Nicolas Grégoire

Nullcon 2017

Nearly generic fuzzing of XML-based formats

nicolas.gregoire@agarri.fr

@agarri_fr

Me?

- Nicolas Grégoire
- Working in InfoSec for the last 15 years
- Owner and Pwner at AGARRI
  - Web hacking
    - Published about XXE and SSRF in bug bounties
  - Teaching
    - Trainings (Burp Suite Pro) and talks
  - Fuzzing
    - Mostly client-side nowadays

Me Vs XSLT
Inspirational work
Project goals
Design
Implementation
Findings
Future work

@AGARRI_FR

Me Vs XSLT

Inspirational work

Project goals

Design

Implementation

Findings

Future work

@AGARRi_FR

Abuse of features

- Talk "Offensive XSLT" (2011)
  - No memory corruption, simply abuse the features
  - Read and create files, execute arbitrary code
  - Highly reliable exploits
- Positive side effect
  - Produced a large corpus covering most features
  - Combine 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
  - Mutation done by Radamsa
  - Basic wrappers
    - Linux: ASan + bash + grep
    - Windows: Python + WinAppDbg
  - Limited depth, found some bugs anyway
- Take-away
  - Producing XML for fuzzing purposes is hard!

Me Vs XSLT

Inspirational work

Project goals

Design

Implementation

Findings

Future work

@AGARRi_FR

Reusing code fragments

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

Production of testcases

- QuickFuzz Project (Gustavo Grieco and al.)
  - Chain different production steps
  - Mix generation and mutation
- High-level production
  - Grammar-based generation
  - Haskell's QuickCheck and Hackage
- Low-level production
  - Dumb mutation
  - Off-the-shelf tools like zzuf or radamsa

Guided fuzzing

- American fuzzy lop
  - By Michael Zalewski @lcamtuf, since 2013
  - Easy to use but hard to master
- Disadvantages
  - Doesn’t run on Windows
  - Mutation engine aimed at binary formats
- Advantages
  - Impressive track record
  - Large and active community
    - Forks (WinAFL), patches (external mutators), helpers (afl-cmin.py)

Python mutators for AFL

- External Python mutation routines
  - Patch by Christian Holler @decoder (2016)
- Add a *Python* stage calling an external module
- The module implements a custom mutator
  - init() called once
    - Do costly tasks
  - fuzz() called for each mutation
    - As fast as possible...

Me Vs XSLT
Inspirational work
Project goals
Design
Implementation
Findings
Future work

GOALS

- Hierarchical mutations
  - Structure (high-level), reuse known fragments
  - Dialect (medium-level), optional
  - Characters (low-level)
- Every XML dialect is supported
  - First step: XSLT and SVG
  - Final target: everything based on XML (SMIL, RSS, TT, ...)
- Coverage-guided path exploration
  - First step: Open Source applications under Linux
  - Final target: support for cross-platform + closed-source

Me Vs XSLT
Inspirational work
Project goals
Design
Implementation
Findings
Future work

XML fragments

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

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

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

Mutation strategy

- High-level mutators
  - Perfect understanding of XML
  - Fully generic
  - Except for fragments (which are specific to a XML dialect)
- Medium-level mutators
  - Optional (and specific to a XML dialect)
- Low-level mutators
  - Work with bytes / characters
  - Fully generic
  - Done by off-the-shelf tools

High-level mutators

- First of all, a compliant XML processor
  - Full support of
    - Namespaces
    - Document types aka DTD
  - Provides parsing, manipulation and serialization
    - Wisely select the XML library (lxml vs ElementTree)
- No knowledge of XML dialects
  - Only interact with nodes and attributes
  - But use (optional) dialect-specific fragments

High-level mutators

- Three families of actions
  - *Add*, *Replace* and *Remove*
  - Each family covers trees and attributes
- *Replace* try to use similar fragments
  - How to define “similarity”?
    - Attribute: attribute name, node name, type of value, …
    - Tree: top-node name, depth, ...
- *Remove* doesn’t need fragments
  - And can be used alone as a XML minimizer

Medium-level mutators

- Optional dialect-specific mutations
- May increase coverage significantly
- For XSLT
  - Switch "Forwards-Compatible Processing" mode
    - Ignores unknown and misplaced nodes/attributes
  - Fix references to variables, parameters and keys
    - Helps to find UAF and double-free
- For SVG
  - Currently none, finds bugs nonetheless

Low-level mutators

- No knowledge of XML or its dialects
  - Byte-level mutations by off-the-shelf tools
- May break valid XML documents
  - Acceptable trade-off if we fuzz fast enough
- Outside of AFL
  - Explicit calls to Radamsa / Surku / zzuf / ...
- When using AFL
  - “trim”, “splice” and “havoc” stages

Me Vs XSLT
Inspirational work
Project goals
Design
Implementation
Findings
Future work

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

XmlMutator

- Python module exposing a few functionalities
- Adding fragments
  - Parse a sample and add its fragments to the database
- Creating a mutator
  - Takes optional parameters (seed, name of dialect)
- Producing mutations
  - Initialize mutator from a string or file
  - Reset mutator to its initial state
  - Modify state of mutator (*Mutate* or *Reduce*)
  - Serialize to a string or file

XmlMutator

- Mutation API
  - Mutate
    - Execute some high-level mutations (*Add*, *Replace* or *Remove*)
    - Then some medium-level mutations (if available)
  - Reduce
    - Only execute some high-level *Remove* mutations
- Possible work-flows
  - Initialize / Mutate / Mutate / Mutate / Serialize
    - One file (depth=3)
  - Initialize / Mutate / Serialize / Reset / Mutate / Serialize
    - Two files (both with depth=1)
- Useless without additional code calling the API

Wrapper: chxml

- Main front-end
- Features
  - Reduce to a file
  - Mutate to a file or directory
  - Extract fragments and add them to the database
- Used by other tools
  - As an external mutator for HonggFuzz / Malamute
  - As a crash minimizer

Wrapper: AFL bridge

- Bridge between AFL and XmlMutator
- init() may take seconds
  - Generate a list of backup samples
  - Copy fragments database to memory
  - 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 string
  - If unsuccessful (invalid XML), initialize mutator from samples
  - Mutate a few times
  - Serialize to bytes and send back to AFL

Fuzzing setup

- For each fuzzed target, two sets of binaries
  - Path exploration
    - Use AFL+LLVM 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
  - But generated corpus is reused against them
  - For closed-source, exploitability heuristics are useful
- Crash collection and bucketization
  - CrashManager by Mozilla Security

Harnesses

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

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

Me Vs XSLT
Inspirational work
Project goals
Design
Implementation
Findings
Future work

Findings

- Section removed in this version of the slides
  - You should have come to Goa ;-)
- Next public edition
  - Allstars 2017, during OWASP AppSec EU

Me Vs XSLT
Inspirational work
Project goals
Design
Implementation
Findings
Future work

MOAR!

- More time
  - Triaging and reporting is time-consuming
- More targets
  - Path exploration + reuse of generated corpus
- More dialects
  - Convert corpus to fragments
  - 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
  - Very complete XLST fragments database
  - More than 10 vulnerabilities found
- Expect more bugs!!

Nicolas Grégoire

Nullcon 2017

Nearly generic fuzzing of XML-based formats

Thanks for reading!

nicolas.gregoire@agarri.fr

@agarri_fr