Aegis™ Platform

User Manual - 0.1.0

AEGIS
CONSORTIUM

Aegis™ Platform:

User Manual - 0.1.0

Table of Contents

1. Introduction 1

  What is Fuzzing? 1

  Your mileage may vary 1

    Code coverage 1

    Dynamic analysis 1

2. Installation 2

  Requirements 2

  Distribution 2

3. Using the Console 3

  The Basics 3

  Protocol independent parameters 3

4. DNP3 5

  Known Gaps 5

  Health Checking 5

  DNP3 specific parameters 5

  Test Procedures 6

  Recommended Tests Plans 6

  Example usages 7

Chapter 1. Introduction

What is Fuzzing?

Fuzzing is an automated software testing technique that stresses any software accepting external input by injecting malformed, unexpected, or random data. Fuzzers can test file parsers, network protocols, and any other software that takes inputs.

Aegis™ is a framework for building *smart fuzzers* for ICS/SCADA protocols. It combines aspects of generational and mutational fuzzing to provide excellent code coverage of the target software.

Your mileage may vary

Fuzzing cannot prove that your software is free of all defects. Most software has a virtually infinite set of inputs, and fuzzing can only prove that certain defects in an infinite input space don't exist. As a software engineer, it is recommended that you apply the same consideration to fuzzing that you do to other types of testing.

Code coverage

Code coverage describes what lines of your source code are executed when a program runs. This technique is frequently used to identify gaps in unit or functional testing coverage. It is also a very important metric for fuzzing. If your fuzzer isn't running a line of code, how can it possibly find a bug on that line? Feedback using the source code is important and we need the help of our users and members to improve the tools. Some code coverage frameworks for popular languages are listed below.

- C/C++ - GCOV [http://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html#Gcov-Intro]
- .NET - opencover [https://www.nuget.org/packages/OpenCover]
- Java - cobertura [http://cobertura.github.io/cobertura/] or emma [http://emma.sourceforge.net/]

Dynamic analysis

Dynamic analysis refers to analyzing the runtime properties of a piece of software. How much CPU is it using? Are resources being leaked? These runtimes proper can help you identify more subtle failure modes than a simple crash. The most effective tools fully virtualize your software, linking hooks between all OS calls and memory allocations.

- C/C++ - Valgrind [http://valgrind.org/]

Chapter 2. Installation

Requirements

The first release of Aegis is written in Scala (www.scala-lang.org). It requires the Java Runtime 7 or later to execute. Aegis has been verified to work on Windows, Linux, and OSX. The "aegis-console" script may require that your JRE installer define the JAVA_HOME environment variable.

Distribution

Aegis Platform is distributed as a platform-neutral ZIP archive. It consists of two directories:

- */bin* - .bat/.sh scripts for launching the tool
- */repo* - java-based dependencies

Copy the distribution to a directory of your choosing. Add the 'bin' subdirectory to your system's PATH.

Chapter 3. Using the Console

The Basics

The first release of Aegis provides a single-run console application. Aegis will always have a console application to simplify scripting and integration with testing servers. Future releases to Aegis members may include UI components with additional target monitoring capability. Run the console by executing the 'aegis-console' script.

```
$ aegis-console
   _____                .__
  /  _  \   ____   ____ |__| ______
 /  /_\  \_/ __ \ / ___\|  |/  ___/
/    |    \  ___// /_/  >  |\___ \
\____|__  /\___  >___  /|__/____  >
        \/     \/_____/         \/

Aegis Platform - CONFIDENTIAL - Automatak, LLC

Required argument not found: mid (Module id of protocol)

usage: aegis-console [flags ... ]

Valid module ids: [dnp3]

Parameters follow.....
```

Protocol independent parameters

The first set of parameters displayed when running the console are independent of the protocol module.

- **Module id (-mid)**

  The module id is a unique identifier that specifies which protocol plugin to run. Valid module id's are displayed on program startup.

- **Procedure id (-pid)**

  The procedure id is an identifier unique to a module that specifies which set of tests to run. Refer to the specific module for a list of procedure ids.

- **Host (-host)**

  The IP address or domain name of the target. The is used only when acting as a TCP/IP. Defaults to localhost (127.0.0.1).

- **Port (-port)**

  The port to use for TCP/IP clients (initiating) or servers (listening). Defaults to 20000. Default will be protocol dependent in future Aegis releases.

- **Listen (-listen)**

  Listen for a connection instead of initiating one. Uses the specified or default port.

- **Start (-start)**

Start at the specified test case (integer id). Aegis will run through the specified random seed to guarantee you get the *exact* same output as if you ran the fuzzer from the first test case.

- **Count (-count)**

  Run the specified number of test cases only.

- **Fill (-fill)**

  When a test case needs a random byte, the framework supplies a default of 0xFF. Override this default value here. This setting is ignored if the 'seed' parameter is specified for pseudo-random filling.

- **Seed (-seed)**

  Use a pseud-random number generator with a specified seed to fill values. Not all test cases use the random number source or fill value. Refer to the specific procedure id to see if it uses these values.

Chapter 4. DNP3

Known Gaps

The DNP3 fuzzer provides fairly exhaustive coverage of the DNP3 link, transport, and application layers. Specialized test cases are provided for each layer and some targeted test cases are provided for known failure points within layers.

There are some known gaps in the current version. Notably, the the following object groups are not tested:

- Group 0 - Device Attributes
- Some object groups above 60, including:
  - File transfer / free-form qualifier code 0x5B
  - Datasets
  - Octet Strings and virtual terminal objects
  - Secure authentication objects

Health Checking

All DNP3 tests use a feature of the link layer to identify if a target has failed. After every attack frame, the fuzzer sends a REQUEST_LINK_STATES message to the target. It then waits for the specified timeout expecting a LINK_STATUS reply. If no reply is received, the fuzzer will retry the request if there are timeouts remaining. If no timeouts remaining, fuzzing is aborted.

The fuzzer can perform some handshaking if it receives a message from the target other than a LINK_STATUS response.

- UNCONFIRMED_USER_DATA - Parse the APDU header and respond with a NULL application message and matching sequence number.
- CONFIRMED_USER_DATA - ACK the frame, Parse the APDU header and respond with a NULL application message and matching sequence number.
- RESET_LINK_STATES - ACK the reset link request
- REQUEST_LINK_STATES - Send the request LINK_STATUS reply

DNP3 specific parameters

- **Destination address (-dest)**

  The link layer destination address. This is always the link layer address of the target you are fuzzing.

- **Source address (-src)**

  The link layer source address. This is address of the fuzzer itself, i.e. who you are pretending to be.

- **Fuzz master (-master)**

  This setting configures the link layer 'master' bit for fuzzing masters. This is required for a master to process link layer frames sent from an outstation. By default, this setting is configured for fuzzing outstations.

- **Link Status Retries (-retries)**

  The number of failed attempts to 'ping' the outstation with a REQUEST_LINK_STATES request before the target is considered failed. This setting defaults to 3, but you may need to increase this number for some implementations.

- **Link Timeout (-linktimeout)**

  The timeout (in milliseconds) for reading a link layer frame from the target. The default of 1000 is usually more than sufficient for a lab setup.

- **App Timeout (DEPRECATED) (-apptimeout)**

  This setting is no longer used and will be removed in a future release.

Test Procedures

- **Link layer (lfuzz)**

  Tests the link layer of an outstation or master using all types of link function codes. This procedure uses the random seed/fill.

- **Transport function (tfuzz)**

  Tests the transport layer of an outstation or master by sending unconfirmed user data packets of varying length and sequence numbers. This procedure uses the random seed/fill.

- **Application layer headers (ahfuzz)**

  Stresses the application layer header parser of an outstation or master by sending malformed messages or messages without function codes that should include object headers but do not. This procedure does not use the seed/fill.

- **Application objects headers and functions (aofuzz)**

  Sends many combinations of function codes, objects, headers, and malformed contents. This tests is most likely to cause issues with an outstation. Masters are unlikely to be affected by this test as they should ignore the vast majority of the function codes. This procedure uses the random seed/fill.

- **Unsolicited object and header fuzzing (aufuzz)**

  Sends many combinations of objects, headers, and malformed contents using the unsolicited (0x82) function code. This test is for masters only, as outstations will (hopefully) just ignore unsolicited responses entirely. This procedure uses the random seed/fill.

Recommended Tests Plans

Recommended test procedures for outstations and masters differ slightly. Don't forget the **-listen** and **-master** flags for master fuzzing!

- **Outstations and Master**

  - lfuzz - run with default 0xFF fill and at least 2 random seeds

  - tfuzz - run with default 0xFF fill and at least 2 random seeds

  - ahfuzz - run with default 0xFF, no random seeds required

- **Outstations only**

  - aofuzz - run with default 0xFF fill and at least 2 random seeds
- **Masters only**
  - aufuzz - run with default 0xFF fill and at least 2 random seeds

Example usages

Run 10 link layer test cases starting at #123

```
$ aegis-console -mid dnp3 -pid lfuzz -start 123 -count 10
```

Unsolicited response fuzzing of a master listening on default port 20000 with master address of 0 and an outstation address of 1

```
$ aegis-console -mid dnp3 -pid aufuzz -dest 0 -src 1 -master -listen
```

Outstation link layer fuzzing test case #100 only

```
$ aegis-console -mid dnp3 -pid lfuzz -start 100 -count 1
```

Outstation link layer fuzzing against 192.168.1.55:20001 with default addressing

```
$ aegis-console -mid dnp3 -id lfuzz -host 192.168.1.55 -port 20001
```