

Fuzzing Defined

- Automated testing technique used to find bugs in software
 1. Focus on the *attack surface*, which are the areas of code used to access or interface between systems
 - Files, APIs, Network interfaces, Reg keys, etc.
 - Also, focus on *boundary conditions*
 - Integer types, and privilege types, etc
 2. Two main types of fuzzing and one experimental
 - Mutation, Generation, and Evolutionary
 3. “Most” of the data needs to be delivered correctly for fuzzing to work well

Fuzzing in 2008

- Doesn't replace traditional testing, software audits, or reverse engineering
- But, it is one of the primary methods of 0-day discovery
- All the big companies such as Microsoft, Cisco, IBM, etc. are outsourcing it or doing it in-house – to improve software quality

When is it most effective?

- Very effective on languages like C/C++ where memory is unmanaged
 - I.e. programmer can manually define the size of static buffers as in `'char buf[100]'`, etc
- Don't forget languages like python/ruby are built with C and can call directly to C
- Could be used elsewhere like web, etc

Why not just create all test cases?

- The idea is that creating test cases is hard
 - Software testing is an NP-Hard problem
- Imagine a program that can parse a file 1 megabyte in size. How many possible files is this if you created a file for every possible combination of bytes?
 - Too many... thus fuzzing
 - Fuzzing hits many cases, quickly, and comes up with combinations the tester might not consider

Simple old-fashioned Example

- Imagine a typical client-server type protocol
 - What would happen if you totally don't follow the rules

- Either randomly or deterministically make up your own data
- Is the code robust?

Client → "Hello"

"Give me a username" ← Server

Client → "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...."

Fuzzer Types Again

- Two main types
 - Mutation – the key is good samples
 - Bit flipping (dword slidding ... “dumb” activities)
 - Effective in the past, still is via file fuzzing
 - Generation – the key is a good model
 - Protocol knowledge or “Intelligent fuzzing”
 - Allows the fuzzer to get *deep* into the protocol
 - More expensive to build
 - Evolutionary
 - Sort of like starting as a mutation fuzzer, and auto-learning how to be a generation fuzzer
 - Prototypes like EFS
 - Microsoft's new work isn't evolutionary but is cool

More ying for your yang: Monitoring

- How will we detect faults?
 - Always start by manually causing a fault in the application under test if possible.
 - This shows you what a fault will look like
 - Fuzzing is pointless if you're not sure how to detect errors
 - Debuggers, OS logs, network sniffs, etc
 - Some fuzzing tools can “binning” crashes
 - 5000 crashes could be the result of one bug

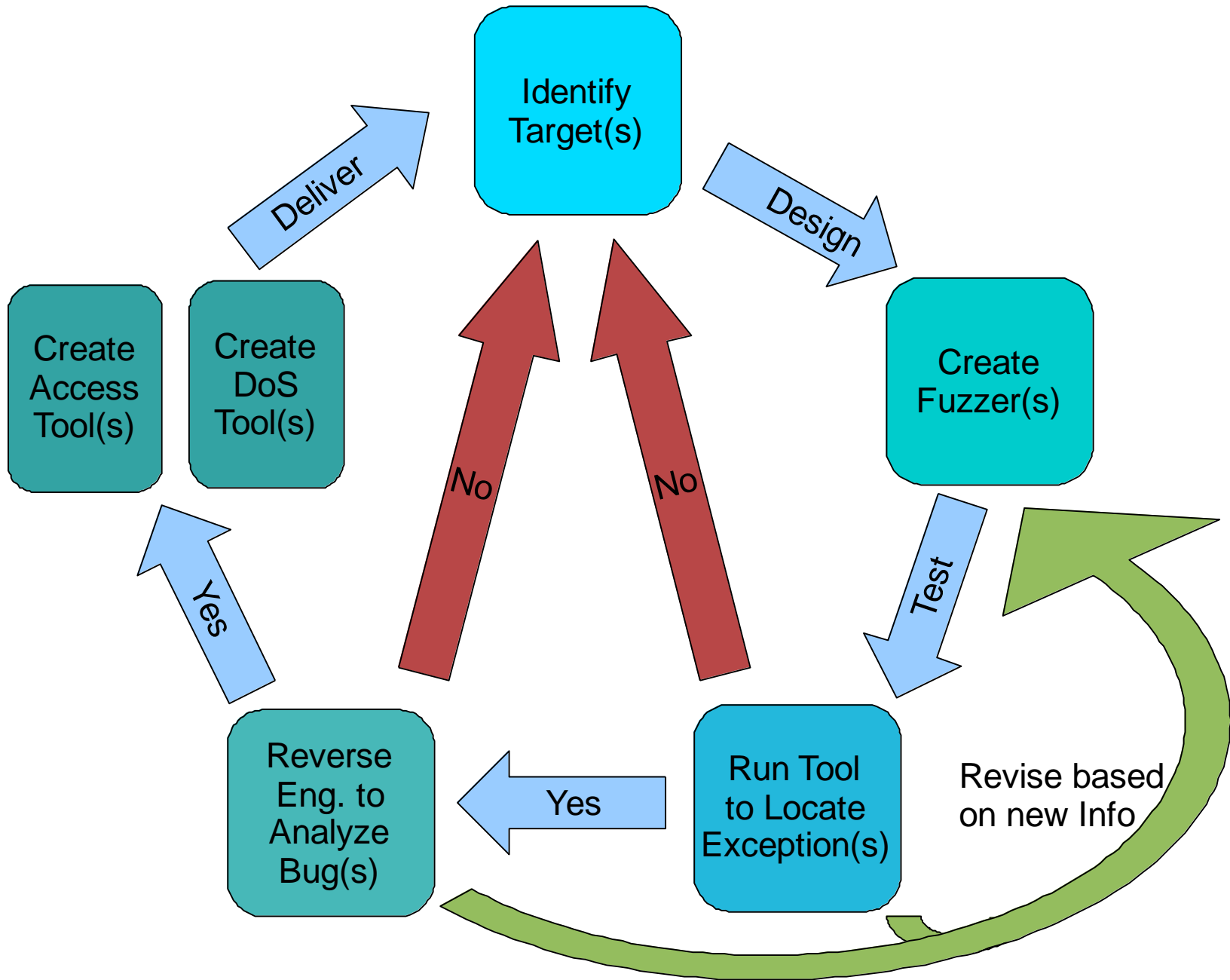
Mutation Fuzzing Ideas

- General Purpose Fuzzer (GPF)
 - Grab a network capture and fuzz!
 - Uses many heuristics ... not just random
- File fuzzing
 - Get samples for an application such as Word or Adobe Reader, randomly mutate, deliver files to application under debugger monitor, and fuzz!
 - We “dropped 0day” in our book with a simple, yet effective file fuzzer that is on the last few pages

Generation Fuzzing Ideas

- SPIKE or SPIKE file
 - Create a definition file that manually describes the protocol under test
 - based on either the RFC, a network sniff, or reverse engineering
- Sulley or build a custom fuzzer
 - We give away some Sulley TLS code in our book
- Peach fuzzing frame is kind of both
 - Uses a sample file, and parses with definition file to create some good stuff

Vulnerability Research Methodology



Which Fuzzy Approach is best?

- Mutation (dumb) tends to be quicker, but yield less total results
- Generation (intelligent) tends to yield better results, but is more expensive
- A mix is almost always the best
 - Try mutation as quickly and cheaply as possible. Than move on for better coverage if nothing was found.

Which Approach to use?

- As many as possible, but all depends on time, contract, requirements, protocol
 - Some clear text protocols such as FTP, SMTP, etc. lend themselves well to mutation fuzzing
 - Some protocols such as SSH, IKE, etc will require generation fuzzing because of the complexity of the protocol
 - **USE BOTH WHEN POSSIBLE!**

Summary

- For MUCH more information please read our book!

