1 Google Android - 'rkp\_set\_init\_page\_ro' RKP Memory Corruption

1. Introduction: As part of Samsung KNOX, Samsung phones include a security hypervisor called RKP (Real-time Kernel Protection), running in EL2. This hypervisor is meant to ensure that the HLOS kernel running in EL1 remains protected from exploits and aims to prevent privilege escalation attacks by "shielding" certain data structures within the hypervisor.
2. Exploit steps:

During the initialization of RKP, a special command can be issued by EL1 kernel in order to mark the RKP read-only page as such in the stage 2 translation table. This command, "rkp\_set\_init\_page\_ro" (command code 0x51) has the following approximate high-level logic:

\_\_int64 rkp\_set\_init\_page\_ro(struct args\* args\_buffer)

{

unsigned long page\_pa = rkp\_get\_pa(args\_buffer->arg0);

if ( page\_pa < rkp\_get\_pa(text) || page\_pa >= rkp\_get\_pa(etext) )

{

if ( !rkp\_s2\_page\_change\_permission(page\_pa, 128LL, 0, 0) )// RO, XN

return rkp\_debug\_log("Cred: Unable to set permission for init cred", 0LL, 0LL, 0LL);

}

else

{

rkp\_debug\_log("Good init CRED is within RO range", 0LL, 0LL, 0LL);

}

rkp\_debug\_log("init cred page", 0LL, 0LL, 0LL);

return rkp\_set\_pgt\_bitmap(page\_pa, 0);

}

As we can see above, the function receives an address in the kernel VAS, and converts it to a physical address by adding a constant offset to it (the virt\_to\_phys offset for the kernel VAS). Then, the function proceeds to mark the resulting physical address as read-only and non-executable in the stage 2 translation table. Finally, the function proceeds to unset the bit in the RKP page-table bitmap corresponding to the given address. This is meant to indicate to EL1 that the address is protected by a stage 2 mapping.

1. Vulnerabilities:

However, the function fails to validate the bounds of the given virtual address (or the resulting physical address). This means that an attacker can supply any arbitrary address and the function will accept it as valid input. Similarly, the implementation of "rkp\_set\_pgt\_bitmap" performs no such validations:

signed \_\_int64 \_\_fastcall rkp\_set\_pgt\_bitmap(\_\_int64 phys\_addr, unsigned char set\_or\_unset)

{

unsigned long phys\_off = phys\_addr - 0x80000000LL;

unsigned long bitmap\_index = (phys\_off >> 18) & 0x3FFFFFFFFFFFLL;

if ( !rkp\_pgt\_bitmap )

return 0LL;

unsigned long bit\_offset = (phys\_off >> 12) & 0x3F;

if ( set\_or\_unset & 0x80 )

{

spin\_lock(&rkp\_bitmap\_spinlock);

\*(rkp\_pgt\_bitmap + 8 \* bitmap\_index) |= 1LL << bit\_offset;

spin\_unlock(&rkp\_bitmap\_spinlock);

result = 1LL;

}

else

{

spin\_lock(&rkp\_bitmap\_spinlock);

\*(rkp\_pgt\_bitmap + 8 \* bitmap\_index) &= ~(1LL << bit\_offset);

spin\_unlock(&rkp\_bitmap\_spinlock);

result = 1LL;

}

return result;

}

The RKP page-table bitmap is only 0x20000 bytes large (each bit denotes a 4KB page, resulting in a supported range of at-most 0x100000000 bytes). The base physical address for the bitmap is the physical base address for the kernel range - 0x80000000.

This means that if an attacker supplies any virtual address that is converted to a physical address not in the range of 0x80000000-0x180000000, the resulting "bitmap\_index" will not be within the bitmap's bounds, causing the function to modify a bit out-of-bounds.

An attacker can use this in order to specifically craft an input virtual address so that the resulting calculated "bitmap\_index" will have any arbitrary value, thus resulting in a modification at an arbitrary offset from the base of the page-table bitmap, within the context of RKP.

As the bitmap resides directly before RKP's code, an attacker can trivially use this primitive in order to modify the code or data pages belonging to RKP, thus gaining privilege escalation from EL1 to the context of RKP.

I've verified this issue on an SM-G935F device, build version "XXS1APG3". The RKP version present on the device is "RKP4.2\_CL7572479".

Proof of concept for the RKP memory corruption in "rkp\_set\_init\_page\_ro".

This PoC modifies an instruction within RKP's address space by repeatedly calling "rkp\_set\_init\_page\_ro" with faulty input addresses.