Vous avez obtenu un trophée - PS4 jailbreaké

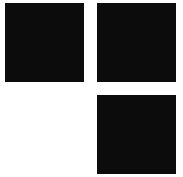
Exploitation d'une vulnérabilité WebKit sur la PlayStation 4

2 Juin 2021

Synacktiv

Quentin Meffre Mehdi Talbi

Table des matières



Qui sommes-nous ?

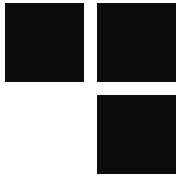
Mehdi Talbi
@abu_y0ussef

- Chercheur en sécurité @Synacktiv
- Recherche académique (dans une vie antérieure)
- Recherche de vulnérabilités & exploitation

Quentin Meffre
@0xdagger

- Chercheur en sécurité @Synacktiv
- Recherche de vulnérabilités & exploitation

Introduction



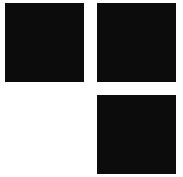
Motivation

- Communauté active sur le hacking de consoles de jeux
- … Mais très peu d'exploits publics
- À quel point est difficile l'exploitation de vulnérabilités sur la PS4 ?

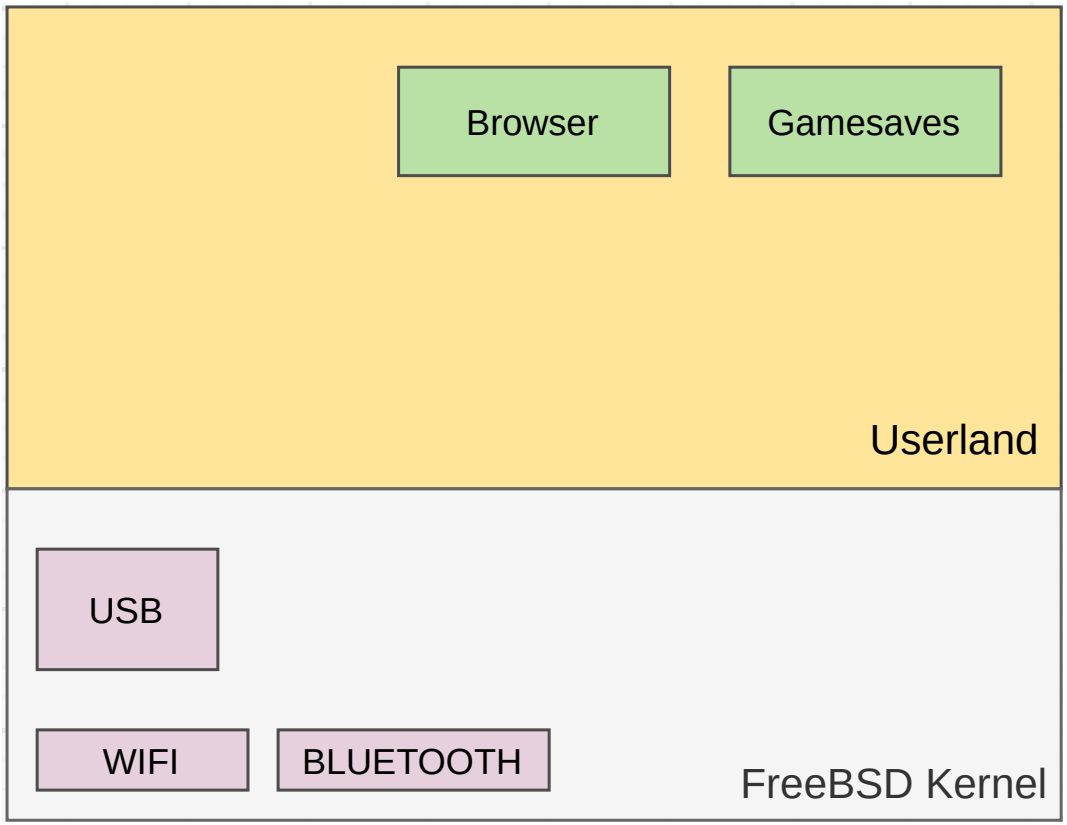
Objectif

- Exploitation pas-à-pas d'une vulnérabilité 0-day dans WebKit sur la PS4

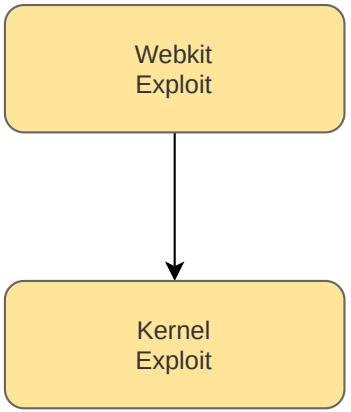
Table des matières



Surface d'attaque



Chaine d'exploitation



Attaque du navigateur

- Navigateur basé sur WebKit
- Sandbox
- Pas de JIT
- Absence de certaines protections
  - Gigacage
  - Randomisation des `StructureID`
- ASLR : partiel ? Faible ?
- Absence de capacité de débogage

Exploits WebKit

CVE-2018-4386

- Identifié par Lokihardt (de Projet Zero)
- Exploité par @Fire30_
- Dernier exploit public en date (“bad-hoist”)
- Primitive de lecture/écriture arbitraire
- Firmwares 6.00-6.72

CVE-2018-4441

- Également identifié par Lokihardt
- Exploité par @SpecterDev
- Primitive de lecture/écriture arbitraire
- Firmwares 6.00-6.20

Autres exploits ...

- Pour les firmwares plus anciens (< 6.xx)
- Exploits de @qwertyoruiopz, @SpecterDev, @CTurt, ...

Exploits Kernel

CVE-2020-9892

- Identifié par @theflow0
- Double Free dans la pile IPv6
- Firmwares <= 7.55

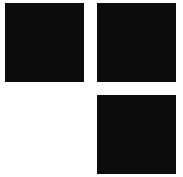
CVE-2020-7457

- Identifié par @theflow0
- Primitives de lecture/écriture dans le kernel
- Atteignable depuis la sandbox WebKit
- Firmwares 7.02 & 6.xx
- Combiné avec l'exploit bad-hoist

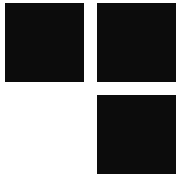
Vulnérabilités dans BPF

- Identifiées & exploitées par @qwertyoruiopz
- Firmwares <= 5.07.
- Excellent write-up par @SpecterDev

Table des matières



Allocateurs WebKit



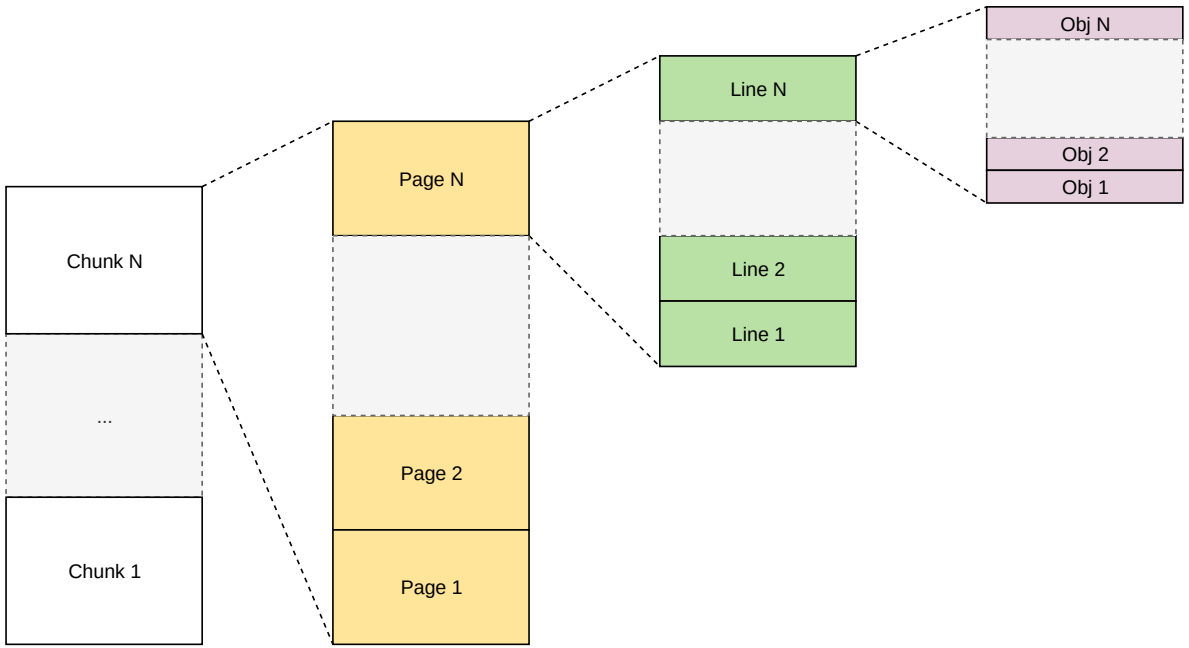
Plusieurs allocateurs

- **FastMalloc** : allocateur standard
- **IsoHeap** : utilisé par le moteur DOM
  - Tri des allocations en fonction du type
- **GC** (Garbage Collector) : utilisé par le moteur JavaScript
- **IsoSubspace** : utilisé par le moteur JavaScript
  - Trie des allocations en fonction de leur taille
- **GigaCage** : protections contre les lectures/écritures OOB sur certains objets

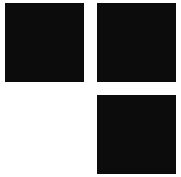
Allocateur standard

Présentation

- **Heap** constitué de **chunks**
- **Chunk** divisé en **pages** (4 kB)
- **Page** subdivisée en **lignes** (256 octets)
- **Ligne** : dessert des allocations d'objets de mêmes tailles



Allocation (1/2)



Allocations via le chemin rapide (fast path)

- Allocateur de type “bump-pointer”

```
--m_remaining;
char* result = m_ptr;
m_ptr += m_size;
return result;
```

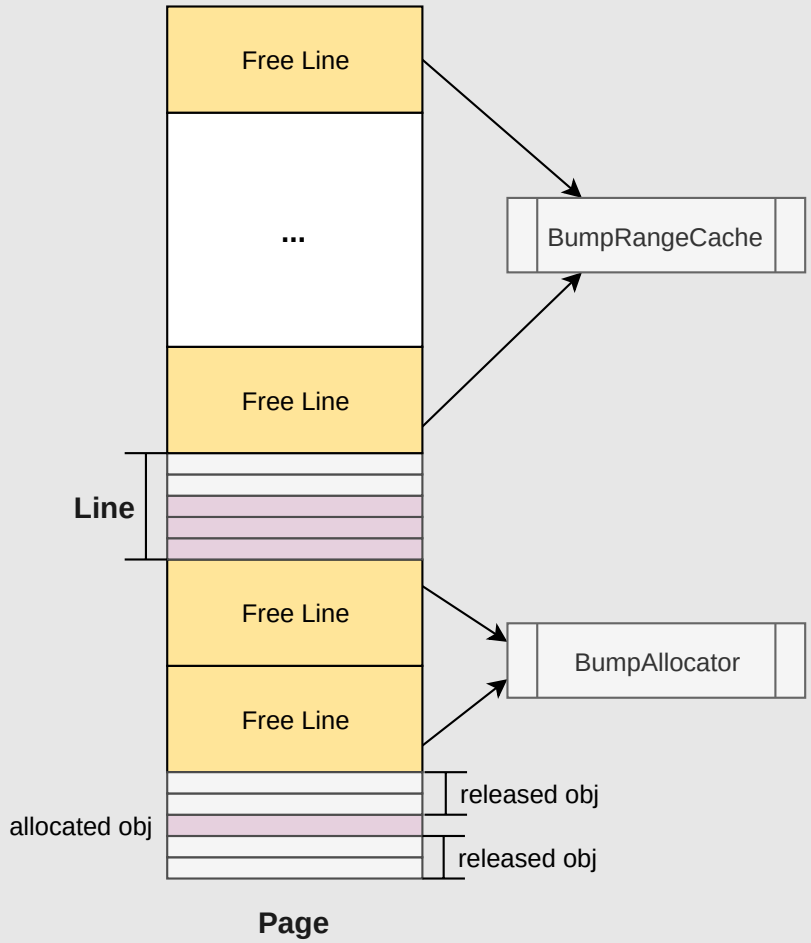
Allocations via le chemin lent (slow path)

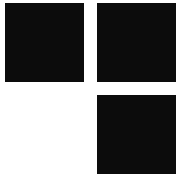
- Réapprovisionnement de l’allocateur
  1. À partir du cache **BumpRangeCache** (fast path)
  2. À partir de l’allocation d’une nouvelle page (slow path)

Allocation (2/2)

Réapprovisionnement de l'allocateur

- Allocation d'une nouvelle page
  1. Depuis le cache `CacheLine`
  2. Depuis le dernier chunk alloué
- Alimenter l'allocateur avec les lignes libres & contiguës
- Alimenter le cache avec les lignes libres restantes

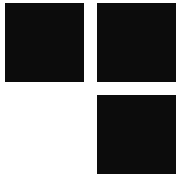


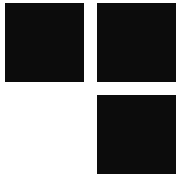


Déallocation

- Les objets libérés ne sont pas immédiatement mis à disposition → placés dans un vecteur dédié (`m_objectLog`).
- Traitement des objets du vecteur `m_objectLog` :
  - Lorsqu'il atteint sa capacité maximale (512)
  - Lors du réapprovisionnement de l'allocateur
- Chunks / Pages / Lignes disposent d'un compteur de référence (refcount)
- Chunks / Pages / Lignes libérés **SI** `refcount == 0`

Table des matières





Vulnérabilité

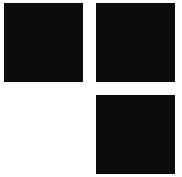
- Présente dans le moteur DOM de WebKit
- Trouvée via fuzzing
- Impacte tous les firmwares < 8.00 de la PS4
  - … et de la PS Vita aussi
- Remontée à Sony via leur programme de Bug Bounty (2500$)
- Corrigée le 11 Sept. 2020

Code vulnérable

- Vulnérabilité présente dans la méthode `WebCore::ValidationMessage::buildBubbleTree`
- La fonction `updateLayout` met à jour la disposition de la page :
  - Exécution de callbacks JS enregistrées par l'utilisateur
  - Possibilité de destruction de l'instance `ValidationMessage` → UAF
- Erreur dans la définition d'un `WeakPtr`

```
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());

    /* ... */
}
```

Le bon et le mauvais correctif (1/2)



Protect frames during style and layout changes Browse files

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.

Le mauvais correctif

```
void ValidationMessage::buildBubbleTree()
{
    /* ... */
+
+   auto weakElement = makeWeakPtr(*m_element);
+
    document.updateLayout();

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

    adjustBubblePosition(m_element->renderer()->absoluteBoundingBoxRect(), m_bubble.get());
    /* ... */
}
```

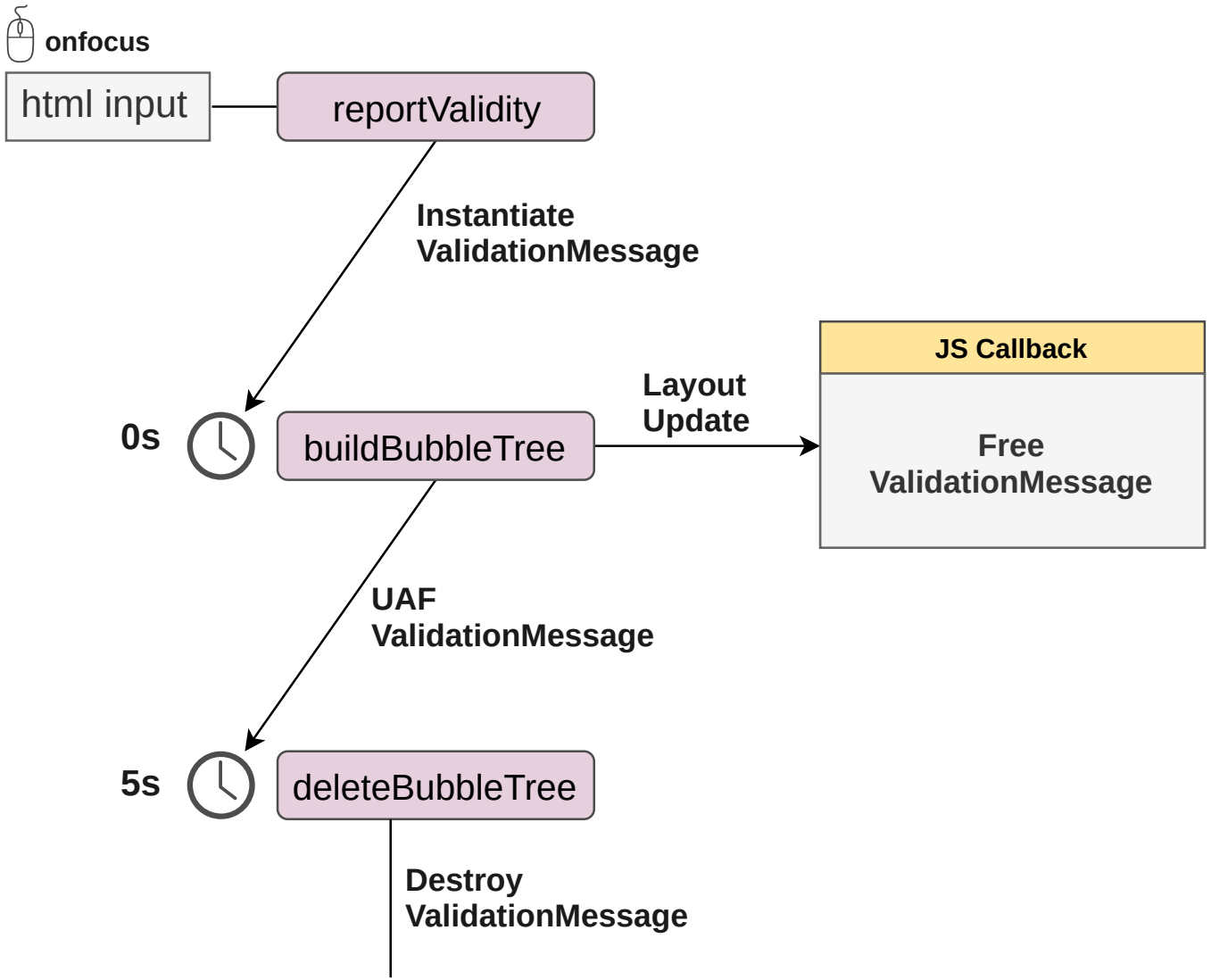
Le bon et le mauvais correctif (2/2)

Le bon correctif

- Ne pas permettre des mises à jour de la disposition de la page dans `buildBubbleTree`

```
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();
+   });
}
```

Chemin vulnérable



Tentatives de déclenchement du bug (1/2)

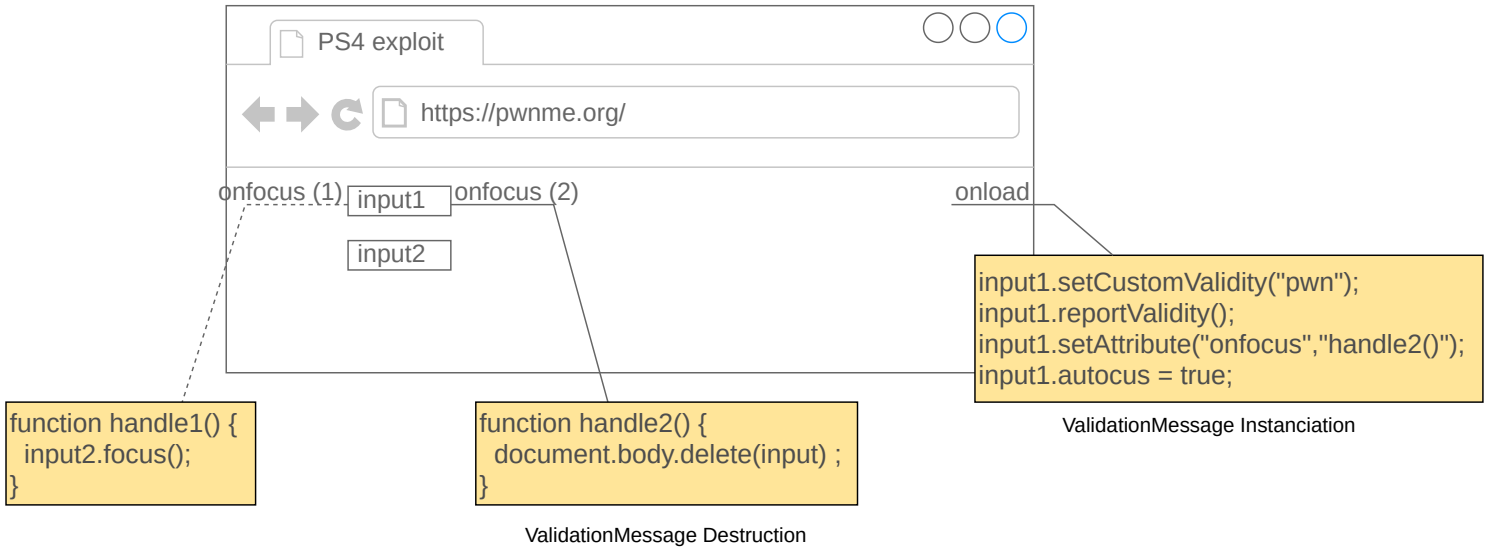
Tentative initiale

1. Enregistrer un événement JS (e.g. onfocus) sur un champ de texte HTML
2. Instancier l'objet `ValidationMessage`
   - → Déclenchement d'un minuteur pour exécuter `buildBubbleTree`
   - → Exécution de la callback JS
3. Détruire l'instance `ValidationMessage` dans la callback JS
4. Pas de crashs !!
   - `reportValidity` positionne le focus sur l'élément HTML cible
   - Exécution prématurée de la callbask JS

Tentatives de déclenchement du bug (2/2)

Solution

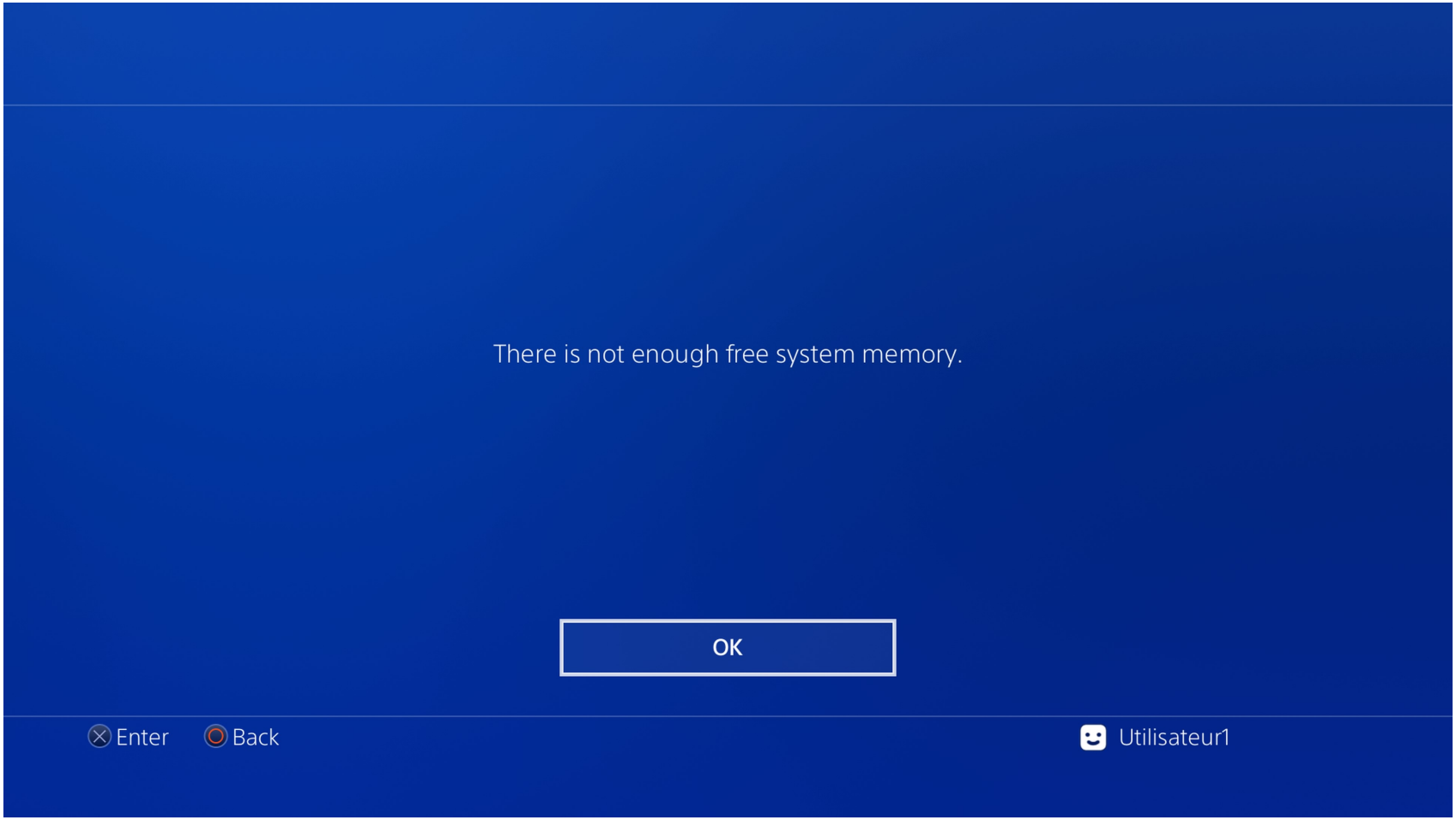
1. Enregistrer un événement JS **handler1** sur un champ HTML **input1**
2. Instancier l'objet **ValidationMessage** (sur **input1**)
   - Le focus est positionné sur **input1** → **handler1** est exécuté
   - **handler1** positionne le focus ailleurs (**input2**)
3. Définir un nouveau handler **handler2** sur **input1**
4. Détruire l'instance **ValidationMessage** dans le **handler2**



ValidationMessage Instanciation

ValidationMessage Destruction

Crash !


There is not enough free system memory.
OK
Enter
Back
Utilisateur1

Débogage

Problème

- Absence de capacité de débogage

Solution 1 - Installation d'un environnement similaire

- Installer un FreeBSD
- Compiler les sources WebKit récupérées depuis doc.dl.playstation.net
- → Utile **Mais** un exploit fonctionnel sur notre environnement n'implique pas un portage assuré sur la PS4

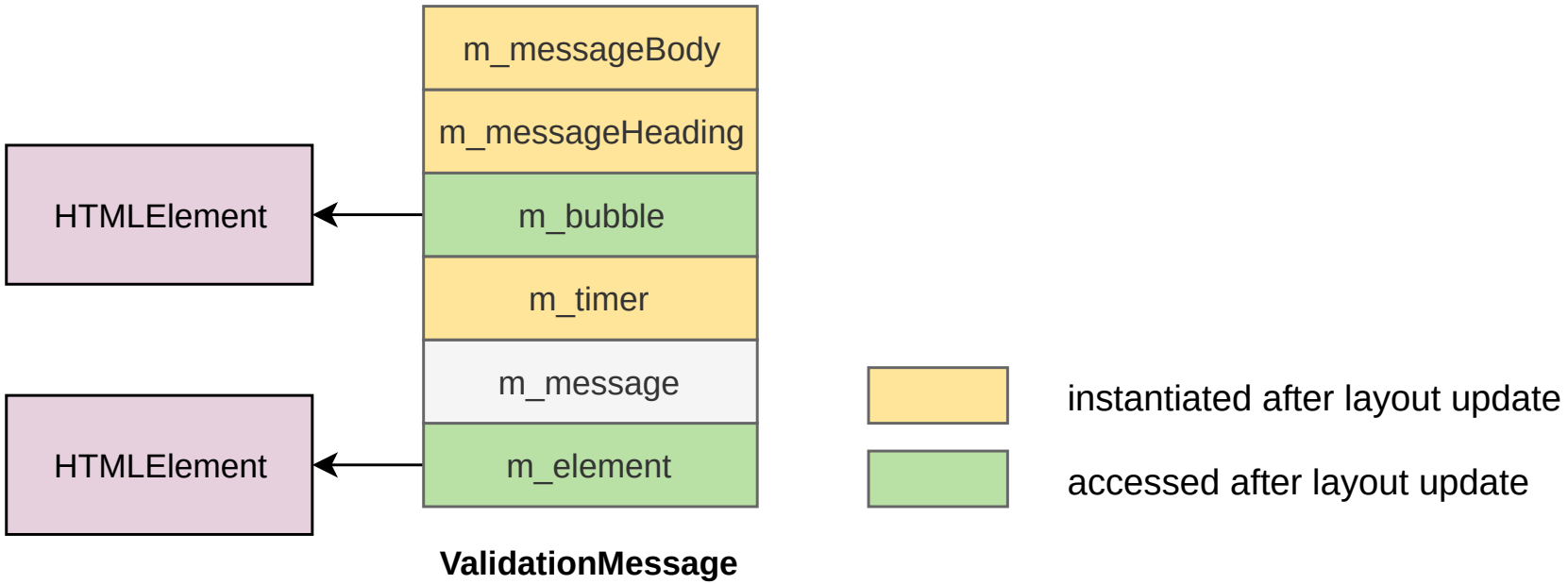
Solution 2 - Débogage d'une 0-day avec une 1-day

- Utilier l'exploit "bad-hoist"
  - (+) Primitives de lecture/écriture
  - (+) Primitives addrof/fakeobj
  - (-) Fonctionne uniquement sur les firmwares 6.xx
  - (-) Peut influencer notre stratégie de shaping de la mémoire
  - (-) Peu fiable

Anatomie de l'objet vulnérable

Objet ValidationMessage

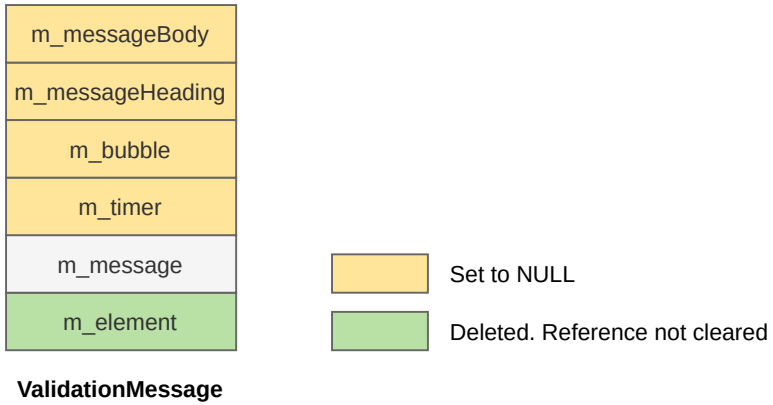
- Instancié par `reportValidity()` (via `fastMalloc()`)
- Accédé par `buildBubbleTree()`
- Détruit par `deleteBubbleTree()`



Survivre au crash initial (1/3)

Situation après destruction de l'objet vulnérable

- Déréférencement d'un pointeur NULL (`m_bubble`) → crash du navigateur
- Situation : **Inconfortable**



Exploitabilité

- Nécessite
  1. Un leak Mémoire, ou bien
  2. … un bypass de l'ASLR

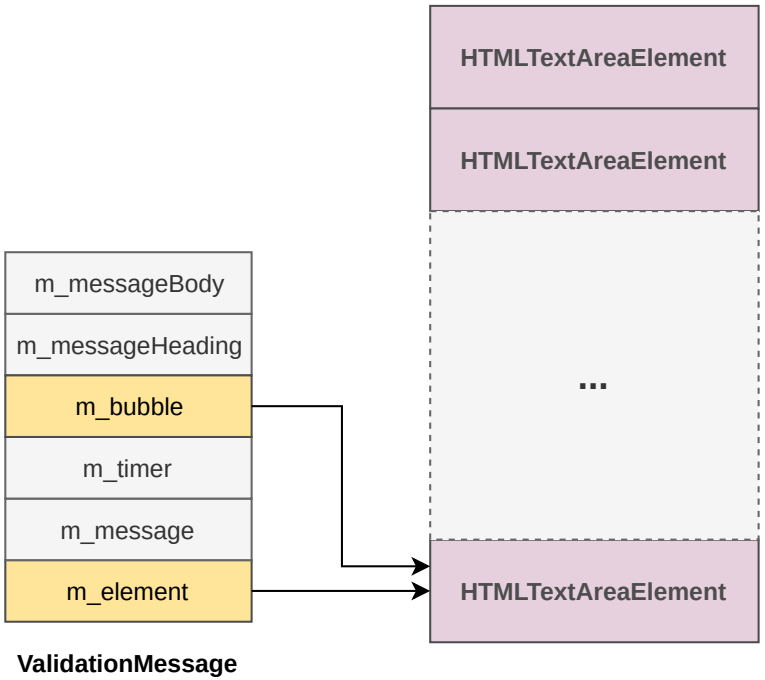
Survivre au crash initial (2/3)

Bypass de l'ASLR

- Heap Spray → **adresses prédictibles**
- Nécessite une connaissance préalable du mapping mémoire

Survivre au crash

- Allocation de plusieurs milliers objets `HTMLElement` (e.g. `HTMLTextAreaElement`)
- UAF de l'objet `ValidationMessage`



Survivre au crash initial (3/3)

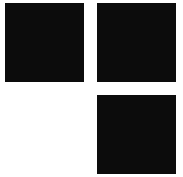
Épilogue de la fonction vulnérable

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

Ré-évaluation de la situation

- `ValidationMessage` de nouveau détruit
- Surcharge de l'opérateur "**=**"
  - Affectation de `nullptr` sur un objet "refcounté" → décrément du compteur de référence
- Décrément d'un compteur de références sur plusieurs objets *contrôlés*
- **UAF** → Décrément arbitraire
- **Exploitable**

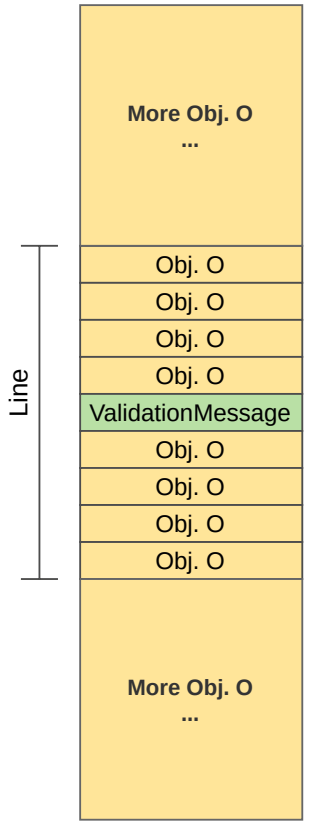
Table des matières



Réutiliser l'objet cible (1/3)

Contrôler le tas

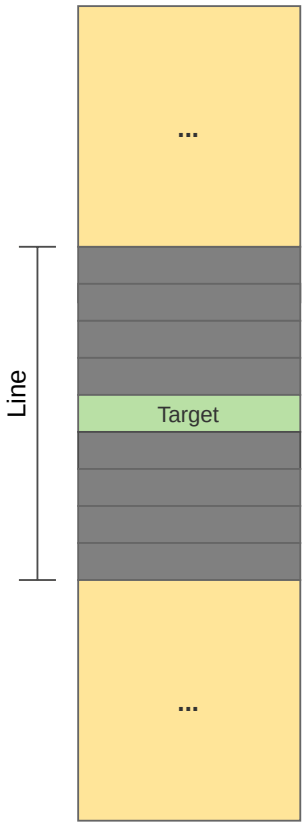
1. Allouer N/2 d'objets dit O
   - `sizeof(O) == sizeof(ValidationMessage)`
2. Instancier un objet `ValidationMessage`
3. Allouer N/2 d'objets dit O



Réutiliser l'objet cible (2/3)

Contrôler le tas

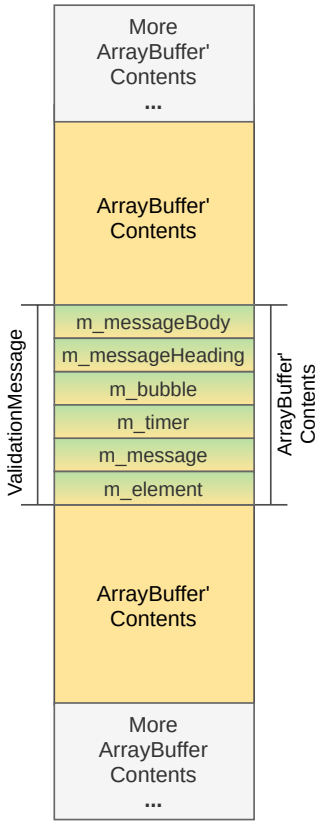
1. Libérer les objets dit O se trouvant autour de l'objet alloué `ValidationMessage`
2. Libérer l'objet `ValidationMessage`
   - La ligne associée est libérée
   - La page associée peut être ajoutée dans le cache



Réutiliser l'objet cible (3/3)

Contrôler le tas

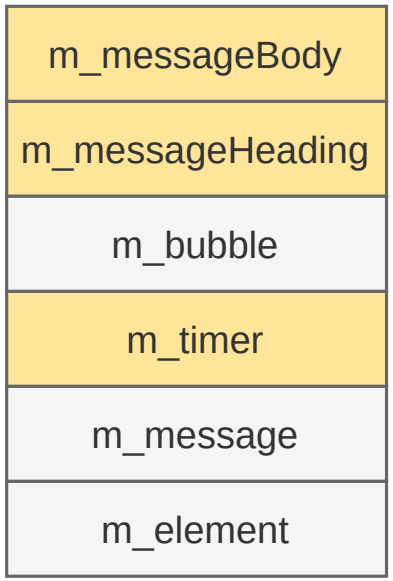
- Allouer de nouveaux objets pour réutiliser l'objet `ValidationMessage` libéré
- La taille de l'objet alloué doit être égale à celle de l'objet `ValidationMessage`
- Le tampon de données d'un objet `ArrayBuffer` est un bon candidat



Fuite de données initiale

Fuite de données

- `m_messageBody`, `m_messageHeading` et `m_timer` sont alloués après avoir réutilisé l'objet `ValidationMessage`
- `m_timer` est alloué par FastMalloc
  - Permet d'inférer l'adresse d'objets alloués dans ce tas

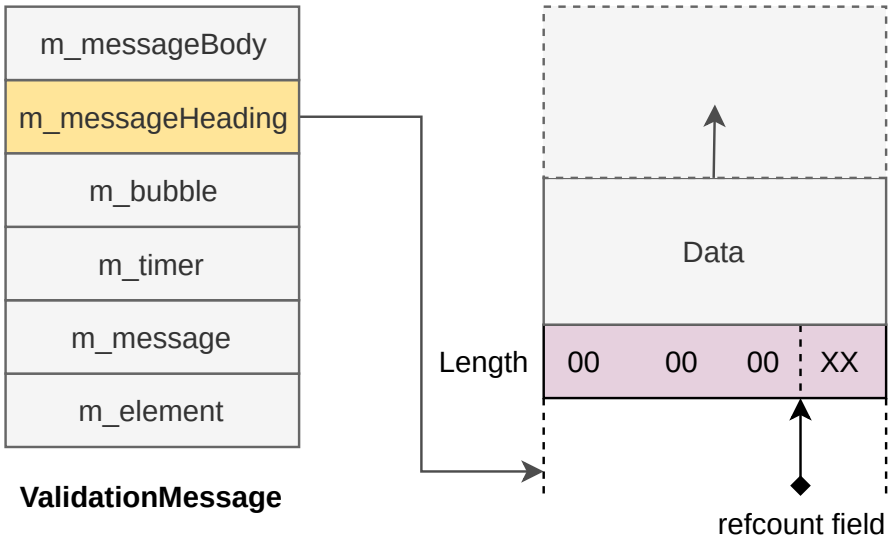


ValidationMessage

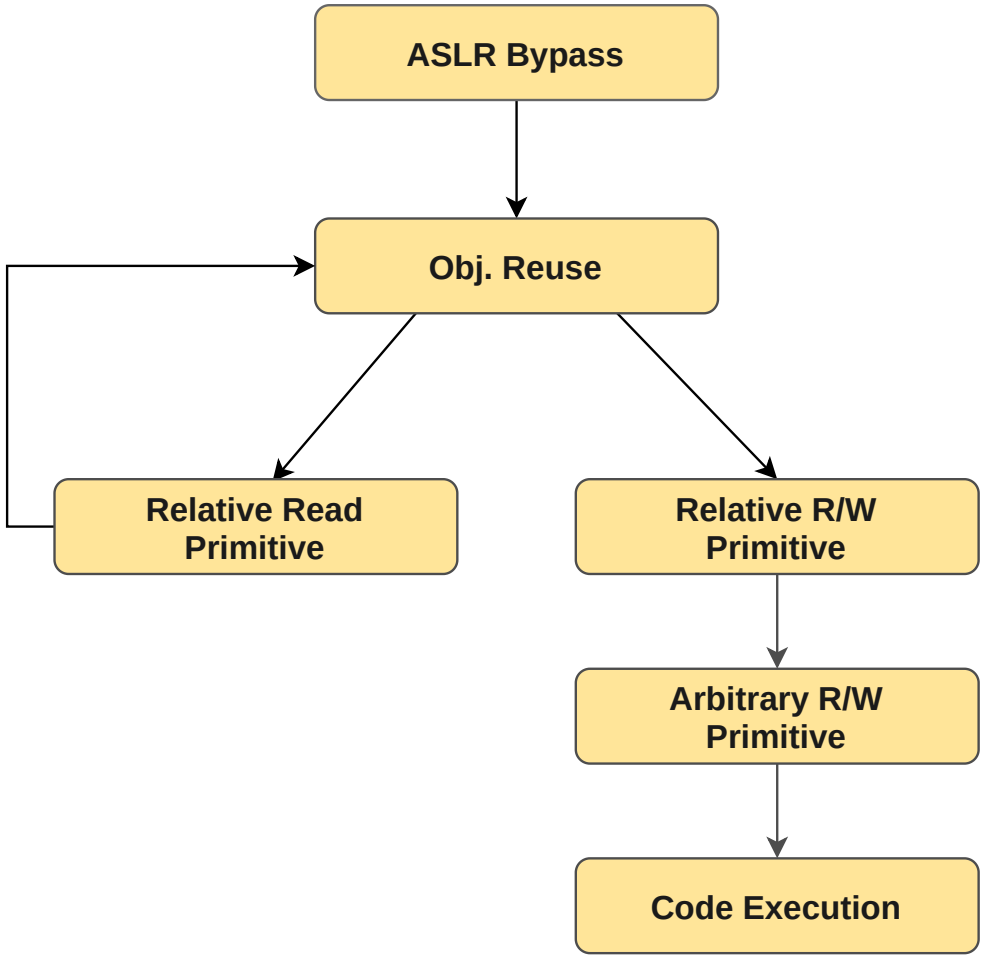
Primitive de décrément arbitraire

Exploitation

- Corrompre le pointeur `m_messageHeading`
- Cibler les attributs `length` des objets permettant de manipuler des données
- Confondre la taille d'un objet avec le compteur de références de l'objet `m_messageHeading`
- Aligner l'objet visé de manière à élargir son attribut `length` et non le réduire
- L'objectif est d'obtenir une primitive de lecture et écriture relative



Stratégie d'exploitation



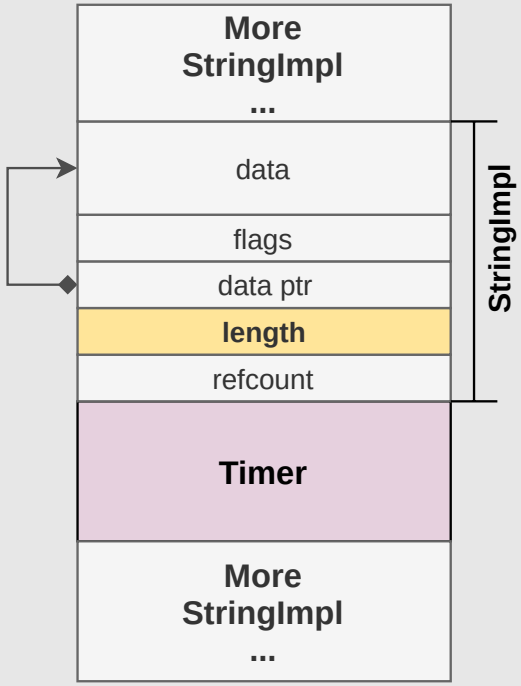
Primitive de lecture relative (1/3)

Objectif

- Obtenir l'adresse d'un objet `JSArrayBufferView` alloué par le Garbage Collector

Comment ?

1. Allouer beaucoup d'objets `StringImpl` sur le tas
   - Avant/après l'allocation de `Timer`
   - `sizeof(Timer) == sizeof(StringImpl)`
2. Utiliser le décrément arbitraire sur l'attribut `length` d'un objet `StringImpl`
3. Il est possible de lire au-delà des limites fixées par l'objet `StringImpl`



Primitive de lecture relative (2/3)

Fuiter l'adresse d'un objet JSArrayBufferView (1/2)

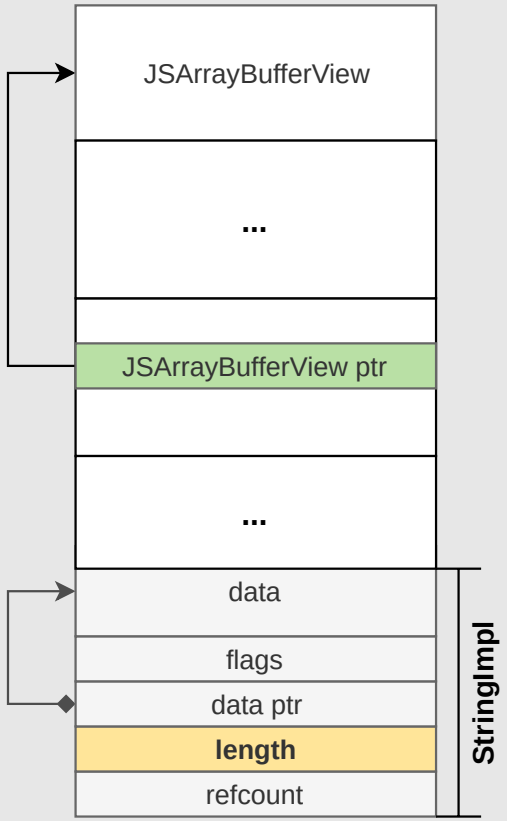
- Les objets du DOM et les objets JavaScript utilisent deux allocateurs différents
  - On ne peut pas atteindre un `JSObject` à partir de notre primitive de lecture relative
- La méthode JavaScript native `Object.defineProperties` alloue deux objets qui permettent de stocker des références vers des `JSObject`
  - Les objets `Vector` et `MarkedArgumentBuffer` sont nos cibles

```
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] Stocke une JSValue sur le tas de FastMalloc
    /* ... */
    markBuffer.append(descriptor.value()); // [2] Stocke une seconde fois une JSValue sur le tas de
           FastMalloc
}
```

Primitive de lecture relative (3/3)

Fuiter l'adresse d'un objet JSArrayBufferView (2/2) [1]

1. Allouer plusieurs objets `JSArrayBufferView`
2. Ajouter des références vers ces objets à l'aide de la méthode `Object.defineProperties`
   - Les deux objets stockant les références vers les `JSObject` sont libérés à la fin de `Object.defineProperties`
   - Il ne faut pas réutiliser ces allocations sous peine de perdre nos références
3. Utiliser la primitive de lecture relative pour chercher ces références
   - L'objet `JSArrayBufferView` doit être alloué après l'objet utilisé par la primitive de lecture relative afin de pouvoir lire son contenu

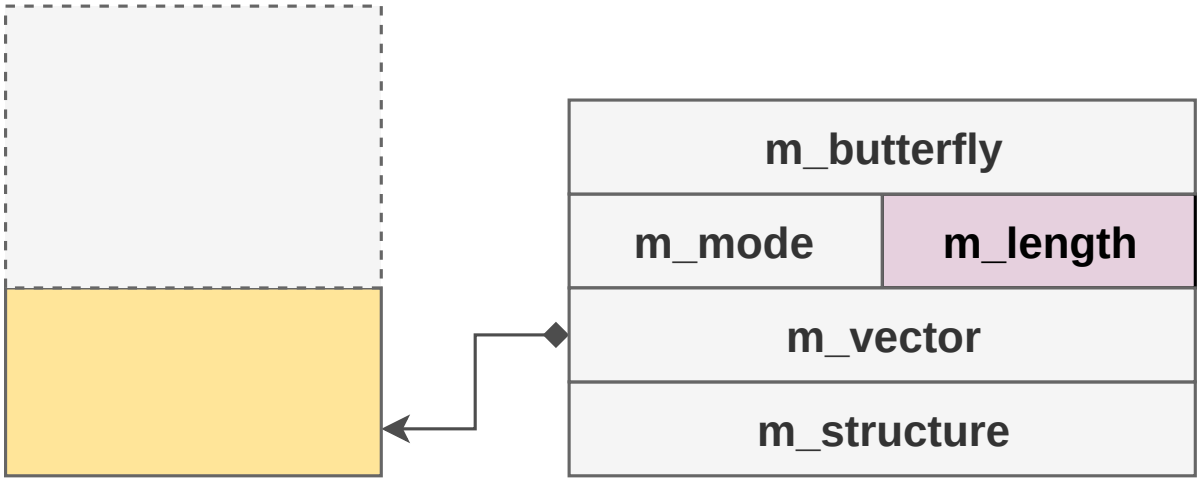


1. Merci @qwertyoruiopz pour la technique utilisant Object.defineProperties

Primitive de lecture/écriture relative

Comment ?

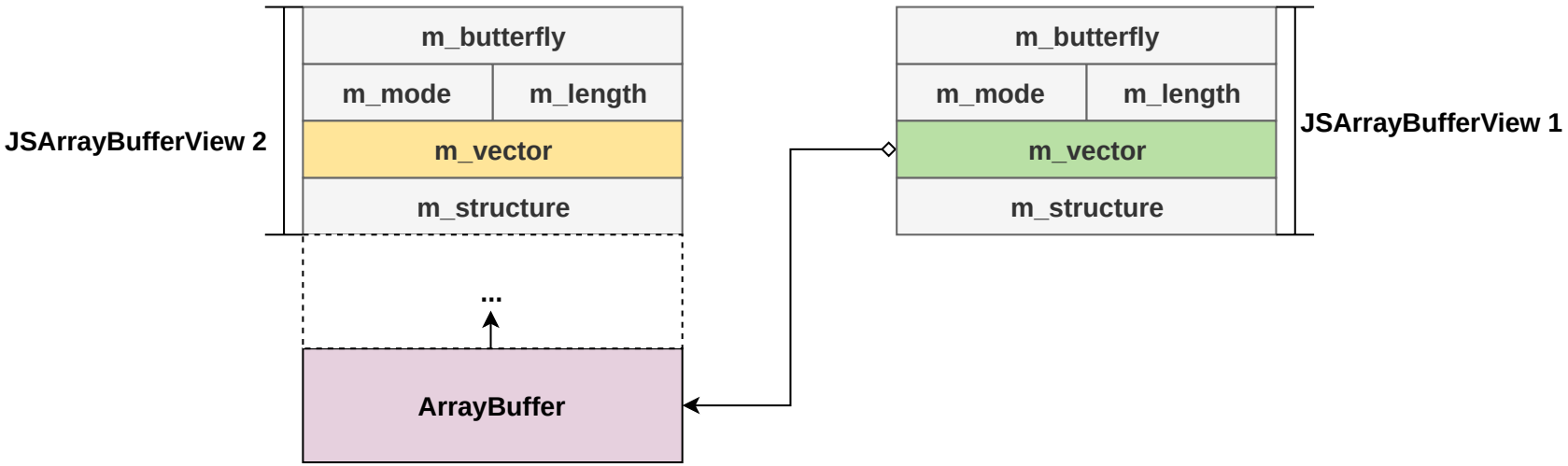
1. Utiliser à nouveau la primitive de décrément arbitraire
2. Corrompre la taille d'un `JSArrayBufferView` à l'aide de l'adresse obtenue précédemment
3. L'objet `JSArrayBufferView` corrompu permet de lire et écrire relativement en mémoire



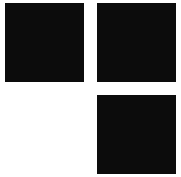
Primitive de lecture/écriture arbitraire

Comment ?

- Primitive de lecture et écriture relative depuis un premier objet `JSArrayBufferView`
  - Corrompre le pointeur du tampon de données d'un second `JSArrayBufferView`
- Primitive de lecture et écriture arbitraire à partir du second `JSArrayBufferView`



Exécution de code



Comment ?

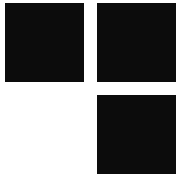
- La PS4 ne permet pas de mapper une page RWX dans le contexte du processus de rendu HTML
- Nous pouvons contrôler le registre d'instruction RIP
  - Nous avons l'adresse d'un objet `HTMLElement`
  - Corrompre la vtable d'un `HTMLElement`
  - Appeler la méthode JavaScript associée afin de faire un appel utilisant la vtable corrompue
- L'étape suivante peut-être implémentée à l'aide d'une chaine de ROP/JOP
  - Méthode couramment utilisée dans le jailbreak PS4

Démonstration

2

2. image credit : TheRegisti

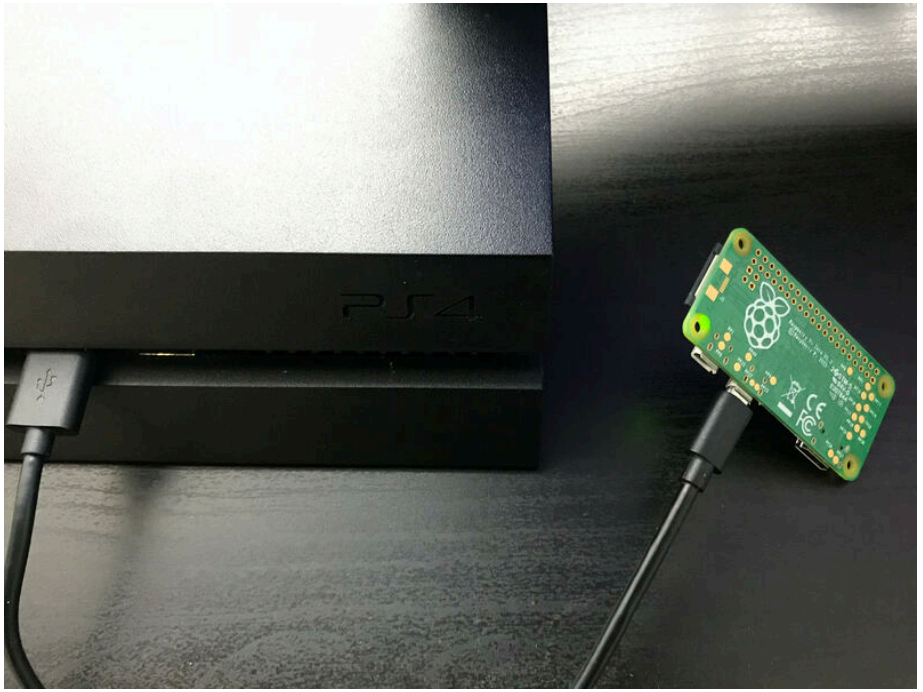
Porter l'exploit sur la version 7 (1/3)



Problématique

- L'adresse utilisée pour contourner l'ASLR sur la version 6 ne fonctionne pas sur la version 7 de la PS4
- Le navigateur crash lors de l'étape de la réutilisation de l'objet libéré `ValidationMessage`
  - Des connaissances préalables sur le mapping mémoire sont nécessaires

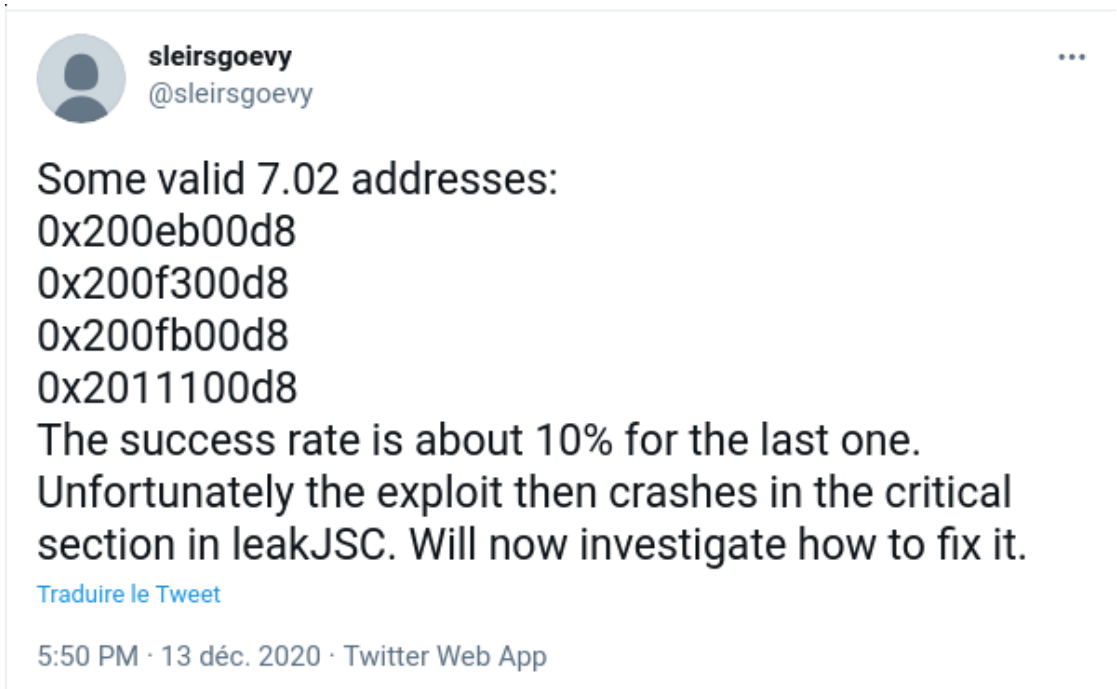
Porter l'exploit sur la version 7 (2/3)



Solution

- Bruteforcer l'ASLR
  - Deviner l'adresse de l'un des `HTMLElement` alloué
- Brancher un Raspberry Pi (détecté comme un clavier)
- Appuyer sur la touche "Entrée" à une fréquence régulière (5s)
  - Redémarrer le navigateur après un crash
- Pas de résultats :-(

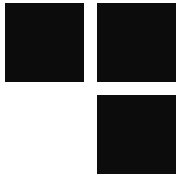
Porter l'exploit sur la version 7 (3/3)



... Quelques jours après la publication de l'exploit

- Publication d'adresses valides pour la version 7.02
- Adaptation de l'exploit (modifications mineures)
- Enchainement avec l'exploit kernel
- **→ Premier Jailbreak de la PS4 en version 7**

Table des matières



Conclusion

Conclusion

- Exploitation d'une vulnérabilité 0-day WebKit sur la PS4
  - L'exploit est disponible sur le github de Synacktiv
- Contribution au premier jailbreak sur le firmware 7.xx

Perspectives

AVEZ-VOUS DES QUESTIONS ?

MERCI DE VOTRE ATTENTION

SYNACKTIV