Microsoft Forefront Edge

Microsoft Israel R&D Center

Fuzzing in Microsoft and FuzzGuru framework

OWASP-IL
May-2007

John Neystadt
jney@microsoft.com
Lead Program Manager
Forefront Edge, Microsoft

Agenda

- Overview
  - Introduction to Fuzzing
  - FuzzGuru Architecture
  - Demo
- Testing
  - Designing Fuzzing Test
  - Preparation for Fuzz Testing
  - Performing Fuzz Testing
  - Completing Fuzzing Coverage
  - Fuzzing in ISA

Introduction to Fuzzing

- Tired of Code Reviews?
  - Fuzz testing is about automatic testing of malformed input handling
  - Fuzzing Target is code, not data. Its white box testing - no need to exhaust all data combinations, that code doesn't distinguish
  - In the past dumb fuzzing was made to randomly corrupt bits of data., This penetrates only top data handling layers, since something basic (checksum, first fields) would be broken and inner code paths wouldn't be reached
  - Smart is about systematic testing of all code paths that process data. Corrupt fields one by one, preserving general validity of the packet (Content-Length, Checksums, Base64 encoding, etc)

Case for Fuzzing

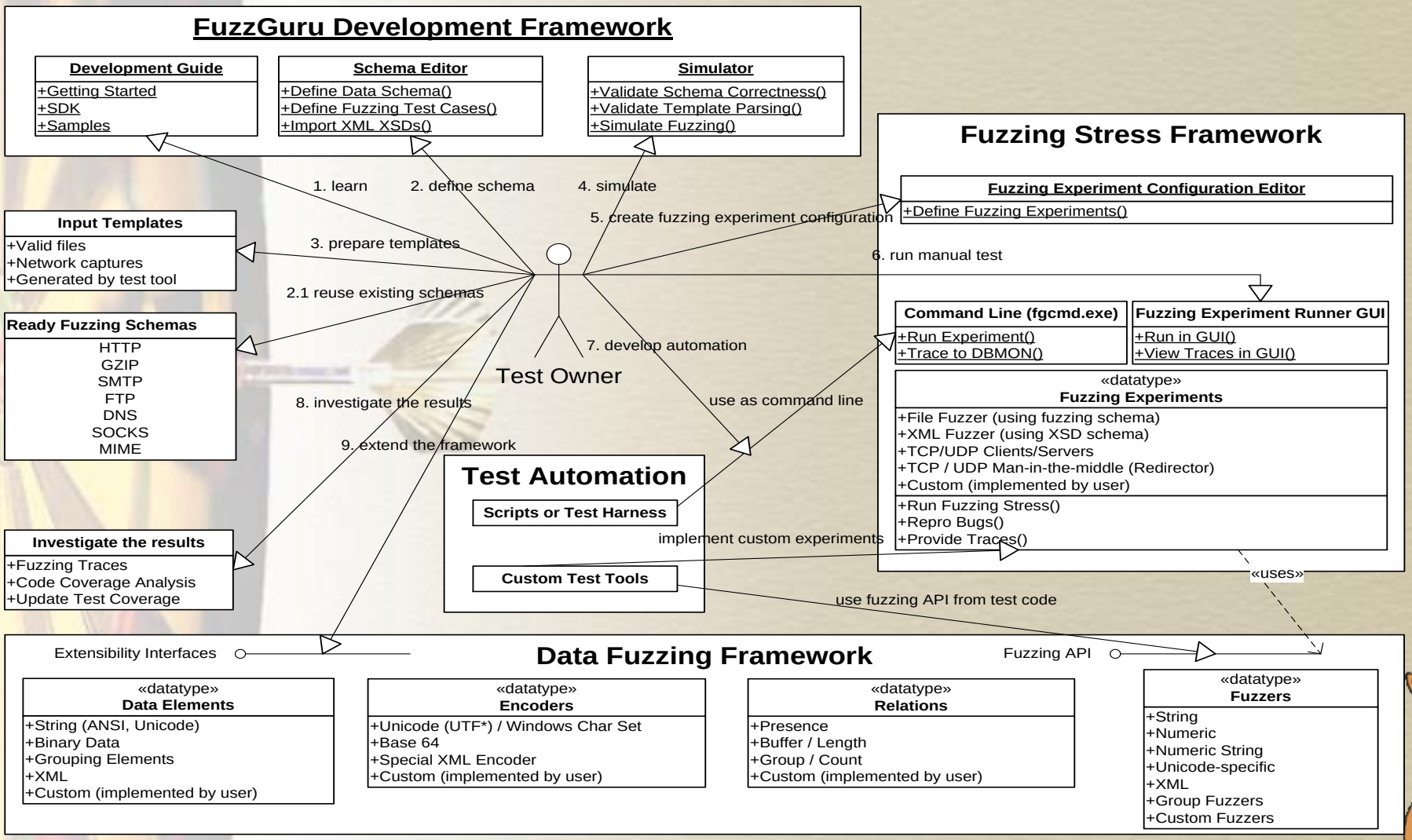
- Over 70% of security vulnerabilities Microsoft patched in 2006 were found by fuzzing
- Many open-source tools and some commercial products that do fuzzing of specific protocols or formats are available

- SDL - Microsoft Security Development Lifecycle
  - Internal “security” bible mandatory for all Microsoft teams
  - Mandates fuzzing for Files, RPC, DCOM, Network, ActiveXs as part of product development lifecycle

What I wanted to know about Fuzzing, but didn't dare to ask

- Fuzzing finds Access Violations and Buffer Overruns, not functional problems
- Fuzzing requests usually fail, since they are malformed
- Often fuzzing bugs are in clean-up code paths: leaks, synchronization and timing issues.
- Fuzz both client and server code – simulating malicious server has high chance of finding bugs!
- If a bug happens when two independently fields are malformed, FuzzGuru will not find it
- Both C++ and C# (or Java) code should be fuzzed if it processes data from untrusted source

FuzzGuru Architecture



Designing Fuzzing Test

- Fuzzing a protocol - create schema or customize a ready one
  - Define schema for parts of the protocol, need to be fuzzed
  - More later
- Defining fuzzing Test Cases
  - Fuzzing Test Cases choose a subset of schema (elements and fuzzers) to be applied for relevant scenario
  - FuzzGuru command line tool gets test case id as parameter
  - FuzzGuru API requires test case id to generate malformed packets
  - Test Case can be used for focused fuzzing to test specific fixes
- Create fuzzing Templates
  - Obtain real world data captures or generated by test programmatically
  - Templates are “equivalence classes”, choose minimum number to cover all your scenarios
  - Template must adhere to schema, you can use placeholder elements in schema to skip irrelevant parts
- Use FuzzGuru GUI Simulator to verify test cases and templates
  - Verify template is parsed correctly
  - See which fuzzers are chosen (simulator trace)
  - Inspect output to inspect malformed requests

Manual or Automated Fuzz Testing

- FuzzGuru Experiments – ready tools for common fuzzing scenarios
  - Gets schema, mutation template and test case name
  - Ready tcp/udp server/client, man-in-the middle redirector, file, xml modes
  - Extensive - can develop custom experiments
  - Generates “stress” of upto 500 tcp cps or 11,000 udp packets
  - Supports recording/playback of repro buffer
  - Generates perf counters
  - Writes fuzzing traces to dbmon
- Manual Test Running (GUI)
  - Define Experiment Configuration (FuzzGuru GUI, Tools | Experiment Configuration Editor)
  - Run experiments interactively (FuzzGuru GUI, Tools | Experiment Runner)
- Scriptable Automation
  - Define Experiment Configuration (FuzzGuru GUI, Tools | Experiment Configuration Editor)
  - Call experiments from command line interface (fgcmd.exe)
- Integrate into your own test tools
  - Call DataFuzzer APIs from your own test tool, providing it schema, mutation template and test case name

Demo

- Run BinUrt2\DemoApp.exe
- Run HTTP Server Attack Demo
  - See Demo\Readme.mhtml => Part2 for instructions
- Results (5 minutes each run):

Scenario	**Iterations**	**Bugs found**	**Test Cases coverage**
Dumb Fuzzing without template	6500	0 out of 4	24%
Dumb Fuzzing with template	11000	2 from 4 (hits: 0, 0, 2, 10)	56%
Fuzzing with HTTP Demo Schema without template	700 (slow ☹)	2 from 4 (hits: 0, 0, 4, 8)	80%
Fuzzing with HTTP Demo Schema with one template (**valid**)	10200	4 from 4 (hits: 1, 7, 64, 143)	100%

Completing Fuzzing Coverage (Code Coverage)

- After running a fuzzing test for a first round it is important to inspect whether all code paths were covered
- Magellan Code Coverage tool set is recommended
- You should not hunt the numbers, rather review all uncovered code paths
  - Fuzzing should traverse to all branches that depend on untrusted data
  - All parsing code should be covered
  - All error handling and cleanup code should be covered
- Note: If there is no code to handle malformed data exist, Code Coverage will not help you!
- To add coverage, you may need:
  - Extend test matrix and fuzzing tests additional configurations of the feature
  - Add additional fuzzing templates
  - Extend FuzzGuru with custom fuzzers or elements
- Metrics
  - Expected to reach 75-85% code coverage of parsers code blocks
  - Per fuzzed DLL: fuzzing usually adds 3%-6% blocks and 3%-8% arcs additional coverage to compared to stress
  - Per fuzzed source file: fuzzing usually significantly adds the coverage of ~13%.
  - Per fuzzed function: fuzzing usually significantly adds the coverage of over 17%.