香港中文大學
The Chinese University of Hong Kong

Ph.D. Oral Defense

Software Obfuscation with Layered Security

Ph.D. Candidate:	Hui Xu
Thesis Supervisor:	Michael R. Lyu
Thesis Committee:	Qiang Xu, Patrick Lee, Jiannong Cao

Sep 21st, 2018

The Problem of Software IP Protection

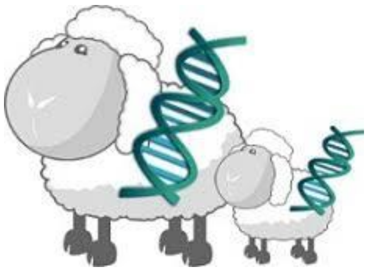
- **Software intellectual property:**
  - Server side (secure)
  - Client side (vulnerable)

- **MATE (Man-At-The-End) attack** [Collberg'11]: reverse engineer

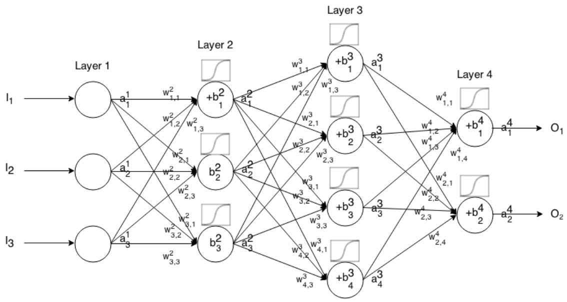
- **Examples of MATE attacks:**

```
if (verifyLicense (key))
    startProgram();
else{
    printf ("invalid key");
    exit(-1);
}
```

Disable License Checking



Clone Codes



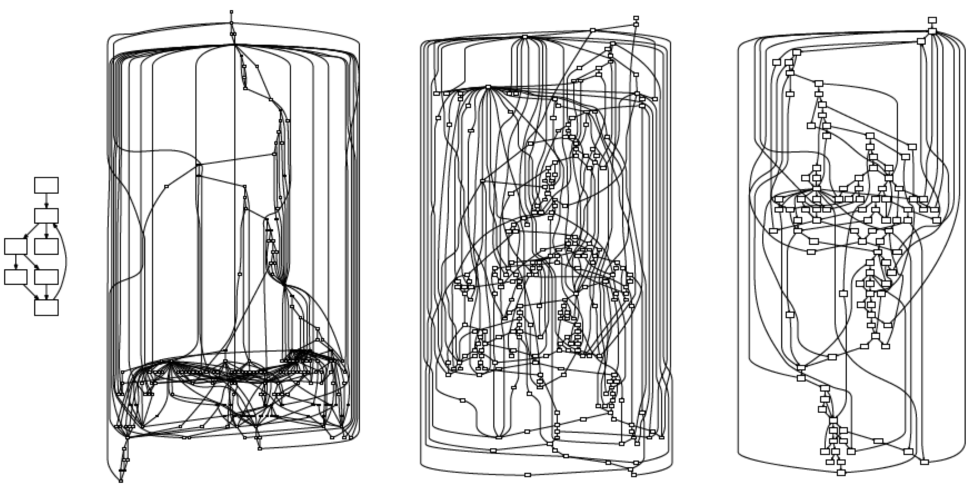
Steal Algorithms

[Collberg'11] C. Collberg, *et al*. Toward digital asset protection. IEEE Intelligent Systems, 2011.

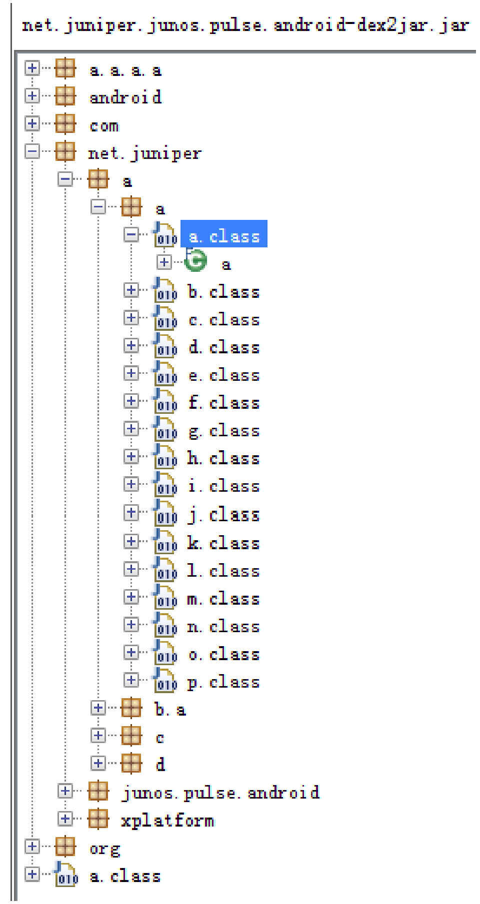
Software Obfuscation for IP Protection

❑Transform codes to a new version:

- Difficult to read.
- Preserve the semantics.
- Incur little overhead.



Examples of control-flow obfuscation [Yadegari'15]

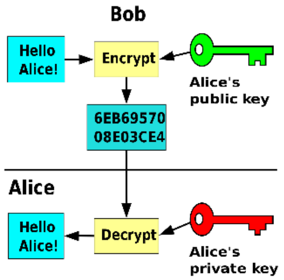


Example of lexical obfuscation

[Yadegari'15] B. Yadegari, *et al.* A generic approach to automatic deobfuscation of executable code. IEEE S&P, 2015.

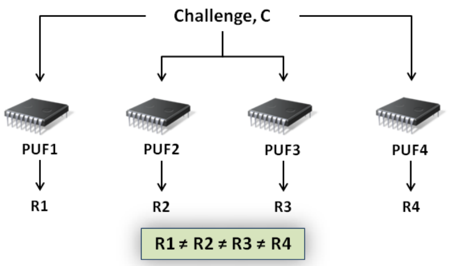
Critical Challenges for Obfuscation

- **Problem:** Obfuscation is not as secure as other security primitives.



Public-key encryption

Fingerprint



Physical unclonable functions

- **Questions:**
  - What is the best security capability of software obfuscation?
  - How to design reliable obfuscation solutions?

Survey Results: Theoretical Area

- [Barak'01]: Negative result for black-box obfuscation.
- [Garg'13]: Positive result for indistinguishability obfuscation based on *graded encoding* (noisy multi-linear maps).
- Problems of graded encoding:
  - Only applicable to circuits: pure arithmetic.
  - Inefficient: polynomial overhead (several Gigabytes to obfuscate a 16-bit point function [Apon'14]).

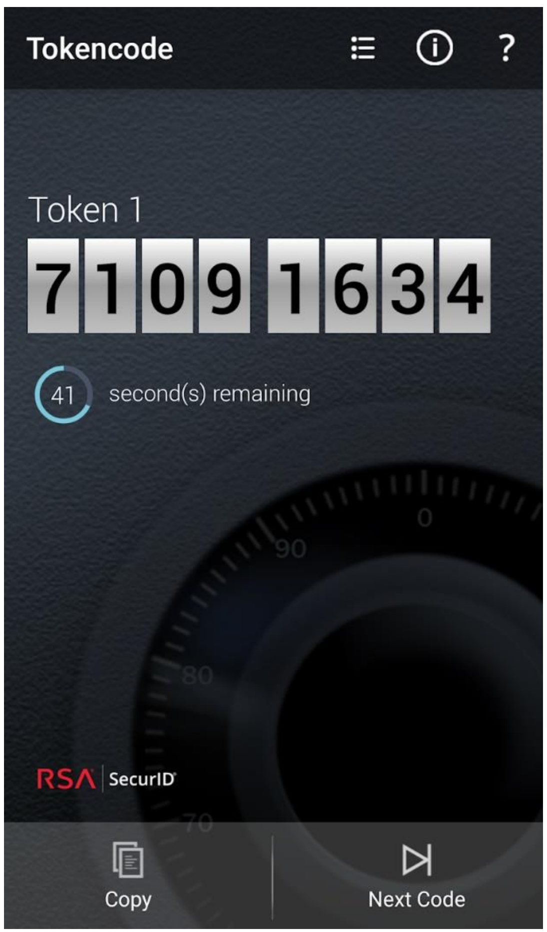
[Barak'01] B. Barak, *et al*. On the (im) possibility of obfuscating programs. CRYPTO. 2001.
[Garg'13] S. Garg, *et al*. Candidate indistinguishability obfuscation and functional encryption for all circuits. FOCS. 2013.
[Apon'14] D. Apon, *et al*. Implementing cryptographic program obfuscation. IACR Cryptology ePrint Archive, 2014.

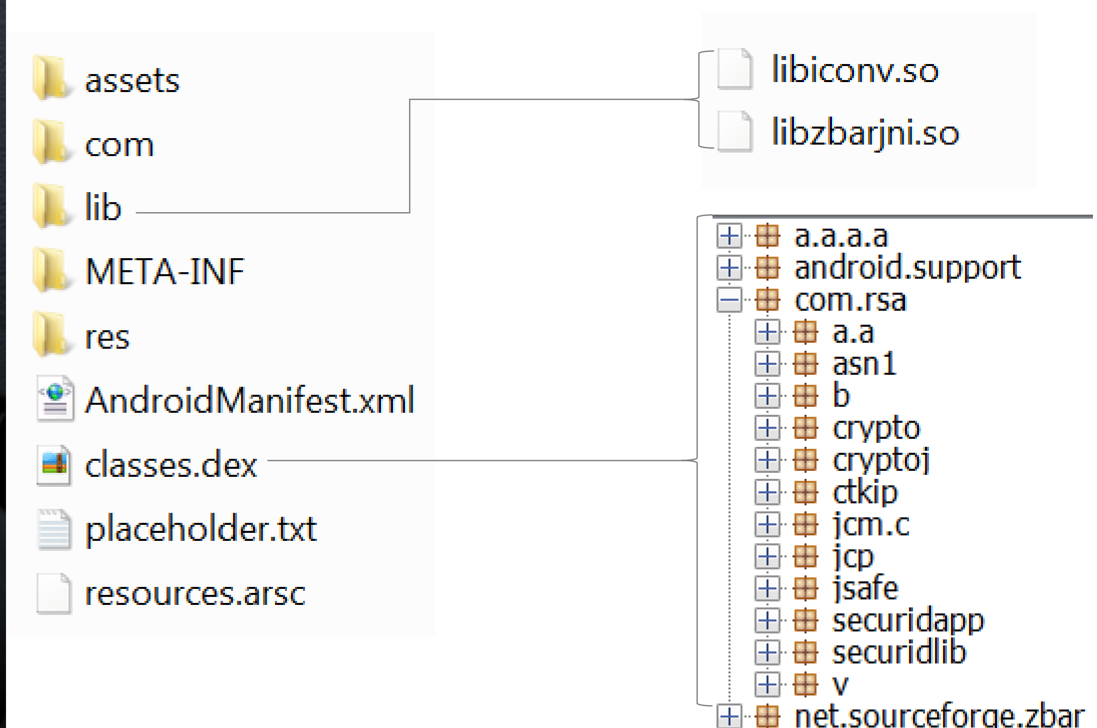
Survey Results: Practical Area

- Most papers assume software written in particular languages, *e.g.,* Java/C/Assembly.
- But real-world software is more complicated with heterogeneous components.
- Two types of software applications:
  - Client-server mode (*e.g., Android applications*).
  - Browser-server mode, *i.e.,* web applications.

Example of Android Apps

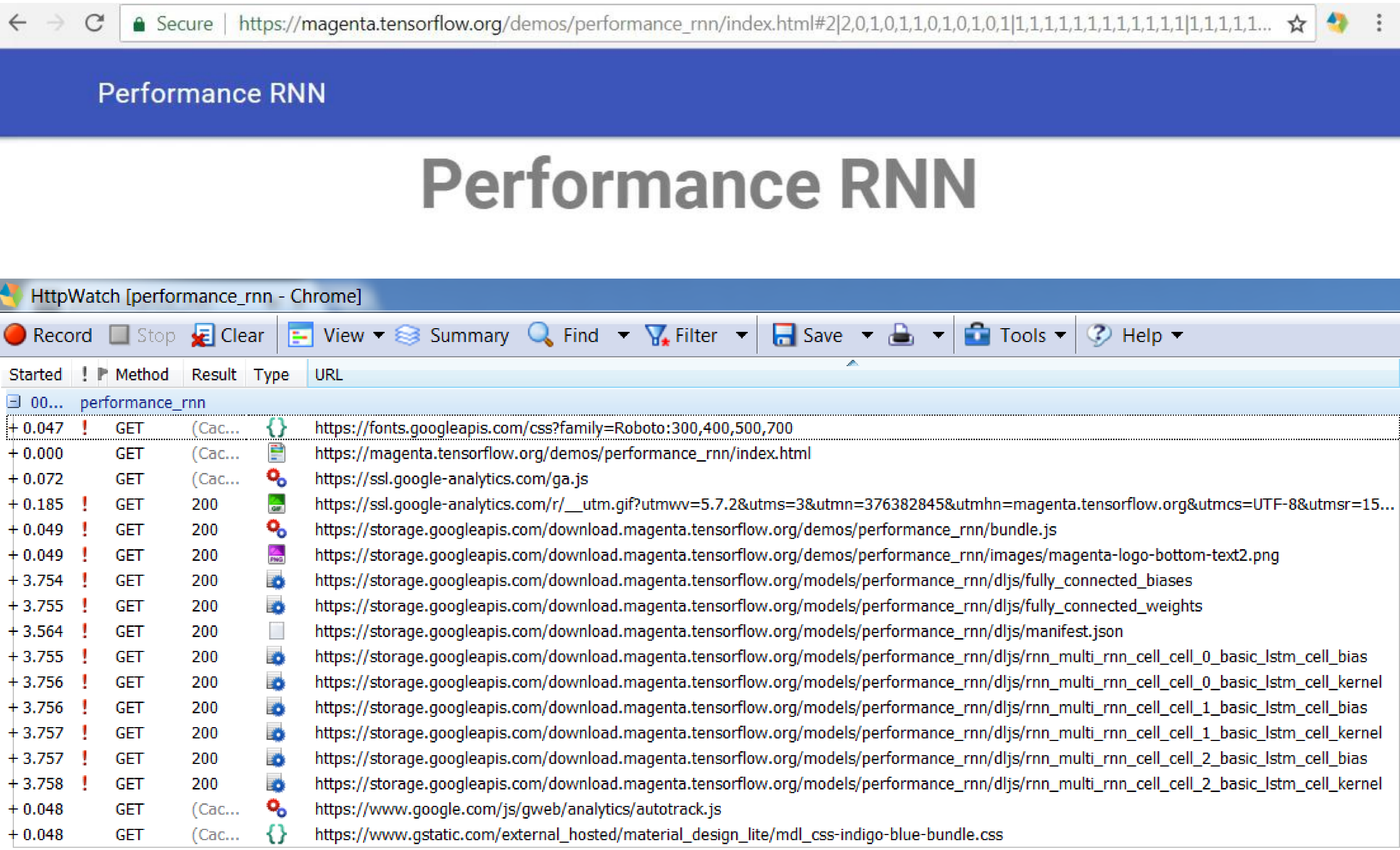
RSA SecureID Software Token





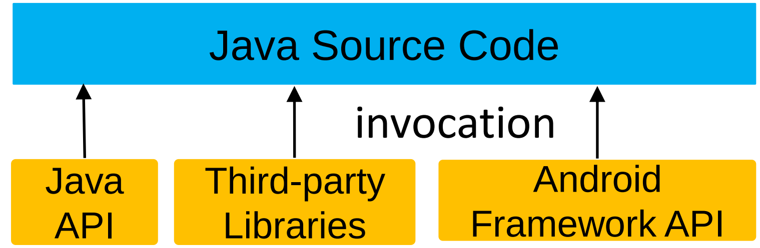
Components

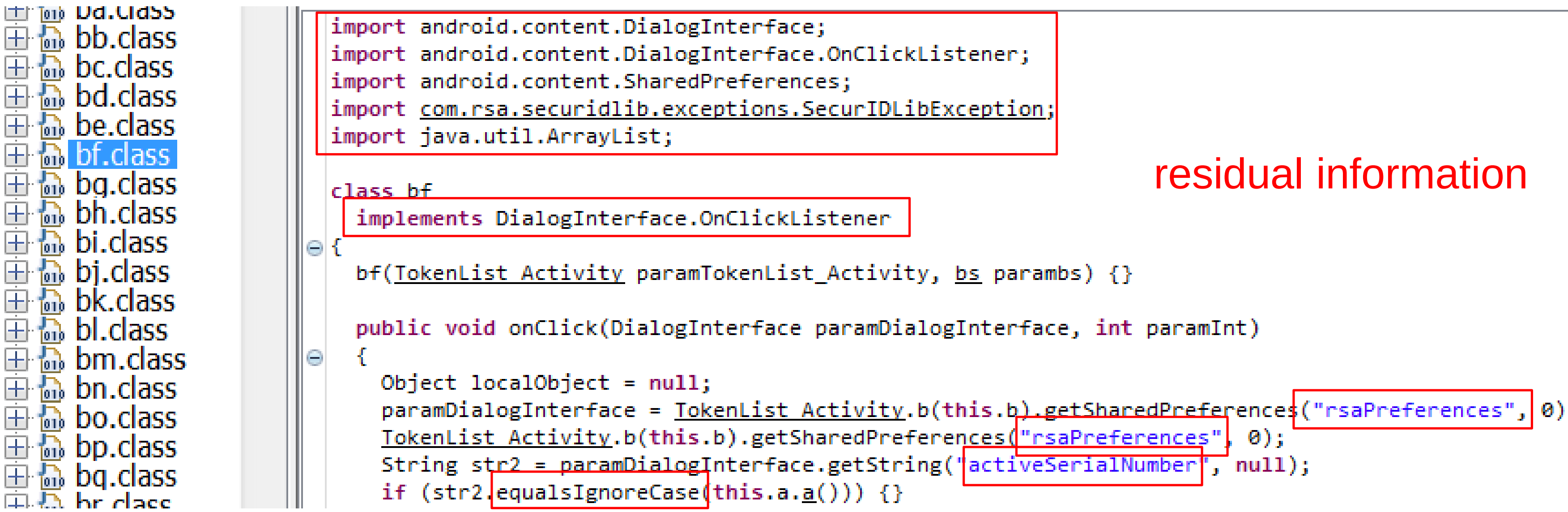
Example of Web Applications



Started	!	Method	Result	Type	URL
00...		performance_rnn			
+ 0.047	!	GET	(Cac...		https://fonts.googleapis.com/css?family=Roboto:300,400,500,700
+ 0.000		GET	(Cac...		https://magenta.tensorflow.org/demos/performance_rnn/index.html
+ 0.072		GET	(Cac...		https://ssl.google-analytics.com/ga.js
+ 0.185	!	GET	200		https://ssl.google-analytics.com/r/__utm.gif?utmwv=5.7.2&utms=3&utmn=376382845&utmhn=magenta.tensorflow.org&utmcs=UTF-8&utmsr=15...
+ 0.049	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/demos/performance_rnn/bundle.js
+ 0.049	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/demos/performance_rnn/images/magenta-logo-bottom-text2.png
+ 3.754	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/fully_connected_biases
+ 3.755	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/fully_connected_weights
+ 3.564	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/manifest.json
+ 3.755	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/rnn_multi_rnn_cell_cell_0_basic_lstm_cell_bias
+ 3.756	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/rnn_multi_rnn_cell_cell_0_basic_lstm_cell_kernel
+ 3.756	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/rnn_multi_rnn_cell_cell_1_basic_lstm_cell_bias
+ 3.757	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/rnn_multi_rnn_cell_cell_1_basic_lstm_cell_kernel
+ 3.757	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/rnn_multi_rnn_cell_cell_2_basic_lstm_cell_bias
+ 3.758	!	GET	200		https://storage.googleapis.com/download.magenta.tensorflow.org/models/performance_rnn/dljs/rnn_multi_rnn_cell_cell_2_basic_lstm_cell_kernel
+ 0.048		GET	(Cac...		https://www.google.com/js/gweb/analytics/autotrack.js
+ 0.048		GET	(Cac...		https://www.gstatic.com/external_hosted/material_design_lite/mdl_css-indigo-blue-bundle.css

Why Hard for Protection?

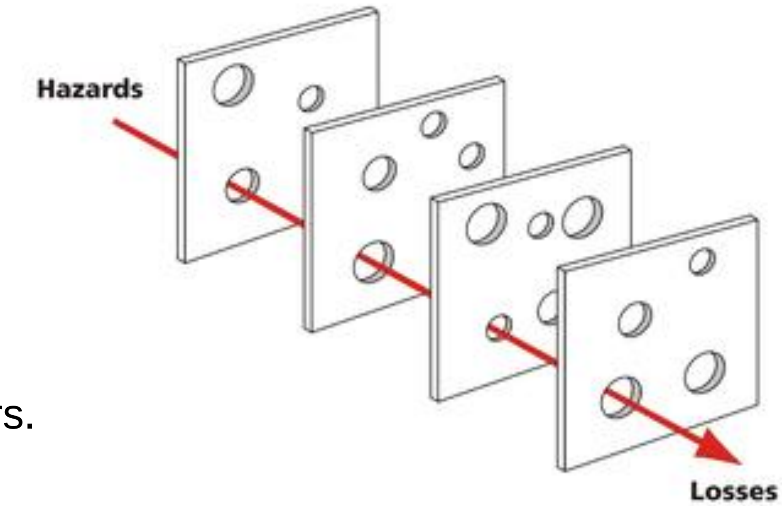




[Bichsel'16]: We can recover a large portion of lexical information based on the residual information, *e.g.*, names of invoked methods and strings.

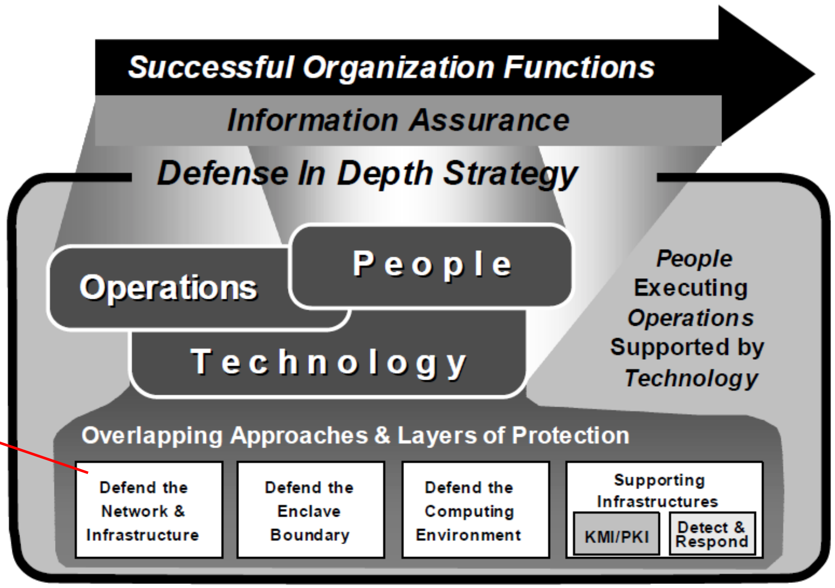
[Bichsel'16] B. Bichsel, et al. Statistical deobfuscation of android applications, CCS, 2016.

Layered Security



- Principle: Swiss cheese model:
  - Mitigate risks through different layers.
  - Avoid single point of failure.
- Employed in aviation safety, healthcare, *etc*.
  - Safety-critical or security-critical.
  - The risks cannot be fully avoided.
- Introduced in IATF3.1.

Each area has multiple layers of protections



[IATF'02] Information Assurance Technical Framework Release 3.1, Department of Defense, 2002

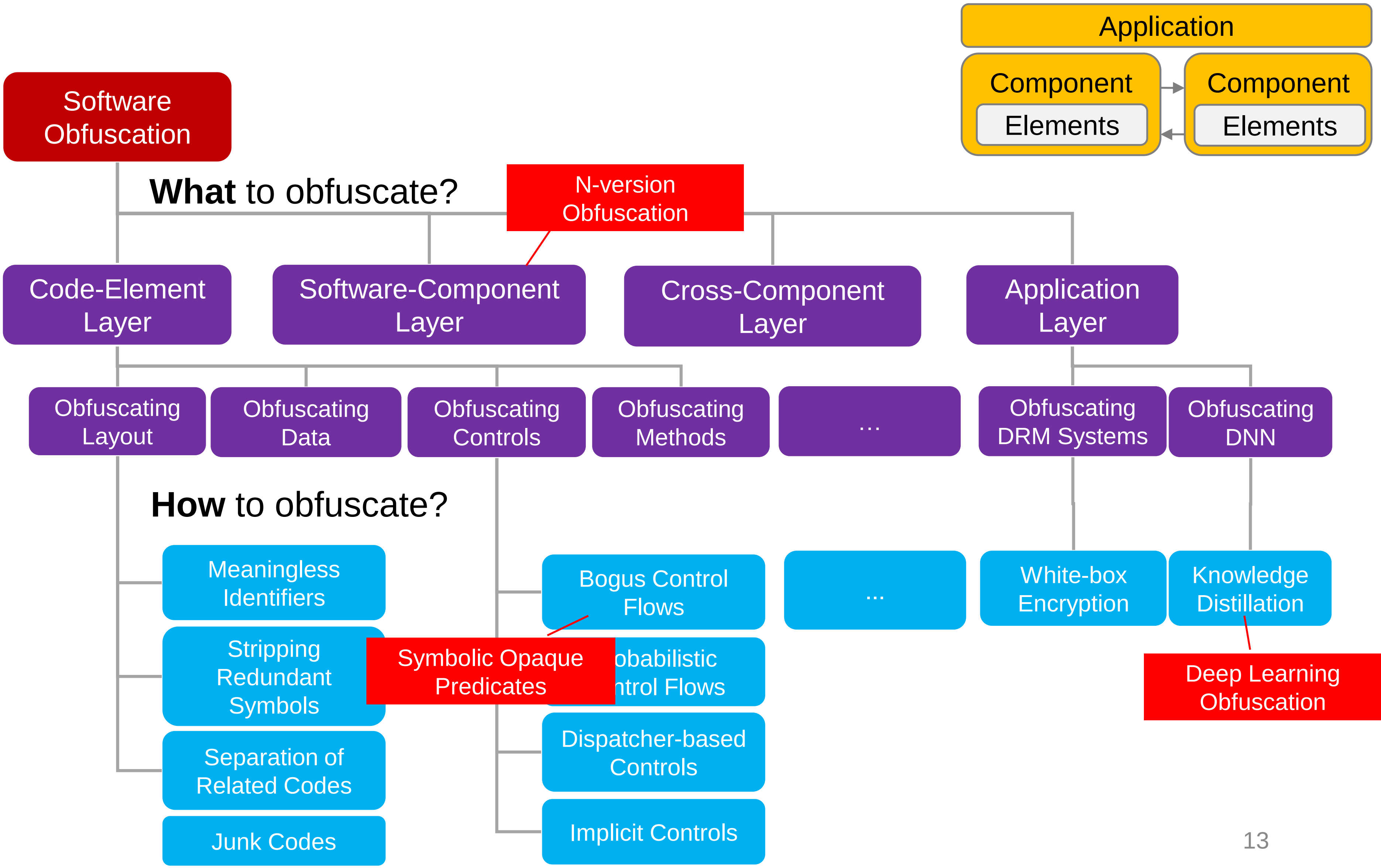
Layered Security for Software Obfuscation

- Why?
  - Software is very complicated.
  - Secure-against-all obfuscation techniques do not exist.
- How?
  - Based on risk management.
  - Integrate multiple obfuscation techniques to mitigate risks.
  - Each obfuscation technique only corresponds to particular threats.

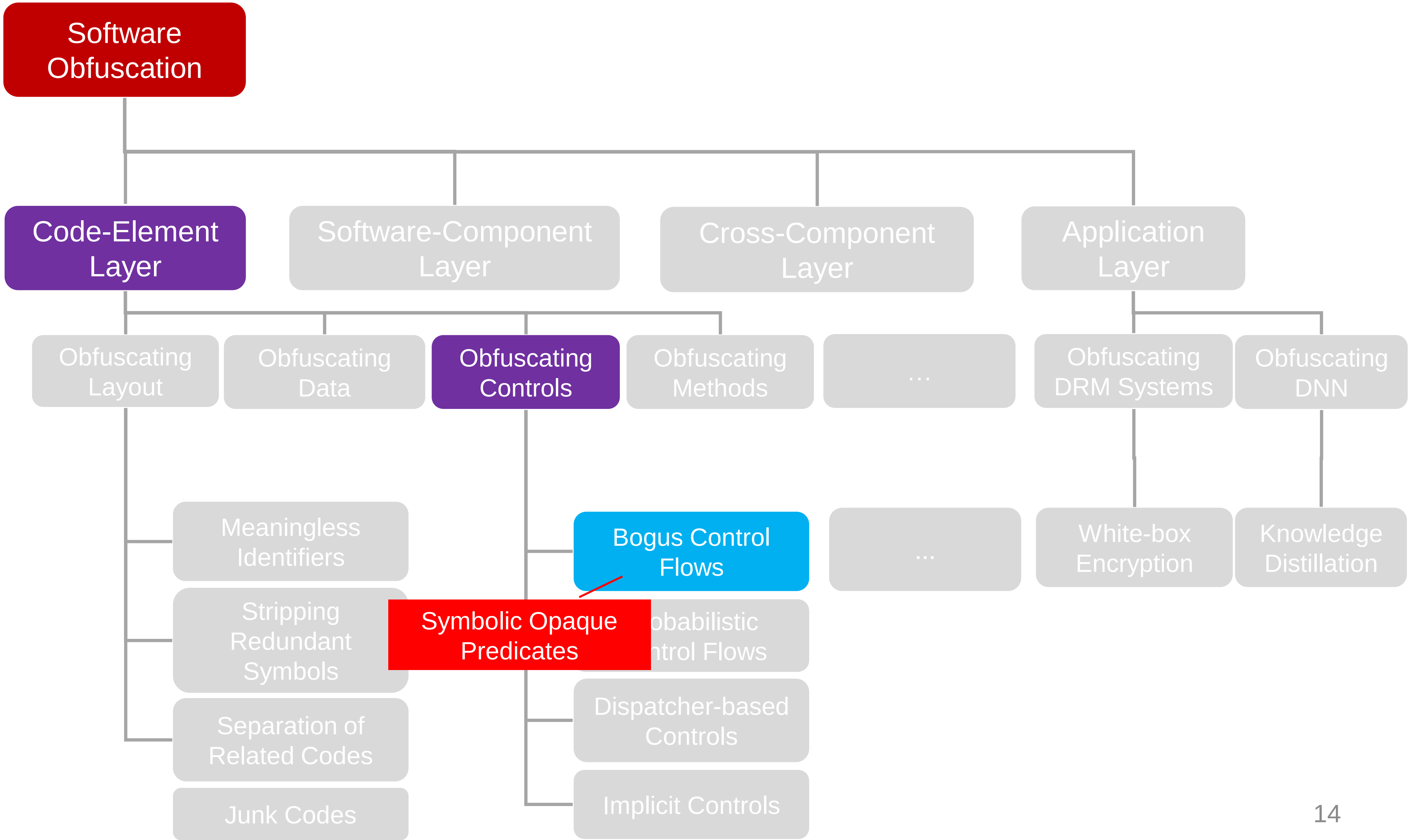
Thesis Contributions

- **Develop a taxonomy** of obfuscation for layered security.
    - Assist developers in designing layered obfuscation solutions.
- **Enrich the taxonomy** with three novel obfuscation techniques.
    - ***Symbolic Opaque Predicates***
        - Enhance the security of control-flow obfuscation.
    - ***N-version Obfuscation***
        - Enable the software with resilience to large-scale tampering attacks.
    - ***Deep Learning Obfuscation***
        - Application-level obfuscation technique for deep learning software.

Taxonomy for Layered Obfuscation

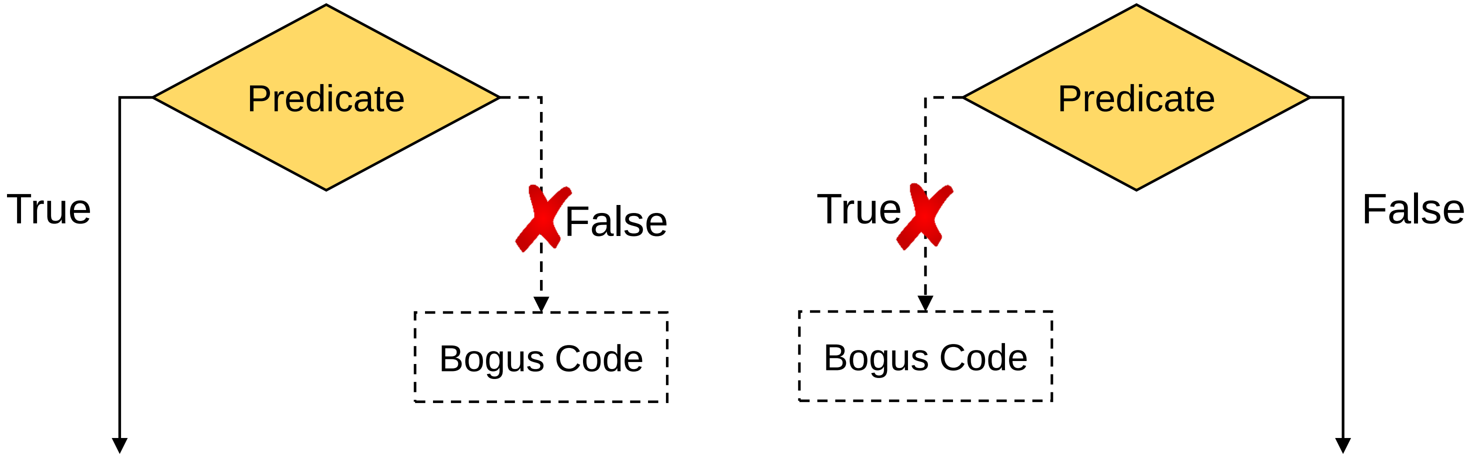


Symbolic Opaque Predicates



Opaque Predicate

- **Definition** [Collberg'97] :
  - The value is known before compilation time.
  - Reverse analysis is difficult.
- **Application**: Control-flow obfuscation.



[Collberg'97] Collberg, *et al*. "A taxonomy of obfuscating transformations." Department of Computer Science, 1997.

Vulnerable Example 1

❑ **Problem:** Real-world opaque predicates are vulnerable.

```
int a, b; //b>0
...
func() {
  ...
  if (a > b) {
    if (a*(a+1)%2 == 0)
      fp = A[((fp)()%2)+2];
    else
      fp = A[((fp)()%2)+4];
    ...
  }
  ...
  if ((b-2)*(b-1)*b%6 != 0)
    fp = A[((fp)()%2)+5];
  else
    fp = A[((fp)()%2)+3];
  ...
}
```

Constantly True

Constantly False

Example in [Ogiso'03]

Vulnerable to automated program analysis tools.

[Ogiso'03] Ogiso, *et al*. "Software obfuscation on a theoretical basis and its implementation." 2003

Vulnerable Example 2

Default opaque predicate generated by Obfuscator-LLVM [Junod’15]

LLVM IR Code:

```
@x7 = common global i32 0
@y8 = common global i32 0
...
define i32 @main() #0 {
  %1 = load i32* @x7
  %2 = load i32* @y8
  %3 = sub i32 %1, 1
  %4 = mul i32 %1, %3
  %5 = urem i32 %4, 2
  %6 = icmp eq i32 %5, 0
  %7 = icmp slt i32 %2, 10
  %8 = or i1 %6, %7
  br i1 %8, label %originalBB, label %originalBBalteredBB
```

Source Code:

```
int x7 = 0;
int y8 = 0;
```

Constantly True

```
if(x7(x7 – 1)%2 == 0||y8<10){
   originalBB;
} else {
   originalBBalteredBB;
}
```

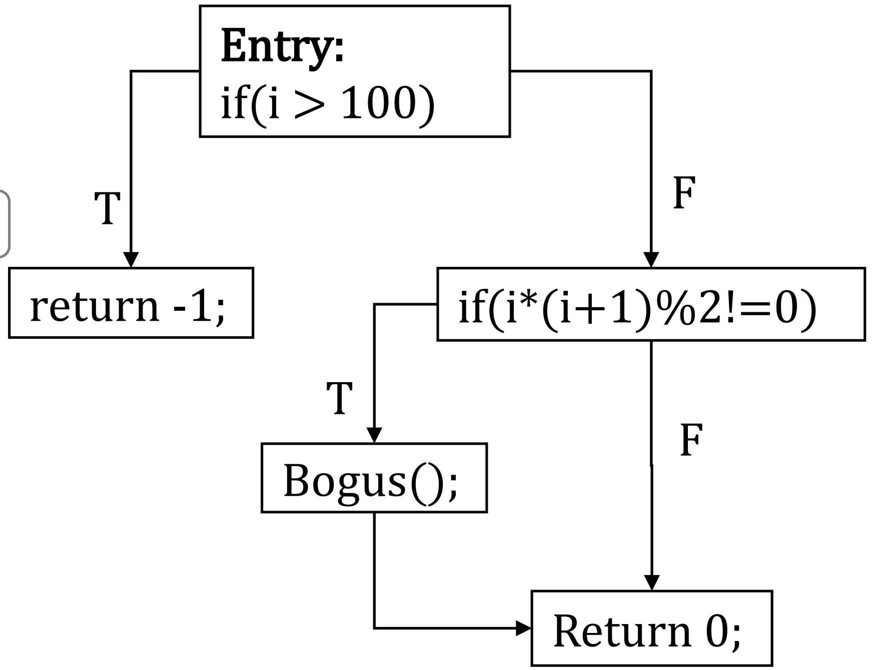
[ollvm’15] Junod, *et al*. “Obfuscator-LLVM--software protection for the masses.” 2015

Adversarial Symbolic Execution

- Symbolic execution can detect opaque predicates by traversing the control-flow graph.

```
int toy_func(int i){
  if(i > 100){
    return -1;
  }
  if(i *(i+1)%2!=0) {
    Bogus();
  }
  return 0;
}
```

opaque predicate

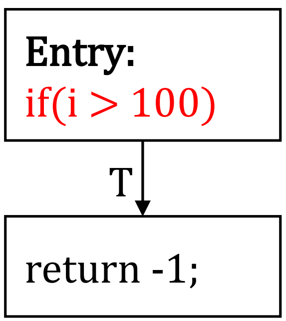


Source Code

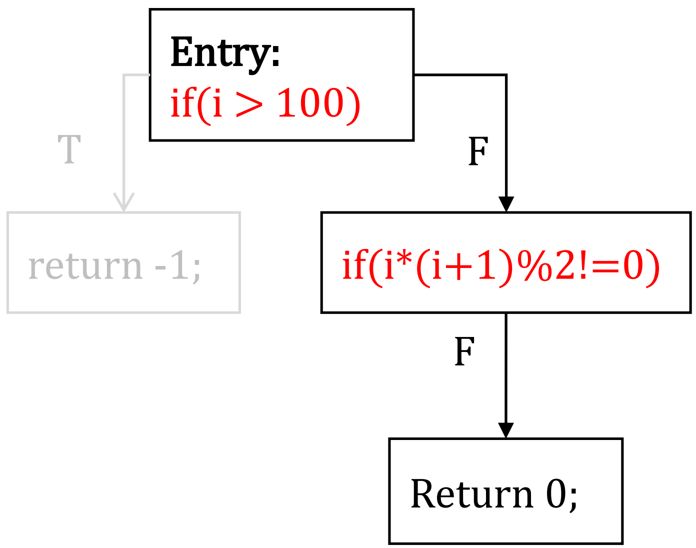
Control-flow Graph

Symbolic Execution Steps

- **Round1**:
Random Input:
i = 1000



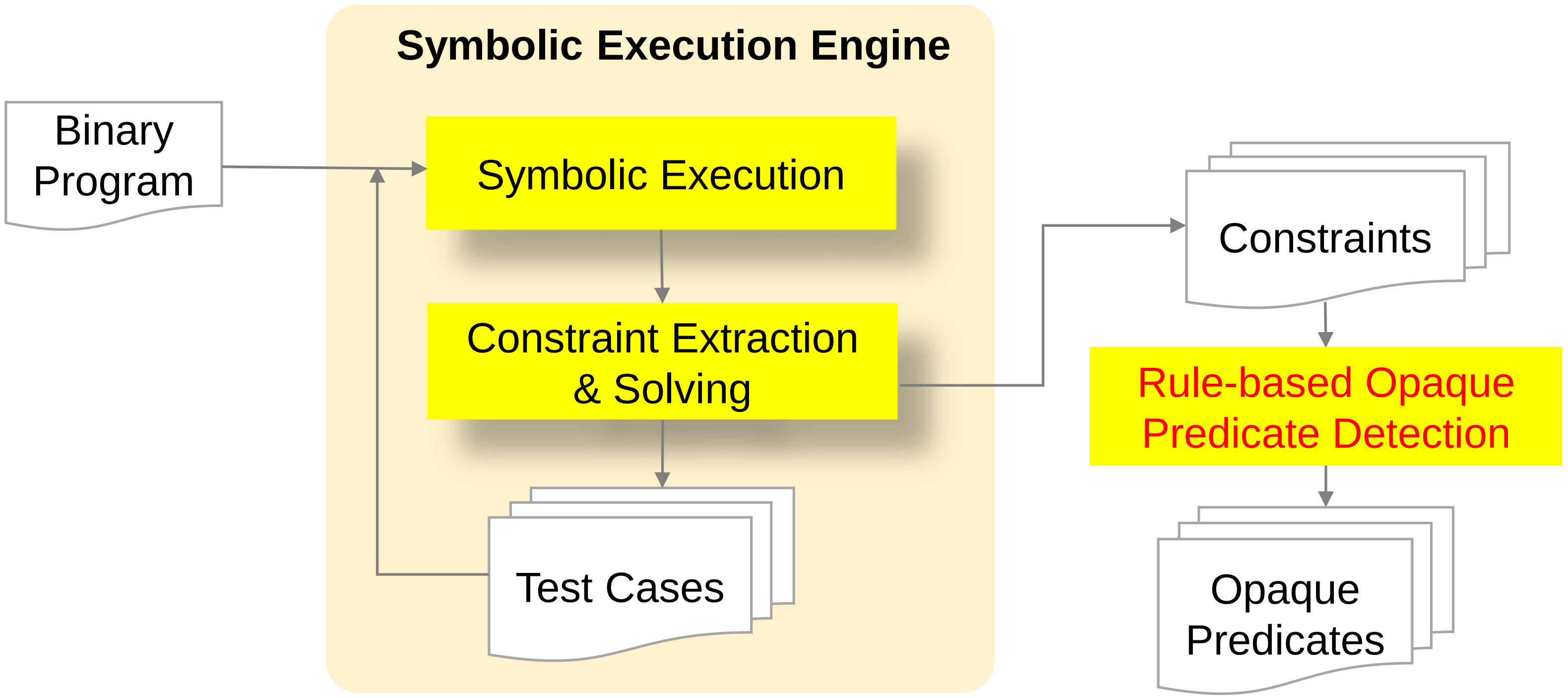
- **Round 2:**
Constraint: *NOT (i > 100)*
=> i = 0



- **Round 3**:
Constraint: NOT(i > 100)
AND i(i+1)%2!=0*
=> No Solution

It implies opaque predicates and bogus codes.

Adversary Model



The constraints are in the form of CNF: $\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n$

[Ming'15] Ming, *et al.* "Loop: Logic-oriented opaque predicate detection in obfuscated binary code." CCS'15.
[Yadegari'15] Yadegari, *et al.* "Symbolic execution of obfuscated code." CCS'15.
[Yadegari'15] Yadegari, *et al.* "A generic approach to automatic deobfuscation of executable code." S&P' 15.
[Xu'17] Xu, *et al.* "Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping." S&P'17.

Our Objective and Approach

❑**Objective** of Symbolic Opaque Predicates:

- Enhance the security of opaque predicates.
- Combat symbolic execution-based attackers.

❑**Approach:**

- Step 1: Investigate the limitations of symbolic execution tools.
- Step 2: Employ these limitations to obfuscate software.

Challenges of Symbolic Execution

	Challenge	Description
Symbolic-Reasoning Challenges	Symbolic Variable Declaration	Contextual variables other than arguments
	Covert Propagations	Propagating symbolic values in covert ways
	Buffer Overflows	Without proper boundary check
	Parallel Executions	Processing symbolic values in parallel codes
	Symbolic Memories	Symbolic values as the offset of memories
	Contextual Symbolic Values	Retrieving contextual values with symbolic values
	Symbolic Jumps	Symbolic values as the address of jump
	Floating-Point Numbers	Symbolic values in float/double
	Arithmetic Overflows	Beyond the scope of an integer type
Path-Explosion Challenges	Loops	Change symbolic values within loops
	Crypto Functions	Processing symbolic values with crypto functions
	External Function Calls	Processing symbolic values with external functions

[Hui’18] Hui Xu, Zirui Zhao, Yangfan Zhou, and Michael R. Lyu, “Benchmarking the Capability of Symbolic Execution Tools,” *TDSC*, 2018.

Example of Symbolic Memories

- Use symbolic values as the offsets to access memory.
- Theoretical challenge: some pointer analysis problems are NP-hard.

```
void func(int symvar){
   int l1_ary[] = {1,2,3,4,5,6,7};
   int l2_ary[] = {symvar,1,2,3,4,5,6,7};
   int i = l2_ary[l1_ary[symvar%7]];
   if(i == 1)
      foobar();
}
```

symbolic variable

a challenging problem: 2-leveled array

condition

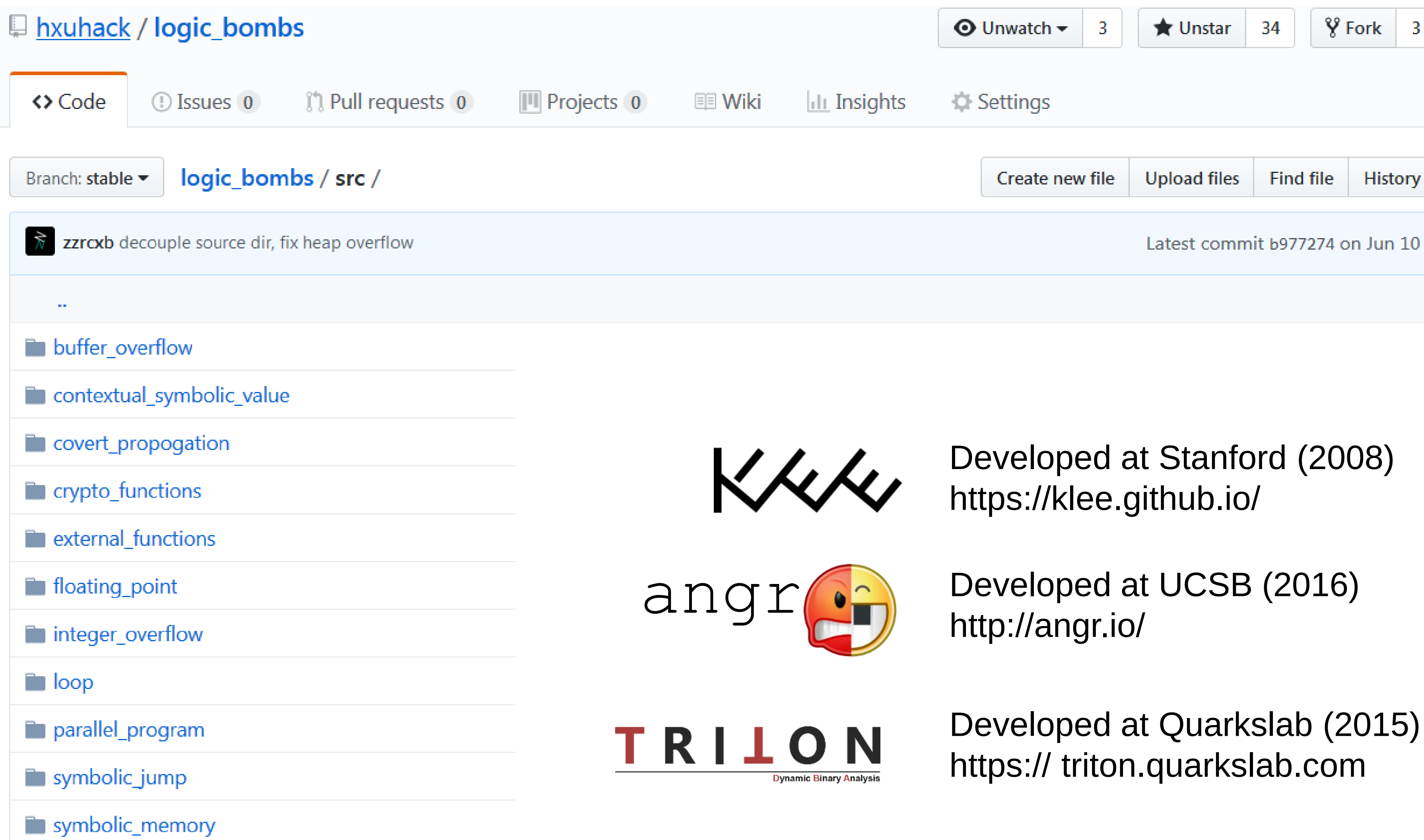
Can symbolic execution tools find a test case for triggering foobar()?

Example of Floating-point Numbers

- Floating-point numbers are approximations of real numbers.
- Defined in IEEE-754: Interval for 32-bit float: [1.401298464324817e-45, 32767.9990234].

```
void func(int symvar){
    float f = symvar/1000000.0;
    if(f==0.1){                                   no solution
        bogus();
    }
    if(1024+f == 1024 && f>0){                    solution: f=0.00001
        foobar();
    }
}
```

Examining the Prevalence of Challenges



Developed at Stanford (2008)
https://klee.github.io/

Developed at UCSB (2016)
http://angr.io/

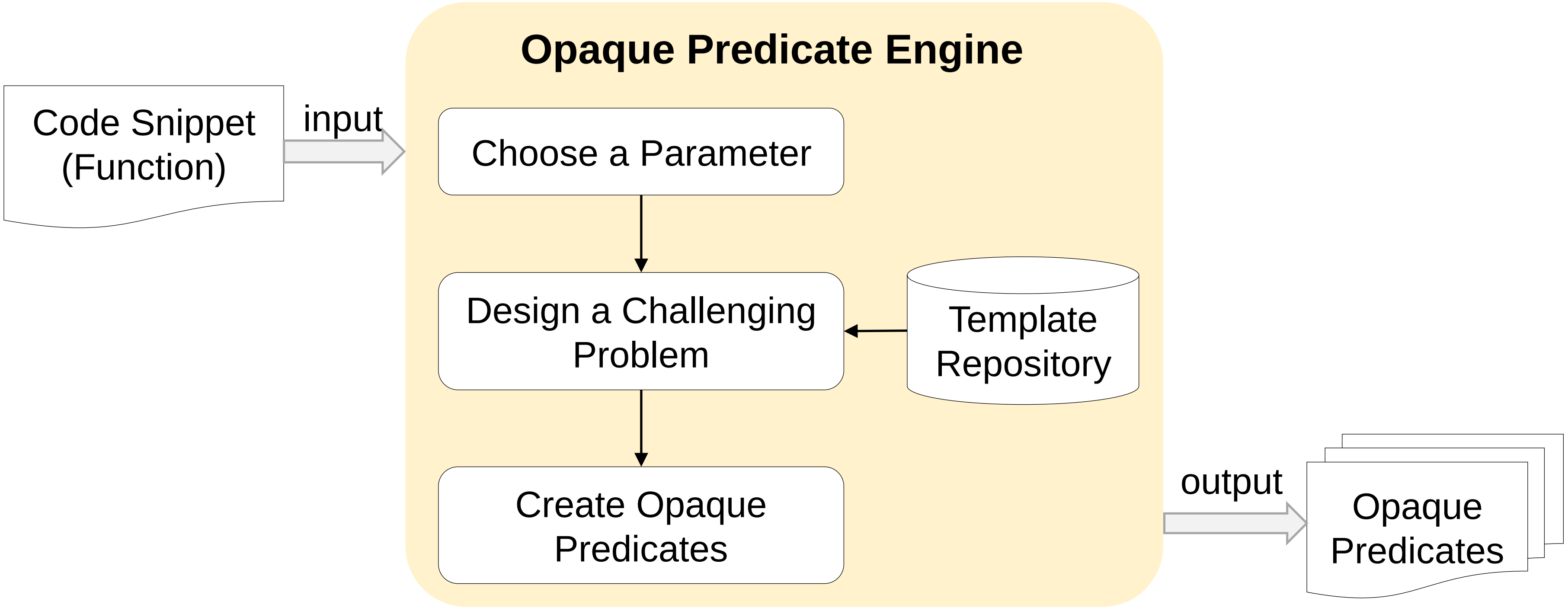
Developed at Quarkslab (2015)
https:// triton.quarkslab.com

Benchmarking Results

These tools failed most of the test cases.

Challenge		Result (pass #/ total #)		
		KLEE	Triton	Angr
Symbolic-Reasoning Challenges	Symbolic Variable Declaration	0/7	0/7	0/7
	Covert Propagations	1/5	1/9	4/9
	Buffer Overflows	0/4	0/4	2/4
	Parallel Executions	0/5	0/5	0/5
	Symbolic Memories	5/6	0/8	7/8
	Contextual Symbolic Values	0/7	0/7	0/7
	Symbolic Jumps	1/1	0/4	2/4
	Floating-Point Numbers	0/5	0/5	2/5
	Arithmetic Overflows	2/2	1/2	2/2
Path-Explosion Challenges	Loops	0/5	0/5	0/5
	Crypto Functions	0/2	0/2	0/2
	External Function Calls	0/8	1/8	3/8
Total		9/54	3/63	22/63

Design Symbolic Opaque Predicates



Template of Symbolic Memories

```
int func(int i){
    if(i == 7){
        foobar();
    }
}
```

Source Code

j=i%7+1

```
int func(int i){
    int l1_ary[] = {1,2,3,4,5,6,7};
    int l2_ary[] = {j,1,2,3,4,5,6,7};
    int j = l2_ary[l1_ary[i%7]];
    if(i == j)
        bogus();
    if(j == 1 && i == 7)
        foobar();
}
```

Type I opaque predicate

Type II

Bi-opaque

Obfuscated Code

Evaluation

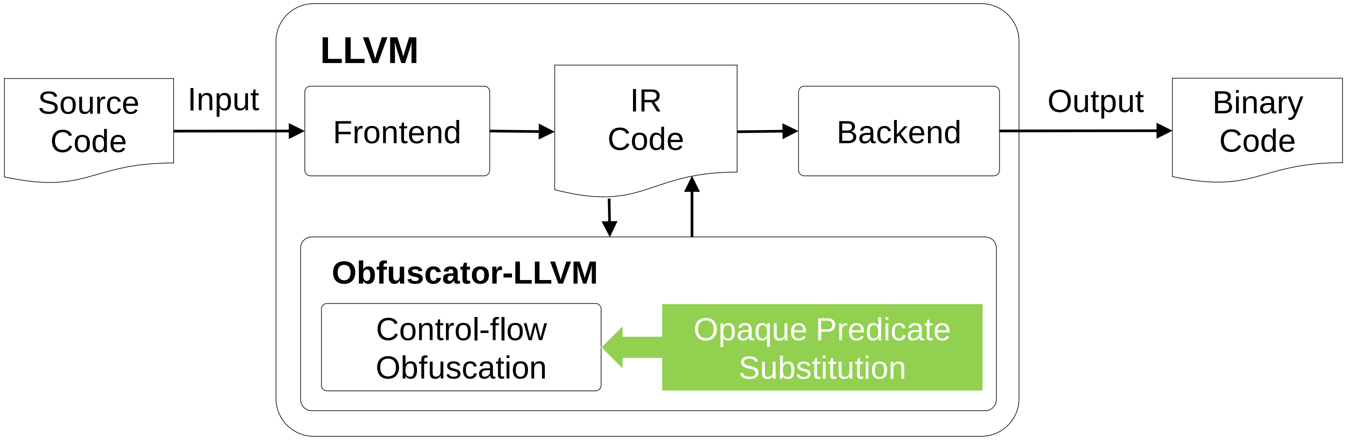
❑ **Performance Metrics:**

- Space overhead
- Execution overhead

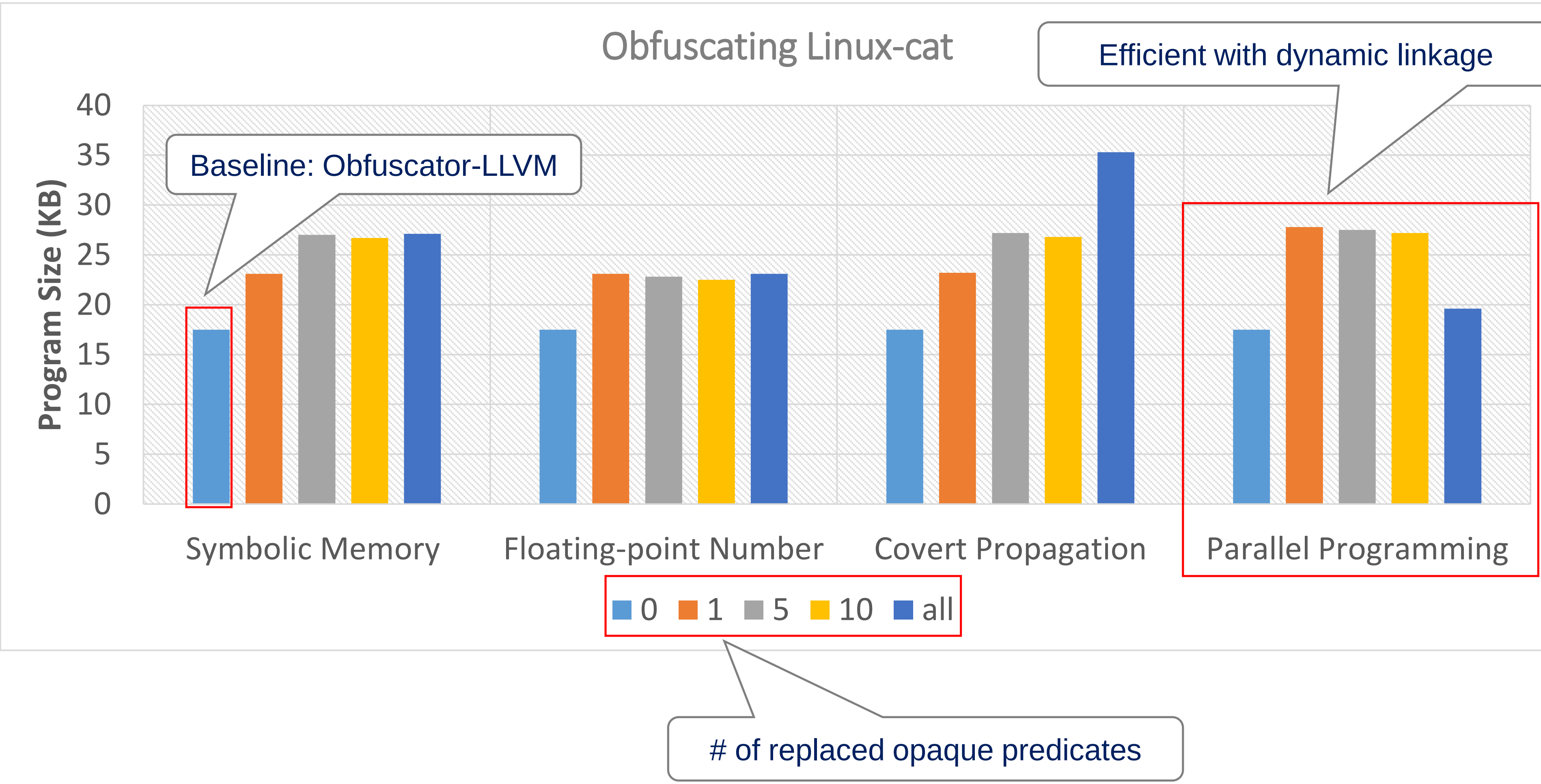
❑ **Target Programs:**

- Linux Busybox (*e.g.*, cat)
- Encryption programs (*e.g.*, AES)

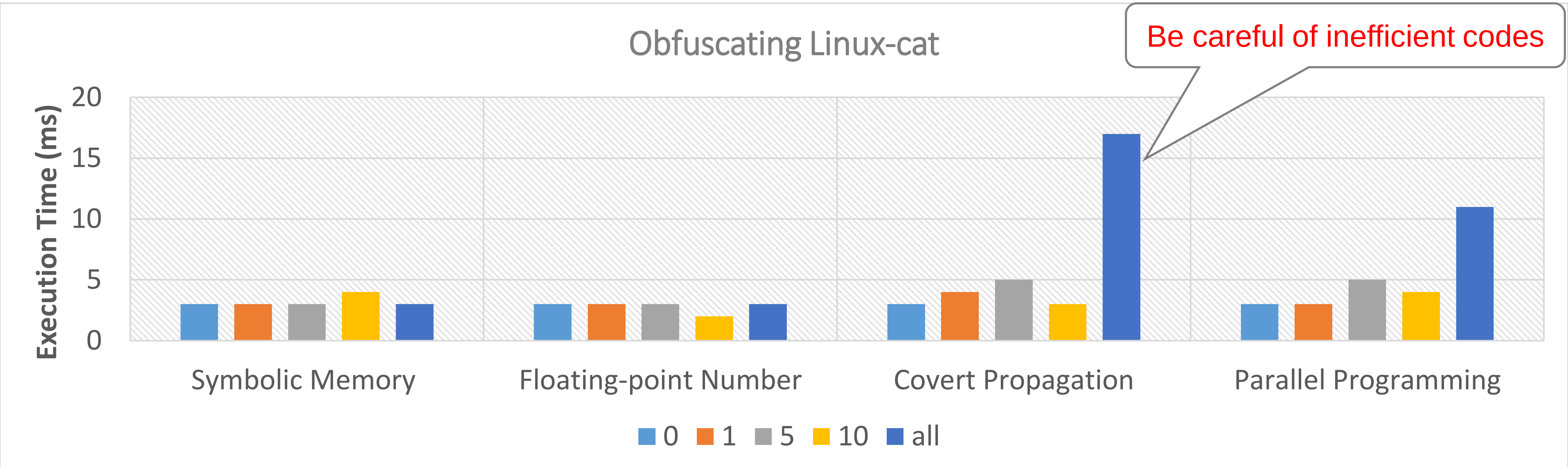
❑ **Prototype** based on Obfuscator-LLVM.

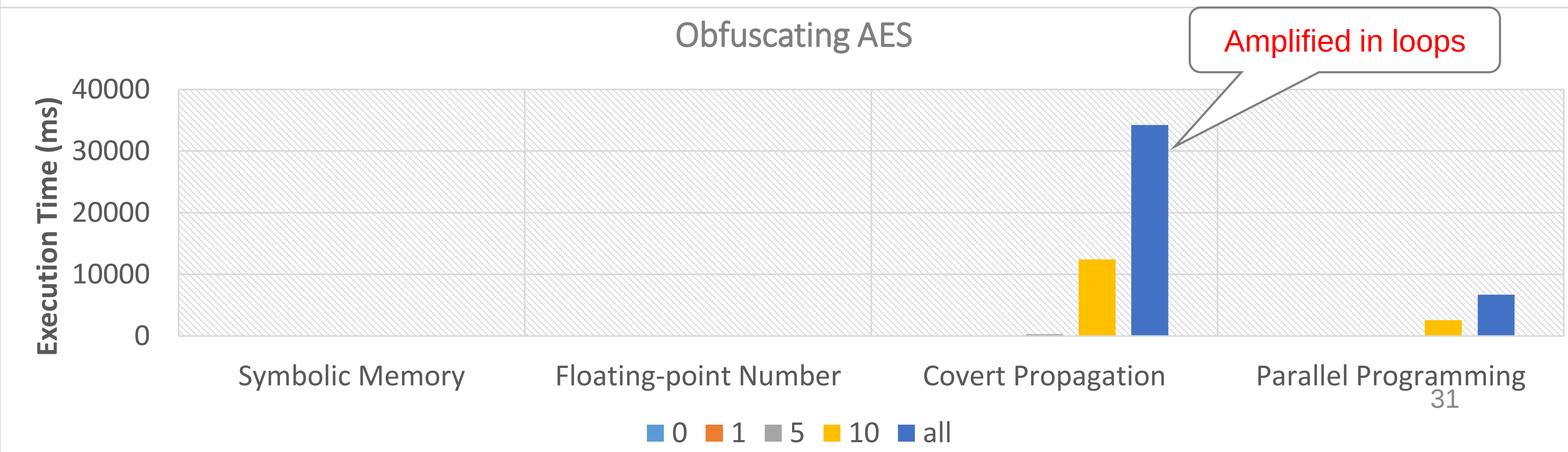


Space Overhead



Execution Overhead





Summary of Symbolic Opaque Predicates

❑**Objective**:

- To secure control-flow obfuscation against symbolic execution.

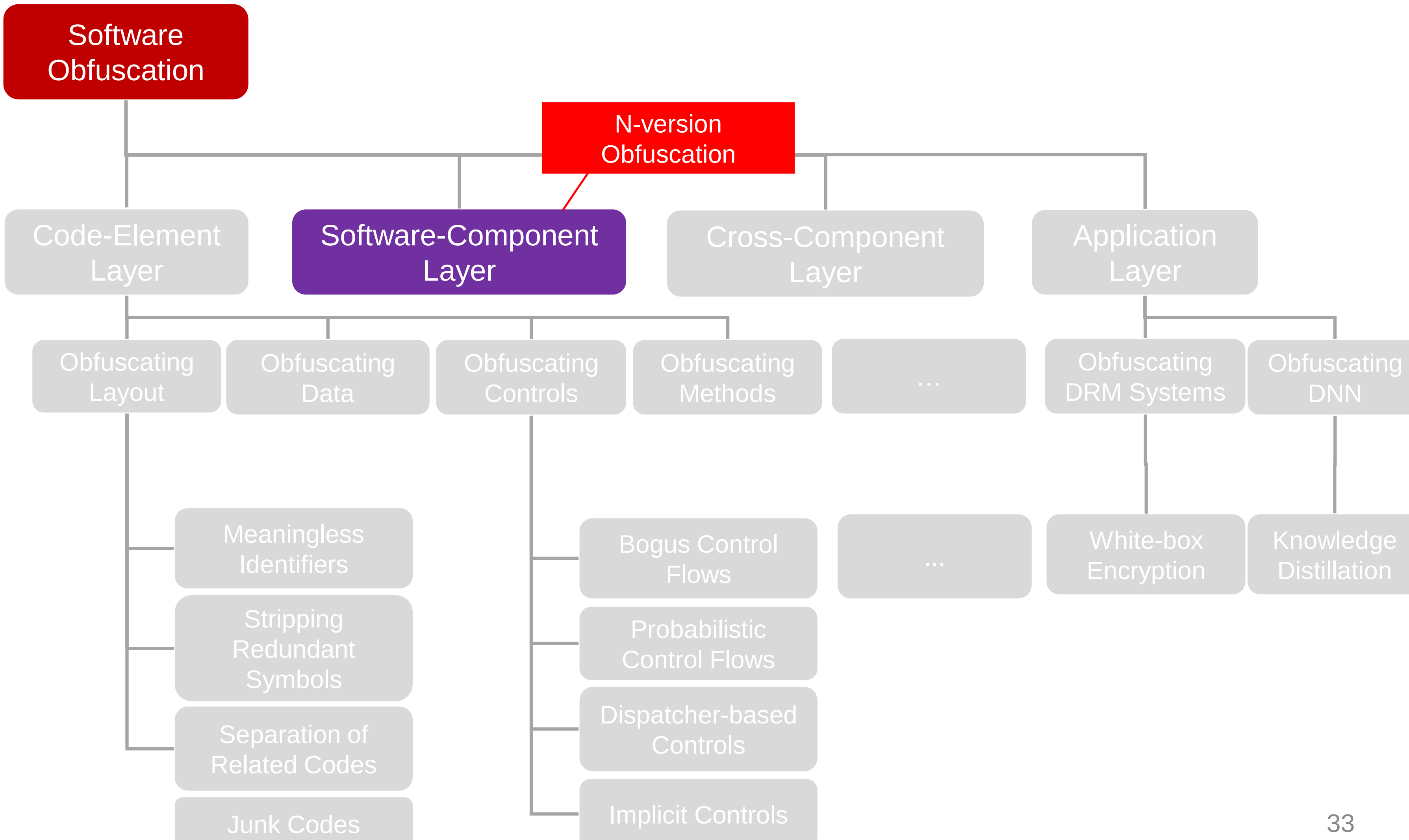
❑**Our Contribution**:

- We investigated the vulnerabilities of symbolic execution and developed a dataset to benchmark symbolic execution tools.
- We proposed a framework to compose opaque predicates leveraging these vulnerabilities.

❑**Current Work**:

- Enrich the template repository with more diversified samples.
- Develop a systematic strategy of opaque predicate insertion with small overhead.

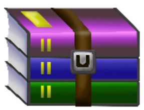
N-version Obfuscation



Motivation

❑ Software tampering attack is popular for smartphones, especially Android.

Static App Repack:



apk file

unpack

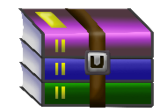
apktool

```
.class Lcom/f5/edge/LogonActivity$1;
.super Ljava/lang/Object;
.source "LogonActivity.java"

# interfaces
.implements Landroid/view/View$OnClickListener;


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Lcom/f5/edge/LogonActivity;->onCreate(Landroid/os/Bundle;)V
.end annotation
```

Smali code

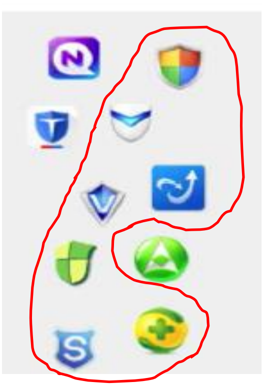
repack

apktool



tampered apk

Dynamic Injection: 7/10 Android security apps in China inject payloads into their "protected" apps.



```
root@android:/ # cat /proc/3789/maps
2a8cd000-2a8e0000 ---p 00000000 00:00 0
2a8e0000-2a8f0000 rw-p 00000000 00:00 0
2a8f0000-2a8fd000 ---p 00000000 00:00 0
2b000000-2b300000 ---p 00000000 00:00 0
2b300000-2b400000 rw-p 00000000 00:00 0
2b400000-2b800000 ---p 00000000 00:00 0
2e800000-2e821000 rw-p 00000000 00:00 0
38f00000-38f49000 rw-p 00000000 00:00 0
3c900000-3c929000 rw-p 00000000 00:00 0
40000000-4000c000 rw-p 00000000 00:00 0
4000c000-4000d000 r--s 00019000 103:0d 32635     /data/data/com.lbe.security/app_hips/client.jar
4000d000-4000e000 r--s 00000000 00:04 5228       /dev/ashmem/SurfaceFlinger read-only heap (delet
4000e000-4000f000 r--s 0029f000 103:0d 99723     /data/data/com.sankuai.meituan/code_cache/secon
```

LBE安全大师
LBESEC.COM

[Zhou'14] Rongyu Zhou, Leave The App Alone! - Attack and Defense of Android App, RSA2014

The Protection Challenge

- We can have a bunch of solutions, but none is overwhelming.

"Given enough time, effort and determination, a competent programmer will always be able to reverse engineer any application."

--Christian Collberg

[Collberg'97] Christian Collberg, *et al*. A taxonomy of obfuscating transformations, 1997.

Our Objective and Approach

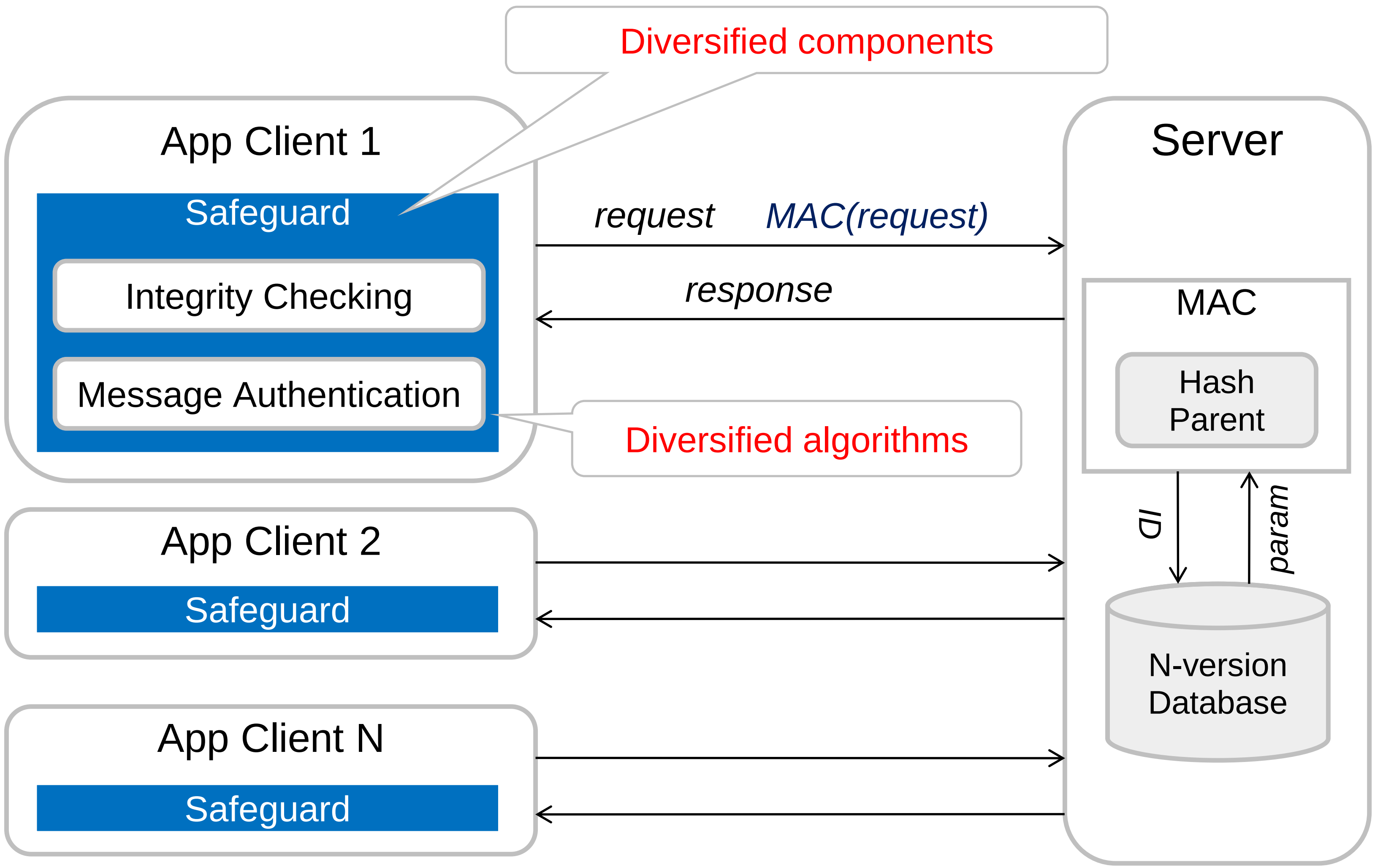
❑**Objective** of N-version obfuscation**:**

- Defend apps against software tampering attack.
- We focus on impeding large-scale attacks only.

❑**Approach:**

- Create diversified apps for different clients.
- Impede the replication of an attack on multiple hosts.

A Candidate Solution for Networked Apps



Diversified MAC Algorithms based on SHA1

Data: $w[80]$
`// blocks of plaintext`
for $i = 0; i < 80; i++$ **do**
  if $0 \le i \le 19$ **then**
    $f \leftarrow (b \text{ AND } c) \text{ OR } ((\text{NOT } b) \text{ AND } d)$;
    $k \leftarrow$ 0X5A827999;
  end
  if $20 \le i \le 39$ **then**
    $f \leftarrow b \text{ XOR } c \text{ XOR } d$;
    $k \leftarrow$ 0X6ED9EBA1;
  end
  if $40 \le i \le 59$ **then**
    $f \leftarrow (b \text{ AND } c) \text{ OR } (b \text{ AND } d) \text{ OR } (c \text{ AND } d)$;
    $k \leftarrow$ 0X8F1BBCDC;
  end
  if $60 \le i \le 79$ **then**
    $f \leftarrow b \text{ XOR } c \text{ XOR } d$;
    $k \leftarrow$ 0XCA62C1D6;
  end
  $temp \leftarrow (a \text{ LEFTROTATE } 5) + f + e + k + w[i]$;
  $e \leftarrow d$;
  $d \leftarrow c$;
  $c \leftarrow b \text{ LEFTROTATE } 30$;
  $b \leftarrow a$;
  $a \leftarrow temp$;
end

Each client employs a random combination

SHA1

Data: $f\_genes[80], k\_genes[80], w[80]$

2^160

for $i = 0; i < 80; i++$ **do**
  Call $f\_genes[i]$;
  `// Pointer to F0, F1, F2 or F3`
  F_TAIL($k\_genes[i], w[i]$);
end
Function *F0()*
  $f \leftarrow (b \text{ AND } c) \text{ OR } ((\text{NOT } b) \text{ AND } d)$;
Function *F1()*
  $f \leftarrow b \text{ XOR } c \text{ XOR } d$;
Function *F2()*
  $f \leftarrow (b \text{ AND } c) \text{ OR } (b \text{ AND } d) \text{ OR } (c \text{ AND } d)$;
Function *F3()*
  $f \leftarrow b \text{ XOR } c \text{ XOR } d$;
Function *F_TAIL(k, w)*
  $temp \leftarrow (a \text{ LEFTROTATE } 5) + f + e + k + w$;
  $e \leftarrow d$;
  $d \leftarrow c$;
  $c \leftarrow b \text{ LEFTROTATE } 30$;
  $b \leftarrow a$;
  $a \leftarrow temp$;

Parent of SHA1

Feasibility of Automation

- Automation of N-version Generation:
  - Can be implemented as a compiler pass.
- Automation of N-version Delivery:
  - Server delivers the safeguard as a dynamic library to each client at the first time of launch.
  - Clients register their versions on the server.

Summary of N-version Obfuscation

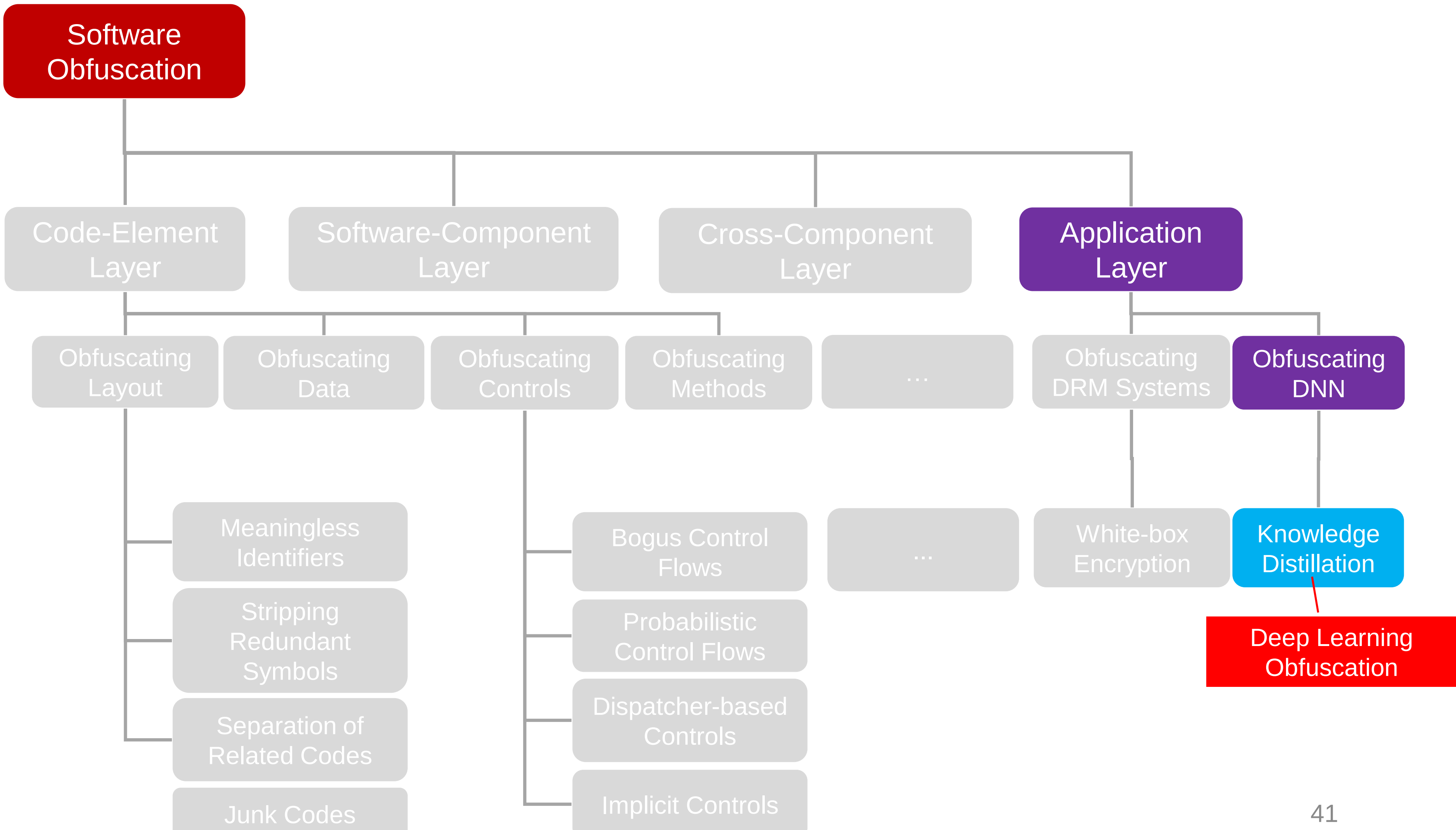
❑Objective:

- To defend software against tampering attacks.
- We focus on impeding large-scale attacks.

❑Our Contribution:

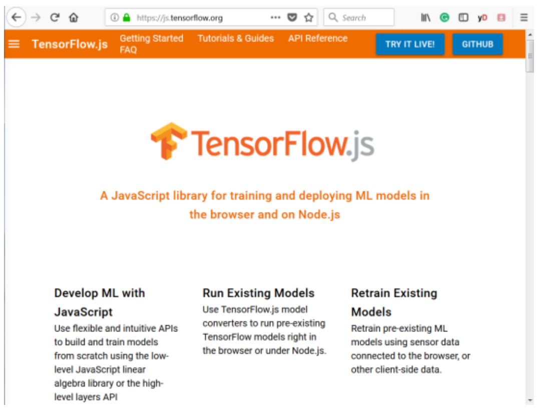
- We proposed an N-version obfuscation solution for networked apps.
- It is efficient to automatically generate and deliver N software versions.

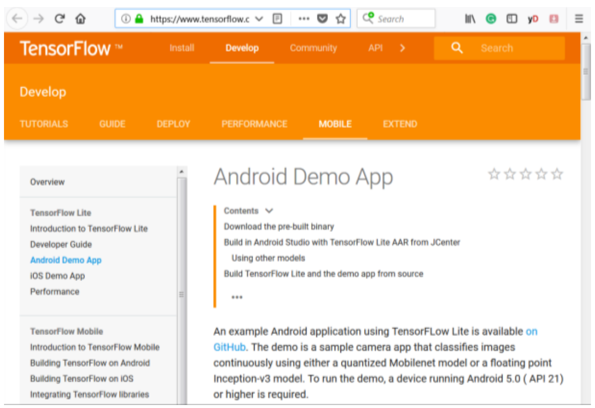
Our Proposed Approaches in the Taxonomy



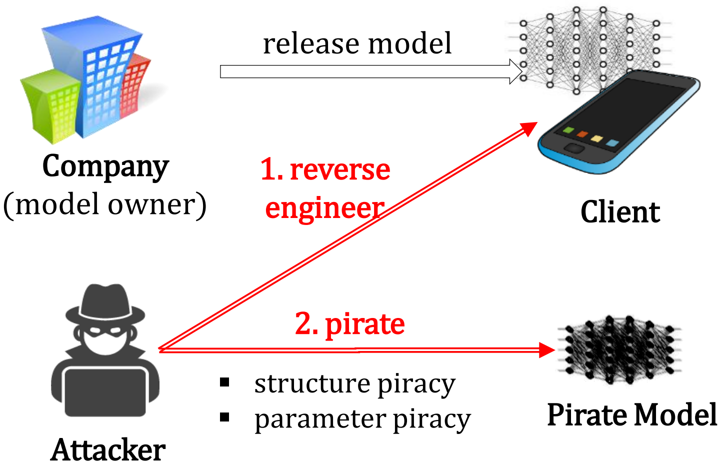
Motivation

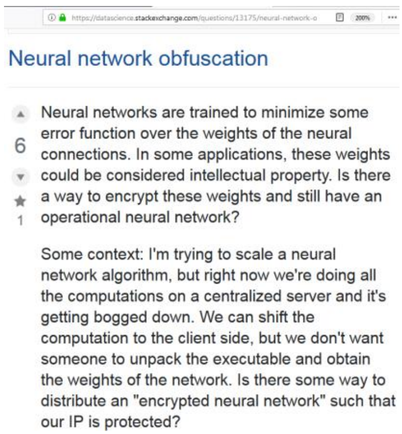
- ❑ Running deep learning models on client sides is a trend.





- ❑ Deep learning models are vulnerable to piracy.

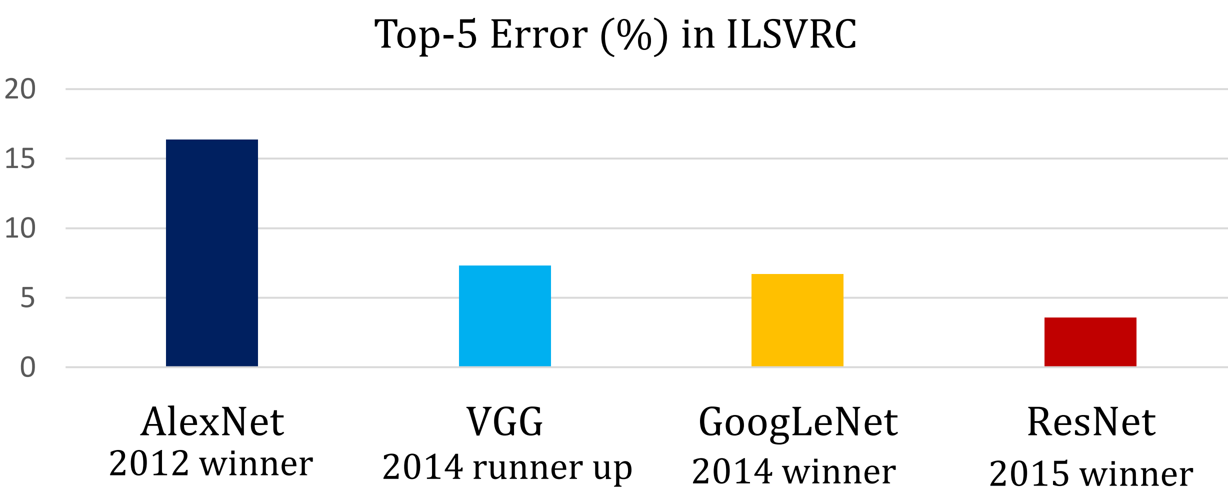


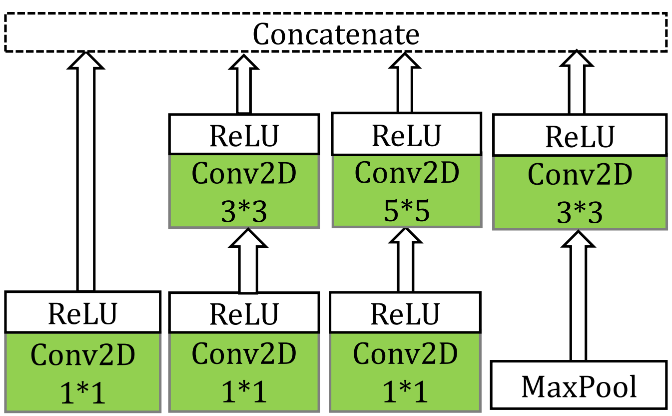


https://datascience.stackexchange.com/questions/13175/neural-network-obfuscation

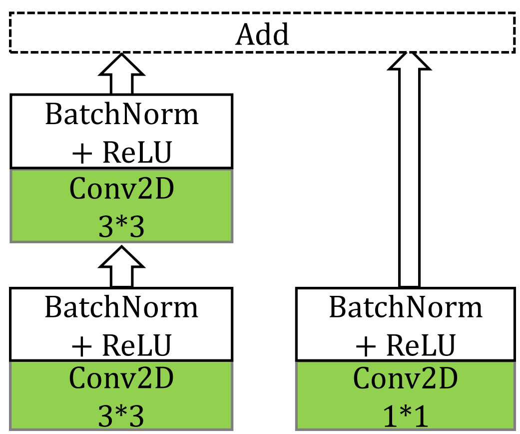
Structure Piracy

- Structure is the key factor for improving accuracy.

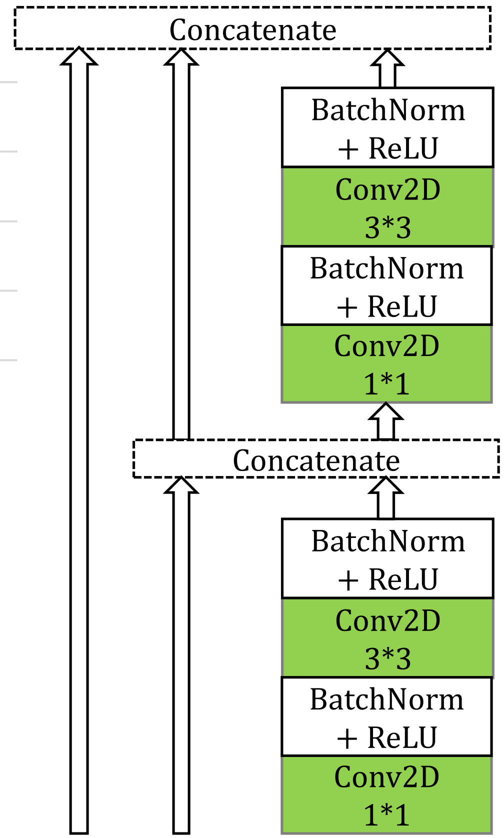




GoogLeNet Inception Module



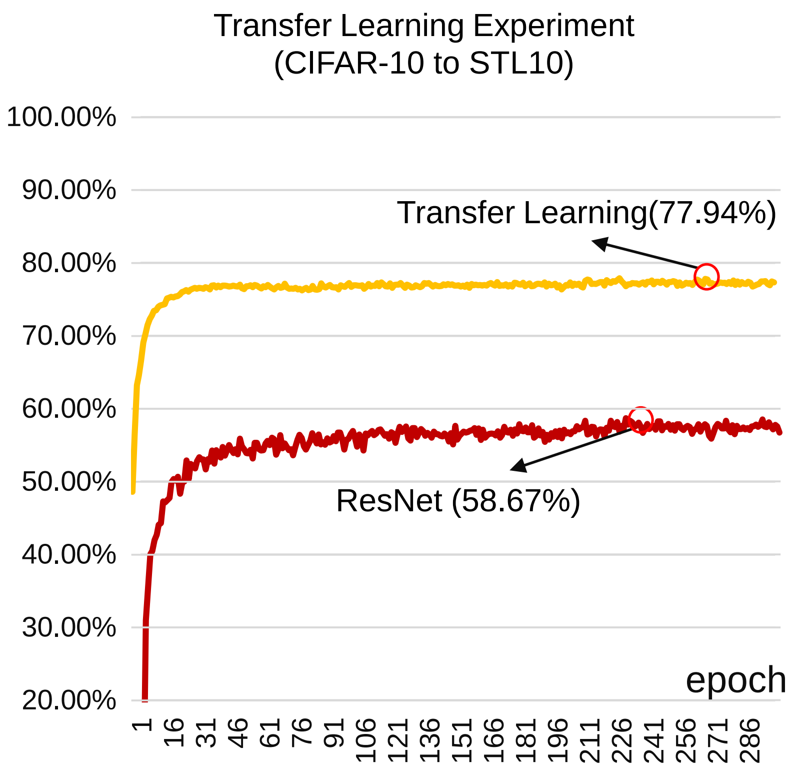
ResNet Inception Module

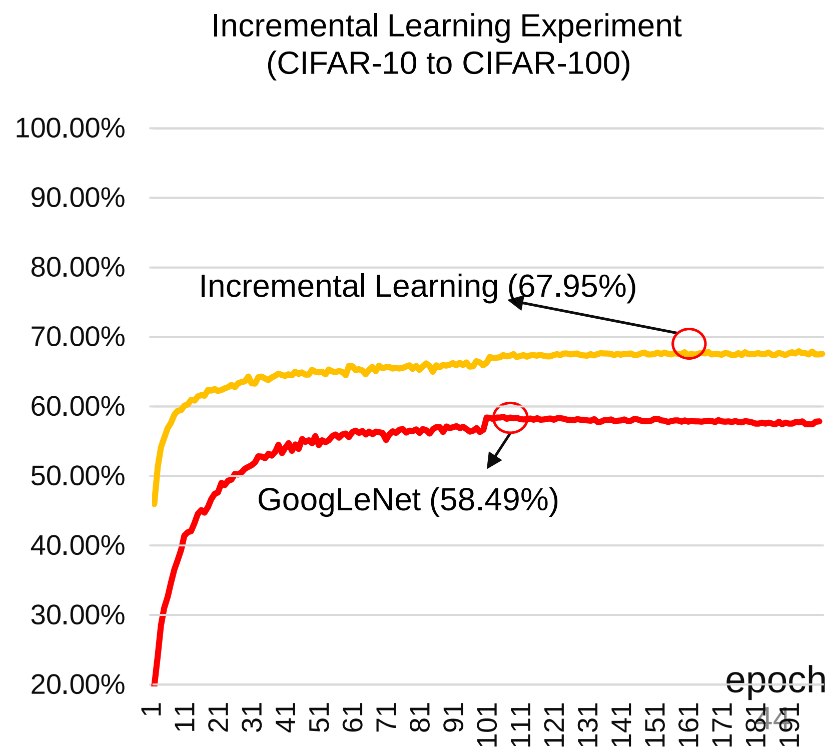


DenseNet Inception Module (2017)

Parameter Piracy

- Employ a well-trained model as the initial state to create new models.
  - Transfer learning.
  - Incremental learning.

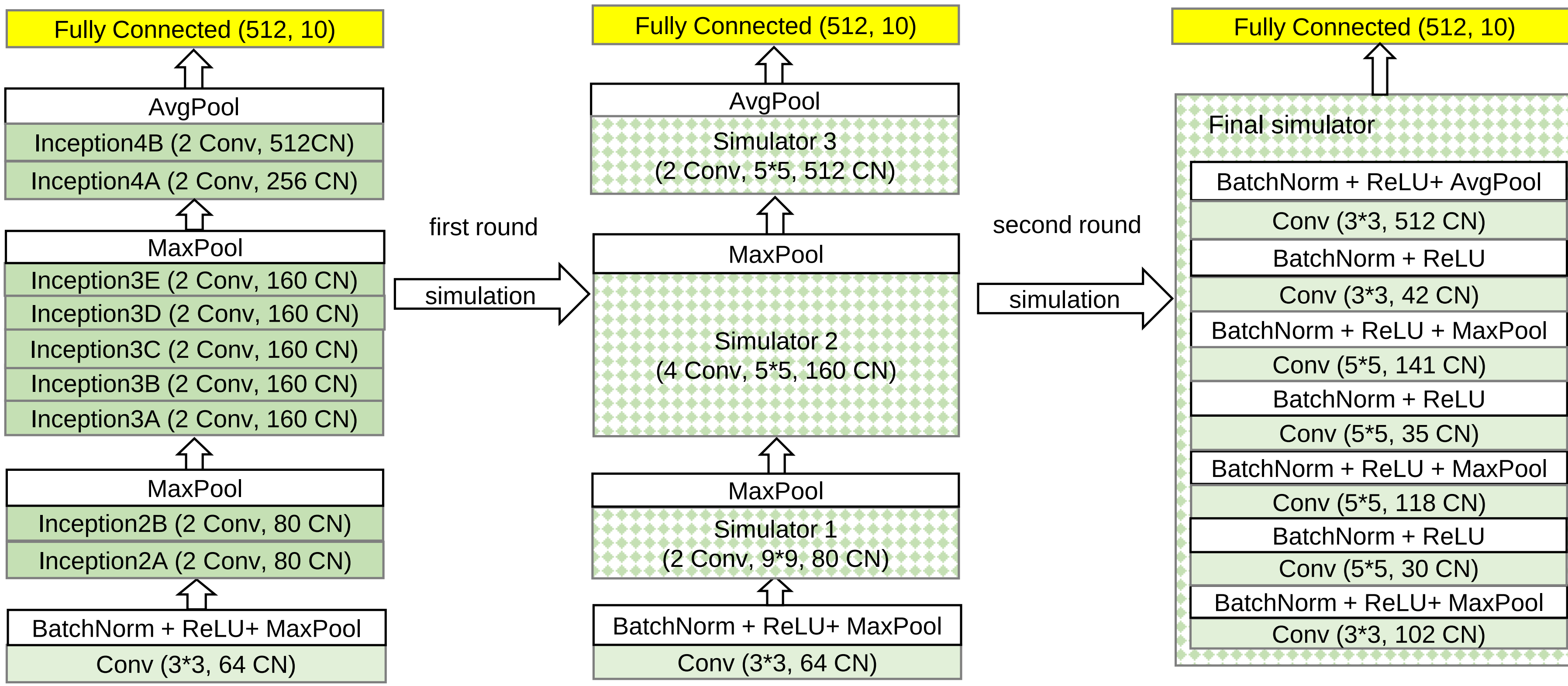




Our Objective and Approach

- **Objective**: defend deep learning models against piracy attacks.
  - Structure piracy.
  - Parameter piracy.
- **Approach**: simulate the model with a shallow network.
  - Combat structure piracy by hiding the critical structures.
  - Combat parameter piracy by degrading the learning ability.
  - We should obtain a simulation network with zero accuracy loss.
    - Recursive simulation.
    - Joint training.

Recursive Simulation of GoogLeNet



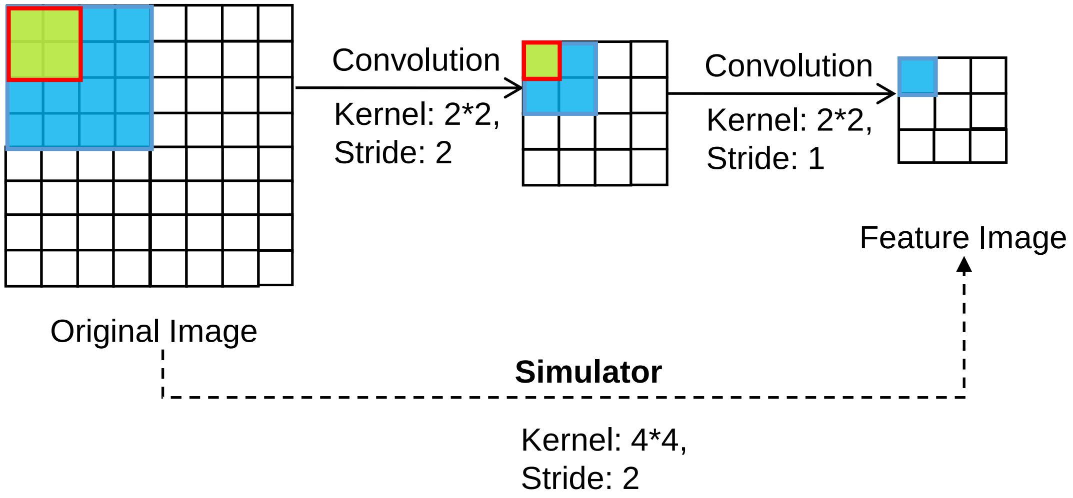
GoogLeNet — **19 conv layers**

Intermediate Result — **9 conv layers**

Obfuscated GoogLeNet — **7 conv layers**

Principle of Simulator Design

Features should be computed from the same (or super) set of pixels.



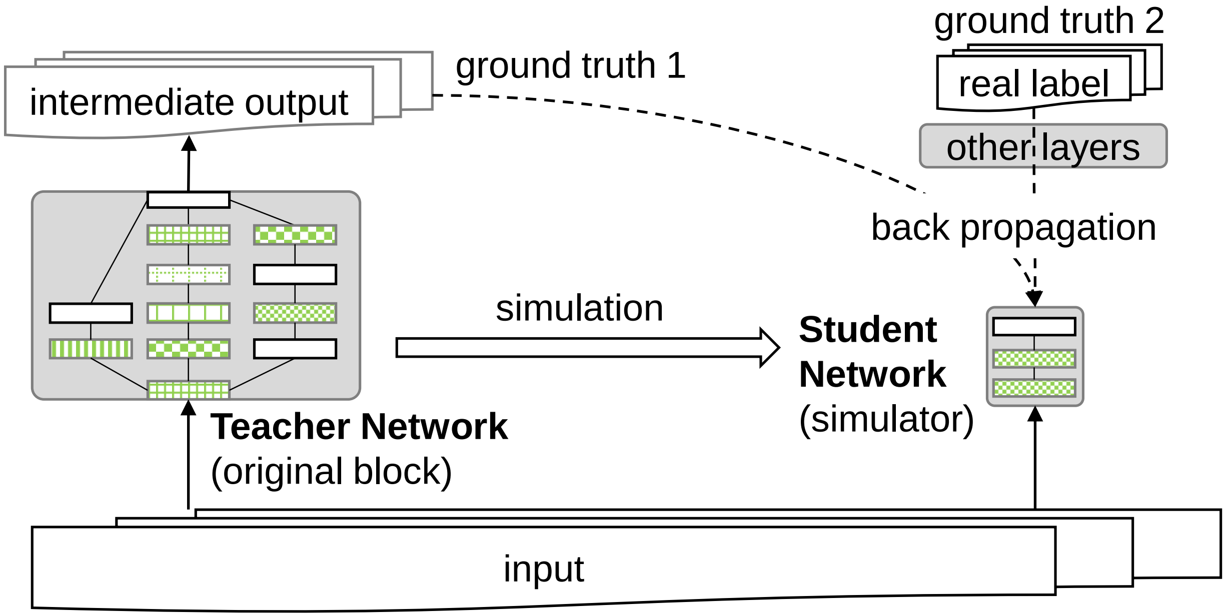
Kernel size of the simulation network:

Compress 2 layers to 1 layer: $h = h_1 + (h_2 - 1) \times s_1$

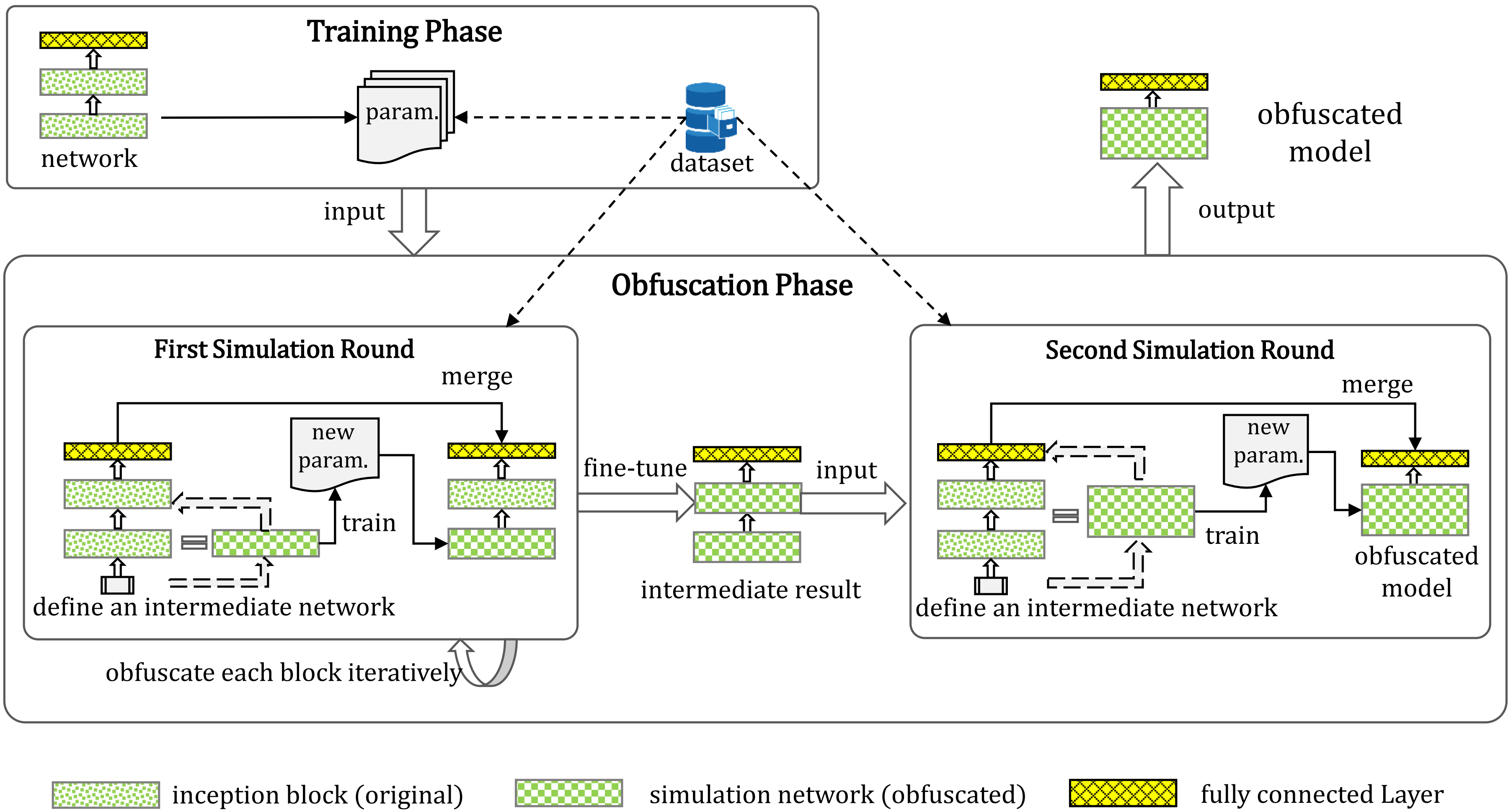
Compress n layers to 1 layer: $h = h_1 + (h_n - 1) \times \cdots \times (h_2 - 1) \times s_{n-1} \times \cdots \times s_1$

Joint Training

- An improvement based on the teacher-student network.
  - The loss of student network cannot be zero.
  - The teacher network itself has errors.



Overall Framework of Obfuscation



Evaluation Experiments

❑Evaluation Purposes:

- Accuracy: can we obfuscate the model with zero accuracy loss?
- Overhead: size and execution cost.
- Security: resilience to parameter piracy.

❑Models for Obfuscation:

- GoogLeNet, ResNet,and DenseNet trained with CIFAR-10.
- ResNet and DenseNet trained with ImageNet (five classes).

Results of Obfuscated Models

Model (CIFAR-10)		Performance			Overhead	
		Accuracy	Size (MB)	Time (us)	Size	Time
GoogLeNet	Original	90.83%	2.51	17.85	-	-
	Obfuscated	90.92%	2.49	7.01	-1%	-63%
ResNet	Original	90.94%	43.36	10.50	-	-
	Obfuscated	91.04%	11.38	5.17	-74%	-51%
DenseNet	Original	90.14%	4.24	35.53	-	-
	Obfuscated	90.31%	4.21	5.52	-1%	-84%

Model (ImageNet)		Performance			Overhead	
		Accuracy	Size (MB)	Time (us)	Size	Time
ResNet	Original	92.4%	43.37	89	-	-
	Obfuscated	92.4%	36.72	59	-15%	-34%
DenseNet	Original	91.6%	4.27	154	-	-
	Obfuscated	92.8%	2.94	56	-31%	-64%

- ❑ No accuracy loss.
- ❑ More efficient.

Resilience to Parameter Piracy

- The accuracy of pirated models based on the obfuscated models declines obviously than based on the original ones.

Model		Incremental Learning (CIFAR-10 to CIFAR-100)		Transfer Learning (CIFAR-10 to STL10)	
		Accuracy	Degradation	Accuracy	Degradation
GoogLeNet	Original	66.5%	-	79.15%	-
	Obfuscated	63.59%	-4.4%	77.95%	-1.5%
ResNet	Original	66.92%	-	78.86%	-
	Obfuscated	64.77%	-3.2%	75.97%	-3.7%
DenseNet	Original	67.16%	-	78.45%	-
	Obfuscated	62.91%	-6.3%	76.90%	-2.0%

Summary of Deep Learning Obfuscation

- **Objective**: to secure deep learning models against piracy.
- **Our Contribution**:
  - We proposed a simulation-based obfuscation approach.
  - We conducted real-world experiments and achieved promising results.
    - No accuracy loss.
    - No overhead.
    - Resilient to parameter piracy.

Conclusion

- We proposed layered obfuscation as a promising way for software obfuscation.
- We presented a taxonomy of obfuscation techniques for layered obfuscation.
- We discussed three novel obfuscation techniques.
  - Symbolic opaque predicates.
  - N-version obfuscation.
  - Deep learning obfuscation.

Future Work

- Practice layered obfuscation with more real-world software.
- Develop a methodology for implementing layered obfuscation.
- Propose new obfuscation techniques for new security issues.
- Develop a practical obfuscation tool integrating multiple techniques.

Publications Related to the Thesis

❑ Motivation and Taxonomy (Chapter 2):

[1] "Assessing the Security Properties of Software Obfuscation," **Hui Xu**, and Michael R. Lyu, in *IEEE Security & Privacy Magazine*, Oct, 2016.

[2] "Layered Obfuscation: A Taxonomy of Software Obfuscation Techniques for Layered Security," **Hui Xu**, Jiang Ming, Yangfan Zhou, and Michael R. Lyu, *under review by Elsevier Computers & Security*.

❑ Newly Proposed Approaches (Chapter 3,4, 5):

[3] "Manufacturing Resilient Bi-Opaque Predicates against Symbolic Execution," **Hui Xu**, Yangfan Zhou, Yu Kang, Fengzhi Tu, and Michael R. Lyu, in *Proc. of the 48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.

[4] "Concolic Execution on Small-Size Binary Codes: Challenges and Empirical Study," **Hui Xu**, Yangfan Zhou, Yu Kang, and Michael R. Lyu, in *Proc. of the 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017.

[5] "Benchmarking the Capability of Symbolic Execution Tools," **Hui Xu**, Zirui Zhao, Yangfan Zhou, and Michael R. Lyu, under review (minor revision) in *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2018.

[6] "N-version Obfuscation," **Hui Xu**, Yangfan Zhou, and Michael R. Lyu, in *Proc. of the 2nd Cyber-Phsical System Security Workshop (in conjunction with AsiaCCS)*, 2016.

[7] "DeepObfuscation: Securing the Structure of Convolutional Neural Networks via Obfuscation," **Hui Xu**, Yuxin Su, Zirui Zhao, Yangfan Zhou, Michael R. Lyu, and Irwin King, *under review by NDSS*, 2019.

Other Publications Related to Software Engineering and Cybersecurity

[1] “IntelliAd: Assisting Mobile App Developers in Measuring Ad Costs Automatically [Poster],” Cuiyun Gao, Yichuan Man, **Hui Xu**, Jieming Zhu, Yangfan Zhou and Michael R. Lyu, in *the 39th International Conference on Software Engineering (ICSE-C)*, 2017.

[2] “DiagDroid: Android Performance Diagnosis via Anatomizing Asynchronous Executions,” Yu Kang, Yangfan Zhou, **Hui Xu**, and Michael R. Lyu, in *Proc. of the 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, 2016.

[3] “SpyAware: Investigating the Privacy Leakage Signatures in App Execution Traces,” **Hui Xu**, Yangfan Zhou, Cuiyun Gao, Yu Kang, Michael R. Lyu, in *Proc. of the 26th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2015.

[4] “AR-Tracker: Track the Dynamics of Mobile Apps via User Review Mining,” Cuiyun Gao, **Hui Xu**, Junjie Hu, and Yangfan Zhou, in Proc. of the *International Workshop on Internet-based Virtual Computing Environment (IVCE)*, 2015.

[5] “Towards Continuous and Passive Authentication via Touch Biometrics: An Experimental Study on Smartphones,” **Hui Xu**, Yangfan Zhou, and Michael R. Lyu, in *Proc. of the USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2014.

[6] “Towards Designing Fault-Tolerant Deep Learning Systems via N-version Programming,” **Hui Xu**, Zhuangbin Chen, Weibin Wu, Irwin King, Michael Lyu, under review.

[7] “Toward Detecting Real-world Adversarial Corner Cases in Deep Neural Networks,” Weibin Wu, **Hui Xu**, Sanqiang Zhong, Michael, R. Lyu, Irwin King, under review.


Q&A