Deterministic Behaviours are your Attacker's Friend
● The Moment of Failure

● Memory/Register state at moment of Failure

● What attackers do with this

● What can be done about it?
Oops, a bug is triggered by unexpected data

• Programmer mishandles some situation, and things go off the rails
  - Generally, code keeps running for a quite a while
  - Reading this, writing that, etc
  - Until it performs an illegal operation
  - SIGSEGV/SIGBUS, etc
• Attacker observes this condition
• Fuzzing is making it easier to find such bugs
What does it actually look like

- Static binary memory layout

```
<table>
<thead>
<tr>
<th>pc</th>
<th>Code</th>
<th>data/bss</th>
<th>heap</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pc</td>
<td>data/bss</td>
<td>heap</td>
<td>sp</td>
</tr>
</tbody>
</table>
```
What does it actually look like

- The Crash state

- Before 1999, every crash state had an identical footprint
What does it actually look like

- Dynamic binary

- Just more state. PIC code model
What the attacker knows

• In 1999
  - Crash-State 100% identical on attacker‘s test host and target host
  - Attack method could be iterated till perfection
  - Then deliver payload, own the target

• By 2010, ASR+W^X+stack-protector had added many unknowns and increased difficulty for the attacker
Consider Constant-Relative offsets

Attacker finds address in a reg or memory

Some shared object

The distance remains constant, and is thus a part of what the attacker knows.

Attacker wants this address
What the attacker does

• Before 1999, classic buffer overflows were the common technique

• After $W^X$ became common, ROP methodology became more common

• With ASR/ASLR now common, sprays and other non-turing-complete "damage the state“ methods become common.

• In JIT systems, the stack protector is usually absent.
What can be done about this?

*REMOVE INFORMATION & INCREASE CHANCE OF SIDE-EFFECT DAMAGE*

- Perturb address spaces
- Strict permissions (RWX, X-only, MAP_STACK, syscall-notX)
- Micro-architectural random cookies (stackghost, SROP)
- Self-protecting data structures (malloc, setjmp)
- Stackprotector / RETGUARD
- PIE, bindnow, ld.so unmapping
- Guards and trapsleds
Perturb address spaces

- PIC to PIE
- Kernel and ld.so do random layout
- All heap allocations via random mmap()
- ASR (not ASLR)

- libc/libcrypto/ld.so boot-time randomization
- KARL kernel boot-time randomization
- amd64 & arm64 VA+PA KASRL
Allocation guards and code trapsleds

- Guards are unmapped memory between objects
- More guards → improved chance an attacker‘s misfire kills him

- Trapsleds are illegal instructions in the instruction sequence, rather than the classic NOP sequences used in the past (for various reasons)
- Requires an attacker to precisely target gadgets, rather than sliding through NOPs
- (Used heavily in the RETGUARD design)
Perturbances - Before

- Dynamic binary

- Just more state. PIC code model, and maybe PIE
Perturbances – After

All objects random placement

Main program is PIE
ld.so random relink @ boot

Heap is guarded micro allocations

ld.so

libc.so

pc integer regs fpu regs sp
Perturbances – After (2nd run)

All objects random placement

- pc
- integer regs
- sp
- fpu regs

- ld.so
- libc.so

Main program is PIE
- ld.so random relink @ boot
- libc.so random relink @ boot

Heap is guarded micro allocations
There are still some knowns

- At moment of crashing
  - A specific register will still have a constant value
  - Pointers will still point at the same objects
- But many relative offsets are disrupted
- libc.so and ld.so boot-time randomization helps
- Cannot use part of a pointer as an integer constant in a calculation
- ROP gadgets & their locations are not known
Strict page permissions

- W^X
- .rodata
- Trying to develop X-only (to prevent blind ROP)
- MAP_STACK pseudo page-protection (prevents ROP-stack pivot into data memory)
- syscalls not permitted from writeable memory
History of our Stack-Protector

- 2001 – stack-protector protects functions $\geq 16$ bytes of local
- 2012 – one stack-protector value per shared-object
- 2014 – stack-protector-strong (more functions protected)
- 2015 – stack-protector values become read-only
- 2018 – RETGUARD: unique read-only stack-protector value per function (all functions protected to eliminate terminator-gadget)
Micro-architectural cookies

- Stackghost (sparc64)
- SROP mitigation (cookie in sigcontext)
- setjmp/longjmp cookie
Self-protecting data structures

- `atexit()` chain storage is write-protected
- `malloc()` tracking datastructures are out-of-band
- Large number of paranoia features in `malloc()` and `free()`
Remove ROP gadgets

• SROP eliminated
• RETGUARD
• OpenBSD/arm64 has no RET-ROP gadgets!
• X86 instr Polymorphic RETs can be significantly reduced (nearly eliminated)
• ld.so unmapping when finished (dlopen gadgetry)
• crt0 gadgetry cleanup
The role of privsep, fork + exec

- Privsep: Rewrite programs and structure them in a micro-kernel way
- Fork+exec: create new & unique address spaces whenever possible
- Moment-of-Failure knowledge from one privsep process, isn't applicable in another privsep process
• Cost of change must be evaluated

• As machines get faster, it is reasonable to use a larger portion of the cpu for self-protection
Pledge and unveil

- If the attacker manages to execute code in your environment
  - Take away as many system calls as possible
  - Restrict the filesystem space available
Conclusions

● Situation is much improved since 1999

● A Full-Stack approach

● Many other systems matching our trajectory

● Open-ended mission: More interesting work to do.