UNIVERSITÀ
DEGLI STUDI
DI GENOVA

Analisi della sicurezza binaria di WebAssembly

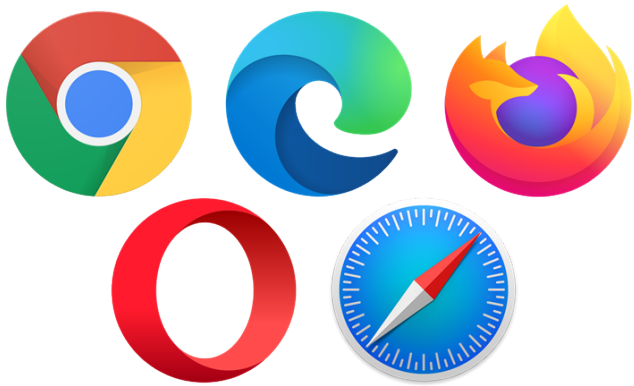
Alessandro Arata

17 Settembre 2021

Relatore: Prof. Giovanni Lagorio

WebAssembly

- linguaggio bytecode
- tempi di esecuzione rapidi
- formato portabile
- target di compilazione
- eseguito su **browser** o in backend



WebAssembly

- linguaggio bytecode
- tempi di esecuzione rapidi
- formato portabile
- **target di compilazione**
- eseguito su **browser** o in backend

WebAssembly

- linguaggio bytecode
- tempi di esecuzione rapidi
- formato portabile
- **target di compilazione**
- eseguito su **browser** o in backend

hello-world.wat

```
puts("Hello, world!");
```

corrisponde nel formato assembly di WebAssembly (**.wat**) a

```
(module
  ;; Imports from JavaScript namespace and log function
  (import "console" "log" (func $log (param i32 i32)))
  ;; Import 1 page of memory
  (import "js" "mem" (memory 1))
  ;; Data section of our module
  (data (i32.const 0) "Hello, world!")
  ;; Function declaration: Exported as helloWorld(), no arguments
  (func (export "helloWorld")
    ;; pass offset 0 to log
    i32.const 0
    ;; pass length 13 to log (strlen of sample text)
    i32.const 13
    call $log
  )
)
```

hello-world.wat

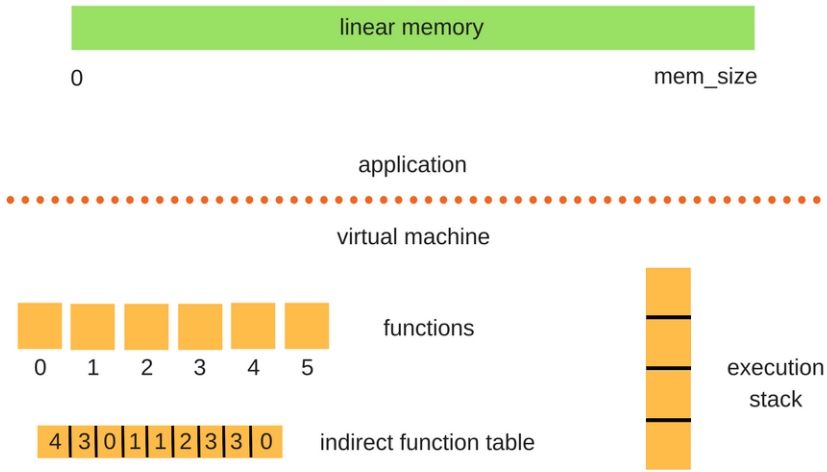
```
puts("Hello, world!");
```

corrisponde nel formato assembly di WebAssembly (**.wat**) a

```
(module
  ;; Imports from JavaScript namespace and log function
  (import "console" "log" (func $log (param i32 i32)))
  ;; Import 1 page of memory
  (import "js" "mem" (memory 1))
  ;; Data section of our module
  (data (i32.const 0) "Hello, world!")
  ;; Function declaration: Exported as helloWorld(), no arguments
  (func (export "helloWorld")
    ;; pass offset 0 to log
    i32.const 0
    ;; pass length 13 to log (strlen of sample text)
    i32.const 13
    call $log
  )
)
```

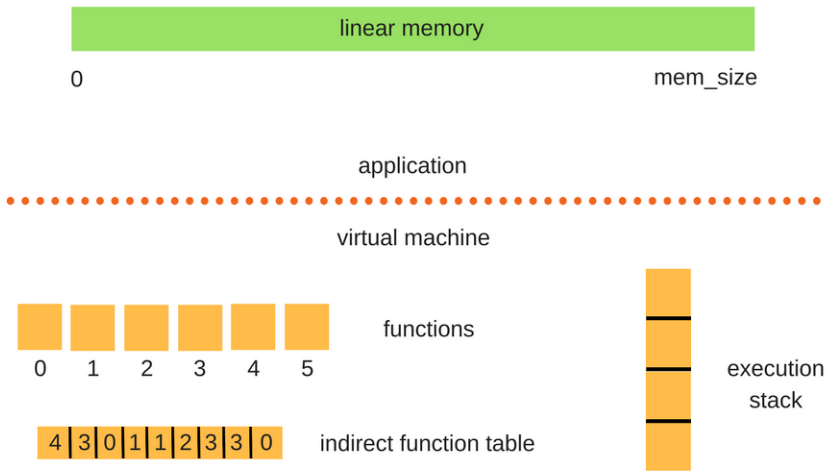
Tipi e gestione della memoria di WebAssembly

- staticamente tipato
  - **i32/64** e **f32/64**
- stack delle chiamate gestito dalla macchina virtuale
- tipi non primitivi salvati nella **memoria lineare**
  - gestita dal programma



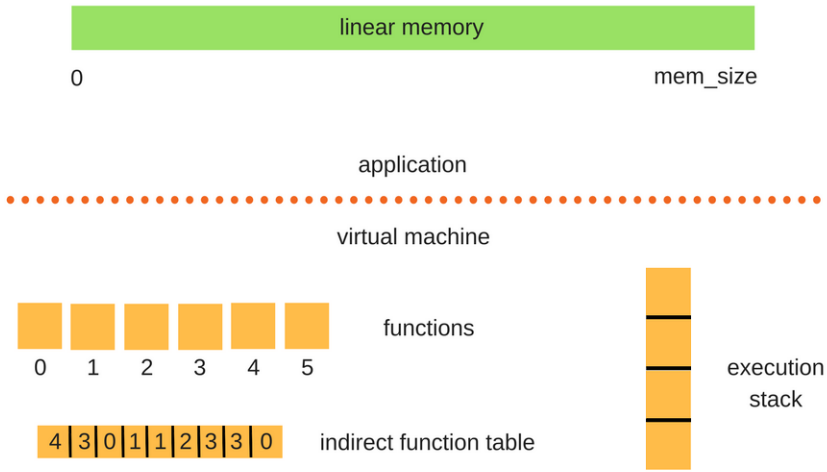
Tipi e gestione della memoria di WebAssembly

- staticamente tipato
  - **i32/64** e **f32/64**
- stack delle chiamate gestito dalla macchina virtuale
- tipi non primitivi salvati nella **memoria lineare**
  - gestita dal programma



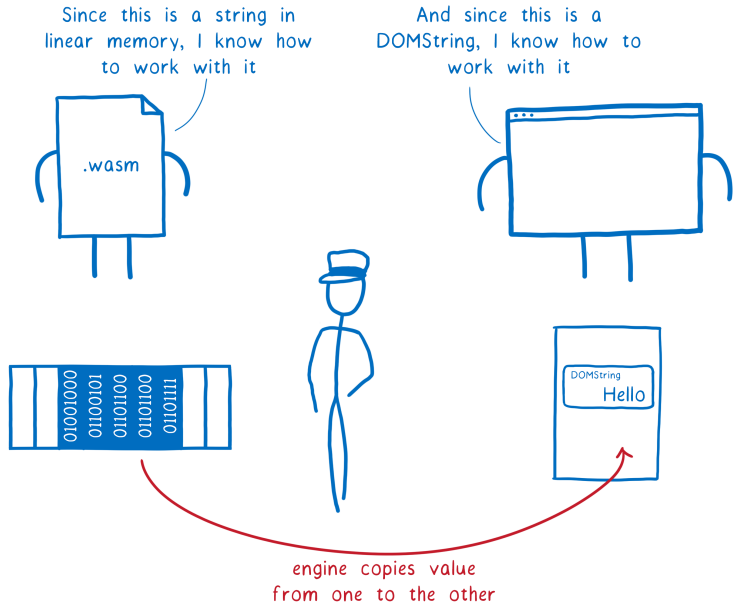
Tipi e gestione della memoria di WebAssembly

- staticamente tipato
  - **i32/64** e **f32/64**
- stack delle chiamate gestito dalla macchina virtuale
- tipi non primitivi salvati nella **memoria lineare**
  - gestita dal programma



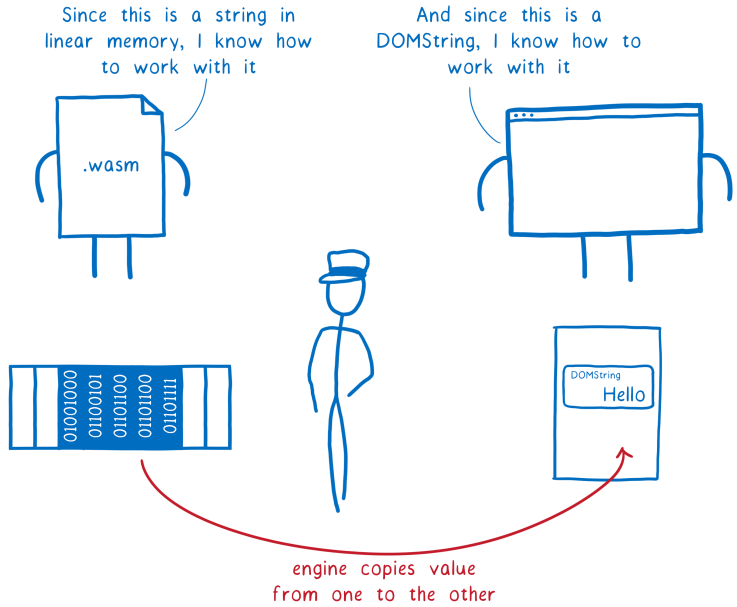
Memoria lineare

- array **globale** di byte
- divisa in **regioni** per heap, stack e dati statici
- sia leggibile che scrivibile ma non eseguibile
- ogni puntatore compreso tra [0, mem_max] è valido



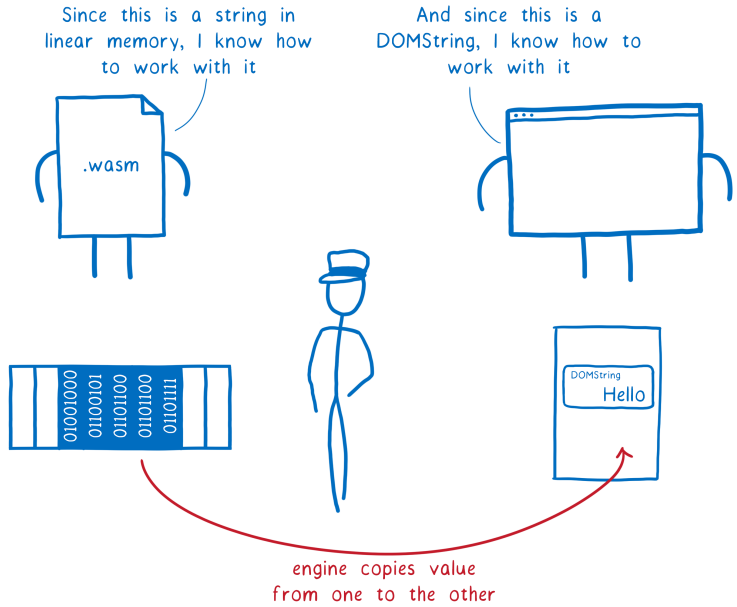
Memoria lineare

- array **globale** di byte
- divisa in **regioni** per heap, stack e dati statici
- sia leggibile che scrivibile ma non eseguibile
- ogni puntatore compreso tra [0, mem_max] è valido



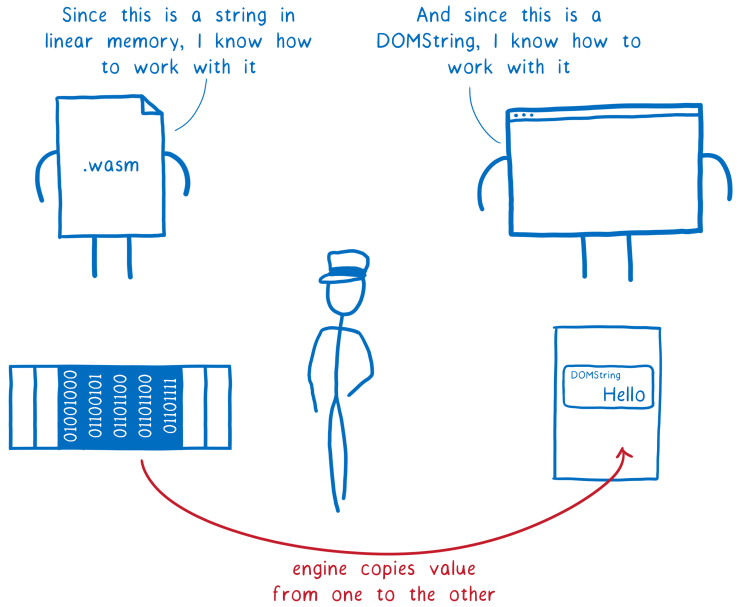
Memoria lineare

- array **globale** di byte
- divisa in **regioni** per heap, stack e dati statici
- sia leggibile che scrivibile ma non eseguibile
- ogni puntatore compreso tra [0, mem_max] è valido



Memoria lineare

- array **globale** di byte
- divisa in **regioni** per heap, stack e dati statici
- sia leggibile che scrivibile ma non eseguibile
- ogni puntatore compreso tra [0, mem_max] è valido



Principali mitigazioni nei binari nativi

- **Address Space Layout Randomization** (ASLR)
- Guard pages
- Stack canaries
- Non-executable stack (NX)
- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Principali mitigazioni nei binari nativi

- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- Stack canaries
- Non-executable stack (NX)
- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Principali mitigazioni nei binari nativi

- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- **Non-executable stack** (NX)
- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Principali mitigazioni nei binari nativi

- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- **Non-executable stack** (NX)
- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Principali mitigazioni nei binari nativi

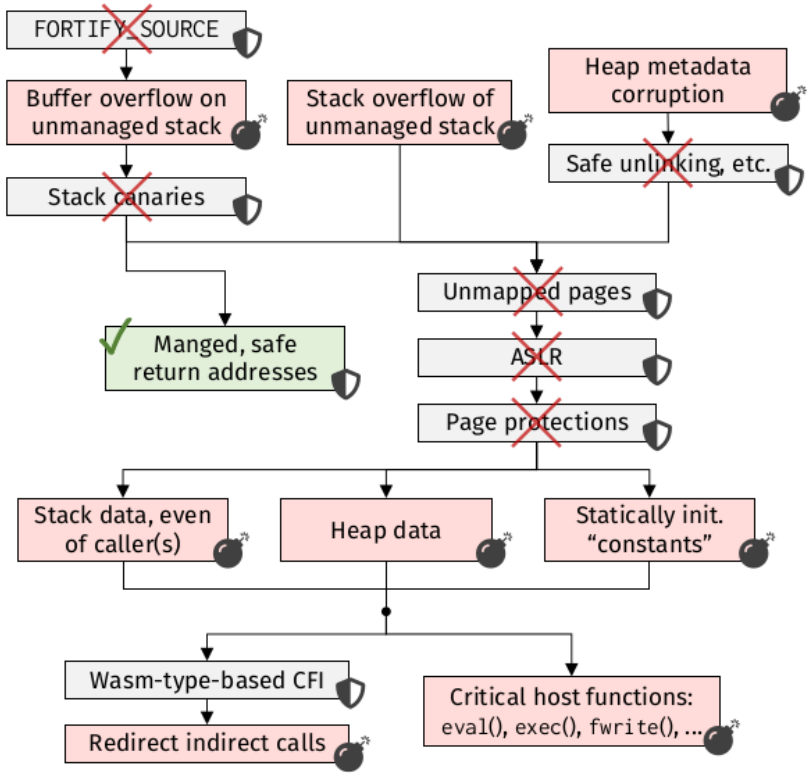
- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- **Non-executable stack** (NX)
- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Principali mitigazioni nei binari nativi

- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- **Non-executable stack** (NX)
- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Sicurezza binaria della memoria lineare

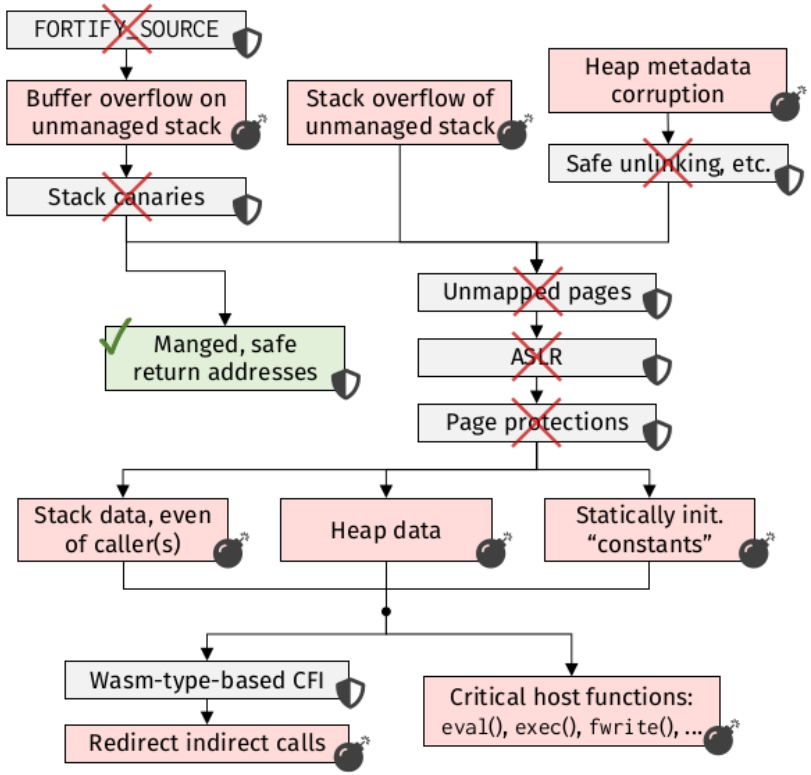
- **Address Space Layout Randomization** (ASLR)
- Guard pages
- Stack canaries
- Non-executable stack (NX)



- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Sicurezza binaria della memoria lineare

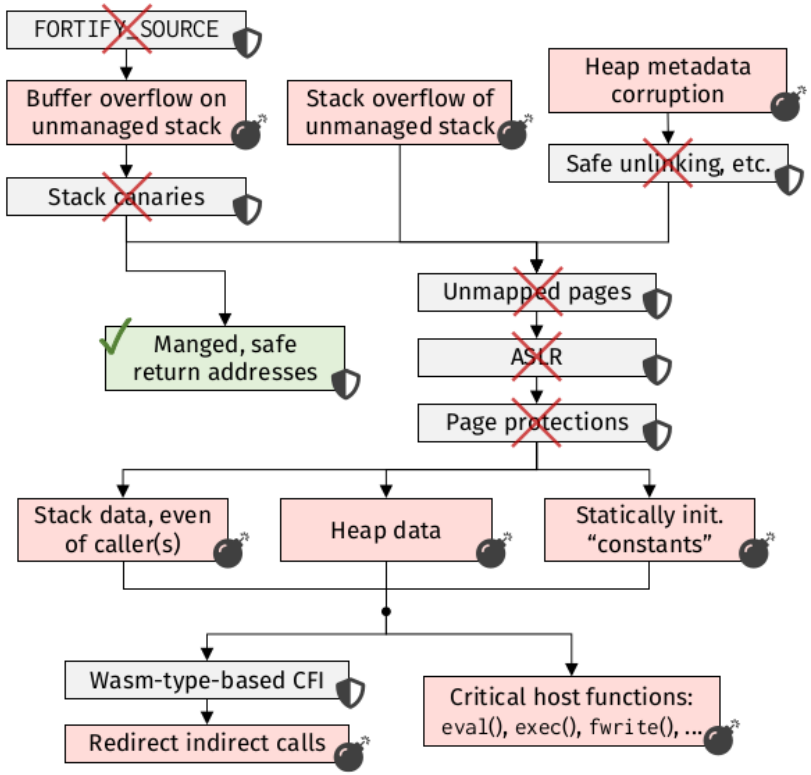
- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- Stack canaries
- Non-executable stack (NX)



- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Sicurezza binaria della memoria lineare

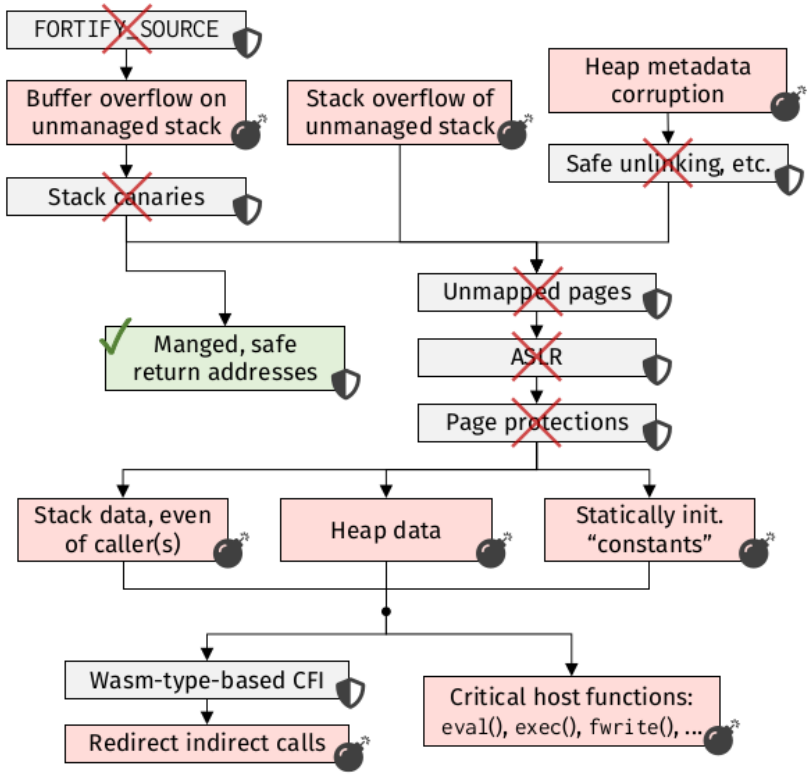
- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- Non-executable stack (NX)



- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Sicurezza binaria della memoria lineare

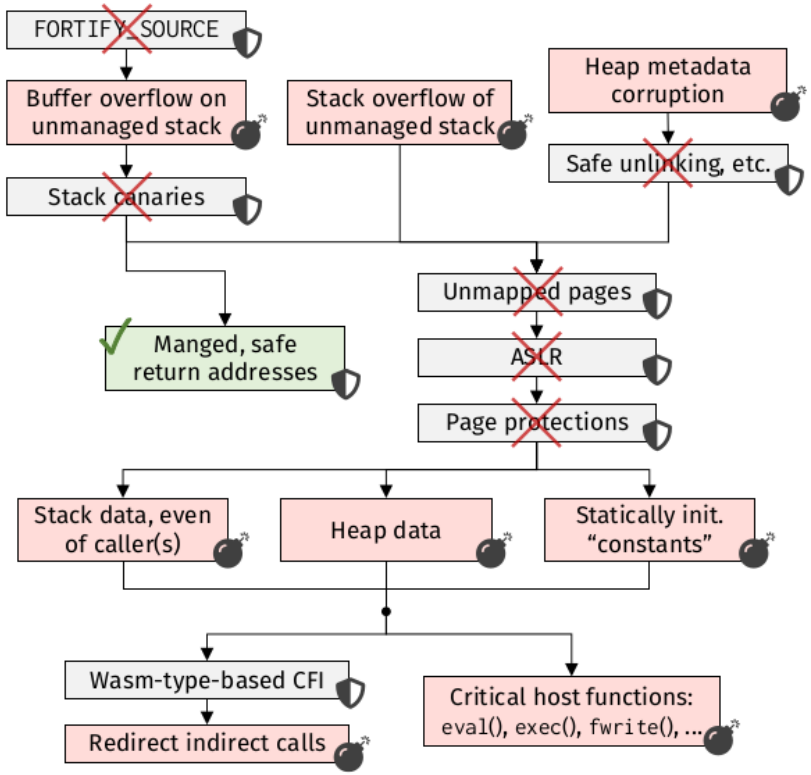
- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- **Non-executable stack** (NX)



- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Sicurezza binaria della memoria lineare

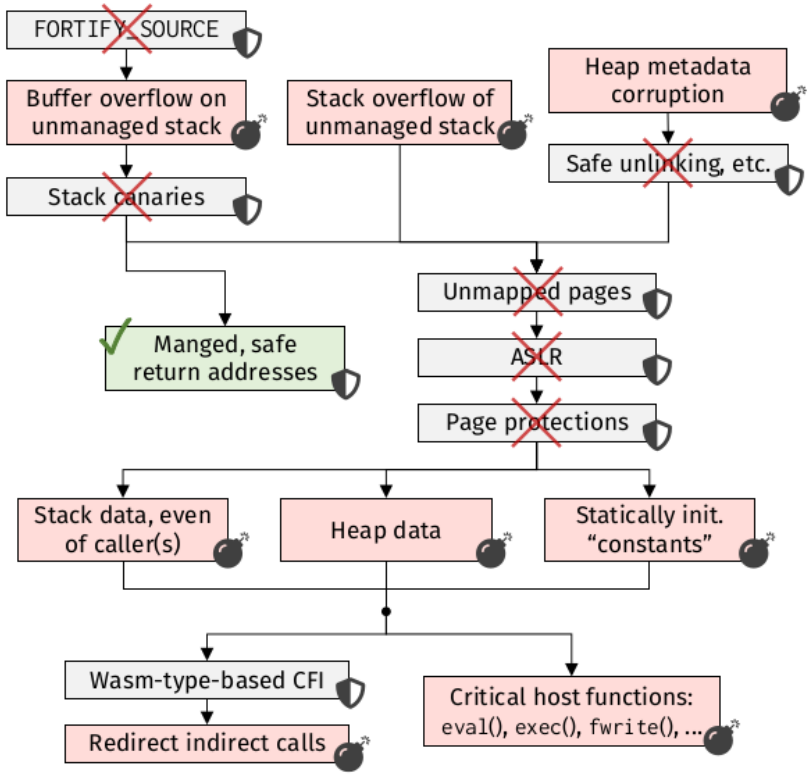
- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- **Non-executable stack** (NX)



- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Sicurezza binaria della memoria lineare

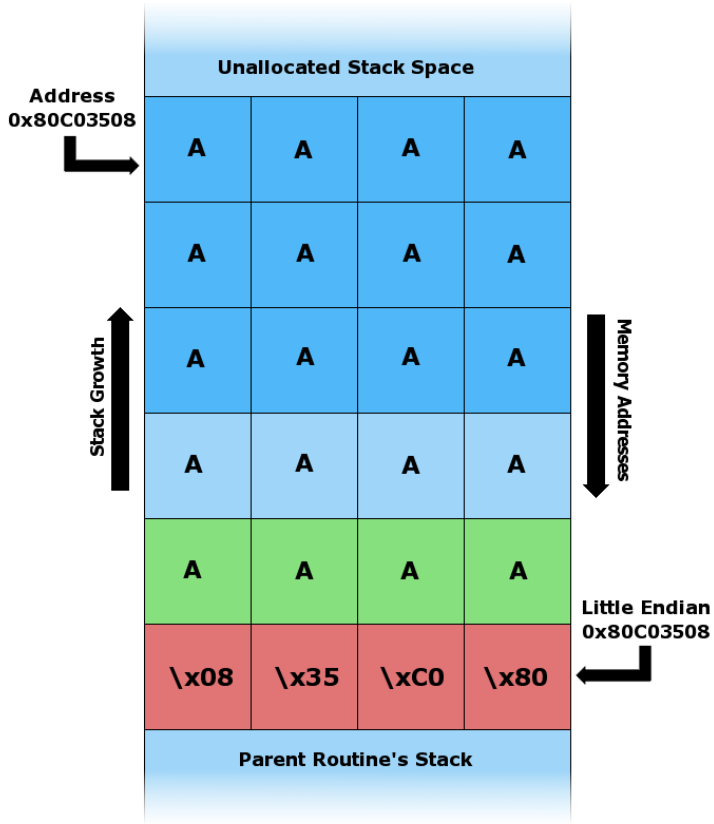
- **Address Space Layout Randomization** (ASLR)
- **Guard pages**
- **Stack canaries**
- **Non-executable stack** (NX)



- esistono sezioni non scrivibili (per i dati costanti)
- aree di memoria del processo potrebbero non essere mappate
  - page fault

Primitiva di attacco: buffer overflow

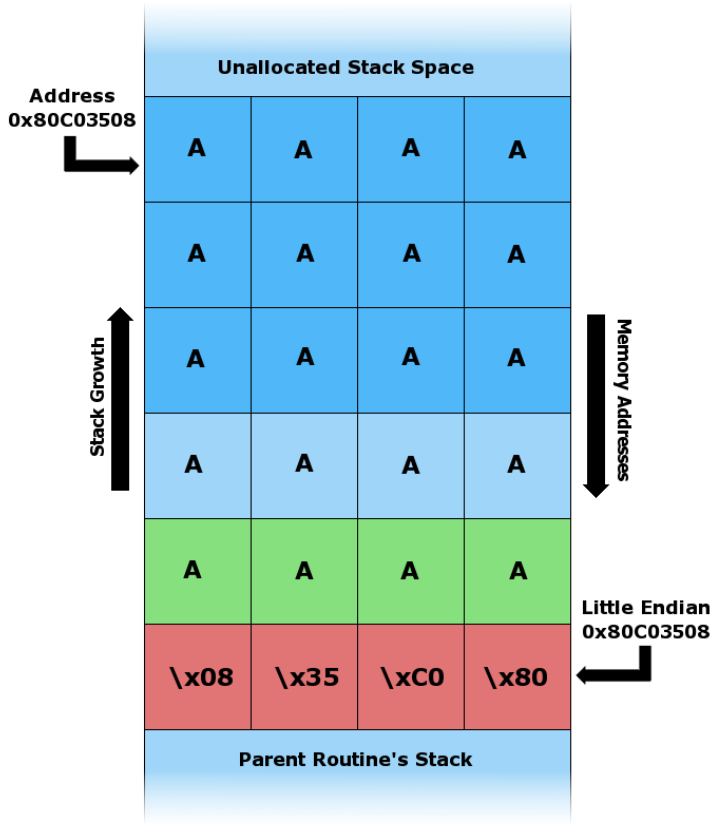
- l'attaccante scrive al di fuori del buffer allocato
  - si possono sovrascrivere variabili locali o indirizzi di ritorno
- gets() e strcpy() permettono questo tipo di attacco



Cosa succede se un programma **vulnerabile** a un buffer overflow viene compilato in WebAssembly?

Primitiva di attacco: buffer overflow

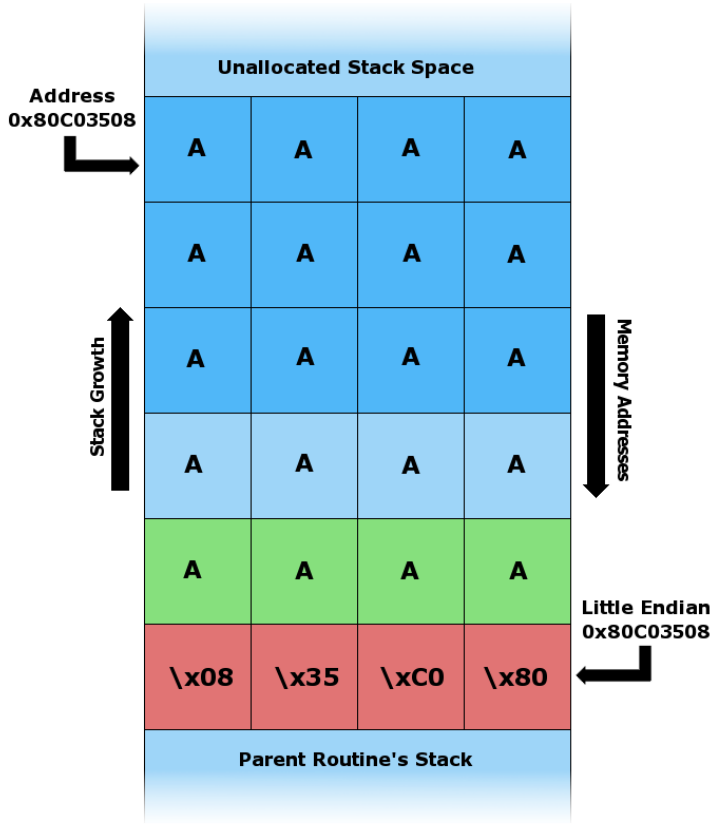
- l'attaccante scrive al di fuori del buffer allocato
  - si possono sovrascrivere variabili locali o indirizzi di ritorno
- **gets()** e **strcpy()** permettono questo tipo di attacco



Cosa succede se un programma **vulnerabile** a un buffer overflow viene compilato in WebAssembly?

Primitiva di attacco: buffer overflow

- l'attaccante scrive al di fuori del buffer allocato
  - si possono sovrascrivere variabili locali o indirizzi di ritorno
- **gets()** e **strcpy()** permettono questo tipo di attacco



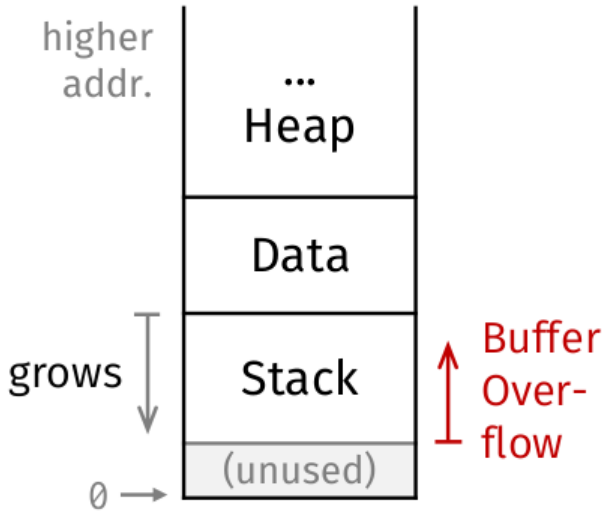
Cosa succede se un programma **vulnerabile** a un buffer overflow viene compilato in WebAssembly?

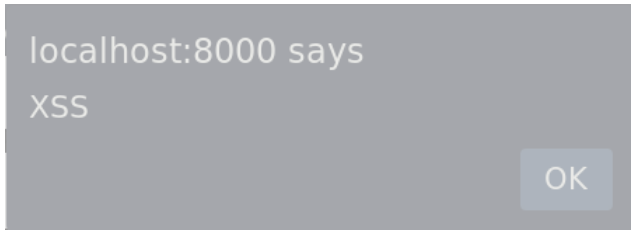
Sovrascrivere dati “costanti”

```
char *other_data = "AAAA";

static char *safe_script =
  "console.log('this should be safe, shouldn\\'t it?')";

int main() {

  emscripten_run_script(safe_script);

}

void vuln(const char* input) {

  strcpy(other_data, input);

}
```

Sovrascrivere dati “costanti”

- variabili statiche salvate nella regione **data**
- **strcpy()** non effettua controlli sulla dimensione dell’input
  - si sovrascrivono dati sullo stack
- l’overflow dallo stack scrive nella regione **data**
  - si sovrascrive **safe_script**
- **XSS** e **RCE**

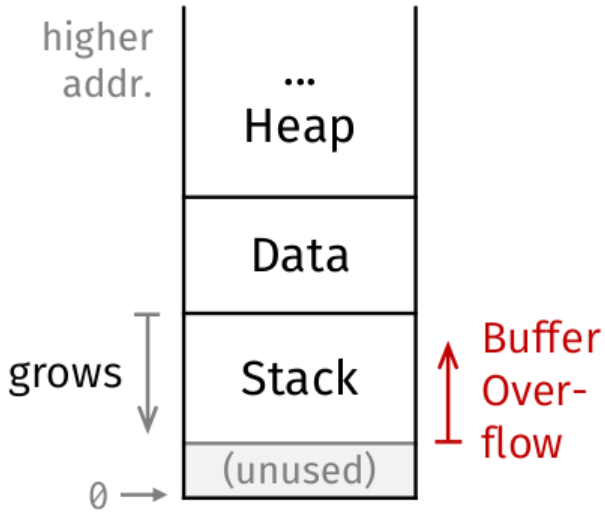


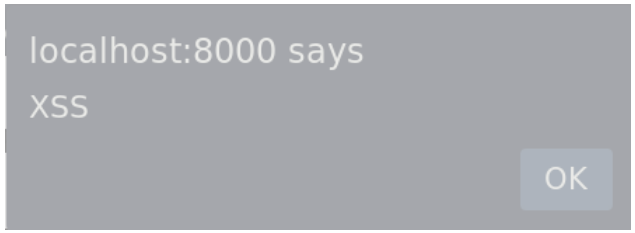


Chiamando **vuln()** con la stringa “.....alert('XSS')”

Sovrascrivere dati "costanti"

- variabili statiche salvate nella regione **data**
- **strcpy()** non effettua controlli sulla dimensione dell'input
  - si sovrascrivono dati sullo stack
- l'overflow dallo stack scrive nella regione **data**
  - si sovrascrive **safe_script**
- **XSS** e **RCE**

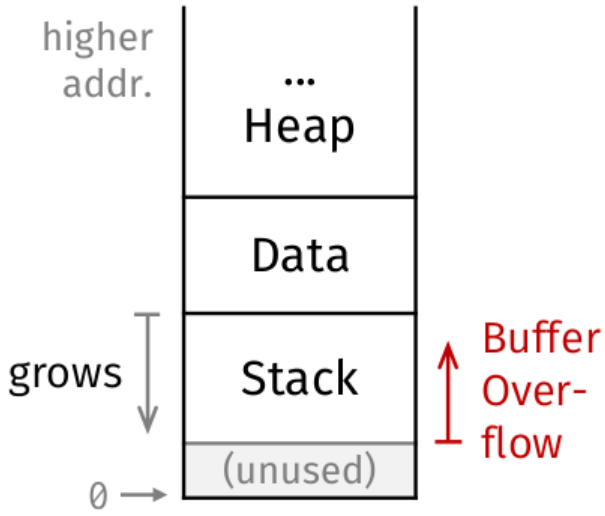


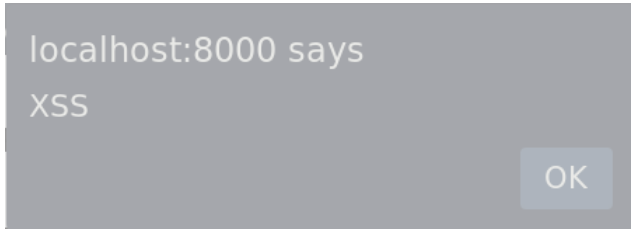


Chiamando **vuln()** con la stringa "...alert('XSS')"

Sovrascrivere dati "costanti"

- variabili statiche salvate nella regione **data**
- **strcpy()** non effettua controlli sulla dimensione dell'input
  - si sovrascrivono dati sullo stack
- l'overflow dallo stack scrive nella regione **data**
  - si sovrascrive **safe_script**
- XSS e RCE

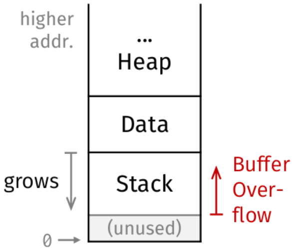




Chiamando **vuln()** con la stringa
"...alert('XSS')"

Sovrascrivere dati “costanti”

- variabili statiche salvate nella regione **data**
- **strcpy()** non effettua controlli sulla dimensione dell’input
  - si sovrascrivono dati sullo stack
- l’overflow dallo stack scrive nella regione **data**
  - si sovrascrive **safe_script**
- **XSS** e **RCE**



Chiamando **vuln()** con la stringa
";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;alert('XSS')"

Heap overflow (a partire dallo stack)

- similmente è possibile sovrascrivere dati sullo heap
- **libpng** contiene un buffer overflow

```
std::string img_tag =
  "<img src='";
pnm2png("input.pnm", "output.png");
img_tag += file_to_base64("output.png") + "'>";
emcc::global("document").call("write", img_tag);
```

- si può sovrascrivere **img_tag** nello heap
  - **XSS**

Heap overflow (a partire dallo stack)

- similmente è possibile sovrascrivere dati sullo heap
- **libpng** contiene un buffer overflow

```
std::string img_tag =
  "<img src='";
pnm2png("input.pnm", "output.png");
img_tag += file_to_base64("output.png") + "'>";
emcc::global("document").call("write", img_tag);
```

- si può sovrascrivere **img_tag** nello heap
  - **XSS**

Heap overflow (a partire dallo stack)

- similmente è possibile sovrascrivere dati sullo heap
- **libpng** contiene un buffer overflow

```
std::string img_tag =
  "<img src='";
pnm2png("input.pnm", "output.png");
img_tag += file_to_base64("output.png") + "'>";
emcc::global("document").call("write", img_tag);
```

- si può sovrascrivere **img_tag** nello heap
  - **XSS**

Heap overflow (a partire dallo stack)

- similmente è possibile sovrascrivere dati sullo heap
- **libpng** contiene un buffer overflow

```
std::string img_tag =
  "<img src='";
pnm2png("input.pnm", "output.png");
img_tag += file_to_base64("output.png") + "'>";
emcc::global("document").call("write", img_tag);
```

- si può sovrascrivere **img_tag** nello heap
  - **XSS**

Heap overflow (a partire dallo stack)

Select input PNM file to convert and show: Choose File No file chosen

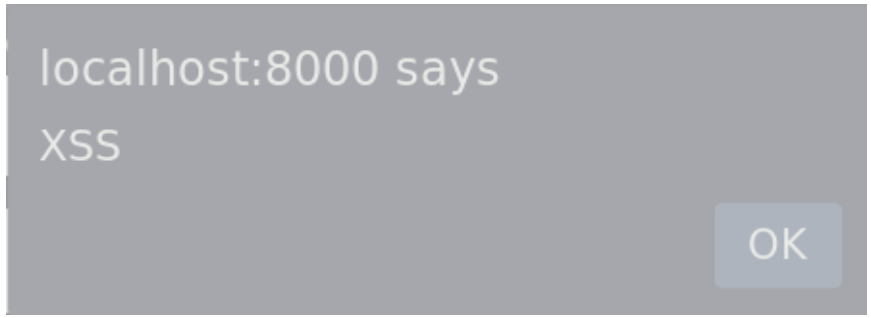
Il sito chiede all'utente un'immagine in input: non vengono fatti controlli sul tipo di file.

Se l'utente immette un'immagine, questa viene convertita e mostrata nel browser.

Utilizzando come payload un file contenente

AAAAAAAAAAAAAAAAAAAAAAAA...
<script>alert('XSS')</script><!- -

si effettua un overflow nello stack che permette di sostituire al tag **<img>** il tag **<script>** nello heap: questo provoca un attacco di tipo XSS.



Heap overflow (a partire dallo stack)

Select input PNM file to convert and show: Choose File No file chosen

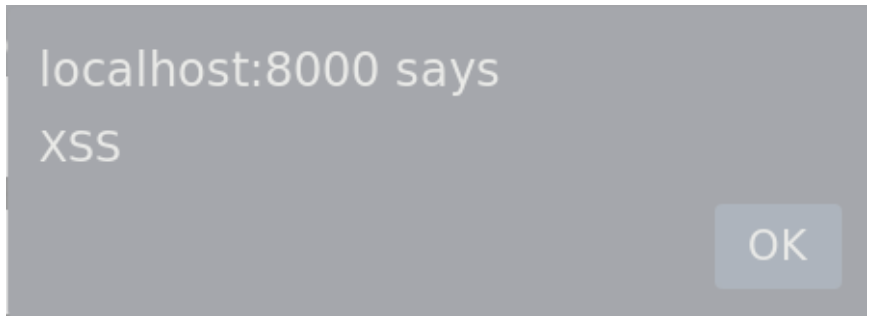
Il sito chiede all'utente un'immagine in input: non vengono fatti controlli sul tipo di file.

Se l'utente immette un'immagine, questa viene convertita e mostrata nel browser.

Utilizzando come payload un file contenente

AAAAAAAAAAAAAAAAAAAAAAAA...
<script>alert('XSS')</script><!- -

si effettua un overflow nello stack che permette di sostituire al tag **<img>** il tag **<script>** nello heap: questo provoca un attacco di tipo XSS.



Heap overflow (a partire dallo stack)

Select input PNM file to convert and show: Choose File No file chosen

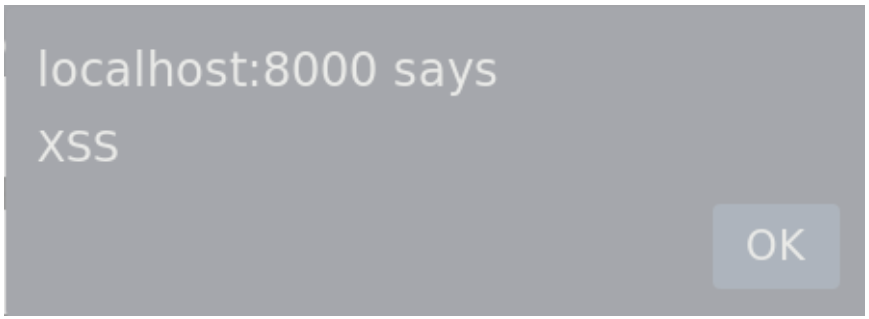
Il sito chiede all'utente un'immagine in input: non vengono fatti controlli sul tipo di file.

Se l'utente immette un'immagine, questa viene convertita e mostrata nel browser.

Utilizzando come payload un file contenente

AAAAAAAAAAAAAAAAAAAAAAAA...
<script>alert('XSS')</script><!- -

si effettua un overflow nello stack che permette di sostituire al tag **<img>** il tag **<script>** nello heap: questo provoca un attacco di tipo XSS.



Heap overflow (a partire dallo stack)

Select input PNM file to convert and show: Choose File No file chosen

Il sito chiede all'utente un'immagine in input: non vengono fatti controlli sul tipo di file.

Se l'utente immette un'immagine, questa viene convertita e mostrata nel browser.

Utilizzando come payload un file contenente

AAAAAAAAAAAAAAAAAAAAAAAA...
<script>alert('XSS')</script><!- -

si effettua un overflow nello stack che permette di sostituire al tag **<img>** il tag **<script>** nello heap: questo provoca un attacco di tipo XSS.

Conclusioni

WebAssembly ha

- reintrodotto vulnerabilità precedentemente mitigate
- introdotto nel web vulnerabilità legate ai binari
  - buffer/heap overflow
  - format string
  - ...
- reso possibili nuovi tipi di attacco
  - sovrascrivere dati "costanti"
  - sovrascrivere dati di una regione a partire da un'altra

Conclusioni

WebAssembly ha

- reintrodotto vulnerabilità precedentemente mitigate
- introdotto nel web vulnerabilità legate ai binari
  - buffer/heap overflow
  - format string
  - ...
- reso possibili nuovi tipi di attacco
  - sovrascrivere dati "costanti"
  - sovrascrivere dati di una regione a partire da un'altra

Conclusioni

WebAssembly ha

- reintrodotto vulnerabilità precedentemente mitigate
- introdotto nel web vulnerabilità legate ai binari
  - buffer/heap overflow
  - format string
  - ...
- reso possibili nuovi tipi di attacco
  - sovrascrivere dati "costanti"
  - sovrascrivere dati di una regione a partire da un'altra

Bibliografia

1. Daniel Lehmann, Johannes Kinder, Michael Pradel: *"Everything Old is New Again: Binary Security of WebAssembly"*
2. Brian McFadden, Tyler Lukasiewicz, Jeff Dileo, Justin Engler: *"Security Chasms of WASM"*