2020: a new year to research iOS security

Liang Chen of Singular Security Lab(@SingularSecLab)

About me

- Director of Singular Security Lab
- Our Team:
  - Vulnerability research on OS kernel, browser, framework, application, hardware & secure boot chain, short distance protocols, etc.
  - Cutting-edge research on automatic vulnerability discovery technology
- Myself:
  - Focus on browser exploitation research, iOS/macOS privilege escalation research
  - Demonstrate several jailbreaks in the past few years
  - Speaker of many conferences (my 6th time at POC)
  - Winner of Mobile Pwn2own 2013 and Pwn2Own 2014
  - Leader of the “Master of Pwn” team at Pwn2own (3 times)

Agenda

- iOS security overview in 2019
- New mitigation & enhancement in 2020
  - Mitigation for checkera1n
  - Userland
  - Kernel
  - Hardware related
- Summary
- Demo

iOS in 2019

- Checkm8 & Checkra1n
  - Bootrom bug for A5-A11 devices
  - Unpatchable
- 0day jailbreak: LIGHTSPEED
  - Introduced in iOS 11, fixed
  - Reintroduced in iOS 13
- SockPuppet by Ned Williamson: fixed in 12.2.2 and reintroduced in 12.4
- oob_timestamp: Looks like unexploitable but Brandon Azad of GP0 made it
- An interesting year

iOS in 2019

- In iOS 12 era, iOS kernel exploit is not a problem
  - Fake port technique + cross zone attack via GC became a exploit “standard” for iOS kernel exploit
- In iOS 13 era, new mitigations tried to kill some exploitation techniques:
  - E.g zone_require is designed to kill “using fake port to achieve tfp0”
    - But it is buggy
  - Make kernel data patching hard. E.g move sandbox related pointer direct to KTRR protected RO regions
  - oob_timestamp defeat all above mitigations, showing how a bad-quality bug became exploitable
- And, with checkra1n, it looks like all A11- devices can be jailbroken forever

2020 is a big year for iOS/iPhone

When everyone thought checkm8/checkra1n is unpatchable

- It is partially mitigated on A10+, How?
  - Checkm8 is bootrom bug
  - Device is pwned at very early stage of boot, except for SEP
  - SEP is still trusted
  - Checkra1n exploited DFU stage, while normal iOS boot doesn’ t enter DFU
  - In DFU mode, SEP nonce is initialized. Normal boot doesn’ t initialize.
  - SEPOS can check if nonce is initialized to detect possible checkra1n attack
- Introduced in iOS 14
  - Need additional SEP bug to achieve jailbreak.

Userland: enhanced sandbox

- In the past, each iOS process has two types:
  - Sandboxed
  - No sandbox
- For no-sandbox process, you can do anything (in most cases):
  - Open most of iokit drivers
  - Access most of the files, including sensitive data like SMS, Mail, Photo, etc.
  - Only very few operations are limited:
    - process-exec in data folder is prohibited
    - Dynamic codesigning needs additional entitlement and additional conditions
- But actually no-sandbox process is sandboxed also
  - Platform sandbox
  - Most operations are allowed

Userland: enhanced sandbox

- Question: Is it a good practice for a modern mobile OS?
  - Of course not. Eg. launchd never needs to access user photos
  - Hackers can just target on no-sandbox processes
- Apple has good architecture to further limit no-sandbox process
  - Platform sandbox + entitlements
- Final goal: Every process should be "sandboxed ”
  - Privileged processes also has their own operation scope

Userland: enhanced sandbox

- Storage-class
  - Introduced around iOS 13.3
  - Important(user sensitive data) data are defined to specific storage class
    - E.g Storage-class "DCIM" for Media/DCIM folder
  - Only with specific entitlement, process can access specific storage-class
    - Some of them still always allowed but with sandbox report (e.g DCIM)
- With storage class, file operation capability is limited to most no-sandbox process
  - Reasonable

Userland: enhanced sandbox in iOS 14

- More processes are sandboxed
- More storage-class defined
  - HomeAI, Biome, etc.
  - DCIM class is removed though (might be compatibility issue)
- iokit-open capabilities are limited for no-sandbox process
  - Opening specific iokit userclient needs specific entitlement
  - Or the executable file must be in /usr/local/bin
- Sandbox can filter mach_msg calls based on msg_id now
- Userland sandbox bypass now can do very limited stuff:
  - As an important stage of full exploit chain, out-of-sandbox kernel bugs are picky now

Userland: different A key

- For A12+ devices, PAC is enabled
  - All userland processes share same A keys(IA DA keys)
- For userland sandbox bypass, attackers can calculate any A-key protected pointers of a different process
- Apple is aware of such attacks, so:
  - Use B-key to protect most of IPC related interfaces
  - But there are still exceptions, which can be used as universal techniques to exploit IPC memory corruption bugs (xpc, nsxpc, etc.)
    - _Block_release involves A key protected pointers (discovered by Ian Beer)

Userland: different A key

- In iOS 14, not all processes share same A key
  - Per entitlement com.apple.pac.shared_region_id
    - Process with same region id will use same A keys
  - Per team identifier
    - Self-developed Apps has different A key with other processes
- Privilege escalation via memory corruption ipc bugs is hard
  - Cannot calculate A key PAC for the target privileged process
  - WebContent has its own region id
  - Self-developed Apps’ s A key also different with others
- Most popular attack surfaces are killed

Kernel: address entropy

- Kslide
  - Before iOS 12.2, kslide is just 1 byte (256 possibilities), and only affect high bits of the lower 4 bytes of the address
  - After iOS 13, kslide became much more complex than before
    - E.g kslide: 19c2c000
- Memory address
  - On my other OS, kernel heap memory is hard to guess/spray without any info leak. (windows/linux)
  - Before iOS 14, kernel memory address entropy is low (kernel TEXT at 0xffffff0 xxxxxxxx, zone 0xffffffe0xxxxxxxx, or 0xffffffe1xxxxxxxx)
  - Usually by spraying around 300MB zone memory, we can obtain a fixed address with our controlled content
  - Heap infoleak not needed. Example: https://i.blackhat.com/us-18/Wed-August-8/us-18-Chen-KeenLab-iOS-Jailbreak-Internals.pdf

Kernel: address entropy

- In iOS 14, address is harder to guess
  - Addresses like 0xffffffe4 xxxxxxxx, 0xffffffe8 xxxxxxxx, 0xffffffe9 xxxxxxx exist
  - Although entropy still not high, it is a big improve.
  - For kernel exploitation, more infoleak is needed.

Kernel: enhanced PAC

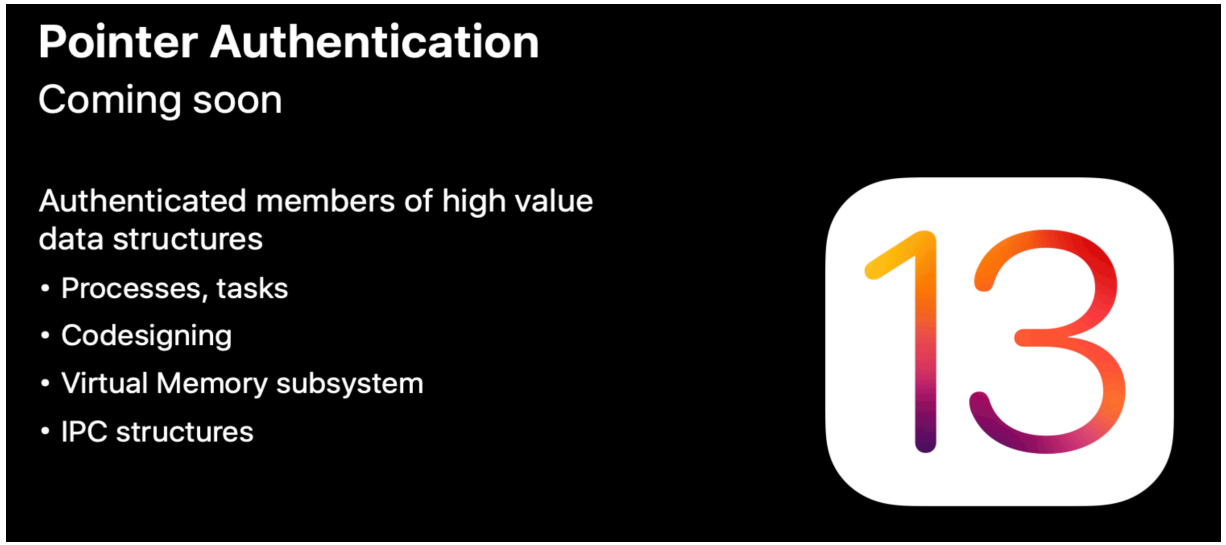
- Thread/context switch is naturally vulnerable to PAC
  - Need to switch register, including X30
  - iOS use G key to hash the key register info and save into context structure
  - And use G key to verify when restoring to register, making sure the context is not modified
- In the past, it is possible to confuse kernel/user thread and sign the kernel thread context using thread_set_state
  - I found this PAC bypass in 2018 (Brandon Azad also mentioned at BlackHat USA 2020)
- In iOS 14, different signing method is used for user thread and kernel thread
  - User thread: G key hash for PC CPSR LR X16 X17
  - Kernel thread: G key hash for 0 0x100004 X30 X16 X17

Kernel: enhanced PAC

- In iOS 13, some pointer uses 0-context
  - Etc. vptr, some callback functions
- iOS 14, many of those pointers are PACed with context
  - Hard to perform 0-context pointer replacement attack

Kernel: enhanced PAC

- Data pac in iOS 13? No.
  - Finally arrived in iOS 14
  - Critical pointers(port, tasks in critical structure) are data PACed.
    - With strong context
    - Cannot replace each other
  - E.g replacing proc->cred will panic immediately
  - E.g stealing a ipc_port from another process is impossible
- Impact what?
  - Kernel exploitation harder
    - Fake port techniques killed on A12+ devices
    - From arbitrary read/write to root harder



Kernel: zone_require enhancement

- Fake port exploitation methodology found by Ian Beer
  - First used in mach_portal
  - Idea is to change an ipc_port pointer to kalloc area which we can control, and make a fake task port
- zone_require
  - Ensure the port pointer is in ipc_port zone (also other critical pointers such as task, etc.)
  - In iOS 13(before 13.6), we can still use shared memory or kernel_map address to make fake port.
  - After 13.6, it is not allowed any more.

Kernel: Isolate kalloc zone

- In iOS 14, kalloc zone has 4 types: Default, Data, Kext, Temp
  - XNU struct in default.kalloc (usually metadata, e.g OSData)
  - Data.kalloc stores really data. (e.g OSData->data)
  - Kernel extension mainly uses kext.kalloc (IOMalloc, IOSurfaceRootUserClient, etc.)
  - Temp.kalloc is used to store some temp structure (will be freed soon)

- Kernel exploit is harder
  - Especially UAF (hard to control every bytes of a freed object)

Kernel: bug fixes

- Again, Apple fixes many good kernel bugs
- Some of the drivers are refactored
  - E.g IOGPUFamily, many userclients are abandoned
  - Killed good bugs ☺
- IOSurface 0 issue is also fixed(not a bug, but a logical issue, useful for exploitation)
  - Mentioned in my POC 2019 talk: http://powerofcommunity.net/poc2019/Liang.pdf
  - Looking up IOSurface 0 will always fail

Hardware related improvement for PAC

- In A12 and A13 era, there exists arbitrary signing gadget
  - AUTIA for arbitrary address and PACIA
  - Even for failed AUTIA, the PACIA will calculate the correct PAC of the address (Discovered by Brandon Azad)
- On A14 devices, PACIA stops signing the pointer if error bit is set
  - Hardware solution is better than software

```
uint64_t autia_res, pacia_res;
autia_res = AUTIA(0x140000000,0x5555);
pacia_res = PACIA(autia_res, 0x5555);

printf("PAC for autia res is %llx, PACIA for incorrect AUTIA is %llx.\n",
    autia_res,pacia_res);
```

```
PAC for autia res is 2000000140000000, PACIA for
    incorrect AUTIA is 540d408140000000.
```

Result on A13

```
PAC for autia res is 2000000140000000, PACIA for
    incorrect AUTIA is 140000000.
```

Result on A14

Summary

- In 2020, Apple improves iOS security a lot.
- All universal & public exploit techniques are killed.
- Attack surface hugely reduced:
  - Especially userland: pwning a userland privileged process no longer make you so privileged
- Kernel pwn is still powerful, but is much harder now

But, research still continues….

Demo

Thank you