Smokestack:Thwarting DOP Attacks with Runtime Stack Layout Randomization

Misiker Aga, *Todd Austin* (University of Michigan)

CGO'19

EECS 583

Yunjie Pan, Shibo Chen, Cheng Chi, Yifan Guan

11-20-2019

SmokeStack

- Motivation & Background
- Methodology
- Implementation
- Evaluation

Data-Oriented Programming (DOP) Attack

- A new form of memory exploits
- Only rely on manipulation of local variables
- Does not attack on control-flow elements

Example

Modeled after FTP server

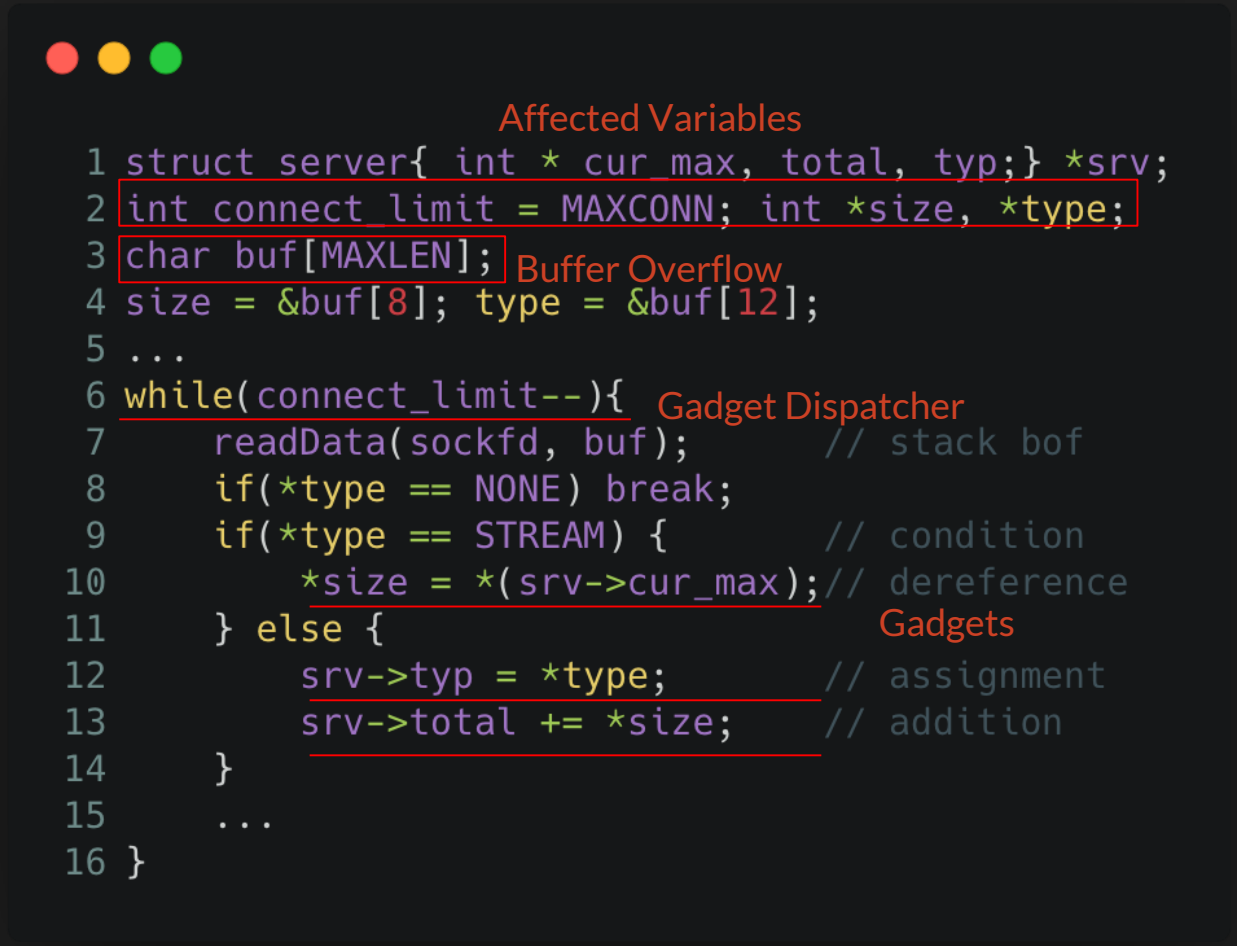
Uses a stack buffer overflow vulnerability to control a few stack variables

```
1 struct server{ int * cur_max, total, typ;} *srv;
2 int connect_limit = MAXCONN; int *size, *type;
3 char buf[MAXLEN];
4 size = &buf[8]; type = &buf[12];
5 ...
6 while(connect_limit--){
7     readData(sockfd, buf);       // stack bof
8     if(*type == NONE) break;
9     if(*type == STREAM) {        // condition
10        *size = *(srv->cur_max);// dereference
11    } else {
12        srv->typ = *type;        // assignment
13        srv->total += *size;     // addition
14    }
15    ...
16 }
```

Example

Modeled after FTP server

Uses a stack buffer overflow vulnerability to control a few stack variables



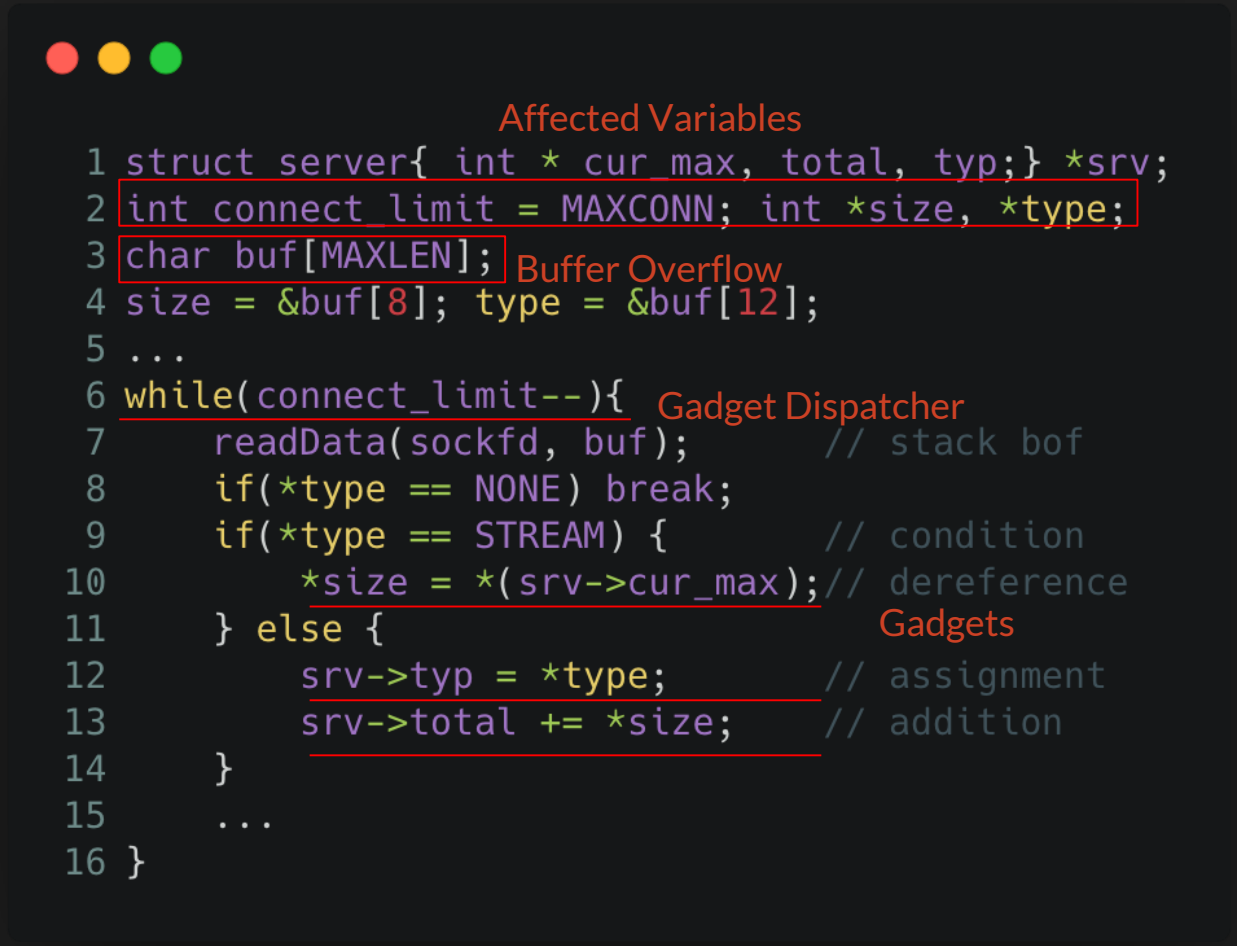
Data-Oriented Programming (DOP) Attack

- Evaluated on 9 real-world programs
- 8 out of 9 have gadgets to **simulate arbitrary computations**
- 2 out of 9 are confirmed **Turing-complete**
- Built 3 attacks that bypass randomization defense
  - Immune to ASLR (only need relative address)
  - Immune to DEP (reuse existing code)
  - Immune to Stack Canary (does not write past stack frames)
- Open source software especially vulnerable

Example

Modeled after FTP server

Uses a stack buffer overflow vulnerability to control a few stack variables

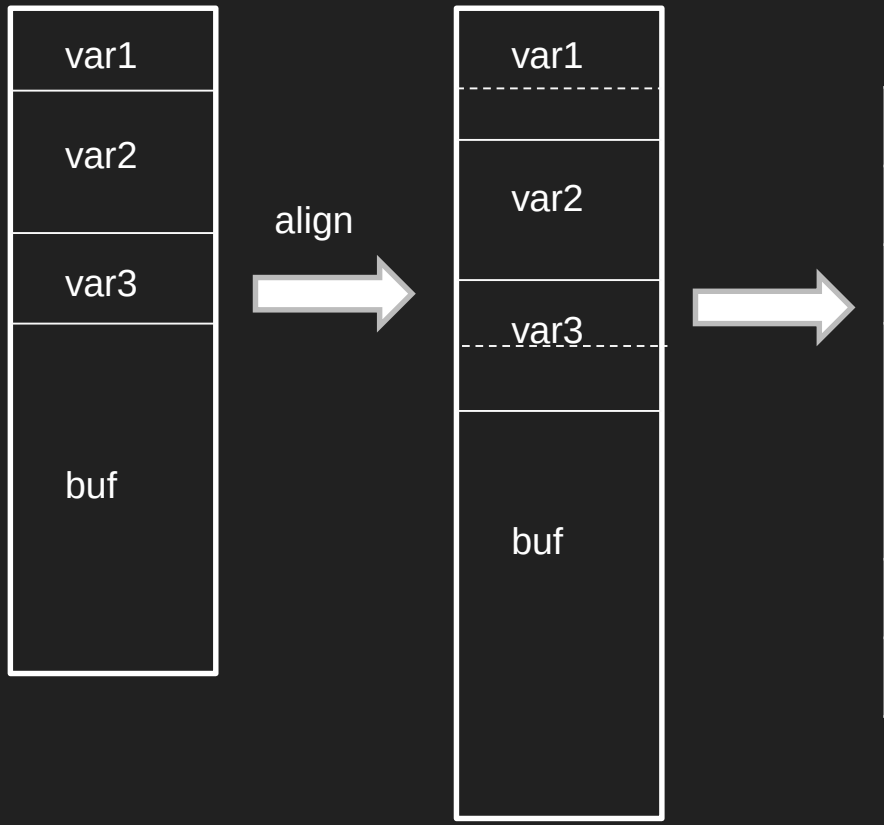


SmokeStack

- Motivation & Background
- Methodology
- Implementation
- Evaluation

Compile time

```
int var1;
double var2;
int var3;
double buf[3];
```



P-BOX

ind	var1	var2	var3	buf
0	1	2	3	4
1	1	3	2	4
……				
N-1	4	5	0	1
N	…	…	…	…

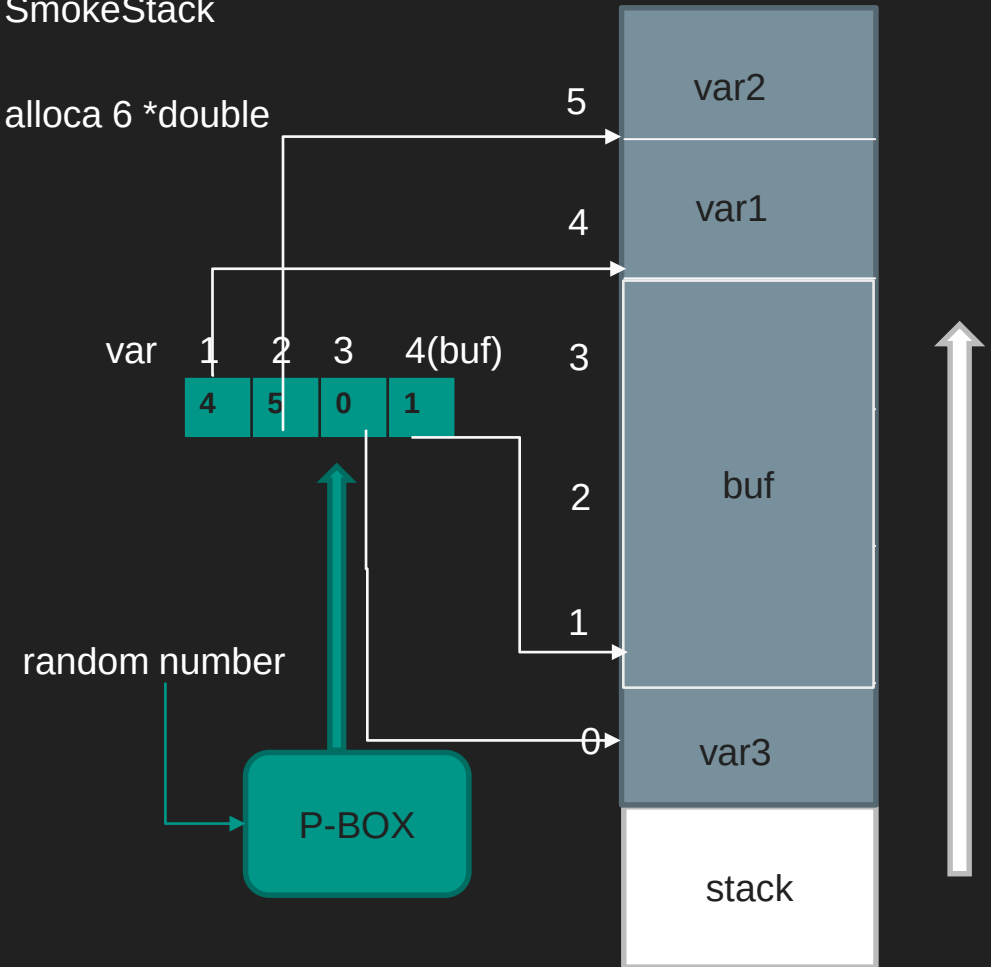
Run time

original code

alloca i32
alloca double
alloca i32
alloca 3 * double

SmokeStack

alloca 6 *double



SmokeStack

- Motivation & Background
- Methodology
- Implementation
- Evaluation

Implementation

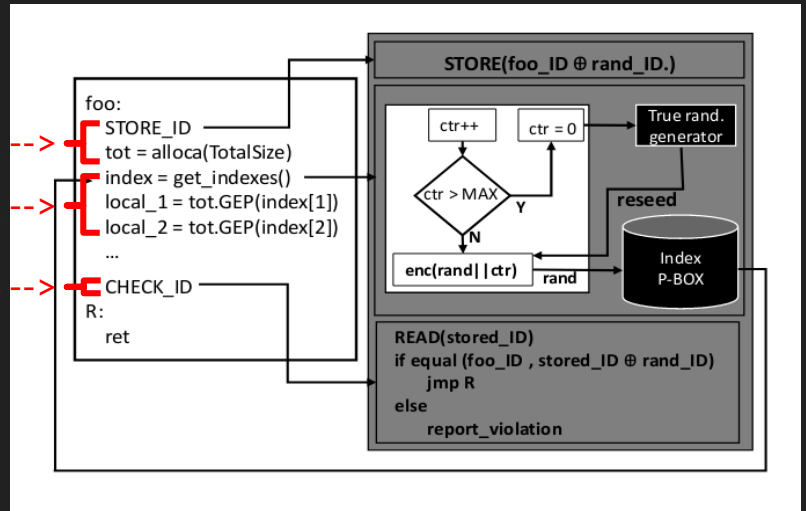
1. Analysis Passes

(1) Gather all stack allocation
(2) Generate P-BOX table for each function
(3) Generate P-BOX for all unique P-BOX entry tables

Implementation

2. Instrumentation Passes

(1) Insert allocation and call to a random number generator function
(2) Replace alloca instr with getelementptr instr
(3) Detect attacks that bypass stack allocation instrumentation



Optimization

Objective: Have low performance and memory overhead

Optimization

Objective: Have low performance and memory overhead

Approach:

- P-BOX size power of 2 -> reduce performance overhead
- Rearranging Stack Allocations -> reduce memory overhead
- Rounding up Allocations -> reduce memory overhead

SmokeStack

- Motivation & Background
- Methodology
- Implementation
- Evaluation

Experiment Setup

Platform:

Xeon D-1541 processor, Ubuntu 16.04 Linux, 32GB of memory

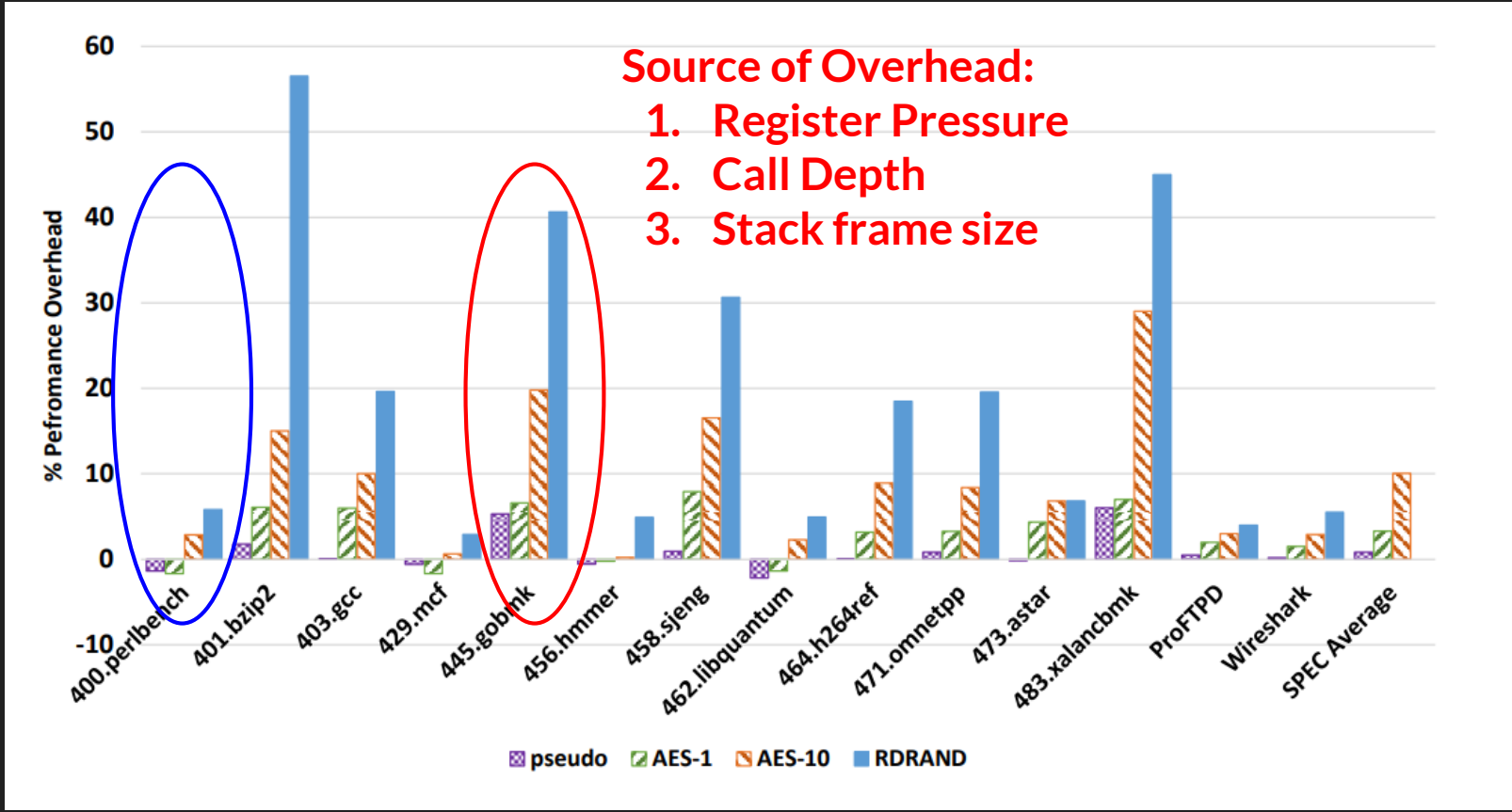
Benchmarks:

SPEC 2006 benchmarks & I/O bound real world applications, e.g., ProFTPD, Wireshark

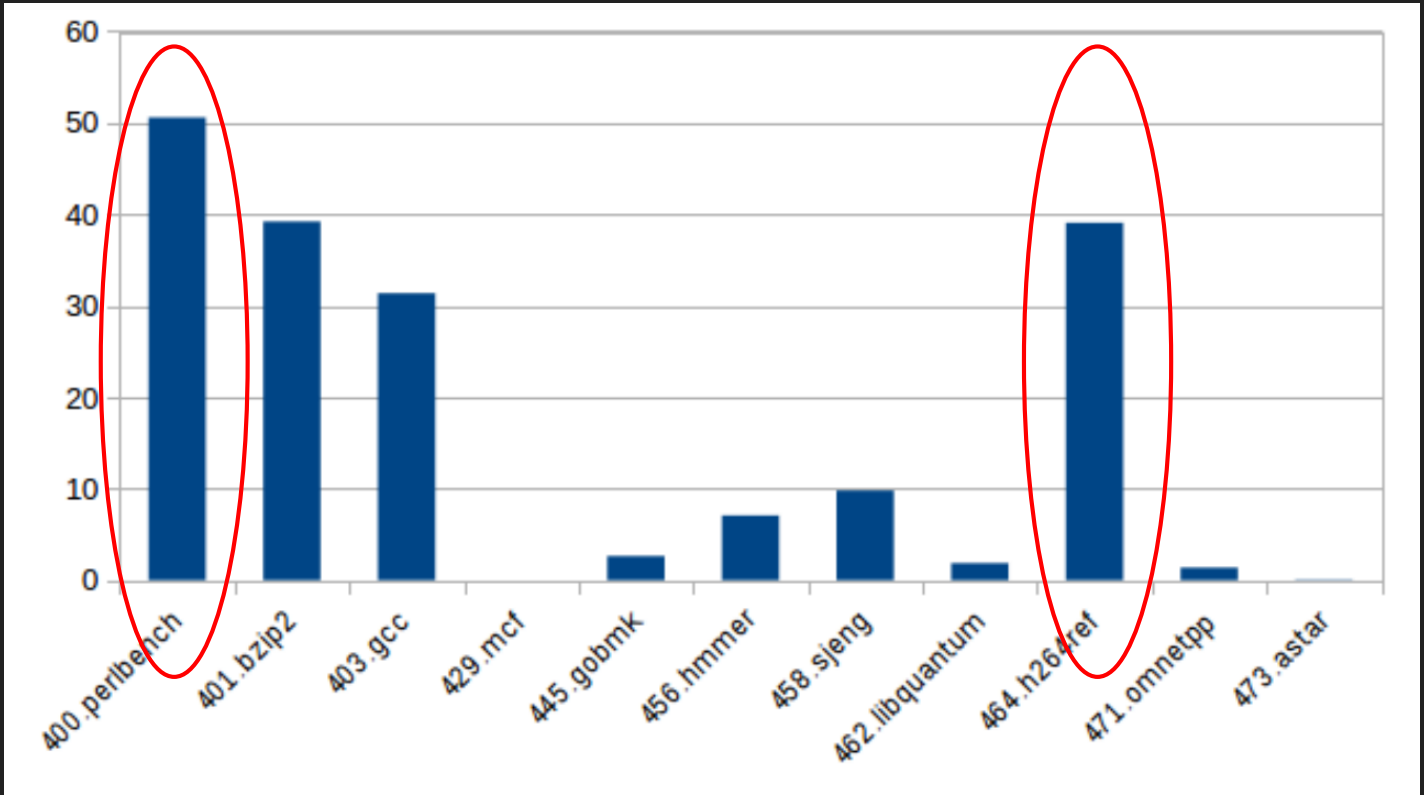
Source of random numbers:

RDRAND, AES-NI

Performance Overhead



Memory Overhead



Security Analysis

Attacks considered:

Attacks in the original paper

RIPE Suite - Control Flow Attack Benchmark Suite

Able to stop them all!

Stop real world attacks:

CVE-2014-2299 in Wireshark:

CVE-2006-5815 in ProFTPD

A stack-based buffer overflow vulnerabilities

Discussion

Strength:

1. Thorough Security Analysis: Protect Against All Known DOP attacks
2. Minimal Overhead if Using PRNG as the Source of Randomness

Weakness:

1. Large Execution Overhead if Using RDRAND
2. Limited Analysis on Source of Overhead & Efficiency of the Optimizations

Question?