RÉPUBLIQUE
FRANÇAISE

Liberté
Égalité
Fraternité

Évolution de la sécurité du moteur Javascript V8

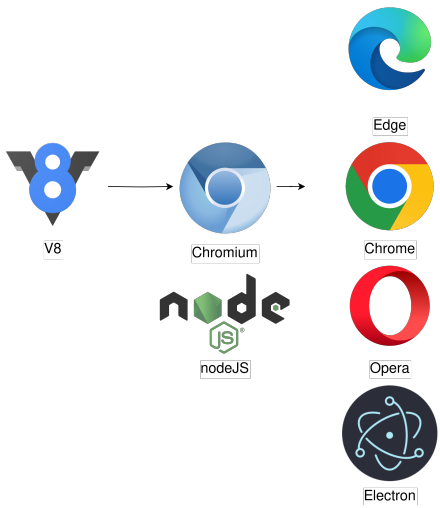
SSTIC 2024

François Jolivet

French National Cyber Security Agency

V8

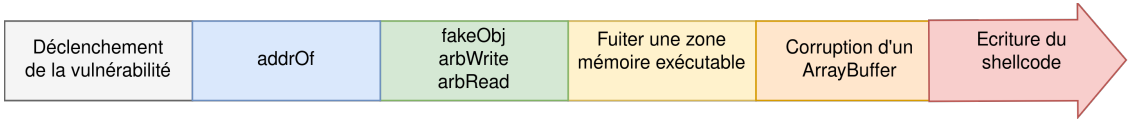
- Moteur JavaScript open-source
- Développé par le projet Chromium
- Utilisé dans de nombreux projets
- Importante exposition aux attaques



Code Javascript malveillant

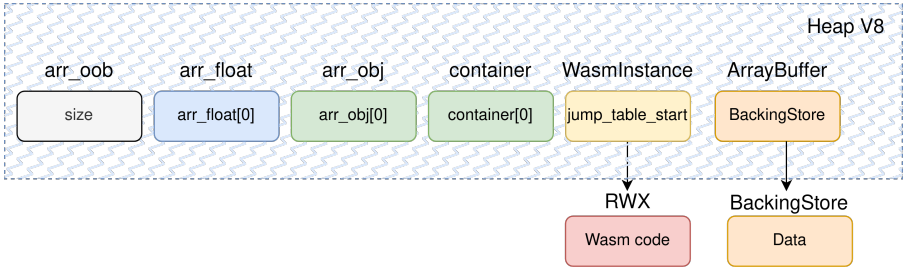
Une attaque contre le moteur *V8* implique

- Du code Javascript **malveillant**
- Le déclenchement d'une vulnérabilité
- La **corruption** successive d'objets Javascript sur le tas via l'utilisation de **primitives d'exploitation**
- Une exécution de code arbitraire à distance

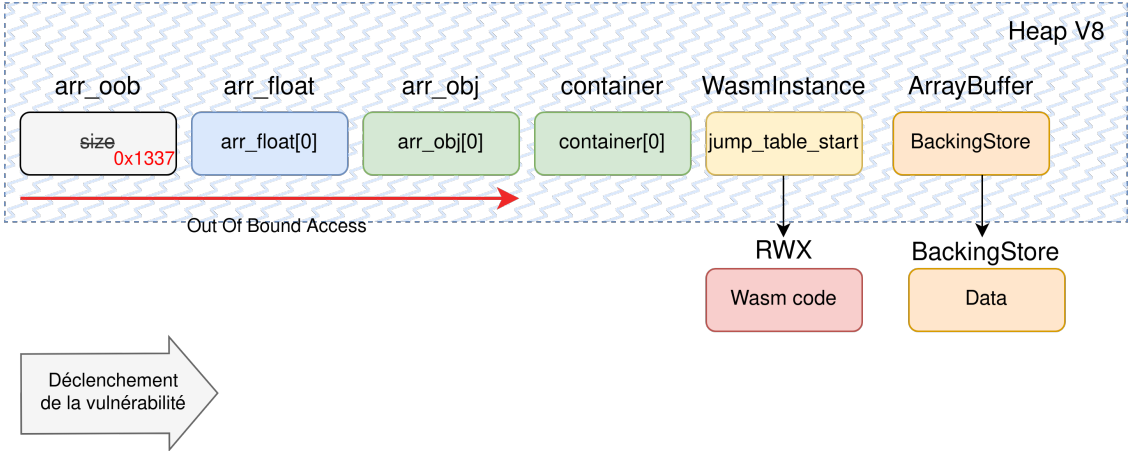


Objets sensibles

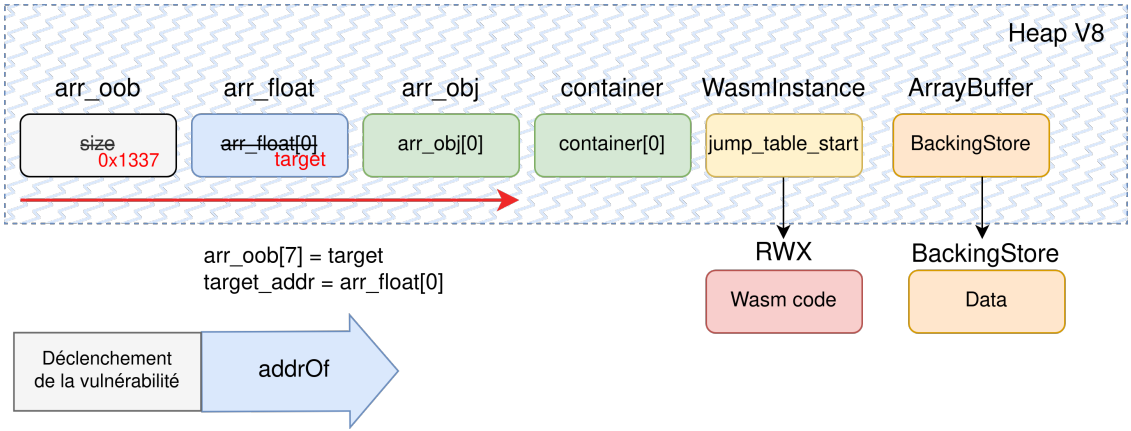
- **ArrayBuffer** : permet de manipuler des données binaires stockées en dehors du tas *V8* avec un accès via des vues (*TypedArray* ou *Dataview*). Les données sont référencées par le pointeur *BackingStore*
- **WasmInstance** : Structure utilisée lors de l'exécution du code *WebAssembly*.
  - La plage RWX utilisée pour le code compilé est référencée par le pointeur *jump_table_start*



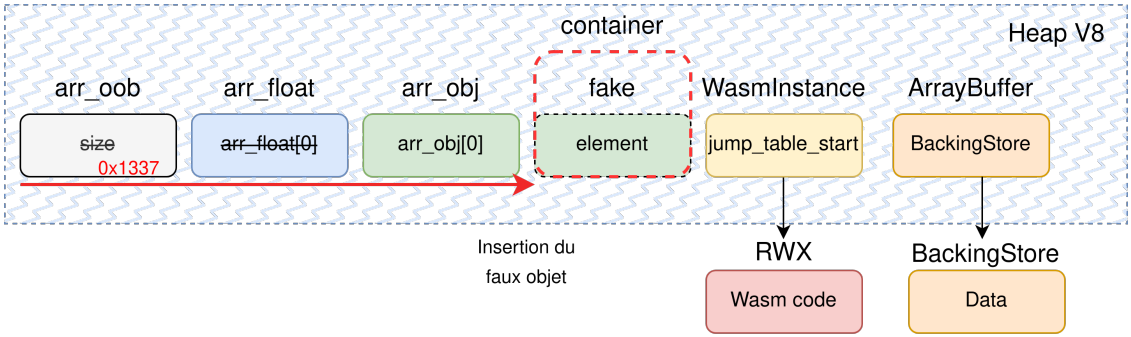
Corruption de la taille d'un tableau



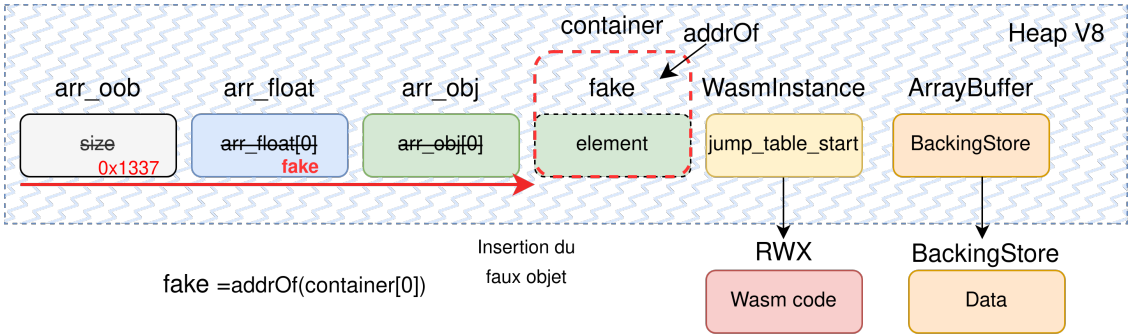
Construction de la primitive addrOf



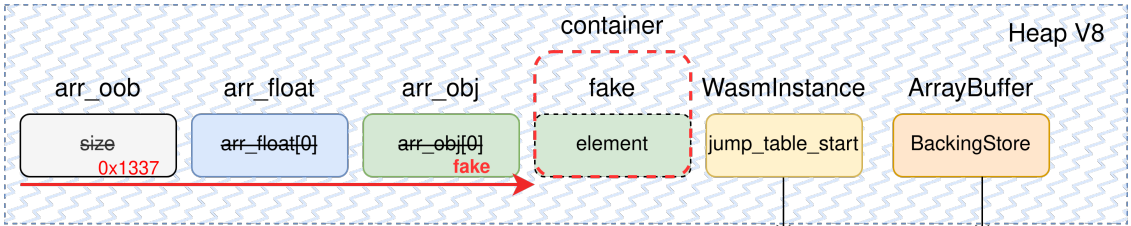
Construction de la primitive fakeObj



Construction de la primitive fakeObj



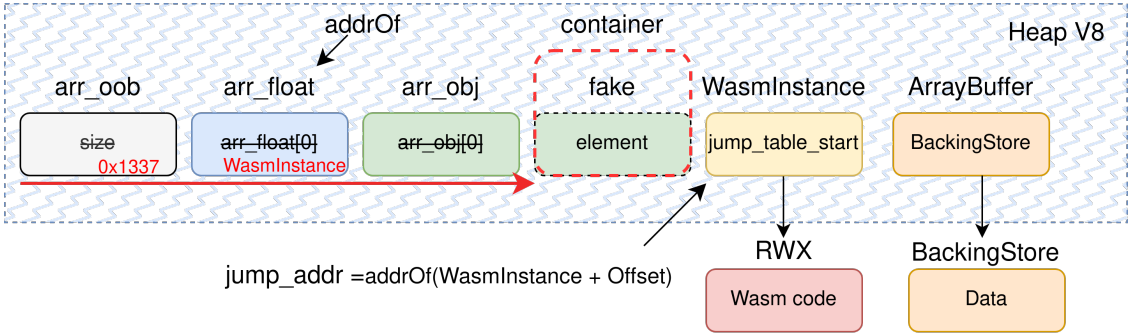
Construction de la primitive fakeObj

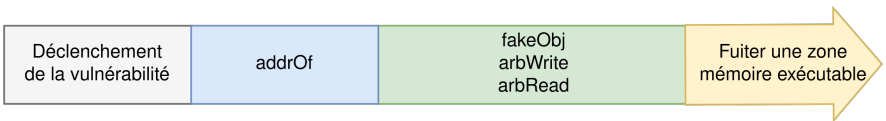


Insertion du faux objet

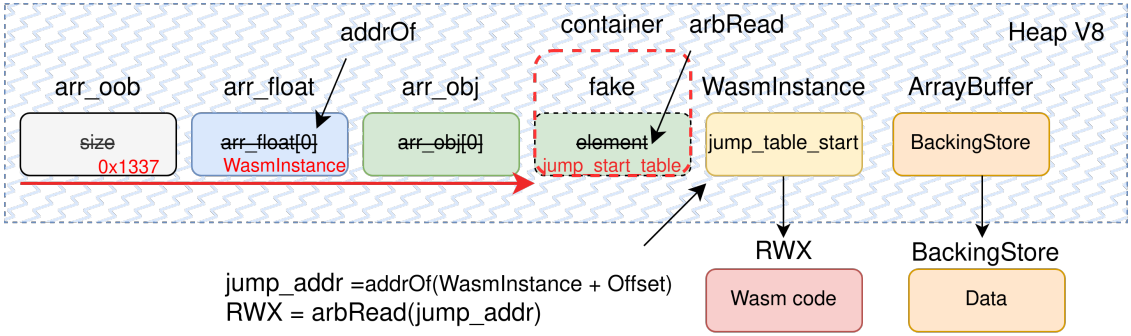
```
fake_addr =addrOf(container[0])
arr_oob[13] = fake_addr
fake = arr_obj[0]
```

Fuite d'une adresse RWX

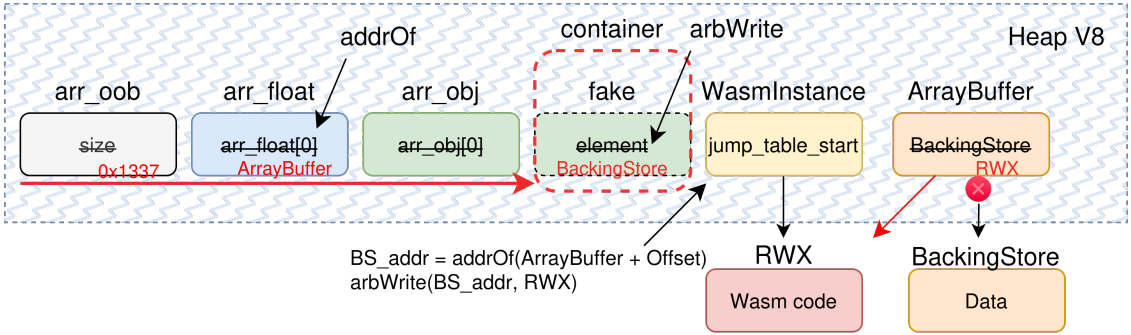




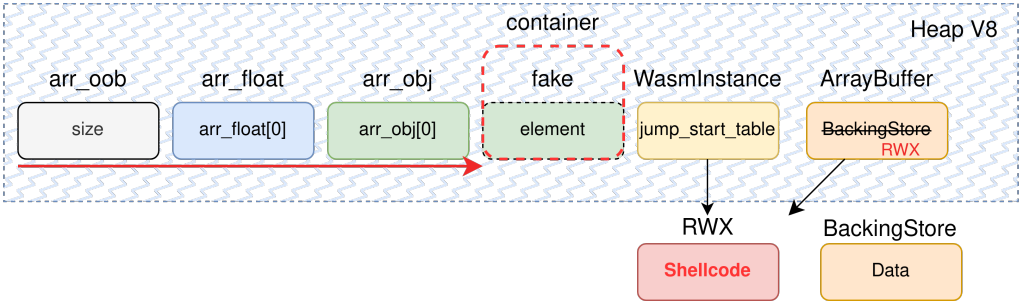
Fuite d'une adresse RWX

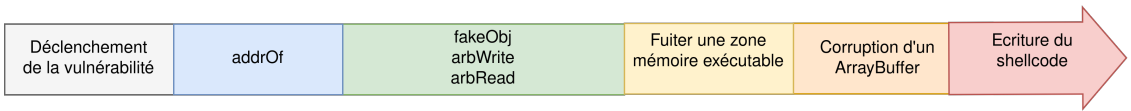


Corruption de l'ArrayBuffer

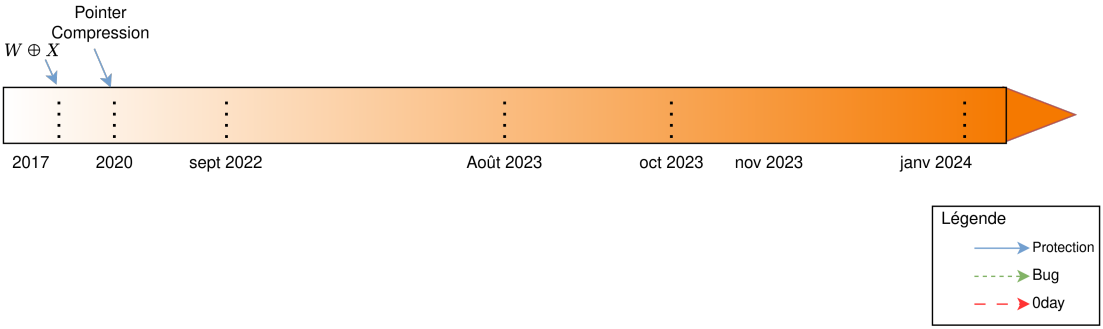


Shellcode

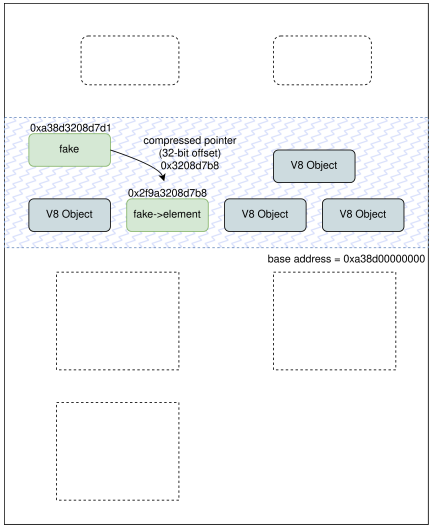

container
Heap V8
arr_oob
arr_float
arr_obj
fake
WasmInstance
ArrayBuffer
size
arr_float[0]
arr_obj[0]
element
jump_start_table
BackingStore
RWX
RWX
BackingStore
Shellcode
Data


Déclenchement de la vulnérabilité
addrOf
fakeObj
arbWrite
arbRead
Fuiter une zone mémoire exécutable
Corruption d'un ArrayBuffer
Ecriture du shellcode

Ligne de temps



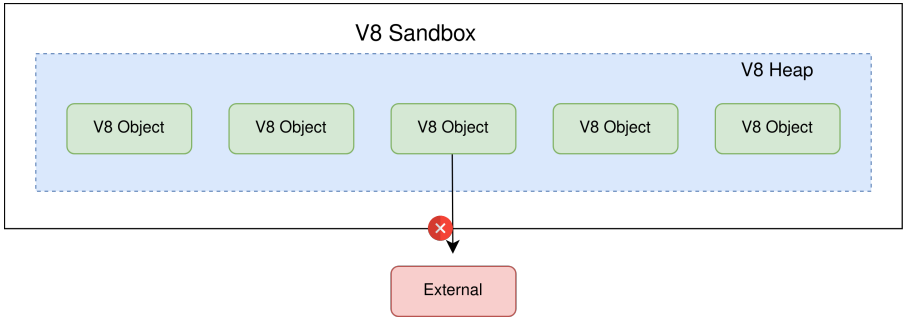
La compression de pointeur



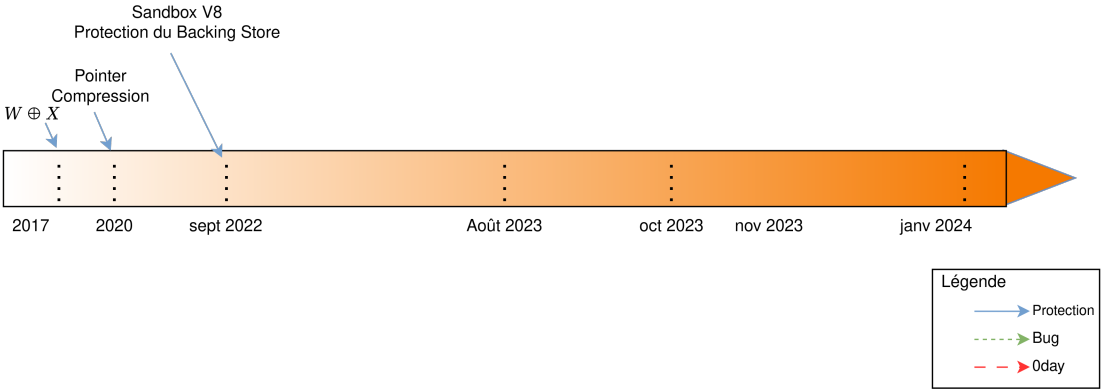
	Avant la compression de pointeur	Après la compression de pointeur
Fake		
address	0xa38d3208d7d1	0x3208d7d1
Fake->element		
element	0x2f9a3208d7b8	0x3208d7b8
WasmInstance		
jump_table_start	0x98406acd000	0x98406acd000
ArrayBuffer		
BackingStore	0x52339310909456	0x52339310909456

Sandbox V8

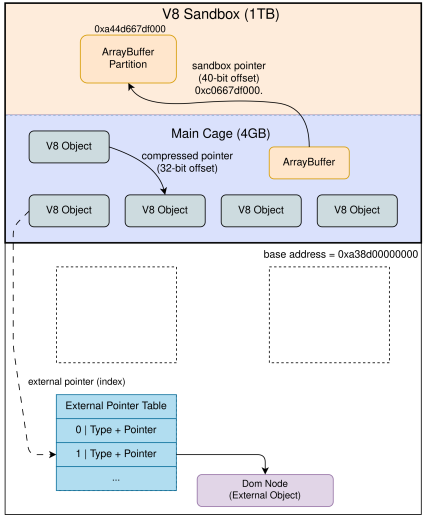
- Objectif : **confiner** les objets Javascript aux tas même si l'attaquant dispose d'une lecture/écriture arbitraire



Ligne de temps



Protection de l'ArrayBuffer

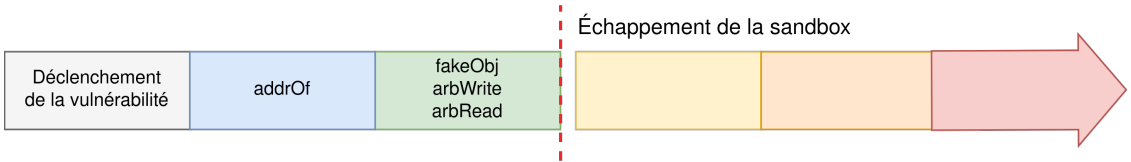


	Après la compression de pointeur	Création de la sandox
Fake		
address	0x3208d7d1	0x3208d7d1
Fake->element		
element	0x3208d7b8	0x3208d7b8
WasmInstance		
jump_table_start	0x98406acd000	0x98406acd000
ArrayBuffer		
BackingStore	0x52339310909456	**0xc0667df000**

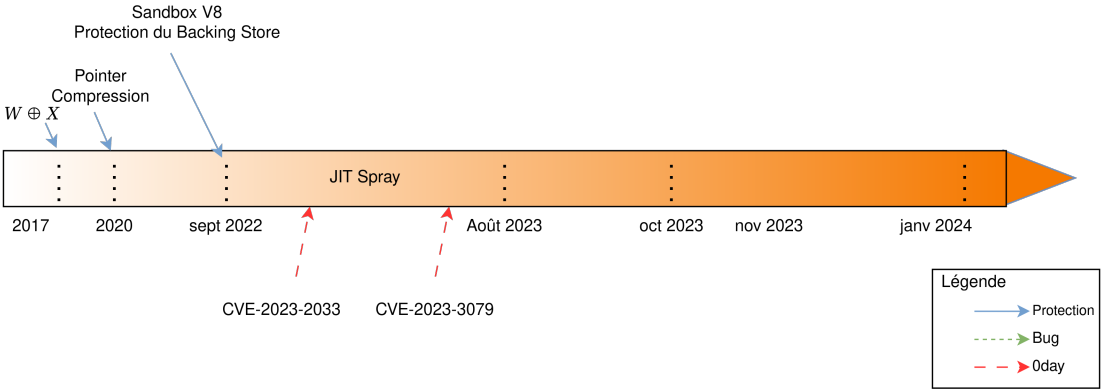
Stratégie

Échappement de la sandbox

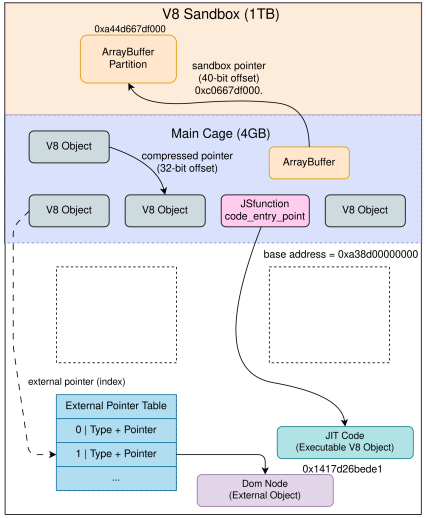
- Pour les attaques suivantes, l'attaquant a **déjà** réalisé les 3 premières étapes et dispose d'une **lecture et écritrire arbitraire** sur la *Main Cage*
- Les 3 dernières étapes :
  - ne repose plus sur l'ArrayBuffer
  - Elles varient pour chaque technique d'évasion



Ligne de temps

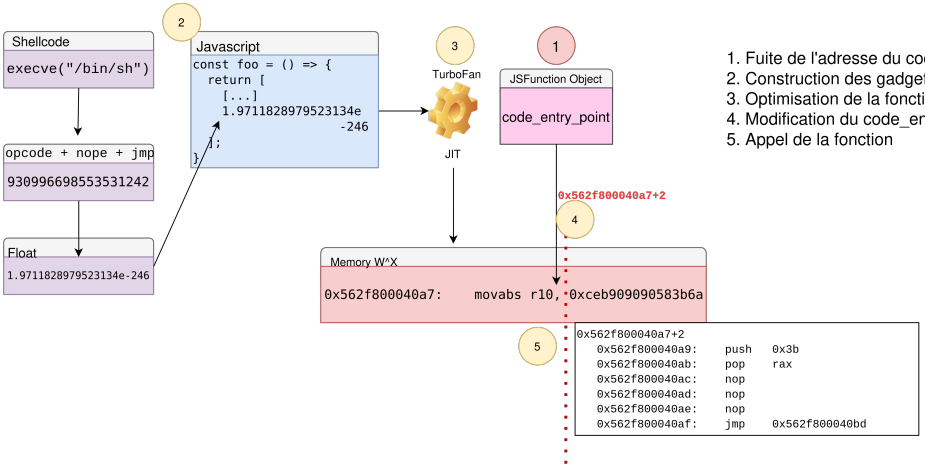


JIT Spray



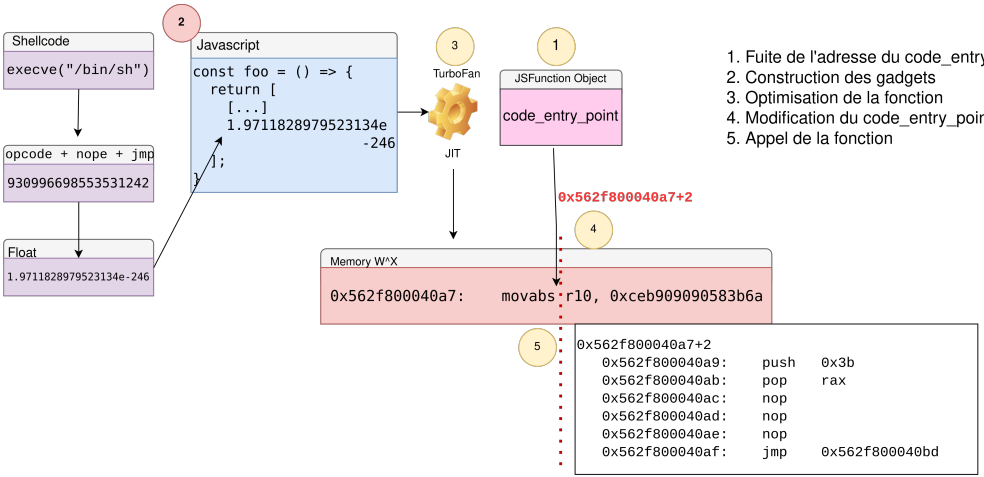
	Après la compression de pointeur	Création de la sandbox
Fake		
address	0x3208d7d1	0x3208d7d1
JSFunction		
code_entry_point	0x1417d26bede1	**0x1417d26bede1**
WasmInstance		
jump_table_start	0x98406acd000	0x98406acd000
ArrayBuffer		
BackingStore	0x5239931090945 6	0xc0667df000

JIT Spray



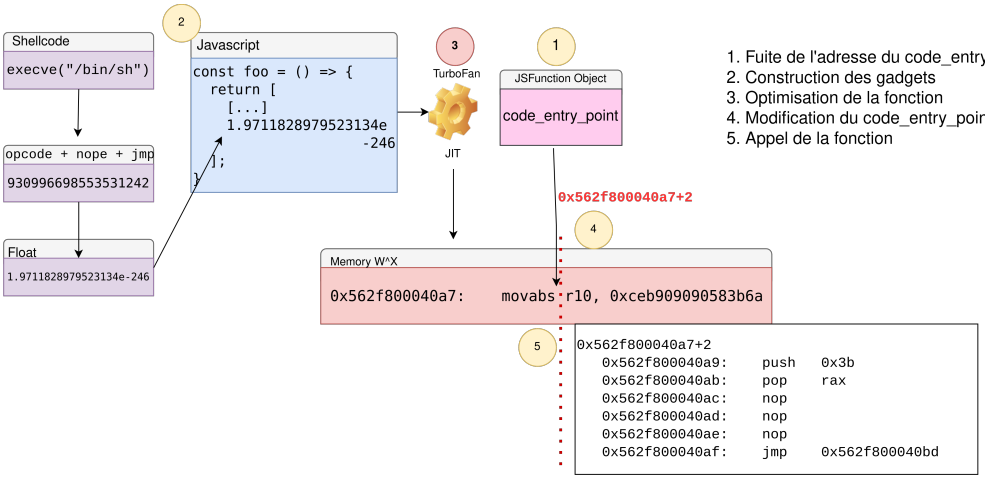
1. Fuite de l'adresse du code_entry_point
2. Construction des gadgets
3. Optimisation de la fonction
4. Modification du code_entry_point
5. Appel de la fonction

JIT Spray



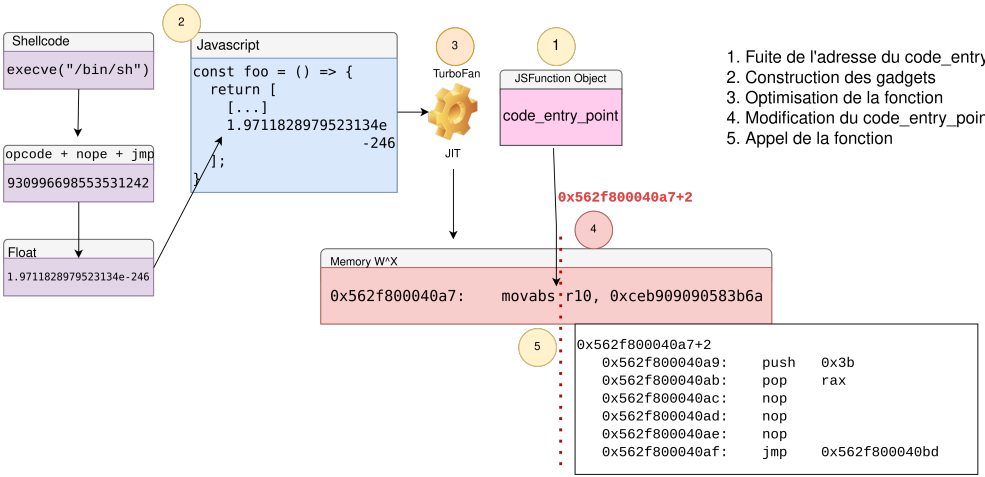
1. Fuite de l'adresse du code_entry_point
2. Construction des gadgets
3. Optimisation de la fonction
4. Modification du code_entry_point
5. Appel de la fonction

JIT Spray



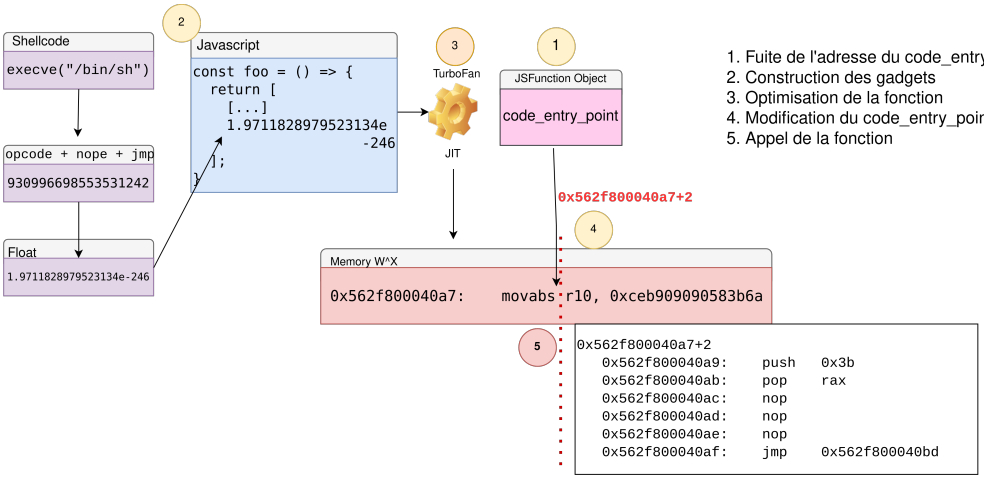
1. Fuite de l'adresse du code_entry_point
2. Construction des gadgets
3. Optimisation de la fonction
4. Modification du code_entry_point
5. Appel de la fonction

JIT Spray



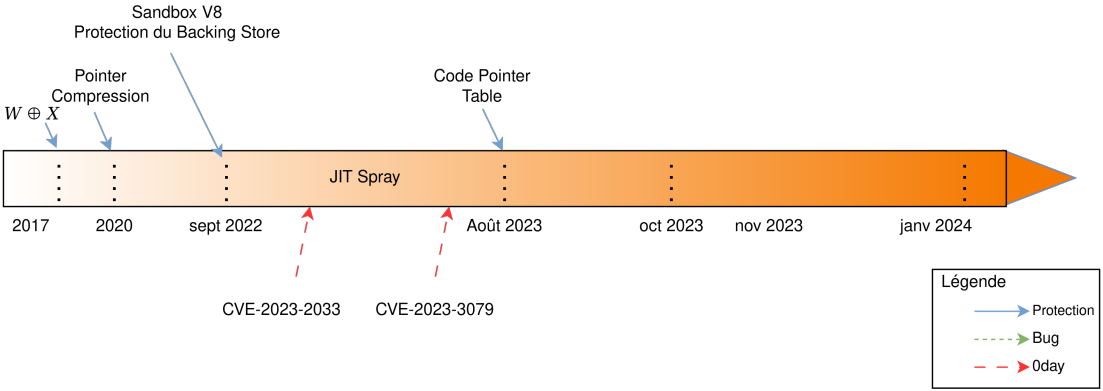
1. Fuite de l'adresse du code_entry_point
2. Construction des gadgets
3. Optimisation de la fonction
4. Modification du code_entry_point
5. Appel de la fonction

JIT Spray

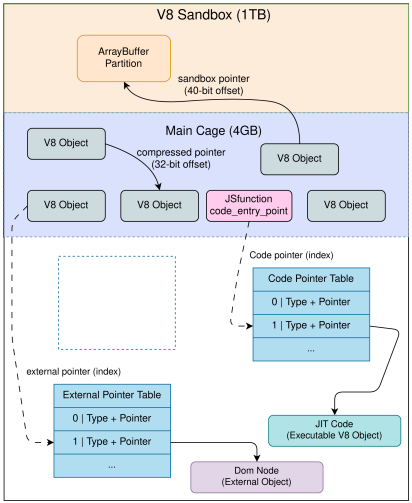


1. Fuite de l'adresse du code_entry_point
2. Construction des gadgets
3. Optimisation de la fonction
4. Modification du code_entry_point
5. Appel de la fonction

Ligne de temps

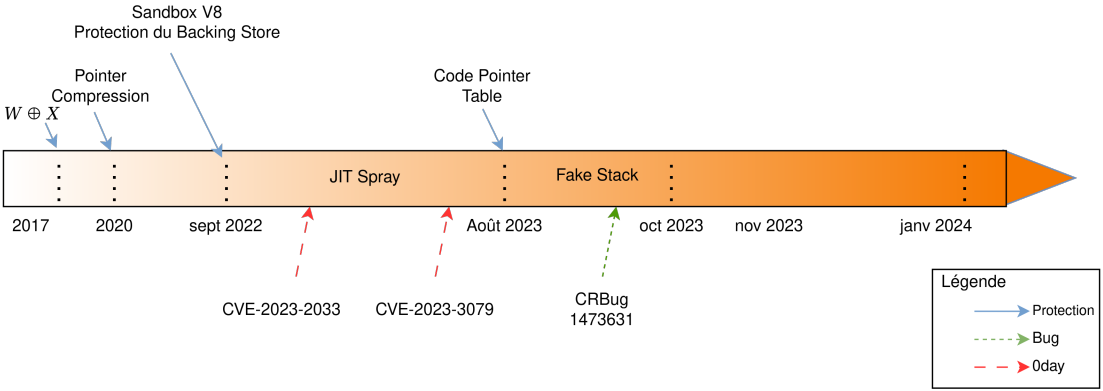


Code External Table

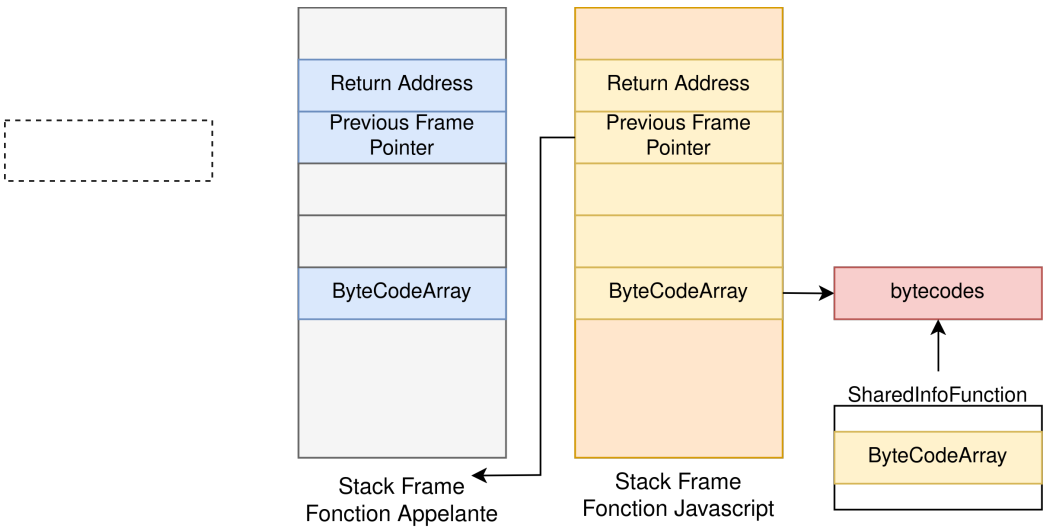


- Le pointeur **code_entry_point** est transformé en un offset par rapport à une table de pointeur appelée la **Code Pointer Table** (CPT)
- L'attaquant n'est donc plus en mesure de **modifier** le pointeur
- Des mesures de sécurité **supplémentaires** sont également ajoutées
  - Positionnement en dehors de la **sandbox**
  - *Control Flow Integrity* (*CFI*) : permet de s'assurer de **l'intégrité** du code avant son exécution
  - **Signature** basée sur le prototype de la fonction

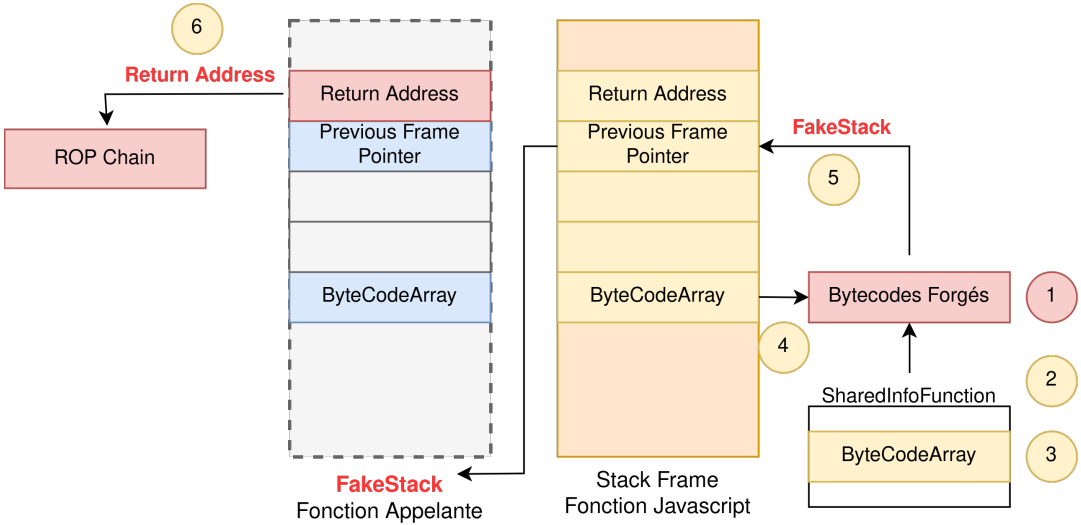
Ligne de temps



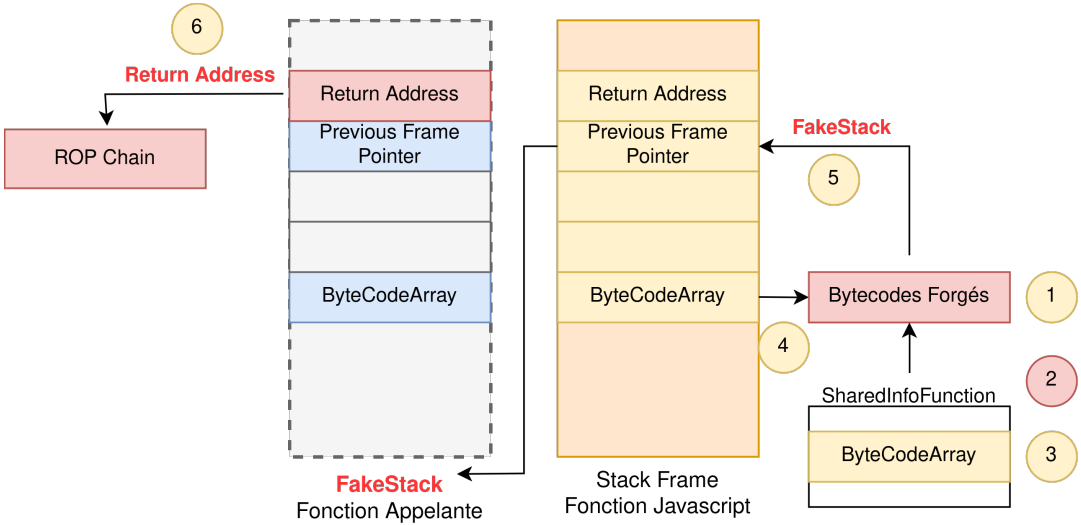
FakeStack


Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Appelante
Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Javascript
bytecodes
SharedInfoFunction
ByteCodeArray

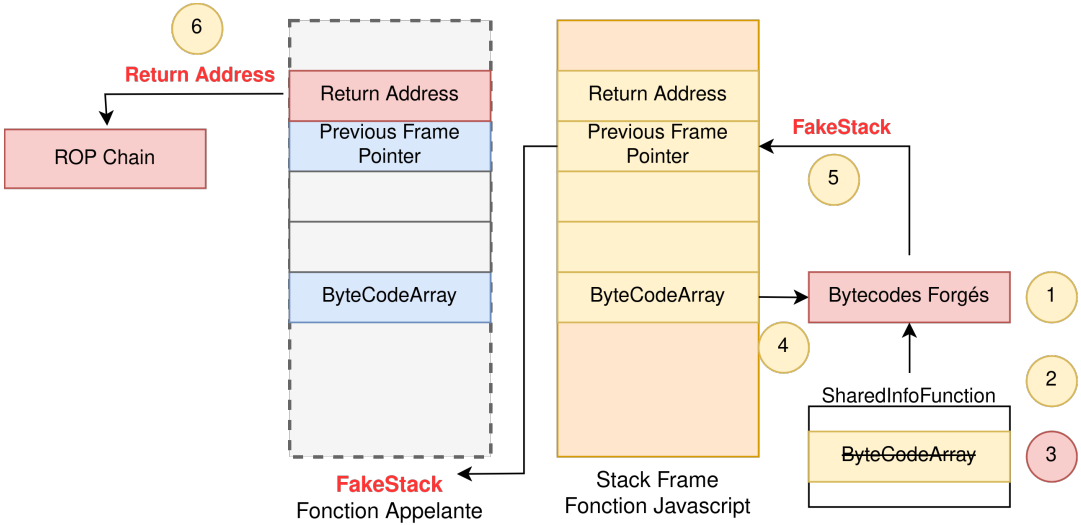
FakeStack


6
Return Address
ROP Chain
Return Address
Previous Frame Pointer
ByteCodeArray
FakeStack
Fonction Appelante
Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Javascript
FakeStack
5
Bytecodes Forgés
1
4
2
SharedInfoFunction
ByteCodeArray
3

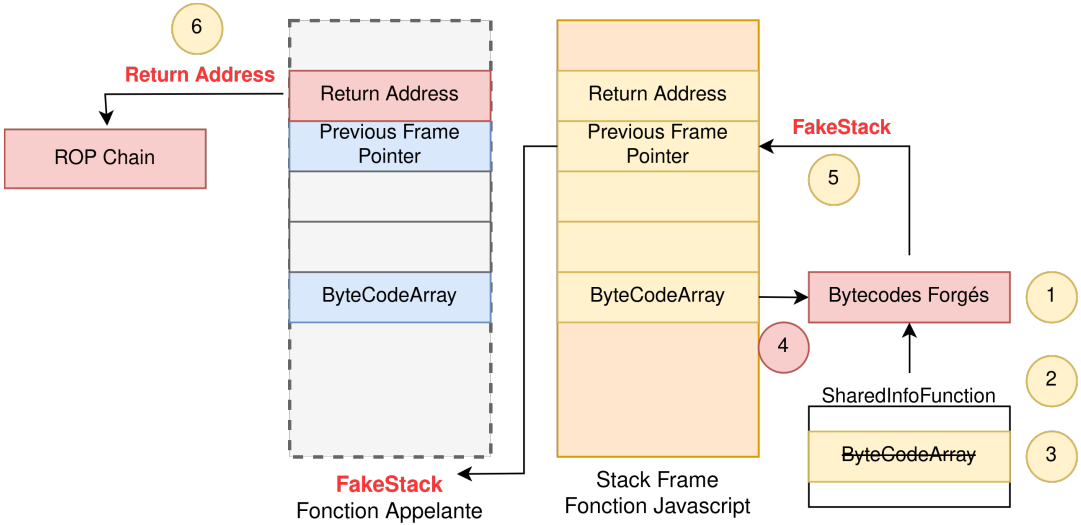
FakeStack


6
Return Address
ROP Chain
Return Address
Previous Frame Pointer
ByteCodeArray
FakeStack
Fonction Appelante
Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Javascript
FakeStack
5
Bytecodes Forgés
1
4
2
SharedInfoFunction
ByteCodeArray
3

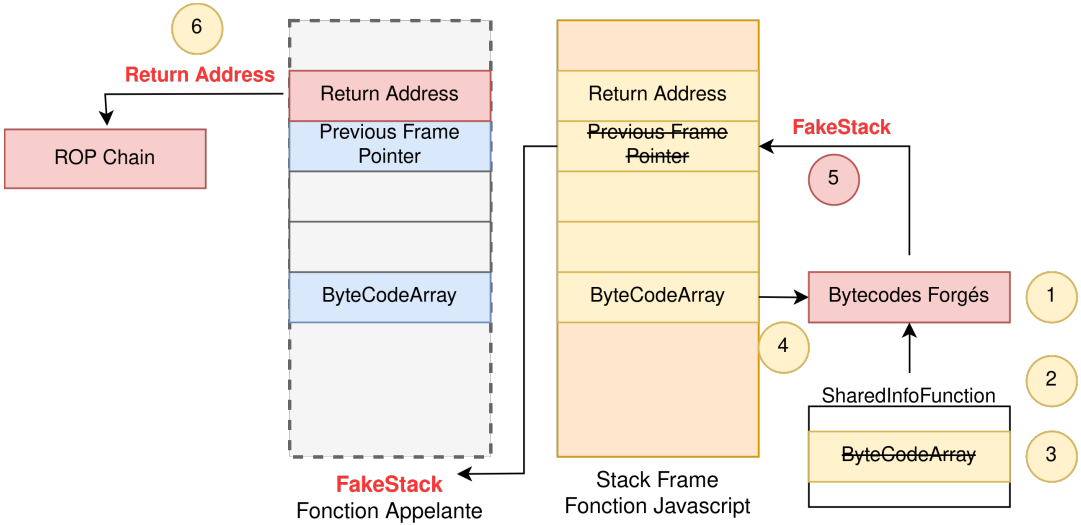
FakeStack


6
Return Address
ROP Chain
Return Address
Previous Frame Pointer
ByteCodeArray
FakeStack
Fonction Appelante
Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Javascript
FakeStack
5
Bytecodes Forgés
1
4
2
SharedInfoFunction
ByteCodeArray
3

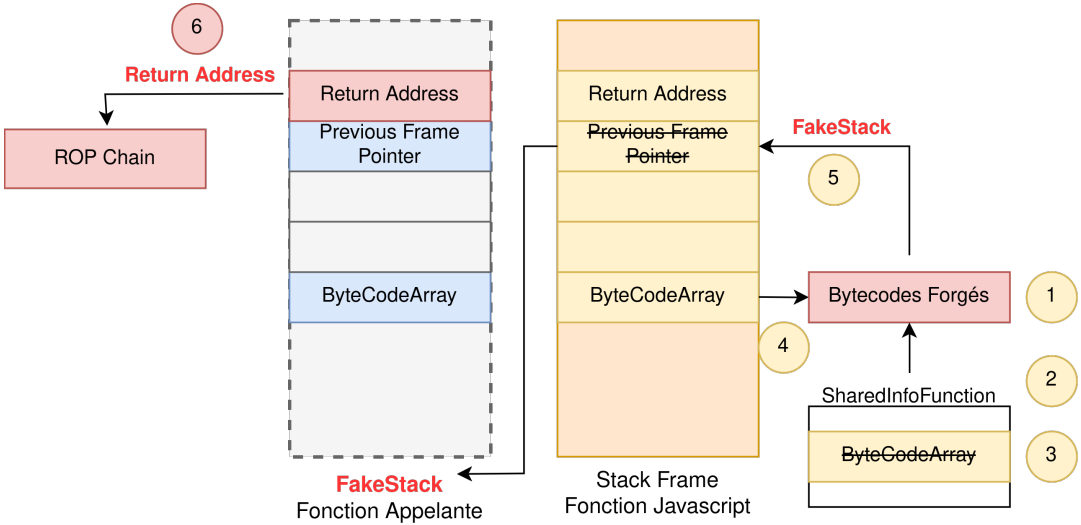
FakeStack


6
Return Address
ROP Chain
Return Address
Previous Frame Pointer
ByteCodeArray
FakeStack
Fonction Appelante
Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Javascript
FakeStack
5
Bytecodes Forgés
1
4
2
SharedInfoFunction
ByteCodeArray
3

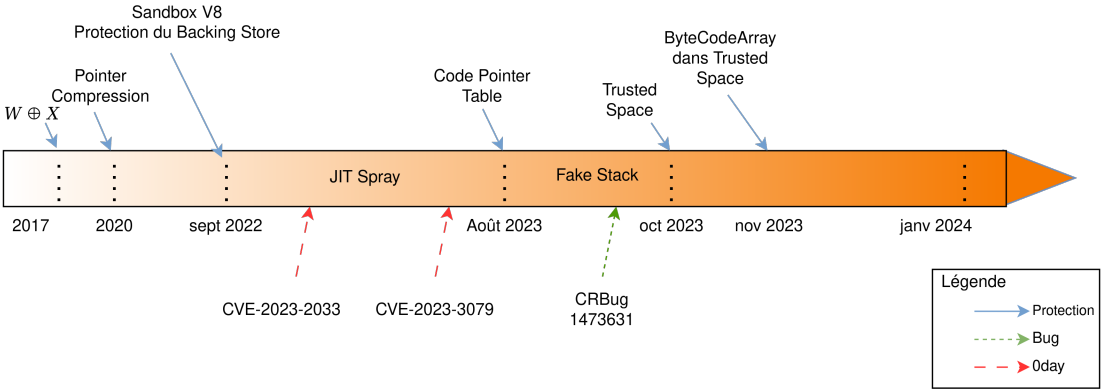
FakeStack


6
Return Address
ROP Chain
Return Address
Previous Frame Pointer
ByteCodeArray
FakeStack
Fonction Appelante
Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Javascript
FakeStack
5
Bytecodes Forgés
1
4
2
SharedInfoFunction
ByteCodeArray
3

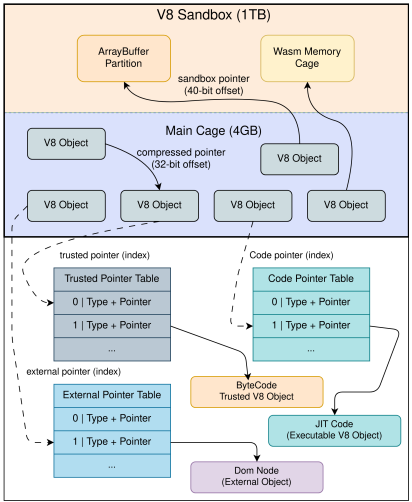
FakeStack


6
Return Address
ROP Chain
Return Address
Previous Frame Pointer
ByteCodeArray
FakeStack
Fonction Appelante
Return Address
Previous Frame Pointer
ByteCodeArray
Stack Frame
Fonction Javascript
FakeStack
5
Bytecodes Forgés
1
4
2
SharedInfoFunction
ByteCodeArray
3

Ligne de temps

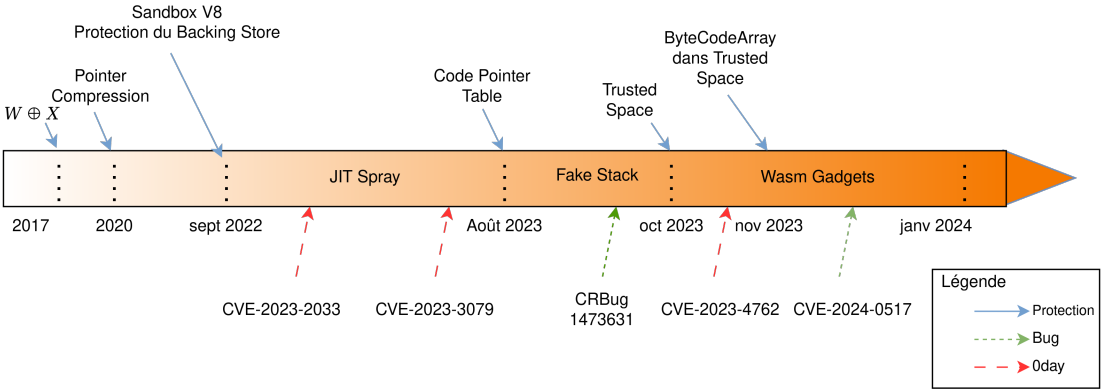


Trusted Space

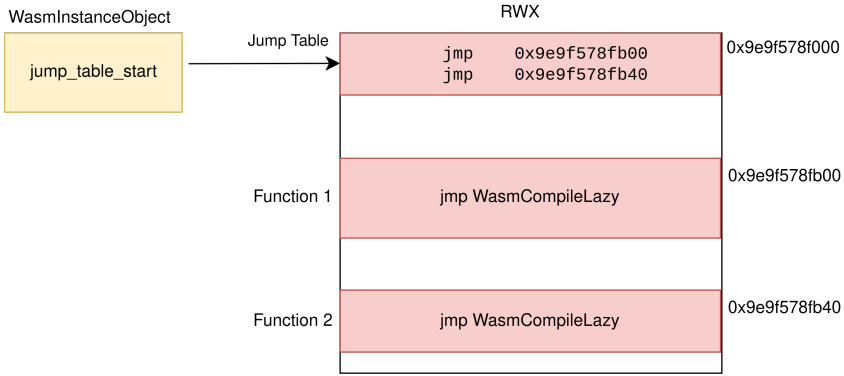


- **Trusted Space** : nouvel espace permettant d'isoler les objets critiques
- **Trusted Pointer Table** : contient les pointeurs vers les objets stockés dans la Trusted Space
- **Trusted Pointer** :
  - Permet à un objet sur la *Main Cage* de référencer un objet sur le *Trusted Space*
  - offset vers la *TPT*

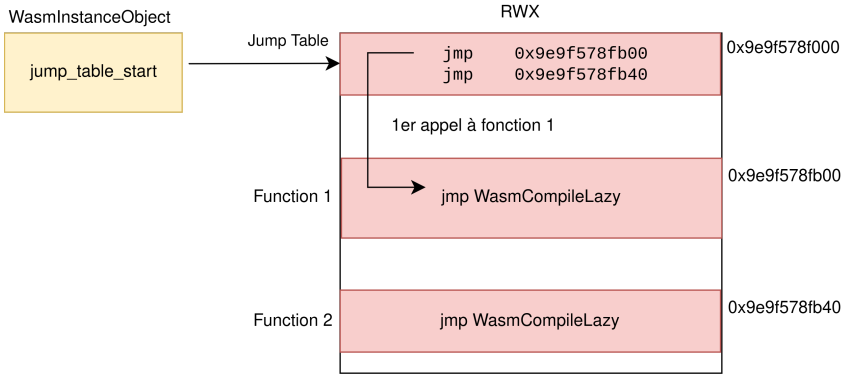
Ligne de temps



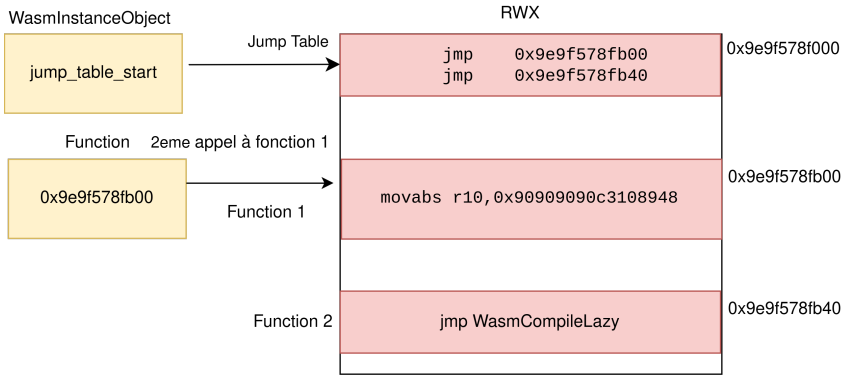
Wasm gadgets



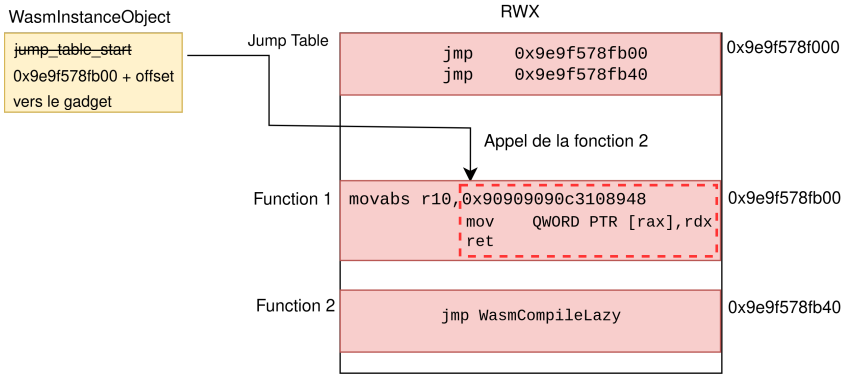
Wasm gadgets



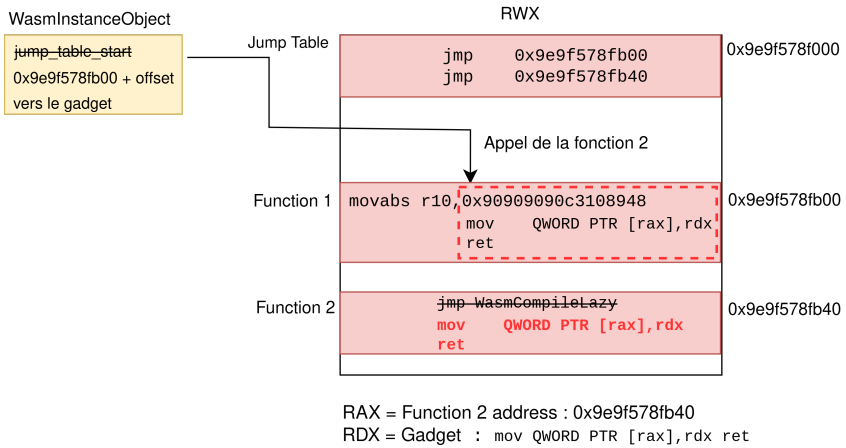
Wasm gadgets



Wasm gadgets

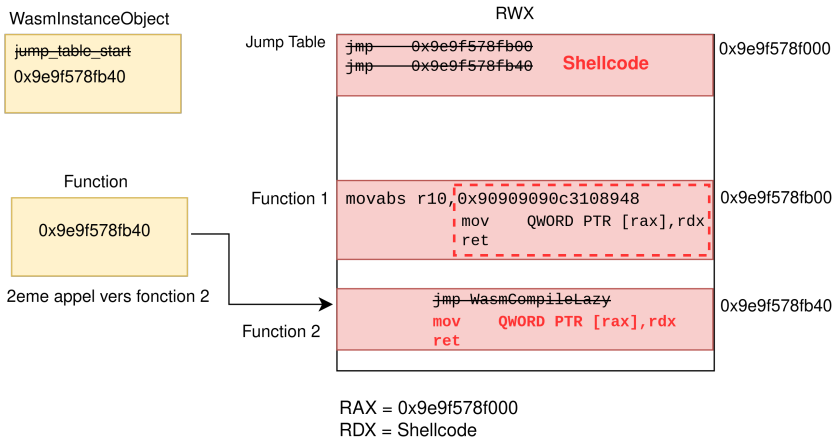


Wasm gadgets

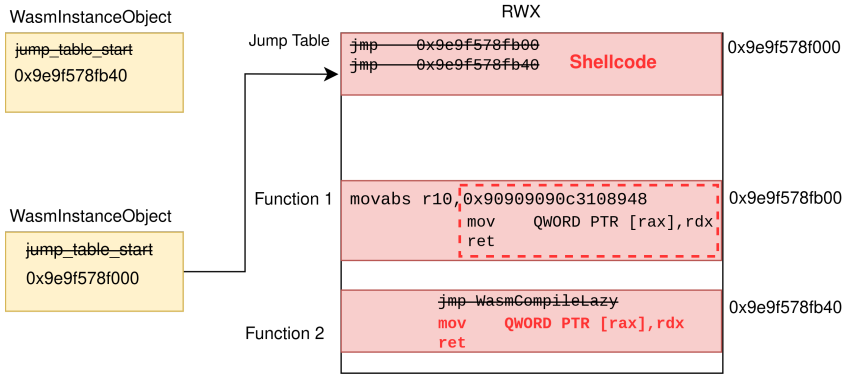


RAX = Function 2 address : 0x9e9f578fb40
RDX = Gadget : `mov QWORD PTR [rax],rdx ret`

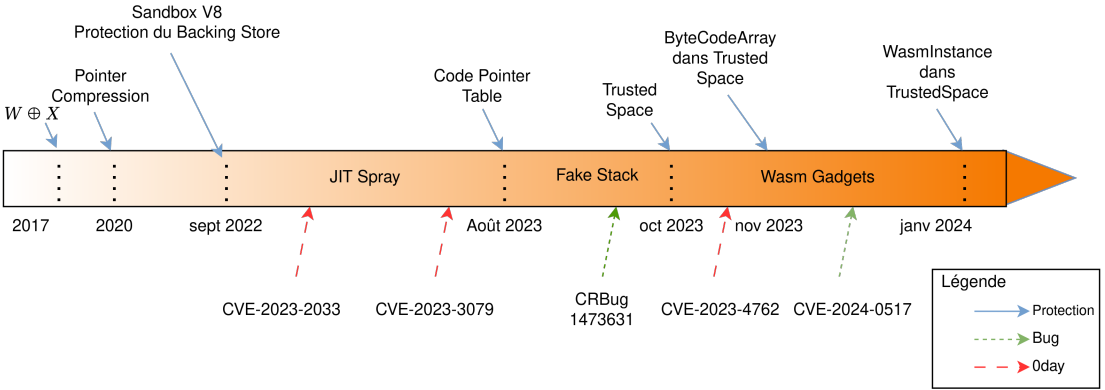
Wasm gadgets



Wasm gadgets



Ligne de temps



Conclusion

- La Sandbox V8 a permis de :
  - **Protéger** les pointeurs *BackingStore* et *jump_table_start*
  - **Neutraliser** les techniques d'évasion connues
  - **Compliquer** l'exploitation
  - Pour exploiter *V8*, il faut désormais disposer d'une **0day** supplémentaire pour évader la sandbox (exemple : **Pwn2Own 2024** exploit de *Manfred Paul*)
- La version **Beta** de la Sandbox *V8* a pris fin en début d'année 2024
- La Sandbox est amenée à évoluer
  - Intégration de protections basées sur des mécanismes **matériels** (*ARM BTI*, *Intel CET-IBT*)
  - **CTF** ouvert depuis octobre 2023

Des questions ?