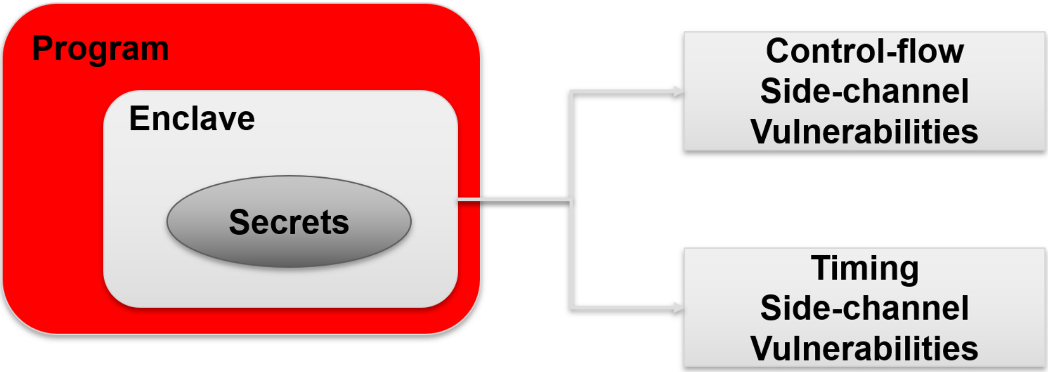
Time and Order: Towards Automatically Identifying Side-Channel Vulnerabilities in Enclave Binaries

Wubing Wang, Yinqian Zhang, and Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

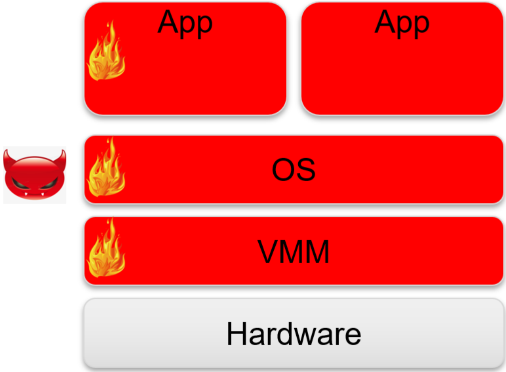
RAID 2019

Objective

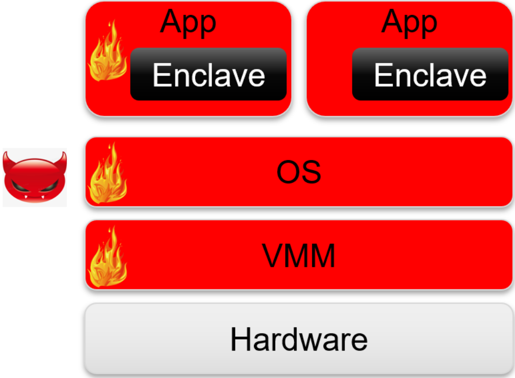

Program
Enclave
Secrets
Control-flow Side-channel Vulnerabilities
Timing Side-channel Vulnerabilities

Intel SGX

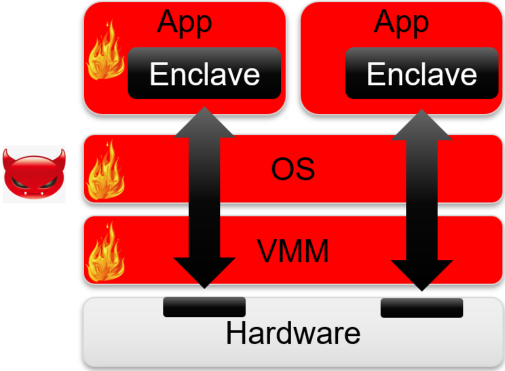
Intel SGX



Intel SGX



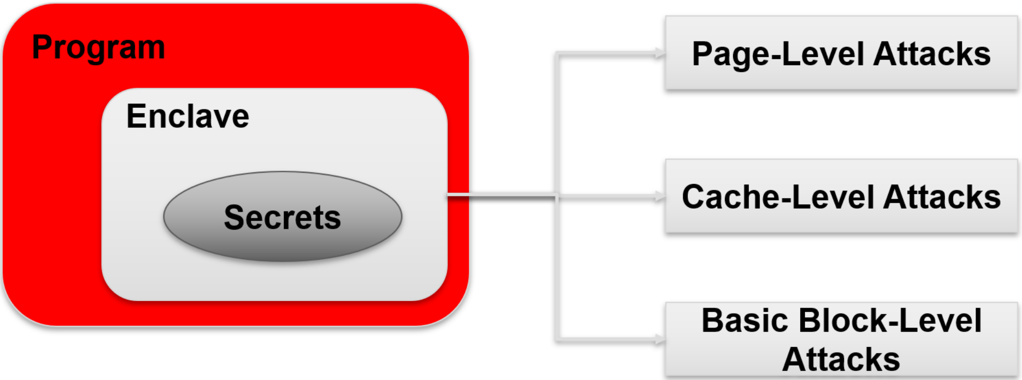
Intel SGX



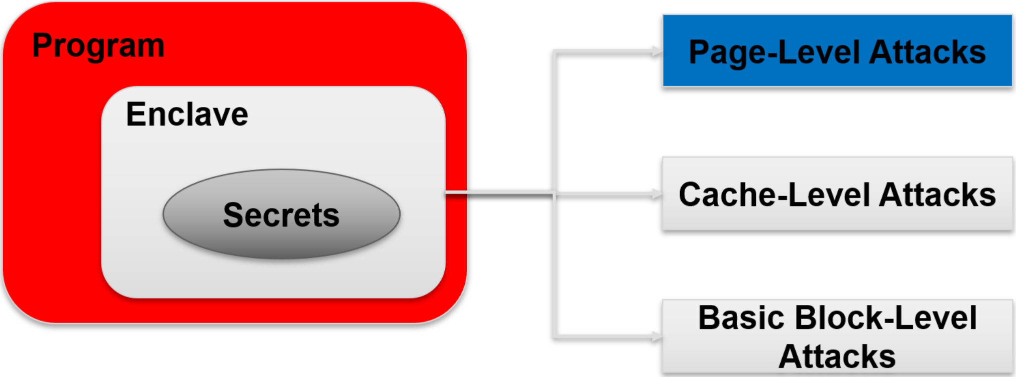
Intel SGX side-channel attacks - Granularity

1. Different Granularities
2. Different Targets

Intel SGX side-channel attacks - Granularity



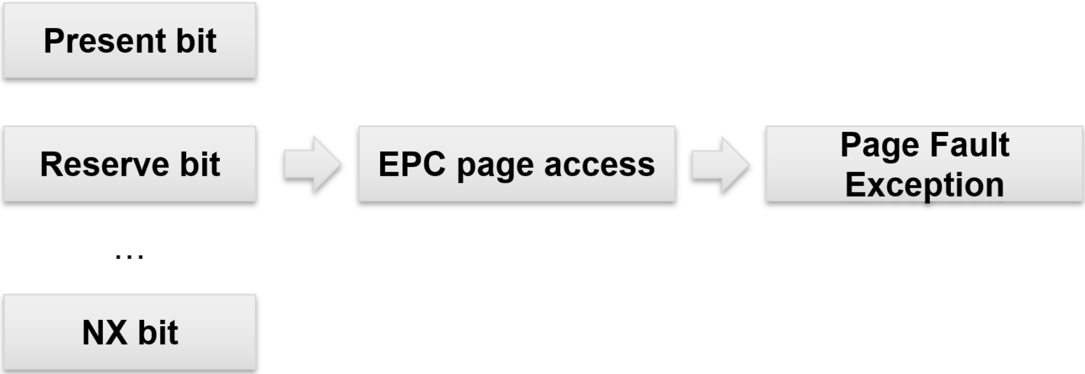
Intel SGX side-channel attacks - Granularity



Page-Level Attacks

1. Approaches to observe page-level pattern
2. The page-level vulnerability

Page-Level Attacks



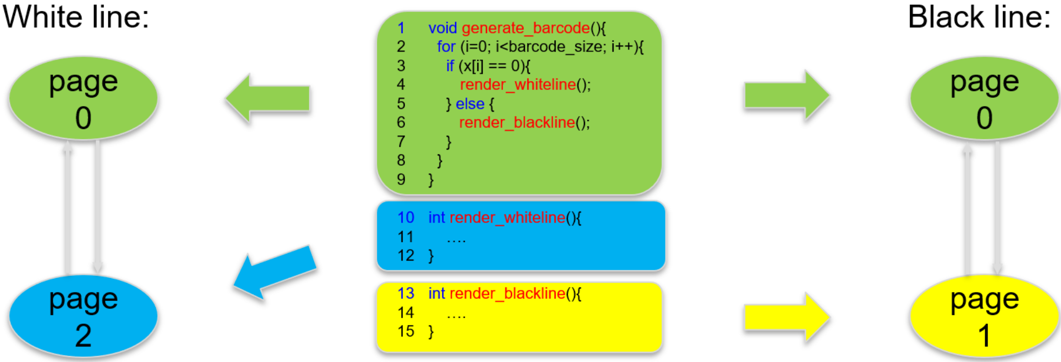
Page-Level Attacks

Barcode:

Page-Level Attacks

Barcode: [black line, white line, black line, black line, black line]

Page-Level Attacks



Page-Level Attacks

Barcode: [black line]

Page Sequence:

page 0, page 1, page 0

Page-Level Attacks

Barcode: [black line, white line]

Page Sequence:

page 0, page 1, page 0, page 2, page 0

Page-Level Attacks

Barcode: [black line, white line, black line]

Page Sequence:

page 0, page 1, page 0, page 2, page 0, page 1,
page 0

Page-Level Attacks

Barcode: [black line, white line, black line, black line]

Page Sequence:

page 0, page 1, page 0, page 2, page 0, page 1,
page 0, page 1, page 0

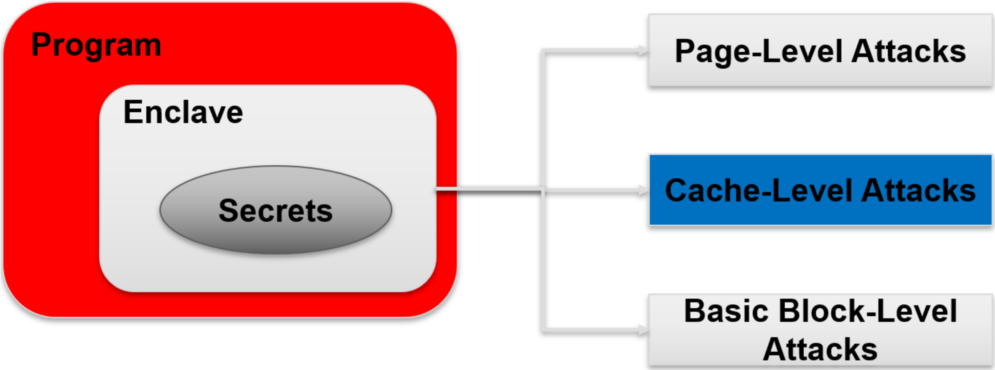
Page-Level Attacks

Barcode: [black line, white line, black line, black line, black line]

Page Sequence:

page 0, page 1, page 0, page 2, page 0, page 1,
page 0, page 1, page 0, page 1, page 0

Cache-Level Attacks



Cache-Level Attacks

1. Approaches to observe cache-level pattern
2. The cache-level vulnerability

Cache-Level Attacks

Prime + Probe

1. Occupy specific cache set
2. Victim program is scheduled
3. Check which cache sets are still occupied

Flush + Reload

1. Map binary into address space
2. Flush a cache line from the cache
3. Victim program is scheduled
4. Check Whether the flushed cache line has been reloaded

Cache-Level Attacks

Prime + Probe

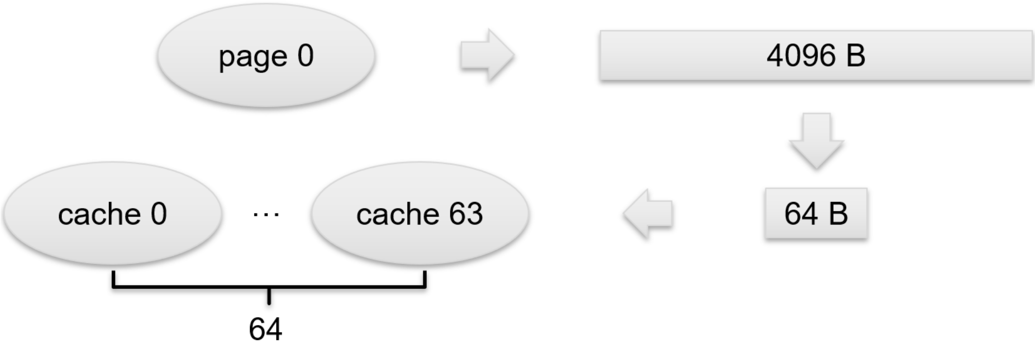
1. Occupy specific cache set
2. Victim program is scheduled
3. Check which cache sets are still occupied

Flush + Reload

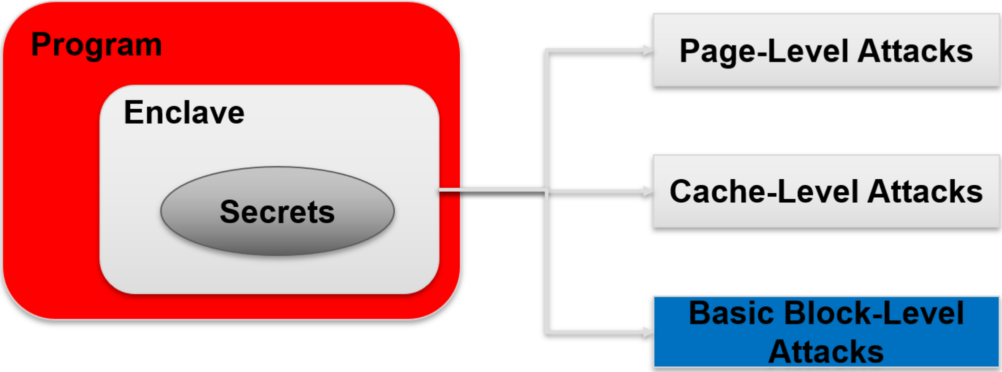
1. Map binary into address space
2. Flush a cache line from the cache
3. Victim program is scheduled
4. Check Whether the flushed cache line has been reloaded

Not applicable: SGX do not share memory with external !

Cache-Level Attacks



Cache-Level Attacks



Attack Targets

Program Inputs (e.g., Hunspell, Libjpeg, Freetype, Apache)
Controlled-channel (S&P'15), Branch Shadowing (USENIX'17)

Encrypted Data (e.g., Padding Oracle attack & Bleichenbacher attack)
Stacco (CCS'17)

Cryptography Key [e.g., RSA, DSA, AES]
DATA (USENIX'18), MicroWalk (ACSAC'18), CacheD (USENIX'17)

Genomic sequences
Software Grand Exposure(WOOT'17)

Motivations

1. The timing information is not thoroughly used
2. No automatic tool to detect the side-channel attack in general

Motivations

1. The timing information is not thoroughly used

"An analysis of covert timing channels" John C. Wray 1992:
Both storage nature (order) and timing nature are attributes of the channel, and a given channel may posses either or both.

Motivations

Storage nature (order):

Input 1: page 0, page 1, page 0, page 2

Input 2 page 0, page 2, page 0, page 1

Motivations

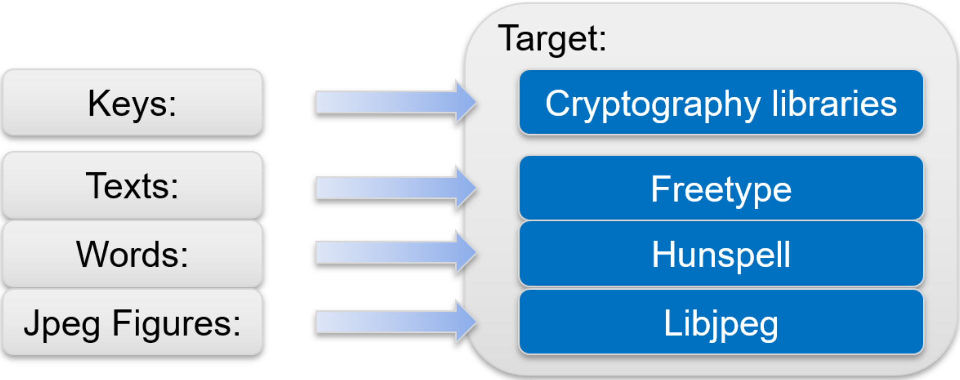
Timing nature:

Input 1:	page 0,	page 1,	page 0,	page 2
	2 ns	5 ns	2 ns	4 ns
Input 2	page 0,	page 1,	page 0,	page 2
	2 ns	50 ns	2 ns	4 ns

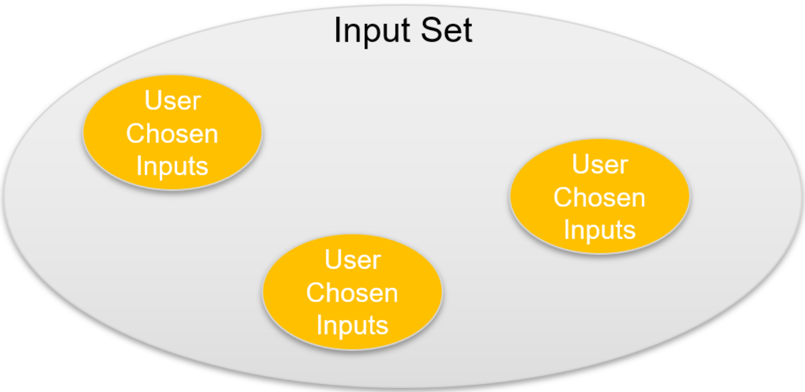
Motivations

Input - execution mapping

Motivations



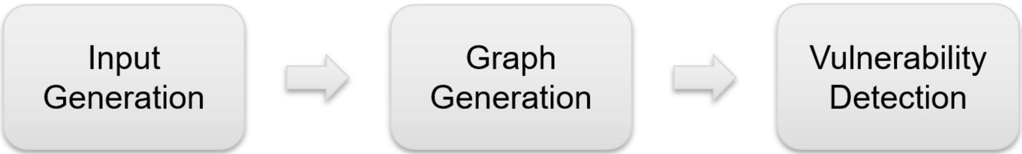
Motivations



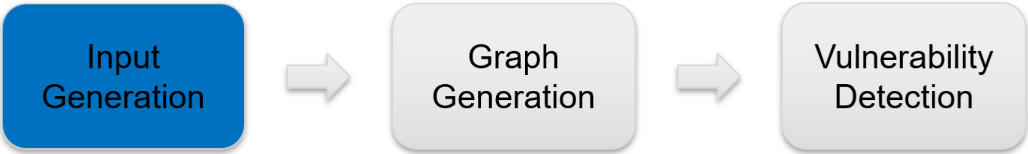
Challenges

1. How to accurately measure the timing information
2. What is the relationship between each input with the whole input set and other inputs

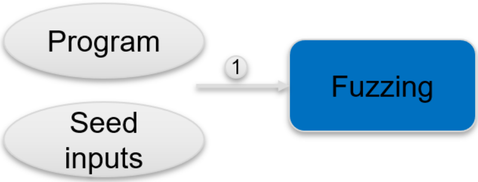
ANABLEPS


Input
Generation
Graph
Generation
Vulnerability
Detection

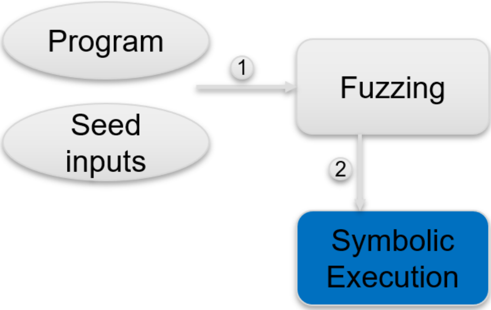
ANABLEPS


Input Generation
Graph Generation
Vulnerability Detection

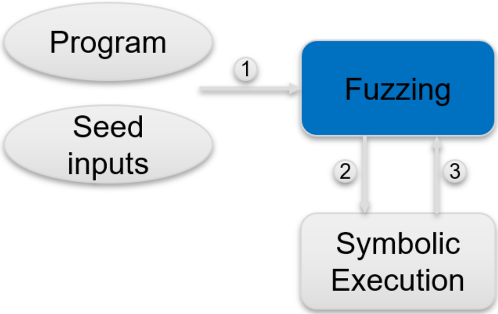
Input Generation



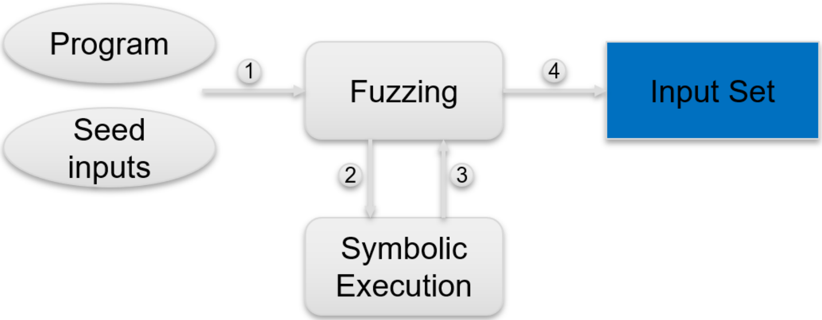
Input Generation



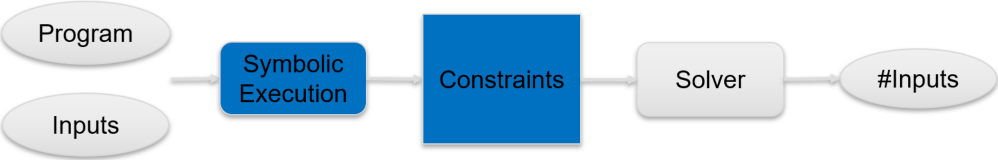
Input Generation



Input Generation



Input Space


Program
Inputs
Symbolic Execution
Constraints
Solver
#Inputs

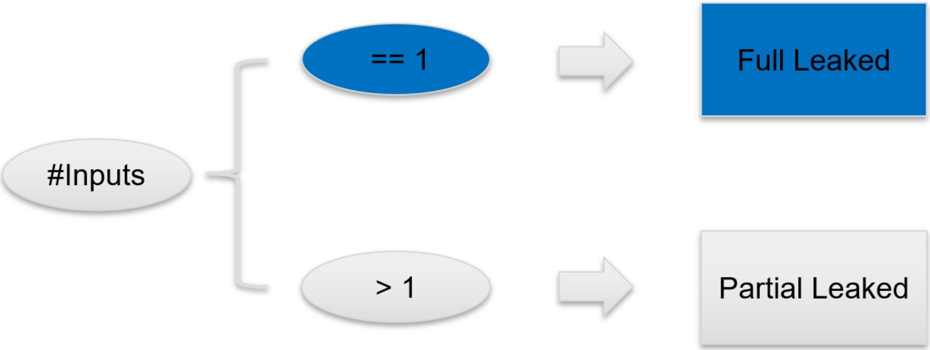
Input Space


Program
Inputs
Symbolic Execution
Constraints
Solver
#Inputs

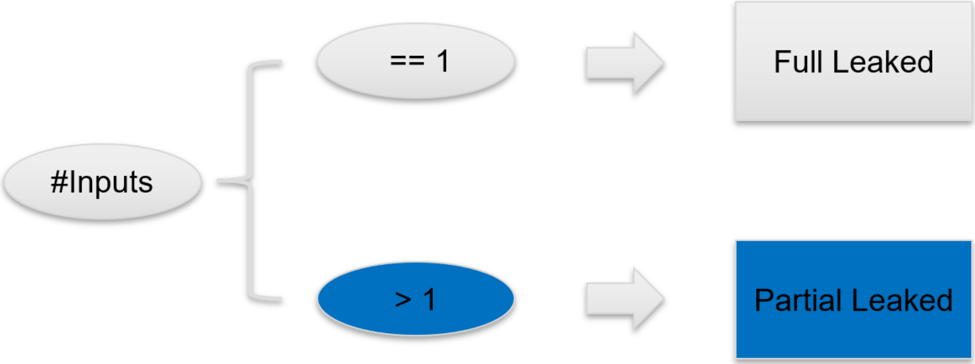
Input Space


Program
Inputs
Symbolic Execution
Constraints
Solver
#Inputs

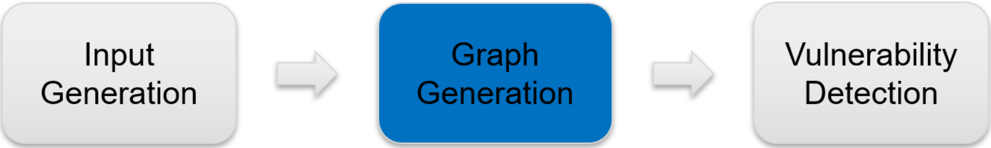
Input Space



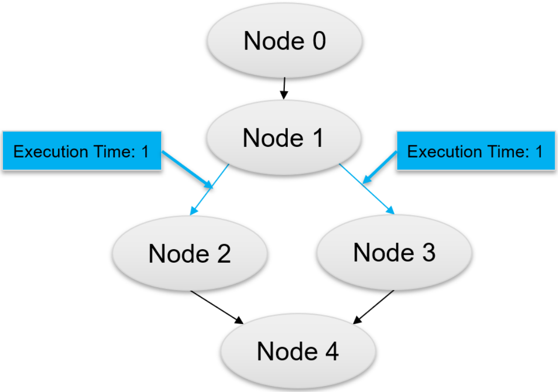
Input Space



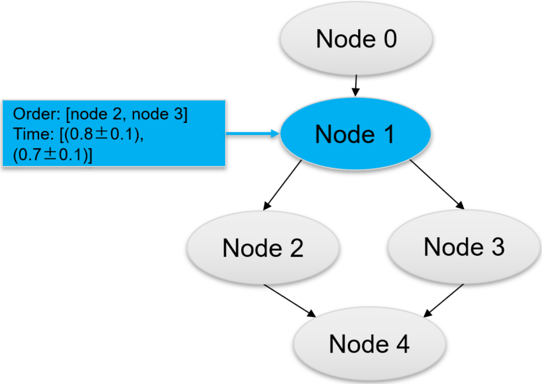
ANABLEPS


Input Generation
Graph Generation
Vulnerability Detection

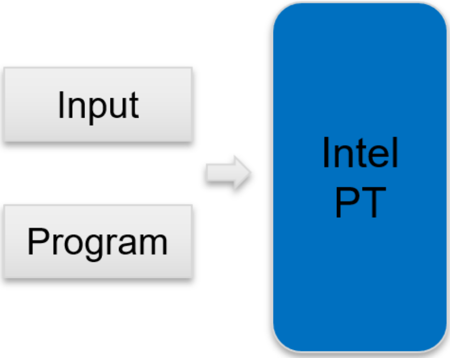
Dynamic Control-Flow Graph



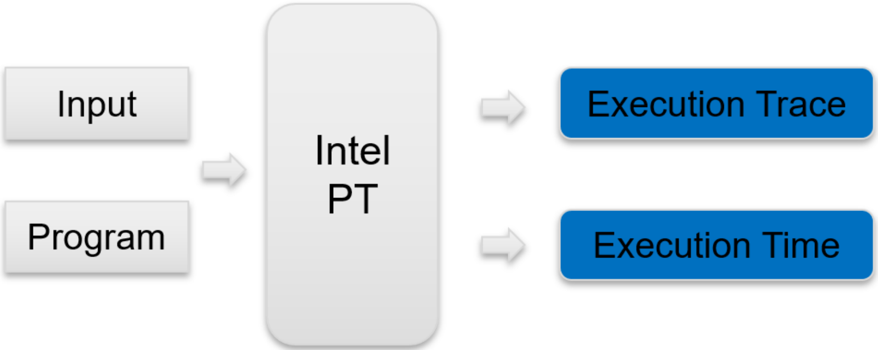
Extended Dynamic Control-Flow Graph (ED-CFG)



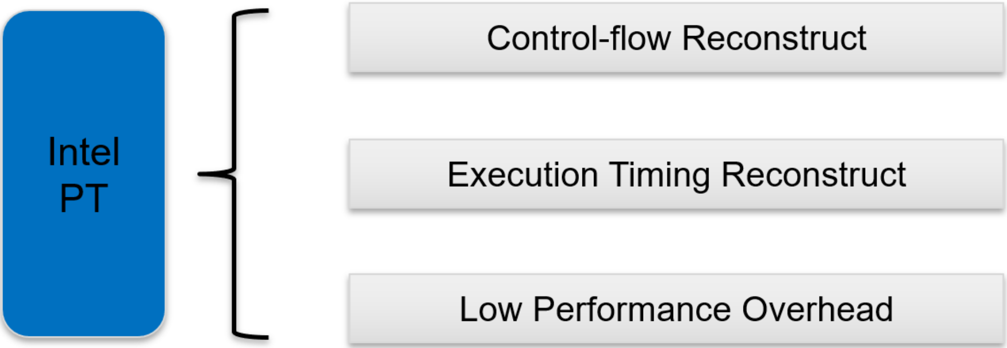
Extended Dynamic Control-Flow Graph (ED-CFG) Generation



Extended Dynamic Control-Flow Graph (ED-CFG) Generation



Extended Dynamic Control-Flow Graph (ED-CFG) Generation



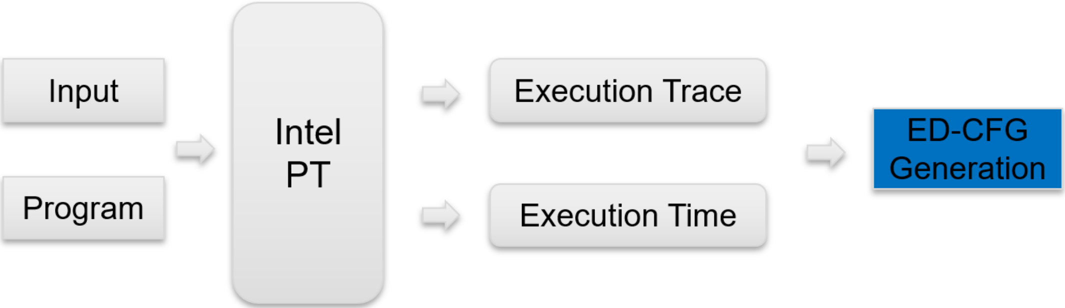
Extended Dynamic Control-Flow Graph (ED-CFG) Generation

Execution Trace	[0x400a08, 0x400cdb, 0x400ce0, ...]
Execution Time	[10, 23, 25, ...,]

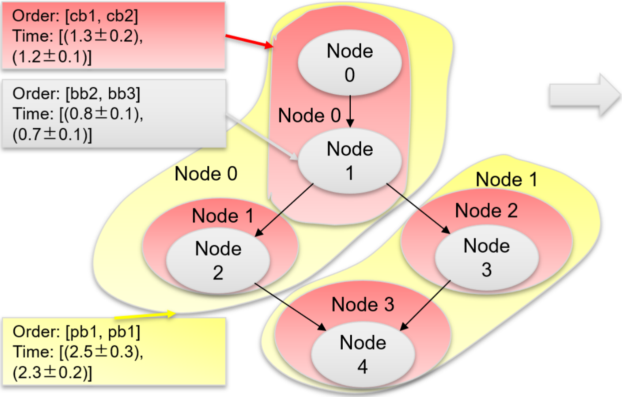
Extended Dynamic Control-Flow Graph (ED-CFG) Generation

Execution Time	[10,	23,	25,	...,]
	[11,	22,	25,	...,]
	[...	...	...	...,]
	[10,	21,	24,	...,]
	[9,	23,	25,	...,]
Mean:	[10,	12,	3,	...,]
Std:	[0.8,	0.9,	0.6,	...,]

Extended Dynamic Control-Flow Graph (ED-CFG) Generation



Extended Dynamic Control-Flow Graph (ED-CFG) Generation



Basic Block Level ED-CFG

Instruction virtual addresses

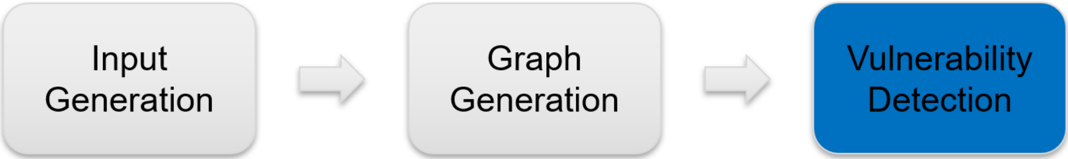
Cache Level ED-CFG

virtual address / cacheline size (64 B)

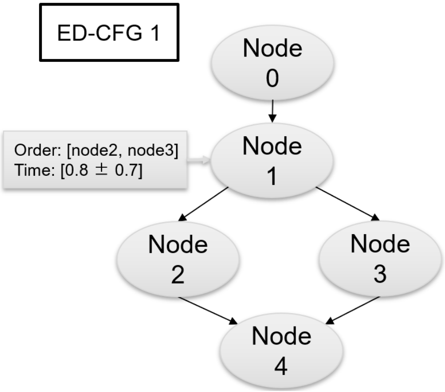
Page Level ED-CFG

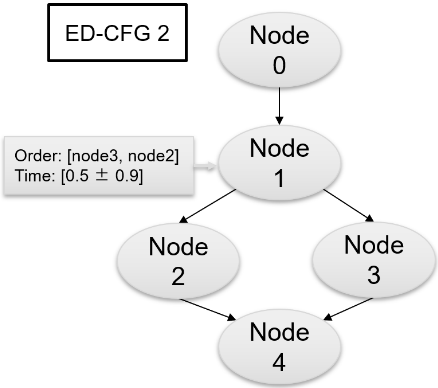
virtual address / page size (4096 B)

ANABLEPS

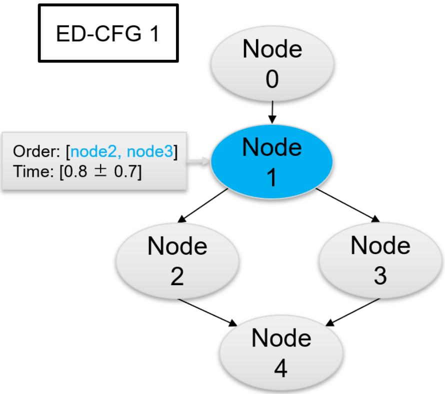

Input Generation
Graph Generation
Vulnerability Detection

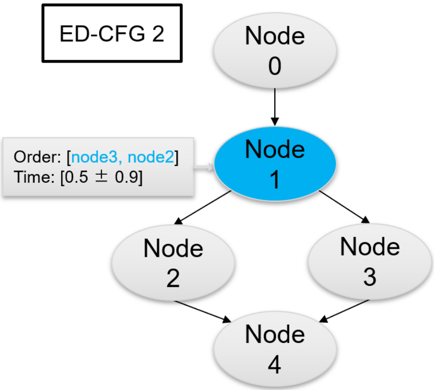
The vulnerability detection - order-based



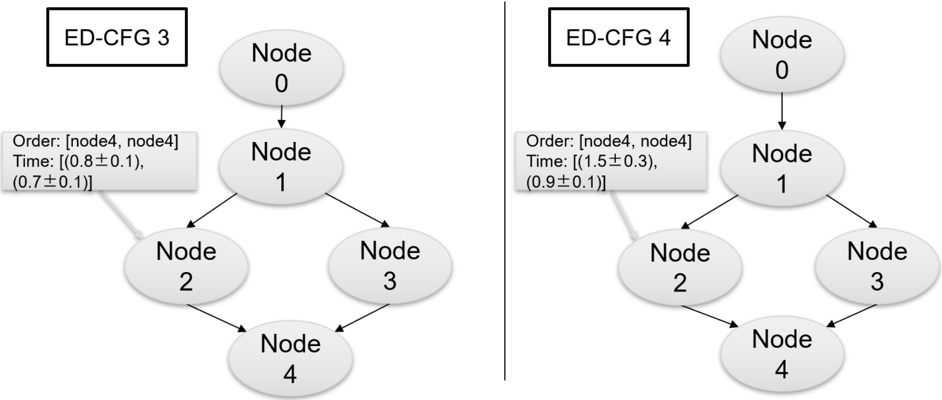


The vulnerability detection - order-based

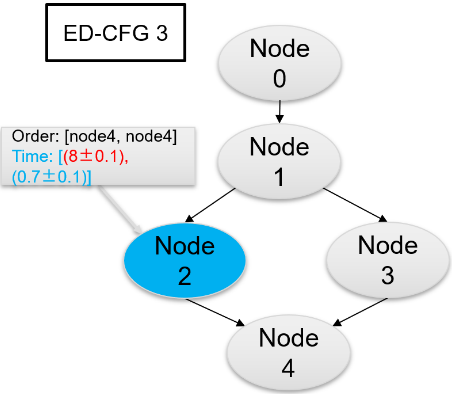


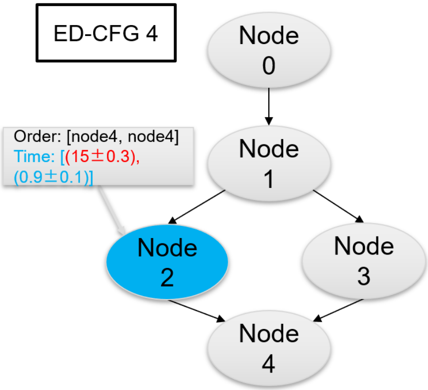


The vulnerability detection - time-based



The vulnerability detection - time-based





Evaluation

1. Detection Results
2. Case Studies

Detection Results

Programs	Functionalities Under Test	Cache Level #Nodes	Cache Level #Order-Based Vulnerable Nodes	Page Level #Nodes	Page Level #Time-Based Vulnerable Nodes
Deep Learning	dA	69	9	13	3
	SdA	109	12	22	3
	DBN	126	17	14	10
	RBM	68	8	13	7
	LogisticRegression	48	2	11	7
gsl	Sort	31	12	11	0
	Permutation	99	30	29	0
Hunspell	Spell checking	302	48	47	10
PNG	PNG Image Render	640	170	53	2
Freetype	Character Render	1054	263	82	13
Bio-rainbow	Bioinfo Clustering	214	16	24	1
QRcodegen	Generatee QR	176	32	15	3
Genometools	bed to gff3 convertion	1901	231	147	5

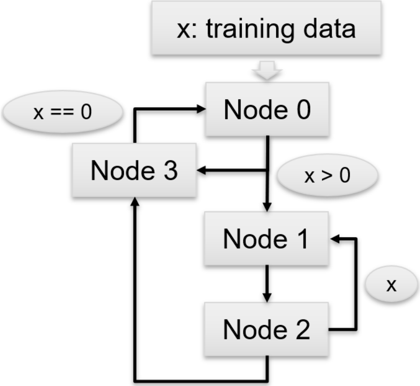
Evaluation

1. Detection Results
2. Case Studies

Detection Results - dA deep learning algorithm

Programs	Functionalities Under Test	Cache Level #Nodes	Cache Level #Order-Based Vulnerable Nodes	Page Level #Nodes	Page Level #Time-Based Vulnerable Nodes
Deep Learning	dA	69	9	13	3
	SdA	109	12	22	3
	DBN	126	17	14	10
	RBM	68	8	13	7
	LogisticRegression	48	2	11	7
gsl	Sort	31	12	11	0
	Permutation	99	30	29	0
Hunspell	Spell checking	302	48	47	10
PNG	PNG Image Render	640	170	53	2
Freetype	Character Render	1054	263	82	13
Bio-rainbow	Bioinfo Clustering	214	16	24	1
QRcodegen	Generatee QR	176	32	15	3
Genometools	bed to gff3 convertion	1901	231	147	5

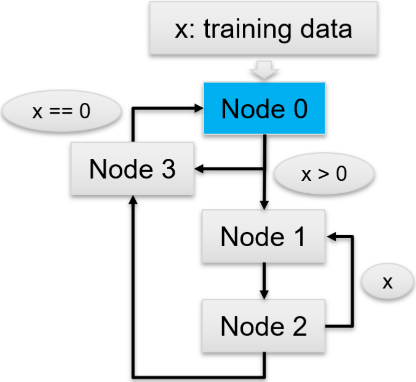
Detection Results - dA deep learning algorithm




```
int biomial*(int n, double p){
  …
  for (i=0; i<n; i++){
    r = rand() / (RAND_MAX + 1.0)
    if (r < p) c++;
  }
  ….
}

void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p){
  int i;
  for (i=0; i<this->n_visible; i++){
    if (x[i] == 0){
      tilde_x[i] = 0;
    } else {
      tilde_x[i] = binomial(x[i], p);
    }
  }
}
```

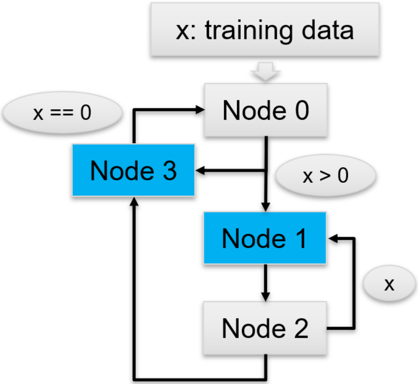
Detection Results - dA deep learning algorithm




```
int biomial*(int n, double p){
 …
 for (i=0; i<n; i++){
   r = rand() / (RAND_MAX + 1.0)
   if (r < p) c++;
 }
 ….
}

void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p){
 int i;
 for (i=0; i<this->n_visible; i++){
   if (x[i] == 0){
     tilde_x[i] = 0;
   } else {
     tilde_x[i] = binomial(x[i], p);
   }
 }
}
```

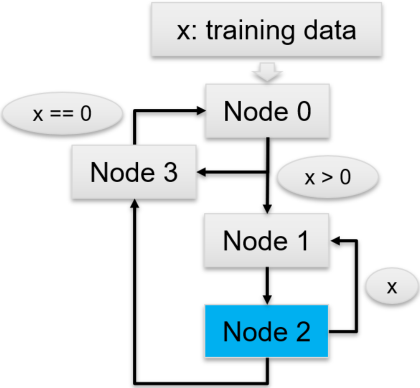
Detection Results - dA deep learning algorithm




```
int biomial*(int n, double p){
  ...
  for (i=0; i<n; i++){
    r = rand() / (RAND_MAX + 1.0)
    if (r < p) c++;
  }
  ....
}

void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p){
  int i;
  for (i=0; i<this->n_visible; i++){
    if (x[i] == 0){
      tilde_x[i] = 0;
    } else {
      tilde_x[i] = binomial(x[i], p);
    }
  }
}
```

Detection Results - dA deep learning algorithm



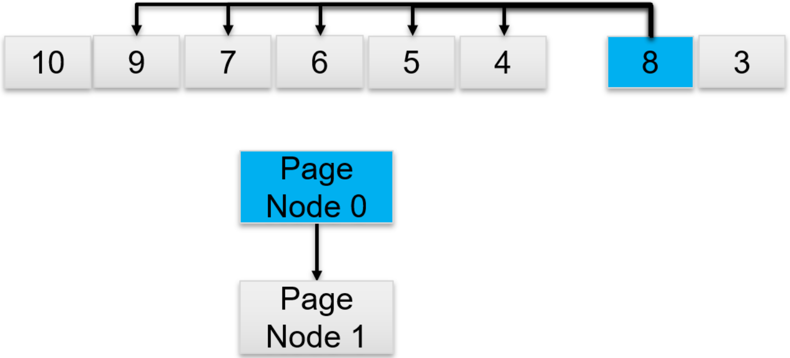

```
int biomial*(int n, double p){
 …
 for (i=0; i<n; i++){
   r = rand() / (RAND_MAX + 1.0)
   if (r < p) c++;
 }
 ….
}

void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p){
 int i;
 for (i=0; i<this->n_visible; i++){
   if (x[i] == 0){
     tilde_x[i] = 0;
   } else {
     tilde_x[i] = binomial(x[i], p);
   }
 }
}
```

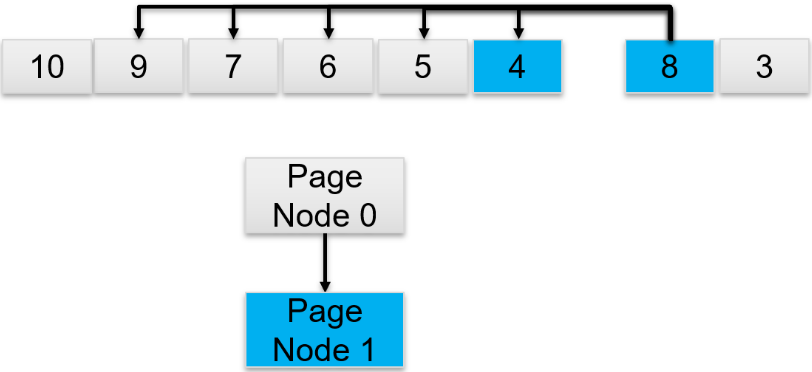
Detection Results - Sorting algorithm

Programs	Functionalities Under Test	Cache Level #Nodes	Cache Level #Order-Based Vulnerable Nodes	Page Level #Nodes	Page Level #Time-Based Vulnerable Nodes
Deep Learning	dA	69	9	13	3
	SdA	109	12	22	3
	DBN	126	17	14	10
	RBM	68	8	13	7
	LogisticRegression	48	2	11	7
gsl	Sort	31	12	11	0
	Permutation	99	30	29	0
Hunspell	Spell checking	302	48	47	10
PNG	PNG Image Render	640	170	53	2
Freetype	Character Render	1054	263	82	13
Bio-rainbow	Bioinfo Clustering	214	16	24	1
QRcodegen	Generatee QR	176	32	15	3
Genometools	bed to gff3 convertion	1901	231	147	5

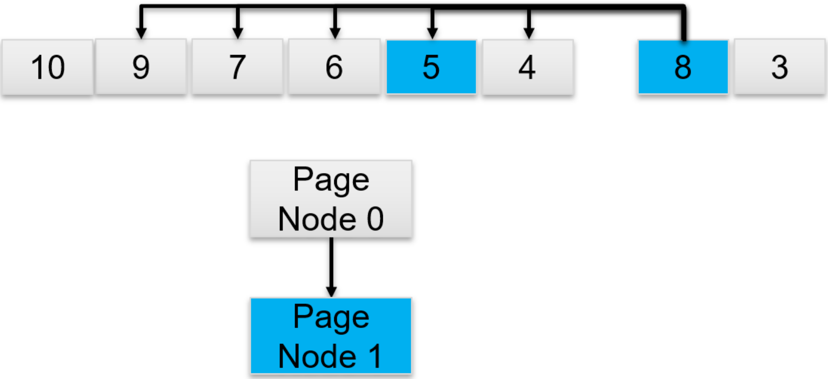
Detection Results - Sorting algorithm



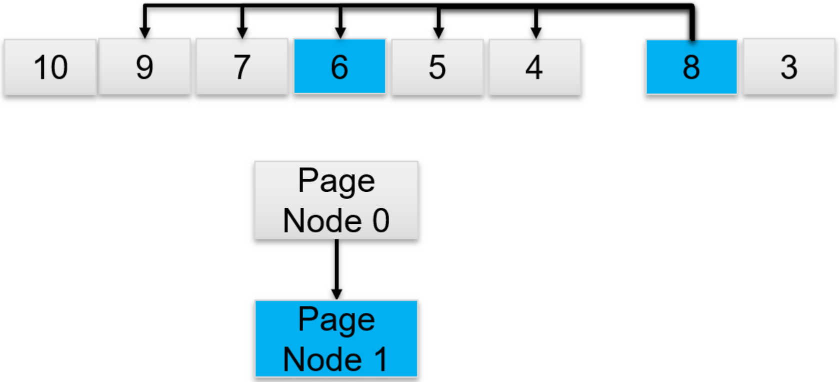
Detection Results - Sorting algorithm



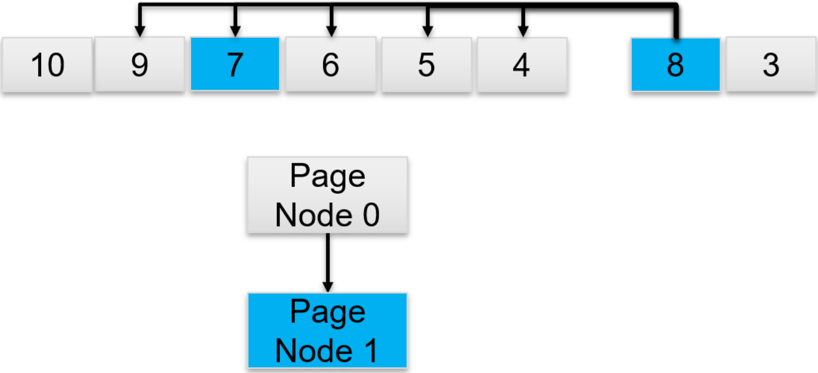
Detection Results - Sorting algorithm



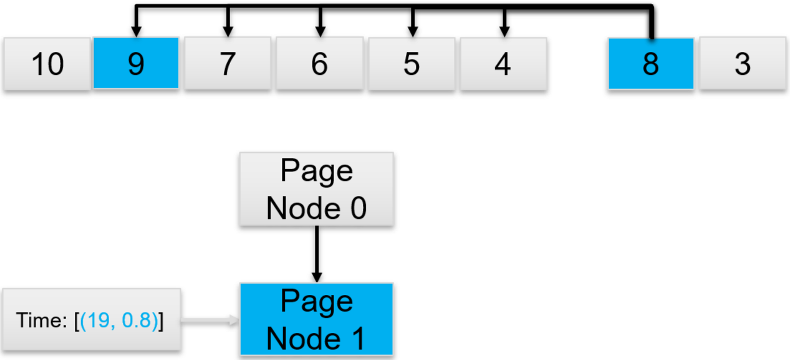
Detection Results - Sorting algorithm



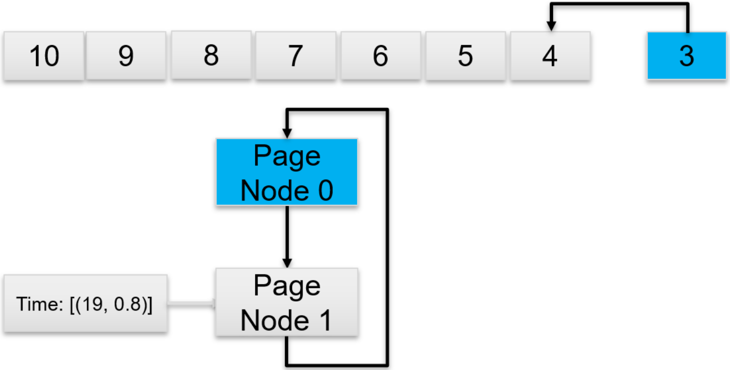
Detection Results - Sorting algorithm



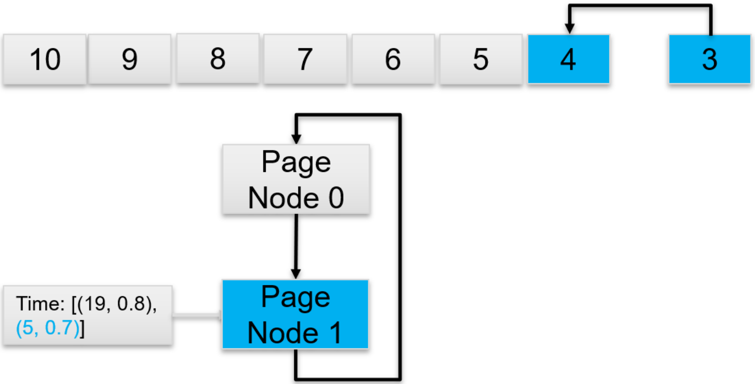
Detection Results - Sorting algorithm



Detection Results - Sorting algorithm



Detection Results - Sorting algorithm



Related Work

1. **Stacco: Differentially Analyzing Side-Channel Traces for Detecting SSL/TLS Vulnerabilities in Secure Enclaves.**
   Yuan Xiao, Mengyuan Li, Sanchuan Cheng, and Yinqian Zhang
2. **MicroWalk: A Framework for Finding Side Channels in Binaries.**
   Jan Wichelmann, Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar
3. **DATA Differential Address Trace Analysis: Finding Address-based Side-Channels in Binaries.**
   Samuel Weiser, Andreas Zankl, Raphael Spreitzer, Katja Miller, Stefan Mangard, and Georg Sigl
4. **CacheD: Identifying Cache-Based Timing Channels in Production Software.**
   Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu

Conclusion

1. **New insights:** With the time information, attacker could get more secret data than only order information.
2. **New methods:** Use the fuzzing and symbolic execution to generate inputs and quantify the leakage is a new attempt.
3. **New tools:** ANABLEPS is an automatically program analysis tool, and will be released to the community. `github.com/OSUSecLab/ANABLEPS`

Thank You

Time and Order: Towards Automatically Identifying Side-Channel Vulnerabilities in Enclave Binaries

Wubing Wang, Yinqian Zhang, and Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

RAID 2019