Fuzzing 101

NYU/Poly.edu
October 23, 2008
Mike Zusman
Hi, I’m Mike Zusman

Past:
- Web Application Developer
- Escalation Engineer @ Whale Communications, Inc (a Microsoft subsidiary)
- Application Security Team @ ADP, Inc
- Spoken at Industry Events: OWASP, BlackHat (and a cameo at DEFCON)

Current:
- Senior Consultant @ Intrepidus Group, Inc.
Great Expectations

☐ You already know how to fuzz 😊

☐ This class will teach you:
  ■ History of fuzzing
  ■ Fuzzing Methodologies
  ■ About Fuzzing tools you can use

☐ At the end of this class you:
  ■ Will have written your own fuzzer
  ■ Found some cool (hopefully exploitable) bugs
The Approach

- Fun entertaining lectures by me
- Lectures contain content from the “book” and my own experience.
- Homework

Michael Sutton
Adam Greene
Pedram Amini
Forward by HD Moore
The Spirit

- HAVE FUN!
  - When the weather's too bad, or my wife won't let me go real "fishing", I go fuzzing instead!
What exactly is fuzzing?

"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 (from Fuzzing)
What exactly is fuzzing?

“Throw sh!t at the wall and see what sticks!” – b3nn
What exactly is fuzzing?

"There are no rules to fuzzing."
- *Fuzzing, the book*

"There are no guarantees in fuzzing."
- Me
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
History

- Millers fuzzer was pretty basic
- It sent random strings of data to the application
- If (CRASH||HANG) {Finding(fuzzStr);}
- Smarter fuzzers would follow...
History

- 1999 brought PROTOS from University of Oulu
- PROTOS began by analyzing PROTOcol specifications
- Packets were modeled that violated the specs
- Testing suites were designed that could be used against multiple vendor products
History

- 2002
  - Microsoft injects cash into PROTOS

- PROTOS members branch out and start Codenomicon
  - First Commercial Fuzzer

Today, Fuzzing is part of Microsoft’s SDL Process!
History

- SPIKE fuzzer also released in 2002
  - Dave Aitel wrote it

- Where Millers fuzzer was dumb, SPIKE is a genius
  - Ability to describe data
  - Built in libraries for known protocols (*RPC)
  - Fuzz strings designed to make software fail
History

- 2004 Browser Fuzzing
- MangleMe – Michal Zalewski
  - Fuzzed HTML to find browser bugs

<table>
<thead>
<tr>
<th>Attachments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input element crash</strong> (25 bytes, text/html) 2004-10-18 13:23 PDT, Daniel Veditz</td>
</tr>
<tr>
<td><strong>attack of the marquees</strong> (361 bytes, text/html) 2004-10-18 13:23 PDT, Daniel Veditz</td>
</tr>
<tr>
<td><strong>col span demo (non-crashing)</strong> (58 bytes, text/html) 2004-10-18 13:24 PDT, Daniel Veditz</td>
</tr>
<tr>
<td><strong>crasher not viewing as a file:///</strong> (62.88 KB, text/html) 2004-10-23 21:41 PDT, Keith Gable</td>
</tr>
</tbody>
</table>

*Add an attachment* (proposed patch, testcase, etc.)

*Description* From Daniel Veditz 2004-10-18 13:22:03 PDT


extract:

A gallery of quick examples I examined to locate the offending tag (total time to find and extract them - circa 1 hour):
History

- 2004 File Format Fuzzing
- Microsoft Security Bulletin MS04-028
  - Buffer Overrun in JPEG Processing (GDI+) Could Allow Remote Code Execution

Nobody, and I mean NOBODY, gets through my firewall.
History

- 2005 File Format Fuzzers Released
- FileFuzz, SPIKEfile, notSPIKEfile
  - Michael Sutton and crew
- Then came the rain.
  - "When Office 2003 shipped, we thought we'd done some good work and that it would be a secure product," said David LeBlanc, a senior software development engineer with the Office team. "For the first two years after release, it held up really well, only two bulletins. [But] then people shifted their tactics and started finding problems in fairly large numbers." -
  http://www.infoworld.com/article/07/09/21/Microsoft-developer-Fuzzing-key-to-Office-security_1.html?DESKTOP%20SECURITY
History

- 2005 More Browser Fuzzers
- Hamachi
  - HD Moore, Aviv Raff
  - Fuzzed Dynamic HTML
- CSSDIE
  - HD Moore and crew
  - Fuzzed CSS Style Sheets
History

- **2006 Month of Browser Bugs**
  - HD Moore and crew released a browser bug every day, with “no direct path to code execution.”
    - http://www.foxnews.com/story/0,2933,202547,00.html
  - Controversial?
    - Blogger thinks so

This blog is under review due to possible Blogger Terms of Service violations and is open to authors only

http://browserfun.blogspot.com/
History

- 2006 ActiveX Fuzzing
  - When his car insurance went up, the GEICO caveman started selling ActiveX 0days (just kidding)
  - Too easy?
History

- 2006 ActiveX Fuzzing
- COMRaider
  - GUI Based
  - Point and Click
  - iDefense tool
- AxMan
  - More complicated to use than COMRaider
  - IMO, a better fuzzer
- Why so easy?
  - ActiveX/COM objects have exportable typelibs that describe all methods, interfaces, properties.
2007 More Browser Fuzzing

Advisory: a specially crafted JavaScript can make Opera execute arbitrary code

A specially crafted JavaScript can make Opera execute arbitrary code.

Severity:
Highly severe

Problem description
A virtual function call on an invalid pointer that may reference data crafted by the attacker can be used to execute arbitrary code.

Opera’s response
Opera Software has released Opera 9.23, where this issue has been fixed.

Credits
Thanks to Mozilla.org for providing their JavaScript fuzzer.
History

- 2008
  - First time fuzzing is taught in a University setting? Maybe...
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

- 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)
Types of Fuzzers

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

- Environment Variable fuzzers
  - Because:
    ```c
    #include <string.h>
    int main (int argc, char **argv)
    {
        char buffer[10];
        strcpy(buffer, getenv("HOME"));
    }
    ```
Types of Fuzzers

- File Format Fuzzers
  - Fuzz valid files
  - Pass them to an executable

- Remote Fuzzers (my favorite)
  - Listen on a network connects
  - When client connects, fuzz them!
Types of Fuzzers

- 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
      - MSRPC (Supported by SPIKE)
Types of Fuzzers

</fuzzers>

<fuzzing>
The Process of Fuzzing

1. Identify Targets
2. Identify Inputs
3. Generate Fuzzed Data
4. Execute Fuzzed Data
5. Monitor for Exceptions
6. Determine Exploitability
1. Identify Targets

- Internet
- Windows
- DMZ Subnet
  - Reverse Proxy
  - Microsoft IIS
- Secure Network
  - DC/DNS/AD
  - Client (Windows)
- Oracle
- Mainframe
- Commercial FTP Server
- Apache Tomcat
The Process of Fuzzing

1. Target: Commercial FTP Server (WARFTP)
2. Identify Inputs
3. Generate Fuzzed Data
4. Execute Fuzzed Data
5. Monitor for Exceptions
6. Determine Exploitability
2. Identify Inputs

1. Target: Commercial FTP Server (WARFTP)
2. Identify Inputs

- TCP Port 21, p1, p2 (PASV or active?)
- Commands: USER, PASS, CWD, DELE, etc
- Special CHARs: \r\n, <space>
The Process of Fuzzing

1. Target: Commercial FTP Server (WARFTP)
2. Inputs: TCP21, Ftp Commands, Special Chars, binary files

3. Generate Fuzzed Data
   
   USER <username>
   Generation one: 1024 x A
   Generation two: 2048 X A
   Generation three: 4096 X String(random)
   PASS <password>
   Generation one: 1024 x A
   Generation two: 2048 X A
The Process of Fuzzing

1. Target: Commercial FTP Server (WARFTP)
2. Inputs: TCP21, Ftp Commands, Special Chars, binary files
3. Fuzzed Data: <username>, <password>
4. Execute Fuzzed Data

```java
for (int w=0; w<maxIterations; w++){
  openhost();
  for (int commandIndex = 0; commandIndex < commandCount; commandIndex++)
  {
    userInput = commands[commandIndex].command;
    if (commands[commandIndex].argument.equals("%f")){
      //FUZZ IT
      if (AttackID != 4) {
        userInput = userInput + stringAttacks[w].replace("\r", "").replace("\n", "");
      } else {
        userInput = userInput + stringAttacks[w];
      }
    }
  }
}
```
for (int w=0; w<maxIterations; w++){
  openhost();
  for (int commandIndex = 0; commandIndex < commandCount; commandIndex++)
  {
    userInput = commands[commandIndex].command;
    if (commands[commandIndex].argument.equals("%f")){
      //FUZZ IT
      if (AttackID != 4) {
        userInput = userInput + stringAttacks[w].replace("\r", "").replace("\n", "");
      } else {
        userInput = userInput + stringAttacks[w];
      }
    } else {
      userInput = userInput + " " + commands[commandIndex].argument;
    }
    userInput = userInput + "\r\n";
    try {
      toServer.write(userInput.getBytes(),0,userInput.getBytes().length);
    } catch (Exception e) {
      System.out.println("Connection dropped on write");
    }
  }
}
The Process of Fuzzing

1. Target: Commercial FTP Server (WARFTP)
2. Inputs: TCP21, Ftp Commands, Special Chars, binary files
3. Fuzzed Data: <username>, <password>
4. Send the Fuzzed Data to the Target (code)
5. Monitor for Exceptions

```java
try {
    response = in.readLine();
    if (!response.equals("")) {
        // do nothing
    }
} catch (IOException e) {
    System.out.println("ANOMALY: Connection was dropped.");
    if (AttackID != 4) System.out.println("ANOMALY: String length was "+
        stringAttacks[w].replace("\r", "").replace("\n", "").length());
} catch (Exception e) ...
```
The Process of Fuzzing

1. Target: Commercial FTP Server (WARFTP)
2. Inputs: TCP21, Ftp Commands, Special Chars, binary files
3. Fuzzed Data: <username>, <password>
4. Send the Fuzzed Data to the Target (code)
5. Monitor for Exceptions

```java
try {
    response = in.readLine();
    if (!response.equals("")) {
        // do nothing
    }
} catch (IOException e) {
    System.out.println("ANOMALY: Connection was dropped.");
    if (AttackID != 4) System.out.println("ANOMALY: String length was " +
             stringAttacks[w].replace("\r", ").replace("\n", ").length());
} catch (Exception e) ..."
```
The Process of Fuzzing

1. Target: Commercial FTP Server (WARFTP)
2. Inputs: TCP21, Ftp Commands, Special Chars, binary files
3. Fuzzed Data: <username>, <password>
4. Send the Fuzzed Data to the Target (code)
5. Monitor the socket for any exceptions
6. **Determine Exploitability**
   Depends...
The Process of Fuzzing

- 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
Fuzzing Logistics

- “A good fuzzer needs to allow a user to quickly narrow down the iteration that caused the crash.”
  - stryde_hax

- Log all fuzz attempts

- The last one before an anomaly (exception) is the best place to start
Fuzzing Logistics

- Reproducibility Challenges
  - What if you are two days in on a fuzzing exercise, and you find a flaw.
  - How can you quickly reproduce the scenario that caused the crash?

<fuzzing_AX>
Fuzzing ActiveX Objects

- Target: Windows Workstations
- Inputs:
  - Internet -> Internet Explorer -> ActiveX -> Interface -> Vulnerable Method/Property
- All fuzzing and fault detection is handled by COMRaider

- COMRaider Demo
Fuzzing ActiveX Objects

- Target: Windows Workstations
- Inputs:
  Internet -> Internet Explorer -> ActiveX -> Interface -> Vulnerable Method/Property

- Fuzzing is handled by AxMan, but not detection. We need an external debugger.

- AxMan Demo
Fuzzing ActiveX Objects

- Target: Windows Workstations
- Inputs:
  Internet -> Internet Explorer -> ActiveX -> Interface -> Vulnerable Method/Property

- Fuzzing is handled by AxMan, but not detection. We need an external debugger.

- AxMan Demo