# At ARMs Length Yet So Far Away

Brad Spengler

Open Source Security, Inc.

October 2013

H2HC

# The Story of KERNEXEC and UDEREF on ARM

# What is it?

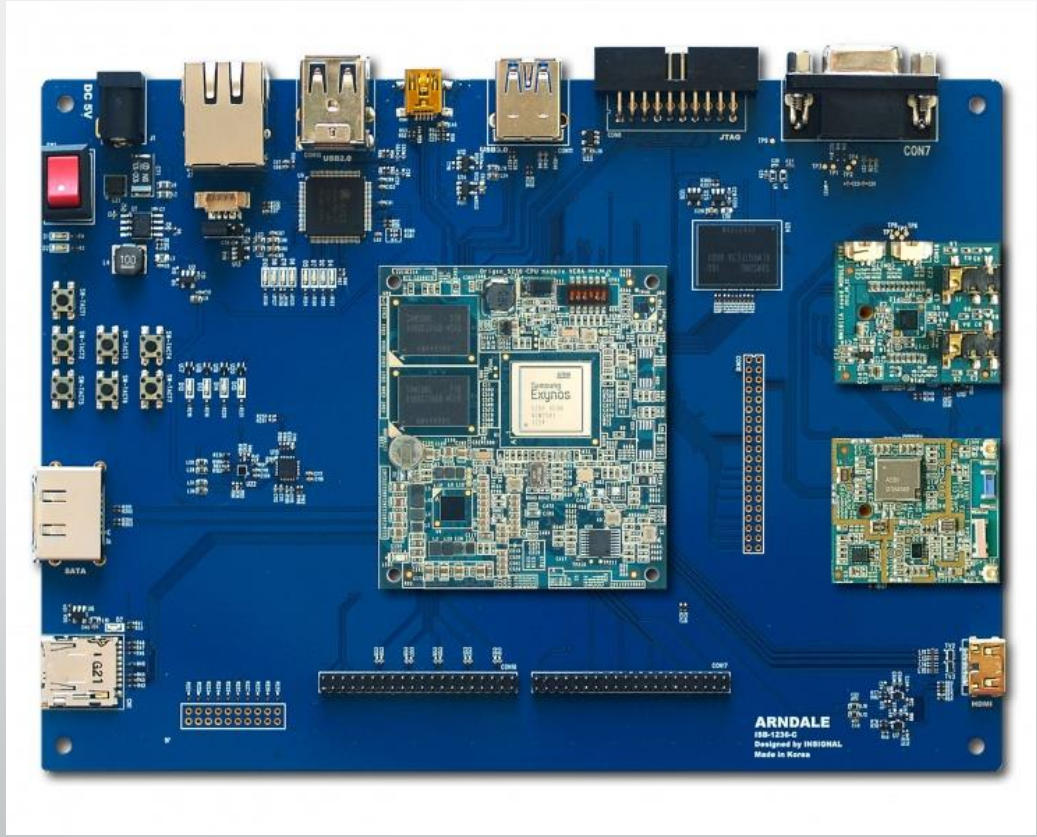
- ARM port of two of PaX's crucial arch-specific kernel self-protection features
- UDEREF: prevents accidental/malicious direct userland access via the kernel
  - NULL derefs
  - Poorly-chosen "magic" poisoned pointers
  - OOB index + trivially forged structs (PERF_EVENTS)
- KERNEXEC: remove RWX memory from the kernel
  - Implies no execution of userland memory
  - Protects against effectively-RWX memory via virtual aliases
  - Allows for read-only protection above and beyond simple 'const'

# Why?

- Upstream kernel self-protection features are non-existent
- Some vendor kernels have CONFIG_STRICT_MEMORY_RWX
  - Config description is a lie
  - Implementation is a joke
  - Lets vendors mark a checkbox
- Self-demonstrate quickly applying security concepts to an arch mostly “new” to me, armed only with the manual
- Spite
  - See last year’s H2HC presentation

# Beginning steps

- Acquired an Arndale development board (with Linaro userland)
- Samsung Exynos5, ARMv7, Cortex A15



# Beginning steps

- Started with ARMv7 since it supports Privileged Execute-Never (PXN)
  - Think x86 SMEP
  - PXN didn’t exist upstream yet, so I wanted to add support
- Focused on Large Physical Address Extension (LPAE) support first
  - 3-level paging structures instead of 2-level
  - More uniform layout of fields (easier to work with)
  - Think x86 PAE

# KERNEXEC on ARM LPAE

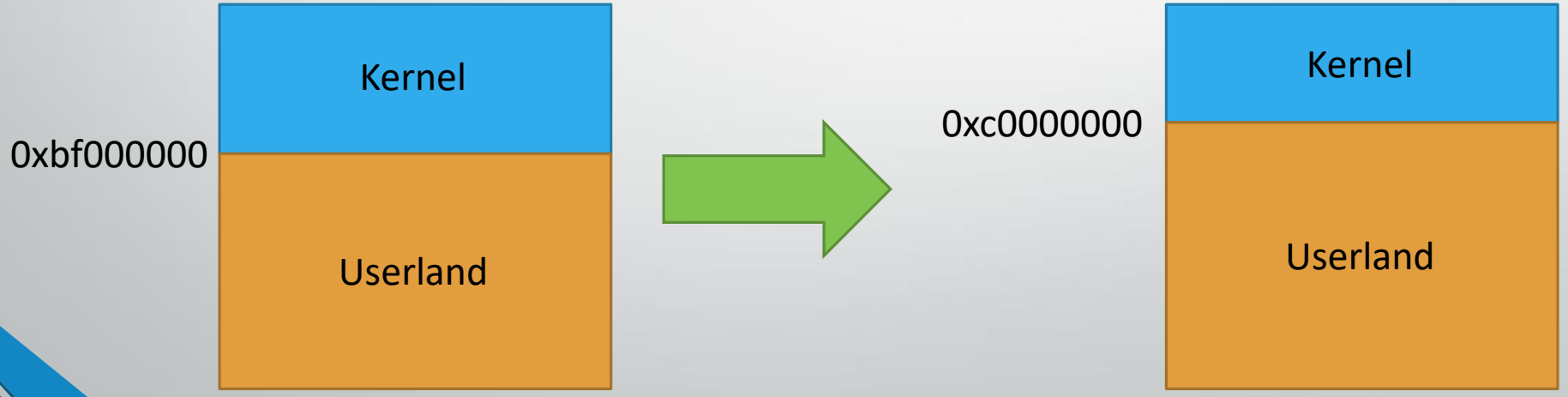
- arch/arm/mm/mmu.c handles setup of protections on most kernel mappings
  - mem_types array – base domain/page protection information for each level descriptors
  - build_mem_type_table() – amends information in mem_types with additional flags based on CPU capabilities
- Most used is MT_MEMORY, used for RWX kernel mappings
  - Completely eliminated, replaced with MT_MEMORY_RW and MT_MEMORY_RX to fail safely during forward porting
- Modified kernel linker script to group up sections with same protections
- Boot with weakened protections on the kernel image, lock in the final protections when freeing initmem
  - __read_only

# KERNEXEC on ARM LPAE

- Problem! PaX allows temporary suppression of page protections to allow privileged code to write to read-only areas
  - pax_open_kernel(void) / pax_close_kernel(void)
  - LPAE seemingly offers no way to do this as on x86
  - Creating temporary aliases would require per-cpu pgds to be secure
  - Would need to muddy up all open/close calls with arguments that would be completely ignored on x86
- Punted on this
  - Moved on to LPAE UDEREF, but this set the seed for an approach that would work for most modern ARM users

# UDEREF on ARM LPAE

- Modules are located at 0xBF000000, need to move to match Translation Table Base Register (TTBR*) granularity
- Now TTBR0 fully covers userland, TTBR1 fully covers kernelland
- Can disable userland access by disabling TTBR0 and changing ASID



# UDEREF on ARM LPAE

- Performed a quick test demonstrating previous technique and need for changing ASID
- Moved on at this point
  - Got lazy and didn’t feel like rewriting ASID generation code
  - Started disliking LPAE already for its inability to support the tighter KERNEXEC
  - Could split ASID space in half, or perhaps something smarter?
- I discovered after writing the blog that the previous description covers exactly how Apple iOS’ UDEREF-like feature works

# KERNEXEC for ARMv6+

- Able to reuse most of the work put into KERNEXEC for LPAE
  - CPU/LPAE-specific details mostly abstracted out by use of #defines
  - PMD_SECT_RDONLY
  - Found an upstream deficiency here
- On ARMv7 we can still use PXN to prevent userland code exec from kernel
  - It’s not as fine-grained as with LPAE, but it ends up not mattering
- Without LPAE, we have a much more powerful feature to exploit
  - Domains!

# KERNEXEC for ARMv6+

- Domain Access Control Register (DACR)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

  - 16 domains (Linux only uses 3)
  - Each domain supports several access types:
    - DOMAIN_NOACCESS – reject access regardless of page protections
    - DOMAIN_CLIENT – obey normal page protections
    - DOMAIN_MANAGER – ignore any page protections
- Domain is a 4-bit field in page table entries
- Important: domain also included in TLB entry, DACR always consulted

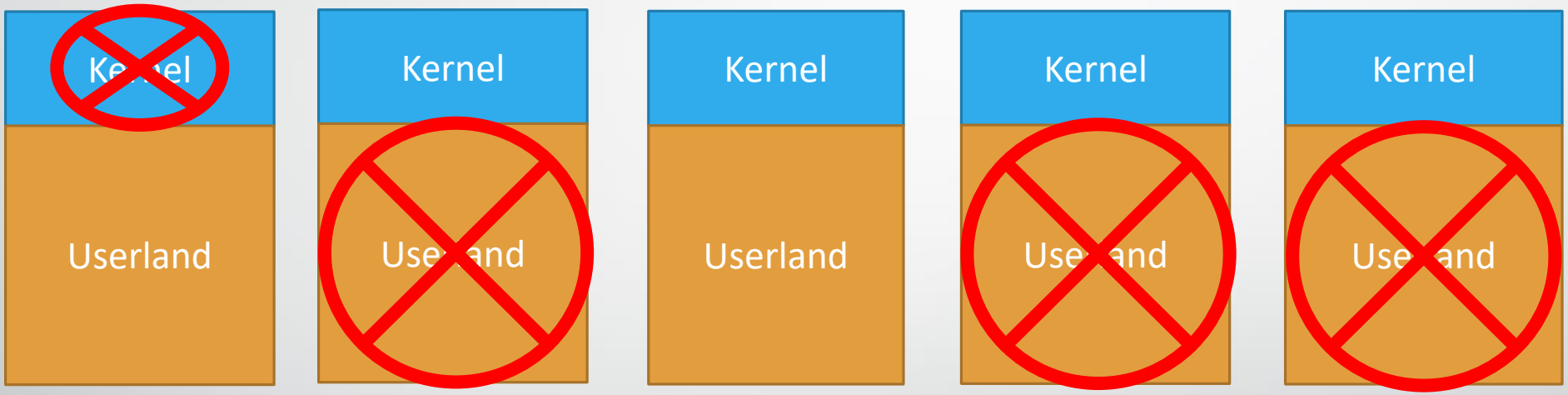
# KERNEXEC for ARMv6+

- Domains are an extremely powerful feature
  - Ability to block access to nearly-arbitrary memory ranges/”kinds” of memory on a per-CPU basis
  - Possible KERNSEAL uses
  - What else could you think of for this?
- For KERNEXEC, main use is to solve the pax_open/close_kernel problem
  - Set access of kernel domain to DOMAIN_MANAGER on open
  - Switch back to DOMAIN_CLIENT access on close
  - But….

# UDEREF for ARMv6+

- Again using domains
  - pgalloc.h:#define _PAGE_USER_TABLE (PMD_TYPE_TABLE | PMD_BIT4 | PMD_DOMAIN(DOMAIN_USER))
  - So by setting the user domain to DOMAIN_NOACCESS, we cut off all access to userland
- Kernel has approved userland accessors
  - copy_*_user()
  - strnlen_from_user()
  - csum_partial_copy_from_user()
  - …
- Introduce pax_open_userland() and pax_close_userland() to these

# UDEREF for ARMv6+




Executing in userland

Ambient kernel permissions

Userland accessor copying to userland (USER_DS)

Userland accessor copying to kernel (KERNEL_DS)

Interrupt handler code servicing interrupt during userland accessor

# KERNEXEC/UDEREF for ARMv6+

- Via domains we achieve protection equivalent to PaX's KERNEXEC/UDEREF on i386
- This makes pipacs and myself happy ☺
  - I hate the shadow region on the non-PCID version of amd64 UDEREF
- With both features enabled, 1.6% performance hit observed in NGINX Benchmark 1.0.1.1, below stddev of test
  - This performance can be improved further, my assembly was written for clarity

# Notes on 3.10 upstream ARM fixes

- Special page installed into each task, sigreturn stubs located at system-wide “random” offset within that page
  - Installed page is subject to mmap randomization
- I don’t know of any userland that can work without the kuser helpers, requiring one to enable the option that adds them all back
  - Leaves fixed-address vector map accessible
  - These helpers are still necessarily at fixed addresses (thanks to glibc/toolchain)
  - My Linaro user perhaps needs the fewest, just get_tls()
- Kernel address leaks from the vector page should be dead now
  - Relevant code/data moved to an adjacent kernel-only page

# Our ARM vector page fixes

- As part of KERNEXEC work, kernel RWX on the vector page via virtual aliasing (one RW, another RX) was eliminated
- No special page installed into each task for sigreturn stubs
  - Kernel controls the address of the sigreturn stub userland will try to execute
  - Unique random inaccessible kernel address assigned to each task's mmu_context struct
  - We cause userland to try to execute at this random address, catch the fault, and perform the sigreturn
- Vector page is inaccessible to userland
  - We emulate get_tls() in the kernel

# Testing

- Previous mentioned kernel “backdoors” to trigger exploit-like activity
- Created page table dumper for both short and long mode descriptor format
  - !LPAE: https://grsecurity.net/~spender/kmaps-arm-v6.c
  - LPAE: https://grsecurity.net/~spender/kmaps-arm-lpae.c
  - Uses /dev/mem
  - Finds and reports pages of memory that are RWX through virtual aliasing
- Verified full removal of RWX from the kernel
- Verified inability to execute/access userland directly from the kernel

# Testing

root:~$ ./test

PaX: Kernel tried to access userland memory at 0x00008010, fsr=00000206

Internal error: : 206 [#1] PREEMPT SMP ARM

Modules linked in:

CPU: 0  Not tainted  (3.7.1-grsec-00071-gac214bd-dirty #49)

pc : [<c02296a4>]  lr : [<c02295b4>]  psr: 60000013

sp : ee847f90  ip : 30c7387d  fp : 00000000

r10: 00000000  r9 : ee846000  r8 : c0206128

r7 : 000000d5  r6 : 00007a69  r5 : b390a788  r4 : 00000000

r3 : 00008000  r2 : 40003000  r1 : b390a8c4  r0 : 00007a69

Flags: nZCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment user

Control: 30c5387d  Table: 40003000  DAC: fffffffd

Process test (pid: 2450, stack limit = 0xee846238)

[ … ]

Code: e1a00007 e8bd41f0 ea0041fd e3a03902 (e5934010)

Kernel panic - not syncing: grsec: halting the system due to suspicious kernel crash caused by root

# Lessons Learned

- Spite fails to motivate when realization sets in that the code must be maintained for free forever ☹
- Fragmented/old userland is a maintenance nightmare for the kernel
- Mobile Linux vendors care more about checking a box than real security improvements
  - Expect Apple to continue to dominate over them, SEAndroid/KNOX are just same tired “security = access control” frauds
- Focus on fundamentals, not fads
  - Makes it easy to apply security to new platforms

# Exploit Weaponization

(For the Linux Kernel)

(Real quick-like)

# Why?

- Because namespace/virtualization/LSM usage is increasing with little discussion of tradeoffs or importance of kernel self-protection
- Because ring-0 can do whatever it wants
- Because I've been told making weaponization/reliability information public reduces the value of exploit sellers
  - I am all for this!
- Because it's embarrassingly easy, as you'll see

# Disable SELinux

- Set security_ops to &default_security_ops (always works)
- Thanks to some SELinux “code cleanup” you can also return to void reset_security_ops(void) (works for all other LSMs too)
- If CONFIG_SECURITY_SELINUX_DEVELOP enabled, you can also modify ‘selinux_enforcing’ (read by getenforce tool when reading /selinux/enforce, also in /selinux/status)
- By patching sel_read_enforce and selinux_kernel_status_page, you can put SELinux into permissive mode while making userland think it’s in enforcing mode

# Disable AppArmor

- Set security_ops to &default_security_ops
  - Or return to reset_security_ops
- Older kernels had some toggles: apparmor_enabled, apparmor_audit, apparmor_logsyscall, apparmor_complain
- Newer kernels (3.x) uses new variables: aa_g_profile_mode, aa_g_audit, aa_g_audit_header, aa_g_logsyscall, aa_g_lock_policy

# Disable IMA

- Set security_ops to &default_security_ops
  - Or return to reset_security_ops
- Patch out ima_bprm_check, ima_file_mmap, ima_path_check, and ima_file_check to all return 0 (\x31\xc0\xc3)

# Disable TOMOYO/all other LSMs

- Set security_ops to &default_security_ops
  - Or return to reset_security_ops

# Disable Auditing

- Clear audit_enabled

# Disable No-New-Privs (NNP)

- Bit field located between current->personality and current->pid (both known values)
- Other fields are unimportant, just clear all 4 bytes

# Break out of user namespaces

- Perhaps surprisingly, my commit_creds(prepare_kernel_cred(NULL)) technique does it automatically
- struct cred {

  …

  struct user_namespace *user_ns; /* user_ns the caps and keyrings are relative to. */

  …

  }

# Break out of chroots

- Get task_struct ->fs offset either through simple static analysis on kernel image or finding init_fs offset within init_task
- Find offset of root and pwd within fs_struct
- Call set_fs_root(current->fs, init_fs.root), set_fs_pwd(current->fs, init_fs.pwd)

# Break out of vserver

- Find offsets of xid, vx_info, nid, nx_info in task struct
- Will be 0 in init_task, but set in a confined process
- Clear the fields

# Break out of OpenVZ

- Call prepare_ve0_process(current)

# Reliability under Xen

- Don’t blindly change cr0.WP and attempt to modify kernel code, it will cause a GPF
- Call make_lowmem_page_readwrite(addr) instead
- Clean up with make_lowmem_page_readonly(addr)

# Reliability under CONFIG_DEBUG_PAGEALLOC

- Don’t blindly scan through kernel memory, kernels with this option enabled have been observed to have a guard page in the kernel image
- Enlightenment parses page tables to determine safe regions to scan

# Reliability under CONFIG_KALLSYMS

- Unknown kernel, no vmlinux, no /proc/kallsyms, no System.map?  No problem!
  - Assuming ring0 execution can be obtained without them
- Payload reliability obtained by using the kernel's own symbol tables in memory
- Enlightenment finds the kernel's own symbol resolution routines and can thus resolve module symbols as well

# Reliability when returning to userland

- Make sure the userland code is locked into physical memory with mlock()
  - Unprivileged users can lock 64KB
- Linux is not Windows!  No kernel memory is paged, attempting to access non-present memory (outside of exception-handled areas) will result in a visible oops, or worse
- More likely under low memory, memory pressure, high exploit memory reqs, higher time between allocation and use
- A decade of exploits returning to userland, and no one gets this right!
- How was my PERF_EVENTS exploit so reliable despite the 64KB lock limit?
  - Read the enlightenment source for this one ;)

# X86 PERF_EVENTS Exploit

[!] Array base is 0xc1a57a60

[!] Detected structure size of 12 bytes

[!] Targeting 0xc1a69b10

[+] Got ring0!

[+] Detected 2.6/3.x style 8k stacks, with current at 0xf1f2cc20 and cred support

[+] Disabled security of : AppArmor LSM

[+] Found ->fs offset at 0x388

[+] Broke out of any chroots or mnt namespaces

[+] Got root!

root@ubuntu:/home/spender/enlightenment#id

uid=0(root) gid=0(root) groups=0(root)

root@ubuntu:/home/spender/enlightenment#uname -a

Linux ubuntu 3.5.0-23-generic #35~precise1-Ubuntu SMP Fri Jan 25 17:15:33 UTC 2013 i686 i686 i386 GNU/Linux

# References

ARM Manual

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0406c/index.html

# Questions?

All code described in this talk is available at https://grsecurity.net

More details available on blog: https://forums.grsecurity.net/viewtopic.php?f=7&t=3292

Thanks to Rodrigo and all our sponsors!