

SpiderMonkey Type Confusion

CVE-2024-8381



Jack Ren

OS: Ubuntu 24.04

GLIBC: Ubuntu GLIBC 2.39-0ubuntu8.3

Clang: 18.1.7, Installed by ./mach bootstrap

Git Commit: 198d5fc1bebaaf114197a529ebdd4b9601045719

PoC Execute Command: `obj-debug-x86_64-pc-linux-gnu/dist/bin/js PoC.js`

Exploit Execute Command: `setarch x86_64 -R obj-opt-x86_64-pc-linux-gnu/dist/bin/js Exp.js`

2025/1/30

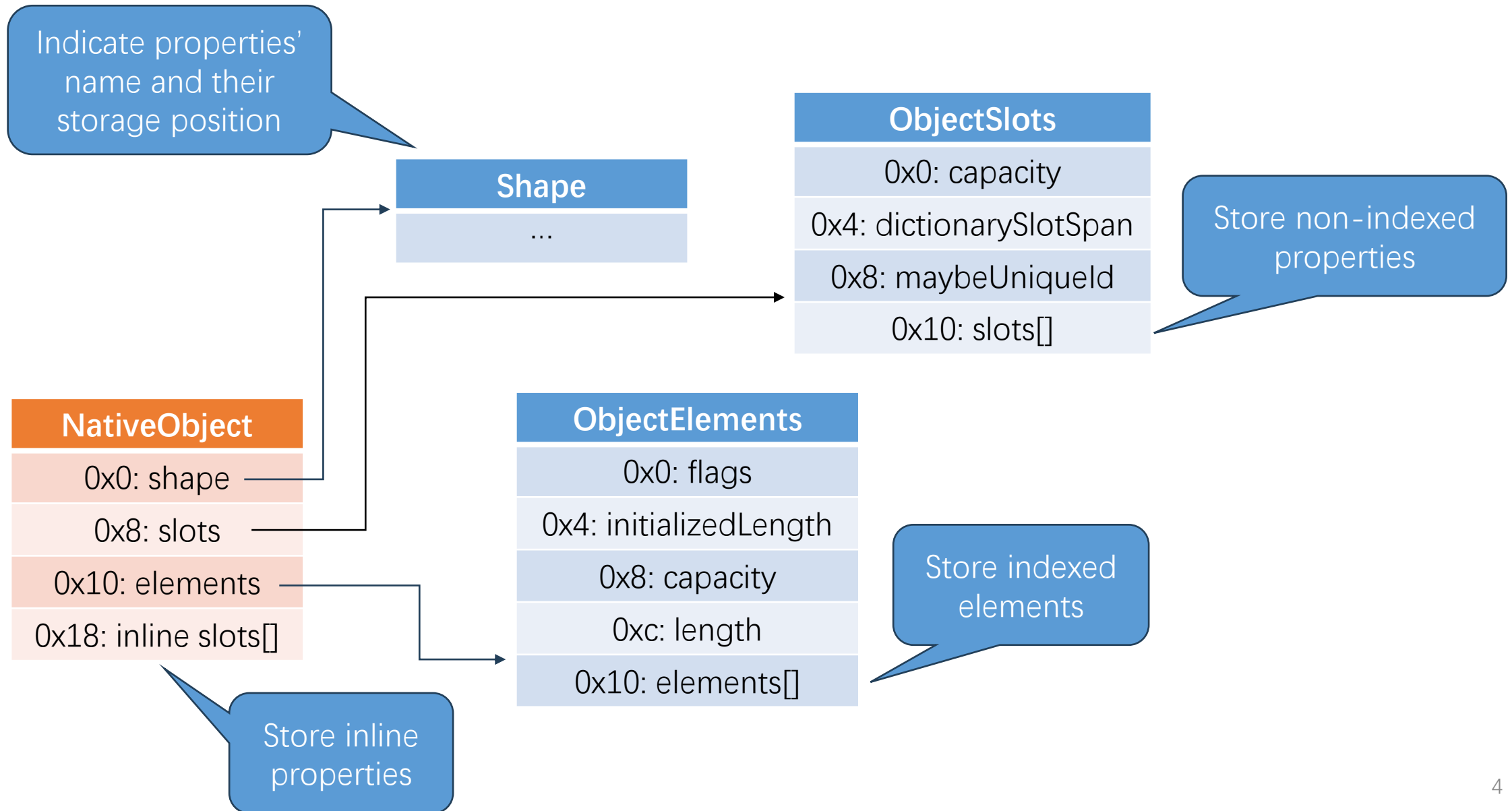
Contents

- Background
- Proof of Concept
- Exploitation

Contents

- Background
 - SpiderMonkey Engine
 - Object Layout

Object Layout




Contents

- Proof of Concept
 - PoC
 - Type Confusion Figure
 - Background
 - with statement
 - `Symbol.unscopables`
 - PoC Explanation

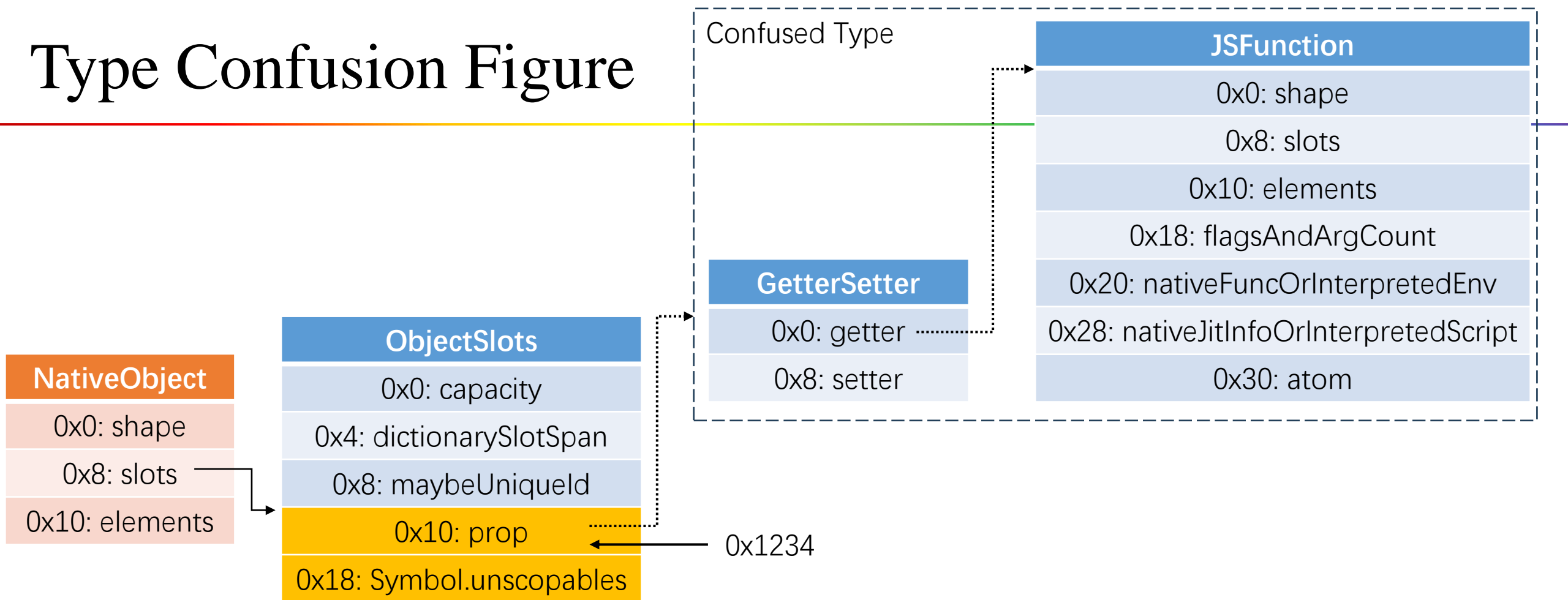
PoC

```
const obj = {  
  get prop() {  
    Object.defineProperty(this, "prop", { enumerable: true, value: 0x1234 });  
    return false;  
  },  
};  
obj[Symbol.unscopables] = obj;  
  
with (obj) {  
  assertEq(prop, 0x1234);  
}
```



SIGSEGV on
dereferencing
0x1234

Type Confusion Figure



Type Confusion Between **GetterSetter** and **Integer**

with statement

- The `with` statement extends the scope chain for a statement.
 - The `with` statement adds the given object to the head of this scope chain during the evaluation of its statement body. Every *unqualified name* would first be searched within the object (through a `in` check) before searching in the upper scope chain.
 - The following `with` statement specifies that the `Math` object is the default object. The statements following the `with` statement refer to the `PI` property and the `cos` and `sin` methods, without specifying an object. JavaScript assumes the `Math` object for these references.

```
let a, x, y;  
const r = 10;  
  
with (Math) {  
  a = PI * r * r;  
  x = r * cos(PI);  
  y = r * sin(PI / 2);  
}
```


Symbol.unscopables

- The `Symbol.unscopables` static data property represents the well-known symbol `Symbol.unscopables`.
 - The `with` statement looks up this symbol on the scope object for a property containing a collection of properties that should not become bindings within the `with` environment.

```
const object1 = {
  property1: 42,
};

object1[Symbol.unscopables] = {
  property1: true,
};

with (object1) {
  console.log(property1);
  // Expected output: Error: property1 is not defined
}
```

PoC Explanation

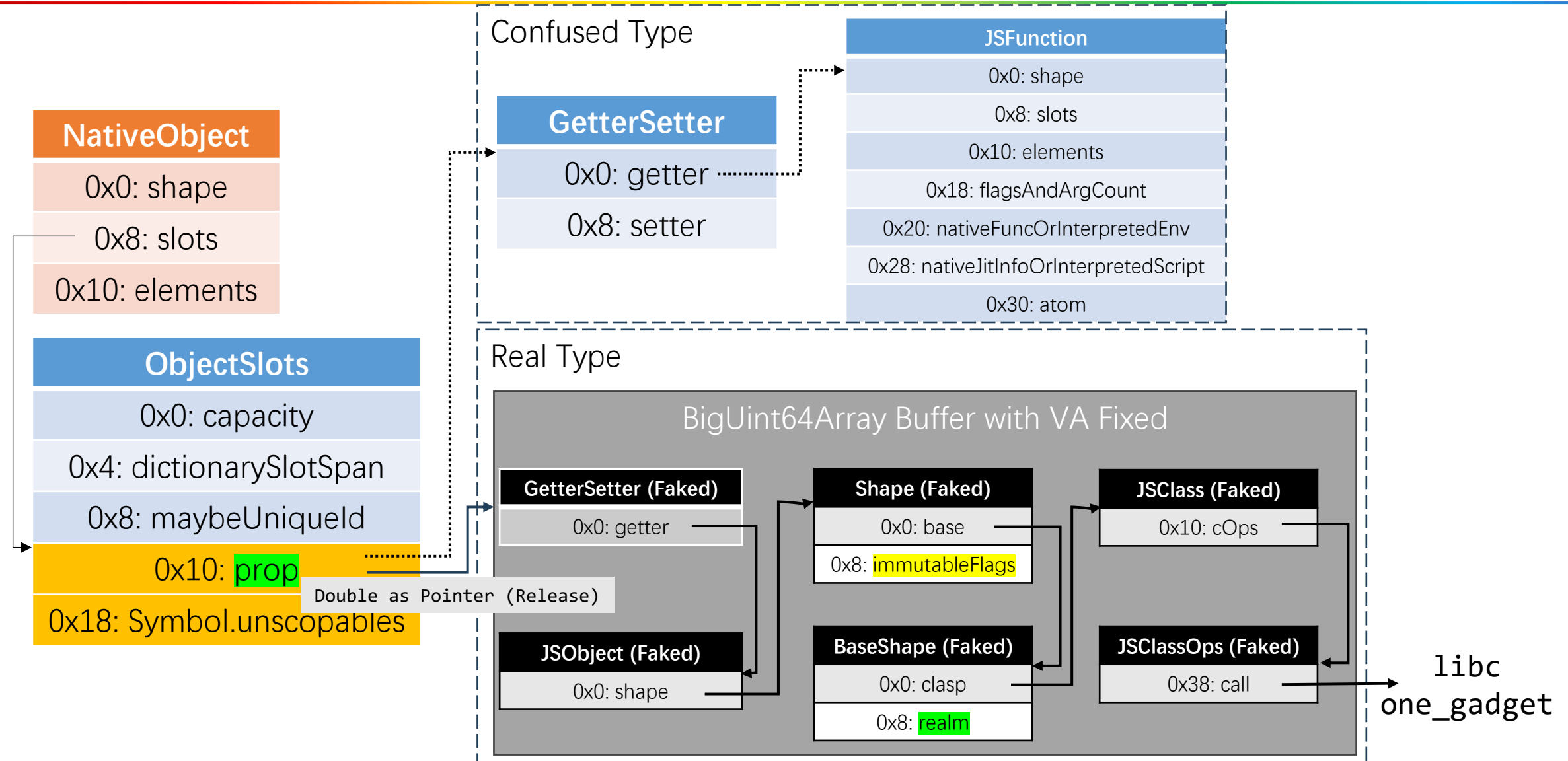
```
const obj = {
  get prop() {
    Object.defineProperty(this, "prop", { enumerable: true, value: 0x1234 });
    return false;
  },
};
obj[Symbol.unscopables] = obj;
with (obj) {
  assertEq(prop, 0x1234);
}
```

- When getting property `prop` from `obj`,
 1. Engine finds `obj.prop` is an accessor property.
 2. Engine tries to know `prop` should be a binding by getting `obj[Symbol.unscopables].prop`, which makes `obj.prop` become a data property and tell engine `prop` should be a binding.
 3. Engine tries to execute the accessor, but find the pointer to accessor becomes an integer. The engine crashes.

Contents

- Exploitation
 - RCE without ASLR
 - Demo
 - Tried methods to RCE

RCE without ASLR: Type Confusion Figure



Demo

```
○ jack@ubuntu2404-server-vmware:~/JavaScriptEngine/gecko-dev$
```

Tried methods to RCE with ASLR but failed

- Are we able to transform this Type Confusion with more exploitability?
 - Try 1: Mutate the PoC to call setter, instead of getter
 - Type Confusion doesn't happen anymore because
 - The variable GET operation is translated into `GetName` in bytecode.
 - `GetName` do cache the type of property, `GetterSetter`, for later use.
 - The variable SET operation is translated into `BindName` + `SetName` in bytecode.
 - `BindName` + `SetName` don't cache anything.

Tried methods to RCE with ASLR but failed

- Are we able to transform this Type Confusion with more exploitability?
 - Try 2: Assign the prop with a real object whose critical fields are controllable, instead of an integer
 - Selected possible object candidates: `Symbol` & `BigInt`
 - Selection Principle: The field at offset 0x0 of object candidates, which is the original place of pointer to `GetterSetter`, must be a pointer to a callable object to achieve control flow hijacking.
 - `Symbol`: The field at offset 0x0 of `Symbol` is a pointer to `JSAtom`.
 - The field at offset 0x0 of `JSAtom` is `lengthAndFlags`. To make engine consider `JSAtom` a callable object, we should mutate `lengthAndFlags` to a real `Shape` pointer or point to a fake `Shape`. To implement any of these, prerequisite is to bypass ASLR.
 - `BigInt`: The field at offset 0x0 of `BigInt` is `lengthAndFlags`.
 - We must fake an `JSObject` and make it callable, then make `lengthAndFlags` point to it. This will also need to bypass ASLR.
 - What's more, `lengthAndFlags` is hard to mutate due to it cannot be controlled by user directly.
 - To conclude, this try also seems unexploitable unless disabling ASLR and solving the mutating difficulty of `lengthAndFlags`.

Tried methods to RCE without ASLR

- Are we able to transform this Type Confusion with more exploitability?
 - Give up bypassing ASLR 🙄
 - Try 3: Assign the `prop` with a double, instead of an integer, to make engine consider it a pointer to object in release compilation.
 - Once we give up bypassing ASLR, we'll find that the address of data buffer of an allocated tremendous large `TypedArray` is fixed without ASLR.
 - Then, we fake all need data structures in the address-fixed buffer to achieve control flow hijacking.
 - This is easier than previous try because we don't need to mutate `lengthAndFlags`.
 - This type confusion figure is presented in preceded slides.

Thank you!

Reference

1. <https://www.cve.org/CVERecord?id=CVE-2024-8381>
2. <https://github.com/mozilla/gecko-dev/commit/fab7e5c28e628ddc2b873a723838562c9b41205e>
3. <https://github.com/mozilla/gecko-dev/commit/0ca509a3a7fbf4ff5d34cf25083a4427f3205549>
4. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/with>
5. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol/unscopables