THIS IS FOR THE PLAYERS PWNERS

QUENTIN MEFFRE & MEHDI TALBI
1 Introduction & Motivation
2 Attack Surface
3 The bug
4 The FastMalloc Allocator
5 Exploitation
6 Conclusion & Future work
Who are we?

Mehdi Talbi
@abu_y0ussef
- Security researcher @Synacktiv
- Academia (in a previous life)
- Vulnerability research & exploitation

Quentin Meffre
@0xdagger
- Security researcher @Synacktiv
- Vulnerability research & exploitation

Synacktiv
- Offensive security company
- Based in France
- ~70 Ninjas
- We are hiring!!!
Introduction

Disclaimer

- This research is done purely out of curiosity and presented for educational purposes.
- This research does not help/support/enable/endorse to break the copyright law.
Introduction

Motivation
- Active console hacking community...
- .. but only few public exploits

Goal
- A walk through of a 0-Day WebKit Exploit
- How hard is it to exploit a vulnerability on the PS4?
Outline

1 Introduction & Motivation

2 Attack Surface

3 The bug

4 The FastMalloc Allocator

5 Exploitation

6 Conclusion & Future work
PS4 attack surface
PS4 attack surface

Exploit chain

- Typical exploit chain: Webkit exploit → Kernel exploit

Attacking the browser

- WebKit-based browser
- Sandboxed
- No JIT
- No modern mitigations
  - Gigacage
  - StructureID randomization
- ASLR. Weak? Partial?
- ... And no debug
## WebKit exploits

### CVE-2018-4386
- Found by Lokihardt (from P0)
- A.k.a Bad-Hoist exploit by @Fire30
- Last known public exploit
- Arbitrary Read/Write primitives
- Works on 6.00-6.72 firmwares

### CVE-2018-4441
- Found by Lokihardt (from P0)
- Exploit by @SpecterDev
- Arbitrary Read/Write primitives
- Works on 6.00-6.20 firmwares

### More exploits ...
- For older firmwares (< 6.xx)
- By @qwertyoruiopz, @SpecterDev, @CTurt, ...
Kernel exploits

CVE-2020-7457
- Reported by @theflow0
- Kernel Read/Write primitives
- Reachable from WebKit Sandbox
- Present in firmware 7.02 and 6.xx
- Used in conjunction with Bad-Hoist exploit

Berkeley Packet Filter vulnerability
- Discovered and exploited by @qwertyoruiopz
- Works on firmwares up to 5.07.
- Excellent write-up by @SpecterDev
<table>
<thead>
<tr>
<th></th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction &amp; Motivation</td>
</tr>
<tr>
<td>2</td>
<td>Attack Surface</td>
</tr>
<tr>
<td>3</td>
<td>The bug</td>
</tr>
<tr>
<td>4</td>
<td>The FastMalloc Allocator</td>
</tr>
<tr>
<td>5</td>
<td>Exploitation</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion &amp; Future work</td>
</tr>
</tbody>
</table>
Introducing the bug (1/2)

Presentation

- Vulnerability in WebKit DOM engine
- Triggered by our internal fuzzers
- Impacts all PS4 firmwares (and PS Vita as well) prior to 8.00
- Reported to Sony through their Bug Bounty Program
- Awarded 2500$
- Fixed on Webkit: **11 Sept. 2020**
Introducing the bug (2/2)

The vulnerable code

- Use-After-Free in `WebCore::ValidationMessage::buildBubbleTree` method
- Extra dereference while making a weak pointer
- `ValidationMessage` could be destroyed during a layout update
- `updateLayout` runs user registered JS callbacks

```cpp
void ValidationMessage::buildBubbleTree()
{
    /* ... */

    auto weakElement = makeWeakPtr(*m_element);

    document.updateLayout(); // [1] call user registered JS events

    if (!weakElement || !m_element->renderer())
        return;

    adjustBubblePosition(m_element->renderer()->absoluteBoundingBoxRect(), m_bubble.get());

    /* ... */
} 
```
Fixing the code (1/2)

The so close fix

```cpp
void ValidationMessage::buildBubbleTree()
{
    /* ... */
    auto weakElement = makeWeakPtr(*m_element);
    document.updateLayout();
    if (!weakElement || !m_element->renderer())
        return;
    adjustBubblePosition(m_element->renderer()->absoluteBoundingBoxRect(), m_bubble.get);
    /* ... */
}
```

Protect frames during style and layout changes

https://bugs.webkit.org/show_bug.cgi?id=198047
<rdar://problem/50954082>

Reviewed by Zalan Bujtas.

Be more careful about the scope and lifetime of objects that participate in layout or style updates. If a method decides a layout or style update is needed, it needs to confirm that the elements it was operating on are still valid and needed in the current operation.
Fixing the code (2/2)

The good fix

Avoid doing layout update in `ValidationMessage::buildBubbleTree`

```cpp
void ValidationMessage::buildBubbleTree()
{
    /* ... */

    auto weakElement = makeWeakPtr(*m_element);
    document.updateLayout();

    if (!weakElement || !m_element->renderer())
    return;

    adjustBubblePosition(m_element->renderer()->absoluteBoundingBoxRect(), m_bubble.get());

    /* ... */

    if (!document.view())
        return;
    document.view()->queuePostLayoutCallback([weakThis = makeWeakPtr(*this)] {
        if (!weakThis)
            return;
        weakThis->adjustBubblePosition();
    });
}
```
The vulnerable path

html input

- reportValidity
- Instantiate ValidationMessage
- buildBubbleTree
- Layout Update
- Free ValidationMessage
- JS Callback
- Destroy ValidationMessage

- onFocus

0s
- buildBubbleTree
- UAF ValidationMessage

5s
- deleteBubbleTree
- Destroy ValidationMessage
Triggering the bug (1/2)

First Attempt

1. Register a JS event (e.g. onfocus) on some input text field.
2. Instantiate a `ValidationMessage` object → Fire-up a timer to call `buildBubbleTree` → Run user registered JS events
3. Destroy `ValidationMessage` instance on JS callback
4. No crashes!!
   - `reportValidity` sets the focus on input
   - user JS callback called too early.

PS4 exploit
https://pwnme.org/

```javascript
input.setCustomValidity("pwn");
input.reportValidity();
input.autocus = true;
```
Triggering the bug (2/2)

Solution

1. Register a JS event handler `handler1` on `input1`
2. Instanciate a `ValidationMessage` (on `input1`)
   - focus is set on `input1` → `handler1` is executed
   - `handler1` sets the focus elsewhere (`input2`)
3. Set `handler2` as new handler for focus event on `input1`
4. `handler2` is executed while running JS user callback from `buildBubbleTree`
   - Destroy `ValidationMessage` instance
5. PS4 browser crashes and restarts
Crash!

There is not enough free system memory.
### Problem
- No debugging capabilities on PS4
- All we get are crashes :-(

### Option 1: Setup a similar environment
- Install a FreeBSD box
- Compile WebKit sources from [doc.dl.playstation.net](http://doc.dl.playstation.net)
- Helpful **BUT** working exploit on our env does not fully work on PS4
- MORE DEBUG
Option 2: Debugging a 0-day with a 1-day

- Get insights on memory mappings
- Dump the content on allocated pages
- Use bad-hoist exploit by @Fire30_
  (+) Read/Write primitives
  (+) Addrof/fakeobj primitives
  (-) Works on 6.xx firmware only
  (-) Adds some noise on heap shaping
  (-) Reliability

Also while I am on twitter :P
github.com/Fire30/bad_hoi...

PS4 Webkit exploit for 6.XX consoles. Gains addrof/fakeobj and arbitrary read and write primitives. Fixed in 7.00.
Anatomy of a vulnerable object

**ValidationMessage object**

- Instantiated by `reportValidity()` (fastMalloc'ed)
- Accessed by `buildBubbleTree()`
- Destroyed by `deleteBubbleTree()`

```
m_element
m_message
m_messageHeading
m_messageBody
m_timer
m_bubble
```

**HTML Element**

- Instantiated after layout update
- Accessed after layout update
Surviving an (inevitable) crash (1/3)

Back from user JS callback

- 2 UAFs: `this` and `m_element` are freed
  - But we still have a reference on `m_element`
- Crash on first virtual call (on `m_bubble`)
- Situation: Not comfortable

Exploitability

1. A memory Leak, Or
2. ... An ASLR Bypass
Surviving an (inevitable) crash (2/3)

Bypassing ASLR

- Heap spraying → objects end-up allocated at a **predictable location**!!
  - Spraying ~ 2MB is enough to predict a heap address
- Require a prior knowledge on the memory mapping
  - Works on 6.xx firmware
  - May work on 7.xx. More on this later ...
Surviving an (inevitable) crash (3/3)

Surviving the crash

- Spray HTMLElement obj. (e.g. HTMLTextAreaElement)
- Shape the heap → Reuse ValidationMessage Obj.
Exploitation primitive

Vulnerable path epilogue

```cpp
void ValidationMessage::deleteBubbleTree()
{
    if (m_bubble) {
        m_messageHeading = nullptr;
        m_messageBody = nullptr;
        m_element->userAgentShadowRoot()->removeChild(*m_bubble);
        m_bubble = nullptr;
    }
    m_message = String();
}
```

Exploitation primitive

- `nullptr` assignment on refcounted classes are overloaded
- `refcount decrement on multiple controlled ValidationMessage pointer fields`
- UAF ➔ Arbitrary Decrement (refcount decrement)
- Exploitable
- Requires multiple heap shaping/spraying stages
Outline

1. Introduction & Motivation
2. Attack Surface
3. The bug
4. The FastMalloc Allocator
5. Exploitation
6. Conclusion & Future work
Webkit Allocator

Many allocators

- **FastMalloc**: standard allocator
- **IsoHeap**: sort each allocation using its type to mitigate Use-After-Free
  - Used by the DOM engine
- **Garbage Collector**: used to allocate **JSObject(s)**
- **IsoSubspace**: same as the **IsoHeap** but used in the Javascript engine
- **Gigacage**: provide mitigation to prevent out-of-bound R/W on specific objects
  - Disabled on PS4
The Primary Heap Allocator

Overview

- **Heap** is made of **chunks**
- **Chunk** split into **pages** (4 kB)
- **Page** divided into **lines** (256 Bytes)
- **Line** holds several **objects**
- Each page serves allocations for same-sized obj.
The Fast Path

- Bump Allocator (per size class)

```c
--m_remaining;
char* result = m_ptr;
m_ptr += m_size;
return result;
```
FastMalloc (2/2)

The Slow Path

- No more available free slots → Refill allocator:
  1. From cache *BumpRangeCache* (fast path)
  2. From newly allocated page (slow path)
     - After processing previously released obj.

Refilling the allocator - The slow path

- Allocate a new page
  1. Pick it from cache (another one)
  2. Pick the last released page from the last allocated chunk
- Fill allocator with the first free contiguous lines
- Fill the cache with the rest of the freed lines
Deallocation

- Released objects are not made immediately available → pushed in a dedicated vector ($m_{\text{objectLog}}$).
- Released objects are processed if $m_{\text{objectLog}}$ reaches its maximal capacity (512).
- Chunks/Pages/Lines are refcounted.
- Chunks/Pages/Lines are released if $\text{refCount} == 0$. 
Outline

1 Introduction & Motivation
2 Attack Surface
3 The bug
4 The FastMalloc Allocator
5 Exploitation
6 Conclusion & Future work
Exploiting UAF (1/3)

Shaping the heap

1. Allocate \( \frac{N}{2} \) of objects \( O \)
   - \( \text{sizeof}(O) = \text{sizeof(ValidationMessage)} \)

2. Instanciate a ValidationMessage Obj

3. Allocate \( \frac{N}{2} \) of objects \( O \)

```
Obj. O
Obj. O
Obj. O
Obj. O
ValidationMessage
Obj. O
Obj. O
Obj. O
Obj. O
More Obj. O
...
More Obj. O
...
```

Line
Exploiting UAF (2/3)

Shaping the heap

1. Delete few objects $O$ around $ValidationMessage$
2. Destroy $ValidationMessage$ object
   - Line released $\rightarrow$ Page cached
Exploiting UAF (3/3)

**Shaping the heap**

- Spray few objects $T$ to get back `ValidationMessage`:
- E.g. spray with `ArrayBuffer(ValidationMessageSize)`
Initial memory leak

**Memory leak**

- *m_messageBody, m_messageHeading & m_timer* instantiated after obj. reuse
- *m_timer* is “fastMalloc’ed”
  - → Guess the address of objects allocated on the same page

- m_element
- m_message
- m_messageHeading
- m_timer
- m_bubble
- ValidationMessage
Arbitrary decrement primitive

**Exploitation**

- Corrupt the \textit{m\_messageHeading} pointer
- Target: obj with length and data field
- Confuse some obj length field with \textit{m\_messageHeading} refcount
- Misaligned write on length field $\rightarrow$ Enlarge size of data buffer
- $\rightarrow$ Relative read/[write] primitive.
Exploitation strategy

- ASLR Bypass
- Obj. Reuse
- Relative Read Primitive
- Relative R/W Primitive
- Arbitrary R/W Primitive
- Code Execution
Relative read primitive (1/2)

**Goal**

- Leak the address of JSC allocated obj. (*JSArrayBufferView*)

**How**

1. Spray heap with multiple *StringImpl*
   - Obj:
     - Before/After *Timer* allocation leak
     - `sizeof(Timer) = sizeof(StringImpl)`

2. Use arbitrary decrement on *StringImpl* length's field

3. → read beyond *data* frontier in fastMalloc heap
Relative read primitive (2/2)

Leaking JSArrayBufferView pointers (1/2)

- DOM objects and JS objects use two different allocators
  - We can't access JSObject using our relative read
- The JS builtins `Object.defineProperty` allocate two objects that store JSObject references
  - `Vector` and `MarkedArgumentBuffer` are our target

```c
static JSValue defineProperties(ExecState* exec, JSObject* object, JSObject* properties) {

    Vector<PropertyDescriptor> descriptors;
    MarkedArgumentBuffer markBuffer;

    /* ... */
    JSValue prop = properties->get(exec, propertyNames[i]);
    /* ... */
    PropertyDescriptor descriptor;
    toPropertyDescriptor(exec, prop, descriptor);
    /* ... */
    descriptors.append(descriptor); // [1] store JSValue reference on fastMalloc
    /* ... */
    markBuffer.append(descriptor.value()); // [2] store one more JSValue reference on fastMalloc
}
```
Relative read primitive (2/2)

Leaking JSArrayBufferView pointers (2/2)

1. Allocate multiple JSArrayBufferView
2. Get reference on fastMalloc heap using `Object.defineProperties`
   - Both target objects are freed at the end of the builtin
   - Must not re-use these allocations otherwise we lose our references
3. Use relative read to find these references
   - We want a JSArrayBufferView that is allocated after our relative read object to read its content

1. Thanks @qwertyoruiopz for the Object.defineProperties technique
Relative read/write primitive

How

1. Run the exploit again
2. Use arbitrary decrement on **leaked** `JSArrayBufferView` address
3. Enlarge size of backing buffer
4. → read/write primitive

```
  m_butterfly
  m_mode  m_length
  m_vector
  m_structure
```
Arbitrary read/write primitive

How

- Relative R/W primitive through JSArrayBufferView 1
  - → corrupt JSArrayBufferView 2’s vector
- Arbitrary R/W primitive through JSArrayBufferView 2
Code execution

How

- We can’t allocate RWX memory page on PS4
- We can control RIP
  - We have a leak of a `HTMLElement` instance
  - Overwrite one vtable ptr of a `HTMLElement`
  - Call the JS method that will trigger the overwritten pointer
- We can do code-reuse to implement the next stage
  - The old previous PS4 jailbreak used this method
Demo

2. image credit: TheRegisti
Outline

1. Introduction & Motivation
2. Attack Surface
3. The bug
4. The FastMalloc Allocator
5. Exploitation
6. Conclusion & Future work
## Conclusion & Future work

### Conclusion
- Working WebKit exploit on 6.xx firmwares
- Exploit available on Github [https://github.com/Synacktiv](https://github.com/Synacktiv)

### Exploit stability
- Not really stable
- Take ~11 sec to gain arbitrary R/W

### Improvements
- The exploit reliability could be improved
- The ASLR bruteforce could be more deterministic
  - Our spray mixed fastMalloc and IsoHeap pages
  - It happens that we guess the address of the wrong virtual page
- We could find a better exploitation path that avoid triggering two times the vulnerability
What about 7.xx firmwares (1/2)

**Problem**

- ASLR bypass not working on 7.xx firmwares
- Cannot survive to crash during obj. reuse:
  - Requires prior knowledge on memory mapping
What about 7.xx firmwares (2/2)

Solution

- Bruteforce ASLR
  - Guess address of sprayed HTMLElement objs.
- Plug a Raspberry Pi (detected as a keyboard)
- Hit *Enter* keystroke at periodical time (5s)
  - Automatically reload exploit after a crash
- No results so far :-( 
Acknowledgements

Thanks to

- Synacktiv
  - For letting us do the research
- Our colleagues
  - For all the help while developing the exploit
- BlackHat
  - For the great event
- You
  - For your attention!
QUESTIONS?