BROWSER FUZZING IN 2014
aka
Learn where to throw your stones

Rosario valotta
agenda

1. BROWSER FUZZING: THE STATE OF THE ART
2. INTRODUCTION OF A NEW fuzzing APPROACH
3. FUZZING WITH TIME
4. FUZZING cross-engine
5. ENJOY FILEJA!
6. SOME RESULTS
7. Repro tricks
Hello!

Daily job: project manager in mobile telco operator

Indipendent security researcher for fun and passion

Mainly focused on Web security and browser security

Speaker at sec conferences:
• HITB – Deepsec – nuit du hack – phDays – swiss cyber storm – syscan 360

https://sites.google.com/site/tentacoloviola/
Why fuzzing?

Quick answer: spotting memory corruption

Ubiquitous platform

Huge attack surface

Browsers are buggy

Not so fast boy...

…and why browsers?

WebKit is basically a collection of use-after-free bugs that somehow manages to render HTML (probably via a buffer overflow in WebGL)
David vs goliath

Be fast or die:
• Browser vendors are becoming quick to patch
• Big and competitive market

If you’re a lonely security researcher with a slingshot you cannot compete with that bug-killing armada out there…

Old fuzzing approaches don’t work any more

You need **new ideas** and a **new approach**, you need to know where to throw your stones
Dom fuzzing - RENDERING ENGINE

MOST FREQUENT FUZZING target

• Most complex module

RENDERS HTML, XML, SVG, MATHML, XML documents and images

RENDERS other data types via plug-ins / extensions
  • PDF
  • Media file
  • Fonts

Manages DOM

• Frequently patched/updated with new capabilities
DOM fuzzers

**Crossfuzz** by Michal Zalewski
- Set the standard for dom references fuzzing
- Lot of mods, widespread coverage
- Hard to spot new crashes

**Nduja fuzzy** by me
- Introduced some new concepts from DOM Level 2 and 3 specs
- Collections / Ranges
- NodeIterators / TreeWalkers
- Mutation Events

"Events are the ATM of IE vulnerabilities”
- Chen Zhang – Smashing the browser - 2014]
A new IDEA: targeting new modules

an extended perimeter for browser fuzzing going beyond RENDERING ENGINE

introducing time dependencies into your fuzzing logic:
1. Synch / Asynch events
2. Network interactions

race conditions
A new IDEA: targeting new modules

- Extend your fuzzing perimeter across multiple scripting ENGINES
- Can run multiple scripting engines simultaneously
- Objects can be referenced cross-context
- Fuzz cross-context references
Stone #1 - Fuzzing/with/TIME
JS RACE conditions

All modern browsers implement their JS engines using one os-thread.

The Only exception are web workers, but with little security risk as they don’t access DOM.

**Question:** Given that 2 js events cannot happen at the same time, do I really need to care about race conditions?

- **Short answer:** Yes
- **LONG ANSWER:** RACE conditions may still happen if your code execution is interrupted by events and some objects/functions you rely on are modified during the event handling.
JS RACE conditions

Several race condition vulns have been spotted in the past:
- APPLE WEBKIT - CVE-2012-3748
- Mozilla firefox - CVE-2006-4253
- Google chrome - CVE-2006-4253
- Microsoft IE - CVE-2011-1257
  But NO targeted fuzzing algorithm to stress race condition insurgence

Three main sources of trouble:

- events
- Set timeout
- Set interval
- Network interactions
Most every action performed on a browser results in an event being generated and appended to the event queue.

JS processes events from the queue one at a time (event loop).

JS is not I/O blocking.

Events are generally handled asynchronously:
- callbacks
- event handlers
- timeouts
- promises

Some special events (mutation) and events fired with `dispatchEvent` are synchronous.
An event handler is a JavaScript function that is registered to an object and an event name

```
myelem.addEventListener("MouseClick", myHandlerFunction, captureIsOn)
myelem.removeEventListener("MouseClick", myHandlerFunction, captureIsOn)
```

Dom holds a map of listeners and event handlers connected to each node keyed by event type

When the corresponding event is fired on the object, all event handlers registered on the node are executed

```
n.addEventListener("MouseClick", eh1, false)
n.addEventListener("MouseClick", eh2, false)
```

//mouse clicked
//call eh1()
//call eh2()
Events threat #1

Mutation events (DOM Level 3) are synchronous

Block js execution & immediately call handling routine

```javascript
function handler(e){
  //delete some js objects/ DOM elements
}
document.body.addEventListener("DOMNodeInserted",handler,false);
p=document.createElement("p");
document.body.appendChild(p);
//mutation event (synch) fires!
//use deleted objects
```

This may cause Use after free errors
Events PROPAGATION

When fired, Event objects are dispatched to an event target.

At the begin of the dispatch, browser determines the propagation path for the event.

Pp is a hierarchical list of dom nodes through which the event must pass.

Propagation path is divided in 3 phases:
1. Capture
2. Target
3. Bubble
Events threat #2

If registered for the event being propagated, other nodes belonging to pp can handle the event

- On bubble phase (if provided by specs for that event)
- On capture phase (if specified in `addEventListener` declaration)

```javascript
function handler(e){
  //handle event
}

body.addEventListener("DOMNodeInserted",handler,true);
p.addEventListener("DOMNodeInserted",handler,true);

//append some nodes under P
//mutation event fires on target P
//1. call handler #1 for BODY (capture=true)
//2. call handler #2 for P
//3 .call handler #3 for BODY (bubble)
```
Events threat #2

• “Once determined, the propagation path **MUST NOT** be changed. For DOM implementations, this applies even if an element in the propagation path is moved within the DOM or removed from the DOM”

[W3C DOM Level 3 Specifications]

```javascript
function handler(e){
  if (e.currentTarget != e.target){
    //DOM mutations here...
  }
  //handle event
}

body.addEventListener("DOMNodeInserted",handler,true);
p.addEventListener("DOMNodeInserted",handler,true);
//mutation event fires on target P
//1. call handler #1 for BODY (capture=true)
//2. call handler #2 for P
//3. call handler #3 for BODY (bubble)
```

Is the propagation continuably? ...UAF
Look ma’ more events!

Starting with DOM level 4, `MutationEvents` are deprecated for performance reasons

Introducing `MutationObservers`...

Every mutation is now queued in a `mutation` collection

```javascript
// select the target node
var target = document.querySelector('#some-id');
// create an observer instance
var observer = new MutationObserver(function(mutations) {
    mutations.forEach(function(mutation) {
        //whatever
    });
});
// configuration of the observer:
var config = { attributes: true, childList: true};
// pass in the target node and the observer options
observer.observe(target, config);
// later, you can stop observing
observer.disconnect();
```

Queue means... asynchronous
Events threat #4

Even if deprecated, MutationEvents are still usable

You can use both MutationEvents and MutationObservers on the same object

```javascript
function handler(e) {
  console.log("mutation event handler");
}
var target = document.body;
var observer = new MutationObserver(function(mutations) {
  mutations.forEach(function(mutation) {
    console.log("mutation observer");
  });
});
var config = {childList: true};
observer.observe(target, config);
target.addEventListener("DOMNodeInserted", handler, false);
p = document.createElement("p");
target.appendChild(p);
```

- Both handling routines will fire
  - Mutation Event handler will fire first
  - Observer handler is fired asynchronously
Events threat #4

SOME interesting APIs will provide an array/list of mutated nodes.

Both mutation and event objects provide references to mutated object:

- takeRecords
- addedNodes
- removedNodes

Mutated Node

- event.target
- event.currentTarget
- ...

Fuzzing mutation AND event objects not updated in atomic operations may lead to memory inconsistencies.

UAF again...
setTimeout threats

Add entropy to your fuzzing logic

Can introduce arbitrary delays in callbacks

When applied to event handlers and callback functions `setTimeout/setInterval` can alter execution order

Invert MutationObservers AND MutationEvents Handling order

`setTimeout(fn, 0)` will:
- Force fn to the next available tick in execution stack
- Force repaint/reflow  → useful for fuzzing with css properties

`setImmediate(fn)`
FUZZing with XHR

XHRs can be synch (deprecated) or asynch operations

XHRs state can be monitored using some event listeners:
-readystatechange, progress, abort, error, load, timeout, loadend

«An xhr object must not be garbage collected if its state is opened and the send flag is set or its state is loading or it has one or more event listeners registered»

In order to trigger GC before callback you can rely on synch events
XHR threats

body.addEventListener("DOMNodeInserted",mutationHandler,true);

xhr = new XMLHttpRequest();
xhr.addEventListener('progress',whatever,false);
//open and send xhr

xhr.onreadystatechange = function() {
  if (this.readyState == 4) {
    //fuzz with xhr object
  }
}

//create DOM node & append under body

function mutationHandler(e){
  //try whatever possible to free xhr
  //1. redefine xhr object
  //2. or unload/reload document
  //3. or removeEventListener from xhr
}
Fuzzing with network calls

The idea here is to combine classical approach of dom fuzzing with network calls

Snippets of valid Js are retrieved using `xhr`s or `ws`s and processed in context of the dom

On the server side there are a bunch of applications, implementing http and WS servers:

- Create `xhr/WS` object
- Open `xhr/WS`
- Send `xhr/WS`
- `Nodejs app`
- `js`
- `Eval (JS)`
- `Execute callback`
Client side logic

```javascript
xhr = new XMLHttpRequest();
xhr.open("GET", "http://127.0.0.1:8887", RBool());
xhr.onreadystatechange = function()
{
  if (this.readyState == 4) {
    var s=document.createElement("script");
    s.innerText=xhr.responseText;
    document.body.appendChild(s);
  }
}

socket = new WebSocket("ws://127.0.0.1:9999", "fuzz");
socket.addEventListener("message", function(event) {
  s=document.createElement("script");
  s.src=(window.URL.createObjectURL(new Blob([data],{type:"text/html"})))
  document.body.appendChild(s);
  f.contentWindow.eval(s.innerText);
});
```
server side logic

For every request:

1. A random **delay** is applied before generating the response → this affects timing on client side

2. A **fragment** of valid js is composed and returned to the client or...

3. ...a reference to a function declared on the client side is returned

```javascript
var snippet=[array of js snippets here...];
var responseBody = generateResponse();

function generateResponse(){
  //may be apply a delay before generating response
  returnString=generateScript(10);
}

function generateScript(num){
  //create and append «num» DOM elements
  or
  //recall some client fuzzer functions
  or
  //compose some random JS using snippets
}
Network calls threats

The evaluation of the js fragments is influenced by:

- synch DOM mutations occurred in the middle of call processing
- xhr/ws references disposed in the meanwhile
- race conditions in request/response management
Fuzzing with time - The masterplan

- Use Several namespaces
- Add listeners
- Tweak elements' attributes
- Collect element references
- Perform random mutations on DOM
- Trigger GC
- Crawl DOM references

- Create random elements
- Open xhr/WS
- Send xhr/WS
- Node.js app
- Eval (JS)

- Use mutation events and mutation observers
- Event handler
- Crossfuzz layer
- fileja layer
- js

- Also ON elements on collections
- Perform random mutations on DOM
- Crawl DOM references
Stone #2 - Fuzzing/cross-engine
IE’ SCRipting engines

In Windows, js is implemented as a **COM DLL** that can be hosted by various applications:

- Windows Script Host (wsh)
- Internet Explorer

Before version 9, IE used the same JavaScript engine as WSH: Jscript.dll
IE’ SCRipting engines

In IE9 a different DLL has been shipped – Jscript9.dll, designed specifically for the browser.

For backward compatibility, Jscript9 engine can emulate IE8 and IE7 doc modes, keeping the same behavior as legacy engine.

Emulating the legacy engine doesn’t actually mean loading the legacy engine!
Using legacy engines

By default when defining `<script>` or `<script language='javascript'>`, IE 9+ loads jscript9.dll

You can force IE to load legacy engine by declaring: `<script language='Jscript.Encode'>`

`Jscript.Encode` was designed for interpreting encoded scripts, but also works for clear text ones

IE also supports a Vbscript scripting engine managed always by jscript.dll `<script language='vbscript'>`

Vbscript is deprecated starting with IE11, but you can summon it back forcing browser to work in emulation mode of previous versions
Using legacy engines

IE9+ doesn’t support `Jscript.Encode` by default

In order to be able to load `Jscript.Encode` scripts you have to force IE=8 emulation

```html
<html>
<head>
<title>test encode</title>
<meta http-equiv="X-UA-Compatible" content="IE=8"></meta>
</head>
<body>

<script language="Jscript.Encode">
  function a(){alert(1);}
</script>

<script> a();</script>
</body>
</html>
```
Host - engine interactions

Communication between the host and the script engine is carried on using these interfaces: IActiveScript, ActiveScriptSite, iDispatch

ActiveScriptSite is implemented on the host side and enables the js engine to call its host.

ActiveScript, implemented on the script engine side, provides necessary function calls to initialize the engine.

IDispatch interface is implemented on both sides and is used to retrieve handles of objects and execute get, set and function calls operations on them.
Cross engines interactions

When a `Jscript.Encode` script is found, IE9 hosts both Jscript9 and Jscript engines at runtime.

Both engines can talk to the other one.

IE uses `IDispatch` interface for resolving cross-engines references.
Cross-engine Threats

Any engine has no knowledge of the status of objects created in other engines' contexts.

Objects could have been deleted on the other engine.

The whole other engine context could have been deleted.

It is the host's responsibility to maintain consistency among objects and objects' references in different scripting contexts.

Guess what?
Threat model

To trigger memory corruption the Strategy is to use a classic dom fuzzing approach but all crawling-tweaking and mutating operations are performed cross-engine

**Parent window**
Jscript9.dll

- Create random elements
- Tweak elements attributes
- Collect element reference
- Perform random mutations on DOM
- Trigger GC
- Crawl DOM reference

**Child window (oR iframe)**
Jscript.dll

- Create random elements
- Tweak elements attributes
- Collect element reference
- Perform random mutations on DOM
- Trigger GC
- Crawl DOM reference
Introducing fileja

A prototype combining a traditional dom fuzzing approach with time events and cross engines fuzzing

Written in javascript & Nodejs

Tested on grinder framework but totally agnostic from fuzzing framework

Applied on nduja-like fuzzer

6 months of full time testing on my PC with a couple of win7 vms mainly on ie11 and chrome

4 exploitable bugs found in the first week, 13 more in the following months
SOME RESULTS #1

Registers:
- EAX = 0x45454545 -
- EBX = 0x0303FCC0 - RW-
- ECX = 0x68EB08B3 - R-X - jscript9!NativeCodeGenerator::CheckCodeGen
- EDX = 0x019AD684 - RW-
- ESI = 0x00000003 -
- EDI = 0x0AC2F89C - RW-
- EBP = 0x0AC2F720 - RW-
- ESP = 0x0AC2F6D4 - RW-
- EIP = 0x45454545 -

Call Stack:
- 0x68EAD364 - jscript9!Js::JavascriptFunction::CallRootFunction
- 0x68EAD2B6 - jscript9!ScriptSite::CallRootFunction
- 0x68EAD23D - jscript9!ScriptSite::Execute
- 0x68EAEC1 - jscript9!ScriptEngineBase::ExecuteInternal<0>
- 0x68EAEBFD - jscript9!ScriptEngineBase::Execute
- 0x659DE34E - mshtml!CMutationObserver::PerformMicrotaskCheckpoint
- 0x659DE284 - mshtml!CObserverManager::InvokeObserversForCheckpoint
- 0x655AA45DE - mshtml!GlobalWndOnMethodCall
- 0x654B7C5E - mshtml!GlobalWndProc
- 0x753CC4E7 - user32!InternalCallWinProc
- 0x753CC5E7 - user32!UserCallWinProcCheckWow
- 0x753CCC19 - user32!DispatchMessageWorker
- 0x753CCC70 - user32!DispatchMessageW
- 0x65DD8DDC - mshtml!ModelessThreadProc
SOME RESULTS #2

Registers:

EAX = 0x02F7AE34 - RW-
EBX = 0x00000000 - 
ECX = 0x771B179F - R-X - ntdll!vDbgPrintExWithPrefixInternal
EDX = 0x02F7ABD1 - RW-
ESI = 0x00620000 - RW-
EDI = 0x00645478 - RW-
EBP = 0x02F7AE9C - RW-
ESP = 0x02F7AE24 - RW-
EIP = 0x77253873 - R-X - ntdll!RtlReportCriticalFailure

Call Stack:

0x772547A3 - ntdll!RtlpReportHeapFailure
0x77254883 - ntdll!RtlpLogHeapFailure
0x77219D8A - ntdll!RtlpCoalesceFreeBlocks
0x771E6287 - ntdll!RtlpFreeHeap
0x771E65A6 - ntdll!RtlFreeHeap
0x761FC3C4 - kernel32!HeapFree
0x020E0034 - 
0x03DA9539 - 
0x03DA3167 -
0x673FCEAB - jscript9!Js::JavascriptFunction::CallFunction<1>
0x674B46F2 - jscript9!Js::InterpreterStackFrame::Process
0x67552226 - jscript9!Js::InterpreterStackFrame::OP_TryCatch
0x674B4712 - jscript9!Js::InterpreterStackFrame::Process
0x67400AA3 - jscript9!Js::InterpreterStackFrame::InterpreterThunk<1>
SOME RESULTS #3

Registers:

- EAX = 0x02C4400
- EBX = 0x00000000
- ECX = 0x047C87F8
- EDX = 0x02C40000
- ESI = 0x00000000
- EDI = 0x08A2BDA0
- EBP = 0x08315D50
- ESP = 0x08315D44
- EIP = 0x62ACBE7D

Code:

```assembly
0x62ACBE7D - mov eax, [edx]
0x62ACBE7F - push edx
0x62ACBE80 - call dword ptr [eax+8]
0x62ACBE83 - jmp 62a7f579h
0x62ACBE88 - mov eax, [ecx]
0x62ACBE8A - push ecx
0x62ACBE8B - call dword ptr [eax+8]
0x62ACBE8E - jmp 62a7f58ah
```

Call Stack:

- 0x62A7F6FD - mshtml!CWindow::SetTimeoutHelper
- 0x62A7F914 - mshtml!CWindow::SetTimeoutFromScript
- 0x62A7F9A6 - mshtml!CFastDOM::CWindow::Trampoline_setTimeout

IE 11
Type: r.A. Violation
Exploitable: YES
SOME RESULTS #4

Registers:
  EAX  = 0x00004444  -
  EBX  = 0x0086EAB8  - RW-
  ECX  = 0x009320BC  - RW-
  EDX  = 0x032D1A01  - RW-
  ESI  = 0x00000001  -
  EDI  = 0x0020003A  -
  EBP  = 0x02EBC3F8  - RW-
  ESP  = 0x02EBC3E8  - RW-
  EIP  = 0x6A362B1F  - R-X  - mshtml!CScriptData::AsyncFireOnError

Code:
  0x6A362B1F  - call dword ptr [eax]
  0x6A362B21  - mov esi, eax
  0x6A362B23  - lea ecx, [ebp-8]
  0x6A362B26  - push 0
  0x6A362B28  - push esi
  0x6A362B29  - call mshtml!CElement::CLock::CLock
  0x6A362B2E  - push -1
  0x6A362B30  - mov ecx, esi

Call Stack:
  0x69EF1A3C  - mshtml!GlobalWndOnMethodCall
  0x69ED9A52  - mshtml!GlobalWndProc
SOME RESULTS #5

Registers:

EAX = 0x00454545
EBX = 0x00000000
ECX = 0x00410000
EDX = 0x00000008
ESI = 0x0020F000
EDI = 0x0020F000
EBP = 0x0015E494
ESP = 0x0015E47C
EIP = 0x00454545

Call Stack:

0x668907A2 - chrome_child!WTF::DefaultAllocator::backingMalloc<WebCore::SVGTextChunk *,void>
0x66CE6896 - chrome_child!WebCore::MutationObserver::enqueueMutationRecord
0x66CEE2B6 - chrome_child!WebCore::MutationObserverInterestGroup::enqueueMutationRecord
0x66AF3188 - chrome_child!WebCore::Element::willModifyAttribute
0x66B8B668 - chrome_child!WebCore::Element::setAttribute
0x66B8B29A - chrome_child!WebCore::ElementV8Internal::setAttributeMethod
0x66B8AFFE - chrome_child!WebCore::ElementV8Internal::setAttributeMethodCallback

Chrome 35
Type: D.E.P.
Violation
SOME RESULTS #6

Registers:
EAX = 0x68CC0056 - R-X - mshtml!CWritableCSSStyleDeclaration::`vftable'
EBX = 0x03A3BEE0 - RW-
ECX = 0x03DF3BA8 - RW-
EDX = 0x00000000 -
ESI = 0x00000000 -
EDI = 0x02D0755C - RW-
EBP = 0x02D0753C - RW-
ESP = 0x02D07504 - RW-
EIP = 0x45454545 -

Call Stack:
0x68FBCB63 - mshtml!CJScript9Holder::VarToVARIANT
0x6985FEED - mshtml!CFastDOM::CSSStyleDeclaration::Trampoline_Set_stopColor
0x6AA0858B - jscript9!Js::JavascriptExternalFunction::ExternalFunctionThunk
0x6AA090BA - jscript9!Js::JavascriptOperators::CallSetter
0x6AA3F815 - jscript9!Js::JavascriptOperators::SetProperty_Internal<0>
0x6AA3FDE7 - jscript9!Js::JavascriptOperators::OP_SetProperty
0x6AA3FE43 - jscript9!Js::JavascriptOperators::PatchPutValueNoFastPath
0x6AA3F931 - jscript9!Js::InterpreterStackFrame::Process
0x6AA07028 - jscript9!Js::InterpreterStackFrame::InterpreterThunk<1>

IE 11
Type: D.E.P. Violation
Caught a Write Access Violation in process 5896 at 2014-04-12 17:51:14 with a crash hash of D17449B5.C819B416

Registers:
EAX = 0x0016F17C - RW-
EBX = 0xFFFCE3CE
ECX = 0x0016F17C - RW-
EDX = 0x00000004 -
ESI = 0x00000000 -
EDI = 0x00C7B000 -
EBP = 0x0016F15C - RW-
ESP = 0x0016F14C - RW-
EIP = 0x6386228C - R-X

Code:
0x6386228C - mov [edi+4], ebx
0x6386228F - call 638622ech
0x63862294 - mov ecx, [ebp+8]
0x63862297 - mov [edi], ebx
0x63862299 - mov [ecx], edi
0x6386229B - add esp, 4
0x6386229E - pop esi
0x6386229F - pop ebx
SOME RESULTS #8

Registers:

EAX = 0xFEEEEE
EBX = 0x02EAA84C - RW-
ECX = 0x02EAA750 - RW-
EDX = 0x02EAA750 - RW-
ESI = 0x02EAA840 - RW-
EDI = 0x72D9102D - R-X - pthread_mutex_unlock
EBP = 0x024AF448 - RW-
ESP = 0x024AF440 - RW-
EIP = 0x66ECD873 - R-X - CFHostUnscheduleFromRunLoop

Code:

0x66ECD873 - mov edx, [eax+1ch]
0x66ECD876 - call edx
0x66ECD878 - pop ebp
0x66ECD879 - ret
0x66ECD87A - int 3
0x66ECD87B - int 3
0x66ECD87C - int 3
0x66ECD87D - int 3

Stack:

0x66ECA6FE - CFHostUnscheduleFromRunLoop
0x66ECA676 - CFHostUnscheduleFromRunLoop
0x66EC8F8F - CFHostUnscheduleFromRunLoop
SOME RESULTS #9

Registers:

- EAX = 0x082CFBA8 - RW-
- EBX = 0x00680AAC - RW-
- ECX = 0x45454546 -
- EDX = 0x45454541 -
- ESI = 0x082CFBA8 - RW-
- EDI = 0x00000281 -
- EBP = 0x0D49CF2C - RW-
- ESP = 0x0D49CF1C - RW-
- EIP = 0x6AAEF775 - R-X - ieframe!CImpWndProc::s_WndProc

Code:

- 0x6AAEF775 - call dword ptr [ecx]
- 0x6AAEF777 - push dword ptr [ebp+14h]
- 0x6AAEF77A - mov edx, [esi]

Stack:

- 0x75E6C4E7 - user32!InternalCallWinProc
- 0x75E6C5E7 - user32!UserCallWinProcCheckWow
- 0x75E65294 - user32!SendMessageWorker
- 0x75E65582 - user32!SendMessageW

- 0x6AB1D18D - ieframe!EnumInvokeCallback
- 0x6AB1CFE9 - ieframe!EnumConnectionPointSinks
- 0x6AB1CEC7 - ieframe!IConnectionPoint_InvokeIndirect

Type: R.A.Violation
Exploitable: YES
Closing notes

Fileja is publicly available at https://sites.google.com/site/tentacoloviola/fileja-fuzzer

Pick the code and play with it
• Use your creativity
• Read w3c specifications
• Fuzz with new browser features

Not widely tested areas:
• non-windows platforms
• mobile devices
• Server side events
• Object.observe()
• Cross-engine fuzzing with VBScript
Thank you

valotta.rosario@gmail.com

@tentacolo_Viola

https://sites.google.com/site/tentacoloviola/