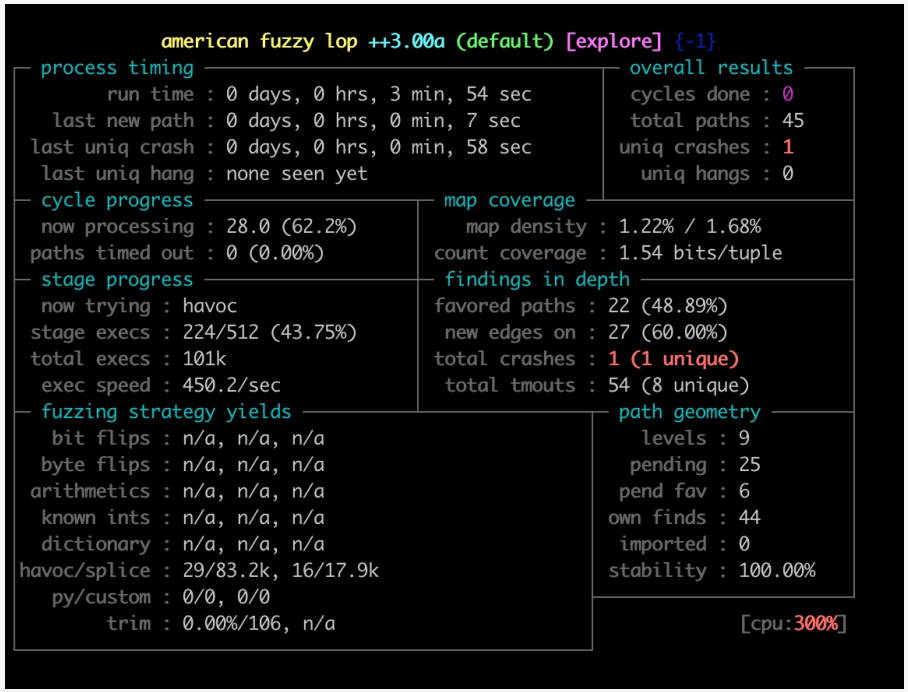
Introduction to Fuzz Testing (Fuzzing)

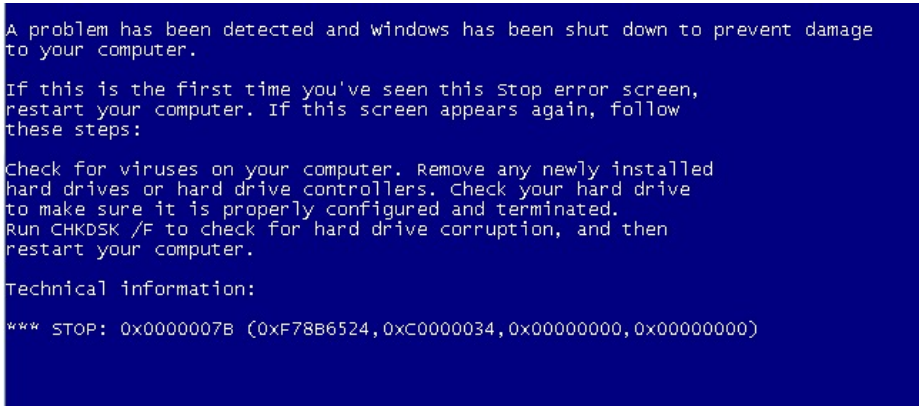


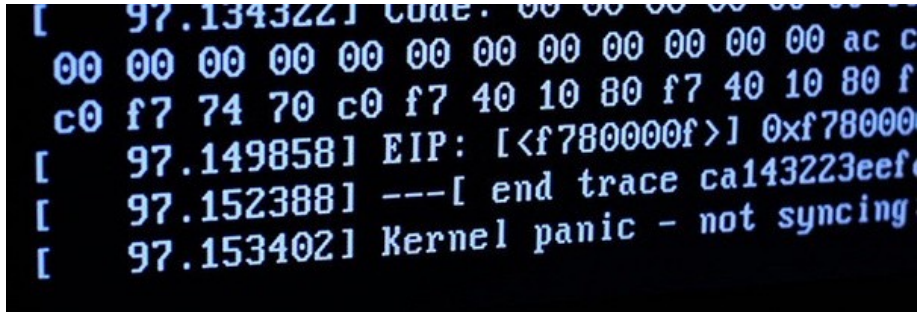
Miguel Morillo

What is fuzzing

fuzz /fʌz/

- data that has been fuzzed;
- data that has been mutated, malformed, invalid data to fuzz;
- to input/feed malformed data into a hardware or software system with the intention of crashing the system and exposing exploitable vulnerabilities.
- manipulate systems inputs to uncover implementation errors
- negative testing





Fuzzing or fuzz testing is a strategy or testing technique to find bugs in software, firmware, libraries, APIs, Network Protocols, etc. in which the system under test is stressed with unexpected inputs and data structures.

What is fuzzing

The standard definition of Fuzzing (according to the Standard Glossary of Software Engineering Terminology, IEEE):

» The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

Mozilla Fuzz testing term definition:

» Fuzzing is a technique for testing software using automated tools to provide invalid or unexpected input to a program or function in a program, then checking the results to see if the program crashes or otherwise acts inappropriately.

Similar Terms:

» Negative testing
» Protocol mutation
» Robustness testing
» Syntax testing
» Fault injection
» Rainy-day testing
» Dirty testing

Why do fuzzing

From a Security point of view

- To determine the strength of an application, protocol, host, etc.
- To test for remotely exploitable bugs
- To identify bugs reliably without false positives often missed by static program analysis and manual code inspection.

From a QA point of view

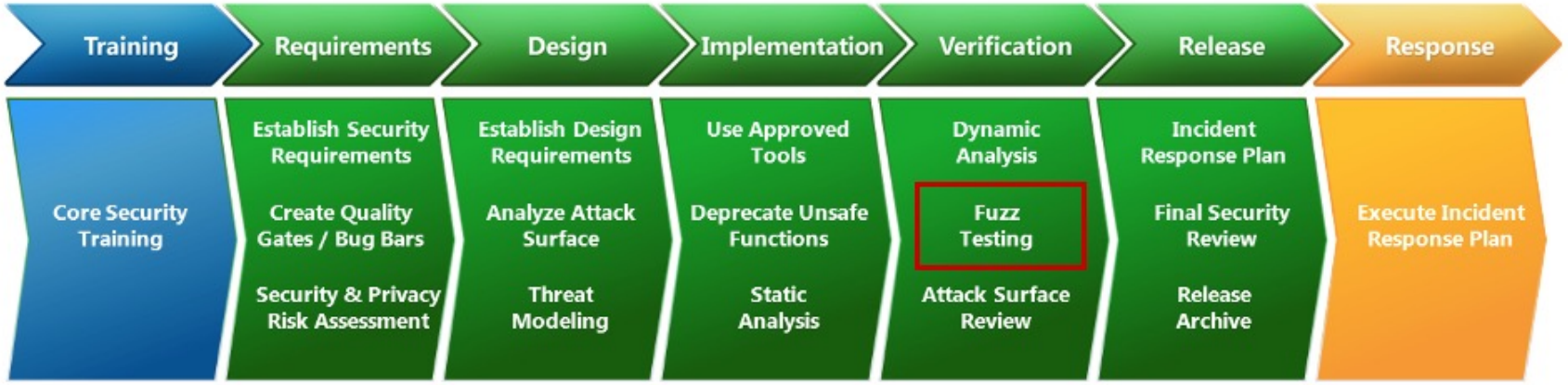
- Testing for Quality
- Code Auditing
- Conformance Testing
- Interoperability Testing
- Performance Testing
- Robustness Testing
- Stress Testing

From a compliance point of view:

- In ISA/IEC 62443 Security for industrial automation and control systems Part 4-1 includes the following in the requirements:
  - Static code analysis (tools to check and debug source code)
  - Software composition analysis (tools that generate an inventory of open source code components)
  - Malformed input testing (e.g., **fuzz testing**)
- NIST (Guidelines on Minimum Standards for Developer Verification of Software):
  - Do dynamic analysis
    - Run the program with built-in checks and protections
    - Create "black box" test cases
    - Create code-based (structural) test cases. Add cases as necessary to reach at least 80% coverage
    - Use test cases that were designed to catch previous bugs
    - Run a **fuzzer**. If the software runs a web service, run a web application scanner.

When do fuzzing

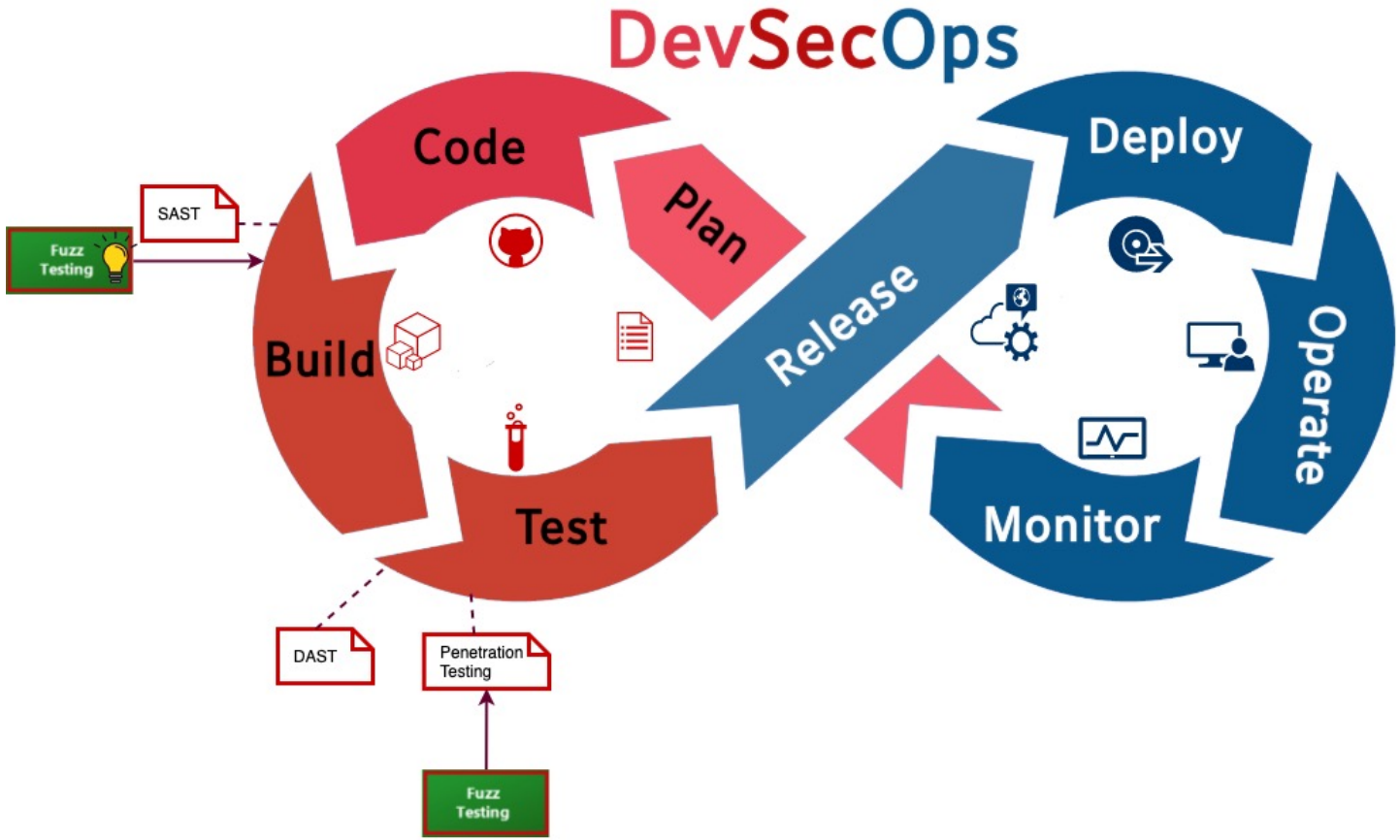
During a typical software development lifecycle usually is defined in the lasts steps or phases the fuzz testing, in conjunction with Dynamic analysis and Pentesting, like defined in the *Microsoft Security Development Lifecycle*:



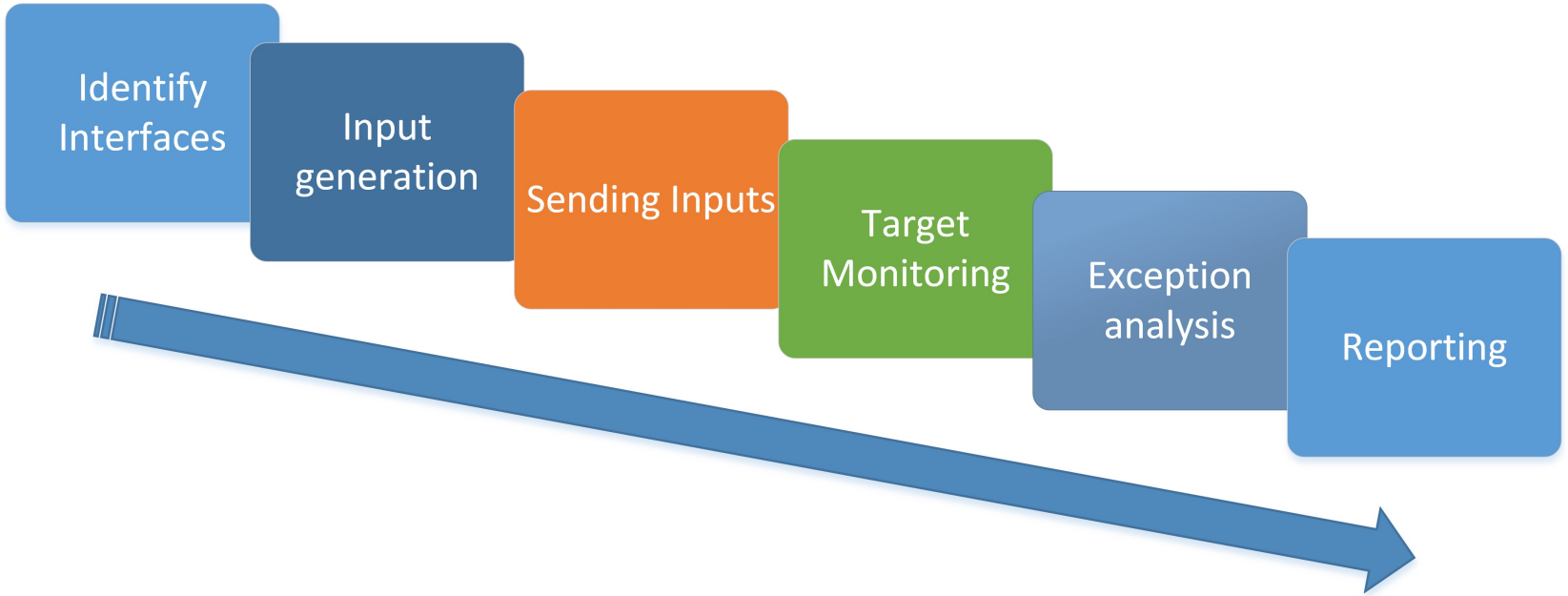
SDL Practice 12: Fuzz Testing

Fuzz testing is a specialized form of dynamic analysis used to induce program failure by deliberately introducing malformed or random data to an application. The fuzz testing strategy is derived from the intended use of the application and the functional and design specifications for the application. The security advisor may require additional fuzz tests or increases in the scope and duration of fuzz testing.

When do fuzzing

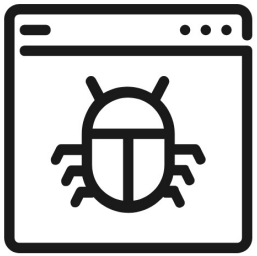


Fuzzing Life Cycle



Types of bugs or attacks

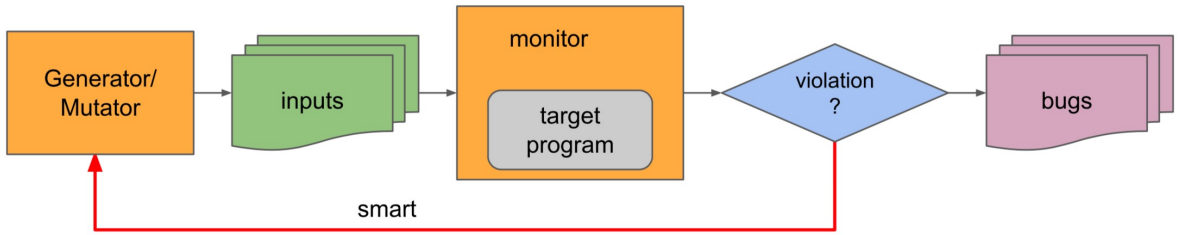
- Brute Force Login
- Race Conditions
- Denials of Service
- Cryptographic Attacks
- Web Applications
  - File Inclusions
  - SQL Injections
  - XPath, XQuery, XXE, and Other Injection Attacks
  - Cross-Site Scripting (XSS)



- Memory Corruption
  - Stack Overflows
  - Format String Errors
  - Integer Errors
  - Heap Overflow
  - (Uninitialized) Stack or Heap Variable Overwrites
  - Memory Overwrites
  - Segmentation Faults

Types of Fuzzers - Based on inputs

A fuzzer can be classified as **generation** or **mutation** based depending on whether inputs are generated from scratch or by modifying existing inputs.



Mutation (Dumb Fuzzer)

» A mutation-based fuzzer leverages an existing corpus of seed inputs during fuzzing. It generates inputs by modifying (mutating) the provided seeds.

Generation (Intelligent Fuzzer)

» A generation-based fuzzer generates inputs from scratch, based on the model previously created. Unlike mutation-based fuzzers, a generation-based fuzzer does not depend on the existence or quality of a corpus of seed inputs.

Types of Fuzzers - Based on target and input structure

A fuzzer can be dumb or smart depending on whether it is aware of input structure. A smart (model-based, grammar-based, or protocol-based) fuzzer leverages the input model to generate a greater proportion of valid inputs.

Input models

- Local applications (e.g. arguments)
- File Formats
- OS Kernel functions
- Network protocols
- Web/APIs
- General Fuzzers: File and network, and also others via custom I/O interfaces. For example: COM, shared libraries, RPC, etc.
- Custom or One-off Fuzzers: These are custom written fuzzers that target a specific format or network protocol. Typically these hand written, many times by testers.

Types of Fuzzers - Based on program structure

A fuzzer can be whitebox, greybox, or blackbox, depending on whether it is aware of program structure.

BlackBox (Dump fuzzer)

» A black-box fuzzer treats the program as a black box and is unaware of internal program structure.

WhiteBox (Test cases/Grammar-based)

» A white-box leverages program analysis to systematically increase code coverage or to reach certain critical program locations.

GreyBox (Coverage guided fuzzing)

» Coverage guided fuzzing (also known as greybox fuzzing) uses program instrumentation to trace the code coverage reached by each input feed to a fuzz target. Fuzzing engines use this information to make informed decisions about which inputs to mutate to maximize coverage.

» For every target, the fuzzing engine builds a corpus (set of inputs for a fuzz target) of inputs. These grow in coverage over time as the engine discovers new inputs through mutation.

List of Fuzzers

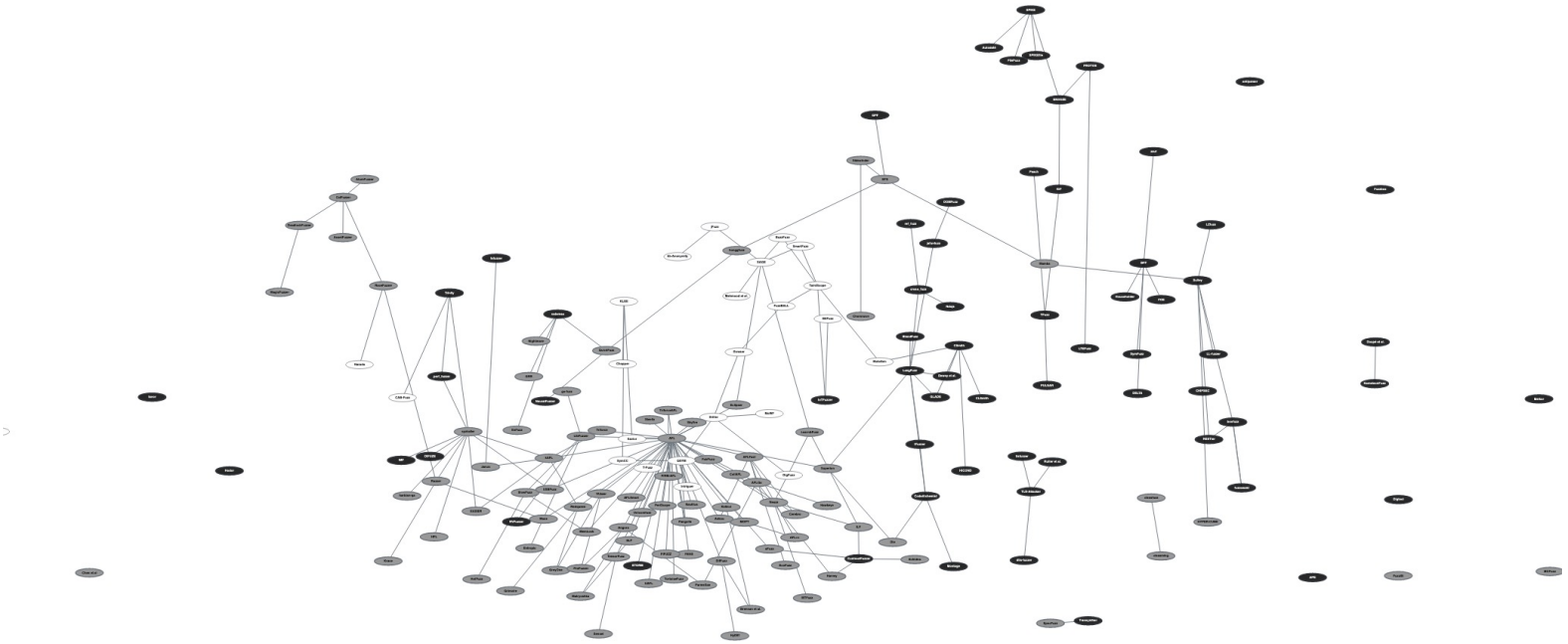
OpenSource Blackbox/Greybox	
Tool	**Link**
Project OneFuzz	https://github.com/microsoft/onefuzz
LibAFL	https://github.com/AFLplusplus/LibAFL
AFL++	https://github.com/AFLplusplus/AFLplusplus
AFLnet	https://github.com/aflnet/aflnet
Libfuzzer	https://github.com/Dor1s/libfuzzer-workshop
oss-fuzz	https://github.com/google/oss-fuzz
ffuf	https://github.com/ffuf/ffuf
Wfuzz	http://www.edge-security.com/wfuzz.php
Peach	https://github.com/MozillaSecurity/peach https://sourceforge.net/projects/peachfuzz/
Sulley	https://github.com/OpenRCE/sulley
Boofuzz	https://github.com/jtpereyda/boofuzz
Fuzzowski	https://github.com/nccgroup/fuzzowski
RESTler	https://github.com/microsoft/restler-fuzzer
radamsa	https://gitlab.com/akihe/radamsa
Mutiny	https://github.com/Cisco-Talos/mutiny-fuzzer
Morph	https://github.com/walkerfuz/morph
Manul	https://github.com/mxmssh/manul
Honggfuzz	https://github.com/google/honggfuzz

OpenSource WhiteBox (code coverage)	
Language	**Fuzzing Engine**
C/C++	http://lcamtuf.coredump.cx/afl/
C/C++	https://llvm.org/docs/LibFuzzer.html
GoLang	https://github.com/dvyukov/go-fuzz
Swift	https://github.com/apple/swift/blob/master/docs/libFuzzerIntegration.md
Rust	https://github.com/rust-fuzz/cargo-fuzz
Java	https://github.com/rohanpadhye/JQF
Java	https://gitlab.com/gitlab-org/security-products/analyzers/fuzzers/javafuzz
Python	https://github.com/fuzzitdev/pythonfuzz
JavaScript	https://github.com/fuzzitdev/jsfuzz
Python	https://github.com/google/atheris

Commercial	
Tool	**Link**
PeachFuzzer	https://www.peach.tech/
Defensics	https://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html
mayhem	https://forallsecure.com/mayhem
CI Fuzz	https://www.code-intelligence.com/
beSTORM	https://beyondsecurity.com/solutions/bestorm.html
FuzzLabs	https://fuzzlabs.guardara.com/

Fuzzer's genealogy

https://fuzzing-survey.org/



This site is maintained by SoftSec Lab. and CSRC, KAIST.

Demo *1*

Code coverage-guided fuzzer for python

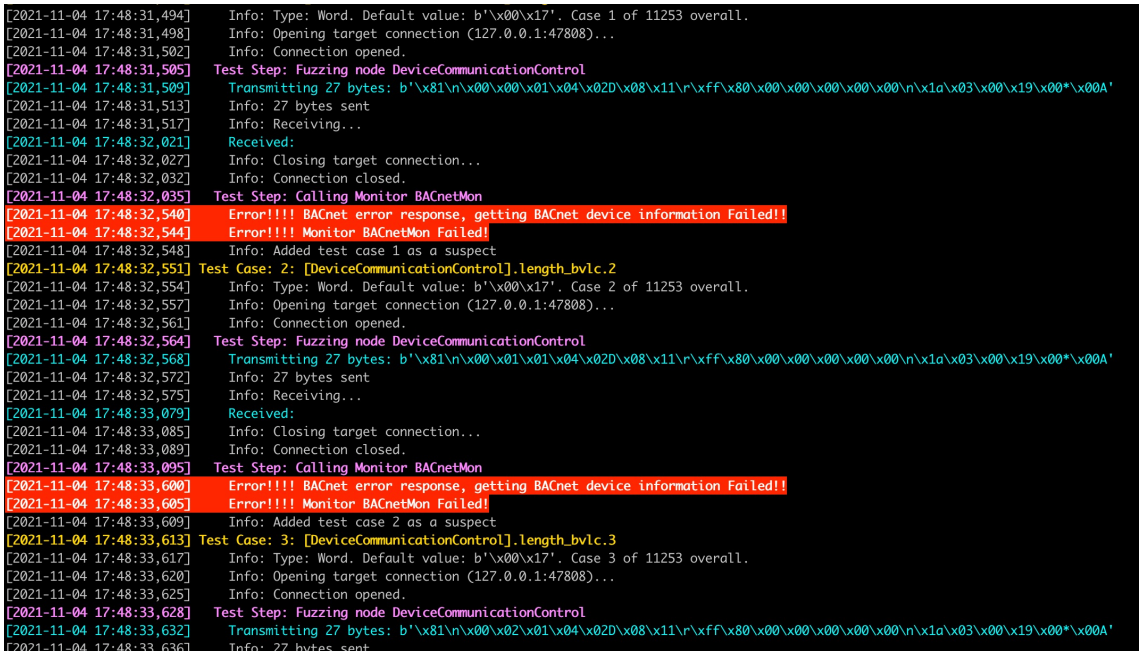
```python
from html.parser import HTMLParser
from pythonfuzz.main import PythonFuzz


@PythonFuzz
def fuzz(buf):
    try:
        string = buf.decode("ascii")
        parser = HTMLParser()
        parser.feed(string)
    except UnicodeDecodeError:
        pass


if __name__ == '__main__':
    fuzz()
```

Code coverage-guided fuzzer for python

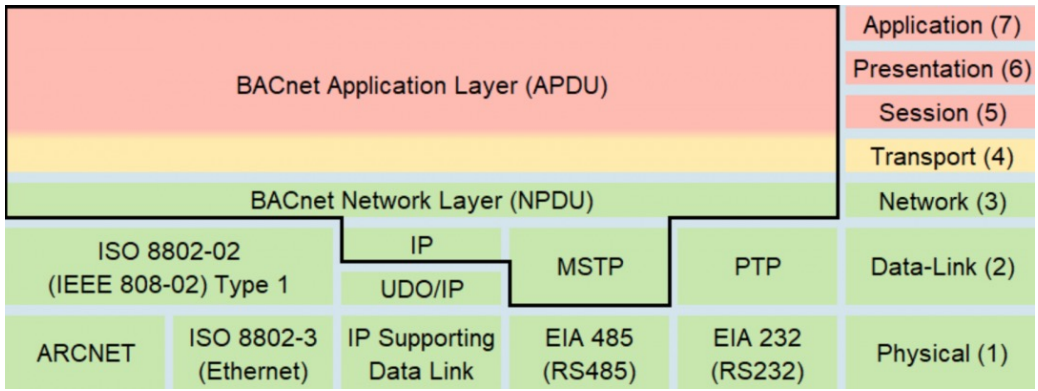
Example of a simple fuzz function for the built-in html module

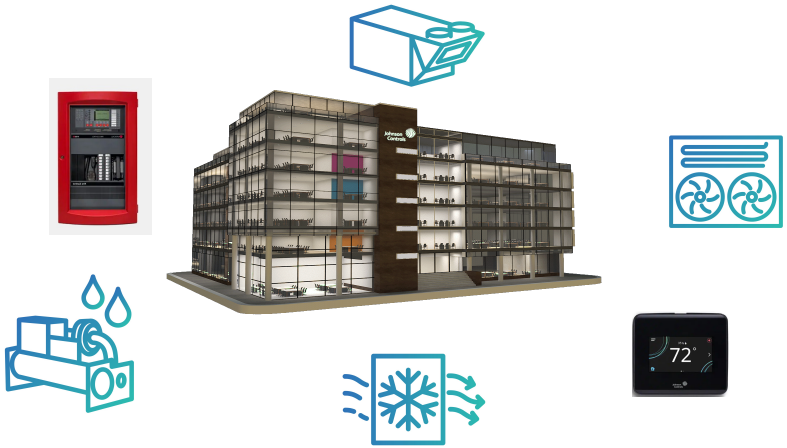
Demo *2*

BACnet network protocol



Demo2 - BACnet network protocol





SERVICE	BACnet	DESCRIPTION
AcknowledgeAlarm	C	Used to tell sender of alarm that a human has seen the alarm.
ConfirmedCOVNotification	C	Tells subscribing devices of the COV that occurred in a property.
ConfirmedEventNotification	C	Used to tell sender of a possible error condition.
GetAlarmSummary	C	Requests from a device a list of "active alarms," if any.
GenEnrollmentSummary	C	Requests a list of "event" (possible error) generating objects.
SubscribeCOV	C	Sent by a device to request that it be told of COVs in an object.
UnconfirmedCOVNotification	U	Tells subscribing devices that a change has occurred to one or more properties of a particular object.

SERVICE	BACnet	DESCRIPTION
DeviceCommunicationControl	C	Tells a device to stop (and start!) accepting network messages.
ConfirmedPrivateTransfer	C	Sends a vendor-proprietary message to a device.
UnconfirmedPrivateTransfer	U	Sends a vendor-proprietary message to one or more devices.
ReinitializeDevice	C	Orders the receiving device to cold- or warm-boot itself.
ConfirmedTextMessage	C	Conveys a text message to another device.
UnconfirmedTextMessage	U	Sends a text message to one or more devices.
TimeSynchronization	U	Sends the current time to one or more devices.
Who-Has	U	Asks which BACnet devices contain a particular Object.
I-Have	U	Affirmative response to Who-Has, broadcast.
Who-Is	U	Asks about the presence of specified BACnet devices.
I-Am	U	Affirmative response to Who-Is, broadcast.

SERVICE	BACnet	DESCRIPTION
AddListElement	C	Adds one or more items to a property that is a list.
RemoveListElement	C	Removes one or more items from a property that is a list.
CreateObject	C	Used to create a new instance of an object in the serving device.
DeleteObject	C	Used to delete a particular object in the serving device.
ReadProperty	C	Returns a value of one property of one object.
ReadPropertyConditional	C	Returns the values of multiple properties in multiple objects.
ReadPropertyMultiple	C	Returns the values of multiple properties of multiple objects.
WriteProperty	C	Writes a value to one property of one object.
WritePropertyMultiple	C	Writes values to multiple properties of multiple objects.

SERVICE	BACnet	DESCRIPTION
AtomicReadFile	C	Requests part or all of a File object's file.
AtomicWriteFile	C	Writes to part or all of a File object's file.

Advantages and Disadvantages of Fuzz Testing

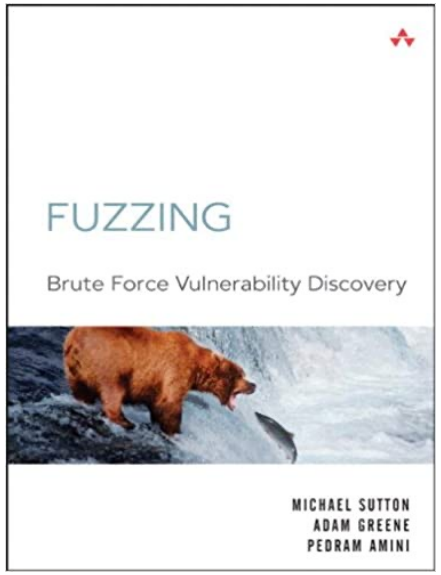
Advantages

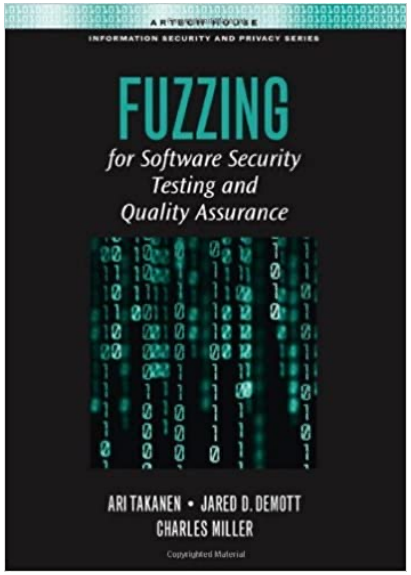
- Fuzz testing improves software quality and the security of the applications and services.
- Bugs found during fuzzing are more interesting, useful for further investigation and severe including crashes, memory leak, unhandled exception, information exposure, etc.
- Help DevOps to identify and find bugs not identified during deployment and deployment phases.

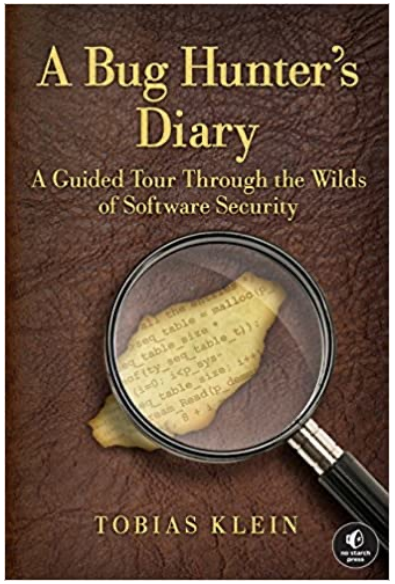
Disadvantages

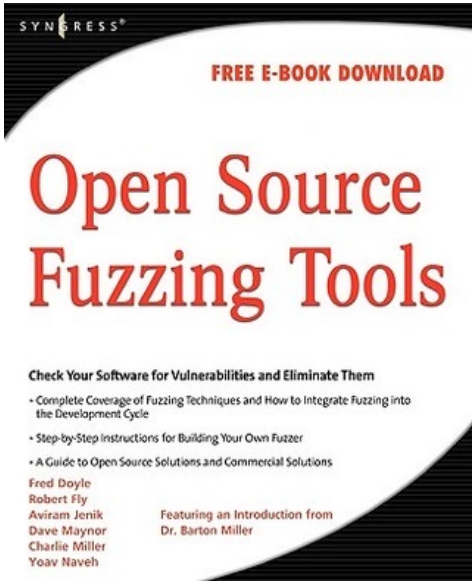
- Fuzz testing is not magic and cannot identify all security bugs or provide a complete picture of all threats.
- Fuzz testing is less effective than QA testing from a performance point of view and don't cover other security issues (e.g. malware infection)
- Fuzz testing require big efforts and amount of time to identify and do fuzzing in the most effective parts of a program or service.

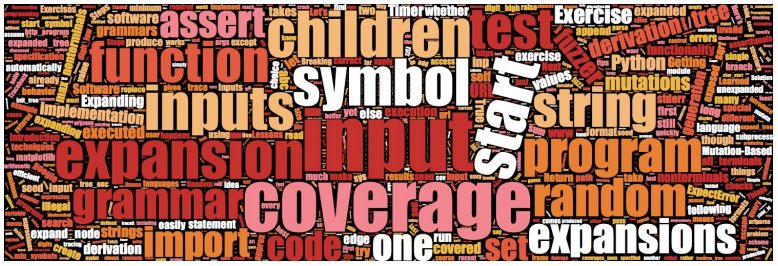
Books and resources











https://www.fuzzingbook.org/

https://github.com/roddux/fuzref

https://www.fuzzcon.eu/

https://github.com/antonio-morales/Fuzzing101

https://fuzzinglabs.com/

Any Question?

Thank you very much

Miguel Morillo

https://www.linkedin.com/in/mmorillo/