HackerPraktikum - RUB

Nov 2017

Nearly Generic Fuzzing of XML-based formats

Nicolas.Gregoire@agarri.fr

@agarri_fr

Me

- Hacking stuff since 1998
- Current skills
  - Web hacking
    - Published about XXE and SSRF
  - Teaching
    - Official Burp Suite Pro training partner
  - Fuzzing
    - Mostly client-side nowadays

Younger Me Vs XSLT
On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

@agarri_fr

Younger Me Vs XSLT

On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

@agarri_fr

Abuse of features

- Talk *"Offensive XSLT"* (2011)
  - No memory corruption, simply abuse the features
    - Using hand-crafted XSLT stylesheets
  - Highly reliable exploits
    - Read and create files, execute arbitrary code
- Positive side effect
  - Produced a large corpus covering most features
    - Combining nodes, attributes and namespaces
    - <sx:output file="/tmp/pwned">31337</sx:output>

Basic mutation-based fuzzing

- Talk *"Dumb-fuzzing XSLT engines"* (2013)
  - Reuse XSLT corpus from 2011
  - Mutate with radamsa
  - Minimalist tools
    - Linux: ASan + bash + grep
    - Windows: Python + WinAppDbg
  - Limited depth, found some bugs anyway
- Take-away
  - Producing XML for fuzzing purposes is hard!

Younger Me Vs XSLT
On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

@agarri_fr

Reuse of code fragments

- Aimed at fuzzing of interpreters
  - Tested on JavaScript, PHP and Ruby
- Christian Holler @mozdeco (2012)
  - Paper: *"Fuzzing with Code Fragments"*
  - Toolbox: LangFuzz (only for Mozilla and Google)
- Sean Heelan @seanhn
  - Talk: *"Ghosts of Christmas Past"* (2014)
  - Toolbox: Malamute (on Github), FragFuzz (private)

Production of testcases

- QuickFuzz Project (2015)
  - Chain different production steps
    - Generate then (dumbly) mutate then fix
- Grammar-based generation
  - Using Haskell’s QuickCheck and Hackage
- Dumb mutation
  - Using off-the-shelf tools like zzuf or radamsa
- Fix-up (aka "variable coherence")
  - Every referenced variable must first be declared

Guided fuzzing

- American Luzzy Lop (2013)
  - By Michael Zalewski @lcamtuf
  - Easy to use, hard to master (Bushnell's Law)
- Disadvantages
  - Doesn't (natively) run on Windows
  - Mutation engine aimed at binary / dense formats
- Advantages
  - Impressive track record
  - Large and active community
    - Forks (Pythia, WinAFL), patches (external mutators), helpers

Python mutators for AFL

- External mutation routines
  - By Christian Holler @decoder (2016)
- New fuzzing stage calling a Python module
- This module implements a custom mutator
  - init() called once
    - Execute all the costly initialization tasks
  - fuzz() called for each mutation
    - Should be as fast as possible
  - Optional trimming API
    - Applies format-aware minimization

Younger Me Vs XSLT
On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

@agarri_fr

GOALS

- Hierarchical mutations
  - High-level → Manipulate the structure (here the XML DOM)
  - Medium-level → Dialect-specific mutations (variable coherence, ...)
  - Low-level → Zero understanding of the format
- Comprehensive database of fragments for each XML dialect
  - Short-term → XSLT and SVG
  - Long-term → Everything based on XML (MathML, SMIL, RSS, TT, …)
    - Generating a database only requires a bunch of samples
- Coverage-guided path exploration
  - Short-term → Open Source applications under Linux
  - Long-term → Windows and Linux, with or without source-code

Younger Me Vs XSLT
On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

@agarri_fr

XML fragments

 <d e="f"/> <g h="i"> <j/> </g>

Trees

Name	Value	Depth
a	<d e="f"/><g h="i"><j/></g>	0
d	<d e="f"/>	1
g	<g h="i"><j/></g>	1
j	<j/>	2

Attributes

Name	Value	Node	Depth
b	c	a	0
e	f	d	1
h	i	g	1

Mutation strategy

- High-level mutator
  - Fully generic
  - Works on attributes and (trees of) nodes
- Medium-level mutator
  - Optional
  - Specific to a XML dialect
- Low-level mutator
  - Fully generic
  - Works on bytes or characters

High-level mutator

- Mainly a compliant XML processor
  - Supports all the XML features
    - DTD, CDATA sections, namespaces, ...
  - Provides parsing, manipulation and serialization
    - Wisely select the XML library
    - Use lxml and not ElementTree!
- Zero knowledge of the XML dialects
  - Only interact with (trees of) nodes and attributes
  - Refer to a DB of dialect-specific fragments

High-level mutator

- Three types of actions
  - *Add* + *Replace* + *Remove*
  - Each type covers trees and attributes
- *Remove* doesn't need fragments
  - Quite useful by itself (PoC minimization, trimming API)
- *Add* uses known fragments
- *Replace* uses similar known fragments
  - How to define "similarity"?
    - Attribute → attribute name, node name, type of value, …
    - Tree → top-node name, depth, ...

Medium-level mutator

- Optional dialect-specific mutations
- May increase coverage significantly
- For XSLT
  - Switch "Forwards-Compatible Processing" mode
    - Instructs parser to (not) ignore unknown nodes/attributes
  - Fix references to variables and keys
    - Helps to find UAF and double-free
- For SVG
  - None at the moment (not needed)

Low-level mutator

- Zero knowledge of XML or its dialects
  - Byte-level mutation by off-the-shelf tools
- May break valid XML documents
  - Acceptable trade-off if we fuzz fast enough
  - TODO → Apply only to attributes values and text nodes
- Implementation
  - Outside of AFL
    - Use radamsa / surku / zzuf / ...
  - When using AFL
    - Use the "trim", "splice" and "havoc" stages
    - TODO → Disable them with AFL_PYTHON_ONLY=1

Younger Me Vs XSLT
On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

@agarri_fr

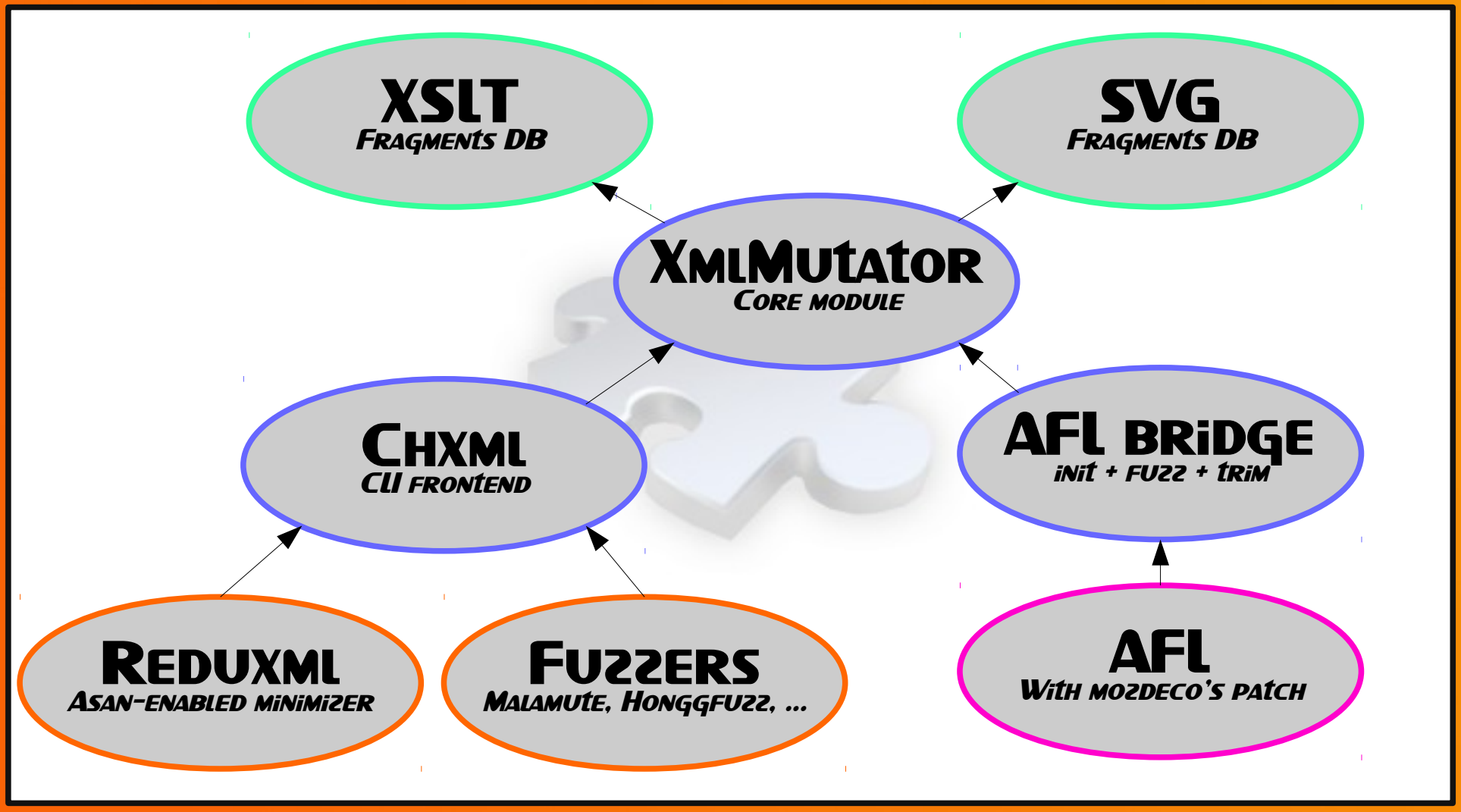
Fragments database

- Based on SQLite
  - Super fast
  - Easy to manage
  - One database file per XML dialect
- Write to the DB only when adding fragments
  - No need for optimization
- But read access is on the critical path
  - Must be as efficient as possible
  - Fast medium (SSD or RAM) + optimized queries

Optimization of queries

- Task → Select a random row from a table
- Naive approach
  - SELECT id, name, value FROM table
  - ORDER BY random() LIMIT 1
- Efficient approach
  - SELECT id, name, value FROM table
  - WHERE rowid = (abs(random()) %
  - (SELECT (SELECT max(rowid) FROM attribute)+1))
- Speed gain ~ 200x

Relationships



XmlMutator overview

- Implements the XML mutation engine
  - Useless without additional programs calling it
    - I call them "wrappers"
- Provides a simple Mutation API
  - *`Mutate`*
    - Execute some high-level mutations
      - Among *`Add`*, *`Replace`* or *`Remove`*
    - Then some medium-level mutations (if available)
  - *`Reduce`*
    - Only execute some high-level *`Remove`* mutations

XmlMutator features

- Adding fragments
  - Parse samples and add extracted fragments to the database
- Creating a mutator
  - Takes optional parameters (seed, name of dialect)
- Producing mutations
  - Get input from a string or file
  - Use the Mutation API
    - Call *`Mutate`* or *`Reduce`* several times
  - Serialize to a string or file
- Trimming input
  - TODO → Implement that (simple call to *`Reduce`*)

Wrapper: chxml

- Main front-end
  - Command-line tool
- Features
  - Reduce to a file
  - Mutate to a file or directory
  - Extract fragments and add them to the database
- Used for testing + by other tools
  - HonggFuzz and Malamute
  - ASan-enabled crash minimizer

Wrapper: AFL bridge

- Bridge between AFL and XmlMutator
- init() may take seconds
  - Generate a list of samples (used only when DOM is corrupted)
    - TODO → Remove it after implementing XML-aware trimming
  - Copy SQLite database to RAM
  - Create a long-lived mutator
- fuzz() need to be fast (thousands of calls / second)
  - Convert bytes received from AFL to a string
  - Initialize mutator from this string (if possible)
  - Mutate a few times
  - Serialize to bytes and send back to AFL

Fuzzing setup

- For each fuzzed target, two sets of binaries
  - Path exploration
    - Compiled with LLVM and afl-clang-fast
    - Use AFL's deferred or persistent modes as much as possible
  - Crash collection
    - Early and verbose crash detection with Asan
- Slow or closed-source applications aren't fuzzed
  - Instead, generated corpus is reused against them
- Crash collection and bucketization
  - Using CrashManager (by Mozilla Security)

Harnesses

- xsltproc (libxslt)
  - Use AFL's deferred mode / speed x 2
  - Strategically placed call to __AFL_INIT
- xpcshell (Firefox)
  - Use AFL's persistent mode / speed x 100
  - JavaScript function aflloop() exposes __AFL_LOOP
  - Thanks @mozdeco for the patch!
- Inkscape
  - Designed to loop through input files
  - Easy to switch to AFL's persistent mode / speed x 10

Numbers

- XSLT
  - Four targets (libxslt, sablotron, transformiix, xalan-c)
  - Two Xeon E5-2630v3 CPU (32 threads)
  - One billion execs per day
  - 360 execs per second per thread
- SVG
  - One target (Inkscape)
  - Half a Core i7-6700 CPU (4 threads)
  - Nine million execs per day
  - 25 execs per second per thread

Younger Me Vs XSLT
On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

XSLT: libxslt / Chrome

- Also used by PHP, Postgres, …
- Already fuzzed a lot (by Google and others)

- A single low-impact bug was found
  - But it shows that my strategy works
- Need a call to a undocumented namespace
  - Which is included in my corpus from 2011

XSLT: Sablotron / Adobe Reader

- I was both lucky and unlucky at the same time
- Two other teams (at least) working on the same target
  - ZDI: *"Transforming Open Source to Open Access in Closed Applications"*
    - Published at Recon Brussels and OPCDE Dubai
  - Nanyang University (SG): *"Skyfire: Data-Driven Seed Generation for Fuzzing"*
    - Published at IEEE SP2017
- Two bugs found by me in 2012 were reintroduced
  - Heap overflow during UTF16 conversion
    - CVE-2017-2948 = CVE-2012-1525
  - Type confusion via node()[lang]
    - CVE-2017-2962 = CVE-2012-1530
- And I found three new ones
  - CVE-2017-3031: two distinct but similar bugs
  - CVE-2017-2949: identical to CVE-2016-6963 (detected as a DUP by ZDI)

XSLT: TransformiiX / Firefox

- Four use-after-free reported and fixed
  - CVE-2017-5376 (MFSA 2017-01)
  - CVE-2017-5438, 39 and 40 (MFSA 2017-10)
- Three of them deal with variables
  - Found thanks to medium-level mutators

XSLT: Xalan-c / Shibboleth

- Apache project
  - Used by Shibboleth (Open Source SSO)
    - Only if XSLT isn't disabled (cf research from 2011)
  - Is it *really* not used anywhere else?
- Low priority
  - Mostly used for generating new paths
- Some security bugs are public since 2015
  - https://issues.apache.org/jira/browse/XALANC-762
  - Is reporting additional bugs useful?

SVG: Inkscape

- Before integration with CrashManager
  - Quick grep on /var/log/syslog
- Write to printable address (string "a-mode")
  - inkscape[15046]: segfault at 65646f6d2d61 ip 00007f5641xxxxxx sp 00007ffc9209e128 error 6 in libstdc++.so.6.0.21[7f5641132000+172000]
- Invalid instruction pointer
  - inkscape[19963] general protection ip:1affa3b sp:7ffc97f5e2e0 error:0 in inkscape[400000+xx]

SVG: Inkscape

- After integration with CrashManager
  - Much easier to analyze
  - TODO → Disable built-in crash handler

Short Description	Bucket Size
[ink] heap-buffer-overflow [illegible] with READ of size 4	6
[ink] heap-use-after-free [illegible] with READ of size 4	3
[ink] heap-buffer-overflow [illegible] with READ of size 8	16
[ink] UNKNOWN SIGNAL on unknown address 0x00000132ee93 [illegible]	1
[ink] heap-use-after-free [illegible] with READ of size 4	4
[ink] UNKNOWN SIGNAL on unknown address 0x000000000018 [illegible]	3
[ink] UNKNOWN SIGNAL on unknown address 0x7f73652a9000 [illegible]	1

Younger Me Vs XSLT
On the shoulders of giants

Project's goals
Design choices
Implementation

Findings
Future work

Moar!

- More time
  - Enhancing the tool is time-consuming
  - Triaging and reporting too!
- More targets
  - For path exploration + bug finding (aka corpus reuse)
- More dialects
  - Generate a bunch of databases
  - Write medium-level mutators (if needed)
- More guided fuzzers
  - LibFuzzer, covFuzz, HonggFuzz, Talos IntelPT, ...

Conclusion

- Project is very young
- A few goals already reached
  - Guided fuzzing of Open Source Linux applications
  - High-level XML mutator
  - Medium-level XSLT mutator
  - XML-aware minimizer
  - Large databases of XSLT and SVG fragments
  - More than 10 vulnerabilities found and reported
- Expect more bugs!!

Q and A

Or a few minutes of uncomfortable silence

You decide...

@AGARRi_FR

HackerPraktikum - RUB

Nov 2017

Nearly Generic Fuzzing of XML-based formats

Thanks for coming!

nicolas.gregoire@agarri.fr

@agarri_fr