University of Pisa

Department of Information Engineering

Language-Based Technology for Security

Nicolò Mariano Fragale
March 2025

Contents

Information

These notes are intended for educational purposes only and cover essential concepts in the field of data systems and security. The aim is to provide a comprehensive understanding of topics such as system vulnerabilities, protection techniques, and defense strategies in cybersecurity.

This document includes topics related to access control, authentication mechanisms, database security, cryptographic methods, and advanced persistent threats, with a particular focus on practical applications in real-world scenarios.

1 Web Assembly

1.1 Introduction WASM is not a programming language, but a binary format generated from other language like C, C++ or Rust. WASM permit this hig level language to run efficiently and properly. It is executed in safe place like browser or other runtime enviroment.

It is safe because it runs in isolated sandbox.

It is used to increase performance in web application as:

1. 3D games in Browser;
2. Figma etc...
3. Editing image/video software online...
4. Ai, ML, blockchain, criptoghraphy...
5. Allow to execute C, C++, Rust online;
6. It can be used on server.

1.2 Key characteristics

1. Stack-Based (push and pop) <-> Does not use registers; -> Operations;
2. Executabel in web broser -> using WebAssembly JavaScript API. -> API is the only way to communicate from sandbox to outside;
3. Secure -> Sandbox and permission denied to access system resources;
4. Platform-independent -> runs on any device that has WASM runtime.

Listing 1: Code Example

```
(func $calcola (param $x i32) (result i32)
    local.get $x
    local.get $x
    i32.mul
    i32.const 2
    i32.mul
    i32.const 1
    i32.add
)
```

Analyze the example:

1. *func $calcola (param $x i32) (result i32)* :
   (a) *func* it is the key word declaring the function;
   (b) *$calcola* function's name;
   (c) *param* it is the key word declaring the parameter;

(d) *$x* parameter's name;

(e) *i32* indicates the data-type (32 bit integer);

(f) *result i32* indicates the result will be a i32 data type.

(g) **if $ is omitted the code will still work.**

2. *local.get $x* push X in stack with index 0 (Func Starts wih stack empty);
3. *local.get $x* push X in stack with index 1;
4. *i32.mul* pop 0 and 1 mul, then mul them (both x) and push temporary result in index 0;
5. *i32.const 2* push in stack the value 2 as type i32 and index 1;
6. *i32.mul* pop 0 and 1, then mul them and push as temporary result as index 0;
7. *i32.const 1* add 1 as i32 in index 1 ;
8. *i32.add* pop 0 and 1, add index 0 and 1, result is pushed in index 0.

1.3 Data-Type

1. **i32** integer with or without sign in 32 bit -> (from 0 to 4.294.967.295) or (from -2.147.483.648 to 2.147.483.647);
2. **i64** integer with or without sign in 64 bit;
3. **f32** floating poin in 32 bit;
4. **f64** floating point in 64 bit.

1.4 Storing Values

1. Stack -> push and pop (for operations) of the parameter and costant;
2. Function context -> variabale declared inside the function -> Example: *local $temp i32*; ->

Listing 2: Code Example

```
1         (func $quadrato (param $x i32) (result i32)
2         (local $temp i32)
3         local.get $x
4         local.get $x
5         i32.mul
6         local.set $temp
7         local.get $temp
8     )
```

3. Single global memory -> Linear memory to handle complex data structure -> Used by many functions to store data in long term or to share data beetween more functions -> Example:

Listing 3: Code Example

```
1         (global $contatore (mut i32) (i32.const 0))
2
3         (func $incrementa (result i32)
4             global.get $contatore
5             i32.const 1
6             i32.add
7             global.set $contatore
8             global.get $contatore
9         )
```

(mut i32) mutable variabale type i32 (otherwise immutable during the execution), **(i32.const 0)** initialized at 0.