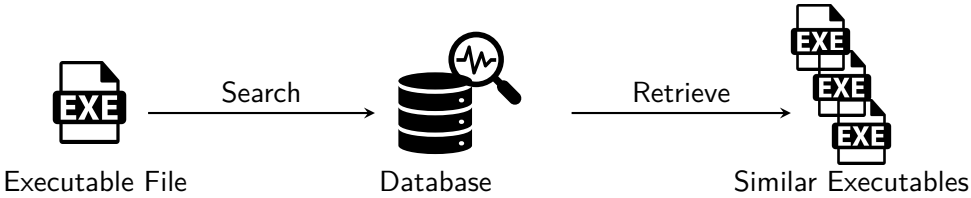
vSim: Semantics-Aware Value Extraction for Efficient Binary Code Similarity Analysis

Huaijin Wang, and Zhiqiang Lin

NDSS 2026

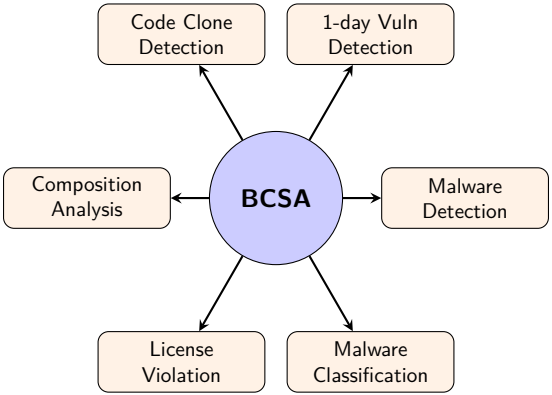
Background: What is Binary Code Similarity Analysis (BCSA)?



Background: What is Binary Code Similarity Analysis (BCSA)?

BCSA acts as a powerful search engine for binary code.

Applications of BCSA



A Core Challenge: Compilation Differences

```
int f2(int x, void* ptr) {
  if (ptr == 0)
    return -1;
  s.a = x * 3;
  f1(ptr, x);
  return 0;
}
```

Source code

```
<f2>:
cmpq 0x0,-0x10(%rbp)
jne  L2
movl 0xffffffff, %eax
<return from procedure>
L2:
imul 0x3,-0x4(%rbp),%eax
mov  %rax, [0x404050]
call <f1>
mov  0x0, %eax
<return from procedure>
```

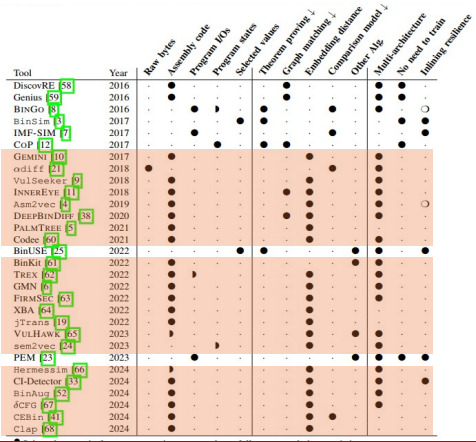
Compiled by clang –O0

```
<f2>:
mov  0x8(%esp),%eax
test %eax,%eax
je   L2
lea  (%eax,%eax,2),%ebx
mov  %ebx, [0x403fc8]
[inlined <f1>]
xor  %eax, %eax
ret
L2:
movl 0xffffffff, %eax
ret
```

Compiled by gcc –O3 (m32)

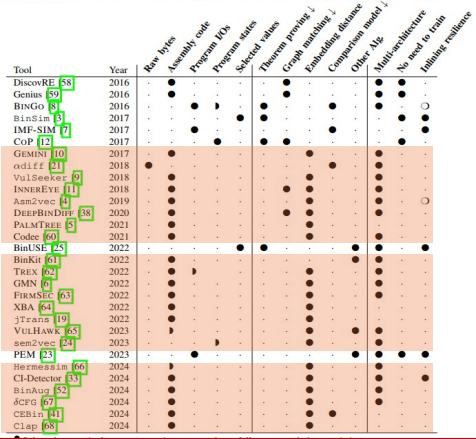
Same source → Different binaries

State of the Art: Machine-learning-based (ML-based) Methods



Tool	Year	Raw bytes	Assembly code	Program IOs	Program states	Selected values	Theorem proving ↘	Graph matching ↓	Embedding distance	Comparison model ↓	Other ML	Multi-architecture	No need to train	Inlining resilience
DiscovRE [58]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
Genius [59]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
BinGO [8]	2016	·	·	●	◗	·	●	·	·	●	·	●	·	○
BinSim [3]	2017	·	·	·	·	●	●	·	·	·	·	·	●	●
IMF-SIM [7]	2017	·	·	●	·	·	·	·	·	●	·	·	·	●
CoP [12]	2017	·	·	·	●	·	●	●	·	·	·	·	●	·
GEMINI [10]	2017	·	●	·	·	·	·	·	●	·	·	●	·	·
αdiff [21]	2018	●	·	·	·	·	·	·	·	●	·	●	·	·
VulSeeker [9]	2018	·	●	·	·	·	·	·	●	·	·	●	·	·
INNEREYE [11]	2018	·	●	·	·	·	·	●	●	·	·	●	·	·
Asm2vec [4]	2019	·	●	·	·	·	·	·	●	·	·	●	·	○
DEEPBINDIFF [38]	2020	·	●	·	·	·	·	●	●	·	·	●	·	·
PALMTREE [5]	2021	·	●	·	·	·	·	·	●	·	·	·	·	·
Codee [60]	2021	·	●	·	·	·	·	·	●	·	·	●	·	·
BinUSE [25]	2022	·	·	·	·	●	●	·	·	·	●	●	·	●
BinKit [61]	2022	·	●	·	·	·	·	·	·	·	●	●	·	·
TREX [62]	2022	·	●	◗	·	·	·	·	●	·	·	●	·	·
GMN [6]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
FIRMSEC [63]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
XBA [64]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
jTrans [19]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
VULHAWK [65]	2023	·	◗	·	·	·	·	·	●	·	●	●	·	·
sem2vec [24]	2023	·	·	·	◗	·	·	·	●	·	·	●	·	·
PEM [23]	2023	·	·	●	·	·	·	·	·	·	●	●	●	●
Hermessim [66]	2024	·	◗	·	·	·	·	·	●	·	·	●	·	·
CI-Detector [33]	2024	·	●	·	·	·	·	·	●	·	·	●	·	●
BinAug [52]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
δCFG [67]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
CEBin [41]	2024	·	●	·	·	·	·	·	●	●	·	·	·	·
Clap [68]	2024	·	●	·	·	·	·	·	●	·	·	·	·	·

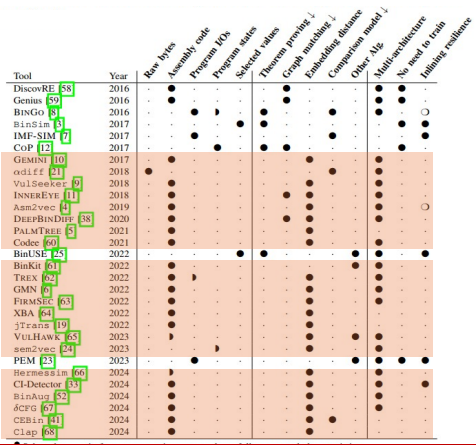
State of the Art: Machine-learning-based (ML-based) Methods



Tool	Year	Raw bytes	Assembly code	Program IOs	Program states	Selected values	Theorem proving ↓	Graph matching ↓	Embedding distance	Comparison model ↓	Other ML	Multi-architecture	No need to train	Inlining resilience
DiscovRE [58]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
Genius [59]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
BINGO [8]	2016	·	·	●	◗	·	●	·	·	●	·	●	·	○
BinSim [3]	2017	·	·	·	·	●	●	·	·	·	·	·	●	●
IMF-SIM [7]	2017	·	·	●	·	·	·	·	·	●	·	·	·	●
CoP [12]	2017	·	·	·	●	·	●	●	·	·	·	·	●	·
GEMINI [10]	2017	·	●	·	·	·	·	·	●	·	·	●	·	·
αdiff [21]	2018	●	·	·	·	·	·	·	·	●	·	●	·	·
VulSeeker [9]	2018	·	●	·	·	·	·	·	●	·	·	●	·	·
INNEREYE [11]	2018	·	●	·	·	·	·	●	●	·	·	●	·	·
Asm2vec [4]	2019	·	●	·	·	·	·	·	●	·	·	●	·	○
DEEPBINDIFF [38]	2020	·	●	·	·	·	·	●	●	·	·	●	·	·
PALMTREE [5]	2021	·	●	·	·	·	·	·	●	·	·	·	·	·
Codee [60]	2021	·	●	·	·	·	·	·	●	·	·	●	·	·
BinUSE [25]	2022	·	·	·	·	●	●	·	·	·	●	●	·	●
BinKit [61]	2022	·	●	·	·	·	·	·	·	·	●	●	·	·
TREX [62]	2022	·	●	◗	·	·	·	·	●	·	·	●	·	·
GMN [6]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
FIRMSEC [63]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
XBA [64]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
jTrans [19]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
VULHAWK [65]	2023	·	◗	·	·	·	·	·	●	·	●	●	·	·
sem2vec [24]	2023	·	·	·	◗	·	·	·	●	·	·	●	·	·
PEM [23]	2023	·	·	●	·	·	·	·	·	·	●	●	●	●
Hermessim [66]	2024	·	◗	·	·	·	·	·	●	·	·	●	·	·
CI-Detector [33]	2024	·	●	·	·	·	·	·	●	·	·	●	·	●
BinAug [52]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
ó CFG [67]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
CEBin [41]	2024	·	●	·	·	·	·	·	●	●	·	·	·	·
Clap [68]	2024	·	●	·	·	·	·	·	●	·	·	·	·	·

Models learn to extract semantics from binaries **built by a fixed set of compilers**.

State of the Art: Machine-learning-based (ML-based) Methods

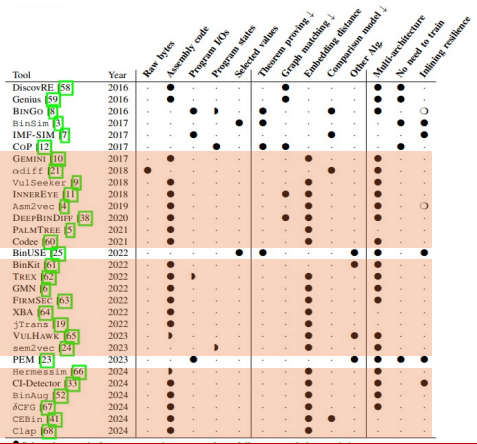


Models learn to extract semantics from binaries **built by a fixed set of compilers**.

Recall@1 of jTrans [WQK+22]

O3 \ O0	GCC-11.2	GCC-4.9	Clang-13	Clang-4
GCC-11.2	0.441	0.395	0.243	0.199
GCC-4.9	0.196	0.217	0.148	0.119
Clang-13	0.128	0.123	0.114	0.087
Clang-4	0.113	0.118	0.089	0.078

State of the Art: Machine-learning-based (ML-based) Methods



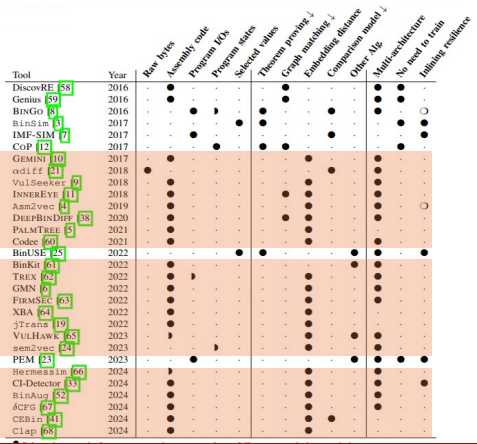
Models learn to extract semantics from binaries **built by a fixed set of compilers**.

Recall@1 of jTrans [WQK+22]

O3 \ O0	GCC-11.2	GCC-4.9	Clang-13	Clang-4
GCC-11.2	0.441	0.395	0.243	0.199
GCC-4.9	0.196	0.217	0.148	0.119
Clang-13	0.128	0.123	0.114	0.087
Clang-4	0.113	0.118	0.089	0.078

- **Not robust** to compiler changes

State of the Art: Machine-learning-based (ML-based) Methods



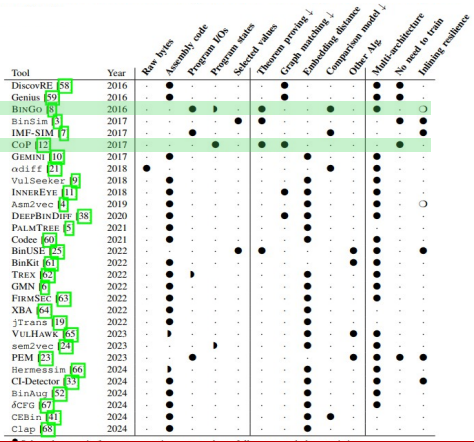
Models learn to extract semantics from binaries **built by a fixed set of compilers**.

Recall@1 of jTrans [WQK+22]

O3 \ O0	GCC-11.2	GCC-4.9	Clang-13	Clang-4
GCC-11.2	0.441	0.395	0.243	0.199
GCC-4.9	0.196	0.217	0.148	0.119
Clang-13	0.128	0.123	0.114	0.087
Clang-4	0.113	0.118	0.089	0.078

- **Not robust** to compiler changes
- **Not interpretable**

State of the Art: Program State-based Methods



Tool	Year	Raw bytes	Assembly code	Program IOs	Program states	Selected values	Theorem proving ↓	Graph matching ↓	Embedding distance	Comparison model ↓	Other Alg.	Multi-architecture	No need to train	Inlining resilience
DiscovRE [58]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
Genius [59]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
BINGO [8]	2016	·	·	●	◗	·	●	·	·	●	·	●	·	○
BinSim [3]	2017	·	·	·	·	●	●	·	·	·	·	·	●	●
IMF-SIM [7]	2017	·	·	●	·	·	·	·	·	●	·	·	·	●
CoP [12]	2017	·	·	·	●	·	●	●	·	·	·	·	●	·
GEMINI [10]	2017	·	●	·	·	·	·	·	●	·	·	●	·	·
αdiff [21]	2018	●	·	·	·	·	·	·	·	●	·	●	·	·
VulSeeker [9]	2018	·	●	·	·	·	·	·	●	·	·	●	·	·
INNEREYE [11]	2018	·	●	·	·	·	·	●	●	·	·	●	·	·
Asm2vec [4]	2019	·	●	·	·	·	·	·	●	·	·	●	·	○
DEEPBINDIFF [38]	2020	·	●	·	·	·	·	●	●	·	·	●	·	·
PALMTREE [5]	2021	·	●	·	·	·	·	·	●	·	·	·	·	·
Codee [60]	2021	·	●	·	·	·	·	·	●	·	·	●	·	·
BinUSE [25]	2022	·	·	·	·	●	●	·	·	·	●	●	·	●
BinKit [61]	2022	·	●	·	·	·	·	·	·	·	●	●	·	·
TREX [62]	2022	·	●	◗	·	·	·	·	●	·	·	●	·	·
GMN [6]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
FIRMSEC [63]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
XBA [64]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
jTrans [19]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
VULHAWK [65]	2023	·	◗	·	·	·	·	·	●	·	●	●	·	·
sem2vec [24]	2023	·	·	·	◗	·	·	·	●	·	·	●	·	·
PEM [23]	2023	·	·	●	·	·	·	·	·	·	●	●	●	●
Hermessim [66]	2024	·	◗	·	·	·	·	·	●	·	·	●	·	·
CI-Detector [33]	2024	·	●	·	·	·	·	·	●	·	·	●	·	●
BinAug [52]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
δCFG [67]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
CEBin [41]	2024	·	●	·	·	·	·	·	●	●	·	·	·	·
Clap [68]	2024	·	●	·	·	·	·	·	●	·	·	·	·	·

- [CXX+16, LMW+17]
  - Collect all symbolic/concrete values during execution
  - **Be influenced by the noises of semantics-irrelevant values**

Observation 1: Inaccurate Semantics (Program State-based)

```
struct S {size_t a, int b};
struct S s;

void f1(void* ptr, int y) {
  if (y < 8) {
    s.b = 0xdeadbeef;
  }
  else {
    s.b = ((struct S*)ptr)->b;
  }
}
```

Source code

```
.bss 0x404050,
L1:
mov  -0x8(%rbp),%rax
mov  0x8(%rax),%eax
mov  %eax, [0x404058]
< return from procedure>
```

Compiled by clang –O0

```
.bss 0x403fc8
mov  0x4(%esp),%eax
mov  0x4(%eax),%eax
L1:
mov  %eax, [0x403fcc]
ret
```

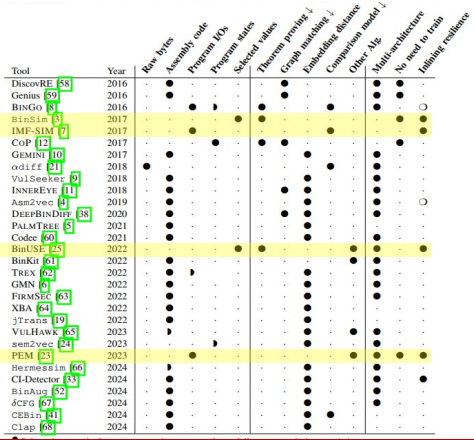
Compiled by gcc –O3 (m32)

```
rax ← ptr
rax ← MEM(ptr + 8, 4)
s.b ← MEM(ptr + 8, 4)
```

Different **address expressions**

```
eax ← ptr'
eax ← MEM(ptr'+4, 4)
s.b ← eax
```

State of the Art: Selected Value-based Methods



Tool	Year	Raw bytes	Assembly code	Program IOs	Program states	Selected values	Theorem proving ↓	Graph matching ↓	Embedding distance	Comparison model ↓	Other Alg.	Multi-architecture	No need to train	Inlining resilience
DiscovRE [58]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
Genius [59]	2016	·	●	·	·	·	·	●	·	·	·	●	●	·
BINGO [8]	2016	·	·	●	◗	·	·	●	·	●	·	●	·	○
BinSim [3]	2017	·	·	·	·	●	●	·	·	·	·	·	●	●
IMF-SIM [7]	2017	·	·	●	·	·	·	·	·	●	·	·	·	●
CoP [12]	2017	·	·	·	●	·	●	●	·	·	·	·	●	·
GEMINI [10]	2017	·	●	·	·	·	·	·	●	·	·	●	·	·
αdiff [21]	2018	●	·	·	·	·	·	·	·	●	·	●	·	·
VulSeeker [9]	2018	·	●	·	·	·	·	·	●	·	·	●	·	·
INNEREYE [11]	2018	·	●	·	·	·	·	●	●	·	·	●	·	·
Asm2vec [4]	2019	·	●	·	·	·	·	·	●	·	·	●	·	○
DEEPBINDIFF [38]	2020	·	●	·	·	·	·	●	●	·	·	●	·	·
PALMTREE [5]	2021	·	●	·	·	·	·	·	●	·	·	·	·	·
Codee [60]	2021	·	●	·	·	·	·	·	●	·	·	●	·	·
BinUSE [25]	2022	·	·	·	·	●	●	·	·	·	●	●	·	●
BinKit [61]	2022	·	●	·	·	·	·	·	·	·	●	●	·	·
TREX [62]	2022	·	●	◗	·	·	·	·	●	·	·	●	·	·
GMN [6]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
FIRMSEC [63]	2022	·	●	·	·	·	·	·	●	·	·	●	·	·
XBA [64]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
jTrans [19]	2022	·	●	·	·	·	·	·	●	·	·	·	·	·
VULHAWK [65]	2023	·	◗	·	·	·	·	·	●	·	●	●	·	·
sem2vec [24]	2023	·	·	·	◗	·	·	·	●	·	·	●	·	·
PEM [23]	2023	·	·	●	·	·	·	·	·	·	●	●	●	●
Hermessim [66]	2024	·	◗	·	·	·	·	·	·	·	·	●	·	·
CI-Detector [33]	2024	·	●	·	·	·	·	·	●	·	·	●	·	●
BinAug [52]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
ő CFG [67]	2024	·	●	·	·	·	·	·	●	·	·	●	·	·
CEBin [41]	2024	·	●	·	·	·	·	·	●	●	·	·	·	·
Clap [68]	2024	·	●	·	·	·	·	·	●	·	·	·	·	·

- [WMY+22, MXJW17, XXF+23]
  - Select semantically-meaningful values (e.g., return value of a function)
  - **Neglect abundant semantics**

Observation 1: Incomprehensive Semantics (Selected Value-based)

```
int f2(int x, void* ptr) {
  if (ptr == 0)
    return -1;
  s.a = x * 3;
  f1(ptr, x);
  return 0;
}
```

Source code

```
<f2>:
cmpq 0x0,-0x10(%rbp)
jne  L2
movl 0xffffffff, %eax          return -1
<return from procedure>
L2:
imul 0x3,-0x4(%rbp),%eax
mov  %rax, [0x404050]
call <f1>
mov  0x0, %eax                 return 0
<return from procedure>
```

Compiled by clang –O0

```
<f2>:
mov  0x8(%esp),%eax
test %eax,%eax
je   L2
lea  (%eax,%eax,2),%ebx
mov  %ebx, [0x403fc8]
[inlined <f1>]
xor  %eax, %eax                return 0
ret
L2:
movl 0xffffffff, %eax          return -1
ret
```

Compiled by gcc –O3 (m32)

Select semantically-meaningful values.
(e.g., return results)

Observation 1: Incomprehensive Semantics (Selected Value-based)

```
int f2(int x, void* ptr) {
  if (ptr == 0)
    return -1;
  s.a = x * 3;
  f1(ptr, x);
  return 0;
}
```

Source code

```
<f2>:
cmpq 0x0,-0x10(%rbp)
jne  L2
movl 0xffffffff, %eax
<return from procedure>
L2:
imul 0x3,-0x4(%rbp),%eax
mov  %rax, [0x404050]
call <f1>
mov  0x0, %eax
<return from procedure>
```

return -1

return 0

Compiled by clang –O0

```
<f2>:
mov  0x8(%esp),%eax
test %eax,%eax
je   L2
lea  (%eax,%eax,2),%ebx
mov  %ebx, [0x403fc8]
[inlined <f1>]
xor  %eax, %eax
ret
L2:
movl 0xffffffff, %eax
ret
```

return 0

return -1

Compiled by gcc –O3 (m32)

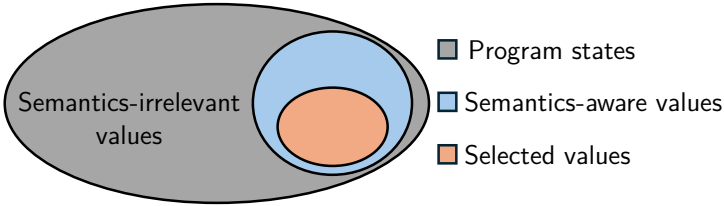
Select semantically-meaningful values.
(e.g., return results)

Limitations

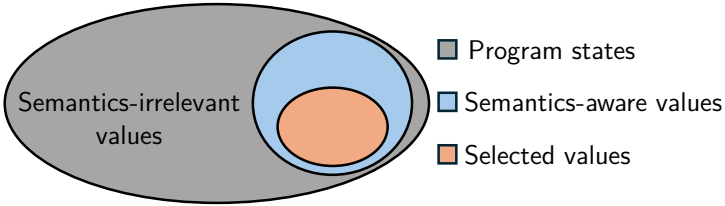
Ignore other valuable semantics.
(e.g., side-effects)

The selected value may not exist.
(e.g., functions with void return)

Motivation 1: Semantics-Aware Value Extraction

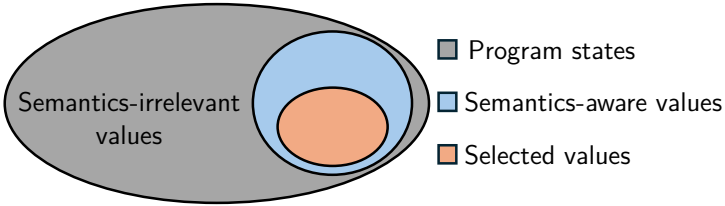


Motivation 1: Semantics-Aware Value Extraction



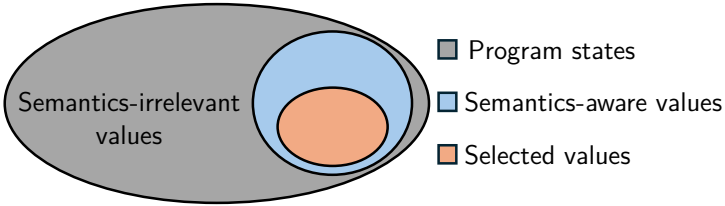
- Program states: collect all values being used while execution

Motivation 1: Semantics-Aware Value Extraction



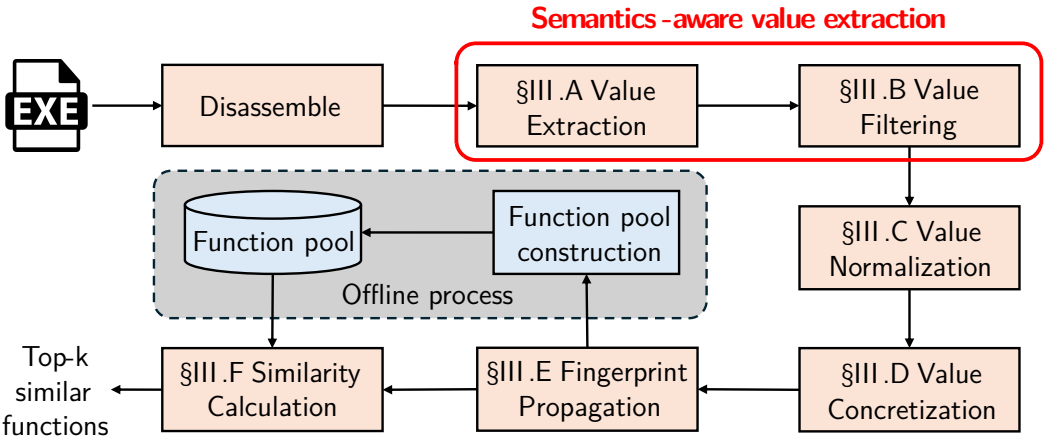
- Program states: collect all values being used while execution
- Selected values: select limited values (e.g., return result of a function)

Motivation 1: Semantics-Aware Value Extraction



- Program states: collect all values being used while execution
- Selected values: select limited values (e.g., return result of a function)
- **Semantics-aware values: collect all values, then *remove values being used as addresses*.**

Design of vSim: Value Extraction & Value Filtering



Observation 2: Unscalable Comparisons (Program Analysis Methods)

Source code

```
int f2(int x, void* ptr) {
  if (ptr == 0)
    return -1;
  s.a = x * 3;
  f1(ptr, x);
  return 0;
}
```

Compiled by clang –O0

```
imul 0x3,-0x4(%rbp),%eax
mov  %rax, [0x404050]
```

$\text{Concat}(\text{BV}(0,32), \text{BV}(3,32) \times \text{BV}(x,32))$

e_1

Compiled by gcc –O3 (m32)

```
lea  (%eax,%eax,2),%ebx
mov  %ebx, [0x403fc8]
```

$\text{BV}(2,32) \times \text{BV}(x',32) + \text{BV}(x',32)$

e_1

To prove: $\text{BV}(x,32) = \text{BV}(x',32) \Rightarrow e_1 = e_2$

Observation 2: Unscalable Comparisons (Program Analysis Methods)

Source code

```
int f2(int x, void* ptr) {
  if (ptr == 0)
    return -1;
  s.a = x * 3;
  f1(ptr, x);
  return 0;
}
```

Compiled by clang –O0

```
imul 0x3,-0x4(%rbp),%eax
mov  %rax, [0x404050]
```

$\text{Concat}(\text{BV}(0,32), \text{BV}(3,32) \times \text{BV}(x,32))$

e_1

Compiled by gcc –O3 (m32)

```
lea  (%eax,%eax,2),%ebx
mov  %ebx, [0x403fc8]
```

$\text{BV}(2,32) \times \text{BV}(x',32) + \text{BV}(x',32)$

e_2

To prove: $\text{BV}(x,32) = \text{BV}(x',32) \Rightarrow e_1 = e_2$

Existing works [WMY+22, LMW+17] depend on theory proving.

Sound but unscalable.

Motivation 2: Normalization & Concretization

Source code

```
int f2(int x, void* ptr) {
  if (ptr == 0)
    return -1;
  s.a = x * 3;
  f1(ptr, x);
  return 0;
}
```

Compiled by clang –O0

```
imul 0x3,-0x4(%rbp),%eax
mov  %rax, [0x404050]
```

Compiled by gcc –O3 (m32)

```
lea  (%eax,%eax,2),%ebx
mov  %ebx, [0x403fc8]
```

$\text{Concat}(\text{BV}(0,32), \text{BV}(3,32) \times \text{BV}(x,32))$ $\quad$ $\text{BV}(2,32) \times \text{BV}(x',32) + \text{BV}(x',32)$

Normalization : Strip bit-vector size

$0 \ll 32 \mid 3 \times x$ $\quad$ $2 \times x' + x'$

Concretization : Iterate x and x' in a random array

$(39, 132, 156, 171, 243)$

Motivation 2: Normalization & Concretization

Source code

```
int f2(int x, void* ptr) {
  if (ptr == 0)
    return -1;
  s.a = x * 3;
  f1(ptr, x);
  return 0;
}
```

Compiled by clang –O0

```
imul 0x3,-0x4(%rbp),%eax
mov  %rax, [0x404050]
```

Compiled by gcc –O3 (m32)

```
lea  (%eax,%eax,2),%ebx
mov  %ebx, [0x403fc8]
```

$\text{Concat}(\text{BV}(0,32), \text{BV}(3,32) \times \text{BV}(x,32))$ $\quad$ $\text{BV}(2,32) \times \text{BV}(x',32) + \text{BV}(x',32)$

Normalization : Strip bit-vector size

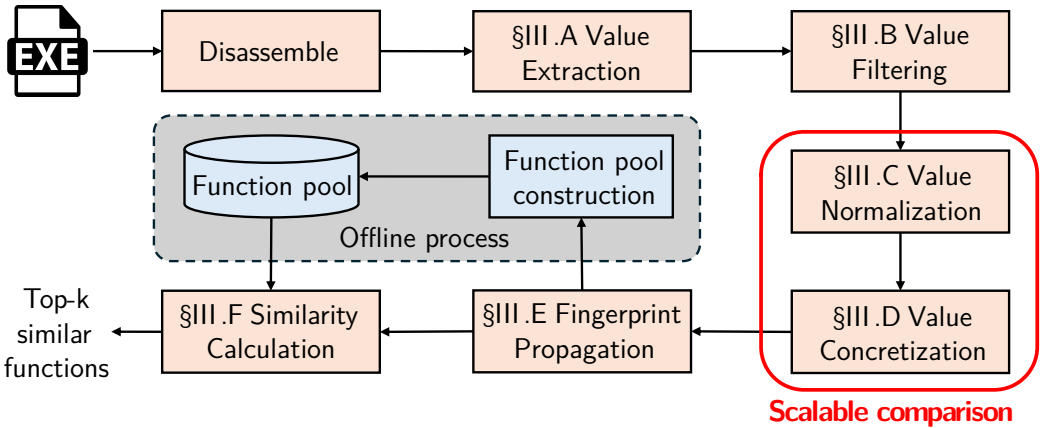
$0 \ll 32 \mid 3 \times x$ $\quad$ $2 \times x' + x'$

Concretization : Iterate x and x' in a random array

$(39, 132, 156, 171, 243)$

*Sacrifice soundness but **significantly increase efficiency**.*

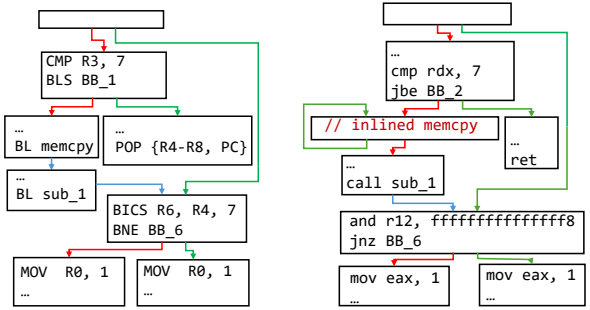
Design of vSim: Value Normalization & Value Concretization



Observation 3: Missing Callees' Semantics

- Function inlining is a *prevalent* compiler optimization
  - GCC, Clang, and Intel CC enable function inlining by default.
- Existing BCSA solutions
  - No solution (e.g., GMN [MGUP+22], jTrans [WQK+22], and etc)
  - Selective inlining based on unreliable heuristics (e.g., BinGO [CXX+16], Asm2vec [DFC19]).

Observation 3: Missing Callees' Semantics



`MDC2_Update` of NetGear `MDC2_Update` (x64)

Similar control flow structures.

GMN [MGUP+22] failed to get a high similarity due to the inlined `memcpy`.

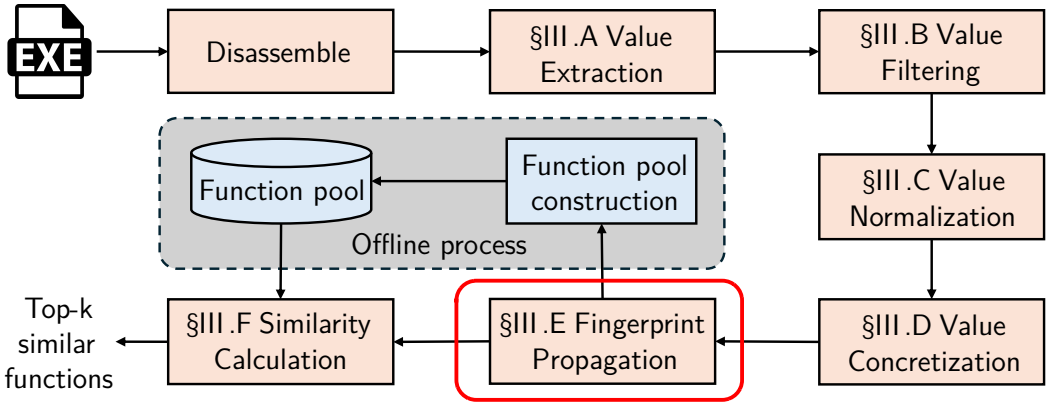
Motivation 3: Fingerprint Propagation

$f2$ calls $f1$

$f2$ does not inline $f1$	$f2$ inlines $f1$
$S_{(b)f1} = \{((>,7),(\geq,8)), -559038737\}$	$S_{(c)f1} = \{((>,7),(\geq,8)), -559038737\}$
$S_{(b)f2} = \left\{ \begin{matrix} (=,0), -1, 0 \\ (39,132,156,171,243) \end{matrix} \right\}$	$S_{(c)f2} = \left\{ \begin{matrix} (=,0), -1, 0, \\ (39,132,156,171,243), \\ ((>,7),(\geq,8)), -559038737 \end{matrix} \right\}$

- If a caller $f2$ inlines the callee $f1$, the values of $f1$ is a subset of the values of $f2$.
- Fingerprint propagation is an **efficient** set union. $FP_{(b)f2} = S_{(b)f1} \cup S_{(b)f2}$
- Jaccard Similarity: $\frac{FP_{(b)f2} \cap S_{(c)f2}}{FP_{(b)f2} \cup S_{(c)f2}} = 1.0$

Design of vSim: Fingerprint Propagation



Evaluation Settings & Baselines

- **Target:** Evaluating across optimization levels, compiler vendors/versions, and CPU architectures.
- **Baselines Compared:**
  - `jTrans` [WQK+22]: Embedding-based model.
  - `Clap` [WGZ+24]: Embedding-based model.
  - PEM-s [XXF+23]: Dynamic analysis technique.
  - `GMN` [MGUP+22]: ML-model computing similarity scores directly.
- **Metrics:** Recall@k and MRR@k (the metrics of information retrieval systems).

Cross-Optimization & Retrieval Cost

Cross-optimization evaluation

Tool	MRR[1]		Recall@1		Retrieval time (s)[2]
	O0,O3	O0,O2	O0,O3	O0,O2	
jTrans	0.467	0.511	0.374	0.420	**6.1**
Clap	0.603	0.647	**0.548**	0.596	**6.1**
PEM-s	0.605	0.701	0.534	0.629	1853
vSim	**0.621**	**0.709**	0.545	**0.642**	87
(No filtering)	0.527	0.614	0.453	0.548	329
(No propagation)	0.508	0.540	0.464	0.498	22

[1] k is `+inf` by default when it is not given.
[2] The comparison processes of three tools are highly parallelized with 64 threads. The reported time is the average of 5 runs.

The scalability of vSim is **comparable** to using machine learning embeddings (a matrix production for similarity calculation).

Usefulness of Semantics Awareness & Propagating Callees' Semantics

Cross-optimization evaluation

Tool	MRR[1]		Recall@1		Retrieval time (s)[2]
	O0,O3	O0,O2	O0,O3	O0,O2	
`jTrans`	0.467	0.511	0.374	0.420	**6.1**
`Clap`	0.603	0.647	**0.548**	0.596	**6.1**
PEM-s	0.605	0.701	0.534	0.629	1853
vSIM	**0.621**	**0.709**	0.545	**0.642**	87
(No filtering)	0.527	0.614	0.453	0.548	329
(No propagation)	0.508	0.540	0.464	0.498	22

[1] k is `+inf` by default when it is not given.
[2] The comparison processes of three tools are highly parallelized with 64 threads. The reported time is the average of 5 runs.

The scalability of vSim is **comparable** to using machine learning embeddings (a matrix production for similarity calculation).

Semantics-awareness and fingerprint propagation significantly contributes to the performance of vSim.

Cross-Compiler Robustness

Cross-compiler evaluation

Recall@1 of `jTrans` (Std Dev 0.107)				
O3 \ O0	**GCC-11.2**	**GCC-4.9**	**Clang-13**	**Clang-4**
GCC-11.2	0.441	0.395	0.243	0.199
GCC-4.9	0.196	0.217	0.148	0.119
Clang-13	0.128	0.123	0.114	0.087
Clang-4	0.113	0.118	0.089	0.078
Recall@1 of vSim (Std Dev 0.026)				
GCC-11.2	0.711	0.689	0.667	0.678
GCC-4.9	0.637	0.701	0.649	0.611
Clang-13	0.638	0.662	0.666	0.623
Clang-4	0.672	0.656	0.660	0.664

The best results are highlighted in light blue and the worst results are highlighted in gray.

jTrans performance decrease 82% due to different compiler versions.

vSim shows better Recall@1 and less Std Dev.

vSim does not rely on the training data and is **robust to the change of compilers** .

Cross-Architecture and 1-day Vulnerability Detection

Cross-architecture evaluation

Tool	MRR@10		Recall@1		Retrieval time (s)[2]
	XO	XA+XO	XO	XA+XO	
GMN[1]	0.75	0.71	0.66	0.61	1005
vSim	0.86	0.70	0.79	0.63	15

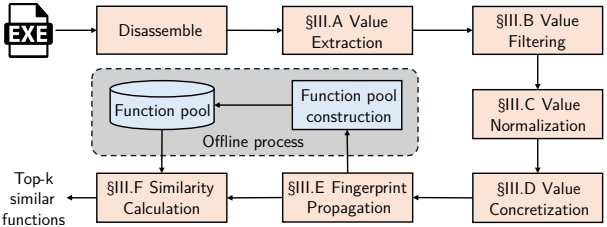
Vulnerable function detection evaluation (MRR@10)

Tool	Netgear R7000				
	x86	AMD64	ARM32	MIPS32	Average
GMN[1]	**0.88**	0.54	**1.00**	0.79	0.80
vSim	**0.88**	**0.81**	0.88	**0.88**	**0.86**
	TP-Link Deco-M4				
GMN[1]	0.67	**0.73**	0.70	0.78	0.72
vSim	**0.8**	0.69	**0.79**	**0.81**	**0.77**

GMN uses a machine-learning model to compute the similarity score (Not embedding-based approach).

vSim shows **better** **accuracy** and **scalability** .

Conclusion



vSim

- **Semantics-Aware Filtering**
- **Scalable Comparison**
- **Inlining Resilience**
- **Robustness**

Thank you!

Thanks for listening!

vSim is open source.

https://github.com/OSUSecLab/vSim

References I

Mahinthan Chandramohan, Yinxing Xue, Zhengzi Xu, Yang Liu, Chia Yuan Cho, and Hee Beng Kuan Tan, *Bingo: Cross-architecture cross-os binary search*, Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, 2016, pp. 678–689.

Steven HH Ding, Benjamin CM Fung, and Philippe Charland, *Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization*, 2019 ieee symposium on security and privacy, IEEE, 2019, pp. 472–489.

Lannan Luo, Jiang Ming, Dinghao Wu, Peng Liu, and Sencun Zhu, *Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection*, IEEE Transactions on Software Engineering **43** (2017), no. 12, 1157–1177.

Andrea Marcelli, Mariano Graziano, Xabier Ugarte-Pedrero, Yanick Fratantonio, Mohamad Mansouri, and Davide Balzarotti, *How machine learning is solving the binary function similarity problem*, 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 2099–2116.

Jiang Ming, Dongpeng Xu, Yufei Jiang, and Dinghao Wu, *Binsim: Trace-based semantic binary diffing via system call sliced segment equivalence checking*, Proceedings of the 26th USENIX Security Symposium, 2017.

Hao Wang, Zeyu Gao, Chao Zhang, Zihan Sha, Mingyang Sun, Yuchen Zhou, Wenyu Zhu, Wenju Sun, Han Qiu, and Xi Xiao, *Clap: Learning transferable binary code representations with natural language supervision*, Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, 2024, pp. 503–515.

Huaijin Wang, Pingchuan Ma, Yuanyuan Yuan, Zhibo Liu, Shuai Wang, Qiyi Tang, Sen Nie, and Shi Wu, *Enhancing DNN-based binary code function search with low-cost equivalence checking*, IEEE Transactions on Software Engineering **49** (2022), no. 1, 226–250.

References II

Hao Wang, Wenjie Qu, Gilad Katz, Wenyu Zhu, Zeyu Gao, Han Qiu, Jianwei Zhuge, and Chao Zhang, *jTrans: jump-aware transformer for binary code similarity detection*, Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA, Association for Computing Machinery, 2022, pp. 1–13.

Xiangzhe Xu, Zhou Xuan, Shiwei Feng, Siyuan Cheng, Yapeng Ye, Qingkai Shi, Guanhong Tao, Le Yu, Zhuo Zhang, and Xiangyu Zhang, *PEM: Representing binary program semantics for similarity analysis via a probabilistic execution model*, Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2023, pp. 401–412.