0x41con, Timisoara

Hacking the PS4

From zero to ring zero in two easy steps

Who am I?

- Luca Todesco aka. qwertyoruiop
  - used to be @qwertyoruiopz on Twitter but it's now suspended
- Background in iOS vulnerability research & exploitation
  - Fan of full chain development, wrote a few over the years
- Recently focusing on running infrastructure while on a security sabbatical
  - From full-chain to block-chain
  - My network has machines running a total of 220 GPUs pulling ~30kWh
    - Actually located here in Timisoara in an industrial park
    - Completely self-built/engineered
- ~~Tried~~ Procrastinated on full chaining the PS4 multiple times over the years

History of my attempts at pwning the PS4

- Bought a PS4 close to launch day to play games with it
  - GTA 5 single-player mods fan on PS3
  - PS4 is not jailbroken, so I kept my PS3 around to play with mods.
- Never bothered to try to do anything with it for a while

History of my attempts at pwning the PS4

- A few years later, multiple groups got badIRET running on PS4 kernel
  - This gets me interested
- Contacted CTurt (one of the various people involved with one of the badIRET attempts) to ask for info and offering my time for the cause
  - Became friends with CTurt, he lets me know of some crashes he'd found fuzzing
    - Investigate on one that looks a lot like a integer overflow

History of my attempts at pwning the PS4

- Investigate on one that looks a lot like a integer overflow
  - Invoking sys_dynlib_prepare_dlclose with a large count (0x80000000) crashes the kernel
    - Yes, the usermode dynamic linker runs in kernelmode
  - Immediately start looking at syscall code, multiple 64 bit integer overflows present (but not exploitable, as they'd require having the syscall return data, but syscall always fails)
    - Doesn't explain the 0x80000000 testcase crashing
  - Comex chimes in and suggests that (uint64_t)count * sizeof(int) is used as argument to malloc() -> maybe malloc() fails for >4G allocations?

History of my attempts at pwning the PS4

- Turns out malloc() had an integer overflow in FreeBSD 9
  - The size of the allocation is passed as a 64 bit integer
  - Two allocation strategies used for small and big allocations
    - UMA for small
    - Direct Alloc for big
  - UMA allocator used full 64 bits, but the direct allocator would actually truncate size to 32 bits
    - >4G is pretty large so Direct Alloc always used
      - All >4G allocations would be overflows in FreeBSD 9
  - Thanks @comex!

History of my attempts at pwning the PS4

- Turns out malloc() had an integer overflow in FreeBSD 9
  - Bug fixed on newer FreeBSD
    - Found accidentally by an user that had several terabytes of RAM and allocating pagetables would request >4GB from malloc crashing the kernel
  - PS4 still affected years later, and sys_dynlib_prepare_dlclose would trigger it and copyin()
    - I exploited this by copyin'ing 4GB+ into a page-sized allocation, but having unmapped memory in user mode to limit the actual copy to page+0x10 and overwrite a kqueue list

History of my attempts at pwning the PS4

- CTurt wrote an article describing the bug and my exploit technique
  - We did not release an exploit, but multiple implementations were made by third parties
- Doesn't affect anything badIRET didn't already affect, but the exploit was a lot simpler thus more reliable, and became the go-to bug for PS4 jailbreak for several years

History of my attempts at pwning the PS4

- Key takeaway is that PS4 security entirely relies on kernel security
  - No hypervisor
  - SAMU (Secure Enclave) is basically powerless
- But we need code execution
  - Thus, we take advantage of two easy steps from zero to ring 0:
    - First, we run a WebKit RCE
    - Then we run a FreeBSD LPE

Pwning the PS4

- 33c3 talk by fail0verflow renewed my PS4 interest
- A couple months later i give a shot at pwning PS4 webkit
  - Actually tricky as it's a fairly old version, and many bugs I had wouldn't work on it
  - Start auditing an old WebKit source tree

Pwning the PS4: Ring3

- Find a reentrancy bug in Function.prototype.apply()
  - Allows to call a function with N arguments of which only M are initialized
  - Converted to UaF by using uninitialized stack to store a reference to a JSObject - but after unwinding the frame, GC wouldn't count it, thus free, and uninitialized use brought reference back to life
  - Later revealed to have been killed (on recent webkit) by Lokihardt at some old P2O

Pwning the PS4: Ring3

- From UaF we can derive arbitrary read/write
- However Sony got rid of JIT on recent firmwares
  - We can only ROP
    - Develop JS framework that scans memory for gadgets and allows arbitrary function calls from JavaScript
      - Use JS VM to implement logic

Pwning the PS4: Ring3

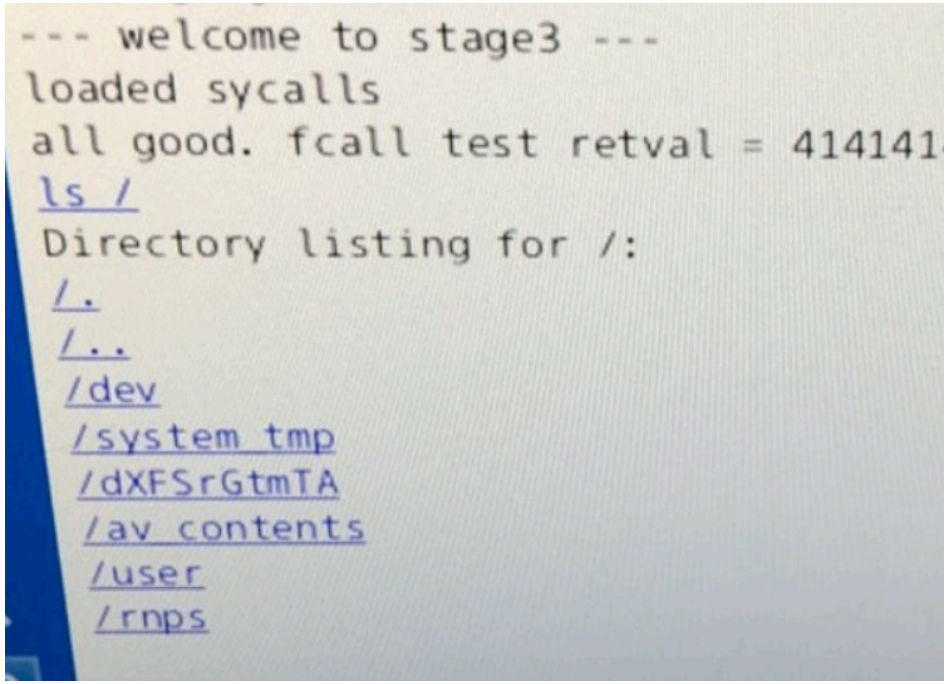
- However issuing syscalls doesn't seem to work
- Calling the syscall wrappers in libkernel works, but ROPing into the |syscall| instruction didn't
  - Actually worked but only when issuing the same syscall the wrapper would issue
    - Later revealed to be a check on the opcodes preceding |syscall| matching a given sequence
      - Scan for fixed sequence to build a syscall wrapper table and just call the right one

Pwning the PS4: Ring3

- Some syscalls are missing
  - Later revealed that WebKit links against libkernel_web, which doesn’t define all syscalls
    - Conflicts with the real libkernel so you can’t just dlopen it
    - Basically some sort of sandboxing as missing wrappers means you can't issue certain syscalls as long as W^X is enforced

Enumerating attack surface

- We can now issue (most) syscalls from javascript
- Time to look around
  - Implement a basic “ls” in javascript



Enumerating attack surface

- Not much in /
  - WebKit is a jailed process
- But we have a “virtual” /dev
  - ls /dev
- Look for the usual suspects
  - Spot /dev/bpf as an accessible device, remember @comex’s BPF_STX bug in early iOS
    - It’s supposed to be a root-only device on iOS/FreeBSD
    - But Sony made it world-accessible and even sandbox-accessible

Pwning the PS4: Ring0

- As bpf is usually root-only, I'd expect it not to be considered as LPE attack surface on FBSD
    - Less audited
- Likely to have *something*, and complex code is all over the place
    - VM with JIT support (albeit JIT is disabled on PS4)
        - Actual VM allows OOB stack access, but updating opcodes runs a validator to prevent it
            - TOCTOU: If the opcodes are changed after being set (ie. heap overflow), you can use the BPF VM to derive kASLR info leak and reliable code execution

Pwning the PS4: Ring0

- VM opcodes are loaded with an ioctl implemented by bpf_setf
- Function allocates new opcodes and frees old ones
  - Old allocation pointer is stored to a local variable before locking
  - If new instructions are valid or input size is 0, old allocation is free'd
  - Race condition: You can enter this function twice, and although the free itself is atomic, the pointer that is free'd will be the same across the two threads as it's copied before asserting locks
    - Double free
      - I think we can do better than this

Pwning the PS4: Ring0

- VM opcodes are parsed by bpf_filter
- Array of opcodes is passed as 1st argument
  - Implicit pointer copy to register by function call conventions
- bpf_filter will be executed on every incoming packet, so it must run unlocked or performance hit will be significant
  - But we can also execute it on-demand by calling write() on the bpf device

Pwning the PS4: Ring0

- At any point in time after the pointer copy it is possible to simply free the opcodes and replace them
  - The VM will run opcodes from a free'd buffer
    - We can replace free chunk with malformed opcodes to take advantage of stack OOB

Pwning the PS4: Ring0

- We need a heap spray
  - We can ioctl an invalid filter on a different bpf file descriptor. This will malloc(), copyin(), free().
    - Heap allocator doesn't taint free'd data, so it's an effective spray strategy
- We need to be able to spawn threads from ROP
  - Just pthread_create a thread running longjmp as pivot, implement complex logic by offloading to JS thread using locking

Pwning the PS4: Ring0

- We need to trigger the bug and to detect whether we won the race or not
  - Make valid bpf filter return 0 and invalid return 1, use write() return value as oracle
- We need to leak the kernel base (and dump kernel text to develop patchset)
  - kASLR wasn't present on earlier firmware, but it is now
  - SMEP is not supported by the APU
    - We can ROP into user mode gadgets of which we know the location

Pwning the PS4: Ring0

- So we create two threads
  - Get them looping over ioctl() calls
    - One on the target fd with a valid program
    - One on a different fd with an invalid program
- From JS thread we loop over write() call on the target fd
  - Break loop when return value is not -1
    - Make valid program return -1 and invalid return success

Pwning the PS4: Ring0

- Eventually the write() call returns success
  - We are now running the invalid opcodes!
    - Smash the stack and pivot into user mode chain
      - Call usermode memcpy to dump kernel
- We have a kernel dump to analyze now!
  - One useful gadget is CR0 set/get, which allows us to disable write protection to patch the kernel

Pwning the PS4: Ring0

- Define primitives: read/write/fcall
  - Just enter write() loop and replace the caller's stack frame with a small chain copying contents of pointer A into pointer B
    - Can derive read/write from here
  - pop rax / pop rdi / pop rsi / pop rdx / call rax; ret
    - Function call

Pwning the PS4: Ring0

- Patch mprotect to allow RWX maps
  - Finally we can run shell code
- Map and enter shellcode, call it once as kernel mode and once as usermode
  - Kernelmode step will perform required patches (breaking out of jail and backdooring ioctl() on some file descriptor to get kernel code exec on demand)
  - Usermode step will implement logic

Shellcode Tricks

- Manually writing shell code sucks
  - So just write it in C and compile as Mach-O for MacOSX.sdk
  - Replace first 4 bytes in mach-o with branch to some function that will act as entry point
- Linking
  - We can enable dlsym() privileges from kernelmode
  - We can redefine dyld_stub_helper to lazy link via dlsym()
    - And link non-lazy references / rebase at runtime
  - Create .tbd file for Sony functions

Modding games

- My aim is to mod GTA V, so we need to dump the game
- Implement task_for_pid() (just iterate over allprocs and return proc pointer for pid)
- Implement vm_read / vm_write to operate on foreign VM space
- Find game text base by sysctl()ing
- Dump VM

Modding games

- We need to allocate some memory on the remote task
  - Implement vm_allocate by changing the current thread's task into foreign process, calling mmap, changing back
    - Dirty but works!
- We need to hook some function on the game
  - GTA V is implemented as VM
    - Function pointer table for "native" functions can be hijacked to override some function
      - MOBILE_PHONE_CREATE was picked (called every time you pull the phone up in-game)

Modding games

- Implement RPC protocol that fills up an “action” buffer
- Write VM that processes the buffer and allows to invoke native functions (using pointer table we previously hijacked) and perform simple logic
- Run VM every time our in-game hook is called
- Make node.js frontend
  - Can now write GTA V scripts in javascript, will get serialized into intermediary format and interpreted when pulling phone up
    - **Can now create money out of thin air!**

Pwning the PS4 again: Ring0

- After a few months Sony “patches” kernel bug by removing bpfwrite()
  - We still have the double free
    - I get lazy and ignore this for a few months
- Exploit targeting 4.55 using this bug was later published by @SpecterDev

Pwning the PS4 again: Ring0

- Get bored of procrastinating
- Want to pop latest PS4 firmware again
  - Get random P0 webkit bug and exploit it
    - setAttributeNodeNS DOM UAF
      - Post exploitation unchanged, copy&paste
  - BPF Double Free race to Ring0
    - Just create one thread repeatedly ioctl()ing on the target fd to set a valid program
    - Repeatedly ioctl() on the same fd with the same program from JS thread to trigger bug

Abusing UMA

- Double free on UMA has interesting behavior
- Free list is out-of-band so we can free the same pointer twice and poison the freelist into having the same element twice
- Allocate a target object and spray, if lucky the target object gets overwritten by spray

Pwning the PS4 again: Ring0

- Target object is knote list just like dlclose bug
- Call close() on poisoned kqueue
  - Destructor is called on each knote
    - Replace pointer to gain RIP control
- Run infinite loop opcode
  - Thread hangs, so exploit works
- Run pivot to chain with infinite loop opcode
  - PS4 crashes - immediately think of SMAP (we're moving RSP to user mode)

Pwning the PS4 again: Ring0

- However APU doesn't support SMEP
  - No way it supports SMAP!
- Maybe Sony added checks on RSP at schedule time
  - Could race maybe?
- JOP seems to work, but exploit is not reliable enough to repeat it multiple times implementing logic in-between (like on the previous bug where every primitive would re-exploit the bug)
  - Pure JOP logic.. let's not go there please

Pwning the PS4 again: Ring0

- Reuse iPhone 7 ROP strategy from Yalu+mach_portal
  - Thread-safe and calling-convention aware, but most importantly pivot-less
- Use JOP to implement a simple loop
  - Loop logic based on deref&branch
  - Every iteration runs a function prolog followed by a branch
    - This pushes lots of stack frames on stack, padding RSP

Pwning the PS4 again: Ring0

- When loop is done, prepare call to memcpy with RDI = RSP, RSI = controlled pointer, RDX = size of pushed stack frame * number of iterations - 1
  - We overwrite all fake frames but one with ROP
- Memcpy will return into our first gadget, kickstarting the chain
  - At tail end of chain we just return into matching function epilog to resume clean execution by popping the one untouched frame
- RSP never pivoted
  - PS4 successfully runs the chain!

Ring 3 -> Ring 0 -> Ring 3

- Dump kernel, patch mprotect, run shellcode, define accessors to foreign address space
- Create money out of thin air again
  - But this time I want to be able to shoot bullets that spawn money on impact
  - Hook a function called on each frame
    - Check if bullets were shot, if so ray trace and spawn money bag object at impact coordinate
      - Become 15yo again

Thanks

- CTurt
- flatz
- Specter
- kiwidog
- Alex
- Comex
- m0rph3us1987
- CMX
- Lokihardt
- Shuffle2
- banty
- The FreeBSD project