Offensive Software Exploitation

Summer 2020

Ali Hadi

@binaryz0ne

Fuzzing & Exploitability Determination

what garbage data can your application handle?

Part #4

Fuzzing

- Original research name “Boundary Value Analysis”

- “An automated method for discovering faults in software by providing unexpected input and monitoring for exceptions.” – Fuzzing

- Also said:

"Fuzzing is the process of sending intentionally invalid data to a product in the hopes of triggering an error condition or fault. These error conditions can lead to exploitable vulnerabilities.“ – *HD Moore (MSF Founder)*

Plz note

- Fuzzing has no rules!
- Not always successful!

Fuzzing History

- Fuzzing is not new
  - It's been named for about 20 years.
- Professor Barton Miller
  - Father of Fuzzing
  - Developed fuzz testing with his students at the University of Wisconsin-Madison in 1988/89
  - GOAL: improve UNIX applications
- Since 1999 with PROTOS till date, Fuzzing has managed to discover a wide range of security vulnerabilities… (Check Fuzzing 101 for further history information)

Fuzzing Methods

- Sending Random Data
  - Least Effective
  - Unfortunately, sometimes, code is bad enough for this to work
- Manual Protocol Mutation
  - You are the fuzzer
  - Time consuming, but can be accurate when you have a hunch
  - Web App Pen-Testing

Fuzzing Methods – Cont.

- Mutation or Brute Force Testing
  - Starts with a valid sample
  - Fuzz each and every byte in the sample
- Automatic Protocol Generation Testing
  - Person needs to understand the protocol
  - Code is written to describe the protocol (a “grammar”)
  - Fuzzer then knows which piece to fuzz, and which to leave alone (SPIKE)

What Data can be Fuzzed?

- Virtually anything!
- Basic types: bit, byte, word, dword, qword
- Common language specific types: strings, structs, arrays
- High level data representations: text, xml

Where can Data be Fuzzed?

Across any security boundary, e.g.:

- An RPC interface on a remote/local machine
- HTTP responses & HTML content served to a browser
- Any file format, e.g. Office document
- Data in a shared section
- Parameters to a system call between user and kernel mode
- HTTP requests sent to a web server
- File system metadata
- ActiveX methods
- Arguments to SUID binaries

Two Approaches

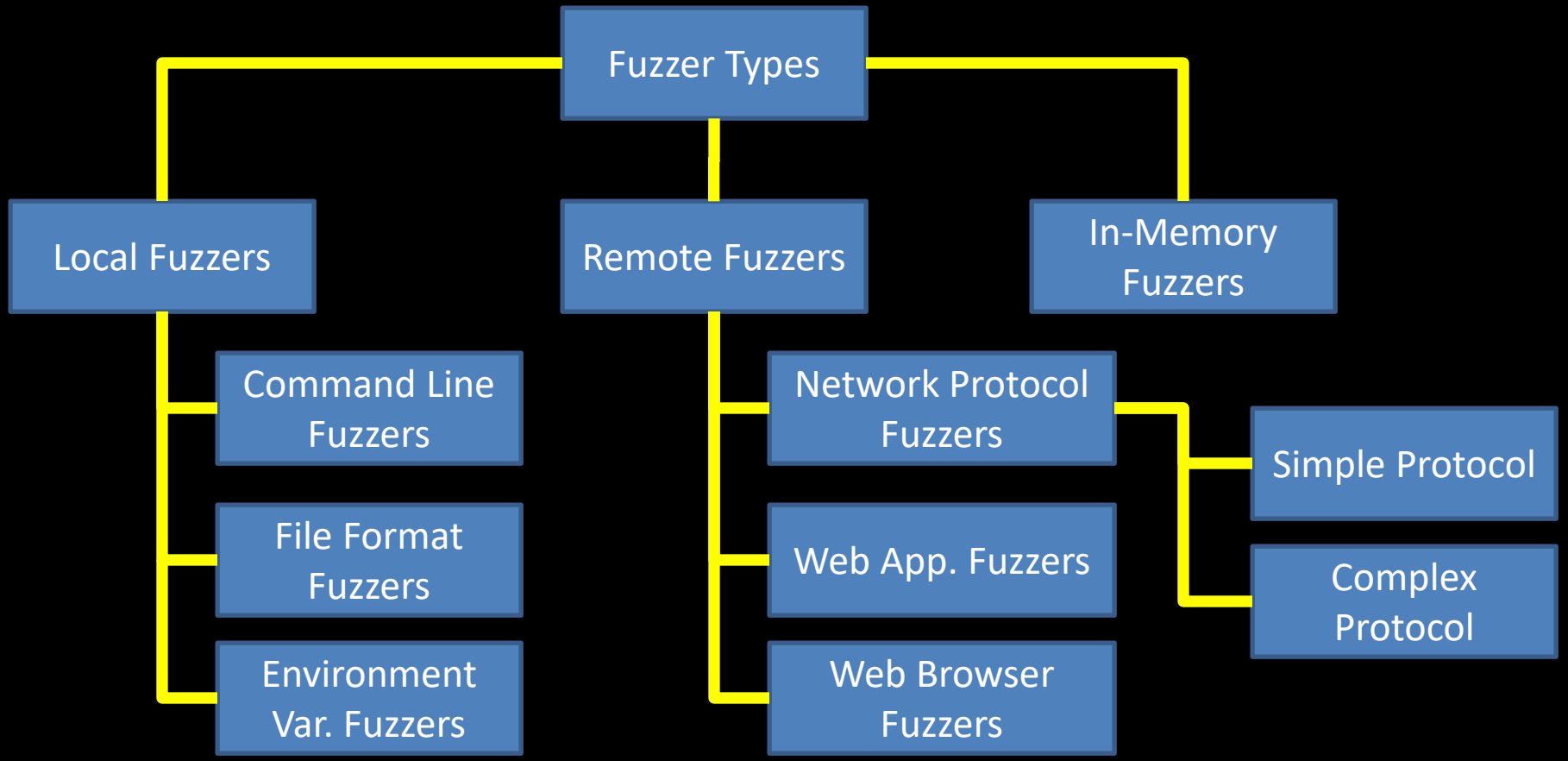
- Dumb (mutational) Fuzzing
- Fuzzer lacks contextual information about data it is manipulating
- May produce totally invalid test
- Up and running fast
- Find simple issues in poor quality code

- Smart (generational) Fuzzing
- Fuzzer is context-aware
  - Can handle relations between entities, e.g. block header lengths, CRCs
- Produces partially well-formed cases test cases
- Time consuming to create
  - What if protocol is proprietary?
- Can find complex issues

Two Approaches – Cont.

- Which approach is better?
- Depends on:
  - Time: how long to develop and run fuzzer
  - [Security] Code quality of target
  - Amount of validation performed by target
    - Can patch out CRC check to allow dumb fuzzing
  - Complexity of relations between entities in data format
- Don't rule out either!
  - My personal approach: get a dumb fuzzer working first
  - Run it while you work on a smart fuzzer

Fuzzer Classifications



The Fuzzing Process – Cont.

- Determine Exploitability – Remotely
  - You need to know what data you sent
    - Record all fuzzed strings, making note of exceptions
    - Network Captures (Wireshark)
  - Try and reproduce the scenario
  - Is it a memory corruption bug?
  - Is it an application logic flaw?

- Determine Exploitability – Locally
  - Attach a debugger

Determining Exploitability

- This process requires experience of debugging security issues, but some steps can be taken to gain a good idea of how exploitable an issue is...
- Look for any cases where data is written to a controllable address – this is key to controlling code execution and the majority of such conditions will be exploitable
- Verify whether any registers have been overwritten, if they do not contain part data sent from the fuzzer, step back in the disassembly to try and find where the data came from

Determining Exploitability – Cont.

- If the register data is controllable, point the register which caused the crash to a page of memory which is empty, fill that page with data (e.g., ‘aaaaa...’)

- Repeat and step through each operation, until another crash occurs, reviewing all branch conditions which are controlled by data at the location of the (modified) register to ensure that they are executed

Determining Exploitability – Cont.

- Are saved return address/stack variables overwritten?
- Are the processor registers derived from data sent by the fuzzer (e.g. 0x61616161)?
- Is the crash triggered by a read operation?
- Is the crash triggered by a write operation?
- Is the crash in a heap management function?
- Do we have full or partial control of the faulting address?
- Do we have full or partial control of the written value?

Types of Fuzzers

- Local Fuzzers
  - Lets you fuzz applications on the command line
    - To what end?
  - Make sure the target has some value (setuid)

- Environment Variable Fuzzers
  - Because:

```
#include <string.h>
int main (int argc, char **argv)
{
        char buffer[10];
        strcpy(buffer, getenv("HOME"));
}
```

The Fuzzing Process

- Identify Targets
- Identify Inputs
- Generate Fuzzed Data
- Execute Fuzzed Data
- Monitor for Exceptions
- Determine Exploitability

Final Tip to Beginners

- OS: Windows XP first!
  - Easy to debugging
  - Almost every RE tool works on XP
  - For example use Kali or BackTrack for developing tools
- Find bugs and debug/exploit them upon XP
- And port it for other versions of OS (Windows 7,8, etc)
- Virtualization Software (VMWARE, Virtualbox, etc) is mandatory
- Use snapshots
- Your OS will be messed up by your Fuzzer

References

- A Bug Hunter's Diary, Tobias Klein, No Starch Press
- Sam Bowne, Malware Analysis Course Slides, http://samsclass.info/126/126_F13.shtml
- Fuzz Testing, http://en.wikipedia.org/wiki/Fuzz_testing
- Fuzzing: Brute Force Vulnerability Discovery, Michael Sutton, et al, Addison-Wesely
- University of Wisconsin Fuzz Testing (the original fuzz project)
- Fuzzing 101, NYU/Poly.edu, Mike Zusman, http://pentest.cryptocity.net/fuzzing/
- Fuzzing for Security Flaws, John Heasman, Stanford University
- EVERYONE HAS HIS OR HER OWN FUZZER, BEIST (BEISTLAB/GRAYHASH), www.codeengn.com
- An Introduction to SPIKE, the Fuzzer Creation Kit, Dave Aitel, http://www.docstoc.com/docs/2687423/An-Introduction-to-SPIKE-the-Fuzzer-Creation-Kit---PowerPoint
- Common Vulnerablities and Exposures, http://cve.mitre.org/
- Common Weakness Enumeration, http://cwe.mitre.org/
- Seven kingdoms of weaknesses Taxonomy, http://cwe.mitre.org/documents/sources/SevenPerniciousKingdomsTaxonomyGraphic.pdf
- Common Configuration Enumeration, http://cce.mitre.org/
- National Vulnerability Database, http://nvd.nist.gov/home.cfm
- Exploit Database, http://exploit-db.com
- http://www.security-database.com/toolswatch/+-Fuzzers-+.html
- http://caca.zoy.org/wiki/zzuf
- https://code.google.com/p/ouspg/wiki/Radamsa