ESORICS

浙江大学 Zhejiang University

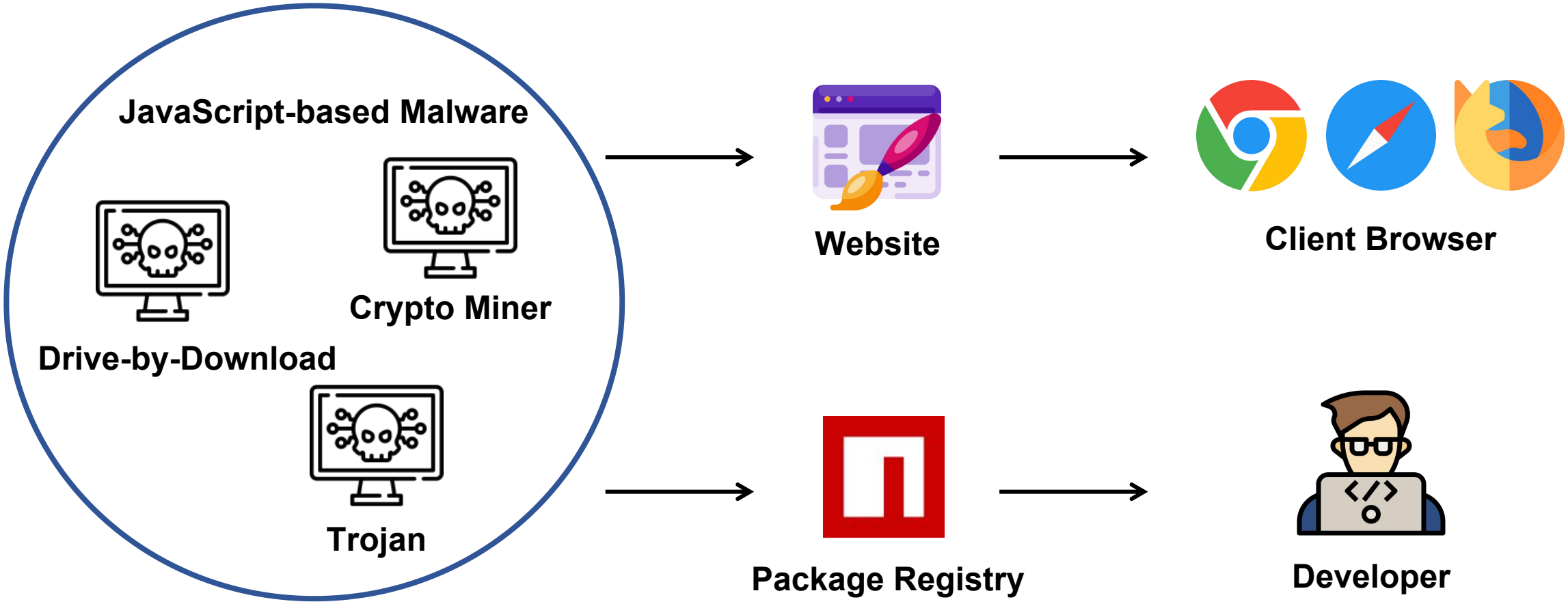
Static Semantics Reconstruction for Enhancing JavaScript-WebAssembly Multilingual Malware Detection

Yifan Xia, Ping He, Xuhong Zhang, Peiyu Liu, Shouling Ji, Wenhai Wang

Zhejiang University

Hangzhou, China

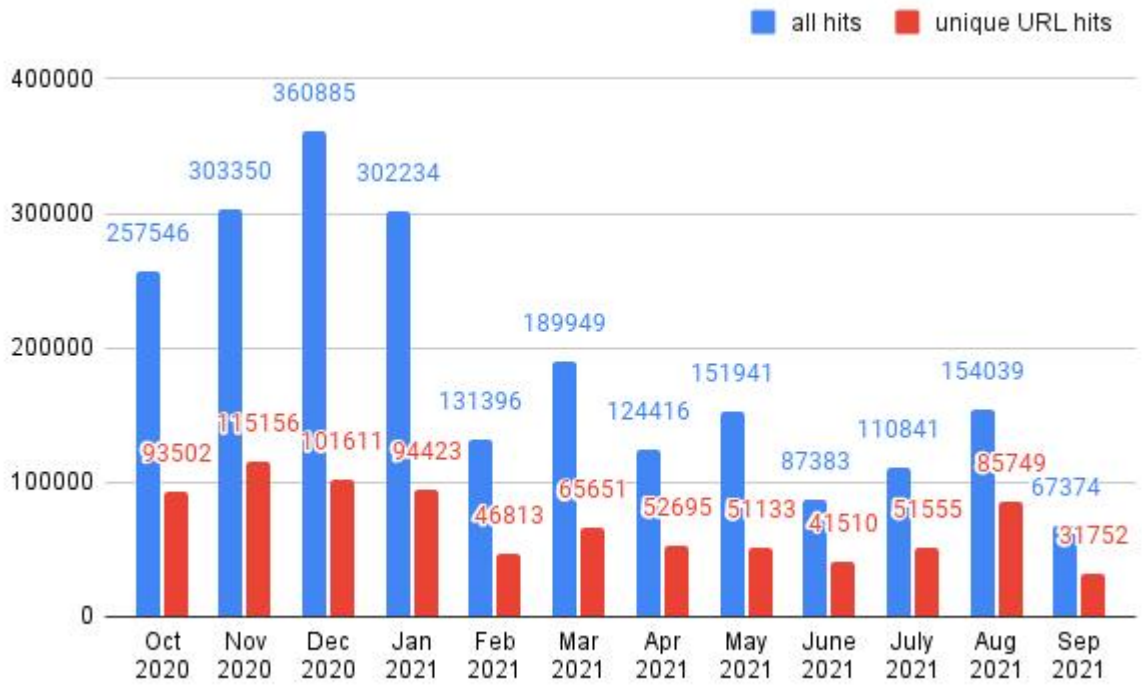
The Threat of JavaScript-based Malware



The Threat of JavaScript-based Malware

Zhejiang University

More than 2 million malicious websites powered by JavaScript were detected during October 2020-September 2021 [1].

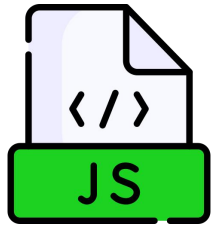


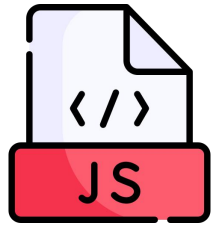
[1] https://unit42.paloaltonetworks.com/web-threats-trends-web-skimmers/

Anti-Virus Solutions for JavaScript-based Malware

Zhejiang University

Benign Samples

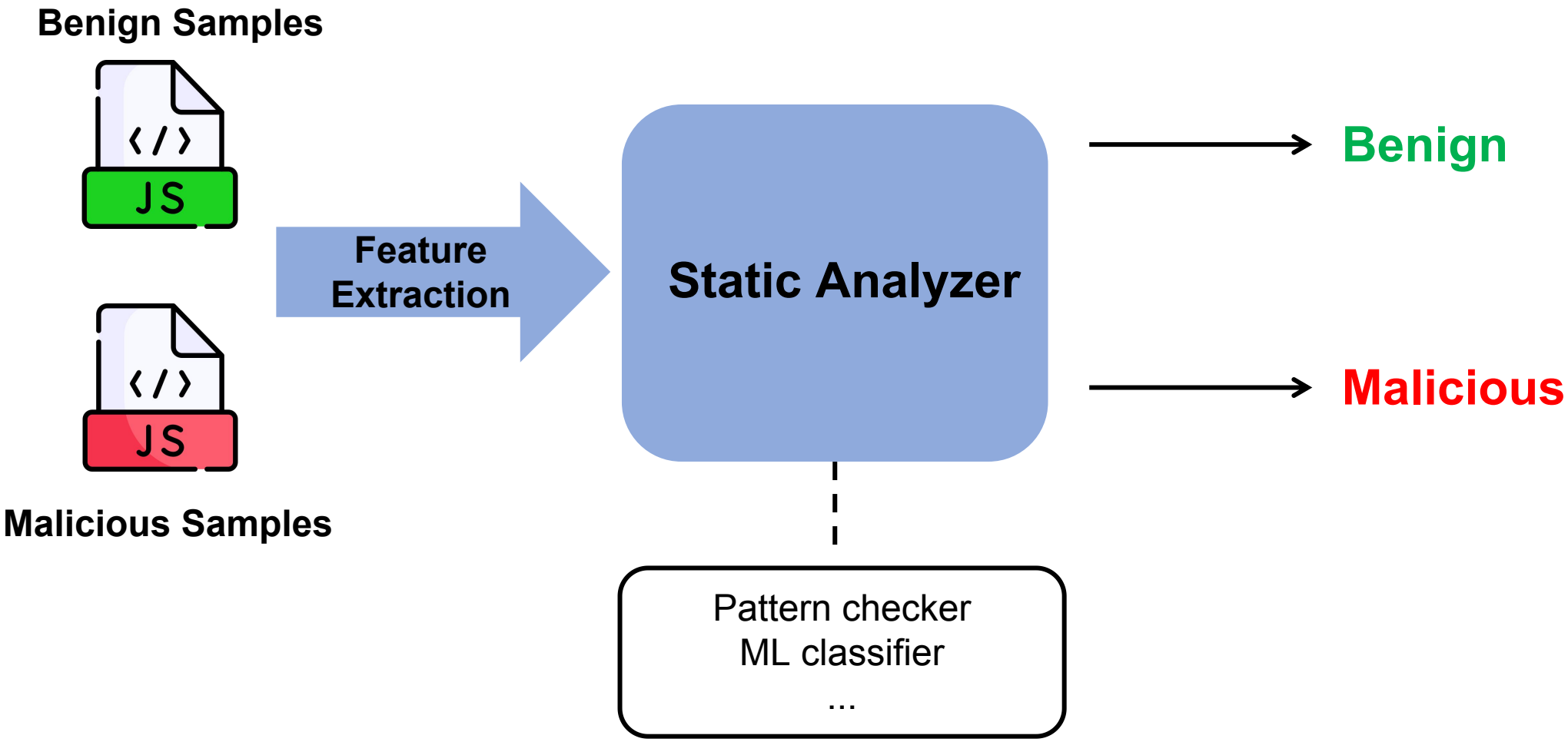




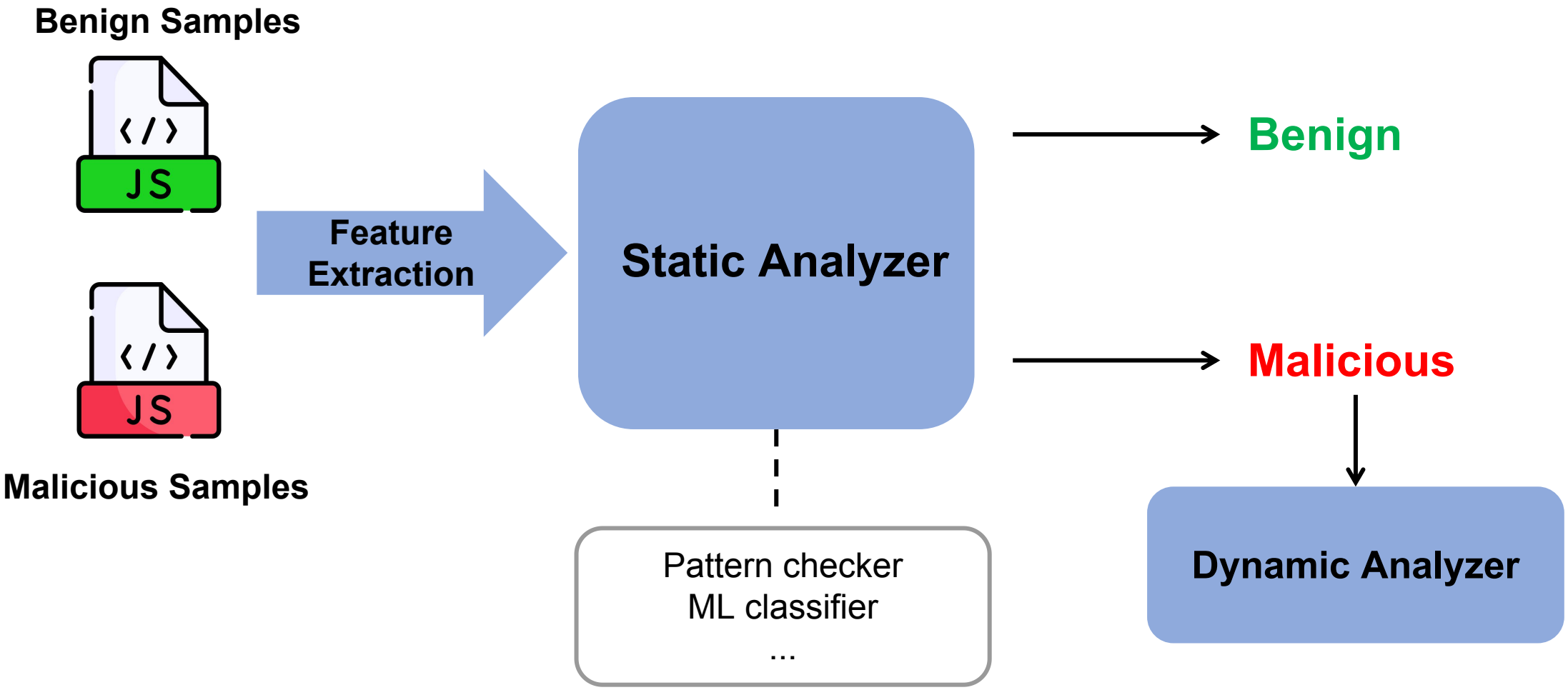
Malicious Samples

Feature Extraction

Anti-Virus Solutions for JavaScript-based Malware

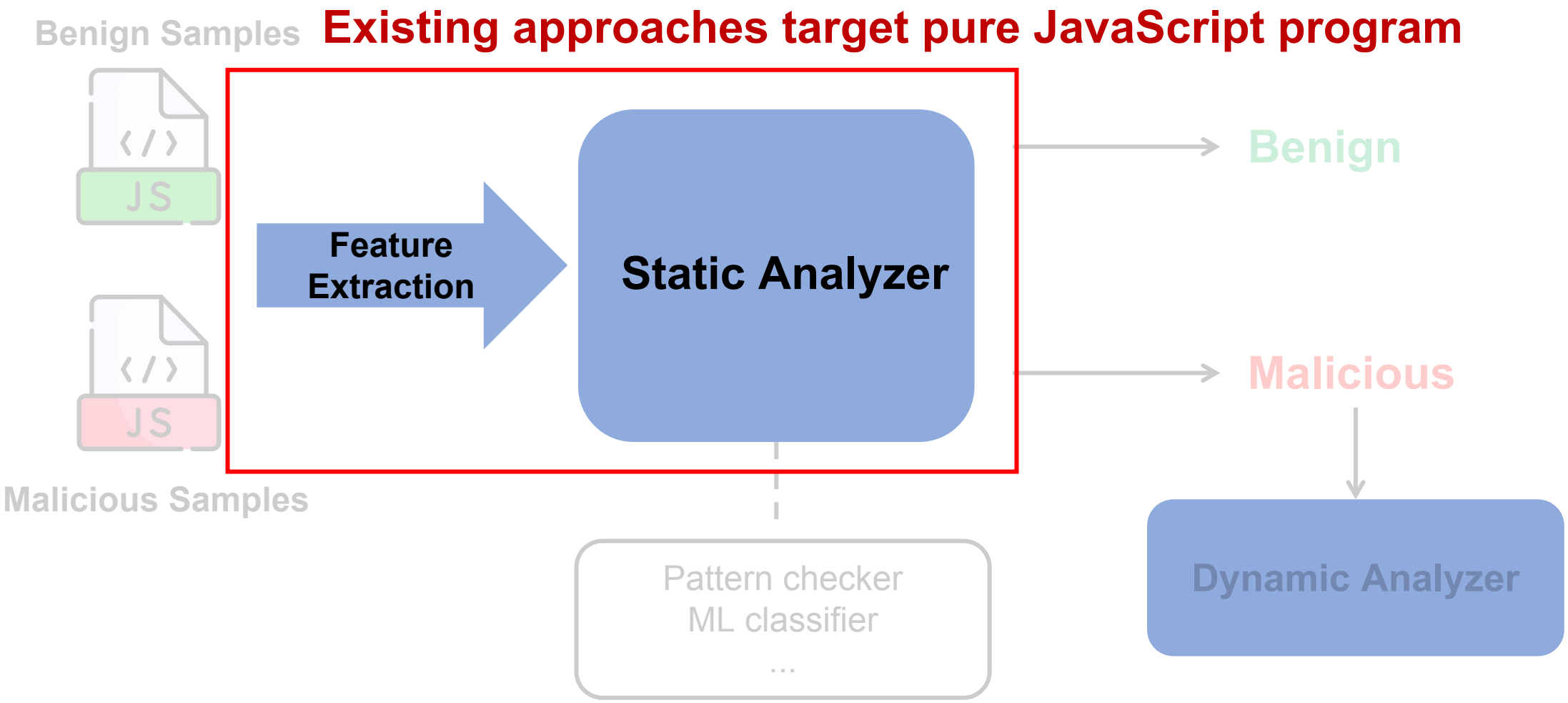


Anti-Virus Solutions for JavaScript-based Malware



Anti-Virus Solutions for JavaScript-based Malware

Zhejiang University



Anti-Virus Solutions for JavaScript-based Malware

Zhejiang University

Benign Samples

Existing approaches target pure JavaScript program

Feature

Benign

The introduction of WebAssembly breaks this assumption

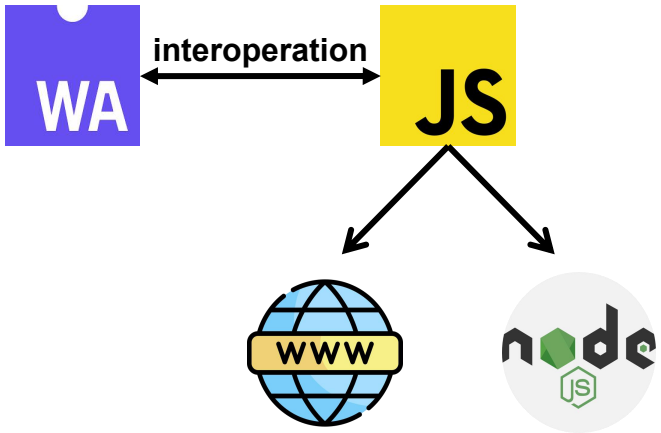
Pattern checker
ML classifier
...

Dynamic Analyzer

Background: WebAssembly

Zhejiang University

WebAssembly: A new client-side language in JavaScript ecosystem

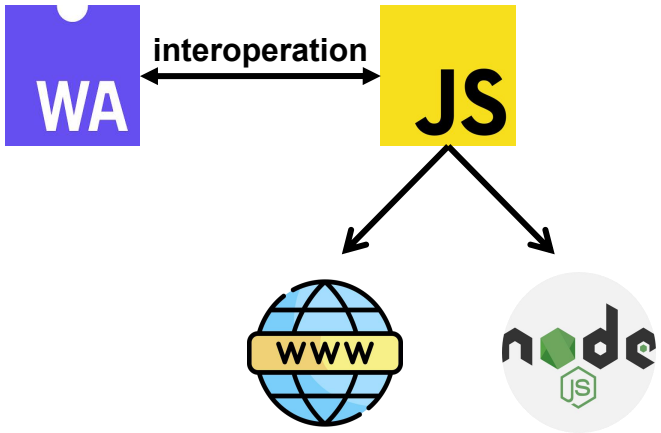


Major Platforms Support

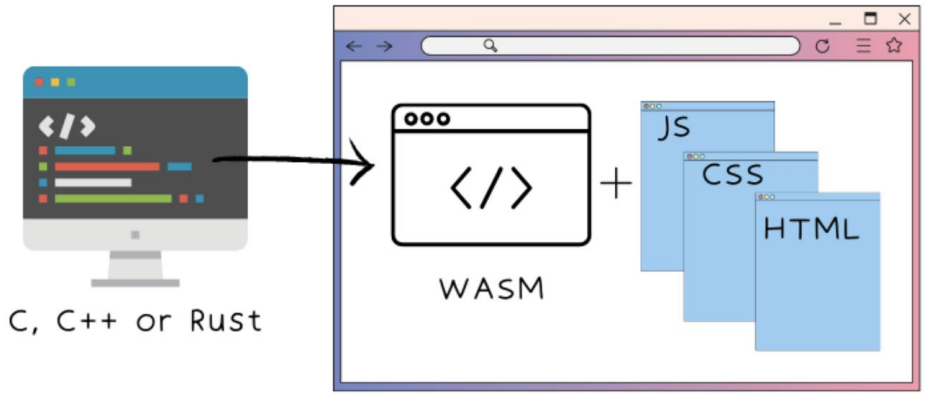
Background: WebAssembly

Zhejiang University

WebAssembly: A new client-side language in JavaScript ecosystem



Major Platforms Support

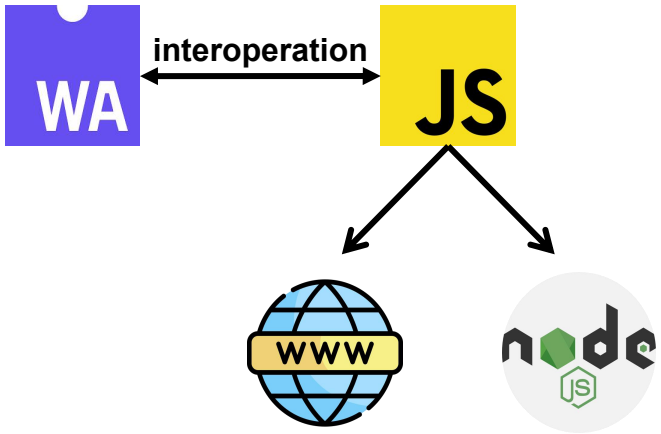


Compilation Targets for Other Languages

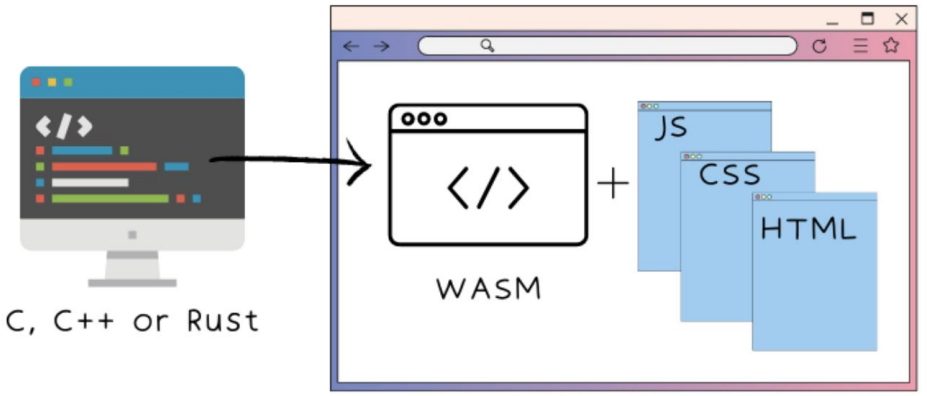
Background: WebAssembly

Zhejiang University

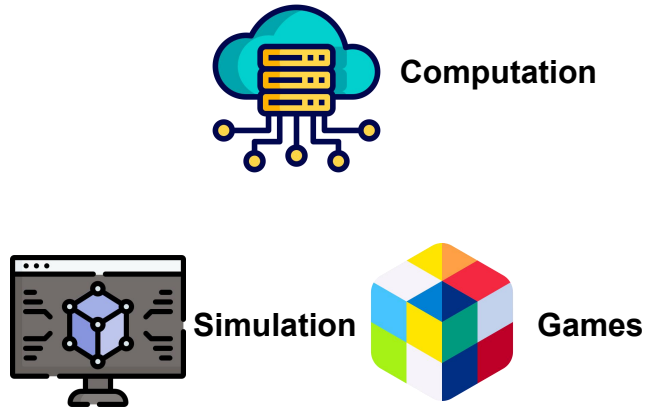
WebAssembly: A new client-side language in JavaScript ecosystem



Major Platforms Support



Compilation Targets for Other Languages

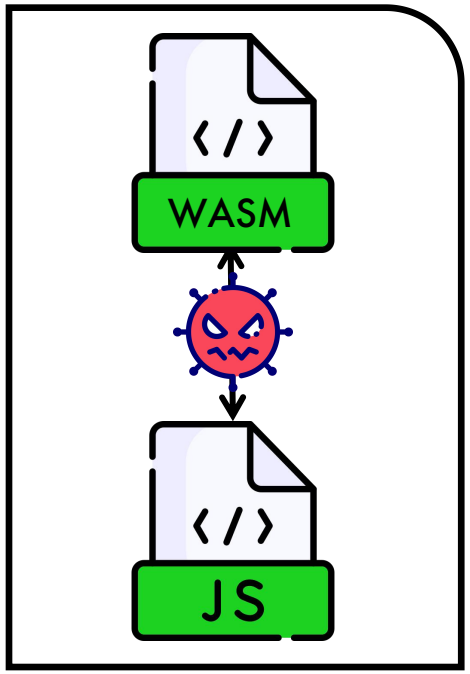


High Performance

Threats of WebAssembly

Zhejiang University

Hiding malicious behaviors in cross-language interoperations -> JWMM.

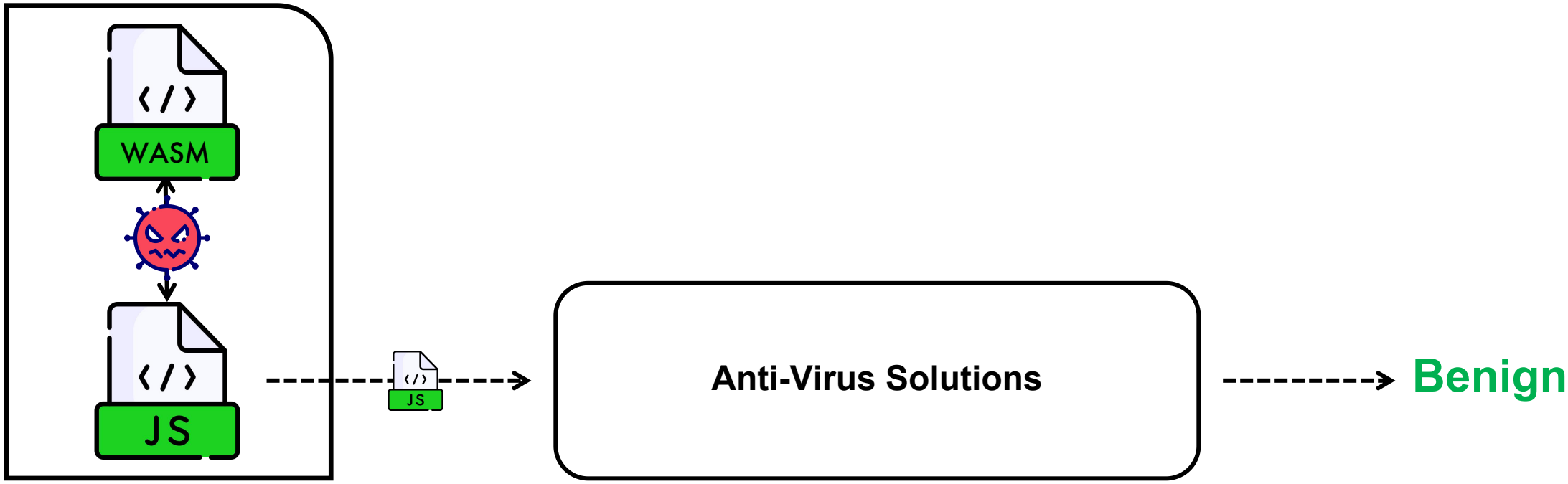


JS-WASM Multilingual Malware (JWMM)

Threats of WebAssembly

Zhejiang University

Hiding malicious behaviors in cross-language interoperations -> JWMM.

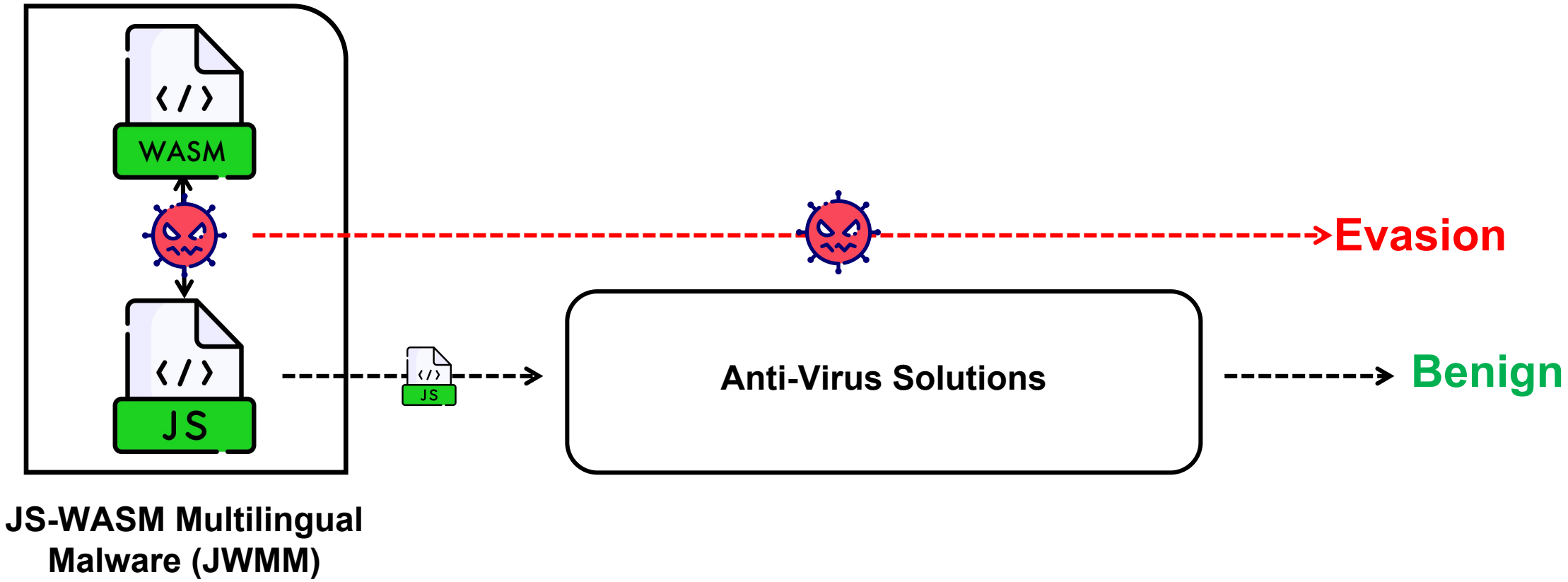


JS-WASM Multilingual Malware (JWMM)

Threats of WebAssembly

Zhejiang University

Hiding malicious behaviors in cross-language interoperations -> JWMM.



Motivation Example

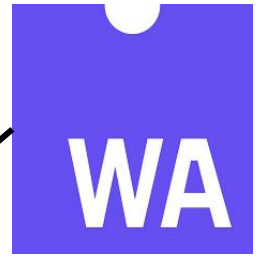

```
window.onload = function(){
    var strings = [];
    strings[0] = "<script>";
    strings[1] = "malicious payloads";
    strings[2] = "</script>";
    for (let i = 0; i < 3; i++){
        document.write(strings[i])
    }
}
```

Motivation Example

Zhejiang University

How does JWMM evade detection?

```
window.onload = function(){
    var strings = [];
    strings[0] = "<script>";
    strings[1] = "malicious payloads";
    strings[2] = "</script>";
    for (let i = 0; i < 3; i++){
        document.write(strings[i])
    }
}
```



WASM Binary Data

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
    }
```

Motivation Example

Conceals strings into WebAssembly

```
window.onload = function(){
    var strings = [];
    strings[0] = "<script>";
    strings[1] = "malicious payloads";
    strings[2] = "</script>";
    for (let i = 0; i < 3; i++){
        document.write(strings[i])
    }
}
```

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
    }
```

```
(data (i32.const 0) "<script>")
(data (i32.const 100) "malicious payloads")
(data (i32.const 200) "<script>")
```

Motivation Example

Zhejiang University

Conceals syntax structures into WebAssembly

```
window.onload = function(){
    var strings = [];
    strings[0] = "<script>";
    strings[1] = "malicious payloads";
    strings[2] = "</script>";
    for (let i = 0; i < 3; i++){
        document.write(strings[i])
    }
}
```

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
    }
```

```
(data (i32.const 0) "<script>")
(data (i32.const 100) "malicious payloads")
(data (i32.const 200) "<script>")
(func $foo (type 0) (local i32)
    ...
    loop
        local.get 0
        load
        call $env.impFunc
        ...
        br 0
    end
    )
```

Motivation Example

Zhejiang University

Conceals function call with WebAssembly

```
window.onload = function(){
    var strings = [];
    strings[0] = "<script>";
    strings[1] = "malicious payloads";
    strings[2] = "</script>";
    for (let i = 0; i < 3; i++){
        document.write(strings[i])
    }
}
```

1

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
    var importObject =
        {env:{impFunc: document.write}});
    }
```

2

3

```
(import "env" "impFunc")
(data (i32.const 0) "<script>")
(data (i32.const 100) "malicious payloads")
(data (i32.const 200) "<script>")
(func $foo (type 0) (local i32)
    ...
    loop
        local.get 0
        load
        call $env.impFunc
        ...
        br 0
    end
)
```

4

Motivation Example

Zhejiang University

Executes WebAssembly functions

```
window.onload = function(){
    var strings = [];
    strings[0] = "<script>";
    strings[1] = "malicious payloads";
    strings[2] = "</script>";
    for (let i = 0; i < 3; i++){
        document.write(strings[i])
    }
}
```

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
        var importObject =
        {env:{impFunc: document.write}});
        var instance =
        WebAssembly.instance(module, importObject);
        if (instance)
            instance.exports.foo();
    }
```

```
(import "env" "impFunc")
(data (i32.const 0) "<script>")
(data (i32.const 100) "malicious payloads")
(data (i32.const 200) "<script>")
(func $foo (type 0) (local i32)
    ...
    loop
        local.get 0
        load
        call $env.impFunc
        ...
        br 0
    end
    )
(export "foo" (func $foo))
```

Exports *foo()* to JavaScript unit

Motivation Example

Zhejiang University

The original semantic infomation is invisible

```
window.onload = function(){
    var strings = [];
    strings[0] = "<script>";
    strings[1] = "malicious payloads";
    strings[2] = "</script>";
    for (let i = 0; i < 3; i++){
        document.write(strings[i])
    }
}
```

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
        var importObject =
        {env:{impFunc: document.write}});
        var instance =
        WebAssembly.instance(module, importObject);
        if (instance)
            instance.exports.foo();
    }
```

```
(import "env" "impFunc")
(data (i32.const 0) "<script>")
(data (i32.const 100) "malicious payloads")
(data (i32.const 200) "<script>")
(func $foo (type 0) (local i32)
    ...
    loop
        local.get 0
        load
        call $env.impFunc
        ...
        br 0
    end
     )
(export "foo" (func $foo))
```

Challenges of Detecting JWMM

Zhejiang University

➢ Interoperation Complexity

- **Previous approachs:**

  Blind to cross-language interoperations

Challenges of Detecting JWMM

Zhejiang University

➢ Interoperation Complexity

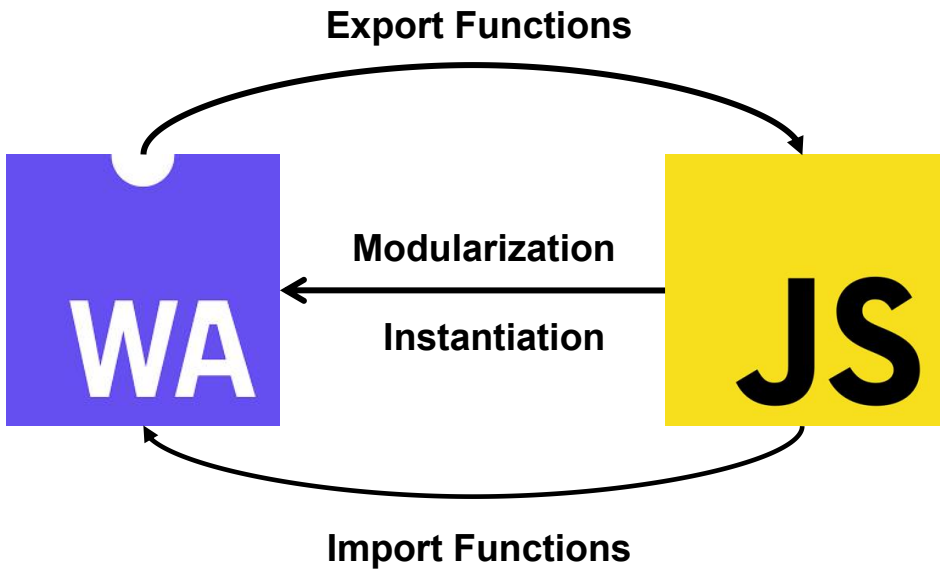
- ➢ **Previous approachs:**

  Blind to cross-language interoperations

- ➢ **Interoperation-aware analysis:**

  Various interfaces,

  Hidden cross-language dependencies



Challenges of Detecting JWMM

Zhejiang University

➢ Semantics Diversity

- ➢ **Previous approachs:**

  Patterns/Features for monolingual malware

Challenges of Detecting JWMM

Zhejiang University

➢ Semantics Diversity

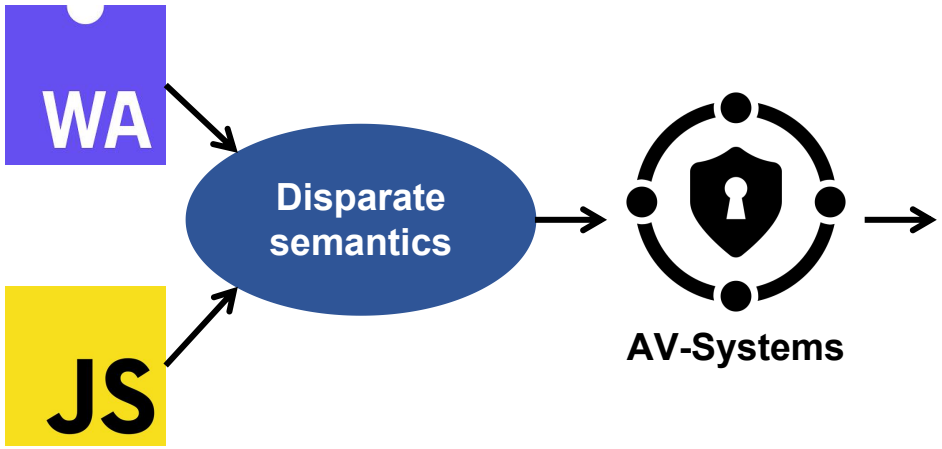
- ➢ **Previous approachs:**

  Patterns/Features for monolingual malware

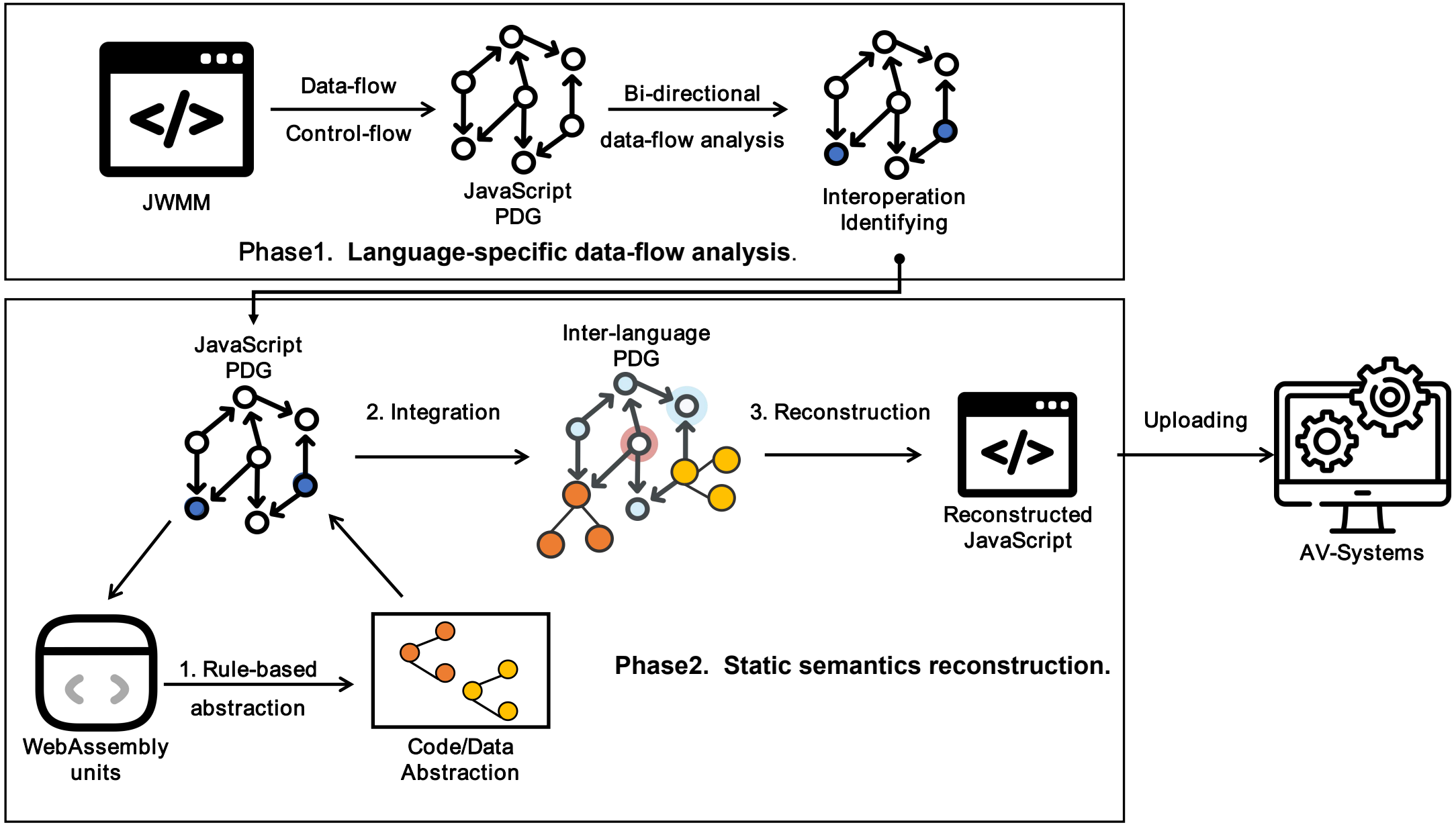
- ➢ **Multilingual program characterization:**

  Consideration of disparate semantics,

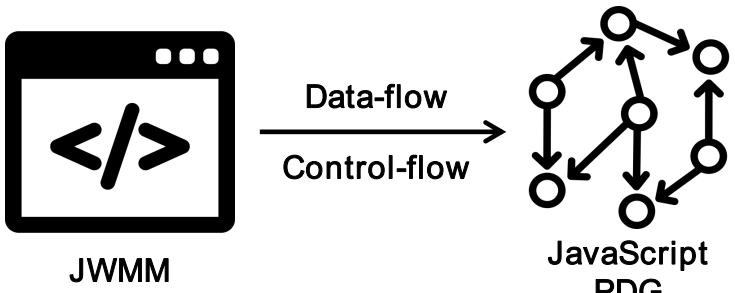
  Hard definition of multilingual malicious patterns



JWBinder Overview

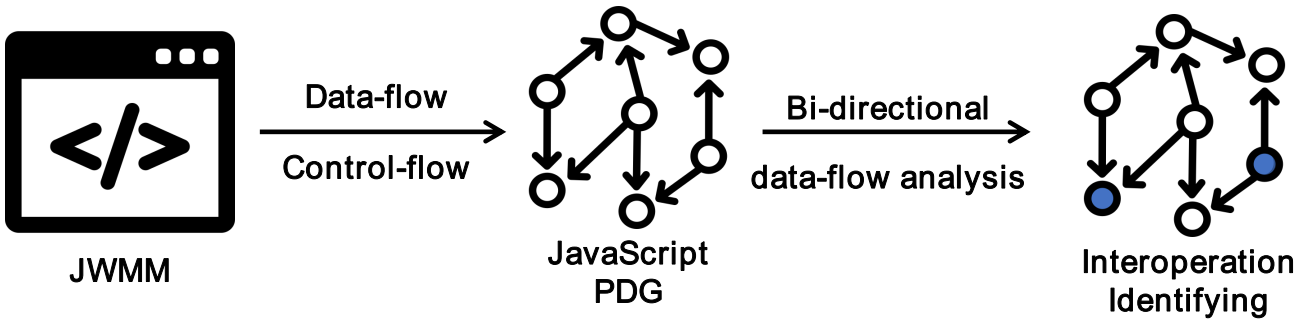


JWBinder Overview



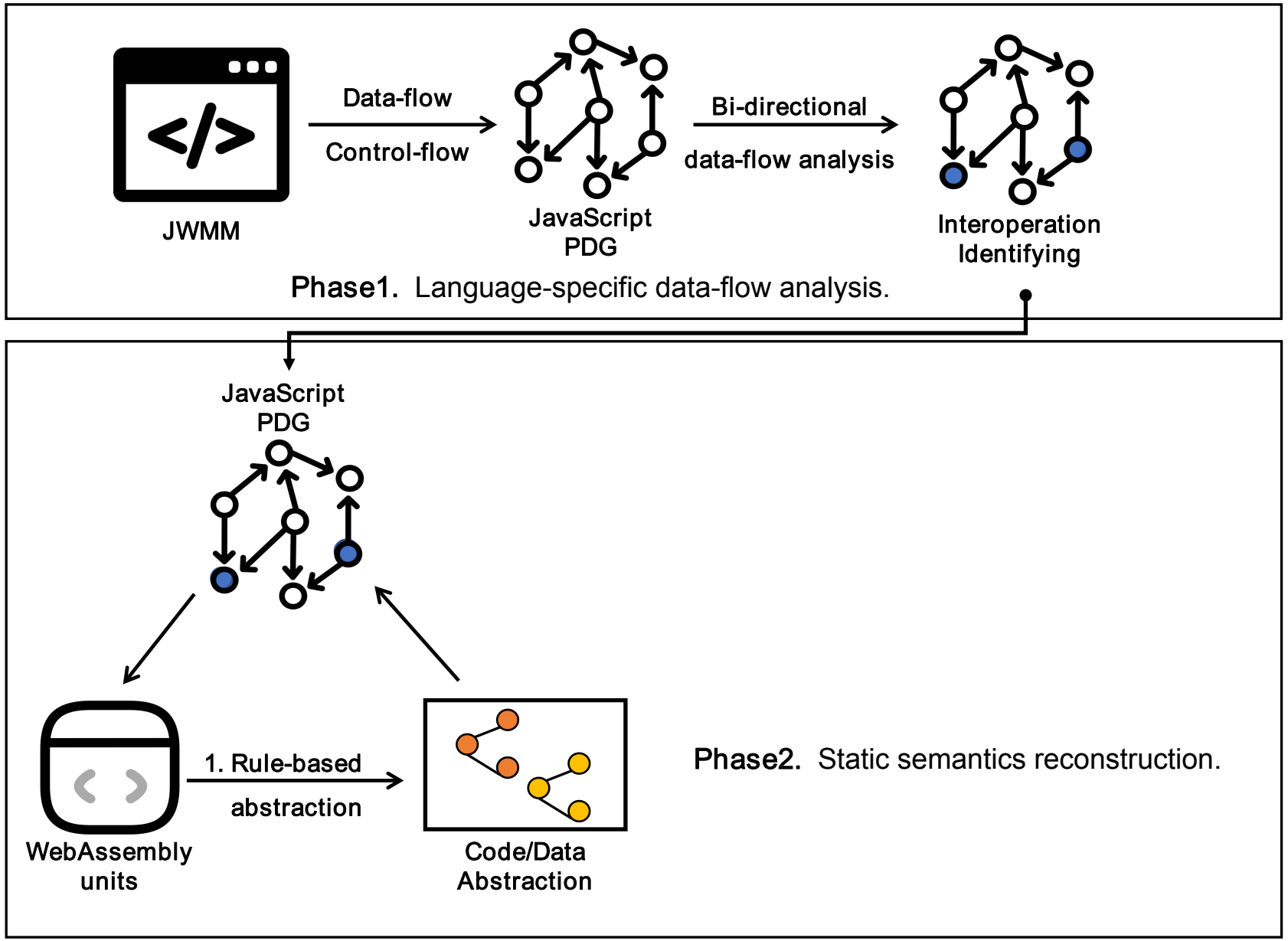
Phase1. Language-specific data-flow analysis.

JWBinder Overview

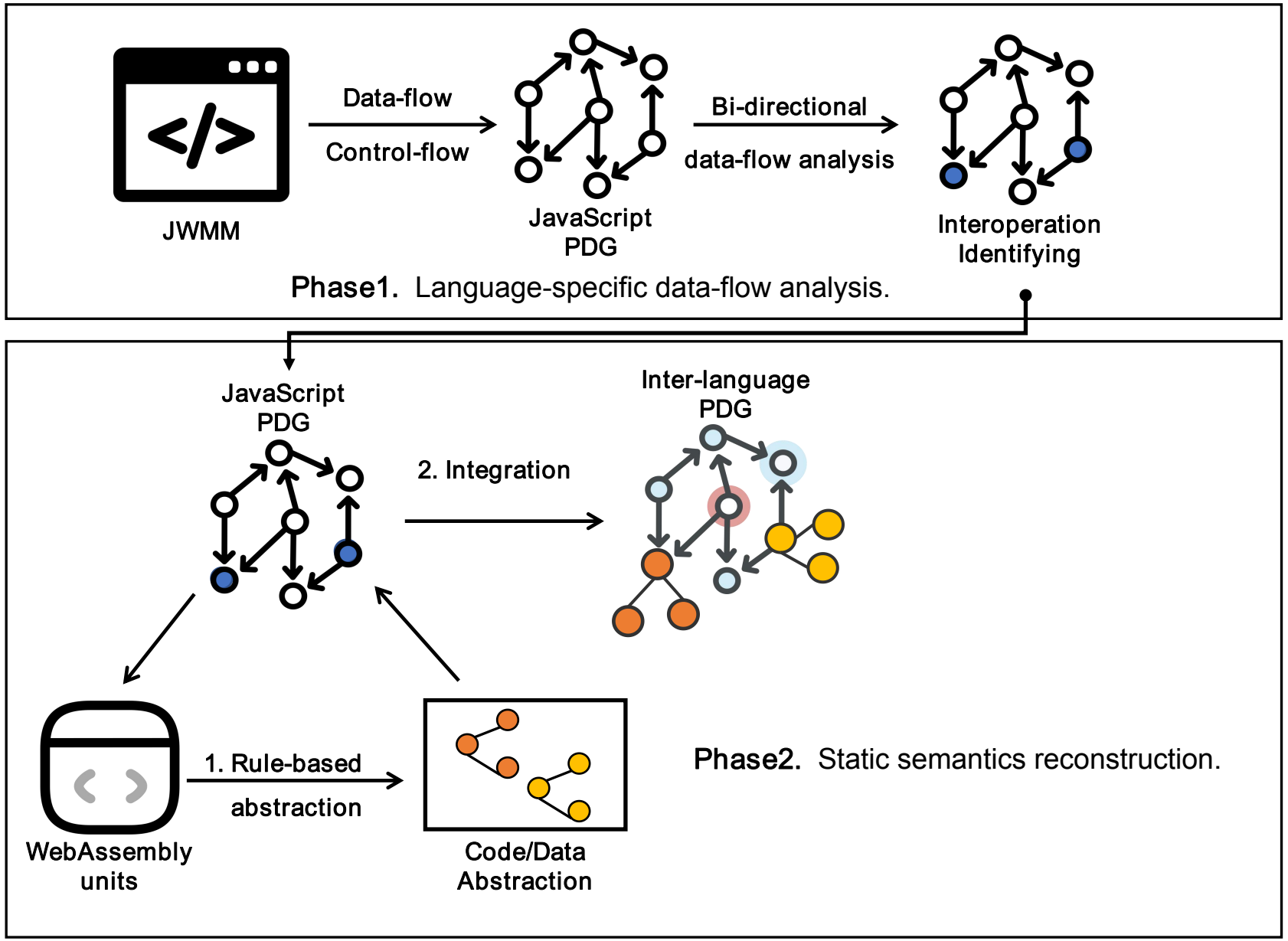


Phase1. Language-specific data-flow analysis.

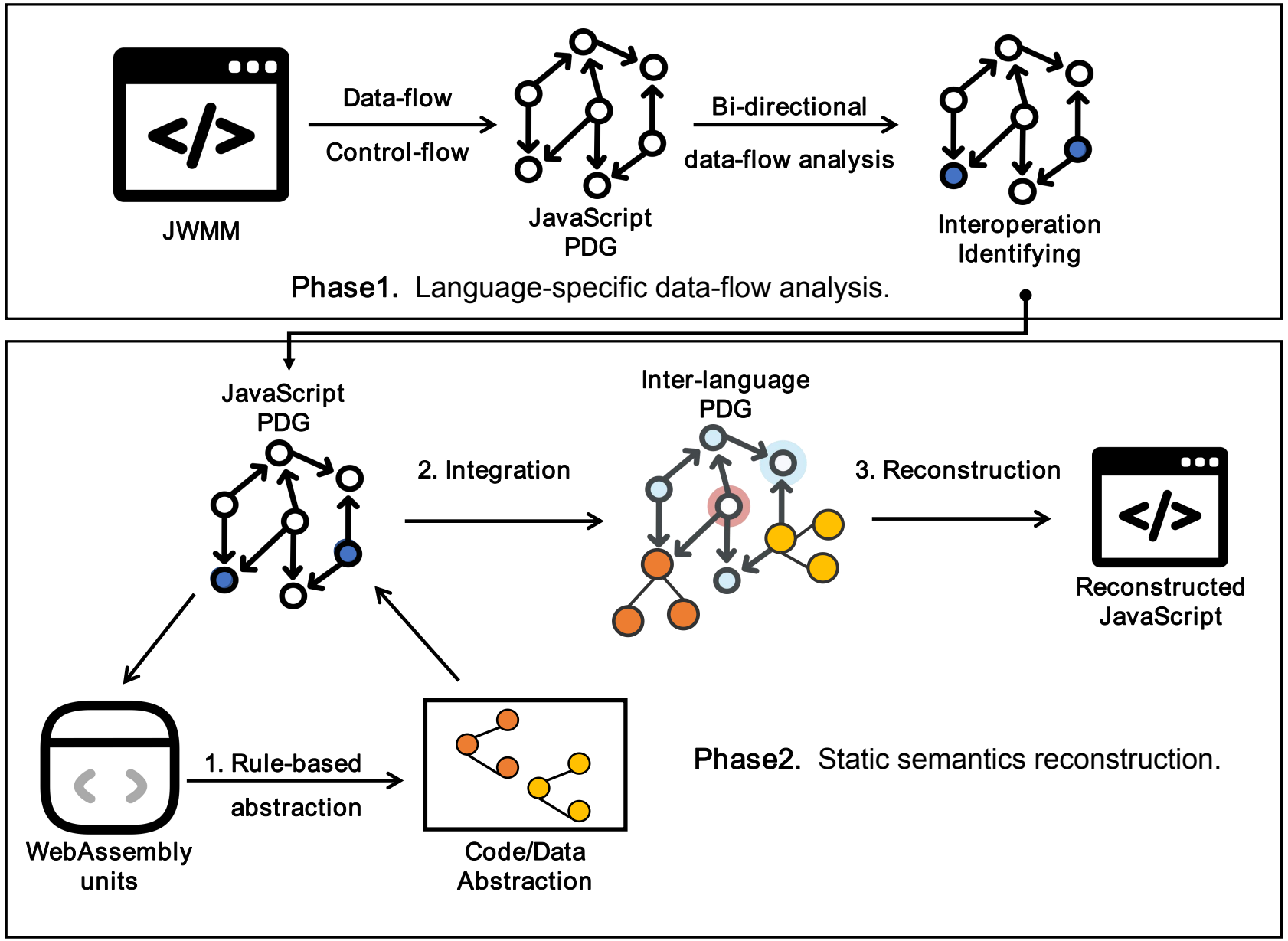
JWBinder Overview



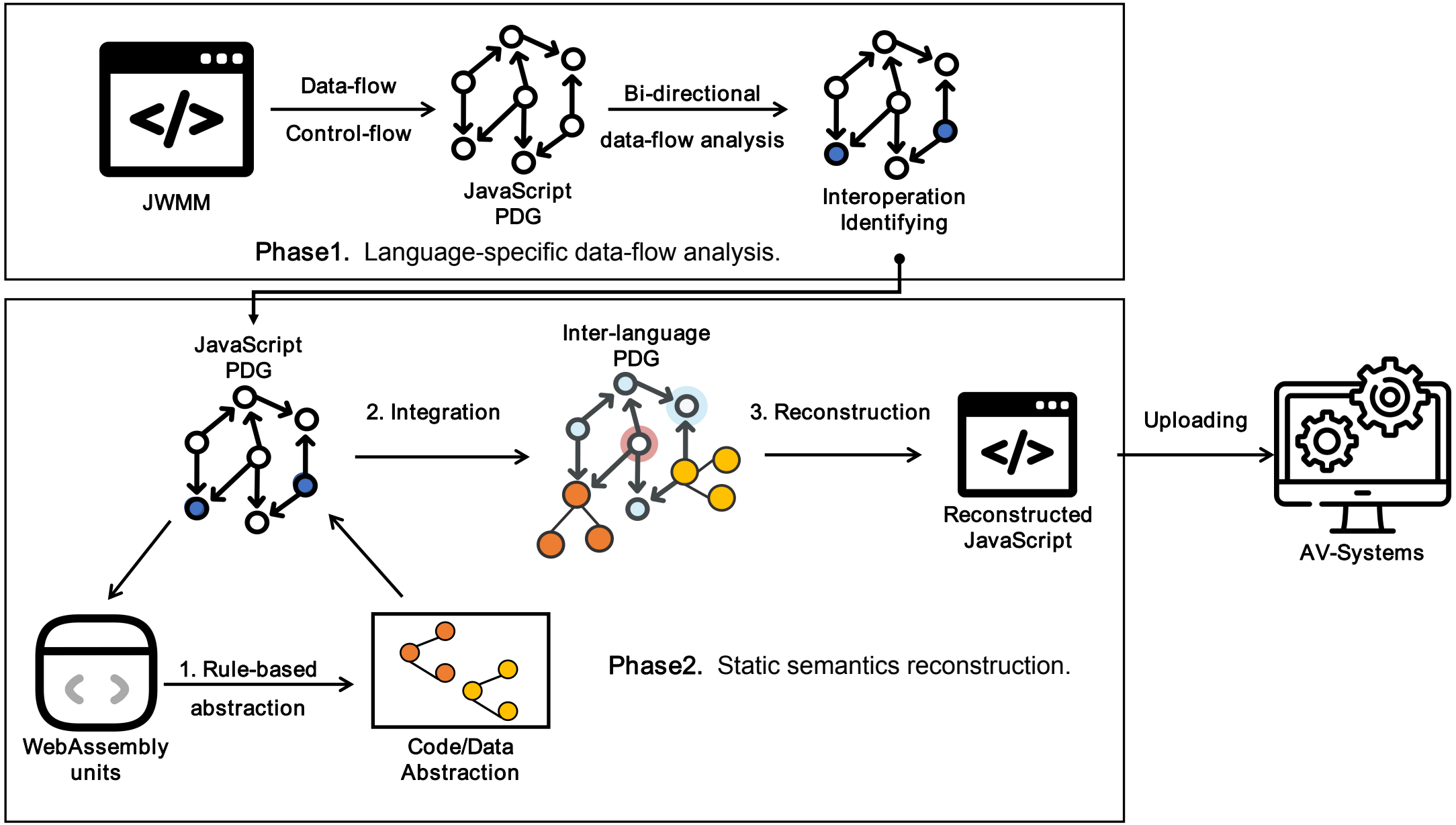
JWBinder Overview



JWBinder Overview



JWBinder Overview



Phase 1: Language-specific Data-flow Analysis

Zhejiang University

- **PDG Generation**

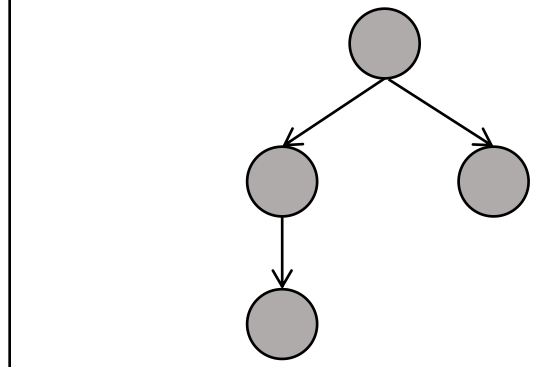
```
window.onload = function(module){
    var instance = WebAssembly.Instance(module);
    if (instance)
            instance.exports.foo();
    }
```

Phase 1: Language-specific Data-flow Analysis

Zhejiang University

- **PDG Generation**

```
window.onload = function(module){
    var instance = WebAssembly.instance(module);
    if (instance)
            instance.exports.foo();
    }
```



Abstract Code Representation → **AST**

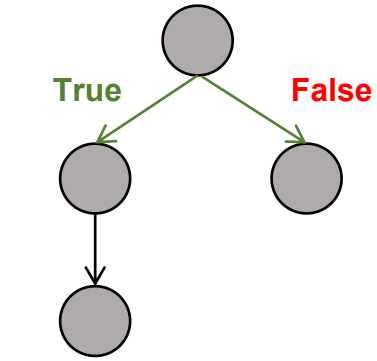
Phase 1: Language-specific Data-flow Analysis

Zhejiang University

- **PDG Generation**

```
window.onload = function(module){
    var instance = WebAssembly.Instance(module);
    if (instance)
        instance.exports.foo();
    }
```

True



Abstract Code Representation → **AST**

Conditions → **Control Flow Graph**

Phase 1: Language-specific Data-flow Analysis

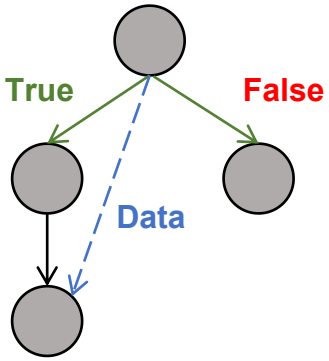
Zhejiang University

- **PDG Generation**

```
window.onload = function(module){
    var instance = WebAssembly.Instance(module);
    if (instance)
        instance.exports.foo();
    }
```

Data

True



Abstract Code Representation → **AST**

Conditions → **Control Flow Graph**

Data Dependenies → **Program Dependency Graph**

Phase 1: Language-specific Data-flow Analysis

Zhejiang University

- **Interoperation Identification**
  - Modelling Interface API

Function/Property Name	Description
WebAssembly.compile() WebAssembly.compileStreaming() WebAssembly.Module	The modularization of WebAssmebly binaries. Return a module for Instantiation.
WebAssembly.instantiate() WebAssembly.instantiateStreaming() WebAssembly.Instance	The Instantiation of WebAssmebly module. Return a instance which can call WebAssembly functions.
Instance.exports	Export WebAssembly internal properties to JavaScript.

Phase 1: Language-specific Data-flow Analysis

Zhejiang University

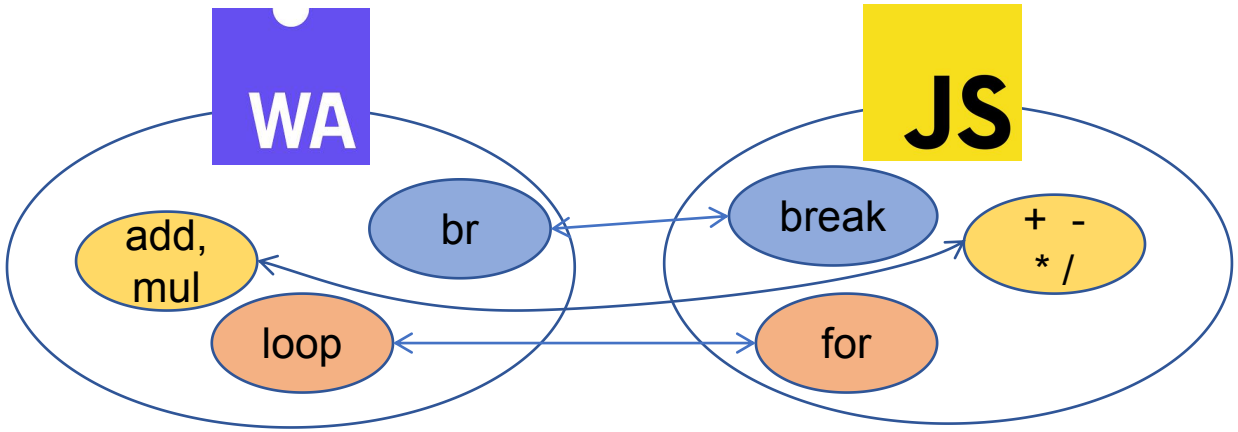
- **Interoperation Identification**
  - Modelling Interface API

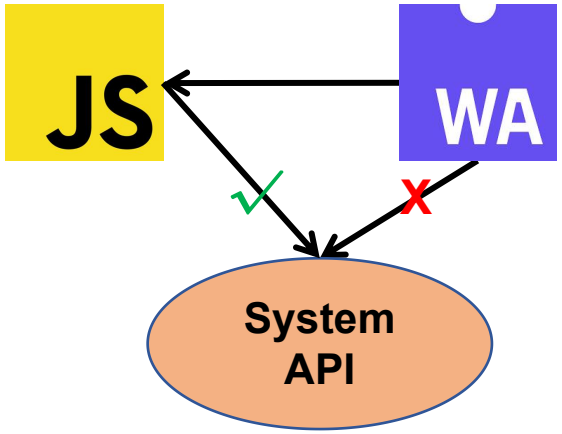
Function/Property Name	Description
WebAssembly.compile() WebAssembly.compileStreaming() WebAssembly.Module	The modularization of WebAssmebly binaries. Return a module for Instantiation.
WebAssembly.instantiate() WebAssembly.instantiateStreaming() WebAssembly.Instance	The Instantiation of WebAssmebly module. Return a instance which can call WebAssembly functions.
Instance.exports	Export WebAssembly internal properties to JavaScript.

  - Capturing cross-language interoperations through bi-directional data-flow analysis
    - Forward - matching WebAssembly execution patterns
    - Backward - capturing cross-language dependencies

Phase 2: Static Semantics Reconstruction

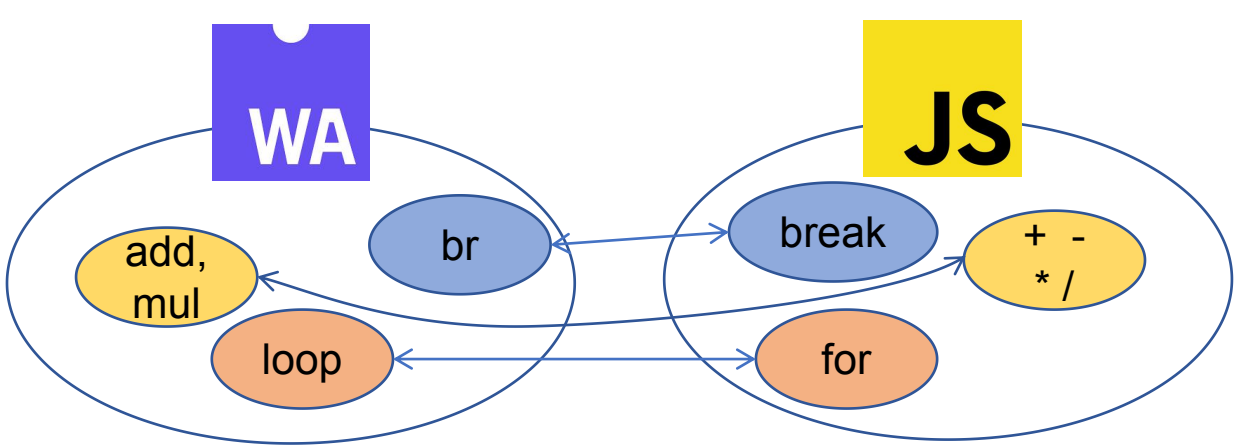
- **Key Intuition**
  - WebAssembly shares some homogeneous features with JavaScript

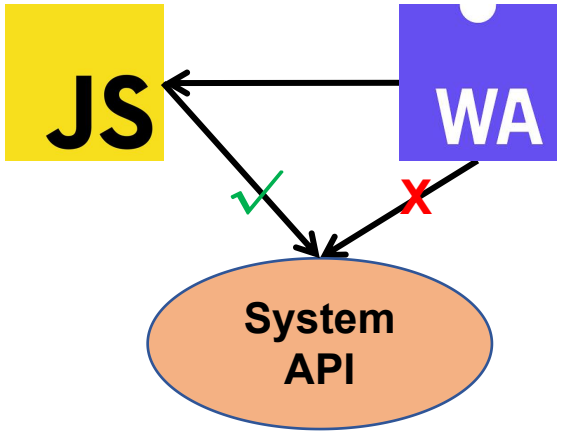




Phase 2: Static Semantics Reconstruction

- **Key Intuition**
  - WebAssembly shares some homogeneous features with JavaScript





  - Integrating semantics into a uniform representation based on such homogeneous features → Inter-language Program Dependency Graph (IPDG)

Phase 2: Static Semantics Reconstruction

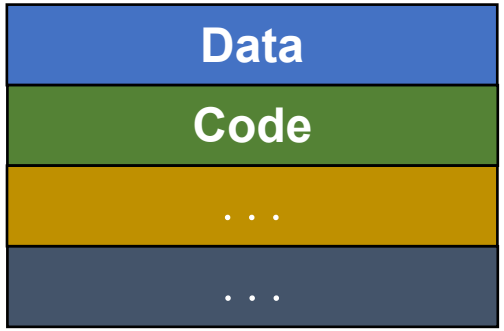
- **Rule-based Abstraction**
  - Extracts high-level semantics from WebAssembly

Phase 2: Static Semantics Reconstruction

Zhejiang University

- **Rule-based Abstraction**
  - Extracts high-level semantics from WebAssembly
  - Targets code and data sections

（a）WebAssembly Sections

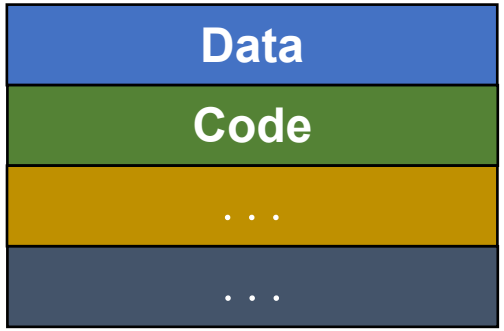


Phase 2: Static Semantics Reconstruction

Zhejiang University

- **Rule-based Abstraction**
  - Extracts high-level semantics from WebAssembly
  - Targets code and data sections
  - Turns WebAssembly instructions into ES6 syntax abstractions

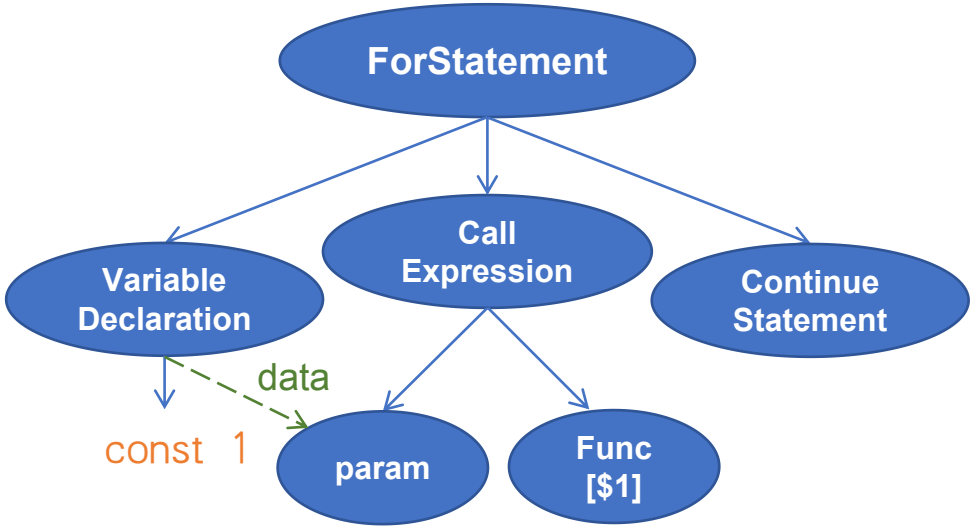
(a) WebAssembly Sections



(b) WebAssembly Instructions

```
loop
    i32.const 1
    call $1
    br 0
end
```

(c) ES6 Abstactions



Phase 2: Static Semantics Reconstruction

- **Code Abstraction**

  - Characterizes the functionalities of WebAssembly internal functions.

  - Two categories of instructions：
    - Data-flow instruction
    - Control-flow instruction

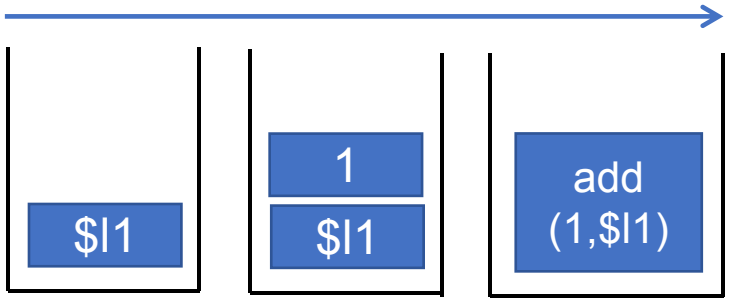
```
(import "env" "impFunc")
(data (i32.const 0) "<script>")
(data (i32.const 100) "malicious payloads")
(data (i32.const 200) "<script>")
(func $foo (type 0) (local i32)
    ...
    loop
        local.get 0
        load
        call $env.impFunc
        ...
        br 0
    end
    )
(export "foo" (func $foo))
```

Phase 2: Static Semantics Reconstruction

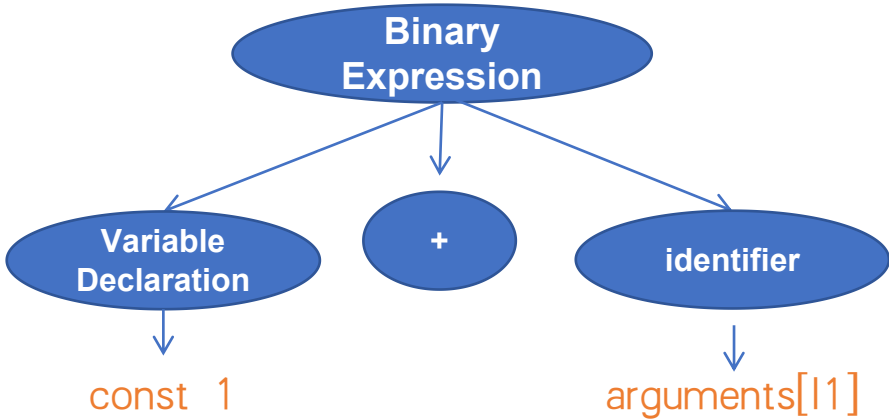
- **Code Abstraction**
  - Data-flow instructions

```
local.get $l1
i32.const 1
i32.add
```

(a) Data-flow instructions



(b) Stack Simulation

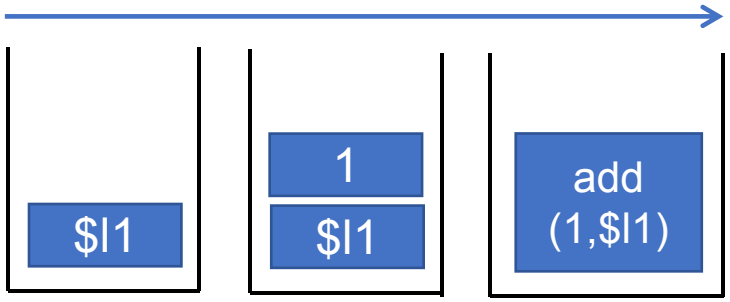


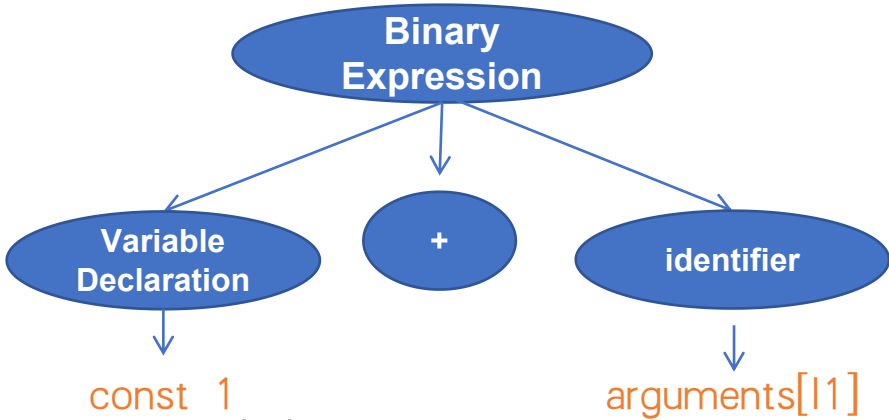
(c) Abstractions

Phase 2: Static Semantics Reconstruction

- **Code Abstraction**
  - Data-flow instructions

```
local.get $l1
i32.const 1
i32.add
```

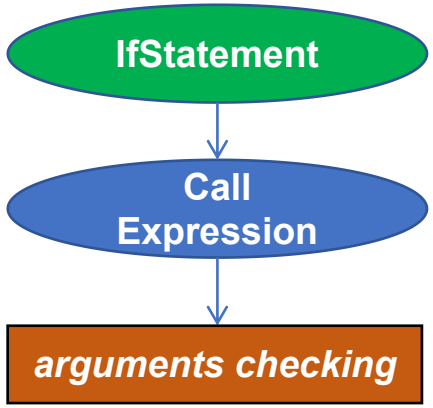




(a) Data-flow instructions　(b) Stack Simulation　(c) Abstractions

  - Control-flow instructions

```
if (local.get 1)
    call env.func
end
```



(a) Control-flow instructions　(b) Homogeneous matching　(c) Abstractions

Phase 2: Static Semantics Reconstruction

- **Data Abstraction**

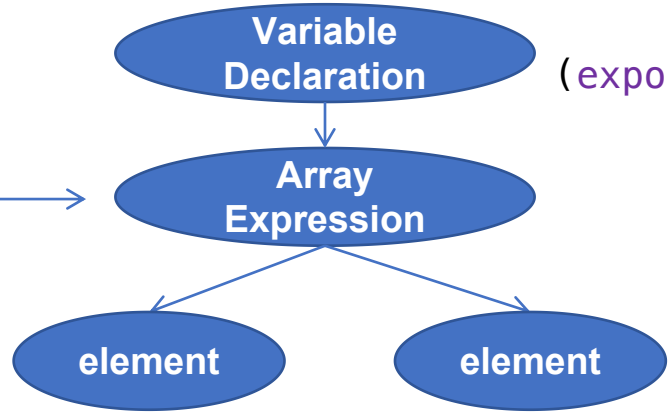
  - Characterizes suspicious values in the WebAssembly linear memory.

  - Limits the scope to the initialization process due to scale issues.

```
(import "env" "impFunc")
(data (i32.const 0) "<script>")
(data (i32.const 100) "malicious payloads")
(data (i32.const 200) "<script>")
(func $foo (type 0) (local i32)
    ...
    loop
        local.get 0
        load
        call $env.impFunc
        ...
        br 0
    end
    )
(export "foo" (func $foo))
```

```
data “payloads1”
data “payloads2”
```

(a) Data sections



(b) Data abstractions

Phase 2: Static Semantics Reconstruction

Zhejiang University

- **Inter-language PDG (IPDG) Integration**
  - Replaces JavaScript PDG nodes with WebAssembly Abstractions

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
        var importObject =
        {env:{impFunc: document.write}});
        var instance =
        WebAssembly.instance(module, importObject);
        if (instance)
            instance.exports.foo();
}
```

Phase 2: Static Semantics Reconstruction

Zhejiang University

- **Inter-language PDG (IPDG) Integration**
  - Replaces JavaScript PDG nodes with WebAssembly Abstractions
  - Targets position:
    - Instantiation site
    - Function invocation site

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
        var importObject =
        {env:{impFunc: document.write}});
        var instance =
        WebAssembly.instance(module, importObject);
        if (instance)
            instance.exports.foo();
}
```

Phase 2: Static Semantics Reconstruction

Zhejiang University

- **Inter-language PDG (IPDG) Integration**
  - Replaces JavaScript PDG nodes with WebAssembly Abstractions
  - Targets position:
    - Instantiation site
    - Function invocation site

```
window.onload = function(){
    var module =
        new WebAssembly.module(binary_data);
        var importObject =
        {env:{impFunc: document.write}});
        var instance =
        WebAssembly.instance(module, importObject);
        if (instance)
            instance.exports.foo();
}
```

- **JavaScript Reconstruction**
  - Our IPDG follows ES6 standards, which can be directly reconstructed into JavaScript.

Evaluation

Experiment Setup

Zhejiang University

- **Dataset**
  - **JWBench:** 21191 JWMM samples deriving from real-world JavaScript malware.

Experiment Setup

Zhejiang University

- **Dataset**
  - **JWBench:** 21191 JWMM samples deriving from real-world JavaScript malware.

  - **Real-world JS malware dataset:**

    44369 samples, 39,450 from Hynek Petrak, 3562 from VirusTotal and 1357 from the GeeksOnSecurity.

Experiment Setup

Zhejiang University

- **Dataset**
  - **JWBench:** 21191 JWMM samples deriving from real-world JavaScript malware.

  - **Real-world JS malware dataset:**

    44369 samples, 39,450 from Hynek Petrak, 3562 from VirusTotal and 1357 from the GeeksOnSecurity.

  - **Filtering samples with syntactic error and invalid cases**

    -> 21191 valid cases (JWBench$_o$).

Experiment Setup

Zhejiang University

Dataset

- **JWBench:** 21191 JWMM samples deriving from real-world JavaScript malware.

- **Real-world JS malware dataset:**

  44369 samples, 39,450 from Hynek Petrak, 3562 from VirusTotal and 1357 from the GeeksOnSecurity.

- **Filtering samples with syntactic error and invalid cases**

  -> 21191 valid cases (JWBench$_o$).

- **Automatically generate JWMM samples using Wobfuscator[1].**

Wobfuscator: Obfuscating JavaScript Malware via Opportunistic Translation to WebAssembly , Oakland'22

Experiment Setup

Zhejiang University

- **Baseline Anti-virus Solutions: VirusTotal**
  - One of the most popular online AV platform.
  - 59 AV-Systems providing detection services for malicious JavaScript.

VIRUSTOTAL

Experiment Setup

Zhejiang University

➢ Baseline Anti-virus Solutions: VirusTotal

- ➢ One of the most popular online AV platforms.
- ➢ 59 AV-Systems providing detection services for malicious JavaScript.

VIRUSTOTAL

➢ Evaluation Metrics

- ➢ **Successful Detection Rate (SDR):** the ratio of successfully detected samples to the overall samples.
- ➢ **Average Detected Engines (ADE):** the mean number of AV-Systems that successfully detect each malicious sample.

Performance of Existing Anti-Virus Solutions

Zhejiang University

- **The detection result of VirusTotal on JWBench_o and JWBench.**

	JWBench_o	JWBench
Successful Detection Rate	99.9%	47.6%
Average Detected Engines	22.4	3.3

More than half of the JWMM samples can elude detection by VirusTotal.

Effectiveness

Zhejiang University

- **How effective is JWBinder when deployed with real-world AV solutions?**
  - **Dataset:** 10k JWMM from JWBench

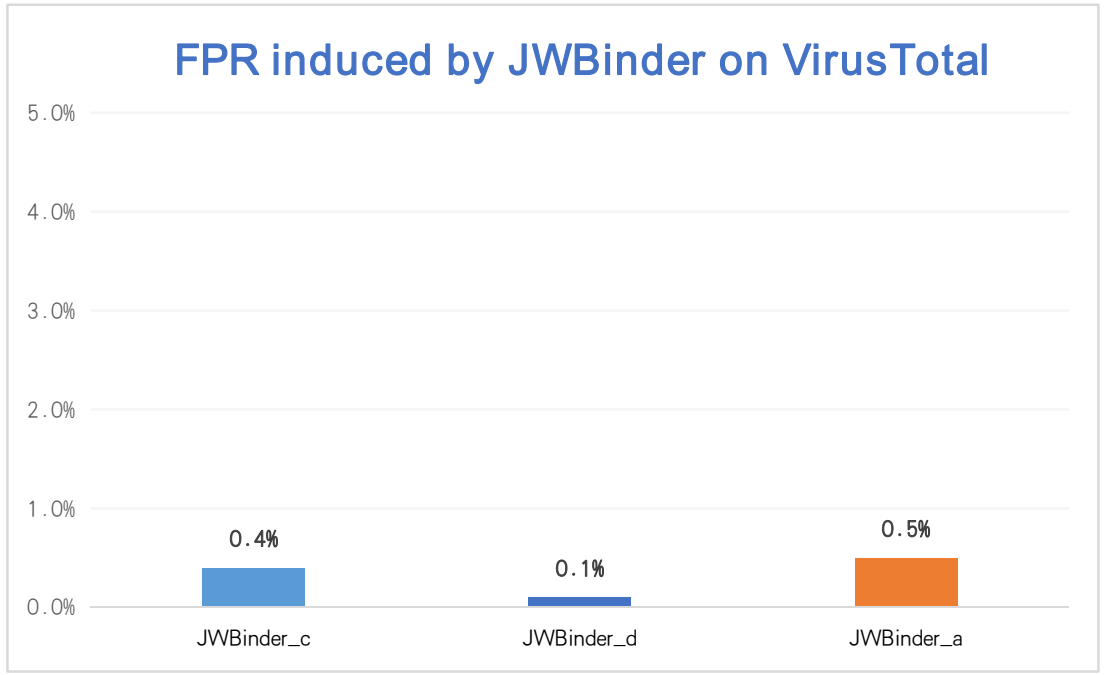
The detection result of VirusTotal on 10k JWMM processed by JWBinder

	Baseline	JWBinder_c	JWBinder_d	$\textbf{JWBinder}_a$
SDR	49.1%	64.7% $(\frac{+16.1\%}{-0.5\%})$	81.0% $(\frac{+33.5\%}{-1.6\%})$	**86.2%** $(\frac{+37.1\%}{-0.0\%})$
ADE	4.1	5.0	7.5	**8.3**

Different SSR levels contribute to the detection of various samples.

Side Effects

- **Does JWBinder introduce side effects to the benign multilingual programs?**
  - **Dataset:** 1000 samples deriving from JS150k which VirusTotal deems benign.



JWBinder introduces minimal side effects when processing JavaScript-WebAssembly multilingual programs.

Generalization Ability

Zhejiang University

- **What is the generalization ability of JWBinder on different commercial AV solutions?**

AV-System	Baseline	JWBinder$_c$	JWBinder$_d$	JWBinder$_a$
Google	31.1%	53.3%	59.4%	**61.3%**
Cyren	29.4%	50.5%	53.4%	**55.9%**
McAfee-GW-Edition	0%	47.5%	2.4%	**47.7%**
Microsoft	19.3%	16.0%	35.0%	**45.7%**
Rising	27.5%	2.6%	44.5%	**45.3%**
Arcabit	14.2%	20.0%	39.8%	**39.9%**
MicroWorld-eScan	14.2%	20.0%	39.8%	**40.0%**
FireEye	14.2%	19.9%	39.5%	**39.8%**
ALYac	13.8%	19.6%	38.9%	**39.4%**
GData	14.2%	19.0%	38.3%	**39.3%**
Emsisoft	14.0%	18.0%	36.9%	**38.6%**
BitDefender	14.2%	19.7%	39.5%	**39.7%**
VIPRE	14.1%	19.5%	39.1%	**39.5%**
MAX	14.2%	19.9%	39.6%	**39.8%**
Ikarus	2.07%	13.7%	23.6%	**24.0%**

- JWBinder significantly benefits AV-Systems from different vendors.

Generalization Ability

Zhejiang University

- **What is the generalization ability of JWBinder on different commercial AV solutions?**

AV-System	Baseline	JWBinder$_c$	JWBinder$_d$	JWBinder$_a$
Google	31.1%	53.3%	59.4%	**61.3%**
Cyren	29.4%	50.5%	53.4%	**55.9%**
McAfee-GW-Edition	0%	47.5%	2.4%	**47.7%**
Microsoft	19.3%	16.0%	35.0%	**45.7%**
Rising	27.5%	2.6%	44.5%	**45.3%**
Arcabit	14.2%	20.0%	39.8%	**39.9%**
MicroWorld-eScan	14.2%	20.0%	39.8%	**40.0%**
FireEye	14.2%	19.9%	39.5%	**39.8%**
ALYac	13.8%	19.6%	38.9%	**39.4%**
GData	14.2%	19.0%	38.3%	**39.3%**
Emsisoft	14.0%	18.0%	36.9%	**38.6%**
BitDefender	14.2%	19.7%	39.5%	**39.7%**
VIPRE	14.1%	19.5%	39.1%	**39.5%**
MAX	14.2%	19.9%	39.6%	**39.8%**
Ikarus	2.07%	13.7%	23.6%	**24.0%**

- JWBinder significantly benefits AV-Systems from different vendors.
- Some AV-Systems favor particular features more than others.

Efficiency

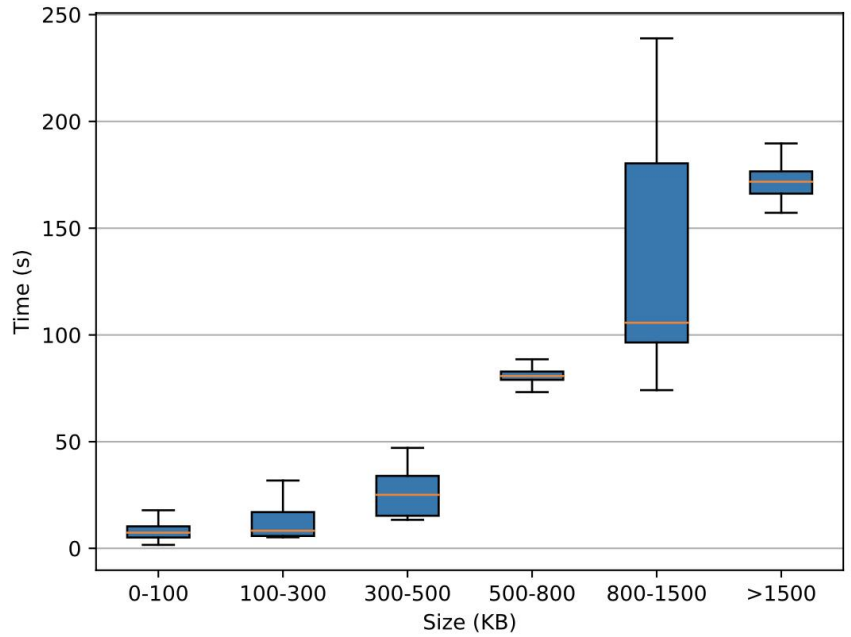
Zhejiang University

- **How efficient is JWBinder in terms of its runtime overhead?**
  - On average 10.0 seconds for data-flow analysis and 15.6 seconds for SSR, which is competitve compared to JS malware detection works (e.g. DoubleX$_1$ 100+ seconds).

Efficiency

Zhejiang University

- **How efficient is JWBinder in terms of its runtime overhead?**
  - On average 10.0 seconds for data-flow analysis and 15.6 seconds for SSR, which is competitve compared to JS malware detection works (e.g. DoubleX[1] 100+ seconds).



Run-time performance of JWBinder depending on the JWMM size

- The runtime overhead increases along with the program size.

Summary

- **JWBinder**: a technique for enhancing the detection of JavaScript-WebAssembly multilingual malware.
  - Cross-language interoperation Identification.
  - Simplifies multilingual malware detection to monolingual problem with SSR.
  - Increases VirusTotal‘s successful detection rate against JWMM from 49.1% to 86.2%.

- **Limitations**: .
  - Threats of run-time code/data generation.
  - Threats of obfuscation.
  - Dataset bias.

THANKS

➢ Example Abstraction Rules

Instruction	Stack Operation	Abstractions	Equivalent JavaScript
get_local v	push(v)		
i32.const c	push(c)	VariableDeclaration -> [id -> C_n , init -> c, kind -> const]	const C_n = c;
set_local v	pop() -> e	AssignStatement -> [id -> v , init -> e]	v = e ;
i32.mul	pop() -> e_1, pop() -> e_2, push(e_1 * e_2)	VariableDeclaration -> [id -> V_n , kind -> let, init -> BinaryExpression -> [left -> e_1, right -> e_2, op: *]]	let V_n = e_1 * e_2;
i32.popcnt	pop() -> e, push(popcnt())	VariableDeclaration -> [id -> V_n , kind -> let, init -> CallExpression -> [callee -> popcnt, arguments -> [e]]	let V_n = popcnt(e);
loop/block g		LabelStatement -> [body -> ForStatement -> [init, test->true, update, body -> [...]] , label -> g]	g: for(;true;){...};
if I	pop() -> e	IfStatement -> [test -> e, body-> [...], alternate -> [...]]	if (e) {...};
br g		BreakStatement -> [label -> g] (in block context) ; ContinueStatement -> [label -> g] (in loop context)	break g ; (in block context) continue g ; (in loop context))
call f	paraNum(f) -> n, for(i=0;i<n;i++) pop -> e_i	CallExpression -> [callee - > f, arguments -> [e_1, e_2, ... , e_n]]	f(e_1,e_2,...,e_n);
call_indirect	pop() -> e, paraNum(f) -> n, for(i=0;i<n;i++) pop -> e_i	CallExpression -> [callee - > f_e, arguments -> [e_1, e_2, ... , e_n]]	f_e(e_1,e_2,...,e_n);
data (type) “payloads”		VariableDeclaration -> [id -> memory , kind -> var, init -> BinaryExpression -> [left -> e_1, right -> e_2, op: *]]	var memory = “payloads”;