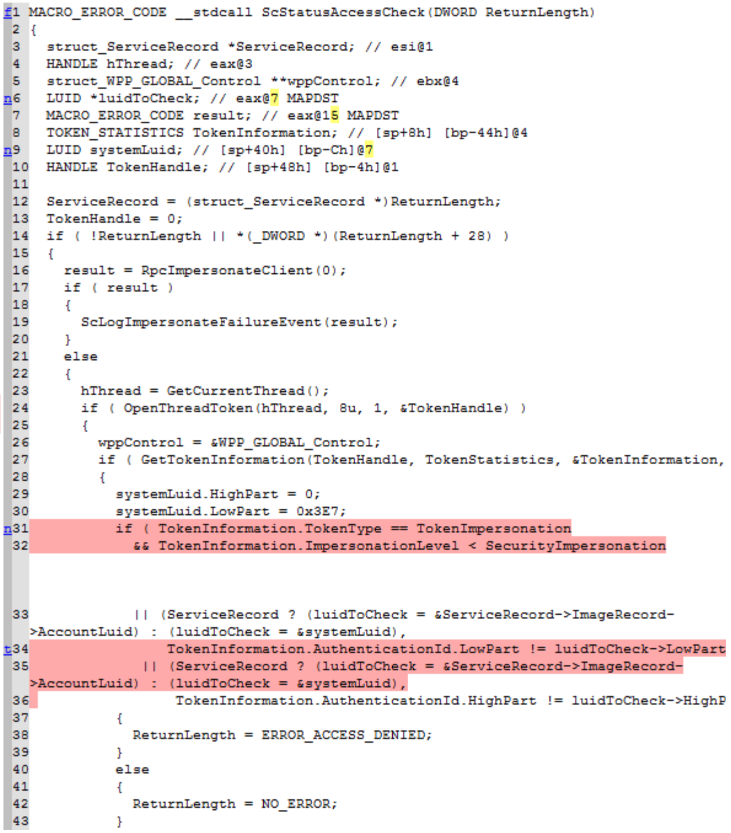


BinDiff locates changes made by a security patch to fix a Null-Pointer Dereference

Binary Diffing Application

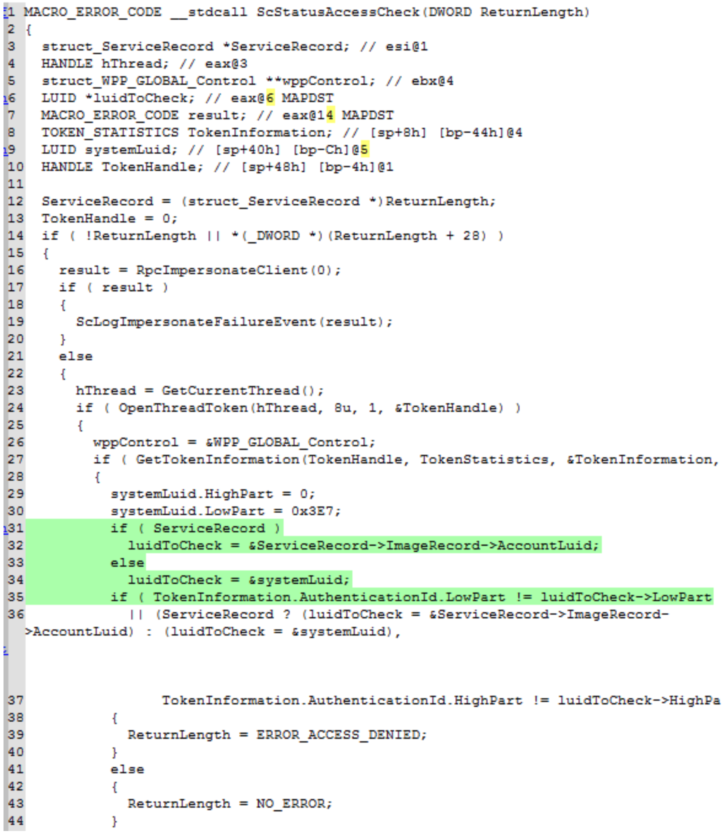
Security Application: locate and analyze N-day vulnerabilities

Buggy Binary



```
MACRO_ERROR_CODE __stdcall ScStatusAccessCheck(DWORD ReturnLength)
{
  struct_ServiceRecord *ServiceRecord; // esi@1
  HANDLE hThread; // eax@3
  struct_WPP_GLOBAL_Control **wppControl; // ebx@4
  LUID *luidToCheck; // eax@7 MAPDST
  MACRO_ERROR_CODE result; // eax@15 MAPDST
  TOKEN_STATISTICS TokenInformation; // [sp+8h] [bp-44h]@4
  LUID systemLuid; // [sp+40h] [bp-Ch]@7
  HANDLE TokenHandle; // [sp+48h] [bp-4h]@1

  ServiceRecord = (struct_ServiceRecord *)ReturnLength;
  TokenHandle = 0;
  if ( !ReturnLength || *(_DWORD *)(ReturnLength + 28) )
  {
    result = RpcImpersonateClient(0);
    if ( result )
    {
      ScLogImpersonateFailureEvent(result);
    }
    else
    {
      hThread = GetCurrentThread();
      if ( OpenThreadToken(hThread, 8u, 1, &TokenHandle) )
      {
        wppControl = &WPP_GLOBAL_Control;
        if ( GetTokenInformation(TokenHandle, TokenStatistics, &TokenInformation,
        {
          systemLuid.HighPart = 0;
          systemLuid.LowPart = 0x3E7;
          if ( TokenInformation.TokenType == TokenImpersonation
            && TokenInformation.ImpersonationLevel < SecurityImpersonation



            || (ServiceRecord ? (luidToCheck = &ServiceRecord->ImageRecord-
>AccountLuid) : (luidToCheck = &systemLuid),
                TokenInformation.AuthenticationId.LowPart != luidToCheck->LowPart
             || (ServiceRecord ? (luidToCheck = &ServiceRecord->ImageRecord-
>AccountLuid) : (luidToCheck = &systemLuid),
                TokenInformation.AuthenticationId.HighPart != luidToCheck->HighP
          {
            ReturnLength = ERROR_ACCESS_DENIED;
          }
          else
          {
            ReturnLength = NO_ERROR;
          }
```

Patched Binary

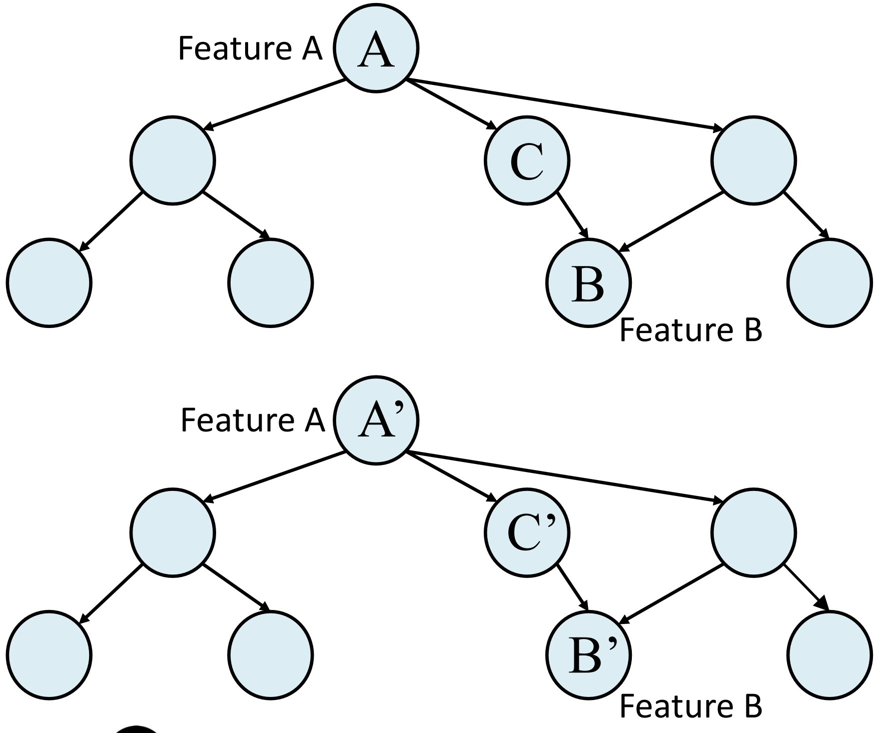


```
MACRO_ERROR_CODE __stdcall ScStatusAccessCheck(DWORD ReturnLength)
{
  struct_ServiceRecord *ServiceRecord; // esi@1
  HANDLE hThread; // eax@3
  struct_WPP_GLOBAL_Control **wppControl; // ebx@4
  LUID *luidToCheck; // eax@6 MAPDST
  MACRO_ERROR_CODE result; // eax@14 MAPDST
  TOKEN_STATISTICS TokenInformation; // [sp+8h] [bp-44h]@4
  LUID systemLuid; // [sp+40h] [bp-Ch]@5
  HANDLE TokenHandle; // [sp+48h] [bp-4h]@1

  ServiceRecord = (struct_ServiceRecord *)ReturnLength;
  TokenHandle = 0;
  if ( !ReturnLength || *(_DWORD *)(ReturnLength + 28) )
  {
    result = RpcImpersonateClient(0);
    if ( result )
    {
      ScLogImpersonateFailureEvent(result);
    }
    else
    {
      hThread = GetCurrentThread();
      if ( OpenThreadToken(hThread, 8u, 1, &TokenHandle) )
      {
        wppControl = &WPP_GLOBAL_Control;
        if ( GetTokenInformation(TokenHandle, TokenStatistics, &TokenInformation,
        {
          systemLuid.HighPart = 0;
          systemLuid.LowPart = 0x3E7;
          if ( ServiceRecord )
            luidToCheck = &ServiceRecord->ImageRecord->AccountLuid;
          else
            luidToCheck = &systemLuid;
          if ( TokenInformation.AuthenticationId.LowPart != luidToCheck->LowPart
            || (ServiceRecord ? (luidToCheck = &ServiceRecord->ImageRecord-
>AccountLuid) : (luidToCheck = &systemLuid),



                TokenInformation.AuthenticationId.HighPart != luidToCheck->HighPa
          {
            ReturnLength = ERROR_ACCESS_DENIED;
          }
          else
          {
            ReturnLength = NO_ERROR;
          }
```

Diaphora locates changes made by a security patch to fix a privilege escalation in Windows SCM

Binary Diffing with Neighborhood Consensus

Binary Diffing with Neighborhood Consensus: correctly matched nodes help identify surrounding matches [1, 2].



❶ **Anchor Point Identification**

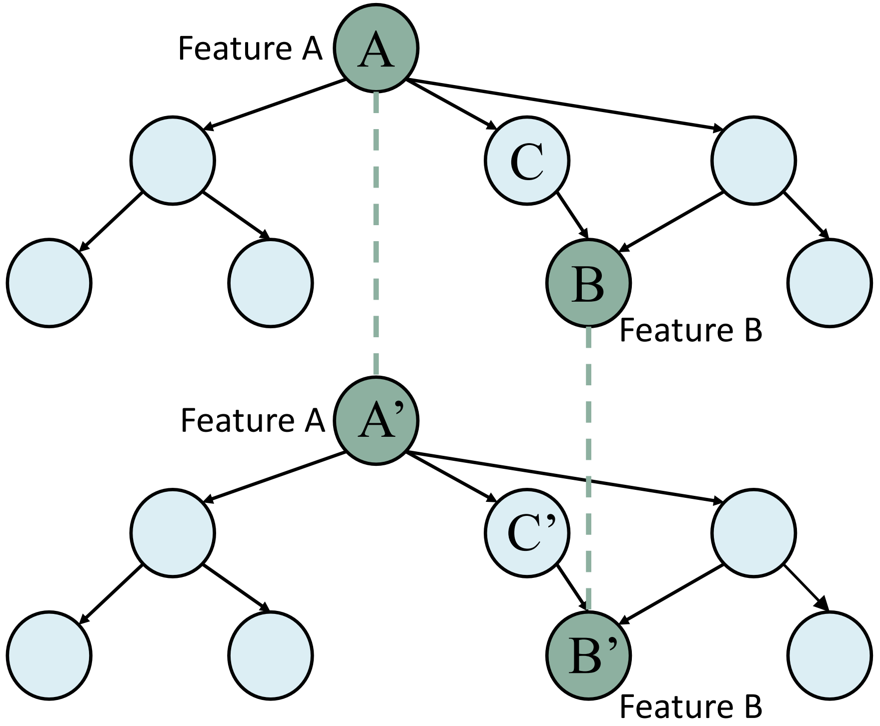
(early match with high confidence feature)

[1] L. Gao, Y . Qu, S. Y u, Y . Duan, and H. Yin, “Sigmadiff: Semantics- aware deep graph matching for pseudocode diffing

[2] Y . Duan, X. Li, J. Wang, and H. Yin, “Deepbindiff: Learning program- wide code representations for binary diffing

Binary Diffing with Neighborhood Consensus

Binary Diffing with Neighborhood Consensus: correctly matched nodes help identify surrounding matches [1, 2].



❶ **Anchor Point Identification**

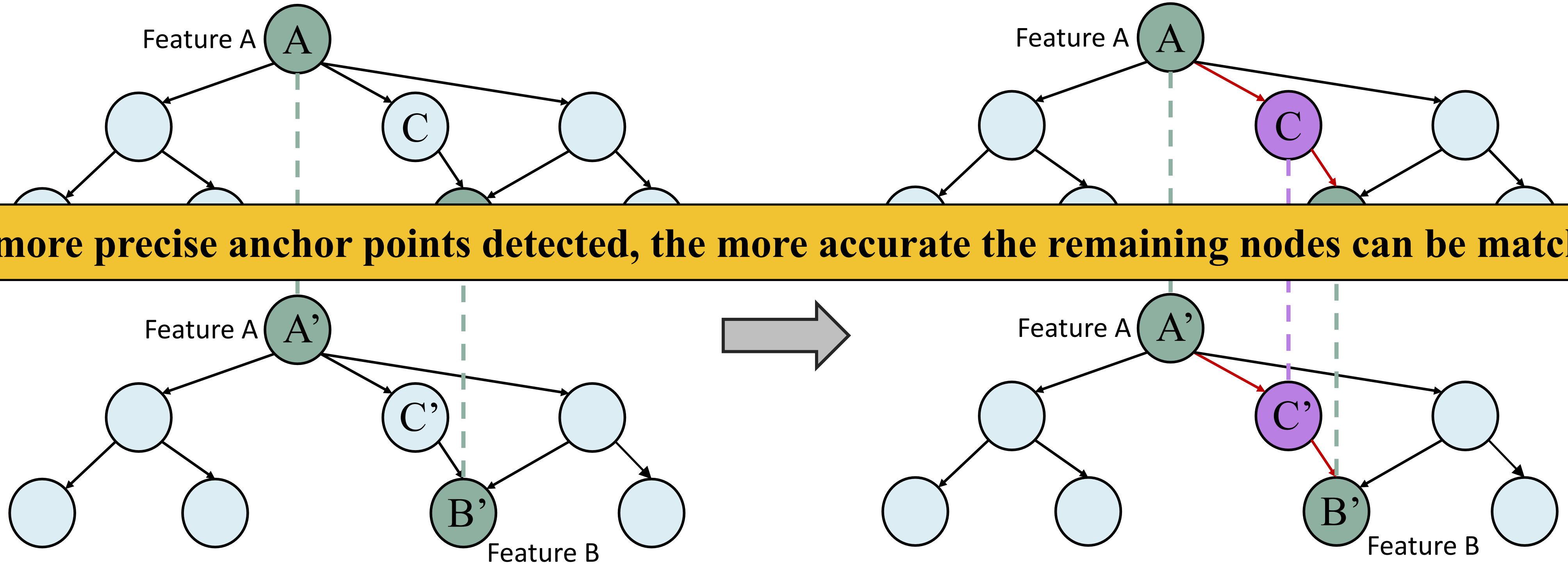
(early match with high confidence feature)

[1] L. Gao, Y . Qu, S. Y u, Y . Duan, and H. Yin, “Sigmadiff: Semantics- aware deep graph matching for pseudocode diffing

[2] Y . Duan, X. Li, J. Wang, and H. Yin, “Deepbindiff: Learning program- wide code representations for binary diffing

Binary Diffing with Neighborhood Consensus

Binary Diffing with Neighborhood Consensus: correctly matched nodes help identify surrounding matches [1, 2].



The more precise anchor points detected, the more accurate the remaining nodes can be matched

❶ **Anchor Point Identification**

(early match with high confidence feature)

❷ **Fuzzy Matching**

(utilize neighborhood matching to match remaining nodes)

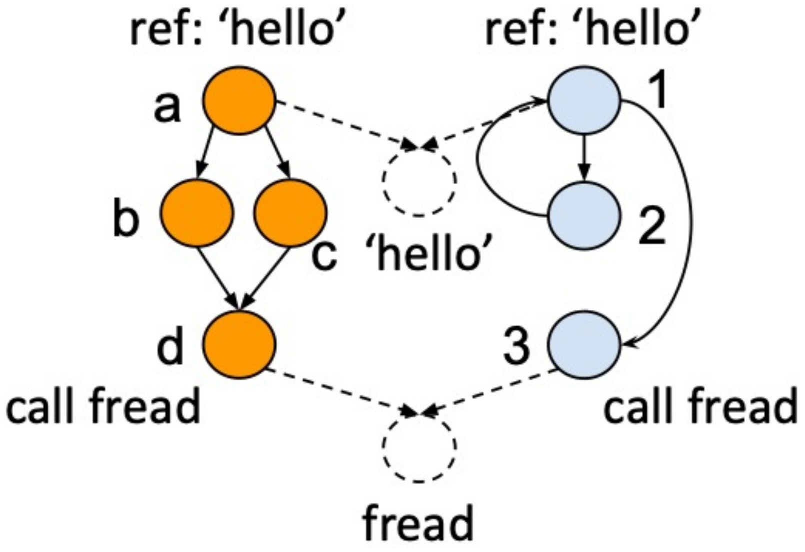
[1] L. Gao, Y . Qu, S. Y u, Y . Duan, and H. Yin, “Sigmadiff: Semantics- aware deep graph matching for pseudocode diffing

[2] Y . Duan, X. Li, J. Wang, and H. Yin, “Deepbindiff: Learning program- wide code representations for binary diffing

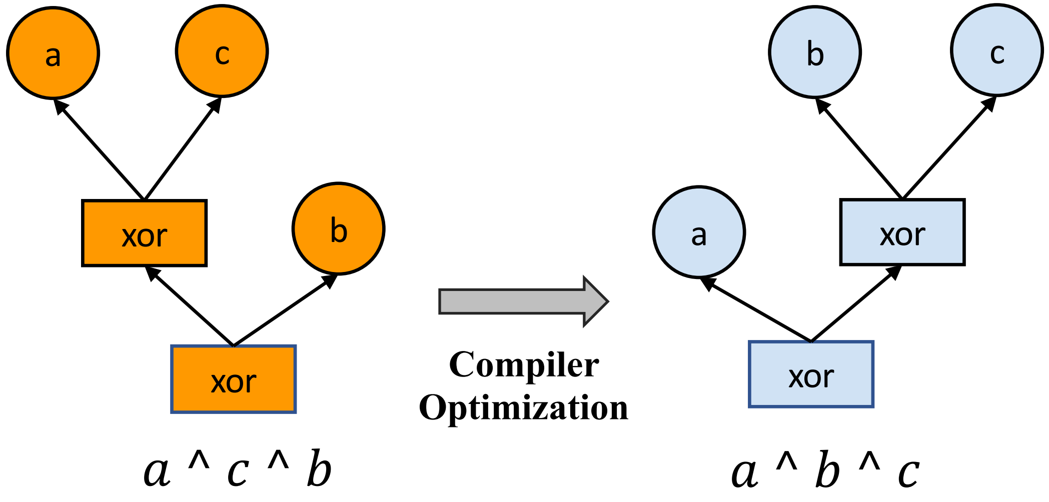
Existing Limitation

Syntactic Matching: match with uniquely appeared syntactic features

- rarely appeared
- vulnerable to compiler optimization



Constant String (DeepBinDiff $^{\text{NDSS 20}}$)

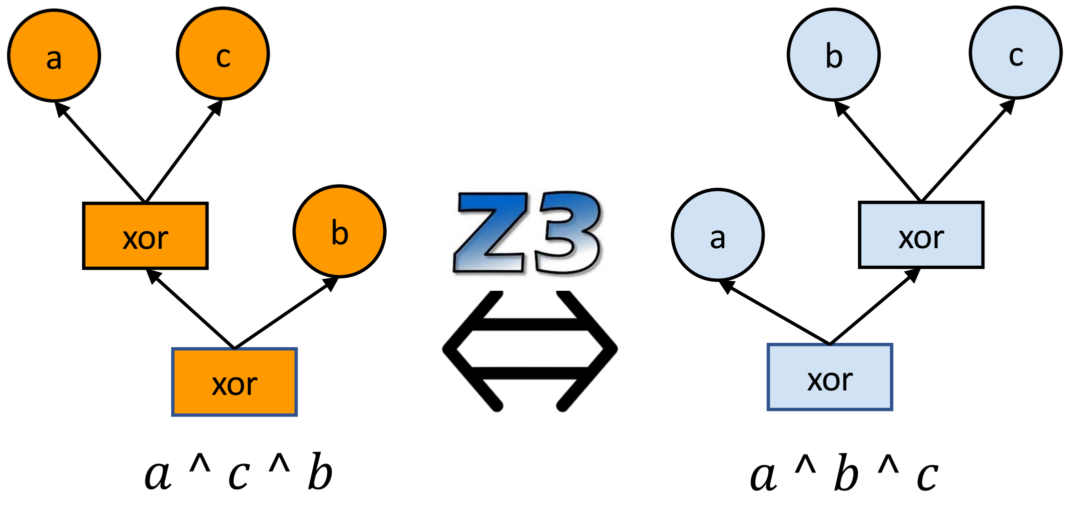


Symbolic Expression Syntax (SigmaDiff $^{\text{NDSS 24}}$)

Existing Limitation

Symbolic Solving: equivalence checking with a theorem prover

- high overhead (e.g., up to 6 hours for basic-block matching on binaries with 5k loc)
- unscalable to fine-grained full-program matching (e.g., instruction-level matching)



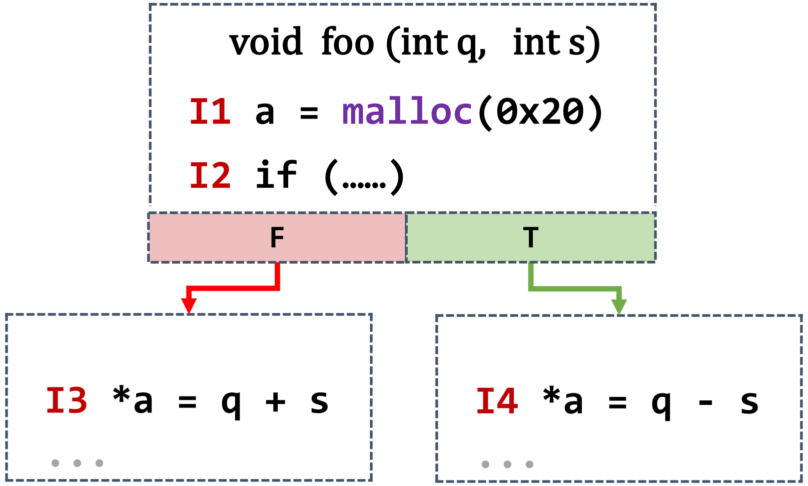
$a \wedge c \wedge b$ $\Leftrightarrow$ $a \wedge b \wedge c$

Symbolic Execution (Binhunt$^{\text{ICICS 08}}$**, BinSim**$^{\text{Security'17}}$**, BinUse** $^{\text{TSE22}}$**)**

Our Technical Direction: Dynamic Forced Execution

Dynamic Forced Execution: take concrete values obtained from sampling paths in binaries as instruction semantic features

- robust against compiler optimization
- efficient compared with a theorem prover

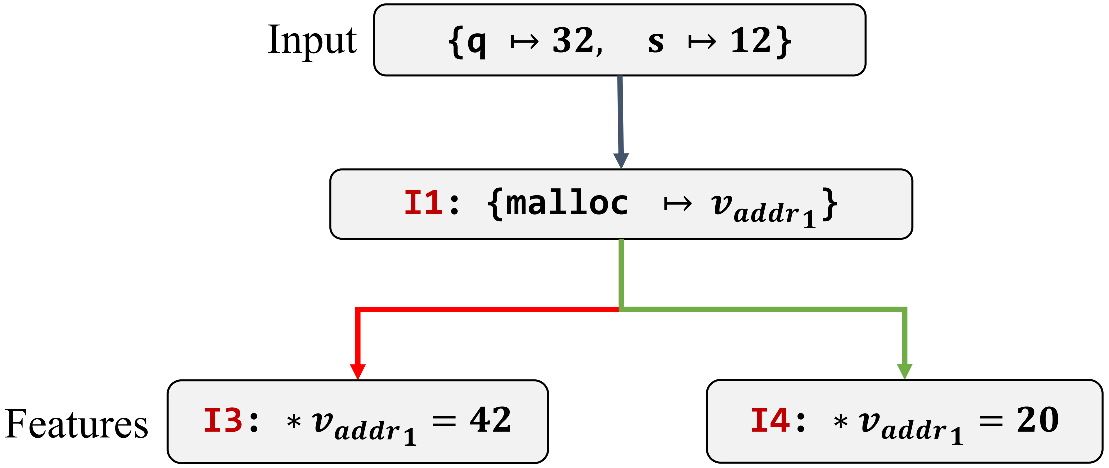


```
void foo (int q,  int s)
I1 a = malloc(0x20)
I2 if (……)
```

```
I3 *a = q + s
...
```

```
I4 *a = q - s
...
```

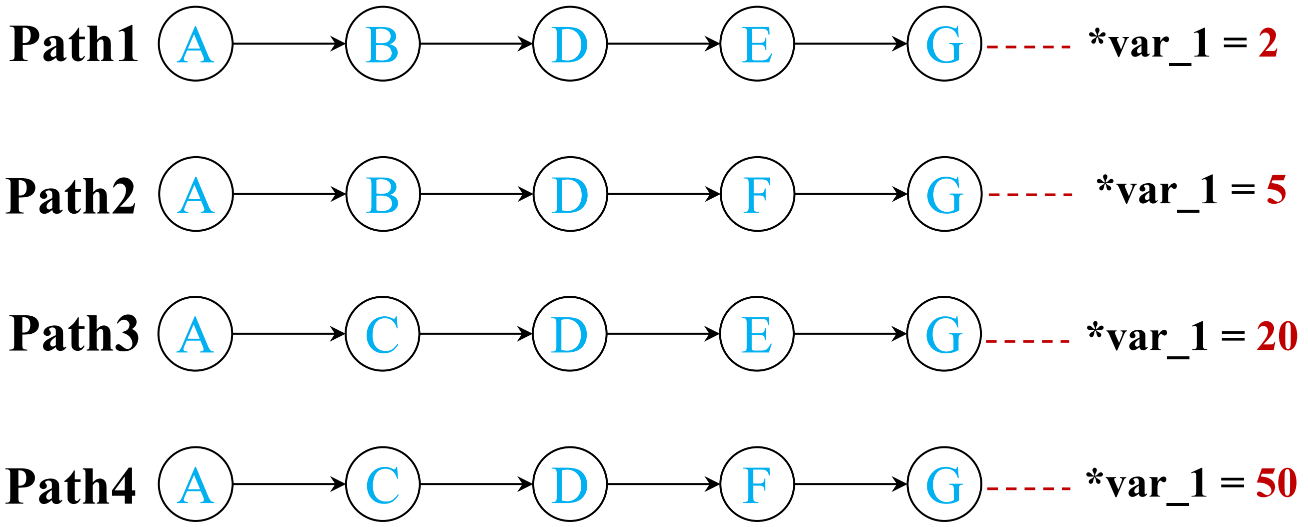
Binary Function

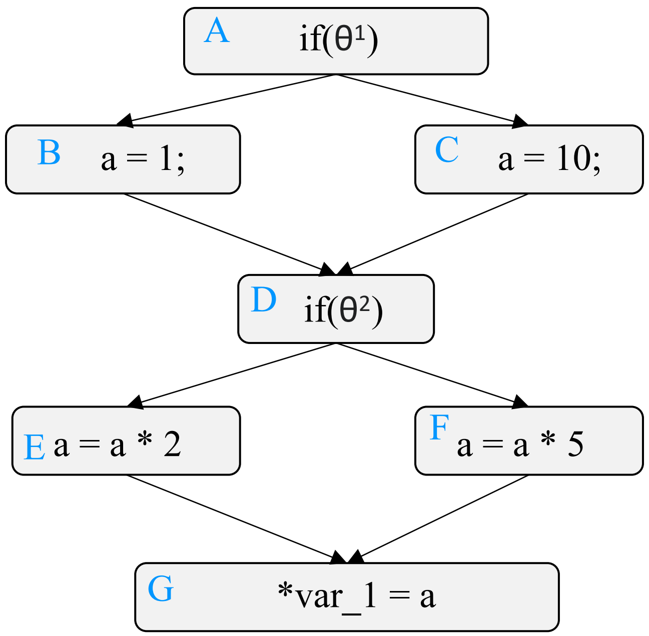


Forced Execution

Dynamic Forced Execution Challenges

Challenge 1: path selection under exponential path growth





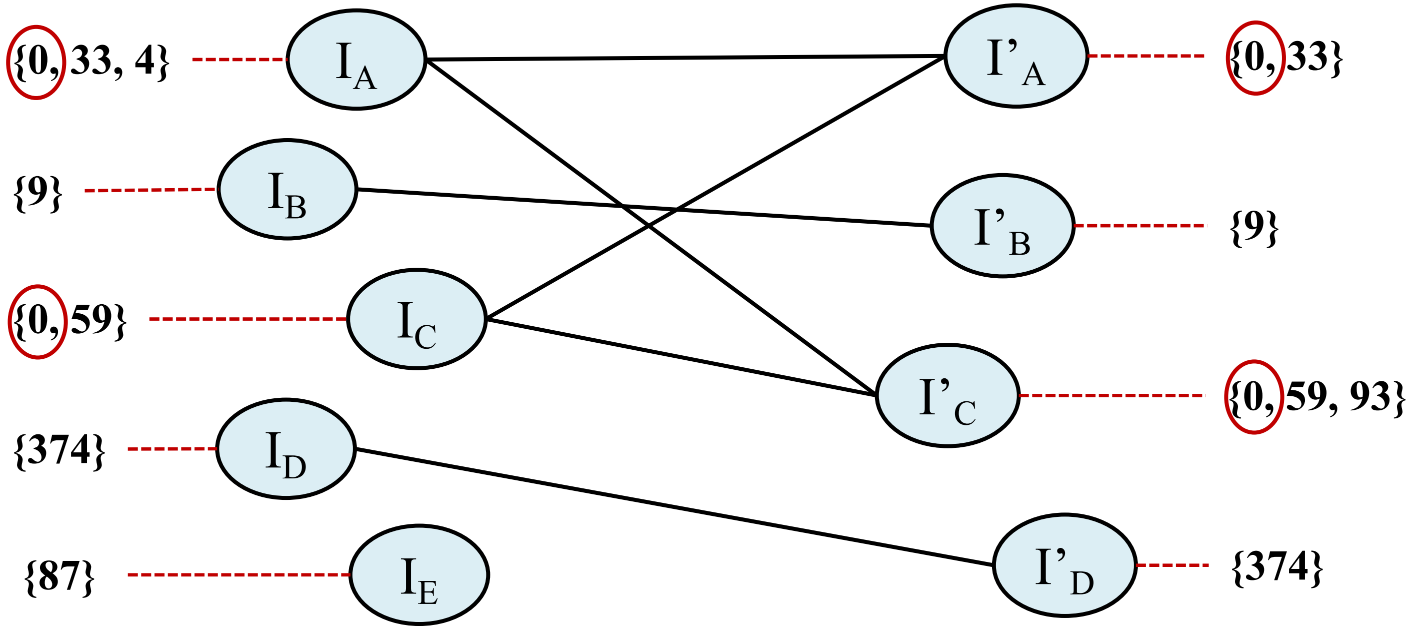
Existing Work Blex (Security 2014), PEM (FSE 2023), ARCTURUS (TOSEM 2024) :

- aim at covering basic blocks to perform coarse-grained function matching

Dynamic Forced Execution Challenges

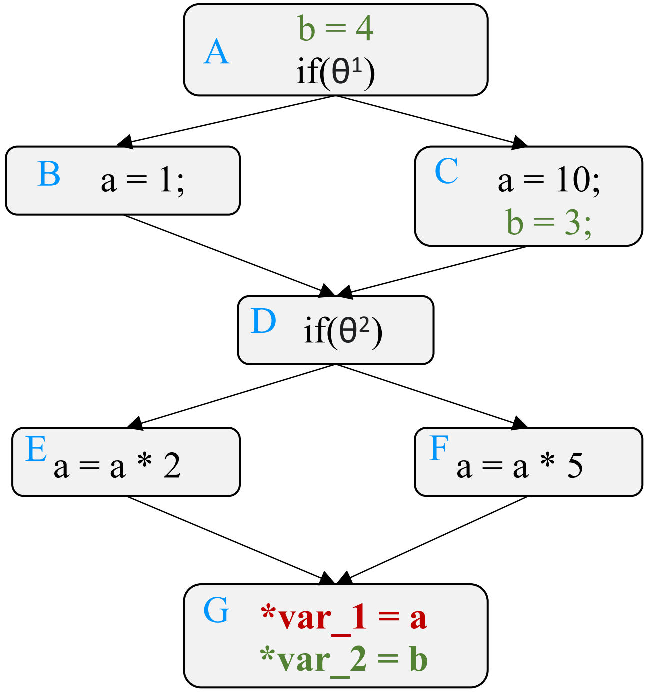
Challenge 2: ambiguous matching due to low-entropy values

- instruction features may overlap with multiple instructions in the target binary
- particularly common for low-entropy constants (e.g., 0, −1)



Observation

Observation: instructions with smaller sets of runtime values are more likely to have values overlapped with their matched counterparts in the compared binary



A b = 4 if(θ¹)

B a = 1;

C a = 10; b = 3;

D if(θ²)

E a = a * 2

F a = a * 5

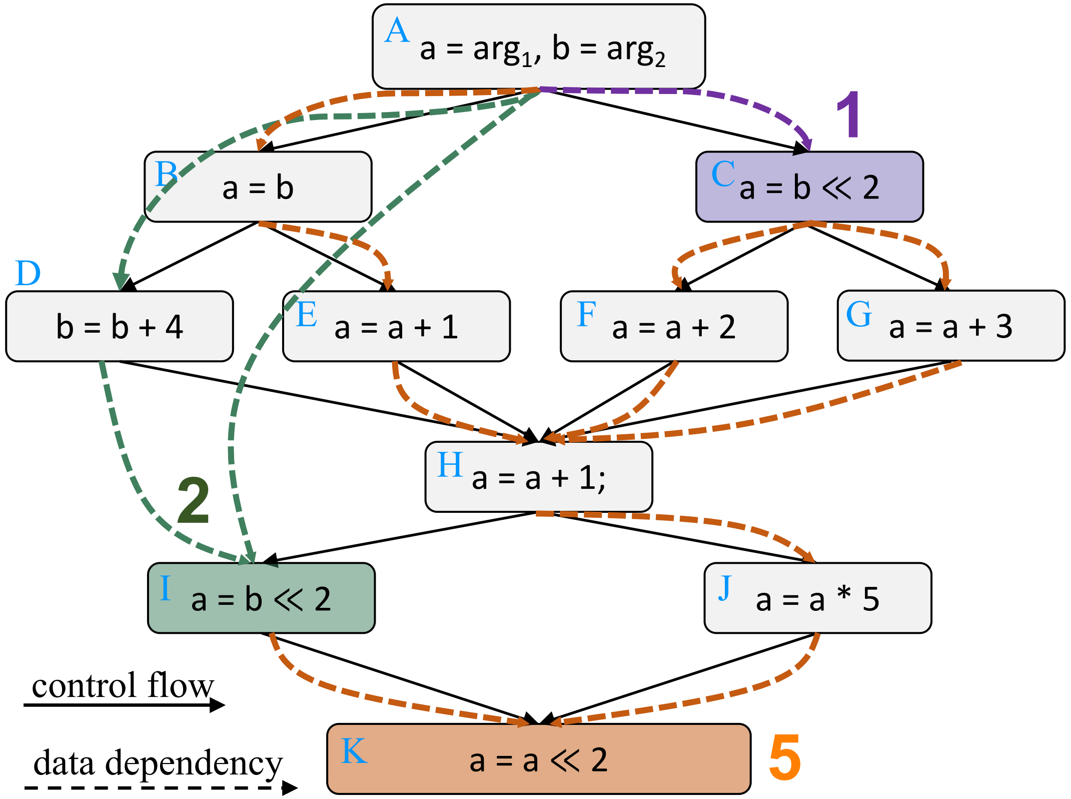
G ***var_1 = a** ***var_2 = b**

Sampled Value Space

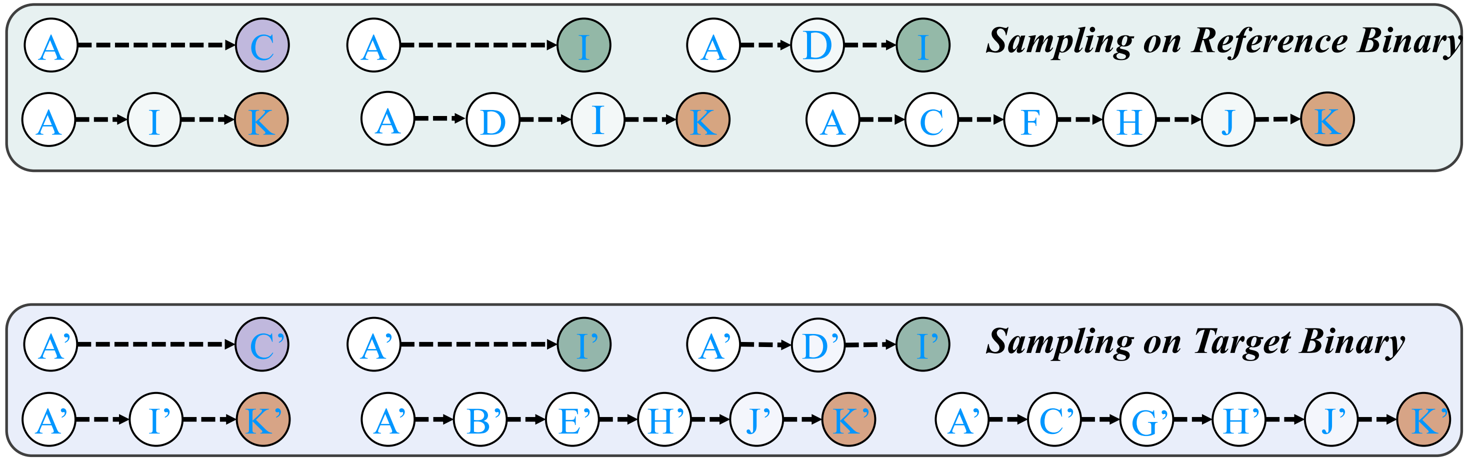
Refer Binary	**Target Binary**	**Tail Possibility of Overlap**
a_G : {2, 5, 20, 50}	a_G : {2, 5, 20, 50}	1/4
b_G : {4, 3}	b_G : {4, 3}	1/2

Prioritized Path Sampling

Prioritized Path Sampling: given a finite path sampling quota, we should prioritize sampling paths to reveal instructions with a smaller value set



CFG and PDG of an example program

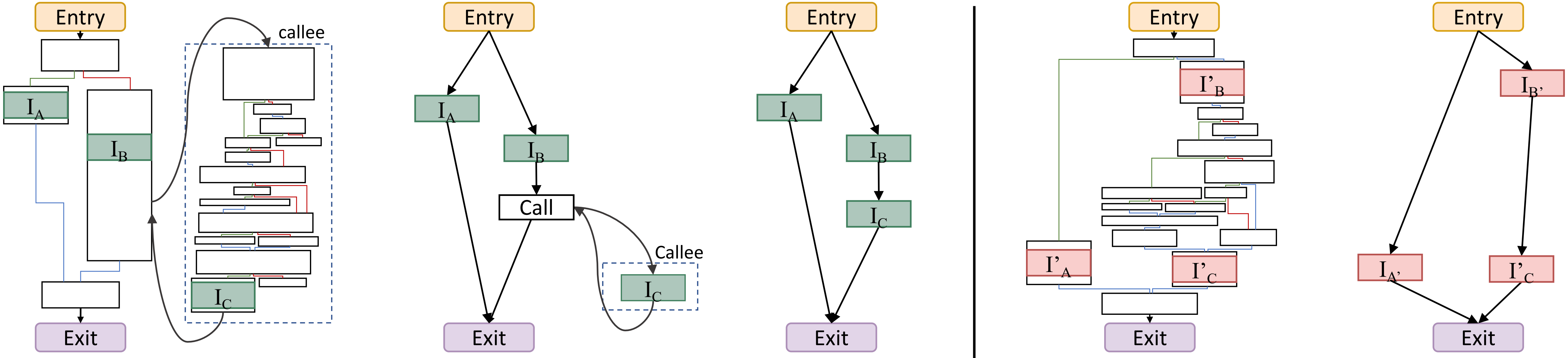


Prioritized path sampling to cover small value-set size instructions first

Reduced-CFG Isomorphism

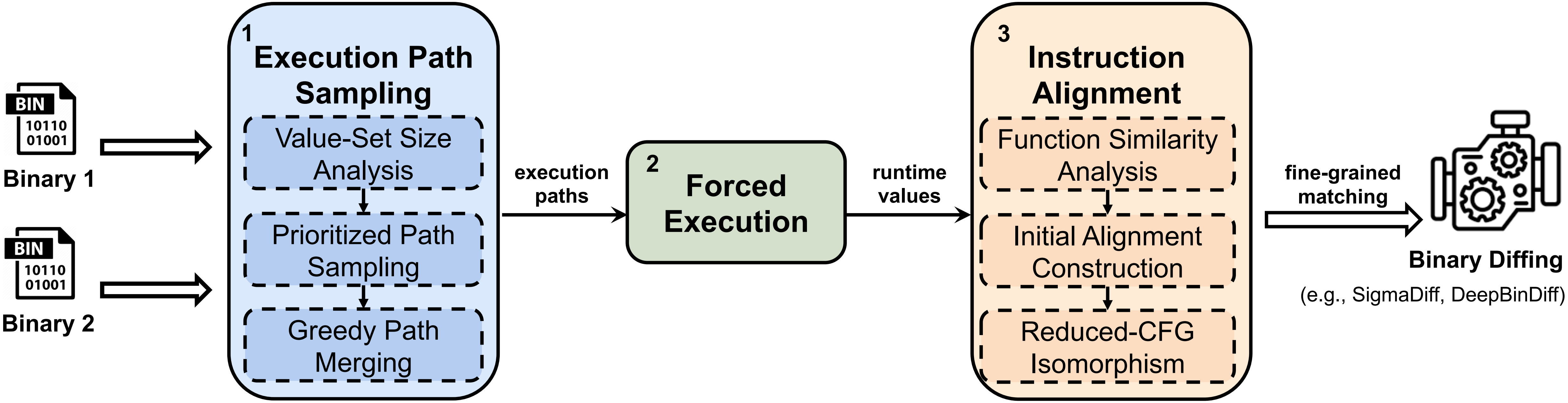
Reduced-CFG Isomorphism: construct a reduced CFG for each instruction group sharing overlapped features (forming a complete bipartite graph), then perform graph isomorphism on the reduced CFG

Advantages: efficient due to small graph, resilient to transformation on other code region



(a) CFG-Coreutils@O0 (b) Reduced CFG (c) Inline callee (d) CFG-Coreutils@O2 (e) Reduced CFG

Workflow


BIN
10110
01001
Binary 1
BIN
10110
01001
Binary 2
1
Execution Path Sampling
Value-Set Size Analysis
Prioritized Path Sampling
Greedy Path Merging
execution paths
2
Forced Execution
runtime values
3
Instruction Alignment
Function Similarity Analysis
Initial Alignment Construction
Reduced-CFG Isomorphism
fine-grained matching
Binary Diffing
(e.g., SigmaDiff, DeepBinDiff)

Implementation

BARRACUDA

- We built our prototype, **Barracuda**, based on the RetDec lifter and LLVM framework
- Extended **DeepBinDiff** and **SigmaDiff** to use **Barracuda** alignments as anchor points

Evaluation

- **RQ1 (Binary Diffing Enhancement):** How effectively does Barracuda improve the performance of semantic-aware binary diffing tools (e.g., Sigmadiff and DeepBindiff)?

- **RQ2 (Anchor Points Accuracy):** How accurately does Barracuda identify instruction alignments compared to existing anchor point detection methods?

- **RQ3 (Efficiency):** Is Barracuda efficient enough to enhance existing binary diffing tools with acceptable overhead?

- **Datasets**: Coreutils, Diffutils, Findutils, GMP, Putty
- **Diffing Scenarios**: Cross-Version, Optimization, Compiler, Architecture, and Obfuscation
- **Metrics:** pseudocode-matching accuracy based on Ghidra

RQ1: Binary Diffing Enhancement – Cross Version

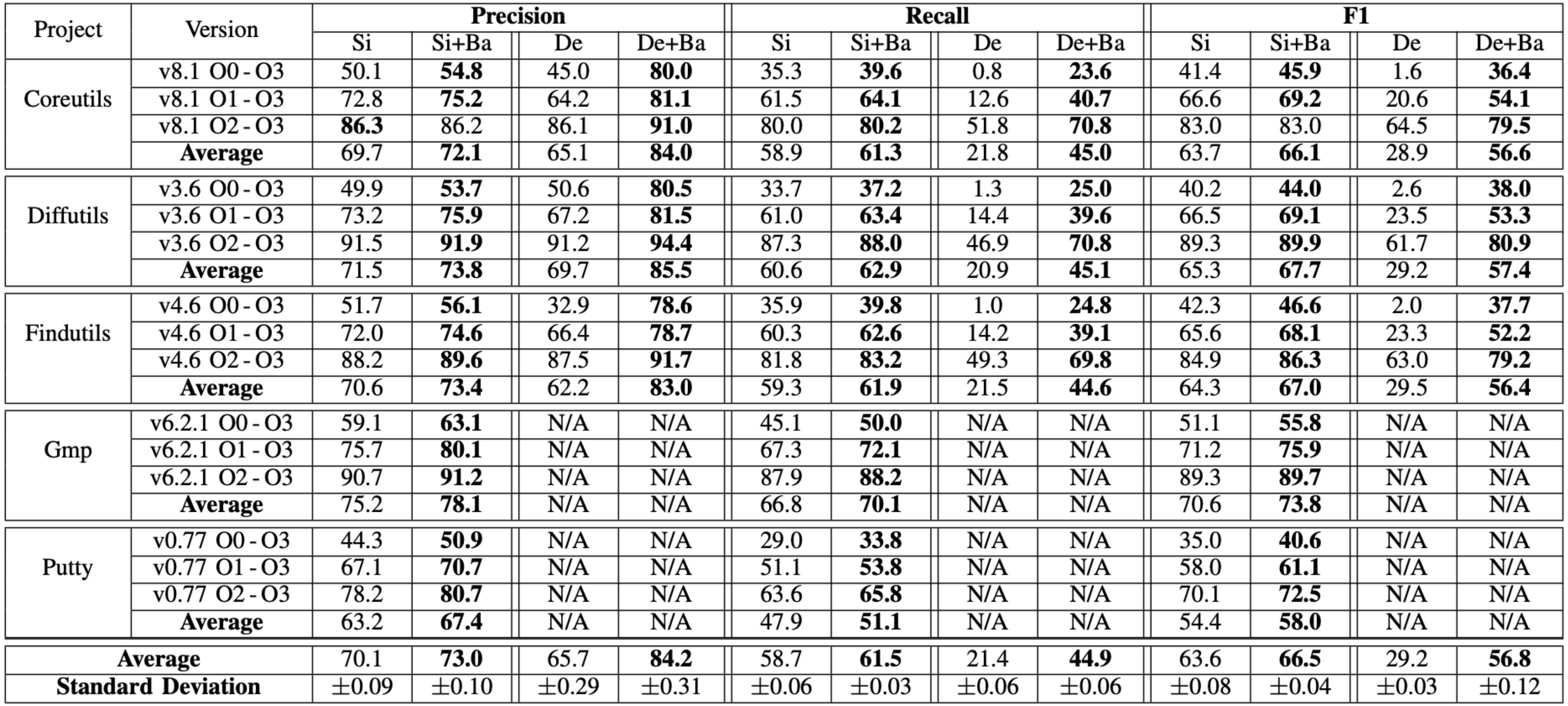
Metrics: pseudocode-matching accuracy on binaries of two versions

Project	Version	Precision				Recall				F1			
		Si	Si+Ba	De	De+Ba	Si	Si+Ba	De	De+Ba	Si	Si+Ba	De	De+Ba
Coreutils	v5.93 - v8.1	74.4	**77.4**	72.6	**77.0**	67.7	**71.5**	27.9	**49.0**	70.9	**74.3**	39.6	**59.6**
	v6.4 - v8.1	74.8	**77.3**	75.3	**79.2**	68.5	**71.6**	32.6	**53.4**	71.5	**74.3**	44.4	**63.4**
	Average	74.6	**77.4**	74.0	**78.1**	68.1	**71.5**	30.3	**51.2**	71.2	**74.3**	42.0	**61.5**
Diffutils	v2.8 - v3.6	78.6	**82.2**	**88.5**	87.7	68.9	**73.6**	35.2	**54.5**	73.4	**77.6**	49.9	**67.2**
	v3.4 - v3.6	94.3	**94.9**	94.6	**97.6**	92.3	**93.2**	64.3	**82.8**	93.3	**94.0**	76.3	**89.5**
	Average	86.5	**88.5**	91.5	**92.6**	80.6	**83.4**	49.8	**68.7**	83.4	**85.8**	63.1	**78.3**
Findutils	v4.233 - v4.6	76.2	**77.8**	73.1	**80.7**	69.6	**71.5**	29.7	**48.2**	72.7	**74.5**	42.0	**60.1**
	v4.41 - v4.6	85.0	**85.4**	86.0	**90.4**	80.0	**80.9**	39.0	**58.5**	82.4	**83.1**	53.2	**70.9**
	Average	80.6	**81.6**	79.5	**85.5**	74.8	**76.2**	34.3	**53.3**	77.6	**78.8**	47.6	**65.5**
Gmp	v6.0.0 - v6.2.1	84.6	**87.6**	N/A	N/A	80.0	**83.6**	N/A	N/A	82.2	**85.6**	N/A	N/A
	v6.1.1 - v6.2.1	87.8	**90.1**	N/A	N/A	84.2	**87.3**	N/A	N/A	86.0	**88.7**	N/A	N/A
	Average	86.2	**88.9**	N/A	N/A	82.1	**85.4**	N/A	N/A	84.1	**87.1**	N/A	N/A
Putty	v0.75 - v0.77	67.7	**68.8**	N/A	N/A	64.6	**65.5**	N/A	N/A	66.1	**67.1**	N/A	N/A
	v0.76 - v0.77	68.1	**69.3**	N/A	N/A	65.4	**65.9**	N/A	N/A	66.7	**67.5**	N/A	N/A
	Average	67.9	**69.1**	N/A	N/A	65.0	**65.7**	N/A	N/A	66.4	**67.3**	N/A	N/A
Average		79.2	**81.1**	81.7	**85.4**	74.1	**76.4**	38.1	**57.7**	76.5	**78.7**	50.9	**68.4**
Standard Deviation		±0.04	±0.12	±0.27	±0.21	±0.09	±0.12	±0.24	±0.65	±0.03	±0.12	±0.23	±0.51

Average F1 score increases from 50.9% to 68.4% (DeepBinDiff) and from 76.5% to 78.7% (SigmaDiff)

RQ1: Binary Diffing Enhancement – Cross Optimization

Metrics: pseudocode-matching accuracy on binaries compiled with different optimization levels

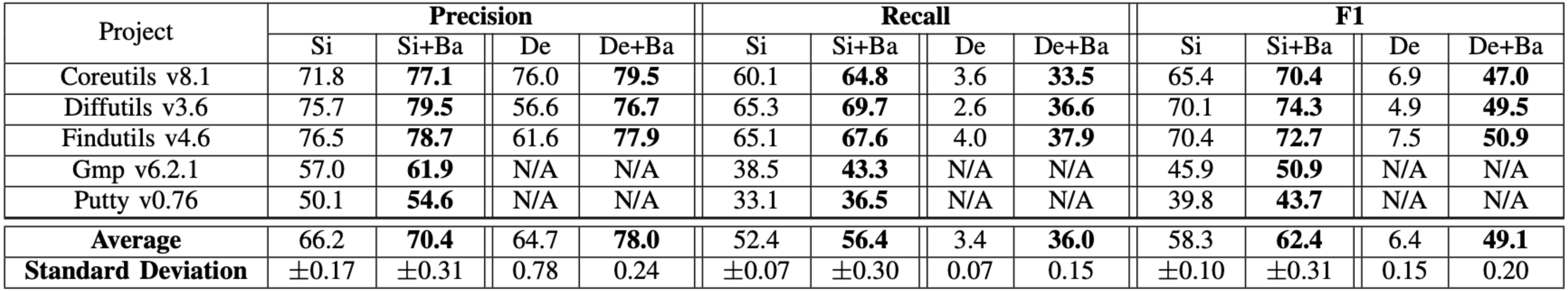


Project	Version	Precision				Recall				F1			
		Si	Si+Ba	De	De+Ba	Si	Si+Ba	De	De+Ba	Si	Si+Ba	De	De+Ba
Coreutils	v8.1 O0 - O3	50.1	**54.8**	45.0	**80.0**	35.3	**39.6**	0.8	**23.6**	41.4	**45.9**	1.6	**36.4**
	v8.1 O1 - O3	72.8	**75.2**	64.2	**81.1**	61.5	**64.1**	12.6	**40.7**	66.6	**69.2**	20.6	**54.1**
	v8.1 O2 - O3	**86.3**	86.2	86.1	**91.0**	80.0	**80.2**	51.8	**70.8**	83.0	83.0	64.5	**79.5**
	Average	69.7	**72.1**	65.1	**84.0**	58.9	**61.3**	21.8	**45.0**	63.7	**66.1**	28.9	**56.6**
Diffutils	v3.6 O0 - O3	49.9	**53.7**	50.6	**80.5**	33.7	**37.2**	1.3	**25.0**	40.2	**44.0**	2.6	**38.0**
	v3.6 O1 - O3	73.2	**75.9**	67.2	**81.5**	61.0	**63.4**	14.4	**39.6**	66.5	**69.1**	23.5	**53.3**
	v3.6 O2 - O3	91.5	**91.9**	91.2	**94.4**	87.3	**88.0**	46.9	**70.8**	89.3	**89.9**	61.7	**80.9**
	Average	71.5	**73.8**	69.7	**85.5**	60.6	**62.9**	20.9	**45.1**	65.3	**67.7**	29.2	**57.4**
Findutils	v4.6 O0 - O3	51.7	**56.1**	32.9	**78.6**	35.9	**39.8**	1.0	**24.8**	42.3	**46.6**	2.0	**37.7**
	v4.6 O1 - O3	72.0	**74.6**	66.4	**78.7**	60.3	**62.6**	14.2	**39.1**	65.6	**68.1**	23.3	**52.2**
	v4.6 O2 - O3	88.2	**89.6**	87.5	**91.7**	81.8	**83.2**	49.3	**69.8**	84.9	**86.3**	63.0	**79.2**
	Average	70.6	**73.4**	62.2	**83.0**	59.3	**61.9**	21.5	**44.6**	64.3	**67.0**	29.5	**56.4**
Gmp	v6.2.1 O0 - O3	59.1	**63.1**	N/A	N/A	45.1	**50.0**	N/A	N/A	51.1	**55.8**	N/A	N/A
	v6.2.1 O1 - O3	75.7	**80.1**	N/A	N/A	67.3	**72.1**	N/A	N/A	71.2	**75.9**	N/A	N/A
	v6.2.1 O2 - O3	90.7	**91.2**	N/A	N/A	87.9	**88.2**	N/A	N/A	89.3	**89.7**	N/A	N/A
	Average	75.2	**78.1**	N/A	N/A	66.8	**70.1**	N/A	N/A	70.6	**73.8**	N/A	N/A
Putty	v0.77 O0 - O3	44.3	**50.9**	N/A	N/A	29.0	**33.8**	N/A	N/A	35.0	**40.6**	N/A	N/A
	v0.77 O1 - O3	67.1	**70.7**	N/A	N/A	51.1	**53.8**	N/A	N/A	58.0	**61.1**	N/A	N/A
	v0.77 O2 - O3	78.2	**80.7**	N/A	N/A	63.6	**65.8**	N/A	N/A	70.1	**72.5**	N/A	N/A
	Average	63.2	**67.4**	N/A	N/A	47.9	**51.1**	N/A	N/A	54.4	**58.0**	N/A	N/A
Average		70.1	**73.0**	65.7	**84.2**	58.7	**61.5**	21.4	**44.9**	63.6	**66.5**	29.2	**56.8**
Standard Deviation		±0.09	±0.10	±0.29	±0.31	±0.06	±0.03	±0.06	±0.06	±0.08	±0.04	±0.03	±0.12

Average F1 score increases from 29.2% to 56.8% (DeepBinDiff) and from 63.6% to 66.5% (SigmaDiff)

RQ1: Binary Diffing Enhancement – Cross Compiler

Metrics: pseudocode-matching accuracy on binaries compiled with different compilers (i.e, GCC versus Clang)

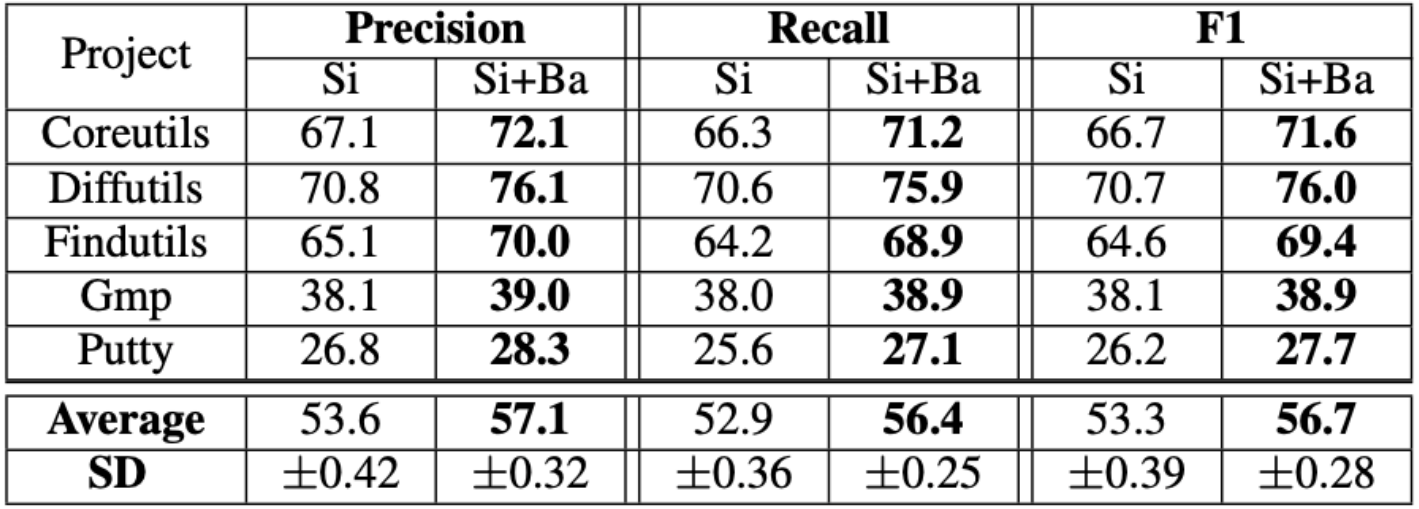


Project	Precision				Recall				F1			
	Si	Si+Ba	De	De+Ba	Si	Si+Ba	De	De+Ba	Si	Si+Ba	De	De+Ba
Coreutils v8.1	71.8	**77.1**	76.0	**79.5**	60.1	**64.8**	3.6	**33.5**	65.4	**70.4**	6.9	**47.0**
Diffutils v3.6	75.7	**79.5**	56.6	**76.7**	65.3	**69.7**	2.6	**36.6**	70.1	**74.3**	4.9	**49.5**
Findutils v4.6	76.5	**78.7**	61.6	**77.9**	65.1	**67.6**	4.0	**37.9**	70.4	**72.7**	7.5	**50.9**
Gmp v6.2.1	57.0	**61.9**	N/A	N/A	38.5	**43.3**	N/A	N/A	45.9	**50.9**	N/A	N/A
Putty v0.76	50.1	**54.6**	N/A	N/A	33.1	**36.5**	N/A	N/A	39.8	**43.7**	N/A	N/A
Average	66.2	**70.4**	64.7	**78.0**	52.4	**56.4**	3.4	**36.0**	58.3	**62.4**	6.4	**49.1**
Standard Deviation	±0.17	±0.31	0.78	0.24	±0.07	±0.30	0.07	0.15	±0.10	±0.31	0.15	0.20

Average F1 score increases from 6.4% to 49.1% (DeepBinDiff) and from 58.3% to 62.4% (SigmaDiff)

RQ1: Binary Diffing Enhancement – Cross Architecture

Metrics: pseudocode-matching accuracy on binaries compiled with different architectures (i.e, Arm versus x86-64)

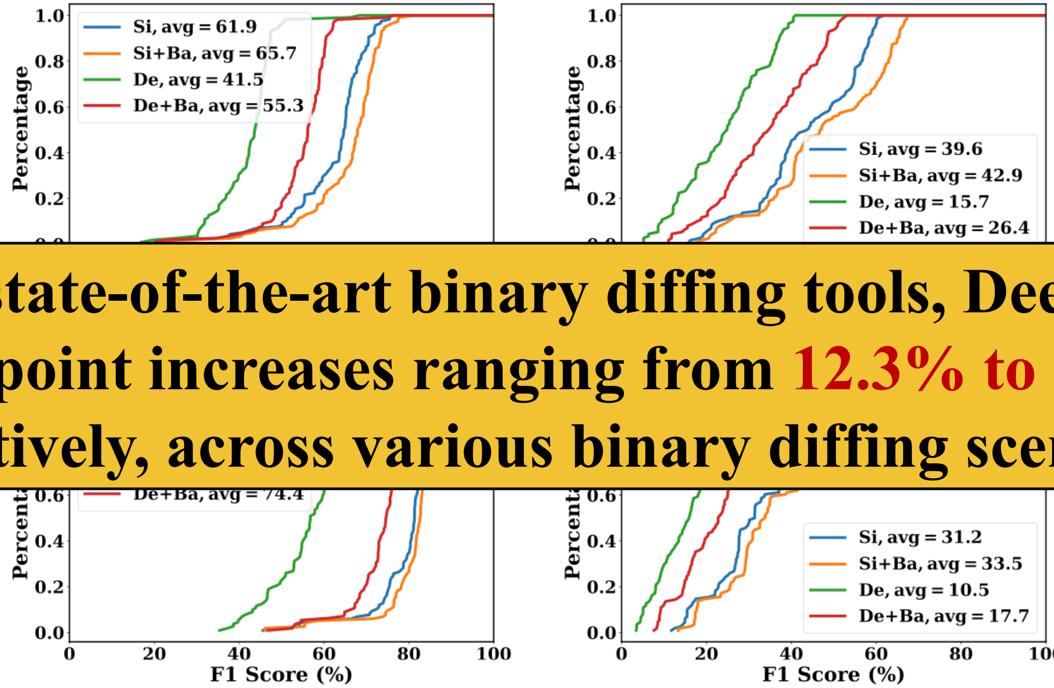


Project	Precision		Recall		F1	
	Si	Si+Ba	Si	Si+Ba	Si	Si+Ba
Coreutils	67.1	**72.1**	66.3	**71.2**	66.7	**71.6**
Diffutils	70.8	**76.1**	70.6	**75.9**	70.7	**76.0**
Findutils	65.1	**70.0**	64.2	**68.9**	64.6	**69.4**
Gmp	38.1	**39.0**	38.0	**38.9**	38.1	**38.9**
Putty	26.8	**28.3**	25.6	**27.1**	26.2	**27.7**
Average	53.6	**57.1**	52.9	**56.4**	53.3	**56.7**
SD	±0.42	±0.32	±0.36	±0.25	±0.39	±0.28

Average F1 score increases from 53.3% to 56.7% (SigmaDiff)

RQ1: Binary Diffing Enhancement – Cross Obfuscation

Metrics: pseudocode-matching accuracy on binaries compiled with different OLLVM obfuscations



(c) OLLVM-SUB

(d) OLLVM-ALL

Barracuda can enhance state-of-the-art binary diffing tools, DeepBinDiff and SigmaDiff, with F1 score percentage point increases ranging from 12.3% to 42.7% and 2.2% to 4.1%, respectively, across various binary diffing scenarios.

Average F1 score increases from 31.2% to 43.5% (DeepBinDiff) and from 53.0% to 56.0% (SigmaDiff)

RQ2: Anchor Points Accuracy

- **Metrics:** precision and recall of BARRACUDA and anchor points detection methods used by existing binary diffing tools, measured on pseudocode tokens
- **Baselines**:
  - Literal: match nodes with the same constant literal (used by DeepBindiff)
  - Expr: match nodes with the same symbolic expression (used by SigmaDiff)

Baseline Comparison

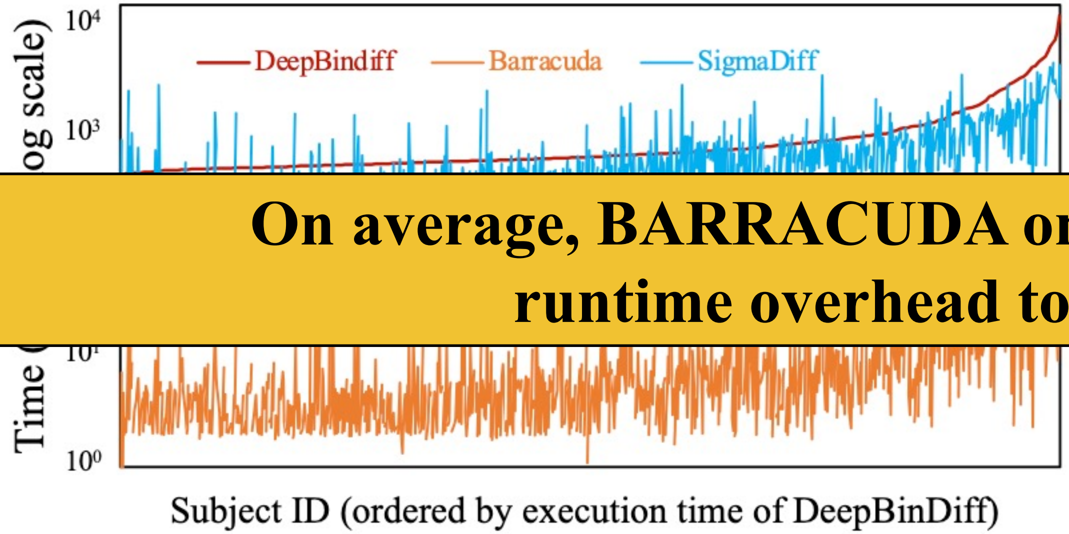
BARRACUDA can detect 24.0% more instruction alignment as anchor points than the best of existing techniques, with a high precision of 92.1%.

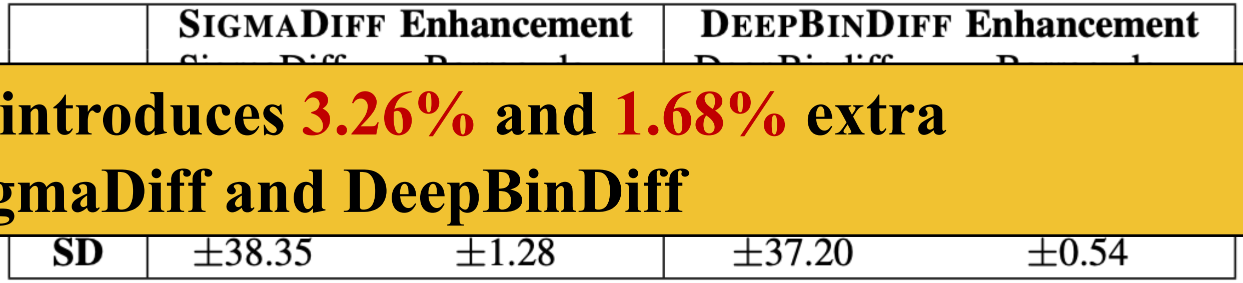
Ablation Study

Two important design leads to 23.9% and 5.0% more instruction alignments

Average	**92.42**	11.68	91.99	30.24	92.08	**37.49**	91.79	30.27	92.06	35.71	91.23	42.82
SD	±0.13	±0.07	±0.04	±0.01	±0.02	±0.02	±0.04	±0.03	±0.05	±0.09	±0.07	±0.27

RQ3: Efficiency





	SigmaDiff Enhancement		DeepBinDiff Enhancement	
SD	±38.35	±1.28	±37.20	±0.54

On average, BARRACUDA only introduces 3.26% and 1.68% extra runtime overhead to SigmaDiff and DeepBinDiff

Enhancing Semantic-Aware Binary Diffing with High-Confidence Dynamic Instruction Alignment

Thank You

More details can be found in the paper: https://dx.doi.org/10.14722/ndss.2026.240663

Artifact is publicly available: https://zenodo.org/records/17885726

Artifact Evaluated
NDSS
SYMPOSIUM
Available
Functional
Reproduced

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系