An Apple a day keeps the exploiter away

SSTIC 2022

Who we are

- **Eloi Benoist-Vanderbeken**
  - @elvanderb
- **Fabien Perigaud**
  - @0xf4b

- **Reverse engineering technical leads**
  - 30+ reversers
  - Focus on low level dev, reverse, vulnerability research/exploitation
  - If there is software in it, we can own it :)
  - We are hiring!

Introduction

Pwning an iPhone in 2019

- **Exploit Safari**
  - Get arbitrary RW
  - Find a way to bypass APRR
    - Might need a PAC bypass to redirect code execution
  - → Execute arbitrary code in the sandbox
- **Get out of the sandbox**
  - Find a way to hook amfid, a userland daemon
    - Might need a kernel vulnerability or several userland ones
  - Use a valid certificate to sign your binary to bypass CoreTrust
- **Easy!**

WebKit Code Execution

Reminder: browser exploitation

- **Usually obtaining addrof / fakeobj primitives**
  - Allow crafting a fake object and getting objects addresses
- **Building arbitrary read / write primitives**
  - Ability to read and modify the whole process memory
- **Getting arbitrary code execution**
  - Depending on the hardware, this can be a hard task
  - APRR → hardware permissions switch on JIT page (RX<>RW)
  - PAC → data and instruction pointers signature

Structure ID randomization

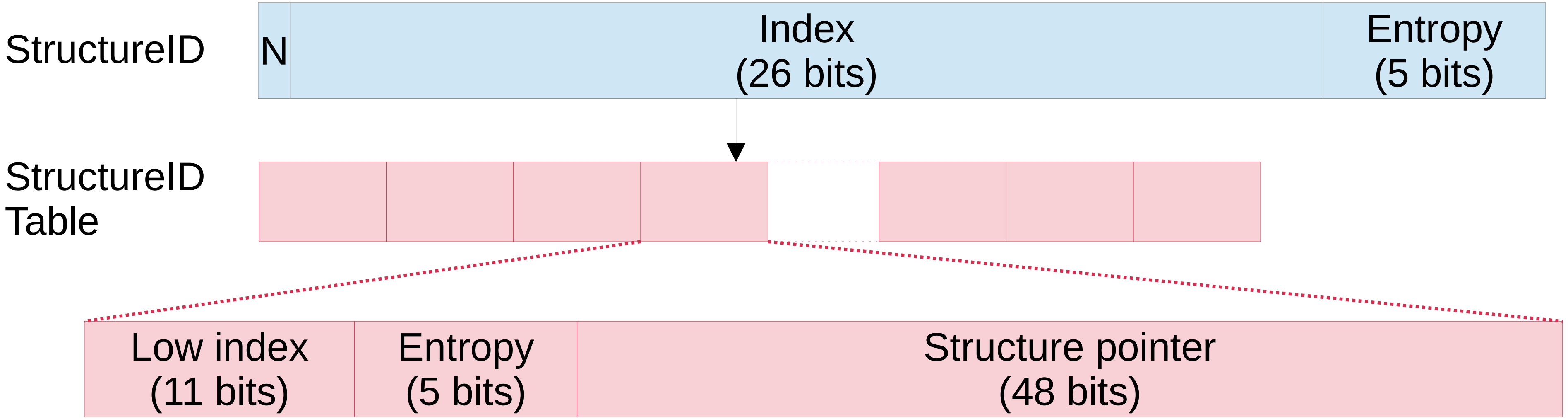
- **Building a fake object requires building a fake JSCell**

```
struct JSC::JSCell {
    JSC::StructureID                     m_structureID;
    JSC::IndexingType                    m_indexingTypeAndMisc;
    JSC::JSType                          m_type;
    JSC::TypeInfo::InlineTypeFlags       m_flags;
    JSC::CellState                       m_cellState;
};
```

- **Previously, a structure ID was an index into a table of structure pointers**
- **A structure describes an object shape**

Structure ID randomization (2)

- **From iOS , the structure ID includes some random bits**



- **Entropy and index bits must match**

Structure ID randomization (3)

- **Mitigation being removed ¯_(ツ)_/¯**
- **Several bypasses have been published**
  - Main idea: only use methods that do not involve object structure manipulation
  - Might be enough to build a R/W primitive

JIT instructions signature

- **Known APRR bypasses**
  - Redirect code execution to the function write code in the JIT page
  - Race the thread writing to the JIT page by modifying its instructions buffer (no code execution required)
- **Now, the JIT code is signed using PAC instructions (on devices supporting PAC)**
  - Each instruction has an associated signature
  - Signature is checked when writing each instruction to the JIT page
- **Attackers need a primitive to sign arbitrary data**
  - No more data only attack to bypass APRR

JIT instructions signature (2)

- **No known bypass**
  - Mitigation relies on PAC, a PAC bypass would bypass it

```
uint32_t nextValue(uint64_t instruction, uint64_t index, uint32_t currentValue)
{
    uint64_t a = tagInt(instruction, makeDiversifier(0x12, index, currentValue));
    uint64_t b = tagInt(instruction, makeDiversifier(0x13, index, currentValue));
    return (a >> 39) ^ (b >> 23);
}
```

tagInt is **PACDB** (sign with data key B)

PAC improvements

- **More signed pointers**
  - Signed data pointers
  - No more unsigned .got pointers
- **More diversity**
  - Pointers signed with a null context are increasingly rare
  - Pointers usually signed on the fly
  - → Pointers substitution attacks are harder
- **Dedicated keys**
  - “com.apple.pac.shared_region_id” entitlement
  - Complicated to attack other processes

PAC improvements (2)

- **Brute-force prevention**
  - Wider signature for instruction pointers: 24 bits
  - AUT instructions always followed by a check (-fptrauth-auth-traps)
  - A14 has EnhancedPAC: no more “flipped” signature on invalid pointer signature → null signature
  - A15 has Armv8.6-A FPAC: exception when an AUT instruction detects an invalid signature
- **Exception termination**
  - Entitlement “com.apple.private.pac.exception”
  - Adds “TF_PAC_EXC_FATAL” flag to the task
  - Task termination on PAC-related exception

Privilege Escalation

Privilege escalation

- **Goal**
  - To execute arbitrary code
  - With arbitrary entitlements
- **Attack surface**
  - User daemons
  - Kernel extensions (KEXTs)
  - Kernel

- **2019 Protections**
  - Sandbox
    - More and more
  - PAC / PPL / RoRgn
  - Code signature
    - Kernel and user

Sandbox

2019

- **120 userland services**
- **15 IOKit User Client Classes**
- **Arbitrary syscalls**
  - With restricted functionalities
- **Arbitrary ioctl/fnctl**

2022

- **4 userland services**
  - Some msgs are restricted
- **0 IOKit User Client Classes**
- **Filtered syscalls**
  - ~ 100/500 syscalls
  - ~ 30/500 kernel mig
- **~20 fnctl**
- **2 ioctls (on /dev/aes_0)**

User

- **WebKit now uses a specific PAC A key**
  - Impossible to sign pointers for another process
- **Objective-C ISA pointers are now signed**
  - Kills a lot of exploit techniques
- **Processes cannot directly get their task port**
  - Cannot easily force a process to send its task port
  - Cannot easily manipulate a foreign process
- **Port labels are used to block ports**
  - Platform Binary task/thread ports cannot be sent to non-PB tasks

Kernel

- **Only two pointers signed with a zero context in memory**
  - mig_strncpy_zerofill and __chkstk_darwin
  - Others are signed on the fly
- **Specific context for function pointers in structures**
- **More and more data pointers are PACed**
  - Breaks exploit and post-exploit primitives
- **Stack variables are always initialized with 0xAA**
  - No more stack leaks

Kernel Heap

- **Zone require**
  - Sensitive objects must come from a specific zone
- **Zone sequestration**
  - Impossible to reuse an object with another zone
  - More specific zones
  - Kills a lot of vulnerabilities
- **Data/pointers Segregation**
  - KHEAP_DATA_BUFFERS / KHEAP_DEFAULT / KHEAP_KEXT
  - Structure split (ipc_kmsg and others)
- **Better randomization**

Hardening

- **Hooking amfid is not enough anymore**
  - Signature was already checked by the kernel with CoreTrust
  - Now entitlements and provisioning profiles are also checked
- **Injecting code in other processes is now more complex**
  - Task ports are now available with various flavors
    - TASK_FLAVOR_CONTROL / READ / INSPECT / NAME
  - No process has the entitlements needed to get tasks control port
  - PPL blocks any non-PB code page in PB process
- **Arbitrary entitlements are needed to access sensitive data**

Hardening

- **PPL is now used to**
  - Validate and protect entitlements in the kernel
  - Provide RO zones
- **RO zones**
  - Can only be written with a special PPL function
  - It checks the type, size, source and destination of the copy
  - Used to protect task credentials, threads exception ports, sandbox profiles, entitlements, etc.
- **No easy way to become root / steal entitlements**

Conclusion

Pwning an iPhone in 2022

- **Exploit Safari**
  - Get arbitrary RW
    - A bit more complicated, more and more pointers are signed
  - Find a way to bypass APRR
    - Need to be able to sign arbitrary data with PAC
    - Or to find a bypass
  - → Execute arbitrary code in the sandbox
- **Exploit the kernel**
  - Fight against the ultra-tight sandbox and the new mitigations
  - Get arbitrary kernel R/W

Pwning an iPhone in 2022

- **Bypass kernel protections**
  - Get root / bypass the sandbox
    - Might require a read-only zone bypass
  - Bypass signature verification in the kernel / PPL
    - Might require a PPL bypass…
    - …which will most probably require a PAC bypass
- **Enjoy your root shell**
  - Cannot inject arbitrary code in arbitrary processes
  - Doesn’t survive a reboot

Conclusion

- **Harder and harder to attack iPhones**
- **Real, constant effort from Apple on all stages**
  - Attack surface reduction
  - Effective mitigations
    - Make whole class of bugs unexploitables
    - Kill generic methods
  - Strong post-exploit mitigations
    - Even with arbitrary kernel R/W it is non trivial to get sensitive data
- **A LOT of time and effort is put in securing iPhones**
  - And we didn't even talk about data at rest, persistance, etc.

Conclusion – bis

“La lecture des deux articles bout à bout (je l'ai fait) risque en effet d'avoir un effet pervers : on en ressort avec le sentiment que beaucoup de choses sont désormais très bien protégées et que l'exploitation d'une vulnérabilité semble impossible ou extrêmement difficile.”

Conclusion – ter

https://www.linkedin.com/company/synacktiv
https://twitter.com/synacktiv
Nos publications sur : https://synacktiv.com