

The logo for SYNACKTIV features a stylized icon on the left consisting of a 3x3 grid of squares, with the top-left square missing and the top-right square containing a red dot. To the right of this icon, the word "SYNACKTIV" is written in a bold, sans-serif font. "SYNA" is in white, "CKTIV" is in red, and the "A" is a slightly larger, bolded character.

SYNACKTIV



Attacking Safari in 2022

/HEXACON/

Who am I?

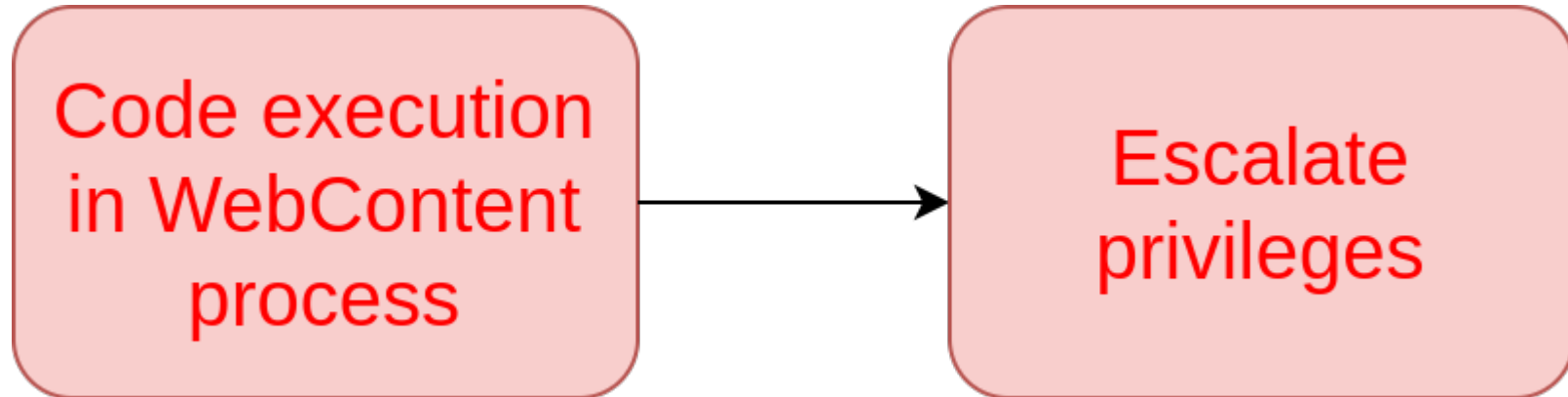


- **Quentin Meffre (@0xdagger)**
 - Security researcher at Synacktiv
 - Vulnerability research & Exploitation
- **Synacktiv**
 - Offensive security company
 - +100 ninjas
 - We are hiring!

Introduction



- Full chain on iPhone using the browser as entry point

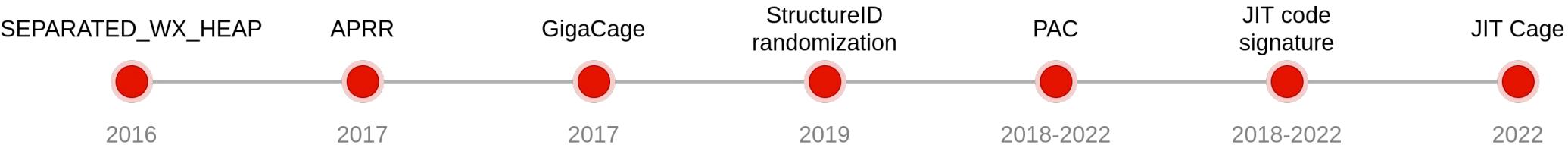


Introduction



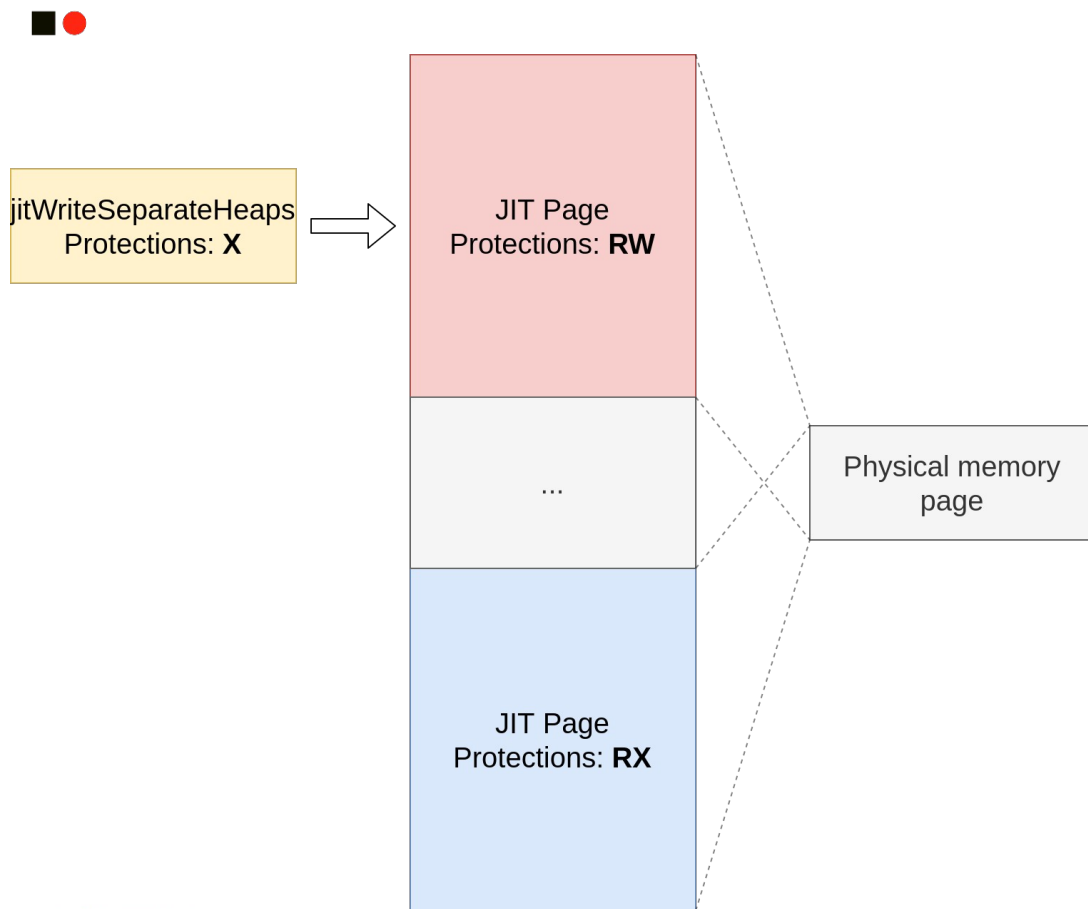
- **Steps to compromise Safari on the iPhone**
 - addrOf/fakeObj
 - Arbitrary R/W
 - Bypass PAC/APRR
 - Overwrite JIT page code
 - Arbitrary code execution!
- **Apple hardened each step of a Safari exploit...**

History of Safari mitigations



SEPARATED_WX_HEAP

- **The JIT page is mapped twice**
 - One has protections RX
 - Second has protections RW
- **A function is jitted to copy data in the JIT page**
 - The function is on a page with **X only** protection
 - The address of the **RW** JIT page is inlined in this function



SEPARATED_WX_HEAP



- **Public bypass still works with this mitigation¹**
 - Build an arbitrary call primitive
 - ROP/JOP
 - Call the `jitWriteSeparateHeaps` function
 - Write arbitrary code in the JIT page
 - Profit!

1: https://www.sstic.org/media/SSTIC2019/SSTIC-actes/WEN_ETA_JB/SSTIC2019-Article-WEN_ETA_JB-benoist-vanderbeken_perigaud.pdf

APRR



- Hardware mitigation
- SEPARATED_WX_HEAP is replaced by APRR on supported hardware
- Atomically switches the JIT page protections using a System Register
 - RX → RW → RX

```
MOVK      X0, #0xC110
MOVK      X0, #0xFFFF, LSL#16
MOVK      X0, #0xF, LSL#32
MOVK      X0, #0, LSL#48
LDR       X0, [X0]
MSR       #6, c15, c1, #5, X0
ISB
```

APRR

9

■ Hard jump in the middle of the function¹

- The System Register value comes from a **R only** page shared with the kernel
- The system register value and the value from the **R only** page are compared
 - Difference → crash

■ Without CFI can be bypassed like **SEPARATED_WX_HEAP**

```
MOVK      X0, #0xC110
MOVK      X0, #0xFFFF, LSL#16
MOVK      X0, #0xF, LSL#32
MOVK      X0, #0, LSL#48
LDR       X0, [X0]
MSR       #6, c15, c1, #5, X0
ISB

MOVK      X1, #0xC110
MOVK      X1, #0xFFFF, LSL#16
MOVK      X1, #0xF, LSL#32
MOVK      X1, #0, LSL#48
LDR       X8, [X1]
MRS      X10, #6, c15, c1, #5

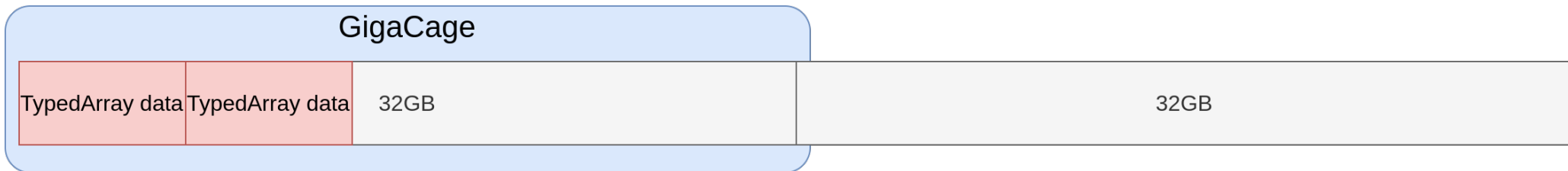
CMP       X8, X10
B.NE     loc_18BA4E060
```

1: https://github.com/phoenix/files/blob/master/exploits/ios-11.3.1/pwn_i8.js

GigaCage



- **TypedArray are JavaScript objects**
 - Often used to build arbitrary R/W
- **TypedArray are allocated in a 32GB zone**
 - Followed by another 32G zone allocated with PROT_NONE
- **The data buffer is now an offset to the cage and no more an address**



- **Cannot R/W outside of the cage anymore...**

GigaCage bypass



- **Many public documentation about the GigaCage¹**
 - Some public bypasses still work...
- **One known bypass is to use other objects**
 - More on this later in this presentation
- **GigaCage is not enabled anymore on latest iOS versions**
 - But attackers still can't use TypedArray to build arbitrary R/W...

1: <https://googleprojectzero.blogspot.com/2020/09/jitsploitation-two.html>

<https://t.me/learningnets>

StructureID randomization



■ **JSObject inherits from the JSCell object**

m_cellState 1 byte	m_flags 1 byte	m_type 1 byte	m_indexingType 1 byte	m_structureID 4 bytes
-----------------------	-------------------	------------------	--------------------------	--------------------------

■ **The StructureID is an index**

- Used to get the *Structure* of a JSObject

■ **Invalid StructureID → crash**

■ **Before randomization the StructureID was incremental**

- Easy to guess a valid *StructureID*
- Build fake objects without crashing

After StructureID randomization

13



- **Randomization is added to the StructureID**

| 1 Nuke Bit | 26 StructureIDTable index bits | 5 entropy bits |

- **Signature is checked every time a JSObject property is accessed...**

- ... but sometimes it is not!¹
- Leads to StructureID randomization bypass

- **StructureID randomization has been removed**

- StructureID uses low 32 bits of Structure address

1: <https://i.blackhat.com/eu-19/Thursday/eu-19-Wang-Thinking-Outside-The-JIT-Compiler-Understanding-And-Bypassing-StructureID-Randomization-With-Generic-And-Old-School-Methods.pdf>
<https://t.me/learningnets>

PAC



- **Pointer Authentication Code**
- **Hardware mitigation**
- **Introduced in ARMv8.3-A**
- **Prevents an attacker from corrupting sensitive pointers**
 - Signature is added to some pointers
 - Corrupting a pointer without signing it correctly often leads to a crash



■ New ARM instructions used in Safari

- PAC*: Add signature to a pointer
- AUT*: Check and remove signature from a pointer
- XPAC*: Remove signature from a pointer
- RETA*: Check X30 with context SP and return to X30 if the signature is correct
- BRA* / BLRA*: Check signature and branch

PAC



- **Two kinds of pointers can be signed**
 - Data
 - Instruction
- **Two keys can be used for each kind**
 - Key A
 - Key B
- **A context is often used to avoid pointer substitution**
 - A pointer can also be signed with a null context...

PAC



- **The signature is stored in the top bits of a pointer**
- **The signature length depends on the key/pointer kind**
 - 16 bits
 - 24 bits



PAC



■ Instruction pointers

- VTable function pointer => PACIA
- Return value stored on the stack => PACIB
- JIT Code pointer => PACIB

■ Data pointers

- VTable pointer => PACDA
- Sensitive data pointer (TypedArray data pointer...) => PACDB
- JIT instructions => PACDB

PAC



- **What is not signed in Safari?**



PAC bypass



- **Bypassing PAC is a security issue in itself**
 - Apple takes PAC bypasses very seriously
- **Many PAC bypasses have been disclosed since PAC introduction**
 - Apple fixes each of them
 - Hardware improvement
 - Software improvement

PAC bypass: design issue



- **If a pointer authentication fails**
 - Signature is removed and one of the top bits is flipped
 - Does not raise an exception
- **If the pointer is signed again after the failed AUT***
 - Correct signature is added, with a flipped bit
 - PAC bypass: flip the bit again to get the correct signature
- **EnhancedPAC is implemented first on A14 SoC**
 - Signing invalid pointers will discard the signature
 - Can't leak the signature anymore...

PAC bypass: bruteforce



- The signature can still be bruteforced...
- ...but Apple killed this bypass again
- The compiler option `-fptrauth-auth-traps` is used
 - Adds a check after all `AUT*` instructions
 - If the signature given to the `AUT*` instruction is invalid → ABORT

```
AUTIB          X16, X17
MOV            X17, X16
XPACI         X17
CMP            X16, X17
B.EQ          loc_18BA4ABD8
BRK            #0xC471
```

PAC bypass: bruteforce



- **Apple added a new feature in the A15 SoC**
 - ARMv8.6-A FPAC extension
 - If an AUT* instruction fails, an exception is now raised
- **Apple killed this exploitation method with this feature**

PAC bypass: null context chained

24



- **Initially, many pointers were signed with a null context**
- **A potential bypass could be to use null signed pointers in a JOP chain**
 - Build powerful primitives
- **Never seen publicly**
- **Since iOS 15 this attack has been almost killed**
 - Very few pointers are still signed with a null context

PAC bypass



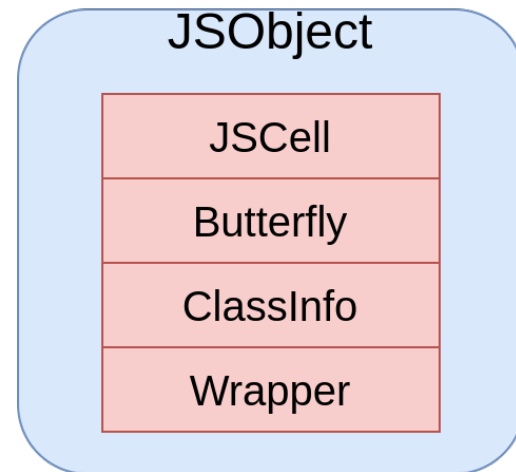
- **More bypasses¹**
 - Unprotected code pointers
 - Race condition with the JIT thread
 - Blocking the JIT thread while copying data on the JIT page
 - Signal handlers corruption
- **All of these bypasses have been fixed**

1: <https://googleprojectzero.blogspot.com/2020/09/jitsploitation-three.html>

PAC R/W



- **PAC doesn't sign a lot of sensitive data pointers**
- **Some object can be wrapped into a JSObject**
 - DOMRect
 - Contains 4 doubles
 - Has methods to read and write these doubles
- **Faking a wrapper to a DOMRect object**
 - Arbitrary R/W
- **Method used by a public exploit¹**



1: <https://blog.google/threat-analysis-group/analyzing-watering-hole-campaign-using-macos-exploits/>
<https://t.me/learningnets>

PAC kill R/W method



- **Method killed by iOS 15.4**
- **Some wrappers to sensitive wrapped objects are now signed**
 - Most of them manipulate floats/doubles
 - Killed many arbitrary R/W methods

```
commit 30199a0aff4a6eff01d9a90aa1c05aac87f1a4cd
Author: [redacted] <[redacted]@apple.com>
Date: Thu Feb 10 14:46:18 2022 +0000

Introduce SignedPtrTraits which enables Ref pointers to be protected with PtrTags.
```

JIT Code signature



- **The JIT compilation can be done in another thread**

- The assembly code is stored in a temporary buffer while doing compilation
- The temporary buffer content is copied in the JIT page at the end of the compilation

- **Before JIT code signature**

- Race the JIT thread to put arbitrary code in the temporary buffer
- Profit!
- But...

JIT Code signature

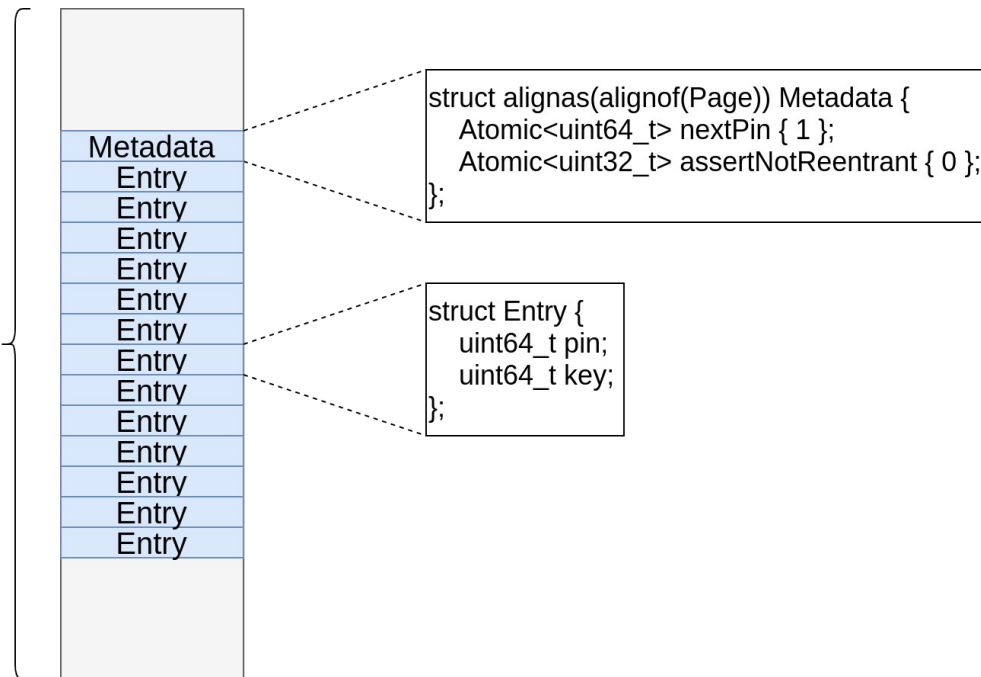


- **Apple introduced the JIT code signature**
 - Stop attackers from overwriting the *JIT* code buffer
- **Software mitigation based on PAC**
- **Instructions stored in the temporary buffer are signed**
 - Each instruction signature generates a hash stored in the hash buffer
 - Signed with previous hash and PACDB
- **Signature is checked when the temporary buffer is copied in the JIT page**
 - If the signature is invalid → Crash

JIT Code signature PIN



- The hash used to sign the next instruction was not protected
- It is now signed with a unique identifier (PIN)
 - Each JIT compilation uses a different PIN
 - PIN informations are stored in the JIT page
 - An attacker can't modify them



JITCage



- **The A15 SoC brings a new complex mitigation**
 - The JITCage!
- **The JITCage stops attackers from calling arbitrary functions from the JIT page**
- **The JIT page is now mapped with a new flag**
 - MAP_JITCAGE?
- **The XNU open-source project doesn't have references about this flag...**



■ ...but the KernelCache has references!

```
void __fastcall enable_jitbox(__int64 thread)
{
    __int64 current_thread; // x0
    __int64 v3; // x8
    unsigned __int64 StatusReg; // x9

    current_thread = get_thread_ro();
    if ( current_thread == thread )
    {
        v3 = *(_QWORD *) (current_thread + 0x358);
        StatusReg = _ReadStatusReg(ARM64_SYSREG(3, 0, 13, 0, 4));
        *(_QWORD *) (*(_QWORD *) (StatusReg + 0x158) + 0x218LL) = v3;
        *(_QWORD *) (*(_QWORD *) (StatusReg + 0x158) + 0x210LL) = *(_QWORD *) (current_thread + 0x350);
        _WriteStatusReg(ARM64_SYSREG(3, 4, 15, 15, 4), *(_QWORD *) (current_thread + 0x358));
        _WriteStatusReg(ARM64_SYSREG(3, 4, 15, 15, 1), *(_QWORD *) (current_thread + 0x350));
        __isb(0xFu);
    }
}
```

JITCage



- **The kernel sets new System Registers using**
 - The size of the JIT page
 - The address of the JIT page
 - Some unknown flags
- **The KernelCache has no other information**
- **The interesting part of the JITCage is implemented in the A15 SoC**

JITCage



- **The following instructions can't be executed in the JITCage**
 - RET
 - BR/BLR/BL
 - SVC
 - MRS/MSR
- **If one tries to execute these instructions in the JITCage**
 - The processor raises an `EXC_BAD_INSTRUCTION` exception

JITCage



- **The PAC IA/IB keys are different in the JITCage**
- **Can't sign instruction pointers in the JITCage**
 - PACIA doesn't add signature if executed in the JITCage
 - PACIB can only sign pointer that points into the JITCage
 - PACD* seems unaffected by the JITCage

JITCage



- **The JIT code has to call functions outside of the JITCage**
- **Setting a System Register allows changing IA key**
 - Instruction pointers used by the JITCage are signed with the IA key
- **Only done once when the JavaScript engine is initialized**
- **Can't be done anymore after**

```
MRS      X8, #4, c15, c15, #6
ORR      X8, X8, #0x8000
MSR      #4, c15, c15, #6, X8
```

- **An attacker can't easily call functions outside of the JITCage**

Conclusion 1/2



- **Getting arbitrary code on latest iPhone involves finding:**
 - A vulnerability
 - A new method to build arbitrary R/W
 - A PAC bypass
 - An APRR bypass
 - A JITCage bypass
- **One solution for attackers could be to implement the next stage using JavaScript only...**

Conclusion 2/2



- **2022 in short**
 - Yet another mitigation
 - Yet other exploitation methods killed
- **What to expect in the next years?**
 - Same as above?
- **Maybe it's time for attackers to find another entry point than the browser...**
 - ...or maybe not? :-)
 - JavaScript is a powerful engine to attack all those mitigations



<https://www.linkedin.com/company/synacktiv>

<https://twitter.com/synacktiv>

Our publications: <https://synacktiv.com>