Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida    Anton Kuijsten    Andrew S. Tanenbaum

Vrije Universiteit Amsterdam

21st USENIX Security Symposium

Bellevue, WA, USA
August 8-10, 2012
Kernel-level Exploitation

- Kernel-level exploitation increasingly gaining momentum.

- Many exploits available for *Windows*, *Linux*, *BSD*, *Mac OS X*, *iOS*.

- Plenty of memory error vulnerabilities to choose from.

- Plethora of internet-connected users running the same kernel version.

- Many attack opportunities for both local and remote exploits.
Existing Countermeasures

- Preserving kernel code integrity [SecVisor, NICKLE, hvmHarvard].
- Kernel hook protection [HookSafe, HookScout, Indexed hooks].
- Control-flow integrity [SBCFI].
- No comprehensive memory error protection.
- Virtualization support required, high overhead.
Address Space Randomization

- Well-established defense mechanism against memory error exploits.

- Application-level support in all the major operating systems.

- The operating system itself typically not randomized at all.

- Only recent *Windows* releases perform basic text randomization.

- Goal: Fine-grained ASR for operating systems.
Challenges in OS-level ASR

Instrumentation

Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
Challenges in OS-level ASR

Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida

Rerandomization
Challenges in OS-level ASR

Information leakage
Challenges in OS-level ASR

Brute forcing
A Design for OS-level ASR

- Make both location and layout of memory objects unpredictable.

- LLVM-based link-time transformations for safe and efficient ASR.

- Minimal amount of untrusted code exposed to the runtime.

- Live rerandomization to maximize unobservability of the system.

- No changes in the software distribution model.
Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
Code Randomization

0x...00  define i32 @my_function() nounwind uwtable {
    entry:
        %6 = call @printf(i8* getelementptr inbounds (@.str, ...))
        [...] 
        ret i32 0
}

Original function (LLVM IR)
Randomize function location

```assembly
define i32 @my_function() nounwind uwtable {
    entry:
        %6 = call @printf(i8* getelementptr inbounds (@str, ...))
        [...] 
        ret i32 0
}
```
Add random-sized padding
Basic block shifting
Static Data Randomization

0x...00  @my_variable = global %struct.my_struct zeroinitializer

%struct.my_struct = type {
   i32 flags,
   i16 id,
   %struct.my_struct *next
   i8* address,
   [8 x i8] string,
}

Original variable and type (LLVM IR)
Static Data Randomization

@my_variable = global %struct.my_struct zeroinitializer

%struct.my_struct = type {
  ; original type
  i32 flags,
  i16 id,
  %struct.my_struct *next
  i8* address,
  [8 x i8] string,
}

Randomize variable location
Static Data Randomization

Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida

0x...a0  @my_variable_padding = global [... x i8] zeroinitializer

0x...b4  @my_variable = global %struct.my_struct zeroinitializer

%struct.my_struct = type {
    ; original type
    +0x00  i32 flags,
    +0x04  i16 id,
    +0x08  %struct.my_struct *next
    +0x0C  i8* address,
    +0x10  [8 x i8] string,
}

Add random-sized padding
Static Data Randomization

0x...a0  @my_variable_padding = global [...] x i8] zeroinitializer

0x...b4  @my_variable = global %struct.my_struct zeroinitializer

%struct.my_struct = type {
   ; randomized type
   +0x00  [... x i8] id_padding,
   +0xa0  i16 id,
   +0xa2  [... x i8] flags_padding,
   +0xb4  i32 flags,
   +0xb8  [... x i8] string_padding,
   +0xc8  [8 x i8] string,
   +0xd0  [... x i8] address_padding,
   +0xe0  i8* address,
   +0xe4  [... x i8] next_padding,
   +0xf4  %struct.my_struct *next
}

Internal layout randomization
Stack Randomization

Stack frame

- Previous frame
- Parameters
- Return address
- Saved base pointer
- Local variables

New stack frame

- Previous frame
- Inter-frame padding
- Parameters
- Return address
- Saved base pointer
- Nonbuffer variables
- Intra-frame padding
- Buffer variables
Dynamic Data Randomization

- Support for `malloc()`/`mmap()`-like allocator abstractions.

- Memory mapped regions are fully randomized.

- Heap allocations are interleaved with random-sized padding.

- Full heap randomization enforced at live rerandomization time.

- ILR for all the dynamically allocated memory objects.
Live Rerandomization

- First **stateful** live rerandomization technique.

- Periodically rerandomize the memory address space layout.

- Support arbitrary memory layout changes at rerandomization time.

- Support all the standard C idioms with minimal manual effort.

- Sandbox the rerandomization code to recover from run-time errors.
Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
ASRR Metadata

- Types
- Global variables
- Static variables
- String constants
- Functions
- Dynamic memory allocations
The Rerandomization Process

Randomization Manager

V1
State
Metadata

V2
State
Metadata
The Rerandomization Process

Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
The Rerandomization Process

Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
The Rerandomization Process

Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
The Rerandomization Process

Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Randomization Manager

V1
State
Metadata
Sync point

Control flow transfer

V2
State
Metadata
Sync point

TRACE

Cristiano Giuffrida
Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization

Cristiano Giuffrida
Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization
Cristiano Giuffrida

ASRR Performance

- SPEC CPU 2006 benchmarks
- devtools benchmark

Runtime overhead (%) vs. Rerandomization latency (s)
A new fine-grained ASR technique for operating systems.

Better performance and security than prior ASR solutions.

Live rerandomization and ILR to counter information leakage.

No heavyweight instrumentation exposed to the runtime.

Process-based isolation to recover from run-time ASRR errors.
Enhanced Operating System Security
Through Efficient and Fine-grained
Address Space Randomization

Thank you!
Any questions?

Cristiano Giuffrida, Anton Kuijsten, Andy Tanenbaum
{giuffrida,kuijsten,ast}@cs.vu.nl

Vrije Universiteit Amsterdam