

Hit the Bullseye: Detecting Browser Exploits Abusing the X Memory in WebAssembly

Tao Yan & Edouard Bochin

Palo Alto Networks



Virus Bulletin 2023 London

About Us

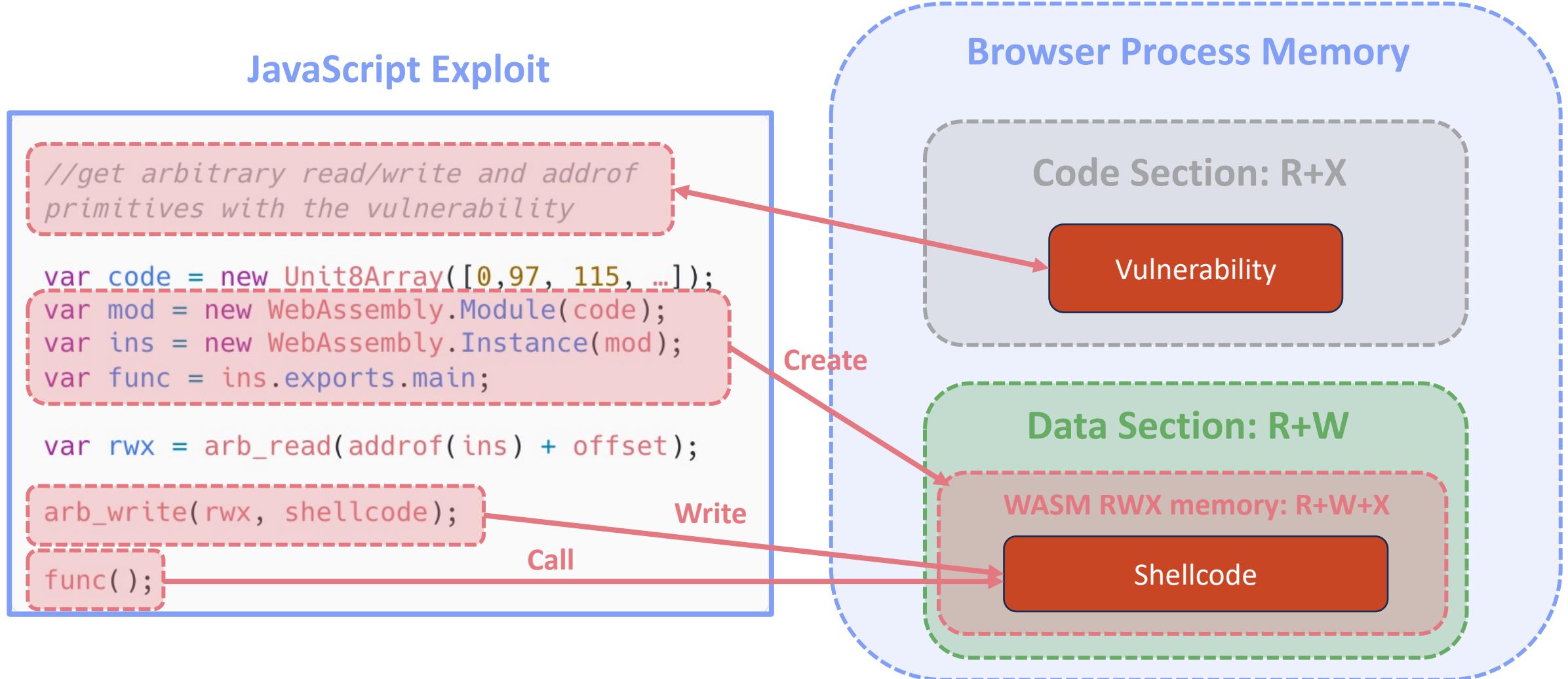
- Security Researchers from Palo Alto Networks
 - Tao Yan (@Ga1ois)
 - Edouard Bochin
- Vulnerability researchers
 - Multiple times for MSRC Top 10 Researchers
- Pwn2Own winner
- Conference speakers
 - Black Hat, CanSecWest, Blue Hat, POC, HITCON, Recon, etc
- Patent inventors
 - New defense and detection techniques

Agenda

- Background & Introduction
- V8 WebAssembly Engine Internals
- WASMGuard
- Demo
- Summary

Background & Introduction

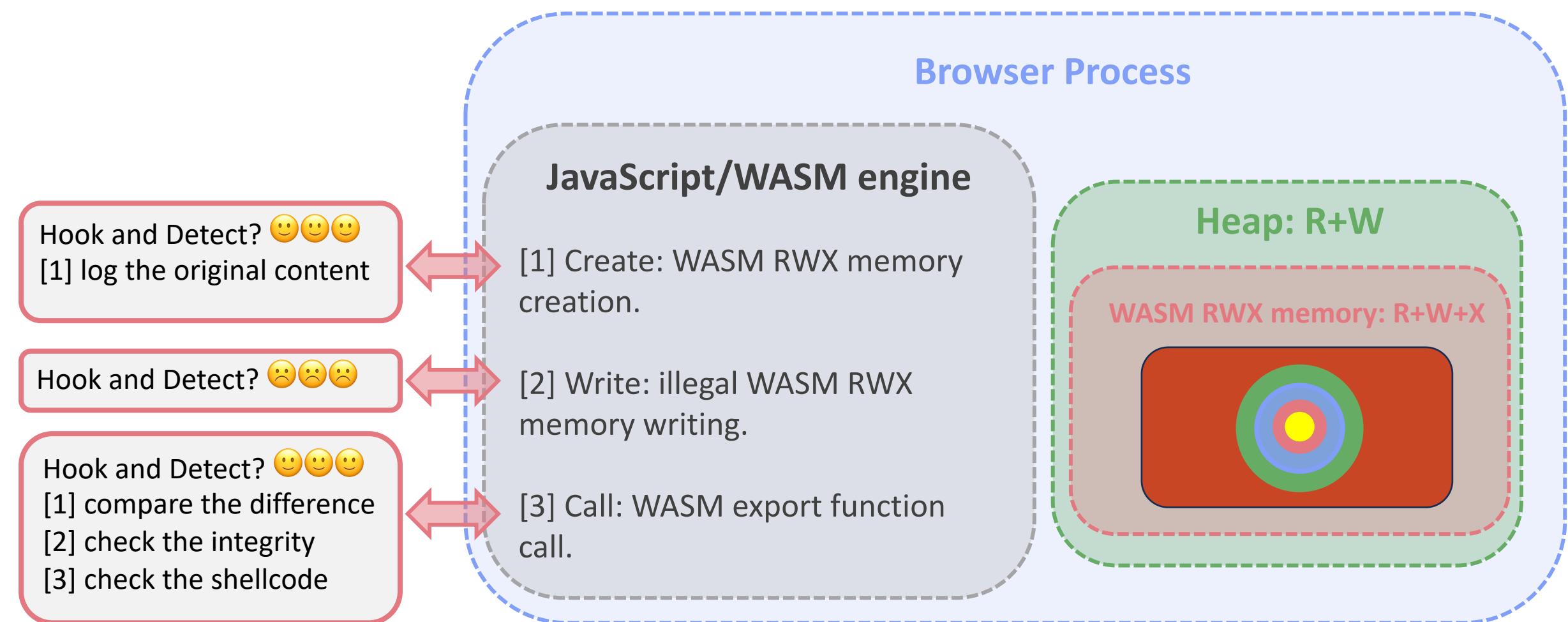
Exploits abusing the X memory in WebAssembly



How popular is it and why?

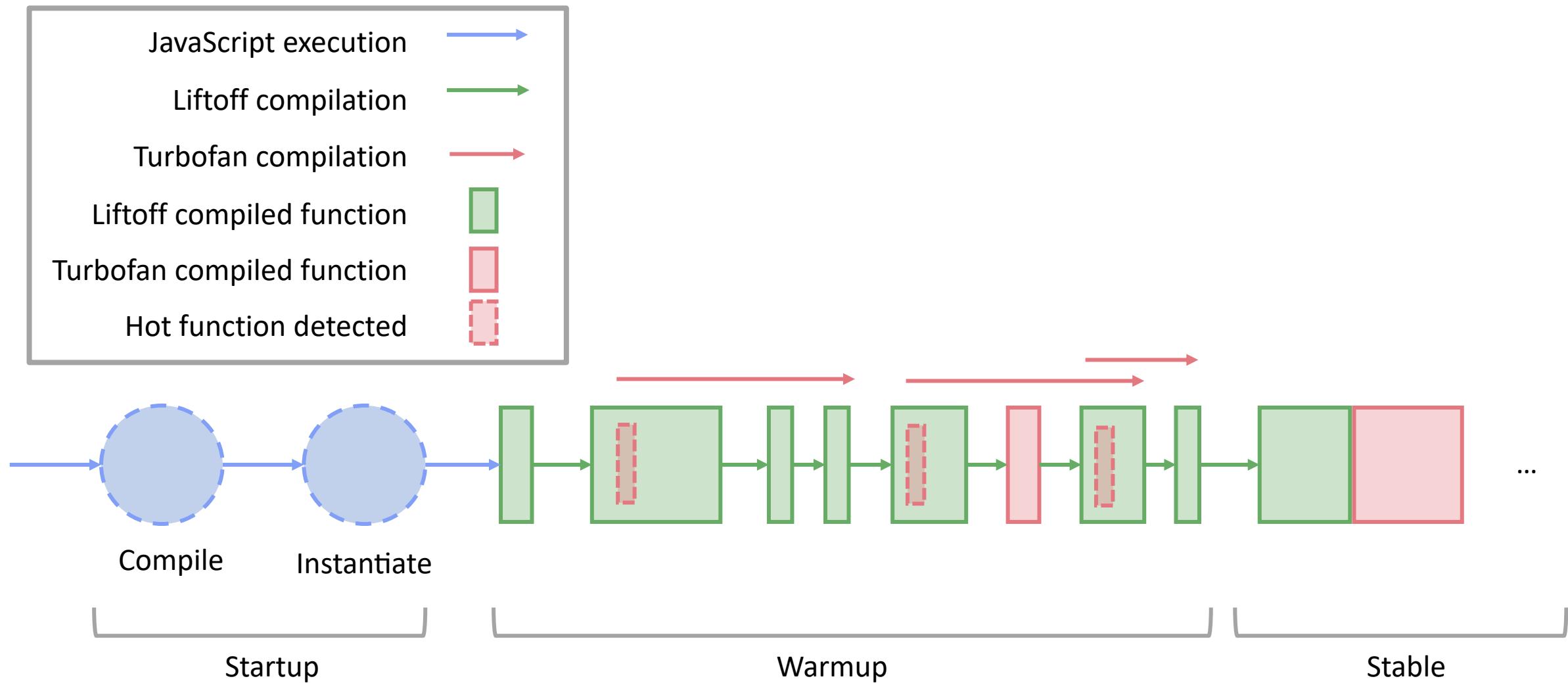
- Statistics of the past 3 years
 - 29 WASM Vs 3 ROP in 32 publicly disclosed exploits (including 0 days, n days, exploits used in Pwn2Own, etc)
- Simple, effective and stable
 - An existing RWX memory region, exploit-friendly in born
 - A perfect way to bypass DEP / NX
- Mitigation
 - There is a `WASM_MEMORY_PROTECTION` flag
 - But easy to be bypassed after getting arbitrary read & write primitives
 - V8 sandbox
 - Efficient but could be bypassed (Example with WASM Globals)
 - Mitigation != Detection

Detecting the exploitation technique



V8 WebAssembly Engine Internals

V8 WebAssembly compilation pipeline



WebAssembly RWX Memory Allocation

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals

```
void WebAssemblyModule(
    const v8::FunctionCallbackInfo<v8::Value>& args);

MaybeHandle<WasmModuleObject>
WasmEngine::SyncCompile(Isolate* isolate, WasmFeatures
enabled, ErrorThrower* thrower, ModuleWireBytes bytes);

std::shared_ptr<NativeModule> CompileToNativeModule(
    Isolate* isolate, ...);

std::shared_ptr<NativeModule>
WasmEngine::NewNativeModule(Isolate* isolate, ...);
```

WASM RWX memory

0x42a81be9000	Int 3
0x42a81be9001	Int 3
0x42a81be9002	Int 3
0x42a81be9003	Int 3
0x42a81be9004	Int 3
0x42a81be9005	Int 3
0x42a81be9006	Int 3
0x42a81be9007	Int 3
0x42a81be9008	Int 3
0x42a81be9009	Int 3
.	.
.	.
.	.

WebAssembly jump tables creation

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals

```
void NativeModule::AddCodeSpaceLocked(base::AddressRegion
region){
...
JumpTableAssembler::GenerateFarJumpTable(...);
...
InitializeJumpTableForLazyCompilation(...);
}

void NativeModule::InitializeJumpTableForLazyCompilation(
    uint32_t num_wasm_functions) {
...
JumpTableAssembler::GenerateLazyCompileTable(...);

    JumpTableAssembler::InitializeJumpsToLazyCompileTable(...);
}
```

WASM RWX memory

0x42a81be9000	int 3
0x42a81be9002	int 3
0x42a81be9003	int 3
0x42a81be9004	int 3
0x42a81be9005	int 3
0x42a81be9006	int 3
0x42a81be9007	int 3
0x42a81be9008	int 3
0x42a81be9009	int 3
0x42a81be900a	int 3
0x42a81be900b	int 3
0x42a81be900c	int 3
0x42a81be900d	int 3
0x42a81be900e	int 3
0x42a81be900f	int 3
0x42a81be9010	int 3
0x42a81be9011	int 3
0x42a81be9012	int 3
...	...

WebAssembly jump tables creation

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals

```
void NativeModule::AddCodeSpaceLocked(base::AddressRegion
region){
    ...
    JumpTableAssembler::GenerateFarJumpTable(...);
    ...
    InitializeJumpTableForLazyCompilation(...);
}

void NativeModule::InitializeJumpTableForLazyCompilation(
    uint32_t num_wasm_functions) {
    ...
    JumpTableAssembler::GenerateLazyCompileTable(...);

    JumpTableAssembler::InitializeJumpsToLazyCompileTable(...);
}
```

WASM RWX memory

0x42a81be9000	int 3
0x42a81be9001	int 3
...	...
0x42a81be9040	jmp qword ptr[rip+0x2]
...	...
0x42a81be9048	0xfffff4b0b600
...	...
0x42a81be9150	jmp qword ptr[rip+0x2]
...	...
0x42a81be9158	0xfffff4609600
...	...
0x42a81be9691	int 3
0x42a81be9692	int 3
0x42a81be9693	int 3
0x42a81be9694	int 3
0x42a81be9695	int 3
0x42a81be9696	int 3
0x42a81be9697	int 3
...	...

Far Jump Table

WebAssembly jump tables creation

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals

```
void NativeModule::AddCodeSpaceLocked(base::AddressRegion
region){
    ...
    JumpTableAssembler::GenerateFarJumpTable(...);
    ...
    InitializeJumpTableForLazyCompilation(...);
}

void NativeModule::InitializeJumpTableForLazyCompilation(
    uint32_t num_wasm_functions) {
    ...
    JumpTableAssembler::GenerateLazyCompileTable(...);
    ...
    JumpTableAssembler::InitializeJumpsToLazyCompileTable(...);
}
```

WASM RWX memory

0x42a81be9000	int 3
0x42a81be9001	int 3
...	...
0x42a81be9040	jmp qword ptr[rip+0x2]
...	...
0x42a81be9048	0xfffff4b0b600
...	...
0x42a81be9150	jmp qword ptr[rip+0x2]
...	...
0x42a81be9158	0xfffff4609600
...	...
0x42a81be96c0	Push 0x0
0x42a81be96c5	jmp 0x42a81be9150
0x42a81be96ca	Push 0x1
0x42a81be96cb	jmp 0x42a81be9150
...	...
0x42a81be9700	add BYTE PTR [rax],al
0x42a81be9702	add BYTE PTR [rax],al
...	...

Far Jump Table

Lazy Compile Table

Start compiled code

WebAssembly jump tables creation

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals

```
void NativeModule::AddCodeSpaceLocked(base::AddressRegion
region){
    ...
    JumpTableAssembler::GenerateFarJumpTable(...);
    ...
    InitializeJumpTableForLazyCompilation(...);
}

void NativeModule::InitializeJumpTableForLazyCompilation(
    uint32_t num_wasm_functions) {
    ...
    JumpTableAssembler::GenerateLazyCompileTable(...);
    ...
    JumpTableAssembler::InitializeJumpsToLazyCompileTable(...);
}
```

WASM RWX memory

0x42a81be9000	int 3
0x42a81be9001	int 3
...	...
0x42a81be9040	jmp qword ptr[rip+0x2]
...	...
0x42a81be9048	0xfffff4b0b600
...	...
0x42a81be9150	jmp qword ptr[rip+0x2]
...	...
0x42a81be9158	pWasmCompileLazy
...	...
0x42a81be96c0	Push 0x0
0x42a81be96c5	jmp 0x42a81be9150
0x42a81be96ca	Push 0x1
0x42a81be96cb	jmp 0x42a81be9150
...	...
0x42a81be9700	add BYTE PTR [rax],al
0x42a81be9702	add BYTE PTR [rax],al
...	...

Far Jump Table

Lazy Compile Table

Start compiled code

WebAssembly jump tables creation

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
})();
```

V8 internals

```
void NativeModule::AddCodeSpaceLocked(base::AddressRegion
region){
  ...
  JumpTableAssembler::GenerateFarJumpTable(...);
  ...
  InitializeJumpTableForLazyCompilation(...);
}

void NativeModule::InitializeJumpTableForLazyCompilation(
    uint32_t num_wasm_functions) {
  ...
  JumpTableAssembler::GenerateLazyCompileTable(...);
  ...
  JumpTableAssembler::InitializeJumpsToLazyCompileTable(...);
}
```

WASM RWX memory

0x42a81be9000	jmp 0x42a81be96c0	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	
...	...	
0x42a81be9048	0xfffff4b0b600	
...	...	
0x42a81be9150	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9158	pWasmCompileLazy	
...	...	
0x42a81be96c0	Push 0x0	
0x42a81be96c5	jmp 0x42a81be9150	Lazy Compile Table
0x42a81be96ca	Push 0x1	
0x42a81be96cb	jmp 0x42a81be9150	
...	...	
0x42a81be9700	add BYTE PTR [rax],al	
0x42a81be9702	add BYTE PTR [rax],al	Start compiled code
...	...	

WebAssembly Module Instantiation

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
})();
```

```
(async () => {
  const module = await WebAssembly.compileStreaming(
    fetch('../out/main.wasm'));
  const instance = await WebAssembly.instantiate(module);
  const result = instance.exports.main();
  console.log(result);
})();
```

```
(async () => {
  const fetchPromise = fetch('../out/main.wasm');
  const { instance } = await
  WebAssembly.instantiateStreaming(fetchPromise);
  const result = instance.exports.main();
  console.log(result);
})();
```

There are different ways to instantiate a WebAssembly Module :

- `WebAssembly.Instance()`
- `WebAssembly.instantiate(module)`
- `WebAssembly.instantiateStreaming()`

But all of them are calling the same low level WebAssembly engine internal function

`v8::internal::wasm::InstantiateToInstanceObject.`

WebAssembly Module Instantiation

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
})();
```

V8 internals

```
MaybeHandle<WasmInstanceObject> InstantiateToInstanceObject(
  Isolate* isolate, ErrorThrower* thrower,
  Handle<WasmModuleObject> module_object, MaybeHandle<JSReceiver>
    imports, MaybeHandle<JSArrayBuffer> memory_buffer);
```

```
MaybeHandle<WasmInstanceObject> InstanceBuilder::Build()
```

WASM RWX memory

0x42a81be9000	jmp 0x42a81be96c0
0x42a81be9005	jmp 0x42a81be96ca
...	...
0x42a81be9040	jmp qword ptr[rip+0x2]
...	...
0x42a81be9048	0x7ffff4b0b600
...	...
0x42a81be9150	jmp qword ptr[rip+0x2]
...	...
0x42a81be9158	pWasmCompileLazy
...	...
0x42a81be96c0	Push 0x0
0x42a81be96c5	jmp 0x42a81be9150
0x42a81be96ca	Push 0x1
0x42a81be96cb	jmp 0x42a81be9150
...	...
0x42a81be9700	add BYTE PTR [rax],al
0x42a81be9702	add BYTE PTR [rax],al
...	...

Main Jump Table

Far Jump Table

Lazy Compile Table

Start compiled code

WebAssembly Module Instantiation

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals

```
class WasmInstanceObject {
public:
    Tagged<WasmModuleObject> module_object;
    Tagged<JSObject> exports_object;
    Tagged<Context> native_context;
    Tagged<FixedArray> memory_objects;
    ...
    Tagged<FixedArray> indirect_function_tables;
    ...
    Tagged<FixedArray> wasm_internal_functions;
    ...
    Address stack_limit_address;
    Address real_stack_limit_address;
    ...
    uint32_t indirect_function_table_size;
    Address jump_table_start;
};
```

WASM RWX memory

0x42a81be9000	jmp 0x42a81be96c0
0x42a81be9005	jmp 0x42a81be96ca
...	...
0x42a81be9040	jmp qword ptr[rip+0x2]
...	...
0x42a81be9048	0xfffff4b0b600
...	...
0x42a81be9150	jmp qword ptr[rip+0x2]
...	...
0x42a81be9158	pWasmCompileLazy
...	...
0x42a81be96c0	Push 0x0
0x42a81be96c5	jmp 0x42a81be9150
0x42a81be96ca	Push 0x1
0x42a81be96cb	jmp 0x42a81be9150
...	...
0x42a81be9700	add BYTE PTR [rax],al
0x42a81be9702	add BYTE PTR [rax],al
...	...

Main Jump Table

Far Jump Table

Lazy Compile Table

Start compiled code

Processing export functions

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
})();
```

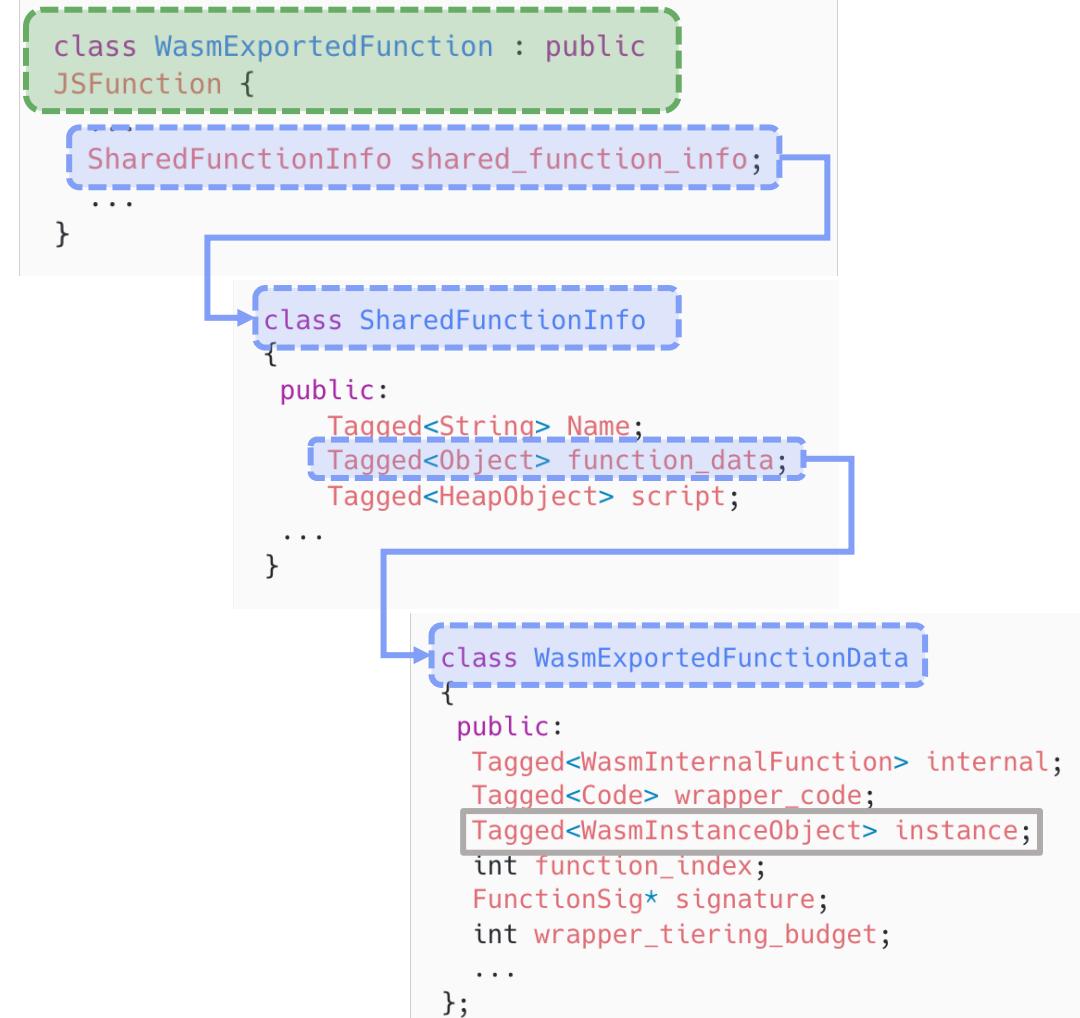
V8 internals

```
void InstanceBuilder::ProcessExports(Handle<WasmInstanceObject>
instance);

Handle<JSFunction> WasmInternalFunction::GetOrCreateExternal(
    Handle<WasmInternalFunction> internal)

Handle<WasmExportedFunction> WasmExportedFunction::New(
    Isolate* isolate, Handle<WasmInstanceObject> instance,
    Handle<WasmInternalFunction> internal, int func_index, int arity,
    Handle<Code> export_wrapper);
```

V8 internal classes



WebAssembly export functions call

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
})();
```

V8 internals

```
void Builtins_GenericJSToWasmWrapper(
    JSFunction rdi, ...)
```

WASM RWX memory

0x42a81be9000	jmp 0x42a81be96c0	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9048	0xfffff4b0b600	
...	...	
0x42a81be9150	jmp qword ptr[rip+0x2]	
...	...	
0x42a81be9158	pWasmCompileLazy	
...	...	
0x42a81be96c0	Push 0x0	Lazy Compile Table
0x42a81be96c5	jmp 0x42a81be9150	
0x42a81be96ca	Push 0x1	
0x42a81be96cb	jmp 0x42a81be9150	
...	...	
0x42a81be9700	add BYTE PTR [rax],al	Start compiled code
0x42a81be9702	add BYTE PTR [rax],al	
...	...	

WebAssembly export functions call

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals



WASM RWX memory

0x42a81be9000	jmp 0x42a81be96c0	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9048	0x7ffff4b0b600	
...	...	
0x42a81be9150	jmp qword ptr[rip+0x2]	
...	...	
0x42a81be9158	pWasmCompileLazy	Lazy Compile Table
...	...	
0x42a81be96c0	Push 0x0	
0x42a81be96c5	jmp 0x42a81be9150	
0x42a81be96ca	Push 0x1	
0x42a81be96cb	jmp 0x42a81be9150	
...	...	
0x42a81be9700	add BYTE PTR [rax],al	Start compiled code
0x42a81be9701	add BYTE PTR [rax],al	
...	...	

WebAssembly export functions call

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
})();
```

V8 internals



WASM RWX memory

0x42a81be9000	jmp 0x42a81be96c0	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9048	0x7ffff4b0b600	
...	...	
0x42a81be9150	jmp qword ptr[rip+0x2]	
...	...	
0x42a81be9158	pWasmCompileLazy	Lazy Compile Table
...	...	
0x42a81be96c0	Push 0x0	
0x42a81be96c5	jmp 0x42a81be9150	
0x42a81be96ca	Push 0x1	
0x42a81be96cb	jmp 0x42a81be9150	
...	...	
0x42a81be9700	push rbp	Start compiled code
0x42a81be9701	mov rbp, rsp	
...	...	

WebAssembly export functions call

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
})();
```

V8 internals



WASM RWX memory

0x42a81be9000	jmp 0x42a81be9700	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9048	0x7ffff4b0b600	
...	...	
0x42a81be9150	jmp qword ptr[rip+0x2]	
...	...	
0x42a81be9158	pWasmCompileLazy	
...	...	
0x42a81be96c0	Push 0x0	Lazy Compile Table
0x42a81be96c5	jmp 0x42a81be9150	
0x42a81be96ca	Push 0x1	
0x42a81be96cb	jmp 0x42a81be9150	
...	...	
0x42a81be9700	push rbp	Start compiled code
0x42a81be9701	mov rbp, rsp	
...	...	

WebAssembly export functions optimization

JavaScript code execution

```
(async () => {
const response = await fetch('main.wasm');
const buffer = await response.arrayBuffer();
const module = new WebAssembly.Module(buffer);
const instance = new WebAssembly.Instance(module);
const result = instance.exports.main();
console.log(result);
for (let i = 0; i < 0x10000; i++){result;}
})();
```

V8 internal classes

```
class WasmExportedFunction : public JSFunction {
    SharedFunctionInfo shared_function_info;
    ...
}

class SharedFunctionInfo {
public:
    Tagged<String> Name;
    Tagged<Object> function_data;
    Tagged<HeapObject> script;
    ...
}

class WasmExportedFunctionData {
public:
    Tagged<WasmInternalFunction> internal;
    Tagged<Code> wrapper_code;
    Tagged<WasmInstanceObject> instance;
    int function_index;
    FunctionSig* signature;
    int wrapper_tiering_budget;
    ...
};
```

V8 internals

```
void Runtime_WasmCompileWrapper(...);
```

```
void ReplaceWrapper(Isolate* isolate,
Handle<WasmInstanceObject> instance, int
function_index, Handle<Code> wrapper_code);
```

WebAssembly export functions optimization

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
  for (let i = 0; i < 0x10000; i++) {result;}
})();
```

V8 internals

```
CompilationExecutionResult ExecuteCompilationUnits(...);

void CompilationStateImpl::SchedulePublishCompilationResults(...);

void CompilationStateImpl::PublishCompilationResults(...);

WasmCode* NativeModule::PublishCode(...)

WasmCode* NativeModule::PublishCodeLocked(
    std::unique_ptr<WasmCode> owned_code);

void NativeModule::PatchJumpTableLocked(const
    CodeSpaceData& code_space_data, uint32_t
    slot_index, Address target);
```

WASM RWX memory

0x42a81be9000	jmp 0x42a81be9700	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9048	0x7ffff4b0b600	
...	...	
0x42a81be96c0	Push 0x0	Lazy Compile Table
0x42a81be96c5	jmp 0x42a81be9150	
...	...	
0x42a81be9700	push rbp	Start compiled code
0x42a81be9701	mov rbp, rsp	
...	...	

Main Jump Table

Far Jump Table

Lazy Compile Table

Start compiled code

WebAssembly export functions optimization

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
  for (let i = 0; i < 0x10000; i++) {result;}
})();
```

V8 internals

```
CompilationExecutionResult ExecuteCompilationUnits(...);

void CompilationStateImpl::SchedulePublishCompilationResults(...);

void CompilationStateImpl::PublishCompilationResults(...);

WasmCode* NativeModule::PublishCode(...)

WasmCode* NativeModule::PublishCodeLocked(
    std::unique_ptr<WasmCode> owned_code);

void NativeModule::PatchJumpTableLocked(const
    CodeSpaceData& code_space_data, uint32_t
    slot_index, Address target);
```

WASM RWX memory

0x42a81be9000	jmp 0x42a81be9700	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9048	0x7ffff4b0b600	
...	...	
0x42a81be96c0	Push 0x0	Lazy Compile Table
0x42a81be96c5	jmp 0x42a81be9150	
...	...	
0x42a81be9700	push rbp	
0x42a81be9701	mov rbp, rsp	
...	...	
0x42a81be97c0	push rbp	WASM Compiled code
0x42a81be97c1	mov rbp, rsp	
0x42a81be97c4	push 0x8	
...	...	

Main Jump Table

Far Jump Table

Lazy Compile Table

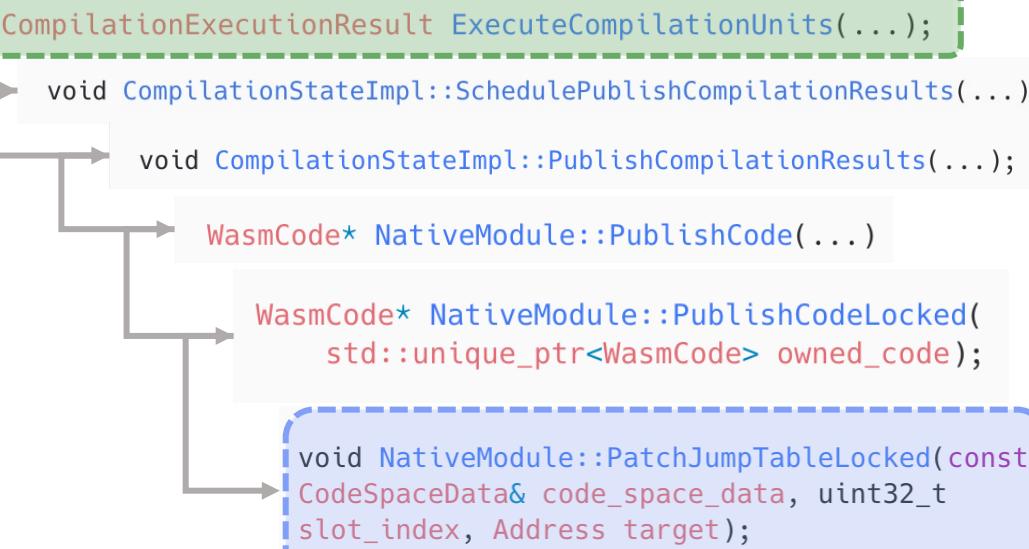
WASM Compiled code

WebAssembly export functions optimization

JavaScript code execution

```
(async () => {
  const response = await fetch('main.wasm');
  const buffer = await response.arrayBuffer();
  const module = new WebAssembly.Module(buffer);
  const instance = new WebAssembly.Instance(module);
  const result = instance.exports.main();
  console.log(result);
  for (let i = 0; i < 0x10000; i++) {result;}
})());
```

V8 internals



WASM RWX memory

0x42a81be9000	jmp 0x42a81be97c0	Main Jump Table
0x42a81be9005	jmp 0x42a81be96ca	
...	...	
0x42a81be9040	jmp qword ptr[rip+0x2]	Far Jump Table
...	...	
0x42a81be9048	0x7ffff4b0b600	
...	...	
0x42a81be96c0	Push 0x0	Lazy Compile Table
0x42a81be96c5	jmp 0x42a81be9150	
...	...	
0x42a81be9700	push rbp	
0x42a81be9701	mov rbp, rsp	
...	...	
0x42a81be97c0	push rbp	WASM Compiled code
0x42a81be97c1	mov rbp, rsp	
0x42a81be97c4	push 0x8	
...	...	

The memory dump shows a sequence of addresses and their corresponding assembly instructions. Addresses 0x42a81be9000 and 0x42a81be96c0 are highlighted with blue dashed boxes and labeled as part of the "Main Jump Table". Address 0x42a81be9040 is labeled as part of the "Far Jump Table". Address 0x42a81be96c0 is also part of the "Lazy Compile Table". The bottom section, starting at address 0x42a81be97c0, is highlighted with a green dashed box and labeled as "WASM Compiled code". Ellipses indicate more memory pages.

Main Jump Table

Far Jump Table

Lazy Compile Table

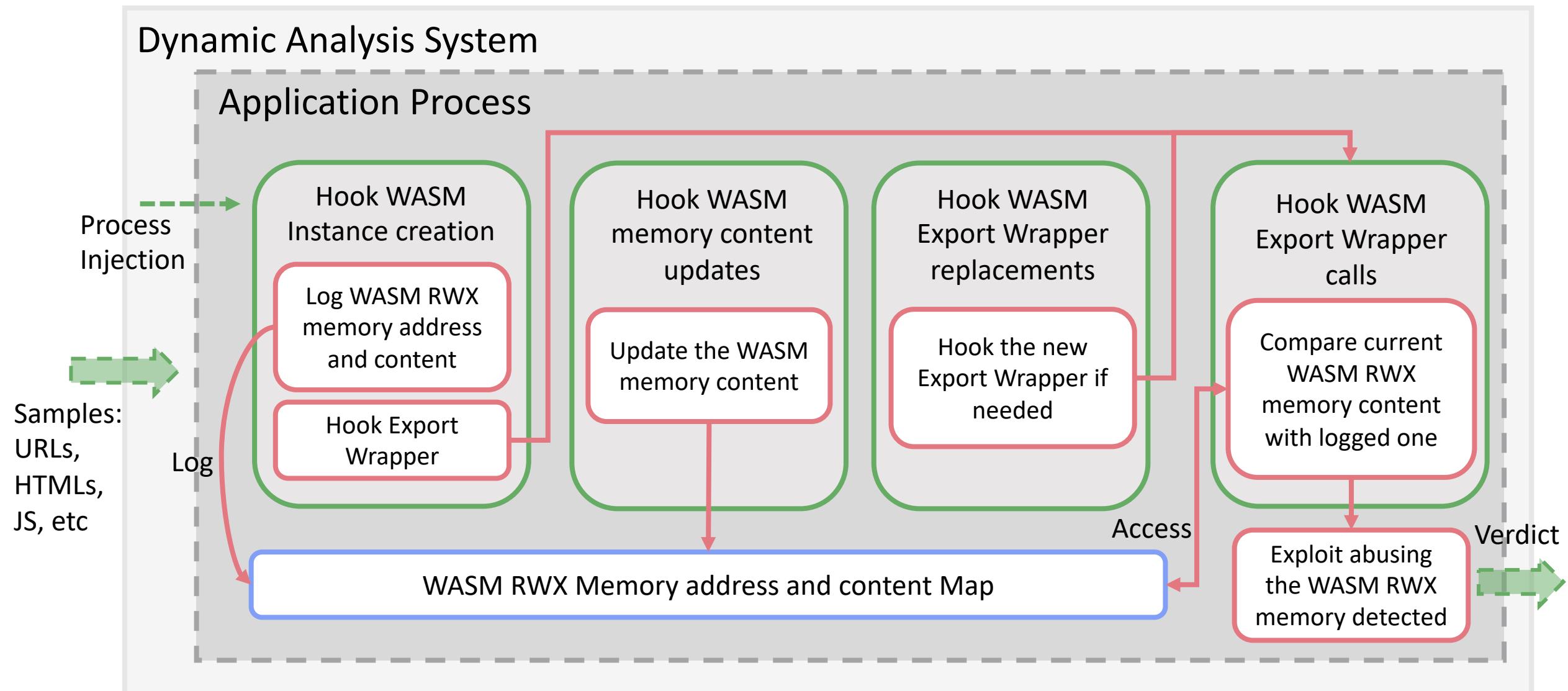
WASM Compiled code

WASMGuard

What is WASMGuard ?

- Hook-based system for runtime detection of browser exploits abusing WASM RWX memory based on 3 detection mechanisms :
 - Detecting illegal content changes in WebAssembly RWX memory by monitoring legitimate ones;
 - Checking the integrity of WebAssembly RWX memory;
 - Detecting shellcode in the WebAssembly RWX memory;
- These mechanisms can be used independently or together to maximize detection efficiency

Detecting illegal content changes by monitoring legitimate ones



Detecting illegal content changes by monitoring legitimate ones

Hook WASM
Instance creation

Log WASM RWX
memory address
and content

Hook Export
Wrapper

- Hooking `v8::internal::wasm::InstantiateToInstanceObject`;

```
void* hooked_InstantiateToInstanceObject(...) {
    ins = org_InstantiateToInstanceObject(...);
    rwx_mem_addr = ins->jmp_table_start;
    rwx_mem_cont = read_mem(rwx_mem_addr, ...);
    update_rwx_mem(rwx_mem_addr, rwx_mem_cont, rwx_mem_map_logged);
    ret = ins;
}
```

- Hooking `v8::internal::WasmExportedFunction::New`;

```
void* hooked_WasmExportedFunction_New(..., Code export_wrapper) {
    JSToWasmWrapper_addr = export_wrapper->instruction_start;
    if (not_hooked(JSToWasmWrapper_addr))
        hook_function(JSToWasmWrapper_addr);
    ret = org_WasmExportedFunction_New(..., export_wrapper);
}
```

Detecting illegal content changes by monitoring legitimate ones

Hook WASM
memory content
updates

Update the WASM
memory content

- Hooking `v8::internal::wasm::NativeModule::PublishCodeLocked;`

```
void* hooked_PublishCodeLocked(..., WasmCode wasm_code) {
    rwx_mem_addr = get_base_address(wasm_code->instruction_start);
    rwx_mem_cont = read_mem(rwx_mem_addr, ...);
    update_rwx_mem(rwx_mem_addr, rwx_mem_cont, rwx_mem_map_logged);
    ret = org_PublishCodeLocked(..., wasm_code);
}
```

- Hooking `v8::internal::wasm::NativeModule::PatchJumpTableLocked;`

```
void hooked_PatchJumpTableLocked(..., Address target) {
    rwx_mem_addr = get_base_address(target);
    rwx_mem_cont = read_mem(rwx_mem_addr, ...);
    update_rwx_mem(rwx_mem_addr, rwx_mem_cont, rwx_mem_map_logged);
    ret = org_PatchJumpTableLocked(..., target);
}
```

Detecting illegal content changes by monitoring legitimate ones

Hook WASM
Export Wrapper
replacements

Hook the new
Export Wrapper if
needed

- Hooking `v8::internal::(anonymous namespace)::ReplaceWrapper;`

```
void hooked_ReplaceWrapper(..., Code wrapper_code) {
    JSToWasmWrapper_addr = wrapper_code->instruction_start;
    if (not_hooked(JSToWasmWrapper_addr))
        hook_function(JSToWasmWrapper_addr);
    ret = org_ReplaceWrapper(..., wrapper_code);
}
```

Detecting illegal content changes by monitoring legitimate ones

Hook WASM
Export Wrapper
calls

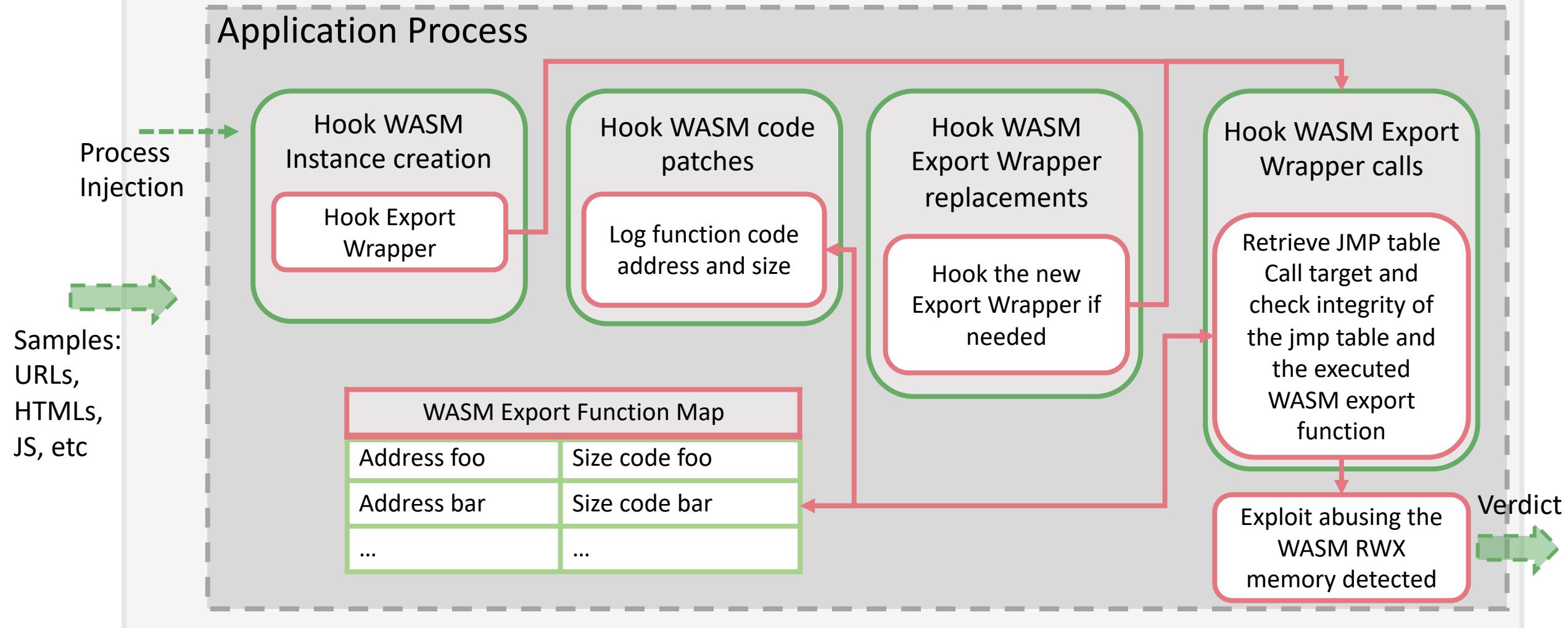
Compare current
WASM RWX
memory content
with logged one

- Hooking `Builtins_*JSToWasmWrapper`:

```
void hooked_JSToWasmWrapper(JSFunction js_func, ...) {
    rwx_mem_addr = js_func->shared_function_info->function_data->instance->jmp_table_start;
    rwx_mem_cont = read_mem(rwx_mem_addr, ...);
    verdict = compare_rwx_mem(rwx_mem_addr, rwx_mem_cont, rwx_mem_map_logged);
    if (verdict) output_result(...);
    else ret = org_JSToWasmWrapper(js_func, ...);
}
```

Checking the integrity of WebAssembly RWX memory

Dynamic Analysis System



Checking the integrity of WebAssembly RWX memory

Hook WASM code patches

Log function code address and size

- Hooking `v8::internal::wasm::NativeModule::PublishCodeLocked`:

```
void hooked_PublishCodeLocked(..., WasmCode wasm_code) {
    func_addr = wasm_code->instructions_ ;
    func_size = wasm_code->instructions_size_ ;
    update_export_functions_map(func_addr, func_size, export_functions_map);
    ret = org_PublishCodeLocked(wasm_code);
}
```

Checking the integrity of WebAssembly RWX memory

- Hooking `Builtins_*JSToWasmWrapper`;

```
void hooked_JSToWasmWrapper(JSFunction js_func, ...) {
    func_call_target = js_func->shared_function_info->function_data->internal_function->call_target;
    jmp_table_entry = get_jmp_table_entry_from_call_target(func_call_target);
    verdict = check_jmp_table_integrity(jmp_table_entry);
    if (verdict) output_result(...);
    else{
        func_addr = get_func_addr_from_jmp_table_entry(jmp_table_entry);
        verdict = check_func_addr_integrity(func_addr, export_functions_map);
        if (verdict) output_result(...);
        else{
            func_size = get_func_size(func_addr, export_functions_map);
            code = read_mem(func_addr, func_size);
            verdict = check_code_integrity(code);
            if (verdict) output_result(...);
            else ret = org_JSToWasmWrapper(js_func, ...);
        }
    }
}
```

Hook WASM Export Wrapper calls

Retrieve JMP table Call target and check integrity of the jmp table and the executed WASM export function

Checking the integrity of the first call to a WASM export function

```
0:000> u 1529a8c51000 110          Call target
00001529`a8c51000 e9bb080000      jmp   00001529`a8c518c0
00001529`a8c51005 e9b6070000      jmp   00001529`a8c517c0
00001529`a8c5100a e971060000      jmp   00001529`a8c51680
00001529`a8c5100f cc             int   3
.
.
```

- Check the integrity of the jmp instruction;
- Check that the address to jump to is pointing to lazy compile table;

```
0:000> u 00006e8a`5cb0168a 110
00006e8a`5cb0168a 6801000000      push  1
00006e8a`5cb0168f e9acfaffff      jmp   00006e8a`5cb01140
00006e8a`5cb01694 6802000000      push  2
00006e8a`5cb01699 e9a2faffff      jmp   00006e8a`5cb01140
```

- Check the integrity of the push and jmp instructions;
 - Check that the address to jump to is pointing to the far jump table;

```
0:000> u 00006e8a`5cb01140
00006e8a`5cb01140 ff2502000000      jmp   qword ptr [00006e8a`5cb01148]
00006e8a`5cb01146 6690              xchg  ax,ax
```

- Check the integrity of the jmp instruction;
- Check that the address to jump to is the address of the WasmCompileLazy function;

```
0:000> dq 00006e8a`5cb01148 11
00006e8a`5cb01148 00007ffa`e665f740
0:000> u 00007ffa`e665f740 110
chrome!Builtins_WasmCompileLazy:
00007ffa`e665f740 415f              pop   r15
```

Checking the integrity of the call to a Liftoff compiled WASM export function

The screenshot shows assembly code from a debugger. A red dashed box highlights the **Prologue** section, which consists of several `push`, `mov`, and `sub` instructions. A blue dashed box highlights a `jbe` instruction that jumps to address `00006e8a`5cb01851`. A green dashed box highlights a `call` instruction at address `00006e8a`5cb012c0`. A green arrow points from this call site to a `jmp` instruction at address `00006e8a`5cb012c8`, which then branches to a `dq` instruction at address `00006e8a`e66f9100`. Another green arrow points from this `dq` instruction to a `u` instruction at address `00007ffa`e66f9100`. A final green arrow points from this `u` instruction to the `chrome!Builtins_WasmStackGuard:` label at address `00007ffa`e66f9103`. The assembly code also includes `mov` and `sub` instructions throughout.

```
00006e8a`5cb017c0 55      push    rbp
00006e8a`5cb017c1 4889e5  mov     rbp, rsp
00006e8a`5cb017c4 6a08  push    8
00006e8a`5cb017c6 56      push    rsi
00006e8a`5cb017c7 488lec20000000 sub    rbp, 20h
00006e8a`5cb017ce 488b4e2f  mov     rcx, qword ptr [rsi+2Fh]
00006e8a`5cb017d2 483b21  cmp     rbp, qword ptr [rcx]
00006e8a`5cb017d5 0f8676000000 jbe    00006e8a`5cb01851
00006e8a`5cb017db 8bc8  mov     ecx, eax
00006e8a`5cb017dd 83f802  cmp     eax, 2
00006e8a`5cb017e0 0f8c05000000 jl    00006e8a`5cb017eb
00006e8a`5cb017e6 e91d000000 jmp    00006e8a`5cb01800
00006e8a`5cb017eb 488b8697000000 mov    rax, qword ptr [rsi+97h]
00006e8a`5cb017f2 8b5004  mov    edx, dword ptr [rax+4]
00006e8a`5cb017f5 83ea2b  sub    edx, 2Bh
00006e8a`5cb017f8 0f8863000000 js    00006e8a`5cb01861
00006e8a`5cb017fe 895004  mov    dword ptr [rax+4], edx
00006e8a`5cb01801 8bc1  mov    eax, ecx

0:000> u 00006e8a`5cb01851 12
00006e8a`5cb01851 50      push    rax
00006e8a`5cb01852 e869faffff call   00006e8a`5cb012c0
0:000> u 00006e8a`5cb012c0 11
00006e8a`5cb012c0 ff2502000000 jmp    qword ptr [00006e8a`5cb012c8]
0:000> dq 00006e8a`5cb012c8 11
00006e8a`5cb012c8 00007ffa`e66f9100
0:000> u 00007ffa`e66f9100 12
chrome!Builtins_WasmStackGuard:
00007ffa`e66f9100 488bd5  mov    rdx, rbp
00007ffa`e66f9103 488b52f0  mov    rdx, qword ptr [rdx-10h]
```

- Checking the integrity of Liftoff **prologue** instructions;
- Checking the V8 internal structure usages:
 - `rsi = WasmInstanceObject`
 - `rsi+0x2f = WasmInstanceObject->stack_limit_address`
 - `rsi+0x97 = WasmInstanceObject->tiering_budget_array`
- Checking the V8 internal function calls:
 - `WasmStackGuard`;
 - `WasmTriggerTierUp`;

Checking the integrity of the call to a Turbofan compiled WASM export function

- We apply the **same logic as Liftoff integrity checks** for Turbofan JIT compiled code BUT:
 - Turbofan can **get rid of the prologue**;
 - Turbofan can **get rid of the structures' access**;
 - Turbofan can **get rid of the internal function calls**;
- It doesn't really matter since it never happens in any existing exploit.
- But if the adversary knows our method and try to call a Turbofan compiled WASM export function to bypass our detection, we still could compute **function content hash** to ensure the integrity of the code.

Checking the integrity of the call to a Turbofan compiled WASM export function

Example of JIT compiled WASM code implementing a simple addition:

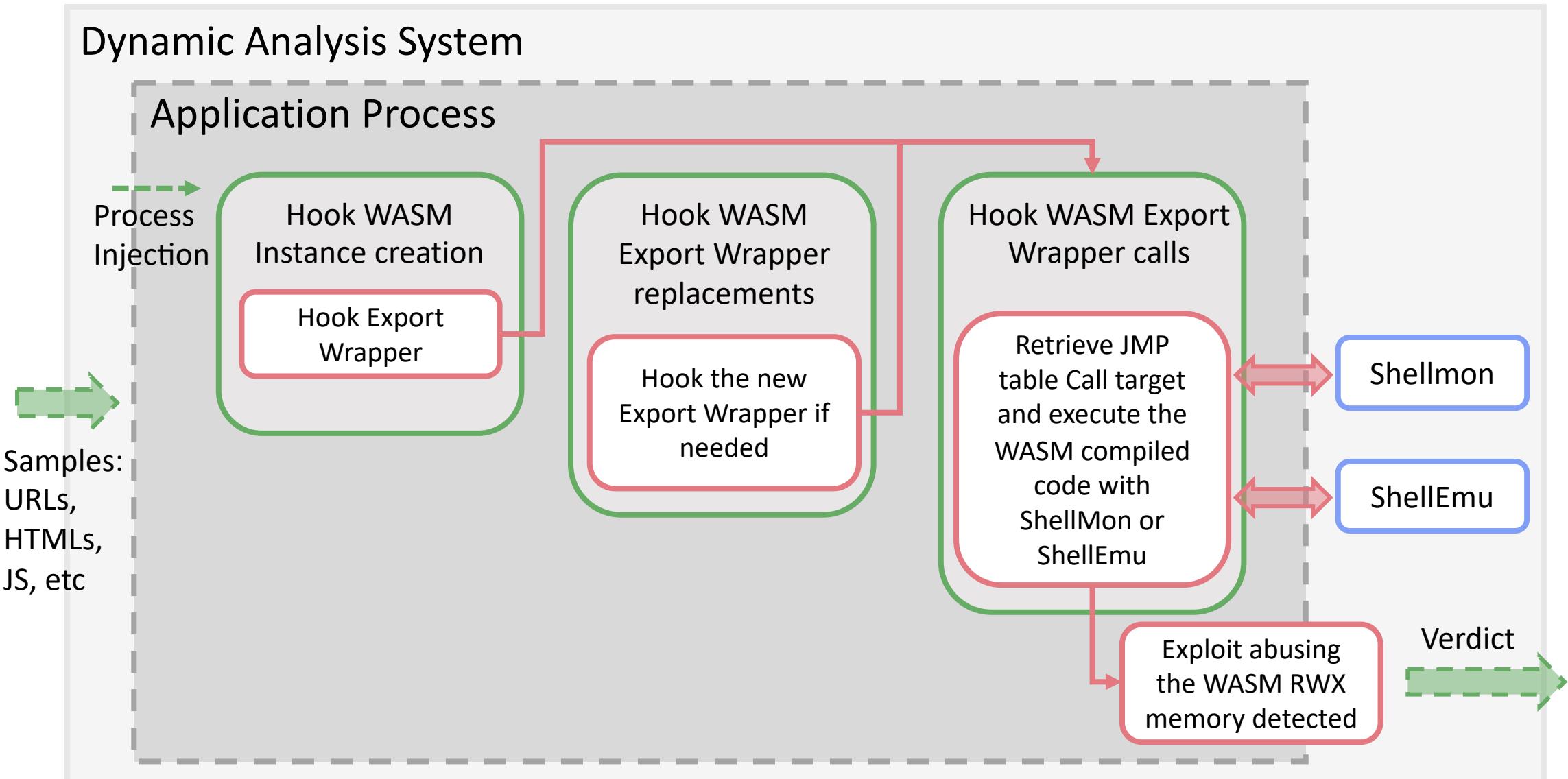
```
00005fab`459e16c0 55          push    rbp
00005fab`459e16c1 4889e5      mov     rbp,rsp
00005fab`459e16c4 6a08        push    8
00005fab`459e16c6 56          push    rsi
00005fab`459e16c7 4881ec10000000 sub    rsp,10h
00005fab`459e16ce 488b4e2f      mov    rcx,qword ptr [rsi+2Fh]
00005fab`459e16d2 483b21        cmp    rsp,qword ptr [rcx]
00005fab`459e16d5 0f8620000000 jbe    00005fab`459e16fb
00005fab`459e16db 8d0c02        lea    ecx,[rdx+rax]
00005fab`459e16de 488b9e97000000 mov    rbx,qword ptr [rsi+97h]
00005fab`459e16e5 8b7b08        mov    edi,dword ptr [rbx+8]
00005fab`459e16e8 83efle        sub    edi,1Eh
00005fab`459e16eb 0f8819000000 js    00005fab`459e170a
00005fab`459e16f1 897b08        mov    dword ptr [rbx+8],edi
00005fab`459e16f4 8bc1          mov    eax,ecx
00005fab`459e16f6 488be5        mov    rbp,rbp
00005fab`459e16f9 5d          pop    rbp
00005fab`459e16fa c3          ret
```



```
00005fab`459e1740 55          push    rbp
00005fab`459e1741 4889e5      mov     rbp,rsp
00005fab`459e1744 6a08        push    8
00005fab`459e1746 56          push    rsi
00005fab`459e1747 03c2        add    eax,edx
00005fab`459e1749 488be5      mov    rbp,rbp
00005fab`459e174c 5d          pop    rbp
00005fab`459e174d c3          ret
```

```
push    rbp
mov    rbp,rsp
push    8
push    rsi
add    eax,edx
mov    rbp,rbp
pop    rbp
ret
```

Detecting shellcode in the WASM RWX memory region



Detecting shellcode in the WASM RWX memory region

Hook WASM Export Wrapper calls

Retrieve JMP table Call target and execute the WASM compiled code with ShellMon or ShellEmu

- Hooking `Builtins_*JSToWasmWrapper`;

```
void hooked_JSToWasmWrapper(JSFunction js_func, ...) {
    func_call_target = js_func->shared_function_info->function_data->internal_function->call_target;
    rwx_mem_addr = get_mem_region_base_address(func_call_target);
    rwx_mem_size = get_mem_region_size(rwx_mem_addr);
    code = read_mem(func_call_target, (rwx_mem_size-(func_call_target-rwx_mem_addr)));
    sm_verdict = shellmon(code);
    se_verdict = shellemu(code);
    if (sm_verdict or se_verdict) output_result(...);
    else ret = org_JSToWasmWrapper(js_func, ...);
}
```

ShellMon

- ShellMon is a shellcode detection system taking as an input a piece of bytecode, executing it in the memory and monitoring its behaviors in a lightweight sandbox.
- Main features:
 - Determine shellcode entry point;
 - Use its fast and minimalist hook engine to trace sensible Windows API calls;

ShellEmu

- Shellcode detection tool based on Qiling emulation Engine.
- Main features:
 - ASM level detection:
 - TEB/PEB Structures access
 - DLLs access: export table, name table, function table, etc.
 - GetPC techniques: Call/Pop, FPU instructions, x64 LEA/Relative addressing, etc.
 - Self decoding
 - API level detection:
 - Sensible Windows API calls

Demo



Summary

- Three exploitation detection mechanisms focusing the WASM RWX memory in WASMGuard.
- The ideas also work for other RWX memory focused DEP/NX bypass techniques.
- The exploitation technique focused exploit detection mechanism rocks!
- A lot more research focusing on the detection of exploitation techniques are needed.

Q&A

References

- [1] WebAssembly: Past and Future (Ben Titzer, Google, V8 WebAssembly team lead) 2019 :
<https://youtu.be/nRArrwEjccI?si=wgN0xcXALC28chRh>
- [2] The Hat Trick: Exploit Chrome Twice from Runtime to JIT (Nan Wang, Zhenghang Xiao) 2023:
<https://www.blackhat.com/us-23/briefings/schedule/#the-hat-trick-exploit-chrome-twice-from-runtime-to-jit-31557>
- [3] Loading WebAssembly modules efficiently (Mathias Bynens) 2018: <https://web.dev/loading-wasm/>