StackArmor: Stopping Stack-based Memory Error exploits in binaries

Xi Chen, Asia Slowinska, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida

Feb 10, 2015

VU VRIJE UNIVERSITEIT AMSTERDAM

Introduction

- Stack memory is an attractive target for attackers
  - CVE-2014-9163, Stack-based buffer overflow in Adobe Flash Player on Windows/OS X/Linux
  - CVE-2014-1593, Stack-based buffer overflow in Mozilla Firefox before 34.0

Introduction

- Stack memory is an attractive target for attackers
  - CVE-2014-9163, Stack-based buffer overflow in Adobe Flash Player on Windows/OS X/Linux
  - CVE-2014-1593, Stack-based buffer overflow in Mozilla Firefox before 34.0
- Protection against stack vulnerabilities in practice.
  - W$\oplus$X, Canaries, ASLR.

Introduction

- Stack memory is an attractive target for attackers
  - CVE-2014-9163, Stack-based buffer overflow in Adobe Flash Player on Windows/OS X/Linux
  - CVE-2014-1593, Stack-based buffer overflow in Mozilla Firefox before 34.0
- Protection against stack vulnerabilities in practice.
  - W$\oplus$X, Canaries, ASLR.
- The predictability of the stack is by design.

Threat model

- Spatial attacks
  - Buffer overflow, Buffer underflow

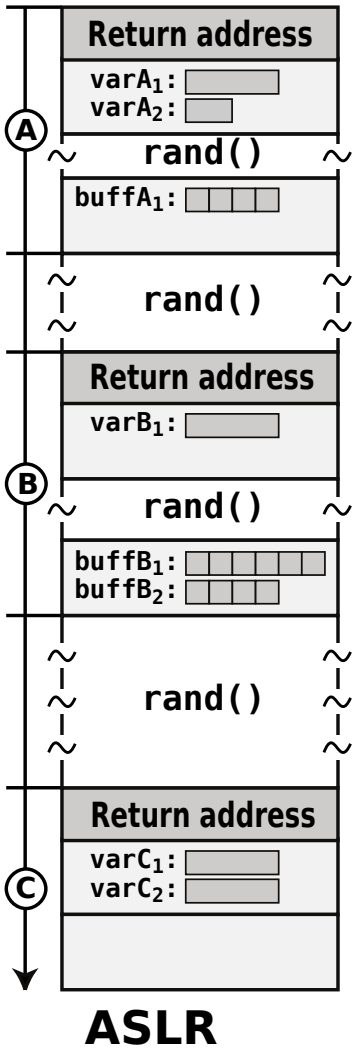
Threat model

- Spatial attacks
  - Buffer overflow, Buffer underflow
- Temporal attacks
  - Use-after-free, Uninitialized read

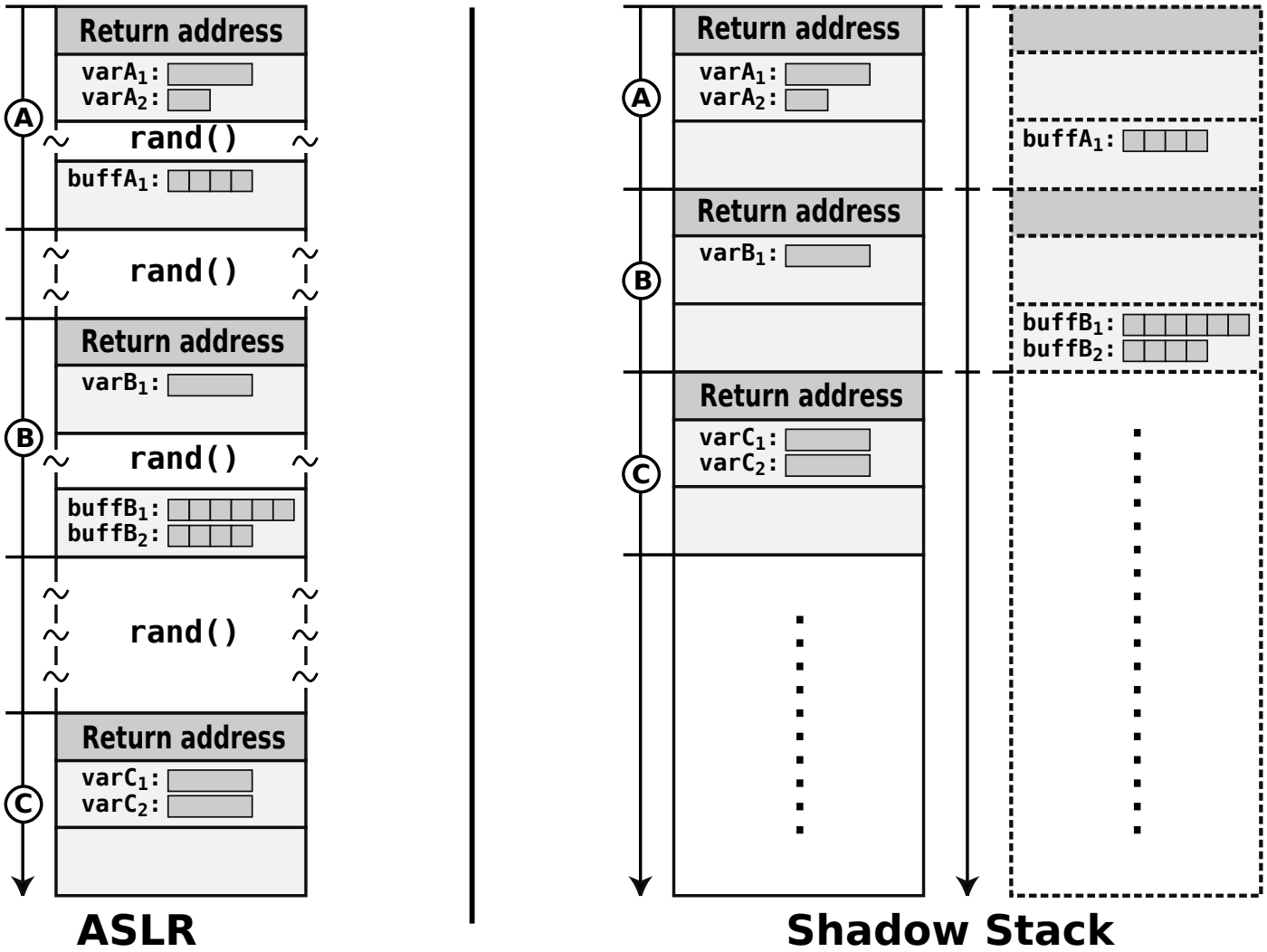
Threat model

- Spatial attacks
  - Buffer overflow, Buffer underflow
- Temporal attacks
  - Use-after-free, Uninitialized read
- Both attacks can happened intra-procedure or inter-procedure

Different stack protection techniques



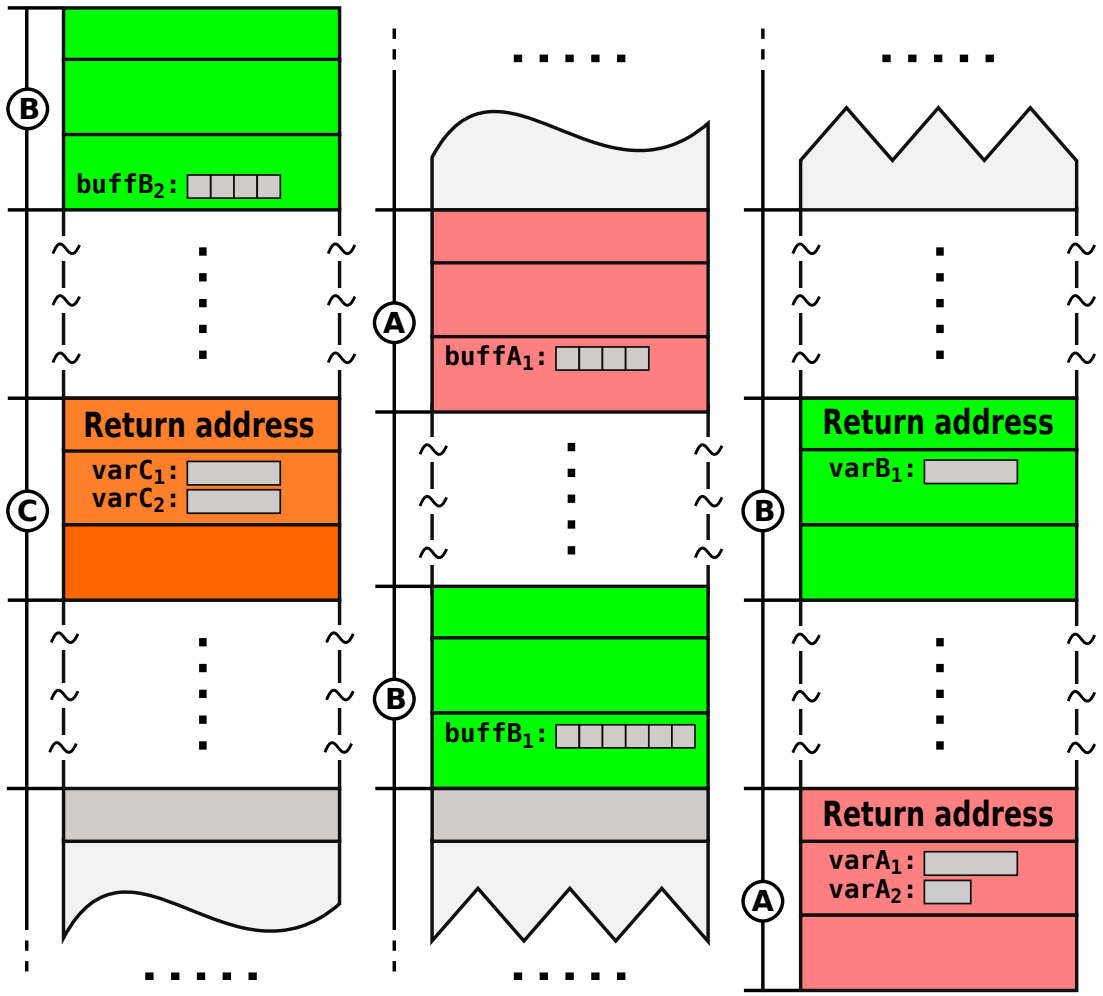
Different stack protection techniques



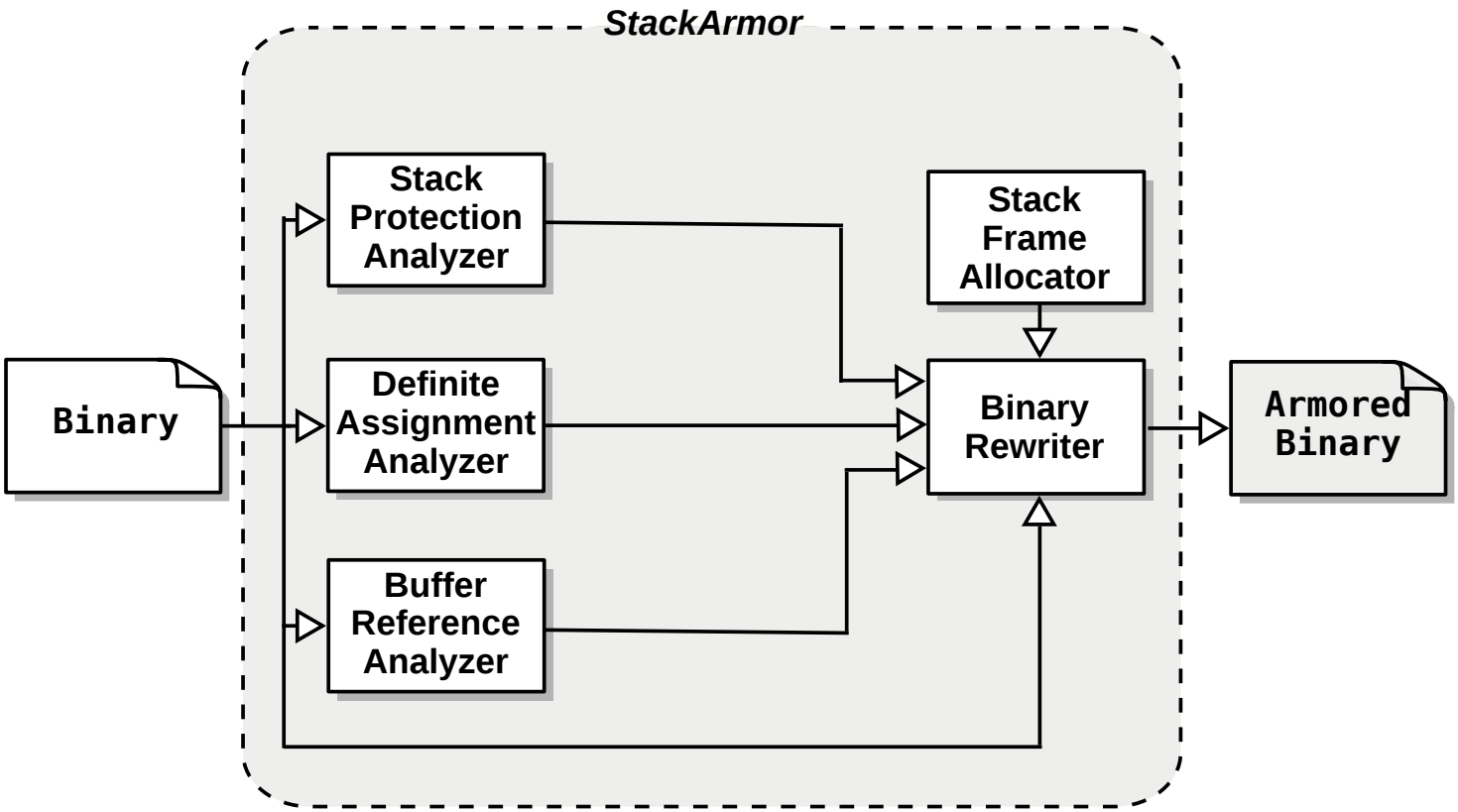
StackArmor

- Comprehensive approach against spatial and temporal Attacks
- A binary rewriting approach.
- No traditional stack, i.e., no predictable stack organization
- Combining stack frame randomization, buffer isolation and stack object zero initialization.

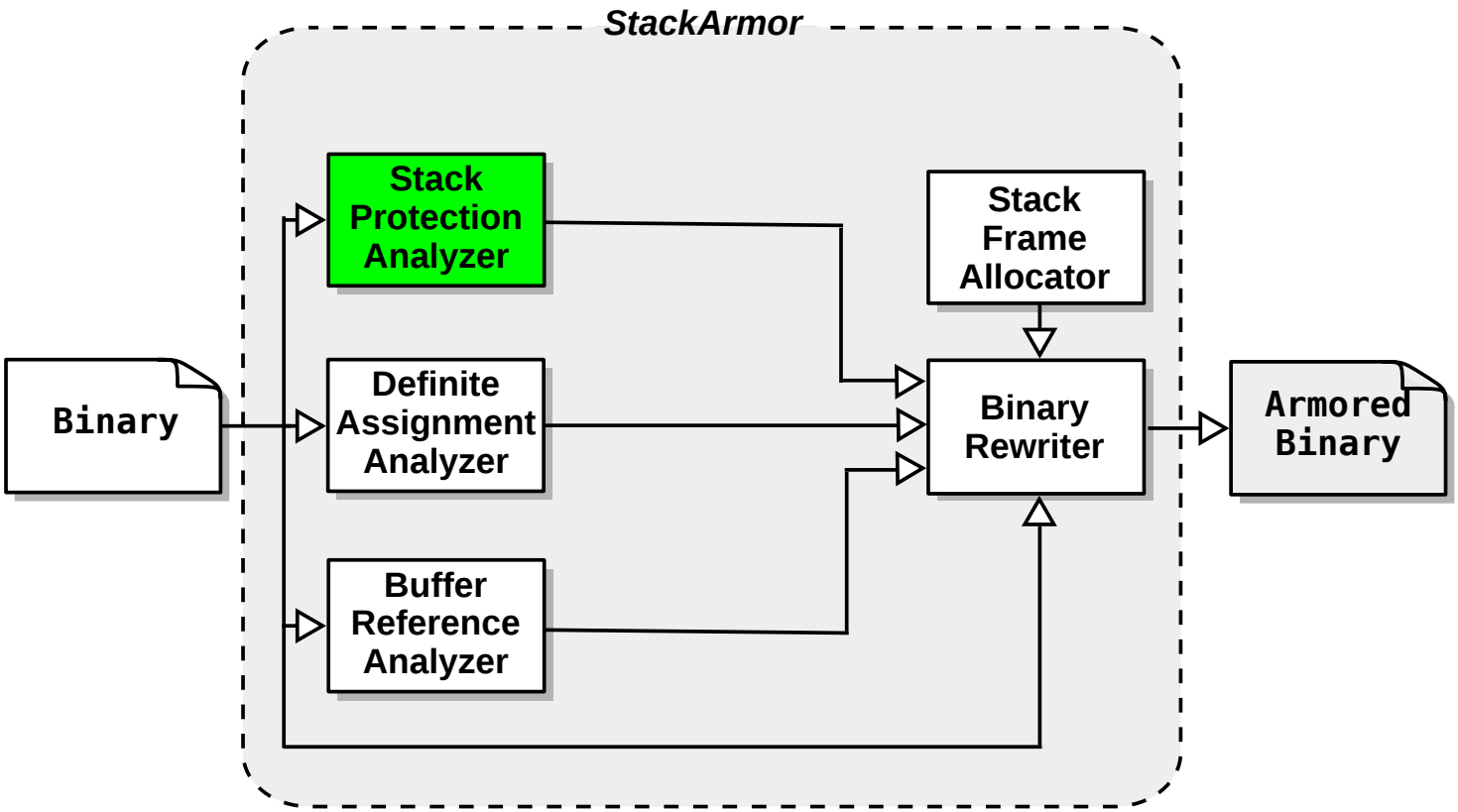
Stack frame layout under StackArmor



Overview of StackArmors's components



Stack protection analyzer



Stack protection analyzer

- Detect functions which have buffers inside.

Stack protection analyzer

- Detect functions which have buffers inside.
- Heuristics
  - Stack variables should only be accessed via stack/frame pointer with constant offset

Stack protection analyzer

- Detect functions which have buffers inside.
- Heuristics
  - Stack variables should only be accessed via stack/frame pointer with constant offset
  - Stack/frame pointer or derived pointer can not store into register/memory outside prologue/epilogue

Stack protection analyzer

- Detect functions which have buffers inside.
- Heuristics
  - Stack variables should only be accessed via stack/frame pointer with constant offset
  - Stack/frame pointer or derived pointer can not store into register/memory outside prologue/epilogue
  - Stack/frame pointer can not be manipulated outside prologue/epilogue

Stack protection analyzer

- Detect functions which have buffers inside.
- Heuristics
  - Stack variables should only be accessed via stack/frame pointer with constant offset
  - Stack/frame pointer or derived pointer can not store into register/memory outside prologue/epilogue
  - Stack/frame pointer can not be manipulated outside prologue/epilogue
- Seems very conservative, but we have simliar result comparing with GCC option

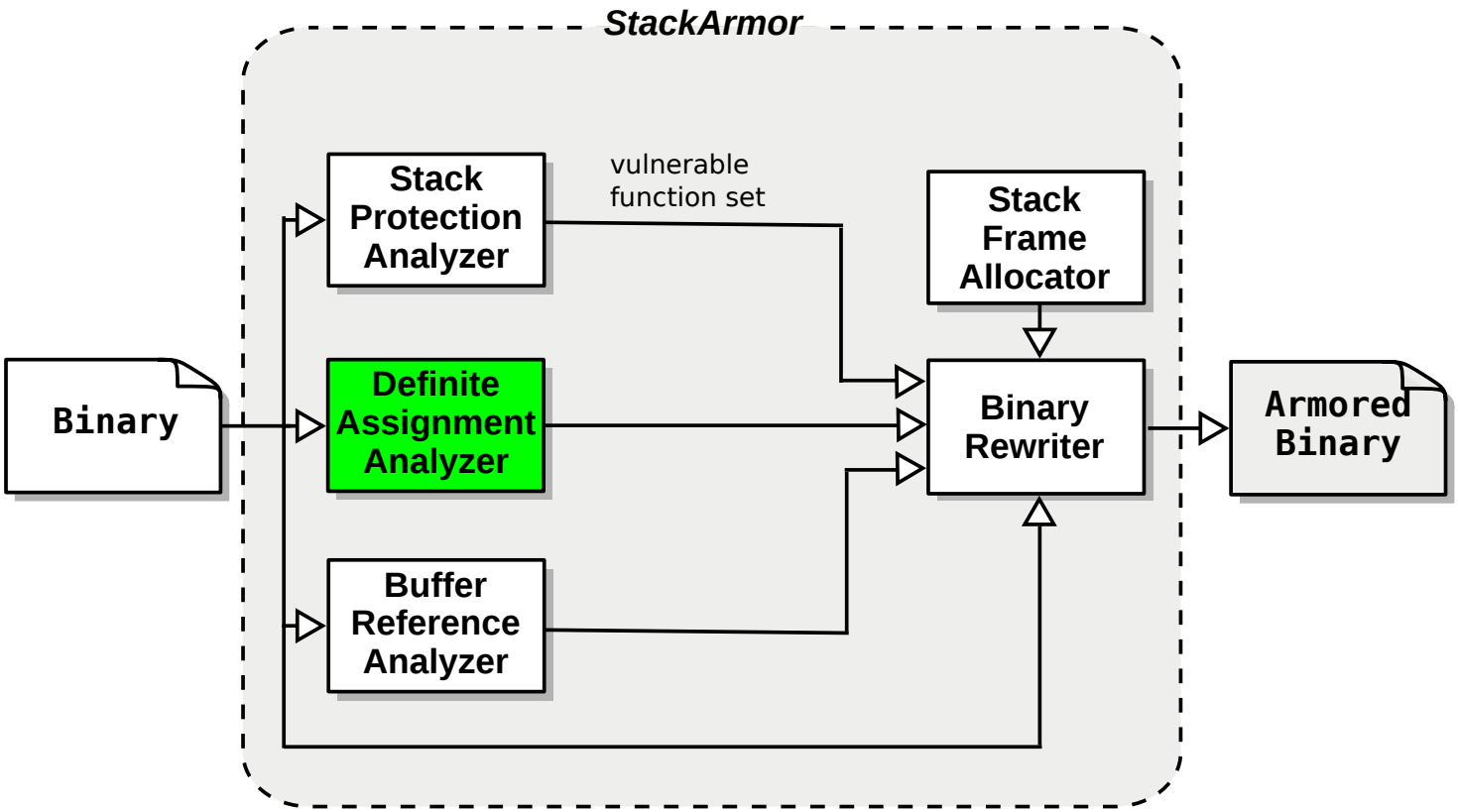
Violation example

```
extern void
helper_sp(int, int *, void *);

int
test_sp(int i, unsigned long size)
{
    int ret;
    char args[] = {1, 2, 3, 4};
    helper_sp(
        args[i],
        &ret,
        alloca(size));
    return ret;
}
```

```
function test_sp:
  pushq   %rbp
  movq    %rsp, %rbp
  subq    $32, %rsp
  movl    %edi, -4(%rbp)
  movq    %rsi, -16(%rbp)
  movl    $67305985, -24(%rbp)
  movslq  -4(%rbp), %rax
  movsbl  -24(%rbp,%rax), %edi
  movq    -16(%rbp), %rax
  addq    $15, %rax
  andq    $-16, %rax
  leaq    -20(%rbp), %rsi
  movq    %rsp, %rdx
  subq    %rax, %rdx
  movq    %rdx, %rsp
  callq   helper_sp
  movl    -20(%rbp), %eax
  movq    %rbp, %rsp
  popq    %rbp
  ret
```

Definite assignment analyzer



Definite assignment analyzer

- Detect stack variables which may be vulnerable to uninitialized read attack
  - In binary, we do initialization at byte granularity

Definite assignment analyzer

- Detect stack variables which may be vulnerable to uninitialized read attack
  - In binary, we do initialization at byte granularity
- Functions that pass stack protection analyzer: no need to be checked.

Definite assignment analyzer

- Detect stack variables which may be vulnerable to uninitialized read attack
  - In binary, we do initialization at byte granularity
- Functions that pass stack protection analyzer: no need to be checked.
- Static analysis remaining functions to find read-before-write bytes.

Definite assignment analyzer

- Detect stack variables which may be vulnerable to uninitialized read attack
  - In binary, we do initialization at byte granularity
- Functions that pass stack protection analyzer: no need to be checked.
- Static analysis remaining functions to find read-before-write bytes.
- False positive is acceptable

Definite assignment analyzer example

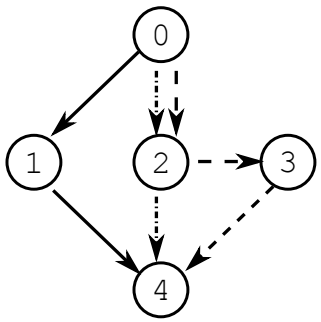
```
extern void
helper_da(int);

int
test_da(unsigned long size)
{
    int arg;
    if (size > 10)
        arg = 10;
    else if (size > 1)
        arg = 1;

    helper_da(arg)
}
```

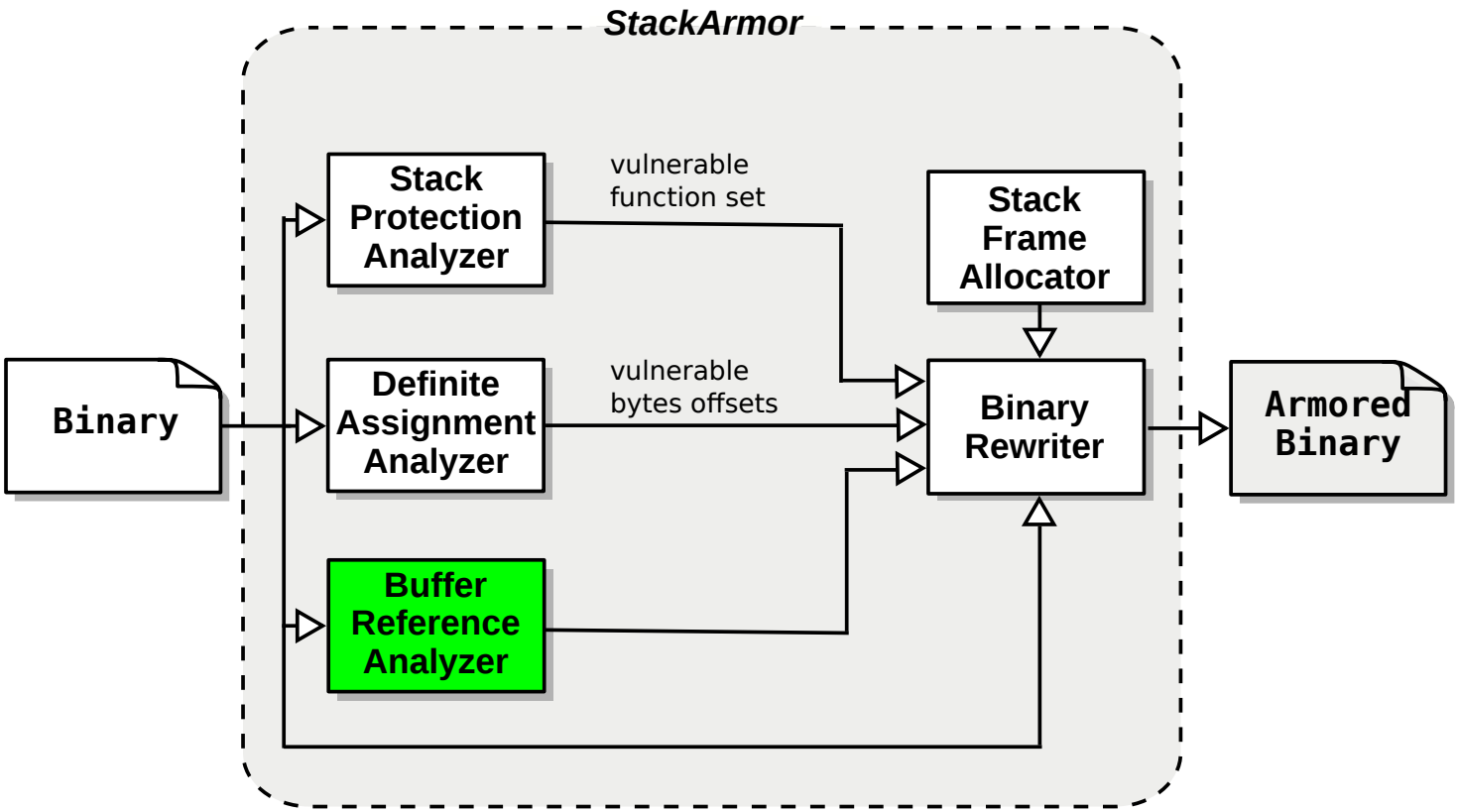
```
function test_da:
.LBB1_0:
  subq    $24, %rsp
  movq    %rdi, 16(%rsp)
  cmpq    $11, %rdi
  jb      .LBB1_2
.LBB1_1
  movl    $10, 12(%rsp)
  jmp     .LBB1_4
.LBB1_2:
  cmpq    $2, 16(%rsp)
  jb      .LBB1_4
.LBB1_3
  movl    $1, 12(%rsp)
.LBB1_4:
  movl    12(%rsp), %edi
  callq   helper_da
  addq    $24, %rsp
  ret
```

Control flow graph and the DA analyzer's results:



	12(%rsp)	16(%rsp)
→	unsafe	safe
-·->	safe	safe
- ->	safe	safe
DA result:	**unsafe**	**safe**

Buffer reference analyzer



Buffer reference analyzer

- Determines whether a stack buffer can be safely isolated

Buffer reference analyzer

- Determines whether a stack buffer can be safely isolated
- Safe isolation requires buffer references are never used to access other memory regions
- Ask buffer location and size information either from debug symbols or dynamic reverse engineering techniques.

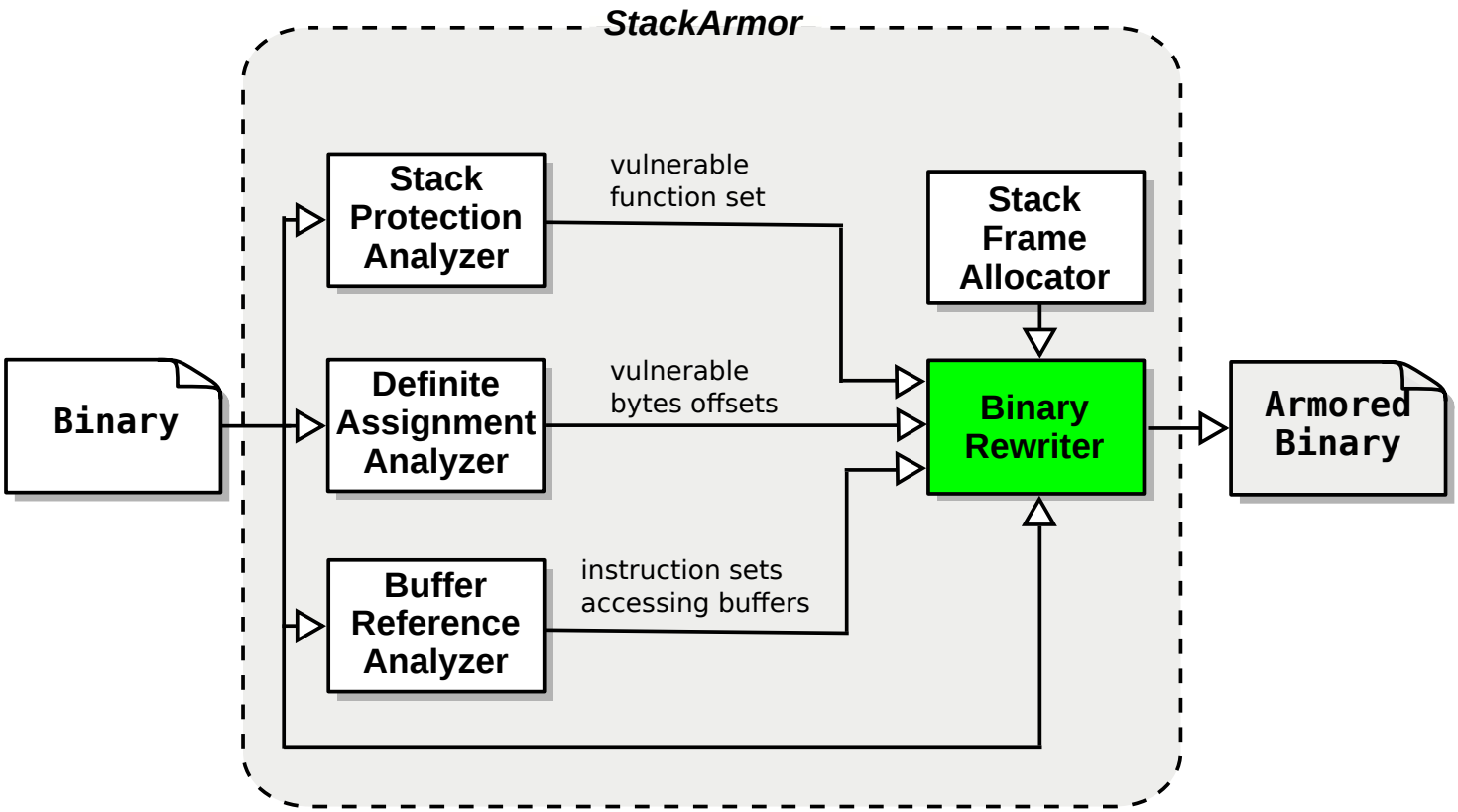
Buffer reference analyzer

- Determines whether a stack buffer can be safely isolated
- Safe isolation requires buffer references are never used to access other memory regions
- Ask buffer location and size information either from debug symbols or dynamic reverse engineering techniques.
- Static data-flow tracking analysis to find instructions which access buffers
  - Can afford neither false positives nor false negatives

Buffer reference analyzer

- Determines whether a stack buffer can be safely isolated
- Safe isolation requires buffer references are never used to access other memory regions
- Ask buffer location and size information either from debug symbols or dynamic reverse engineering techniques.
- Static data-flow tracking analysis to find instructions which access buffers
  - Can afford neither false positives nor false negatives
  - If can not resolve the address being de-referenced, give up
  - If a insturction can access different objects, give up

Binary instrumentation



Binary instrumentation

- Buffer Isolation : Remap stack-referencing instructions

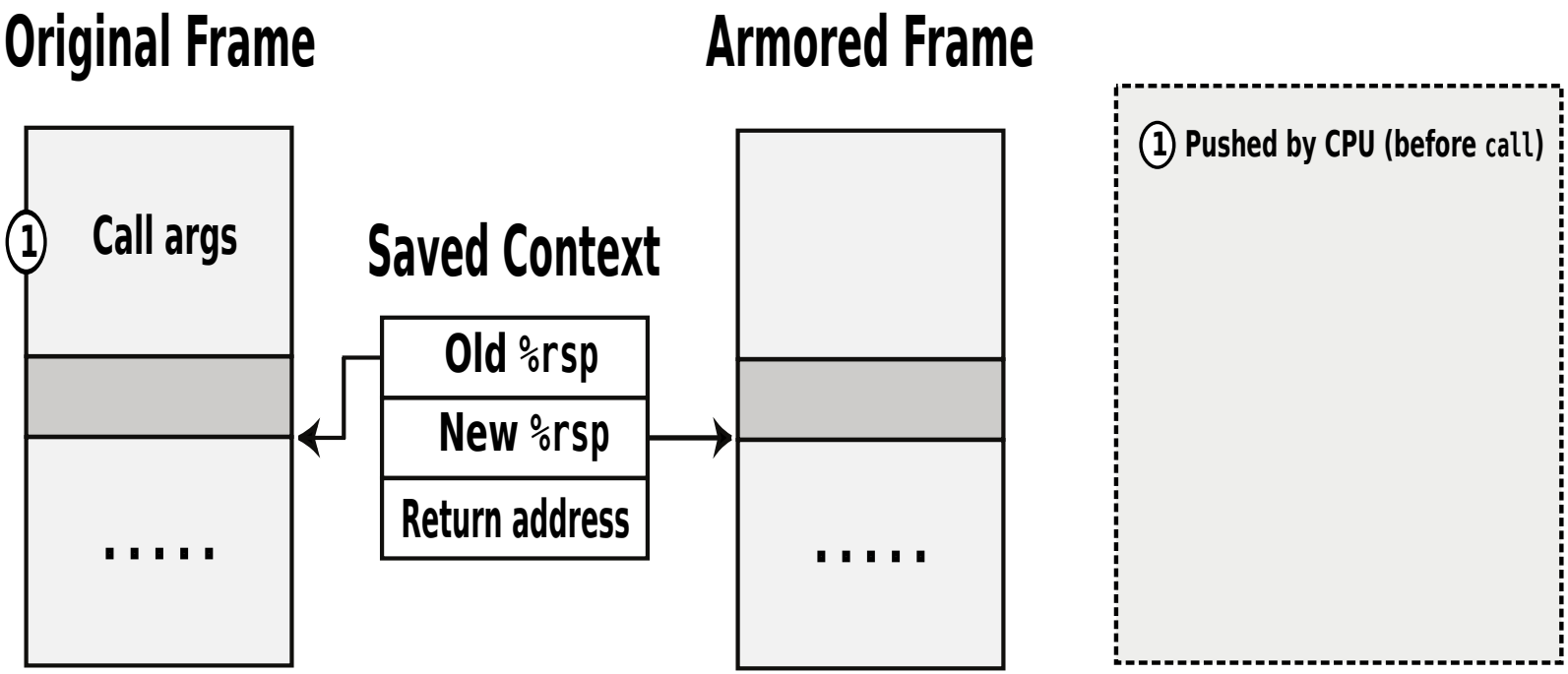
Binary instrumentation

- Buffer Isolation : Remap stack-referencing instructions
- Stack initialization : Zero initialize read-before-write bytes

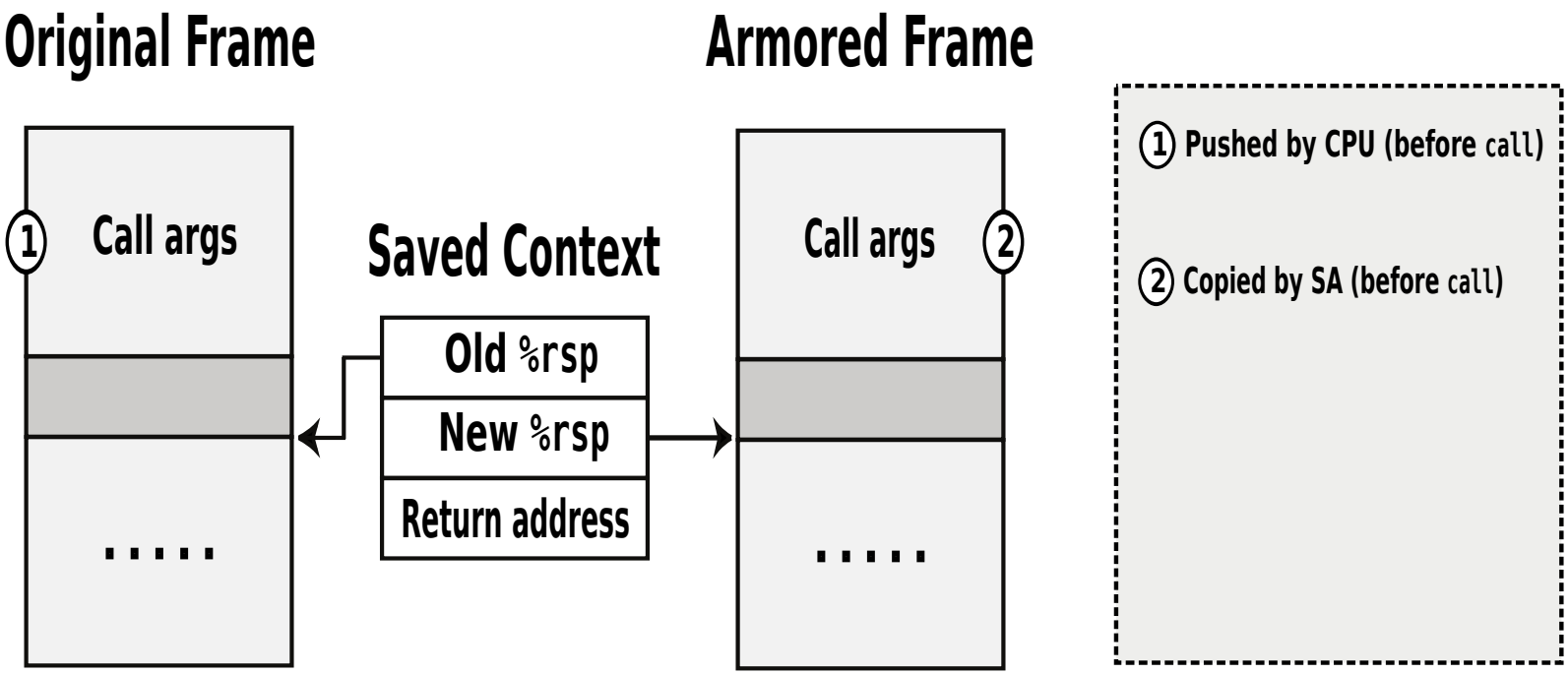
Binary instrumentation

- Buffer Isolation : Remap stack-referencing instructions
- Stack initialization : Zero initialize read-before-write bytes
- Stack frame randomization : Call site instrumentation

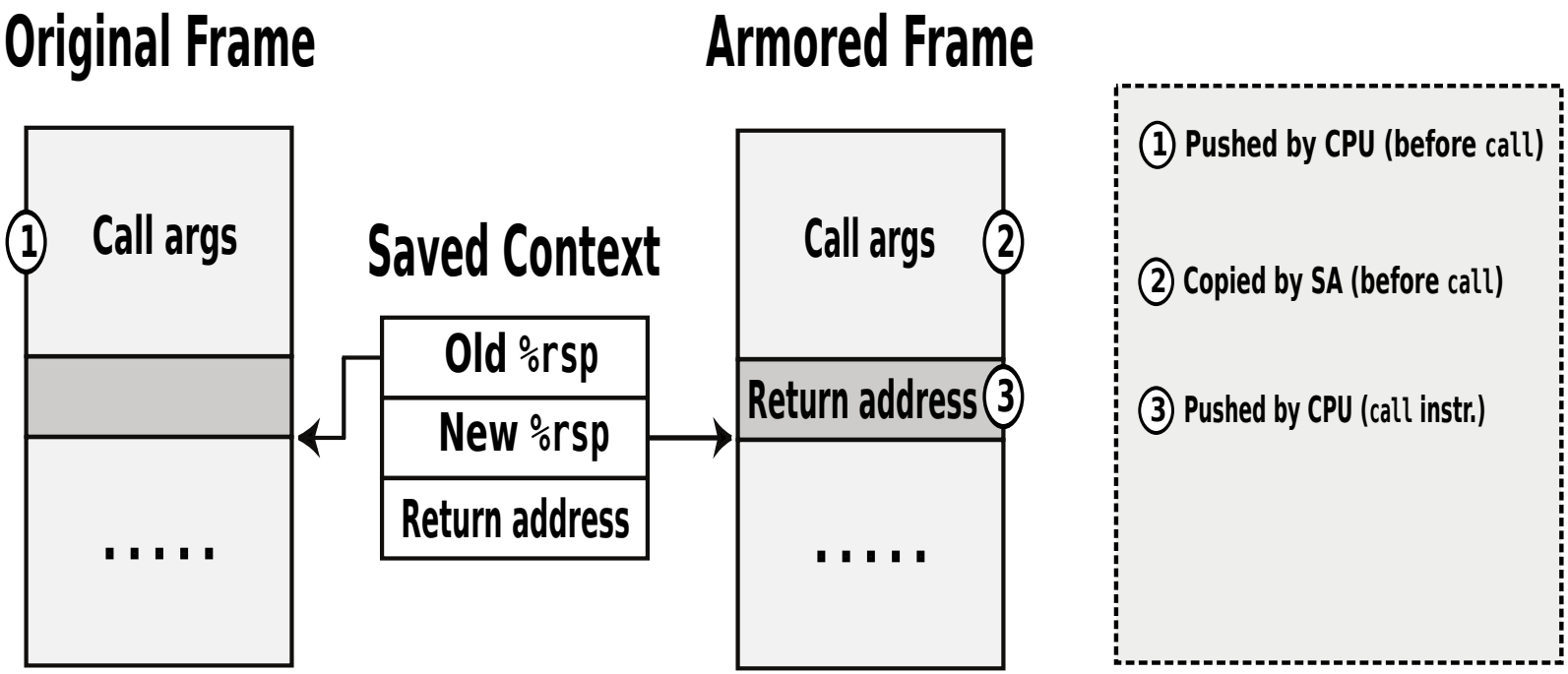
Call site instrumentation



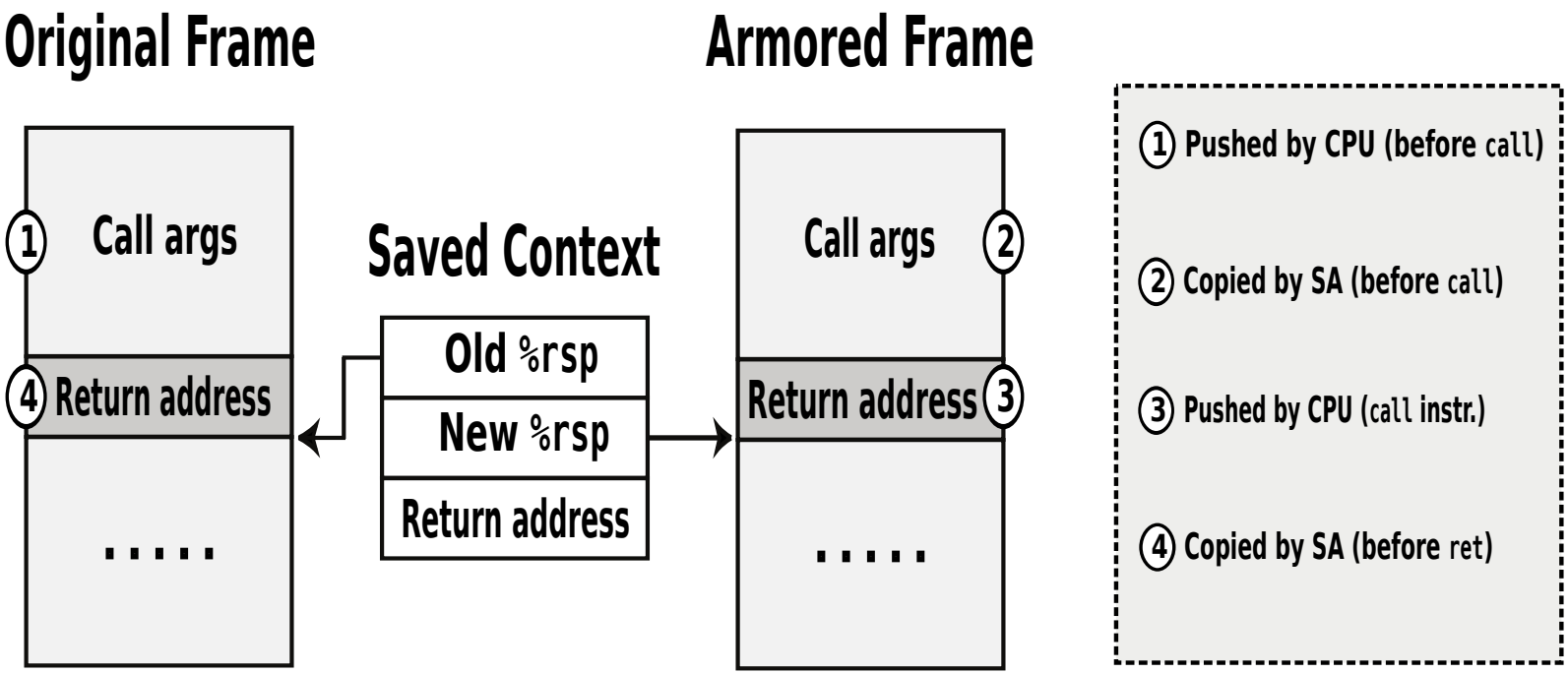
Call site instrumentation



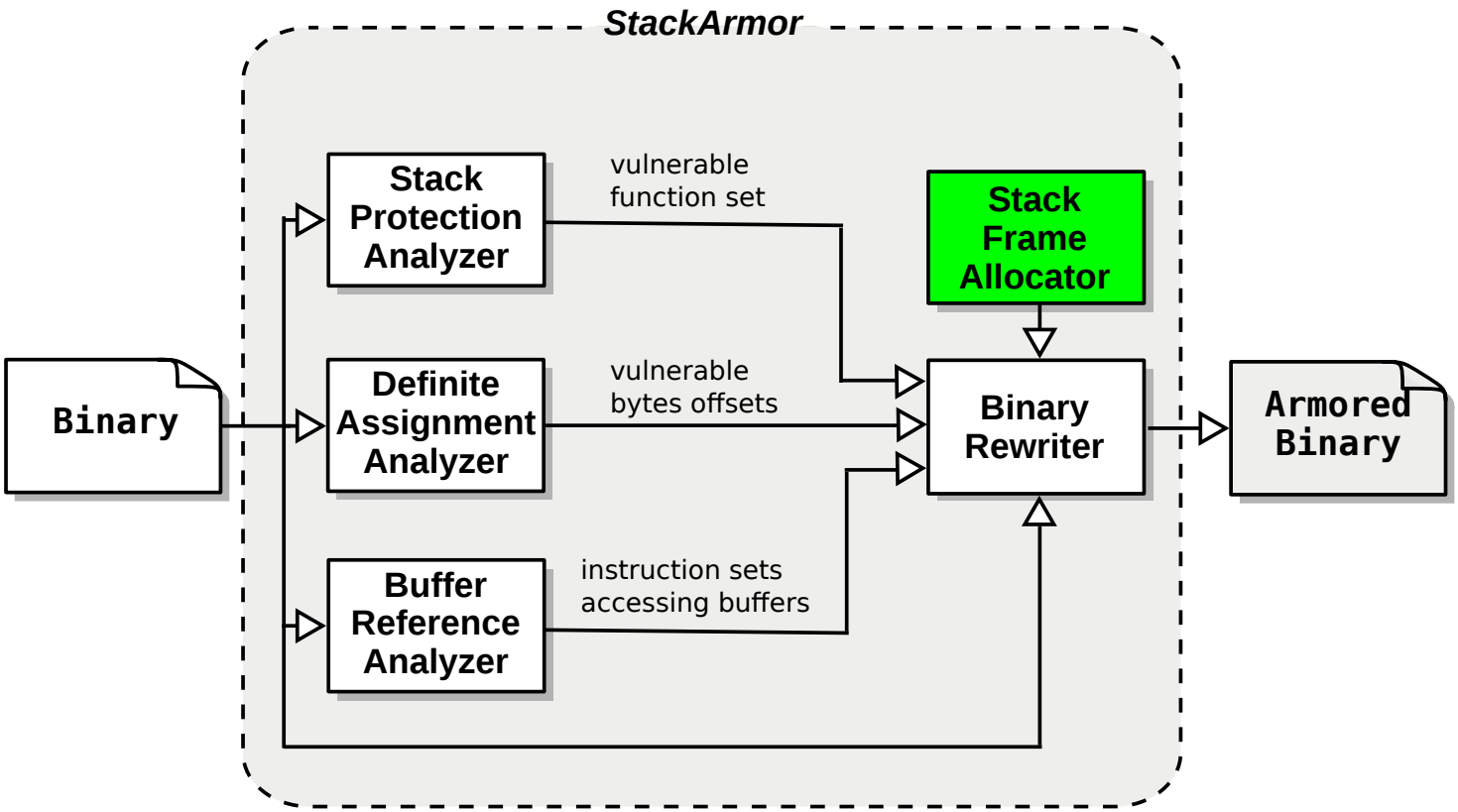
Call site instrumentation



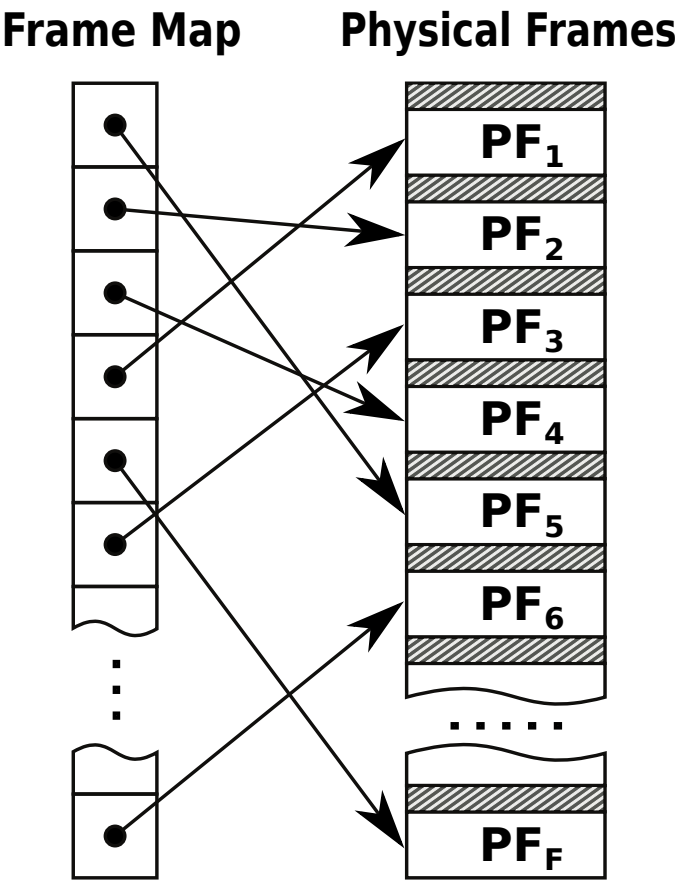
Call site instrumentation



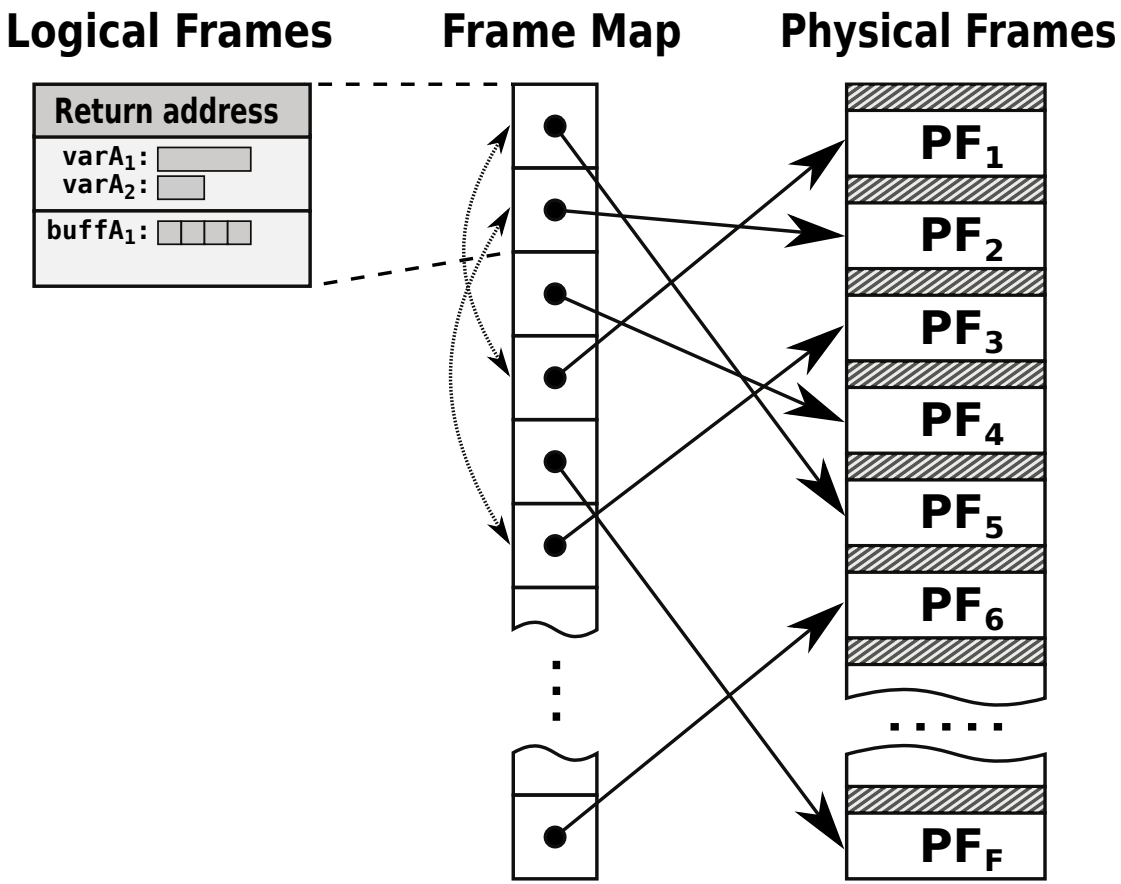
Stack frame allocator



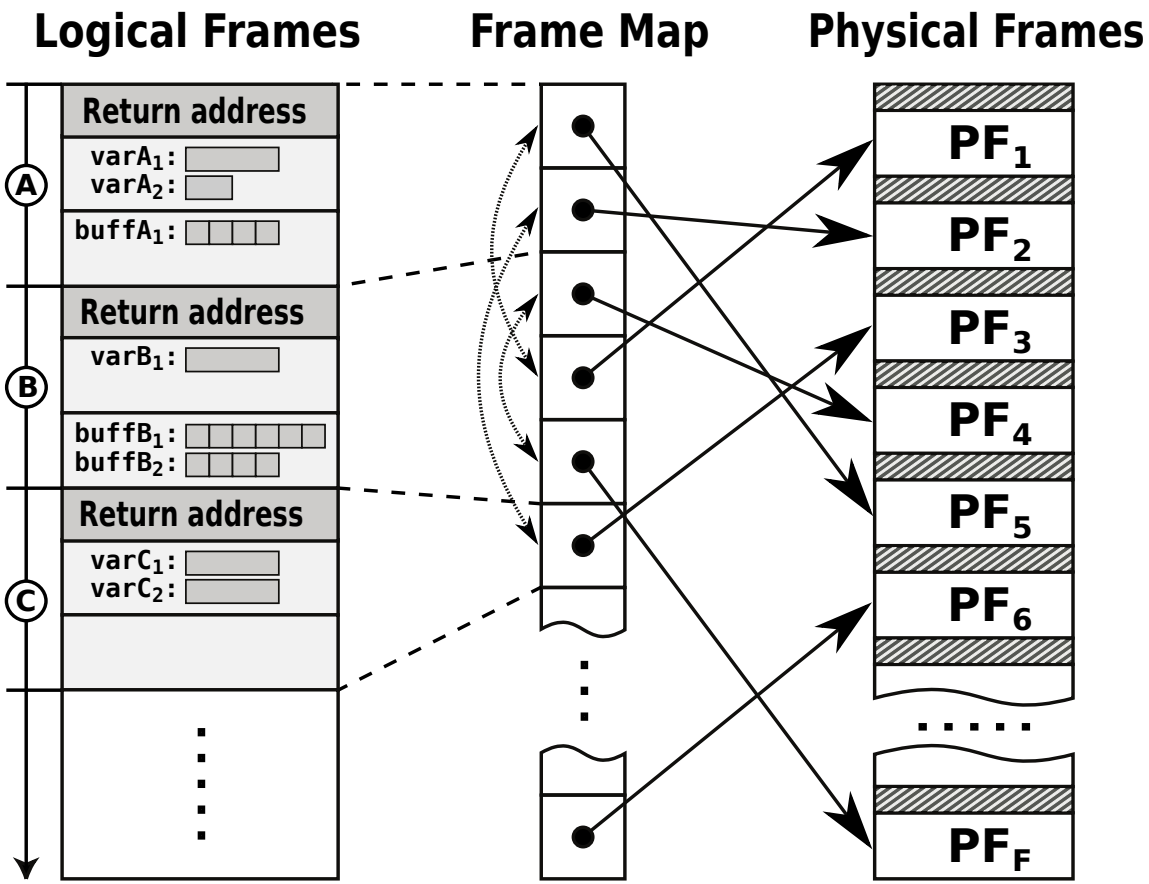
Stack frame allocator



Stack frame allocator



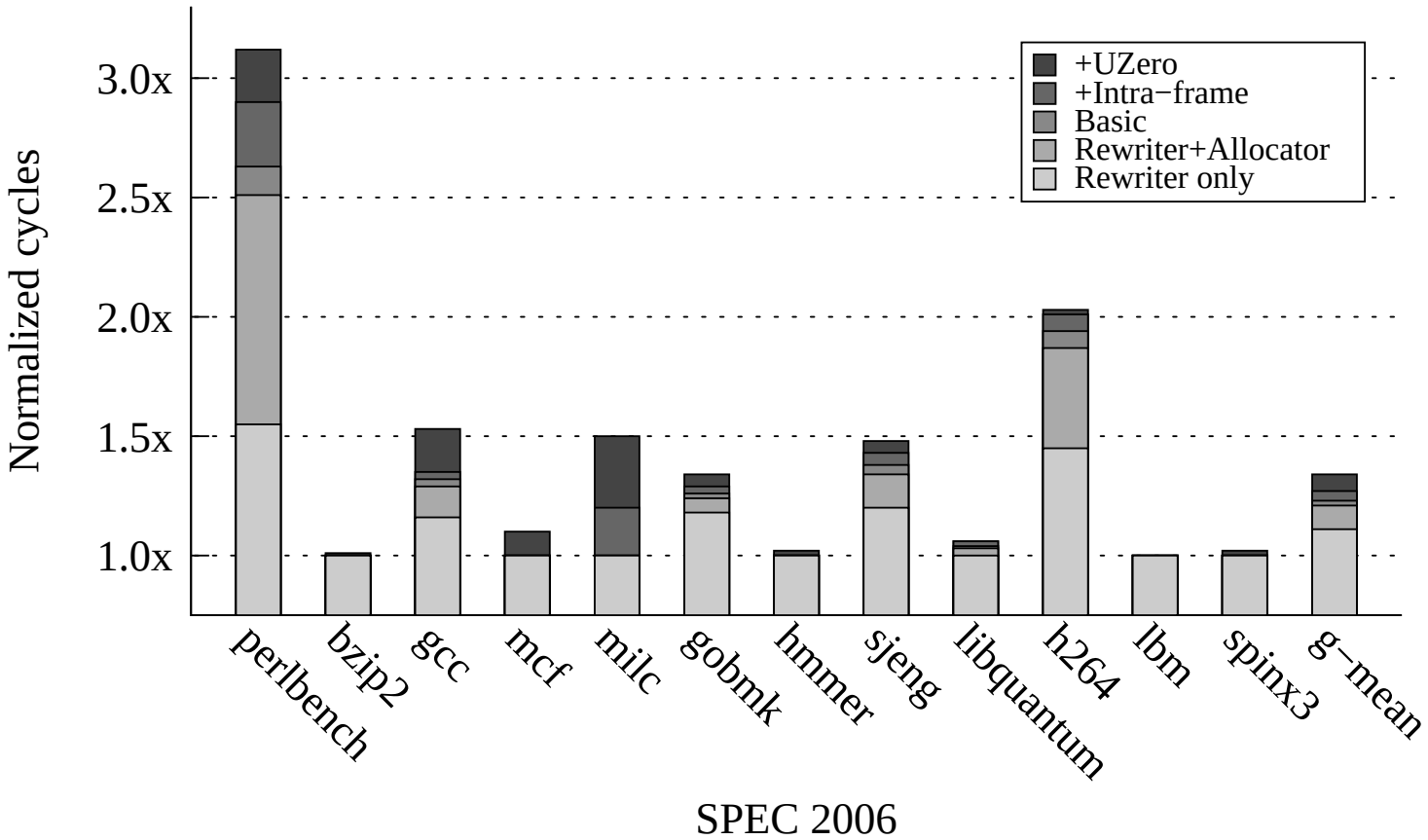
Stack frame allocator



Run time overhead

App	Basic	+Buffer-Isolation	+Zero-Initialization
lighttpd	1.06x	1.07x	1.10x
exim	1.01x	1.04x	1.05x
openssh	1.00x	1.01x	1.01x
vsftpd	1.00x	1.01x	1.04x
SPEC_{gm}	1.16x	1.22x	1.28x

Detailed run time overhead on SPEC 2006



Conclusions

- StackArmor "destroys" traditional stack organization to provide fully randomized stack space
- It can protect against stack-based spatial and temporal attacks
- And it provides tunable trade-off between performance and security

Thanks, any questions?