

Automatic Program Analysis

CS390R - UMass Amherst

Course Information

- Project 4 due April 18
- Gradescope Presentation Checkin Sunday

Today's Content

- Fuzzer theory
- Setting up a fuzzer and using it
 - AFL++ (Project 5 README explains it as well)

Basics

- Origin
 - 1988 class project at University of Wisconsin
 - Crashed ~30% of tested unix utilities, and triaged the crashes
 - Paper published in 1990
- This fuzzer was very dumb, completely random
- Provide invalid/unexpected random data to program to find exceptions
- AFL
 - Released in 2014, first public coverage guided fuzzer
 - Very easy to use
 - Brought fuzzing field back alive bigtime

Types of fuzzers

- Blackbox Fuzzing
 - Blindly send input into a binary and hope something crashes
 - Simplest form of fuzzing, but still surprisingly effective
- Greybox Fuzzing
 - Apply various instrumentation techniques to the target to improve fuzzer
 - This makes the fuzzer much more effective in many circumstances
- Whitebox Fuzzing
 - Whitebox fuzzing relies on having access to source code of the target
 - Might use symbolic execution to basically transform target program into a mathematical equation and attempt to solve it
 - Much slower than previously mentioned approaches

Mutational vs Grammar-based Fuzzer

- Mutational
 - Very useful to fuzz applications that take input files and parse them to work off of the data
 - Works off of existing “corpus”
 - Randomly mutates the files in the corpus, and runs the target with the mutated file as input
 - Target crashes -> Save input file for manual inspection
- Grammar-based
 - This technique is necessary when attempting to fuzz applications that have very strict rules for their input
 - Compilers, Browsers
 - Does not use a corpus, instead generates inputs from scratch using grammar rules
 - Target crashes -> Save input file for manual inspection

Web Fuzzing

- Can be used to find vulnerabilities like xss & sql injections
- Often done with word lists containing various sql and xss payloads that are likely to find bugs
- Some examples:
 - Gobuster/Dirbuster - Bruteforce directories/urls (eg. finding hidden www.some_site/admin.php)
 - Burp Intruder - More control to fuzz specific fields with varying options

The screenshot shows the Burp Suite Intruder interface with the 'Payload Positions' tab selected. The 'Attack type' is set to 'Sniper'. A list of 13 request components is shown, with the second component, the Host header, highlighted. The Host header value is 'ac2d1fac1e7d09d480ce541c00db00f4.web-security-academy.net'. To the right of the list are buttons for 'Add \$', 'Clear \$', 'Auto \$', and 'Refresh'. A 'Start attack' button is located at the top right of the tab.

Dashboard Target Proxy **Intruder** Repeater Sequencer Decoder Comparer Extender Project options User options

1 x 2 x ...

Target Positions **Payloads** Options

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
1 GET /product?productId=$ HTTP/1.1
2 Host: ac2d1fac1e7d09d480ce541c00db00f4.web-security-academy.net
3 Connection: close
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
6 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
7 Sec-Fetch-Site: same-origin
8 Sec-Fetch-Mode: navigate
9 Sec-Fetch-User: ?1
10 Sec-Fetch-Dest: document
11 Referer: https://ac2d1fac1e7d09d480ce541c00db00f4.web-security-academy.net/
12 Accept-Encoding: gzip, deflate
13 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
```

Add \$
Clear \$
Auto \$
Refresh

Start attack

The screenshot shows the Burp Suite Intruder interface with the 'Payload Sets' and 'Payload Options' tabs selected. The 'Payload Sets' tab shows a single payload set named '1' with a count of 0. The 'Payload Options' tab shows a list of items for the 'Username generator' payload type, with a maximum payload count of 50. The 'Add' button is visible at the bottom.

Dashboard Target Proxy **Intruder** Repeater Sequencer Decoder Comparer Extender Project options User options

1 x 2 x ...

Target Positions **Payloads** Options

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 0
Payload type: Username generator Request count: 0

Payload Options [Username generator]

This payload type lets you configure a list of names or email addresses, and derives potential usernames from these using various common schemes. You can enter items as "firstname.lastname" or "firstname.lastname@example.org".

Maximum payloads per item: 50

Items

Paste
Load ...
Remove
Clear

Add Enter a new item

Address Sanitization

- With the fuzzing approaches we have discussed so far, we are only able to detect crashes
- Many bugs such as slight out of bounds reads/writes do not actually result in crashes, they can however still be very relevant security concerns
- With address sanitization, even 1 byte out of bound reads/writes can be recognized and reported
- This is usually done either through compile time instrumentation or emulation

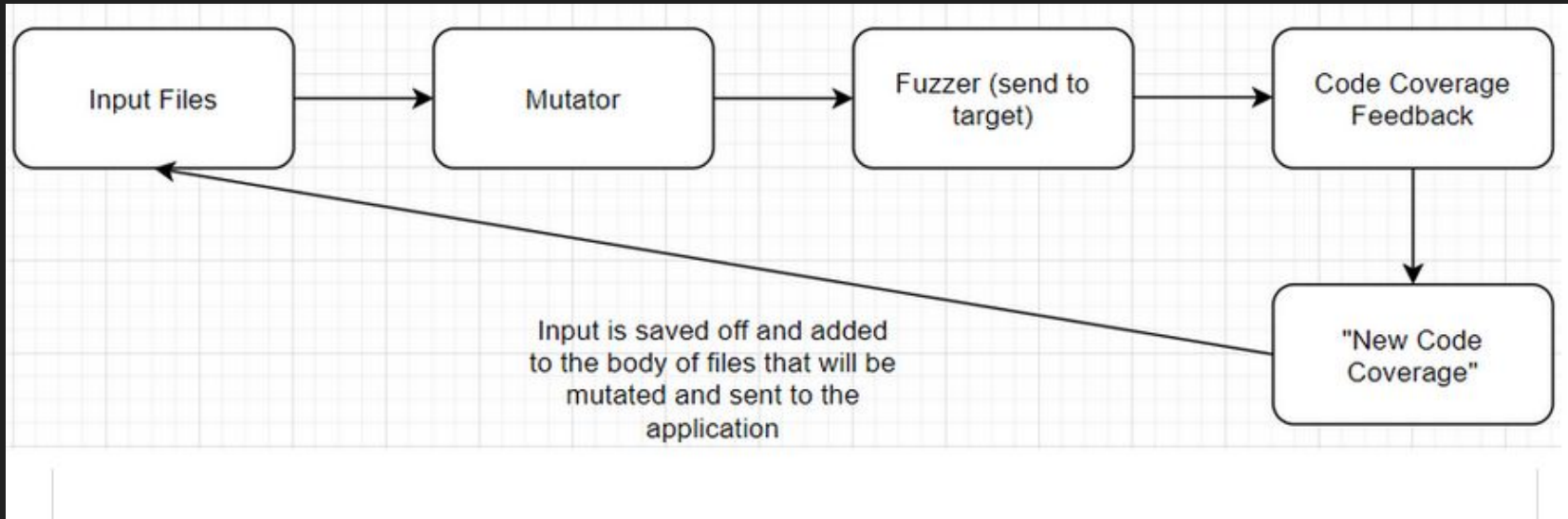

```
1  int func(char *x) {
2      if (x[0] == '\x41') {
3          if (x[1] == '\x42') {
4              if (x[2] == '\x43') {
5                  if (x[3] == '\x44') {
6                      printf("Success\n");
7                      return 1;
8                  }
9              }
10         }
11     }
12     return 0;
13 }
```

Coverage Tracking

- Measure which parts of the program are being executed
- Collection methods
 - Compile-time instrumentation
 - Intel PIN / Intel PT
 - Randomly request fuzzer location at time intervals
- Types of coverage:
 - Block
 - Edge
 - N-gram Edge
 - Path
 - Data
- Why do we care about coverage? (example)

Coverage Guided Fuzzing

- Different heuristics used by modern fuzzers
- Main idea is to add “interesting” inputs to the corpus, so future fuzz cases make use of these cases



Seed Scheduling

Often heuristic based depending on a couple different metrics

- Static
 - Graph centrality analysis - assign weight based on the number of reachable edges from a given seed
 - Determine weight based on if seed is on the path to a frequently vulnerable function (eg. memcpy)
- Dynamic (Power Schedules)
 - Assign weights based on input properties (execution time, shorter, more frequent coverage increases, etc)
 - Use mutation history to decide when to stop focusing on “hard” edges

Corpus Management

- Corpus minimization
 - Delete “slow” entries from corpus
 - Pros: “faster” corpus
 - Cons: potentially less state
- Initial seed selection importance
 - Having a good initial corpus greatly affects the performance of the fuzzer
- Finding seed files:
 - <https://datacommons.anu.edu.au/DataCommons/rest/records/anudc:6106/data/>
 - Writing a web crawler

Mutational Strategies

- Feedback loop approach - use 'good' mutations more frequently
- Havoc: Apply multiple randomly selected mutators simultaneously on inputs
- Individual Strategies:
 - Bit flips
 - Byte exchanges
 - Simple arithmetics (+-x on each byte)
 - Known integers
 - Changing size of input
 - Dictionary of interesting strings
 - Splicing - combine two different inputs at random locations

Crash Triaging

- Crash exploration
 - Separate mode of fuzzer that takes a crashing seed as its input, and attempts to find more crashes based on this input
 - Once multiple crashing inputs are gathered, statistical analysis can be performed to find common cases and to better understand the bug
- Deduping Crashes
 - Group “similar” crashes together to avoid looking at hundreds of similar crashes
- Debugging
 - Simplest: Load into gdb and rootcase
 - Improved: Timeless debugger to step backwards and more easily root-cause the bug

Harnessing

- Many programs don't just take a file as input and operate on it
- In these cases, we need to write a wrapper around the target that allows our fuzzer to interact with it
- Examples include gui tools, embedded devices, libraries,

Performance

- Persistent Mode/Snapshot fuzzing
 - Fuzz in short loop around target functions by saving memory/register state right before this function, to then base future cases off of this location
- In-memory Fuzzing
 - Many targets attempt to read data from disk, loading corpus entirely into memory instead and injecting fuzz cases can greatly improve performance by reducing disk I/O
- Scaling
 - Real world generally runs fuzzers on at least 50-100 cores
 - Can't use too much shared information between threads (corpus/coverage/statistics)
 - Avoid executing syscalls in fuzz-loop since that can quickly trigger kernel locks