Offensive Software Exploitation

SEC-300-01/CSI-301-02

Ali Hadi

@binaryz0ne

Fuzzing

what gibberish data can your application handle?

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

What Does Fuzzed Data Consist Of?

- Fuzzing at the type level:
  - Long strings, strings containing special characters, format strings
  - Boundary case byte, word, dword, qword values
  - Random fuzzing of data buffers
- Fuzzing at the sequence level
  - Fuzzing types within sequences
  - Nesting sequences a large number of times
  - Adding and removing sequences
  - Random combinations
- Always record the random seed!!

When to Fuzz?

Fuzzing typically finds implementation flaws, e.g.:

- Memory corruption in native code
  - Stack and heap buffer overflows
  - Un-validated pointer arithmetic (attacker controlled offset)
  - Integer overflows
  - Resource exhaustion (disk, CPU, memory)
- Unhandled exceptions in managed code
  - Format exceptions (e.g. parsing unexpected types)
  - Memory exceptions
  - Null reference exceptions

When to Fuzz? – Cont.

- Injection in web applications
  - SQL injection against backend database
  - LDAP injection
  - HTML injection (Cross-site scripting)
  - Code injection

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

?

- How can we monitor the target?
- What to monitor?

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

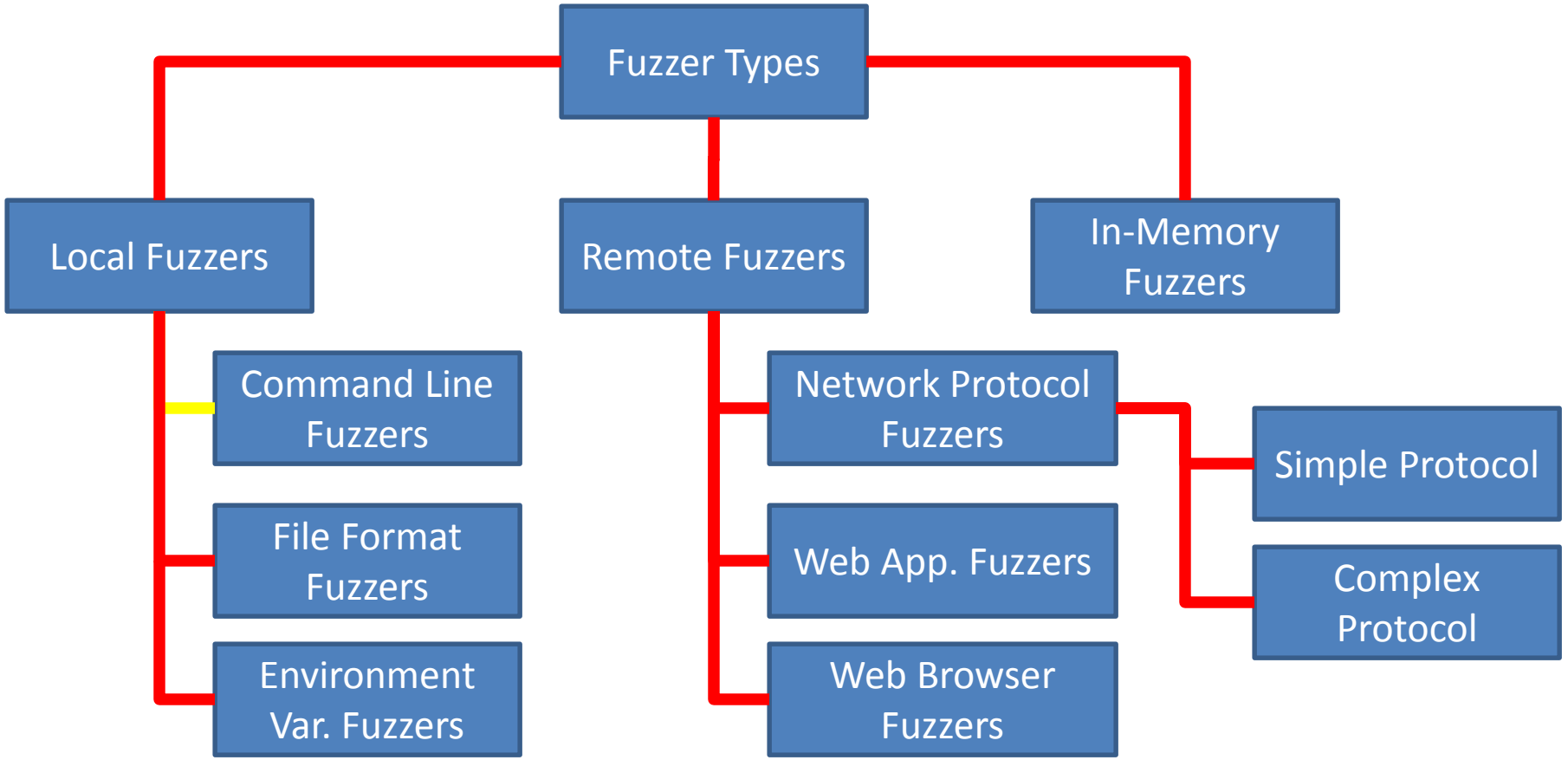
Determining Exploitability – Cont.

- 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 Notes

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

Fuzzer Classifications



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"));
}
```

Types of Fuzzers – Cont.

- File Format Fuzzers
  - Fuzz valid files
  - Pass them to an executable
- Remote Fuzzers (might make you famous ☺)
  - Listen on a network connects
  - When client connects, fuzz them!

Types of Fuzzers – Cont.

- Network Protocol Fuzzers
  - The Fuzzer is the client
  - Need to understand the protocol
  - Simple Protocols
    - Text Based: Telnet, FTP, POP, HTTP
  - Complex Protocols
    - Binary Data (some ASCII)
    - Complex authentication, encryption, etc

Types of Fuzzers – Cont.

- Other types of fuzzers:
  - Web Application and Server Fuzzing
  - Web Browser Fuzzing
  - In-Memory Fuzzing

Common Fuzzers

- Publicly available fuzzing frameworks:
  - Spike, Peach Fuzz, Sulley, Schemer, etc
- Publicly available fuzzing applications
  - Fuzz, FileFuzz, iFuzz, WebFuzz, JBroFuzz, WebScarab,
  - BurpSuite (includes a fuzzer), notSPIKEFile, SPIKEProxy, ProtoFuzz
  - SMUDGE, mangleme, FileP, FileH, MalyBuzz,
  - Dfuz, AxMan, bugger, fuzzdb
  - And the list goes on and on …

The Fuzzing Process

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

The Fuzzing Process

- 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

Protocol Fuzzing

- Find as much data as you can about the target application
  - Google is your friend
  - Maybe someone has fuzzed it
  - Maybe it uses some standard protocol
- What is the transport layer?
  - TCP or UDP?
    - Effects anomaly detection
- What type of protocol (simple or complex)?

Protocol Fuzzing – Cont.

- Do we need to authenticate?
  - What authentication protocol?
- Scoping your assessment
  - You may only care about pre-auth
- Reversing the Protocol
  - Generate Traffic and Sniff
  - Use wireshark (check for plug-ins!)
  - Google
- Once you understand how to communicate with a service, you can send packets to it

Why ???

- Writing a network protocol fuzzer, means eventually you'll be re-inventing the wheel!!!

- Why do that when you can use:

SPIKE

SPIKE

- SPIKE fuzzer released in 2002
  - Written by Dave Aitel (Immunity Inc.)
- SPIKE is a genius
- SPIKE is a fuzzing framework/API
- Ability to describe data
- Built in libraries for known protocols (*RPC)
- Fuzz strings designed to make software fail

SPIKE – Cont.

- Simple Text Based Protocol Fuzzing
- Accepts a “script” of SPIKE commands
- Example: ./generic_send_tcp <IP> <PORT> script.spk 00

```
s_readline()
s_string_variable("USER");
s_string(" ");
s_string_variable("devel_user");
s_string(" ");
s_string_variable("PASS");
s_string(" ");
s_string_variable("secretpassword");
s_string("\r\n");
```

SPIKE's Real Value

- Complex Protocols have length fields and data fields
- Tracking length fields while Fuzzing data is complicated
- SPIKE does this for you
- Block Based Protocol Representation

What is a SPIKE?

- “A SPIKE is a simple list of structures which contain block size information and a queue of bytes.”

```
s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);
```

What is a SPIKE? – Cont.

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- Push 4 NULLs onto BYTE queue (size place holder)
- Then a new BLOCK listener is allocated named "somepacketdata"

What is a SPIKE? – Cont.

```
s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);
```

- Script starts searching the block listeners for one named “somepacketdata”
- Block “start” pointers are updated to reflect the blocks position in the queue

What is a SPIKE? – Cont.

```
s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);
```

- 4 bytes of data are pushed onto the queue

What is a SPIKE? – Cont.

```
s_block_size_binary_bigendian_word("somepacketdata");
s_block_start("somepacketdata")
s_binary("01020304");
s_block_end("somepacketdata");
```

- The block is ended, and the sizes are finalized
- The original 4 null bytes are updated with the appropriate size value

What is a SPIKE? – Cont.

s_block_size_binary_bigendian_word(“somepacketdata”);
s_block_start(“somepacketdata”)
s_binary(“01020304”);
s_block_end(“somepacketdata”);

Block 2
Morepacketdata
Big Endian word
Start Pointer: 1008

Block 1
Somepacketdata
Big Endian word
Start Pointer: 1000

Existing Challenges

- How to measure effectiveness of a fuzzer?
  - Number of test cases?
  - Number of bugs?
  - Severity of bugs?
  - % Code coverage?
- How many test cases to run?
  - How to balance complexity vs. time constraints?

SUMMARY

- Explained what do we mean by Fuzzing, and Fuzzing History
- Also talked about Fuzzing Methods, Types and the Fuzzing Process
- Talked about howto fuzz a protocol, and finally talked about SPIKE

References

- A Bug Hunter's Diary, Tobias Klein, No Starch Press
- 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