Evolution of Safari mitigations and bypasses

iOS 14-15

Nikita Pupyshev 2023

Introduction

Previous works on the matter
(in the order that I found them while googling)

- Attacking Safari in 2022 by Quentin Meffre (@0xdagger) of Synacktiv
- Attacking JavaScript engines in 2022 by Samuel Groß and Amy Burnett
- JITSploitation series of posts by Samuel Groß (in particular the 3rd post)
- ...

A little recap on JSC internals

Value representation

```
Pointer {  0000:PPPP:PPPP:PPPP
         / 0002:****:****:****
Double  {          ...
         \ FFFC:****:****:****
Integer {  FFFE:0000:IIII:IIII
```

Special pointer values

```
False:     0x06
True:      0x07
Undefined: 0x0a
Null:      0x02
```

- A JSValue is represented as a 64-bit integer.
- Objects are pointers to instances of JSCell subclasses.

A little recap on JSC internals

Representation of objects

- let o = {a: 0x41, b: 0x42};

+ 0x00	JSCell
+ 0x08	Bufferfly Pointer
+ 0x10	Inline Property: ‘a’
+ 0x18	Inline Property: ‘b’

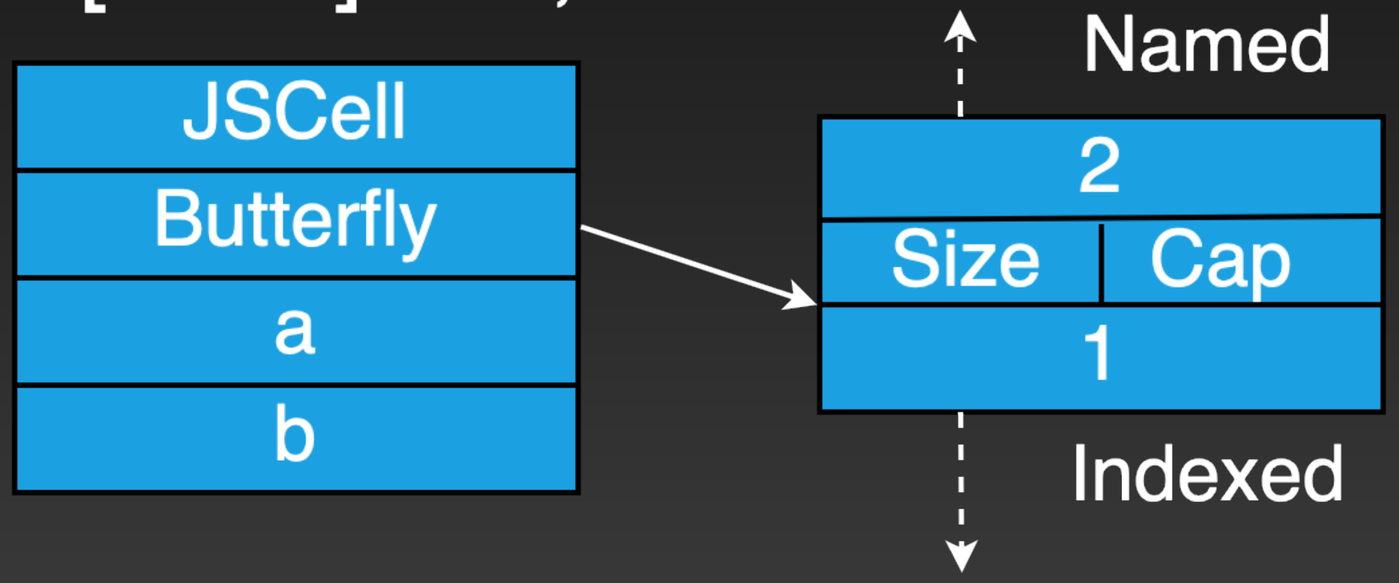
StructureID
IndexingType
JSType
InlineTypeFlags
CellState

Source: BHEU19 Wang, Yong (@ThomasKing2014) Alibaba Security

A little recap on JSC internals

Representation of objects: Butterfly

```
let x = {a: 0x41, b = 0x42};
a[0] = 1;
a["foo"] = 2;
```



Where we at?

Assumed primitives

Butterfly R/W

Fake JSValue construction

Primitives

Butterfly R/W

- Extensively covered in numerous talks.
- You point the butterfly of an object somewhere before or right at your target address.
- You need to ensure that right before there are two 32-bit values that are large enough to be your size and capacity.

Primitives

Fake JSValue construction

This is left as a task to the reader 😉

(visit the Project Zero blog)

Where are we going?

Why not just write to the JIT region?

- Doesn’t work this way on newer Apple devices.
- JIT region is switched from RW to RX using a hardware register (aka fast JIT permissions).
- void pthread_jit_write_protect_np(int enabled)
- We really want to write to the JIT region.

What do we do next?

PAC bypass

What produces signed pointers from unsigned data?

- dlsym produces pointers from export trie; <- easy way
- Objective-C runtime extracts pointers to message implementations from metadata sections;
- Pointers in __got (Global Offset Table) sections are called without a PAC check.
- All of these rely on RO memory being RO to be secure.
- Messing with any of these requires us to call a function.

JSCustomGetterSetterFunction

- Allowed to call a function pointer with up to 4 controlled arguments.
- Allowed to retrieve the returned value.
- Had some limitations we will need to bypass.
- Later split into getter and setter parts in 2b233c8.
- Brutally killed in 757f991.

JSCustomGetterSetterFunction

```
class JSCustomGetterSetterFunction : public JSFunction {
    //...
    enum class Type { Getter = 0, Setter = 1 };
    //...
    WriteBarrier<CustomGetterSetter> m_getterSetter;
    Type m_type;
    PropertyName m_propertyName;
};
```

JSCustomGetterSetterFunction

```
class CustomGetterSetter : public JSCell {
    //...
    using CustomGetter = GetValueFunc;
    using CustomSetter = PutValueFunc;
    //...
    CustomGetter m_getter;
    CustomSetter m_setter;
};
```

JSCustomGetterSetterFunction

```
using GetValueFunc = EncodedJSValue(JIT_OPERATION_ATTRIBUTES*)(
  JSGlobalObject*, EncodedJSValue, PropertyName
);

using PutValueFunc = bool (JIT_OPERATION_ATTRIBUTES*)(
  JSGlobalObject*, EncodedJSValue, EncodedJSValue, PropertyName
);
```

JSCustomGetterSetterFunction

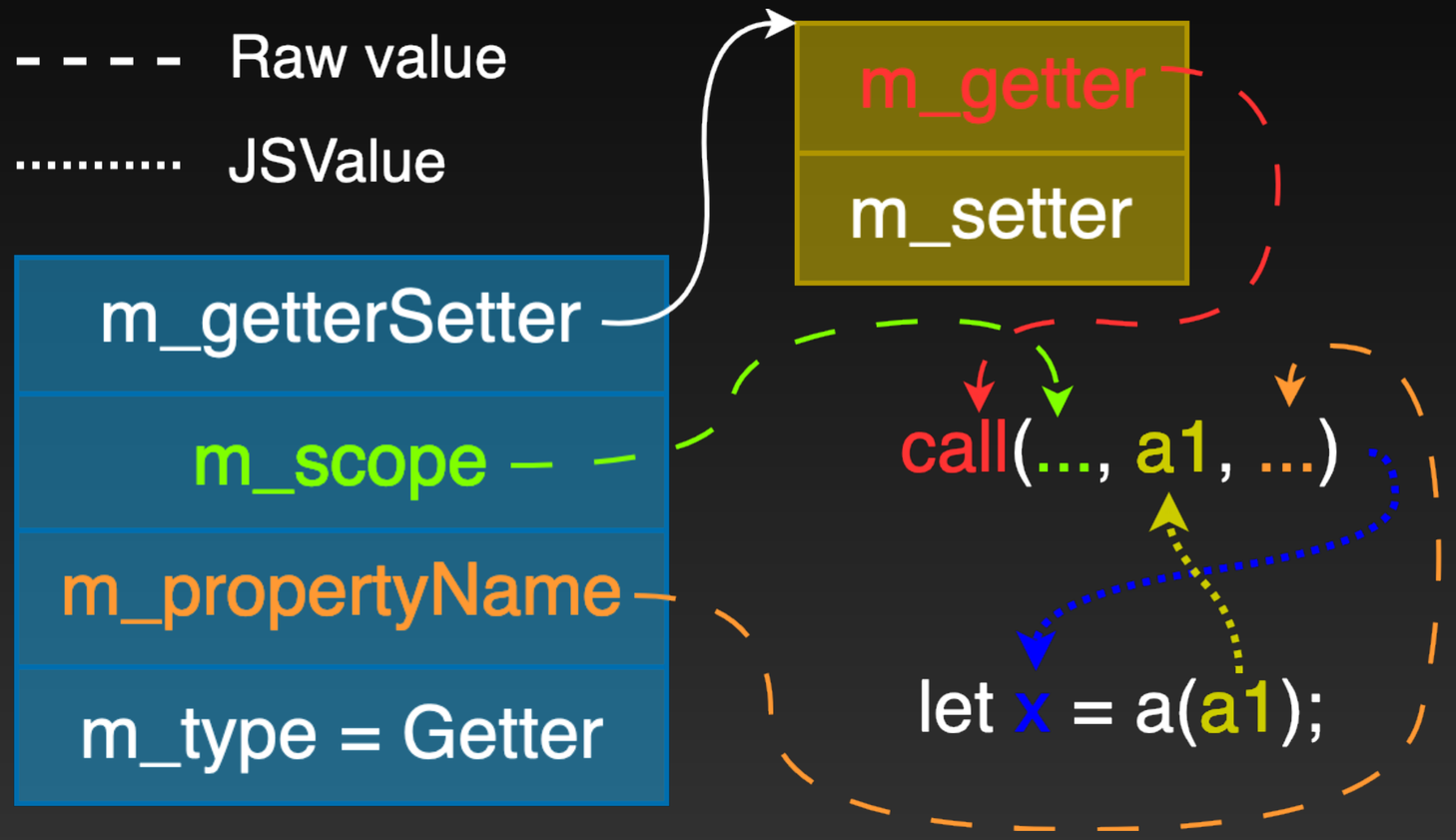
Construction

```
let a = window.__lookupGetter__("name");
let b = window.__lookupSetter__("name");
```

Any r/w window property works.

JSCustomGetterSetterFunction

Getters



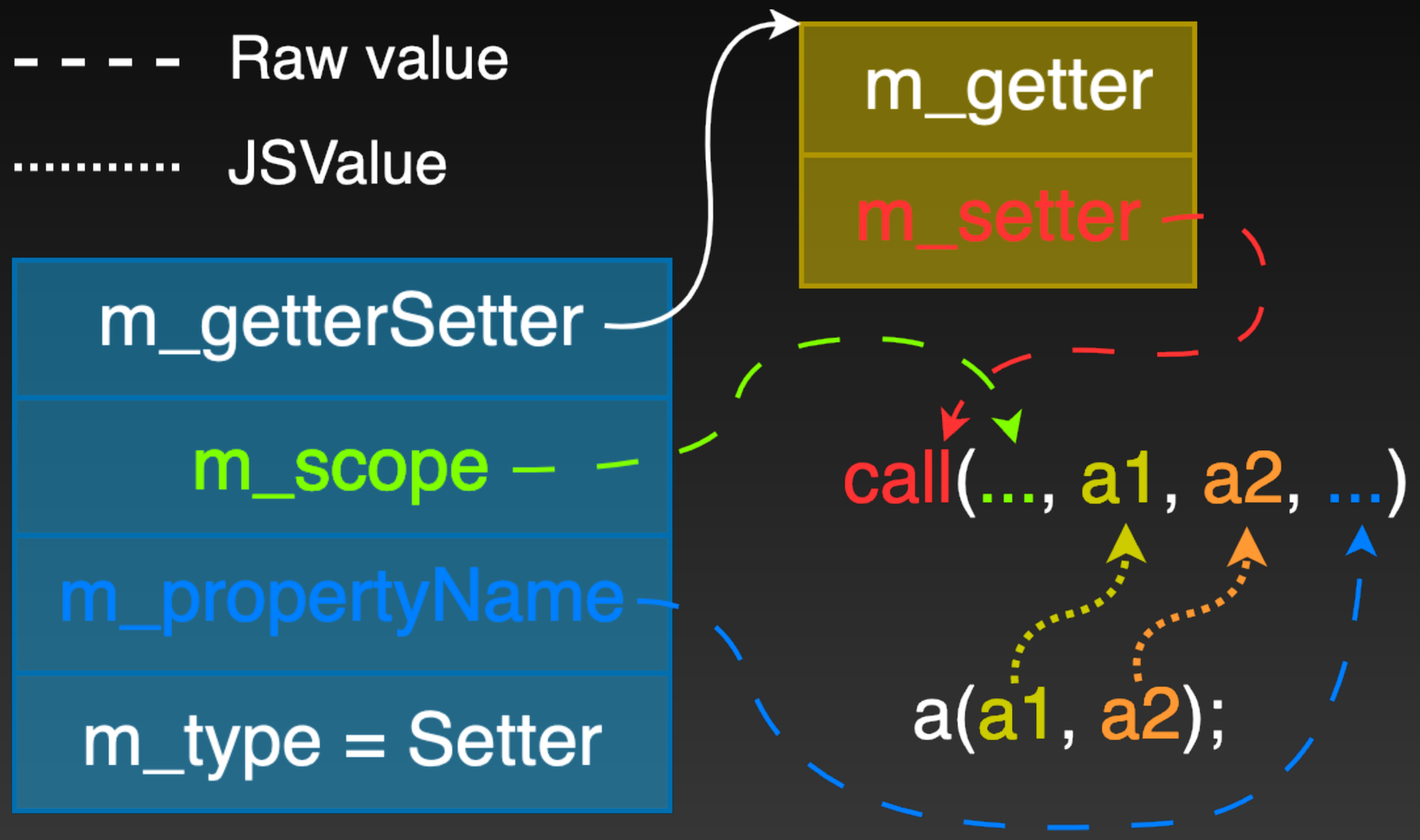
JSCustomGetterSetterFunction

Getter restrictions

- customGetterFunctionCall dereferences the first argument, it must be a valid pointer.
- The returned value is treated as a JSValue and in case it is a pointer the JSC's profiler will record it into one of its buckets and later crash.

JSCustomGetterSetterFunction

Setters



JSCustomGetterSetterFunction

Setter notes

- The first argument may be something other than a pointer, no longer dereferenced.
- No way to get the returned value.
- 2nd and 3rd arguments are JSValue and will thus be profiled if these are pointers.

JSCustomGetterSetterFunction

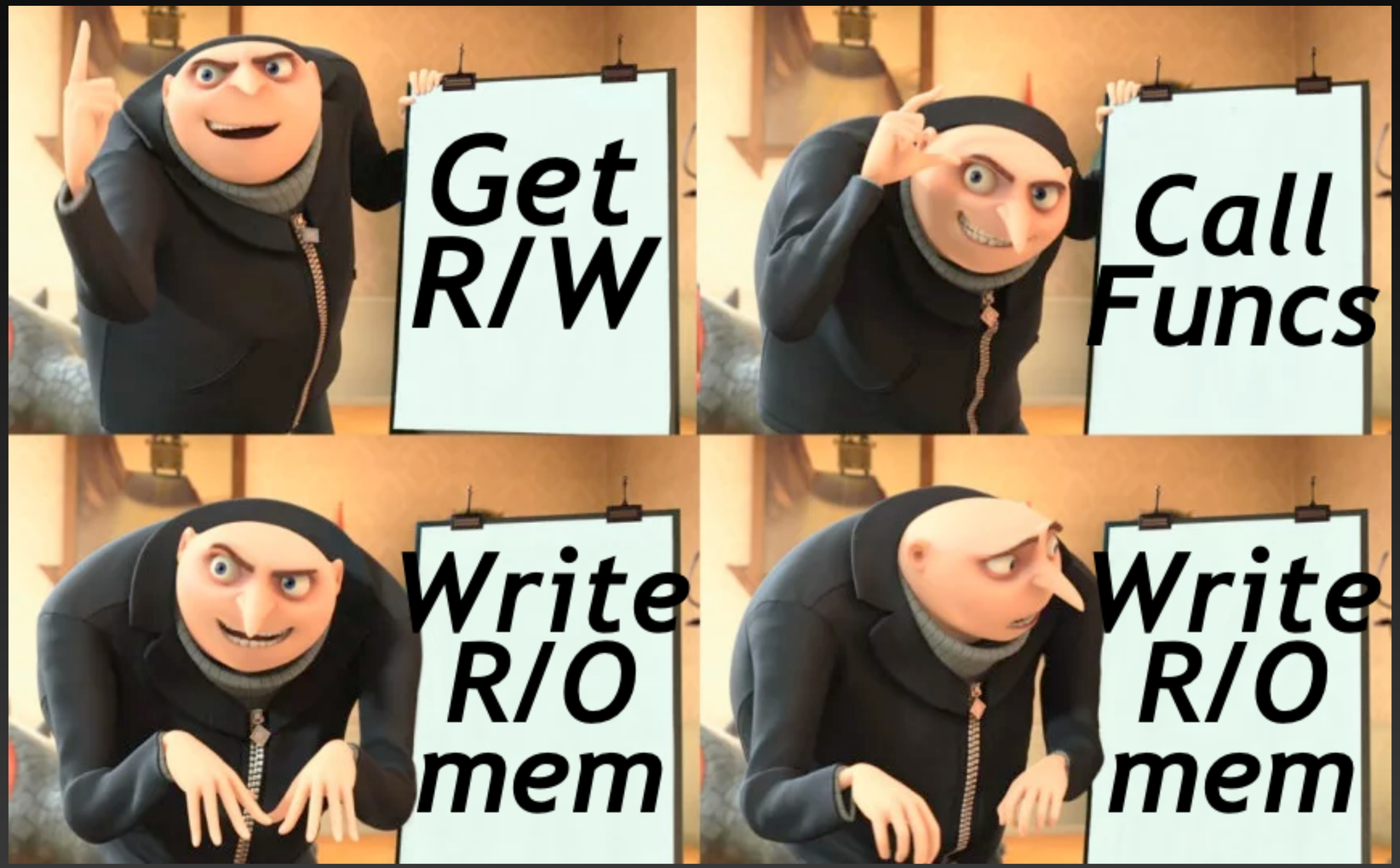
Avoiding the profiler with TBI

- TBI (Top Byte Ignore) allows pointers to have any data in the top 8 bits.
- Double JSValues can be used to pass pointers as function arguments.
- Not useful with objc_msgSend bc of how ObjC selectors work.

```
Pointer {  0000:PPPP:PPPP:PPPP
         / 0002:****:****:****
Double  {          ...
         \ FFFC:****:****:****
Integer {  FFFE:0000:IIII:IIII
```

PAC bypass

dlsym: the easy way?



PAC bypass
The solution

- Calling mmap with an address of any region in the DSC with MAP_FIXED will replace this region with newly allocated zeroed out pages.
- Call mmap on a page belonging to an export table of a target library.
- Restore the contents of the pages while patching an offset of any function to another one.
- Call dlsym with a handle to that library.
- Profit! A pointer to an arbitrary location signed with key IA ctx 0.

PAC bypass

Needed function pointers

- dlopen - exists in DSC at the time ✅
- dlsym - exists in DSC at the time ✅
- memmove - exists in DSC at the time ✅
- mmap - not directly available in DSC ❌

PAC bypass

Solving mmap issue

```
inline void vmZeroAndPurge(void* p, size_t vmSize, VMTag usage)
{
    vmValidate(p, vmSize);
    // ...
    void* result = mmap(
      p, vmSize, PROT_READ | PROT_WRITE,
      MAP_PRIVATE | MAP_ANON | MAP_FIXED | BMALLOC_NORESERVE,
      static_cast<int>(usage), 0
    );
    RELEASE_BASSERT(result == p);
}
```

PAC bypass

Solving mmap issue

```
void* tryLargeZeroedMemalignVirtual(
  size_t requiredAlignment, size_t requestedSize, HeapKind kind
) {
    // ...
    Heap& heap = PerProcess<PerHeapKind<Heap>>::get()->at(kind);
    // ...
    result = heap.allocateLarge(
        lock, alignment, size, FailureAction::ReturnNull);
    // ...
    if (result)
        vmZeroAndPurge(result, size);
    return result;
}
```

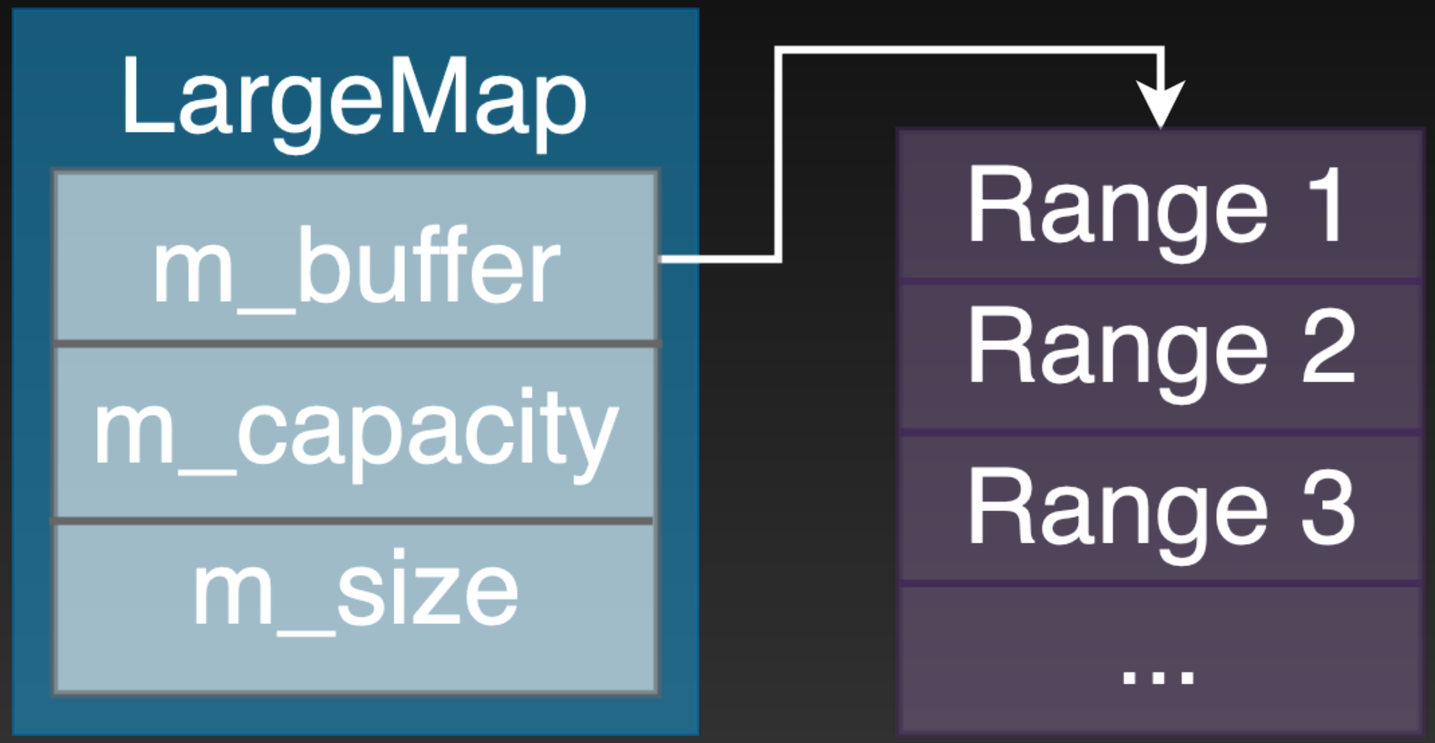
PAC bypass

Solving mmap issue

- Allocating a large buffer from a Heap will lead to mmap being called on it.
- Gigacage happens to be managed by a Heap.
- Wasm.Memory objects are allocated from the Gigacage.
- Modifying the allocator’s freelist will give us our mmap primitive.

PAC bypass

(Ab)using the allocator



PAC bypass

Needed function pointers

- dlopen - exists in DSC at the time ✅
- dlsym - exists in DSC at the time ✅
- memmove - exists in DSC at the time ✅
- mmap - abuse the memory allocator ✅

Writing to the JIT region

NSInvocation yet again

- We can call arbitrary functions now.
- We want to chain pthread_jit_write_protect_np and memcpy calls.
- Create a fake NSArray of 3 fake NSInvocation objects and call makeObjectsPerformSelector: selector on the array.
- Custom function pointer is passed by nesting into another invocation that calls _invokeWithIMP:.

The technique is stolen from Hack Different: Pwning iOS 14 with Generation Z Bugz by Zhi Zhou and Jundong Xie.

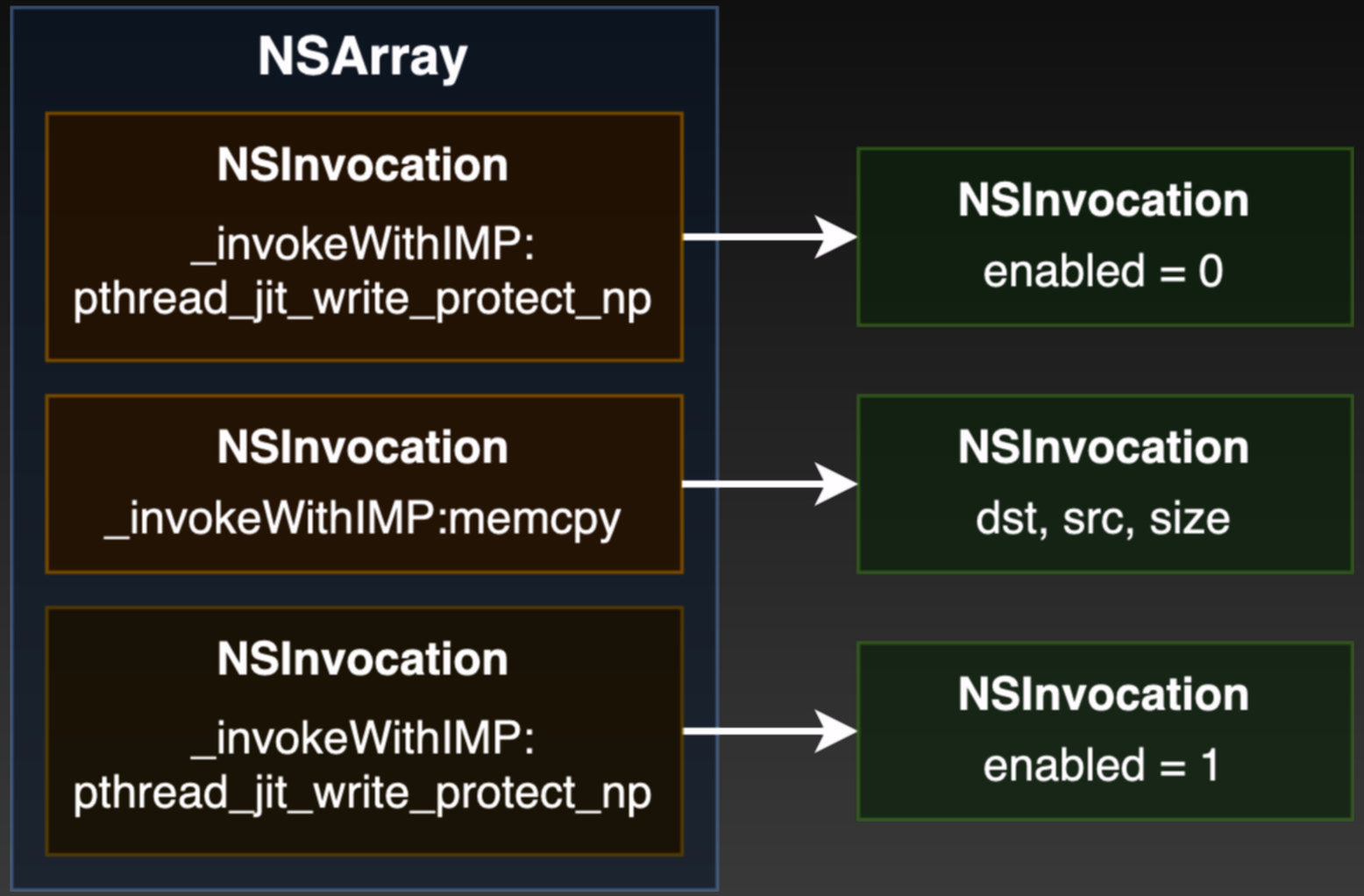
Writing to the JIT region

NSInvocation before ~14.3

```
struct NSInvocation {
  Class *isa;
  void *_frame; // registers X0-X8, FP regs
  void *_retdata; // same as frame, but after the call
  NSMethodSignature *_signature;
  // ...
  uint32_t _magicCookie; // we get it from a static var
  // ...
};
```

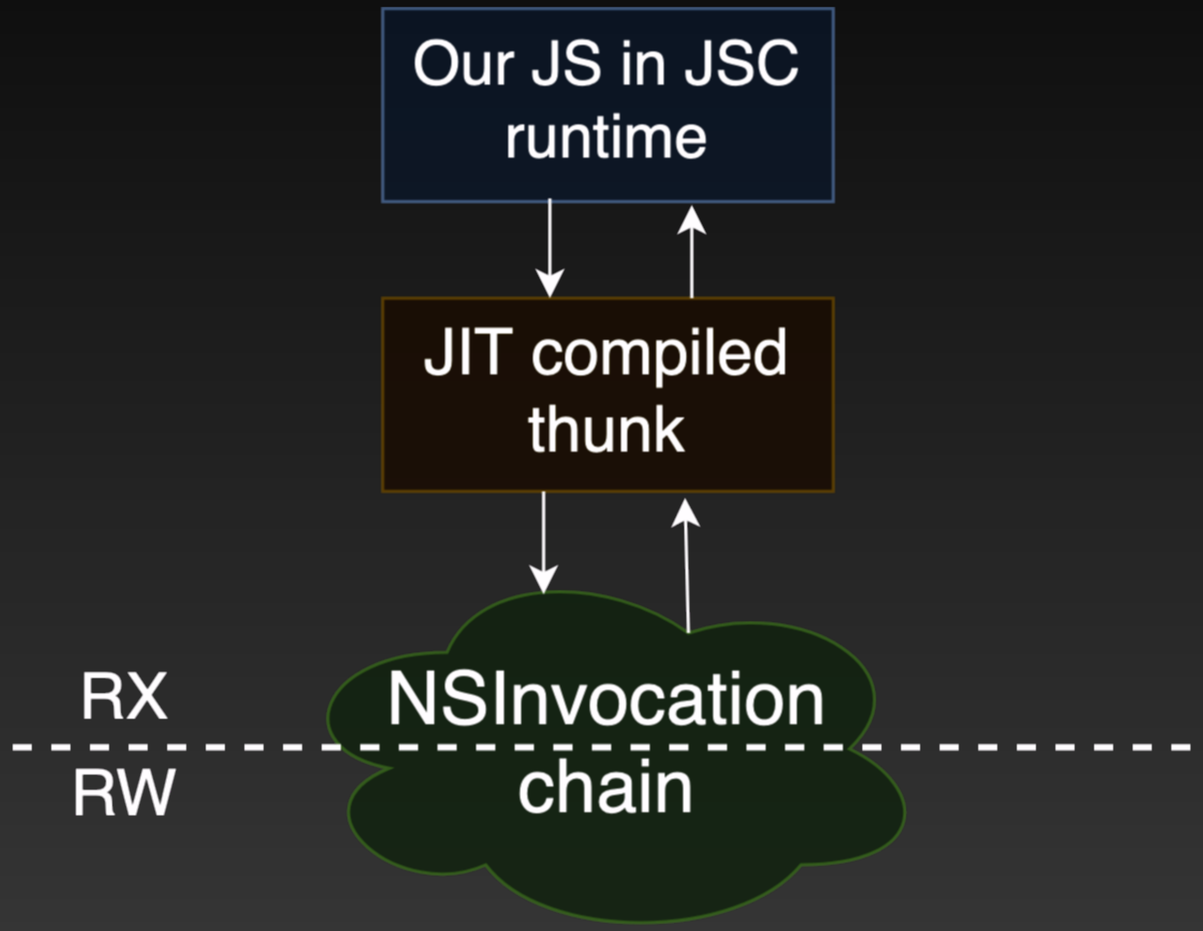
Writing to the JIT region

Chaining invocations



Writing to the JIT region

Avoiding crashes



Profit

JIT is pwned

Evolution

Bypassing new mitigations

NSInvocation after ~14.3

```
struct NSInvocation {
  Class *isa;
  void *_frame; // registers X0-X8, FP regs
  void *_retdata; // same as frame, but after the call
  NSMethodSignature *_signature;
  // ...
  void *_signedSelf; // signed X0 value
  void *_signedSelector; // signed X1 value
  uint32_t _magicCookie; // we get it from a static var
  // ...
};
```

Bypassing new mitigations

NSInvocation after ~14.3

_signedSelf and _signedSelector can be set to NULL to avoid the PAC check.

Bypassing new mitigations

JIT hardening in 14.4

- pthread_jit_write_protect_np is inlined, impossible to obtain a pointer using dlsym.
- Use the legit JIT API to construct a trampoline that calls the end of any MacroAssemblerARM64 method that appends an instruction.
- Use LinkBuffer::linkCode to copy the code to the JIT region.
- Use AssemblerARM64::relinkJump to point a JIT-compiled function to that code.

Bypassing new mitigations

JIT hardening in 14.5

- MacroAssemblerARM64::farJump now requires a pointer PAC-signed with key IB and context 18705, like InternalFunction.
- Use InternalFunction constructor to resign the pointer from key IA ctx 0.

Bypassing new mitigations

NSInvocation after ~15.0

- Setting _signedSelf to NULL no longer works.
- Setting one of the top bits does though since tagged Objective-C values exist.
- Same for _signedSelector, but not checked in case the method signature's second argument is not a selector.
- ISA pointer is now signed. An object may be constructed by obtaining selector implementations.

Bypassing new mitigations

NSInvocation after ~15.1

- Most of the fields are signed with PACGA now.
- Arguments can still be set by calling the implementation of setArgument:atIndex:.
- Need to create an NSMethodSignature now with a large enough number of arguments.

Bypassing new mitigations

Other mitigations

- Little changes in how AssemblyBuffers work, signing in one thread and linking in another isn’t possible anymore. Doesn’t affect this flow at all.
- Some static function pointers were removed. Either replaced by gadgets or taken from the heap buffers they were moved into.
- Introduction of libpas forced to find a new way to call mmap.

Effective Mitigations

Removing function call primitives

Apply C++ type safety to Lookup.h's HashTableValue's ValueStorage union.

https://bugs.webkit.org/show_bug.cgi?id=243680
<rdar://problem/98206776>

Reviewed by Saam Barati.

```
- using GetValueFunc = EncodedJSValue(JIT_OPERATION_ATTRIBUTES*)(JSGlobalOb
- using GetValueFuncWithPtr = EncodedJSValue(JIT_OPERATION_ATTRIBUTES*)(JSG
+ using GetValueFunc = TypedFunctionPtr<GetValueFuncPtrTag, EncodedJSValue(
  FunctionAttributes::JITOperation>;
+ using GetValueFuncWithPtr = TypedFunctionPtr<GetValueFuncWithPtrPtrTag, E
```

Effective Mitigations

Removing function call primitives

- PAC is strong if applied universally and with different contexts/keys/diversifiers.
- This will be bypassable one way or another via non-obvious means.

Effective Mitigations

TPRO

```
/*
 * The X0 index is used for TPRO mappings. To avoid exposing them as --x,
 * the VM code tracks VM_MAP_TPRO requests and couples them with the proper
 * read-write protection. The PMAP layer though still needs to use the right
 * index, which is the older X0-now-TPRO one and that is specially selected
 * here thanks to PMAP_OPTIONS_MAP_TPRO.
 */
if (options & PMAP_OPTIONS_MAP_TPRO) {
    pte = pmap_construct_pte(
      pmap, v, pa, VM_PROT_RORW_TP, fault_type, wired, pt_attr, &pp_attr_bits);
} else {
    pte = pmap_construct_pte(
      pmap, v, pa, prot, fault_type, wired, pt_attr, &pp_attr_bits);
}
```

Effective Mitigations

TPRO

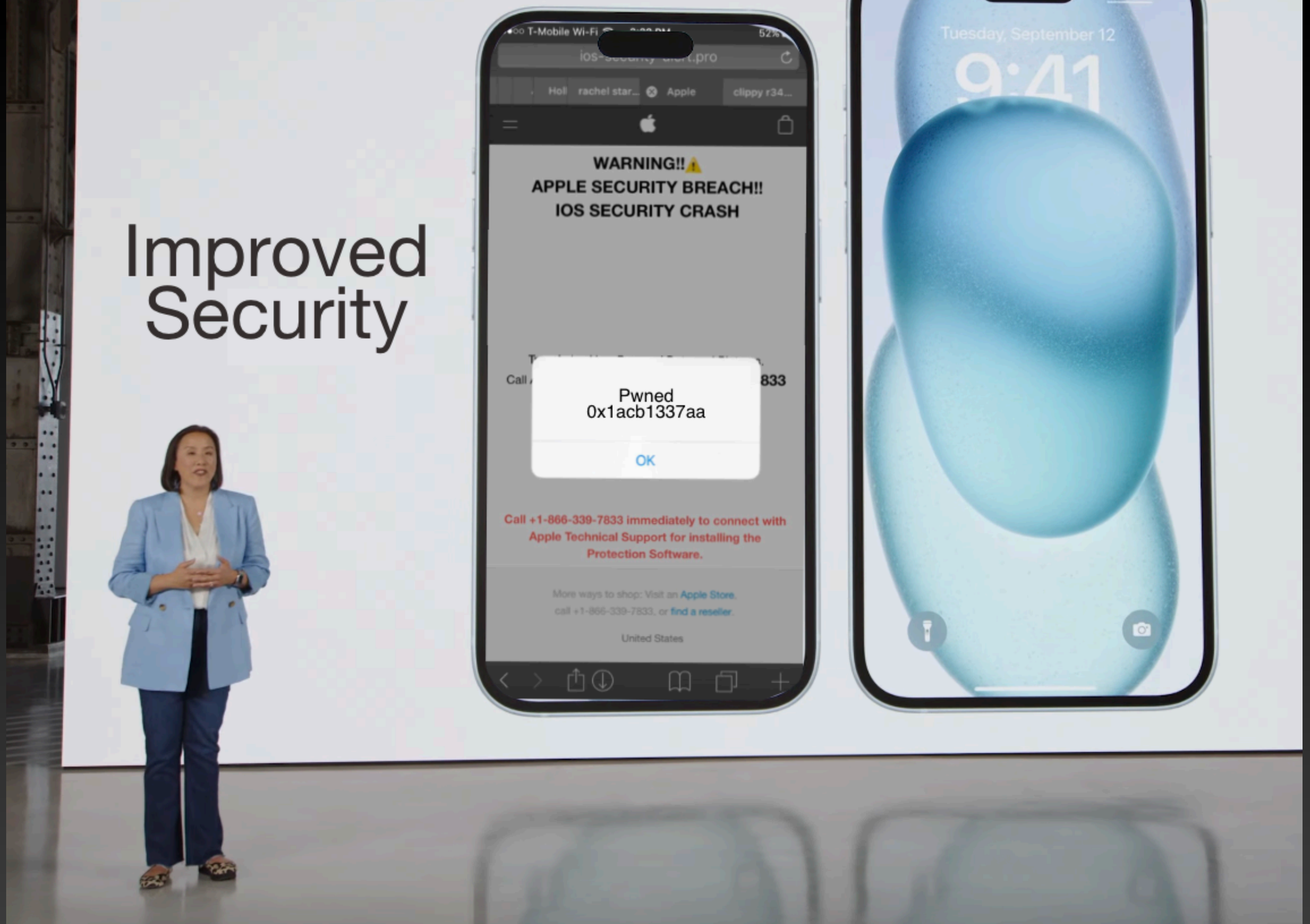
- A new hardware mitigation.
- Prevents regions that are initially RW but set to RO by dyld from being remapped.
- Uses SPRR to remap execute-only permissions to read-only.
- Little information available.

Effective Mitigations

What else can be done?

- Reduce the WebContent attack surface (introduction of the GPU process).
- Reduce the amount of code present in a process' address space (whole DSC is mapped).
- Move to safe languages where possible.

Demo


Improved Security
T-Mobile Wi-Fi
52%
rachel star...
Apple
clippy r34...
WARNING!!
APPLE SECURITY BREACH!!
IOS SECURITY CRASH
Pwned
0x1acb1337aa
OK
Call +1-866-339-7833 immediately to connect with Apple Technical Support for installing the Protection Software.
More ways to shop: Visit an Apple Store, call +1-866-339-7833, or find a reseller.
United States
Tuesday, September 12
9:41

Questions?