StackArmor: Stopping Stack-based Memory Error exploits in binaries

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

Feb 10, 2015
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⊕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

ASLR

Shadow Stack
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

- **buffB2**: ..... 
- **varC1**: ..... 
- **varC2**: ..... 
- **buffB1**: ..... 
- **varA1**: ..... 
- **varA2**: ..... 
- **Return address**: }

### Design

StackArmor: Stopping Stack-based Memory Error exploits in binaries

---

![Diagram](image-url)
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 similar result comparing with GCC option
Violation example

extern void
definer_sp(int, int *, void *);

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

Control flow graph and the DA analyzer's results:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

function test_da:

LBB0:  
    subq $24, %rsp
    movq %rdi, 16(%rsp)
    cmpq $11, %rdi
    jb .LBB1 2

LBB1:
    movl $10, 12(%rsp)
    jmp .LBB3 4

LBB2:
    cmpq $2, 16(%rsp)
    jb .LBB3 4

LBB3:
    movl $1, 12(%rsp)

LBB4:
    movl 12(%rsp), %edi
    callq helper_da
    addq $24, %rsp
    ret

12(%rsp) 16(%rsp)

unsafe  safe
unsafe  safe
safe  safe
safe  safe
DA result: unsafe  safe
Buffer reference analyzer

StackArmor: Stopping Stack-based Memory Error exploits in binaries

Design

StackArmor

Stack Protection Analyzer

Definite Assignment Analyzer

Buffer Reference Analyzer

Stack Frame Allocator

Binary Rewriter

Armored Binary
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 instruction can access different objects, give up
Binary instrumentation

StackArmor: Stopping Stack-based Memory Error exploits in binaries

Design

StackArmor

- Stack Protection Analyzer
- Definite Assignment Analyzer
- Buffer Reference Analyzer
- Stack Frame Allocator
- Binary Rewriter

Binary

Armored Binary

vulnerable function set
vulnerable bytes offsets
instruction sets accessing buffers
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**

- **Original Frame**
  - Call args
  - Saved Context:
    - Old \%rsp
    - New \%rsp
    - Return address

- **Armored Frame**
  - Call args

- **Notes**:
  - 1 Pushed by CPU (before call)
  - 2 Copied by SA (before call)
Call site instrumentation

Original Frame

Armored Frame

1. Call args

2. Saved Context
   - Old %rsp
   - New %rsp
   - Return address

3. Return address

4. Pushed by CPU (before call)
5. Copied by SA (before call)
6. Pushed by CPU (call instr.)
Call site instrumentation
Stack frame allocator

Stack Armor: Stopping Stack-based Memory Error exploits in binaries

Design

Stack Armor

Stack frame allocator

[Diagram showing the flow of information from Binary to Stack Armor, through Stack Protection Analyzer, Definite Assignment Analyzer, Buffer Reference Analyzer, to Stack Frame Allocator, and finally to Armored Binary.]
Stack frame allocator

Frame Map

Physical Frames

PF_1
PF_2
PF_3
PF_4
PF_5
PF_6
PF_F

......
Stack frame allocator
Stack Armor: Stopping Stack-based Memory Error exploits in binaries

Stack frame allocator

Logical Frames

Return address

varA₁: 
varA₂: 

buffA₁: 

Return address

varB₁: 

buffB₁: 
buffB₂: 

Return address

varC₁: 
varC₂: 

... 

Frame Map

Physical Frames

PF₁ 
PF₂ 
PF₃ 
PF₄ 
PF₅ 
PF₆ 

... 

PF₇
## Run time overhead

<table>
<thead>
<tr>
<th>App</th>
<th>Basic</th>
<th>+Buffer-Isolation</th>
<th>+Zero-Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>lighttpd</td>
<td>1.06x</td>
<td>1.07x</td>
<td>1.10x</td>
</tr>
<tr>
<td>exim</td>
<td>1.01x</td>
<td>1.04x</td>
<td>1.05x</td>
</tr>
<tr>
<td>openssh</td>
<td>1.00x</td>
<td>1.01x</td>
<td>1.01x</td>
</tr>
<tr>
<td>vsftpd</td>
<td>1.00x</td>
<td>1.01x</td>
<td>1.04x</td>
</tr>
<tr>
<td>SPEC$_{gm}$</td>
<td>1.16x</td>
<td>1.22x</td>
<td>1.28x</td>
</tr>
</tbody>
</table>
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?