SigmaDiff: Semantics-Aware Deep Graph Matching for Pseudocode Diffing

Lian Gao, Yu Qu, Sheng Yu, Yue Duan, Heng Yin

Pseudocode Diffing

Pseudocode of Two Given Binaries

Pseudocode Diffing

Pseudocode of Two Given Binaries

Locate Similar Tokens

Capture Different Tokens

Pseudocode Diffing

Pseudocode of Two Given Binaries

Locate Similar Tokens

Capture Different Tokens

Vulnerability/patch Detection

Pseudocode Diffing

Pseudocode of Two Given Binaries

Locate Similar Tokens

Capture Different Tokens

Vulnerability/patch Detection

Plagiarism Detection

Pseudocode Diffing

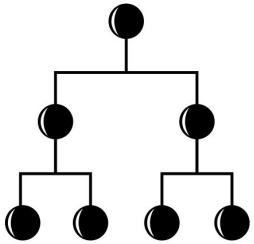
Pseudocode of Two Given Binaries

Locate Similar Tokens

Capture Different Tokens

Vulnerability/patch Detection

Plagiarism Detection



Lineage Analysis

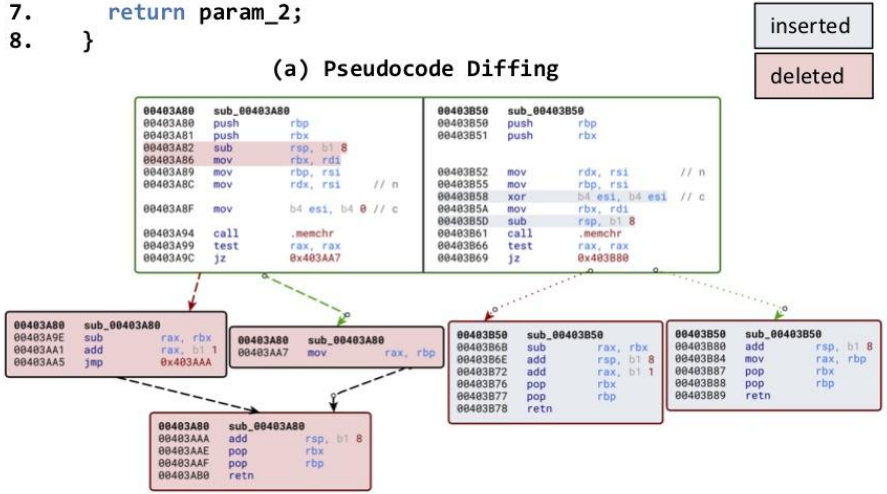
Comparison with Binary Diffing

Benefits:

- More concise and human-readable

```
1.    size_t FUN_00403a80(void *param_1,size_t param_2) {
2.      void *pvVar1;
3.      pvVar1 = memchr(param_1,0,param_2);
4.      if (pvVar1 != (void *)0x0) {
5.  -       param_2 = (long)pvVar1 + (1 - (long)param_1);
    +       return (long)pvVar1 + (1 - (long)param_1);
6.      }
7.      return param_2;
8.    }
```

(a) Pseudocode Diffing



(b) Binary Diffing

Comparison with Binary Diffing

Benefits:

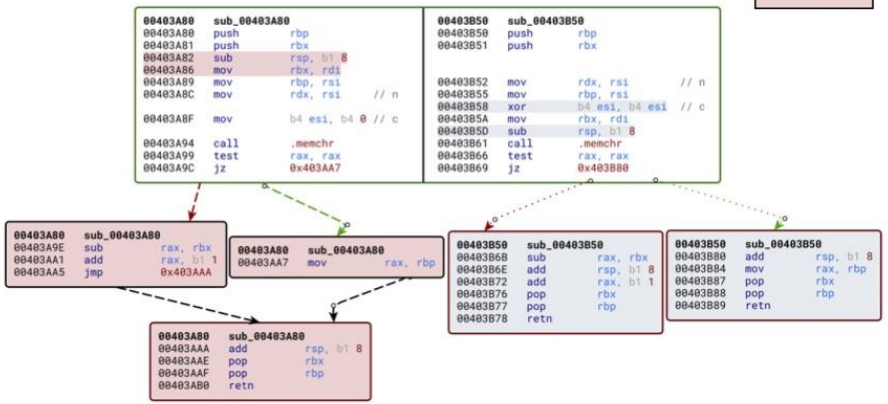
- More concise and human-readable
- More fine-grained

```
1.   size_t FUN_00403a80(void *param_1,size_t param_2) {
2.     void *pvVar1;
3.     pvVar1 = memchr(param_1,0,param_2);
4.     if (pvVar1 != (void *)0x0) {
5.   -     param_2 = (long)pvVar1 + (1 - (long)param_1);
     +     return (long)pvVar1 + (1 - (long)param_1);
6.     }
7.     return param_2;
8.   }
```

inserted

deleted

(a) Pseudocode Diffing



(b) Binary Diffing

Comparison with Binary Diffing

Benefits:

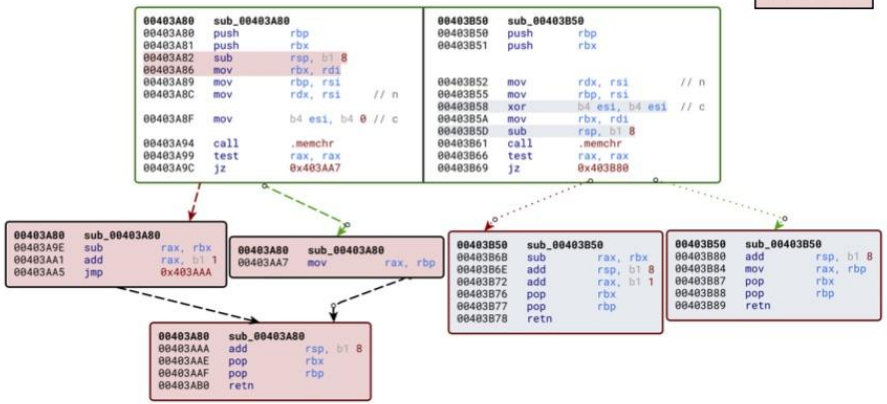
- More concise and human-readable
- More fine-grained
- The recovered semantic information could be leveraged

```
1.   size_t FUN_00403a80(void *param_1,size_t param_2) {
2.     void *pvVar1;
3.     pvVar1 = memchr(param_1,0,param_2);
4.     if (pvVar1 != (void *)0x0) {
5.   -     param_2 = (long)pvVar1 + (1 - (long)param_1);
     +     return (long)pvVar1 + (1 - (long)param_1);
6.     }
7.     return param_2;
8.   }
```

inserted

deleted

(a) Pseudocode Diffing



(b) Binary Diffing

Comparison with Binary Diffing

Benefits:

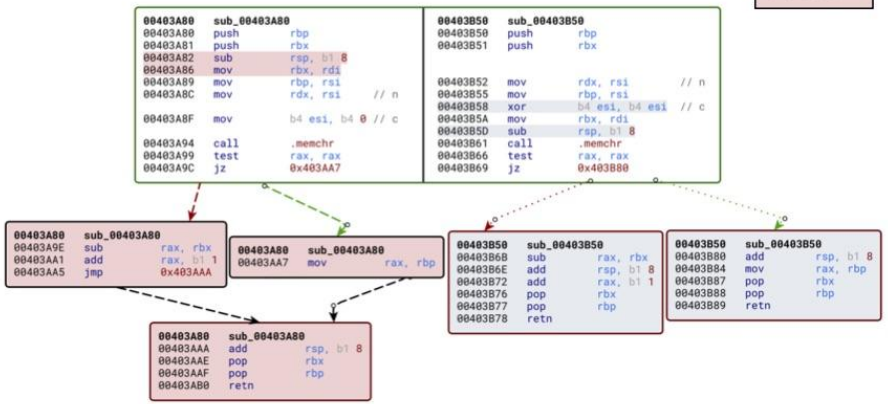
- More concise and human-readable
- More fine-grained
- The recovered semantic information could be leveraged
- Natural support of cross-architecture diffing

```
1.   size_t FUN_00403a80(void *param_1,size_t param_2) {
2.     void *pvVar1;
3.     pvVar1 = memchr(param_1,0,param_2);
4.     if (pvVar1 != (void *)0x0) {
5.   -     param_2 = (long)pvVar1 + (1 - (long)param_1);
     +     return (long)pvVar1 + (1 - (long)param_1);
6.     }
7.     return param_2;
8.   }
```

inserted

deleted

(a) Pseudocode Diffing



(b) Binary Diffing

Existing Work

Traditional approaches (e.g., BinDiff, Diaphora)

- rely on heuristics (e.g., function name) to find similar functions, perform basic block matching along the control flow graph (BinDiff), or simple string-based matching at the token level (Diaphora)
- not robust and can be easily thwarted by compiler optimizations

Existing Work

Traditional approaches (e.g., BinDiff, Diaphora)

- rely on heuristics (e.g., function name) to find similar functions, perform basic block matching along the control flow graph (BinDiff), or simple string-based matching at the token level (Diaphora)
- not robust and can be easily thwarted by compiler optimizations

Dynamic analysis-based approaches (e.g., BLEX, iBinHunt)

- capturing the semantics of binaries and have good resilience against code obfuscation
- low code coverage

Existing Work

Traditional approaches (e.g., BinDiff, Diaphora)

- rely on heuristics (e.g., function name) to find similar functions, perform basic block matching along the control flow graph (BinDiff), or simple string-based matching at the token level (Diaphora)
- not robust and can be easily thwarted by compiler optimizations

Dynamic analysis-based approaches (e.g., BLEX, iBinHunt)

- capturing the semantics of binaries and have good resilience against code obfuscation
- low code coverage

Learning-based approaches (e.g., DeepBinDiff)

- encode graph information into numerical vectors, a.k.a, graph embeddings, and perform binary diffing
- distill unique semantic-level features of a program
- does not scale well on large binaries

Motivating Example (CVE-2020-13790)

Challenges:

Pseudocode-level changes (noises) that are caused by compilers and decompilers

Motivating Example (CVE-2020-13790)

Challenges:

Pseudocode-level changes (noises) that are caused by compilers and decompilers

Source Code

```
1.    source->rescale = (JSAMPLE *)
      (*cinfo->mem->alloc_small)
      ((j_common_ptr)cinfo, JPOOL_IMAGE,
      -     (size_t)(((long)maxval + 1L) *
      +     (size_t)(((long)MAX(maxval, 255) + 1L) *
            sizeof(JSAMPLE)));
      ......
      ......
2.    for (val = 0; val <= (long)maxval; val++) {
3.      source->rescale[val] = (JSAMPLE)((val *
      MAXJSAMPLE + half_maxval) / maxval);
4.    }
```

Pseudocode (old version)

```
1.    uVar18 = (ulong)uVar12;
2.    lVar16 = (**(code
      **)param_1[1])(param_1,1,uVar18 + 1);
      ......
      ......
3.    while( true ) {
4.      *(char *)(lVar16 + lVar17) =
      (char)((long)uVar14 / (long)uVar18);
5.      lVar17 = lVar17 + 1;
6.      if ((long)uVar18 < lVar17) break;
7.      lVar16 = *(long *)(param_2 + 0x48);
8.      uVar14 = uVar14 + 0xff;
9.    }
```

Pseudocode (new version)

```
1.    uVar13 = 0xff;
2.    if (0xfe < uVar12) {
3.      uVar13 = (ulong)uVar12;}
4.    __s = (void *)(**(code
      **)param_1[1])(param_1,1,uVar13 + 1);
      ......
5.    do {
6.      *(char *)(*(long *)(param_2 + 0x48) +
      lVar14) = (char)((long)uVar13 /
      (long)(ulong)uVar12);
7.      lVar14 = lVar14 + 1;
8.      uVar13 = uVar13 + 0xff;
9.    } while (lVar14 != (ulong)uVar12 + 1);
```

inserted
deleted
moved
updated

Motivating Example (CVE-2020-13790)

Challenges:

Pseudocode-level changes (noises) that are caused by compilers and decompilers

- Different variable names

Source Code

```
1.    source->rescale = (JSAMPLE *)
      (*cinfo->mem->alloc_small)
      ((j_common_ptr)cinfo, JPOOL_IMAGE,
      -     (size_t)(((long)maxval + 1L) *
      +     (size_t)(((long)MAX(maxval, 255) + 1L) *
            sizeof(JSAMPLE)));
      ......
      ......
2.    for (val = 0; val <= (long)maxval; val++) {
3.      source->rescale[val] = (JSAMPLE)((val *
      MAXJSAMPLE + half_maxval) / maxval);
4.    }
```

Pseudocode (old version)

```
1.    uVar18 = (ulong)uVar12;
2.    lVar16 = (**(code
      **)param_1[1])(param_1,1,uVar18 + 1);
      ......
      ......
3.    while( true ) {
4.      *(char *)(lVar16 + lVar17) =
      (char)((long)uVar14 / (long)uVar18);
5.      lVar17 = lVar17 + 1;
6.      if ((long)uVar18 < lVar17) break;
7.      lVar16 = *(long *)(param_2 + 0x48);
8.      uVar14 = uVar14 + 0xff;
9.    }
```

Different Variable Names

Pseudocode (new version)

```
1.    uVar13 = 0xff;
2.    if (0xfe < uVar12) {
3.      uVar13 = (ulong)uVar12;}
4.    __s = (void *)(**(code
      **)param_1[1])(param_1,1,uVar13 + 1);
      ......
5.    do {
6.      *(char *)(*(long *)(param_2 + 0x48) +
      lVar14) = (char)((long)uVar13 /
      (long)(ulong)uVar12);
7.      lVar14 = lVar14 + 1;
8.      uVar13 = uVar13 + 0xff;
9.    } while (lVar14 != (ulong)uVar12 + 1);
```

inserted
deleted
moved
updated

Motivating Example (CVE-2020-13790)

Challenges:

Pseudocode-level changes (noises) that are caused by compilers and decompilers

- Different variable names
- Different expressions

Source Code

```
source->rescale = (JSAMPLE *)
(*cinfo->mem->alloc_small)
((j_common_ptr)cinfo, JPOOL_IMAGE,
-     (size_t)(((long)maxval + 1L) *
+     (size_t)(((long)MAX(maxval, 255) + 1L) *
      sizeof(JSAMPLE)));
......
......
for (val = 0; val <= (long)maxval; val++) {
  source->rescale[val] = (JSAMPLE)((val *
MAXJSAMPLE + half_maxval) / maxval);
}
```

Pseudocode (old version)

```
uVar18 = (ulong)uVar12;
lVar16 = (**(code
**)param_1[1])(param_1,1,uVar18 + 1);
......
......
while( true ) {    Different Expressions
  *(char *)(lVar16 + lVar17) =
(char)((long)uVar14 / (long)uVar18);
  lVar17 = lVar17 + 1;
  if ((long)uVar18 < lVar17) break;
  lVar16 = *(long *)(param_2 + 0x48);
  uVar14 = uVar14 + 0xff;
}
```

Pseudocode (new version)

```
uVar13 = 0xff;
if (0xfe < uVar12) {
  uVar13 = (ulong)uVar12;}
__s = (void *)(**(code
**)param_1[1])(param_1,1,uVar13 + 1);
......
do {
  *(char *)(*(long *)(param_2 + 0x48) +
lVar14) = (char)((long)uVar13 /
(long)(ulong)uVar12);
  lVar14 = lVar14 + 1;
  uVar13 = uVar13 + 0xff;
} while (lVar14 != (ulong)uVar12 + 1);
```

inserted
deleted
moved
updated

Motivating Example (CVE-2020-13790)

Challenges:

Pseudocode-level changes (noises) that are caused by compilers and decompilers

- Different variable names
- Different expressions
- Different control constructs

Source Code

```
1.    source->rescale = (JSAMPLE *)
      (*cinfo->mem->alloc_small)
      ((j_common_ptr)cinfo, JPOOL_IMAGE,
      -     (size_t)(((long)maxval + 1L) *
      +     (size_t)(((long)MAX(maxval, 255) + 1L) *
            sizeof(JSAMPLE)));
      ......
      ......
2.    for (val = 0; val <= (long)maxval; val++) {
3.      source->rescale[val] = (JSAMPLE)((val *
      MAXJSAMPLE + half_maxval) / maxval);
4.    }
```

Pseudocode (old version)

```
1.    uVar18 = (ulong)uVar12;
2.    lVar16 = (**(code
      **)param_1[1])(param_1,1,uVar18 + 1);
      ......
      ......
3.    while( true ) {
4.      *(char *)(lVar16 + lVar17) =
      (char)((long)uVar14 / (long)uVar18);
5.      lVar17 = lVar17 + 1;
6.      if ((long)uVar18 < lVar17) break;
7.      lVar16 = *(long *)(param_2 + 0x48);
8.      uVar14 = uVar14 + 0xff;
9.    }
```

Different Control Constructs

Pseudocode (new version)

```
1.    uVar13 = 0xff;
2.    if (0xfe < uVar12) {
3.      uVar13 = (ulong)uVar12;}
4.    __s = (void *)(**(code
      **)param_1[1])(param_1,1,uVar13 + 1);
      ......
5.    do {
6.      *(char *)(*(long *)(param_2 + 0x48) +
      lVar14) = (char)((long)uVar13 /
      (long)(ulong)uVar12);
7.      lVar14 = lVar14 + 1;
8.      uVar13 = uVar13 + 0xff;
9.    } while (lVar14 != (ulong)uVar12 + 1);
```

inserted
deleted
moved
updated

Motivating Example (CVE-2020-13790)

Observations:

- A large number of syntax level changes in decompilation are introduced during the transformation from intermediate representation (IR) to pseudocode

Design Choice:

- Perform diffing at IR level and map the IR-level diffing results up to the pseudocode level

Source Code

```
1.  source->rescale = (JSAMPLE *)
    (*cinfo->mem->alloc_small)
    ((j_common_ptr)cinfo, JPOOL_IMAGE,
-       (size_t)(((long)maxval + 1L) *
+       (size_t)(((long)MAX(maxval, 255) + 1L) *
        sizeof(JSAMPLE)));
    ......
    ......
2.  for (val = 0; val <= (long)maxval; val++) {
3.    source->rescale[val] = (JSAMPLE)((val *
    MAXJSAMPLE + half_maxval) / maxval);
4.  }
```

Pseudocode (old version)

```
1.  uVar18 = (ulong)uVar12;
2.  lVar16 = (**(code
    **)param_1[1])(param_1,1,uVar18 + 1);
    ......
    ......
3.  while( true ) {      Different Expressions
4.    *(char *)(lVar16 + lVar17) =
    (char)((long)uVar14 / (long)uVar18);
5.    lVar17 = lVar17 + 1;
6.    if ((long)uVar18 < lVar17) break;
7.    lVar16 = *(long *)(param_2 + 0x48);
8.    uVar14 = uVar14 + 0xff;
9.  }
```

Pseudocode (new version)

```
1.  uVar13 = 0xff;
2.  if (0xfe < uVar12) {
3.    uVar13 = (ulong)uVar12;}
4.  __s = (void *)(**(code
    **)param_1[1])(param_1,1,uVar13 + 1);
    ......
5.  do {
6.    *(char *)(*(long *)(param_2 + 0x48) +
    lVar14) = (char)((long)uVar13 /
    (long)(ulong)uVar12);
7.    lVar14 = lVar14 + 1;
8.    uVar13 = uVar13 + 0xff;
9.  } while (lVar14 != (ulong)uVar12 + 1);
```

inserted
deleted
moved
updated

Motivating Example (CVE-2020-13790)

Challenges:

- Different variable names

Design Choice:

- Perform a lightweight symbolic analysis to associate each IR variable with a symbolic expression that reveals how the value of that IR variable is calculated

Source Code

```
1.  source->rescale = (JSAMPLE *)
    (*cinfo->mem->alloc_small)
    ((j_common_ptr)cinfo, JPOOL_IMAGE,
-       (size_t)(((long)maxval + 1L) *
+       (size_t)(((long)MAX(maxval, 255) + 1L) *
        sizeof(JSAMPLE)));
    ......
    ......
2.  for (val = 0; val <= (long)maxval; val++) {
3.    source->rescale[val] = (JSAMPLE)((val *
    MAXJSAMPLE + half_maxval) / maxval);
4.  }
```

Pseudocode (old version)

```
1.  uVar18 = (ulong)uVar12;
2.  lVar16 = (**(code
    **)param_1[1])(param_1,1,uVar18 + 1);
    ......
    ......
3.  while( true ) {
4.    *(char *)(lVar16 + lVar17) =
    (char)((long)uVar14 / (long)uVar18);
5.    lVar17 = lVar17 + 1;
6.    if ((long)uVar18 < lVar17) break;
7.    lVar16 = *(long *)(param_2 + 0x48);
8.    uVar14 = uVar14 + 0xff;
9.  }
```

Different Variable Names

Pseudocode (new version)

```
1.  uVar13 = 0xff;
2.  if (0xfe < uVar12) {
3.    uVar13 = (ulong)uVar12;}
4.  __s = (void *)(**(code
    **)param_1[1])(param_1,1,uVar13 + 1);
    ......
5.  do {
6.    *(char *)(*(long *)(param_2 + 0x48) +
    lVar14) = (char)((long)uVar13 /
    (long)(ulong)uVar12);
7.    lVar14 = lVar14 + 1;
8.    uVar13 = uVar13 + 0xff;
9.  } while (lVar14 != (ulong)uVar12 + 1);
```

inserted
deleted
moved
updated

Motivating Example (CVE-2020-13790)

Challenges:

- Unstable control flow

Design Choice:

- Leverage data and control dependencies to capture contextual information for each IR

Source Code

```
1.    source->rescale = (JSAMPLE *)
      (*cinfo->mem->alloc_small)
      ((j_common_ptr)cinfo, JPOOL_IMAGE,
      -     (size_t)(((long)maxval + 1L) *
      +     (size_t)(((long)MAX(maxval, 255) + 1L) *
            sizeof(JSAMPLE)));
      ......
      ......
2.    for (val = 0; val <= (long)maxval; val++) {
3.      source->rescale[val] = (JSAMPLE)((val *
      MAXJSAMPLE + half_maxval) / maxval);
4.    }
```

Pseudocode (old version)

```
1.    uVar18 = (ulong)uVar12;
2.    lVar16 = (**(code
      **)param_1[1])(param_1,1,uVar18 + 1);
      ......
      ......
3.    while( true ) {
4.      *(char *)(lVar16 + lVar17) =
      (char)((long)uVar14 / (long)uVar18);
5.      lVar17 = lVar17 + 1;
6.      if ((long)uVar18 < lVar17) break;
7.      lVar16 = *(long *)(param_2 + 0x48);
8.      uVar14 = uVar14 + 0xff;
9.    }
```

Different Control Constructs

Pseudocode (new version)

```
1.    uVar13 = 0xff;
2.    if (0xfe < uVar12) {
3.      uVar13 = (ulong)uVar12;}
4.    __s = (void *)(**(code
      **)param_1[1])(param_1,1,uVar13 + 1);
      ......
5.    do {
6.      *(char *)(*(long *)(param_2 + 0x48) +
      lVar14) = (char)((long)uVar13 /
      (long)(ulong)uVar12);
7.      lVar14 = lVar14 + 1;
8.      uVar13 = uVar13 + 0xff;
9.    } while (lVar14 != (ulong)uVar12 + 1);
```

inserted
deleted
moved
updated

Motivating Example (CVE-2020-13790)

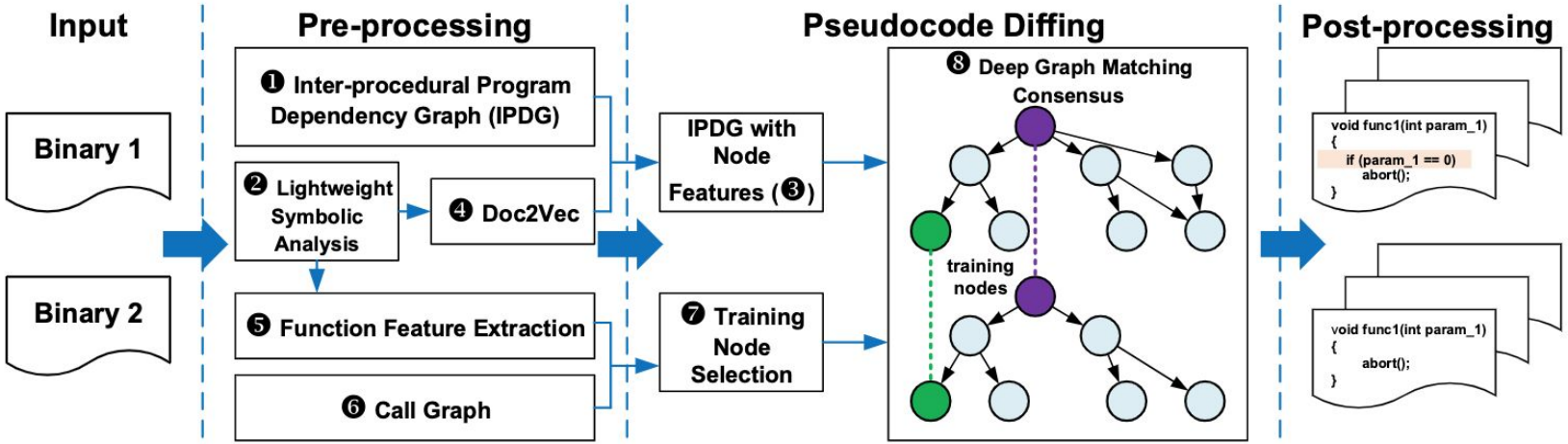
Challenges

- Noises in the graph matching
- Large number of tokens
- NP-hard graph matching problem

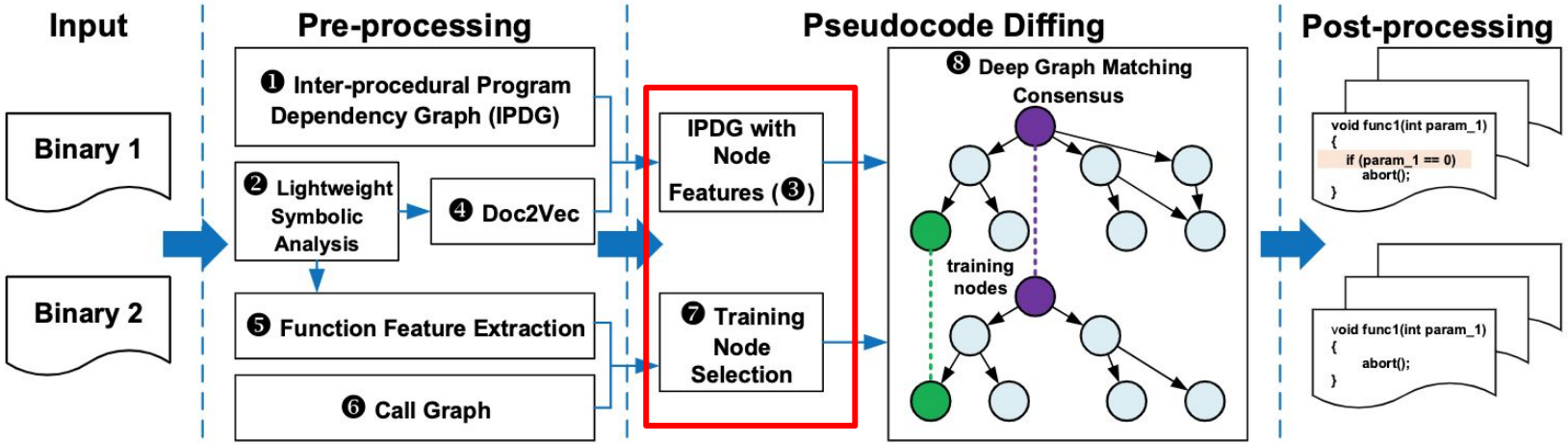
Design Choice

- Apply deep graph matching consensus (DGMC) model to fully exploit the neighboring contextual information
- Leverages the computing power of modern GPUs

Approach Overview



Approach Overview



IPDG with Node Features

```
1.  void foo(undefined8* param_1) {
2.    lVar2 = bar(param_1);
3.    if (lVar2 != 0) {
4.      *(param_1 + 8) = lVar2;
5.    }
6.  }
```

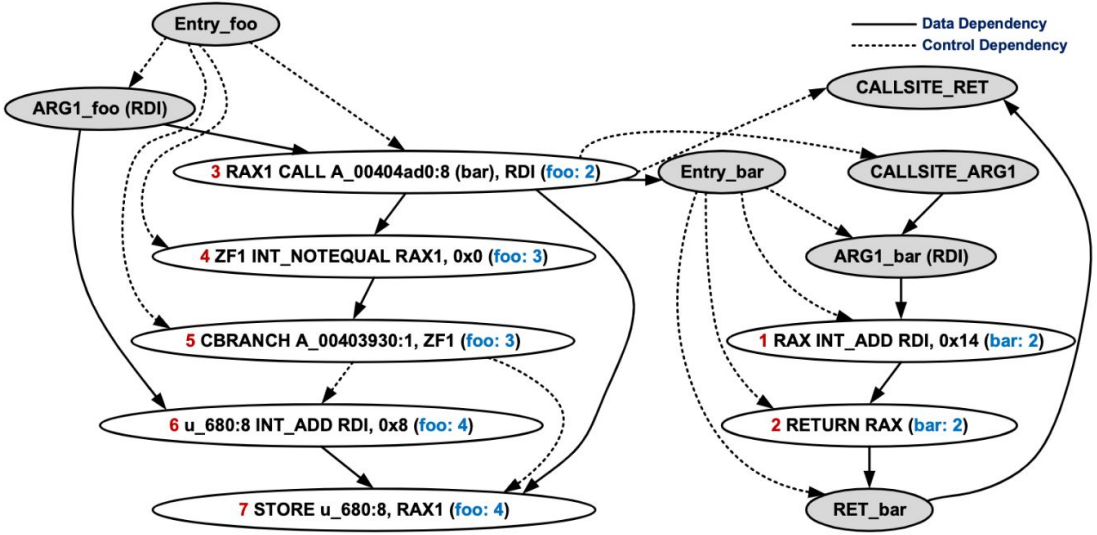
```
1.  undefined8* bar(undefined8* param_1){
2.    return param_1 + 20;
3.  }
```

IPDG with Node Features

```
1.  void foo(undefined8* param_1) {
2.    lVar2 = bar(param_1);
3.    if (lVar2 != 0) {
4.      *(param_1 + 8) = lVar2;
5.    }
6.  }
```

```
1.  undefined8* bar(undefined8* param_1){
2.    return param_1 + 20;
3.  }
```

(1) Generate inter-procedural program dependency graph



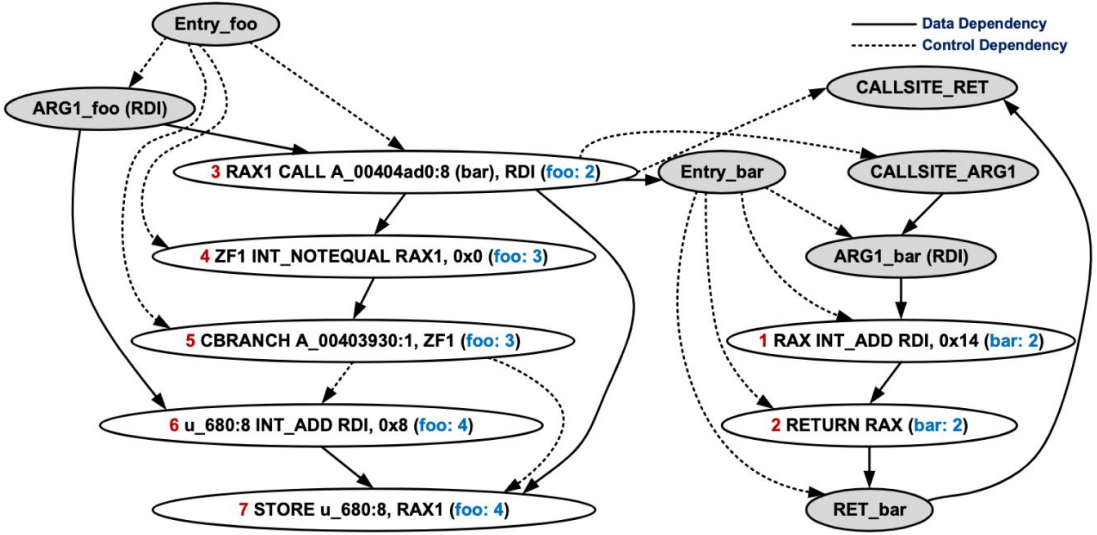
IPDG with Node Features

```
1.  void foo(undefined8* param_1) {
2.    lVar2 = bar(param_1);
3.    if (lVar2 != 0) {
4.      *(param_1 + 8) = lVar2;
5.    }
6.  }
1.  undefined8* bar(undefined8* param_1){
2.    return param_1 + 20;
3.  }
```

ID	Symbolic Analysis Results	Node Features
1	α(RDI) := ARG1 α(RAX) := ARG1 + 20	ARG1 + 20 INT_ADD ARG1 20
2	N/A	RETURN ARG1 + 20
3	α(RDI) := ARG1 α(RAX1) := ARG1 + 20	ARG1 + 20 CALL ADDR ARG1
4	α(ZF1) := 0/1	0 INT_NOTEQUAL ARG1 + 20 CONST
5	N/A	CBRANCH ARG1 + 20 CONST
6	α(u_680:8) := ARG1 + 8	ARG1 + 8 INT_ADD ARG1 CONST
7	α([ARG1 + 8]) := ARG1 + 20	STORE CONST ARG1 + 8 ARG1 + 20

(2) Perform lightweight symbolic analysis
=> Node Features

(1) Generate inter-procedural program dependency graph



Training Node Selection

Training Nodes:

- Node pairs with high similarity and uniqueness

Used in semi-supervised learning:

- Source and target graphs + small
  set of training nodes => mappings for the rest of the nodes

Training Node Selection

Select IRs (nodes):

- that have unique node feature
- that appear in both graphs

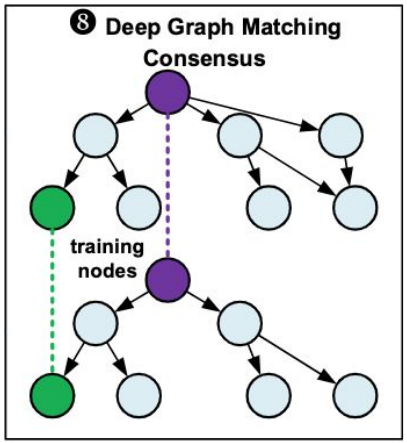
```
1.  void foo(undefined8* param_1) {
2.    lVar2 = bar(param_1);
3.    if (lVar2 != 0) {
4.      *(param_1 + 8) = lVar2;
5.    }
6.  }
```

```
1.  void foo(undefined8* param_1) {
2.    lVar2 = bar(param_1);
3.    if (lVar2 == 0) {
4.      return;
5.    }
6.    *(param_1 + 8) = lVar2;
7.  }
```

STORE CONST ARG1+8 ARG1+20

ID	Symbolic Analysis Results	Node Features
1	α(RDI) := ARG1 α(RAX) := ARG1 + 20	ARG1 + 20 INT_ADD ARG1 20
2	N/A	RETURN ARG1 + 20
3	α(RDI) := ARG1 α(RAX1) := ARG1 + 20	ARG1 + 20 CALL ADDR ARG1
4	α(ZF1) := 0/1	0 INT_NOTEQUAL ARG1 +20 CONST
5	N/A	CBRANCH ARG1 + 20 CONST
6	α(u_680:8) := ARG1 + 8	ARG1 + 8 INT_ADD ARG1 CONST
7	α([ARG1 + 8]) := ARG1 + 20	STORE CONST ARG1 + 8 ARG1 +20

Diffing

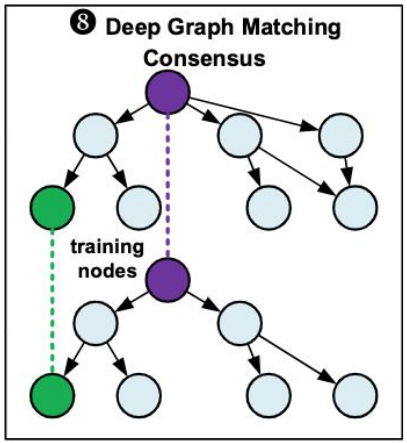


Deep Graph Matching Consensus (DGMC) model [1]

- A two-stage deep graph matching architecture
  - Leveraging Graph Neural Network (GNN) to obtains the initial matchings
  - Iteratively reaching neighborhood consensus (ensuring the neighbors of the matched nodes are correctly matched to each other as well)

[1] Fey, Matthias, et al. "Deep Graph Matching Consensus." *International Conference on Learning Representations*. 2019.

Diffing



Deep Graph Matching Consensus (DGMC) model [1]

- A two-stage deep graph matching architecture
  - Leveraging Graph Neural Network (GNN) to obtains the initial matchings
  - Iteratively reaching neighborhood consensus (ensuring the neighbors of the matched nodes are correctly matched to each other as well)

Modifications:

- Increase the penalty of incompatible types
- Add more hops
- Introduce the “pre-training and fine-tuning” schema
- Design an iterative algorithm to avoid the out-of-memory problem in GPU

[1] Fey, Matthias, et al. "Deep Graph Matching Consensus." *International Conference on Learning Representations*. 2019.

Evaluation

- Compare with Diaphora and DeepBinDiff
  - Cross-version
  - Cross-optimization-level
  - Cross-compiler
  - Cross-architecture
- Conduct Patch Detection
  - Case studies on real-world vulnerabilities
  - Zoom

Evaluation

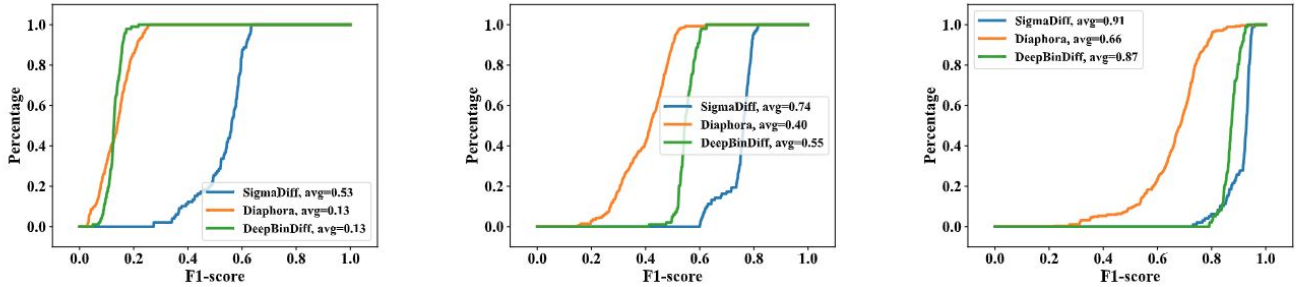
TABLE V: Cross-version Pseudocode Diffing Results. Si: SIGMADIFF, De: DeepBinDiff, Di: Diaphora

		Recall			Precision			F1		
		Si	De	Di	Si	De	Di	Si	De	Di
Coreutils	v5.93 - v8.1	**0.624**	0.589	0.438	**0.770**	0.743	0.759	**0.687**	0.654	0.549
	v6.4 - v8.1	**0.689**	0.631	0.551	**0.793**	0.767	0.643	**0.735**	0.691	0.592
	Average	**0.656**	0.610	0.494	**0.781**	0.755	0.701	**0.711**	0.673	0.571
Diffutils	v2.8 - v3.6	**0.694**	0.660	0.348	**0.848**	0.806	0.406	**0.763**	0.725	0.375
	v3.4 - v3.6	**0.928**	0.909	0.788	0.957	**0.960**	0.844	**0.942**	0.934	0.815
	Average	**0.811**	0.784	0.568	**0.903**	0.883	0.625	**0.853**	0.829	0.595
Findutils	v4.233 - v4.6	**0.685**	0.579	0.366	**0.825**	0.758	0.773	**0.748**	0.655	0.487
	v4.41 - v4.6	**0.769**	0.690	0.499	**0.868**	0.847	0.786	**0.814**	0.759	0.609
	Average	**0.727**	0.635	0.433	**0.847**	0.803	0.779	**0.781**	0.707	0.548
Gmp	v6.0.0 - v6.2.1	**0.815**	N/A	0.691	0.871	N/A	**0.892**	**0.842**	N/A	0.779
	v6.1.1 - v6.2.1	**0.858**	N/A	0.771	0.894	N/A	**0.920**	**0.876**	N/A	0.839
	Average	**0.836**	N/A	0.731	0.882	N/A	**0.906**	**0.859**	N/A	0.809
Putty	v0.75 - v0.77	**0.781**	N/A	0.741	0.899	N/A	**0.897**	**0.836**	N/A	0.812
	v0.76 - v0.77	**0.798**	N/A	0.732	**0.908**	N/A	0.881	**0.849**	N/A	0.800
	Average	**0.789**	N/A	0.737	**0.904**	N/A	0.889	**0.842**	N/A	0.806

TABLE VII: Cross-architecture Pseudocode Diffing Results (ARM vs. x86-64). Si: SIGMADIFF, Di: Diaphora

	Recall		Precision		F1	
	Si	Di	Si	Di	Si	Di
Coreutils 8.1	**0.720**	0.429	0.725	**0.954**	**0.723**	0.586
Diffutils 3.6	**0.751**	0.003	**0.754**	0.674	**0.752**	0.006
Findutils 4.6	**0.620**	0.003	**0.642**	0.521	**0.631**	0.006
Gmp 6.2.1	**0.362**	0.165	0.386	**0.827**	**0.373**	0.275
Putty 0.76	**0.143**	0.038	0.436	**0.591**	**0.216**	0.072

Evaluation



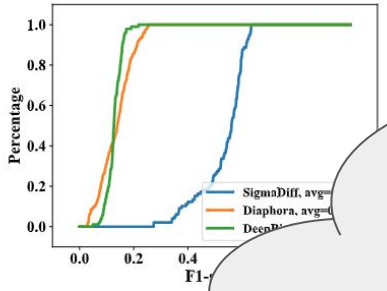
(a) O0 vs. O3 (b) O1 vs. O3 (c) O2 vs. O3

Fig. 6: Cross-optimization-level Pseudocode Diffing F1-score CDF (merging results of Coreutils, Diffutils, and Findutils)

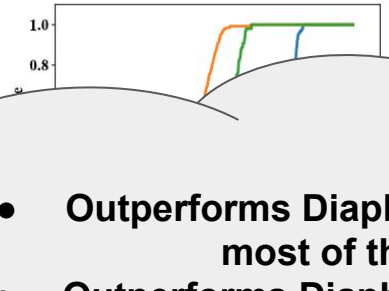
TABLE VI: Cross-compiler Pseudocode Diffing Results (Clang vs. GCC). Si: SIGMADIFF, De: DeepBinDiff, Di: Diaphora

	Recall			**Precision**			**F1**		
	Si	De	Di	Si	De	Di	Si	De	Di
Coreutils 8.1	**0.595**	0.203	0.262	**0.807**	0.482	0.720	**0.681**	0.285	0.382
Diffutils 3.6	0.295	0.187	0.029	**0.574**	0.445	0.535	**0.390**	0.263	0.055
Findutils 4.6	0.363	0.202	0.051	**0.619**	0.458	0.597	**0.457**	0.279	0.094
Gmp 6.2.1	**0.393**	N/A	0.227	0.597	N/A	**0.821**	**0.474**	N/A	0.356
Putty 0.76	**0.273**	N/A	0.095	0.507	N/A	**0.610**	**0.354**	N/A	0.164

Evaluation



(a) O0



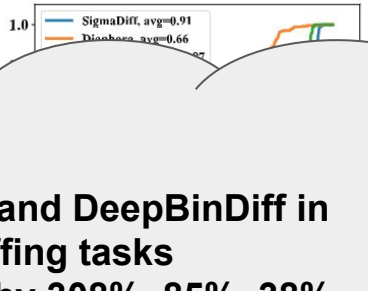


Fig. 6: Cross-optimizat

- **Outperforms Diaphora and DeepBinDiff in most of the diffing tasks**
- **Outperforms Diaphora by 308%, 85%, 38% in terms of F1-scores in O0 vs. O3, O1 vs. O3, and O2 vs. O3, respectively**

TABLE VI: Cross-comp

Coreutils 8.1	**0.595**								
Diffutils 3.6	0.295	0.187	0.029	**0.574**	0.445				0.055
Findutils 4.6	0.363	0.202	0.051	**0.619**	0.458	0.597		0.279	0.094
Gmp 6.2.1	**0.393**	N/A	0.227	0.597	N/A	**0.821**	**0.474**	N/A	0.356
Putty 0.76	**0.273**	N/A	0.095	0.507	N/A	**0.610**	**0.354**	N/A	0.164

Vulnerability/Patch Analysis

TABLE XI: Diffing results for Zoom. Si: SIGMADIFF, Di: Diaphora, Bi: BinDiff

Library	Ver.	Confirmed CVEs	Function-lvl			Token-lvl		
			Si	Di	Bi	Si	Di	Bi
OpenSSL	1.1.1k	CVE-2023-0464	✓		✓	✓		
		CVE-2023-0215	✓					
		CVE-2022-4450	✓	✓		✓		
		CVE-2022-0778	✓	✓	✓	✓		✓
		CVE-2021-3712	✓	✓		✓		
		CVE-2021-3711	✓			✓		
SQLite	3.33.0	CVE-2022-35737	✓					
		CVE-2021-20227	✓					
resiprocate	1.11	CVE-2021-3672	✓	✓				
		CVE-2017-9454	✓	✓				
libjpeg-turbo	2.0.4	CVE-2020-13790	✓	✓	✓	✓	✓	✓
FFmpeg	4.2.3	CVE-2021-38291	✓	✓	✓	✓	✓	✓
		CVE-2020-22037	✓	✓	✓	✓		

Vulnerability/Patch Analysis

TABLE XI: Diffing results for Zoom. Si: SIGMADIFF, Di: Diaphora, Bi: BinDiff

Library	Ver.	Confirmed CVEs	Function-lvl			Token-lvl		
			Si	Di	Bi	Si	Di	Bi
OpenSSL	1.1.1k	CVE-2023-0464 CVE-2023-0215 CVE-202 CVE CVI	✓ ✓		✓	✓		
SQLite	3.33.0							
resiprocate	1.11							
libjpeg-turbo	2.0.4	CVE-202						
FFmpeg	4.2.3	CVE-2021-3829 CVE-2020-22037	✓	✓	✓	✓		

- **Identify thirteen vulnerabilities in Windows (v5.9.7.3931) and Linux (v5.9.6.2225)**
- **Precisely pinpoint eight vulnerabilities at token level**

Open Source Project

https://github.com/yijiufly/SigmaDiff

Thank you!