


Guarded Control Stack on arm64

Challenges in Enabling Shadow Stack
Support for CRIU

GCS - intro

- Armv9.4-A feature 
- No silicon out (yet)
- Separate stack (of return addresses) -> protected VMA
- On RET → [LR] is verified against GCS top
- GCSPR_ELO → GCS top
- GCS grows low ↗ high
- Host support auxval AT_HWCAP → HWCAP_GCS entry (glibc uses that too)

GCS - use case

CET (x86):

- CRIU based impl. → shstk (Mike Rapoport)
- Arm64 GCS = new return-address protection → CRIU must adapt

GCS (arm64):

- Hardens against ROP attacks
- Main difference GCS cannot be **re-enabled** once **disabled**

#1

(challenge)

GCS — Environment

- HW: none
- QEMU: not impl. (it is now)
- Steve Capper's MiniDebConf Yocto build using FVP
- Tried to re-create in Debian (applied exact patch+revisions)
 - patch drift?
- Fedora Rawhide had it all patched

GCS — Feature Verification



- Bundle = (FVP + patched kernel, binutils, gcc, glibc, and gdb)
 - but does it work?
- GCS Kernel tests
- GCSPR_EL0 readable via MRS
- basic-gcs.c → prctl
 - enabling | set_status
 - map_guarded_stack
- gcsstr.S → writes to shadow stack
- libc-gcs.c → has GCSSES1 tests
 - critical to switch from **tmp** → original GCS (during restoration)

TEST FILES:

- asm-offsets.h
- **basic-gcs.c**
- gcs-locking.c
- gcspushm.S
- gcs-stress.c
- gcs-stress-thread.S
- gcsstr.S
- gcs-util.h
- libc-gcs.c
- Makefile

(linux-v6.15-rc7/tools/testing/selftests/arm64/gcs)

GCS — Kernel Functions

FILE: arch/arm64/mm/gcs.c

arch_set_shadow_stack_status()	enable/disable GCS mode (prctl)
--------------------------------	---------------------------------

map_shadow_stack() [SYSCALL]	Map shadow stack VMA
------------------------------	----------------------

alloc_gcs()	Helper function calls do_mmap with VM_SHADOW_STACK
-------------	--

FILE: arch/arm64/kernel/signal.c

restore_gcs_context()	Validate + apply GCS state from the sigframe
-----------------------	--

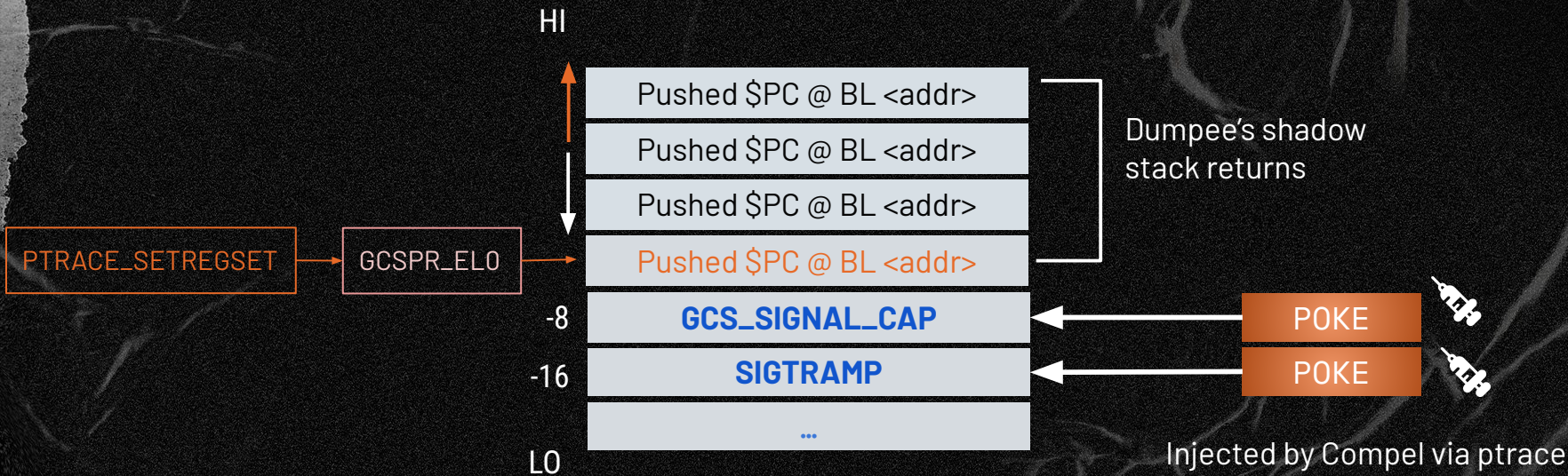
gcs_restore_signal	Check + consume the token on rt_sigreturn
--------------------	---

rt_sigreturn() [SYSCALL]	RT_SIGRETURN
--------------------------	--------------

GCS — CRIU dump

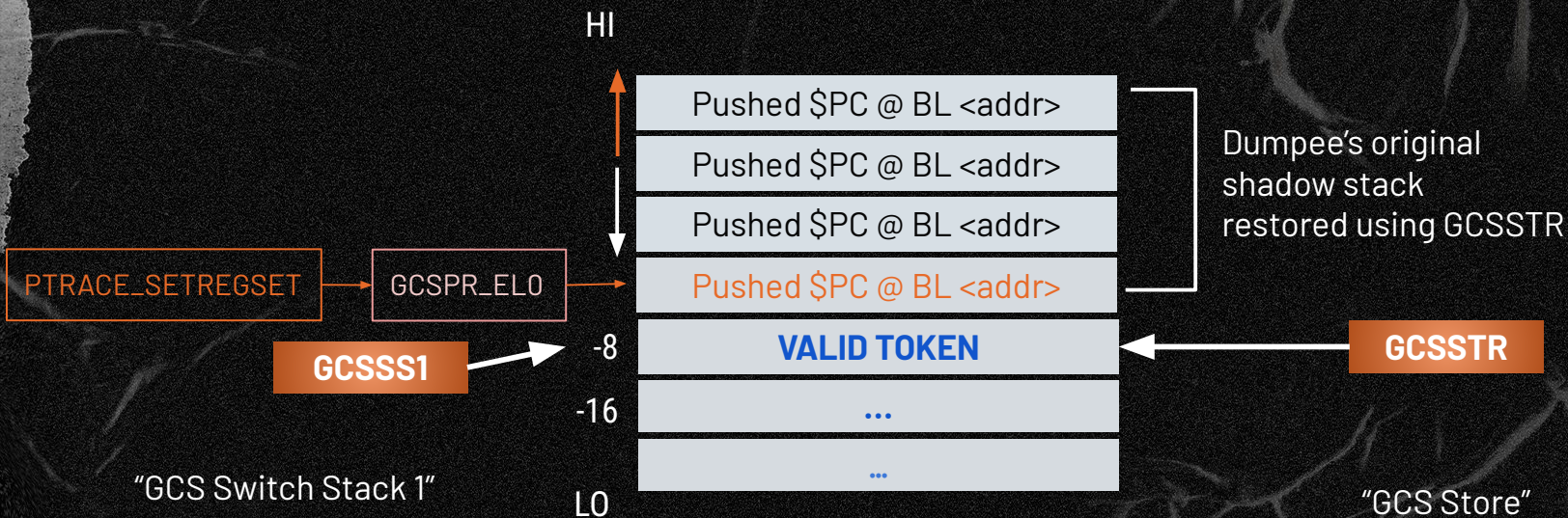
- Setup GCS (registers + memory) in the dumpee
- Plant gcs_context → sigframe
- Ensure rt_sigreturn doesn't fail
 - cap token @ GCSPR_ELO - 8
 - sa_restorer @ GCSPR_ELO - 16
- Injects token + restorer using ptrace
- GCS_ENABLE= flags
- GCSPR_ELO → point to sa_restorer (parasite instruction pointer)
- (will not damage the cap_token or sa-restorer)

GCS — CRIU dump



★ required to setup for `compel_cure` to work until the end

GCS — CRIU restore




GCS — Kernel Debugging

- printk >>
 - restore_gcs_context
 - restore_sigframe
 - SYSCALL_DEFINE(rt_sigreturn)
 - do_el0_gcs(struct pt_regs *regs, unsigned long esr)
 - alloc_gcs
 - SYSCALL_DEFINE3(map_shadow_stack
 - arch_set_shadow_stack_status(
 - SYSCALL_DEFINE5(prctl
- Macro to filter out by comm=env00
- to avoid noise from other programs
- Currently exploring FTrace and other dynamic tracing interfaces

GCS — From printk to Ftrace

- started with → printk
- Goal:
 - observe GCS VMA allocations, expected cap token etc..
 - try mostly dynamic ftrace points
 - keep tracing instrumentation lean and maintainer-friendly
- Constraint → inlined functions (unreachable by dynamic tracing)
- Best candidate for a justified static → cap-token mismatch information



potential
improvement

GCS — Ftrace setup probes

```
# kprobes: alloc + map_shadow_stack
echo 'p:kprobes/gcs_alloc alloc_gcs addr=%x0 size=%x1 comm=$comm' \
>> kprobe_events
echo 'r16:kprobes/gcs_alloc_ret alloc_gcs retval=$retval' \
>> kprobe_events

echo 'p:kprobes/map_ss __arm64_sys_map_shadow_stack \
      addr=+0(%x0):u64 size=+8(%x0):u64 flags=+16(%x0) comm=$comm' \
>> kprobe_events
echo 'r16:kprobes/map_ss_ret __arm64_sys_map_shadow_stack \
      retval=$retval comm=$comm' \
>> kprobe_events

# fprobes: track EL0 mode + GCS state
echo 'f:fprobes/gcs_status_entry arch_set_shadow_stack_status \
      task=task arg=arg' \
>> dynamic_events
echo 'f:fprobes/gcs_set_el0_mode_entry gcs_set_el0_mode \
      task=$arg1 mode=$arg2 \
      thread_gcspr_el0=task->thread.gcspr_el0:x64' \
>> dynamic_events
echo 'f:fprobes/gcs_set_el0_mode_exit gcs_set_el0_mode$return' \
>> dynamic_events
```


GCS — Ftrace

```
TRACE_EVENT(gcs_restore_sigcap_invalid,  
    TP_PROTO(u64 gcspr_el0, u64 cap, u64 expected),  
    TP_ARGS(gcspr_el0, cap, expected),  
    TP_STRUCT__entry(  
        __field(u64, gcspr_el0)  
        __field(u64, cap)  
        __field(u64, expected)  
    ),  
    TP_fast_assign(  
        __entry->gcspr_el0 = gcspr_el0;  
        __entry->cap        = cap;  
        __entry->expected    = expected;  
    ),  
    TP_printk("gcspr_el0=%#llx cap=%#llx expected=%#llx",  
        __entry->gcspr_el0, __entry->cap, __entry->expected)  
);
```


GCS — prctl default size

- arch_set_shadow_stack_status → no size arg
- arch_shstk_trampoline → **prctl** → PR_SET_SHADOW_STACK_STATUS ← 1st
 - restoree gets large [ss] VMA
- gcs_restore → restorer gets 1 page [ss] VMA (temporary) ← 2nd
- unmap (temporary 1_page sizedd)
- restore original shadow stack GCSSTR ← 3rd
- rt_sigreturn original process

potential
improvement

— GCS — mremap → VMA [ss]

- 1st → CRIU → pre-maps VMAs
- 2nd → CRIU → mremap(s) VMAs
- Initially we were not sure if mremap + shadowstacks = OK
- Kernel selftests extended to prove
- GCSSS1 to mremap-ed shadow stack
- also verified stack switching token contract

GCS — Conversion Interface

- ability to convert regular VMA → shadow stack VMA
- removes need for sequential (by doubleword) restoration of original shadow stack (GCSSTR)
- maybe madvise on VMA?

potential
improvement

GCS — Potential Improvements

CHANGE	PROBLEM	SOLVES
Shadow Stack Conversion Interface	Currently have to alloc	Would simplify CRIU restore and avoid costly remaps
Enabling shadow stack without kernel auto-allocating it	Large default. CRIU has to cleanup the extra mapping the reallocate with a specific pointer.	Allows re-use of pre-allocated user shadow stack, instead of having to deal with the auto-created one
FTrace cap token tracepoint	Hard to see why token validation failed	Lets developers trace bad/forged caps
mremap GCS test	No test that shadow stack survives move	Ensures kernel preserves GCS invariants on mremap

Thanks!

Mentors:

Alexander Mikhailsyn
Andrei Vagin
Mike Rapoport

Do you have any questions?

svilenkov@gmail.com | [linkedin.com/in/svilenkov/](https://www.linkedin.com/in/svilenkov/)

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**



Potential Improvements