Controlling Your Control Flow Graph

Arun Kanuparthi, Jeyavijayan Rajendran†, Ramesh Karri‡

Intel Corporation, The University of Texas at Dallas†, New York University‡

Disclaimer

The views expressed in this presentation solely belong to the authors and do not in anyway reflect the views of Intel Corporation. The countermeasures described here were implemented in experimental hardware (implemented using cycle accurate simulators) and software environments. The authors of this article have not explored the potential applicability of these countermeasures to commercially available hardware and software.

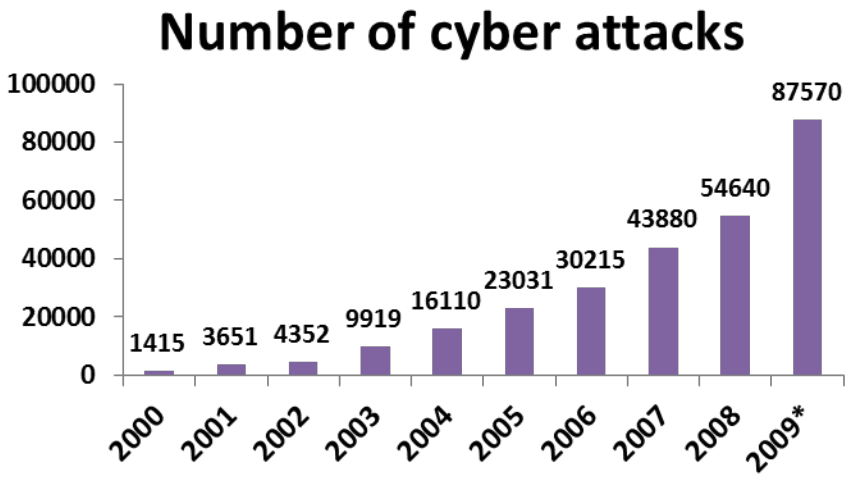
Outline

- Introduction
- Motivation & Threat Model
- Dynamic Sequence Checking (DSC)
- Architecting DSC
- Evaluation
- Security Analysis
- Summary

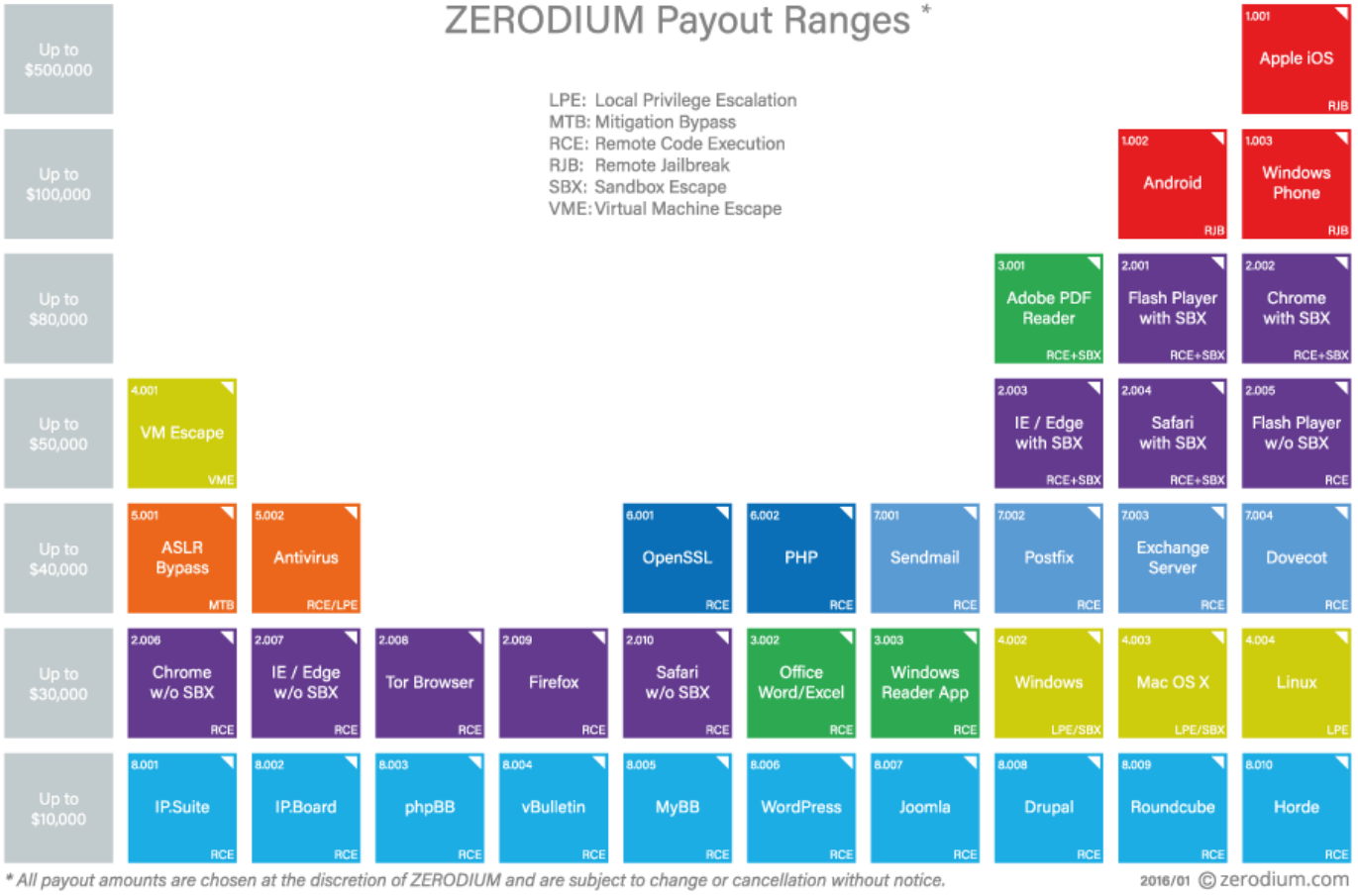
Introduction

- Attackers steal sensitive data such as bank account numbers, passwords, SSN, medical records, etc.
- With the increase in the number of smart devices, the number of cyber attacks is on the rise

Problem is only getting worse!



Motivation – For Attackers



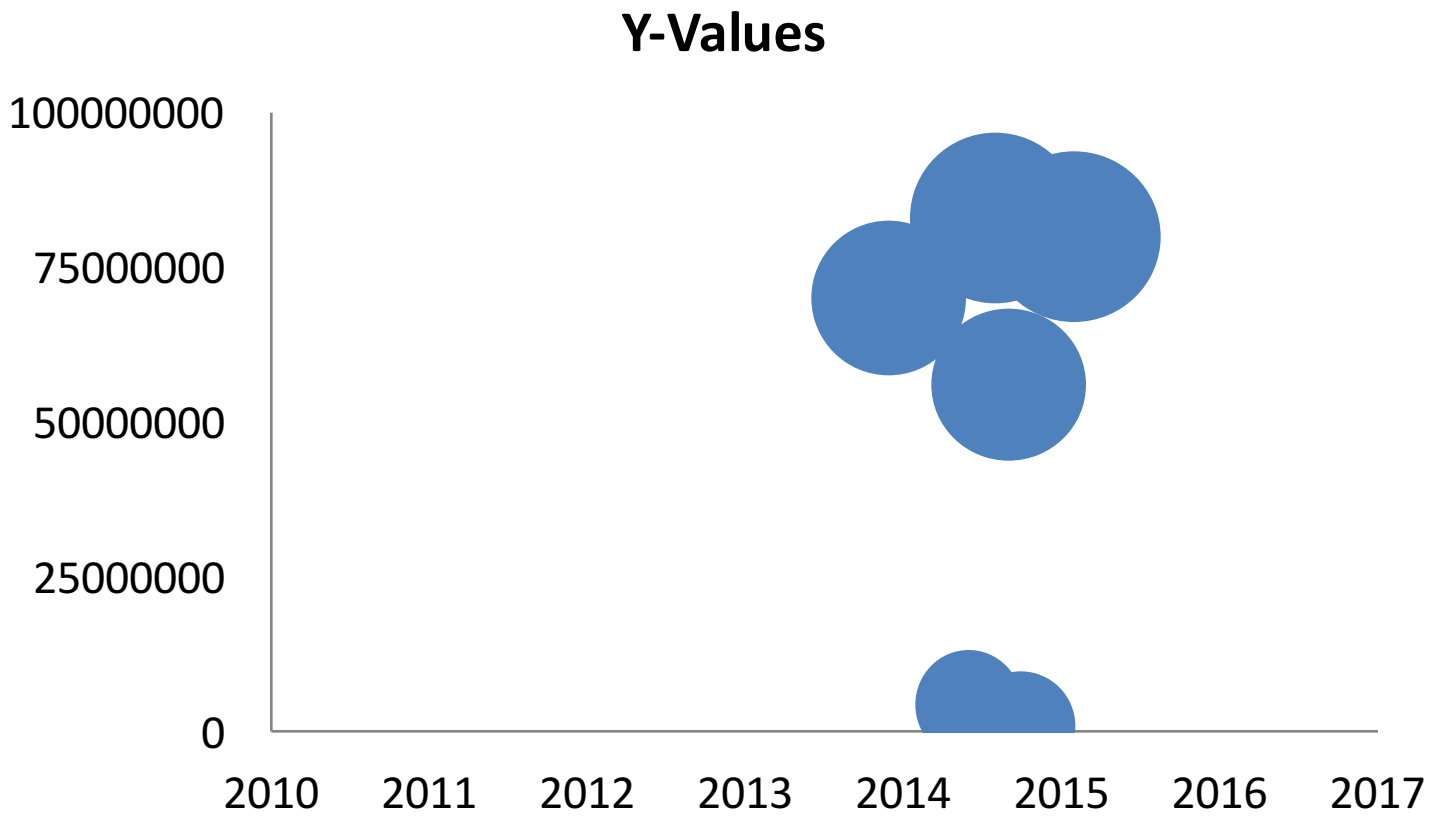
Bug bounties and black markets offer premium money for vulnerabilities

Attackers may also choose to exploit vulnerabilities themselves

Motivation- Incidents

- Heart bleed (2014)
- Target attack (2013)
- Security vulnerabilities in iPhone (2011)
- Sony PlayStation Network hack (2011)
- Medical records stolen in Utah (2011)
- US Department of Defense incident (2008)

Incidents


Y-Values
100000000
75000000
50000000
25000000
0
2010
2011
2012
2013
2014
2015
2016
2017

Code Reuse Attacks

Example code:

Binary	Assembly code
f7 c7 07 00 00 00	test $0x00000007, (%edi)
0f 95 45 c3	setnzb -61(%ebp)

Starting one byte later:

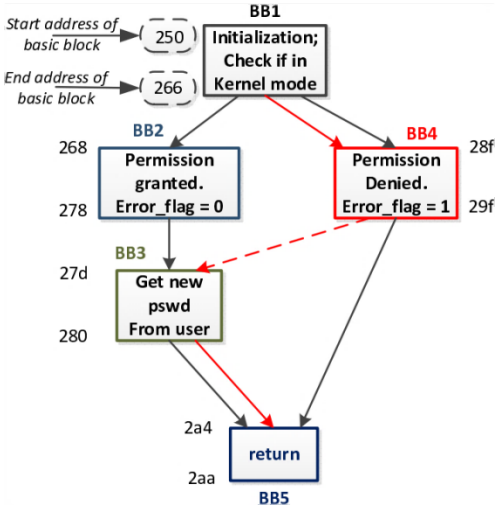
Binary	Assembly code
c7 07 00 00 00 0f	movl $0x0f000000, (%edi)
95 45 c3	xchg %ebp, %eax
45	inc %ebp
c3	ret

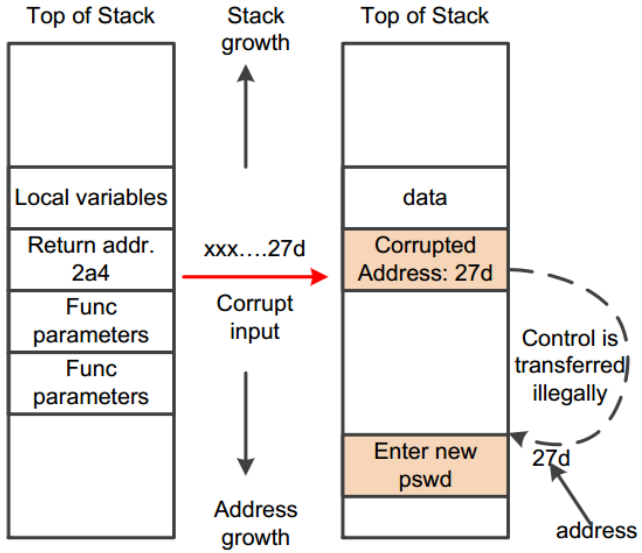
Example of Return Oriented Programming

Example attack

```
char cur_pswd[20];
scanf("%d", &cur_pswd);
if(kernel_mode){
  scanf("%d", &new_pswd);
  pswd = new_pswd;
  error_flag=0;
}
else {
  error_flag=0;
  printf("\nNo Permission!!");
}
```

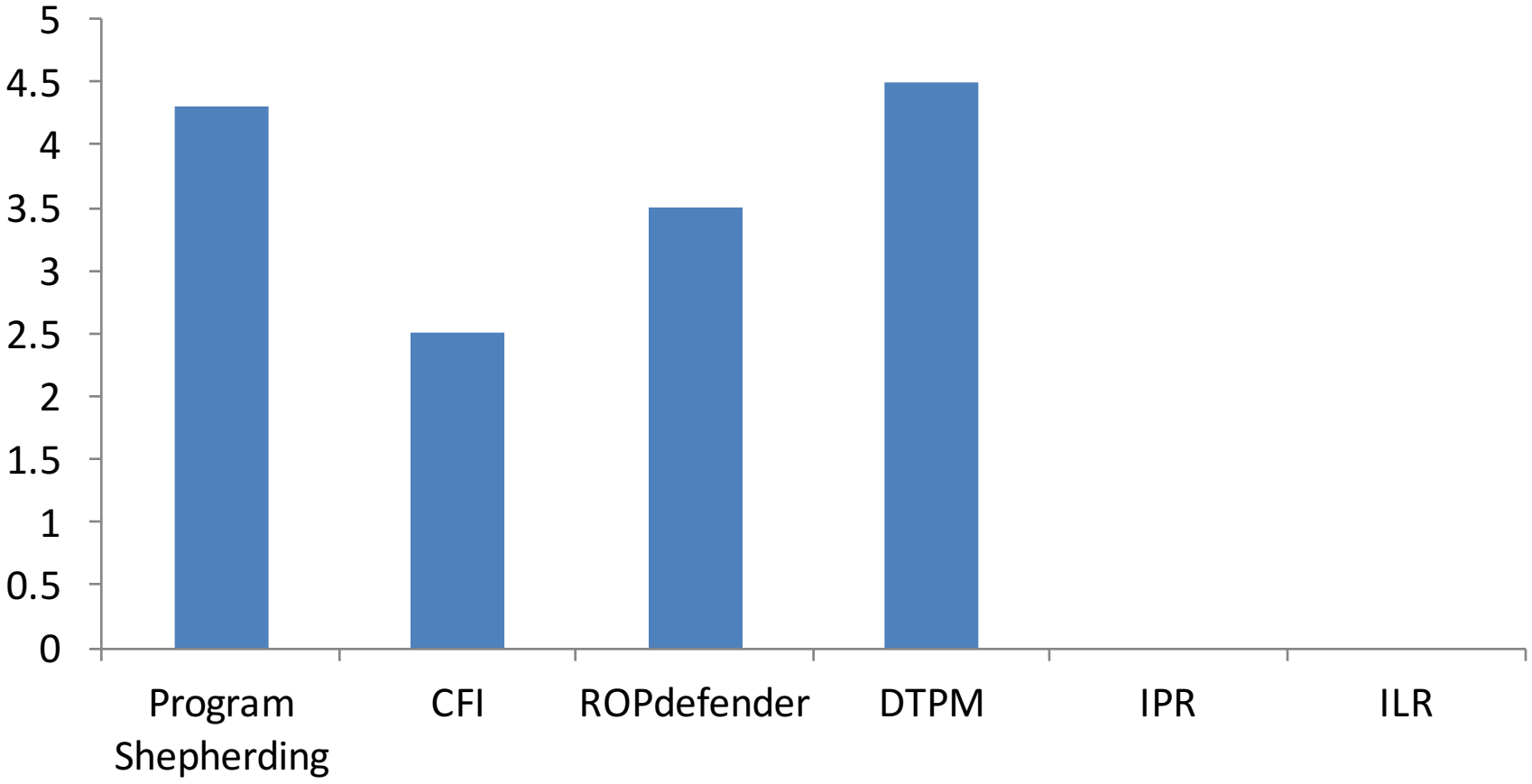
```
BB1  8048250  push %ebp                ;Initialize
     8048251  mov  %esp, %ebp
                    .
                    .
     8048261  cmpl $0x0, 0x1c(%esp)   ; Check if in
     8048266  je   804828f            ; kernel mode.
BB2  8048268  mov  $0x80a7688, %eax   ; If in kernel
     804826d  lea  0x10(%esp), %edx   ; mode, get
     8048271  mov  %edx, 0x4(%esp)    ; new_passwd
     8048275  mov  %eax, (%esp)       ; from user
     8048278  call 8048dc0            ; call 'scanf'
BB3  804827d  mov  0x10(%esp), %eax   ; update
     8048281  mov  %eax, 0x14(%esp)   ; old pswd
     8048285  movl $0x0, 0x18(%esp)   ; with new
     804828d  jmp  80482a4            ; pswd
BB4  804828f  movl $0x1, 0x18(%esp)   ; If not,
     8048297  mov  0x80a768b, %eax    ; display
     804829c  mov  %eax, (%ebp)       ; message
     8048289  call 8048d90            ; call 'printf'
BB5  80482a4  mov  $0x80a768b, %eax
     80482a9  leave
     80482aa  ret                     ; return
```





Motivation – For Defenders

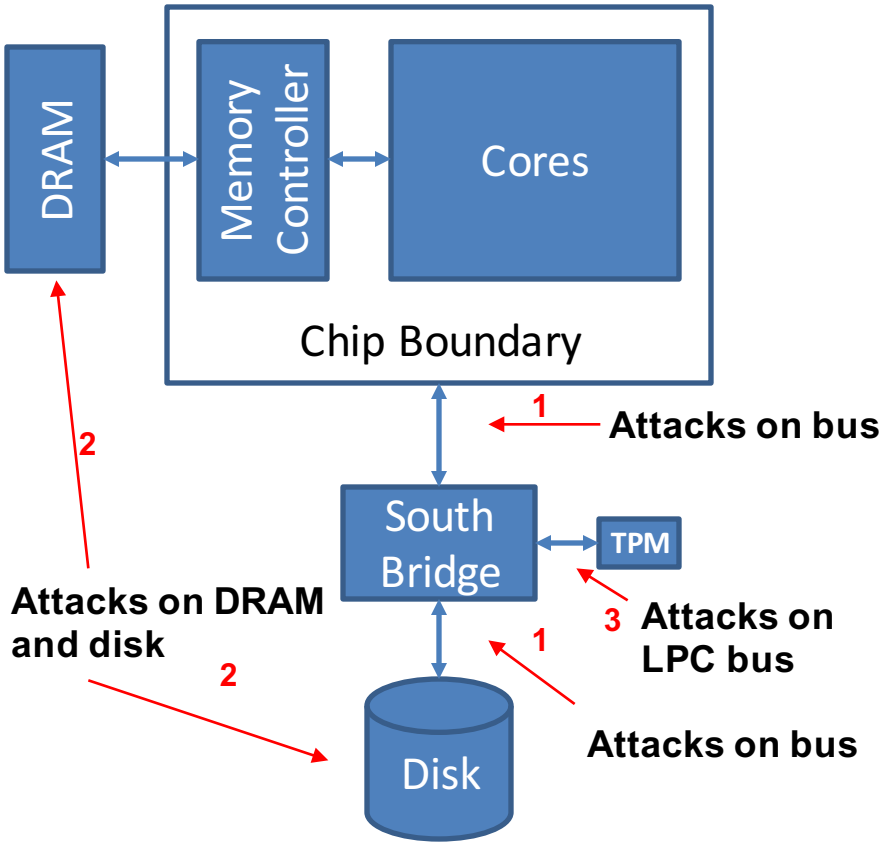
Cyber threat actors continue to exploit vulnerable software to conduct attacks. **As many as 85 percent of targeted attacks are preventable**.*



*http://www.publicsafety.gc.ca/cnt/ntnl-scrt/cbr-scrt/tp-strtgs-eng.aspx

Threat Model

- **Goal of the attacker**: *Tamper with program execution and/or obtain protected information*
- Attacker can:
  - Tamper with the external buses (arrows 1 and 3)
  - Modify the contents of disk and DRAM (arrow 2)
- Attacker cannot:
  - Observe the internal states of the processor
  - Tamper with the interconnects internal to the processor



Dynamic Sequence Checking (DSC)

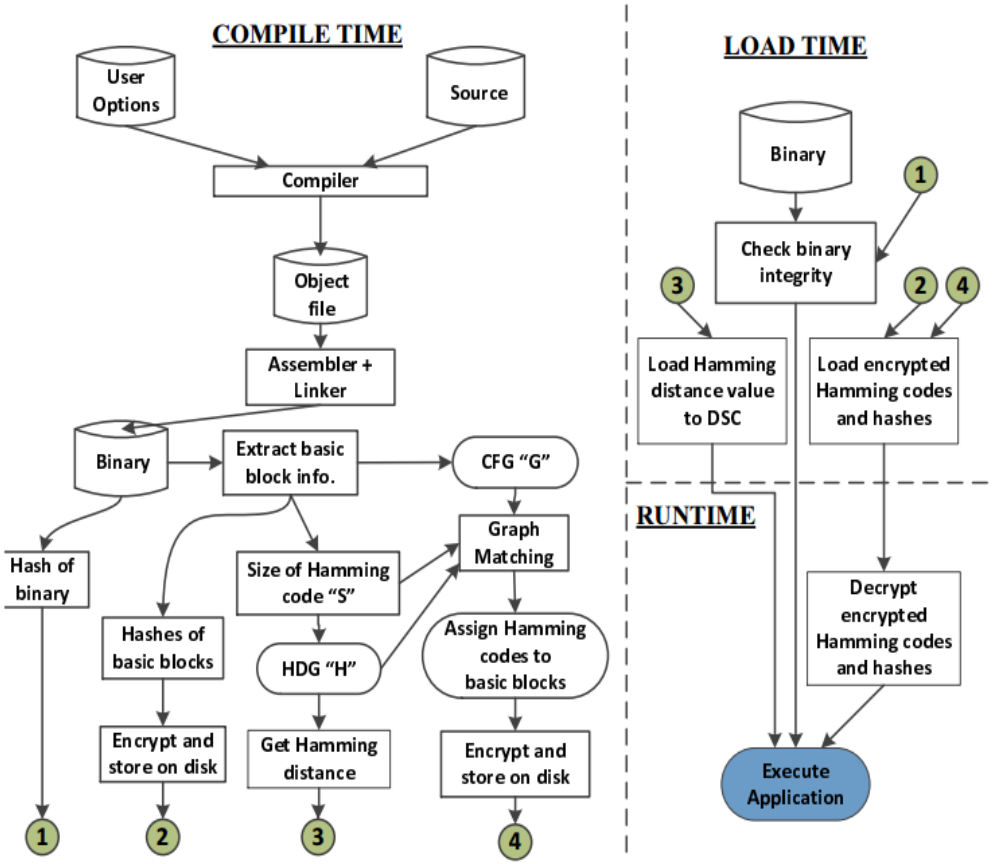
- Key Ideas
  - What:
    - Verify validity of control flow between basic blocks
  - How:
    - Assign unique codes to every basic block in the program at compile time such that Hamming distance between any two legally connected blocks is a known constant
    - Verify the Hamming distance source basic block and destination basic block at runtime

Why DSC works

Benchmark	Basic Blocks (N)	Possible transitions (E') x 10^6	Actual transitions (E)	% Actual transitions (%E)
400.Perlbench	20149	202.98	29111	0.0143
401.bzip2	2508	3.14	3008	0.0956
403.Gcc	51224	1311.92	68142	0.0052
462.Libquantum	1648	1.35	2037	0.1501
482.Xalancbmk	24165	291.96	31266	0.0107
003.Clustalw	2554	3.26	2920	0.0896
006.Phylip	1745	1.52	2124	0.1396
STREAM	1248	0.77	1473	0.1893

DSC Design Flow

- Size of Hamming Code:
  - $S = \max(C, P, ceil(logN) + 1)$
- Determination of Hamming distance
  - HD ε (1, ceil(HD/2))
- Match Control Flow Graph with Hamming distance graph
  - Use Karp Sisper graph matching algorithm



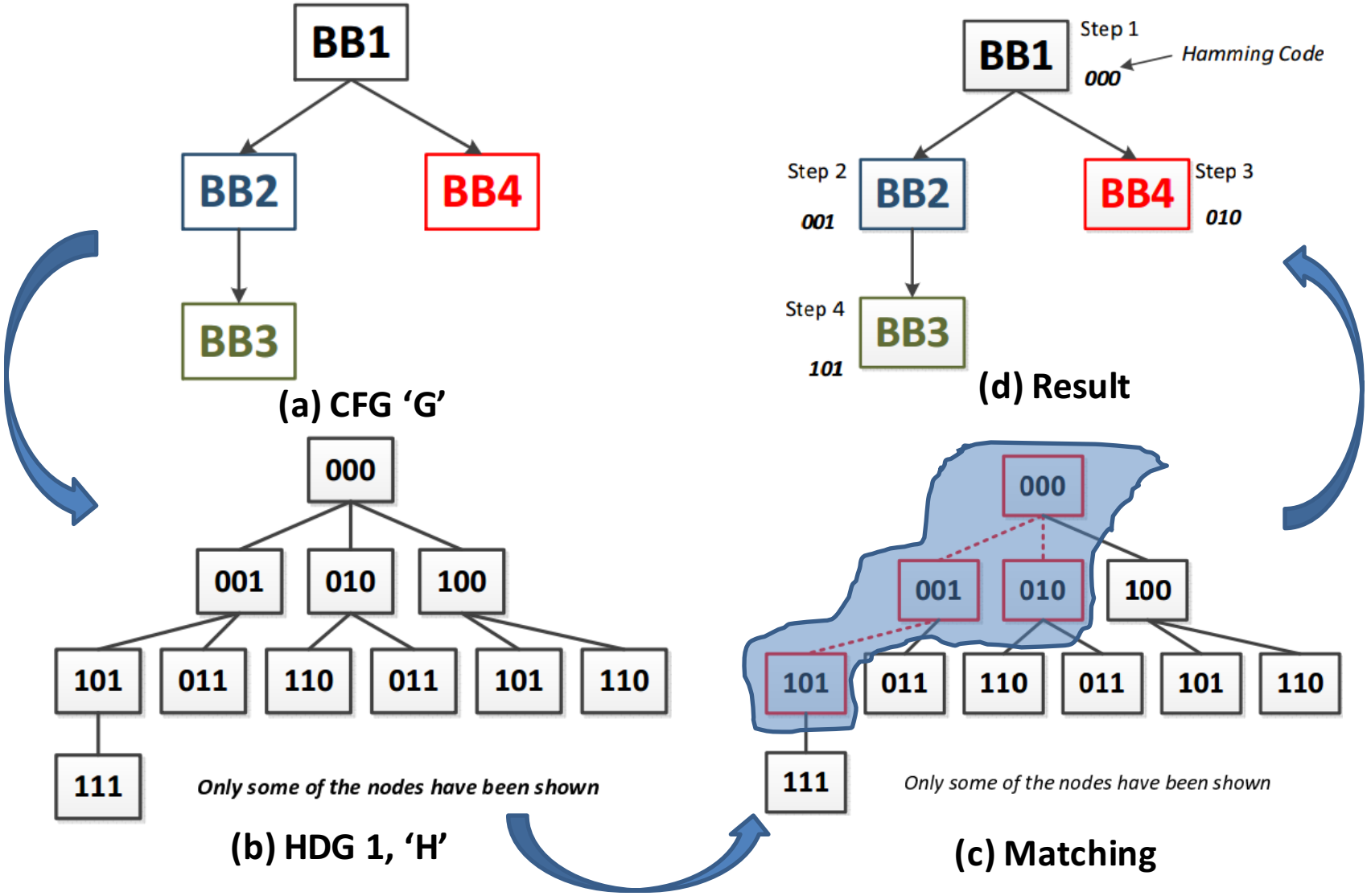
Characteristics of children and parents in benchmarks

Benchmark	Children	Parents	ceil (log N)
400.Perlbench	27	24	15
401.bzip2	7	9	12
403.Gcc	37	31	16
462.Libquantum	9	9	11
482.Xalancbmk	19	22	15
003.Clustalw	6	8	12
006.Phylip	8	9	11
STREAM	3	3	11

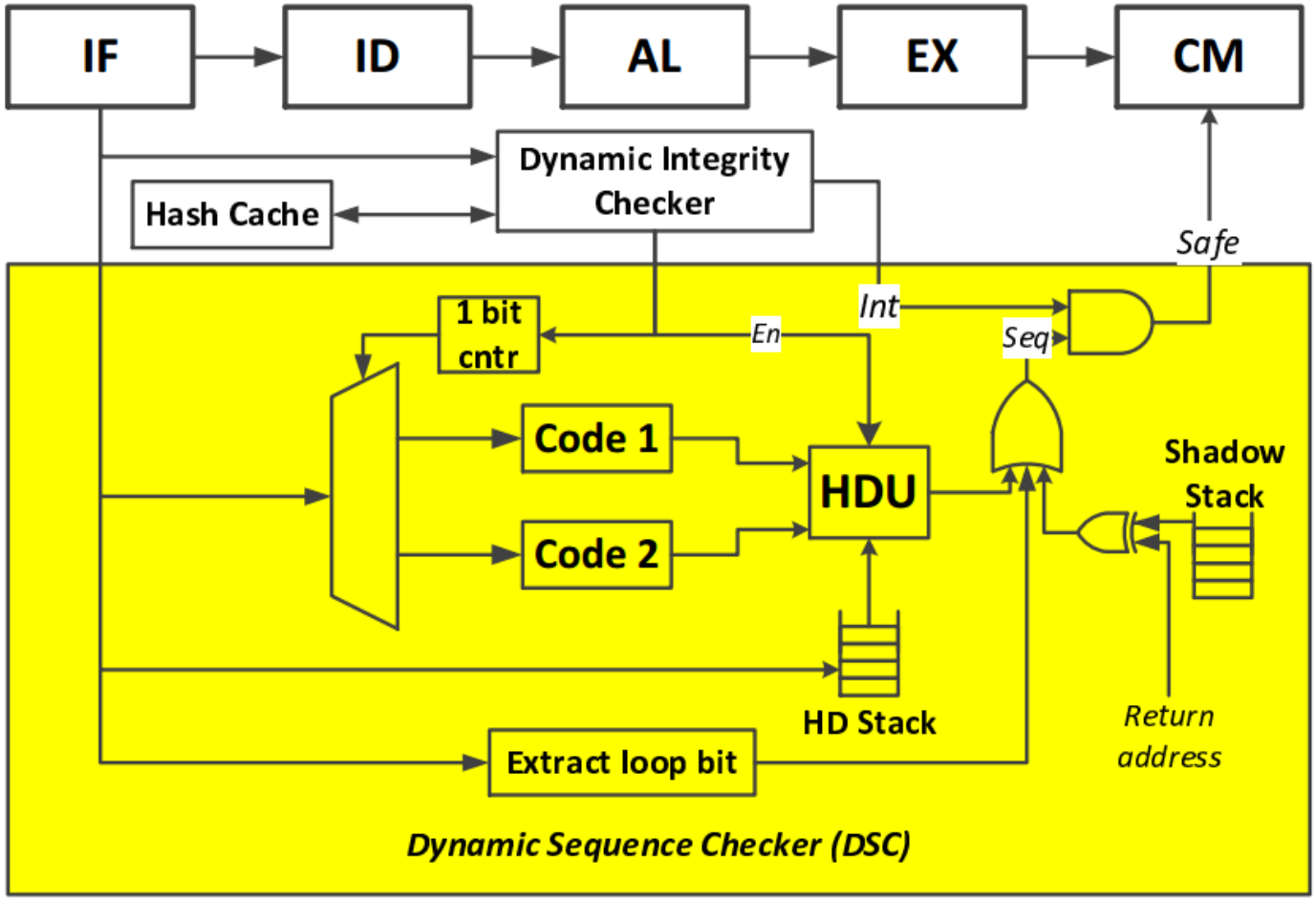
Algorithm for assigning unique codes

1: HD $\leftarrow$ 1;
2: Root node of G is matched with root node of H ;
3: **while** *G still has nodes left* **do**
    3.1: Search for G within H;
    3.2: Begin with leftmost child in current level in G and assign leftmost code in same level in H;
    3.3: Remove edge between parent node and child node and update G;
end
4: **if** $G \not\subseteq H$ **then**
    4.1: HD $\leftarrow$ HD + 1;
    4.2: **if** $HD < \lceil S/2 \rceil$ **then**
        4.2.1: Go to step 2;
    else
        4.2.2: S $\leftarrow$ S + 1;
        4.2.3: Go to step 2;
    end
else
end

Assigning unique codes to basic blocks



DSC Microarchitecture



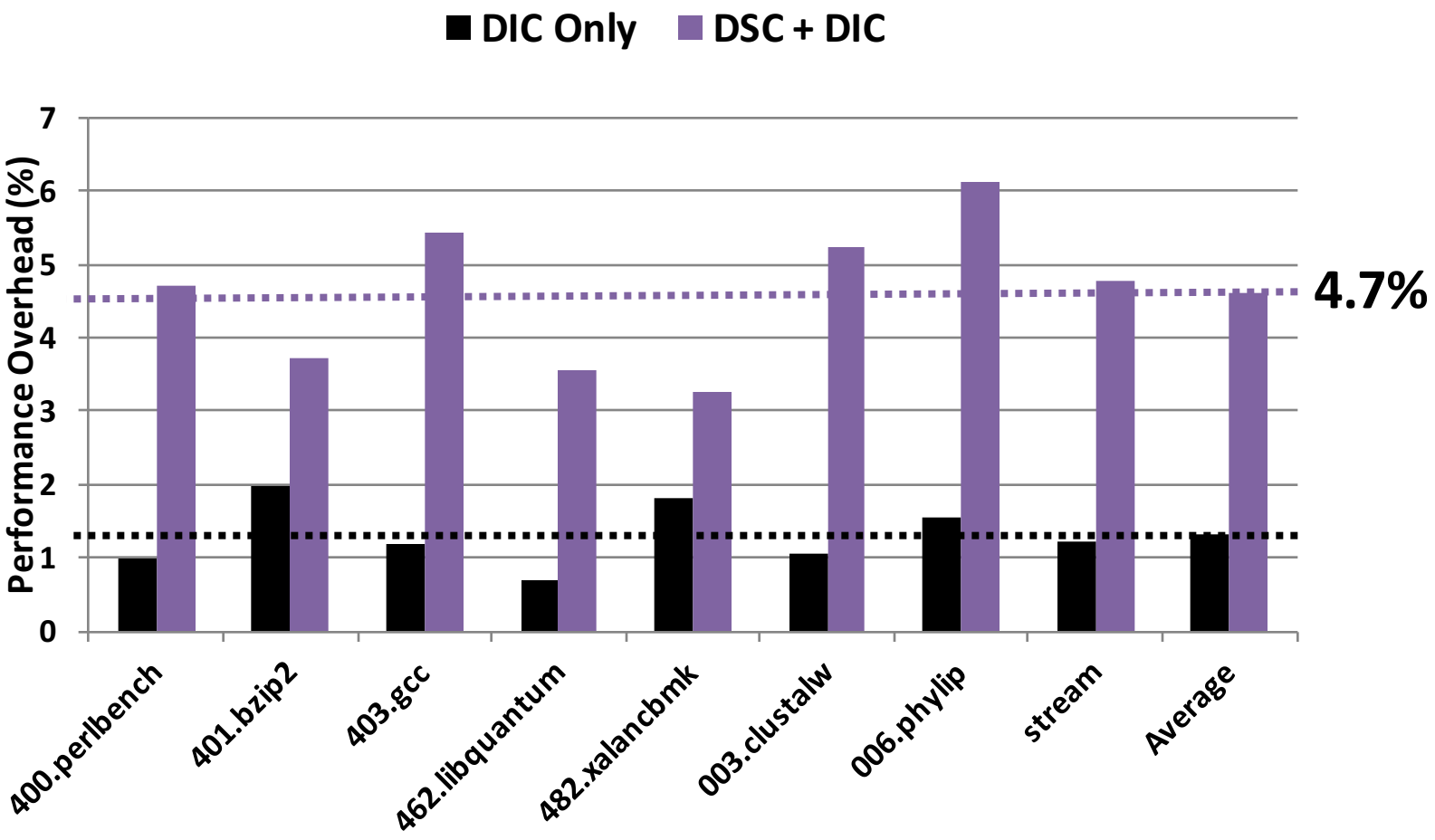
Experimental Setup

Parameter	Specification
Core	2 GHz
L1-I$, D$	32 KB 4-way, 4 cycles
L2	8 cycles
L3	20 cycles
Main Memory round trip latency	50 cycles
SHA-3 Hash Engine	12 cycles
Comparator	1 cycle
Hash Cache	64 KB 8-way, 6 cycles
Hamming Distance calculation	1 cycle
HD stack	
Shadow stack	
Benchmarks	SPEC CPU2006, BioBench, STREAM

Results

Benchmark	Code Size (bits)	Hamming Distance (HD)	Storage (G + H) in MB	Storage (codes) in KB
400.Perlbench	28	2	64	68.75
401.bzip2	13	2	8	4
403.Gcc	38	2	112	237.5
462.Libquantum	12	2	2	2.5
482.Xalancbmk	23	2	56	67
003.Clustalw	13	2	8	4
006.Phylip	12	2	2	2.5
STREAM	12	2	2	1.8

Results



Performance Impact of DSC

Resiliency of DSC against ROP gadgets

Benchmark	ret instructions	Unintended ret instructions	Gadgets thwarted	% detected
400.Perlbench	3292	1915	51/51	100
401.bzip2	971	549	34/34	100
403.Gcc	6794	4668	60/60	100
462.Libquantum	1006	554	44/44	100
482.Xalancbmk	17495	10213	68/68	100
003.Clustalw	1258	631	46/46	100
006.Phylip	1921	871	42/42	100
STREAM	779	324	28/28	100

Security Analysis

- DSC can protect against
  - Control transfer to the middle of an instruction
    - No unique code to block in the middle of instruction
  - Control transfer to the middle of a basic block
    - No unique code to the block in the middle of basic block
  - Code reuse attacks using return/ jump
    - Hamming distance does not match
  - Indirect jumps/ branches
    - Hamming distance does not match
  - Indirect Calls
    - Hamming distance does not match
- Limitations of DSC
  - Attacker redirects code to basic blocks with similar Hamming distance

Conclusion

- Microarchitecture support to ensure control-flow integrity
- Hamming-distance based approach to thwart code reuse attacks
- All gadgets identified by ROPgadget for various benchmarks have been detected
- Very low performance overhead (4.7%)

Thank You

BACKUP