Dumb fuzzing XSLT engines in a smart way

Me & Myself

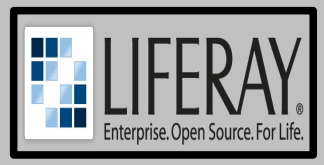
Founder & owner of Agarri

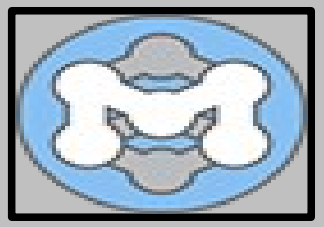
Consulting, training, ...

Before 2011

Me & XML

2011: XSLT abuse of features + XML External Entities






2012: XSLT implementation bugs

- At least 3 components: XML, XSLT, XPath
- May include DOM interaction, XInclude, XPointer, ...

Dumb fuzzing XSLT engines in a smart way

“Dumb fuzzing”

Aka “monkey typing”

Aka “mutation-based”

Should I really explain?

“Smart way”

Anything which could allow a lonesome hacker to beat, in his free time, the security teams of some large vendors

My - 

- No SAGE, LangFuzz
- nor ClusterFuzz

My + 

- Motivated
- Good knowledge of XSLT

Overview

Introduction

XSLT applications and vectors

The fuzzing campaign

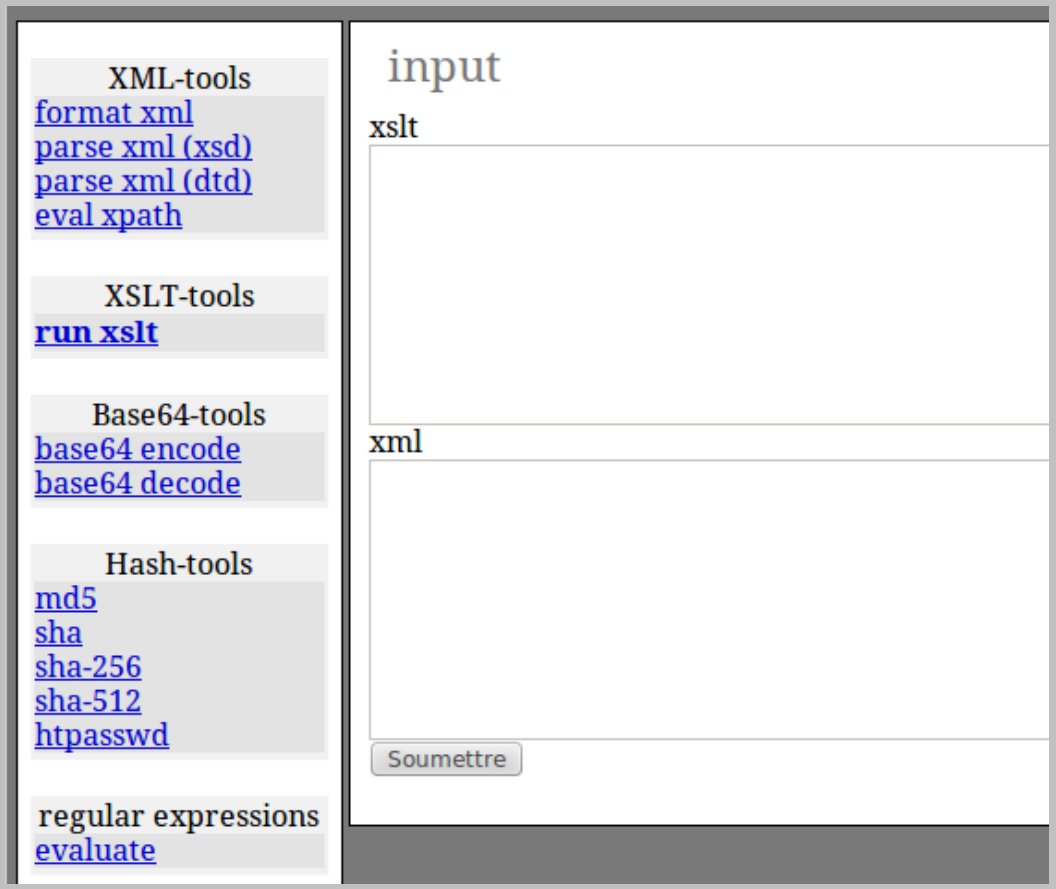
Summary

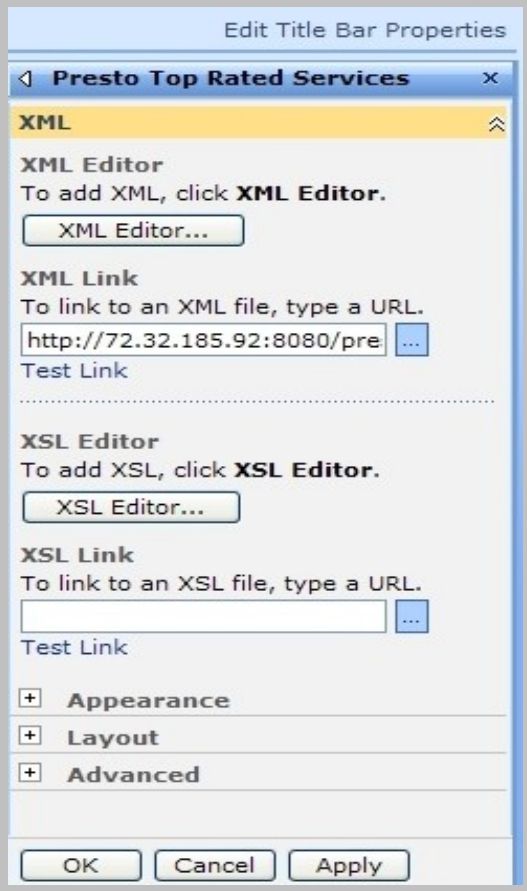
Future work

XSLT apps & vectors

DOTNETNUKE™

Microsoft® SharePoint®





XSLT apps & vectors

XML doc with a PI pointing to external or embedded XSLT

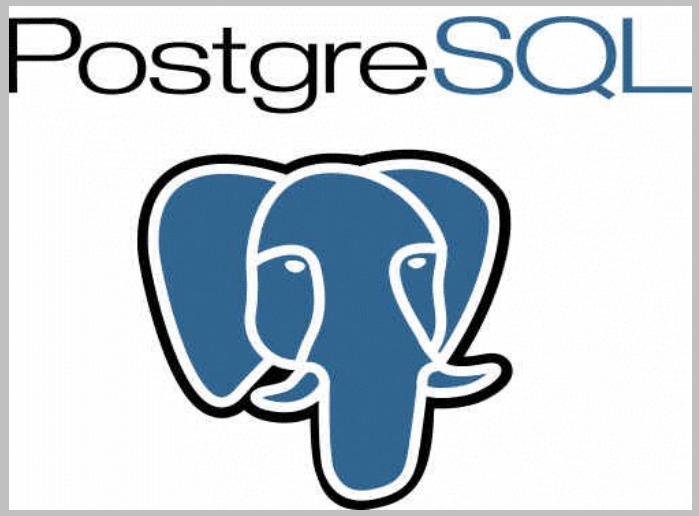
Via JavaScript / MSIE has its own syntax

```
<?xml-stylesheet type="text/xsl" href="#pwn" ?>
<!DOCTYPE foo [
  <!ATTLIST xsl:stylesheet id ID #IMPLIED>
]>
<foo>[...]<xsl:stylesheet id="pwn" [...]</foo>
```

XSLT apps & vectors


```
<ds:Signature […]

    <ds:Transform Algorithm =
  "http://www.w3.org/TR/1999/REC-xslt-19991116">

        <xsl:stylesheet […]

  </ds:Signature>
```

XSLT apps & vectors




```
select xslt_process(
    $$<foo/>$$::text,
    $$<xsl:stylesheet[...]$$::text
);
```

XSLT apps & vectors


```
select xmltransform(
 xmltype('<foo/>'),
 xmltype('<xsl:stylesheet [... ]')
) from dual;
```

XSLT apps & vectors

JavaScript code


```
var xslt = xfa.data.nodes.item(0)
    .nodes.item(1).saveXML();
var res = xfa.data.nodes.item(0)
    .nodes.item(0).applyXSL(xslt);
```

XSLT apps & vectors

XFA data

```
<xfa:datasets xmlns:xfa=
"http://www.xfa.org/schema/xfa-data/1.0/">
<xfa:data>[...]
        <xml>Da XML Data</xml>
        <xsl:stylesheet [...]
```

Analysis

Fuzzing long-running applications is hard 

Management, resources, reproducibility

Starting large software is costly 

Adobe Reader, browsers

Vectors are complex 

Context-dependent escaping ($$ in PostgreSQL)

XSLT stored in a XML container (Reader, xmlsec)

Oh, an idea!

Libxslt is a Open-Source XSLT engine

Provides a CLI wrapper, xsltproc

Fuzzing the wrapper could solve many problems!

- Performance
- Reproducibility
- Passing mutated XSLT code

Next steps

Identify the XSLT engine used in each common application listed earlier

Setup a basic/fast CLI wrapper for each engine

Identify XSLT engines

The usual way: docs, code, strings, behavior

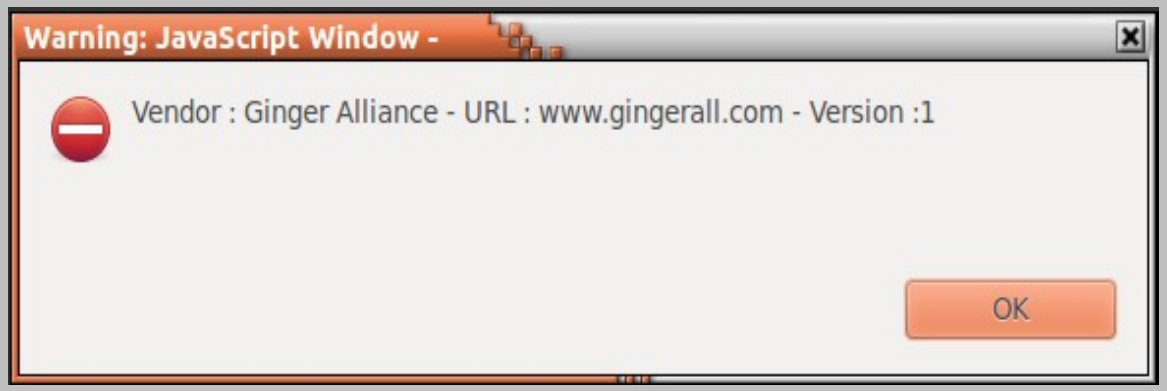
The XSLT way: introspection via system-property()

- Standardized info
  - xsl:vendor / xsl:vendor-url / xsl:version
- Extensions
  - sablotron:version / adobe:version / msxml:version

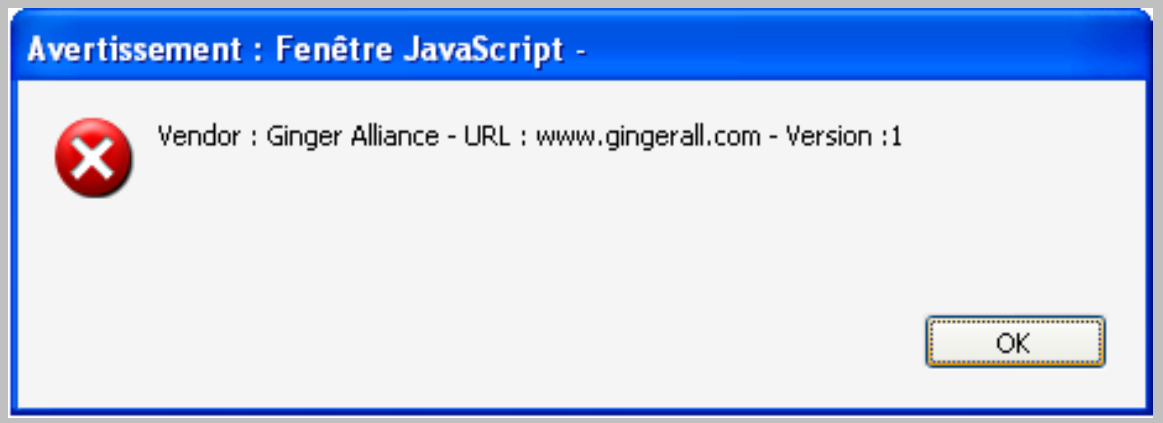
Identify XSLT engines

Adobe Reader uses an old open-source XSLT engine 

v9.5.4 / Unix



v10.1.4 / Windows



Identify XSLT engines

But they have some Secure Devloppment skills

```
* Various code hygiene changes:
    - eliminated all write accesses to constant strings
    - eliminated the creation of unterminated strings
    - eliminated all potential buffer overruns
    - eliminated all known potential floating-point and integer overflows
    - replaced some hard-coded constants for initial list sizing with e.g. LIST_SIZE_2
    - eliminated all newly discovered memory leaks
```

http://partners.adobe.com/

Engines	Applications
Transformiix	Firefox
Libxslt	Chrome Safari xmlsec PostgreSQL PHP 5
Presto	Opera
Oraxsl	Oracle SQL
MSXML	Internet Explorer Microsoft Sharepoint and Office DotNetNuke
Sablotron	Adobe Reader PHP 4

Setup CLI wrappers

Open source:

- Libxslt & Sablotron => wrapper provided 
- Transformiix => no CLI 

Closed source:

- Oraxsl & Intel => binary provided 
- MSXML => VBS script

Generate test cases

Mutation done via Radamsa

Why Radamsa?

Microsoft Excel

- 3 hours to find an exploitable bug
- Handling of SLK files aka MS11-045 / CVE-2011-1276

Banking applications using an unknown protocol

- 1 hour to found remotely triggerable overflows
- From a single captured packet!

Input files

Covered features

- Mostly XSLT 1.0, some other versions like 0.9, 1.1 and 2.0
- Some extensions from EXSLT or vendor-specific

Sources

- Standardized features (interoperability suites, unit tests)
- Public bugs from bug trackers and forums
- My research from 2011, incl. crypto functions in libxslt

Computing resources

4 “small but free” VM

- Amazon AWS Free Usage Tier / Microsoft Azure via MSDN

1 high-CPU VM

- Amazon AWS “c1.medium”
- 2 months 24/7, a few hundred euros

No human intervention

A Python script using WinAppDbg manages everything

- Generate test cases
- Run X parallel CLI engines with a timeout
- Wait for crashes
- Classify using a few heuristics
- Generate a report and send it by email (stored as Maildir)

Crashes

```
<xsl:stylesheet xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform"
version="2.0">

    <xsl:template match="/">
        <xsl:value-of
            select="1094861636 idiv 1.0"/>
    </xsl:template>

</xsl:stylesheet>
```

Crashes

Second Chance Exception Type:

0xC0000005

Exception Faulting Address:

0x41424344

1094861636 = 0x41424344

Crashes

```
EAX: 0xBFFFD470  EBX: 0x08298B40  ECX: 0xB2DEC151
EDX: 0x41424344  ESI: 0x082DDFD4  EDI: 0x082B2340
EBP: 0xBFFFD59C  ESP: 0xBFFFD470  EIP: 0x080B8FD4
```

```
gdb$ bt
#0  0x080b8fd4 in xxxxxx ()
#1  0x61626364 in ?? ()
#2  0x082ddfd4 in ?? ()
#3  0x0000002f in ?? ()
```

Crashes

```
<!DOCTYPE whatever [
  <!ATTLIST magic blabla CDATA "anything">
  <!ENTITY foobar "abcdefgh****ijklmnop">
]>
```

```
<magic xsl:version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform"/>
```

Crashes

```
#0  xmlStrEqual__internal_alias (
str1=0x2a2a2a2a <Address 0x2a2a2a2a out of bounds>,
str2=0x1cf444 "http://www.w3.org/1999/XSL/Transform"
) at xmlstring.c:162
```

```
#1  0x001aa384 in xsltParseTemplateContent
(style=0x805cc58, templ=0x80598e8) at xslt.c:4849

#2  0x001ac824 in xsltParseStylesheetProcess
(ret=0x805cc58, doc=0x80598e8) at xslt.c:6456

#3  0x001acd2c in xsltParseStylesheetImportedDoc
(doc=0x80598e8, parentStyle=0x0) at xslt.c:6627
```

Crashes

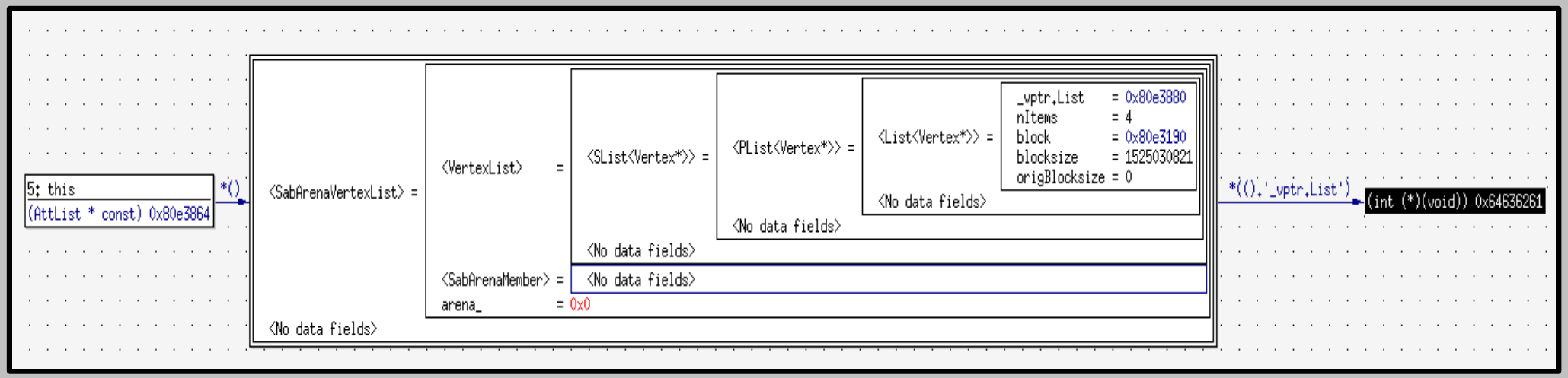
```
<xsl:template match="node()">
   <xsl:apply-templates
      select="node()[lang('foo')]"/>
</xsl:template>


<x>
   <abcd/>
</x>
```

Crashes

```
Program received signal SIGSEGV, Segmentation fault.
AttList::findNdx (this=0x6c6ef8, attName=...) at verts.cpp:1282
1282            if (attName == a -> getName())
(gdb) p/x $rdi
$2 = 0x64636261
(gdb) x/3i $rip
=> 0x415d24 <_ZN7AttList7findNdxERK5QName+96>:    mov    (%rdi),%rax
   0x415d27 <_ZN7AttList7findNdxERK5QName+99>:    callq  *0x40(%rax)
   0x415d2a <_ZN7AttList7findNdxERK5QName+102>:   mov    %rax,%rsi
```

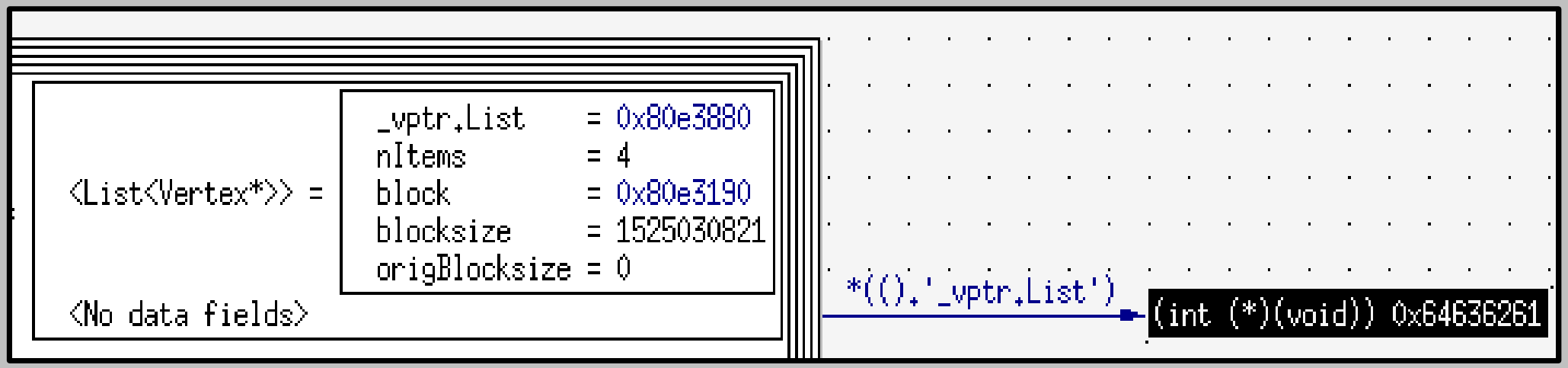
Crashes

GNU DDD, an useful visualization tool



Crashes

GNU DDD, zoomed



Crashes

The bug works perfectly well on Reader X

```
01EE2C36   33FF            XOR EDI,EDI
01EE2C38   85DB            TEST EBX,EBX
01EE2C3A   7E 1E           JLE SHORT AXSLE.01EE2C5A
01EE2C3C   8B46 08         MOV EAX,DWORD PTR DS:[ESI+8]
01EE2C3F   8B0CB8          MOV ECX,DWORD PTR DS:[EAX+EDI*4]
01EE2C42   8B01            MOV EAX,DWORD PTR DS:[ECX]
01EE2C44   FF50 1C         CALL DWORD PTR DS:[EAX+1C]
01EE2C47   8B4C24 10       MOV ECX,DWORD PTR SS:[ESP+10]
01EE2C4B   50              PUSH EAX
01EE2C4C   E8 173FFEFF     CALL AXSLE.01EC6B68
01EE2C51   85C0            TEST EAX,EAX
```

```
Registers (FPU)
EAX 61616161 AcroRd_1.61616161
ECX 6F007400
EDX 04F70388
EBX 46784578
ESP 0012B290
EBP 0012B8F8
ESI 04F935E0 ASCII "xCxDxExFaaaaGxHxIx"
EDI 00000000

EIP 01EE2C42 AXSLE.01EE2C42
```

Non crashes

Looking only for crashes, I missed

- Uninitialized memory
- Use After Free & Double Free
- Small heap/stack/global overflows and underflows

Tools depend on the context (OS, code available)

Non crashes

OS	Src ?	Engine	Toolbox
Windows	No	Intel MSXML	Gflags Dr. Memory
Linux	No	Oraxsl	Valgrind Efence
Linux	Yes	Libxslt Sablotron Transformiix	ASan

Non crashes

```
<xsl:variable
   name="foo" as="xs:string *">
   <xsl:sequence
        select="'Do NOT remove me!'"/>
</xsl:variable>

<xsl:value-of
     select="remove($foo,1)" />
```

Non crashes

```
Error #1: UNADDRESSABLE ACCESS: reading 0x02e2d228-0x02e2d229 1 byte(s)
# 0 xslt2cmd.exe!?      +0x0       (0x00655710 <xslt2cmd.exe+0x255710>)
# 1 xslt2cmd.exe!?      +0x0       (0x0064e645 <xslt2cmd.exe+0x24e645>)
# 2 xslt2cmd.exe!?      +0x0       (0x004ce845 <xslt2cmd.exe+0xce845>)
# 3 xslt2cmd.exe!?      +0x0       (0x004e2c28 <xslt2cmd.exe+0xe2c28>)
# 4 xslt2cmd.exe!?      +0x0       (0x004028f7 <xslt2cmd.exe+0x28f7>)
# 5 napa2::tick         +0x4b5101  (0x010c4cde <xslt2cmd.exe+0xcc4cde>)
[...]
Note: next higher malloc: 0x02e2e058-0x02e2ef38
```

Note: 0x02e2d228-0x02e2d229 overlaps memory 0x02e2d158-0x02e2de60 that was freed

Non crashes

```
<xsl:template match="/">
 <xsl:value-of select="
 format-number(1 div HUGE, '#')"/>
</xsl:template>
```

Non crashes

```
ERROR: AddressSanitizer heap-buffer-overflow on address 0x7fafa86aa2dd at pc
0x7fafd761d317 bp 0x7ffff9c0aeb0 sp 0x7ffff9c0aea8

READ of size 1 at 0x7fafa86aa2dd thread T0
    #0 0x7fafd761d317 (/home/nicob/Asan/firefox/firefox+0x7fafd761d317)
    #1 0x7fafd74b0100 (/home/nicob/Asan/firefox/firefox+0x7fafd74b0100)
    #2 0x7fafd75f8989 (/home/nicob/Asan/firefox/firefox+0x7fafd75f8989)
    […]

0x7fafa86aa2dd is located 163 bytes to the left of 31-byte region
[0x7fafa86aa380,0x7fafa86aa39f) allocated by thread T0 here:
    #0 0x42cb94 (/home/nicob/Asan/firefox/firefox+0x42cb94)
    #1 0x7fafeced862e (/home/nicob/Asan/firefox/firefox+0x7fafeced862e)
    #2 0x7fafd76187f8 (/home/nicob/Asan/firefox/firefox+0x7fafd76187f8)
    [...]
```

Non crashes

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:cry="http://exslt.org/crypto"
  extension-element-prefixes="cry">


    <xsl:template match="/">
       <xsl:value-of select="cry:rc4_decrypt('simple_demo',
'0262ee34196ae2df1ab850c1705ee0c38dcaaaa034021015446ae42bb
eecf140dea99675fb35539a4dcbeaf5c2e6a6cae679843dbf3650275a2
147483647be07464047dc17eff2661b8f065f0ae3abcd3b33e9fd3c1a3
6f2201ae65e093fa45b0a1b55cd408ec815a8dada050b8881b99e95770
4dc5f17208d105966680a26f')"/>
    </xsl:template>


</xsl:stylesheet>
```

Non crashes

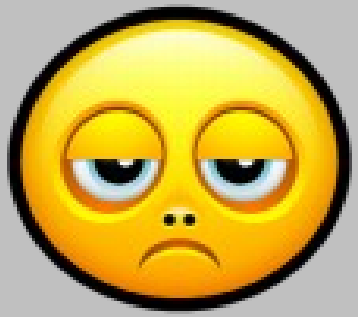
```
AddressSanitizer heap-buffer-overflow on address 0x7f6450acde02 at
pc 0x40b4c1 bp 0x7fff5c2175a0 sp 0x7fff5c217588

WRITE of size 1 at 0x7f6450acde02 thread T0

    #0  000000000040b4c1 <memcpy+0xd1>:
  40b4c1:       48 89 df                mov    %rbx,%rdi

0x7f6450acde02 is located 2 bytes to the right of 128-byte region
[0x7f6450acdd80,0x7f6450acde00) allocated by thread T0 here:

    #0  0000000000413ee2 <malloc+0x22>:
  413ee2:       4c 8d b5 d8 fd ff ff    lea    -0x228(%rbp),%r14
```

Summary - Before

Large applications

Complex vectors

Graphical user interfaces



Complicated & costly fuzzing

Summary - After

Source-code available for Reader's engine

CLI wrappers (5 out of 6 engines)

ASan instrumentation (3 out of 6)

Easy vector, logging and repro

Free cloud resources

Reports via email

Results

CVE-2012-1525

CVE-2012-1530

CVE-2012-2825

CVE-2012-2870

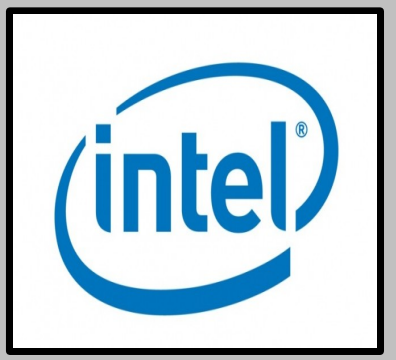
CVE-2012-2871

CVE-2013-0007

aka MS13-002

CVE-2012-0449

CVE-2012-3972



No CVE / unpatched

... and more ...

Future work

- Focus on DTD and embedded network clients
  - Already found a few bugs 
- Try some other execution contexts
  - DOM interaction / embedded stylesheet
- Use a fuzzer dedicated to interpreters
  - LangFuzz by Mozilla

That's all, folks!

Thanks for your attention

Any questions?

@Agarri_FR

nicolas.gregoire@agarri.fr