CSC 574 Computer and Network Security

Fuzzing

Alexandros Kapravelos kapravelos@ncsu.edu

Let's find some bugs (again)

- We have a potentially vulnerable program
- The program has some inputs which can be controlled by the attacker

Can we generate automatic tests?

Fuzzing

- A form of vulnerability analysis
- Steps
 - Generate random inputs and feed them to the program
 - Monitor the application for any kinds of errors

- Simple technique
- Inefficient
 - Input usually has a specific format, randomly generated inputs will be rejected
 - Probability of causing a crash is very low

Example

Standard HTML document

• <html></html>

Randomized HTML

- <html>AAAAAA</html>
- <html><></html>
- <html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></htm
- <html>html</html>
- <html>/</<>></html>

Types of Fuzzers

- Mutation Based
 - mutate existing data samples to create test data
- Generation Based
 - define new tests based on models of the input
- Evolutionary
 - Generate inputs based on response from program

Mutation Based Fuzzing

- Little or no knowledge of the structure of the inputs is assumed
- Anomalies are added to existing valid inputs
- Anomalies may be completely random or follow some heuristics
- Requires little to no setup time
- Dependent on the inputs being modified
- May fail for protocols with checksums, those which depend on challenge response, etc.
- Example Tools:
 - Taof, GPF, ProxyFuzz,
 - Peach Fuzzer, etc.

Fuzzing a pdf viewer

- Google for .pdf files (about 1,640,000,000 results)
- Crawl pages and build a pdf dataset
- Create a fuzzing tool that:
 - Picks a PDF file
 - Mutates the file
 - Renders the PDF in the viewer
 - Check if it crashes

Mutation Based Fuzzing

- East to setup and automate
- Little to no protocol knowledge required
- Limited to the initial dataset
- May fail on protocols with checksums, or other challenges

Generation-Based Fuzzing

- Generate random inputs with the input specification in mind (RFC, documentation, etc.)
- Add anomalies to each possible spot
- Knowledge of the protocol prunes inputs that would have been rejected by the application

Word (.doc) Binary File Format

File	Edit	View	٦	ools	H	lelp																		
rser:	Cases.	dll : Wo	ordBi	naryFo	ormat	Dete	tion	Logic	CVE	-2006	-453	4, CV	/E-20	07-05	6 <mark>15</mark> , C	•	F	arse						
aw File	e Conten	its																		Parsing Results				
0000	00940	0.0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			Name		Offset	Size	
	00950	- 50.5	00	5.5	00	00	00	00	00	00	00	00	00	00	00	00	00					0x00004f00	0x00000080	-
0.0.0.0	00960		00				00		00	00	00	00			00	00	00			EleName		0x00004f00	0x00000040	
	00970		00				00		00	00	00	00	00	00	00	00	00					the second second second		
	00980	333	00	3131	00	00	00		00	00	00	00	00	00	00	00	00	• • • • • • • • • • • • • • • • • • • •		CbEleName		0x00004f40	0x0000002	
	009900	202	00	5.5	0.0	00	00	- 5,5,	00	00	2,2,	00	5/5/	- 5/5/	00	00	00			- Туре		0x00004f42	0x0000001	
	009B0		00							00		- 5 5		- 5.5		00	00			TbyFlags		0x00004f43	0x00000001	
0000	00900	0.0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			sidLeft		0x00004f44	0x00000004	
	009D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			sidRight		0x00004f48	0x00000004	
	009E0		00		00	00	00		00	00	00	00	- 5.5	- 5.5	00	00	00					121222220112	10100000000	
	009F0		00		- 5.5		00		00	00		00		00	00	00	00			sidChild		0x00004f4c	0x0000004	
	00A00 00A10	5.7	65		- 7.5		20	2.2	77	6F	0.77	6C		2.2	0D	00	00	Hello, world!		😥 dsidThis		0x00004f50	0x00000010	
	00A10	- 50-5	00	0.000	- 5/5		00	- 5/5/	00	00	- 5/5/	00	- 5/5	- 5/5/	00	00	00	• • • • • • • • • • • • • • • • • • • •		UserFlags		0x00004f60	0x00000004	
	00A20		00				0.0		00	00	00	00		0.0	00	00	00			CreateTime		0x00004f64	0x00000008	
	00A40		00		00	00	00		00	00	00	00	00	00	00	00	00							
000	00A50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			- ModifyTime		0x00004f6c	0x00000008	
000	00A60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			StartSect		0x00004f74	0x00000004	
	00A70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			··· SizeLow		0x00004f78	0x00000004	
	08A00	333	00		- 5.5		00	2.2	00	00	- 5151	00	- 515	- 5.5	00	2020	00			SizeHigh		0x00004f7c	0x00000004	
	00A90		00	5.5	00	- 5.5	00	- 7/7/	00	00	00	00	- 5.5	- 5.5	00	00	00			- Data	100	0x00000200	0x00001000	
	00AA0 00AB0		00		00	00	00		00	00	00	00	00	00	00	00	00	• • • • • • • • • • • • • • • • • •			and the second	and the second second second second		
	00AD0	-	00		00		00		00	00		00			00	00	00			OneTableDocumentStr.		0x00004e80	0x00000080	
	OOADO	333	00	5.5		- 5.5	0.0	- 5151	00	00	2.2	00		- 5151	00	00	00			+ Clx		0x00002681	0x00000015	
	OOAEO	- 50.5	00	5.5	- 5.5	- 5/5/	- 5/5	5.5	- 5/5/	00	- 5/5/	00	- 5/5/	- 5/5/	- 5/5/	- 5/5/	00							>
0000	OOAFO	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				_			-
	00B00		00			~~~	00		00	00		00	00	00	00	00	00		-	Parsing Notes				

Generation-Based Fuzzing

- Completeness
- Can deal with complex input, like checksums
- Input generator is labor intensive for complex protocols
- There has to be a specification

Evolutionary Fuzzing

- Attempts to generate inputs based on the response of the program
- Autodafe
 - Fuzzing by weighting attacks with markers
 - Open source
- EFS
 - Generates test cases based on code coverage metrics

Challenges

- Mutation based
 - Enormous amount of generated inputs
 - Can run forever
- Generation based
 - Less inputs (we have more knowledge)
 - Is it enough?

Code Coverage

- A metric of how well your code was tested
- Percent of code that was executed during analysis
- Profiling tools
 - Gcov
- Code coverage types:
 - Line coverage
 - which lines of source code have been executed
 - Branch coverage
 - which branches have been taken
 - Path coverage
 - which paths were taken

Fuzzing Chrome

- AddressSanitizer
- ClusterFuzz
- SyzyASAN
- ThreadSanitizer
- libFuzzer
- more...

Chrome's fuzzing infrastructure

- Automatically grab the most current Chrome LKGR (Last Known Good Revision)
- Hammer away at it to the tune of multi-million test cases a day
- Thousands of Chrome instances
- Hundreds of virtual machines

AddressSanitizer

- Compiler which performs instrumentation
- Run-time library that replaces malloc(), free()and friends
- custom malloc() allocates more bytes than requested and "poisons" the redzones around the region returned to the caller
- Heap buffer overrun/underrun (out-of-bounds access)
- Use after free
- Stack buffer overrun/underrun
- Chromium's "browser_tests" are about 20% slower

SyzyASAN

- AddressSanitizer works only on Linux and Mac
- Different instrumenter that injects instrumentation into binaries produced by the Microsoft Visual Studio toolchain
- Run-time library that replaces malloc, free, et al.

ThreadSanitizer

- Runtime data race detector based on binary translation
- Supports also compile-time instrumentation
 - Greater speed and accuracy
- Data races in C++ and Go code
- Synchronization issues
 - deadlocks
 - unjoined threads
 - destroying locked mutexes
 - use of async-signal
 - unsafe code in signal handlers
 - others...

libFuzzer

- Engine for in-process, coverage-guided, white-box fuzzing
- In-process
 - don't launch a new process for every test case
 - mutate inputs directly in memory
- Coverage-guided
 - measure code coverage for every input
 - accumulate test cases that increase overall coverage
- White-box
 - compile-time instrumentation of the source code
- Fuzz individual components of Chrome
 - don't need to generate an HTML page or network payload and launch the whole browser

libFuzzer

==9896==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x62e000022836 at

pc 0x000000499c51 bp 0x7fffa0dc1450 sp 0x7fffa0dc0c00

WRITE of size 41994 at 0x62e000022836 thread T0

SCARINESS: 45 (multi-byte-write-heap-buffer-overflow)

#0 0x499c50 in __asan_memcpy

#1 0x4e6b50 in Read third_party/woff2/src/buffer.h:86:7

#2 0x4e6b50 in ReconstructGlyf third_party/woff2/src/woff2_dec.cc:500

#3 0x4e6b50 in ReconstructFont third_party/woff2/src/woff2_dec.cc:917

#4 0x4e6b50 in woff2::ConvertWOFF2ToTTF(unsigned char const*, unsigned long,

woff2::WOFF2Out*) third_party/woff2/src/woff2_dec.cc:1282

#5 0x4dbfd6 in LLVMFuzzerTestOneInput

testing/libfuzzer/fuzzers/convert_woff2ttf_fuzzer.cc:15:3

Cluster Fuzzing

ClusterFuzz uses the following memory debugging tools with libFuzzer-based fuzzers:

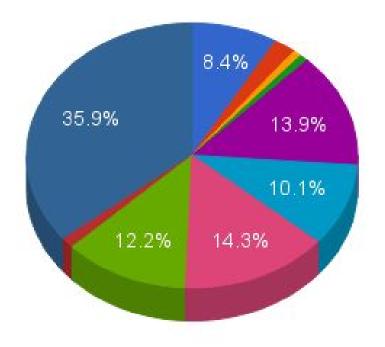
- AddressSanitizer (ASan): 500 GCE VMs
- MemorySanitizer (MSan): 100 GCE VMs
- UndefinedBehaviorSanitizer (UBSan): 100 GCE VMs

NC STATE UNIVERSITY

July 2016

14,366,371,459,772 unique test inputs **112** bugs filed

Analysis of the bugs found so far



Heap-buffer-overflow (ASan)
Stack-buffer-overflow (ASan)
Global-buffer-overflow (ASan)
Heap-use-after-free (ASan)
Use-of-uninitialized-value (MSan)
Direct-leak (LSan)
Undefined-shift (UBSan)
Integer-overflow (UBSan)
Floating-point-exception (UBSan)
Other crashes

Chrome's Vulnerability Reward Program

- Submit your fuzzer
- Google will run it with ClusterFuzz
- Automatically nominate bugs they find for reward payments