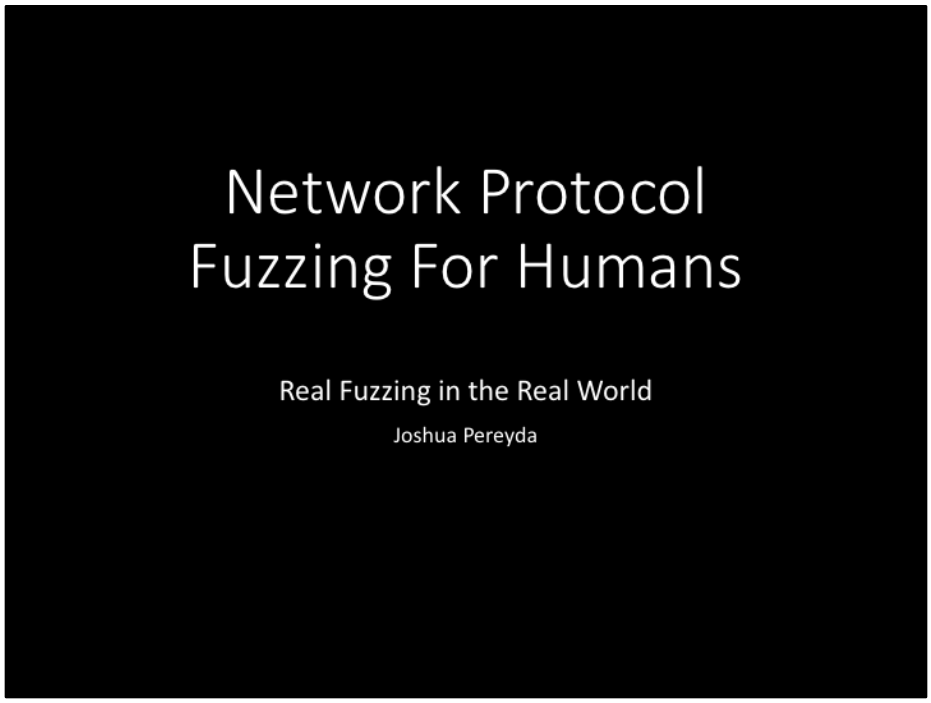


Intro:

- "I'm going to talk about Network Protocol Fuzzing for Humans"
- I spent the last year or so researching and implementing fuzzers, working with both commercial and open source tools. I found that, on the open source side, most fuzzers have a pretty typical lifecycle. You could call it a fuzzer development lifecycle.

Step 1: Build a fuzzer

Step 2: Present the fuzzer at Blackhat/Defcon

Step 3: Abandon your fuzzer

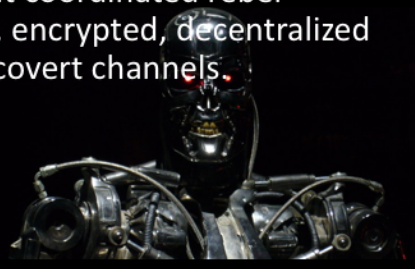
- So I found a lot of presentations and talks about fuzzers, but not a lot of hands on guidance for how to use the existing tools. And when I did try to use those tools, I found that none of them were very mature.
- In this talk, I'll
    - Introduce network protocol fuzzers
    - Why you would want to use an open source tool in this case
    - Best open source tools available
    - Why I decided to fork to a new tool called boofuzz
    - Walkthrough: Make your own network protocol fuzzer

Fuzzer Development Lifecycle

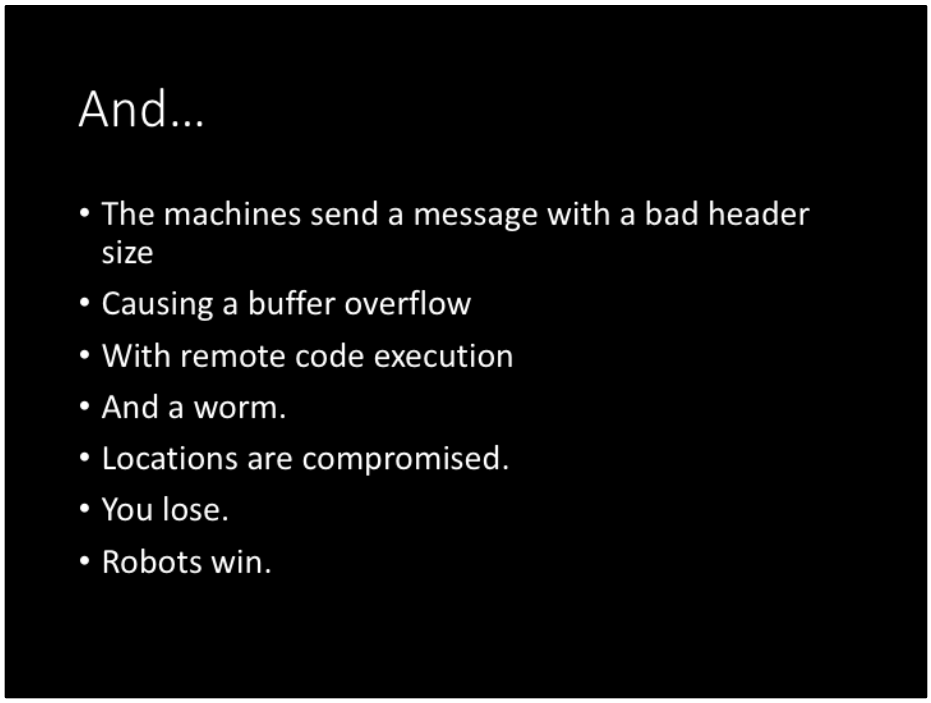
- 1. Build a fuzzer
- 2. Present the fuzzer at Blackhat/Defcon
- 3. Abandon the fuzzer

Rise of the Machines

- You are organizing a resistance to the robot army rogue AI super-human distributed hive-mind human-enslaving overlord matrix net.
- You organize disparate but coordinated rebel fighters using a high tech, encrypted, decentralized comms system riding on covert channels.
- Good job.



The Earth stands in suspense as the fate of the world hangs on the efforts of a few brave souls. And one lone programmer.
Everything so far has led up to this.
The final battle.



All your fighters are shipped off to hominid re-programming camps

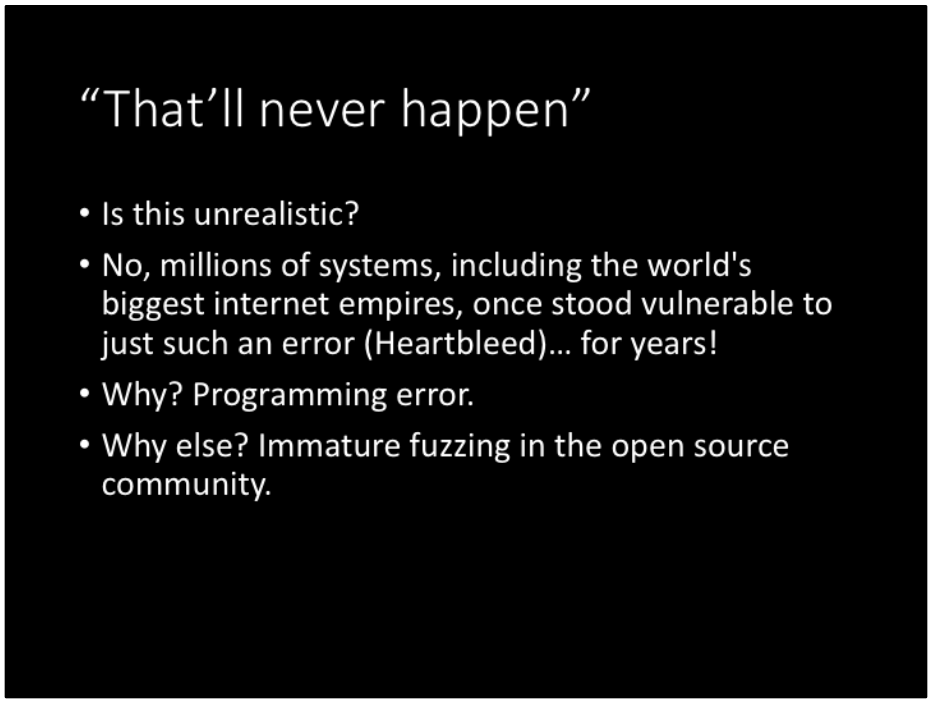
Why?

- You forgot to validate the header length in an untrusted network packet.

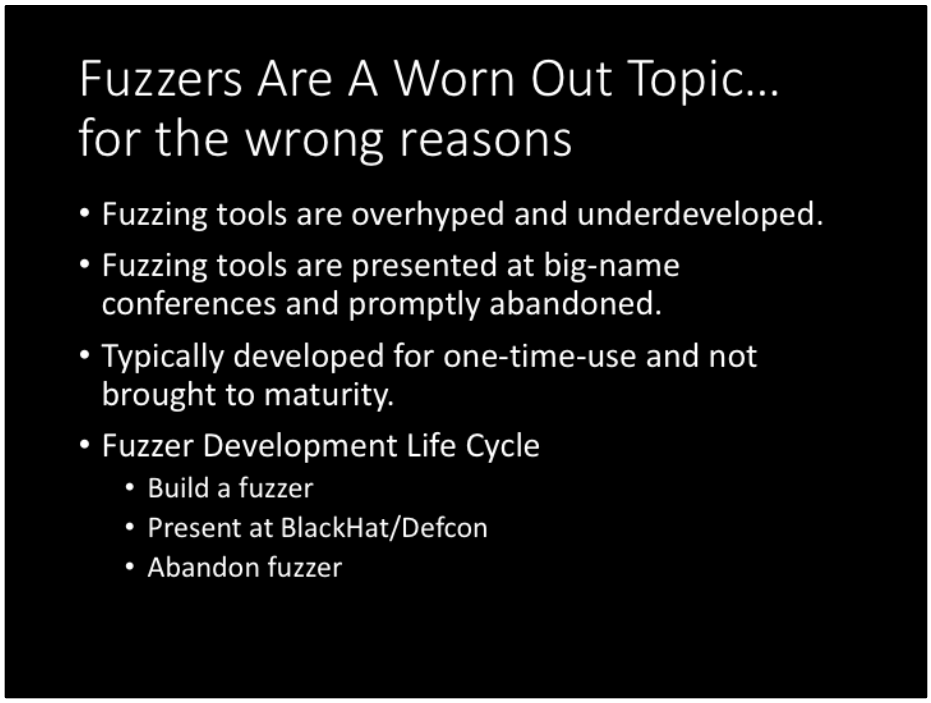
<emphasize: NEVER trust data from the network!>

Why else?

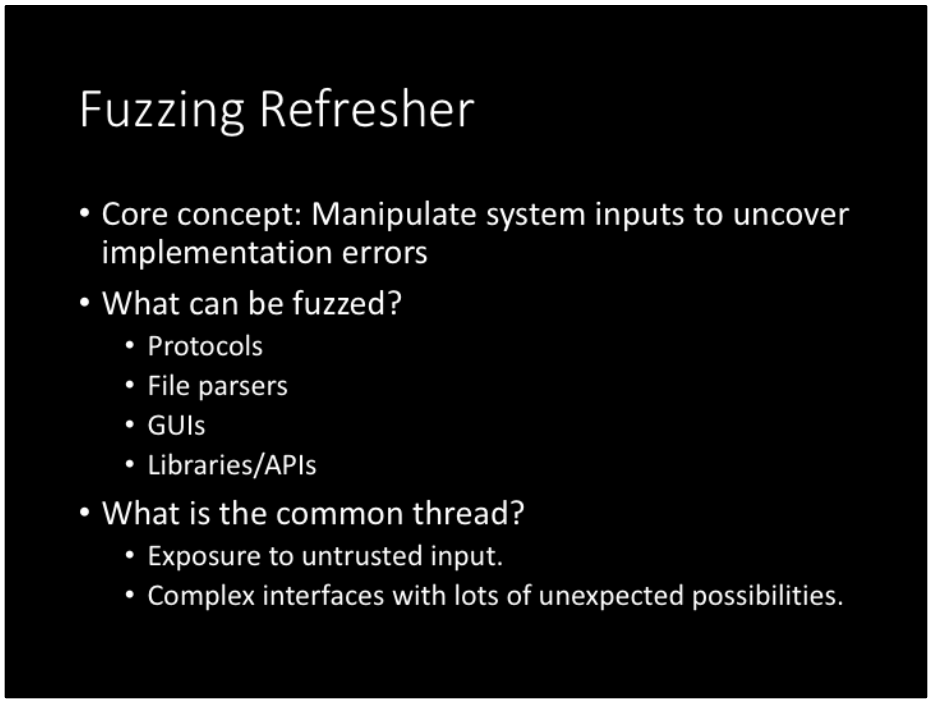
- You should have fuzzed.



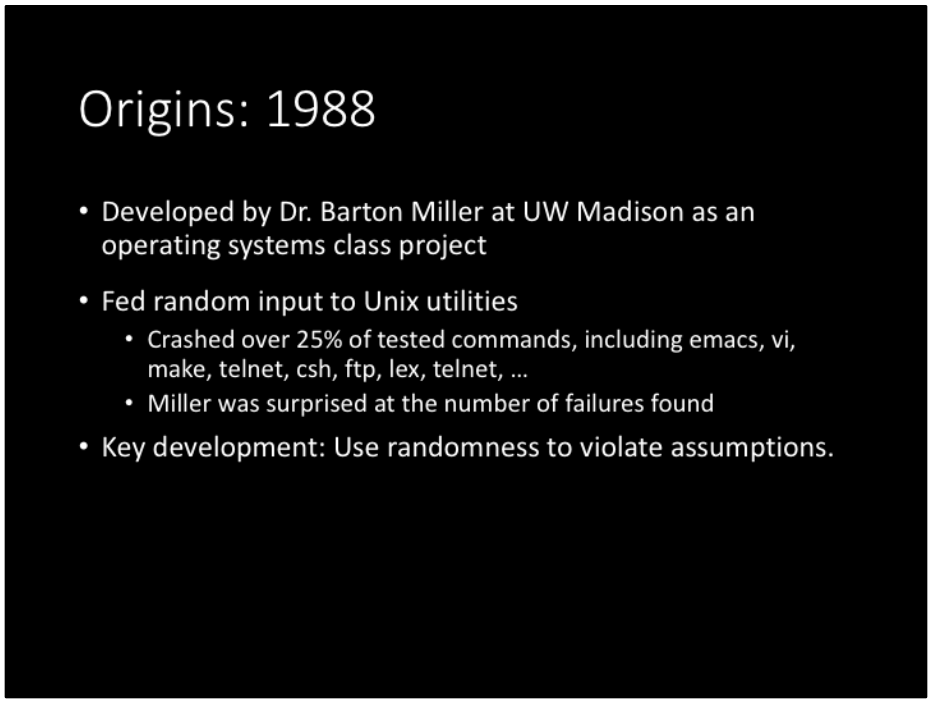
But could the fate of the world ever really hang on your network code?
Isn't this unrealistic?
These kinds of network errors cause severe vulnerabilities and go undetected for months or years, even in high quality and highly critical software.
So yes, your company's home grown app is likely to have a similar issue.



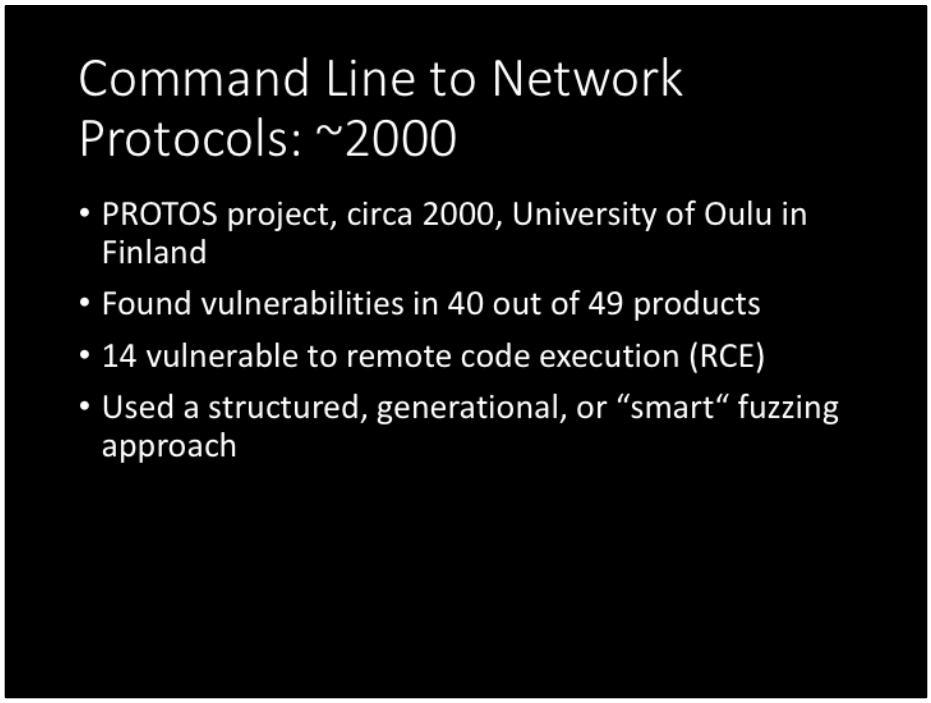
“Josh, you’re telling me that fuzzers are a big deal, but I’ve been going to security conferences for years and I’ve seen lots of fuzzers. This is old news.”
“Yeah yeah, fuzzers are important.”
Fuzzing tools are presented at big-name conferences and promptly abandoned.



What do these have in common?
- Exposure to untrusted input.
- Relatively complex; lots of invalid/unexpected inputs.

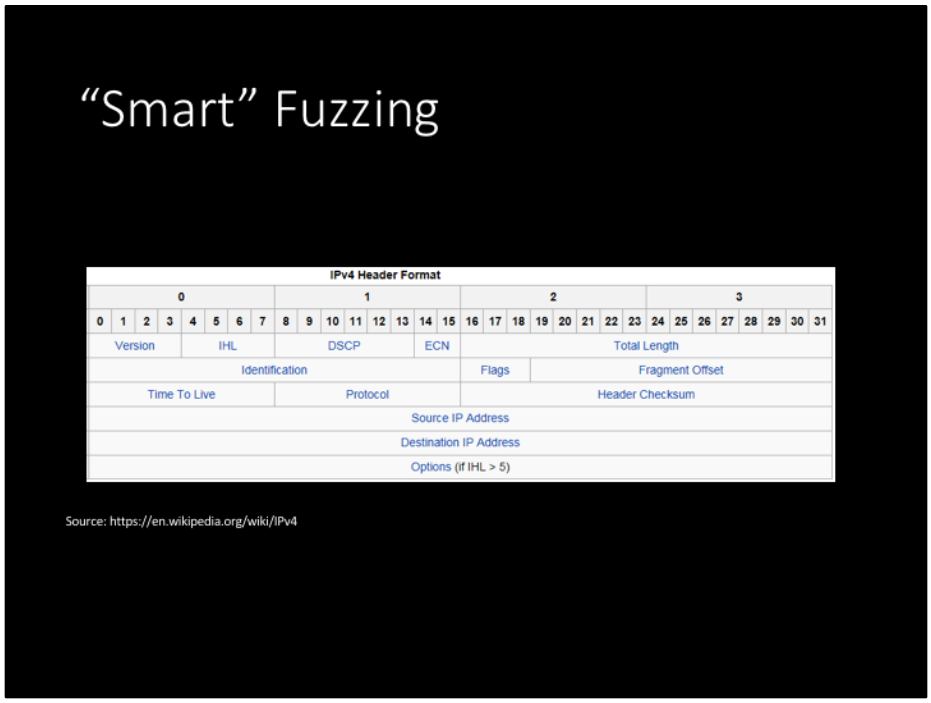


This is pretty profound: Software developed by world-renowned cutting-edge programmers (mighty men of old, the men of renown, superior men of yore, the legends, the predecessors, the forerunners, the people who used computers before it was cool) can be broken…. With totally random input.
Complete randomness… can defeat, genius.
It's surprising on the one hand, but it also makes sense if you think about it…. Totally random, unstructured input is exactly what these programmers weren't expecting. They were expecting humans to use these tools, and to make human mistakes.



Two things
- Command line to the network
- Randomness to structured randomness

“Smart” Fuzzing



IPv4 Header Format

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Version				IHL				DSCP						ECN		Total Length															
Identification																Flags			Fragment Offset											
Time To Live								Protocol								Header Checksum															
Source IP Address																															
Destination IP Address																															
Options (if IHL > 5)																															

Source: https://en.wikipedia.org/wiki/IPv4

Version, Length, …
Lots of combinations. Lots of rules to break.

Example

- -> Anomalous Message
- <- Response (optional)
- Health Check
  - -> Request
  - <- Response
  - If no response:
    - Log failure
    - Restart device

Mutational Fuzzing

- Start with a valid data sample
- Mutate valid sample to get test cases
- Advanced techniques: Use code coverage feedback to find the “interesting” test cases

Mutational vs. Generational

Mutational

- Does not know protocol
- Less protocol-specific work
- Less coverage, less bugs
- Better bugs to work ratio
- Favored use case: Offensive

Generational

- Knows protocol
- More work
- More coverage, more bugs
- Lower bugs per work ratio
- Favored use case: Defensive

Recap

- Fuzzers
- Random input
- Finds bugs
- Hackers like bugs
- -> Hackers like fuzzers
- Businesses want to fin
- -> Businesses like fuzzers

Open Source Tools

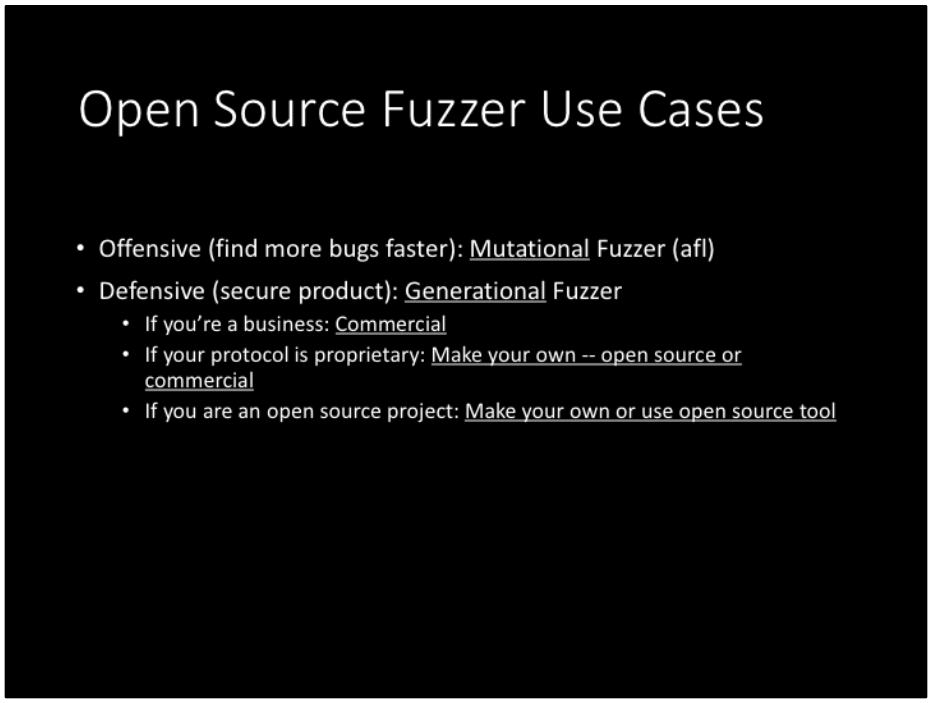
Public Domain

- Open Source Fuzzers
  - American Fuzzy Lop (afl) – Mutational
  - Peach Fuzzer – Forked to closed source by corporate sponsor.
  - Spike – Obsolete
  - Sulley – State of the art… ? Lots of bugs,
    ates
    00,000+ obscure fuzzin

Picture by Fan Wen, CC BY-SA 4.0

Open Source Generational Fuzzers: Immature

- Why?
  - Some tools lend themselves to a proprietary model. Tools that are:
    - Very valuable to a few people/businesses
    - Expensive to develop
    - In low demand in open source community
    - E.g., most defensive security tools
  - Other tools could be open source, but are plagued by anti-incentives like:
    - Commercialization
    - Businesses desire to keep methodologies secret



Why should a business bother with open source tools?
Defensive – Assurance
(Generational vs Mutational: Jared DeMott talk, Defcon 14)
"There are times when even a larger business would want to use an open source generational fuzzer framework."

Opinionated Fuzzer Decision Tree

- Offensive (find more bugs faster) or defensive (assurance)?
  - Offensive: Use a mutational/evolutionary fuzzer.
  - Defensive: Use a generational fuzzer with good protocol coverage.
    - Are you a business (time scarce) or open source project (money scarce)?
      - Open source: Must use open source tools.
      - Business: Is your protocol open or proprietary?
        - Open: Buy a tool if available.
        - Proprietary/Obscure: Must make your own. I recommend open source tools.

State of the… Fuzzing

- Commercial
  - Codenomicon Defensics (now owned by Synopsys)
  - Peach Fuzzer
- Open Source
  - AFL – Mutational
  - Sulley – Generational
    - Poor adoption
    - Buggy
    - Hard to use

The Future is Open Source

- Over time, open source dominates
  - gcc: Compilers were once for sale… when's the last time you bought a compiler?
  - git: Even enterprises are dumping "enterprise" VCS tools.
  - Metasploit: Biggest hacking tool ever. Open source.
  - Linux. Enough said.

Note: This is not a slam on commercial tools. They have an invaluable place in our industry, and the fuzzing and security world wouldn't be where it is without these businesses.

Custom Network Protocol Fuzzers Today

- Tool recap
  - Peach Fuzzer – Split to closed source
  - Spike – Superseded by Sulley
  - Sulley
    - Easy and quick data generati
    - Instrumentation and target r
    - Extensible (kind of)
    - Buggy
    - Doesn't work out of the box
    - Not actively developed
    - Poor responsiveness to pull r
    - Solution?

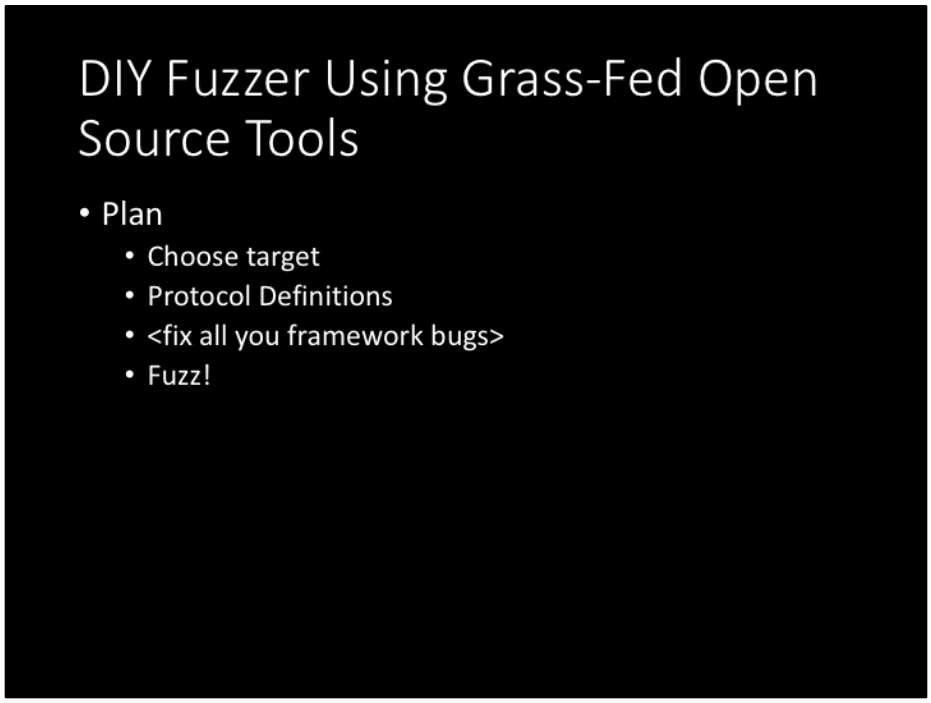
Fork!

- Got tired of waiting for Sulley pull requests
- Sulley had too many bugs to use without serious time investment
- Introducing…

boofuzz

- Sulley plus...
  - Support for arbitrary communicati
  - Extensible failure detection
  - Better test logs
  - Far fewer bugs
  - About 200 hours more engineering time

The upshot of all of this is that you can write a custom fuzzer in literally ten minutes. I'll show you how.



At end: This is what fuzzers should be. A good framework makes fuzzers easy to write, and those fuzzers can easily be applied to a wide range of software. This will make you happy if you're a hacker, because you can find bugs, or if you're a software developer, because you can find your own bugs.

Finding a Target

- FTP
- Text-based protocol
- Easy target practice:
  - Search Github
    - ftp in title
    - server in body
    - C language
    - “ftp in:title server language:C”
  - Sort by fewest stars

Boilerplate

```
from boofuzz import *
session = Session(
    target=Target(
        connection=SocketConnection("127.0.0.1", 8021, proto='tcp')))
```

Protocol Definition

```
s_initialize("user")
s_string("USER")
s_delim(" ")
s_string("anonymous")
s_static("\r\n")
```

← 220 FTP server ready
→ USER username
← 331 Password required
→ PASS asdfzxcvqwer
← 230 You may enter
→ STOR my_new_file
← 150 Opening connection

Protocol Definition

```
s_initialize("pass")
s_string("PASS")
s_delim(" ")
s_string("james")
s_static("\r\n")
```

← 220 FTP server ready
→ USER username
← 331 Password required
→ PASS asdfzxcvqwer
← 230 You may enter
→ STOR my_new_file
← 150 Opening connection

Protocol Definition

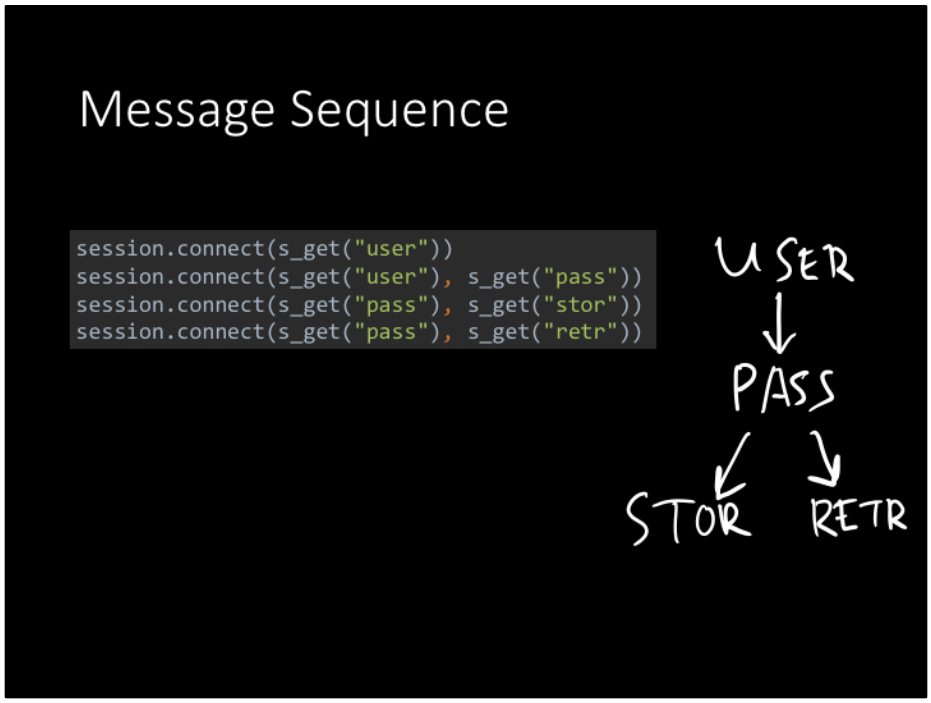
```
s_initialize("stor")
s_string("STOR")
s_delim(" ")
s_string("AAAA")
s_static("\r\n")
```

← 220 FTP server ready
→ USER username
← 331 Password required
→ PASS asdfzxcvqwer
← 230 You may enter
→ STOR my_new_file
← 150 Opening connection

Protocol Definition

```
s_initialize("retr")
s_string("RETR")
s_delim(" ")
s_string("AAAA")
s_static("\r\n")
```

← 220 FTP server ready
→ USER username
← 331 Password required
→ PASS asdfzxcvqwer
← 230 You may enter
→ STOR my_new_file
← 150 Opening connection



Notice: We didn't tell boofuzz what to do… we only described the data.

Run

```
session.fuzz()
```

```
>python ftp.py
```

Target 1

- Target: https://github.com/[redacted]/ftp
  - First repo found when sorting by fewest stars
  - 0 stars

Target 1 – Test Case 1

```
[2016-07-20 19:19:30,530] Test Case: 1
[2016-07-20 19:19:30,563]     Transmitting 12 bytes: 20 61 6e 6f 6e
79 6d 6f 75 73 0d 0a b' anonymous\r\n'
[2016-07-20 19:19:30,585]     Info: 12 bytes sent
[2016-07-20 19:19:30,598]     Received: 32 32 30 20 57 65 6c 63 6f 6d
65 20 74 6f 20 46 54 50 20 73 65 72 76 69 63 65 2e 0a 35 30 30 20 55
6e 6b 6e 6f 77 6e 20 63 6f 6d 6d 61 6e 64 0a b'220 Welcome to FTP
service.\n500 Unknown command\n'
[2016-07-20 19:19:30,607]     Check: Verify some data was received
from the target.
[2016-07-20 19:19:30,634]       Check OK: Some data received from
target.
```

Target 1 – Test Case 2

```
[2016-07-20 19:19:30,673] Test Case: 2
Traceback (most recent call last):
  File "ftp.py", line 48, in <module>
    main()
  File "ftp.py", line 44, in main
    session.fuzz()
  File "c:\users\joshpere\code\boofuzz\boofuzz\sessions.py", line 404, in fuzz
    self._fuzz_current_case(*fuzz_args)
  File "c:\users\joshpere\code\boofuzz\boofuzz\sessions.py", line 836, in _fuzz_current_case
    target.open()
  File "c:\users\joshpere\code\boofuzz\boofuzz\sessions.py", line 70, in open
    self._target_connection.open()
  File "c:\users\joshpere\code\boofuzz\boofuzz\socket_connection.py", line 118, in open
    self._sock.connect((self.host, self.port))
  File "C:\Python27\lib\socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 10061] No connection could be made because the target machine actively
refused it
```

Target 1 Server Output

Results – Target 1

- Success!
  - Server put in infinite loop (denial of service)
  - Offending test case: Test Case 1
  - Reproduce:

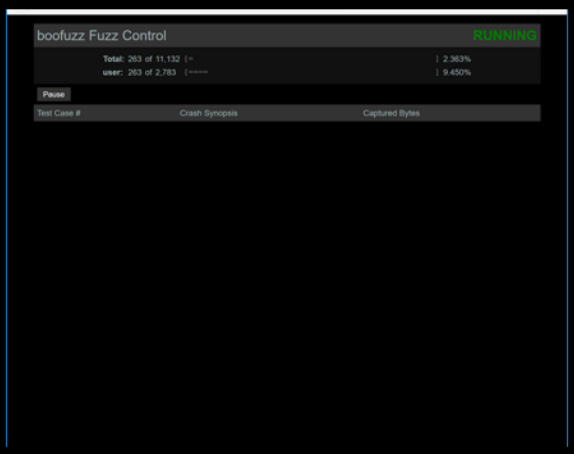
```
$ cat test
USER anonymous
$ nc 127.0.0.1 8021 < test
```

  - Likely cause: Race condition

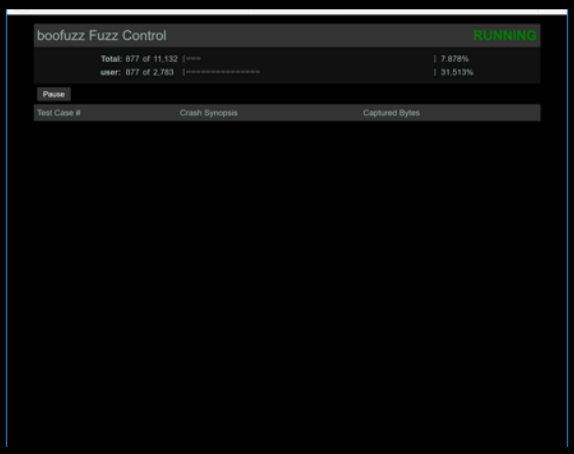
Target 2

- How about another…
  https://github.com/[redacted]/ftp
  - First page when searching for “ftp in:title server language:C”
  - 6 stars
  - Easy to build

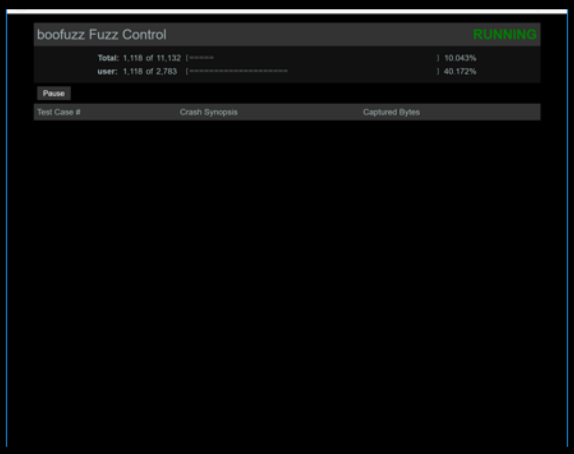
Target 2 – Web Interface



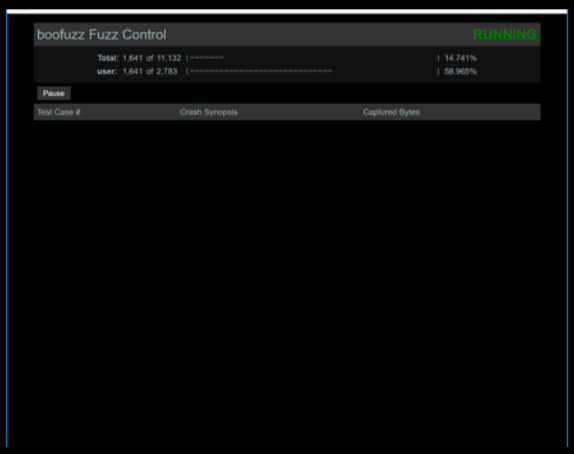
Target 2 – Web Interface



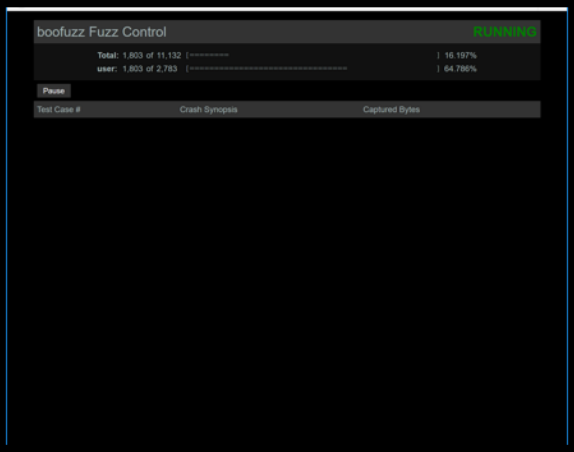
Target 2 – Web Interface



Target 2 – Web Interface



Target 2 – Web Interface



Target 2 – Crash ☹

```
C:\Users\joshpere\code\boofuzz-ftp>python ftp.py > results
Traceback (most recent call last):
  File "ftp.py", line 44, in <module>
    main()
  File "ftp.py", line 40, in main
    session.fuzz()
  File "c:\users\joshpere\code\boofuzz\boofuzz\sessions.py", line 414, in fuzz
    self._fuzz_current_case(*fuzz_args)
  File "c:\users\joshpere\code\boofuzz\boofuzz\sessions.py", line 856, in _fuzz_current_case
    self.transmit(target, self.fuzz_node, edge)
  File "c:\users\joshpere\code\boofuzz\boofuzz\sessions.py", line 727, in transmit
    self.last_recv = self.targets[0].recv(10000)
  File "c:\users\joshpere\code\boofuzz\boofuzz\sessions.py", line 110, in recv
    data = self._target_connection.recv(max_bytes=max_bytes)
  File "c:\users\joshpere\code\boofuzz\boofuzz\socket_connection.py", line 136, in recv
    data = self._sock.recv(max_bytes)
socket.error: [Errno 10054] An existing connection was forcibly closed by the remote host
```

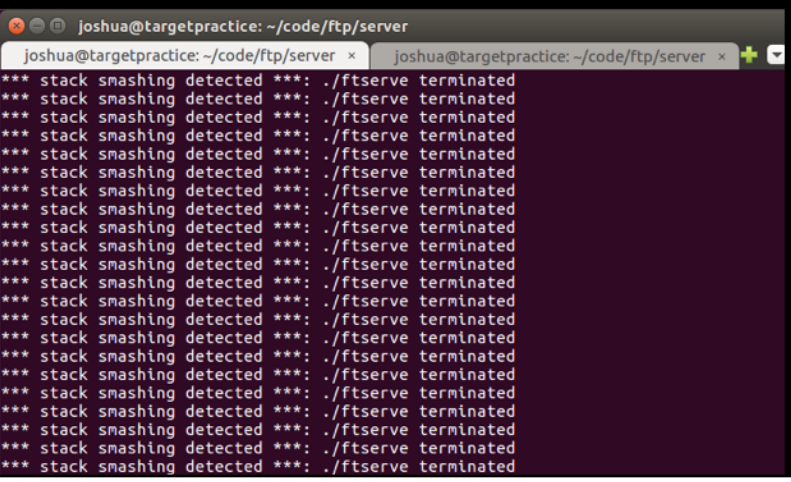
Target 2 – Server Output

```
joshpere@joshperelx4:~/code/        -ftp/server$ ./ftserve 8021
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
```

Target 2 – Server Output

```
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
error sending...
: Connection reset by peer
recv error
: Connection reset by peer
*** stack smashing detected ***: ./ftserve terminated
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
: Connection reset by peer
recv error
```

Target 2 – Server Output



Results – Target 2

- Probable buffer overflow
- Stack protection enabled -> harder to exploit
- Hard to reproduce: race condition?

boofuzz

- Quick Test Generation
- Extensible
- Few(er) bugs
- Needs
  - More users
  - More published protocols
  - More documentation
  - Features and fixes

Takeaways

- Fuzzing is an important testing activity and rapidly becoming normal
- No mature open source network fuzzing tools
- Good open source tools are critical for secure protocol implementations
- Needs for open source protocol fuzzing
  - Improved open source frameworks (boofuzz?)
  - Published protocol definitions
- Write your own protocol fuzzer… getting started is easy!

Thank You and Happy Fuzzing!

- Learn More
  - https://github.com/jtpereyda/boofuzz
  - https://github.com/jtpereyda/boofuzz-ftp
- Questions
  - @jtpereyda
  - https://gitter.im/jtpereyda/boofuzz

What can YOU do?

- Employed programmers: Does your company fuzz? Should it?
- Hackers:
  - Publish and share tools
  - Use and improve open source tools
- Open source programmers:
  - Write clean code.
  - Write tests.
  - Use CI.
  - In short, be a responsible engineer. :)
  - (But don't let messy code keep you from sharing!)