SYNACKTIV

Escaping the Safari Sandbox: A tour of WebKit IPC

OFFENSIVECON

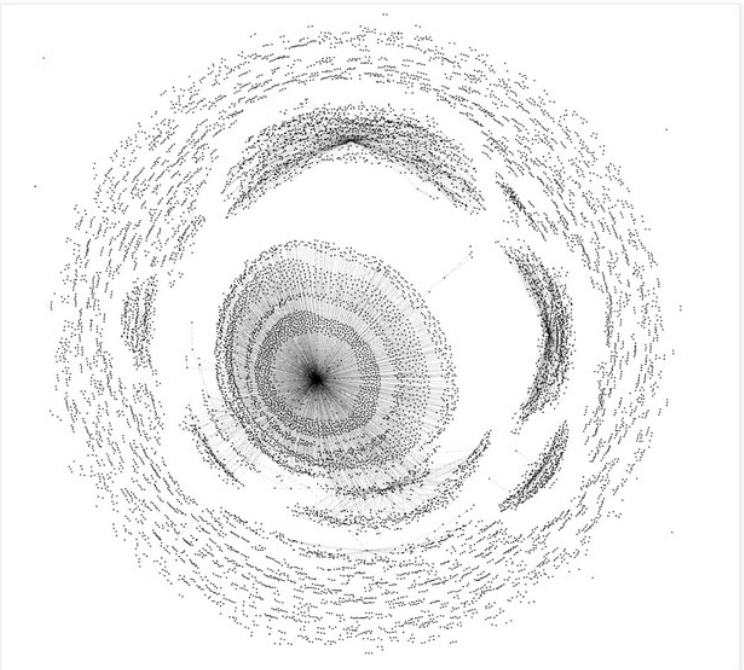
Who am I?

- **Quentin Meffre (@0xdagger)**
  - Security researcher at Synacktiv
  - Vulnerability research & Exploitation
- **Synacktiv**
  - Offensive security company
  - +170 ninjas
  - We are hiring!

Introduction

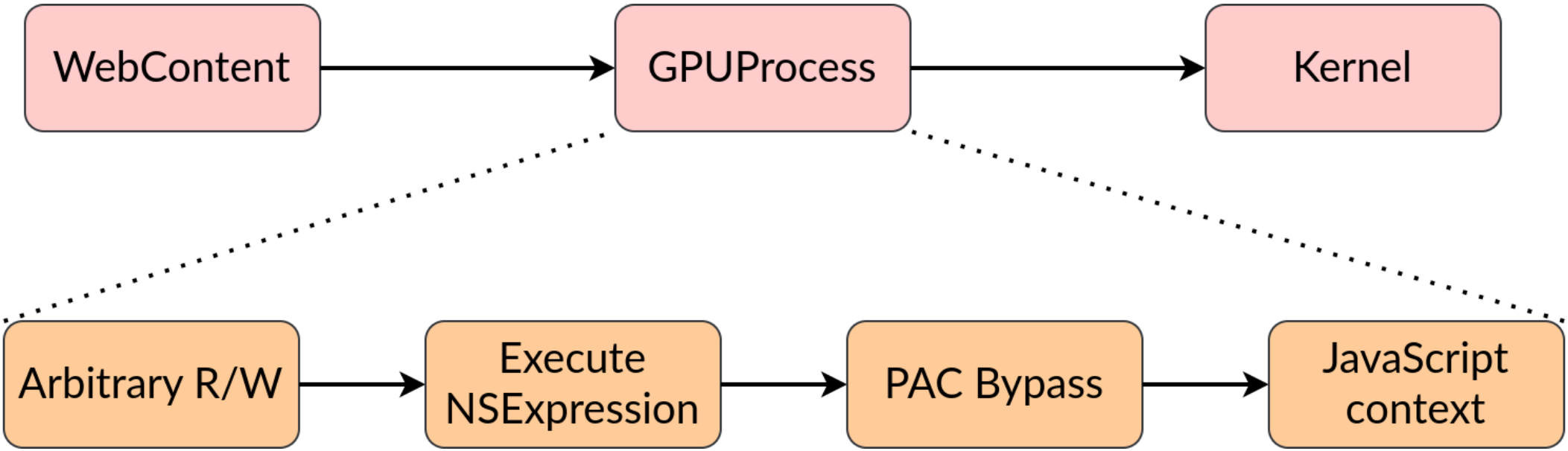
An analysis of an in-the-wild iOS Safari WebContent to GPU Process exploit

By Ian Beer



SYNACKTIV

Introduction

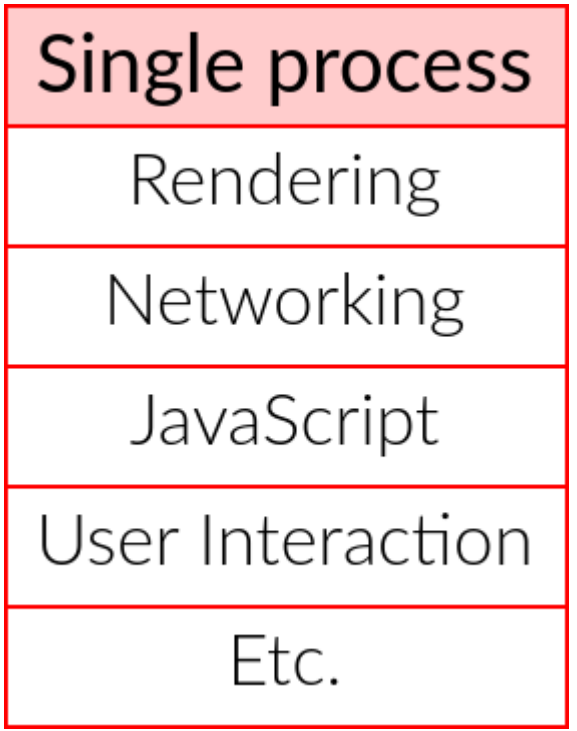


WebKit

- **Browser engine**
  - Developed by Apple
  - Mainly used by Safari
  - Initial release in 2005
- **Ships everything to build a browser**
  - JavaScript engine
  - DOM/rendering engine
  - Web APIs
  - User Interface API
  - Etc.

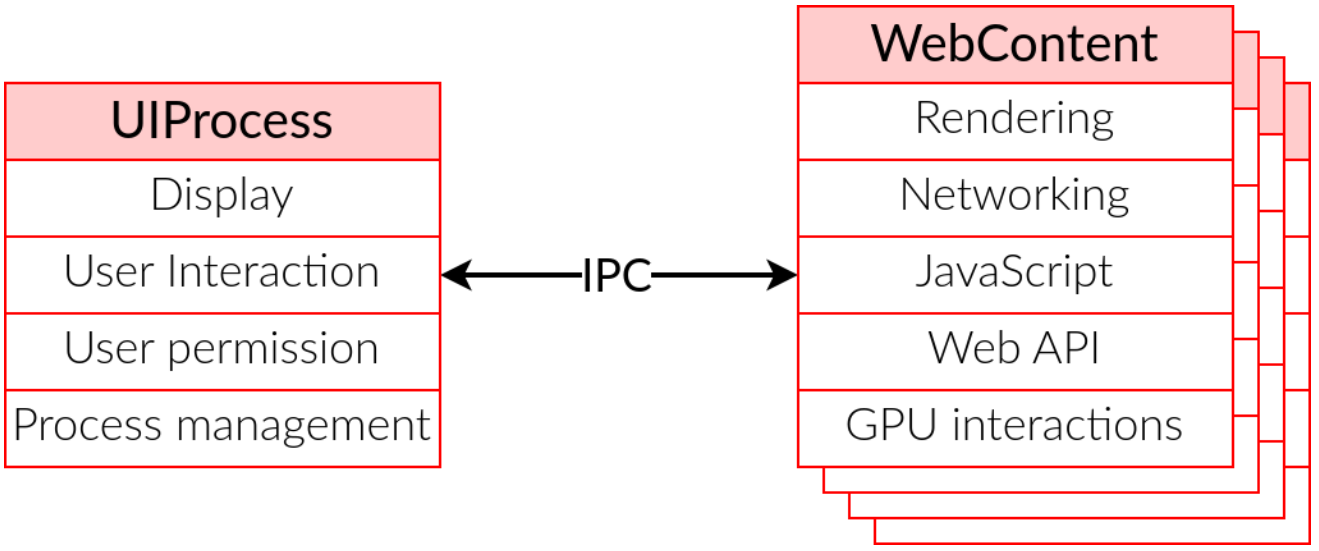
Architecture

- **Initial architecture**
  - Single process
  - Too much privileges
- **Bad from a security point of view**
  - Compromise the process → Game Over

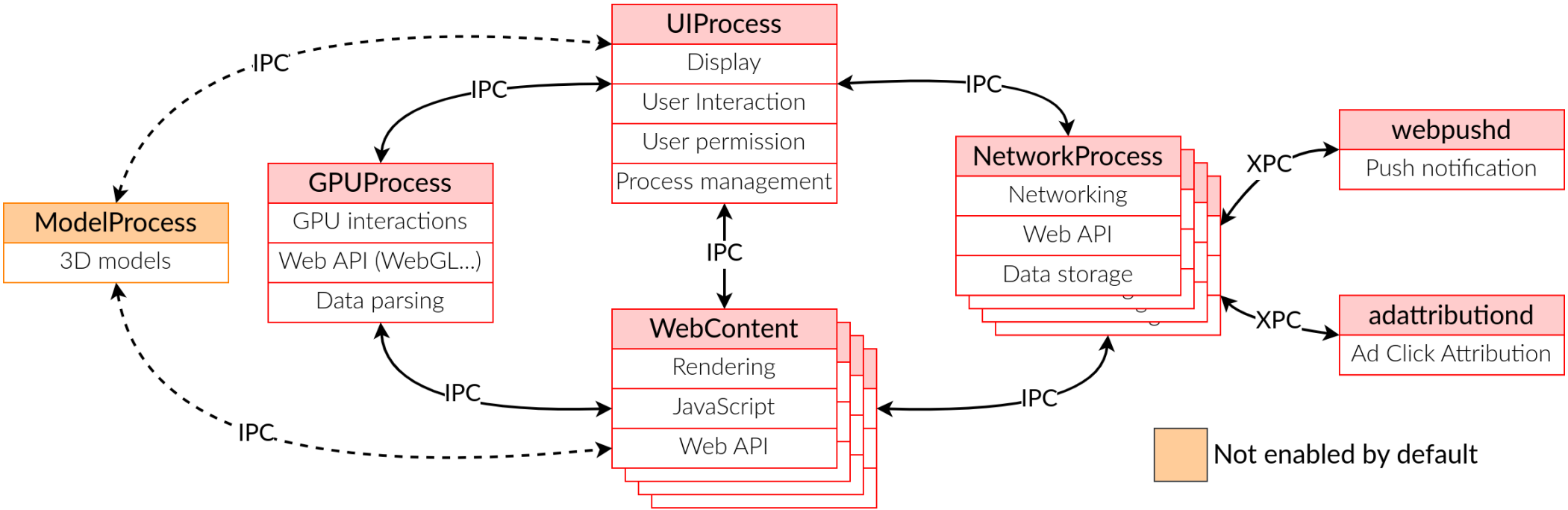


Architecture

- **WebKit2!**
- **Multi-process**
  - UIProcess
    - *Most privileged*
  - WebContent
    - *Less privileged…*
    - *…still too much*



Architecture



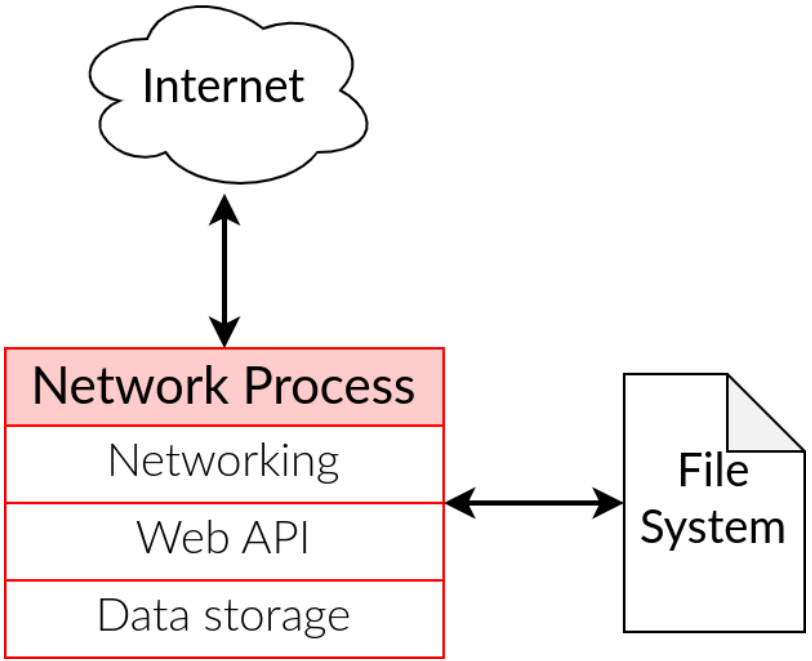
WebContent

- **Most exposed process**
  - DOM rendering
  - JavaScript engine
  - Web APIs implementation
- **Almost no privileges**
  - Hardened sandbox profile
- **Can use sandbox extensions…**
  - …require user permission
- **WebKit processes expose a large surface**

IOKit	0
Unix syscalls	~90
Mach traps	~30
MIG routines	~20
Userland services	0
WebKit process	3

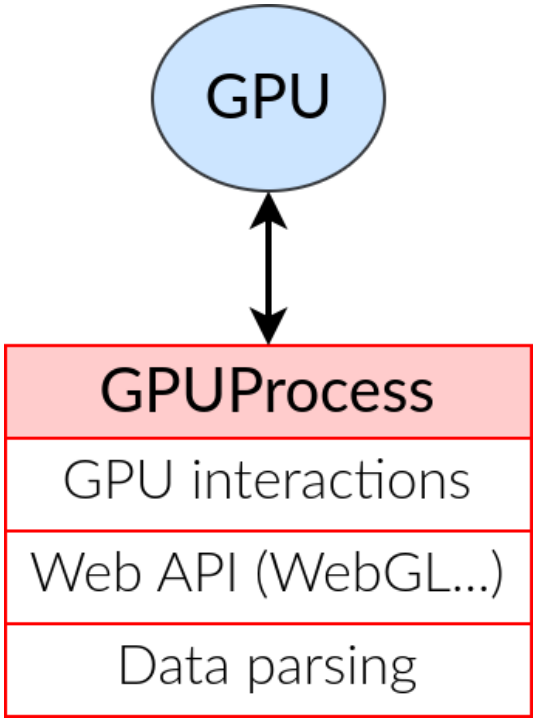
NetworkProcess

- **Network-related process**
  - Loading remote/local resources
  - Web APIs implementation
    - *CacheAPI*
    - *SharedWorker*
    - *etc.*
- **Larger kernel and userland surface**
  - Few network-related syscalls
  - Access to 1 IOKit
  - Communicates with some services



GPUProcess

- **Video and graphics processing**
  - Communicates with GPU (via ANGLE)
  - Web APIs implementation
    - *WebGL*
    - *WebGPU!*
      - Not reachable anymore
    - Etc.
  - Data parsing
    - Font, WebRTC
- **Almost same sandbox as WebContent**
  - Few IOKits and services

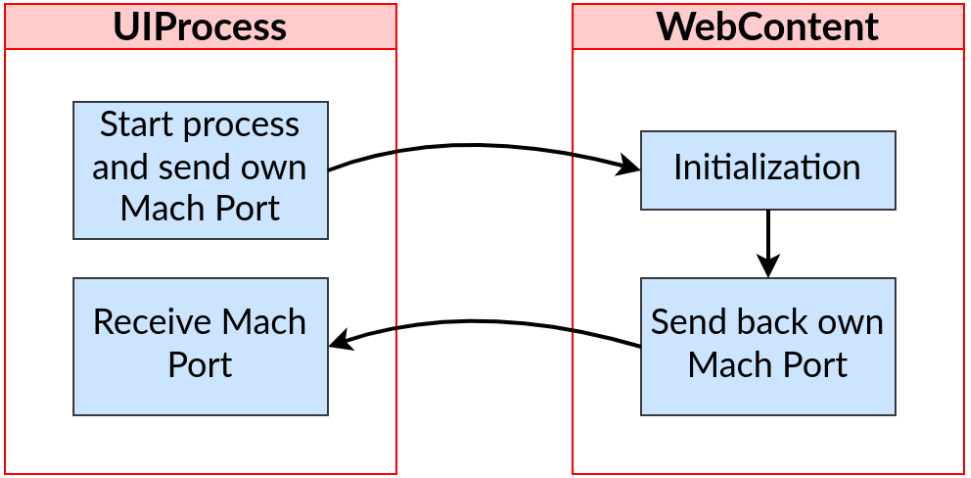


UIProcess

- **Main Safari process**
- **Most privileged WebKit process**
  - Display on screen
  - User interaction
  - Process management
  - User permissions management
    - *Camera*
    - *Microphone*
    - *etc.*
- **No specific sandbox**

WebKit2

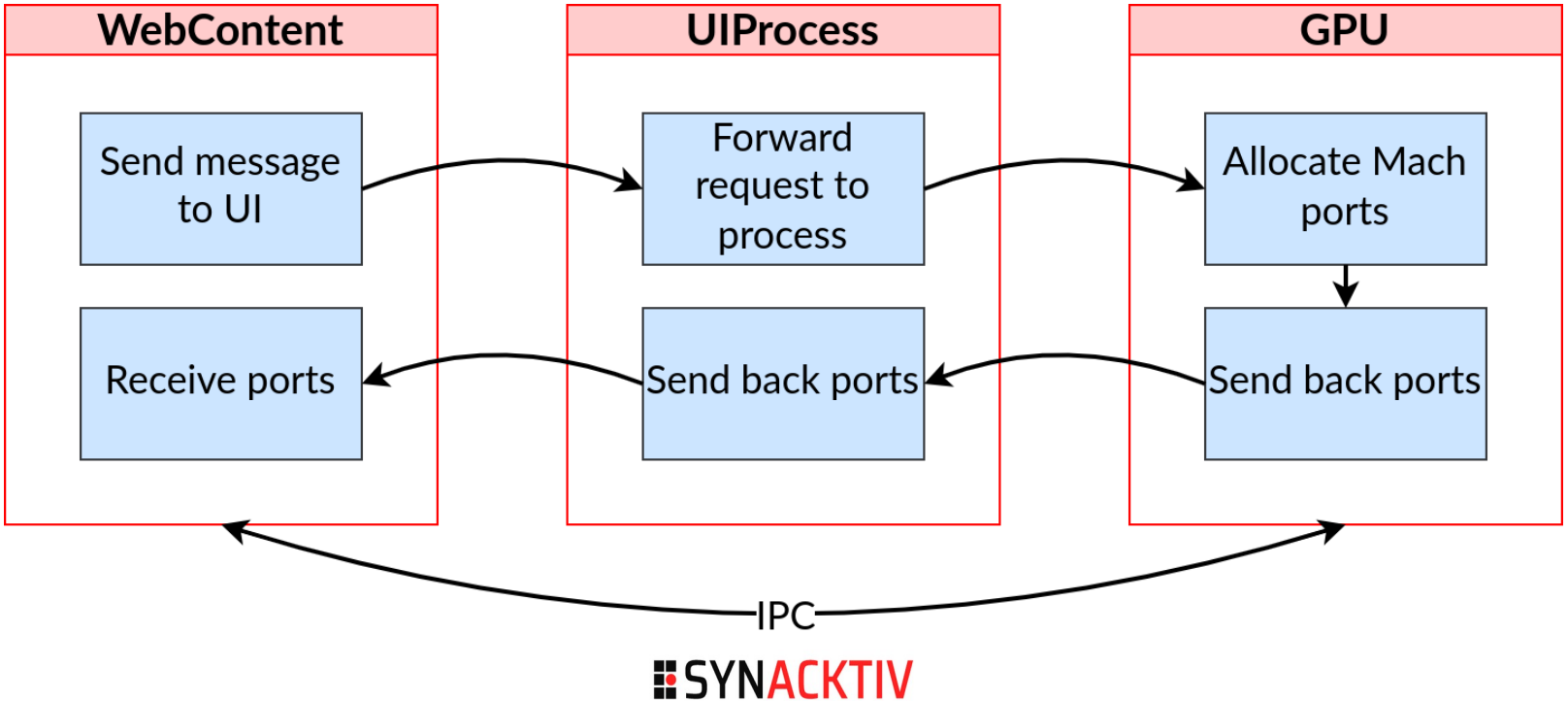
- **WebKit processes are extensions[1]**
  - Services before iOS 17.4
- **Communicate through Mach messages**
  - UIProcess starts every WebKit process



1: https://developer.apple.com/documentation/extensionkit?language=objc

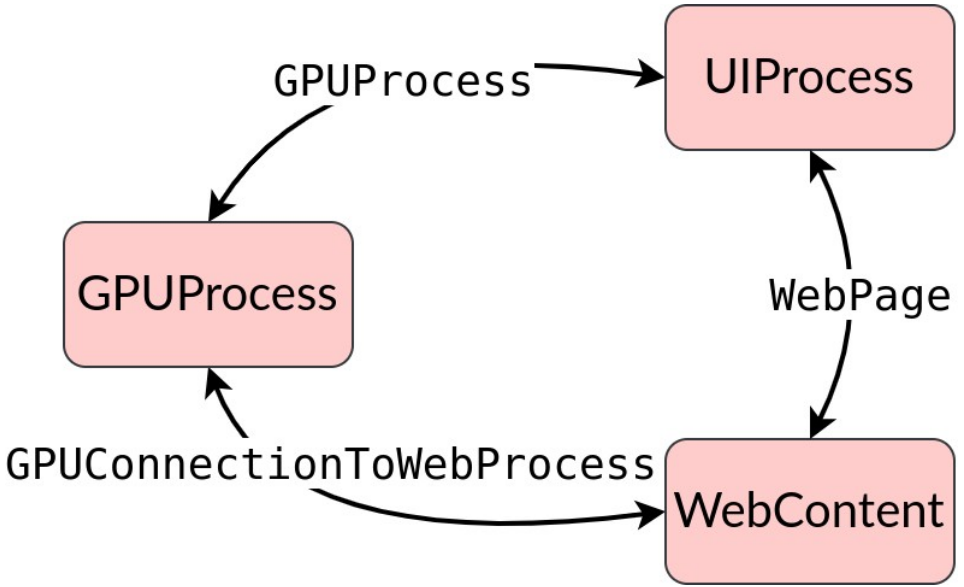
WebKit2

- **UIProcess allows WebContent to communicate with other processes**



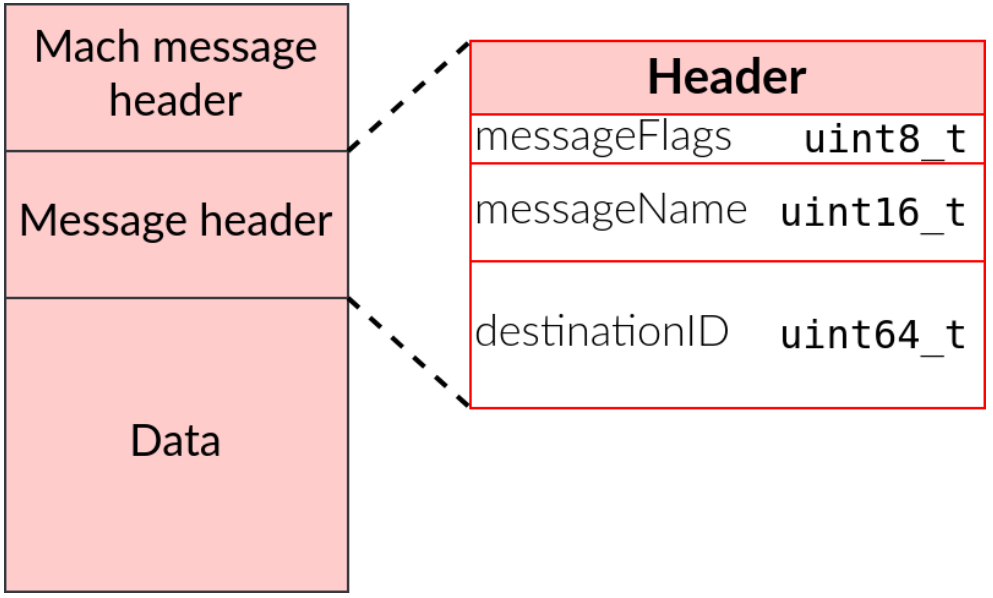
WebKit2

- **Processes have their own dedicated connection**
  - Messages are filtered based on connection type



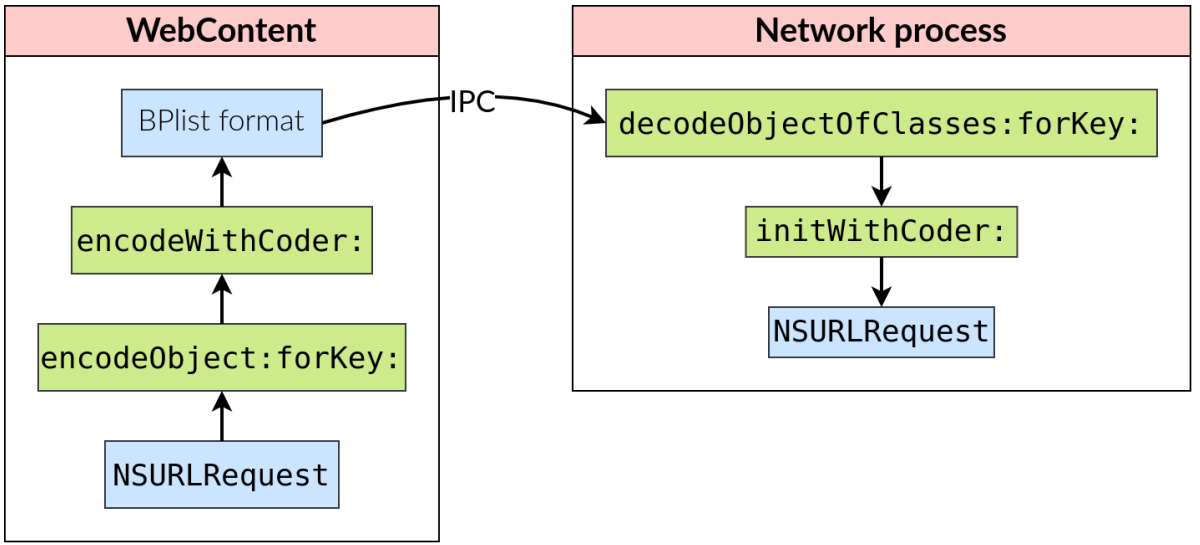
WebKit2

- **Message starts with a `mach_msg_header_t`**
- **Followed by a message header**
- **Custom encoder/decoder**
  - Integer, string, floating number
  - Memory entry, Objective-C object



Objective-C decoding

- **WebKit2 can send Objective-C objects**
  - Based on `NSKeyedArchiver` *and* `NSKeyedUnarchiver`
    - Objects are serialized as BPlist



Objective-C decoding

- **Very powerful**
  - Lots of objects can be encoded/decoded
  - Supports cyclic decoding
- **Historically lots of exploits abused the Objective-C deserializer[1][2][3]**
- **Apple starts killing exploitation methods...**

1: https://googleprojectzero.blogspot.com/2020/01/remote-iphone-exploitation-part-1.html
2: https://googleprojectzero.blogspot.com/2022/03/forcedentry-sandbox-escape.html
3: https://googleprojectzero.blogspot.com/2023/10/an-analysis-of-an-in-the-wild-ios-safari-sandbox-escape.html

Objective-C decoding

- **`NSSecureCoding`**
  - Must specify decoded type
  - Raise exception if decoded object type != specified type
  - Allows to decode subclasses of the specified type!
    - If `NSObject` is in the allowed list → arbitrary deserialization!

```
- (id)decodeObjectForKey:(NSString*);
```

With Secure Coding

```
- (id)decodeObjectOfClass:(Class) ForKey:(NSString*);
```

Objective-C decoding

- **Trust restrictions**
  - Applied to Platform Binary and Apple applications
    - Raises an exception if `NSObject` is in the allowed list
    - Collection classes must explicitly be in the allowed list
      - `NSArray`
      - `NSSet`
      - etc.
    - Disable many features of `NSPredicate`
    - Disable cyclic decoding
    - Decoding must use `NSSecureCoding`
- **Can't easily trigger arbitrary Objective-C deserialization anymore**

Objective-C decoding

- **Strict mode**
  - Applied to WebKit processes

```
- (void)_enableStrictSecureDecodingMode;
```

  - Even more restrictive than Secure Coding mode
    - Can't decode subclasses anymore
    - Stops attacker from decoding some sensitive object fields
- **Breaks some exploitations methods**
- **Very few Objective-C objects can still be decoded in WebKit**

libPas

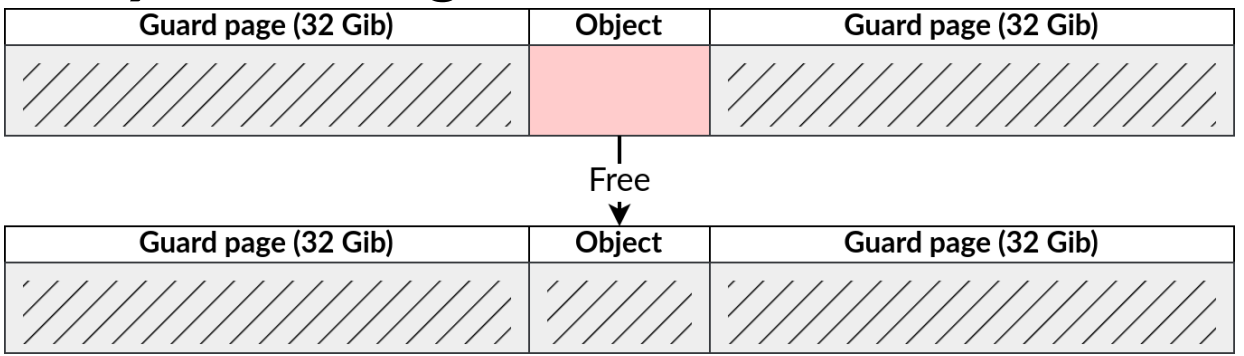
- **WebKit has its own heap allocator**
  - "Libpas is a beast of a malloc, designed for speed, memory efficiency, and type safety.", Filip Pizlo
  - Exposes API
    - *FastMalloc*
    - *ISOHeap*
      - *Still documented*
      - *Few WebKit objects uses this API*
    - *GigaCage, JITHeap*
      - *Not relevant for this talk*

libPas

- **FastMalloc**
  - Based on Thread Local Cache
  - Almost every WebKit object uses this API
  - Sorts allocations based on their sizes
  - Few security protections
    - *Good control over the heap*

libPas

- **Probabilistic Guard Malloc**
  - Tries to catch memory corruption bugs in the wild
    - *Adds guard pages and segregation*
  - 1/1000 probability to have the feature enabled
    - *1 allocation every 4000-5000 is guarded*
- **Not a security hardening**



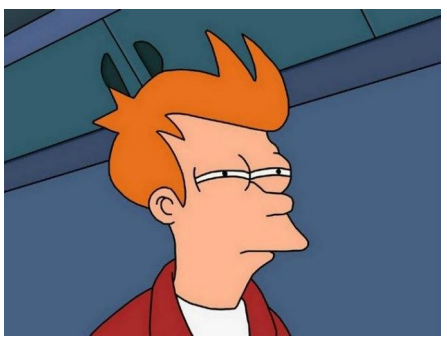
libPas

- **TZone**
  - Disabled by default! (for now…)
  - Objects information is stored in Mach-O section `__tzone_descs`
  - Allocations are stored into buckets
    - Based on their size and alignment
    - AND a random seed
      - Generated by the kernel
    - Can't predicate which objects share the same buckets
  - Tries to break heap-based exploit reliability

Bucket 1	Bucket 2
Object A	Object B
Object A	Object D
Object C	Object B
Object C	Object B
Object A	Object B

Default userland malloc

- **Almost every process uses this heap allocator**
  - Historically hacker friendly[1]
- **iOS 17 introduced a little change...**



```
MOV     W0, #0x58
BL      _malloc
```

iOS 17

```
MOV     W0, #0x58
MOV     X1, #0x100008038113C5C
BL      _malloc_type_malloc
```


1: https://www.synacktiv.com/ressources/Sthack_2018_Heapple_Pie.pdf

Default userland malloc

- **`malloc()` is replaced by `malloc_type_malloc()`**
  - Second parameter is a tag generated by the compiler
  - Looks like a new hardened allocator, but…
- **… `malloc_type_malloc()` still uses the old implementation**
  - The tag is never used (as of iOS 17.4)
  - At least WebKit processes don’t use it
- **Is typed malloc coming to userland?**

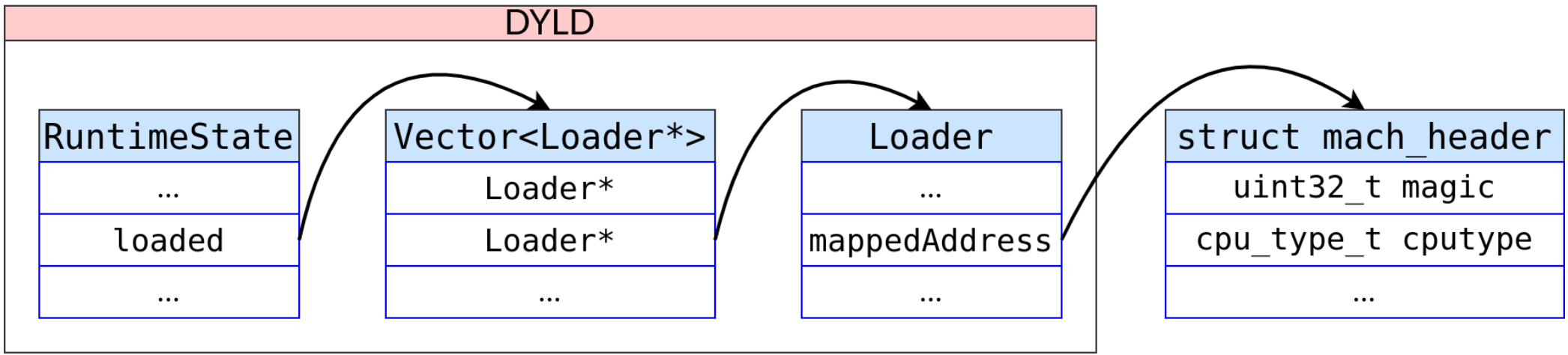
PAC Bypass

- **Need to bypass PAC again outside of WebContent**
  - WebContent has its own PAC keys
- **Latest PAC bypasses targeted the DYLD loader[1][2]**
- **Very interesting target**
  - Lots of optimizations
  - Has to sign pointers at runtime
    - `dlsym()`
    - Relocation

1: https://googleprojectzero.blogspot.com/2023/10/an-analysis-of-an-in-the-wild-ios-safari-sandbox-escape.html
2: https://media.ccc.de/v/37c3-11859-operation_triangulation_what_you_get_when_attack_iphones_of_researchers

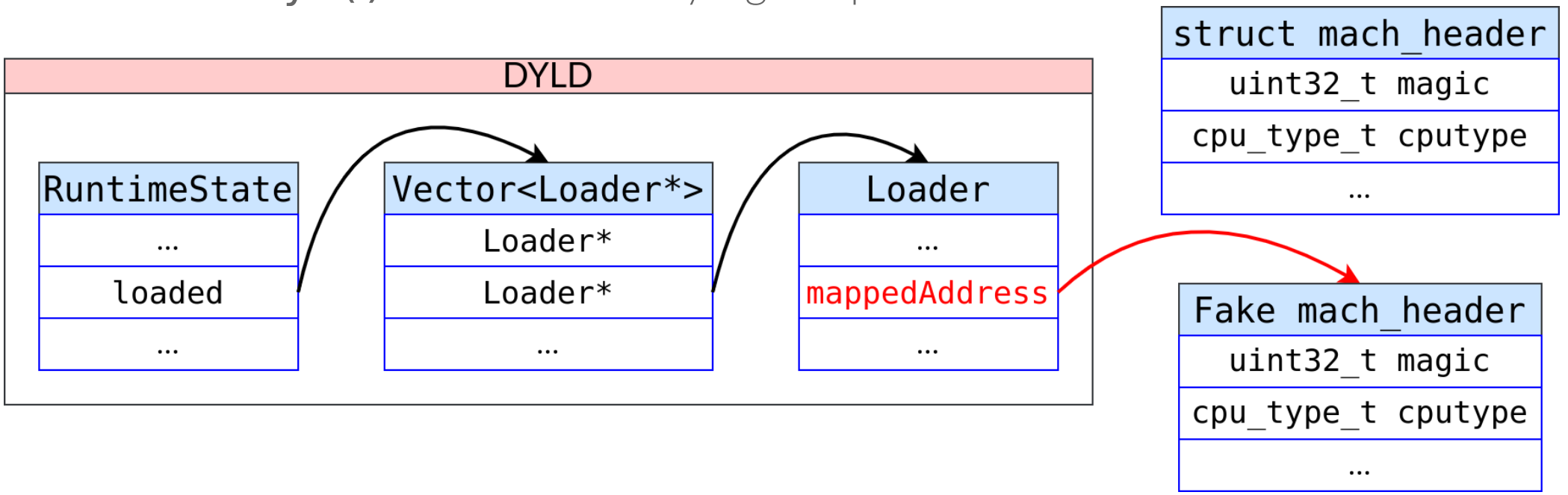
PAC Bypass

- **Structures used to keep information about loaded images**
  - Initially not protected



PAC Bypass

- **Build fake Mach-O in memory**
  - `dlsym()` returns arbitrary signed pointers

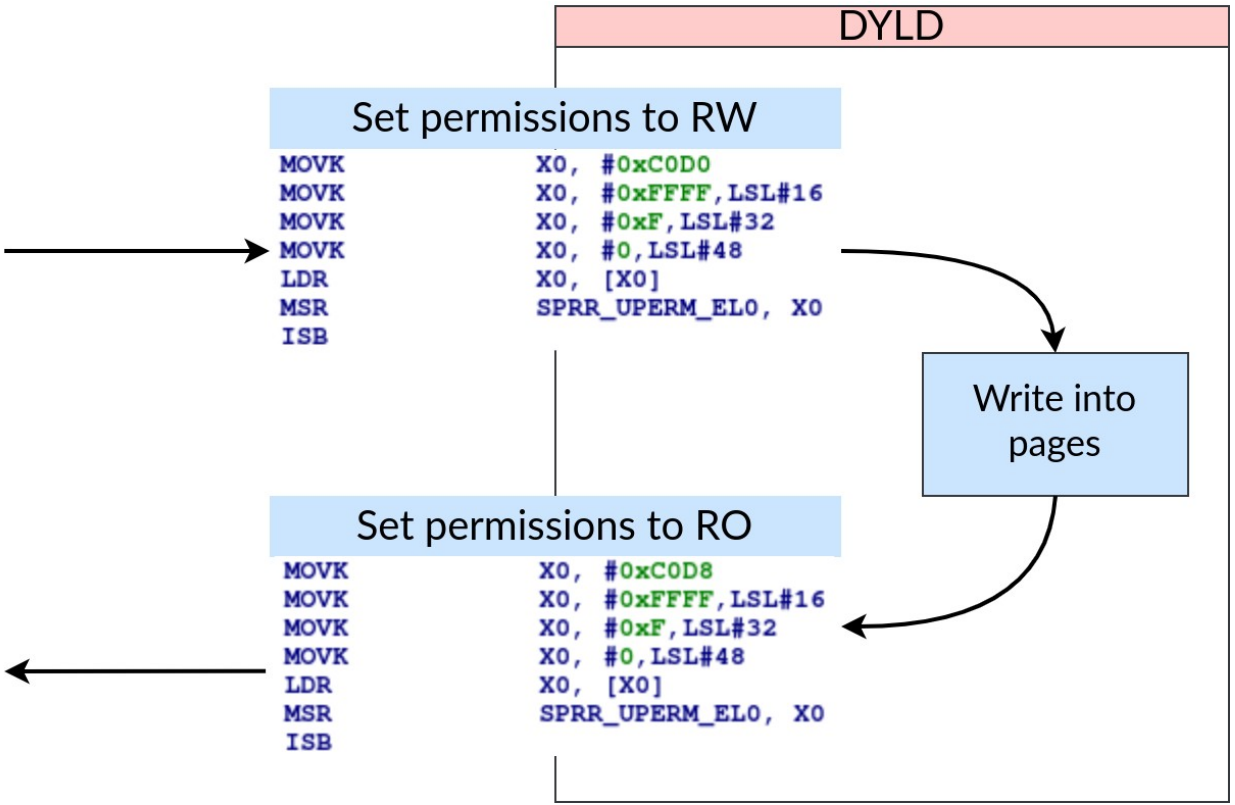


PAC Bypass

- **DYLD now protects its internal structures**
  - Structures are allocated in `VM_PROT_READ` pages
  - Switches to `VM_PROT_WRITE` when it needs to write
  - Switches back to `VM_PROT_READ` after writing
- **Attackers can't corrupt DYLD structures anymore…**
  - …but if attackers can call `mprotect()` they can change pages protections
    - Operation Triangulation did that

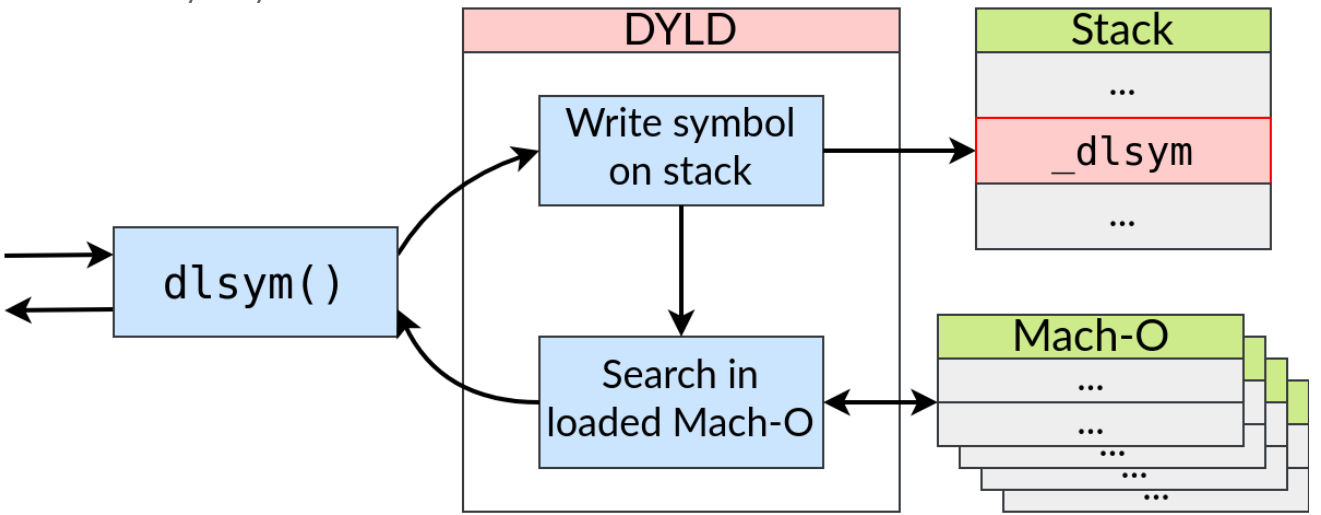
PAC Bypass

- **DYLD pages are now protected using SPRR**
  - Pages mapped with `VM_FLAGS_TPRO`
  - Protections dynamically changed by DYLD
- **Operation Triangulation PAC bypass doesn't work anymore**



PAC Bypass

- **The GPU full chain exploits a race condition in `dlsym()`**
  - Corrupts the symbol name on the stack before it is used
  - Sign arbitrary symbols



Execution context

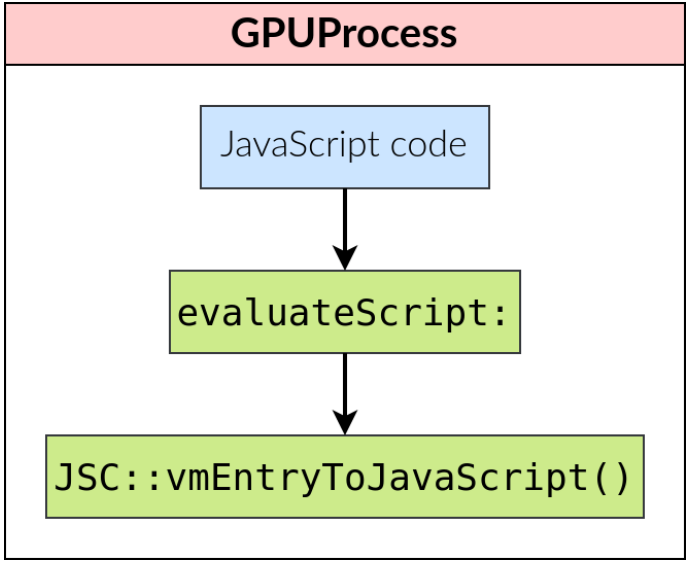
- **Can’t map RWX pages**
  - Only WebContent and few other processes
- **Useful to have an execution context in the compromised process**
  - To pivot into the compromised process
  - To implement the next stage
- **Spawn a JavaScript engine!**

Execution context

- **JavaScriptCore exposes an Objective-C API**

```
- (JSValue*)evaluateScript:(NSString*);
- (JSValue*)objectForKeyedSubscript:(id);
- (void)setObject:(id) forKeyedSubscript:(id);
```

- **Corrupt `JSValue` inside the JavaScript engine**
  - Transfer primitives
- **Apple doesn't like this exploitation method...**



Execution context

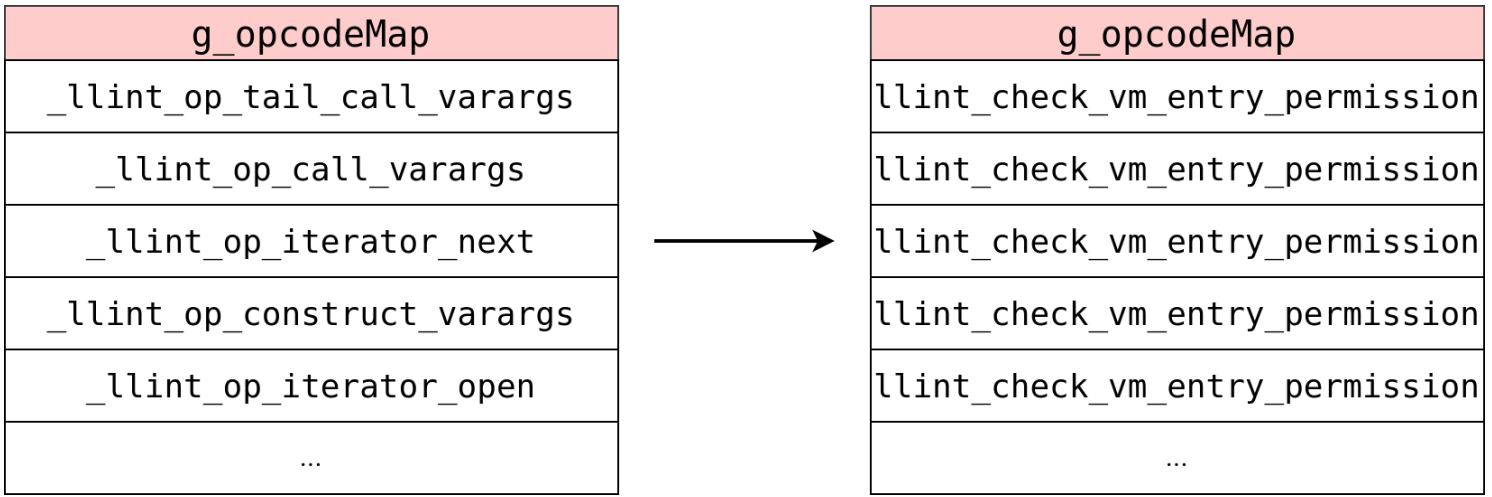
Forbid JS execution in the GPU Process.

https://bugs.webkit.org/show_bug.cgi?id=254101
rdar://106869810

The GPU Process does not need to execute any JS code. We should enforce this invariant.

Execution context

- **Can’t spawn JavaScript engine in the GPU process anymore**
  - Opcode list is trashed at process initialization
  - `VM::VM` initialization is forbidden
    - Or is it?



Execution context

- **Checked in the `VM` constructor**
  - `vmCreationDisallowed` must be set to crash the process

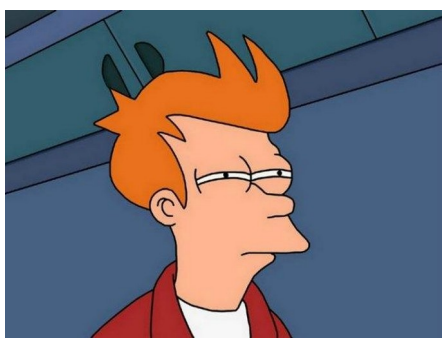
```
VM::VM(VMType vmType, HeapType heapType, WTF::RunLoop* runLoop, bool* success)
// ...
    if (UNLIKELY(vmCreationShouldCrash || g_jscConfig.vmCreationDisallowed))
        CRASH_WITH_EXTRA_SECURITY_IMPLICATION_AND_INFO(/* ... */);
```

Execution context

- **Developers forgot (or not?) to set `vmCreationDisallowed`**

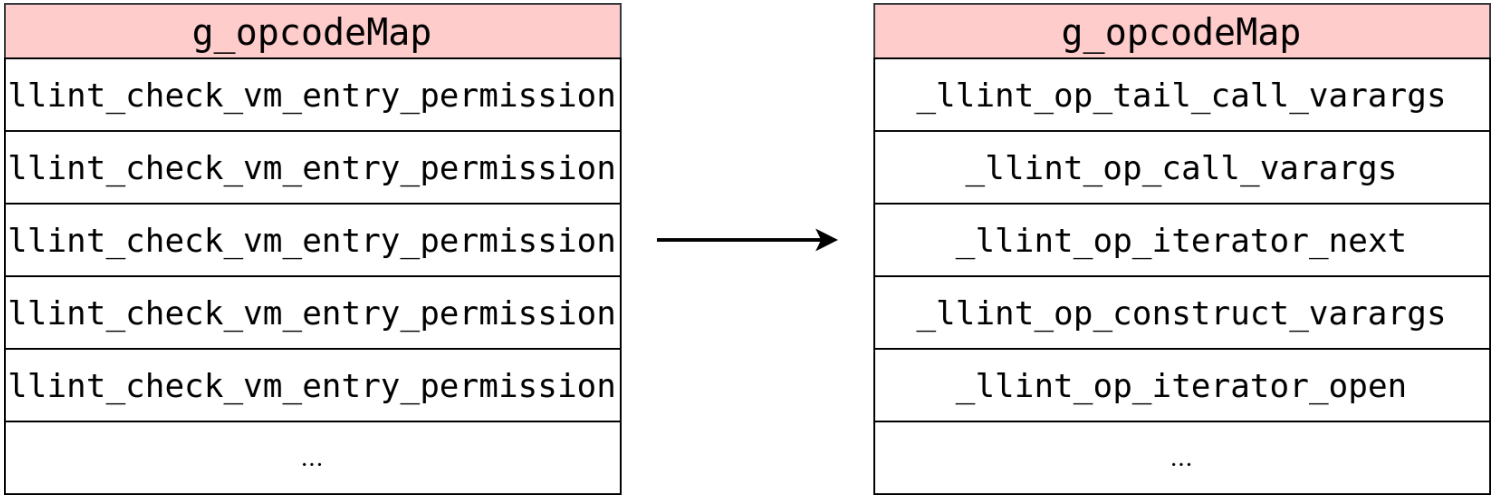
```
void GPU_SERVICE_INITIALIZER(xpc_connection_t connection, xpc_object_t initializerMessage)
{
    g_jscConfig.vmEntryDisallowed = true;
    g_wtfConfig.useSpecialAbortForExtraSecurityImplications = true;

    WTF::initializeMainThread();
```



Execution context

- **Bypass JavaScript engine hardening**
  - PAC bypass is mandatory
  - Restore each signed functions pointers in `g_opcodeMap`
  - Profit!



Conclusion

- **Escaping the WebContent sandbox through WebKit processes looks promising…**
  - … but increases full-chains complexity
- **DYLD is a good PAC bypass target…**
  - … lots of PAC bypasses killed
- **iOS has never had so many userland mitigations…**
  - … but in 2023 attackers were still able to build a full-chain from WebContent :-)

https://www.linkedin.com/company/synacktiv
https://twitter.com/synacktiv
Our publications: https://synacktiv.com