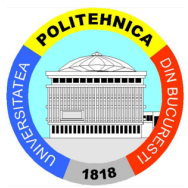

Session 7
Stateless Fuzzing

Security Summer School
July 14th 2014

ACS/Ixia/Hexcellents

What is fuzzing anyway?

- Sofware testing technique (often automated)
- Involves providing invalid, unexpected or random data as input to a program
- Used for discovering new software vulnerabilities

Fuzzing Phases

- Identify target

Fuzzing Phases

- Identify target
- Identify inputs

Fuzzing Phases

- Identify target
- Identify inputs
- Generate fuzzed data

Fuzzing Phases

- Identify target
- Identify inputs
- Generate fuzzed data
- Execute fuzzed data

Fuzzing Phases

- Identify target
- Identify inputs
- Generate fuzzed data
- Execute fuzzed data
- Monitor for exceptions

Fuzzing Phases

- Identify target
- Identify inputs
- Generate fuzzed data
- Execute fuzzed data
- Monitor for exceptions
- Determine exploitability

Target inputs

- Command line parameters
- Files (configuration/application specific files) that the application parses
- Network sockets
- Stdin
- Environment variables
- In-Memory Fuzzing

Types of Fuzzers

- Protocol Fuzzers (eg. Sulley, Peach, SPIKE, etc.)
- File Format Fuzzers (eg. SPIKEFile, notSPIKEFile, FileFuzz)
- CLI arguments fuzzers (eg. iFuzz)
- Specialized fuzzers (eg. ActiveX fuzzers, RPC fuzzers, web fuzzers, etc.)

Generating the data

Depending on how you derive the fuzzing data, you can have:

- Mutation-based fuzzers where you take valid data and then modify it
- Generation-based fuzzers where you contruct the data from the ground up based on its specific structure

When generating fuzzing data, we're trying to pick interesting values for the fields being fuzzed

- Integers
  - Negative numbers (0xFFFFFFFF, 0x80000000, etc.)
  - Large numbers (0x7FFFFFFF, 0x20000000, etc.)
  - Small values such as 0-10
  - Header values identifying the length of header/data segments
- ASCII
  - Large strings / empty strings
  - Strings with inaccurate length tags
  - Strings with accurate but long length tags
  - Strings with format specifiers

Stateless vs. Stateful Fuzzing

- Depends on whether we need to get the application into a certain state before we can commence fuzzing
- Think of HTTP vs. SMTP/FTP
- Protocol state machine
- Stateful fuzzing is a bit more complicated to do

Fuzzing Frameworks

- Generate the fuzzed data
- Orchestrate running the program against the generated test data and see whether it crashed
- Automate collecting debugging and run information for the test cases that crashed or caused the application to go into an abnormal state
- Track the progress and time needed to run the fuzz tests
- Generate reports and automatically assess the exploitability of the discovered vulnerability

Sulley Fuzzing Framework

Sulley

- Fuzzer development and fuzz testing framework consisting of multiple extensible components
- Python based so it can basically run on any platform
- Overall usage of Sulley breaks down to the following
  - Data Representation: First step in using any fuzzer. Run your target and tickle some interfaces while snagging the packets. Break down the protocol into indvidual requests and represent that as blocks in Sulley.
  - Session: Link your developed requests together to form a session, attach the various available Sulley monitoring agents (network, debugger, etc.) and commence fuzzing.
  - Post Mortem: Review the generated data and monitored results. Replay individual test cases.

Fuzzing with Sulley

When fuzzing with Sulley, we need to write a python script that defines all required objects that Sulley needs in order to fuzz specific target:

- Data Model: data model defines the properties of the network protocol that we're going to fuzz.
- State Model: state model is used to define possible interactions between different states of the fuzzed network protocol.
- Target: target defines what we're going to fuzz.
- Agents: agents are special programs running on the target computer, which do various things, amongst others are: monitoring the fuzzed process for crashes, intercepting the relevant network packets, restarting the crashed process, etc.
- Monitoring Interface: monitoring interface allows us to easily see the result of fuzzing process, how many of the test cases were already sent, how many of them caused a crash, etc.

File Format Fuzzing

- SPIKEfile
  - Linux generation-based file fuzzer based on SPIKE
  - Fuzzers may be defined for it with the same SPIKE script block based notation as we've seen with Sulley.
- notSPIKEfile
  - Linux mutation-based file fuzzer
  - Starting from a valid input file it will attempt and modify the file byte by byte (or some other pattern) in order to produce test cases
- Both of them support application orchestration and monitoring

Resources

1. Fuzzing.org: `www.fuzzing.org/`
2. Sulley: `github.com/OpenRCE/sulley`
3. Sulley User Manual: `www.fuzzing.org/wp-content/SulleyManual.pdf`