PMP: Cost-effective Forced Execution with Probabilistic Memory Pre-planning

Wei You, Zhuo Zhang, Yonghwi Kwon, Yousra Aafer, Fei Peng, Yu Shi, Carson Harmon, Xiangyu Zhang

PURDUE UNIVERSITY

UNIVERSITY of VIRGINIA

Background

- Difficulties in Malware Behavior Analysis
  - the needed environment or setup may not present
  - recent malware makes use of time-bomb and logic-bomb to hide payload
  - sophisticated malware even use cloaking technique to anti-analysis

- Forced Execution
  - penetrate malware self-protection mechanisms and various trigger conditions
  - works by force-setting branch outcomes of some conditional instructions
  - challenge: maintain *crash-free* execution

X-Force v.s. PMP

- X-Force: heavy-weight
  - track individual instructions
  - reason about pointer alias relations on-the-fly
  - repair invalid pointers by on-demand memory allocation

- PMP: light-weight
  - no tracking individual instructions
  - no on-demand memory allocation and pointer repair
  - pre-allocate a large memory buffer
  - fill the buffer and variables with carefully crafted random values before execution

Example

takes the false branch

is forced to take the true branch

```c
typedef struct{double *f1; long *f2;} T;
void foo() {
  long **a = malloc(...);
  T *b;
  if (cond1()) init(b);
  if (cond2()) {
    long *c = b->f2;
    *(b->f2) = **a; // [0x0008] = [0x0010]
    *(b->f1) = 0.1; // [0xffd0] = 0.1
    long tmp = *c;
  }
}
```

Local Variables

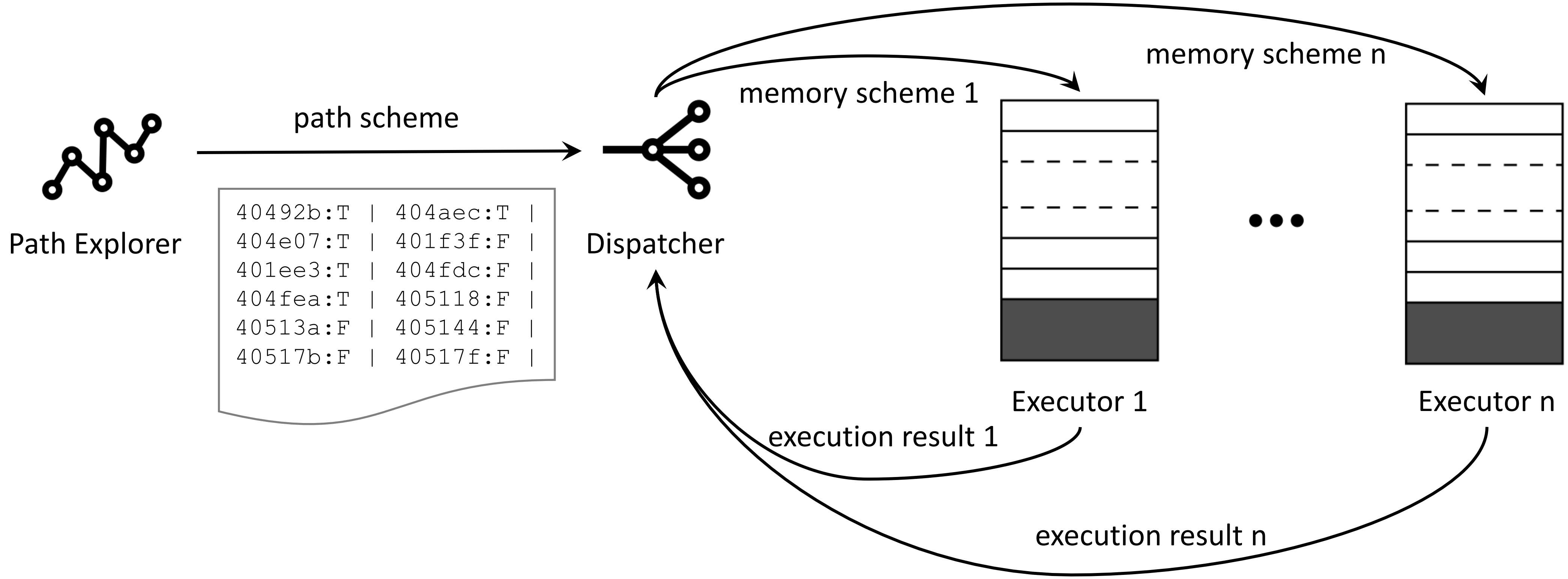
c: 0x08
b: 0x20

b

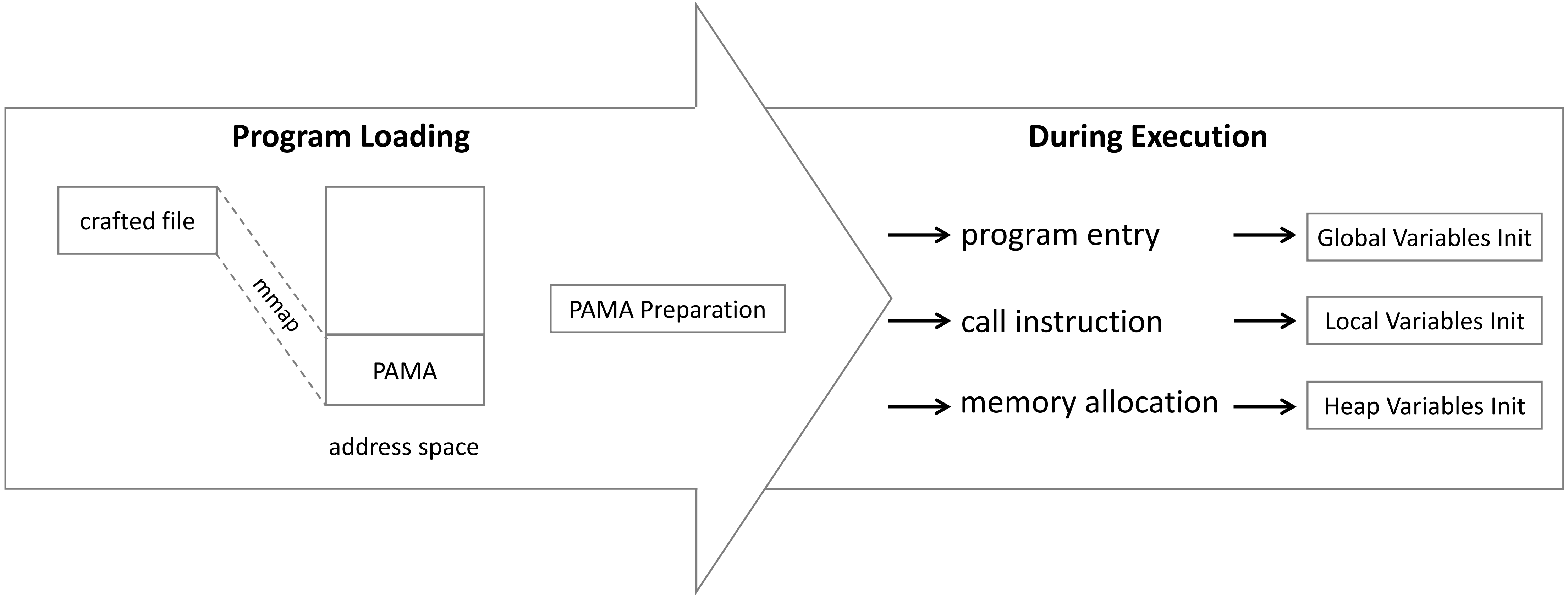
b->f2

PAMA	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0000	80	fe	00	00	00	00	00	00	50	38	00	00	00	00	00	00
0x0010	48	74	00	00	00	00	00	00	f8	04	00	00	00	00	00	00
0x0020	d0	ff	00	00	00	00	00	00	08	00	00	00	00	00	00	00
	...															
0xffd0	88	19	00	00	00	00	00	00	30	30	00	00	00	00	00	00
0xffe0	40	fc	00	00	00	00	00	00	98	20	00	00	00	00	00	00
0xfff0	20	50	00	00	00	00	00	00	e8	a7	00	00	00	00	00	00

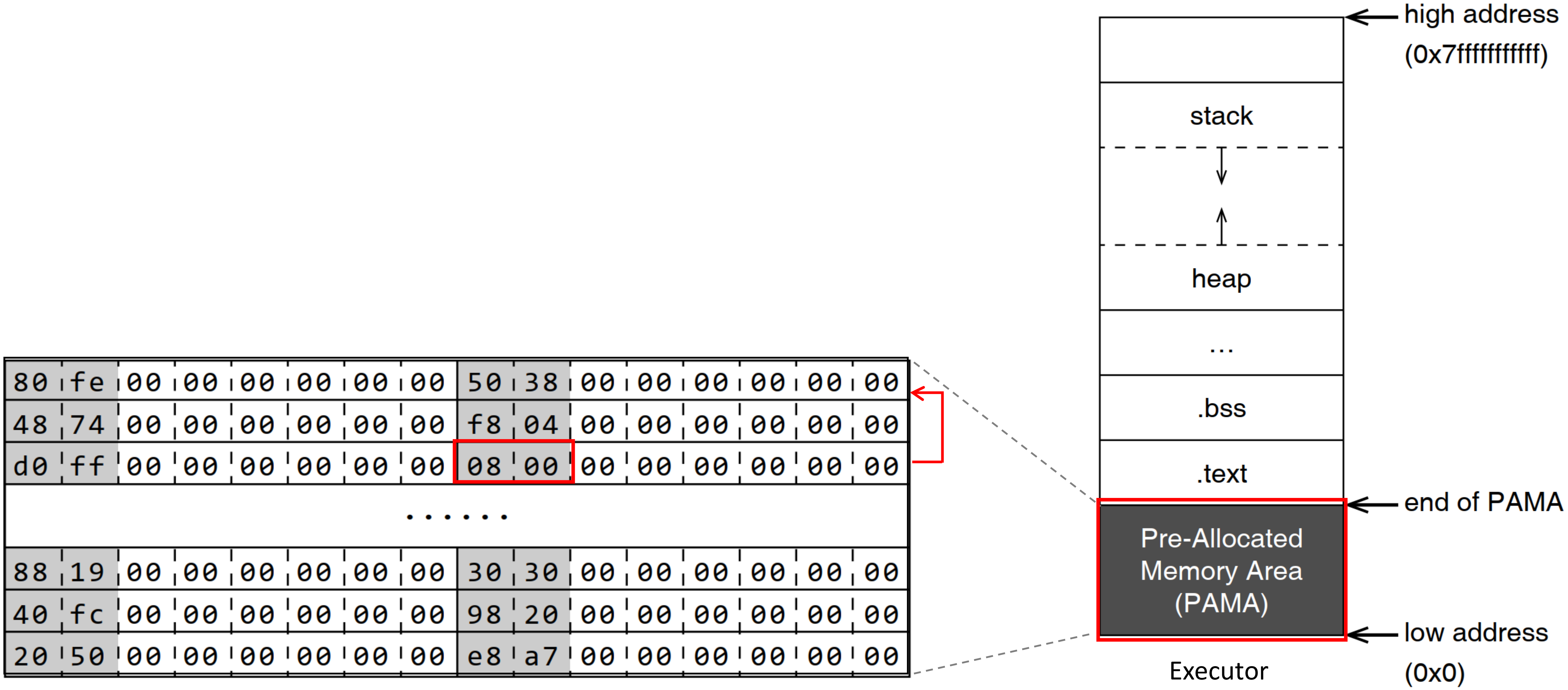
Architecture of PMP



Memory Pre-planning



PAMA Preparation



Variable Initialization

- Global Variables
  - read the offset and size information of the .bss segment from the ELF header
  - set .bss segment with random values indicating word-aligned PAMA addresses
- Heap Variables
  - intercept all memory allocations
  - set the allocated regions to contain random word-aligned PAMA addresses
- Local Variables
  - initialize the entire stack region like a heap region during program loading
  - intercept each function invocation to reinitialize the overwritten stack regions

SCMB and SDMB Properties

- SCMB (Self-Contained Memory Behavior)
  - if the filling values are interpreted as memory address, the corresponding accesses still fall into PAMA
  - violations of SCMB lead to memory access exceptions

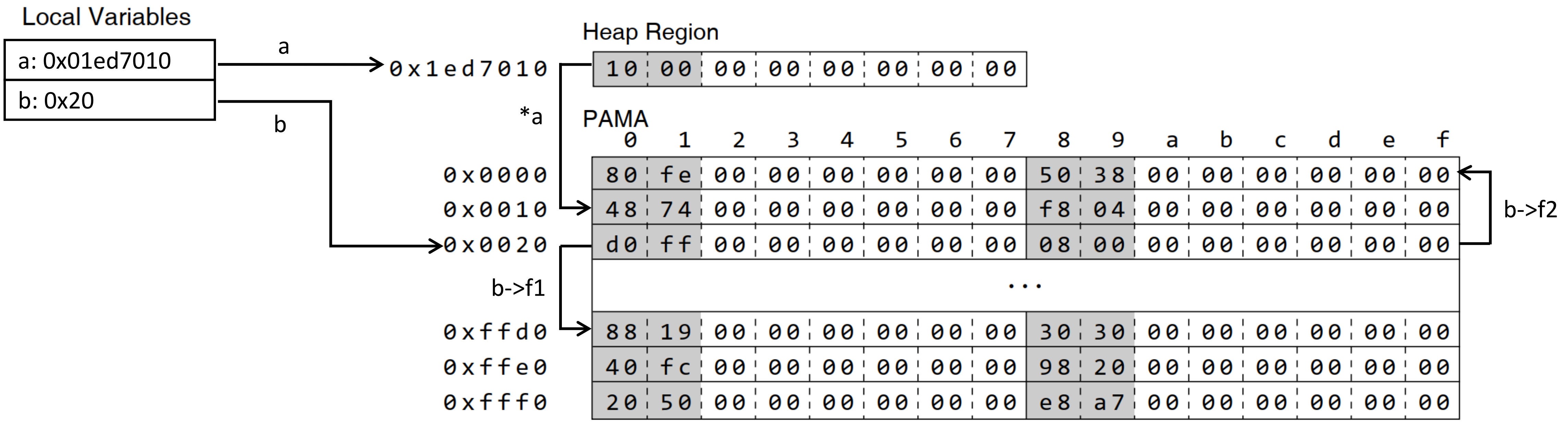
- SDMB (Self-Disambiguated Memory Behavior)
  - it is highly unlikely that two semantically unrelated memory operations access the same random address
  - violations of SDMB lead to bogus dependences and corrupted variable values

Example

takes the false branch

is forced to take the true branch

```
typedef struct{double *f1; long *f2;} T;
void foo() {
  long **a = malloc(...);
  T *b;
  if (cond1()) init(b);
  if (cond2()) {
    long *alias = b->f2;
    *(b->f2) = **a; // [0x0008] = [0x0010]
    *(b->f1) = 0.1; // [0xffd0] = 0.1
    long tmp = *alias;
  }
}
```



Implementation

- Based on QEMU User-Mode Emulator
  - instrument conditional jumps and indirect jumps to enforce path scheme
  - currently supports ELF binary on x86_64 platform
- Practical Challenges
  - handling file and network I/O, infinite loop and recursion
  - allocation of large PAMA
  - misaligned memory access

Probability Analysis

- Definition
  - $\texttt{PA}$: set of all possible addresses within PAMA
  - $\texttt{WA}$: word-aligned subset of $\texttt{PA}$, $\texttt{FV}$: random subset of $\texttt{WA}$
  - $\texttt{S=|PA|=|WA|}\times\texttt{8}$: size of PAMA, $\texttt{d=|FV|/|WA|}$: diversity of filling values
- Probabilistic Guarantee of SCMB

$$P_{err1} = P\left((x+\alpha) \notin \mathrm{PA} \mid x \in \mathrm{FV}\right) = \frac{\alpha}{S-8} \cdot \left(1 - \frac{8}{d \cdot S}\right) \qquad (1)$$

error1: out-of-bound access

- Probabilistic Guarantee of SDMB

$$P_{err2} = P\left(x = y \mid x \in \mathrm{FV},\ y \in \mathrm{FV}\right) = \frac{8}{d \cdot S} \qquad (2)$$

error2: coincidental address collision

$$\begin{aligned} P_{err3} &= P\left(l(x,\beta) \cap l(y,\gamma) \neq \emptyset \mid x \in \mathrm{FV},\ y \in \mathrm{FV}\right) \\ &\leq \frac{64}{d^2 \cdot S^2} + \left(1 - \frac{8}{d \cdot S}\right)^2 \cdot \frac{\beta+\gamma-8}{S-8} \end{aligned} \qquad (3)$$

error3: coincidental address overlap

Probabilities of Errors in a Typical Setting

- Typical Setting
  - 4-MB pre-allocated memory area ($\mathtt{S}$ = 0x400000)
  - 2 executors ($\mathtt{n}$ = 2)
  - diversity of filling values $\mathtt{d}$ is set to be 1
  - α = 8, β = 0x1000, γ = 0x1000

- Probabilities of Errors
  - P_{err1} = 1.9073e-06
  - P_{err2} = 1.9073e-06
  - $_{Perr3}$ = 0.00195

Evaluation Settings

- Subjects: SPEC2000 Malware

- Computing Resources
  - 8-core CPU (Intel®Core™ i7-8700@ 3.20GHz)
  - 16G main memory

- Time budget
  - no time limit for Spec2000
  - 5 minutes for each malware sample

Evaluation on SPEC2000

- SPEC2000: a well-known benchmark set
  - 12 real-world programs
  - some of them are large (e.g., 176.gcc)

- Comparison
  - execution outcomes
  - code coverage
  - memory dependence

VS X-force

Evaluation on SPEC2000

- Execution Outcomes
  - PMP is 84 times faster than X-Force
  - the failure rate is similar
- Code Coverage
  - PMP has comparable code coverage with X-Force (83.8% v.s. 82.7%)
  - PMP achieves 100% code coverage for some programs while X-Force does not
- Memory Dependence
  - X-Force has 6.5 times more false positives than PMP
  - X-Force has 10% more false negatives than PMP

Evaluation on Malware

- 400 Malware Samples
  - half of them are from VirusTotal
  - half of them are from Padawan

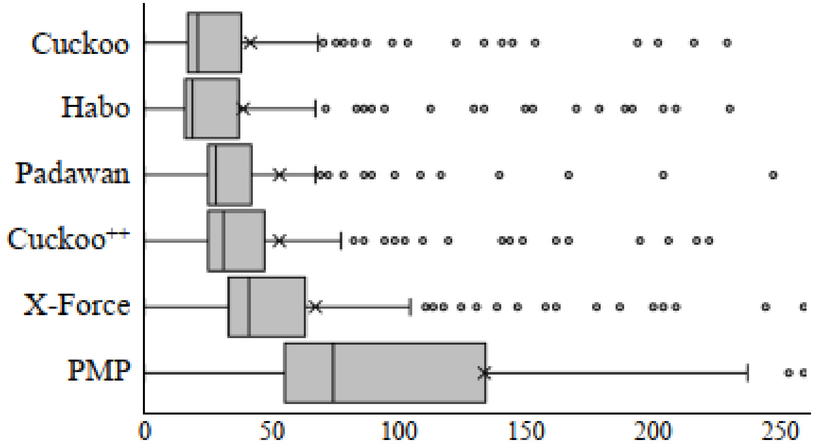
PADAWAN VS X-force

Habo Analysis System

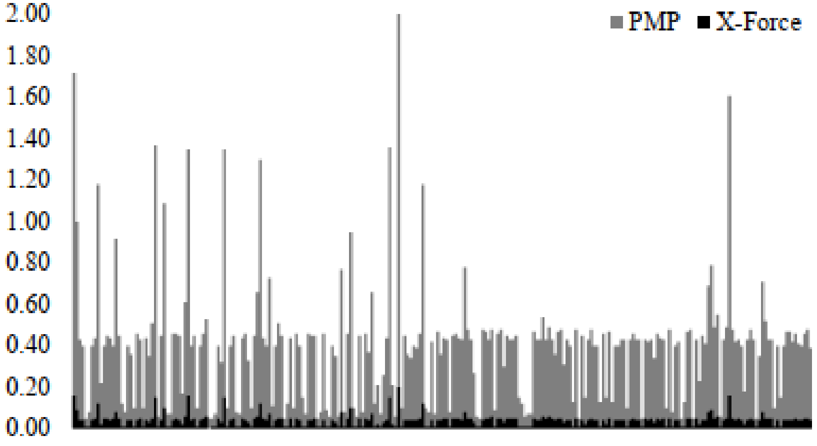
cuckoo $\xrightarrow[\text{anti-virtualization-detection}]{\text{system time fast-forwarding}}$ cuckoo++

Evaluation on Malware

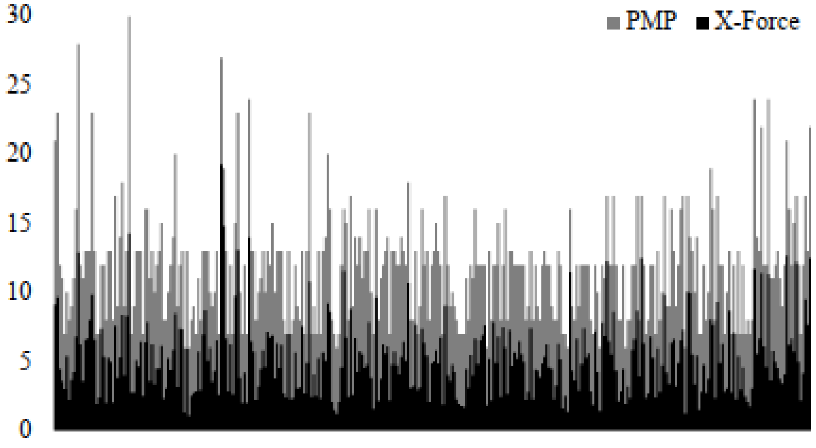
- PMP reports more than twice syscall sequences of that of other tools
- PMP is 9.8 times faster than X-Force
- PMP yields 1.5 times longer path schemes than X-Force



(a) number of exposed syscall sequences



(b) executions per second



(c) length of path scheme

Case Study: C&C Bot Malware Sample

- Simplified Code Snippet

```
char *data = read_file("/sys/class/dmi/id/product_name");
if (contains(data, "VirtualBox", "VMware"))
  remove_self_and_exit();
while (1) {
  char *ip = select_intranet_ip(ip_list);
  char *vuln = select_known_vuln(vuln_list);
  if (connect_and_check(ip, vuln)) {
    send_info_to_server(ip, vuln);
    send_payload(ip, vuln);
  }
}
```

VM detector

communication to the selected IP address

sending host info and payload

- Comparison of Different Tools

Tools	Cuckoo	Habo	Padawan	Cuckoo++	X-Force	PMP
# syscall sequences	153	169	292	221	274	705

Availability

Experimental version of PMP:

https://github.com/pmp-tool/PMP

Thank you!

Q & A