Assessing the Effectiveness of the Tigress Obfuscator Against MOPSA and BinaryNinja

Nicolò Altamura
University of Verona

Enrico Bragastini
University of Verona

Marco Campion
INRIA & ENS Paris

Mila Dalla Preda
University of Verona

Taipei, Taiwan – CheckMATE Workshop (CCS) 2025

Context

- **The measurement challenge**

Context

- **The measurement challenge**
  - How do you formally define obfuscation effectiveness?

Context

- **The measurement challenge**
  - How do you formally define obfuscation effectiveness?
  - Assessing obfuscation is inherently challenging and often confusing for the variety of tools and techniques

Context

- **The measurement challenge**
  - How do you formally define obfuscation effectiveness?
  - Assessing obfuscation is inherently challenging and often confusing for the variety of tools and techniques
- **Our contribution**

Context

- **The measurement challenge**
  - How do you formally define obfuscation effectiveness?
  - Assessing obfuscation is inherently challenging and often confusing for the variety of tools and techniques
- **Our contribution**
  - We constrain the attacker to specific static analysis tools

Context

- **The measurement challenge**
  - How do you formally define obfuscation effectiveness?
  - Assessing obfuscation is inherently challenging and often confusing for the variety of tools and techniques
- **Our contribution**
  - We constrain the attacker to specific static analysis tools
  - This allows us to systematically measure the confusion added by simple code obfuscation strategies

Context

- **The measurement challenge**
  - How do you formally define obfuscation effectiveness?
  - Assessing obfuscation is inherently challenging and often confusing for the variety of tools and techniques
- **Our contribution**
  - We constrain the attacker to specific static analysis tools
  - This allows us to systematically measure the confusion added by simple code obfuscation strategies
- **The goal**

Context

- **The measurement challenge**
  - How do you formally define obfuscation effectiveness?
  - Assessing obfuscation is inherently challenging and often confusing for the variety of tools and techniques
- **Our contribution**
  - We constrain the attacker to specific static analysis tools
  - This allows us to systematically measure the confusion added by simple code obfuscation strategies
- **The goal**
  - We want to measure how much obfuscation degrades the static analysis at the source code level and at the binary level

Tigress

- A source-level obfuscator designed for C programs

Tigress

- A source-level obfuscator designed for C programs
- Aims at protecting software against reverse engineering and tampering

Tigress

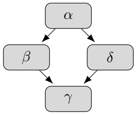
- A source-level obfuscator designed for C programs
- Aims at protecting software against reverse engineering and tampering
- Applies a wide range of transformations to hinder code analysis

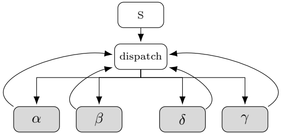
Tigress

- A source-level obfuscator designed for C programs
- Aims at protecting software against reverse engineering and tampering
- Applies a wide range of transformations to hinder code analysis
- Employs key techniques such as data encoding,

Tigress

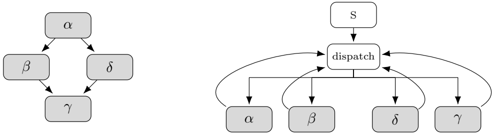
- A source-level obfuscator designed for C programs
- Aims at protecting software against reverse engineering and tampering
- Applies a wide range of transformations to hinder code analysis
- Employs key techniques such as data encoding, control flow obfuscation

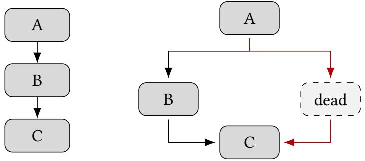




Tigress

- A source-level obfuscator designed for C programs
- Aims at protecting software against reverse engineering and tampering
- Applies a wide range of transformations to hinder code analysis
- Employs key techniques such as data encoding, control flow obfuscation and the insertion of opaque predicates





Mopsa

- Source-level static analyzer built on the principles of abstract interpretation

Mopsa

- Source-level static analyzer built on the principles of abstract interpretation
- Targets C, Python languages

Mopsa

- Source-level static analyzer built on the principles of abstract interpretation
- Targets C, Python languages
- The interval domain (**non relational**):

$$Int = \{[a, b] \mid a, b \in \mathbb{R} \cup \{-\infty, +\infty\}, a \leq b\} \cup \{\bot\},$$

Mopsa

- Source-level static analyzer built on the principles of abstract interpretation
- Targets C, Python languages
- The interval domain (**non relational**):

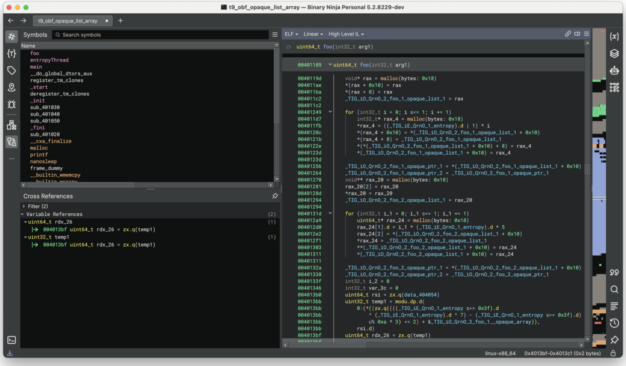
$$Int = \{[a, b] \mid a, b \in \mathbb{R} \cup \{-\infty, +\infty\}, a \leq b\} \cup \{\perp\},$$

- The polyhedra abstract domain (**relational**):

$$\mathcal{P} = \left\{P \subseteq \mathbb{R}^n \mid P = \{x \in \mathbb{R}^n \mid Ax \leq b\}, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m\right\} \cup \{\perp\}.$$

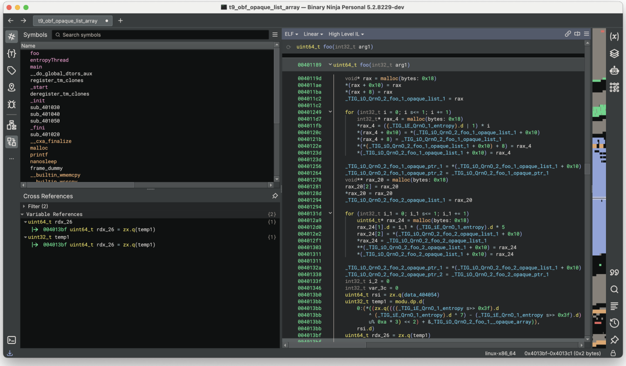
Binary Ninja

- Reverse engineering framework with decompilation capabilities



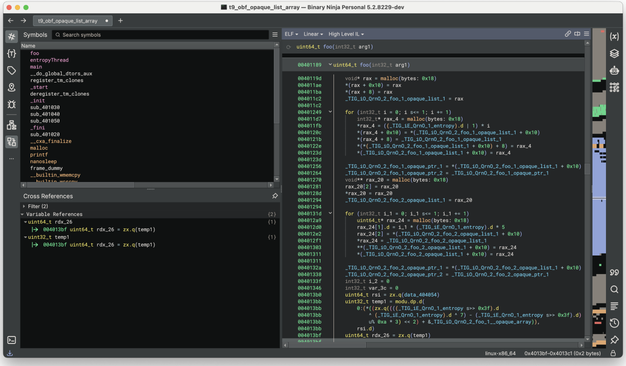
Binary Ninja

- Reverse engineering framework with decompilation capabilities
- Offers code inspection through intermediate representations at three abstraction levels



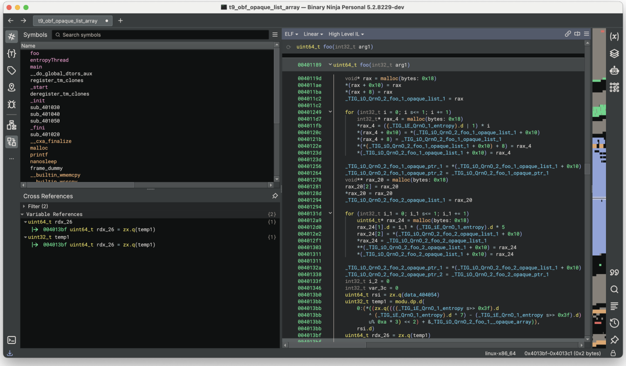
Binary Ninja

- Reverse engineering framework with decompilation capabilities
- Offers code inspection through intermediate representations at three abstraction levels
- Generates Pseudo-C code



Binary Ninja

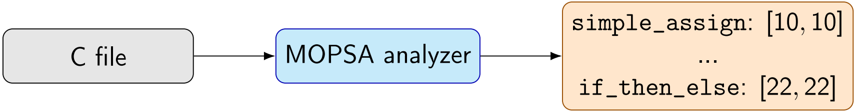
- Reverse engineering framework with decompilation capabilities
- Offers code inspection through intermediate representations at three abstraction levels
- Generates Pseudo-C code
- Provides an API for automating analysis and exporting results



BinaryNinja – https://binary.ninja/

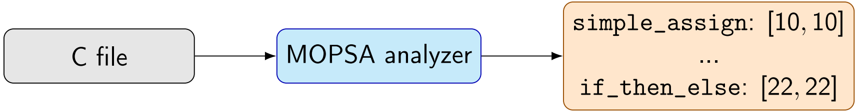
Evaluation setup at source level

- Without obfuscation:

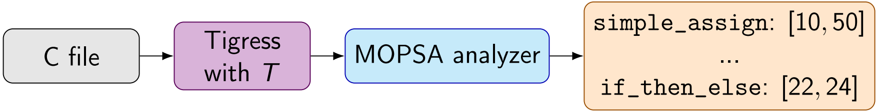


Evaluation setup at source level

- Without obfuscation:



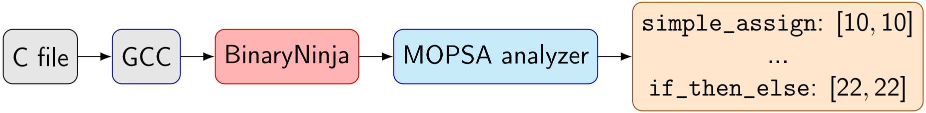
- for $T \in \mathit{Transformations}$:



$$\mathit{Transformations} = \{\mathit{DataEncoding} + \mathit{DataArithmetic}, \mathit{Opaque}, \mathit{Flatten}, \mathit{OpaqueConst}\}$$

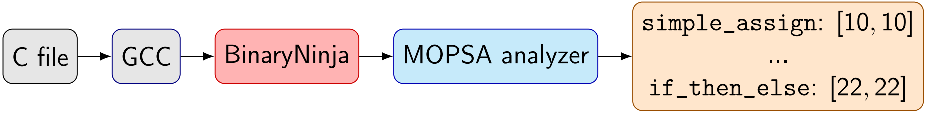
Evaluation setup at binary level

- Without obfuscation:

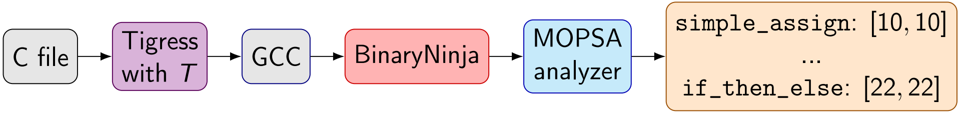


Evaluation setup at binary level

- Without obfuscation:



- for $T \in \mathit{Transformations}$:



$$\mathit{Transformations} = \{\mathit{DataEncoding} + \mathit{DataArithmetic}, \mathit{Opaque}, \mathit{Flatten}, \mathit{OpaqueConst}\}$$

Experiment 1: Mopsa on the original dataset

- MOPSA executed on all programs of the two datasets

Experiment 1: Mopsa on the original dataset

- MOPSA executed on all programs of the two datasets
- Retrieved the output via `_mopsa_print(res)`

Experiment 1: Mopsa on the original dataset

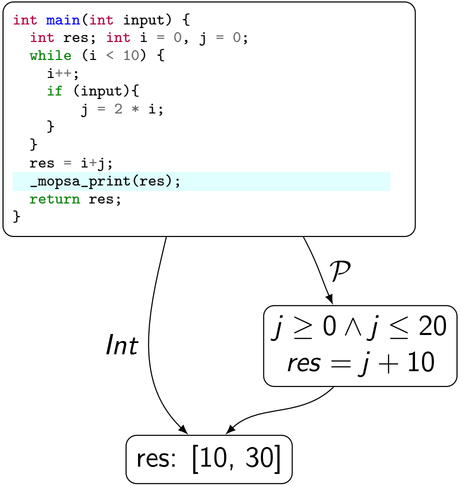
- MOPSA executed on all programs of the two datasets
- Retrieved the output via `_mopsa_print(res)`
- Used fixed loop unrolling (10000)

Experiment 1: Mopsa on the original dataset

- MOPSA executed on all programs of the two datasets
- Retrieved the output via `_mopsa_print(res)`
- Used fixed loop unrolling (10000)
- Result expressed in the interval form for both abstract domains

Experiment 1: Mopsa on the original dataset

- MOPSA executed on all programs of the two datasets
- Retrieved the output via `_mopsa_print(res)`
- Used fixed loop unrolling (10000)
- Result expressed in the interval form for both abstract domains



E1-BN: Mopsa with Binary Ninja

- Extracted the code using *Pseudo-C* representation

E1-BN: Mopsa with Binary Ninja

- Extracted the code using *Pseudo-C* representation
- Obtained the same results at binary level

E1-BN: Mopsa with Binary Ninja

- Extracted the code using *Pseudo-C* representation
- Obtained the same results at binary level
- In absence of obfuscation, the combination of BinaryNinja and mopsa can extract meaningful information from the binary

E1-BN: Mopsa with Binary Ninja

- Extracted the code using *Pseudo-C* representation
- Obtained the same results at binary level
- In absence of obfuscation, the combination of BinaryNinja and mopsa can extract meaningful information from the binary

file	E1: original	E1-BN: original
for_loop	[70, 70]	[70, 70]
for_loop_3	[480, 480]	[480, 480]
t04	[100, 127]	[100, 127]
t05	[100, 100]	[100, 100]

E2: Data obfuscation

- Two main transformations
  - `EncodeData` (poly1)

E2: Data obfuscation

- Two main transformations
  - `EncodeData` (poly1)
  - `EncodeArithmetic` (MBA)

E2: Data obfuscation

- Two main transformations
  - `EncodeData` (poly1)
  - `EncodeArithmetic` (MBA)
- No confusion added: The analysis with mopsa returns either the same results obtained on the original code or goes to $\top$

E2: Data obfuscation

- Two main transformations
  - `EncodeData` (poly1)
  - `EncodeArithmetic` (MBA)
- No confusion added: The analysis with mopsa returns either the same results obtained on the original code or goes to $\top$

file	**E1: original**	**E2: data_obf**
for_loop_1	[62, 62]	[62, 62]
for_loop_2	[5, 5]	[5, 5]
while_cycle_1	[12, 12]	[12, 12]
while_cycle_2	[27, 27]	[27, 27]

E2-BN: Data obfuscation

- `EncodeData` and `EncodeArithmetic`
- In some programs, MOPSA returns an unsound solution

file	E1-BN: original	E2-BN: data-obf
for_loop	[70, 70]	[46, 46]
t01	[100, 100]	[100, 100]
t04	[100, 127]	$\top$
t05	[100, 100]	[13825, 13825]
t09	[10, 30]	[0, 0]

E2-BN: Data obfuscation

- `EncodeData` and `EncodeArithmetic`
- In some programs, MOPSA returns an unsound solution

file	E1-BN: original	E2-BN: data-obf
for_loop	[70, 70]	[46, 46]
t01	[100, 100]	[100, 100]
t04	[100, 127]	$\top$
t05	[100, 100]	[13825, 13825]
t09	[10, 30]	[0, 0]

```
uint64_t foo(){
    int32_t var_18 = 0, var_14 = 0;
    while ((((var_18 - 0xa) & (var_18 ^ 0xfffffff5)) | (var_18
    ↪  & 0xfffffff5)) < 0)  {
        var_18 += 1;
        ...
    }

    int64_t res = ((((var_14 + var_18) * ....;
    _mopsa_print(res); // [0, 0]
    // It should have been [10, 22]
    return res;
}
```

E2-BN: Data obfuscation

- `EncodeData` and `EncodeArithmetic`
- In some programs, MOPSA returns an unsound solution
- `while (v < N)` $\Longrightarrow$ `while (((v - N) & (v ^ 0xfffffff5)) | (v & 0xfffffff5) < 0)`

file	E1-BN: original	E2-BN: data-obf
for_loop	[70, 70]	[46, 46]
t01	[100, 100]	[100, 100]
t04	[100, 127]	$\top$
t05	[100, 100]	[13825, 13825]
t09	[10, 30]	[0, 0]

```
uint64_t foo(){
    int32_t var_18 = 0, var_14 = 0;
    while ((((var_18 - 0xa) & (var_18 ^ 0xfffffff5)) | (var_18
    ↪ & 0xfffffff5)) < 0)  {
        var_18 += 1;
        ...
    }

    int64_t res = ((((var_14 + var_18) * ....;
    _mopsa_print(res); // [0, 0]
    // It should have been [10, 22]
    return res;
}
```

E2-BN: Data obfuscation

- `EncodeData` and `EncodeArithmetic`
- In some programs, MOPSA returns an unsound solution

file	E1-BN: original	E2-BN: data-obf
for_loop	[70, 70]	[46, 46]
t01	[100, 100]	[100, 100]
t04	[100, 127]	$\top$
t05	[100, 100]	[13825, 13825]
t09	[10, 30]	[0, 0]

- `while (v < N)` $\Longrightarrow$ `while (((v - N) & (v ^ 0xfffffff5)) | (v & 0xfffffff5) < 0)`
- MOPSA reports an underflow value for variable `var_18`

```
uint64_t foo(){
    int32_t var_18 = 0, var_14 = 0;
    while ((((var_18 - 0xa) & (var_18 ^ 0xfffffff5)) | (var_18
    ↪  & 0xfffffff5)) < 0)  {
        var_18 += 1;
        ...
    }

    int64_t res = ((((var_14 + var_18) * ....;
    _mopsa_print(res); // [0, 0]
    // It should have been [10, 22]
    return res;
}
```

E2-BN: Data obfuscation

- `EncodeData` and `EncodeArithmetic`
- In some programs, MOPSA returns an unsound solution

file	E1-BN: original	E2-BN: data-obf
for_loop	[70, 70]	[46, 46]
t01	[100, 100]	[100, 100]
t04	[100, 127]	$\top$
t05	[100, 100]	[13825, 13825]
t09	[10, 30]	[0, 0]

- `while (v < N)` $\Longrightarrow$ `while (((v - N) & (v ^ 0xfffffff5)) | (v & 0xfffffff5) < 0)`
- MOPSA reports an underflow value for variable `var_18`
- **Disrupts loop unrolling for MOPSA**

```
uint64_t foo(){
    int32_t var_18 = 0, var_14 = 0;
    while ((((var_18 - 0xa) & (var_18 ^ 0xfffffff5)) | (var_18
    ↪ & 0xfffffff5)) < 0)  {
        var_18 += 1;
        ...
    }

    int64_t res = ((((var_14 + var_18) * ....;
    _mopsa_print(res); // [0, 0]
    // It should have been [10, 22]
    return res;
}
```

E3: Control flow flattening

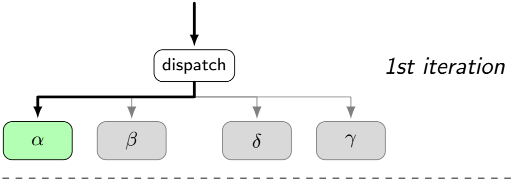
- TIGRESS provides 5 possible dispatchers
- only suitable option for our experiments: switch

E3: Control flow flattening

- Tigress provides 5 possible dispatchers
- only suitable option for our experiments: switch
- strongly impacted by the fixed number of loop unrolling iterations

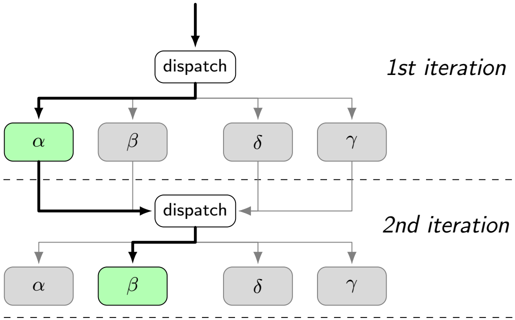
E3: Control flow flattening

- TIGRESS provides 5 possible dispatchers
- only suitable option for our experiments: switch
- strongly impacted by the fixed number of loop unrolling iterations



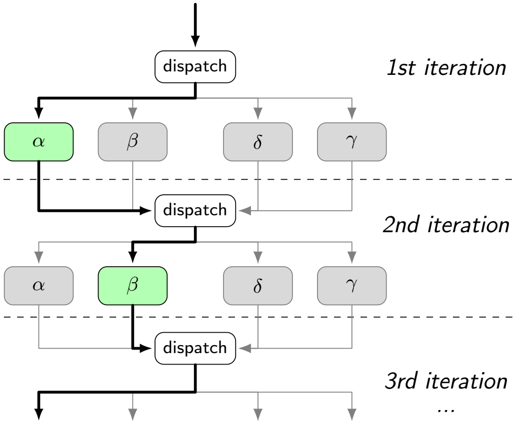
E3: Control flow flattening

- TIGRESS provides 5 possible dispatchers
- only suitable option for our experiments: switch
- strongly impacted by the fixed number of loop unrolling iterations



E3: Control flow flattening

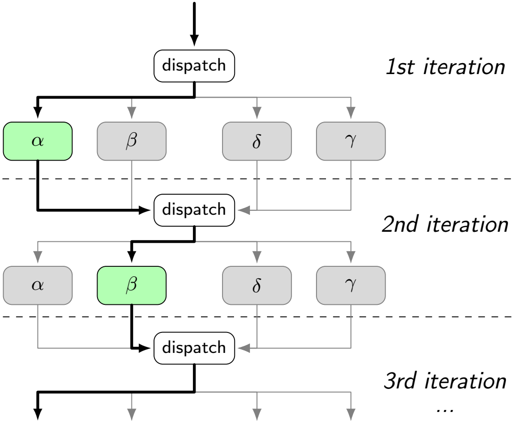
- TIGRESS provides 5 possible dispatchers
- only suitable option for our experiments: switch
- strongly impacted by the fixed number of loop unrolling iterations



E3: Control flow flattening

- TIGRESS provides 5 possible dispatchers
- only suitable option for our experiments: switch
- strongly impacted by the fixed number of loop unrolling iterations

file	E1: original	E3: flat
t01	[100, 100]	[100, 100]
t04	[100, 127]	$\top$
t06	[10, 10]	[10, 10]
t19	[202, 202]	[202, 202]



E3-BN: Control flow flattening

- switch translated as a jump table during compilation

E3-BN: Control flow flattening

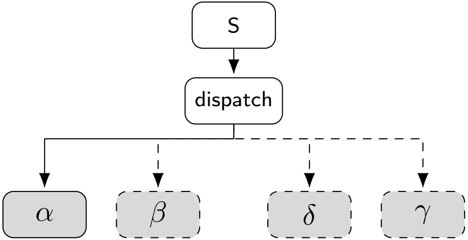
- switch translated as a jump table during compilation
- identifying boundaries of a jump table: difficult problem in binary analysis

E3-BN: Control flow flattening

- switch translated as a jump table during compilation
- identifying boundaries of a jump table: difficult problem in binary analysis
- disrupts control flow graph if jump table is not correctly read

E3-BN: Control flow flattening

- switch translated as a jump table during compilation
- identifying boundaries of a jump table: difficult problem in binary analysis
- disrupts control flow graph if jump table is not correctly read



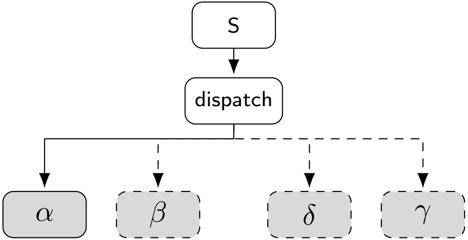

```
004011d0    void foo() __noreturn

004011d8        int32_t rdi
004011d8        int32_t var_2c = rdi
004011db        int64_t var_10 = 4
004011db
004011e8        while (true)
004011e8            if (var_10 u<= 0xc)
00401201                jump_table_402008[var_10]
00401213                var_10 = 0xb

00401220  83 45 ec 01 83 6d f0 01-48 c7 45 f8 02 00 00 00
00401230  eb 74 c7 45 ec 00 00 00-00 8b 45 dc 89 45 f0 48
00401240  c7 45 f8 02 00 00 00 eb-5d 81 7d dc 0f 27 00 00
00401250  7f 0a 48 c7 45 f8 03 00-00 00 eb 4a 48 c7 45 f8
00401260  00 00 00 00 eb 40
```

E3-BN: Control flow flattening

- switch translated as a jump table during compilation
- identifying boundaries of a jump table: difficult problem in binary analysis
- disrupts control flow graph if jump table is not correctly read




```
004011d0    void foo() __noreturn

004011d8        int32_t rdi
004011d8        int32_t var_2c = rdi
004011db        int64_t var_10 = 4
004011db
004011e8        while (true)
004011e8            if (var_10 u<= 0xc)
00401201                jump_table_402008[var_10]
00401213                var_10 = 0xb

00401220  83 45 ec 01 83 6d f0 01-48 c7 45 f8 02 00 00 00
00401230  eb 74 c7 45 ec 00 00 00-00 8b 45 dc 89 45 f0 48
00401240  c7 45 f8 02 00 00 00 eb-5d 81 7d dc 0f 27 00 00
00401250  7f 0a 48 c7 45 f8 03 00-00 00 eb 4a 48 c7 45 f8
00401260  00 00 00 00 eb 40
```

file	E1-BN: original	E3-BN: flat
t04	[100, 127]	FAIL
t05	[100, 100]	FAIL
t07	[100, 100]	FAIL
t09	[10, 30]	FAIL

E4: Opaque predicates

E4: Opaque predicates

List, array

```
p7 = malloc(sizeof(struct opaqueNode));
p7->data = VALUE;
p7->next = opaque_list_1->next;
p7->prev = opaque_list_1;
...
if (opaque_array[5] != opaque_array[2])
↪  ...
```

E4: Opaque predicates

List, array

```
p7 = malloc(sizeof(struct opaqueNode));
p7->data = VALUE;
p7->next = opaque_list_1->next;
p7->prev = opaque_list_1;
...
if (opaque_array[5] != opaque_array[2])
↪  ...
```

No loss of precision. MOPSA was able to recognize the live branch.

E4: Opaque predicates

List, array

```
p7 = malloc(sizeof(struct opaqueNode));
p7->data = VALUE;
p7->next = opaque_list_1->next;
p7->prev = opaque_list_1;
...
if (opaque_array[5] != opaque_array[2])
↪  ...
```

No loss of precision. MOPSA was able to recognize the live branch.

Input

e.g $input \in [0, 42]$

```
if (input - 42) ....
```

E4: Opaque predicates

List, array

```
p7 = malloc(sizeof(struct opaqueNode));
p7->data = VALUE;
p7->next = opaque_list_1->next;
p7->prev = opaque_list_1;
...
if (opaque_array[5] != opaque_array[2])
↪  ...
```

No loss of precision. MOPSA was able to recognize the live branch.

Input

e.g $input \in [0, 42]$

```
if (input - 42) ....
```

MOPSA was not able to recognize the live branch. Impacts depend on the content of dead branch.

E4: Opaque predicates

List, array

```
p7 = malloc(sizeof(struct opaqueNode));
p7->data = VALUE;
p7->next = opaque_list_1->next;
p7->prev = opaque_list_1;
...
if (opaque_array[5] != opaque_array[2])
↪  ...
```

No loss of precision. MOPSA was able to recognize the live branch.

Input

e.g *input* $\in [0, 42]$

```
if (input - 42) ....
```

MOPSA was not able to recognize the live branch. Impacts depend on the content of dead branch.

Env

```
unsigned long alwaysZero = 0;
int foo(){
    ...
    if (alwaysZero * (entropy_env)) ...
    ...
}
```

E4: Opaque predicates

List, array

```
p7 = malloc(sizeof(struct opaqueNode));
p7->data = VALUE;
p7->next = opaque_list_1->next;
p7->prev = opaque_list_1;
...
if (opaque_array[5] != opaque_array[2])
↪  ...
```

No loss of precision. MOPSA was able to recognize the live branch.

Input

e.g *input* $\in [0, 42]$

```
if (input - 42) ....
```

MOPSA was not able to recognize the live branch. Impacts depend on the content of dead branch.

Env

```
unsigned long alwaysZero = 0;
int foo(){
    ...
    if (alwaysZero * (entropy_env)) ...
    ...
}
```

No loss of precision. MOPSA was able to recognize the live branch.

E4: Opaque predicates / 2

Content of the dead branch includes:

- trivial instructions: semantically equivalent to `nop`
- buggy: add statements similar to live branch
- call: adds a call to a random function

E4: Opaque predicates / 2

Content of the dead branch includes:

- trivial instructions: semantically equivalent to `nop`
- buggy: add statements similar to live branch
- call: adds a call to a random function

Other TIGRESS options were discarded:

- `true`: encapsulates statement into an always true branch
- `junk`: adds data into the dead branch
- `fake`: call to an non existent function
- `question`: dead and live branch are the same

E4: Opaque predicates / 2

Content of the dead branch includes:

- trivial instructions: semantically equivalent to `nop`
- buggy: add statements similar to live branch
- call: adds a call to a random function

Other TIGRESS options were discarded:

- `true`: encapsulates statement into an always true branch
- `junk`: adds data into the dead branch
- `fake`: call to an non existent function
- `question`: dead and live branch are the same

file	**E1: original**	**E4 a: list_array**	**E4 b: input**	**E4 c: env**
for_loop	[70, 70]	[70, 70]	[10, 70]	[70, 70]
for_loop_3	[480, 480]	[480, 480]	[0, 480]	[480, 480]
if_then_else_level_1	[-2, -2]	[-2, 2]	[-2, 2]	[-2, -2]
if_then_else_simple	[4, 4]	[4, 4]	[4, 4]	[4, 4]

E4: Opaque predicates / 2

Content of the dead branch includes:

- trivial instructions: semantically equivalent to `nop`
- buggy: add statements similar to live branch
- call: adds a call to a random function

Other TIGRESS options were discarded:

- `true`: encapsulates statement into an always true branch
- `junk`: adds data into the dead branch
- `fake`: call to an non existent function
- `question`: dead and live branch are the same

file	E1: original	E4 a: list_array	E4 b: input	E4 c: env
for_loop	[70, 70]	[70, 70]	[10, 70]	[70, 70]
for_loop_3	[480, 480]	[480, 480]	[0, 480]	[480, 480]
if_then_else_level_1	[-2, -2]	[-2, 2]	[-2, 2]	[-2, -2]
if_then_else_simple	[4, 4]	[4, 4]	[4, 4]	[4, 4]

E4-BN: Opaque predicates

- During the export with BinaryNinja, two situations depending on whether or not initialize data structures from memory.

E4-BN: Opaque predicates

- During the export with BinaryNinja, two situations depending on whether or not initialize data structures from memory. Example using the `Env` option:

E4-BN: Opaque predicates

- During the export with BinaryNinja, two situations depending on whether or not initialize data structures from memory. Example using the Env option:

Global values initialized

```
// global symbol
unsigned long alwaysZero = 0;
...
if (alwaysZero * (entropy_env)) {
    live_branch
} else {
    dead_branch
}
```

E4-BN: Opaque predicates

- During the export with BinaryNinja, two situations depending on whether or not initialize data structures from memory. Example using the `Env` option:

Global values initialized

```
// global symbol
unsigned long alwaysZero = 0;
...
if (alwaysZero * (entropy_env)) {
    live_branch
} else {
    dead_branch
}
```

Global values NOT initialized

```
// global symbol
unsigned long alwaysZero;
...
if (alwaysZero * (entropy_env)) {
    live_branch
} else {
    dead_branch
}
```

E4-BN: Opaque predicates

- During the export with BinaryNinja, two situations depending on whether or not initialize data structures from memory. Example using the Env option:

Global values initialized

```c
// global symbol
unsigned long alwaysZero = 0;
...
if (alwaysZero * (entropy_env)) {
    live_branch
} else {
    dead_branch
}
```

Global values NOT initialized

```c
// global symbol
unsigned long alwaysZero;
...
if (alwaysZero * (entropy_env)) {
    live_branch
} else {
    dead_branch
}
```

	original	Opaque env (data init)	Opaque env (data not init)
t12	[112, 112]	[112, 112]	[102, 112]
t18	[12, 12]	[12, 12]	[2, 12]
t20	[4, 4]	[4, 4]	[4, 4]
t22	[7, 7]	[7, 7]	[0, 31]

E5: Opaque Constant

- Opaque predicates partially impacts MOPSA analysis

E5: Opaque Constant

- Opaque predicates partially impacts MOPSA analysis
- **Idea**: extend Tigress with opaque constant*

*Extracted from: "Information Hiding in Software with Mixed Boolean-Arithmetic Transforms" [Zhou, 2007]

E5: Opaque Constant

- Opaque predicates partially impacts MOPSA analysis
- **Idea**: extend Tigress with opaque constant*
- Obfuscation primitive for building opaque predicates

*Extracted from: "Information Hiding in Software with Mixed Boolean-Arithmetic Transforms" [Zhou, 2007]

E5: Opaque Constant

- Opaque predicates partially impacts MOPSA analysis
- **Idea**: extend Tigress with opaque constant*
- Obfuscation primitive for building opaque predicates

$$f(a,b) = 89 + (142 + 10 \cdot a + 246 \cdot b + 20 \cdot \neg(a \vee \neg b) + 246 \cdot (a \oplus b)) \cdot (160 + 224 \cdot a + 32 \cdot b + 192 \cdot \neg(a \vee \neg b) + 32 \cdot (a \oplus b)) + 170 \cdot a + 86 \cdot b + 84 \cdot \neg(a \vee \neg b) + 86 \cdot (a \oplus b)$$

$$\forall a, b \in \mathbb{Z}/2^8,\ f(a,b) = 25$$

E5: Opaque Constant

- Opaque predicates partially impacts MOPSA analysis
- **Idea**: extend Tigress with opaque constant*
- Obfuscation primitive for building opaque predicates

$$f(a,b) = 89 + (142 + 10 \cdot a + 246 \cdot b + 20 \cdot \neg(a \vee \neg b) + 246 \cdot (a \oplus b)) \cdot (160 + 224 \cdot a + 32 \cdot b + 192 \cdot \neg(a \vee \neg b) + 32 \cdot (a \oplus b)) + 170 \cdot a + 86 \cdot b + 84 \cdot \neg(a \vee \neg b) + 86 \cdot (a \oplus b)$$

$\forall a, b \in \mathbb{Z}/2^8,\ f(a,b) = 25$

We replaced a, b with random stack variables for thwarting evaluation.

*Extracted from: "Information Hiding in Software with Mixed Boolean-Arithmetic Transforms" [Zhou, 2007]

E5: Opaque Constant

- Opaque predicates partially impacts MOPSA analysis
- **Idea**: extend Tigress with opaque constant*
- Obfuscation primitive for building opaque predicates

file	E1-BN: original	E5-BN: opaque_const
t01	[100, 100]	[0, 100]
t02	[100, 100]	[0, 100]
t03	[10000, 32766]	$\top$
t04	[100, 127]	[-2147483648, 127]

$$f(a,b) = 89 + (142 + 10\cdot a + 246\cdot b + 20\cdot\neg(a\vee\neg b) + 246\cdot(a\oplus b))\cdot(160 + 224\cdot a + 32\cdot b + 192\cdot\neg(a\vee\neg b) + 32\cdot(a\oplus b)) + 170\cdot a + 86\cdot b + 84\cdot\neg(a\vee\neg b) + 86\cdot(a\oplus b)$$

$\forall a,b \in \mathbb{Z}/2^8,\ f(a,b) = 25$

We replaced a, b with random stack variables for thwarting evaluation.

Extracted from: "Information Hiding in Software with Mixed Boolean-Arithmetic Transforms" [Zhou, 2007]

E5-BN: Opaque Constant

- Analysis impacted by the content of the dead branch
  - Same effect of the source level
  - t04, t52 have more points added
  - Others, such as `if_then_else_level_1`, show little to no impact.

file	E1-BN: original	E5-BN: opaque_const
if_then_else_level_1	[2, 2]	[-2, 2]
t04	[100, 127]	[-2147483648, 127]
t05	[100, 100]	$\top$
t07	[100, 100]	[5, 100]
t09	[10, 30]	[0, 31]
t52	[0, 0]	[-2147483648, 0]

Takeaways

- Without obfuscation, compilation/decompilation introduces no confusion (E1, E1-BN)

Takeaways

- Without obfuscation, compilation/decompilation introduces no confusion (E1, E1-BN)
- Stronger obfuscation should leverage context beyond local scope (E2, E2-BN)

Takeaways

- Without obfuscation, compilation/decompilation introduces no confusion (E1, E1-BN)
- Stronger obfuscation should leverage context beyond local scope (E2, E2-BN)
- Injecting non-determinism or randomness in guards/loops increases complexity (E2)

Takeaways

- Without obfuscation, compilation/decompilation introduces no confusion (E1, E1-BN)
- Stronger obfuscation should leverage context beyond local scope (E2, E2-BN)
- Injecting non-determinism or randomness in guards/loops increases complexity (E2)
- Control flow flattening and opaque predicates may break loop unrolling by MOPSA analysis (E3)

Takeaways

- Without obfuscation, compilation/decompilation introduces no confusion (E1, E1-BN)
- Stronger obfuscation should leverage context beyond local scope (E2, E2-BN)
- Injecting non-determinism or randomness in guards/loops increases complexity (E2)
- Control flow flattening and opaque predicates may break loop unrolling by MOPSA analysis (E3)
- Hard static analysis problems (e.g., jump table boundaries) are effective obfuscation (E3-BN)

Takeaways

- Without obfuscation, compilation/decompilation introduces no confusion (E1, E1-BN)
- Stronger obfuscation should leverage context beyond local scope (E2, E2-BN)
- Injecting non-determinism or randomness in guards/loops increases complexity (E2)
- Control flow flattening and opaque predicates may break loop unrolling by MOPSA analysis (E3)
- Hard static analysis problems (e.g., jump table boundaries) are effective obfuscation (E3-BN)
- Non-trivial opaque predicates reduce precision due to misleading dead branches. The loss of precision depends on the dead branch (E4, E4-BN, E5, E5-BN)

Today:

- Evaluation of common transformations from TIGRESS against MOPSA static analyzer at source level and with BINARYNINJA at binary level

Today:

- Evaluation of common transformations from Tigress against mopsa static analyzer at source level and with BinaryNinja at binary level
- Obtained precise results at source level, less precise at binary level

Today:

- Evaluation of common transformations from TIGRESS against MOPSA static analyzer at source level and with BINARYNINJA at binary level
- Obtained precise results at source level, less precise at binary level
- Extended Tigress with a new kind of opaque predicate based on opaque constant

Today:

- Evaluation of common transformations from TIGRESS against MOPSA static analyzer at source level and with BINARYNINJA at binary level
- Obtained precise results at source level, less precise at binary level
- Extended Tigress with a new kind of opaque predicate based on opaque constant
- New insights for obfuscation transformations

Today:

- Evaluation of common transformations from TIGRESS against MOPSA static analyzer at source level and with BINARYNINJA at binary level
- Obtained precise results at source level, less precise at binary level
- Extended Tigress with a new kind of opaque predicate based on opaque constant
- New insights for obfuscation transformations

Thanks for your attention!

Questions? Join the live Q&A.

Further information

PDF available at

https://dl.acm.org/doi/10.1145/3733817.3762702

```
@inproceedings{altamura2025assessing,
  title={Assessing the Effectiveness of the Tigress Obfuscator
  Against MOPSA and BinaryNinja},
  author={Altamura, Nicol{\`o} and Bragastini, Enrico and Campion,
  Marco and Dalla Preda, Mila},
  booktitle={Proceedings of the 2025 Workshop on Research on
  Offensive and Defensive Techniques in the Context of Man At The
  End (MATE) Attacks},
  pages={29--37},
  year={2025}
}
```