Generalized Dynamic Opaque Predicates: A New Control Flow Obfuscation Method

Authors: Dongpeng Xu, Jiang Ming, Dinghao Wu

Presenters: Gabe Garfinkel, Marshall Rhea, Michael Wolf

Background: Obfuscation

- Impede attempts to:
  - Reverse engineer
  - Find/exploit vulnerabilities
  - Recognize
- Used on networks to impede attackers
- Malware uses it too

“Opaque Predicates”

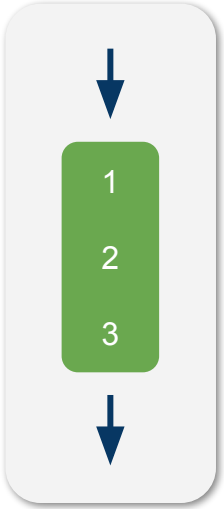
- An expression that is always true or false
- Known to compiler, tough to detect at runtime
- Can construct branches that always choose one way
  - But “look like” they are doing a real comparison

$$\forall x \in Z.\ (4x^2 + 4) \bmod 19 \not\equiv 0$$

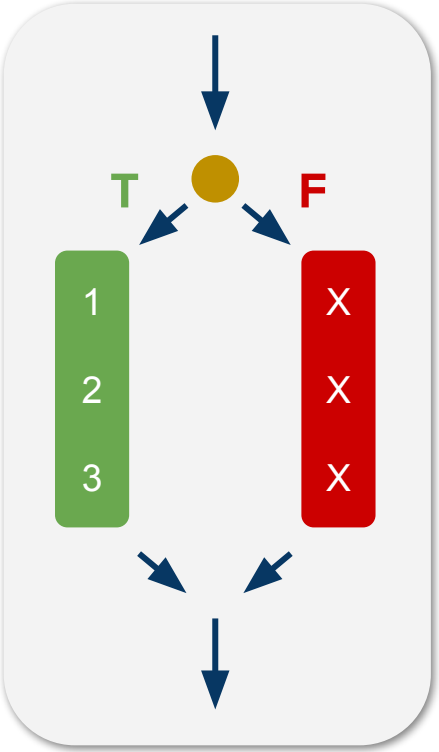
“Opaque Predicates”

- Use to insert superfluous branches

Original Code



Opaque Predicate

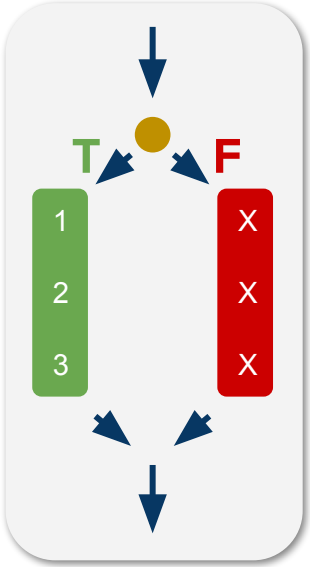


Correlated Predicates/Variables

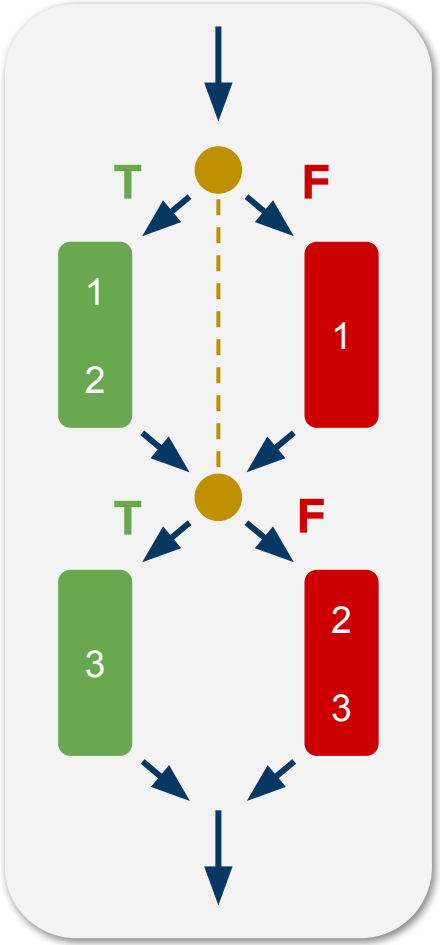
- **Correlated Predicates** - A set of predicates that always evaluate to the same value in a given program execution.
  - Check DU chain to ensure variables are not re-assigned between predicates
  - Based on **correlated variables**, which always have the same value

CV	CP_1	CP_2	CP_3
x	`x > 0`	`x%2 == 1`	`x+x > 0`
y	`y > 0`	`y%2 == 0`	`2*y <= 0`
z	`z <= 0`	`z%2 == 1`	`z<<1 > 0`

“Dynamic Opaque Predicates”



Opaque Predicate



Dynamic Opaque

“Dynamic Opaque Predicates”

- Problems:
  - Detectable using symbolic execution
  - A program called LOOP exploits this

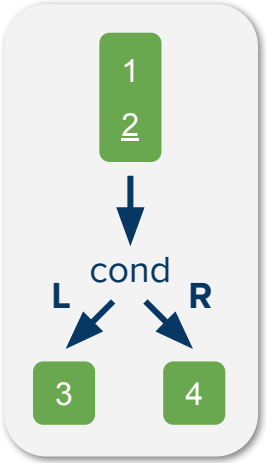
“Generalized Dynamic Opaque Predicates”

- Use dynamic opaque predicates across branch and loop
- More resistant to program analysis tools
  - But in reality, it’s just an “arms race”
- Additionally, do multiple passes

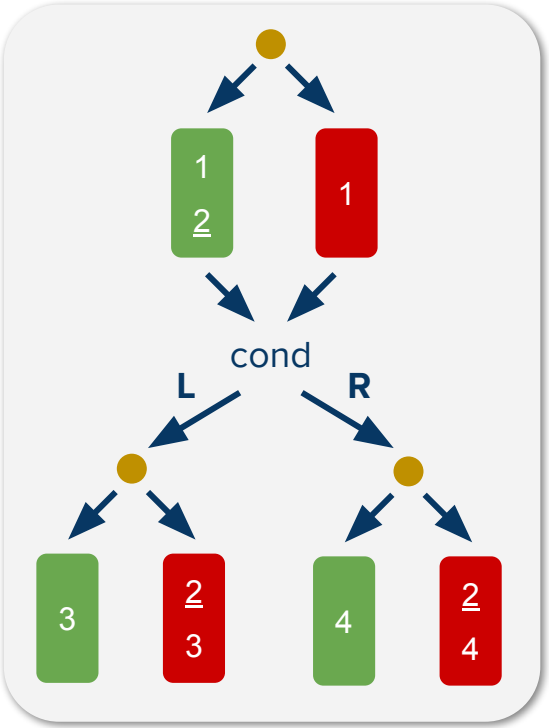
Branches

- Move an invariant instruction across a branch
  - Execution before/after branch depends on predicate

Original Code



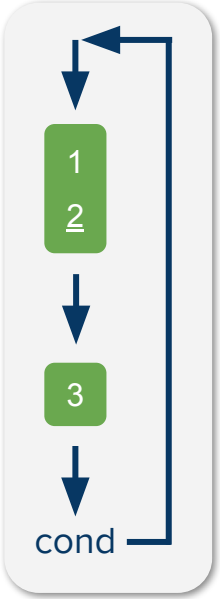
Across Branch



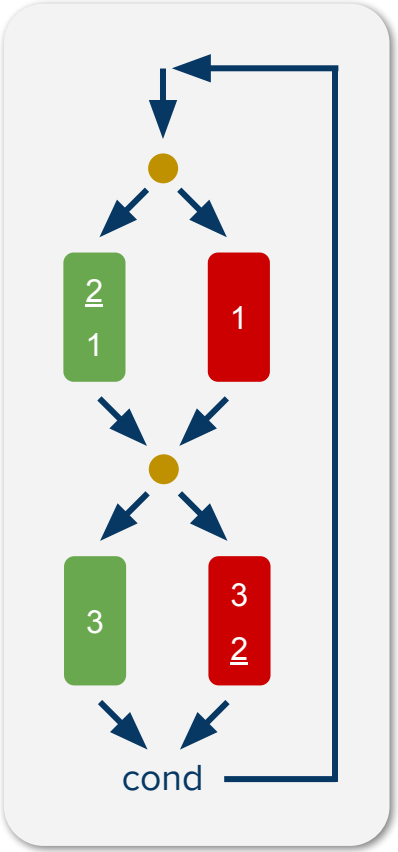
Loop

- Move a loop-invariant to be first
  - Then, in right branch, move it across backedge

Original Code



Across Loop



Results

- Ran obfuscator on five “hot” functions of GNU Coreutils 8.23
  - More basic blocks, CFG edges, and higher cyclomatic number
- Opaque predicate algorithm (LOOP) performed very poorly
- ➡ Great performance against current standards

Table 2: Obfuscation metrics and BinDiff scores of hot functions in Coreutils.

Function	# of Basic Blocks			# of CFG Edges			Cyclomatic Number			Bindiff Score	
	Orig.	50%	100%	Orig.	50%	100%	Orig.	50%	100%	50%	100%
1	43	171	229	62	258	338	21	89	111	0.05	0.02
2	20	75	105	30	114	158	12	41	55	0.02	0.01
3	30	94	120	49	141	177	21	49	59	0.02	0.02
4	46	138	208	80	220	320	36	84	114	0.04	0.01
5	76	272	376	117	425	573	43	155	199	0.05	0.02

Table 3: The result of LOOP detection.

Function	Straight Line DOP			Branch DOP			Loop DOP		
	Total	Detected	Ratio	Total	Detected	Ratio	Total	Detected	Ratio
1	52	3	5.77%	21	0	0.00%	8	0	0.00%
2	28	2	7.14%	15	0	0.00%	6	0	0.00%
3	27	2	7.41%	23	0	0.00%	6	0	0.00%
4	54	5	9.26%	26	0	0.00%	8	0	0.00%
5	82	8	9.76%	52	0	0.00%	14	0	0.00%

Cost

- Obfuscation caused marginal increases in runtime, file size
- ➡ Predicates do not appreciably change programs

Table 4: Cost evaluation of the dynamic opaque predicate obfuscation.

Function	Program	Binary Size (Bytes)				Execution Time (ms)		
		Orig.	50%	100%	Ratio	Orig.	50%	100%
1	`tr`	132,084	132,826	133,491	0.53%	2.2	2.2	2.4
2	`stat`	210,864	211,355	211,710	0.20%	4.0	4.0	4.1
3	`ls`	350,076	350,916	351,527	0.21%	23.2	23.4	23.7
4	`ls`	350,076	351,083	351,742	0.24%	23.2	23.3	23.8
5	`expr`	129,696	130,836	131,409	0.66%	0.6	0.6	0.6

Strengths

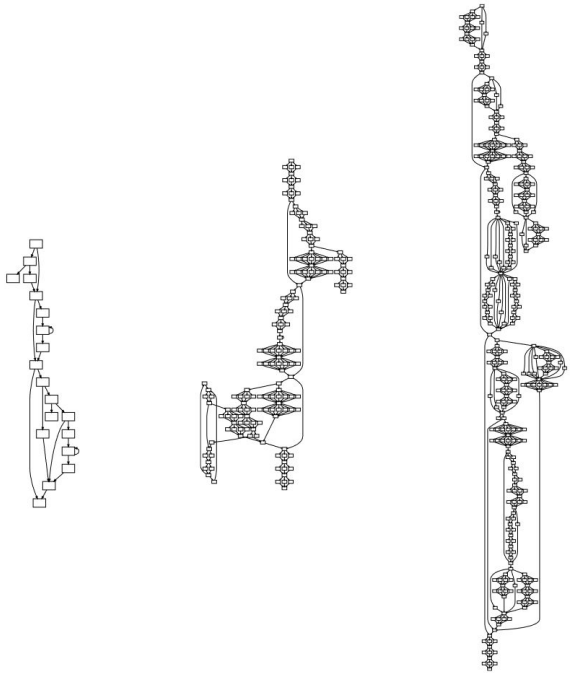
- Creates a novel obfuscation with no known detection method
- Code base provided
- Simple yet effective methods

Weaknesses

- Lack of consideration of future de-obfuscation techniques
  - E.g., always using correlated predicates, not considering loop or branch recognition methods
- Only used on one codebase
- Did not completely eliminate detection of Straight-Line DOPs

Conclusion

- Generalized Dynamic Opaque Predicates allow for efficient and efficient obfuscation
- There exists no current algorithm to detect these predicates effectively, especially in loops and branches



(a) The original CFG. (b) The CFG after one round of obfuscation. (c) The CFG after two rounds of obfuscation.

Fig. 7: Comparison between CFGs after different rounds of dynamic opaque predicate obfuscation.

Thank you!

Old Slides

“Dynamic Opaque Predicates”

Original Code

Dynamic Opaque