Fuzzing

Fuzzing

"Finding bugs by bombarding target with nonconform data"

- Think: Flip a few bits in a PDF, then start Acrobat with that PDF
- Just more automated

Steps:

- Create input corpus
- Select an input
- Modify input file ("fuzz it")
- Start program with input file
- (Observe program)
- Identify crashes

Fuzzer

A program which generates new inputs

Mutation:

- Modify existing test samples
- Shuffle, change, erase, insert

Generation:

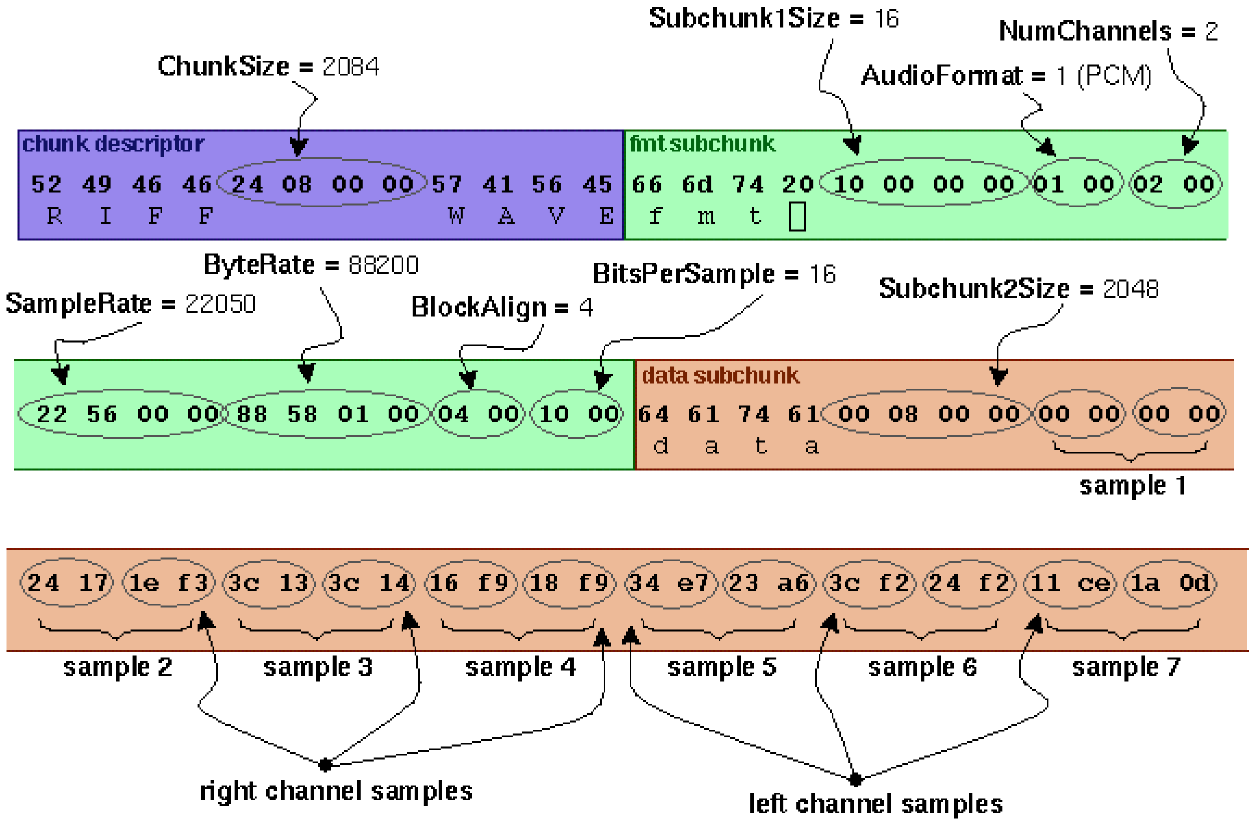
- Define new test sample based on models, templates, RFCs or documentation

Fuzzer: Mutation

Mutation fuzzing examples:

- Ffmpeg: Movie files
- Winamp: MP3 files
- Antivirus: ELF files

Take an input file, modify it a bit, continue



Fuzzer: Generation

Generation fuzzing:

- Browser: JavaScript
- Browser: HTML

Cannot just bit flip etc, as it is not a binary protocol

```
alert(1);
```

- is valid:

```
al1rt(e);
```

- is garbage

HTTP RFC

5 Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

```
Request       = Request-Line              ; Section 5.1
                *(( general-header        ; Section 4.5
                 | request-header         ; Section 5.3
                 | entity-header ) CRLF)  ; Section 7.1
                CRLF
                [ message-body ]          ; Section 4.3
```

5.1.1 Method

The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.

```
Method         = "OPTIONS"                ; Section 9.2
               | "GET"                    ; Section 9.3
               | "HEAD"                   ; Section 9.4
               | "POST"                   ; Section 9.5
               | "PUT"                    ; Section 9.6
               | "DELETE"                 ; Section 9.7
               | "TRACE"                  ; Section 9.8
               | "CONNECT"                ; Section 9.9
               | extension-method
extension-method = token
```

HTTP RFC

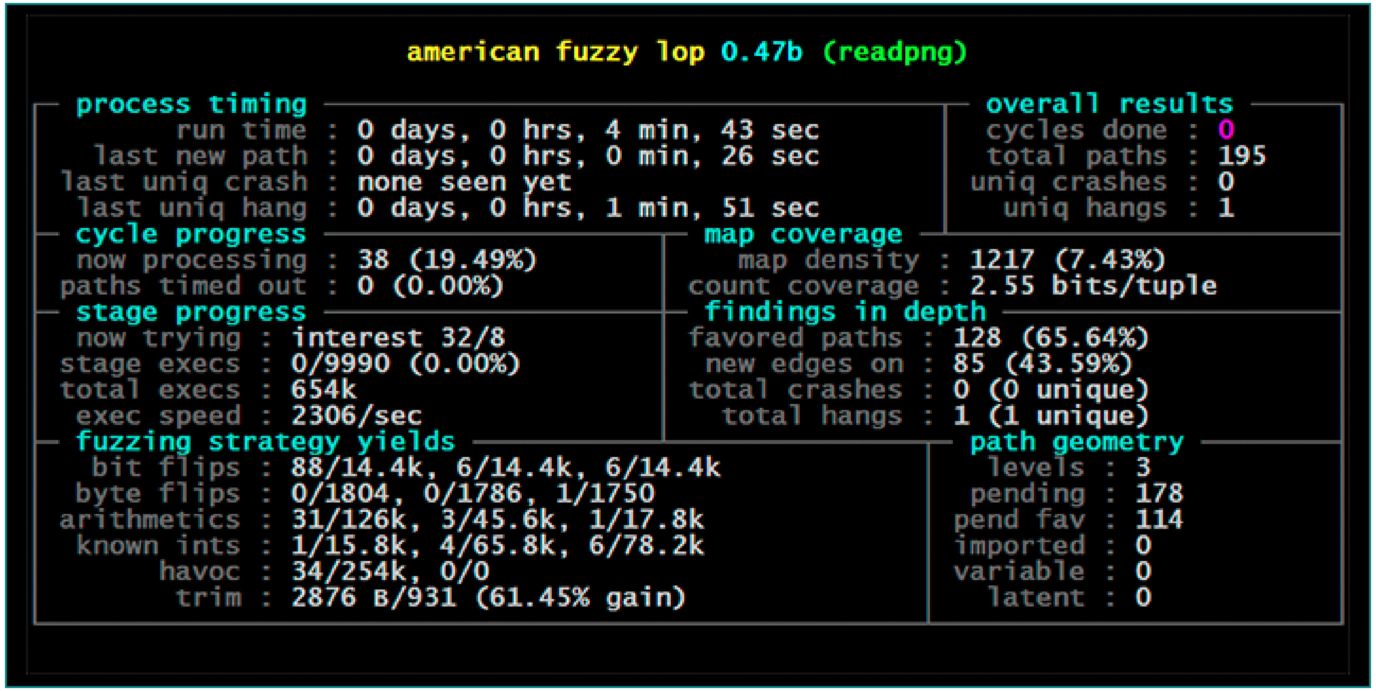
```
HTTP-date     = rfc1123-date | rfc850-date | asctime-date
rfc1123-date = wkday "," SP date1 SP time SP "GMT"
rfc850-date  = weekday "," SP date2 SP time SP "GMT"
asctime-date = wkday SP date3 SP time SP 4DIGIT
date1         = 2DIGIT SP month SP 4DIGIT
                ; day month year (e.g., 02 Jun 1982)
date2         = 2DIGIT "-" month "-" 2DIGIT
                ; day-month-year (e.g., 02-Jun-82)
date3         = month SP ( 2DIGIT | ( SP 1DIGIT ))
                ; month day (e.g., Jun  2)
time          = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                ; 00:00:00 - 23:59:59
wkday         = "Mon" | "Tue" | "Wed"
              | "Thu" | "Fri" | "Sat" | "Sun"
weekday       = "Monday" | "Tuesday" | "Wednesday"
              | "Thursday" | "Friday" | "Saturday" | "Sunday"
month         = "Jan" | "Feb" | "Mar" | "Apr"
              | "May" | "Jun" | "Jul" | "Aug"
              | "Sep" | "Oct" | "Nov" | "Dec"
```

AFL

AFL

american fuzzy lop (2.38b)

American fuzzy lop is a security-oriented fuzzer that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary. This substantially improves the functional coverage for the fuzzed code. The compact synthesized corpora produced by the tool are also useful for seeding other, more labor- or resource-intensive testing regimes down the road.



Compared to other instrumented fuzzers, *afl-fuzz* is designed to be practical: it has modest performance overhead, uses a variety of highly effective fuzzing strategies and effort minimization tricks, requires essentially no configuration, and seamlessly handles complex, real-world use cases - say, common image parsing or file compression libraries.

AFL Overview

https://lcamtuf.blogspot.ch/2014/08/a-bit-more-about-american-fuzzy-lop.html

Fuzzing is one of the most powerful strategies for identifying security issues in real-world software. Unfortunately, it also offers fairly shallow coverage: it is impractical to exhaustively cycle through all possible inputs, so even something as simple as setting three separate bytes to a specific value to reach a chunk of unsafe code can be an insurmountable obstacle to a typical fuzzer.

There have been numerous attempts to solve this problem by augmenting the process with additional information about the behavior of the tested code. These techniques can be divided into three broad groups:

- **Simple coverage maximization.** This approach boils down to trying to isolate initial test cases that offer diverse code coverage in the targeted application - and them fuzzing them using conventional techniques.
- **Control flow analysis.** A more sophisticated technique that leverages instrumented binaries to focus the fuzzing efforts on mutations that generate distinctive sequences of conditional branches within the instrumented binary.
- **Static analysis.** An approach that attempts to reason about potentially interesting states within the tested program and then make educated guesses about the input values that could possibly trigger them.

AFL Code Coverage

American fuzzy lop tries to find a reasonable middle ground between sophistication and practical utility.

In essence, it's a fuzzer that **relies on a form of edge coverage measurements to detect subtle, local-scale changes to program control flow** without having to perform complex global-scale comparisons between series of long and winding execution traces - a common failure point for similar tools.

AFL Input Generation

The output from this instrumentation is used as a part of a simple, vaguely "genetic" algorithm:

1) Load user-supplied initial test cases into the queue,
2) Take input file from the queue,
3) Repeatedly mutate the file using a balanced variety of traditional fuzzing strategies
4) If any of the generated mutations resulted in a new tuple being recorded by the instrumentation, add mutated output as a new entry in the queue.
5) Go to 2.

AFL Conclusion

What does this all mean?

- User gets several representative example files (e.g. valid WAV files)
- Put them into a directory
- AFL will:
  - find similarities of these files
  - create new input files based on the existing
  - start the target program with these input files
  - check which code path has been taken in the target program (coverage)
  - check if the program crashes
  - Repeat
- Result: Input files and corresponding core files

Demo

DARPA CDC

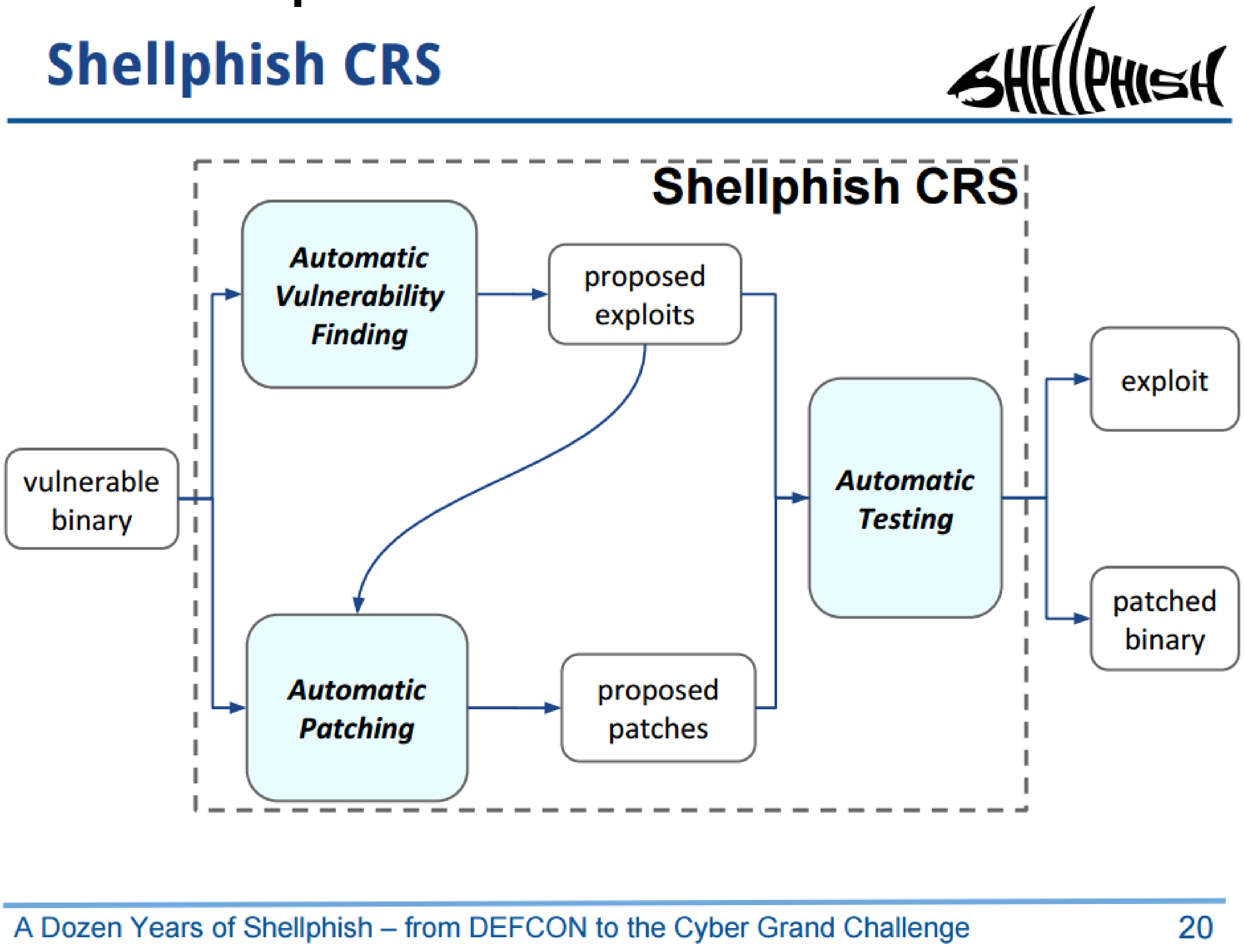
CBC

DARPA Cyber Grand Challenge 2016

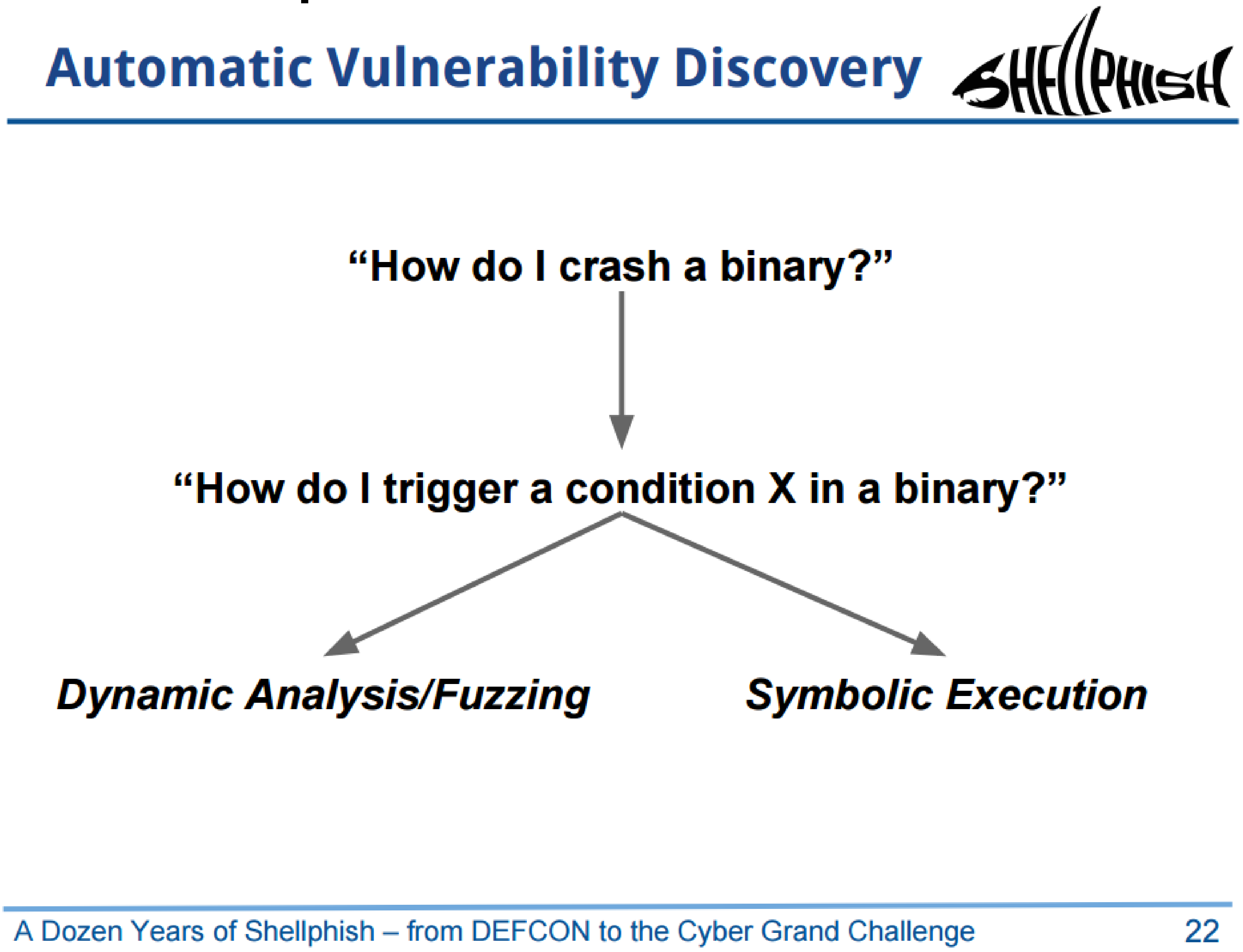
- Like the autonomous car challenge
- Teams create an autonomous system to attack and defend programs
- Programs are not real x86, but a more simplistic version
- Find bugs
  - Patch bugs in your teams computers
  - Exploit bugs in the other team computers
- Some serious HW (one rack per team, ~1000 cores, 16TB RAM)
- Finals @ Defcon Las Vegas 2016 (I was there!)

CDC

CDC Shellphish



CDC Shellphish



CDC Shellphish

Dynamic Analysis/Fuzzing

SHELLPHISH

- **How do I trigger the condition: "You win!" is printed?**

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

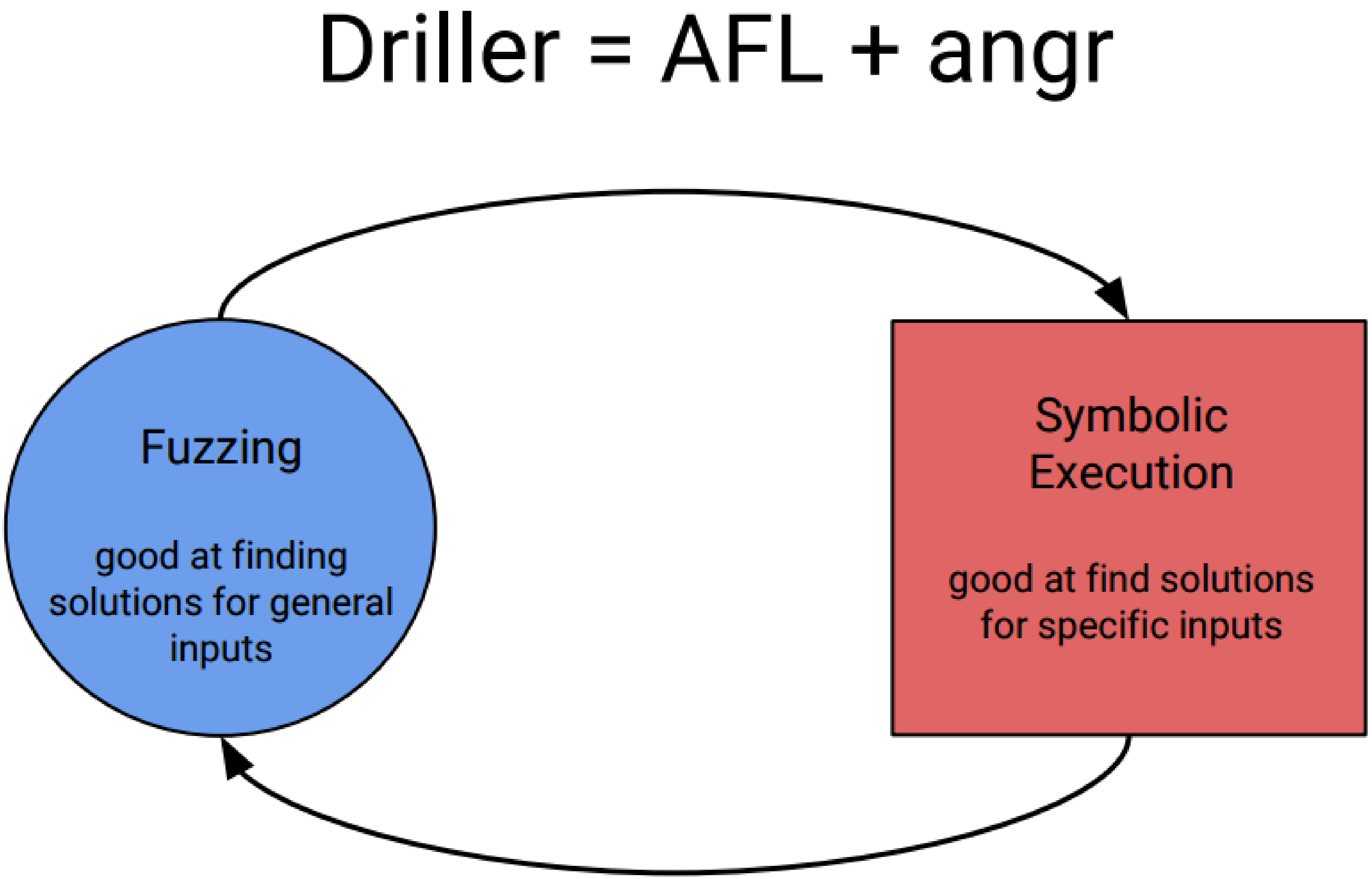
- Try "1" → *"You lose!"*
- Try "2" → *"You lose!"*
- ...
- Try "10" → *"You win!"*

Dynamic Analysis/Fuzzing

- **How do I trigger the condition: "You win!" is printed?**

```
x = int(input())
if x >= 10:
    if x == 123456789012:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

CDC Shellphish



Compiler Flags

Compiler Flags

Compiler options to enable advanced error detection routines

- GCC
- Clang

Will slow down the program massively

Will find bugs which do not directly lead to crash

Use together with fuzzing

Compiler Flags

AddressSanitizer (ASAN)

`-fsanitize=address`

- Fast memory error detector
- Out-of-bounds access to heap, stack, globals
- Use-after-free
- Use-after-return
- Use-after-scope
- Double free, invalid free
- For testing only (do not compile public releases with it!)

UndefinedBehaviourSanitizer (Bsan)

`-fsanitize=undefined`

- Finds various kinds of undefined behaviour
- Null ptr, signed integer overflow, …
- For testing only

Other fuzzing related things…

Intentionally break protocols

The future:

https://cayan.com/developers/blog-articles/how-to-protect-your-api-clients-against-breaking-c

```
Roughtime is like a small “chaos monkey” for protocols, where the Roughtime
server intentionally sends out a small subset of responses with various forms of
protocol error
```

Fuzzing: Recap

Fuzzing Recap

Fuzzing is:

- Finding bugs in programs
  - Especially exploitable bugs
- By bombard a program with:
  - Mutated/modified valid data
  - Generated semi-valid data

References

http://slides.com/revskills/fzbrowsers

- Browser Bug Hunting and Mobile (Syscan 360)

Shellphish:

- http://cs.ucsb.edu/~antoniob/files/hitcon_2015_public.pdf
- https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEFCON-24-Shellphish-Cyber%20Grand%20Shellphish-UPDATED.pdf