# boofuzz Documentation

*Release 0.0.7*

**Joshua Pereyda**

**Nov 23, 2017**

# Contents

Boofuzz is a fork of and the successor to the venerable Sulley fuzzing framework. Besides numerous bug fixes, boofuzz aims for extensibility. The goal: fuzz everything.

CHAPTER 1

# Why?

Sulley has been the preeminent open source fuzzer for some time, but has fallen out of maintenance.

CHAPTER 2

# Features

Like Sulley, boofuzz incorporates all the critical elements of a fuzzer:

- Easy and quick data generation.
- Instrumentation – AKA failure detection.
- Target reset after failure.
- Recording of test data.

Unlike Sulley, boofuzz also features:

- Much easier install experience!
- Support for arbitrary communications mediums.
- Built-in support for serial fuzzing, ethernet- and IP-layer, UDP broadcast.
- Better recording of test data – consistent, thorough, clear.
- Test result CSV export.
- *Extensible* instrumentation/failure detection.
- Far fewer bugs.

Sulley is affectionately named after the giant teal and purple creature from Monsters Inc. due to his fuzziness. Boofuzz is likewise named after the only creature known to have scared Sulley himself: Boo!

Fig. 2.1: Boo from Monsters Inc

CHAPTER 3

# Installation

```
pip install boofuzz
```

Boofuzz installs as a Python library used to build fuzzer scripts. See *Installing boofuzz* for advanced and detailed instructions.

CHAPTER 4

# User Guide

## 4.1 Installing boofuzz

### 4.1.1 Prerequisites

Boofuzz requires Python. Recommended installation requires `pip`.

Ubuntu: `sudo apt-get install python-pip`

Windows: See this help site but make sure to get Python 2.x instead of 3.x (pip is included).

### 4.1.2 Install

```
pip install boofuzz
```

### 4.1.3 From Source

1. Download source code: https://github.com/jtpereyda/boofuzz
2. Install. Run `pip` from within the boofuzz directory:
   - Ubuntu: `sudo pip install .`
   - Windows: `pip install .`

Tips:

- Use the `-e` option for developer mode, which allows changes to be seen automatically without reinstalling:

  ```
  `sudo pip install -e .`
  ```

- To install developer tools (unit test dependencies, test runners, etc.) as well:

```
`sudo pip install -e .[dev]`
```

- If you're behind a proxy:

```
`set HTTPS_PROXY=http://your.proxy.com:port`
```

  - On Linux, also use `sudo`'s `-E` option:

    ```
    sudo -E pip install -e .
    ```

### 4.1.4 Extras

#### process_monitor.py (Windows only)

The process monitor is a tool for detecting crashes and restarting an application on Windows (process_monitor_unix.py is provided for Unix).

The process monitor is included with boofuzz, but requires additional libraries to run. While boofuzz typically runs on a different machine than the target, the process monitor must run on the target machine itself.

If you want to use process_monitor.py, follow these additional steps:

1. Download and install pydbg.
   (a) Make sure to install and run pydbg using a 32-bit Python interpreter, not 64-bit!
   (b) The OpenRCE repository doesn't have a setup.py. Use Fitblip's fork.
   (c) `C:\Users\IEUser\Downloads\pydbg-master>pip install ./pydbg-master`
2. Download and install pydasm.
   (a) `C:\Users\IEUser\Downloads\libdasm-master\libdasm-master\pydasm>python setup.py build_ext**`
   (b) `C:\Users\IEUser\Downloads\libdasm-master\libdasm-master\pydasm>python setup.py install`
3. Verify that process_monitor.py runs:

```
C:\Users\IEUser\Downloads\boofuzz>python process_monitor.py -h
usage: procmon [-h] [--debug] [--quiet] [-f STR] [-c FILENAME] [-i PID]
               [-l LEVEL] [-p NAME] [-P PORT]

optional arguments:
  -h, --help            show this help message and exit
  --debug               toggle debug output
  --quiet               suppress all output
  -f STR, --foo STR     the notorious foo option
  -c FILENAME, --crash_bin FILENAME
                        filename to serialize crash bin class to
  -i PID, --ignore_pid PID
                        PID to ignore when searching for target process
  -l LEVEL, --log_level LEVEL
                        log level: default 1, increase for more verbosity
  -p NAME, --proc_name NAME
                        process name to search for and attach to
  -P PORT, --port PORT  TCP port to bind this agent to
```

** Building pydasm on Windows requires the Visual C++ Compiler for Python 2.7.

#### Deprecated: network_monitor.py

The network monitor was Sulley's primary tool for recording test data, and has been replaced with boofuzz's logging mechanisms. However, some people still prefer the PCAP approach.

## 4.2 Quickstart

The *Session* object is the center of your fuzz... session. When you create it, you'll pass it a *Target* object, which will itself receive a *Connection* object. For example:

```
session = Session(
    target=Target(
        connection=SocketConnection("127.0.0.1", 8021, proto='tcp')))
```

Connection objects implement *ITargetConnection*. Available options include *SocketConnection* and *SerialConnection*.

With a Session object ready, you next need to define the messages in your protocol. Once you've read the requisite RFC, tutorial, etc., you should be confident enough in the format to define your protocol using the various *static protocol definition functions*.

Each message starts with an *s_initialize* function.

Here are several message definitions from the FTP protocol:

```
s_initialize("user")
s_string("USER")
s_delim(" ")
s_string("anonymous")
s_static("\r\n")

s_initialize("pass")
s_string("PASS")
s_delim(" ")
s_string("james")
s_static("\r\n")

s_initialize("stor")
s_string("STOR")
s_delim(" ")
s_string("AAAA")
s_static("\r\n")

s_initialize("retr")
s_string("RETR")
s_delim(" ")
s_string("AAAA")
s_static("\r\n")
```

Once you've defined your message(s), you will connect them into a graph using the Session object you just created.:

```
session.connect(s_get("user"))
session.connect(s_get("user"), s_get("pass"))
session.connect(s_get("pass"), s_get("stor"))
session.connect(s_get("pass"), s_get("retr"))
```

After that, you are ready to fuzz:

```
session.fuzz()
```

Note that at this point you have only a very basic fuzzer. Making it kick butt is up to you.

To do cool stuff like checking responses, you'll want to use *Session.post_send*. You may also want be interested in *Making Your Own Block/Primitive*.

Remember boofuzz is all Python, so everything is there for your customization. If you are doing crazy cool stuff, check out the *community info* and consider contributing back!

Happy fuzzing, and Godspeed!

# CHAPTER 5

# Public Protocol Libraries

The following protocol libraries are free and open source, but the implementations are not at all close to full protocol coverage:

- boofuzz-ftp
- boofuzz-http

If you have an open source boofuzz protocol suite to share, please *let us know*!

CHAPTER 6

# API Documentation

## 6.1 Session

**class** boofuzz.**Session**(*session_filename=None*, *skip=0*, *sleep_time=0.0*, *restart_interval=0*, *web_port=26000*, *crash_threshold=3*, *restart_sleep_time=5*, *fuzz_data_logger=None*, *check_data_received_each_request=True*, *log_level=20*, *logfile=None*, *logfile_level=10*, *ignore_connection_reset=False*, *ignore_connection_aborted=False*, *target=None*)

Bases: boofuzz.pgraph.graph.Graph

Extends pgraph.graph and provides a container for architecting protocol dialogs.

**Parameters**

- **session_filename** (*str*) – Filename to serialize persistent data to. Default None.
- **skip** (*int*) – Number of test cases to skip. Default 0.
- **sleep_time** (*float*) – Time in seconds to sleep in between tests. Default 0.
- **restart_interval** (*int*) – Restart the target after n test cases, disable by setting to 0 (default).
- **crash_threshold** (*int*) – Maximum number of crashes allowed before a node is exhaust. Default 3.
- **restart_sleep_time** (*int*) – Time in seconds to sleep when target can't be restarted. Default 5.
- **web_port** (*int*) – Port for monitoring fuzzing campaign via a web browser. Default 26000.
- **fuzz_data_logger** (*fuzz_logger.FuzzLogger*) – For saving test data and results.. Default Log to STDOUT.
- **check_data_received_each_request** (*bool*) – If True, Session will verify that some data has been received after transmitting each node, and if not, register a failure. If False, this check will not be performed. Default True.

- **ignore_connection_reset** (*bool*) – Log ECONNRESET errors ("Target connection reset") as "info" instead of failures.
- **ignore_connection_aborted** (*bool*) – Log ECONNABORTED errors as "info" instead of failures.
- **target** (Target) – Target for fuzz session. Target must be fully initialized. Default None.
- **log_level** (*int*) – DEPRECATED Unused. Logger settings are now configured in fuzz_data_logger. Was once used to set the log level.
- **logfile** (*str*) – DEPRECATED Unused. Logger settings are now configured in fuzz_data_logger. Was once the name of the log file.
- **logfile_level** (*int*) – DEPRECATED Unused. Logger settings are now configured in fuzz_data_logger. Was once used to set the log level for the logfile. Default logger.INFO.

**add_node**(*node*)

Add a pgraph node to the graph. We overload this routine to automatically generate and assign an ID whenever a node is added.

**Parameters** **node** (*pgraph.Node*) – Node to add to session graph

**add_target**(*target*)

Add a target to the session. Multiple targets can be added for parallel fuzzing.

**Parameters** **target** (Target) – Target to add to session

**build_webapp_thread**(*port=26000*)

**connect**(*src*, *dst=None*, *callback=None*)

Create a connection between the two requests (nodes) and register an optional callback to process in between transmissions of the source and destination request. Leverage this functionality to handle situations such as challenge response systems. The session class maintains a top level node that all initial requests must be connected to. Example:

```
sess = sessions.session()
sess.connect(sess.root, s_get("HTTP"))
```

If given only a single parameter, sess.connect() will default to attaching the supplied node to the root node. This is a convenient alias and is identical to the second line from the above example:

```
sess.connect(s_get("HTTP"))
```

If you register callback method, it must follow this prototype:

```
def callback(session, node, edge, sock)
```

Where node is the node about to be sent, edge is the last edge along the current fuzz path to "node", session is a pointer to the session instance which is useful for snagging data such as session.last_recv which contains the data returned from the last socket transmission and sock is the live socket. A callback is also useful in situations where, for example, the size of the next packet is specified in the first packet. As another example, if you need to fill in the dynamic IP address of the target register a callback that snags the IP from sock.getpeername()[0].

**Parameters**

- **src** (*str or Request (pgrah.Node)*) – Source request name or request node
- **dst** (*str or Request (pgrah.Node), optional*) – Destination request name or request node

- **callback** (*def, optional*) – Callback function to pass received data to between node xmits. Default None.

**Returns** The edge between the src and dst.

**Return type** pgraph.Edge

**export_file**()
Dump various object values to disk.

@see: import_file()

**fuzz**()
Call this routine to get the ball rolling. Iterates through and fuzzes all fuzz cases, skipping according to self.skip and restarting based on self.restart_interval.

If you want the web server to be available, your program must persist after calling this method. helpers.pause_for_signal() is available to this end.

**Returns** None

**fuzz_single_case**(*mutant_index*)
Fuzz a test case by mutant_index.

**Parameters** **mutant_index** (*int*) – Non-negative integer.

**Returns** None

**Raises** sex.SulleyRuntimeError – If any error is encountered while executing the test case.

**import_file**()
Load various object values from disk.

@see: export_file()

**log**(*msg*, *level=1*)

**num_mutations**(*this_node=None*, *path=()*)
Number of total mutations in the graph. The logic of this routine is identical to that of fuzz(). See fuzz() for inline comments. The member variable self.total_num_mutations is updated appropriately by this routine.

**Parameters**

- **this_node** (*request (node)*) – Current node that is being fuzzed. Default None.
- **path** (*list*) – Nodes along the path to the current one being fuzzed. Default [].

**Returns** Total number of mutations in this session.

**Return type** int

**pause**()
If that pause flag is raised, enter an endless loop until it is lowered.

**poll_pedrpc**(*target*)
Poll the PED-RPC endpoints (netmon, procmon etc...) for the target.

**Parameters** **target** (Target) – Session target whose PED-RPC services we are polling

**post_send**(*target*, *fuzz_data_logger*, *session*, *sock*, **args*, ***kwargs*)
Overload or replace this routine to specify actions to run after to each fuzz request. The order of events is as follows:

```
pre_send() - req - callback ... req - callback - post_send()
```

**Potential uses:**

- Closing down a connection.
- Checking for expected responses.

@see: pre_send()

**Parameters**

- **target** (Target) – Target with sock-like interface.
- **fuzz_data_logger** (*ifuzz_logger.IFuzzLogger*) – Allows logging of test checks and passes/failures. Provided with a test case and test step already opened.
- **session** (Session) – Session object calling post_send. Useful properties include last_send and last_recv.
- **sock** – DEPRECATED Included for backward-compatibility. Same as target.
- **args** – Implementations should include *args and **kwargs for forward-compatibility.
- **kwargs** – Implementations should include *args and **kwargs for forward-compatibility.

**pre_send**(*sock*)

Overload or replace this routine to specify actions to run prior to each fuzz request. The order of events is as follows:

```
pre_send() - req - callback ... req - callback - post_send()
```

When fuzzing RPC for example, register this method to establish the RPC bind.

@see: pre_send()

**Parameters** **sock** (*Socket*) – Connected socket to target

**restart_target**(*target*)

Restart the fuzz target. If a VMControl is available revert the snapshot, if a process monitor is available restart the target process. Otherwise, do nothing.

**Parameters** **target** (*session.target*) – Target we are restarting

@raise sex.BoofuzzRestartFailedError if restart fails.

**server_init**()

Called by fuzz() to initialize variables, web interface, etc.

**transmit**(*sock*, *node*, *edge*)

Render and transmit a node, process callbacks accordingly.

**Parameters**

- **sock** (Target, *optional*) – Socket-like object on which to transmit node
- **node** (*pgraph.node.node (Node), optional*) – Request/Node to transmit
- **edge** (*pgraph.edge.edge (pgraph.edge), optional*) – Edge along the current fuzz path from "node" to next node.

## 6.2 Target

**class** boofuzz.**Target**(*connection*, *procmon=None*, *procmon_options=None*, *netmon=None*)

Bases: object

Target descriptor container.

Takes an ITargetConnection and wraps send/recv with appropriate FuzzDataLogger calls.

Encapsulates pedrpc connection logic.

Contains a logger which is configured by Session.add_target().

#### Example

tcp_target = Target(SocketConnection(host='127.0.0.1', port=17971))

**close**()
Close connection to the target.

**Returns** None

**open**()
Opens connection to the target. Make sure to call close!

**Returns** None

**pedrpc_connect**()
Pass specified target parameters to the PED-RPC server.

**recv**(*max_bytes*)
Receive up to max_bytes data from the target.

**Parameters** **max_bytes** (*int*) – Maximum number of bytes to receive.

**Returns** Received data.

**send**(*data*)
Send data to the target. Only valid after calling open!

**Parameters** **data** – Data to send.

**Returns** None

**set_fuzz_data_logger**(*fuzz_data_logger*)
Set this object's fuzz data logger – for sent and received fuzz data.

**Parameters** **fuzz_data_logger** (*ifuzz_logger.IFuzzLogger*) – New logger.

**Returns** None

## 6.3 Connections

Connection objects implement *ITargetConnection*. Available options include *SocketConnection* and *SerialConnection*.

### 6.3.1 ITargetConnection

**class** boofuzz.**ITargetConnection**
Bases: object

Interface for connections to fuzzing targets. Target connections may be opened and closed multiple times. You must open before using send/recv and close afterwards.

**close**()
Close connection.

**Returns** None

**open**()
Opens connection to the target. Make sure to call close!

**Returns** None

**recv**(*max_bytes*)
Receive up to max_bytes data.

**Parameters** **max_bytes** (*int*) – Maximum number of bytes to receive.

**Returns** Received data. bytes('') if no data is received.

**send**(*data*)
Send data to the target.

**Parameters** **data** – Data to send.

:rtype int :return: Number of bytes actually sent.

### 6.3.2 SocketConnection

**class** boofuzz.**SocketConnection**(*host*, *port=None*, *proto='tcp'*, *bind=None*, *timeout=5.0*, *ethernet_proto=2048*, *l2_dst='xffxffxffxffxffxff'*, *udp_broadcast=False*)
Bases: boofuzz.itarget_connection.ITargetConnection

ITargetConnection implementation using sockets.

Supports UDP, TCP, SSL, raw layer 2 and raw layer 3 packets.

Examples:

```
tcp_connection = SocketConnection(host='127.0.0.1', port=17971)
udp_connection = SocketConnection(host='127.0.0.1', port=17971, proto='udp')
udp_connection_2_way = SocketConnection(host='127.0.0.1', port=17971, proto='udp',
↪ bind=('127.0.0.1', 17972)
udp_broadcast = SocketConnection(host='127.0.0.1', port=17971, proto='udp', bind=(
↪'127.0.0.1', 17972),
                                 udp_broadcast=True)
raw_layer_2 = (host='lo', proto='raw-l2')
raw_layer_2 = (host='lo', proto='raw-l2',
               l2_dst='\xFF\xFF\xFF\xFF\xFF\xFF', ethernet_proto=socket_
↪connection.ETH_P_IP)
raw_layer_3 = (host='lo', proto='raw-l3')
```

**Parameters**

- **host** (*str*) – Hostname or IP address of target system, or network interface string if using raw-l2 or raw-l3.
- **port** (*int*) – Port of target service. Required for proto values 'tcp', 'udp', 'ssl'.
- **proto** (*str*) – Communication protocol ("tcp", "udp", "ssl", "raw-l2", "raw-l3"). Default "tcp". raw-l2: Send packets at layer 2. Must include link layer header (e.g. Ethernet frame). raw-l3: Send packets at layer 3. Must include network protocol header (e.g. IPv4).
- **bind** (*tuple (host, port)*) – Socket bind address and port. Required if using recv() with 'udp' protocol.

- **timeout** (*float*) – Seconds to wait for a send/recv prior to timing out. Default 5.0.
- **ethernet_proto** (*int*) – Ethernet protocol when using 'raw-l3'. 16 bit integer. Default ETH_P_IP (0x0800). See "if_ether.h" in Linux documentation for more options.
- **l2_dst** (*str*) – Layer 2 destination address (e.g. MAC address). Used only by 'raw-l3'. Default 'ÿÿÿÿÿÿ' (broadcast).
- **udp_broadcast** (*bool*) – Set to True to enable UDP broadcast. Must supply appropriate broadcast address for send() to work, and '' for bind host for recv() to work.

**MAX_PAYLOADS = {'raw-l3': 1500, 'raw-l2': 1514, 'udp': 65507}**

**close**()
Close connection to the target.

**Returns** None

**open**()
Opens connection to the target. Make sure to call close!

**Returns** None

**recv**(*max_bytes*)
Receive up to max_bytes data from the target.

**Parameters** **max_bytes** (*int*) – Maximum number of bytes to receive.

**Returns** Received data.

**send**(*data*)
Send data to the target. Only valid after calling open! Some protocols will truncate; see self.MAX_PAYLOADS.

**Parameters** **data** – Data to send.

**Returns** Number of bytes actually sent.

**Return type** int

### 6.3.3 SerialConnection

**class** boofuzz.**SerialConnection**(*port=0*, *baudrate=9600*, *timeout=5*, *message_separator_time=0.3*, *content_checker=None*)
Bases: boofuzz.itarget_connection.ITargetConnection

ITargetConnection implementation for generic serial ports.

Since serial ports provide no default functionality for separating messages/packets, this class provides several means:

- timeout: Return received bytes after timeout seconds.
- msg_separator_time: Return received bytes after the wire is silent for a given time. This is useful, e.g., for terminal protocols without a machine-readable delimiter. A response may take a long time to send its information, and you know the message is done when data stops coming.
- content_check: A user-defined function takes the data received so far and checks for a packet. The function should return 0 if the packet isn't finished yet, or n if a valid message of n bytes has been received. Remaining bytes are stored for next call to recv(). Example:

```
def content_check_newline(data):
if data.find('\n') >= 0:
    return data.find('\n')
else:
    return 0
```

If none of these methods are used, your connection may hang forever.

**Parameters**

- **port** (*Union[int, str]*) – Serial port name or number.
- **baudrate** (*int*) – Baud rate for port.
- **timeout** (*float*) – For recv(). After timeout seconds from receive start, recv() will return all received data, if any.
- **message_separator_time** (*float*) – After message_separator_time seconds _without receiving any more **data_**, recv() will return. Optional. Default None.
- **content_checker** (*function(str) -> int*) – User-defined function. recv() will pass all bytes received so far to this method. If the method returns n > 0, recv() will return n bytes. If it returns 0, recv() will keep on reading.

**close**()
Close connection to the target.

**Returns** None

**open**()
Opens connection to the target. Make sure to call close!

**Returns** No ne

**recv**(*max_bytes*)
Receive up to max_bytes data from the target.

**Parameters** **max_bytes** (*int*) – Maximum number of bytes to receive.

**Returns** Received data.

**send**(*data*)
Send data to the target. Only valid after calling open!

**Parameters** **data** – Data to send.

**Returns** Number of bytes actually sent.

**Return type** int

## 6.4 Logging

Boofuzz provides flexible logging. All logging classes implement *IFuzzLogger*. Built-in logging classes are detailed below.

### 6.4.1 Logging Interface

**class** boofuzz.**IFuzzLogger**
Bases: object

Abstract class for logging fuzz data.

**Usage while testing:**

1. Open test case.
2. Open test step.
3. Use other log methods.

IFuzzLogger provides the logging interface for the Sulley framework and test writers.

The methods provided are meant to mirror functional test actions. Instead of generic debug/info/warning methods, IFuzzLogger provides a means for logging test cases, passes, failures, test steps, etc.

This hypothetical sample output gives an idea of how the logger should be used:

**Test Case: UDP.Header.Address 3300**

**Test Step: Fuzzing** Send: 45 00 13 ab 00 01 40 00 40 11 c9 ...

**Test Step: Process monitor check** Check OK

**Test Step: DNP Check** Send: ff ff ff ff ff ff 00 0c 29 d1 10 ... Recv: 00 0c 29 d1 10 81 00 30 a7 05 6e ... Check: Reply is as expected. Check OK

**Test Case: UDP.Header.Address 3301**

**Test Step: Fuzzing** Send: 45 00 13 ab 00 01 40 00 40 11 c9 ...

**Test Step: Process monitor check** Check Failed: "Process returned exit code 1"

**Test Step: DNP Check** Send: ff ff ff ff ff ff 00 0c 29 d1 10 ... Recv: None Check: Reply is as expected. Check Failed

A test case is opened for each fuzzing case. A test step is opened for each high-level test step. Test steps can include, for example:

- Fuzzing
- Set up (pre-fuzzing)
- Post-test cleanup
- Instrumentation checks
- Reset due to failure

Within a test step, a test may log data sent, data received, checks, check results, and other information.

**log_check**(*description*)
Records a check on the system under test. AKA "instrumentation check."

**Parameters** **description** (*str*) – Received data.

**Returns** None

**Return type** None

**log_error**(*description*)
Records an internal error. This informs the operaor that the test was not completed successfully.

**Parameters** **description** (*str*) – Received data.

**Returns** None

**Return type** None

**log_fail**(*description=''*)
Records a check that failed. This will flag a fuzzing case as a potential bug or anomaly.

**Parameters** **description** (*str*) – Optional supplementary data.

**Returns** None

**Return type** None

**log_info**(*description*)
Catch-all method for logging test information

**Parameters** **description** (*str*) – Information.

**Returns** None

**Return type** None

**log_pass**(*description=''*)
Records a check that passed.

**Parameters** **description** (*str*) – Optional supplementary data..

**Returns** None

**Return type** None

**log_recv**(*data*)
Records data as having been received from the target.

**Parameters** **data** (*bytes*) – Received data.

**Returns** None

**Return type** None

**log_send**(*data*)
Records data as about to be sent to the target.

**Parameters** **data** (*bytes*) – Transmitted data

**Returns** None

**Return type** None

**open_test_case**(*test_case_id*)
Open a test case - i.e., a fuzzing mutation.

**Parameters** **test_case_id** – Test case name/number. Should be unique.

**Returns** None

**open_test_step**(*description*)
Open a test step - e.g., "Fuzzing", "Pre-fuzz", "Response Check."

**Parameters** **description** – Description of fuzzing step.

**Returns** None

boofuzz.**IFuzzLoggerBackend**
alias of IFuzzLogger

### 6.4.2 Text Logging

**class** boofuzz.**FuzzLoggerText**(*file_handle=<colorama.ansitowin32.StreamWrapper object>*, *bytes_to_str=<function hex_to_hexstr>*)

Bases: boofuzz.ifuzz_logger.IFuzzLogger

This class formats FuzzLogger data for text presentation. It can be configured to output to STDOUT, or to a named file.

Using two FuzzLoggerTexts, a FuzzLogger instance can be configured to output to both console and file.

**DEFAULT_TEST_CASE_ID = 'DefaultTestCase'**

**INDENT_SIZE = 2**

**LOG_CHECK_FORMAT = 'Check: {0}'**

**LOG_ERROR_FORMAT = '\x1b[41m\x1b[1mError!!!! {0}\x1b[0m'**

**LOG_FAIL_FORMAT = '\x1b[31m\x1b[1mCheck Failed: {0}\x1b[0m'**

**LOG_INFO_FORMAT = 'Info: {0}'**

**LOG_PASS_FORMAT = '\x1b[32m\x1b[1mCheck OK: {0}\x1b[0m'**

**LOG_RECV_FORMAT = '\x1b[36mReceived: {0}\x1b[0m'**

**LOG_SEND_FORMAT = '\x1b[36mTransmitting {0} bytes: {1}\x1b[0m'**

**TEST_CASE_FORMAT = '\x1b[33m\x1b[1mTest Case: {0}\x1b[0m'**

**TEST_STEP_FORMAT = '\x1b[35m\x1b[1mTest Step: {0}\x1b[0m'**

**log_check**(*description*)

**log_error**(*description*)

**log_fail**(*description=''*)

**log_info**(*description*)

**log_pass**(*description=''*)

**log_recv**(*data*)

**log_send**(*data*)

**open_test_case**(*test_case_id*)

**open_test_step**(*description*)

### 6.4.3 CSV Logging

**class** boofuzz.**FuzzLoggerCsv**(*file_handle=<colorama.ansitowin32.StreamWrapper object>*, *bytes_to_str=<function hex_to_hexstr>*)

Bases: boofuzz.ifuzz_logger.IFuzzLogger

This class formats FuzzLogger data for pcap file. It can be configured to output to a named file.

**log_check**(*description*)

**log_error**(*description*)

**log_fail**(*description=''*)

**log_info**(*description*)

**log_pass**(*description=''*)

**log_recv**(*data*)

**log_send**(*data*)

**open_test_case**(*test_case_id*)

**open_test_step**(*description*)

### 6.4.4 File Logging

### 6.4.5 FuzzLogger Object

**class** boofuzz.**FuzzLogger**(*fuzz_loggers=None*)
Bases: boofuzz.ifuzz_logger.IFuzzLogger

Implementation for IFuzzLogger.

FuzzLogger takes logged data and directs it to the appropriate backends. It aggregates an arbitrary number of logger backends, and functions like a multiplexer.

FuzzLogger also maintains failure and error data.

**failure_summary**()
Return test summary string based on fuzz logger results.

**Returns** Test summary string, may be multi-line.

**log_check**(*description*)

**log_error**(*description*)

**log_fail**(*description=''*)

**log_info**(*description*)

**log_pass**(*description=''*)

**log_recv**(*data*)

**log_send**(*data*)

**open_test_case**(*test_case_id*)

**open_test_step**(*description*)

## 6.5 Static Protocol Definition

Static functions are used in boofuzz to assemble messages for a protocol definition. They may be obsoleted in future releases by a less static approach to message construction. For now, you can see the *Quickstart* guide for an intro.

Requests are messages, Blocks are chunks within a message, and Primitives are the elements (bytes, strings, numbers, checksums, etc.) that make up a Block/Request.

### 6.5.1 Request Manipulation

boofuzz.**s_initialize**(*name*)
Initialize a new block request. All blocks / primitives generated after this call apply to the named request. Use s_switch() to jump between factories.

**Parameters** **name** (*str*) – Name of request

boofuzz.**s_get**(*name=None*)
Return the request with the specified name or the current request if name is not specified. Use this to switch from global function style request manipulation to direct object manipulation. Example:

```
req = s_get("HTTP BASIC")
print req.num_mutations()
```

The selected request is also set as the default current. (ie: s_switch(name) is implied).

**Parameters** **name** (*str*) – (Optional, def=None) Name of request to return or current request if name is None.

**Return type** blocks.request

**Returns** The requested request.

boofuzz.**s_mutate**()
Mutate the current request and return False if mutations are exhausted, in which case the request has been reverted back to its normal form.

**Return type** bool

**Returns** True on mutation success, False if mutations exhausted.

boofuzz.**s_num_mutations**()
Determine the number of repetitions we will be making.

**Return type** int

**Returns** Number of mutated forms this primitive can take.

boofuzz.**s_render**()
Render out and return the entire contents of the current request.

**Return type** Raw

**Returns** Rendered contents

boofuzz.**s_switch**(*name*)
Change the currect request to the one specified by "name".

**Parameters** **name** (*str*) – Name of request

### 6.5.2 Block Manipulation

boofuzz.**s_block**(*name*, *group=None*, *encoder=None*, *dep=None*, *dep_value=None*, *dep_values=()*, *dep_compare='=='*)
Open a new block under the current request. The returned instance supports the "with" interface so it will be automatically closed for you:

```
with s_block("header"):
    s_static("\x00\x01")
    if s_block_start("body"):
        ...
```

**Parameters**

- **name** (*str*) – Name of block being opened
- **group** (*str*) – (Optional, def=None) Name of group to associate this block with

- **encoder** (*Function Pointer*) – (Optional, def=None) Optional pointer to a function to pass rendered data to prior to return
- **dep** (*str*) – (Optional, def=None) Optional primitive whose specific value this block is dependant on
- **dep_value** (*Mixed*) – (Optional, def=None) Value that field "dep" must contain for block to be rendered
- **dep_values** (*List of Mixed Types*) – (Optional, def=[]) Values that field "dep" may contain for block to be rendered
- **dep_compare** (*str*) – (Optional, def="==") Comparison method to use on dependency (==, !=, >, >=, <, <=)

boofuzz.**s_block_start**(*name*, **args*, ***kwargs*)

Open a new block under the current request. This routine always returns an instance so you can make your fuzzer pretty with indenting:

```
if s_block_start("header"):
    s_static("\x00\x01")
    if s_block_start("body"):
        ...
s_block_close()
```

:note Prefer using s_block to this function directly :see s_block

boofuzz.**s_block_end**(*name=None*)

Close the last opened block. Optionally specify the name of the block being closed (purely for aesthetic purposes).

**Parameters** **name** (*str*) – (Optional, def=None) Name of block to closed.

boofuzz.**s_checksum**(*block_name*, *algorithm='crc32'*, *length=0*, *endian='<'*, *fuzzable=True*, *name=None*, *ipv4_src_block_name=None*, *ipv4_dst_block_name=None*)

Create a checksum block bound to the block with the specified name. You *can not* create a checksum for any currently open blocks.

**Parameters**

- **block_name** (*str*) – Name of block to apply sizer to
- **algorithm** (*str*) – (Optional, def=crc32) Checksum algorithm to use. (crc32, adler32, md5, sha1, ipv4, udp)
- **length** (*int*) – (Optional, def=0) NOT IMPLEMENTED. Length of checksum, specify 0 to auto-calculate
- **endian** (*Character*) – (Optional, def=LITTLE_ENDIAN) Endianness of the bit field (LITTLE_ENDIAN: <, BIG_ENDIAN: >)
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing.
- **name** (*str*) – Name of this checksum field
- **ipv4_src_block_name** (*str*) – Required for 'udp' algorithm. Name of block yielding IPv4 source address.
- **ipv4_dst_block_name** (*str*) – Required for 'udp' algorithm. Name of block yielding IPv4 destination address.

boofuzz.**s_repeat**(*block_name*, *min_reps=0*, *max_reps=None*, *step=1*, *variable=None*, *fuzzable=True*, *name=None*)

Repeat the rendered contents of the specified block cycling from min_reps to max_reps counting by step. By default renders to nothing. This block modifier is useful for fuzzing overflows in table entries. This block modifier MUST come after the block it is being applied to.

**See** Aliases: s_repeater()

**Parameters**

- **block_name** (*str*) – Name of block to apply sizer to
- **min_reps** (*int*) – (Optional, def=0) Minimum number of block repetitions
- **max_reps** (*int*) – (Optional, def=None) Maximum number of block repetitions
- **step** (*int*) – (Optional, def=1) Step count between min and max reps
- **variable** (*Sulley Integer Primitive*) – (Optional, def=None) An integer primitive which will specify the number of repitions
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_size**(*block_name*, *offset=0*, *length=4*, *endian='<'*, *output_format='binary'*, *inclusive=False*, *signed=False*, *math=None*, *fuzzable=True*, *name=None*)

Create a sizer block bound to the block with the specified name. You *can not* create a sizer for any currently open blocks.

**See** Aliases: s_sizer()

**Parameters**

- **block_name** (*str*) – Name of block to apply sizer to
- **offset** (*int*) – (Optional, def=0) Offset to calculated size of block
- **length** (*int*) – (Optional, def=4) Length of sizer
- **endian** (*Character*) – (Optional, def=LITTLE_ENDIAN) Endianess of the bit field (LITTLE_ENDIAN: <, BIG_ENDIAN: >)
- **output_format** (*str*) – (Optional, def=binary) Output format, "binary" or "ascii"
- **inclusive** (*bool*) – (Optional, def=False) Should the sizer count its own length?
- **signed** (*bool*) – (Optional, def=False) Make size signed vs. unsigned (applicable only with format="ascii")
- **math** (*Function*) – (Optional, def=None) Apply the mathematical operations defined in this function to the size
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this sizer
- **name** (*str*) – Name of this sizer field

boofuzz.**s_update**(*name*, *value*)

Update the value of the named primitive in the currently open request.

**Parameters**

- **name** (*str*) – Name of object whose value we wish to update
- **value** (*Mixed*) – Updated value

### 6.5.3 Primitive Definition

boofuzz.**s_binary**(*value*, *name=None*)
Parse a variable format binary string into a static value and push it onto the current block stack.

**Parameters**

- **value** (*str*) – Variable format binary string
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_delim**(*value*, *fuzzable=True*, *name=None*)
Push a delimiter onto the current block stack.

**Parameters**

- **value** (*Character*) – Original value
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_group**(*name*, *values*)
This primitive represents a list of static values, stepping through each one on mutation. You can tie a block to a group primitive to specify that the block should cycle through all possible mutations for *each* value within the group. The group primitive is useful for example for representing a list of valid opcodes.

**Parameters**

- **name** (*str*) – Name of group
- **values** (*List or raw data*) – List of possible raw values this group can take.

boofuzz.**s_lego**(*lego_type*, *value=None*, *options=()*)
Legos are pre-built blocks... TODO: finish this doc

**Parameters**

- **lego_type** (*str*) – Function that represents a lego
- **value** – Original value
- **options** – Options to pass to lego.

boofuzz.**s_random**(*value*, *min_length*, *max_length*, *num_mutations=25*, *fuzzable=True*, *step=None*, *name=None*)
Generate a random chunk of data while maintaining a copy of the original. A random length range can be specified. For a static length, set min/max length to be the same.

**Parameters**

- **value** (*Raw*) – Original value
- **min_length** (*int*) – Minimum length of random block
- **max_length** (*int*) – Maximum length of random block
- **num_mutations** (*int*) – (Optional, def=25) Number of mutations to make before reverting to default
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **step** (*int*) – (Optional, def=None) If not null, step count between min and max reps, otherwise random

- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_static**(*value*, *name=None*)
Push a static value onto the current block stack.

**See** Aliases: s_dunno(), s_raw(), s_unknown()

**Parameters**

- **value** (*Raw*) – Raw static data
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_string**(*value*, *size=-1*, *padding='\x00'*, *encoding='ascii'*, *fuzzable=True*, *max_len=0*, *name=None*)
Push a string onto the current block stack.

**Parameters**

- **value** (*str*) – Default string value
- **size** (*int*) – (Optional, def=-1) Static size of this field, leave -1 for dynamic.
- **padding** (*Character*) – (Optional, def="x00") Value to use as padding to fill static field size.
- **encoding** (*str*) – (Optonal, def="ascii") String encoding, ex: utf_16_le for Microsoft Unicode.
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **max_len** (*int*) – (Optional, def=0) Maximum string length
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_from_file**(*value*, *encoding='ascii'*, *fuzzable=True*, *max_len=0*, *name=None*, *filename=None*)
Push a value from file onto the current block stack.

**Parameters**

- **value** (*str*) – Default string value
- **encoding** (*str*) – (Optonal, def="ascii") String encoding, ex: utf_16_le for Microsoft Unicode.
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **max_len** (*int*) – (Optional, def=0) Maximum string length
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive
- **filename** (*str*) – (Mandatory) Specify filename where to read fuzz list

boofuzz.**s_bit_field**(*value*, *width*, *endian='<'*, *output_format='binary'*, *signed=False*, *full_range=False*, *fuzzable=True*, *name=None*)
Push a variable length bit field onto the current block stack.

**See** Aliases: s_bit(), s_bits()

**Parameters**

- **value** (*int*) – Default integer value

- **width** (*int*) – Width of bit fields
- **endian** (*Character*) – (Optional, def=LITTLE_ENDIAN) Endianess of the bit field (LITTLE_ENDIAN: <, BIG_ENDIAN: >)
- **format** (*output_format*) – (Optional, def=binary) Output format, "binary" or "ascii"
- **signed** (*bool*) – (Optional, def=False) Make size signed vs. unsigned (applicable only with format="ascii")
- **full_range** (*bool*) – (Optional, def=False) If enabled the field mutates through *all* possible values.
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_byte**(*value*, *endian='<'*, *output_format='binary'*, *signed=False*, *full_range=False*, *fuzzable=True*, *name=None*)

Push a byte onto the current block stack.

**See** Aliases: s_char()

**Parameters**

- **value** (*int|str*) – Default integer value
- **endian** (*Character*) – (Optional, def=LITTLE_ENDIAN) Endianess of the bit field (LITTLE_ENDIAN: <, BIG_ENDIAN: >)
- **output_format** (*str*) – (Optional, def=binary) Output format, "binary" or "ascii"
- **signed** (*bool*) – (Optional, def=False) Make size signed vs. unsigned (applicable only with format="ascii")
- **full_range** (*bool*) – (Optional, def=False) If enabled the field mutates through *all* possible values.
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_word**(*value*, *endian='<'*, *output_format='binary'*, *signed=False*, *full_range=False*, *fuzzable=True*, *name=None*)

Push a word onto the current block stack.

**See** Aliases: s_short()

**Parameters**

- **value** (*int*) – Default integer value
- **endian** (*chr*) – (Optional, def=LITTLE_ENDIAN) Endianess of the bit field (LITTLE_ENDIAN: <, BIG_ENDIAN: >)
- **output_format** (*str*) – (Optional, def=binary) Output format, "binary" or "ascii"
- **signed** (*bool*) – (Optional, def=False) Make size signed vs. unsigned (applicable only with format="ascii")
- **full_range** (*bool*) – (Optional, def=False) If enabled the field mutates through *all* possible values.
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive

- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_dword**(*value*, *endian='<'*, *output_format='binary'*, *signed=False*, *full_range=False*, *fuzzable=True*, *name=None*)

Push a double word onto the current block stack.

**See** Aliases: s_long(), s_int()

**Parameters**

- **value** (*int*) – Default integer value
- **endian** (*Character*) – (Optional, def=LITTLE_ENDIAN) Endianess of the bit field (LITTLE_ENDIAN: <, BIG_ENDIAN: >)
- **output_format** (*str*) – (Optional, def=binary) Output format, "binary" or "ascii"
- **signed** (*bool*) – (Optional, def=False) Make size signed vs. unsigned (applicable only with format="ascii")
- **full_range** (*bool*) – (Optional, def=False) If enabled the field mutates through *all* possible values.
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

boofuzz.**s_qword**(*value*, *endian='<'*, *output_format='binary'*, *signed=False*, *full_range=False*, *fuzzable=True*, *name=None*)

Push a quad word onto the current block stack.

**See** Aliases: s_double()

**Parameters**

- **value** (*int*) – Default integer value
- **endian** (*Character*) – (Optional, def=LITTLE_ENDIAN) Endianess of the bit field (LITTLE_ENDIAN: <, BIG_ENDIAN: >)
- **output_format** (*str*) – (Optional, def=binary) Output format, "binary" or "ascii"
- **signed** (*bool*) – (Optional, def=False) Make size signed vs. unsigned (applicable only with format="ascii")
- **full_range** (*bool*) – (Optional, def=False) If enabled the field mutates through *all* possible values.
- **fuzzable** (*bool*) – (Optional, def=True) Enable/disable fuzzing of this primitive
- **name** (*str*) – (Optional, def=None) Specifying a name gives you direct access to a primitive

### 6.5.4 Making Your Own Block/Primitive

Now I know what you're thinking: "With that many sweet primitives and blocks available, what else could I ever conceivably need? And yet, I am urged by joy to contribute my own sweet blocks!"

To make your own block/primitive:

1. Create an object that implements `IFuzzable`.
2. Create an accompanying static primitive function. See boofuzz's *__init__.py* file for examples.

3. ???
4. Profit!

If your block depends on references to other blocks, the way a checksum or length field depends on other parts of the message, see the `Size` source code for an example of how to avoid recursion issues. Or otherwise be careful. :)

CHAPTER 7

# Contributions

Pull requests are welcome, as boofuzz is actively maintained (at the time of this writing ;)). See contributing.

# CHAPTER 8

# Community

For questions that take the form of "How do I… with boofuzz?" or "I got this error with boofuzz, why?", consider posting your question on Stack Overflow. Make sure to use the `fuzzing` tag.

If you've found a bug, or have an idea/suggestion/request, file an issue here on GitHub.

For other questions, check out boofuzz on gitter or Google Groups.

For updates, follow @fuzztheplanet on Twitter.

CHAPTER 9

# Indices and tables

- genindex
- modindex
- search

# Index