Software Security IV: Fuzzing

Chengyu Song

Slides modified from Dawn Song
Administrivia

- Homework 1
  - Due: Friday Oct 27 11:59pm
  - Questions regarding reading materials
- Talk
  - Security R&D in a Security Company: Career and Challenges
  - Claud Xiao, Palo Alto Networks
  - Monday, October 23rd, 12:00-1:00pm in Bourns A265
Memory error

• Spatial error and temporal error
• Root cause: many types of bugs
• How to exploit
  • Overwrite/overlap critical data
  • Overread/overlap critical data
• Defenses
Switch gear: finding vulnerabilities

• From attackers perspective
  • Vulnerabilities are the pass into the system
  • 0-day vulnerabilities are especially valuable
    • $5k ~ $1.5M, even higher in blackmarket
  • Bug bounties

• From defender/vendors perspective
  • Finding and fixing bugs before release is way cheaper than patches
But how?

- Two general approach: static analysis and dynamic analysis
  - Static analysis does not execute the program
    - Example: compiler warnings and errors
    - Sacrifice accuracy for code coverage -> no false negative
  - Dynamic analysis performs the analysis while running the program
    - Example: unit tests
    - Sacrifice code coverage for accuracy -> no false positive
Dynamic testing

- Unit/regression tests
  - Goal: target code behave as **expected**
  - How: (mostly) manually generated test cases
- Exploits
  - Inputs that can trigger **unexpected** behaviors
- Fuzzing
  - Finding such inputs
## Fuzzing vs. regression

<table>
<thead>
<tr>
<th></th>
<th>Regression</th>
<th>Fuzzing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition</strong></td>
<td>Run program on many normal inputs, look for badness.</td>
<td>Run program on many abnormal inputs, look for badness.</td>
</tr>
<tr>
<td><strong>Goals</strong></td>
<td>Prevent normal users from encountering errors (e.g. assertion failures are bad).</td>
<td>Prevent attackers from encountering exploitable errors (e.g. assertion failures are often ok).</td>
</tr>
</tbody>
</table>
Fuzzing

- Inputs generator: **automatically** generates test inputs
- Executor: executes the target program with the inputs
- Monitor: detects **abnormal** behaviors
How to generate inputs?

- Idea 1: randomly
Random inputs

- Advantages: easy to implement, do not look like normal inputs
- Disadvantages: inefficient
  - Programs usually have input validation logic, random inputs are unlikely to pass
  - No indication of progress
Mutation

- Idea 2: take a well-formed input, randomly perturb (e.g., flipping bits)
  - Little or no knowledge of the structure of the inputs is assumed
  - Anomalies are added to existing valid inputs
  - Anomalies may be completely random or follow some heuristics (e.g. remove NUL, shift character forward)
Example: Building a PDF fuzzer

- Google for PDF files (filetype:pdf more than 1 billion results)
- Crawl them to build a corpus
- Using a fuzzing tool (or script)
  1. Grab a file
  2. Mutate the file
  3. Feed it to the PDF reader
  4. Look for crash
Limitations

- Only as good as the initial corpus
- Corpus may contain lots of redundant or dull inputs
  - Solution: corpus distillation
- Not making use of semantic information
Syntax-guided generation

- Test cases are generated from description of the format: grammar, RFC, documentation, etc.
- Anomalies are added to each possible spot in the inputs
- Knowledge of syntax should give better results than random fuzzing
Example: png specification

//png.spk
//author: Charlie Miller

// Header - fixed.
s_binary("89504E470D0A1A0A");

// IHDRChunk
s_binary_block_size_word_bigendian("IHDR"); //size of data field
s_block_start("IHDRcrc");
    s_string("IHDR"); // type
    s_block_start("IHDR");
// The following becomes s_int_variable for variable stuff
// 1=BINARYBIGENDIAN, 3=ONEBYTE
    s_push_int(0x1a, 1); // Width
    s_push_int(0x14, 1); // Height
    s_push_int(0x8, 3); // Bit Depth - should be 1,2,4,8,16, based on colortype
    s_push_int(0x3, 3); // ColorType - should be 0,2,3,4,6
    s_binary("00 00"); // Compression || Filter - shall be 00 00
    s_push_int(0x0, 3); // Interlace - should be 0,1
    s_block_end("IHDR");
    s_binary_block_crc_word_littleendian("IHDRcrc"); // crc of type and data
    s_block_end("IHDRcrc");
...
Limitations

• Writing specifications and corresponding generators are not easy, especially for complex format
  • Solution: learn spec through machine learning
• May be too well-formed
Measuring progress

• Q: with limited computation resources, how to evaluate inputs and prioritize good ones?

• Feedback fuzzing
Genetic programming

Inspired by biological evolution and its fundamental mechanisms, Genetic Programming software systems implement an algorithm that uses random mutation, crossover, a fitness function, and multiple generations of evolution to resolve a user-defined task. -- GeneticProgramming.com
Fitness function

- How to we measure the "fitness" of an input?
- Metrics
  - Program states coverage
  - Code coverage: line, branch, path
Line coverage

• Line/block coverage: measures how many lines of source code have been executed.

```java
if (a > 2) a = 2;
if (b > 2) b = 2;
```

• For the code above, how many test cases (values of `pair(a, b)`) is needed for full (100%) line coverage?
Branch coverage

- Branch coverage: measures how many branches in code have been taken (conditional jumps)

```java
if (a > 2) a = 2;
if (b > 2) b = 2;
```

- For the code above, how many test cases is needed for full branch coverage?
Path coverage

- Path coverage: measures how many execution paths have been taken

```
if (a > 2) a = 2;
if (b > 2) b = 2;
```

- For the code above, how many test cases is needed for full path coverage?
- Q: how to calculate the total number of paths?
Problems of code coverage

mySafeCpy(char *dst, char* src) {
    if(dst && src)
        strcpy(dst, src);
}

• Does full line coverage guarantee finding the bug?
• Does full branch coverage guarantee finding the bug?
Mutation strategy

- Random mutation
- Feedback-based mutation
  - Checksum
  - Magic number
  - Likelihood to trigger vulnerability
Monitors

• How to we know if there is bug?
  • Crash?
  • Manually inserted assertions?
  • Error detectors
    • AddressSanitier, ThreadSanitizer, MemorySanitizer, UndefinedBehaviorSanitizer, DataFlowSanitizer, LeakSanitizer
Best available fuzzer?

- american fuzzy lop (AFL)
  - [http://lcamtuf.coredump.cx/afl/](http://lcamtuf.coredump.cx/afl/)
For next class ...

- Software Security V: symbolic execution