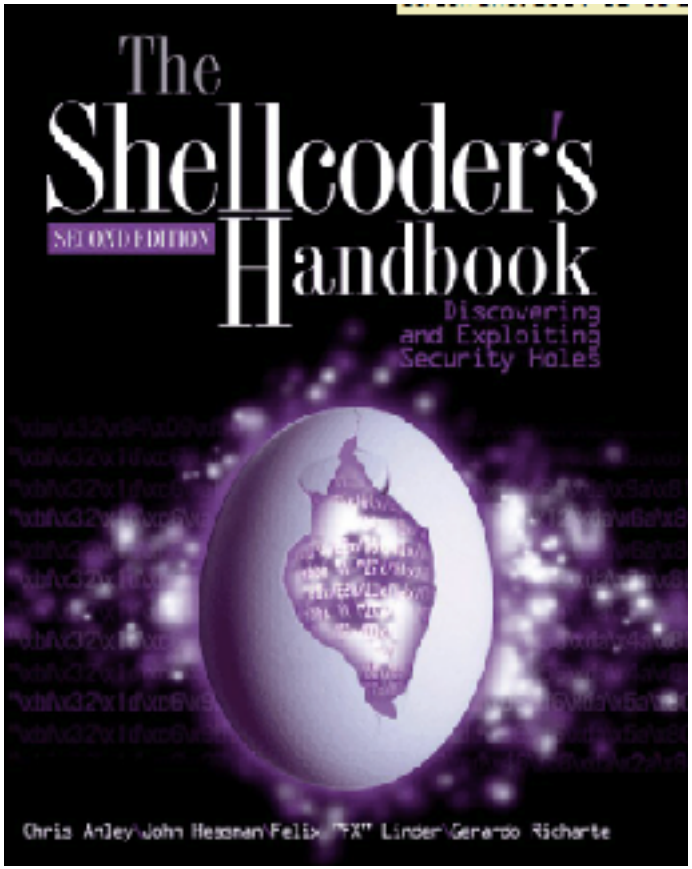
CNIT 127: Exploit Development
Vulnerability Discovery
Ch 16: Fault Injection



Updated 5-10-22

Fault Injection

- Long used to verify the fault tolerance of hardware, such as
  - Automobile and airplane components
  - Coffee makers
- Faults are injected through
  - Pins of integrated circuits
  - Bursts of EMI (Electromagnetic Interference)
  - Altered voltage levels, etc.

QA (Quality Assurance)

- Engineers test software for weaknesses with fault injection
- Automating these tests makes their work much more efficient
- They also use manual auditing techniques
  - Reverse engineering
  - Source code auditing

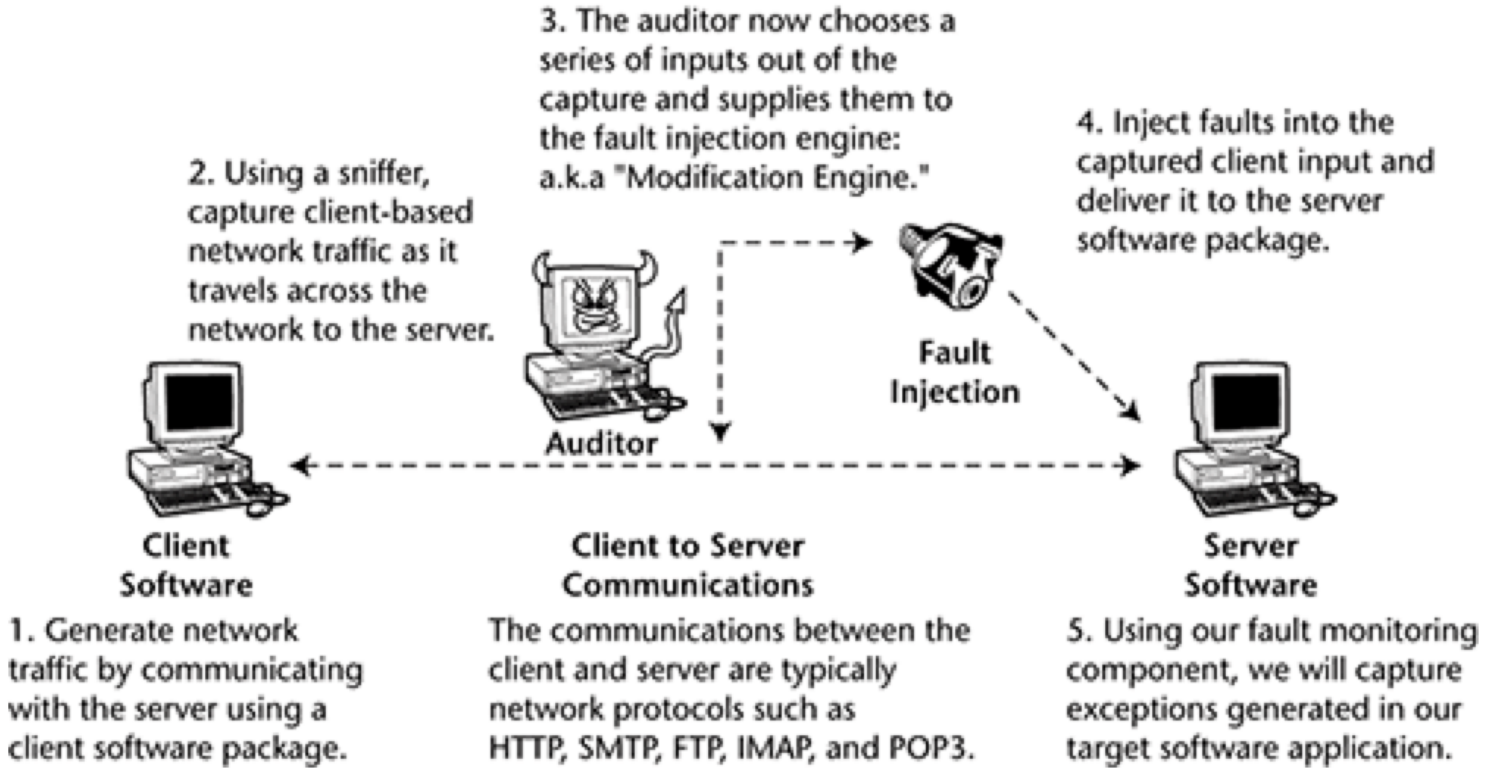
How I hacked a hardware crypto wallet and recovered $2 million

- https://www.youtube.com/watch?v=dT9y-KQbqi4

Topics

- Design Overview
- Fault Monitoring
- Putting It Together

Design Overview



Input Generation

- Select input that uses esoteric and untested software features
- This request uses the uncommon .ida filetype
  - An ISAPI filter included in IIS web server

```
GET /search.ida?group=kuroto&q=riot HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: 192.168.1.1
Connection: Keep-Alive
Cookie: ASPSESSIONIDQNNNNTEG=ODDDDIOANNCXXXXIIMGLLNNG
```

Generating Input

- Manual generation
  - Build inputs in a text editor
  - Time-consuming, but produces best results
- Automated generation
  - Creating fake input with a program
  - May waste time on buggy input

Generating Input

- Live capture
  - Inject faults directly into live network traffic
  - Requires complex adjustment of data size fields, checksums, etc.
- Fuzz generation
  - Researchers noticed core dumps when using a dial-up modem during a thunderstorm
  - Random data injection found many new faults

Fault Injection

- Open-source apps
  - Can be recompiled with special added code to improve fuzzing
  - Such as American Fuzzy Lop (link Fuzz 15)
- Closed-source apps
  - Only input data is modified

Modification Engines

- To find buffer overflows
  - Inject variable-sized data to elements
  - Use non-alphanumeric characters to delimit elements
  - Inject into elements, without altering delimiters

```
GET /index.html HTTP/1.1
Host: test.com
```

A sample run with ten iterations using the fault `EEYE2003` would produce the following faulted input streams.

Sequential fault injection:

```
EEYE2003GET /index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GEEYE2003ET /index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GEEEYE2003T /index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GETEEYE2003 /index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET EEYE2003/index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /EEYE2003index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /iEEYE2003ndex.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /inEEYE2003dex.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /indEEYE2003ex.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /indeEEYE2003x.html HTTP/1.1\r\nHost: test.com\r\n\r\n
```

Fault injection using delimiter logic:

```
GETEEYE2003 /index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET EEYE2003/index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET EEYE2003/index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /EEYE2003index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /indexEEYE2003.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /index.EEYE2003html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /index.htmlEEYE2003 HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /index.html EEYE2003HTTP/1.1\r\nHost: test.com\r\n\r\n
```

Defeating Input Sanitization

- Repeat existing characters instead of injecting new ones

```
GETTTTTTTTT /index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET ////////////index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET           /index.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /iiiiiiiiiiindex.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /indexxxxxxxxxxx.html HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /index.hhhhhhhhhhtml HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /index.htmmmmmmmmmm HTTP/1.1\r\nHost: test.com\r\n\r\n
GET /index.html HHHHHHHHHHTTP/1.1\r\nHost: test.com\r\n\r\n
GET /index.html HTTPPPPPPPPPP/1.1\r\nHost: test.com\r\n\r\n
GET /index.html HTTP/11111111111.1\r\nHost: test.com\r\n\r\n
```

Fault Delivery

1. Create network connection to target application.
2. Send our modified input data over the created connection.
3. Wait momentarily for a response.
4. Close the network connection.

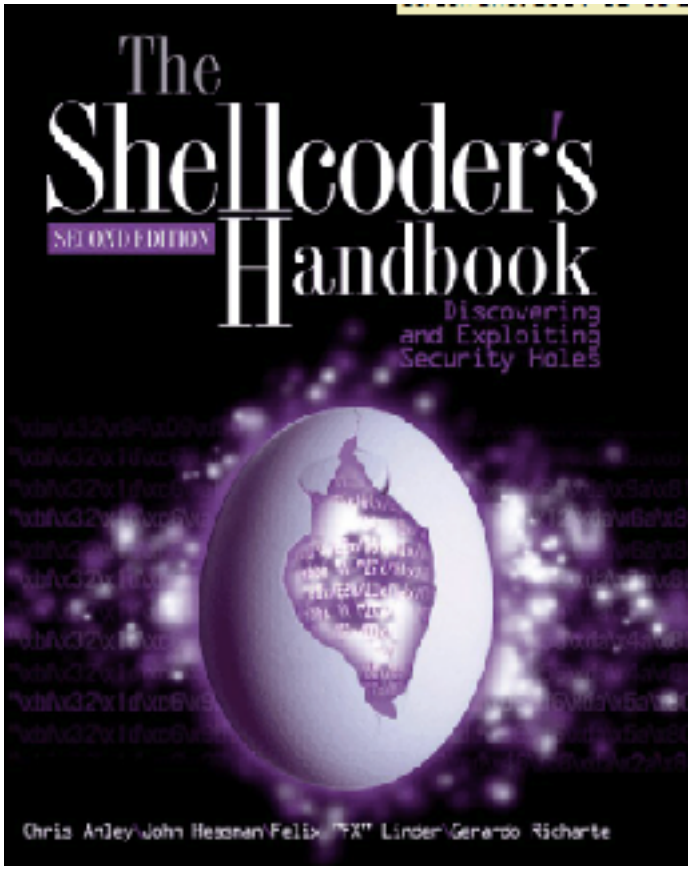
- Nagel algorithm
  - Delays transmission of small datagrams so they can be grouped together
  - Enabled by default in Windows
  - Must be disabled with NO_DELAY flag
    - Link Ch 16a

Fault Monitoring

Using a Debugger

- Good for interactive fault testing
- Capture every exception, if possible
  - Instead of passing them to the application first ("First chance")
- Access-violation exceptions are the most important
  - Indicate that data structures used to read or write to RAM were corrupted

CNIT 127: Exploit Development
Ch 17: The Art of Fuzzing



Static Analysis

- Analyzing code that is not running
- Source code or binary
- Many bugs found this way are unimportant in practice
  - Because there is no input from the user that "reaches" the buggy code
  - There's no easy way to determine the reachability of a bug from static analysis

Fuzzing is Scalable

- An SMTP fuzzer can test any SMTP server
- No need to rewrite it
- Very simple strings may apply to many protocols
  - Such as "../" * 5000

Weaknesses in Fuzzers

- Some parts of code won't be hit by a fuzzer
  - Because it requires special input values we don't know about
- Fuzzing gets very slow if many parameters vary
- Fuzzing should be supplemented by static analysis and runtime binary analysis

SPIKE

- Builds a network packet by adding data one field at a time to a "spike" data structure
- Automatically fills in size fields, checksums, etc.
- Has various sending programs
  - Such as **generic_send_tcp**

SPIKE Functions

- s_string("Hello, world!");
  - Adds the literal string **Hello World!** to the spike
- s_string_variable("MESSAGE");
  - Adds a series of varying strings to the spike
  - The first one is **MESSAGE**
- s_readline();
  - Reads a message from the server

Very Simple SPIKE Script

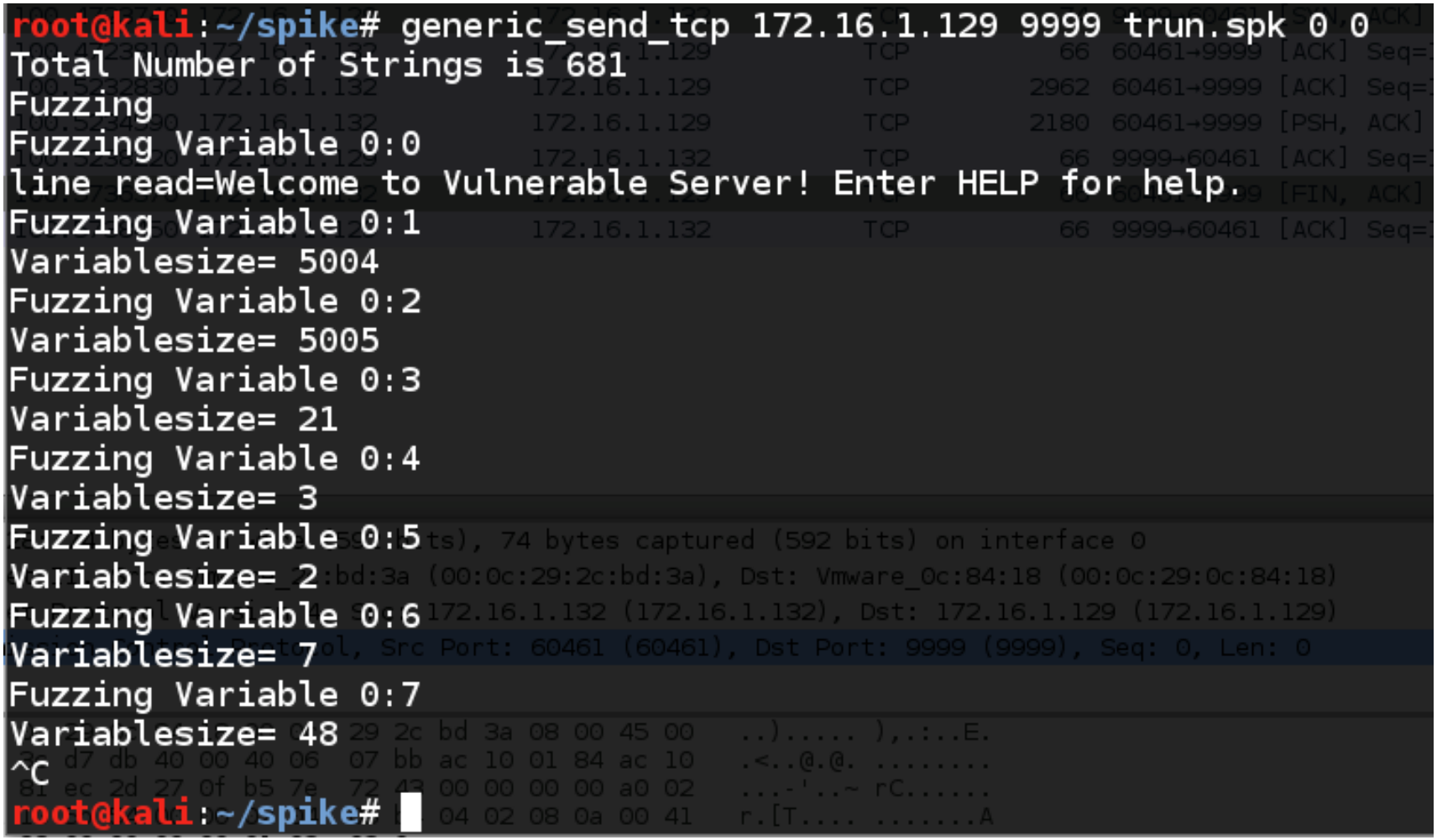
- Enough to fuzz "Vulnerable Server"

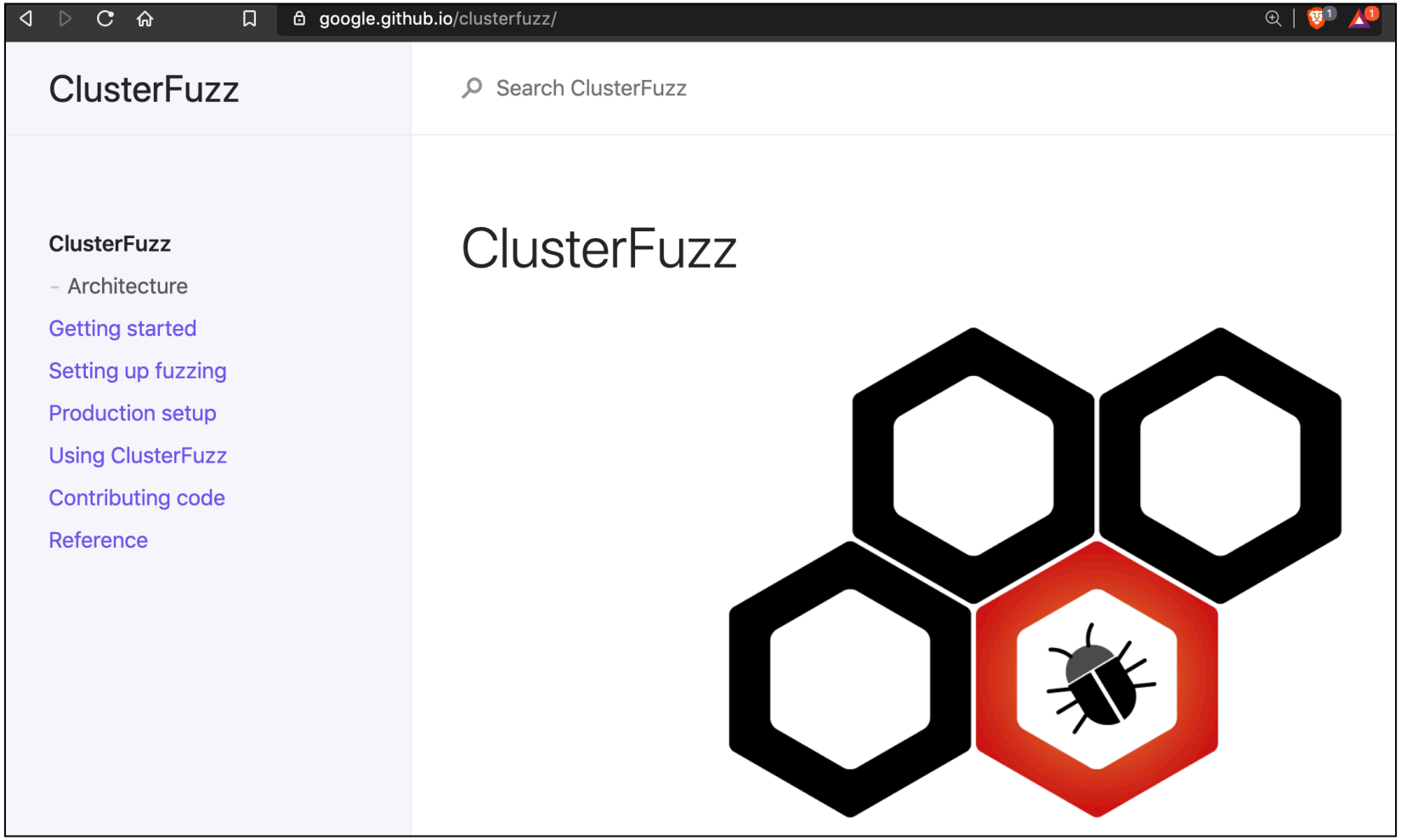
```
  GNU nano 2.2.6                File: trun.spk

s_readline();
s_string("TRUN ");
s_string_variable("COMMAND");
```

Spike Script (Partial)

```
//version
s_binary("00 01");
//Opcode (request=07)
//3 is onebyte
//5 is two byte big endian
s_int_variable(0x0007,5);
//message length
//s_binary("00 17 ");
s_binary_block_size_halfword_bigendian("message");
s_block_start("message");
//display number
s_int_variable(0x0001,5);
//connections
s_binary("01");
//internet type
s_int_variable(0x0000,5);
//address 192.168.1.100
//connection 1
s_binary("01");
//size in bytes
//s_binary("00 04");
s_binary_block_size_halfword_bigendian("ip");
//ip
s_block_start("ip");
s_binary("c0 a8 01 64");
s_block_end("ip");
//authentication name
//s_binary("00 00");
s_binary_block_size_halfword_bigendian("authname");
s_block_start("authname");
```

Fuzzing with SPIKE




google.github.io/clusterfuzz/
ClusterFuzz
Search ClusterFuzz
ClusterFuzz
Architecture
Getting started
Setting up fuzzing
Production setup
Using ClusterFuzz
Contributing code
Reference
ClusterFuzz


Kahoot!