Shachi Shah
Course: CSE543 Spring B
17 April 2025

Fuzz Them All Project

Dependencies

This fuzzer is implemented in Python 3 and relies only on standard library modules:

- random
- sys
- struct

To Run

- chmod +x fuzzer.py
- ./fuzzer.py <prng_seed> <num_iterations> > output.crash

No external dependencies or libraries are required to run the fuzzer. The code is compatible with Linux environments and has been tested on both the challenge VM and local Linux setup.

Input Generation Strategy

This project implements a dumb mutational fuzzer, which reads a binary seed file called _seed_, and applies random mutations to it. The fuzzer was designed to comply with the project specification:

- The seed is loaded from a file named _seed_ in the current directory.
- A pseudorandom number generator (PRNG) is seeded with the prng_seed command-line argument.
- The fuzzer performs num_iterations iterations. In each iteration:
    - Each byte of the current input is mutated to a random byte with a probability of 13%.
    - Every 500 iterations, the input is extended by 10 random bytes.
- The final mutated input is printed to stdout, with no additional output.
- The fuzzer maintains determinism, meaning the same combination of _seed_, prng_seed, and num_iterations will always produce the same output. This makes results reproducible by the TA for grading purposes.

Crashing Inputs and Parameters

Level 1:

- Seed file format: 4-byte integer length + payload
- Payload: b'A' * 180

    -
      ```
      python3 -c "
      import struct
      payload = b'A' * 180
      ```

```
seed = struct.pack('<i', 100) + payload
open('_seed_', 'wb').write(seed)
"
```

- Crash Location: Buffer overflow on input size > 76
- prng_seed: 1337
- num_iterations: 30000
- Crashing input saved as level-1.crash

Level 2:

- Seed file format: 4-byte flag + 4-byte size + payload
- Payload: b'A' * 64, flag set to 2

    -
      ```
      python3 -c "
      import struct
      payload = b'A' * 64
      flag = 2
      length = len(payload)
      seed = struct.pack('<II', flag, length) + payload
      open('_seed_', 'wb').write(seed)
      "
      ```
- Crash Location: Buffer overflow on input > 72 bytes
- prng_seed: 1337
- num_iterations: 30000
- Crashing input saved as level-2.crash

Level 3:

- Seed file format: 4-byte flag + 2-byte size + payload
- Payload: b'A' * 240 + b'\xde\xad\xbe\xef' * 8, flag set to 0xFFFFFFFF

    -
      ```
      python3 -c "
      import struct
      payload = b'A' * 240 + b'\xde\xad\xbe\xef' * 8
      flag = 0xFFFFFFFF
      length = len(payload)
      seed = struct.pack('<IH', flag, length) + payload
      open('_seed_', 'wb').write(seed)
      "
      ```
- Crash Location: Buffer overflow due to unchecked byte-wise reading
- prng_seed: 1337
- num_iterations: 50000
- Crashing input saved as: level-3.crash

Level 4:

- Seed file format: 4-byte flag + 2-byte size + payload

- Payload: b'A' * 128, flag set to 2

  -
    ```
    python3 -c "
    import struct
    payload = b'A' * 128
    flag = 2
    length = len(payload)
    seed = struct.pack('<IH', flag, length) + payload
    open('_seed_', 'wb').write(seed)
    "
    ```
- Crash Location: Buffer overflow from size-controlled read
- prng_seed: 1337
- num_iterations: 50000
- Crashing input saved as: level-4.txt

Level 5:

- Seed file format: 4-byte size + payload
- Payload: b'A' * 400

  -
    ```
    python3 -c "
    import struct
    payload = b'A' * 400
    seed = struct.pack('<i', len(payload)) + payload
    open('_seed_', 'wb').write(seed)
    "
    ```
- Crash Location: Buffer overflow due to read beyond buffer (264-byte buffer in binary)
- prng_seed: 1337
- num_iterations: 200000
- Crashing input saved as: level-5.crash

Level 6:

- Seed file format: 4-byte big-endian integer length + payload
- Payload: b'A' * 400

  -
    ```
    python3 -c "
    import struct
    payload = b'A' * 400
    seed = struct.pack('>I', len(payload)) + payload
    open('_seed_', 'wb').write(seed)
    "
    ```
- Crash location: Buffer overflow on local_118[268] due to unchecked size read with endian conversion
- Endian note: Length field must be encoded as big-endian
- prng_seed: 1337
- num_iterations: 100000

- Crashing input saved as: level-6.crash

Level 7:

- Seed file format:
  - 2-byte little-endian chunk count
  - For each chunk: 2-byte little-endian chunk length + chunk size
- Payload: 2 chunks
  - Chunk 0: b’A’*20
  - Chunk 1: b’B’*20

    -
      ```
      python3 -c "
      import struct
      chunks = [b'A' * 20, b'B' * 20]
      seed = struct.pack('<H', len(chunks))  # 2 chunks
      for chunk in chunks:
          seed += struct.pack('<H', len(chunk)) + chunk
      open('_seed_', 'wb').write(seed)
      "
      ```
- Crash Location: Buffer overflow due to mutation corrupting chunk length field (e.g., chunk 1's length mutated to a very large value, triggering over-read)
- Additional Handing: After fuzzing, if the mutated input caused the target to request more bytes than were present, the crash file was patched with zero padding to fulfill the required length.
- prng_seed: 1337
- num_iterations: 50000
- Crashing input saved as: level-7.crash

Level 8:

- Seed file format:
  - 2-byte little-endian chunk count
  - For each chunk: 2-byte little-endian chunk length + chunk size
- Payload: 2 chunks
  - Chunk 0: b’A’*20
  - Chunk 1: b’B’*20
  - The fuzzer mutates one of the 2-byte size fields to a large positive number, causing an oversized read beyond local_38[40].

    -
      ```
      python3 -c "
      import struct
      chunks = [b'A' * 20, b'B' * 20]
      seed = struct.pack('<H', len(chunks))  # 2 chunks
      for chunk in chunks:
          seed += struct.pack('<H', len(chunk)) + chunk
      open('_seed_', 'wb').write(seed)
      ```

"

- Crash Location: Buffer overflow due to unchecked read(0, local_38, chunk_size) when chunk_size becomes large but not negative (it passes the signed check!)
- Endian Note: All fields are little-endian (<H)
- prng_seed: 1337
- num_iterations: 50000
- Crashing input saved as: level-8.crash

Level 9:

- Uncrashable

Level 10:

- Seed file format:
  - 2-byte little-endian chunk count
  - For each chunk: 2-byte little-endian chunk length + chunk size
- Payload: 2 chunks
  - Chunk 0: b'A'*20
  - Chunk 1: b'B'*20

    -
      ```
      python3 -c "
      import struct
      chunks = [b'A' * 20, b'B' * 20]
      seed = struct.pack('<H', len(chunks))  # 2 chunks
      for chunk in chunks:
          seed += struct.pack('<H', len(chunk)) + chunk
      open('_seed_', 'wb').write(seed)
      "
      ```
- Crash Location: Negative chunk length → error → _exit(1)
- prng_seed: 1337
- num_iterations: 50000
- Crashing input saved as: level-10.crash

Notes

- The same fuzzer was used for all levels.
- The _seed_ file was customized per level to match the input structure observed through reverse engineering.
- The fuzzer was run locally on Linux to avoid VM resource limitations, and crash-inducing inputs were later uploaded to the challenge VM for flag verification.

- Level 7 required extra post-processing to ensure the mutated input matched the length specified in the corrupted chunk headers, demonstrating a deeper mutation side-effect vulnerability.
- Level 9 is not crashable.