SYNOPSYS®
Silicon to Software™

Network and Application Level Protocol Fuzzing -Theory and demo/Hands-on

Vishwas Sharma

Why Fuzzing

Vulnerabilities: Key to Hack a System

Known vulnerabilities are tracked by NIST

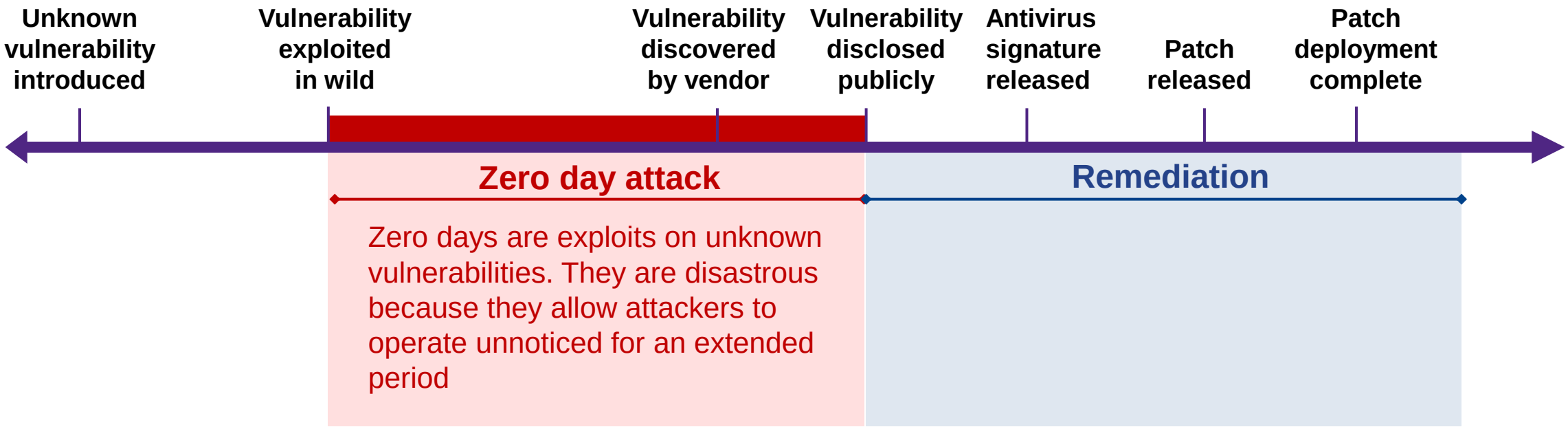
Known vulnerabilities can be found by a vulnerability scanner and then patched

Unknown vulnerabilities

Nobody knows except skilled hackers

and … **Defensics®**

Unknown Vulnerability, Zero Day Attack

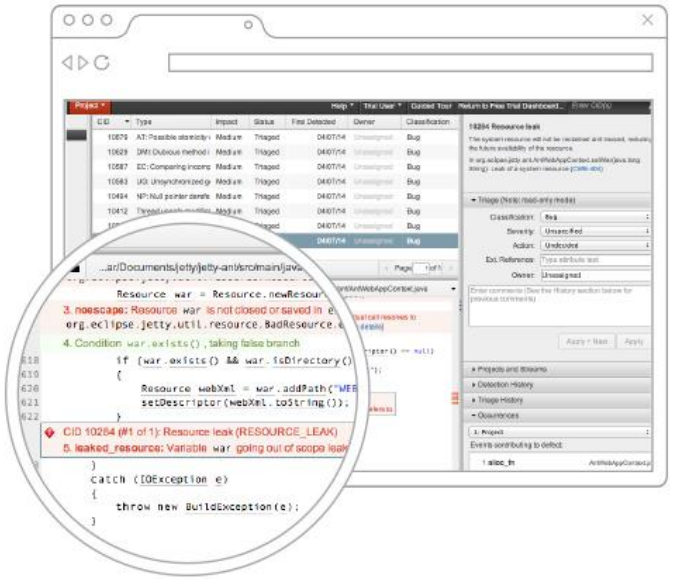


Window of exposure

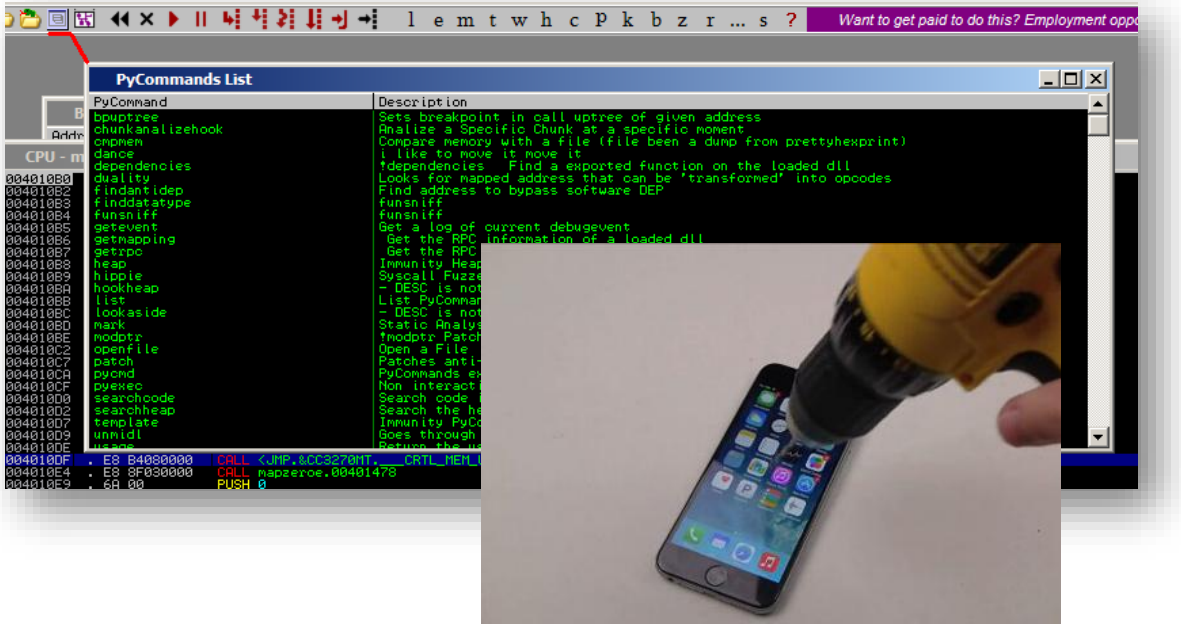
Reactive security (i.e., firewalls, IDS, SIEM, and antivirus) is ineffective against zero day attacks

How Hackers Find Zero Day Vulnerabilities

- **Open source or access to source code**
  - Manual code review
  - Static Code Analysis

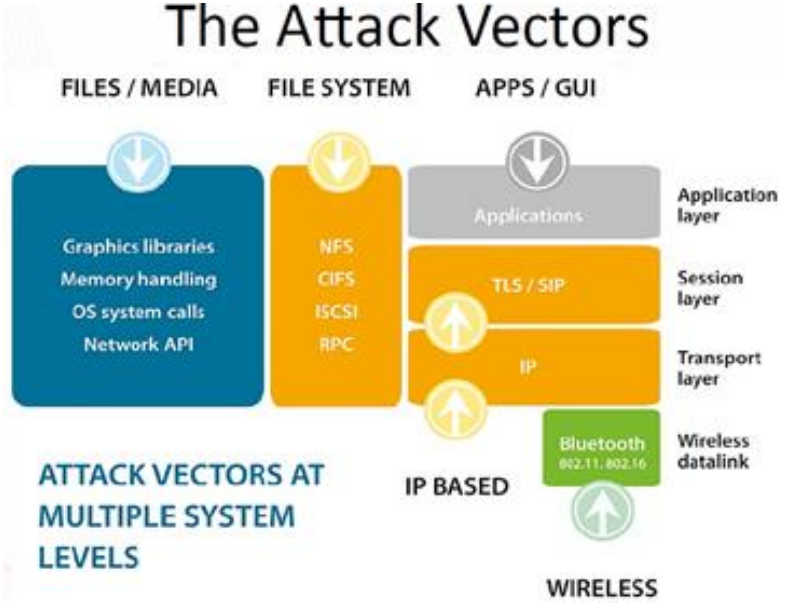


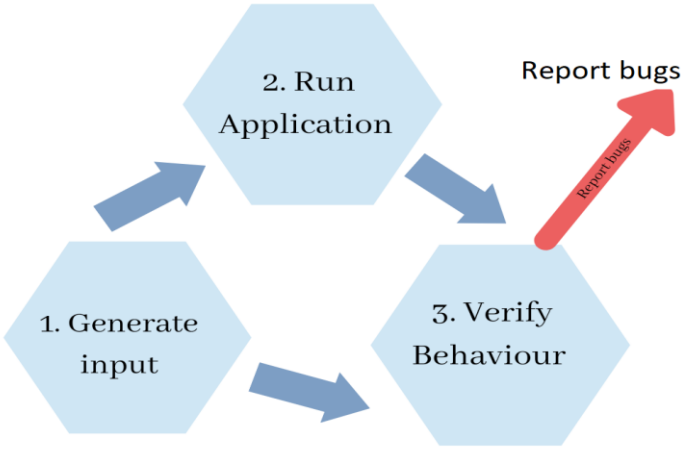
- **No access to source code**
  - Reverse Engineering
  - Binary Static Analysis
  - Fuzzing (malformed input testing)



Why You Should Fuzz

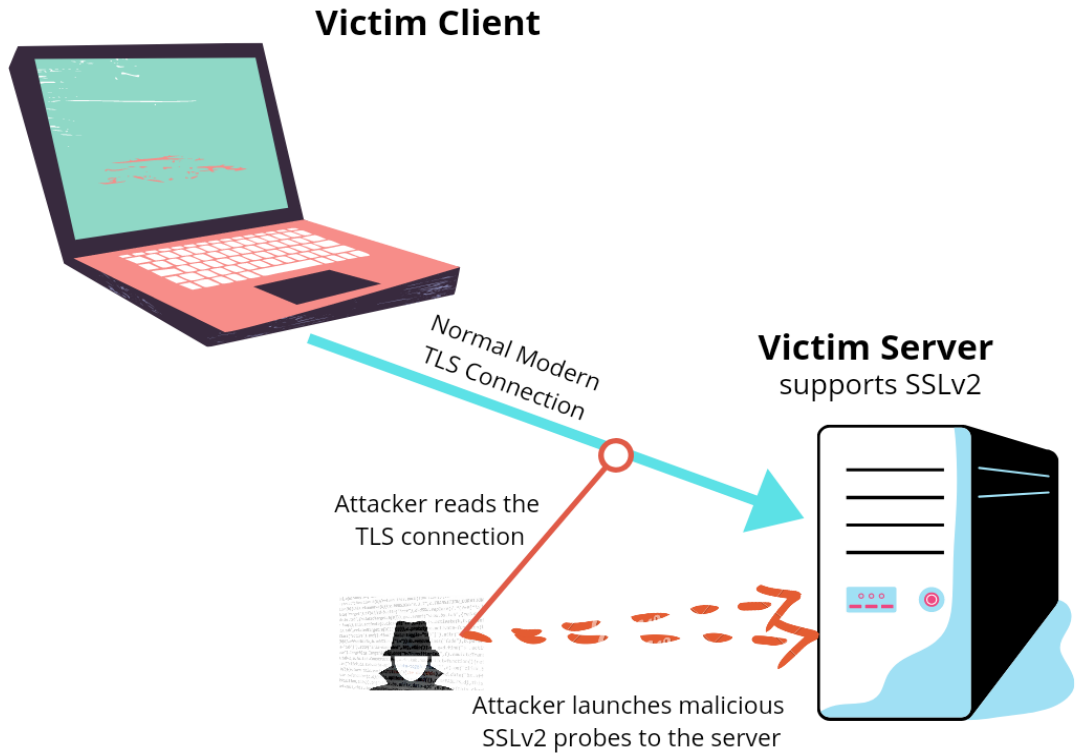
- If you develop software and it takes inputs without validation, the question should be ***how to fuzz,*** not ***whether to fuzz***
- A fuzzer closely emulates the behavior of an external attacker who does not have the access to source code
- Fuzzing relates to abuse/misuse, and misuse has historically been the most common way to trigger vulnerabilities
- ***How good is your software at gracefully handling and rejecting invalid inputs?***





How Fuzzing Relates to Protocol Security

- Attackers have used fuzzing to target remote systems
- Use of malformed inputs allows identification of the weak implementation
- Not all software validate inputs properly



- ❖ **Protocols define the external/internet-facing communication**
- ❖ **A protocol implementation needs to be matured/hardened enough before deployment**

Real world experience with Fuzzing

Event 1 : USB device connected

Event 2: Windows crashed with an irrecoverable exception

Event 3: Windows generated an error log

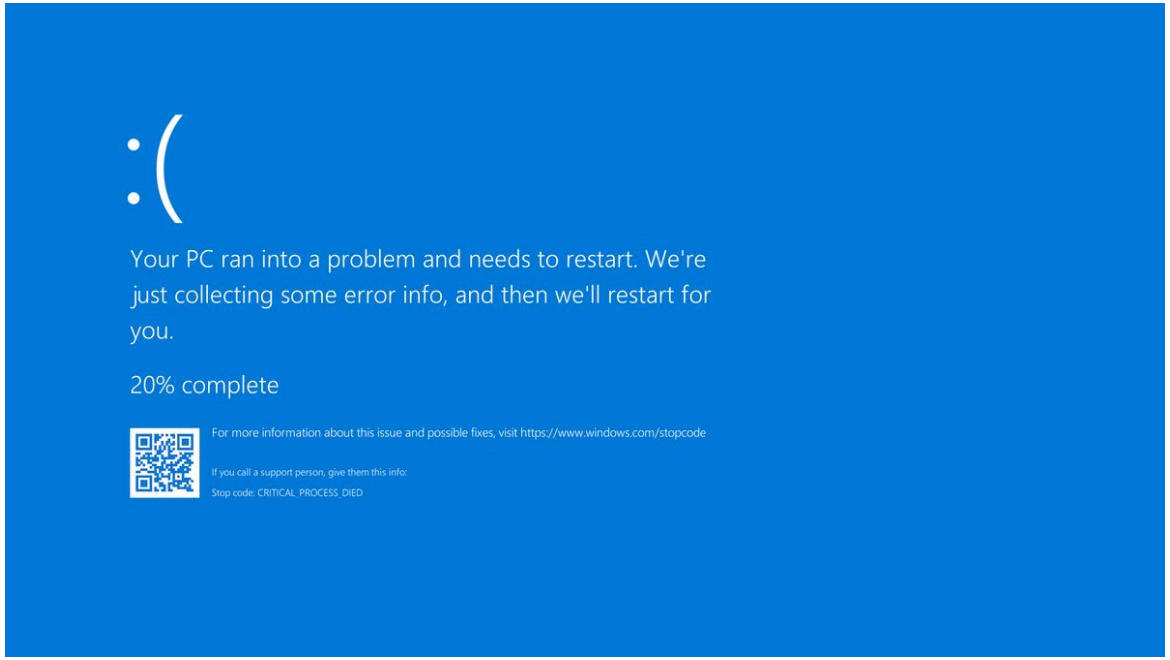
Event 4: Windows uploaded the error log

Event 5: Microsoft analyzed the error log

Event 6: Microsoft created a patch

Event 7: Microsoft updated the patch

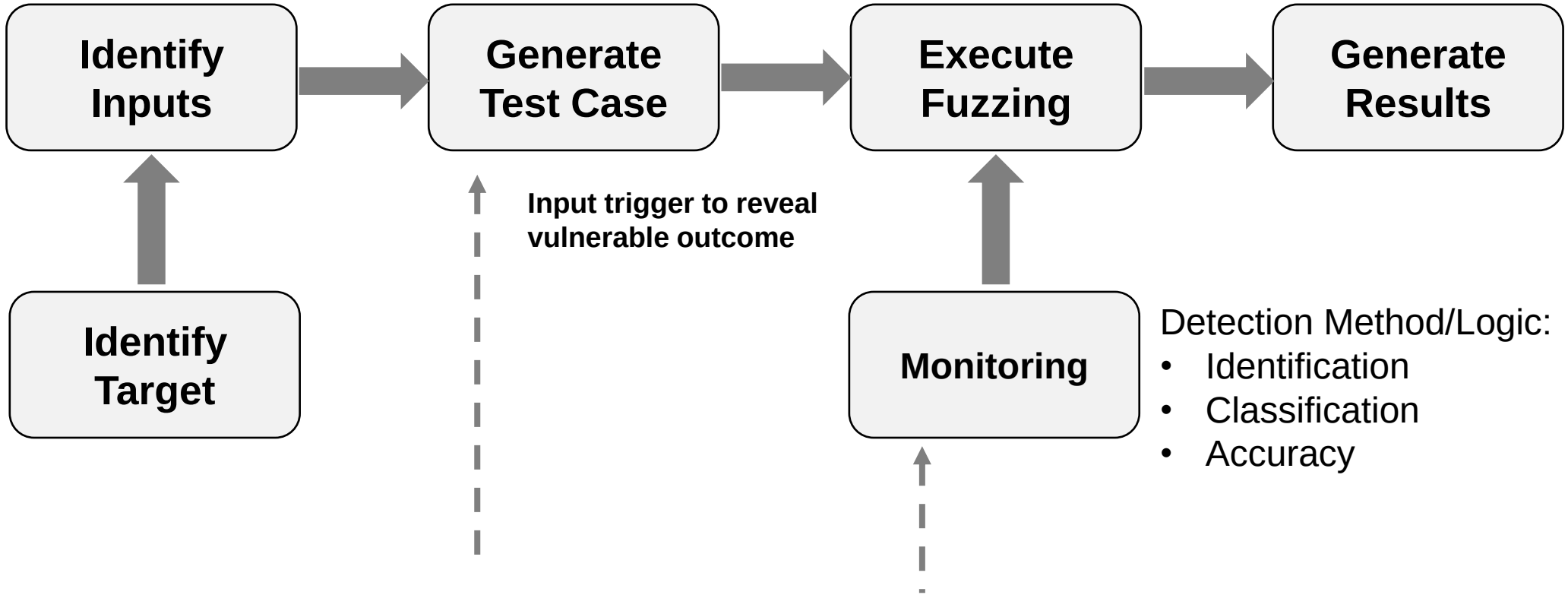
Event 8: Bug was fixed, and exploit (if any) was neutralized



→ **Detection, Identification, and Remediation**

Defensics protocol fuzzer

Fuzzing: A Stepwise Process



How a test case is generated?
- Random
- Template based (mutation method)
- Model based (Generational/Spec based)

Monitoring requires certain instrumentation methods to be in place:
- Built-in methods
- Custom methods

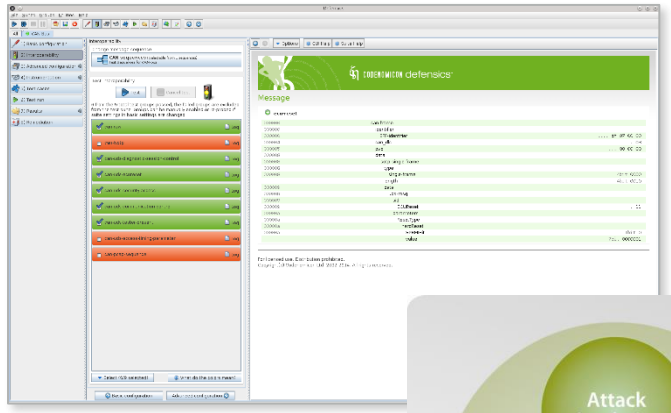
Defensics®

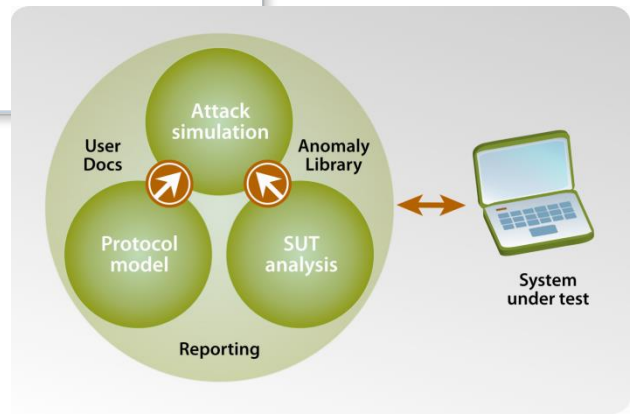
Intelligent Fuzz Testing

Find unknown, exploitable vulnerabilities in your software … before hackers do

An advanced **Fuzz Testing Platform** for discovering and remediating **Unknown Vulnerabilities** in software and devices

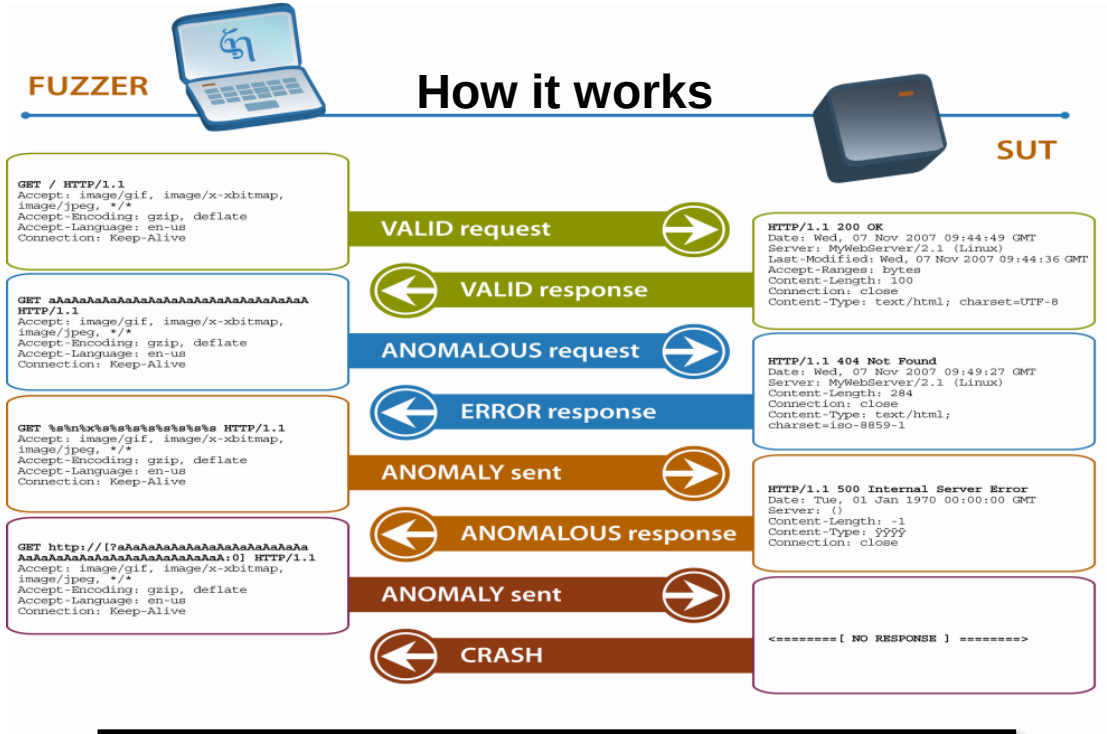
- **Prebuilt test suites** for 300+ standard network protocols, file formats, and other interfaces
- Utilizes **template, generation, and evolutionary-based testing techniques**
- Detailed documentation and workflows that **outline clear paths to remediation**





Defensics: Breadth of Technology

- **Fully State-Aware Testing** for all protocols
  - Maintains complete awareness of protocol states
  - Shrinks the infinite space security problem into a manageable number of misuse test cases
- **Model-Based** test material and complete protocol spec coverage
  - Tests extend to rare and often vulnerable protocol elements
  - Stateful test case injection

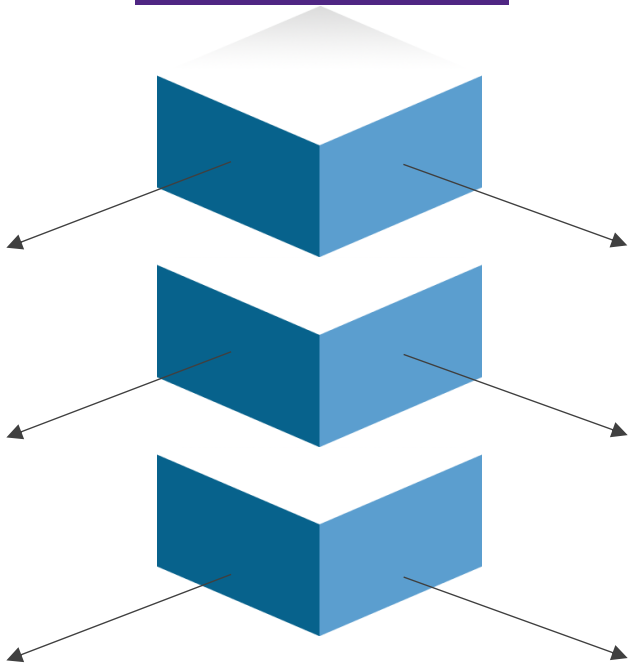


How We Find Vulnerabilities

Input
ANOMALIES

Expose
VULNERABILITIES

DUT/SUT



FIELD LEVEL
- Overflows
- Integer anomalies

STRUCTURAL
- Underflows
- Repetition of elements
- Unexpected elements

SEQUENCE LEVEL
- Out of sequence
- Omitted/unexpected
- Repetition/spamming

NOT RANDOM FUZZING
- Intelligent targeting
- Automatic/systematic
- Ability to sequence
- Repeatable (regression)

WHAT WE FIND

- ✓ Crashes
- ✓ Denial of service (DOS)
- ✓ Security exposures
- ✓ Performance degradation
- ✓ Slow responses
- ✓ Thrashing
- ✓ Anomalous behaviour

Superdemo1.4.zip

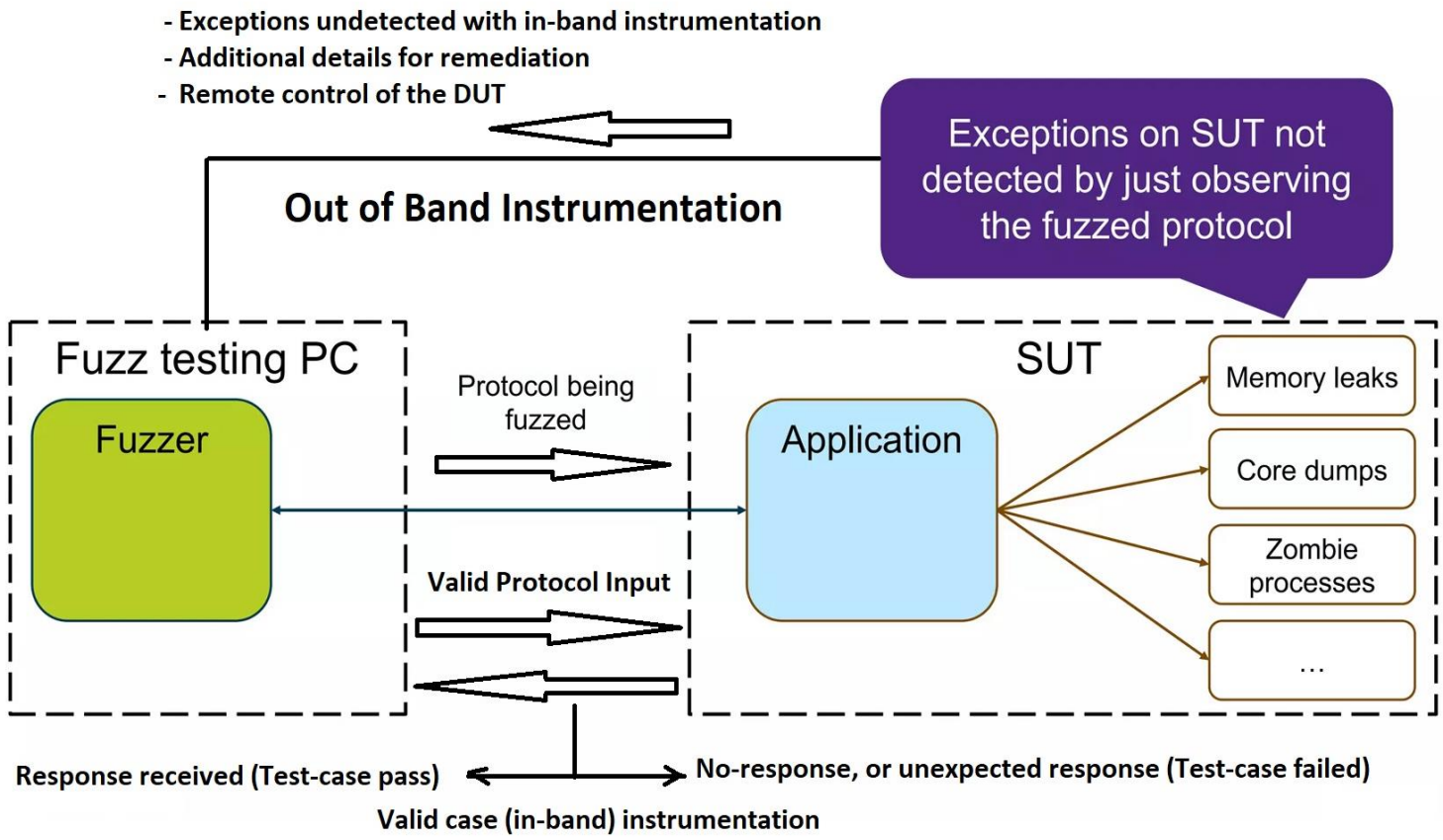
Fuzzing with Instrumentation

Test case : A protocol input designed to trigger exception or error

Instrumentation : Monitoring methods to detect/confirm the exception and collect necessary data for remediation

- ***In-band :*** Use of tested protocol itself for health check and test-case verdict (pass/fail)
- ***Out of band :*** Any method outside of tested protocol (e.g syslog) for additional details to aid remediation of the found issues

*A combination of In-band and out-of-band instrumentation should be used for **detection** and **remediation** of the robustness issues*

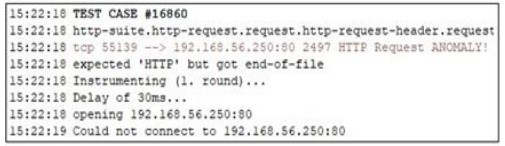


Remediation With Built-In (in-band) and Extendable (out of band) Instrumentations

- Test cases have anomaly, CVSS scoring, and CWE details
- Support for built-in and extendable instrumentation methods that help ascertain the root cause of the failure

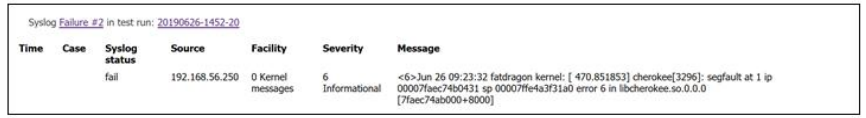
  - **Valid-case instrumentation** is performed after each test case to check that the tested system is still working as expected.
  - **Protocol semantics instrumentation** is protocol-specific semantic checks for SUT messages.
  - **Connection-based instrumentation** is always enabled for the test suites, which use TCP sockets or other connection-oriented transport.
  - **External instrumentation** runs any script or command as a method of instrumentation.
  - **SNMP query instrumentation** checks if certain parameters are outside a specified range.
  - **SNMP trap instrumentation** listens to SNMP trap messages from critical events in the SUT.
  - **Syslog instrumentation** listens to Syslog messages from critical events in the SUT.

Valid Case Instrumentation

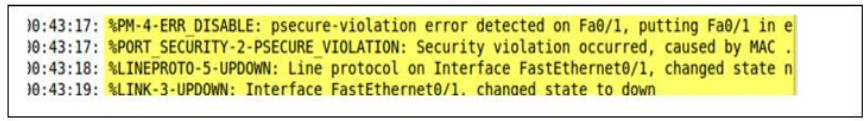


SNMP Instrumentation

SYSLOG Instrumentation



External Instrumentation



Defensics : Key Strengths

Strength	Impact
Test Target Focus	**Defensics** can test an application protocol in either client, server or peer/proxy mode. This helps in one arm testing, two arm testing and proxy/transit mode testing
Generational Fuzzing	**Defensics** protocol test suites are based on popular specification (IETF, ITU, 3GPP, ISO and so on) based generational fuzzing. Defensics test cases are ready to use and do not require any template, or custom development
Anomaly Profiling	Enables user to define which type of anomaly (test case category) to emphasize in testing. This helps a tester to select the test scenarios relevant to their test plan and DUT (protocol) implementation
Remediation package	Remediation package contains protocol level logs, packet traces and other instrumentation checks which are useful in bug reproduction and fixing
Scalable	Solution allows using multiple stateful test instances with proper hardware support
CVSS Scoring	Test reports are based on **CVSS** scores to help in issue prioritization based on criticality
CWE based and proprietary anomalies	Defensics supports both **CWE** based and proprietary anomalies in the test cases

DUT failure scenarios under Fuzz-testing

DUT condition	Observable behaviours/indicators	Required instrumentations
The DUT crashes and is unable to restart	Irrecoverable Non-responsiveness, or Port closed permanently. The tested process is down.	-Valid case -External, or log based
The DUT crashes and then possibly restarts	Recoverable non-responsiveness. The tested process is restarted after crash.	-Valid case -External, or log based
The DUT hangs in a busy loop, causing a permanent Denial of Service situation.	Irrecoverable Non-responsiveness. Process is running but hanged.	-Valid case -External, or log based
The DUT slows down momentarily causing a temporary Denial-of- Service situation.	Slow responsiveness (increased timeout). DUT resumes normal working after certain time.	-Valid case -External, or log based
The DUT fails to provide useful services causing a Denial-of-Service situation (i.e. new network connections are refused)	TCP/UDP connection failure. Process is still running.	-Valid case -External, or log based
DUT does not crash, or restart but develops a critical exception shown in the logs	Log message	External, or Log based

Test-roles for DUT and Defensics

→For testing DUT in server mode, Defensics test-suite acts as client

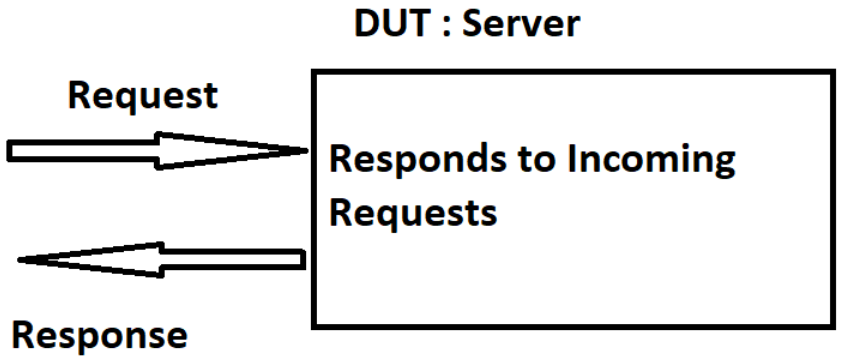
→For testing DUT in client mode, Defensics test-suite acts as server

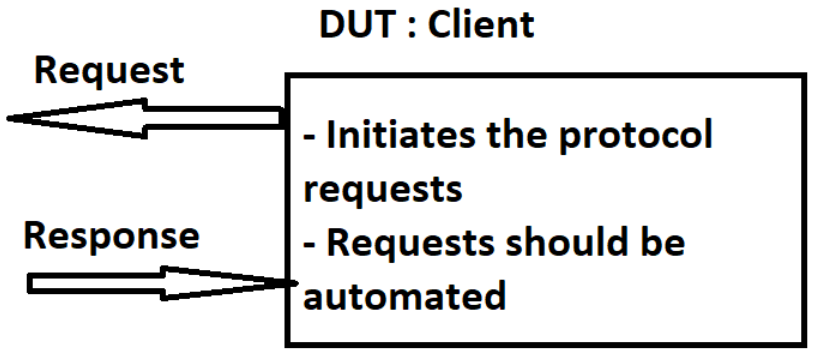
→Testing a DUT in client mode requires automation of the requests

Client – Test-initiator (requester)

Server –Test terminator (responder)

Transit: Both Client and Server

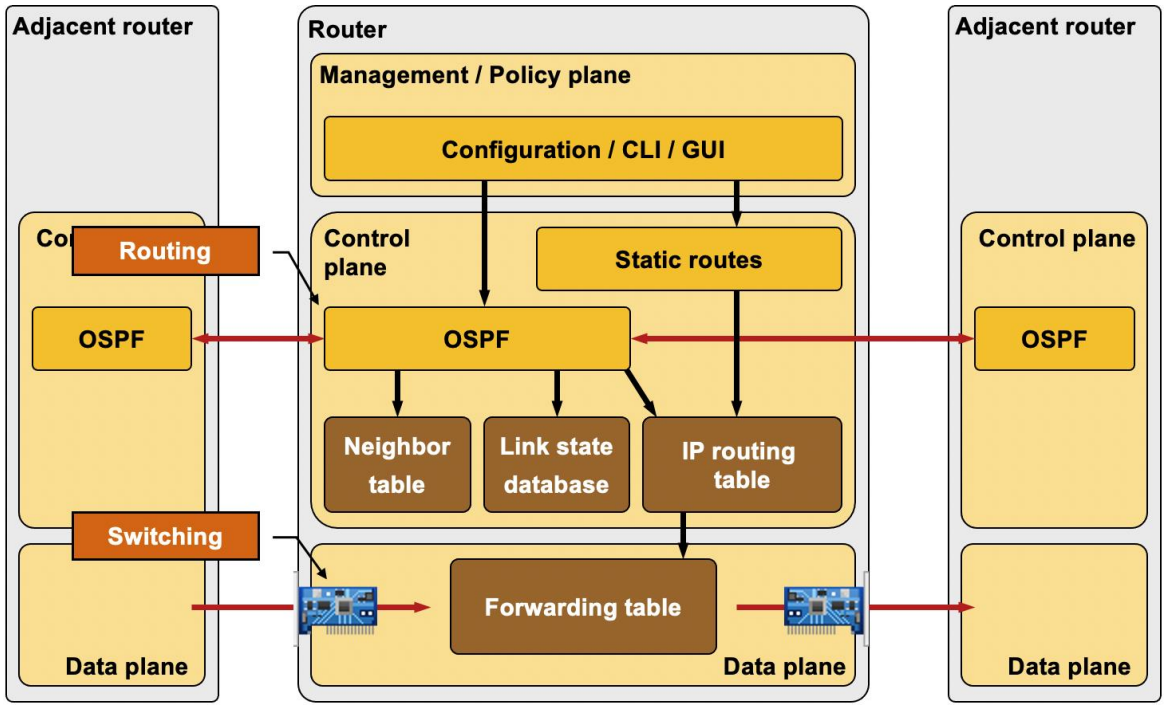




Use cases and Test-scenarios

IP Router - Traffic Planes concept

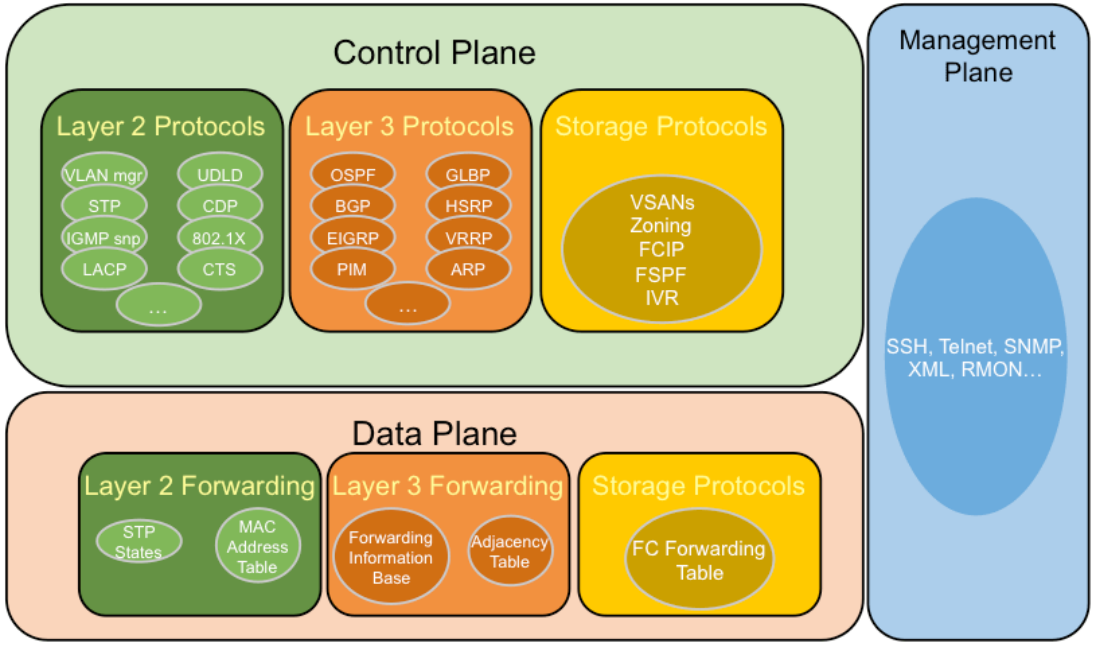
Reference : https://blog.ipspace.net/2013/08/management-control-and-data-planes-in.html



- Router handles the receive and transit traffic using dedicated and logically separated traffic planes
- The attack surface analysis of router should also be done as per the traffic planes concept
- The router can act as source (client), sink (server), or transit for the network traffic
- The test-bed for the router should closely emulate the traffic that is handled by the router
- The router under test should generate logs for errors and exceptions that arise during testing

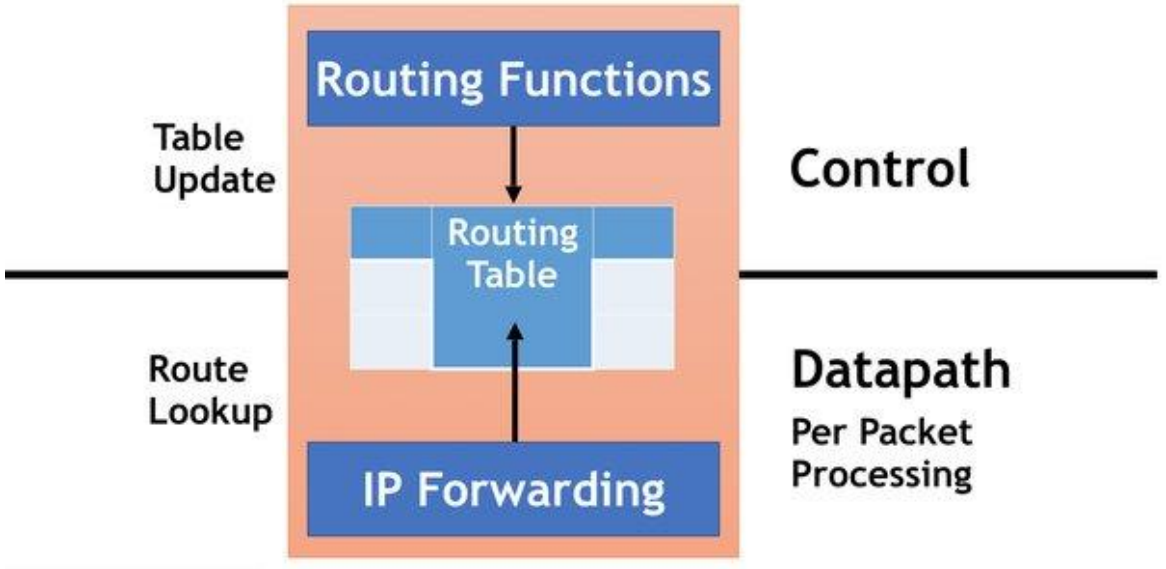
Fuzzing IP router

- **Data Plane:** It involves only pass through traffic. Not the packets that originate, or terminate at the router.
- **Control Plane:** Protocols related to core functionality of the router. These protocols require router to act as requester/responder (E.g routing protocols: BGP, OSPF, MPLS, LDP..)
- **Service/Management Plane:** The protocols used for local administration of the router. These protocols require router to act as requester/responder (HTTP, SSH, SNMP, TFTP…)



How to visualize router fuzzing

- Does fuzzing cause disruption of router's main functionality – the route lookup and forwarding?
- Does the fuzzing cause an error or exception in tested process (BGP, OSPF..)



Protocol stacks architecture in a CPE device

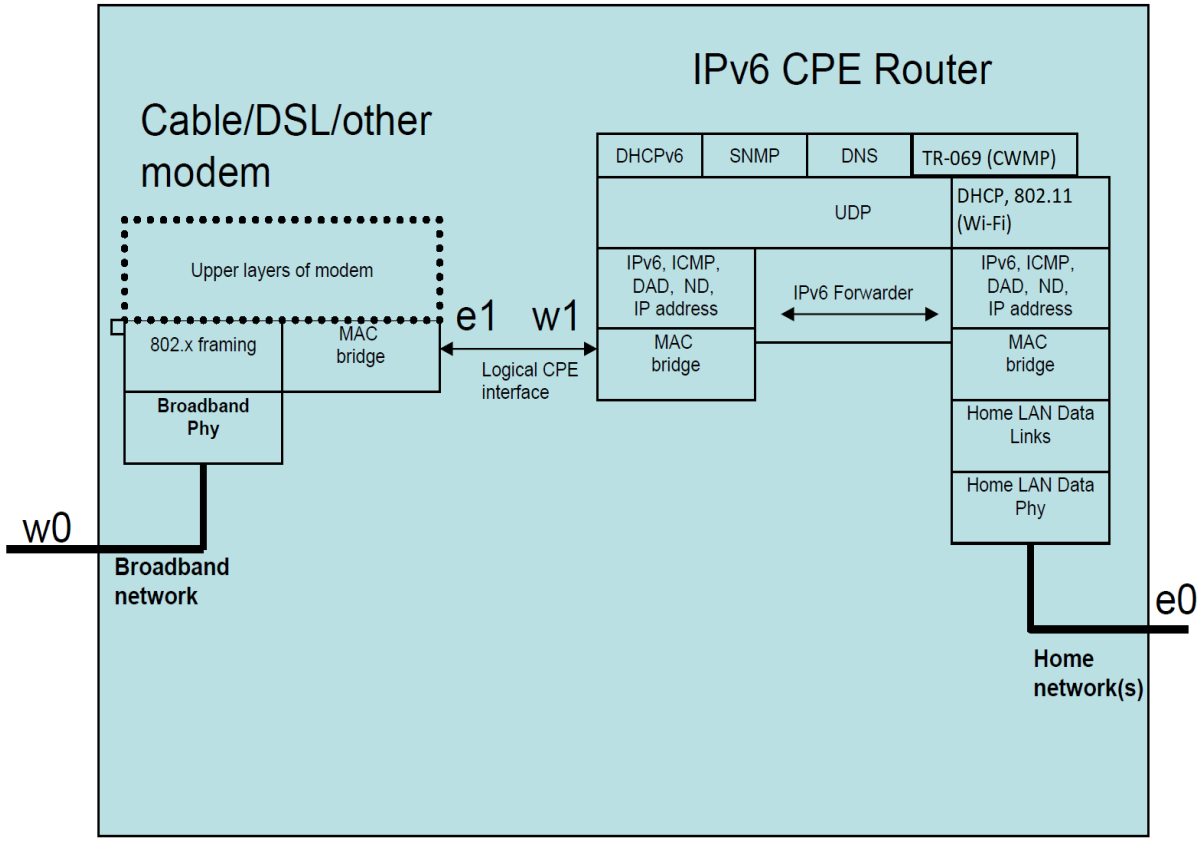
A CPE device has two interfaces

-LAN

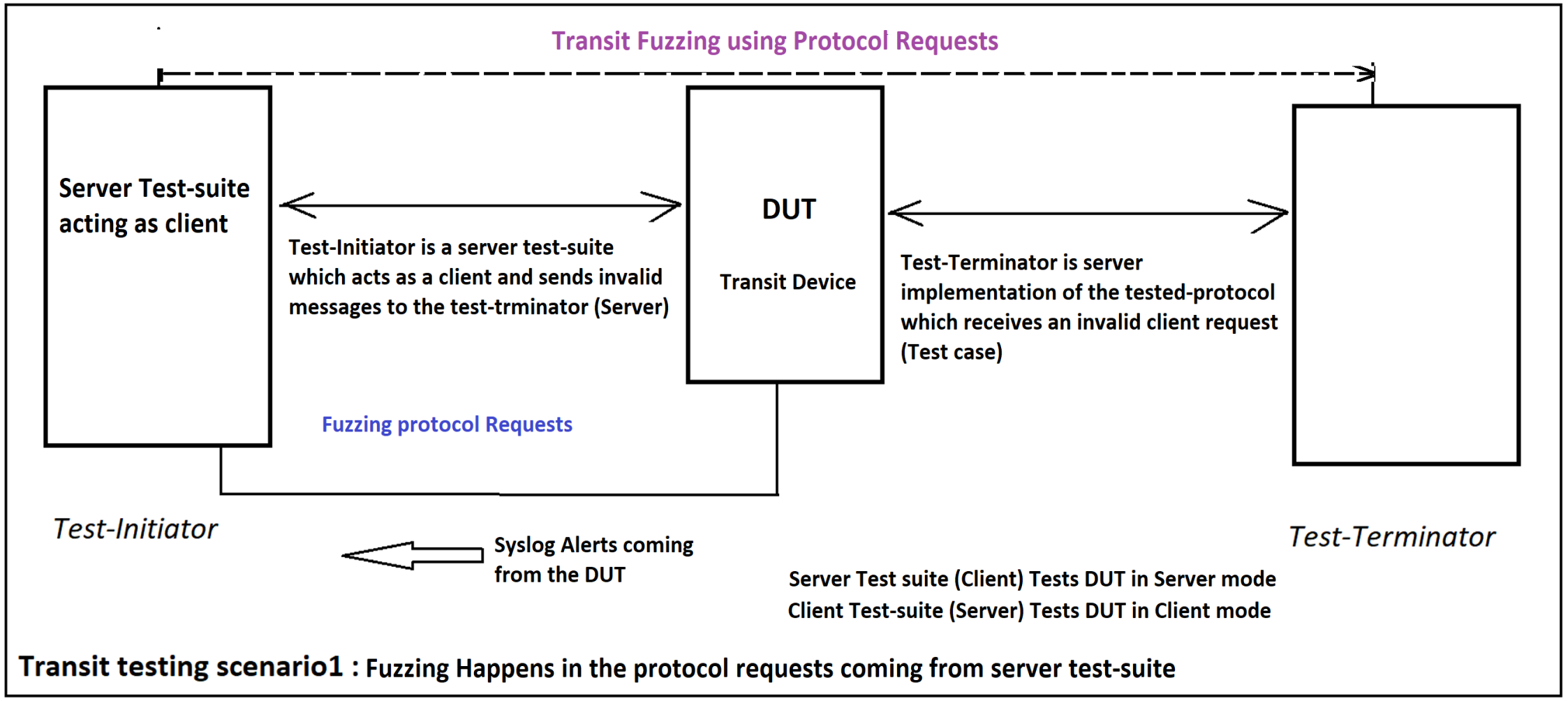
-WAN

→ *The protocols on LAN side operate mostly as server*

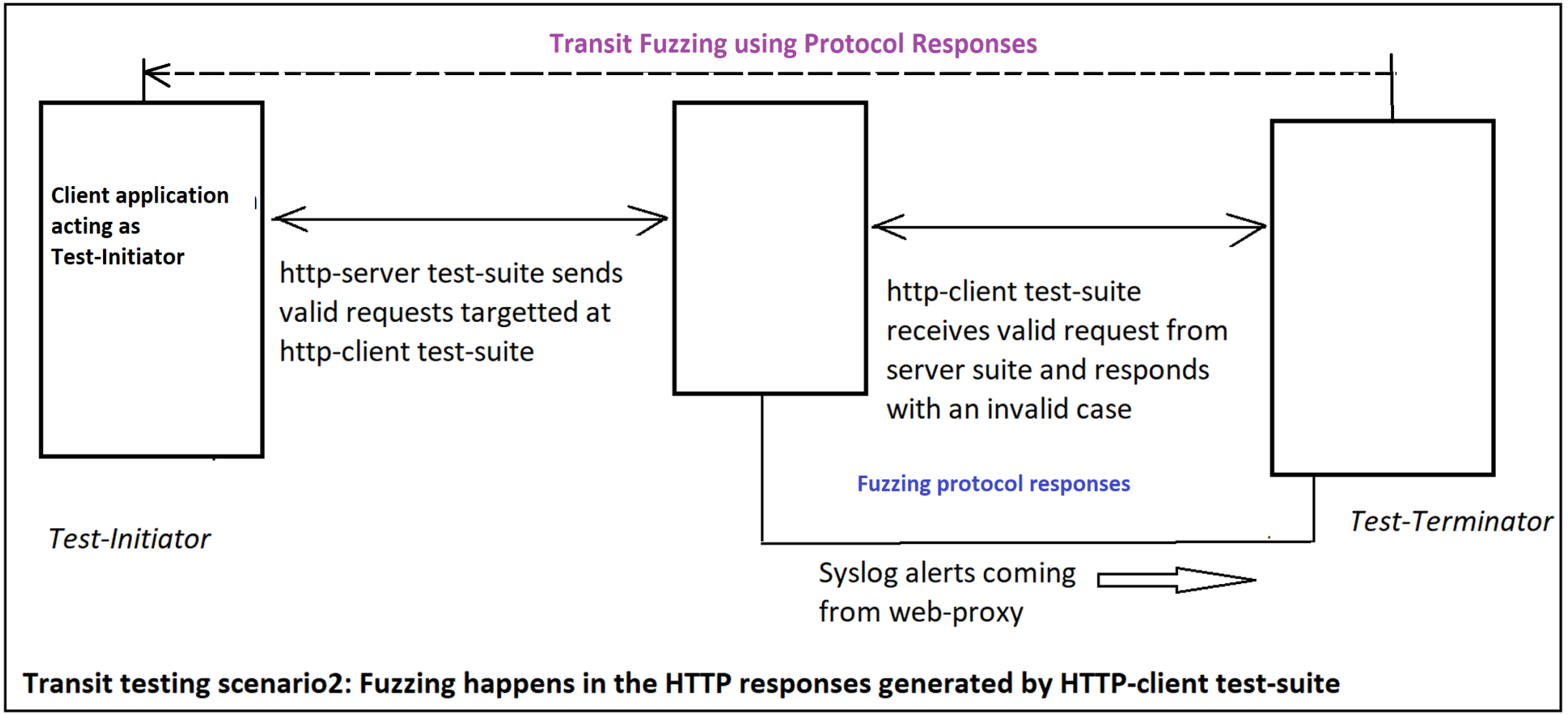
→ *The protocols on WAN side may operate both as client, or server*



Transit mode fuzzing scenario1 : Example - Fuzzing HTTP requests

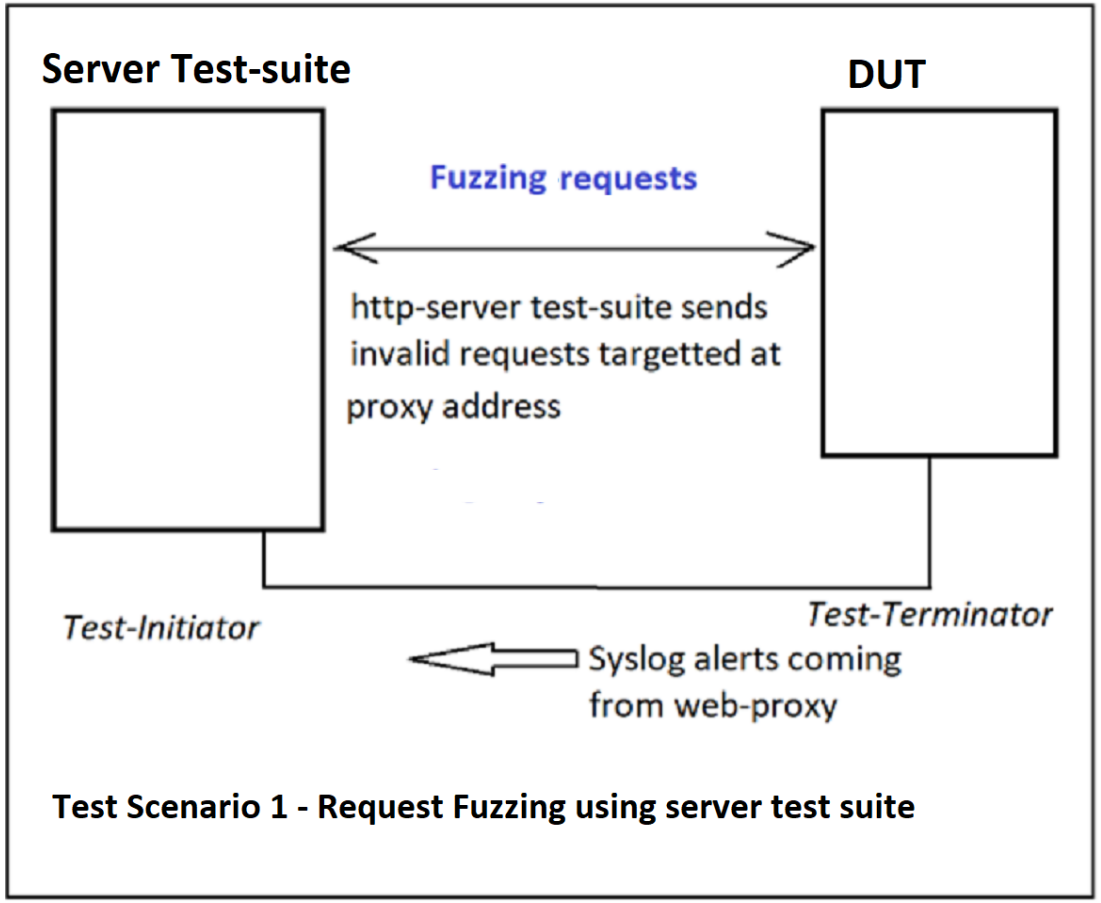


Transit mode fuzzing scenario2: Fuzzing HTTP responses

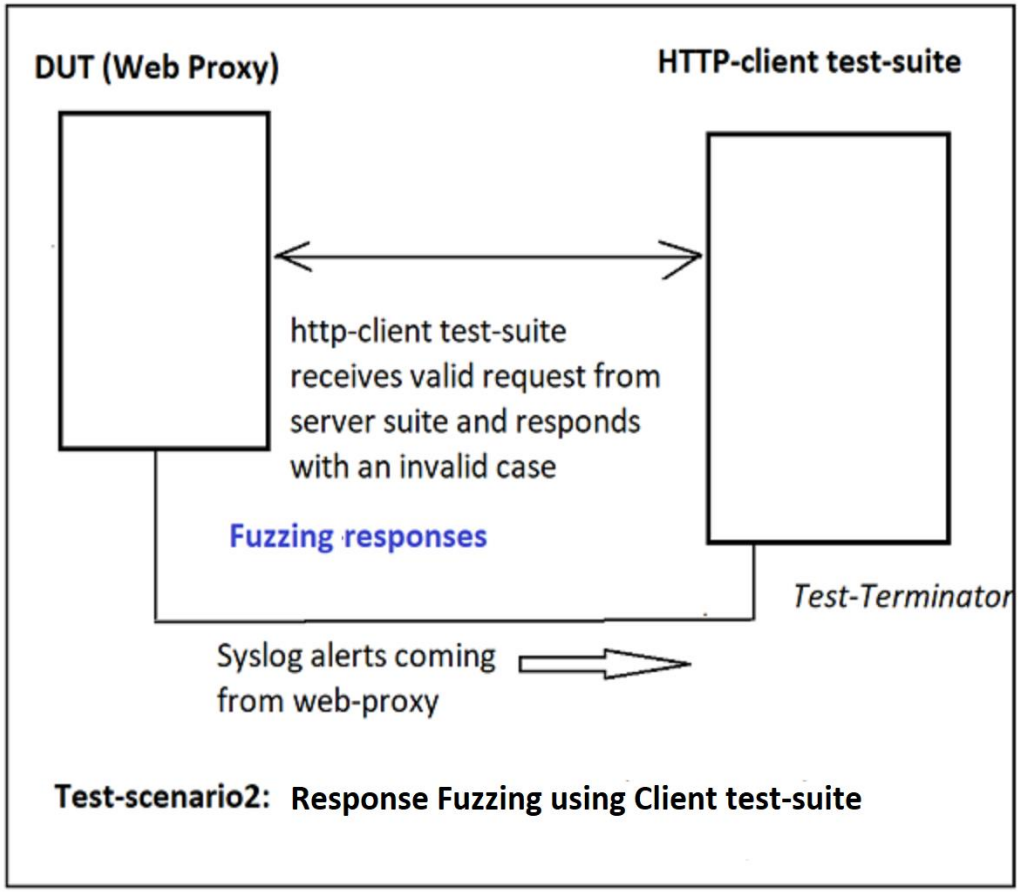


Transit testing scenario2: Fuzzing happens in the HTTP responses generated by HTTP-client test-suite

Receive mode fuzzing scenarios



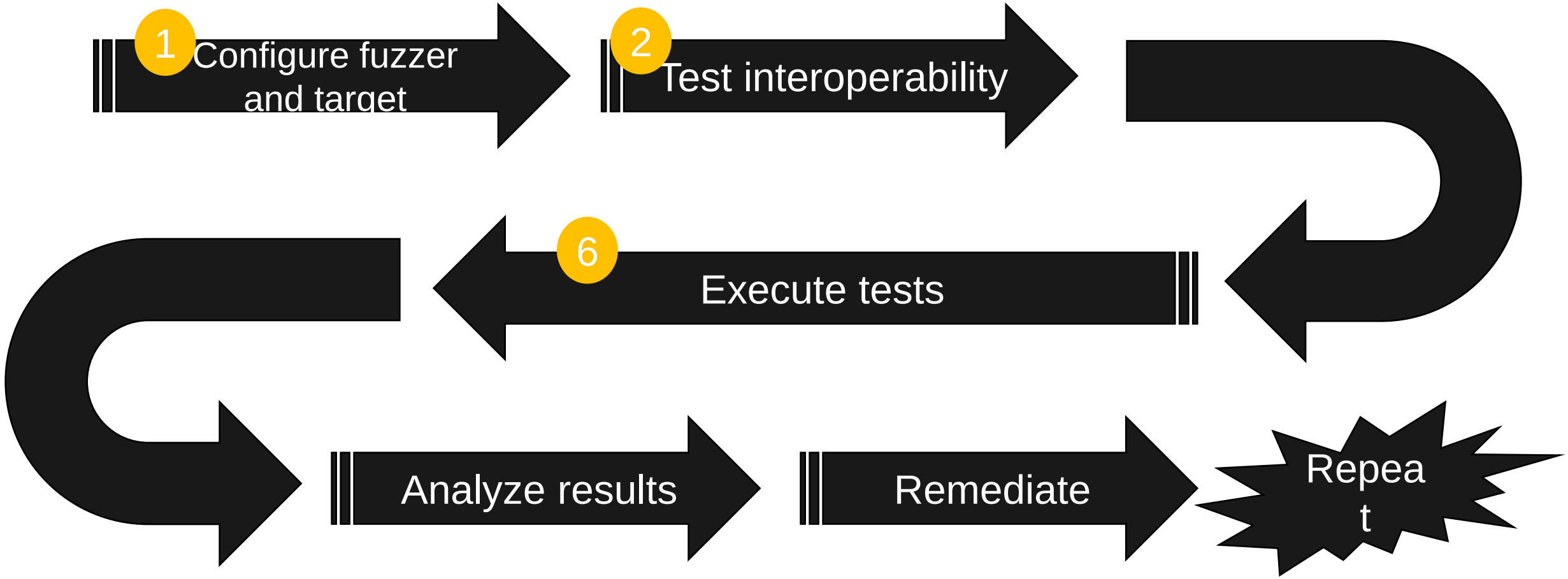
Test Scenario 1 - Request Fuzzing using server test suite



Test-scenario2: Response Fuzzing using Client test-suite

Fuzzing - Workflow

Workflow



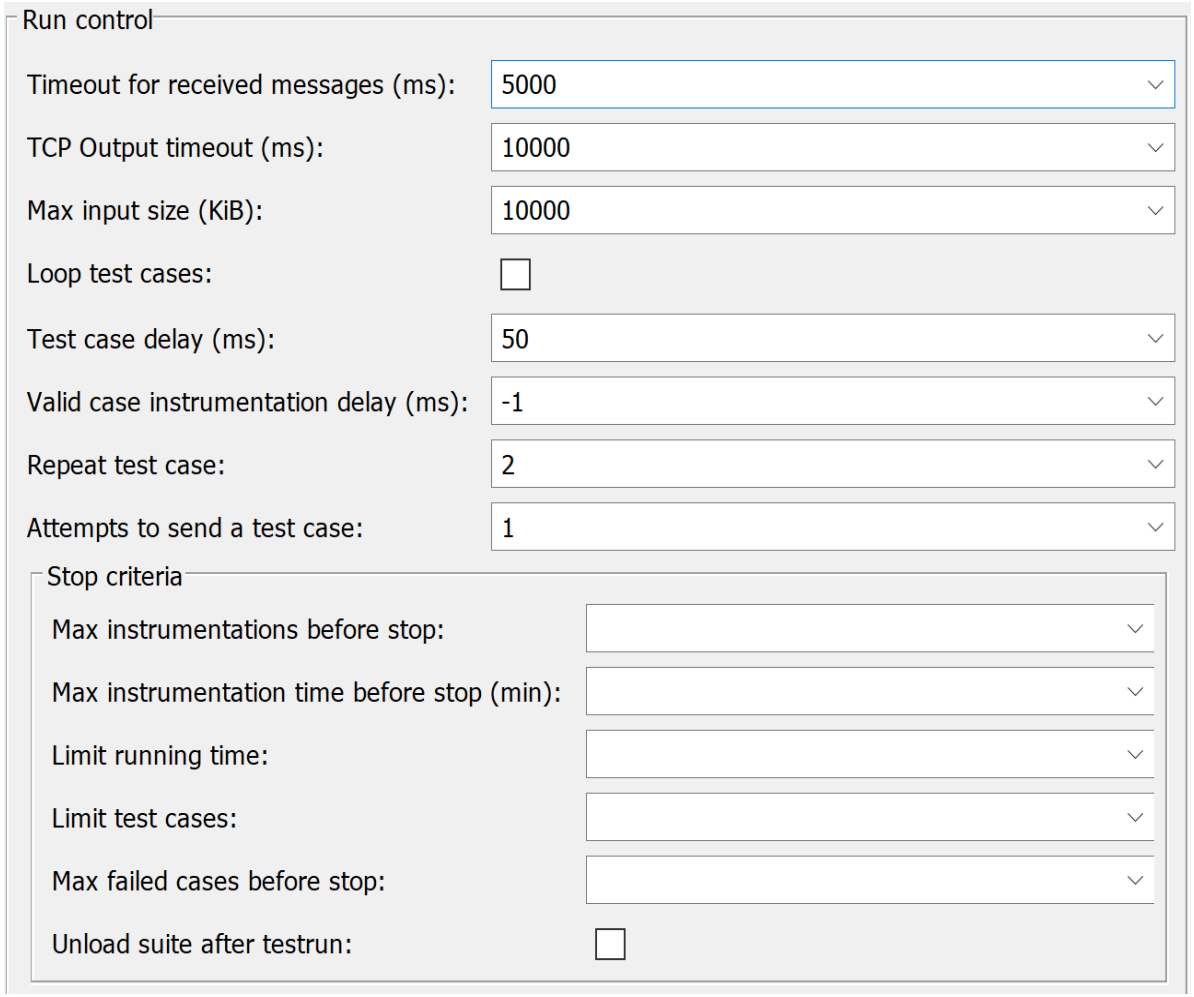
Prepare the DUT and the tool

- Enable the protocols to be tested
- Check if the protocols (ports) show up in port scan
- Get a packet capture of a valid handshake for the tested protocol from the OEM/Operator
- Enable error/exception logging and make it accessible either in the form of a text file, or syslog output
- Get the error codes from the DUT documentation that indicate critical exceptions
- Define acceptable timeout values for the tested protocols

Configure protocol specific settings in Defensics

- Details such as destination hostname, IP address, ports, MAC address
- Authentication details (if applicable)
- Any other protocol specific setting needed for the valid handshake
- It is also advised to look at a packet capture of the tested protocol to know the exact test-sequence and associated settings

Run Control – Settings to control test-execution



Timeout – Acceptable response time before DUT is considered in DOS. This affects pass/fail criteria for the test-cases. It should be a value that is mutually agreeable between tester and the DUT vendor

TCP Timeout - Time that the test driver will wait for a request to be sent to the tested implementation when using TCP transport, provided in milliseconds

Test Interoperability

- Once the matching configuration has been applied, run interop
- If Interop passes, proceed to next step
- If Interop fails, try these:

-Check packet capture to know if there is IP level bidirectional connectivity

-Check if there is transport layer connectivity

-Check protocol response from the DUT to know what is to be changed in the settings

-Check DUT logs to know the required change in the test-settings

Instrumentation settings – Types of instrumentations

Instrumentation is needed to give pass/fail verdict to the test cases and to gather details for issue remediation (reporting the issue to OEM)

1. Valid case instrumentation - Pass/Fail Criteria

- To be chosen from the Interop sequences
- This is used for first level health check after a test-case (invalid input)
- Serves as pass/fail criteria (unless you use external, or agent instrumentation)

2. External instrumentation – Issue Validation, DUT control, Additional details

- Additional checks to validate the reported failures by valid case instrumentations
- Can be used to gather additional details or to control the DUT (Reading logs, port scan and so on)

3. Log based Instrumentation – Remediation details (Conclusive evidence of the failure)

Syslog Instrumentation to be used with critical error codes

Instrumentations settings : Timeout

- Timeout is the time that the test driver will wait for a response from the system under test (SUT), in milliseconds
- SUT does not often respond to anomalous message which is acceptable but should respond to valid message.
- Failure to respond to a valid message within given timeout is marked as failure if fail limit is exceeded
- Timeout is function of processing time by the DUT and the receive/forward delays (if any)

Observed time out : Calculate the difference in the timestamp for sent and received messages for valid cases

→Observed time out during instrumentation may be higher than that during interop test because invalid messages can slow down the DUT (temporarily)

Maximum acceptable time-out : Define the baseline as per your threat model for DOS

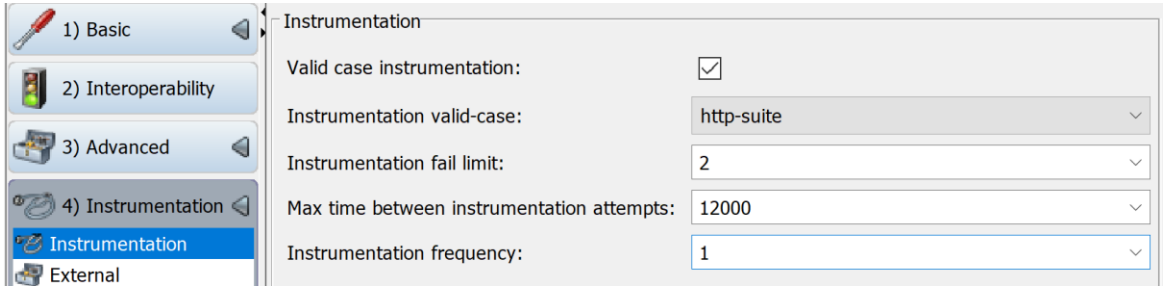
→This the maximum allowed timeout before the DUT is considered to be in DOS

→This allows us to set the threshold for passing/failing the test-cases without reporting a false positive and without ignoring a true positive

→Maximum acceptable timeout to be chosen as the “Timeout” under advanced settings

Instrumentations settings : Fail limit and frequency

Fail limit defines the number of consecutive instrumentations that need to be unsuccessful for a test-case to be regarded as “Failed”



→*This setting works with connection based instrumentation, valid case instrumentation, external instrumentation and synchronous agent instrumentation.*

→*A lower fail limit results in quicker instrumentation but is prone to higher false positives*

→*A higher fail limit provides more than one health check attempts and is thus results in more accurate pass/fail verdict*

→*Typically, a value of **either 2 or 3** is recommended.*

Instrumentation Frequency determines after how many test cases are run before *instrumentation* is executed.

→*Recommended value is 1*

→*Values larger than 1 can be used to speed up testing, since instrumentation is done less often. Regardless of the used value, instrumentation is always run after the first test case.*

Typical timeout values for tested-protocols (Please verify with DUT documentation)

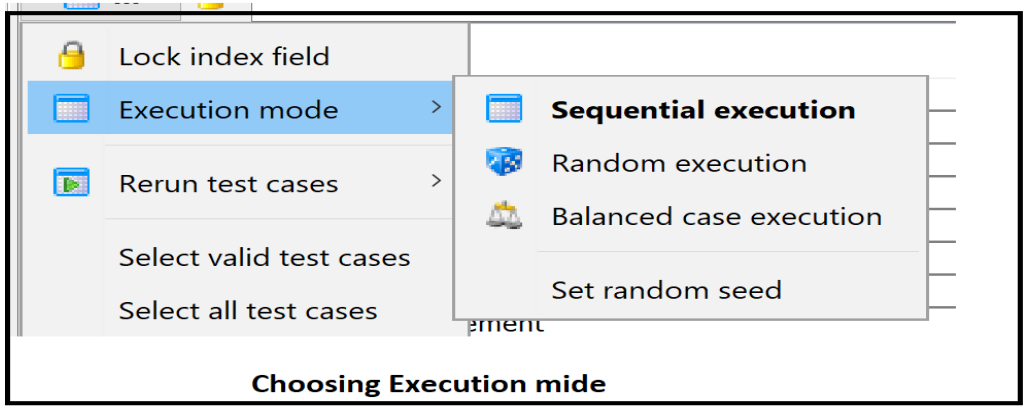
- Time constrained protocols have stricter timeout
- Non real time protocols have relaxed timeout
- Time out depends on how much delay in the protocol response can be allowed before DUT is considered in DOS

→Triggering a vulnerability does not depend on timeout. It depends on the test-case

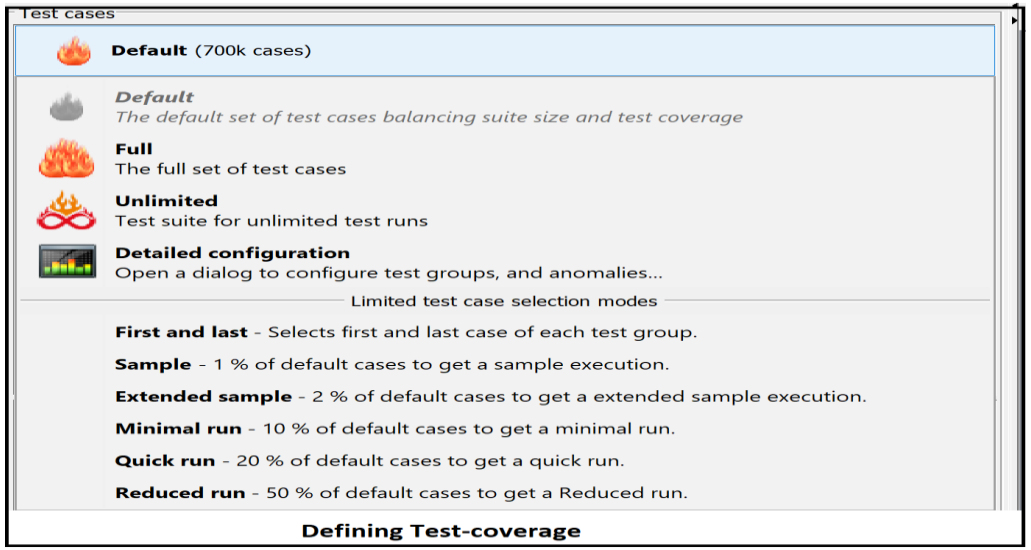
→Timeout only affects pass/fail verdict

Choose the test-cases and execution mode

- Choose a combination of “sequential” and “random” execution mode with at 20 to 50% test coverage.
- Balanced mode to strike balance between the two modes
- The test case anomalies can be changed depending on the type of tests that are considered important as per the given threat model.



Choosing Execution mide



Defining Test-coverage

Start Instrumented test execution

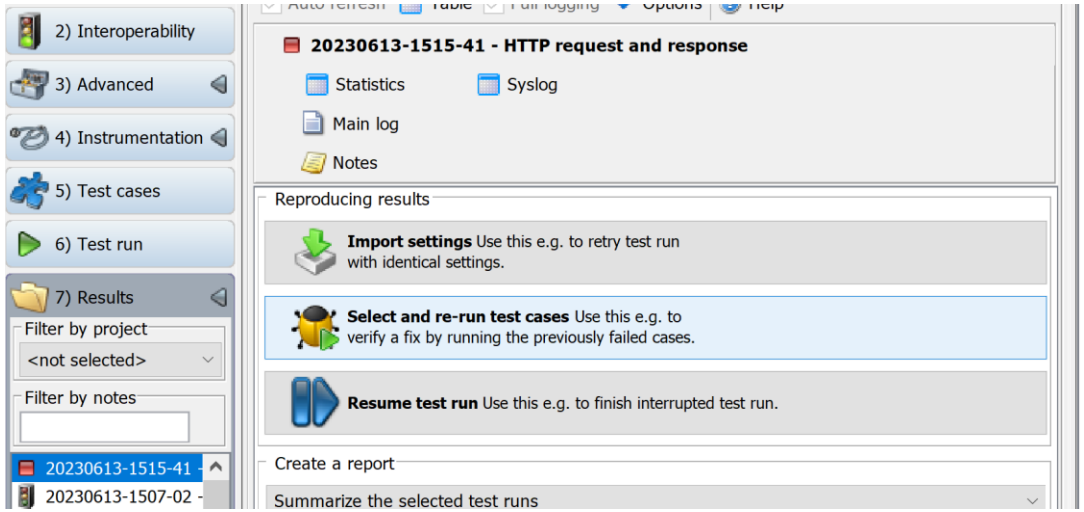
- Start fuzzing with at least two instrumentations
- Watch for instrumentation failures
- Have a watch on the DUT and look for reboot/restart events
- Look for errors/exceptions in the logs

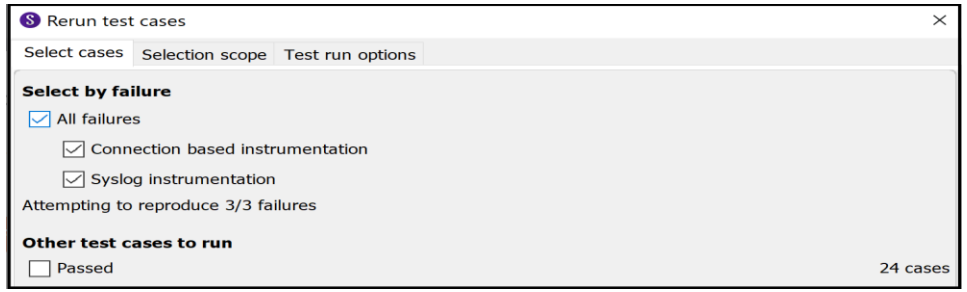
Reproducibility test : To ensure results are accurate and repeatable

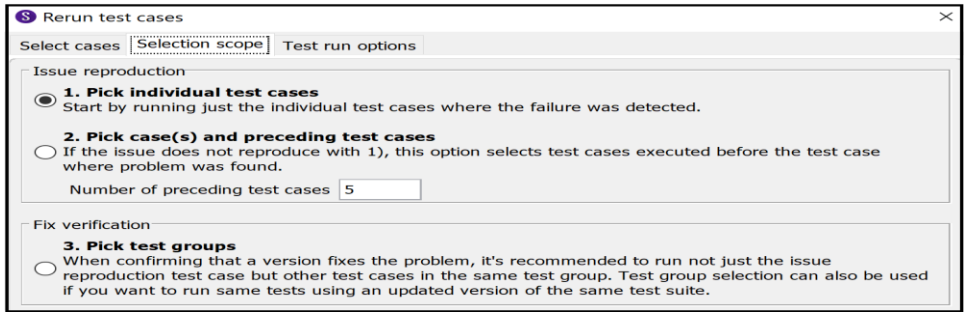
- Choose “Select and re-run” setting
- Relax the time out (to a permissible value) and add delay between test cases

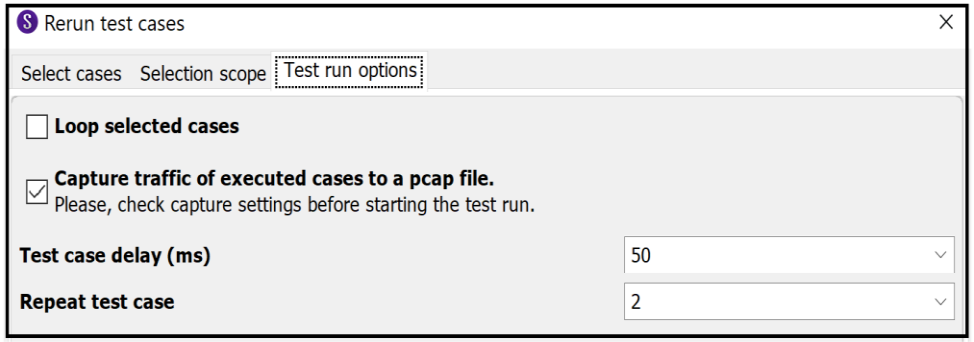
→ *A permissible timeout is maximum allowed response time before DUT is considered in a DOS*

- Add additional instrumentation for validation (Port scan, for example) if there was none during first test-run
- Collect DUT logs and packet captures and attach them with test-report for issue reporting and remediation to be shared with OEM, or DUT vendor









How to confirm and report the failed test-case

Run reproducibility test

-- Is the test-case failing again?

Run External instrumentation

-- Does it confirm the failure due to target crash?

Run Syslog, or Agent instrumentation

-- DUT log should provide process level evidence of failure

Demo