



Peach Fuzzer Professional User Guide

Peach Fuzzer, LLC

Version 0.0.0

Copyright © 2016 Peach Fuzzer, LLC. All rights reserved.

This document may not be distributed or used for commercial purposes without the explicit consent of the copyright holders.

Peach Fuzzer® is a registered trademark of Peach Fuzzer, LLC.

Peach Fuzzer contains Patent Pending technologies.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Peach Fuzzer, LLC
1122 E Pike St
Suite 1064
Seattle, WA 98122

Table of Contents

1. Preface.....	1
1.1. What is Peach Fuzzer Professional?.....	1
1.2. Additional Resources	2
1.3. Bug Reporting Guidelines	2
2. Getting Started with the Peach Fuzzer Platform.....	4
2.1. Picking your fuzzing target.....	4
2.2. Installing Peach	5
2.3. Launching Peach.....	5
2.4. Selecting a Peach Pit.....	5
2.5. Specifying a Test Configuration.....	5
2.6. Running the Fuzzing Job	6
2.7. Interpreting Results	6
2.8. Fixing Issues and Re-running Test Cases.....	7
3. Introduction to Fuzzing.....	8
3.1. Dumb Fuzzing	10
3.2. Smart Fuzzing	11
3.3. When to Stop Fuzzing	12
4. Installation.....	15
4.1. Hardware Requirements	15
4.2. Downloading	16
4.3. Windows.....	20
4.4. Linux	21
4.5. macOS	26
4.6. License Activation	28
4.7. Enabling HTTPS And Authentication.....	31
5. What's new in Peach Fuzzer Professional v0.0.....	33
6. Running Peach	34
6.1. The Peach Web Interface	35
6.2. The Peach Command Line Interface	91
6.3. PeachAgent.....	98
6.4. Minset	99
6.5. Peach Multi-Node CLI Tool.....	102
6.6. Pit Tool - Ninja	107
7. Monitoring Recipes	110
7.1. Recipe: Monitoring a File Consumer (File Fuzzing)	110
7.2. Recipe: Monitoring a Linux Network Service Client	116

7.3. Recipe: Monitoring a Linux Network Service	128
7.4. Recipe: Monitoring a Network Device.....	139
7.5. Recipe: Monitoring a Windows Network Service Client.....	149
7.6. Recipe: Monitoring a Windows Network Service.....	162
8. Agents	176
8.1. Agent Channels	178
9. Monitors	180
9.1. Android Monitor.....	182
9.2. AndroidEmulator Monitor.....	187
9.3. APC Power Monitor	190
9.4. ButtonClicker Monitor.....	193
9.5. CanaKitRelay Monitor	198
9.6. CAN Capture Monitor	204
9.7. CAN Error Frame Monitor	205
9.8. CAN Send Frame Monitor.....	206
9.9. CAN Timing Monitor	210
9.10. CAN Threshold Monitor	212
9.11. CleanupFolder Monitor.....	215
9.12. CleanupRegistry Monitor (Windows)	218
9.13. CrashReporter Monitor (OS X).....	221
9.14. CrashWrangler Monitor (OS X).....	222
9.15. Gdb Monitor (Linux, OS X).....	225
9.16. GdbServer Monitor (Linux, OS X).....	231
9.17. IpPower9258 Monitor	237
9.18. LinuxCoreFile Monitor (Linux).....	242
9.19. Memory Monitor	244
9.20. NetworkCapture Monitor.....	246
9.21. PageHeap Monitor (Windows)	251
9.22. Ping Monitor	252
9.23. PopupWatcher Monitor (Windows).....	254
9.24. Process Monitor	255
9.25. ProcessKiller Monitor	258
9.26. Run Command.....	259
9.27. SaveFile Monitor.....	263
9.28. Serial Port Monitor.....	264
9.29. SNMP Power Monitor	271
9.30. Socket Monitor	275
9.31. SshCommand Monitor.....	277

9.32. SshDownloader Monitor	282
9.33. Syslog Monitor	284
9.34. TcpPort Monitor	288
9.35. Vmware Monitor	292
9.36. WindowsDebugger Monitor (Windows)	296
9.37. WindowsKernelDebugger Monitor (Windows)	300
9.38. WindowsService Monitor (Windows)	302
10. Reproducing Faults	303
10.1. Iteration Based Targets	303
10.2. Session Based Targets	304
10.3. How to Control the Automated Fault Reproduction	305
10.4. Replay the Fuzzing Session	305
11. Appendixes	306
11.1. Using Virtual Machines	306
11.2. Setting a Static IPv4 Address	307

1. Preface

This document is one of two books that together form the official documentation of Peach Fuzzer Professional v0.0. This documentation set is written by the Peach Fuzzer Professional team, and represents a concerted effort to fully document all of the Peach Fuzzer Professional features.

This book, Peach Fuzzer Professional User Guide, focuses on activities encountered in day-to-day usage of Peach. The organization of this book is predominantly task-driven. The second book, Peach Fuzzer Professional Developer Guide, focuses on developer needs such as advanced configurations, building custom fuzzing definitions, and extending the various areas of Peach.

As a User Guide, this book provides support for the user in daily issues that can otherwise impede tasks associated with fuzzing. It is not theoretical in nature. Nor is it designed to tackle advanced configurations, such as those used to fuzz embedded hardware or kernel drivers.

The primary audience for this guide is a daily user who is not a developer. The user is also not knowledgeable about XML.

Peach Fuzzer has been in active development through three major revisions since 2004. Until 2014, no complete documentation existed. This is the first edition of the User Guide, bearing witness that documenting Peach is an on-going effort. The majority of effort focuses on documenting the most common uses of Peach Fuzzer Professional.

1.1. What is Peach Fuzzer Professional?

Peach Fuzzer Professional (Peach) is a fuzzer for data consumers. Fuzzing is a software testing technique that introduces malformed data to parts of a computer system. How the tested item reacts to unexpected data becomes the source of security bugs.

Peach is a smart fuzzer that operates by performing the following actions:

- Understands the data structure and the flows of the test target
- Creates and feeds malformed data to the test target
- Monitors the test target to record interesting information when unintended or undesirable behavior occurs (monitors include debuggers and network packet sniffers)

Peach is a versatile product and has been used to fuzz a wide range of products and devices:

- web browsers (file consumers)
- web servers (network servers)
- mobile devices (such as Android iOS)
- robots
- SCADA systems

- Semiconductor chips

Because Peach can easily extend its interfacing and monitoring, it is the most adaptable fuzzer that exists.

1.2. Additional Resources

More information about Peach is available on the Web:

- [Peach Fuzzer website](#)
- [Current Pits and Pit Packs](#)
- [Peach User Forums](#)
- [Installing Peach Fuzzer Software video](#)

1.3. Bug Reporting Guidelines

Support for Peach Fuzzer is available in two ways:

- The Peach Forums site
- Using the Peach ticketing system to open a support ticket

1.3.1. Peach Forums

There are two sets of forums for Peach, the community forums and the professional forums. Both forums are hosted at <https://forums.peachfuzzer.com>.

Peach Fuzzer Professional users should access the private forums to receive support for the commercial versions of Peach. Responses on the commercial forums are prioritized over the public community forums.

To access the Peach Fuzzer Professional forums, follow these steps:

1. Register an account on the forums site.
2. Send an email to support@peachfuzzer.com with your license email and forum username.
3. Your account will be granted access to the commercial forums within 24 business hours.

Forums are monitored by the team at Peach Fuzzer, LLC; however, there is no guarantee of response time.

1.3.2. Support Tickets

You can open a support ticket by sending an email to support@peachfuzzer.com. You will receive an initial response within 24 business hours of opening the ticket. Peach support is available Monday through Friday. Peach support is not currently available on weekends or holidays.

When opening a ticket, please provide the following information in your email:

- Operating system(s) in use by Peach and any agents
- Exact version of Peach being used. This is available from the console output and in the *status.txt* log file
- Detailed description of the issue and expected behavior
- Console output using the *--trace* argument
- (if possible) the full Pit file and configuration files

2. Getting Started with the Peach Fuzzer Platform

This section provides a workflow that identifies the steps involved in setting up and running a fuzzing session. Stepping through the Peach fuzzing workflow is the quickest way to begin fuzzing.

The workflow in this section is conceptual. It tells you what needs to be done; not how to do it. Yet, when we can, we refer to the appropriate section of the Peach User Guide to perform that part of the workflow.

If you're new to fuzzing or want to learn how fuzzing works, read the section [Introduction to Fuzzing](#).



This approach applies to any step of the workflow that you are unfamiliar with or that you are curious about. Read the section. Then, come back to the workflow for the next task.

The workflow includes the following steps. The first item in the workflow is your test target. You should have this item locked down, as it makes the other steps of the workflow easier to get through.

1. [Picking Your Fuzzing Target](#)
2. [Installing Peach](#)
3. [Launching Peach](#)
4. [Selecting a Peach Pit \(fuzzing definition for your test target\)](#)
5. [Specifying a Test Configuration](#)
6. [Running the Fuzzing Job](#)
7. [Interpreting Results](#)
8. [Fixing Issues and Re-running the Faulting Test Cases](#)

The following is a restatement of the workflow that includes descriptions and appropriate references. After you've been through the workflow once or twice, you shouldn't need to access the details again.

2.1. Picking your fuzzing target

A fuzzing target can be just about anything; typically, the target accepts and dispenses data through security boundaries. Fuzzing target examples include:

- An application that consumes files
- A protocol service
- A protocol client

- A device
- Other

Get a good idea of what you intend to fuzz.

2.2. Installing Peach

The [Installation](#) section includes instructions for installing Peach Fuzzer on Windows, three Linux distributions, and OS X. The scope of material includes downloading distribution images with your Peach license through launching the platform to ensure a correct installation.

The installation also identifies other applications that Peach uses and where needed, provides installation on these items.

A video of the installation titled [Installing Your Peach Fuzzer Professional or Enterprise Software](#) is available at [vimeo.com](#).

2.3. Launching Peach

The [Starting the Peach Web Interface](#) section describes how to launch Peach and the Peach Web UI.



1. A command processor or command shell at a heightened/administrative level is used to launch Peach.
2. In the Windows environment, Peach automatically starts a browser and loads the Web UI.

For other environments, you need to copy the URL generated by Peach, launch a browser, and connect to Peach by pasting the URL into the browser address bar.

2.4. Selecting a Peach Pit

From the Peach Web Home page, click the “Library” icon in the group of icons on the left side of the page that are above the menu entries. This action lets you start with a pit installed on your system.

If you need a Pit that is not in your library, please contact the Peach Fuzzer Sales team at sales@peachfuzzer.com. For a list of available Pits, visit the [Peach Fuzzer](#) web site. From the Peach Fuzzer homepage, click on "Products" in the banner at the top of the page, then click the "Peach Pits" menu entry.

2.5. Specifying a Test Configuration

Specifying a configuration consists of setting variables specific to the selected Pit and defining how to monitor the test subject.

Variables are used to configure and target a Pit. This includes information such as IP addresses and port numbers for network Pits and any configuration required a specific Pit. Many of the variables you already know or can obtain quickly. Sometimes, the value of a variable depends on the monitoring setup, so some tweaking might be appropriate once you select the monitors to use.

Several monitoring recipes are included to provide base monitor configurations for the different fuzzing targets. Monitors focus on detecting faults, collecting data, and automating the test environment. The monitoring recipes also are great starting points for developing your own configurations that precisely monitor the test target for your needs. The recipes in this release consist of the following:

- [Monitoring a File Consumer \(File Format\)](#)
- [Monitoring a Linux Network Service](#)
- [Monitoring a Linux Network Client](#)
- [Monitoring a Network Device](#)
- [Monitoring a Windows Network Service](#)
- [Monitoring a Windows Network Client](#)

2.6. Running the Fuzzing Job

Peach Fuzzer typically runs hundreds of thousands of test cases. The Peach Fuzzer engine will generate an unlimited number of test cases, though this is not typically the most efficient use of resources. For more information, see [When to Stop Fuzzing](#). In each fuzzing job, Peach generates test cases randomly with fresh data values. The Peach platform weights the mutations used to generate the most faults.

Once your configuration is complete, or if you selected an existing configuration, you can start a fuzzing job by clicking Start at the bottom of the configuration page. For more information on starting and re-running fuzzing jobs, see the [Fuzzing Session](#) section for more information.



A fuzzing job runs indefinitely. It will stop running if you specify a value for Stop Test Case or if you manually click Stop on the Peach dashboard.

2.7. Interpreting Results

When the fuzzing job completes, Peach provides two sets of results:

- A report of the fuzzing job in a [.pdf](#) file.
- Online information that buckets the results, and provides drill-down capability

The report provides an overall status of the job and provides many views of the fuzzing job by focusing on the metrics used in fuzzing. The report is available by locating the appropriate fuzzing job entry,

then clicking on the document in the Action column at the right side of the entry.

Review the overall metrics and the buckets/categories of faults that occur. You should be able to see what worked well and where the issues were. After that, you'll need to investigate the faults, find the root cause of the fault, and then deal with it accordingly.

The online results focus on the faults that occurred during the fuzzing job. The faults are where you need to focus, and where Peach Fuzzer adds value to the SDL. You can access faults from the Dashboard page or from the home page, where you can access all of the stored fuzzing job results.

See the [Faults](#) section for a description of the information that Peach captures when a fault occurs; visit the [Metrics](#) section for a description of how Peach Fuzzer provides meaningful views into the fuzzing job by rolling up test case results.

2.8. Fixing Issues and Re-running Test Cases

The last step is to address the faults/issues uncovered during fuzzing, and to verify the fixes.

- Address the faults

This item is for the developer, who needs to edit the code where the fault occurred. Use your normal debugging practices here.

- Verifying fixes of issues

When the fixes are in place, you can re-run a fuzzing job in whole or in part by selecting the Pit Configuration, and then specifying the Seed value of the fuzzing job that you found the fault, the Start Test Case (optional), and the End Test Case (optional). For more information, see [Re-running a Fuzzing Job](#).



Specifying the same seed value as in the original fuzzing job ensures that the same test cases are run, in the same sequence and with the same data as in the original fuzzing job.

3. Introduction to Fuzzing

Welcome to the world of fuzzing!

What is fuzzing?

Fuzzing is the art of performing unexpected actions that result in target misbehavior.

What is our goal when we fuzz?

Our goal is to find new, previously unknown security vulnerabilities. Finding vulnerabilities is limited only by our ability to detect them automatically.

The most common target for fuzzing is a data consumer such a network service or a web browser. For a data consumer, the unexpected actions consist of sending data to the consumer that is malformed in some way. When the target consumes the fuzzed data, the malformed data could trigger a target vulnerability. The likelihood that a single change causes a vulnerability might not be high, but when the act of sending fuzzed data repeats over and over, a very good chance exists that the target will misbehave in some interesting manner.

Fuzzers submit hundreds, thousands, even millions of data samples malformed in some way; fuzzers call the malformed data mutations. Once written, fuzzers typically run for long periods of time finding more and more vulnerabilities, called faults.

Consider the following C structure and corresponding data shown in hex:

```
struct Header
{
    ushort ver;
    ushort len; // 33
    char* data; // 33
}

char* msg_data = malloc(len);
strcpy(msg_data, data);
```

59	4D	00	21	41	4F	41	42
44	B1	45	55	56	0F	43	44
45	50	2D	4A	38	39	C0	80
62	68	5F	70	65	61	63	68
5F	66	75	7A	00	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	00	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	4D	44	55	2D

The data is parsed by the data consumer into the structure. The data and structure are color coded to

show which parts of the data are loaded into each member of the structure *Header*. Following the structure, two lines of code occur that use the *Header* structure. Looking at that code, we can identify several issues that could likely lead to crashing our program that we could trigger by changing our input data.

The first thing we could do is change the *len* in our data from `0x0021` to `0x0001`. This would cause the memory allocation to return a smaller amount of memory than what is needed for copying data using the *strcpy* function. Once *strcpy* executes, we will have written past the end of *msg_data*, possibly causing the target process to crash.

```
struct Header
{
    ushort ver;
    ushort len; // 1
    char* data; // 33
}

char* msg_data = malloc(len);
strcpy(msg_data, data);
```

59	4D	00	01	41	4F	41	42
44	B1	45	55	56	0F	43	44
45	50	2D	4A	38	39	C0	80
62	68	5F	70	65	61	63	68
5F	66	75	7A	00	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	00	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	4D	44	55	2D

Another change we could make to the data is to remove or change the null byte that terminates the string *data*. In C, string functions operate on data up to the null character. If the null byte is changed, the *strcpy* function copies more than 33 bytes into *msg_data*. This could cause the target process to crash.

```

struct Header
{
    ushort ver;
    ushort len; // 33
    char* data; // 74
}

char* msg_data = malloc(len);
strcpy(msg_data, data);

```

59	4D	00	21	41	4F	41	42
44	B1	45	55	56	0F	43	44
45	50	2D	4A	38	39	C0	80
62	68	5F	70	65	61	63	68
5F	66	75	7A	55	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	00	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	4D	44	55	2D

The two changes we made to trigger two similar but different bugs in the code is something we look to automate with fuzzing. Taking this input data and randomly changing a byte to a random value, would, over the course of many attempts, eventually find both of these issues.

One of the goals with fuzzing is to find as many faults in targets with the least amount of human time.

3.1. Dumb Fuzzing

Several types of fuzzing technology exist; one of the most common technologies is dumb fuzzing. Dumb fuzzers are a class of fuzzers that operate with limited or no information or understanding of the target or data being consumed.

Dumb fuzzers are popular because they take very little effort to get running and can produce good results with certain targets. Since they lack any knowledge of the data being consumed, they have limited practical use.

The most common mutation performed by dumb fuzzers is bit flipping. Bit flipping selects a part of data to change and modifies it in a simple manner. One might make a single change, or multiple changes depending on the fuzzer. The resulting data is then sent to the data consumer to see if it causes the data consumer to misbehave (such as crash).

Dumb fuzzing makes a great starting point when first fuzzing as it is the easiest method to use. However, in many cases, dumb fuzzing does not work. One example is the use of checksums in the data format. Checksums validate integrity of the data during transmission or storage. Any change to the data will result in a checksum that does not match. This is shown in the following image:

CRC-32:
98 1B A7 75



New CRC-32:
DA 38 80 19

59	4D	53	47	00	0F	00	00
02	B1	00	55	00	00	00	00
00	50	2D	4A	38	39	C0	80
62	68	5F	FF	65	61	63	68
5F	66	75	7A	7A	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	35	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	98	1B	A7	75

The highlighted FF in the data stream is a byte that was changed during fuzzing. The new checksum value does not match the one that already exists in the file. If the checksum is validated by the data consumer, it could be rejected early on, limiting the number of faults that can be found.

Smart fuzzers allow updating the data to correct for things like checksums, encryption, encoding, and compression so that the data passes through the initial system checks and is process by the data consumer.

Peach makes it easy to shift from dumb fuzzing into smart fuzzing.

3.2. Smart Fuzzing

Smart fuzzers are a class of fuzzers that operate with some knowledge and understanding of the target, and of the data being consumed. The amount of knowledge depends on the fuzzer being used.

A typical smart fuzzer does the following things:

- Understand the data format being consumed by the target application
- Monitor the target for fault conditions
- Modify the data to gain better coverage or to increase the ability to detect certain types of issues

Data understanding includes:

- Type information (string, integer, byte array)
- Relationships between fields in the data (length, offset, count)
- Ability integrity fields such as a checksum or a CRC

At this level, understanding the data structures and data types allows the fuzzer to make more informed changes (mutations) to the data. Smart fuzzers use this understanding level to find more bugs.

Smart fuzzers can control and monitor the fuzzing environment. Environment and instrumentation controls start all the components of the system so they are ready to fuzz. On a faulting condition, they reset the environment to a known, good state. Smart fuzzers can detect a faulting condition and collect any interesting data in the system at the time of the fault (including output from a debugger, a network capture, or files on the file system), and log the data for later review. High-quality smart fuzzers can run unattended for long periods of time and capture enough information to allow a resource to reasonably reproduce and investigate the faults that occurred.

Smart fuzzers also perform bug bucketing and basic risk analysis. Fuzzing commonly finds the same issue multiple times during a long test session. Bucketing is an industry term for cataloging similar, and possible duplicate, issues into a "bucket". Bucketing is typically done at a major and minor level.

- Major differences are generally distinct issues. The bucket for a major issue might associate with a specific processor family, operating system, or monitor name.
- Minor differences that generate faults mean that the issues might be identical, or are probably similar; yet, issues in minor buckets are worth reviewing to ensure the issues have the same root cause.

Along with buckets, initial risk analysis allows you to direct your attention first on higher risk faults before spending time on lower risk issues. Risk analysis is not always possible, but is useful when it can be performed.

3.3. When to Stop Fuzzing

A fuzzing bar sets requirements for a fuzzing job and answers the question, "How long do we fuzz?"

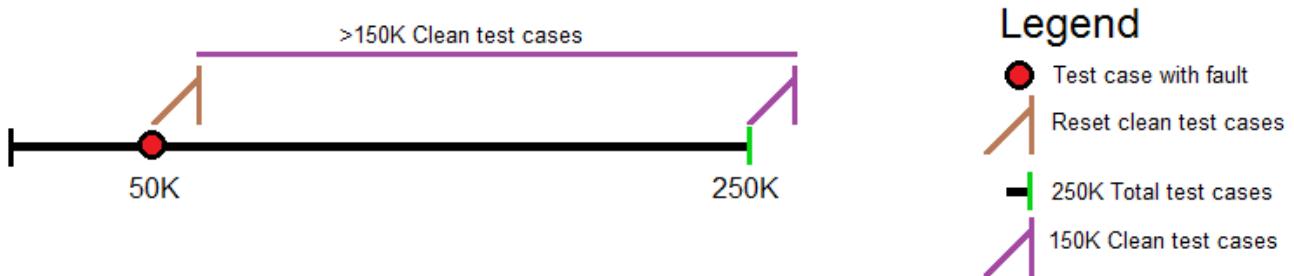
Here is one set of exiting criteria that consists of two requirements:

- The fuzzer must run a set number of test cases.
The number of test cases performed must meet or exceed the specified threshold.
- The fuzzer must have at least Y consecutive clean test cases.
A clean test case generates zero new faults.

As soon as both requirements occur, fuzzing can stop.

For example, when fuzzing a new product, the release criteria might be set to 250,000 total fuzzing test cases on the product and yield 150,000 consecutive clean test cases.

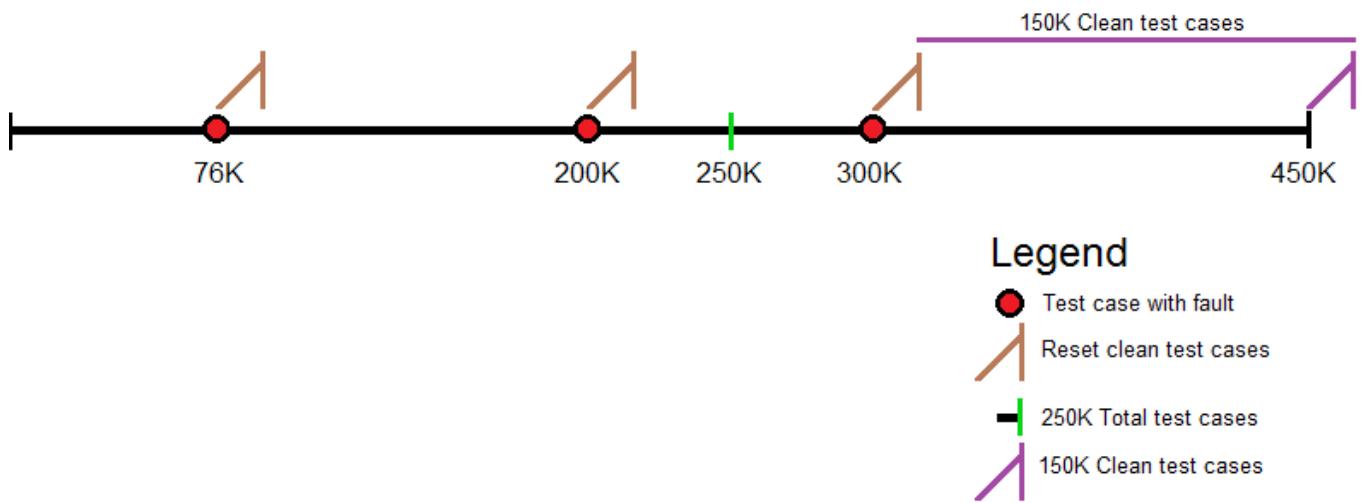
The following diagram shows one way of meeting this criteria and focuses on the number of test cases performed, test case failures, and the number of consecutive clean test cases.



A few items are worth noting:

- The total number of test cases performed is 250K.
- The number of consecutive clean test cases is 200K, surpassing the requirement of 150K clean test cases. The last 50K test cases were needed to meet the first requirement.
- The count of consecutive clean test cases reset to zero when a new fault was found at the 50,000th test case.

The next diagram shows another way of meeting this criteria.



Again, a few of items are worth noting:

- The total number of test cases performed is 450K.
- The number of consecutive clean test cases is 150K.
- The count of the consecutive clean test cases restarted three times due to new faults occurring in test cases: after 76K, 200K, and 300K.

The requirements used in the previous example are reasonable for a first release of a product. However, each successive version of a product should become more and more stable. This can be realized by increasing the total number of test cases performed by the fuzzer, and the number of consecutive clean test cases that result from fuzzing. The following table provides increasing requirements as a product matures.

Year	Required Iterations	Clean Iterations
Year 1	250,000	150,000
Year 2	500,000	250,000
Year 4	High 750,000 Medium 500,000 Low 500,000	High 500,000 Medium 250,000 Low 250,000
Year 5	High 1,000,000 Medium 750,000 Low 500,000	High 750,000 Medium 500,000 Low 250,000



What was once the Software Development Life Cycle (SDLC) has evolved into the Secure Development Lifecycle (SDL) to protect corporate assets from zero-day attacks. Fuzzing is part of the SDL, specifically in the security push and the Verification Phase. Peach Fuzzer can operate on non-executable file formats, protocol stacks, and data that originates from a lesser-privileged trust boundary.

4. Installation

The following list contains links to the sections that describe in detail how to download, install and activate Peach Fuzzer.

Recommended Hardware

This section describes the minimum and recommended hardware requirements for common fuzzing scenarios.

Product Download

This section contains the steps required to download the product for your desired platform.

Product Installation

This section lists the software prerequisites, OS specific configuration, and steps for installing the product on each of the three supported platforms:

- [Windows](#)
- [Linux](#)
- [OSX](#)

Product Activation

The list below contains links to the steps for activating the different types of Peach Fuzzer licenses.

- [Usage Based \(Online Synchronization\)](#)
- [Usage Based \(Offline Synchronization\)](#)
- [Node Locked](#)
- [Enterprise](#)

Optional Configuration

The list below contains links to optional post-install configurations.

- [Enabling HTTPS And Authentication](#)

4.1. Hardware Requirements

The following are generic hardware recommendations. Adjust based on your needs.

4.1.1. Local Target

When fuzzing a local target (software running on the same machine as Peach), additional resources are required for the target process.

Target Type	Architecture	Cores	Ram	Disk
Network	64-bit	4	8GB	60GB SSD
File	64-bit	4	16GB	60GB SSD
Other	64-bit	4	8GB	60GB SSD

4.1.2. Remote Target

Remote target fuzzing occurs when the target is not located on the machine running Peach.

	Architecture	Cores	Ram	Disk
Minimum	64-bit	2	4GB	60GB Any
Recommended	64-bit	2	8GB	60GB SSD



It's possible to run Peach on 32-bit systems, but it's not recommended as it places severe limits on memory usage (max 2GB).

4.2. Downloading

The first step of the installation is to download the Peach Fuzzer distribution files from the Peach download site. Once the appropriate files have been downloaded, follow the instructions for your specific operating system found in the next section.

User Account Download

If you were assigned an account with a username/password, follow these instructions to sign in and download Peach Fuzzer.

Enterprise Download

If you were provided an enterprise license file, follow the instructions to sign in and download Peach Fuzzer.

4.2.1. User Account Download

1. When your Peach Fuzzer welcome email arrives, click the link to reset your initial password.
2. Navigate to <https://portal.peachfuzzer.com> with your preferred web browser.
3. At the login prompt under the Portal Login section, enter the new username/password that was recently reset and click **Sign In**.

PEACHTECH

Portal Login

Username

Password

Sign In

Enterprise Users

If you were provided a license file 'Peach.license', upload that file here to log in.

No file chosen



Contact support@peach.tech if you need your account password reset.

4. On the [Downloads](#) page, select the Peach Fuzzer release version and operating system to install.
 - a. Choose a release version from the items on the left side.
 - b. Click the download icon on the right side after deciding which OS and architecture is needed.

The screenshot shows the PeachTech website's navigation bar with 'Downloads' highlighted. The main content area displays the 'PEACH FUZZER' section, where 'v4.3' is selected (indicated by a red box). A list of files for v4.3.284 is shown, including various PDF guides and installation files for different operating systems and architectures. The 'PEACH API SECURITY' section is also visible on the left.

File	Size
Client_Peach_Trial_Guide-ICS.pdf	1 MB
Client_Peach_Trial_Guide-Network.pdf	1 MB
Hosted_Peach_Trial_Guide-Fileformat.pdf	1 MB
Hosted_Peach_Trial_Guide-Healthcare.pdf	1 MB
Hosted_Peach_Trial_Guide-ICS.pdf	1 MB
Hosted_Peach_Trial_Guide-Network.pdf	1 MB
Peach_Pro_Changes.pdf	0 MB
Peach_Pro_Installation_Guide.pdf	1 MB
Peach_Pro_User_Guide.pdf	7 MB
peach-pro-4.3.284-linux_x86_64_release.zip	69 MB
peach-pro-4.3.284-linux_x86_release.zip	69 MB
peach-pro-4.3.284-osx_release.zip	73 MB
peach-pro-4.3.284-sdk.zip	36 MB
peach-pro-4.3.284-win_x64_release.zip	83 MB
peach-pro-4.3.284-win_x86_release.zip	80 MB

5. If your organization has multiple entitlements, you may need to select an Activation ID that corresponds to the license the download should be tied to. This selection page will not be displayed

if there is only one entitlement for your organization. Contact licensing@peach.tech for more information if you are unsure which Activation ID to select.

6. After a few moments, an End User License Agreement acceptance page appears. Click **I ACCEPT** to continue.

The screenshot shows a blue header bar with the PeachTech logo and navigation links: Solutions, Forums, Tickets, Downloads, Licensing, and Logout. Below the header is a white content area with a light gray border. At the top of the content area, it says "END-USER LICENSE AGREEMENT: PEACH FUZZER™". Below this, a paragraph of text states: "BY DOWNLOADING, ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES, YOU AUTOMATICALLY ACKNOWLEDGE, ACCEPT AND AGREE TO ALL THE TERMS OF THE PEACH FUZZER™ END-USER LICENSE AGREEMENT LOCATED AT <https://www.peach.tech/eula/flex/>. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, YOU MAY NOT DOWNLOAD, ACCESS OR OTHERWISE USE THE SOFTWARE AND ACCOMPANYING FEATURES." Underneath this text is a section titled "CONTACT INFORMATION" with the instruction: "If you have any questions about this EULA, or if you want to contact Peach Fuzzer, LLC for any reason, please direct all correspondence to: contact@peach.tech". At the bottom of the content area are two buttons: a blue "I ACCEPT" button and a dark gray "Cancel" button. At the very bottom of the page, outside the main content area, is a small copyright notice: "Copyright © 2019 Peach Fuzzer, LLC . All rights reserved."

7. The download will begin. Depending on your network connection, this could take a few minutes.

4.2.2. Enterprise Download



You need a copy of your Peach Fuzzer license on your system to perform the download.

1. Using a web browser, navigate to <https://portal.peachfuzzer.com>

PEACHTECH

Portal Login

Username
Password

Sign In

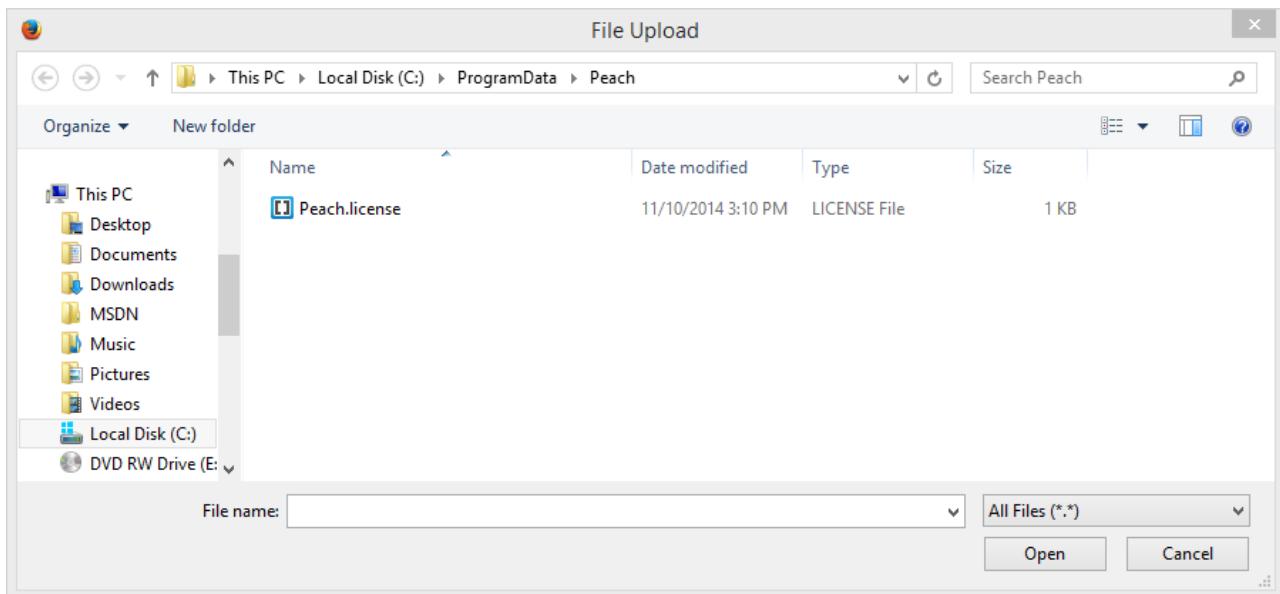
Enterprise Users

If you were provided a license file 'Peach.license', upload that file here to log in.

No file chosen

Sign In with License

- a. Click the **Choose File** button. The upload dialog display allows you to select the **Peach.license** file.



- b. Navigate to the location of the license
 - c. Select the license (**Peach.license**)
 - d. Click **Open** to return to the download home page.
2. Upon returning to the Peach download home page, click **Sign In with License**.
 3. On the **Downloads** page, select the Peach Fuzzer release version and operating system to install.
 - a. Choose a release version from the items on the left side.
 - b. Click the download icon on the right side after deciding which OS and architecture is needed.

PEACH FUZZER

[v4.3](#)[v4.2](#)

PEACH API SECURITY

[v1.5](#)[v1.4](#)[v1.3](#)[v1.2](#)[v1.1](#)[v1.0](#)

v4.3.284 (Thursday, November 15, 2018)

 Client_Peach_Trial_Guide-ICS.pdf	1 MB
 Client_Peach_Trial_Guide-Network.pdf	1 MB
 Hosted_Peach_Trial_Guide-Fileformat.pdf	1 MB
 Hosted_Peach_Trial_Guide-Healthcare.pdf	1 MB
 Hosted_Peach_Trial_Guide-ICS.pdf	1 MB
 Hosted_Peach_Trial_Guide-Network.pdf	1 MB
 Peach_Pro_Changes.pdf	0 MB
 Peach_Pro_Installation_Guide.pdf	1 MB
 Peach_Pro_User_Guide.pdf	7 MB
 peach-pro-4.3.284-linux_x86_64_release.zip	69 MB
 peach-pro-4.3.284-linux_x86_release.zip	69 MB
 peach-pro-4.3.284-osx_release.zip	73 MB
 peach-pro-4.3.284-sdk.zip	36 MB
 peach-pro-4.3.284-win_x64_release.zip	83 MB
 peach-pro-4.3.284-win_x86_release.zip	80 MB

4. After a few moments, an End User License Agreement acceptance page appears. Click **I ACCEPT** to continue.

END-USER LICENSE AGREEMENT: PEACH FUZZER™

BY DOWNLOADING, ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES, YOU AUTOMATICALLY ACKNOWLEDGE, ACCEPT AND AGREE TO ALL THE TERMS OF THE PEACH FUZZER™ END-USER LICENSE AGREEMENT LOCATED AT <https://www.peach.tech/eula/flex/>. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, YOU MAY NOT DOWNLOAD, ACCESS OR OTHERWISE USE THE SOFTWARE AND ACCOMPANYING FEATURES.

CONTACT INFORMATION

If you have any questions about this EULA, or if you want to contact Peach Fuzzer, LLC for any reason, please direct all correspondence to:
contact@peach.tech

I ACCEPT**Cancel**

Copyright © 2019 Peach Fuzzer, LLC . All rights reserved.

5. The download will begin. Depending on your network connection, this could take a few minutes.

4.3. Windows

Peach is officially supported on the following Windows® Operating Systems:

- Windows 7 SP1 (x86 and x64)
- Windows 8 (x86 and x64)

- Windows 8.1 (x86 and x64)
- Windows 10 (x64)
- Windows Server 2008 SP2 (x86 and x64)
- Windows Server 2008 R2 SP1 (x64)
- Windows Server 2012 (x64)
- Windows Server 2012 R2 (x64)

The only required software is the Microsoft .NET Framework v4.5.

1. Download and install the [Microsoft .NET Framework v4.5.2 \(Installer\)](#).
2. Install the [Microsoft Debugging Tools for Windows](#) (optional).



This is only required if you want to use a debugger to detect crashes in fuzzed programs.

3. Install [Wireshark](#) (optional).



This is only required if you want to collect network captures during fuzzing runs.

4. Unzip the Peach distribution to the appropriate folder. The file is a zip file with the extension `.zip`. Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-win_x64_release.zip`.
5. When fuzzing, many security products (such as anti-virus programs) can interfere or slowdown fuzzing. For network fuzzing, make sure none of the network or host-based network intrusion detection systems (IDS) are running. For file fuzzing, disable anti-virus software; or mark Peach, the target application, and any directories that might have files used in fuzzing, as out of scope for real time monitoring.

4.4. Linux

Peach is supported on three distributions of Linux; Peach may run on other Linux distributions, but are not officially supported. This section provides instructions for installing Peach on the following supported Linux systems, and includes a checklist for installing Peach on other Linux systems:

- Ubuntu/Debian Linux
- Redhat Enterprise Linux (RHEL and CentOS)
- SUSE Enterprise Linux (SLES)

4.4.1. Ubuntu/Debian Linux

When installing Peach on an Ubuntu or Debian Linux system, the operating system is ready for Peach without modification. The installation starts with the Mono .NET runtime, then Peach. If you want to

attach a debugger to a target process, install GDB when the Mono installation completes.

Peach Fuzzer, LLC, recommends using Ubuntu Linux version 16.04 LTS, and Mono .NET runtime version 4.8.1 from the mono project. The mono project has [apt](#) packages for Ubuntu.

Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

 Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

Some Linux kernel versions have known issues with the Mono runtime. When using Ubuntu 14.04 LTS, avoid using kernel versions 3.13.0-48 through 3.13.0-54 inclusive. Version 14.04 LTS might require that you update the Linux kernel. If so, perform the following to update the kernel: [sudo apt-get install linux-image-generic](#).

The Peach installer checks for compatibility and alerts the user if an incompatibility has been detected.

The following steps will prepare Peach to run properly:

1. Install the latest [mono-complete](#) package.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF  
  
echo "deb http://download.mono-project.com/repo/debian wheezy/snapshots/4.8.1.0 main"  
| sudo tee /etc/apt/sources.list.d/mono-xamarin.list  
  
sudo apt-get update  
  
sudo apt-get install mono-complete
```

2. Install libpcap using the following command:

```
sudo apt-get install libpcap
```

3. Optionally, install the GNU Debugger (GDB) to enable debugging of local processes.

```
sudo apt-get install gdb
```

4. Unzip the Peach distribution to the appropriate folder. The file is a zip file with the extension [.zip](#).

Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-linux_x86_64_release.zip`.

4.4.2. Redhat Enterprise Linux (RHEL and CentOS)

Installing Peach on a RHEL CentOS platform requires additional steps. Begin by installing Extra Packages for Enterprise Linux (EPEL), followed by the Mono package and Peach.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project. The mono project has `yum` packages for RHEL and CentOS distributions.

 Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

 The following Mono installation steps are taken from the [Mono Project](#).

The following steps provide the needed details:

1. Install `yum-utils` using the following command:

```
sudo yum install yum-utils
```

2. Install Extra Packages for Enterprise Linux (EPEL) using the following command:

```
sudo yum install epel-release
```

3. Import the GPG signing key for the mono package using the following command. Note the long search key:

```
sudo rpm --import  
"http://keyserver.ubuntu.com/pks/lookup?op=get&search=0x3FA7E0328081BFF6A14DA29AA6A19B  
38D3D831EF"
```

4. Add and enable the mono project repository for CentOS using the yum configuration manager:

```
sudo yum-config-manager --add-repo http://download.mono-project.com/repo/centos/
```

5. Install the latest version of Mono using the following command:

```
sudo yum install mono-complete-4.8.1.0-0.xamarin.1
```

6. Install libpcap using the following command:

```
sudo yum install libpcap
```

7. Unzip the Peach distribution to the appropriate folder. The file is a zip file with the extension **.zip**. Use the filename that begins with **peach-pro** and contains the appropriate architecture for your system, such as **peach-pro-0.0.0-linux_x86_64_release.zip**.

If you receive an error regarding libMonoPosixHelper the **/etc/mono/config** file may need to be edited. To edit locate a line that looks like the following (the path may be different):

```
<dllmap dll="MonoPosixHelper" target="/usr/lib/libMonoPosixHelper.so" os="!windows" />
```



Once found use the Linux **find** command to locate the shared library:

```
find /usr -name "*libMonoPosixHelper.so"
```

And finally, update the **/etc/mono/config** entry to the correct path.

i For more information, see the following resources:
* <http://www.mono-project.com/docs/getting-started/install/linux#centos-fedora-and-derivatives>
* https://fedoraproject.org/wiki/EPEL#How_can_I_use_these_extra_packages.3F

4.4.3. SUSE Enterprise Linux (SLES)

To install Peach on a SUSE Enterprise Linux platform, use the 1-click SUSE mono-complete installation file. If you want to attach a debugger to a target process, install GDB when the Mono installation completes.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project. The mono project has packages for SLES distributions.

 Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

The following steps provide the needed details:

1. Import the GPG signing key for the mono package using the following command. Note the long search key:

```
sudo rpm --import  
"http://keyserver.ubuntu.com/pks/lookup?op=get&search=0x3FA7E0328081BFF6A14DA29AA6A19B  
38D3D831EF"
```

2. Add and enable the mono project repository using the zypper configuration manager:

```
sudo zypper ar -f http://download.mono-project.com/repo/centos/ mono
```

3. Install the latest supported version of Mono using the following command:

```
sudo zypper in mono-complete=4.8.1.0-0.xamarin.1
```

4. Install libpcap using the following command:

```
sudo zypper in libpcap
```

5. Optionally, install the GNU Debugger (GDB) for debugging local processes.

```
sudo yum install gdb
```

6. Unzip the Peach binary distribution to the appropriate folder. The file is a zip file with the extension **.zip**. Use the filename that begins with **peach-pro** and contains the appropriate architecture for your system, such as **peach-pro-0.0.0-linux_x86_64_release.zip**.

If you receive an error regarding libMonoPosixHelper the `/etc/mono/config` file may need to be edited. To edit locate a line that looks like the following (the path may be different):



```
<dllmap dll="MonoPosixHelper" target="/usr/lib/libMonoPosixHelper.so" os="!windows" />
```

Once found use the Linux `find` command to locate the shared library:

```
find /usr -name "*libMonoPosixHelper.so"
```

And finally, update the `/etc/mono/config` entry to the correct path.

4.4.4. Other Linux Distributions

For other Linux versions, the installation steps are a checklist, not specific commands. The checklist follows:

1. Install the Mono runtime. Version 4.8.1 is recommended.
2. Unzip the Peach distribution to an appropriate folder. The file is a zip file with the extension `.zip`. Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-linux_x86_64_release.zip`.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project.



Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

4.5. macOS

To install on macOS, follow the installation steps provided below. Installation will require installing the Mono .NET runtime, then Peach. To enable support for the `CrashWrangler` monitor, install CrashWrangler and Xcode. Note that installing CrashWrangler is optional; it is only needed when running the target locally.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project.



Peach will not run with Mono version 5.0 or newer due to incompatibilities with our Python runtime. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

1. Install the [Mono package](#).
2. Unzip the Peach distribution to an appropriate folder. The file is a zip file with the extension [.zip](#). Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-osx_release.zip`.
3. Install CrashWrangler.

CrashWrangler **MUST** be compiled on each macOS machine. Peach includes the CrashWrangler source files in the peach distribution. Here are instructions to install and compile CrashWrangler from the peach zip.

- a. Ensure XCode is installed.
- b. Open [Terminal.app](#).
- c. Navigate to the folder where you extracted `peach-pro-0.0.0-osx_release.zip`.
- d. Finish installing CrashWrangler using the following commands.

```
# Navigate to the folder containing CrashWrangler distribution
cd CrashWrangler

# Extract CrashWrangler sources
unzip 52607_crashwrangler.zip

# Navigate to the folder containing the extracted CrashWrangler sources
cd crashwrangler

# Compile CrashWrangler
$ make

# Ensure installation directory exists
sudo mkdir -p /usr/local/bin

# Install CrashWrangler
sudo cp exc_handler /usr/local/bin

# Navigate to the folder containing peach
cd ../../

# Verify CrashWrangler can run
exc_handler
```

4.6. License Activation

The list below contains links to the steps for activating the different types of Peach Fuzzer licenses.

Usage Based (Online Synchronization)

The most common method for activating a usage based license. A Cloud License Server will be automatically provisioned and managed for you. However, Peach Fuzzer will require a persistent connection to the Internet.

Usage Based (Offline Synchronization)

Users who wish to use Peach Fuzzer in an offline without having access to the Internet can deploy a Local License Server onsite to provide offline activation and synchronization of licensing information.

Node Locked

The license is tied to the physical machine running Peach Fuzzer. No license server is required and internet connectivity is only needed for activation.

Enterprise

No activation is required for enterprise customers.

4.6.1. Usage Based (Online Synchronization)

A Cloud License Server provides functionality for serving and monitoring a counted pool of licenses for Peach Fuzzer. A persistent connection to the Internet is required so that usage data can be uploaded to the Cloud License Server while a Peach Fuzzer job is running. Peach Fuzzer will automatically activate the first time it is run.



If a proxy server is required to connect to the Internet, it must be configured as described below.

Windows Proxy configuration

On Windows, the system proxy setting is the correct way to configure the proxy peach will use to connect to the licensing server.

Windows 10

The system proxy settings are configured at Settings > Network & Internet > Proxy. From there you will be able to enter the IP and Port of the proxy server.

Windows 8

The system proxy settings are configured at PC Settings > Network Proxy. From there you will be able to enter the IP and Port of the proxy server.

Windows 7

The system proxy settings are configured through the Internet Settings dialog. Open the Internet Options window located at Control Panel > Network and Internet > Internet Options.

1. Click the "Connections" tab at the top of the Internet Options window.
2. Click the "LAN Settings" button at the bottom of the window.
3. Click the "Advanced" button under Proxy Server will allow you to change advanced settings and enable a manual proxy server.

Linux Proxy Configuration

The proxy configuration on Linux is controlled via two environment variables `http_proxy` and `https_proxy`. Ensure both variables are set prior to starting Peach Fuzzer.

```
export http_proxy=http://xxxxx  
export https_proxy=https://xxxxx
```

4.6.2. Usage Based (Offline Synchronization)

The Local License Server provides functionality for serving and monitoring a counted pool of licenses for Peach Fuzzer. Users who wish to use Peach Fuzzer without having access to the Internet can deploy

a Local License Server onsite to provide offline activation and synchronization of licensing information.

The instructions for installing and activating a Local License Server can be found on the [Peach Portal](#) by navigating to the "Licensing" tab and clicking the "Local License Server" button for the desired license.

The screenshot shows the Peach Portal interface with a blue header bar containing the logo, navigation links (Solutions, Forums, Tickets, Downloads, Licensing), and a Logout link. The main content area displays a "Peach Fuzzer - Premium" license entry. It includes fields for Activation ID (8264-92dc-cde5-48c7-9f3a-db62-bb15-fad3), Expiration Date (2/1/2021), License Type (Local License Server), and a Products section listing Peach Fuzzer - Premium, PeachPitPack-Premium, and Peach Fuzzer - Test Cases (Limited). At the bottom of the card are two buttons: "View details »" and "Local License Server Instructions", with the second button being highlighted with a red box. A copyright notice at the bottom of the page reads "Copyright © 2019 Peach Fuzzer, LLC . All rights reserved."

4.6.3. Node Locked

No license server is required, as node locked licenses are tied to an individual machine. Peach Fuzzer will automatically activate the first time it is run.

If your license has changed and you want Peach Fuzzer refresh its license, run the following command:

```
peach --activate
```

If you wish to move your license to a new machine, you must first deactivate the existing instance by running the following command:

```
peach --deactivate
```



Peach Fuzzer requires an internet connection in order to perform activation and deactivation. Once activated, no further internet connectivity is required.

4.6.4. Enterprise

No activation is required for enterprise customers. The enterprise license is automatically embedded in the Peach Fuzzer download.

4.7. Enabling HTTPS And Authentication

Peach Fuzzer Professional uses a web interface for configuration and control of the fuzzing engine. By default, the web interface is accessible with no encryption (SSL/TLS) and no authentication. If the use of HTTPS or authentication is required, a reverse proxy (apache/nginx/traefik) can be used to provide both SSL/TLS and authentication.

4.7.1. Reverse Proxy with NGINX

The following steps will configure NGINX as a reverse proxy for Peach adding TLS and authentication:

1. Install nginx using your Linux package manager
2. Create required key. For self signed keys this [online self-signed certificate generator](#) can be used.
3. Install the included NGINX configuration file to /etc/nginx/sites-available/peach
4. Install certificate and key and update configuration file if needed
5. Create .htpasswd with username/passwords replacing **USERNAME** with your username

```
sudo sh -c "echo -n 'USERNAME:' >> /etc/nginx/.htpasswd"
sudo sh -c "openssl passwd -apr1 >> /etc/nginx/.htpasswd"
```

6. Add a firewall rule to block external access to Peach's port 8888. Make sure this rule is enabled on bootup.
7. Link /etc/nginx/sites-available/peach to /etc/nginx/sites-enabled/peach
8. Restart NGINX and verify configuration is working

NGINX Configuration File

```

# HTTPS server
#
server {
    listen 443;
    server_name localhost;

    root html;
    index index.html index.htm;

    ssl on;
    ssl_certificate /etc/ssl/certs/ssl.crt;
    ssl_certificate_key /etc/ssl/private/ssl.key;

    ssl_session_timeout 5m;

    ssl_protocols TLSv1.1 TLSv1.2;
    ssl_ciphers "HIGH:!aNULL:!MD5 or HIGH:!aNULL:!MD5:!3DES";
    ssl_prefer_server_ciphers on;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        #try_files $uri $uri/ =404;
        # Uncomment to enable naxsi on this location
        # include /etc/nginx/naxsi.rules

        auth_basic "Restricted";
        auth_basic_user_file /etc/nginx/.htpasswd;
        proxy_pass http://127.0.0.1:8888/;
    }
}

```

5. What's new in Peach Fuzzer Professional v0.0

This section provides a high-level view of the changes introduced this release of Peach Fuzzer Professional.

6. Running Peach

Peach Fuzzer Professional includes a number of executable files. In most instances, using the web interface will meet your needs. Peach can also be used from the command line. This includes using the Peach Web Interface, which launches by running Peach from the command line without any parameters or switches.

The following list identifies the support applications included with Peach Fuzzer Professional.

Program	Executable	Description
Peach Web Interface	Peach.exe	The Peach Web Interface for Peach Fuzzer.
Peach Command Line	Peach.exe	The Peach Command Line Interface for Peach Fuzzer.
Peach Agent	PeachAgent.exe	The Agent process for Peach Fuzzer.
Minset	PeachMinset.exe	Find the minimal set of sample files for use during fuzzing with the greatest code coverage for a given target.
PitTool - Sample Ninja	PitTool.exe	Create a sample ninja database.
Peach Multi-Node CLI Tool	sdk\tools\peachcli	Control and coordinate multiple Peach instances.

6.1. The Peach Web Interface

The Peach Web Interface is an interactive interface to Peach Fuzzer Professional that simplifies monitoring of local and remote fuzzing jobs. Using Peach Web Interface, you can select, configure, and run fuzzing definitions (Pits). The Peach Web Interface is operating system agnostic. This means that you can use the same interface to run Peach, whether on Windows, Linux, or OS X. When a Pit is running, you can view the state of the fuzzing job and see faults that result.

The Peach Web Interface works with Pits from the included Pit Library. With a little configuration, the Pits will be ready to run. Once you settle on a Pit that you want to use, you can configure the Pit to detect faults, to collect data, or to automate the fuzzing session.

6.1.1. Peach Web Interface Installation Requirements

The Peach Web Interface uses modern web technologies, such as HTML 5. Yet, Peach Web Interface requires two things to run:

- JavaScript, enabled in your web browser
- A supported browser
 - Internet Explorer: version 9 and newer
 - Safari: version 6 and newer
 - Firefox: version 4 and newer
 - Chrome: version 12 and newer
 - Opera: version 12 and newer



Other browsers might work; however, they are not officially supported

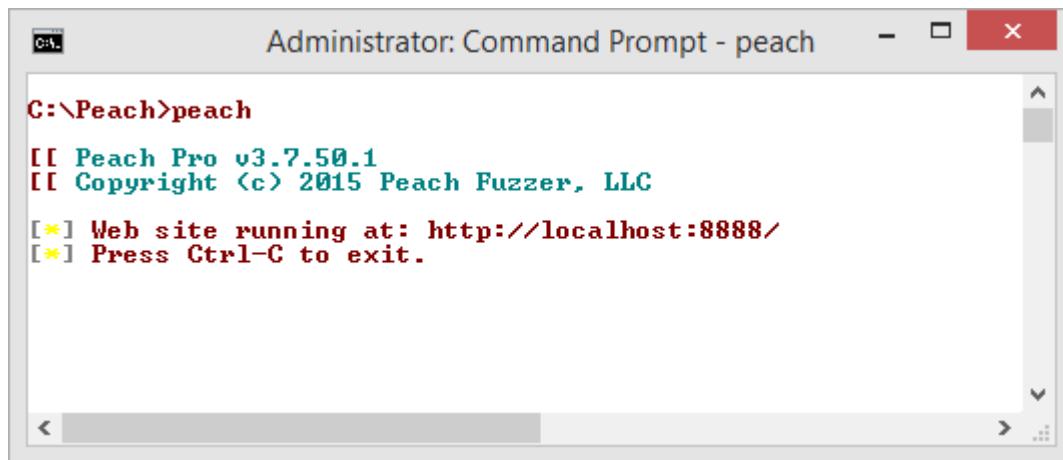
6.1.2. Starting the Peach Web Interface

You can start the Peach Web Interface using the GUI File Manager or from the command line. Both methods include the same functionality, use the method you prefer.

We recommend running Peach with heightened or administrative privileges. Some applications that provide monitoring functions, such as debuggers, need heightened access to run.

You can start Peach from a command line with two actions:

1. Open a command shell with administrative privileges.
2. On the command line, type Peach and press RETURN. The following illustration shows starting Peach in Windows.



The image shows an Administrator Command Prompt window titled "Administrator: Command Prompt - peach". The window contains the following text:
C:\Peach>peach
[[Peach Pro v3.7.50.1
[[Copyright <c> 2015 Peach Fuzzer, LLC
[*] Web site running at: http://localhost:8888/
[*] Press Ctrl-C to exit.

The Peach Web Interface is the default mode of operation for the command line. Launching the Peach Web Interface performs the following tasks:

1. Starts Peach Fuzzer Professional
2. Loads your default web browser with the Peach Web Interface URL

On the first launch of Peach, a licensing page displays.

File Edit View History Bookmarks Tools Help

localhost:8888/eula

END-USER LICENSE AGREEMENT: PEACH FUZZER(TM) ENTERPRISE SOLUTION

IMPORTANT: CAREFULLY READ THIS END USER LICENSE AGREEMENT BEFORE ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES. BY ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES, YOU AUTOMATICALLY ACKNOWLEDGE, ACCEPT AND AGREE TO ALL THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, YOU MAY NOT ACCESS OR OTHERWISE USE THE SOFTWARE AND ACCOMPANYING FEATURES.

This End-User License Agreement (the "Agreement") is a legally binding and enforceable agreement between Peach Fuzzer, LLC, a Washington limited liability company (the "Company") and the customer, individual or entity, set forth on the applicable Customer Invoice (the "Customer"), governing the Customer's license to and use of the Software, Definition Files and Features, as well as the Customer's license to create and use Customer Peach Pits.

1. DEFINITIONS.

(a) "Customer Invoice" means that certain final invoice issued by the Company to the Customer that identifies the specific license to products and services of the Company that the Customer has purchased.

(b) "Definition File(s)" means any Peach Pit files, development tools, and software programs developed by the Company and provided in the Peach Pit library subject to a current Peach Pit License.

BY CLICKING "I ACCEPT" YOU ACKNOWLEDGE THAT YOU HAVE READ, UNDERSTAND, AND AGREE TO BE BOUND BY THE TERMS ABOVE.

Accept Reject

When you have read and understood the End User License Agreement, click *Accept*.

3. Displays the Peach Home Page

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/. The title bar says "Peach Fuzzer". The left sidebar has icons for Home, Library, Jobs, Help, and Forums. The main content area is titled "Home" and includes a welcome message, a list of actions, and a "Recent Jobs" section with a table.

Welcome to the Peach Fuzzer ® UI home page, where you can do the following:

- Review results of fuzzing jobs, recent and not so recent
- Run new fuzzing jobs
- Replay fuzzing jobs to reproduce issues or to validate fixes of issues
- Review fuzzing definitions (pits) installed on your system
- Create and edit test configurations for the various pits

To begin, click on the Library on the left and select a pit to configure, then run in a new fuzzing job.

Recent Jobs

Here are the most recent fuzzing jobs. For any entry, you can perform the following actions:

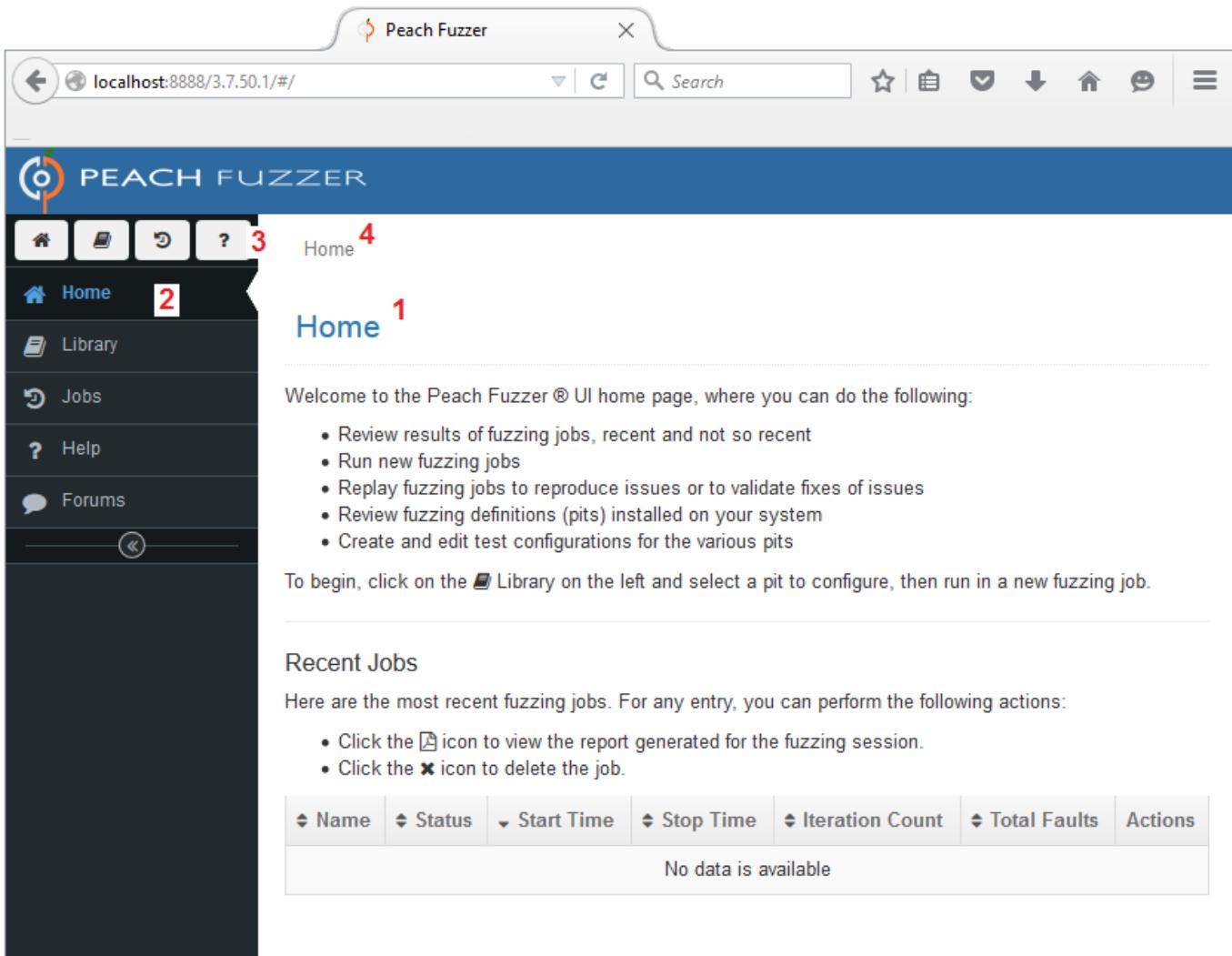
- Click the icon to view the report generated for the fuzzing session.
- Click the icon to delete the job.

Actions	Name	Status	Start Time	Stop Time	Iteration Count	Total Faults
No data is available						

From Home, you can use click entries and buttons on the left side of the page to work with your licensed Pits, view reports and details of previous fuzzing sessions, and to interact with the Peach forums.

Parts of the User Interface

The Web User Interface includes many useful components, the most prominent are called out in the following illustration:



1. Page Title

The Page title identifies the page you are on and establishes the context of your current task.

2. Menu Entries

Menu entries are located along the left edge of the browser window. These buttons identify the functionality immediately at your disposal. Whether you want to explore your Pit Library, investigate a completed fuzzing job, or get some assistance from Help or the Forums, the menu is there for you.

The entries in the menu change according to the work context. For example, when you start working with a Pit, the menu entries change to accommodate editing a Pit definition. Click an entry in the menu to start working on that item. There is also an option to collapse the menu.

3. Menu Icons

Located above the menu entries, the menu icons are always present and do not change. The icons are always:

- Home Page

- Your Pit Library
- Your Fuzzing Job Historical Results
- Peach User's Guide (HTML Help)

Click a menu icon to move there.

4. Breadcrumb menu

The breadcrumb is located above the Page Title and identifies your position within the Peach Web UI. Portions of the breadcrumb are links and marked appropriately by color.

A Quick Tour of the Peach Web Interface

This section provides a brief glimpse of the main areas of Peach that are accessible from the menu and buttons on the left edge of the page. The tour is quick: about a slide per menu item.

Library

The *Library* menu is where you select and configure Pits (fuzzing definitions), and run fuzzing jobs. The page lists the licensed Peach Pits first, then follows with the defined configurations that you create.

The screenshot shows the Peach Fuzzer application running in a web browser. The title bar reads "Peach Fuzzer" and the address bar shows "10.0.1.57:8888/#/library". The main content area is titled "PEACH FUZZER" with a logo. On the left, there's a sidebar with icons for Home, Library, Jobs, Help, and Forums. The "Library" icon is highlighted. The main content area has a breadcrumb "Home / Library" and a title "Library". Below it, a welcome message says: "Welcome to the Peach Pit library. This page consists of the following sections: Pits, Configurations, Legacy". A search bar at the bottom right says "Search for pits...". The main content area is divided into several sections: "Application" (containing HL7 and PPTX), "Image" (containing BMP, DICOM File, GIF, ICO, JPG, JPG2000, PNG), and "Net".

Jobs

The *Jobs* menu provides access to your fuzzing job results. Click on an entry to see the report status, summary, metrics, and drill down detail of individual findings.

The screenshot shows a web browser window titled "Peach Fuzzer". The address bar displays "localhost:8888/3.7.50.1/#/jobs". The main content area is titled "PEACH FUZZER" with a logo. A sidebar on the left contains links for "Home", "Library", "Jobs" (which is selected), "Help", and "Forums". The main content area shows a heading "Jobs" and a message: "Here is a comprehensive list of the fuzzing jobs on this computer. For any entry, you can perform the following actions:". Below this is a table header with columns: "Name", "Status", "Start Time", "Stop Time", "Iteration Count", "Total Faults", and "Actions". A message "No data is available" is displayed in the table body.

Help

The *Help* Menu provides access to the online Peach User Guide. This instructional piece provides workflows for installing Peach, recipes for setting up and running various configurations, and descriptions of the Peach monitors for detecting faults (issues), collecting data, and automating the test environment.

The screenshot shows a web browser window displaying the "Peach Fuzzer Professional" documentation. The title bar reads "File Edit View History Bookmarks Tools Help" and "Peach Fuzzer Professional - X". The address bar shows "localhost:8888/docs/index.html". The page header features the "PEACH FUZZER" logo and navigation links for "SIDEBAR" and "NEXT ▶". The main content area is titled "Peach Fuzzer Professional" and includes a "User Guide" section for "Peach Fuzzer, LLC". A "Table of Contents" sidebar on the left lists chapters such as Preface, Introduction to Fuzzing, Installation, What's new in Peach Platform v3.7 (2015 Q3), The Peach Web Interface, Monitor Recipes, Agents, and Monitors. Each chapter has a list of sub-links, such as "What is Peach Fuzzer Professional?", "Additional Resources", "Bug Reporting Guidelines", and "Peach Fuzzer Professional License".



Additionally, the *User Guide* is available in PDF format.

Peach Fuzzer Professional: User Guide

Table of Contents

Bookmarks

- 1. Preface
 - 1.1. What is Peach Fuzzer Professional?
 - 1.2. Additional Resources
 - 1.3. Bug Reporting Guidelines
- 2. Getting Started with the Peach Fuzzer Platform
 - 2.1. Picking your fuzzing target
 - 2.2. Installing Peach
 - 2.3. Launching Peach
 - 2.4. Selecting a Peach Pit
 - 2.5. Specifying a Test Configuration
 - 2.6. Running the Fuzzing Job
 - 2.7. Interpreting Results
 - 2.8. Fixing Issues and Re-running Test Cases
- 3. Introduction to Fuzzing
 - 3.1. Dumb Fuzzing
 - 3.2. Smart Fuzzing
 - 3.3. When to Stop Fuzzing
- 4. Installation
 - 4.1. Peach Fuzzer Professional License
 - 4.2. Downloading the Peach Fuzzer Distribution Files
 - 4.3. Windows
 - 4.4. Linux
 - 4.5. OS X
- 5. What's new in Peach Platform v3.8 (2015 Q4)
 - 5.1. Fuzzing Definitions (Pits)
 - 5.2. Canbus Publisher
 - 5.3. User Interface Enhancements
 - 5.4. Documentation
- 6. The Peach Web Interface
 - 6.1. Peach Web Interface Installation Requirements
 - 6.2. Starting the Peach Web Interface
 - 6.3. Quick Start Wizard
 - 6.4. Configuration Menu
 - 6.5. Test Pit Configuration
 - 6.6. Fuzzing Session
 - 6.7. Faults
 - 6.8. Metrics
 - 6.9. Switching Pits
- 7. Monitoring Recipes
 - 7.1. Recipe: Monitoring a File Consumer (File Fuzzing)
 - 7.2. Recipe: Monitoring a Linux Network Service Client

Forums

Peach has user forums that serve as a knowledge base of user questions, and as an active platform to raise questions of current need or interest. Feel free to explore the forums. Note that the professional forum provides a service venue to licensed users of Peach Professional and Peach Enterprise solutions.

6.1.3. Configuration Menu

You can configure a Pit by using the Configuration menu. This menu consists of four parts: variables, monitoring, tuning, and test. Only the Variable and Monitoring parts need to be completed to pass a test of the configuration.

To begin from the Home Page,

1. Click the Library menu entry.

- From the Pit Library, select a Pit or an existing configuration.
 - Selecting a configuration means that you are revising the settings of an existing configuration. Peach displays the start screen for the configuration.
 - Selecting a Pit means that you are creating a new configuration. You will need to name the configuration, optionally provide a description, and click "Submit" to reach the start screen for the configuration.

The screenshot shows the Peach Fuzzer web application interface. At the top, there is a navigation bar with links for File, Edit, View, History, Bookmarks, Tools, and Help. Below the navigation bar is a browser-style header with a back button, a search bar containing 'Search', and various icons for refresh, download, and navigation. The main title is 'PEACH FUZZER'. On the left side, there is a sidebar with icons for Home, Pit, Quick Start, Configure, Help, and Forums. The 'Pit' icon is highlighted. The main content area shows the title 'BMP_Demo'. A yellow warning box states: 'Warning! The currently selected Pit should be configured for monitoring the environment. Pit Configuration Quick Start'. Below this, a message says: 'This configuration uses the `BMP_Demo` pit and includes configuration data for your test setup.' There are three buttons in a row: 'Quick Start Wizard', 'Configure Variables', and 'Configure Monitoring'. Underneath these buttons is a section titled 'Start Options' with a 'Hide' link. It contains text explaining that start options specify test cases and identify a seed for generating mutated data. It also describes how to replay a fuzzing run. Below this, it says: 'For replay, changing the Start Options has the following effects:' followed by a bulleted list. The list includes: 'Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.', 'Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.' At the bottom, there are two input fields: 'Seed' set to 'Random Seed' and 'Start Test Case' set to '1'.

3. Click the "Configure Variables" button to define or edit the variables and their values.

Switching Pit Configurations



The active Peach Pit configuration can be changed in the Peach Web Interface by clicking on the Home menu icon above the menu along the left side of the screen. Then, click on the Library menu item to choose a Pit or Pit configuration in your library.

Variables

The variables data entry screen lists the information needed by the selected pit, as in the following illustration. Some information is pit-specific, such as file names used to fuzz file formats and port addresses used to fuzz network protocols. Other information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path.

Variables

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

Save + Add Variable

▼ All

Name	Key	Value
Fuzzed Data File	FuzzedFile	? fuzzed.bmp
Seed File	Seed	? *.bmp
Sample Path	SamplePath	? ##PitLibraryPath##/_Common/Samples/Image

▼ System Defines

Name	Key	Value
Peach OS	Peach.OS	? windows
Peach Installation Directory	Peach.Pwd	? C:\Peach
Peach Working Directory	Peach.Cwd	? C:\Peach
Root Log Directory	Peach.LogRoot	? C:\Peach\Logs
Pit Library Path	PitLibraryPath	? C:\Peach\pits

Pit-specific Variables

In the illustration, three variables are specific to the **BMP** Pit. Because the variables have default values, you can use the supplied values during the fuzzing job.

- If a variable lacks a default value and is not marked optional, you need to supply a value for the variable before you can fuzz the target.
- If a variable is labeled optional or has a default value, you need not supply a value to enable the configuration to run.

When entering a data value, type the value into the appropriate text box of the form. If you'd like to use the value of another variable, type `##`, the name of the variable that contains the value you want to use, and `##` to end the variable name.

The default value of the `Sample Path` variable in the previous illustration uses another variable (`PitLibraryPath`) as part of its value.

System-defined Variables

The Peach environment variables includes the following entries:

Peach OS

The Peach OS identifies the operating system that the Peach Fuzzer is using. The value is selected when downloading the Peach distribution image.

Peach Installation Directory

The Peach Installation Directory is the directory that contains the peach executable file. This directory was created when Peach was installed on the computer system.

Peach Working Directory

The Peach working directory is the current working directory for fuzzing. While this directory is usually the directory containing Peach, the directory can be another location on your system. The Peach Working Directory is set by launching Peach from the shell command line. The value is the current working directory of the command shell when you start Peach.

Root Log Directory

The default name of this directory is Logs. The default location is a subdirectory of the Peach Installation Directory. You can specify another location for it, such as a subdirectory of the Peach Working Directory.



Each pit has its own logs that appear as subdirectories of the logger Path.

Pit Library Path

The Pit library path is the full path of the folder in the Peach installation directory that contains your licensed Peach Pits and Pit Packs. The folder name is "pits", and contains subdirectories that hold your licensed Pits, Pit configurations, and Peach-supplied sample files.

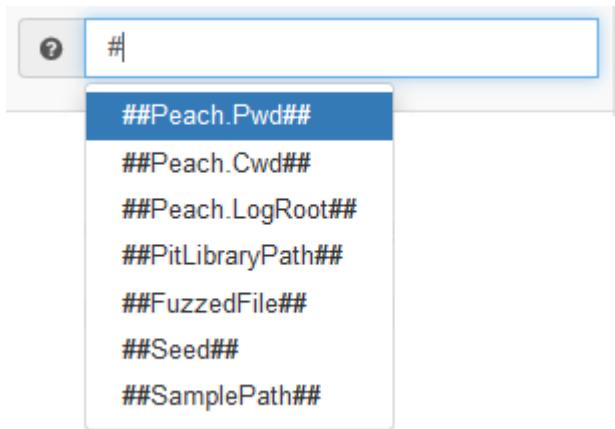
For descriptions on the pit-specific variables, see the documentation for the individual pit.

Custom Variables

You can create your own set variables for Peach. Perhaps you want to identify where the image repository is located. Or, perhaps you want to identify the MAC addresses, interface names, and IP addresses for the network interfaces on your system.

Whatever the resource you'd like to specify, once you've created a variable and given it a value, you can use the variable wherever appropriate when supplying values for Pit variables and monitor variables.

Additionally, once defined, the custom variable is added to the dropdown variable list. The list of variables displays during data entry when a # is typed as the sole character in a data field, as shown in the following illustration.



Creating a custom variable

Here are the steps to add a custom variable to Peach:

1. Begin on the Variables page.

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

- Click the "Add Variable" button immediately above and to the right of list of variables.

The "Add Variable" pop-up dialog displays.

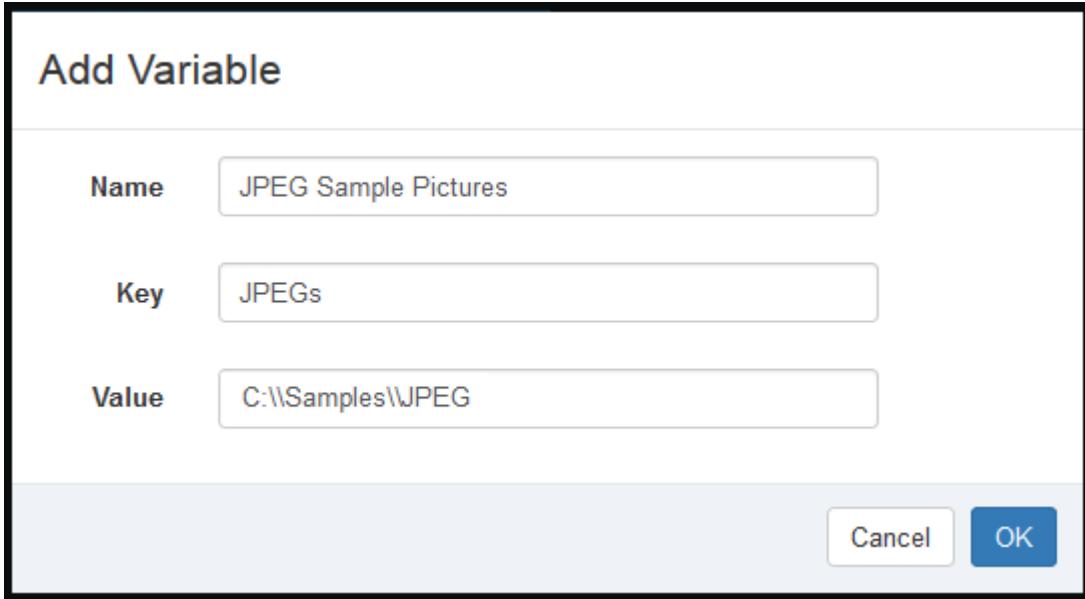
Name	<input type="text" value="Name"/>
Required	
Key	<input type="text" value="Key"/>
Required	
Value	<input type="text" value="Value"/>

Cancel OK

- Fill in values for the name, key, and value fields. The following illustration shows a variable locating a JPEG repository.
 - The **Name** parameter is the name of the custom variable, and allows spaces and punctuation. This is the name to search on if you want to edit the value of the variable.
 - The **Key** parameter is the value, such as **JPEGs**, that you plug into values in the Peach Web UI.

When Peach processes the pit, the fuzzer replaces the key parameter with the value of the **Value** parameter.

- The **Value** parameter stores the value that Peach uses when processing the pit.



4. Click OK.

5. Click save.

The new value is now part of the list, and has been saved for future use.

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

Saved successfully.

Name	Key	Value
Fuzzed Data File	FuzzedFile	fuzzed.bmp
Seed File	Seed	*.bmp
Sample Path	SamplePath	##PitLibraryPath##/_Common/Sa

Name	Key	Value	Remove
JPEG Sample Pictures	JPEGs	C:\Samples\JPEG	X

> System Defines

Using a custom variable

Using a custom variable consists of typing **##**, the variable name, and **##** in the value of another variable. In the following illustration, the "Sample Path" variable refers to the "JPEG Sample Pictures". When parsing the configuration information, Peach inserts the value **C:\Samples\JPEG** for the value of the "Sample Path".

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d30876a0. The left sidebar has a navigation menu with items like Pit, Quick Start, Configure, Variables (which is selected), Monitoring, Test, Help, and Forums. The main content area shows the 'Variables' configuration page. At the top, there's a breadcrumb trail: Home / Library / BMP_Demo / Configure / Variables. Below it, the title 'Variables' is displayed. A text block explains that this page lists information needed by the selected pit, including Peach environment variables and pit-specific variables like port addresses. There are two buttons at the top right: 'Save' and '+ Add Variable'. The main area contains three tables under sections 'All', 'User Defines', and 'System Defines'. The 'All' section has three rows: 'Fuzzed Data File' (Key: FuzzedFile, Value: fuzzed.bmp), 'Seed File' (Key: Seed, Value: *.bmp), and 'Sample Path' (Key: SamplePath, Value: ##JPEGS##). The 'User Defines' section has one row: 'JPEG Sample Pictures' (Key: JPEGs, Value: C:\\Samples\\JPEG). The 'System Defines' section is collapsed.

Monitoring

The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit.

Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers.

Monitors are components that perform one or more of the following functions: detect faults (issues),

collect data associated with faults, and help manage the fuzzing job to reduce the level of human interaction throughout the job.

From the Home Page

To begin configuring from the Home Page:

1. Click the Library menu entry.
2. From the Pit Library, select a Pit or an existing configuration.
 - Selecting a configuration means that you are revising the settings of an existing configuration. Peach displays the start screen for the configuration.
 - Selecting a Pit means that you are creating a new configuration. You will need to name the configuration, optionally provide a description, and click "Submit" to reach the start screen for the configuration.

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d308. The main title bar says "Peach Fuzzer". The left sidebar has a "Pit" section with "Quick Start", "Configure", "Help", and "Forums" buttons. The main content area shows "BMP_Demo" and a warning message: "Warning! The currently selected Pit should be configured for monitoring the environment. Pit Configuration Quick Start". Below this, it says "This configuration uses the **BMP_Demo** pit and includes configuration data for your test setup." There are three buttons: "Quick Start Wizard", "Configure Variables", and "Configure Monitoring". The "Configure Monitoring" button is highlighted. A "Start Options" section follows, with a note about replaying runs and changing start/stop test cases. It also mentions seed selection. At the bottom, there are dropdown menus for "Seed" (set to "Random Seed") and "Start Test Case" (set to "1").

- Click the "Configure Monitoring" button to define or edit agents, monitors, and the data values associated with them. The Monitoring data entry screen displays and is initially empty.

The screenshot shows a web browser window titled "Peach Fuzzer". The address bar displays "localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d30876a0". The main content area is titled "PEACH FUZZER" and shows the "Monitoring" page. On the left, there is a sidebar with icons for Home, Library, BMP_Demo, Configure, Monitoring, Test, Help, and Forums. The "Monitoring" icon is highlighted. The main content area has a breadcrumb navigation path: Home / Library / BMP_Demo / Configure / Monitoring. Below the path, the title "Monitoring" is displayed. A text block explains: "The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit. Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers." A yellow warning box contains the message "Warning! No agents have been configured." At the bottom right of the content area are two buttons: "Save" and "+ Add Agent".

The workflow for this data entry screen has you declare an Agent. Then, you can branch out and do the following in any order:

- Declare one or more monitors for the agent
- Fill in details for a monitor
- Switch focus from one monitor to another
- Declare additional agents, as needed
- Switch focus from one agent to another

In this instance, we're going to complete the agent, then add a monitor and fill in the monitor settings.

Specifying an Agent

Here are the steps to add an agent to a configuration:

1. Click the "Add Agent" button. Peach adds a new agent to the Monitors page, as in the following illustration.

The screenshot shows the Peach Fuzzer web interface. The URL in the address bar is `localhost:8888/3.8.51.1/#/pit/5376991a9f3fe90d308`. The main content area is titled "Monitoring". It contains a brief description of what monitoring does, a red warning box stating "Save is disabled until validation issues are fixed.", and a configuration form for an agent. The form has a dropdown "local://", a "Name" field with a required indicator, and a "Location" field set to "local://". A yellow warning box says "Warning! At least one monitor is advised." There are "Save" and "Add Agent" buttons at the top right of the form.

An agent has a name and location. The location can be local or remote.

- Local agents run in the same process space as the Peach fuzzing engine.
- Remote agents are separate processes that can reside on the same hardware as the test target. Remote agents act as intermediaries between Peach and test targets, sending test cases to the test target and replying with test case results and data back to Peach.

If you use a remote agent, you need to supply location information that conforms to a URL with the following parts: `channel://host:port`

channel

Specify one of the following for the channel type: `local`, `tcp`, or `http`. Typical remote agent configurations should use the `tcp` channel.

host

Specify the hostname of the agent to be used. This value is not required for the *local* channel.

port

Specify the port number of the agent to be used. This value is not required for the *local* channel.

- Example agent using the `tcp` channel: `tcp://192.168.127.128:9001`

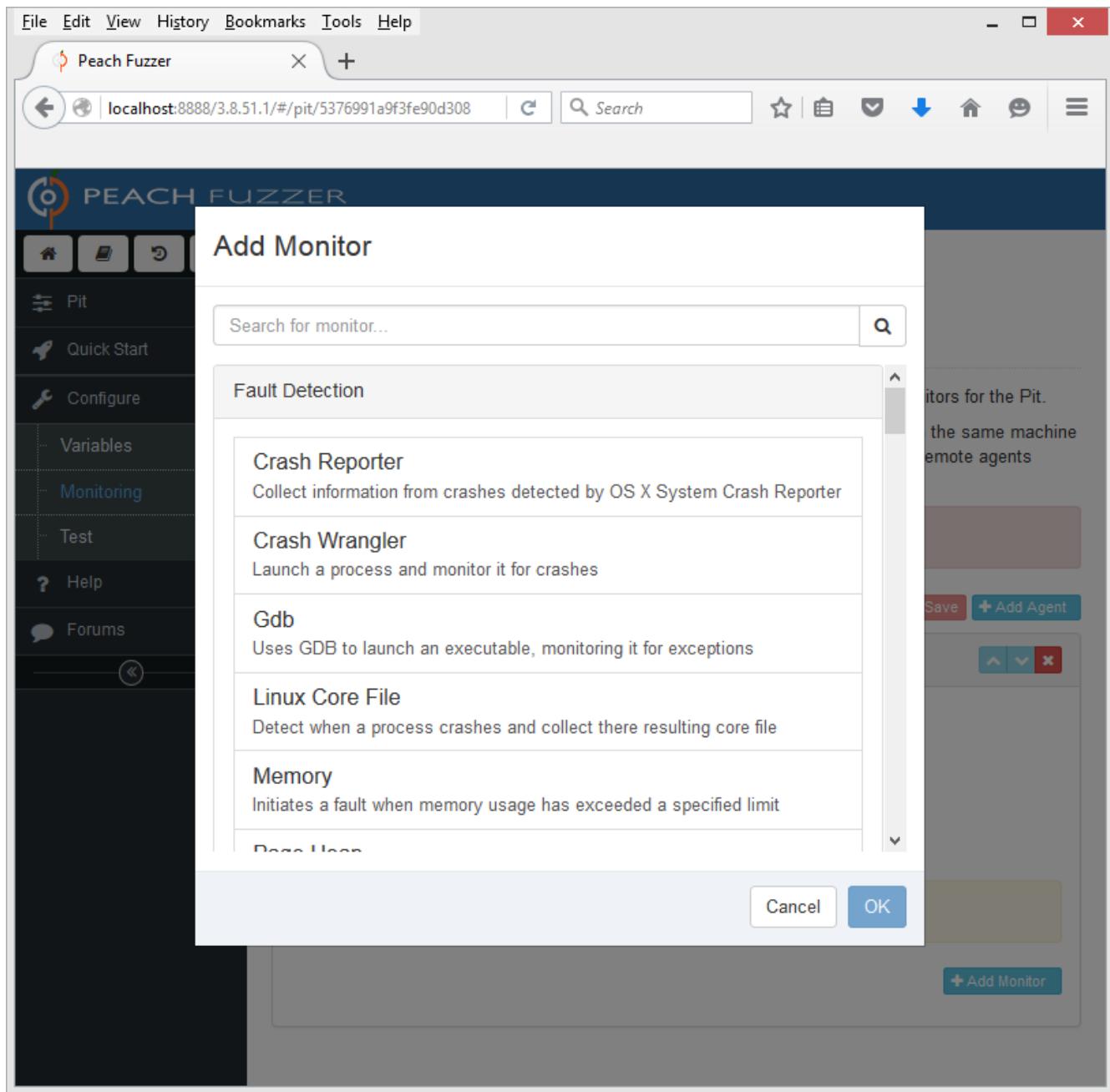
2. Give the agent a name, such as `LocalAgent` and click "Save".

Peach saves the Agent information, provides a visual cue with a "Saved successfully." message in a banner near the top of the page.

Adding a Monitor

1. click the "Add a monitor..." button.

Peach displays a list of monitors that you can use in your configuration. The monitors are categorized by usage. Fault detection monitors appear first, then data collection, automation, android, and lastly, other monitors.



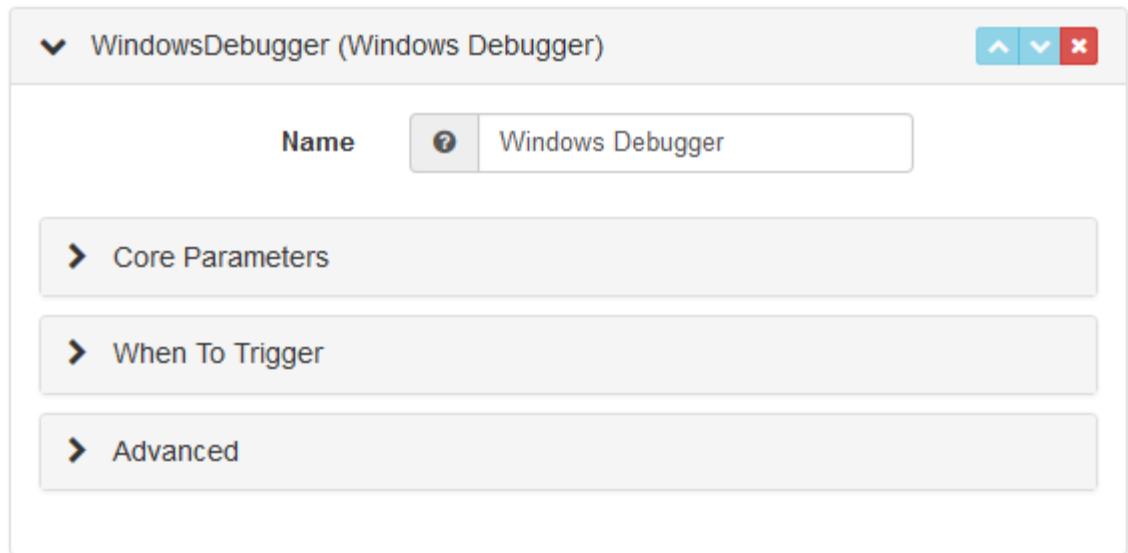
2. Select the **WindowsDebugger** entry, located under the **Fault Detection** section, and click "OK".

The screenshot shows the Peach Fuzzer web application interface. The title bar reads "Peach Fuzzer". The address bar shows the URL "localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d30876a0". The main navigation menu includes "File", "Edit", "View", "History", "Bookmarks", "Tools", and "Help". Below the menu is a toolbar with icons for back, forward, search, and other functions. The left sidebar has a "Pit" icon and a vertical menu with "Quick Start", "Configure", "Variables", "Monitoring" (which is selected), "Test", "Help", and "Forums". The main content area shows the "Monitoring" configuration screen. The URL in the browser is "Home / Library / BMP_Demo / Configure / Monitoring". The page title is "Monitoring". A sub-section header says "The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit. Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers." There are two sections: "local:// (LocalAgent)" and "WindowsDebugger (Windows Debugger)". Each section has a "Name" field (set to "LocalAgent" and "Windows Debugger" respectively) and a "Location" field (set to "local://"). A "Save" button and an "Add Agent" button are at the top right. A "Add Monitor" button is located below the "WindowsDebugger" section. The "Core Parameters" section for the Windows Debugger monitor includes fields for "Executable", "Arguments", and "Process Name".

3. Fill in the details of monitor.

Configuration information for each monitor is available in the [Monitors](#) reference section. A list of monitors appears at the start of the section that links to the individual entries.

The monitor parameters divide into three groups: "Core", "When To Trigger", and "Advanced".

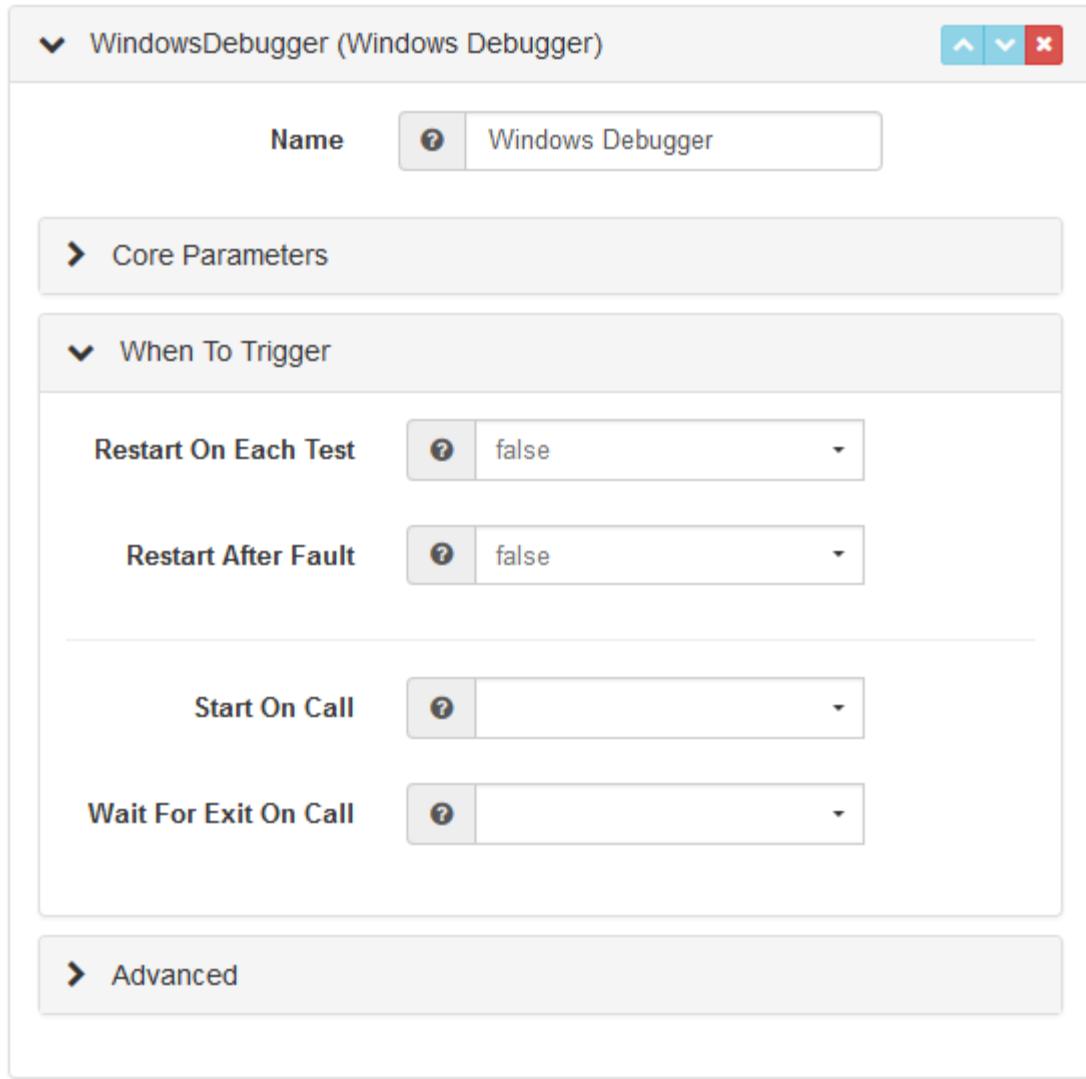


- The "Core" parameters consist of parameters that you should check when creating or editing a configuration. This monitor provides a choice of what to monitor: an executable file, a running process, or a service.

▼ WindowsDebugger (Windows Debugger) ^ ▾ ✕

Name	? Windows Debugger
Core Parameters	
Executable	? <input type="text"/>
Arguments	? <input type="text"/>
Process Name	? <input type="text"/>
Service	? <input type="text"/>
Win Dbg Path	? <input type="text"/>
When To Trigger	
Advanced	

- The "When To Trigger" addresses timing-related issues, such as restarting the test target at the end of an iteration. This is common for file fuzzing, where Peach creates a fuzzed data file, then starts the target with the fuzzed file as input.



- The "Advanced" parameters are items that seldom need to be specified; the default values of these parameters are usually sufficient as is. Yet, once in a while you might need to access one of these parameters. When fuzzing a network client, the "No Cpu Kill" parameter needs to be set to true to give the client an opportunity to close cleanly.

▼ WindowsDebugger (Windows Debugger) ▲ ▼ ×

Name	? Windows Debugger
Core Parameters	
When To Trigger	
Advanced	
No Cpu Kill	? false
Fault On Early Exit	? false
Wait For Exit Timeout	? 10000
Symbols Path	? SRV*http://msdl.microsoft.com
Ignore First Chance Guard Page	? false
Ignore Second Chance Guard Page	? false
Cpu Poll Interval	? 200

Sample Agent with Multiple Monitors

The following illustration is of an agent with multiple monitors. Note that you can show or hide the details for a monitor by clicking the chevron preceding the monitor name. In fact, this is true for agents, too.

The screenshot shows the Peach Fuzzer web application interface. The title bar reads "Peach Fuzzer". The left sidebar has a dark theme with white icons and text, showing navigation links: Home, Pit, Quick Start, Configure, Variables, Monitoring (which is selected and highlighted in blue), Test, Help, and Forums. The main content area has a light blue header with the title "Monitoring". Below it, a text block explains the purpose of the monitoring screen: "The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit." It describes agents as host processes for monitors and publishers, and provides local and remote options. A "Save" button and a "+ Add Agent" button are at the top right. The main form is titled "local:// (Agent0)". It contains fields for "Name" (set to "Agent0") and "Location" (set to "local://"). Below these are four expandable sections: "PageHeap", "WindowsDebugger", "SaveFile", and "Process", each with its own set of up/down and delete buttons.

Sample Remote Agent

The following illustration is of a remote agent. The location of the agent is the IP address of the remote machine, and that the address is stored in a custom variable. The agent manages multiple monitors.

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/ed9a357583baa23c. The title bar says "Peach Fuzzer". The left sidebar has icons for Home, Pit, Quick Start, Configure, Variables, Monitoring (which is selected and highlighted in blue), Test, Help, and Forums. The main content area shows the "Monitoring" page with the URL [Home / Library / IPv4 / Configure / Monitoring](#). The page title is "Monitoring". A text block explains: "The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit. Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers." Below this are two buttons: "Save" (red) and "+ Add Agent" (blue). The main form contains an agent entry for "tcp://##TargetIPv4## (Agent0)". It has fields for "Name" (Agent0) and "Location" (tcp://##TargetIPv4##). Below the agent is a button "+ Add Monitor". There are four monitor entries: "PageHeap", "WindowsDebugger", "Pcap", and "Process", each with up/down and delete buttons.

Tuning

Tuning allows control of how testing is performed on a field-by-field basis.

Fields are shown hierarchically below. A search feature is provided to quickly find fields of interest. Each field, or set of fields, can be excluded from testing or have its testing focus turned up or down. As a field's focus is turned up (High or Highest), test cases will be generated more often for that field. If a

field is turned down (Low, Lowest), fewer test cases will be generated for that field. Fields that are excluded will not have any test cases generated.

Excluding fields from testing should be used judiciously as it can lead to undiscovered faults.



Tuning is not required. In fact it's recommended to use the defaults and let Peach decide how often to test fields.

From the Home Page

To begin configuring from the Home Page:

1. Click the Library menu entry.
2. From the Pit Library, select a Pit or an existing configuration.
 - Selecting a configuration means that you are revising the settings of an existing configuration. Peach displays the start screen for the configuration.
 - Selecting a Pit means that you are creating a new configuration. You will need to name the configuration, optionally provide a description, and click "Submit" to reach the start screen for the configuration.

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d308. The left sidebar has a 'Pit' section expanded, showing 'Quick Start', 'Configure', 'Help', and 'Forums'. The main content area is titled 'BMP_Demo'. A yellow warning box says: 'Warning! The currently selected Pit should be configured for monitoring the environment. Pit Configuration Quick Start'. Below it, text states: 'This configuration uses the **BMP_Demo** pit and includes configuration data for your test setup.' There are three buttons: 'Quick Start Wizard', 'Configure Variables', and 'Configure Monitoring'. A 'Start Options' section explains how to replay a fuzzing job by selecting a run and clicking 'Replay'. It also describes how changing start options affects the job. A 'Seed' dropdown is set to 'Random Seed' and a 'Start Test Case' dropdown is set to '1'.

3. Click the "Configure" menu item on the left of the screen. It will expand to show several configuration options.
4. Click the "Tuning" sub-menu item to access the Tuning page. The Monitoring data entry screen displays and is initially empty.

Peach Fuzzer

10.0.1.57:8888/#/pit/3f8913878cbe2695222e837c3c3d20d3/advanced

PEACH FUZZER

Home / Library / HTTP_Server-Demo / Configure / Tuning

Tuning

Tuning allows control of how testing is performed on a field-by-field basis.

Fields are shown hierarchically below. A search feature is provided to quickly find fields of interest. Each field, or set of fields, can be excluded from testing or have its testing focus turned up or down. As a field's focus is turned up (High or Highest), test cases will be generated more often for that field. If a field is turned down (Low, Lowest), fewer test cases will be generated for that field. Fields that are excluded will not have any test cases generated.

Excluding fields from testing should be used judiciously as it can lead to undiscovered faults.

Search for fields...

Exclude Lowest Low Normal High Highest

- Request-Line
- Method
- URI
- Version
- CRLF
- Headers

Tuning Fields

The Tuning page shows fields in a hierarchy. For file pits such as media formats or application formats, this will be the format of the file being generated. For network pits, the hierarchy will show different packets or messages being transmitted, assuming the protocol has this concept. When modifying a field that contains children, by default all of the children will also be changed. This allows quickly tuning entire messages or sections of a message.

For each field there are several tuning options:

Exclude

Don't perform any testing of this field.



Excluding fields from testing should be used judiciously, as it may lead to missed faults. It's our recommendation that all fields get tested. Instead of excluding fields, consider tuning them to lower or lowest.

Lowest

Produce the least number of test cases for this field. When selected, the field will still receive some testing, but much less than fields marked as normal.

Low

Produce fewer test cases for this field. Fields tuned to low will still be tested, but fewer test cases will be generated compared to fields marked as normal.

Normal

No changes to how this field is tested. Peach will decide how often to generate test cases for fields marked as normal. This is the recommended setting for all fields.

High

Produce more test cases for this field. Fields tuned to high will receive more testing than those marked as normal.

Highest

Produce even more test cases for this field. Fields tuned to highest will receive more testing than those marked as high.

Once you have completed tuning the fields, click the Save button in the upper right of the screen.

Test

In the Test section, Peach performs a test on the selected Pit configuration using settings provided for variables, agents, and monitors. Peach identifies the readiness of the Pit configuration for testing by tracking and reporting the progress of completing settings for the variables, agents, and monitors.

The Test section runs a single test case without any fuzzing.

The test requires that the target device, service, or application be available for use.

This screen issues a warning if the pit is not configured, but lets the user run the test.

- Click the Begin Test button to run the test.

- Click the Begin Test button to run the test.

When the test completes, Peach reports whether the Pit configuration passes the test. If the configuration passes the test, the following message displays:

The screenshot shows the Peach Fuzzer web application running in a browser window titled "Peach Fuzzer". The URL is 10.0.1.57:8888/#/pit/3f8913878cbe269522e837c3c3d20d3/advanced. The left sidebar has a dark theme with white icons and text, showing navigation links like Home, Pit, Quick Start, Configure, Variables, Monitoring, Tuning, and Test. The main content area has a light blue header with the "PEACH FUZZER" logo. The current page is "Test" under "Configure". The main content area contains a section titled "Test" with a description of its purpose: "This section validates the configuration by executing one test case and exposes issues that would hinder a fuzzing session. The output will display any warnings and errors that surface in the control test case. Detailed log messages are provided to help diagnose issues with the configuration." Below this is a note: "Click the Begin Test button to validate the configuration." At the bottom are two buttons: "Continue" (green) and "Begin Test" (orange). A green message box says "Testing passed, click Continue.". Below this is a section titled "Test Output (12:30 pm)" with tabs for "Summary" (selected) and "Log". The "Summary" tab shows a list of successful steps with green checkmarks:

Message
✓ Loading configuration file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml.config'
✓ Loading pit file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml'
✓ Starting fuzzing engine
✓ Connecting to agent 'local://'
✓ Starting monitor 'SshCommand'
✓ Notifying agent 'local://' that the fuzzing session is starting
✓ Connecting to agent 'tcp://192.168.48.129:9001'

You can start a fuzzing job with your pit.

6.1.4. Fuzzing Session

With your Pit configured and tested, you're ready to start fuzzing!

From the Home Page

1. From the Home screen, click the Library menu.
2. From your Pits Library screen, select a configuration.

The configurations are listed in the section that follows the Pits.

Once selected, the configured Pit displays, as in the following illustration.

File Edit View History Bookmarks Tools Help

Bing × Peach Fuzzer ×

localhost:8888/3.7.57.1/#/pit/889c168ecb45814200992a0b751b318e

 PEACH FUZZER

Home / Library / BMP_1_Simple

BMP_1_Simple

The Pit is configured and ready for use. Click the START button below to begin fuzzing.

This configuration uses the **BMP_1_Simple** pit and includes configuration data for your test setup.

Configuration Options

Set or change the test configuration using the following buttons:

- Quick Start Wizard**: Leads a structured question & answer session that covers typical configurations. This choice is recommended for novice users and for simple configurations.
- Configure Variables**: A list of items needed by the configuration, including global pit variables and pit-specific information, such as target IP addresses.
- Configure Monitoring**: A list of the agents and monitors defined for the test configuration. Click this button to select and update the agents, monitors and associated monitor settings for the test configuration. This choice provides the most flexibility in implementing fault detection, data collection, and automation.

Start Options

The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.

If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.

For replay, changing the Start Options has the following effects:

- Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.
- Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.

Click "Start" to begin the fuzzing job.

Seed	Random Seed
Start Test Case	1
Stop Test Case	Default
	

If needed, you can change configuration settings or set some other parameters at the button of the page (typically used in replaying a fuzzing session).

3. Click Start to begin the fuzzing session. The Peach Dashboard displays.

The screenshot shows a web browser window with the title 'Peach Fuzzer'. The address bar shows 'localhost:8888/3.7.59.1/#/job/9e7a0197-16e1-46cc-ad98-204c611ae88'. The main content area is titled 'BMP 1' and displays the message 'Peach is currently fuzzing...'. Below this, there is a summary table with the following data:

Start Time	10/9/15 3:07PM	Running Time	00h 00m 24s
Test Cases/Hour	3450	Seed	6023
Test Cases Executed	23	Total Faults	0

Below the table are three buttons: 'Start' (green), 'Pause' (blue), and 'Stop' (red). Underneath the table is a section titled 'Recent Faults' with a table header:

#	When	Monitor	Risk	Major Hash	Minor Hash	Download
---	------	---------	------	------------	------------	----------

The message 'No data is available' is displayed below the header.

The dashboard allows you to monitor progress as your fuzzing job runs and from it, you can pause, stop, resume, and replay your fuzzing session. The Peach Dashboard provides the following information:

- The Configuration name, above the colored status bar
- The time the job started
- The duration that the job has been running
- The number of fuzzing test cases per hour
- The seed ID for random number generation, so you can replicate the test, if needed
- Number of test cases completed
- Total number of faults found in this run
- A summary of the most recent faults.

NOTES

1. The seed ID influences the fuzzing that occurs during a fuzzing job. If you want to replicate a test, the seed value is required to reproduce the exact sequence of values from the random-number generator used in fuzzing.
2. The STOP button does NOT close Peach. The STOP button only allows you to stop the currently running job.
3. If you have stopped a job and wish to start a new job using a different pit, choose one of the Pits or Pit configurations in your Pit Library. You'll need to re-visit the Home page, and then choose the appropriate entry from the library.
4. The fault summary lists the most recent faults. For information about the faults generated during this fuzzing session, click the Metrics menu, then Faults on the left side of the screen.



From a Configuration Test

When your Pit configuration passes the validation test, the following screen displays with the green banner and the message "Testing Passed, click Continue."

The screenshot shows the Peach Fuzzer web application running in a browser window titled "Peach Fuzzer". The URL in the address bar is 10.0.1.57:8888/#/pit/3f8913878cbe2695222e837c3c3d20d3/advanced. The user is logged in as "Michael". The main content area displays the "Test" configuration page for an "HTTP_Server-Demo" library. The left sidebar includes links for Pit, Quick Start, Configure, Variables, Monitoring, Tuning, and Test. The "Test" link is currently selected. The main content area has a heading "Test" and a descriptive text block about validating configuration. Below it, a message says "Click the Begin Test button to validate the configuration." Two buttons are present: a green "Continue" button and an orange "Begin Test" button. A green success message box states "Testing passed, click Continue." Below this is a section titled "Test Output (12:30 pm)" with tabs for "Summary" and "Log". The "Summary" tab is active, showing a list of successful messages:

Message
✓ Loading configuration file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml.config'
✓ Loading pit file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml'
✓ Starting fuzzing engine
✓ Connecting to agent 'local://'
✓ Starting monitor 'SshCommand'
✓ Notifying agent 'local://' that the fuzzing session is starting
✓ Connecting to agent 'tcp://192.168.48.129:9001'

From here, you can start a fuzzing session with two clicks of the mouse.

1. Click Continue. Peach displays the Pit configuration page.

File Edit View History Bookmarks Tools Help

Bing × Peach Fuzzer ×

localhost:8888/3.7.57.1/#/pit/889c168ecb45814200992a0b751b318e

 PEACH FUZZER

Home / Library / BMP_1_Simple

BMP_1_Simple

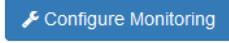
The Pit is configured and ready for use. Click the START button below to begin fuzzing.

This configuration uses the **BMP_1_Simple** pit and includes configuration data for your test setup.

Configuration Options

Set or change the test configuration using the following buttons:

- Quick Start Wizard**: Leads a structured question & answer session that covers typical configurations. This choice is recommended for novice users and for simple configurations.
- Configure Variables**: A list of items needed by the configuration, including global pit variables and pit-specific information, such as target IP addresses.
- Configure Monitoring**: A list of the agents and monitors defined for the test configuration. Click this button to select and update the agents, monitors and associated monitor settings for the test configuration. This choice provides the most flexibility in implementing fault detection, data collection, and automation.

Start Options

The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.

If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.

For replay, changing the Start Options has the following effects:

- Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.
- Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.

Click "Start" to begin the fuzzing job.

Seed	Random Seed
Start Test Case	1
Stop Test Case	Default
	

2. Click Start in the "Start Options" section at the bottom of the page to start a fuzzing job.

The Peach dashboard displays and the fuzzing job starts.

The screenshot shows the Peach Fuzzer web interface. The top navigation bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The title bar says "Peach Fuzzer". The address bar shows "localhost:8888/3.7.59.1#/job/9e7a0197-16e1-46cc-ad98-204c6111ae88". The search bar contains "office online". The main content area has a blue header "PEACH FUZZER". On the left is a sidebar with icons for Home, Dashboard, Faults, Metrics (selected), Help, and Forums. The main content area shows "BMP 1" and a message "Peach is currently fuzzing...". Below this is a table with the following data:

10/9/15 3:07PM	00h 00m 24s
3450	6023
Test Cases/Hour	Seed
23	0
Test Cases Executed	Total Faults

At the bottom are three buttons: "Start" (green), "Pause" (blue), and "Stop" (red). Below this is a section titled "Recent Faults" with a table header and the message "No data is available".

Re-running a Fuzzing Job

Peach allows you to re-run fuzzing sessions in whole or in part, meaning that exactly the same tests run using exactly the same data values. If you run an entire fuzzing job, all of the test cases are performed.

If you run a partial job, the test cases you specify run, again in exactly the same order as in the original fuzzing job using exactly the same data values.

What is needed to re-run a fuzzing job?

- Seed value
- Start test case number
- Stop test case number

Supply the seed value and appropriate Start and Stop Test Case values in the "Start Options" on the Pit Configuration page that follows, and click Start to begin a repeat of the fuzzing session.

The screenshot shows a web browser window with the URL localhost:8888/3.7.57.1/#/pit/889c168ecb45814200992a0b751b318e. The page title is "PEACH FUZZER". The main content area displays the configuration for a "BMP_1_Simple" pit. A sidebar on the left contains links for Home, Pit, Quick Start, Configure, Help, and Forums. The main content includes a message about the pit being ready for use, configuration options, start options, and a start button.

BMP_1_Simple

The Pit is configured and ready for use. Click the START button below to begin fuzzing.

This configuration uses the [BMP_1_Simple](#) pit and includes configuration data for your test setup.

Configuration Options

Set or change the test configuration using the following buttons:

- Quick Start Wizard** Leads a structured question & answer session that covers typical configurations. This choice is recommended for novice users and for simple configurations.
- Configure Variables** A list of items needed by the configuration, including global pit variables and pit-specific information, such as target IP addresses.
- Configure Monitoring** A list of the agents and monitors defined for the test configuration. Click this button to select and update the agents, monitors and associated monitor settings for the test configuration.
This choice provides the most flexibility in implementing fault detection, data collection, and automation.

[Quick Start Wizard](#) [Configure Variables](#) [Configure Monitoring](#)

Start Options

The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.

If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.

For replay, changing the Start Options has the following effects:

- Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.
- Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.

Click "Start" to begin the fuzzing job.

Seed: Random Seed

Start Test Case: 1

Stop Test Case: Default

Start

The next two sections provide information about the Seed Value and the Test Case number used in recreating a fuzzing job.

Seed Value

The seed value of a fuzzing job is located in the following places:

- For a running job, the seed is part of the dashboard display when a fuzzing job is running. See entry 2 of the right column of the dashboard.
- For a completed job, click on the Jobs menu and again on the entry in the Jobs list. This brings up the dashboard for the fuzzing job, where the seed is entry 2 of the right column.
- For a completed job, view the generated report, a [.pdf](#) file. The seed is the last table entry in the Summary section (section 1) of the report.

A seed is a common technique used in scientific and computer experiments to provide reproducible results when a random set of data values is needed. The seed feeds into a random-number generator that produces a sequence of "random numbers". Each time the seed is used, the same sequence of "random numbers" is generated.

Peach uses the "random numbers" in determining the mutators to use in a test case, the sequence of mutators, the data elements to fuzz, and the fuzzed data values. Having the seed value guarantees that the same sequence of numbers is generated and used throughout a fuzzing job.

Start/Stop Test Case

The Start Test Case number identifies the first test case to perform in a fuzzing job. The Stop Test Case number identifies the last test case to perform in a fuzzing job. Together, the Start and Stop Test Case numbers identify a range or a sequence of fuzzing test cases to run, whether the fuzzing job is new or is a re-run.

The number of a specific test case is present in the detail or drill-down report for a specific fault. In this report, the value is at the top of the report and is labeled "Iteration". The value represents the iteration number (or test case number) in a fuzzing job.

Main uses of specifying start and/or stop test case numbers in a fuzzing job are to confirm that an issue reliably occurs, to assist in tracking down an issue, or to verify that an issue is fixed.

If the issue does not reproduce, the issue will be more difficult to solve and might be a HEAP-related memory issue in which the addressable memory layout can have a large impact on the bug occurrence. In short, tracking down the root cause and verifying a fix for an issue will require running Peach for a long time to see whether the issue recurs. There is no easy way to guarantee an effective fix in this case.



Once a fix is in place, run a new fuzzing job to regress around the fix and to determine whether any residual faults surface.

6.1.5. Faults

While Peach Fuzzer Professional is running, you can view all the faults generated during the session by clicking the Faults menu option on the left.

Faults displays the total number of generated faults. There are two Faults views: the Summary view and the Detail view:

The screenshot shows the Peach Fuzzer Professional application window. The title bar says "Peach Fuzzer" and the address bar shows "10.0.1.57:8888/#/job/2f086ba2-ad7c-47e3-963f-2ab52933b0e3/faults/all". The main header is "PEACH FUZZER". The left sidebar has icons for Home, Dashboard, Faults (with 13 notifications), Metrics, Help, and Forums. The "Faults" item is selected. The main content area is titled "Faults" and contains a sub-header: "For each session, the Faults view lists a summary of information about a fault such as:" followed by a bulleted list. Below this is a table of fault data. The table has columns: #, When, Monitor, Risk, Major Bucket, Minor Bucket, and Download. The data rows are as follows:

#	When	Monitor	Risk	Major Bucket	Minor Bucket	Download
3188	1/31/16 8:58 PM	Gdb	EXPLOITABLE	047C1B7E	B4C61093	Download
5701	1/31/16 9:07 PM	Gdb	EXPLOITABLE	0B0DDD19	310A8210	Download
15739	1/31/16 9:43 PM	Gdb	EXPLOITABLE	047C1B7E	B4C61093	Download
15767	1/31/16 9:44 PM	Gdb	EXPLOITABLE	04EC23BE	C78DF99E	Download
18631	1/31/16 9:55 PM	Gdb	EXPLOITABLE	047C1B7E	244E0C17	Download
18784	1/31/16 9:56 PM	Gdb	EXPLOITABLE	14F145F9	F3F8E68E	Download
20296	1/31/16 10:01 PM	Gdb	EXPLOITABLE	047C1B7E	33E26825	Download
24220	1/31/16 10:15 PM	Gdb	EXPLOITABLE	AC60250E	393FF2B1	Download
30558	1/31/16 10:39 PM	Gdb	EXPLOITABLE	047C1B7E	33E26825	Download
32952	1/31/16 10:49 PM	Gdb	EXPLOITABLE	047C1B7E	33E26825	Download
33692	1/31/16 10:52 PM	Gdb	EXPLOITABLE	14F145F9	00D7125B	Download
37007	1/31/16 11:03 PM	Gdb	EXPLOITABLE	047C1B7E	C1D79156	Download

For each session, the Faults Summary view lists a summary of information about the fault such as:

- Identified fault iteration count
- Time and date
- Monitor that detected the fault
- Risk (if known)

- Unique identifiers of the fault (major and minor hashes), if available

Clicking on one of the listed faults from the Summary view opens the Details view for the selected fault.

The screenshot shows the Peach Fuzzer web application interface. The title bar says "Peach Fuzzer" and the address bar shows "10.0.1.57:8888/#/job/2f086ba2-ad7c-47e3-963f-2ab52933b0e3/faults/all/41400". The main content area is titled "Iteration: 41400". On the left is a sidebar with links for Dashboard, Faults (13), Metrics, Help, Forums, and a search bar. The "Faults" link is highlighted. The main content area displays fault details in a table:

Test Case	41400				
Reproducible	Yes				
Title	PossibleStackCorruption (7/22), AccessViolation (21/22)				
When	1/31/16 11:20 PM				
Source	Gdb				
Risk	EXPLOITABLE				
Major Bucket	B9973A01				
Minor Bucket	00507C8A				
Tested Fields	<table border="1"> <thead> <tr> <th>Field</th> <th>Mutator</th> </tr> </thead> <tbody> <tr> <td>HTTP:Request.request-line.uri</td> <td>StringLengthVariance</td> </tr> </tbody> </table>	Field	Mutator	HTTP:Request.request-line.uri	StringLengthVariance
Field	Mutator				
HTTP:Request.request-line.uri	StringLengthVariance				

Below the table is a "Description" section containing assembly code and a stack trace:

```

'exploitable' version 1.3
Linux ubuntu 3.13.0-29-generic #53-Ubuntu SMP Wed Jun 4 21:00:20 UTC 2014
Signal si_signo: 11 Signal si_addr: 140737488351232
Nearby code:
    0x00007ffff789685e <+1070>: add    rdi,0x40
    0x00007ffff7896862 <+1074>: add    rsi,0x40
    0x00007ffff7896866 <+1078>: movdqu XMMWORD PTR [rdi-0x40],xmm4
    0x00007ffff789686b <+1083>: movaps xmm2,XMMWORD PTR [rsi]
    0x00007ffff789686e <+1086>: movdqa xmm4,xmm2
=> 0x00007ffff7896872 <+1090>: movdqu XMMWORD PTR [rdi-0x30],xmm5
    0x00007ffff7896877 <+1095>: movaps xmm5,XMMWORD PTR [rsi+0x10]
    0x00007ffff789687b <+1099>: pminub xmm2,xmm5
    0x00007ffff789687f <+1103>: movaps xmm3,XMMWORD PTR [rsi+0x20]
    0x00007ffff7896883 <+1107>: movdqu XMMWORD PTR [rdi-0x20],xmm6

Stack trace:
# 0 __strcat_sse2_unaligned at 0x7ffff7896872 in /lib/x86_64-linux-gnu/libc.so.6
# 1 CrashMe at 0x401ffe in /home/mike/httpd
# 2 None at 0x6974736f702f7469 in ?

```

Here's where you can find details about the selected fault. Additional information (such as any files collected during the data collection phase) are located in the disk log folder.

6.1.6. Metrics

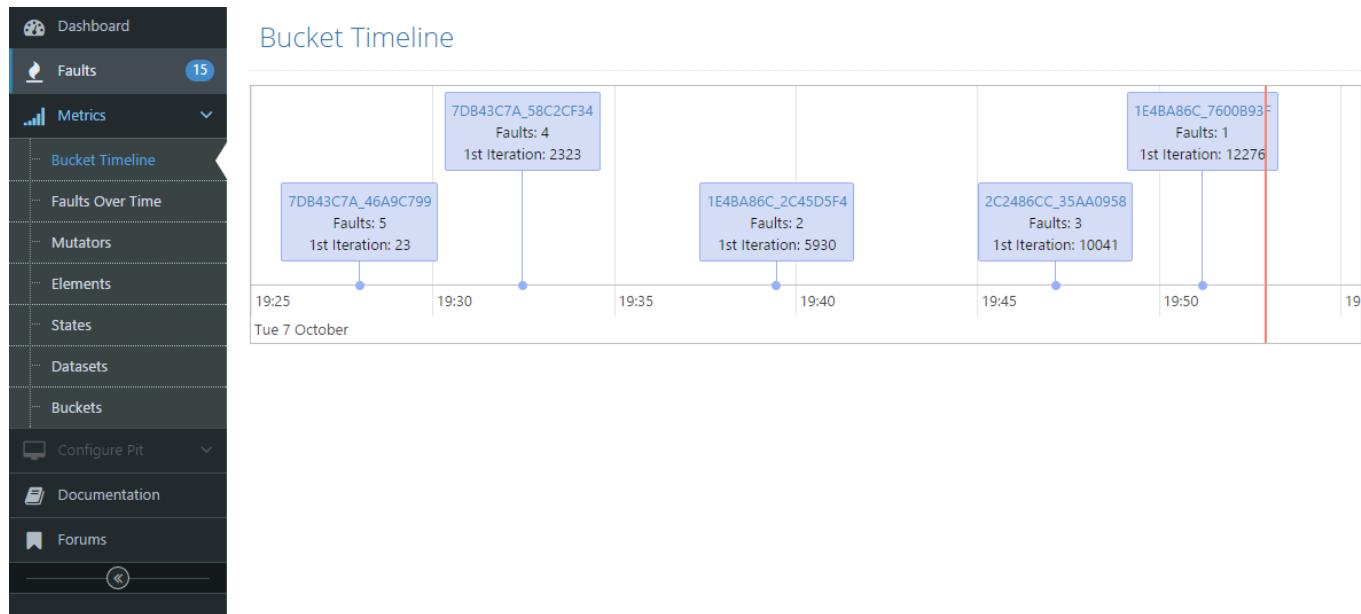
A number of metrics are available for viewing while Peach Fuzzer Professional is running.



The data grids used on many of the metrics displays support multi-column sorting using the *shift* key and clicking on the different columns to sort.

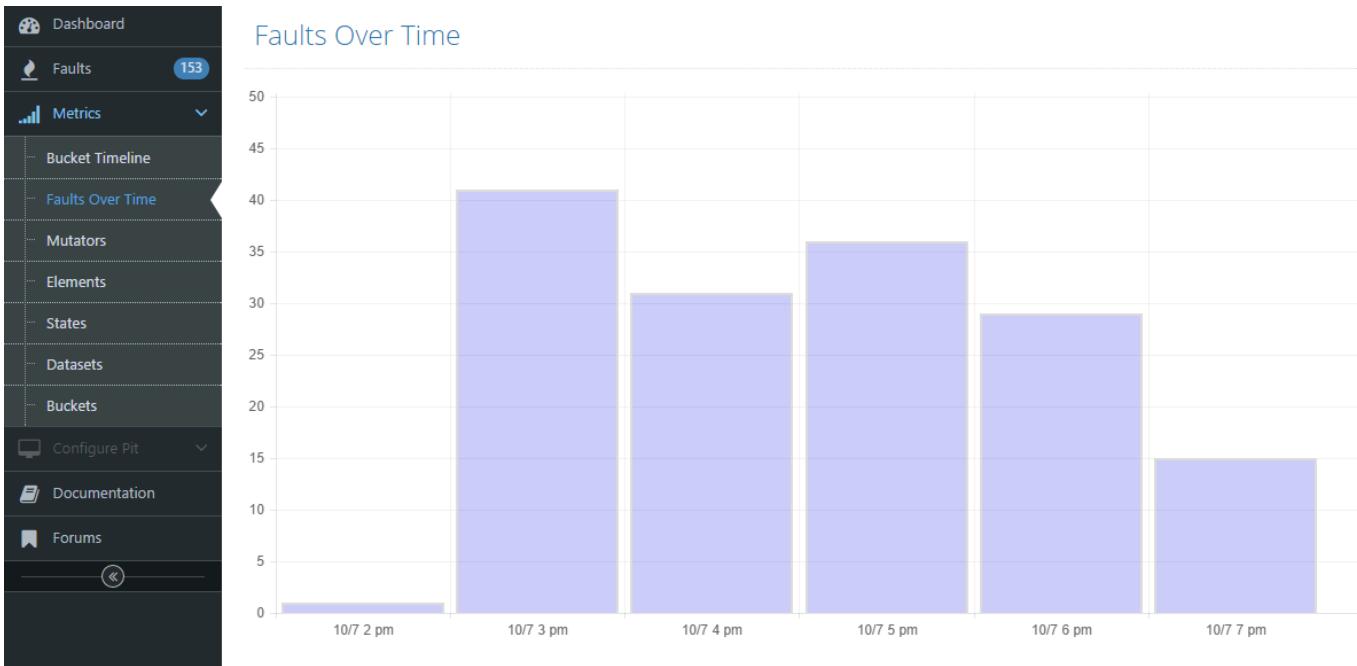
Bucket Timeline

This metric display shows a timeline with new fault buckets listed, and total number of times the bucket was found during the fuzzing session.



Faults Over Time

This metric display shows the count of faults found by hour over the course of the fuzzing run. This is the count of all faults found, not just unique buckets.



Mutators

This metric display shows statistics for each mutator by arranging the information into columns:

Element Count

The number of elements this mutator touched with mutated data.

Iteration Count

The number of iterations this mutator was used during the fuzzing job.

Bucket Count

The number of unique buckets found while this mutator was in use.

Fault Count

The number of faults found while this mutator was in use.

The screenshot shows the PEACH FUZZER application interface. The top navigation bar includes the logo and the user name "randofaulter". The left sidebar contains a navigation tree with the following items:

- Dashboard
- Faults (27)
- Metrics
- Bucket Timeline
- Faults Over Time
- Mutators**
- Elements
- States
- Datasets
- Buckets
- Configure Pit
- Documentation
- Forums
- Logout

The main content area is titled "Mutators" and displays a table with the following data:

Mutator	Element Count	Iteration Count	Bucket Count	Fault Count
BlobExpandAllRandom	175	1019	2	2
BlobExpandSingleIncrementing	176	1047	2	2
BlobExpandZero	176	1027	2	2
DataElementBitFlipper	139	192	1	1
NumberEdgeCase	80	24893	5	13
NumberRandom	80	6012	5	7
NumberVariance	80	24446	5	20
SizedDataEdgeCase	16	2110	2	3
SizedDataVariance	16	4127	2	4

Elements

This metric display shows statistics for all of the elements in your Pit.

This display shows several columns of information:

State

The state this element belongs to

Action

The action this element belongs to

Parameter

The parameter this action belongs to (if any). Parameters are used only with actions of type *call*.

Element

The full name of the element and its associated DataModel.

Mutations

The number of mutations generated from this element.

Buckets

The number of unique buckets found by sending mutating data to this element.

Faults

The number of faults found from the mutated data sent to this element.

Elements

State	Action	Parameter	Element	Iterations	Buckets	Faults
S1	A2		TheDataModel.Headers.Height	456	1	1
S1	A2		TheDataModel.Length	808	1	1
S1	A3		TheDataModel.Data.Count	325	1	1
S1	A3		TheDataModel.Data.DataSegment.DataSegment_1	15	1	1
S1	A3		TheDataModel.Data.Type	455	1	1
S1	A3		TheDataModel.Headers.Depth	453	1	1
S1	A3		TheDataModel.Headers.Height	459	1	1
S1	A3		TheDataModel.Length	781	1	1

States

This metric display presents statistics that are relevant for pits that have state models with more than two or more states. This display shows the number of times a specific state occurred during the fuzzing session. Seldom-used states might hide issues or indicate a problem.

For example, not all states always execute. If an early-occurring state is fuzzed, the outcome of the fuzzing could prevent states that are used late in the state flow from occurring.



Over time, the number of occurrences for most states should trend towards equality.

States

State	Executions
S1	14640
S2	14640
S3	14640
S4	14639

Data Sets

This metric display shows statistics related to the use of two or more data sets in the fuzzing session. This is useful to determine the origin of unique buckets and also faults in terms of the data sources

used in mutating.

This display shows several columns of information:

Data Set

Name of the data set

Iterations

Number of fuzzing iterations performed using this data set

Buckets

Number of unique buckets found with this data set

Faults

Number of faults found with this data set

The screenshot shows a software interface with a dark-themed sidebar on the left and a main content area on the right. The sidebar contains various navigation links: Dashboard, Faults (with 20 notifications), Metrics, Bucket Timeline, Faults Over Time, Mutators, Elements, States, Datasets (selected), Buckets, Configure Pit, Documentation, Forums, and a help icon. The main content area is titled 'Datasets' and displays a table with four columns: Data Set, Iterations, Buckets, and Faults. A single row is shown for 'Data' with values 55734, 3, and 19 respectively.

Data Set	Iterations	Buckets	Faults
Data	55734	3	19

Buckets

This metric display shows the buckets encountered during the fuzzing job. Several columns of information show:

Fault bucket

Identifier of the fault that occurred

Mutator

The mutator that generated the fault

Iteration count

The number of iterations that used the mutator

Faults count

The number of faults that occurred while using the mutator

Fault Bucket	Mutator	Element	Iteration Count	Fault Count
1E4BA86C	BlobExpandSingleIncrementing	S1.A3.TheDataModel.Data.Da...	3	1
1E4BA86C	NumberEdgeCase	S1.A2.TheDataModel.Headers...	230	1
1E4BA86C	NumberEdgeCase	S1.A2.TheDataModel.Length	212	1
1E4BA86C	NumberEdgeCase	S1.A3.TheDataModel.Headers...	224	1
1E4BA86C	NumberEdgeCase	S3.A2.TheDataModel.Headers...	212	1
1E4BA86C	NumberVariance	S1.A3.TheDataModel.Data.Co...	229	1
1E4BA86C	NumberVariance	S1.A3.TheDataModel.Headers...	198	1
1E4BA86C	NumberVariance	S1.A3.TheDataModel.Length	156	1
1E4BA86C	NumberVariance	S2.A2.TheDataModel.Data.Co...	215	1

Accessing Raw Metrics Data

Each job has its own SQLite database that contains metrics and other information about the job. The database is stored with other log assets under the logs folder in the peach application folder. Each job will have its own folder of assets. While we don't document the database schema, advanced users are welcome to mine the database to utilize the metrics data in different ways. The database format may change between versions, though typically changes are small.

6.2. The Peach Command Line Interface

This is the core Peach application which provides the core fuzzing capabilities and also some utility functions for the custom pit developer. This application can be used to start the Peach Web Application and also to launch fuzzing jobs from the command line.

Some options may be disabled depending on your Peach License options.

Please submit any bugs to support@peachfuzzer.com.

6.2.1. Peach Web Application

Starts Peach and provides a web application for configuring, running, and viewing results of a fuzzing job.

Syntax

```
peach [options]
```

--nobrowser

Disable launching browser on start.

--webport=PORT

Specified port the web application runs on.

--plugins=PATH

Change the plugins folder location. Defaults to the *Plugins* folder relative to the Peach installation.

6.2.2. Fuzzing from Command Line

A fuzzing run is started by specifying the Peach Pit Configuration or Peach XML file and the name of a test to perform.

If a run is interrupted, it can be restarted by providing the last successful test case and the seed of the test session. Use the --skipto and --seed parameters to provide this information.

Syntax

```
peach [options] <PEACH_PIT.xml | PEACH_CONFIG.peach> [test_name]
```

PEACH_CONFIG.peach

Peach Pit Configuration generated by Peach Application.

PEACH_PIT.xml

The Peach Pit XML file.

test_name

Name of test to run (defaults to *Default*).

-1

Perform a single test case

--debug

Enable debug messages. Useful when debugging Peach Pit Files.

-DKEY=VALUE

Define a configuration variable via the command line. Multiple defines can be provided as needed.

Example: -DTargetIPv4=127.0.0.1 -DTargetPort=80

--duration=DUR

Duration of fuzzing run. Peach will run for DUR length of time. Commonly integrating Peach into an automated test cycle or continuous integration environment. Argument format is DD.HH:MM:SS.

Examples

- **--duration=12** Duration of 12 days
- **--duration=0:20** Duration of 20 min
- **--duration=5:00** Duration of 5 hours
- **--duration=1.5:00** Duration of 1 day, 5 hrs

--noweb

Disable the Peach Web Application

--plugins=PATH

Change the plugins folder location. Defaults to the *Plugins* folder relative to the Peach installation.

--polite

Disable interactive console mode

--range=S,F

Perform a range of test cases starting at test case S and ending with test case F. Typically combined with the --seed argument.

--seed=SEED

Set the fuzzing jobs seed. The same seed will always produce the same test case sequence. Should only be set when reproducing a historical fuzzing job. Default is a random seed.

--skipto=NUM

Skip to NUM test case and start fuzzing. Normally combined with --seed to reproduce a specific sequence of test cases.

--trace

Enable even more verbose debug messages.

--webport=PORT

Specified port the web application runs on

6.2.3. Debug Peach XML File

This will perform a single iteration (-1) of your pit file while displaying a lot of debugging information (--debug). The debugging information is intended for custom pit developers.

Syntax

```
peach -1 --debug <PEACH_PIT.xml | PEACH_CONFIG.peach> [test_name]
```

6.2.4. Display List of Network Capture Devices

Display a list of all known devices Peach can perform network capture on.

Syntax

```
peach --showdevices
```

6.2.5. Display Known Elements

Print a list of all known:

- Actions
- Agent Channels
- Analyzers
- DataElements
- Fixups
- Loggers
- Monitors
- Mutation Strategies
- Mutators

- Publishers
- Relations
- Transformers

The list includes any associated parameters along with a description and default values. This can be used to verify that custom extensions are found.

Syntax

```
peach --showenv
```

6.2.6. Peach Agent

The Peach Agent functionality has been moved to a separate executable. See [PeachAgent](#) for more information.

6.2.7. Examples

Example 1. Running a Pit Configuration (.peach)

This example shows how to run a fuzzing job from a configuration file (.peach). The following command line launches Peach and fuzzes using `pit_config.peach` as the configuration file.

```
> peach pit_config.peach
```

Example 2. Running a Pit

This example shows how to run a fuzzing definition. The following command line launches Peach and fuzzes using `pit.xml` (and if it exists, `pit.xml.config`) as the configuration file.

```
> peach pit.xml
```

Example 3. Single Iteration with Debug Output

When testing a definition, we recommend running a single non-mutating iteration with debug output.

The following command line launches Peach and fuzzes using `pit.xml` (and if it exists, `pit.xml.config`) as the configuration file. The command line combines the `-1` and `--debug` arguments to run a single iteration; the debugging information is included in the output. Even

more verbose output can be enabled by using `--trace` instead of `--debug`.

```
> peach -1 --debug samples\DebuggerWindows.xml

[*] Test 'Default' starting with random seed 27886.
Peach.Core.Agent.Agent StartMonitor: Monitor WindowsDebugger
Peach.Core.Agent.Agent StartMonitor: Monitor_1 PageHeap
Peach.Core.Agent.Agent StartMonitor: Monitor_2 NetworkCapture
Peach.Core.Agent.Agent SessionStarting: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionStarting
Peach.Core.Agent.Agent SessionStarting: Monitor_1
Establishing the listener...
Waiting for a connection...
Peach.Core.Agent.Agent SessionStarting: Monitor_2

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Publishers.TcpClientPublisher open()
Accepted connection from 127.0.0.1:51466.
Peach.Core.Publishers.TcpClientPublisher output(12 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21          Hello World!

Received 12 bytes from client.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(12 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21          Hello World!

Received 12 bytes from client.
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:4
244
Connection closed by peer.
Shutting connection down...
Connection is down.
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:4244, closing
client connection.
Waiting for a connection...
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:4244
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault()
```

```
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault() - No fault detected
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.TcpClientPublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor_2
Peach.Core.Agent.Agent SessionFinished: Monitor_1
Peach.Core.Agent.Agent SessionFinished: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionFinished
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger

[*] Test 'Default' finished.
```

Example 4. Replay Existing Test Sequence

Once you find a faulting condition, you may want to replicate the exact test (or sequence of tests) to recreate the issue. Peach can reproduce exact test sequences given the following information:

1. Exact version of Peach. This is found in the log file `status.txt`.
2. Seed number used. This is also found in the log file `status.txt`.
3. Same/similar pit file. Data and state models must be the same.
4. If datasets are used, they must be the same set and have the same contents.

1. `status.txt`

Peach Fuzzing Run

=====

```
Date of run: 3/20/2014 1:58:58 PM
Peach Version: 3.1.40.1          ①
Seed: 51816                      ②
Command line: samples\DebuggerWindows.xml
Pit File: samples\DebuggerWindows.xml
. Test starting: Default
```

① Version of Peach used. Must match when reproducing.

② Seed used. Must match when reproducing.

We can use the first command line to skip directly to a specific iteration and start fuzzing. This lets you run a series of iterations starting from a certain point.



The `--seed` argument matches the value from the `status.txt` file.

```
> peach --seed 51816 --skipto 37566
```

We can use the second command line to perform either a specific iteration or a small number of iterations.

```
> peach --seed 51816 --range 37566,37566
```

6.3. PeachAgent

Starts a Peach Agent server process.

A Peach Agent can be started on a remote machine (remote to Peach) to accept connections from a Peach instance. Agents run various utility modules called Monitors and also host remote Publishers. Peach Agents do not need any specific configuration outside of which port to listen on. All configuration is provided by a Peach instance.

6.3.1. Licensing

The Peach Agent server process does not require a license. In a typical deployment, only the machine running the core Peach process (Peach.exe) requires a license.

6.3.2. Syntax

```
peachagent [--port=9001]
```

-h, --help

Display this help and exit

-V, --version

Display version information and exit

-v, --verbose

Increase verbosity, can use multiple times

--plugins=VALUE

Specify the plugins path

--port=VALUE

Port to listen for incoming connections on (defaults to 9001).

--debug

Enable debug messages. Useful when debugging your Peach Pit file. Warning: Messages are very cryptic sometimes.

--trace

Enable even more verbose debug messages.

6.4. Minset

This tool is used when adding additional samples to an existing Pit or creating a custom file fuzzing Pit.

Peach Minset is used to identify the minimum number of sample files required to provide the greatest code coverage for a given target. This process can be distributed across multiple machines to decrease the run time.

There are two steps to the process:

1. Collect traces
2. Compute minimum set coverage

The first step can be distributed and the results collected for analysis by the second step.

6.4.1. Collect Traces

Performs code coverage using all files in the *samples* folder. Collects the .trace files for later analysis. This process can be run in parallel across multiple machines or CPU cores.

Syntax

```
PeachMinset [-k] -s samples -t traces command.exe args %s
```

-k

Kill target when CPU usage drops to near zero. This is used when taking traces of GUI programs.

-s samples

Folder to load sample files from

-t traces

Folder to write traces to

command.exe args %s

Executable and arguments. **%s** is replaced with the path and file name of a file from the samples folder.

6.4.2. Compute Minimum Set Coverage

Analyzes all .trace files to determine the minimum set of samples to use during fuzzing. This process cannot be parallelized.

Syntax

```
PeachMinset -s samples -t traces -m minset
```

-s samples

Folder to load sample files from

-t traces

Folder to write traces to

-m minset

Folder to write minimum set of files to

6.4.3. All-In-One

Both tracing and computing can be performed in a single step.

Syntax

```
PeachMinset [-k] -s samples -m minset -t traces command.exe args %s
```

-k

Kill target when CPU usage drops to near zero. This is used when taking traces of GUI programs.

-s samples

Folder to load sample files from

-m minset

Folder to write minimum set of files to

-t traces

Folder to write traces to

command.exe args %s

Executable and arguments. **%s** is replaced with the path and file name of a file from the samples folder.

6.4.4. Distributing Minset

Minset can be distributed by splitting up the sample files and distributing the collecting of traces to multiple machines. The final compute minimum set coverage cannot be distributed.

6.4.5. Examples

Example 5. Example Run

```
> PeachMinset.exe -s pinsamples -m minset -t traces bin\pngcheck.exe %%  
  
[*] Running both trace and coverage analysis  
[*] Running trace analysis on 15 samples...  
[1:15] Coverage trace of pinsamples\basn0g01.png...done.  
[2:15] Coverage trace of pinsamples\basn0g02.png...done.  
[3:15] Coverage trace of pinsamples\basn0g04.png...done.  
[4:15] Coverage trace of pinsamples\basn0g08.png...done.  
[5:15] Coverage trace of pinsamples\basn0g16.png...done.  
[6:15] Coverage trace of pinsamples\basn2c08.png...done.  
[7:15] Coverage trace of pinsamples\basn2c16.png...done.  
[8:15] Coverage trace of pinsamples\basn3p01.png...done.  
[9:15] Coverage trace of pinsamples\basn3p02.png...done.  
[10:15] Coverage trace of pinsamples\basn3p04.png...done.  
[11:15] Coverage trace of pinsamples\basn3p08.png...done.  
[12:15] Coverage trace of pinsamples\basn4a08.png...done.  
[13:15] Coverage trace of pinsamples\basn4a16.png...done.  
[14:15] Coverage trace of pinsamples\basn6a08.png...done.  
[15:15] Coverage trace of pinsamples\basn6a16.png...done.  
  
[*] Finished  
[*] Running coverage analysis...  
[-] 3 files were selected from a total of 15.  
[*] Copying over selected files...  
[-] pinsamples\basn3p08.png -> minset\basn3p08.png  
[-] pinsamples\basn3p04.png -> minset\basn3p04.png  
[-] pinsamples\basn2c16.png -> minset\basn2c16.png  
  
[*] Finished
```

6.5. Peach Multi-Node CLI Tool

This tool is used to control multiple Peach instances at once. The tool can be used via the command line or as an interactive tool.

The tool utilizes the Peach REST API to perform all actions.

6.5.1. Installation

Installation of this tool has two steps.

Install Python 2.7

Python v2.7 is recommended. Other versions may also work.

Install dependencies

```
easy_install requests  
easy_install cmd2
```

Populate instances.py

The instances.py file contains a list of all Peach instances that will be controlled from this tool. Instances can be placed into groups. An instance can be part of more than one group.

Configurations are pulled from a master instance configured in instances.py. Typically the master instance is running locally, but it can also be one of the fuzzing instances.



Only one master instance can be configured!

6.5.2. Syntax

```
peachcli  
peachcli "jobs all" quit
```

6.5.3. Commands

jobs: View all jobs

```
jobs <group>  
jobs all
```

Show basic job information from each instance in a group.

Example 6. Example

```
# peachcli.py

| Peach CLI v0.1
| Copyright (c) Peach Fuzzer, LLC

>> jobs all

-- http://192.168.48.128:8888 --
Name          Status      Start           Stop        Count
Faults
-----
-----
HTTP Server-Test    stopped   2016-01-12T23:13:11Z 2016-01-12T23:13:30Z 200
-
HTTP Server-Test    stopped   2016-01-13T00:27:02Z 2016-01-13T00:54:05Z 781
-

-- http://192.168.48.129:8888 --
Name          Status      Start           Stop        Count
Faults
-----
-----
HTTP Server-Test    stopped   2016-01-12T23:13:14Z 2016-01-12T23:13:25Z -
-
HTTP Server-Test    stopped   2016-01-13T00:27:04Z 2016-01-13T00:48:35Z 13400
112

>>
```

pause: Pause a set of jobs

```
pause <pit-config> <group>
pause HTTP_Server-Test all
```

Pause pit configuration jobs running on a specified group.

pull: Pull fault details

```
pull faults <pit-config> <group>  
  
pull faults HTTP_Server-Test all
```

Pull fault details for a specific pit configuration. Faults from all jobs in group will be collected. The resulting faults are placed into a **faults** folder, organized by risk, bucket, and node/test case number.

Example 7. Example

```
# peachcli.py  
  
| Peach CLI v0.1  
| Copyright (c) Peach Fuzzer, LLC  
  
>> pull faults HTTP_Server-Test all  
Pulling faults of HTTP_Server-Test for group all:  
  
Pulling 112 faults from http://192.168.48.129:8888...  
  
>> quit  
  
# ls faults\HTTP_Server-Test\EXPLOITABLE\F0FF8D9D\1395E24B\  
192.168.48.129_13290 192.168.48.129_13309 192.168.48.129_13328  
192.168.48.129_13347 192.168.48.129_13366 192.168.48.129_13385  
192.168.48.129_13291 192.168.48.129_13310 192.168.48.129_13329  
192.168.48.129_13348 192.168.48.129_13367 192.168.48.129_13386  
192.168.48.129_13292 192.168.48.129_13311 192.168.48.129_13330  
192.168.48.129_13349 192.168.48.129_13368 192.168.48.129_13387  
192.168.48.129_13293 192.168.48.129_13312 192.168.48.129_13331  
192.168.48.129_13350 192.168.48.129_13369 192.168.48.129_13388  
192.168.48.129_13294 192.168.48.129_13313 192.168.48.129_13332  
192.168.48.129_13351 192.168.48.129_13370 192.168.48.129_13389  
192.168.48.129_13295 192.168.48.129_13314 192.168.48.129_13333  
192.168.48.129_13352 192.168.48.129_13371 192.168.48.129_13390  
192.168.48.129_13296 192.168.48.129_13315 192.168.48.129_13334  
192.168.48.129_13353 192.168.48.129_13372 192.168.48.129_13391  
192.168.48.129_13297 192.168.48.129_13316 192.168.48.129_13335  
192.168.48.129_13354 192.168.48.129_13373 192.168.48.129_13392  
192.168.48.129_13298 192.168.48.129_13317 192.168.48.129_13336  
192.168.48.129_13355 192.168.48.129_13374 192.168.48.129_13393  
192.168.48.129_13299 192.168.48.129_13318 192.168.48.129_13337  
192.168.48.129_13356 192.168.48.129_13375 192.168.48.129_13394  
192.168.48.129_13300 192.168.48.129_13319 192.168.48.129_13338
```

192.168.48.129_13357	192.168.48.129_13376	192.168.48.129_13395
192.168.48.129_13301	192.168.48.129_13320	192.168.48.129_13339
192.168.48.129_13358	192.168.48.129_13377	192.168.48.129_13396
192.168.48.129_13302	192.168.48.129_13321	192.168.48.129_13340
192.168.48.129_13359	192.168.48.129_13378	192.168.48.129_13397
192.168.48.129_13303	192.168.48.129_13322	192.168.48.129_13341
192.168.48.129_13360	192.168.48.129_13379	192.168.48.129_13398
192.168.48.129_13304	192.168.48.129_13323	192.168.48.129_13342
192.168.48.129_13361	192.168.48.129_13380	192.168.48.129_13399
192.168.48.129_13305	192.168.48.129_13324	192.168.48.129_13343
192.168.48.129_13362	192.168.48.129_13381	192.168.48.129_13400
192.168.48.129_13306	192.168.48.129_13325	192.168.48.129_13344
192.168.48.129_13363	192.168.48.129_13382	192.168.48.129_13401
192.168.48.129_13307	192.168.48.129_13326	192.168.48.129_13345
192.168.48.129_13364	192.168.48.129_13383	
192.168.48.129_13308	192.168.48.129_13327	192.168.48.129_13346
192.168.48.129_13365	192.168.48.129_13384	

push: Push pit configuration

```
push <pit-config> <group>
```

```
push HTTP_Server-Test all
```

The push command will copy a configuration from the master instance to a group of remote instances. The pit configuration name must follow a specific naming convention to use this command. The name has two parts. The first part is the source pit name, for example "HTTP_Server". The second part is the configuration name, for example "Test". They are joined with a hyphen (-). The resulting name would be "HTTP_Server-Test".

status: Status of all related jobs

```
status <pit-config> <group>
```

```
status HTTP_Server-Test all
```

Collect information about all jobs for a specific pit configuration.

Example 8. Example

```
# peachcli.py

| Peach CLI v0.1
| Copyright (c) Peach Fuzzer, LLC

>> status HTTP_Server-Test all
Status of HTTP_Server-Test for group all:

Nodes     Running   Stopped   Paused   Count   Faults
-----
2          0         4         0        14381   112

>>
```

start: Start a new set of jobs

```
start <pit-config> <group>
start HTTP_Server-Test all
```

Start a new job using the specified pit configuration on all instances in the specified group.

stop: Stop a set of jobs

```
stop <pit-config> <group>
stop HTTP_Server-Test all
```

Stop pit configuration jobs running on a specified group.

6.6. Pit Tool - Ninja

This tool is used when adding samples to an existing Pit or creating a custom file fuzzing Pit.

Sample Ninja reads sample files used in fuzzing, runs them through the Peach data crackers (which applies sample files to the data models), and places them into a database. During fuzzing, a sample ninja mutator mixes-and-matches file sections to create a new file.



Sample Ninja is not available in the Peach Community version.

The `pittool ninja` command produces and maintains the Sample Ninja database used by the Sample Ninja mutator.

To fuzz using Sample Ninja:

- Generate a Sample Ninja database using `pittool ninja`.
- The Sample Ninja mutator automatically uses the database the next time Peach is run.
- When new or modified sample files are needed, re-run `pittool ninja` to update the database.

The generated database has a name like `PIT.ninja`. This means that if your pit file is `png.xml`, the generated database file is `png.ninja`.

6.6.1. Syntax

Usage:

```
PitTool.exe ninja <pitPath> <dataModel> <samplesPath>
```

Description:

Create a sample ninja database.

General Options:

<code>-h, --help</code>	Display this help and exit
<code>-V, --version</code>	Display version information and exit
<code>-v, --verbose</code>	Increase verbosity, can use multiple times
<code>--plugins=VALUE</code>	Specify the plugins path
<code>--pits=VALUE</code>	Specify the PitLibraryPath.

6.6.2. Parameters

PitPath

Fuzzing definition that refers to sample files.

DataModel

The DataModel used to crack each sample.

SamplesPath

The path to a folder containing the sample files to be used during fuzzing.

6.6.3. Examples

Example 9. Creating a Sample Ninja database

1. Open a new terminal window.
2. Run `pittool < PitPath > < DataModel > < SamplesPath >`.
3. The Sample Ninja database will be generated.

```
> pittool ninja pits\Image\PNG.xml PNG:PNG:File samples_png
Processing: samples_png\ajou_logo.png
Processing: samples_png\apollonian_gasket.png
Processing: samples_png\aquarium.png
Processing: samples_png\baboon.png
...
Processing: samples_png\z00n2c08.png
Processing: samples_png\z03n2c08.png
Processing: samples_png\z06n2c08.png
Processing: samples_png\z09n2c08.png
```

Generated database:

03/17/2014 08:39 PM	9,035	PNG.xml
03/20/2014 03:52 PM	9,651,200	PNG.ninja

Example 10. Adding new samples to an existing Samples Ninja database

1. Put new samples into your samples folder.
2. Open a terminal window.
3. Run `pittool < PitPath> <DataModel> <SamplesPath>`.
4. The new and modified files will be added to the database.

```
> pittool ninja pits\Image\PNG.xml PNG:PNG:File samples_png
Skipping: samples_png\ajou_logo.png
Skipping: samples_png\apollonian_gasket.png
Skipping: samples_png\aquarium.png
Skipping: samples_png\baboon.png
...
Skipping: samples_png\z00n2c08.png
Skipping: samples_png\z03n2c08.png
Skipping: samples_png\z06n2c08.png
Skipping: samples_png\z09n2c08.png
Processing: samples_png\zzzz.png
```

Generated database:

03/17/2014 08:39 PM	9,035	PNG.xml
03/20/2014 03:52 PM	9,651,200	PNG.ninja

7. Monitoring Recipes

Peach Fuzzer has very powerful monitoring capabilities that detect faults that occur in the test target, collect data surrounding each fault, and automate the test environment. This section provides some suggested monitoring configurations for different types of targets. These recipes can be used as-is or as starting points for a custom configurations.

All of the recipes here follow a common process:

1. Define the workflow for the fuzzing session.
2. Define the Peach components to use in configuring the fuzzing setup. This step focuses on the monitoring needs and the agents that house the monitors.
3. Provide configuration settings used to fuzz a sample test target using the recipe.

NOTE: Assumptions/Givens in each recipe include the following:



- A Pit, pre-defined or custom, is ready to use.
- Peach is installed and ready to run.
- All software modules needed to perform the fuzzing job are available for use.

What's left to do?

When you finish developing a recipe, it's time to fill out and test the configuration, then run the fuzzing job.

- See the [Getting Started with the Peach Fuzzer Platform](#) for the workflow to run a fuzzing job.

The remainder of this section provides the following monitor recipes:

- [Monitoring a File Consumer \(File Format\)](#)
- [Monitoring a Linux Network Service](#)
- [Monitoring a Linux Network Client](#)
- [Monitoring a Network Device](#)
- [Monitoring a Windows Network Service](#)
- [Monitoring a Windows Network Client](#)

7.1. Recipe: Monitoring a File Consumer (File Fuzzing)

This recipe identifies the monitors and associated settings suitable for testing a file consumer. When fuzzing a file consumer, Peach creates malformed data files, and then has the file consumer open and access the data from the malformed files.



Using one common approach, Peach can fuzz file consumers of all types, whether the file types are multimedia—sound, images, or video; documents—word processing, text, or presentation; or other types of files, such as archive or compressed data files.

7.1.1. What is the fuzzing session workflow?

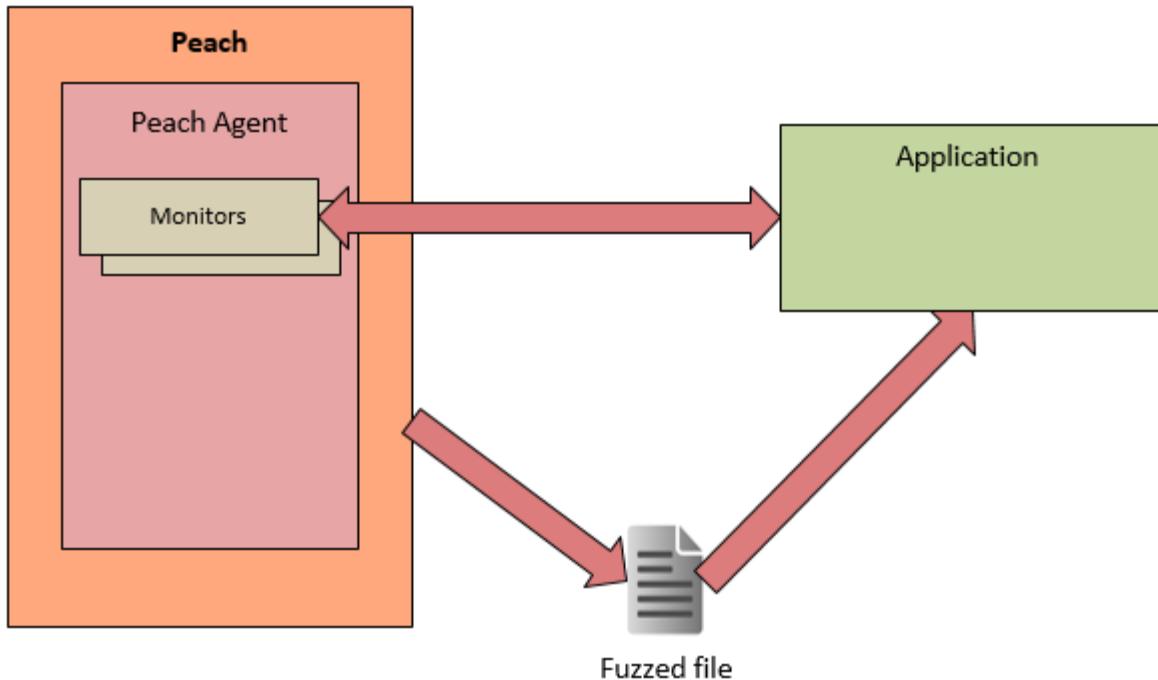
The workflow that we want to follow in the test consists of the following steps:

1. Perform Fuzzing.
 - a. Write a file to disk that contains fuzzed data.
 - b. Run the target application.
 - c. Wait for the target to consume the data file.
 - d. Check for faults.
 - e. If a fault occurs, collect data.
2. Repeat step 1.

The monitoring setup for this recipe is simple because a single monitor, the Windows Debugger monitor, provides most of the functionality needed, including: fault detection, data collection, and automation.

A second monitor, the PageHeap monitor, enables debug memory allocation settings for the target process. PageHeap can improve the likelihood that certain types of issues, such as buffer overflow, will cause the target to crash. Using the debug memory allocation settings normally leads to finding more faults during testing.

Here's a diagram of connections for the test configuration.



7.1.2. Setting up Monitors Using the Peach Web UI

Starting Peach

1. Start a Windows command line session with administrative privileges.
Right-click on the command prompt on the Windows menu to find the "As Administrator" setting.
PageHeap requires administrative (heightened) privileges.
2. Launch Peach from the command line (type `peach` and press the Return key) to start the UI.
3. Select a pit (test definition) of a protocol supported by the network device, such as BMP, so that Peach can communicate with device protocol during fuzzing.
 - Give the pit a name and a description. Peach makes a configuration file of the selections you make, so that you can re-use the setup again.
4. From the configuration menu along the left edge of the window, select Monitoring.
5. Fill in a name for the agent. Since this agent resides within Peach, the default location `local` is appropriate.

local:// (Local)

Name	<input type="text"/> Local
Location	<input type="text"/> local://
Warning! At least one monitor is advised.	
Add a monitor... ▾	

Supplying Monitor Details

Begin each monitor with a name or descriptive text. This helps identify one monitor from another.

Next, fill in the critical parameters for each monitor. These parameters have callouts in the settings diagram of each monitor. Details for these parameters are given in the text that follows.



The order of the monitors listed in the agent is significant. Peach processes the monitors in the order listed (from top to bottom). For example, a blocking situation or incorrect test results might occur if the sequencing is incorrect. In fact, PageHeap sets some settings for the Windows Debugger, so you need to declare the PageHeap monitor first.

For this recipe, use the monitors in the order they are presented:

- PageHeap (Fault Assist)
- WinDebugger (WaitforBootOnStartAndFault)

PageHeap Monitor (FaultAssist)

The [PageHeap Monitor](#) enables heap allocation monitoring for an executable through the Windows debugger. Peach sets and clears the parameters used for monitoring heap allocation at the beginning and end of the fuzzing session.

PageHeap (FaultAssist)

Name	<input type="text"/> FaultAssist
Executable	<input type="text"/> ##Executable##
Win Dbg Path	<input type="text"/>

The **Executable** parameter identifies the application or executable file that is the fuzzing target. Specify the file name, with the file extension. The path is not needed nor wanted.

Windows Debugger Monitor (FaultDataAndAutomation)

The [Windows Debugger Monitor](#) controls a Windows debugger instance. This monitor launches an executable file, a process, a service, or a kernel driver with the debugger attached; or, this monitor can attach the debugger to a running executable, process, or service. This monitor can also attach the debugger to kernel-mode drivers, although that lies outside the scope of this documentation.

WindowsDebugger (FaultDataAndAutomation)

Name	<input type="text" value="FaultDataAndAutomation"/>
Arguments	<input type="text" value="##FuzzedFile##"/> 
Cpu Poll Interval	<input type="text" value="200"/>
Executable	<input type="text" value="##Executable##"/> 
Fault On Early Exit	<input type="text" value="false"/>
Ignore First Chance Guard Page	<input type="text" value="false"/>
Ignore Second Chance Guard Page	<input type="text" value="false"/>
Kernel Connection String	<input type="text"/>
No Cpu Kill	<input type="text" value="false"/>
Process Name	<input type="text"/>
Restart After Fault	<input type="text" value="false"/>
Restart On Each Test	<input type="text" value="false"/>
Service	<input type="text"/>
Start On Call	<input type="text" value="ExitIterationEvent"/> 
Symbols Path	<input type="text" value="SRV*http://msdl.microsoft.com/download/symbols"/>
Wait For Exit On Call	<input type="text"/>
Wait For Exit Timeout	<input type="text" value="10000"/>
Win Dbg Path	<input type="text"/>

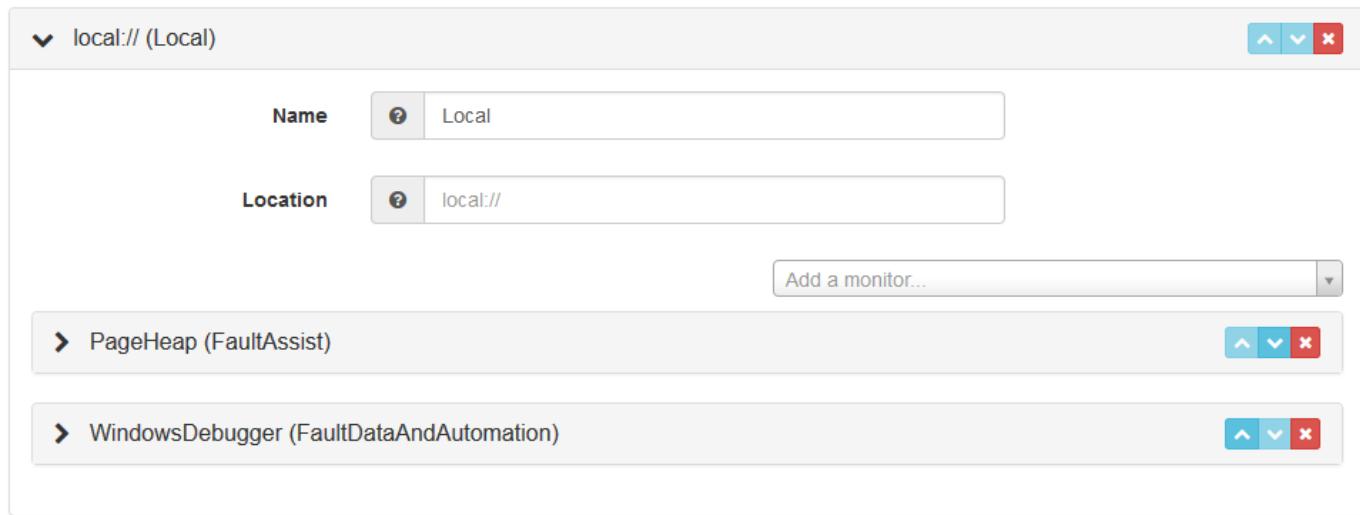
The **Arguments** parameter identifies command-line arguments for the executable file. The value of this parameter provides command-line switches and data file names needed to launch the executable file.

The **Executable** parameter identifies the application or executable file to launch via the debugger. If the executable has command-line arguments, specify these using the "Arguments" parameter.

The **StartOnCall** parameter defers launching the target until the state model issues a call to the monitor to begin. Upon receiving the call, the debugger starts the process.

7.1.3. Summary

When fuzzing files on Windows, you need two monitors: PageHeap for heap allocation and the Windows Debugger that performs the vast majority of work. You also need to launch Peach from an administrative console. A snapshot of the agent setup follows:



When ready, test the configuration. See [Text Pit Configuration](#) for more information.

Also, see [Fuzzing Session](#) for information on running a fuzzing job.

7.2. Recipe: Monitoring a Linux Network Service Client

This recipe describes the base setup needed to fuzz a client of a Linux network service. When fuzzing a network client, Peach impersonates the network service, the other endpoint of the network connection.

The recipe is a model that you can follow closely. Or, use the model as a starting point and augment the model for your specific situation. This recipe consists of the following parts:

1. The workflow for the fuzzing session
2. The Peach components to use in configuring the fuzzing setup. This section focuses on the monitoring needs and the agents that house the monitors.
3. Configuration settings used to fuzz a service client (`sntpget`) running this workflow



Assumptions/Givens in this recipe are that a Pit is ready to use, Peach is ready to run, and any software module needed to perform the fuzzing job is installed.

7.2.1. Workflow for the Fuzzing Session

The workflow lists the task sequence that occurs when running the fuzzing session. The setup needed to implement the workflow follows in the next section. Start with defining the workflow, especially if you plan to embellish the recipe.

Here is the workflow that Peach performs in fuzzing a Linux network service client:

1. Revert to a virtual machine snapshot.
2. Wait for the machine to boot up.
3. Perform fuzzing. Create and run test cases.
 - a. Peach launches the network client. The client initiates contact with the server and sends requests to the server.
 - b. Peach impersonates the server and replies to client queries. Query responses contain fuzzed data.
 - c. Perform fault detection on the client.
 - d. If a fault occurs, collect data surrounding the test case.
 - e. Revert to the VM snapshot.
4. Loop to step three (resume fuzzing).

7.2.2. Peach Components Needed in the Fuzzing Configuration

Defining the Peach components divides in two parts: identifying the monitors to use in the configuration and identifying where to locate the specified monitors.

Identifying Monitors

This part of the recipe revisits each step of the workflow to identify the monitors needed to implement the configuration:

1. Revert to a snapshot of a virtual machine.

Peach needs to automate the test environment and remove human interaction during the fuzzing job. We place the service in a virtual machine (VM) because Peach can use a VM monitor to automatically start and reset the test environment when needed.

The VM snapshot is taken while the guest OS and the Peach agent are running. Using such a snapshot avoids the wait time associated with booting up the virtual machine. Also, the same snapshot is used when Peach refreshes the test environment after a fault occurs.

The monitor for the VM environment, [VMware](#) monitor, resides on the host machine.

2. Wait for the machine to boot up.

Peach waits for the VM snapshot to resume.

3. Perform fuzzing, checking for faults.

a. Launch the network client.

The Peach agent launches the client service through GDB. The client submits requests from the remote machine. The debugger and the client both reside on the remote machine.

b. Reply to the client request with fuzzed data.

Peach impersonates the network server and sends fuzzed replies in response to client queries.

c. Perform fault detection in the VM.

The GDB monitor watches the internals of the services and detects faults such as access violations and exceptions. Again, the debug monitor is located on the same machine as the service.

d. Collect data surrounding each fault as it happens.

When a fault occurs, the packets involved with the fault are interesting. Peach captures the packets using a network capture monitor. This monitor resides on the local machine with Peach Fuzzer.

Peach collects log files generated by the debugger located on the remote machine where the service resides. A monitor that saves files sends files from a specified folder on the remote system to the Peach logging repository on the local machine. The monitor to save the log files is located on the remote machine.

4. Revert to the VM snapshot.

This step uses the VM monitor and VM snapshot from step 1 to refresh the test environment, and the debug monitor from step 3 to start the network service in the refreshed environment. No additional monitors are needed for this step.

5. Resume fuzzing.

Loop to step 3.

Identifying Agents

Peach offers two types of agents to manage monitors and I/O publishers: local and remote.

- Local agents reside inside Peach.

The local agent in this recipe addresses automation involving the VM and data collection that captures network packets. The local agent houses the [VMware](#) and the [NetworkCapture](#) monitors.

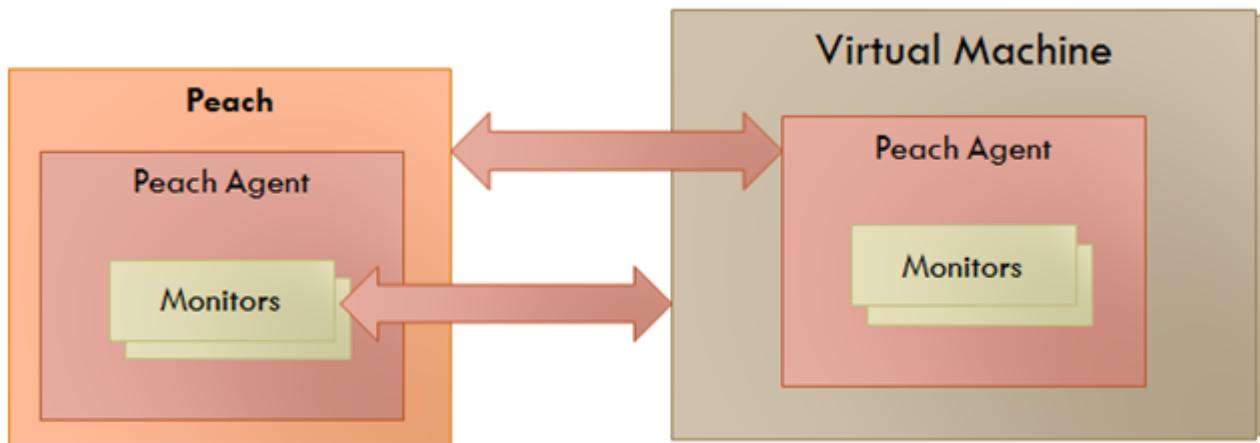
The VMware monitor starts a snapshot VM environment at the beginning of the fuzzing job, as well as restarting the same VM snapshot after a fault occurs.

- Remote agents reside in separate processes on remote machines with the test targets.
In this case, the remote agent and the Linux service reside on the same machine.

The remote agent houses the [Gdb](#) debug monitor that starts the Linux service client at the beginning of the fuzzing job and restarts the service in the refreshed environment after a fault. The Gdb debug monitor detects faults that occur in the client.

In addition, a data collection monitor collects log files after a fault occurs in the network client. The [Save File](#) monitor forwards the log files to the logging repository.

The result is that we end up with the following configuration:



Peach is located on one machine with a local agent that houses the VM monitor and the Network capture monitor. A second agent resides on the remote machine with the service. The remote agent houses the Gdb debug monitor and the SaveFile monitor.

The local agent is simple to implement. All that's needed is to define the agent, then specify the appropriate monitors and monitor settings used with the local agent.

The remote monitor is a little more involved. Like the local agent, the remote agent needs to be defined, then specify the appropriate monitors and monitor settings used with the remote agent. Second, the remote agent needs to run on the same OS as the test target. This step can be done separately from specifying the configuration details. In this recipe, a VM snapshot is used. See [Using Virtual Machines](#), for information on setting up the VM snapshot.

7.2.3. Sample Network Client Configuration

This section shows the recipe implemented for a network service client and consists of the following items:

- Setup on the Target VM Image
- Pit variables
- Peach agents
- Peach monitors
- Debug monitor "No Cpu Kill" parameter
- Configuration test

The configurations for the network client and the network service are very similar. Two significant differences exist:

- The network client configuration uses a client application instead of the network service.
- In the network client configuration, the test target initiates the action instead of responding to a request. The client contacts Peach, acting as the network service, then waits for Peach to provide a response to the query. The debug monitor has additional configuration options that are set to drive this configuration.



Setup on the Target VM Image

Perform the following task on the VM before taking a snapshot of the VM.

- Run the Peach agent from a shell with root access.

Within the shell, navigate to the peach folder and execute the following command:

```
./peach -a tcp
```

When Peach starts the VM, the Peach agent is running in a root shell.

Pit Variables

The following UI display identifies data values typically needed by a network protocol Pit. The variables and values are independent of the monitors used in the configuration. Pit variables are unique to the Pit and might differ with those in the example illustration.

Variables

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

Saved successfully.

 Save  + Add Variable

Name	Key	Value	Remove
.	.		
.	.		
.	.		
Community String	CommString	 public	
Source IPv4 Address	SourceIPv4	 192.168.127.1	
Source Port	SourcePort	 161	
Timeout	Timeout	 5000	

The Pit User Guides describe the Pit-specific variables.

Community String (Authentication)

Community string used for authentication. Peach and the network client must use the same community string. Check the server documentation for consistency of this value. If needed, change the value here to coincide with the value expected by the test target.

Source Port

Port number of the local machine that sends packets to the server. Several services use well-known ports that usually can be left unedited.

Target IPv4 Address

IPv4 address of the target machine (client). For information on obtaining the IPv4 address, see Retrieving Machine Information section of the Pit User Guide.

Target Port

SNMP port number of the remote machine that sends and receives packets. Several services use well-known ports that usually can be left unedited.

Timeout

Duration, in milliseconds, to wait for incoming data. During fuzzing, a timeout failure causes the fuzzer to skip to the next test case.

Agents

The following UI diagram acts as an overview, showing the Peach agents and the monitors within each agent. Peach uses the ordering within the agent to determine the order in which to load and run monitors.

Monitoring

The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit.

Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers.

The screenshot shows the 'Monitoring' configuration interface. At the top right are 'Save' and '+ Add Agent' buttons. Below is a list of agents:

- local:// (LocalAgent)**:
 - Name: LocalAgent
 - Location: local://
 - Monitors: NetworkCapture (InterestingPackets) and Vmware (Remote Client Manager)
- tcp://##TargetIPv4## (Remote Client Manager)**:
 - Name: Remote Client Manager
 - Location: tcp://##TargetIPv4##
 - Monitors: Gdb (Debug Monitor for Client) and SaveFile (CollectLogs)

The local agent is defined first and lists the default information for both name and location. This definition for a local agent is typical and, otherwise, unremarkable. The NetworkCapture and Vmware monitors are independent of one another, allowing either monitor to top the list.

The remote agent, named "Remote Client Manager", has quite a different location specification. The location consists of concatenated pieces of information:

- Channel. The channel for a remote agent is `tcp`. A colon and two forward slashes separate the channel from the IPv4 address of the hardware interface.
- Target IPv4 address of the remote machine. The IPv4 address of the agent is the second component of the location. For more information, see the Retrieving Machine Information section of the Peach Pit User Guide.

The monitor list within each agent is significant, as the monitors are launched in order from top to bottom within an agent.

Monitors

This recipe uses four monitors, two on the machine with Peach and two on the remote machine. The recipe shows each monitor and describes its roles: fault detection, data collection, and automation.

Vmware (Remote Client Manager)

The `Vmware` monitor controls setting up and starting the virtual machine.

Vmware (Remote Client Manager)

Name	<input type="text"/> Remote Client Manager	
Vmx	<input type="text"/> \\Virtual Machines\\Ubuntu 64-bit_2\\Ubuntu 64-bit_2.vmx	
Headless	<input type="text"/> false	
Host	<input type="text"/>	
Host Port	<input type="text"/> 0	
Host Type	<input type="text"/> Workstation	
Login	<input type="text"/>	
Password	<input type="text"/>	
Reset Every Iteration	<input type="text"/> false	
Reset On Fault Before Collection	<input type="text"/> false	
Snapshot Index	<input type="text"/>	
Snapshot Name	<input type="text"/> Snap_Cli1	
Stop On Fault Before Collection	<input type="text"/> false	
Wait For Tools In Guest	<input type="text"/> true	
Wait Timeout	<input type="text"/> 600	

The most significant parameters for the VMware monitor follow:

Vmx

Identifies the full path of the virtual machine image. Peach loads the snapshot of the VM image at the start of the fuzzing job and after a fault occurs.

Headless

Identifies whether the VM has a window associated with it. When developing a configuration, set this parameter to false. When the configuration is complete, change Headless to true.

Host Type

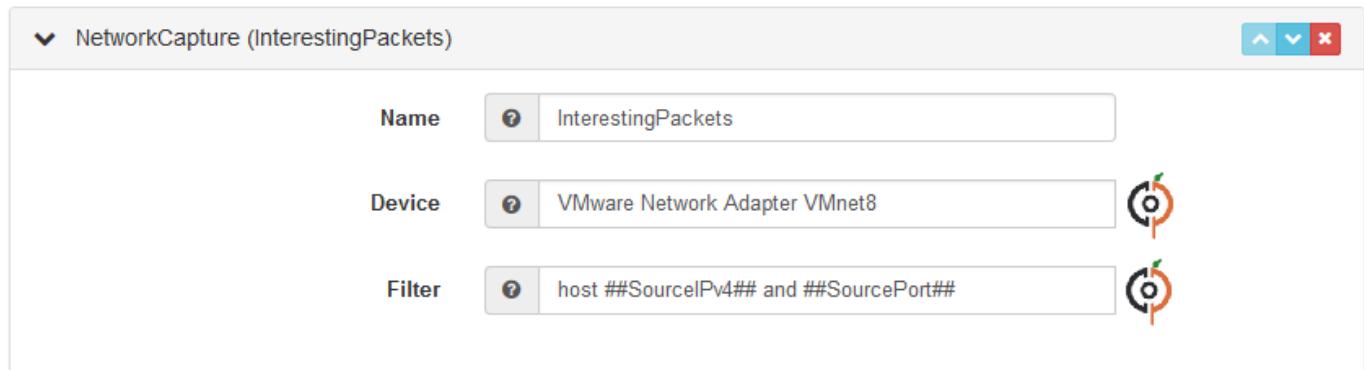
Specifies the VMware product used in the configuration.

Snapshot Name

Identifies the snapshot to use for the specific image.

Network Capture (InterestingPackets)

The [Network Capture Monitor](#) captures network packets sent and received from the test target. When a fault occurs, Peach stores the packets immediately surrounding the fault in the log of the test case.



The most significant parameters for the network capture monitor follow:

Device

Specifies the name of the interface on the local machine (the machine with Peach) used to communicate with the test target. Use [ifconfig](#) to identify the interface(s) available for use.

You can find the appropriate host interface that communicates with the VM using the following steps:



1. Collect a list of interfaces (and their IPv4 addresses) by running [ipconfig](#) or [ifconfig](#).
2. Test each interface in the list. Manually run a capture session with Wireshark using an interface from the list.
3. On the host machine, Ping the target IPv4 (of the VM).
4. If the correct interface of the host is used, you'll see the Ping request and reply packet exchanges through Wireshark,
5. Loop to step 2 and repeat, using another interface.

Filter

Helps capture only those packets associated with the fuzzing session. The filter adheres to the syntax and requirements of the Pcap filter specification.



WireShark refers to the Libpcap filters as capture filters. Use the capture filters. Wireshark also defines its own display filters that it uses to filter entries in its session files. The display filters are not compatible with Libpcap.

GDB (Debugger)

The **GDB** debugger monitor performs two main functions in this recipe:

- Starts the network client at the start of a fuzzing job and restarts the client when the VM snapshot refreshes.
- Detects faults internal to the client.

The Gdb monitor uses the settings in the following illustration:

The screenshot shows the configuration dialog for the Gdb monitor. The title bar says "Gdb (Debug Monitor for Client)". The dialog contains the following fields:

Name	Value
Name	Debug Monitor for Client
Executable	##svc_filename##
Arguments	##svc_filename_args##
Fault On Early Exit	false
Gdb Path	/usr/bin/gdb
No Cpu Kill	true
Restart After Fault	false
Restart On Each Test	false
Start On Call	StartIterationEvent
Wait For Exit On Call	
Wait For Exit Timeout	10000

The most significant parameters follow:

Executable

Identifies the full path to the Linux service client. The client resides on the remote machine; so, the full path is for the Linux file system.

Arguments

Arguments for the executable.

No Cpu Kill

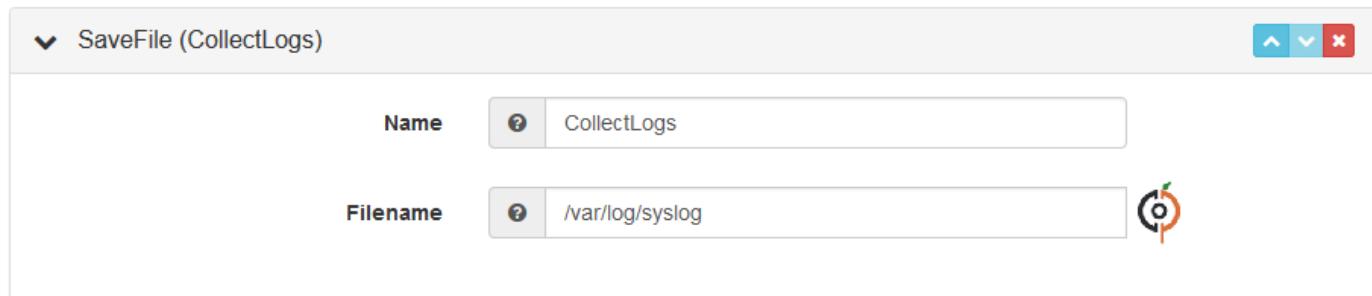
Controls whether the process stays alive if its CPU usage drops to zero. Specify `true` to keep the process running and to allow the process to release or close its resources before exiting. For more information, see the following section *Closing the Client Process*.

Start On Call

Controls when the test target launches, and in turn, initiates contact with the service (Peach). Specify `StartIterationEvent` to launch the client at the start of the test case.

SaveFile (CollectLogs)

The `SaveFile` monitor collects log files from the remote test target and copies them to the Peach Logging folder. The monitor is housed by the remote agent.



The most significant parameter follows:

Filename

Specifies the full path to the Linux logging system used by GDB.

7.2.4. Closing the Client Process

In this recipe, the Peach debug monitor launches the network service client using the "Start On Call" parameter so that the client initiates contact with the server. Then, at the end of the test case after execution complete, the "No Cpu Kill" parameter provides control of how the client closes:

- If "No Cpu Kill" is `true`, Peach waits for the process to exit OR for a time to elapse specified by the "Wait For Exit Timeout" parameter.
- If "No Cpu Kill" is `false`, Peach waits for the CPU usage of the process to reach zero percent OR for the process to exit OR for a time to elapse specified by the "Wait For Exit Timeout" parameter. The "No Cpu Kill" parameter default setting is `false`.



If the waiting period ends, peach kills the target process if it is still running and starts the next iteration.

So, when do you need to let a process with zero CPU activity continue to execute?

Set “No Cpu Kill” to **true** when you’re fuzzing a network service client. In this scenario, Peach Fuzzer starts the network client using the “Start on Call” parameter to initiate contact with the service. When the client receives and processes the reply, Peach waits for the client to run to completion and watches for any faults that occur before the client exits.

Scenarios exempt from the "No Cpu Kill" option include the following:

- Fuzzing network service servers typically do not use the “Start On Call” option, so the “No Cpu Kill” option isn’t needed.
- Fuzzing file formats require “Start on Call” to start the fuzzing target once the fuzzed data file is generated. The “No Cpu Kill” parameter can be used here; however, Peach can save time that will be replicated in each test case by letting the process terminate if the CPU usage falls to zero. In this case, not using “No Cpu Kill” is a performance optimization.
- Embedded devices. Fuzzing configurations for these devices do not use the Peach debugging monitors: GDB, WindowsDebugger, or CrashWrangler. So, the “No Cpu Kill” option isn’t needed.
- Kernel-mode debugging. Kernel-mode debugging has its own set of requirements. “No Cpu Kill” is not used here.

Configuration Test

Once the monitors and associated parameters are part of the configuration, you can test the configuration. From the Configuration menu along the left edge of the window, click on **Test** to run a single iteration (test case) on the configuration. Note that the test checks the connections and communications. It does NOT do any fuzzing.

For more information on testing a configuration, see [Test Pit Configuration](#).

7.3. Recipe: Monitoring a Linux Network Service

This recipe describes the base setup needed to fuzz a Linux network service. When fuzzing a network server, Peach impersonates the client, the other endpoint of the network connection.

The recipe is a model that you can follow closely. Or, you can use the model as a starting point and augment the model for your specific situation. This recipe consists of the following parts:

1. The workflow for the fuzzing session
2. The Peach components to use in configuring the fuzzing setup This section focuses on the monitoring needs and the agents that house the monitors.
3. Configuration settings used to fuzz a sample service (SNMP) running this workflow



Assumptions/Givens in this recipe are that a Pit is ready to use, Peach is ready to run, and any software module needed to perform the fuzzing job is installed.

In this scenario, Peach runs on a host computer; the network server runs in a Virtual Machine (VM) on

the host. With Peach running on the host, it controls the environment. If the network server crashes, the worst thing that happens is that the virtual machine has to restart. Peach recover the data if the network server crashes.

7.3.1. Workflow for the Fuzzing Session

The workflow lists the task sequence that occurs when running the fuzzing session. The setup needed to implement the workflow follows in the next section. Start with defining the workflow, especially if you plan to embellish the recipe.

Here is the workflow that Peach performs in fuzzing a Linux network service:

1. Revert to a virtual machine snapshot.
2. Wait for the machine to boot up.
3. Launch the network service.
4. Perform fuzzing. Create and run test cases.
 - Peach initiates contact with the server and sends packets of fuzzed data to the server.
 - Check for faults, such as crashes and access violations.
5. When a fault occurs, do the following:
 - a. Collect data surrounding the test case.
 - b. Revert to the VM snapshot.
 - c. Launch the network service.
6. Loop to step 4 (resume fuzzing).

7.3.2. Peach Components Needed in the Fuzzing Configuration

Defining the Peach components divides in two parts: identifying the monitors to use in the configuration and identifying where to locate the specified monitors.

Identifying Monitors

This part of the recipe revisits each step of the workflow to identify the monitors needed to implement the configuration:

1. Revert to a snapshot of a virtual machine.

Peach needs to automate the test environment and remove human interaction during the fuzzing job. We place the service in a virtual machine (VM) because Peach can use a VM monitor to automatically start and reset the test environment when needed.

The VM snapshot is taken while the guest OS and the Peach agent are running. Using such a snapshot avoids the wait time associated with booting up the virtual machine. Also, the same snapshot is used when Peach refreshes the test environment after a fault occurs.

The monitor for the VM environment, [VMware](#) monitor, resides on the host machine.

2. Wait for the machine to boot up.

Peach waits for the VM snapshot to resume.

3. Launch the network service.

The Peach agent in the VM starts the network service via the [Gdb](#) debugging monitor and the GDB system debugger.

4. Perform fuzzing, checking for faults.

Perform fault detection in the VM. The Gdb monitor watches the internals of the services and detects faults such as access violations and exceptions. Again, the debug monitor is located on the same machine as the service.

5. Collect data surrounding each fault as it happens.

Peach sends and receives network packets to the service. When a fault occurs, the packets involved with the fault are interesting. Peach captures the packets using the [NetworkCapture](#) monitor.

Peach collects log files generated by the debugger located on the remote machine where the service resides. A [Save File](#) monitor sends files from a specified folder on the remote system to the Peach logging repository on the local machine.

6. Resume fuzzing.

This step uses the VM monitor and VM snapshot from step 1 to refresh the test environment, and the debug monitor from step 3 to start the network service in the refreshed environment. No additional monitors are needed for this step.

Identifying Agents

Peach offers two types of agents to manage monitors and I/O publishers: local and remote.

- Local agents reside inside Peach.

The local agent in this recipe addresses automation involving the VM and data collection that captures network packets. The local agent houses the [VMware](#) and the [NetworkCapture](#) monitors.

The VMware monitor starts a snapshot VM environment at the beginning of the fuzzing job, as well as restarting the same VM snapshot after a fault occurs.

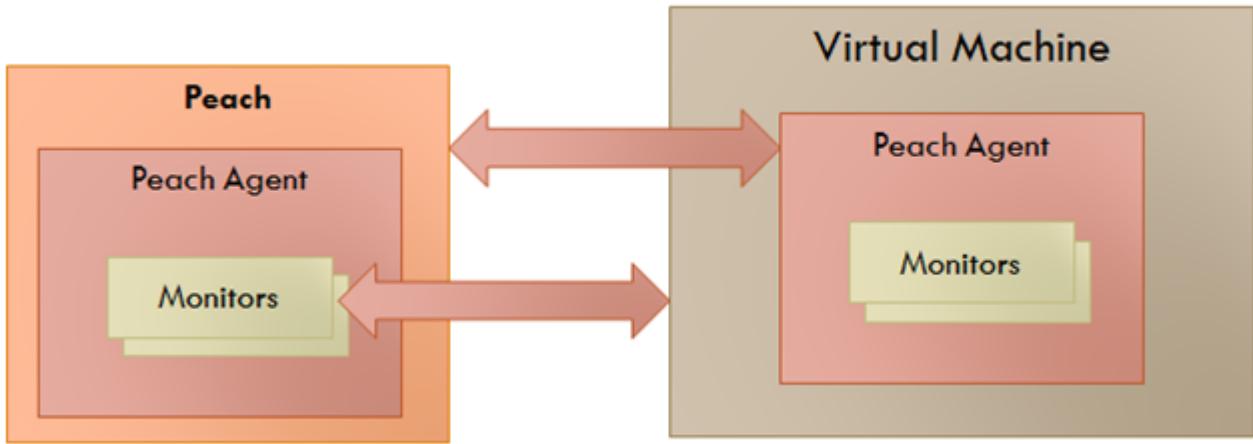
- Remote agents reside in separate processes on remote machines with the test targets.
In this case, the remote agent and the Linux service reside on the same machine.

The remote agent houses the [Gdb](#) debug monitor that starts the network service at the beginning of the fuzzing job and restarts the service in the refreshed environment after a fault. The Gdb debug

monitor detects faults that occur in the service.

In addition, a data collection monitor collects log files after a fault occurs in the network service. The [Save File](#) monitor forwards the log files to the logging repository.

The result is that we end up with the following configuration:



Peach is located on one machine with a local agent that houses the VM monitor and the Network capture monitor. A second agent resides on the remote machine with the service. The remote agent houses the Gdb debug monitor and the SaveFile monitor.

The local agent is simple to implement. All that's needed is to define the agent, then specify the appropriate monitors and monitor settings used with the local agent.

The remote monitor is a little more involved. Like the local agent, the remote agent needs to be defined, then specify the appropriate monitors and monitor settings used with the remote agent. Second, the remote agent needs to run on the same OS as the test target. This step can be done separately from specifying the configuration details. In this recipe, a VM snapshot is used. See the previous section, Using Virtual Machines, for information on setting up the VM snapshot.

7.3.3. Sample configuration

This section shows the recipe implemented for a network service and consists of the following items:

- Setup on the Target VM Image
- Pit variables
- Peach agents
- Peach monitors
- Configuration test

Setup on the Target VM Image

Perform the following items on the VM before taking a snapshot of the VM.

1. Run the Peach agent from a shell with root access.

Within the shell, navigate to the peach folder and execute the following command:

```
./peach -a tcp
```

When Peach starts the VM, the Peach agent is running in a root shell.

2. In the VM, edit the configuration file to have the service listen for connections on all IPv4 interfaces.

3. Stop the service.

During fuzzing, the debugger (GDB) will start the service.

Pit Variables

The following UI display identifies data values typically needed by a network protocol Pit. The variables and values are independent of the monitors used in the configuration. Pit variables are unique to the Pit and might differ with those in the example illustration.

Variables

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

				 Save	 Add Variable
Name	Key	Value		Remove	
		:	:		
Community String	CommString	 public			
Source Port	SourcePort	 162			
Target IPv4 Address	TargetIPv4	 192.168.127.163			
Target Port	TargetPort	 161			
Timeout	Timeout	 1000			

The Pit User Guides describe the Pit-specific variables.

Community String (Authentication)

Community string used for authentication by the network server. Check the network service documentation for consistency of this value. If needed, change the value here to coincide with the value expected by the test target.

Source Port

Port number of the local machine that sends packets to the server. Several services use well-known ports that usually can be left unedited.

Target IPv4 Address

IPv4 address of the target machine (server). For information on obtaining the IPv4 address, see Retrieving Machine Information in the Pit documentation.

Target Port

Port number of the server that receives packets. Several services use well-known ports that usually

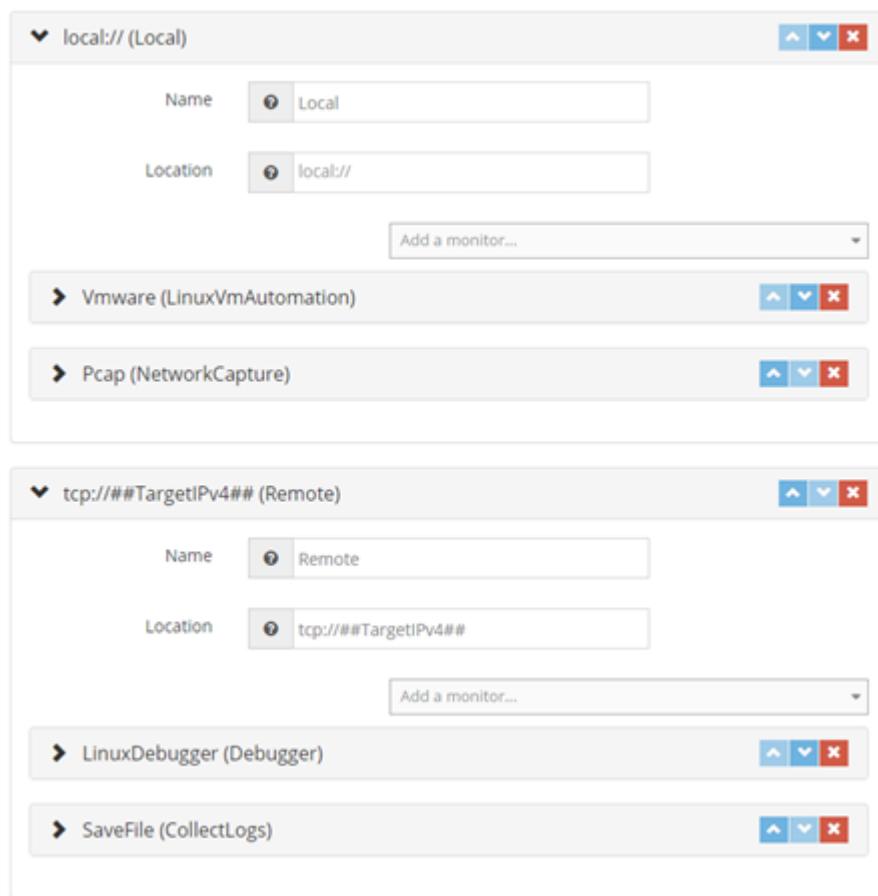
can be left unedited.

Timeout

Duration, in milliseconds, to wait for incoming data. During fuzzing, a timeout failure causes the fuzzer to skip to the next test case.

Agents

The following UI diagram acts as an overview, showing the Peach agents and the monitors within each agent. Peach uses the ordering within the agent to determine the order in which to load and run monitors.



The local agent is defined first and lists the default information for both name and location. This definition for a local agent is typical and, otherwise, unremarkable. The Vmware monitor, that starts the virtual machine, is the first monitor listed, as that action is not dependent on actions from another monitor.

The remote agent, named "Remote", has quite a different location specification. The location consists of concatenated pieces of information:

- Channel. The channel for a remote agent is **tcp**. A colon and two forward slashes separate the channel from the IPv4 address of the hardware interface.
- IPv4 address. The IPv4 address of the agent is the second component of the location. Use **ifconfig** to

find this address of the remote machine.

The monitor list within each agent is significant, as the monitors are launched in order from top to bottom within an agent.

Monitors

This recipe uses four monitors, two on the machine with Peach and two on the remote machine. The recipe shows each monitor and describes its roles: fault detection, data collection, and automation.

Vmware (Linux virtual machine Automation)

The [Vmware](#) monitor controls setting up and starting the virtual machine.

The screenshot shows the configuration dialog for a Vmware monitor named "LinuxVmautomation". The dialog contains the following fields:

Setting	Value	Status
Name	LinuxVmautomation	Valid
Vmx	C:\vms\linux\linux.vmx	Valid
Headless	false	Valid
Host	(empty)	Valid
Host Port	0	Valid
Host Type	Workstation	Valid
Login	(empty)	Valid
Password	(empty)	Valid
Reset Every Iteration	false	Valid
Reset On Fault Before Collection	false	Valid
Snapshot Index	(empty)	Valid
Snapshot Name	FuzzingServer	Valid
Stop On Fault Before Collection	false	Valid
Wait For Tools In Guest	true	Valid
Wait Timeout	600	Valid

The most significant parameters for the VMware monitor follow:

Vmx

Identifies the full path of the virtual machine image. Peach loads the snapshot of the VM image at the start of the fuzzing job and after a fault occurs.

Headless

Identifies whether the VM has a window associated with it. When developing a configuration, set this parameter to false. When the configuration is complete, change Headless to true.

Host Type

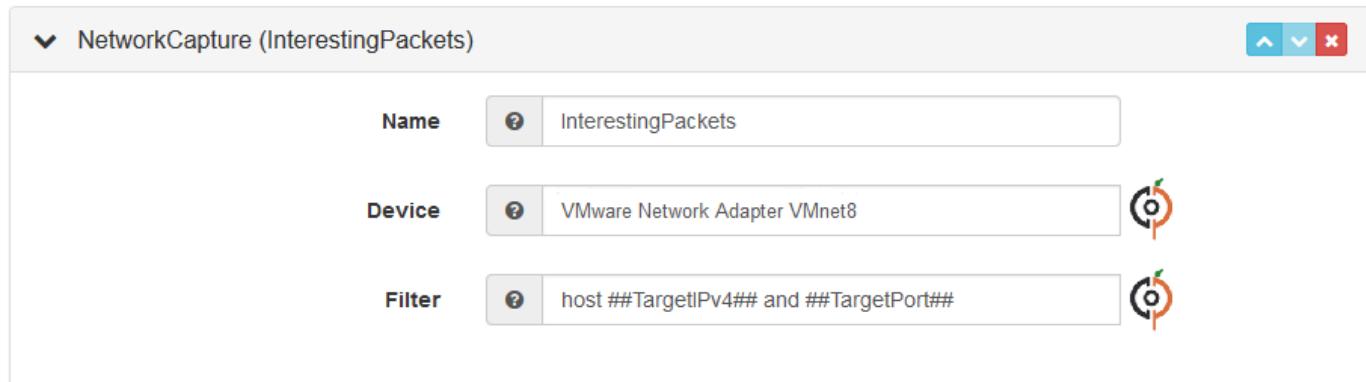
Specifies the VMware product used in the configuration.

Snapshot Name

Identifies the snapshot to use for the specific image.

Network Capture (InterestingPackets)

The [Netowrk Capture Monitor \(InterestingPackets\)](#) captures network packets sent and received from the test target. When a fault occurs, Peach stores the packets immediately surrounding the fault in the log of the test case.



The most significant parameters for the network capture monitor follow:

Device

Specifies the name of the interface on the local machine (the machine with Peach) used to communicate with the test target. Use [ifconfig](#) to identify the interface(s) available for use.

You can find the appropriate host interface that communicates with the VM using the following steps:



1. Collect a list of interfaces (and their IPv4 addresses) by running `ipconfig` or `ifconfig`.
2. Test each interface in the list. Manually run a capture session with Wireshark using an interface from the list.
3. On the host machine, Ping the target IPv4 (of the VM).
4. If the correct interface of the host is used, you'll see the Ping request and reply packet exchanges through Wireshark,
5. Loop to step 2 and repeat, using another interface.

Filter

Helps capture only those packets associated with the fuzzing session. The filter adheres to the syntax and requirements of the Pcap filter specification.



WireShark refers to the Libpcap filters as capture filters. Use the capture filters. Wireshark also defines its own display filters that it uses to filter entries in its session files. The display filters are not compatible with Libpcap.

Gdb (Debugger)

The `Gdb` debugger monitor performs two main functions in this recipe:

- Starts the network service at the start of a fuzzing job and restarts the service when the VM snapshot refreshes.
- Detects faults internal to the service.

▼ Gdb (Debugger)

Name Debugger

Executable /user/sbin/snmpd

Arguments -f

Fault On Early Exit true

Gdb Path /usr/bin/gdb

No Cpu Kill false

Restart After Fault false

Restart On Each Test false

Start On Call

Wait For Exit On Call

Wait For Exit Timeout 10000

The most significant parameters follow:

Executable

Identifies the full path to the Linux service executable. This monitor, managed by the remote agent, resides on the remote machine, so the full path is for the Linux file system.

Arguments

Arguments for the executable.

SaveFile (CollectLogs)

The [SaveFile](#) monitor collects log files from the remote test target and copies them to the Peach Logging folder. The monitor is housed by the remote agent.

SaveFile (CollectLogs)

Name	<input type="text"/> CollectLogs
Filename	<input type="text"/> /var/log/syslog



The most significant parameter follows:

Filename

Specifies the full path to the Linux logging system used by GDB.

Configuration Test

Once the monitors and associated parameters are part of the configuration, you can test the configuration. From the Configuration menu along the left edge of the window, click on **Test** to run a single iteration (test case) on the configuration. Note that the test checks the connections and communications. It does NOT do any fuzzing.

For more information on testing a configuration, see [Test Pit Configuration](#).

7.4. Recipe: Monitoring a Network Device

This recipe identifies the monitors and associated settings suitable for testing a network device, such as a router or a switch. The network device connects to the computer running Peach via a network interface. Peach tests the connection using a specific protocol or service. The network device's serial console connects to the computer running Peach. Peach uses the serial console to capture and inspect device messages.

This recipe uses a router to demonstrate the procedure.



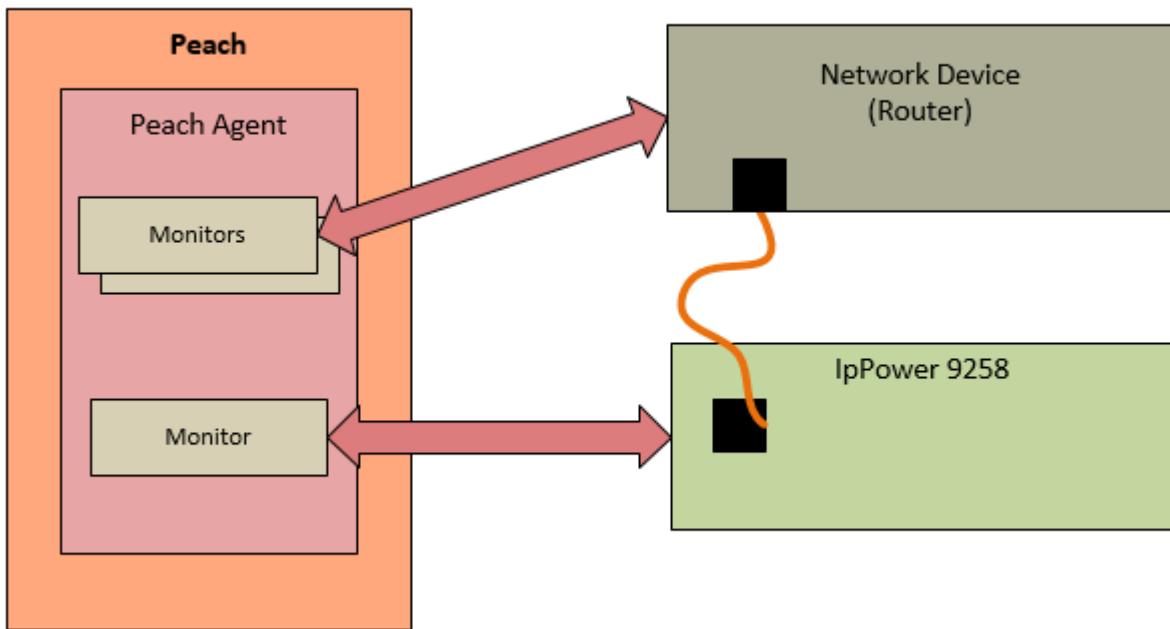
When testing protocols in layers 2 through 4 of the ISO network model, consider using a private test network so that the testing doesn't affect other systems on the network.

7.4.1. What is the fuzzing session workflow?

The workflow that we want to follow in the test consists of the following steps:

1. Power on the device.
2. Wait for the device to complete its boot process.
3. Fuzz the device and look for faults.
4. When a fault occurs, do the following:
 - a. Reset the device by cycling the power off and then on.
 - b. Wait for the device to complete its boot process.
5. Loop to step 3 and continue fuzzing.

Given this, here's a diagram of connections for the test configuration.



Peach handles communications with the device using a single, local agent. Peach also connects to a programmable power supply, the IpPower 9258, that can be powered down and up on command. In step 4a of the workflow, Peach sends the power cycling command to the IpPower 9258 to perform this task.

Now for setting up the monitors, consisting of detecting faults, collecting data, and automating the test environment.

The first item in detecting faults is to ensure the service is still listening for incoming connections. With more sophisticated devices, a service can stop, but the device still responds. Use the TCP port monitor for detecting when the service stops. As a backup, the Ping monitor can be used.

The second item in detecting faults is watching the device console for error messages. The Serial monitor with a regular expression specified, such as “Error”, can perform this task. You will need to construct the regular expression to watch for key word(s) in the error message text because the messages vary among manufacturers. Consult the device documentation or an expert on the test device for the message content.

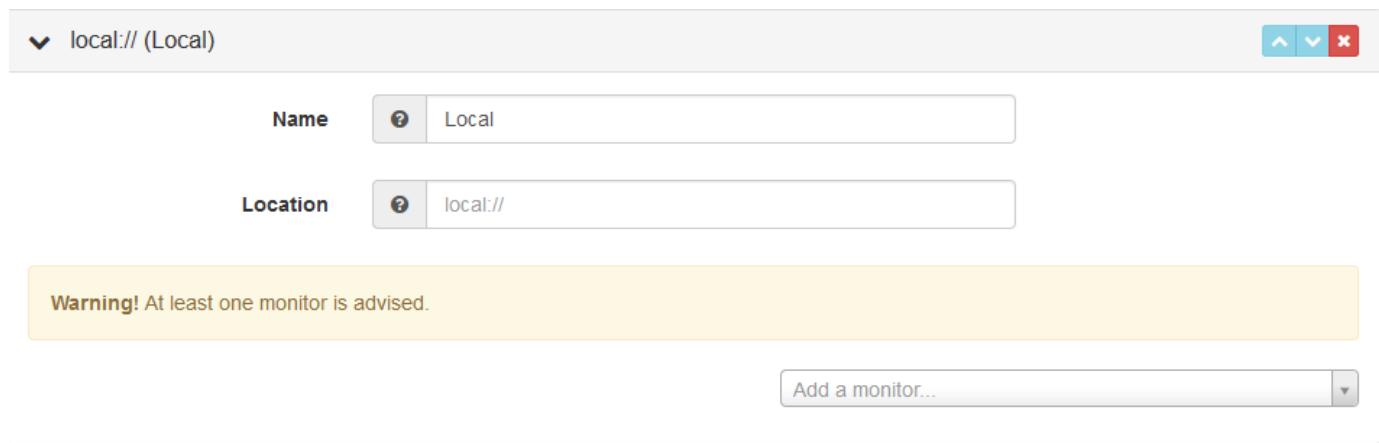
When a fault occurs, we want to collect data relevant to the fault. The NetworkCapture monitor (InterestingPackets) provides the mechanism to capture network packets around the time that the fault occurs. The Serial monitor provides console logs with messages leading up to the fault.

Finally, we want to automatically reset the device after a fault occurs by cycling the power off, then on again. The IpPower9258 monitor provides control over outlets. A second aspect of automation is that the system must wait for the device reboot to complete before resuming fuzzing activities. The Serial monitor performs this task by watching logging messages for a message that indicates the boot process completed.

7.4.2. Setting up Monitors Using the Peach Web UI

Starting Peach

1. Launch Peach from the command line (type `peach` and press the Return key) to start the UI.
2. Select a pit (test definition) of a protocol supported by the network device, so that Peach can communicate with the device during fuzzing.
 - Give the pit a name and a description. Peach makes a configuration file of the selections you make, so that you can re-use the setup again.
3. From the configuration menu along the left edge of the window, select Monitoring.
4. Fill in a name for the agent. Since this agent resides within Peach, the default location `local` is appropriate. Then, click the `Save` button.



Supplying Monitor Details

Begin each monitor with a name or descriptive text. This helps identify one monitor from another, and is especially evident with the two serial monitors.

Next, fill in the critical parameters for each monitor. These parameters have callouts in the settings diagram of each monitor. Details for these parameters are given in the text that follows.



The order of the monitors listed in the agent is significant. Peach processes the monitors in the order listed (from top to bottom). For example, if the IpPower monitor were last on the list, a blocking situation might arise because one of the Serial monitors would be waiting for the device boot process to finish before the power is recycled to initiate the device boot process.

For this recipe, use the monitors in the order they are presented:

- IpPower9258 (BootDevice)
- IpPower9258 (RebootDevice)
- SerialPort (WaitforBootOnStartAndFault)
- SerialPort (WaitForBootAfterFault)
- TcpPort (VerifyDeviceAlive)
- NetworkCapture (InterestingPackets)

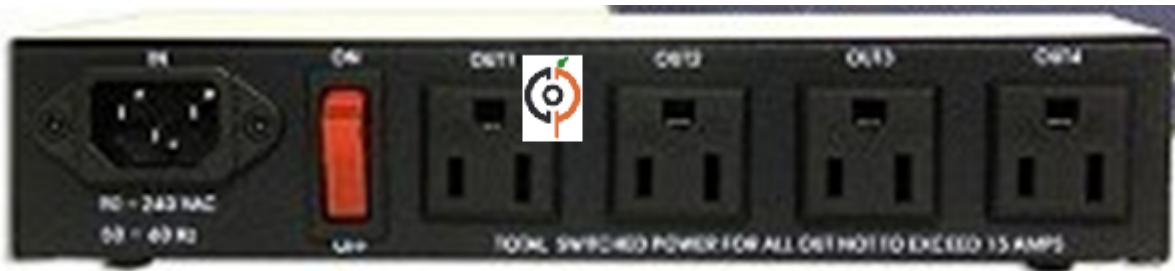
IpPower9258 (BootDevice)

The [IpPower 9258 Monitor](#) controls the IpPower 9258 device to start the network device at the beginning of the fuzzing session.

IpPower9258 (BootDevice)

Name	<input type="text"/> BootDevice
Host	<input type="text"/> 192.168.1.10
Password	<input type="text"/> Password!
Port	<input type="text"/> 1 
User	<input type="text"/> peach
Power Off On End	<input type="text"/> false
Power On Off Pause	<input type="text"/> 500
Start On Call	<input type="text"/> 
When	<input type="text"/> OnStart 

The **Port** parameter identifies the receptacle of the IpPower9258 device. Numbers range from 1 to 4. Receptacle 1 is the leftmost receptacle.



The **When** parameter "OnStart" identifies that the power cycle occurs at the start of the fuzzing session.

IpPower9258 (RebootDevice)

The second [IpPower 9258 Monitor](#) controls the IpPower 9258 device to recycle power, ensuring that the network device starts in a clean state following a fault.

IpPower9258 (RebootDevice)

Name	<input type="text"/> RebootDevice	
Host	<input type="text"/> 192.168.1.10	
Password	<input type="text"/> Password!	
Port	<input type="text"/> 1	
User	<input type="text"/> peach	
Power Off On End	<input type="text"/> false	
Power On Off Pause	<input type="text"/> 500	
Start On Call	<input type="text"/>	
When	<input type="text"/> OnIterationStartAfterFault	

The **Port** parameter identifies the receptacle of the IpPower9258 device. Numbers range from 1 to 4. Receptacle 1 is the leftmost receptacle.

The **When** parameter "OnIterationStartAfterFault" identifies that the power cycle occurs at the start of the test case that immediately follows a fault.

Serial Port (WaitforBootOnStartAndFault)

This [Serial Port Monitor](#) addresses two configuration settings: one automation setting and one fault detection setting. The automation setting causes Peach to wait for the device to complete its boot processing before starting the fuzzing session. The fault detection setting causes peach to monitor the console for messages that indicate a fault occurred on the device.

SerialPort (WaitForBootOnStartAndFault)

Name	WaitForBootOnStartAndFault	
Port	COM1	
Baud Rate	115200	
Data Bits	8	
Dtr Enable	false	
Fault Regex	(CRITICAL ERROR ASSERT CRASH)	
Handshake	None	
Max Buffer Size	1048576	
Parity	None	
Rts Enable	false	
Stop Bits	One	
Wait On Call		
Wait Regex	Bootup Completed	
Wait When	OnStart	

The **Port** parameter identifies the serial port on the computer that receives monitoring.

- In Windows, the port map is accessible from the Device Manager located in the Control Panel\System applet. In the illustration, the value is **COM1**.
- In Linux and OS X systems the port map is accessible with the following command: `dmesg | grep tty`. Specify the value of the appropriate port, such as **tty0**.

The **Fault Regex (CRITICAL | ERROR | ASSERT | CRASH)** identifies words that indicate a fault occurred. When the monitor encounters any word of a message that matches any word in the regular expression, Peach issues a fault.

The **Wait When** and **Wait Regex** parameters are automation oriented. **Wait When** identifies when peach should enter a waiting state. The value **OnStart** identifies that the waiting period is when the test target goes through the boot process at the beginning of the fuzzing session. The **Wait Regex** identifies the message text that the monitor looks to match. Here, the value is **Bootup Completed**. Peach waits until this message appears to begin fuzzing the target.



The regular expressions used with this monitor are used to identify faults that have occurred in the device. You will need to construct the regular expression to watch for key word(s) in the error message text. Consult the device documentation or an expert of the test device for the message content.

Serial Port Monitor (WaitForBootAfterFault)

This second instance of the [Serial Port Monitor](#) addresses resumption of a fuzzing session after a fault.

When a fault occurs, the IpPower9258 recycles the power causing the network device to reboot. Then, this monitor causes Peach to wait until the network device completes its boot process and becomes available for use before resuming the current fuzzing session.

SerialPort (WaitforBootAfterFault)

Name	WaitforBootAfterFault
Port	COM1
Baud Rate	115200
Data Bits	8
Dtr Enable	false
Fault Regex	
Handshake	None
Max Buffer Size	1048576
Parity	None
Rts Enable	false
Stop Bits	One
Wait On Call	
Wait Regex	Bootup Completed
Wait When	OnIterationStartAfterFault

The **Port** parameter identifies the serial port on the computer that receives monitoring. Since only one serial port is in this fuzzing setup, the value should be identical with the value for the WaitforBootOnStartAndFault monitor.

The **Wait When** and **Wait Regex** parameters identify when Peach should enter a waiting state. The **Wait When** value `OnIterationStartAfterFault` identifies that the waiting period follows each fault occurrence. The **Wait Regex** identifies the message text that the monitor looks to match. Here, the value is `Bootup Completed`. Peach waits until this message appears to resume the fuzzing session in progress.



The regular expressions used with this monitor are used to identify conditions that have occurred in the device. You will need to construct the regular expression to watch for key word(s) in the error message text. Consult the device documentation or an expert of the test device for the message content.

TcpPort (VerifyDeviceAlive)

The [TcpPort Monitor](#) periodically checks two things when the State Model issues a [Call](#) event during a test case:

1. The state of a TCP port on the target.
2. The state of the service or protocol on the target, that uses the same TCP port on the target.

If the status of the port is [Closed](#), Peach reports an error.



The TcpPort monitor can be used when the target runs TCP. When testing other protocols, use the Ping monitor instead to see whether the device as a whole is responding.

Name	VerifyDeviceAlive
Host	##TargetIPv4##
Port	80
Action	Fault
State	Closed
Timeout	-1
Wait On Call	(disabled)
When	OnCall

The **Host** parameter specifies the hostname of the target or the IPv4 address of the target. The [##TargetIPv4##](#) value is a configuration variable that you set to identify the target. Its operation is similar to operating system environment variables. For more information, see [Variables](#).

The **Port** value should be set to the appropriate port number of the target device used by the service

under test. In this example, the service is using TCP on port 80 of the network device.

For example, some common port values follow: HTTP uses port 80; SSH uses port 22; and, FTP uses port 21.

The **Action** parameter specifies the type of action that the monitor performs. Here, the value used is **Fault**; and causes the monitor to report a fault when the TCP port is closed and unresponsive.

The **State** parameter specifies the fault condition. The value **Closed** indicates a fault occurs when the communication channel changes to **Closed**.

NetworkCapture (InterestingPackets)

The [NetworkCapture Monitor](#) captures network traffic (packets) sent and received from the test target.



The **Device** parameter specifies the hostname of the target or the IPv4 address of the target. The value given is **eth0**.

The **Filter** parameter is a capture filter (Berkeley Packet Syntax filter used by Libpcap) that limits the network packets under consideration to those packets that match the specified filter. The packets that match the filter are captured from the wire as they arrive or leave the test target.

Here, the filter consists of the hostname combined with the TCP port number of the test target. As previously mentioned, **port 80** is the test target TCP port number.



WireShark refers to the Libpcap filters as capture filters. Use the capture filters. Wireshark also defines its own display filters that it uses to filter entries in its session files. The display filters are not compatible with Libpcap.

7.5. Recipe: Monitoring a Windows Network Service Client

This recipe describes the base setup needed to fuzz a client of a Windows network service. When fuzzing a network client, Peach impersonates the network service, the other endpoint of the network connection.

The recipe is a model that you can follow closely. Or, use the model as a starting point and augment the model for your specific situation. This recipe consists of the following parts:

1. The workflow for the fuzzing session
2. The Peach components to use in configuring the fuzzing setup. This section focuses on the monitoring needs and the agents that house the monitors.
3. Configuration settings used to fuzz a service client running this workflow



Assumptions/Givens in this recipe are that a Pit is ready to use, Peach is ready to run, and any software module needed to perform the fuzzing job is installed.

7.5.1. Workflow for the Fuzzing Session

The workflow lists the task sequence that occurs when running the fuzzing session. The setup needed to implement the workflow follows in the next section. Start with defining the workflow, especially if you plan to embellish the recipe.

Here is the workflow that Peach performs in fuzzing a Windows network service client:

1. Revert to a virtual machine snapshot.
2. Wait for the machine to boot up.
3. Perform fuzzing. Create and run test cases.
 - a. Peach launches the network client. The client initiates contact with the server and sends requests to the server.
 - b. Peach impersonates the server and replies to client queries. Query responses contain fuzzed data.
 - c. Perform fault detection on the client.
 - d. If a fault occurs, collect data surrounding the test case.
 - e. Revert to the VM snapshot.
4. Loop to step three (resume fuzzing).

7.5.2. Peach Components Needed in the Fuzzing Configuration

Defining the Peach components divides in two parts: identifying the monitors to use in the configuration and identifying where to locate the specified monitors.

Identifying Monitors

This part of the recipe revisits each step of the workflow to identify the monitors needed to implement the configuration:

1. Revert to a snapshot of a virtual machine.

Peach needs to automate the test environment and remove human interaction during the fuzzing job. We place the service in a virtual machine (VM) because Peach can use a VM monitor to automatically start and reset the test environment when needed.

The VM snapshot is taken while the guest OS and the Peach agent are running. Using such a snapshot avoids the wait time associated with booting up the virtual machine. Also, the same snapshot is used when Peach refreshes the test environment after a fault occurs.

The monitor for the VM environment, [VMware](#) monitor, resides on the host machine.

2. Wait for the machine to boot up.

Peach waits for the VM snapshot to resume.

3. Perform fuzzing, checking for faults.

- a. Launch the network client.

The Peach agent launches the client service through the Windows debugger. The client submits requests from the remote machine. The debugger and the client both reside on the remote machine.

NOTE: The PageHeap monitor complements the WindowsDebugger monitor by allowing in-depth heap analysis.

- b. Reply to the client request with fuzzed data.

Peach impersonates the network server and sends fuzzed replies in response to client queries.

- c. Perform fault detection in the VM.

The WindowsDebugger monitor watches the internals of the services and detects faults such as access violations and exceptions. Again, the debug monitor is located on the same machine as the service.

- d. Collect data surrounding each fault as it happens.

When a fault occurs, the packets involved with the fault are interesting. Peach captures the packets using a network capture monitor. This monitor resides on the local machine with Peach Fuzzer.

- e. Revert to the VM snapshot.

This step uses the VM monitor and VM snapshot from step one to refresh the test environment, and the debug monitor from step three to start the network service in the refreshed environment. No additional monitors are needed for this step.

4. Resume fuzzing.

Loop to step three, perform fuzzing.

Identifying Agents

Peach offers two types of agents to manage monitors and I/O publishers: local and remote.

- Local agents reside inside Peach.

The local agent in this recipe addresses automation involving the VM and data collection that captures network packets. The local agent houses the [VMware](#) and the [NetworkCapture](#) monitors.

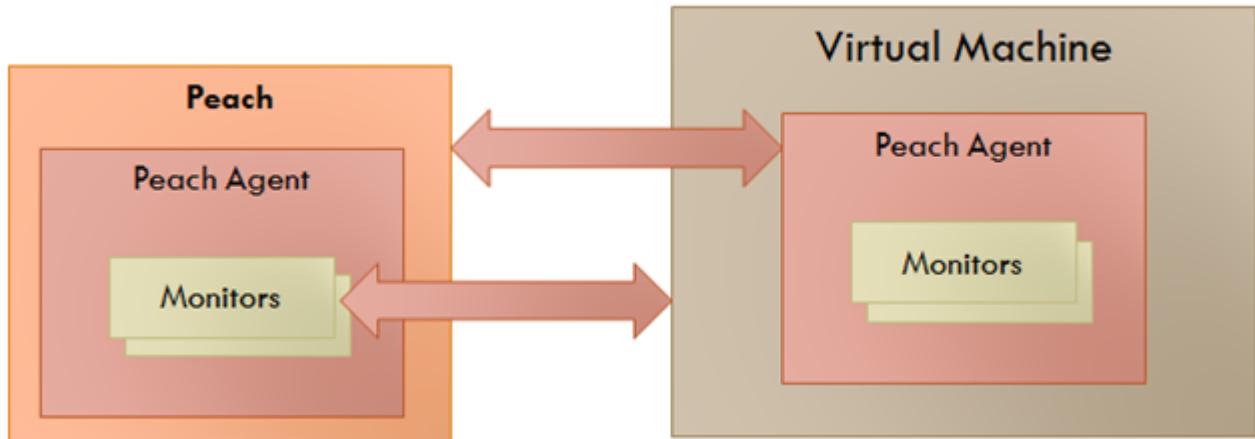
The VMware monitor starts a snapshot VM environment at the beginning of the fuzzing job, as well as restarting the same VM snapshot after a fault occurs.

- Remote agents reside in separate processes on remote machines with the test targets.

In this case, the remote agent and the Windows service client reside on the same machine.

The remote agent houses the [PageHeap](#) and the [WindowsDebugger](#) monitors that start the Windows service client at the beginning of the fuzzing job and restart the service in the refreshed environment after a fault. The WindowsDebugger monitor detects faults that occur in the client.

The result is that we end up with the following configuration:



Peach is located on one machine with a local agent that houses the VM monitor and the Network capture monitor. A second agent resides on the remote machine with the service. The remote agent houses the PageHeap and WindowsDebugger monitors.

The local agent is simple to implement. All that's needed is to define the agent, then specify the appropriate monitors and monitor settings used with the local agent.

The remote monitor is a little more involved. Like the local agent, the remote agent needs to be defined, then specify the appropriate monitors and monitor settings used with the remote agent. Second, the remote agent needs to run on the same OS as the test target. This step can be done separately from specifying the configuration details. In this recipe, a VM snapshot is used. See [Using](#)

[Virtual Machines](#), for information on setting up the VM snapshot.

7.5.3. Sample Network Client Configuration

This section shows the recipe implemented for a Windows network service client and consists of the following items:

- Setup on the Target VM Image
- Pit variables
- Peach agents
- Peach monitors
- Debug monitor "No Cpu Kill" parameter
- Configuration test

The configurations for the network client and the network service are very similar. Two significant differences exist:

- The network client configuration uses a client application instead of the network service.
- In the network client configuration, the test target initiates the action instead of responding to a request. The client contacts Peach, a surrogate network service, then waits for Peach to provide a response to the query. The debug monitor has additional configuration options that are set to drive this configuration.

Setup on the Target VM Image

Perform the following task on the VM before taking a snapshot of the VM.

- Run the Peach agent from a command processor as an administrator.

Within the command processor, navigate to the peach folder and execute the following command:

`peach -a tcp`

When Peach starts the VM, the Peach agent is running in a root shell.

Pit Variables

The following UI display identifies data values typically needed by a network protocol Pit. The variables and values are independent of the monitors used in the configuration. Pit variables are unique to the Pit and might differ with those in the example illustration.

Variables

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

Saved successfully.

 Save  + Add Variable

Name	Key	Value	Remove
.	.		
.	.		
.	.		
Community String	CommString	 public	
Source IPv4 Address	SourceIPv4	 192.168.127.129	
Source Port	SourcePort	 161	
Timeout	Timeout	 5000	

The Pit User Guides describe the Pit-specific variables.

Community String (Authentication)

Community string used for authentication. Peach and the network client must use the same community string. Check the server documentation for consistency of this value. If needed, change the value here to coincide with the value expected by the test target.

Source Port

Port number of the local machine that sends packets to the server. Several services use well-known ports that usually can be left unedited.

Target IPv4 Address

IPv4 address of the target machine (client). For information on obtaining the IPv4 address, see Retrieving Machine Information section of the Pit User Guide.

Target Port

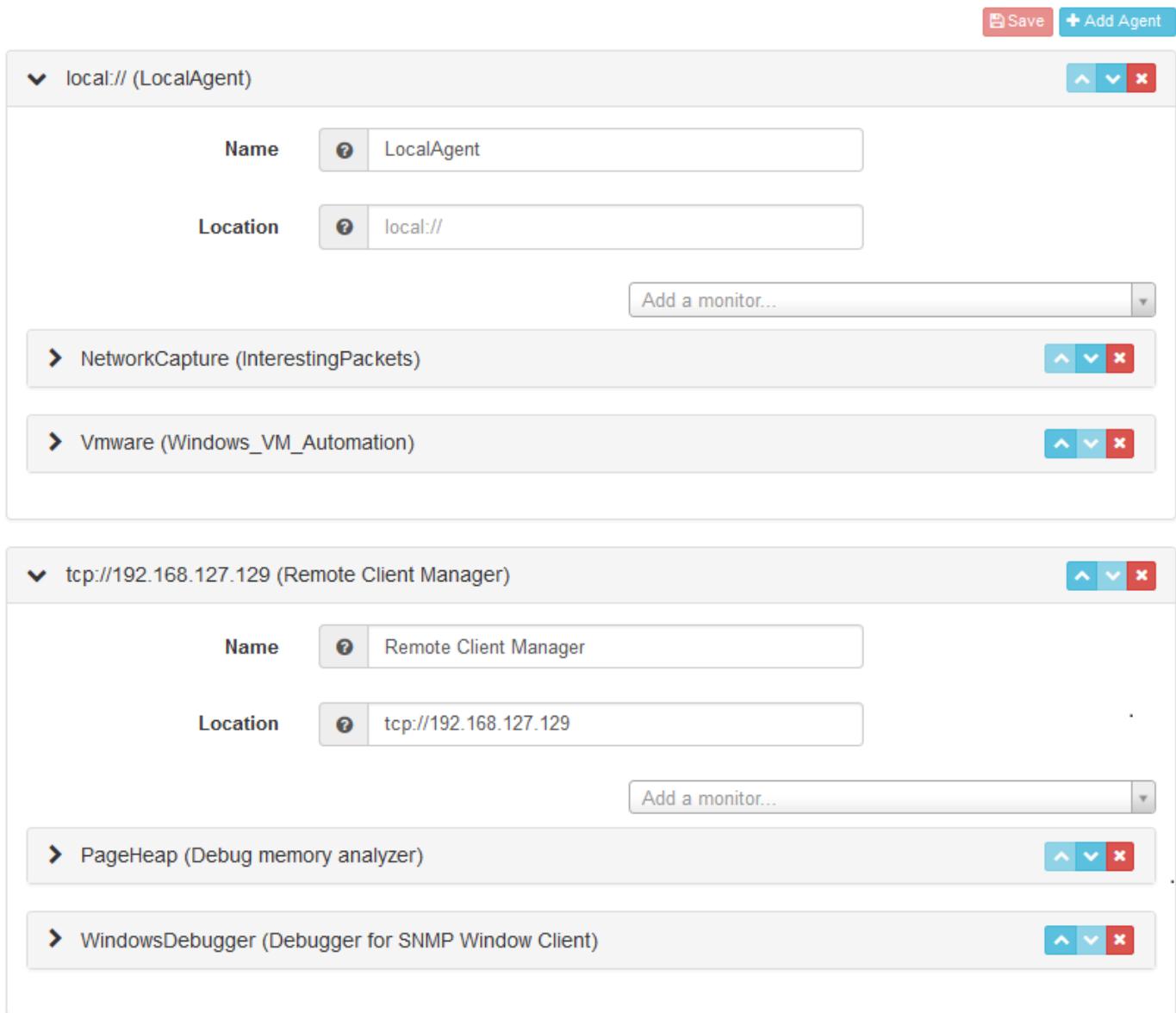
SNMP port number of the remote machine that sends and receives packets. Several services use well-known ports that usually can be left unedited.

Timeout

Duration, in milliseconds, to wait for incoming data. During fuzzing, a timeout failure causes the fuzzer to skip to the next test case.

Agents

The following UI diagram acts as an overview, showing the Peach agents and the monitors within each agent. Peach uses the ordering within the agent to determine the order in which to load and run monitors.



The local agent is defined first and lists the default information for both name and location. This definition for a local agent is typical and, otherwise, unremarkable. The NetworkCapture and Vmware monitors are independent of one another, allowing either monitor to top the list.

The remote agent, named "Remote Client Manager", has quite a different location specification. The location consists of concatenated pieces of information:

- Channel. The channel for a remote agent is **tcp**. A colon and two forward slashes separate the channel from the IPv4 address of the hardware interface.
- Target IPv4 address of the remote machine. The IPv4 address of the agent is the second component

of the location. For more information, see the Retrieving Machine Information section of the **SNMP Peach Pit User Guide**.

The monitor list within each agent is significant, as the monitors are launched in order from top to bottom within an agent.

Monitors

This recipe uses four monitors, two on the machine with Peach and two on the remote machine. The recipe shows each monitor and describes its roles: fault detection, data collection, and automation.

Vmware (Remote Client Manager)

The [Vmware](#) monitor controls setting up and starting the virtual machine.

Vmware (Windows_VM_Automation)

Name	<input type="text" value="Windows_VM_Automation"/>	
Vmx	<input type="text" value="C:\\bix\\PeachProClient\\Peach Pro Client.vmx"/>	
Headless	<input type="text" value="false"/>	
Host	<input type="text"/>	
Host Port	<input type="text" value="0"/>	
Host Type	<input type="text" value="Workstation"/>	
Login	<input type="text"/>	
Password	<input type="text"/>	
Reset Every Iteration	<input type="text" value="false"/>	
Reset On Fault Before Collection	<input type="text" value="false"/>	
Snapshot Index	<input type="text"/>	
Snapshot Name	<input type="text" value="Snap_WinSvcCli_1"/>	
Stop On Fault Before Collection	<input type="text" value="false"/>	
Wait For Tools In Guest	<input type="text" value="true"/>	
Wait Timeout	<input type="text" value="600"/>	

The most significant parameters for the VMware monitor follow:

Vmx

Identifies the full path of the virtual machine image. Peach loads the snapshot of the VM image at the start of the fuzzing job and after a fault occurs.

Headless

Specifies whether the VM connects to a viewing window in the VMware window. When developing a configuration, set this parameter to false. When performing a fuzzing job, the setting doesn't matter.

Host Type

Specifies the VMWare product used in the configuration.

Snapshot Name

Identifies the snapshot to use for the specific image.

Network Capture (InterestingPackets)

The [Netowrk Capture Monitor](#) (InterestingPackets) captures network packets sent and received from the test target. When a fault occurs, Peach stores the packets immediately surrounding the fault in the log of the test case.



The most significant parameters for the network capture monitor follow:

Device

Specifies the name of the interface on the local machine (the machine with Peach) used to communicate with the test target. Use [ipconfig -all](#) to identify the interface(s) available for use.

You can find the appropriate host interface that communicates with the VM using the following steps:

1. Collect a list of interfaces (and their IPv4 addresses) by running [ipconfig](#).
2. Test each interface in the list. Manually run a capture session with Wireshark using an interface from the list.
3. On the host machine, Ping the target IPv4 (of the VM).
4. If the correct interface of the host is used, you'll see the Ping request and reply packet exchanges through Wireshark,
5. Loop to step 2 and repeat, using another interface.



Filter

The packet filter helps capture only those packets associated with the fuzzing session. The filter adheres to the syntax and requirements of the Pcap filter specification.



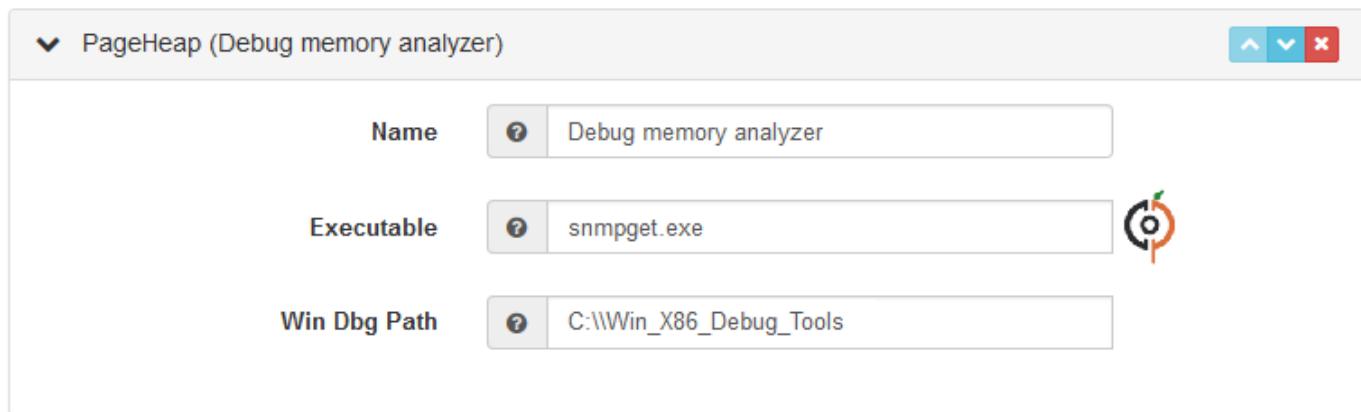
WireShark refers to the Libpcap filters as capture filters. Use the capture filters in Peach. Wireshark also defines its own display filters that it uses to filter entries in its session files. The display filters are not compatible with Libpcap.

PageHeap

The [PageHeap](#) monitor manages registry settings that enables the Windows debugger to perform heap analysis. This monitor sets the appropriate registry values at the start of a fuzzing session and clears them at the session's end. The monitor is housed by the remote agent.



PageHeap requires administrative privileges to run correctly.



The most significant parameter follows:

Executable

Specifies the file name and file extension of the test target.

WindowsDebugger

The [WindowsDebugger](#) debugger monitor performs two main functions in this recipe:

- Starts the network client at the start of a fuzzing job and restarts the client when the VM snapshot refreshes.
- Detects faults internal to the client.

The WindowsDebugger monitor uses the settings in the following illustration:

▼ WindowsDebugger (Debugger for SNMP Window Client) ▲ ▼ ✕

Name	<input type="text"/> Debugger for SNMP Window Client	
Arguments	<input type="text"/> -v:2c -c:"public" -r:192.168.127.1 -o:.1.3.6.1.2.1.1.1	
Cpu Poll Interval	<input type="text"/> 200	
Executable	<input type="text"/> C:\snmpget\snmpget.exe	
Fault On Early Exit	<input type="text"/> false	
Ignore First Chance Guard Page	<input type="text"/> false	
Ignore Second Chance Guard Page	<input type="text"/> false	
No Cpu Kill	<input type="text"/> true	
Process Name	<input type="text"/>	
Restart After Fault	<input type="text"/> false	
Restart On Each Test	<input type="text"/> false	
Service	<input type="text"/>	
Start On Call	<input type="text"/> StartIterationEvent	
Symbols Path	<input type="text"/> SRV*http://msdl.microsoft.com/download/symbols	
Wait For Exit On Call	<input type="text"/>	
Wait For Exit Timeout	<input type="text"/> 10000	
Win Dbg Path	<input type="text"/> C:\Win_X86_Debug_Tools	

The most significant parameters follow:

Executable

Identifies the full path to the Windows service client. The client resides on the remote machine; so, the full path is for the Windows file system.

Arguments

Arguments for the executable.

No Cpu Kill

Controls whether the process stays alive if its CPU usage drops to zero. Specify `true` to keep the process running and to allow the process to release or close its resources before exiting. For more information, see the following section *Closing the Client Process*.

Start On Call

Controls when the test target launches, and in turn, initiates contact with the service (Peach). Specify `StartIterationEvent` to launch the client at the start of the test case.

7.5.4. Closing the Client Process

In this recipe, Peach launches the network service client using the "Start On Call" parameter so that the client initiates contact with the server. Then, at the end of the test case after execution complete, the "No Cpu Kill" parameter provides control of how the client closes:

- If "No Cpu Kill" is `true`, Peach waits for the process to exit OR for a time to elapse specified by the "Wait For Exit Timeout" parameter.
- If "No Cpu Kill" is `false`, Peach waits for the CPU usage of the process to reach zero percent OR for the process to exit OR for a time to elapse specified by the "Wait For Exit Timeout" parameter. The "No Cpu Kill" parameter default setting is `false`.



If the waiting period ends, peach kills the target process if it is still running and starts the next iteration.

So, when do you need to let a process with zero CPU activity continue to execute?

Set "No Cpu Kill" to `true` when you're fuzzing a network service client. In this scenario, Peach Fuzzer starts the network client using the "Start on Call" parameter to initiate contact with the service. When the client receives and processes the reply, Peach waits for the client to run to completion and watches for any faults that occur before the client exits.

Scenarios exempt from the "No Cpu Kill" option include the following:

- Fuzzing network service servers typically do not use the "Start On Call" option, so the "No Cpu Kill" option isn't needed.
- Fuzzing file formats require "Start on Call" to start the fuzzing target once the fuzzed data file is generated. The "No Cpu Kill" parameter can be used here; however, Peach can save time that will be replicated in each test case by letting the process terminate if the CPU usage falls to zero. In this

case, not using "No Cpu Kill" is a performance optimization.

- Embedded devices. Fuzzing configurations for these devices do not use the Peach debugging monitors: GDB, WindowsDebugger, or CrashWrangler. So, the "No Cpu Kill" option isn't needed.
- Kernel-mode debugging. Kernel-mode debugging has its own set of requirements. "No Cpu Kill" is not used here.

Configuration Test

Once the monitors and associated parameters are part of the configuration, you can test the configuration. From the Configuration menu along the left edge of the window, click on **Test** to run a single iteration (test case) on the configuration. Note that the test checks the connections and communications. It does NOT do any fuzzing.

For more information on testing a configuration, see [Test Pit Configuration](#).

7.6. Recipe: Monitoring a Windows Network Service

This recipe describes the base setup needed to fuzz a Windows network service. The recipe is a model that you can follow exactly as is. Or, you can use the model as a starting point and augment the model for your specific situation. This recipe consists of the following parts:

1. The workflow for the fuzzing session
2. The Peach monitoring and agent components to use in configuring the fuzzing setup
3. Configuration settings used to fuzz a sample service running this workflow



Assumptions/Givens in this recipe are that a Pit is ready to use; Peach is ready to run; and any software module needed to perform the fuzzing job is installed.

In this scenario, Peach runs on a host computer; the network server runs in a Virtual Machine (VM) on the host. With Peach running on the host, it controls the environment. If the network server crashes, the worst thing that happens is that the virtual machine has to restart. Peach recovers the data if the network server crashes.

7.6.1. Workflow for the Fuzzing Session

The workflow lists the task sequence that occurs when running the fuzzing session. The setup needed to implement the workflow follows in the next section. Start with defining the workflow, especially if you plan to embellish the recipe.

Here is the workflow that Peach performs in fuzzing a Windows network service:

1. Revert to a virtual machine snapshot.
2. Wait for the machine to boot up.

3. Launch the network service.
4. Perform fuzzing. Create and run test cases.
 - Peach initiates contact with the server and sends packets of fuzzed data to the server.
 - Check for faults, such as crashes and access violations.
5. When a fault occurs, do the following:
 - a. Collect data surrounding the test case.
 - b. Revert to the VM snapshot.
 - c. Launch the network service.
6. Loop to step 4 (resume fuzzing).

7.6.2. Peach Components Needed in the Fuzzing Configuration

Defining the Peach components divides in two parts: identifying the monitors to use in the configuration and identifying where to locate the specified monitors.

Identifying Monitors

This part of the recipe revisits each step of the workflow to identify the monitors needed to implement the configuration:

1. Revert to a snapshot of a virtual machine.

Peach needs to automate the test environment and remove human interaction during the fuzzing job. We place the service in a virtual machine (VM) because Peach can use a VM monitor to automatically start and reset the test environment when needed.

The VM snapshot is taken while the guest OS and the Peach agent are running. Using such a snapshot avoids the wait time associated with booting up the virtual machine. Also, the same snapshot is used when Peach refreshes the test environment after a fault occurs.

The monitor for the VM environment, [VMware](#) monitor, resides on the host machine.

2. Wait for the machine to boot up.

Peach waits for the VM snapshot to resume.

3. Launch the network service.

In Windows, Peach needs to explicitly launch the network service when PageHeap and the Windows debugger are used. The [WindowsService](#) monitor manages the service on the remote machine with the test target.

4. Perform fuzzing, checking for faults.

Perform fault detection in the VM. The [WindowsDebugger](#) monitor watches the internals of the services and detects faults such as access violations and exceptions.

The [PageHeap](#) monitor complements the Windows debugger by enabling heap analysis in the debugger.

Both monitors run on the remote machine with the test target.

5. Collect data surrounding each fault as it happens.

Peach sends and receives network packets to the service. When a fault occurs, the packets involved with the fault are interesting. Peach captures the packets using the [NetworkCapture](#) monitor. This monitor resides on the local machine with Peach Fuzzer.

6. Resume fuzzing.

This step uses the VM monitor and VM snapshot from step 1 to refresh the test environment, and the [WindowsService](#) monitor from step 3 to start the network service in the refreshed environment. No additional monitors are needed for this step.

Identifying Agents

Peach offers two types of agents to manage monitors and I/O publishers: local and remote.

- Local agents reside inside Peach.

The local agent in this recipe addresses automation involving the VM and data collection that captures network packets. The local agent houses the [VMware](#) and the [NetworkCapture](#) monitors.

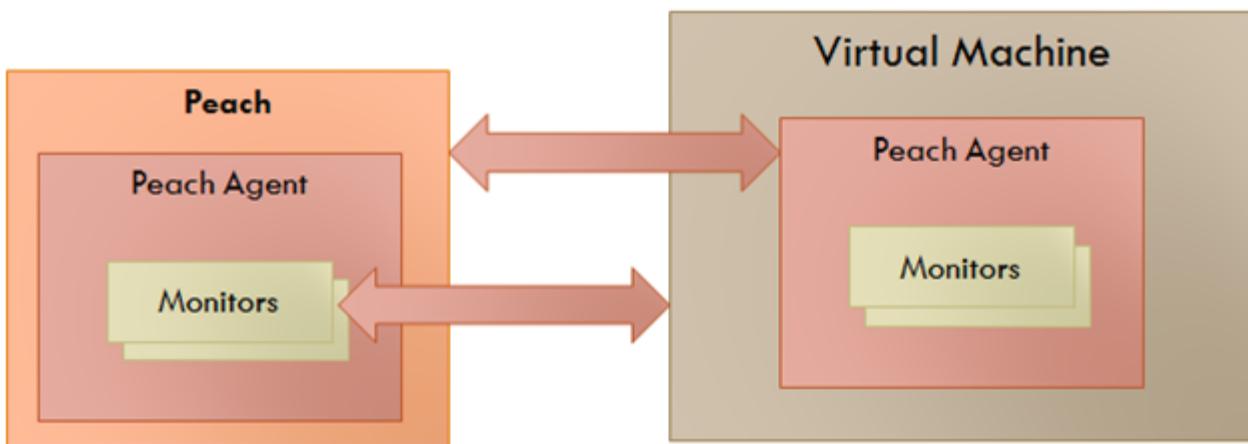
The VMware monitor starts a snapshot VM environment at the beginning of the fuzzing job, as well as restarting the same VM snapshot after a fault occurs.

- Remote agents reside in separate processes on remote machines with the test targets.
In this case, the remote agent and the Windows service reside on the same machine.

The remote agent houses the [WindowsService](#), the [PageHeap](#), and the [WindowsDebugger](#) monitors.

- The [WindowsService](#) monitor starts the network service at the beginning of the fuzzing job and restarts the service in the refreshed environment after a fault.
- The [PageHeap](#) monitor manages registry settings at the beginning and the end of a fuzzing session, that in turn enables heap memory analysis by the debugger.
- The [WindowsDebugger](#) monitor detects faults that occur in the service.

The result is that we end up with the following configuration:



Peach is located on one machine with a local agent that houses the VM monitor and the Network capture monitor. A second agent resides on the remote machine with the service. The remote agent houses the PageHeap, the WindowsDebugger, and the WindowsService monitors.

The local agent is simple to implement. All that's needed is to define the agent, then specify the appropriate monitors and monitor settings used with the local agent.

The remote monitor is a little more involved. Like the local agent, the remote agent needs to be defined, then specify the appropriate monitors and monitor settings used with the remote agent. Second, the remote agent needs to run on the same OS as the test target. This step can be done separately from specifying the configuration details. In this recipe, a VM snapshot is used. See the appendix, Using Virtual Machines, for information on setting up the VM snapshot.

7.6.3. Sample Windows Service Configuration

This section shows the recipe implemented for a Windows network service and consists of the following items:

- Setup on the Target VM Image
- Settings for the service on the Windows VM
- Pit variables
- Peach agents
- Peach monitors
- Configuration Test

Setup on the Target VM Image

Perform the following items on the VM before taking a snapshot of the VM.

1. Run the Peach agent from a command processor with administrative access.

Within the command processor, navigate to the peach folder and execute the following command:

```
peach -a tcp
```

When Peach starts the VM, the Peach agent is running in a root shell.

Windows Service Setup

The sample configuration uses a Windows network service as the fuzzing target. Some services are included with Windows; but, might be turned off. Other services are either custom or available on the Web.

Use the following steps to ensure a service is ready for use:

1. From the Windows Start button, right-click “Computer”, then select “Manage” from the shortcut menu.
2. Expand the “Services and Applications” entry in the Computer Management pane.
3. Double-click “Services”.
4. Search for the Service you are targeting.
5. If the status is not “Stopped”, right-click the service name and choose “Stop”.

Some properties of the service need to be configured to use the service. Right click on the Service entry to display its properties and adjust the necessary settings in the dialog.

The following action is performed on the local system.

- Allow access to run the service through the firewall on the local system.

Pit Variables

The following UI display identifies data values typically needed by a network protocol Pit. The variables and values are independent of the monitors used in the configuration. Pit variables are unique to the Pit and might differ with those in the example illustration.

Variables

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

				 Save	 Add Variable
Name	Key	Value		Remove	
	:				
	:				
Community String	CommString	 public			
Source Port	SourcePort	 162			
Target IPv4 Address	TargetIPv4	 192.168.127.148			
Target Port	TargetPort	 161			
Timeout	Timeout	 1000			

The Pit User Guides describe the Pit-specific variables.

Community String (Authentication)

Community string used for authentication by the network server. Check the network service documentation for consistency of this value. If needed, change the value here to coincide with the value expected by the test target.

Source Port

Port number of the local machine that sends packets to the server. Several services use well-known ports that usually can be left unedited.

Target IPv4 Address

IPv4 address of the target machine (server). For information on obtaining the IPv4 address, see Retrieving Machine Information in the Pit documentation.

Target Port

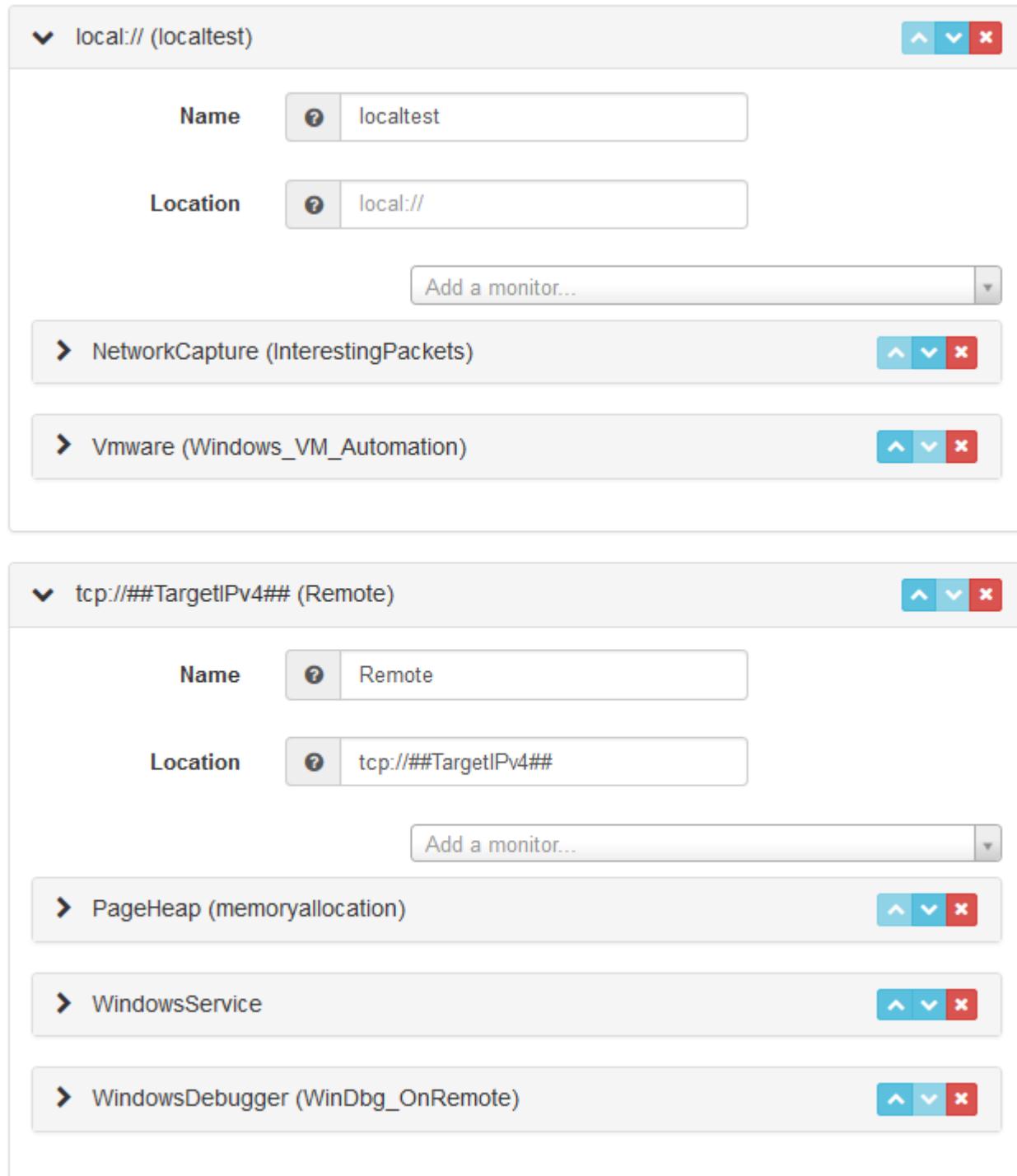
Port number of the server that receives packets. Several services use well-known ports that usually can be left unedited.

Timeout

Duration, in milliseconds, to wait for incoming data. During fuzzing, a timeout failure causes the fuzzer to skip to the next test case.

Agents

The following UI diagram acts as an overview, showing the Peach agents and the monitors within each agent. Peach uses the ordering within the agent to determine the order in which to load and run monitors.



The local agent is defined first and lists the default information for both name and location. This definition for a local agent is typical and, otherwise, unremarkable. The monitor list includes the NetworkCapture and the Vmware monitors that are independent of each other.

The remote agent, named "Remote", has quite a different location specification. The location consists of concatenated pieces of information:

- Channel. The channel for a remote agent is `tcp`. A colon and two forward slashes separate the channel from the IPv4 address of the hardware interface.
- IPv4 address. The IPv4 address of the agent is the second component of the location. Use `ipconfig -all` to find this address of the remote machine.

The monitor list within each agent is significant, as the monitors launch in sequence from top to bottom within an agent.



For first-time users, we recommend that you build incrementally to the final configuration by testing each monitor along the way. You can run the VM manually until you're ready to automate the environment.

1. Start with the local agent and the network capture monitor to capture network packets.
2. Add the remote agent and the WindowsService monitor.
3. Add the WindowsDebugger monitor to the remote agent.
4. Add the PageHeap monitor to the remote agent and reposition it atop the remote agent monitor list.
5. Add automation to the local agent using the Vmware monitor.

Monitors

This recipe uses five monitors, two on the machine with Peach and three on the remote machine. The recipe shows each monitor and describes its roles: fault detection, data collection, and automation.



When specifying a backslash (\) in the Peach Web user interface, double them, as the parser treats the first \ as a meta character.

NetworkCapture Monitor

The [Netowrk Capture Monitor \(InterestingPackets\)](#) monitor captures network packets when a fault occurs and stores them in the log for the test case that generates the fault.

Name	InterestingPackets
Device	VMware Network Adapter VMnet8
Filter	host ##TargetIPv4## and ##TargetF

The most significant parameters for the network capture monitor follow:

Device

Name of the interface on the local machine (the machine with Peach) used to communicate with the test target. Use `ipconfig -all` to identify the interface(s) available for use.

You can find the appropriate host interface that communicates with the VM using the following steps:

1. Collect a list of interfaces (and their IPv4 addresses) by running `ipconfig`.
2. Test each interface in the list. Manually run a capture session with Wireshark using an interface from the list.
3. On the host machine, Ping the target IPv4 (of the VM).
4. If the correct interface of the host is used, you'll see the Ping request and reply packet exchanges through Wireshark,
5. Loop to step 2 and repeat, using another interface.

Filter

The packet filter helps capture only those packets associated with the fuzzing session. The filter adheres to the syntax and requirements of the Pcap filter specification.



WireShark refers to the Libpcap filters as capture filters. Use the capture filters in Peach. Wireshark also defines its own display filters that it uses to filter entries in its session files. The display filters are not compatible with Libpcap.

Vmware (Windows virtual machine Automation)

The **Vmware** monitor controls setting up and starting the virtual machine and uses the settings in the following illustration:

Vmware (Windows_VM_Automation)

Name	<input type="text" value="Windows_VM_Automation"/>	
Vmx	<input type="text" value="C:\\bix\\Peach Pro Server\\Peach Pro Server.vmx"/>	
Headless	<input type="text" value="false"/>	
Host	<input type="text"/>	
Host Port	<input type="text" value="0"/>	
Host Type	<input type="text" value="Workstation"/>	
Login	<input type="text"/>	
Password	<input type="text"/>	
Reset Every Iteration	<input type="text" value="false"/>	
Reset On Fault Before Collection	<input type="text" value="false"/>	
Snapshot Index	<input type="text"/>	
Snapshot Name	<input type="text" value="Snap6_IT"/>	
Stop On Fault Before Collection	<input type="text" value="false"/>	
Wait For Tools In Guest	<input type="text" value="true"/>	
Wait Timeout	<input type="text" value="600"/>	

The most significant parameters for the VMware monitor follow:

Vmx

Identifies the full path of the virtual machine image. Peach loads the snapshot of the VM image at the start of the fuzzing job and after a fault occurs.

Headless

Specifies whether the VM connects to a viewing window in the VMware window. When developing a configuration, set this parameter to false. When performing a fuzzing job, the setting doesn't matter.

Host Type

Specifies the VMWare product used in the configuration.

Snapshot Name

Identifies the snapshot to use for the specific image.

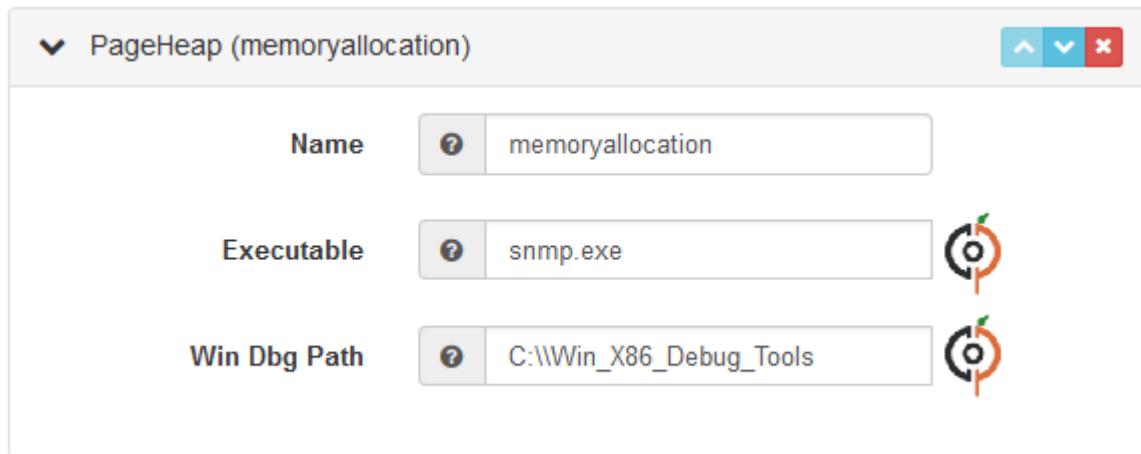
PageHeap (Memory Heap Analyzer)

The [PageHeap](#) monitor enables the Windows Debugger to analyze heap memory allocations throughout the fuzzing session. This monitor manages the registry entries that govern heap monitoring. The monitor sets the entries at the beginning of the fuzzing session and clears them at the end of the session.



PageHeap requires administrative privileges to run correctly.

The PageHeap monitor uses the settings in the following illustration:



The most significant parameters for the PageHeap monitor follow:

Executable

Name of the test target executable file. Provide the file name and extension. The path is not needed.

Win Dbg Path

Folder on the test target containing the Windows debugging tools. Use absolute path from the file system root to the folder.



When using PageHeap with Windows services, run the PageHeap monitor when the service is stopped.

WindowsService

The [WindowsService](#) monitor manages a Windows service. This monitor starts the network service at the start of the fuzzing job, and restarts the service when the VM is refreshed (after a fault). The monitor is housed by the remote agent.

The WindowsService monitor uses the settings in the following illustration:

The screenshot shows a configuration dialog box titled "WindowsService". It contains six settings: "Name" (empty), "Service" (set to "snmp" with a status icon showing green and orange), "Fault On Early Exit" (set to "false"), "Machine Name" (empty), "Restart" (set to "false"), and "Start Timeout" (set to "1"). The dialog has standard window controls (minimize, maximize, close) at the top right.

Name	Value
Name	
Service	snmp
Fault On Early Exit	false
Machine Name	
Restart	false
Start Timeout	1

The most significant parameter for the WindowsService monitor is “Service” that specifies the name of the Windows service to monitor.

WindowsDebugger

The [WindowsDebugger](#) debugger monitor performs two major functions in this recipe:

- Detects faults internal to the service.
- Create log files when a faulting condition occurs.

The WindowsDebugger monitor uses the settings in the following illustration:

WindowsDebugger (WinDbg_OnRemote)

Name: WinDbg_OnRemote

Arguments:

Cpu Poll Interval: 200

Executable:

Fault On Early Exit: false

Ignore First Chance Guard Page: false

Ignore Second Chance Guard Page: false

No Cpu Kill: false

Process Name:

Restart After Fault: false

Restart On Each Test: false

Service: snmp 

Start On Call:

Symbols Path: SRV*http://msdl.microsoft.com/dov

Wait For Exit On Call:

Wait For Exit Timeout: 10000

Win Dbg Path: C:\Win_X86_Debug_Tools 

The most significant parameters for the WindowsDebugger monitor follow:

Service

Name of the test target service on the remote machine. Provide the service name given in the service properties.

Win Dbg Path

Folder on the test target containing the Windows debugging tools. Use absolute path from the file system root to the folder.

Configuration Test

Once the monitors and associated parameters are part of the configuration, you can test the configuration. From the Configuration menu along the left edge of the window, click on **Test** to run a single iteration (test case) on the configuration. Note that the test checks the connections and communications. It does NOT do any fuzzing.

For more information on testing a configuration, see [Test Pit Configuration](#).

8. Agents

A Peach agent is a light weight process that can host multiple monitors. These monitors perform tasks such as fault detection, data collection and automation.

The Peach agent communicates over a network channel to the main Peach process where all captured information is reported. This allows the main Peach process to perform monitoring locally or on the device under test.

Monitors that perform fault detection provide methods to identify when a problem with the target under test during testing. For example, you might use a debugger monitor to detect when a target crashes and collect information about the crash. Peach finds more issues when robust fault detection is configured.

Data collection monitors are used to gather additional information about a fault that has occurred. This can include taking a network capture of the test traffic, collecting log files and running scripts to collect information about the target state. The goal is to collect any information that will be useful in tracking down the root cause of the faulting condition.

Automation monitors are used to automate the target and target environment. This can include startup automation such as configuring an environment and starting the target. Restarting the environment when a fault occurs so testing can continue. And finally shutting down the target/environment when testing has completed. Some configurations may also require triggering the target to connect/consume data.

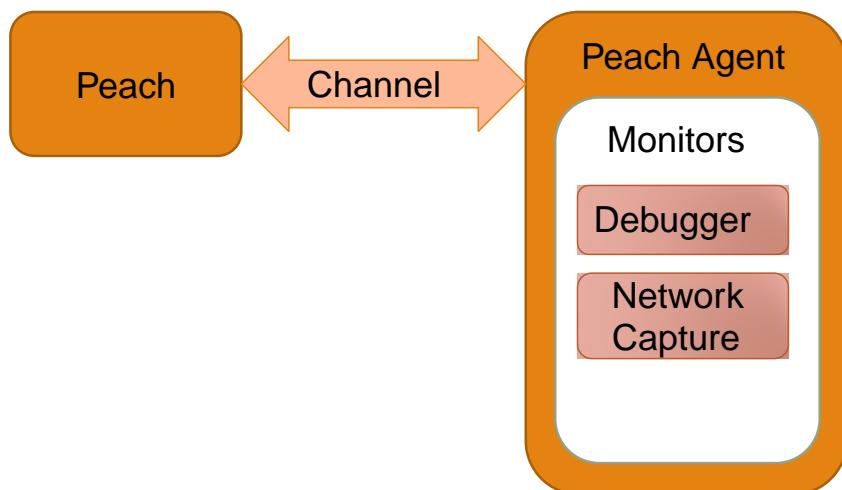


Figure 1. Peach Agent Block Diagram

Agent Privileges

Some monitors or publishers hosted by an agent process sometimes require heightened privileges. If you receive an error regarding permissions, try running the agent with root or administrative privileges.

OS X/Linux/Unix

On OS X, Linux and other Unix systems, a process can be launched as root using the *sudo* utility.

For local agents:

```
sudo ./peach
```

For remote agents:

```
sudo ./peachagent
```

Windows

On Windows a process can be started with administrator rights by right-clicking and selecting "Run as Adminsitritor". Optionally the user can be given additional needed priviledges by the system administrator.

Privileged Monitors

Which monitors require additional privileges depends on the specific OS and privileges of the user, but typically the following require special rights:

- PageHeap (Windows)
- NetworkCapture (OS X/Linux/Unix)

Network Firewall

Many modern operating systems such as Windows and Linux come with a firewall that prevents incoming or outgoing network traffic. This can prevent a remote agent from communicating with Peach. On Windows the user is asked when program first runs if a firewall rule should be added.

On Linux, specifically Ubuntu distributions, this command allows packets through the firewall using port 9001 (default the port for remote agents).

```
sudo ufw allow 9001
```

8.1. Agent Channels

Local Agent (local)

The agent is hosted in the current Peach process. This is the default channel. An example agent URL is `local://`

Remote Agent (tcp)

The external Peach agent communicates using network calls over TCP. Peach can run as a remote agent using the [PeachAgent](#) program. Remote agents do not require a valid license and are only used to host monitors or remote publishers. An example agent URL is `tcp://192.168.1.2:9002`

Custom Agent (RESTful API) (http)

This agent protocol uses RESTful style calls over HTTP. This channel is best suited for custom Peach agents. For more information, see the [Agent](#) topic in the [Extending Peach](#) section of the *Peach Professional Developer Guide*. An example agent URL is `tcp://192.168.1.2:8080`

8.1.1. Local Agent

The peach runtime supports a local agent that runs in process. LocalAgent is the default agent unless another agent type is specified.

Agent URL:

```
local://
```

8.1.2. Remote Agent

Peach includes a remote [agent server](#) that can be used to host Monitors and Publishers on remote machines. Usage of a remote agent requires a location URL of the following format:

Agent URL:

```
tcp://HOST:PORT
```

HOST

Remote host the agent is running on

PORT

Remote port the agent is bound to (defaults to 9001)

Example:

```
tcp://192.168.1.100:9001
```

9. Monitors

Agents are special Peach processes that can be run locally or remotely. These processes host one or more Monitors that can perform such actions as attaching debuggers, watching memory consumption, or detecting faults.

The following table lists each monitor by name, function type (Fault detection, Data collection, Automation), and by the operating systems that support the monitor. If the operating systems column is blank for an entry, that monitor is available in all supported operating systems.

Table 1. Monitors

Monitor	Fault Detection	Data Collection	Automation	Operating Systems [1: When an operating system is listed, the monitor is available only for the listed operating system. A blank entry indicates the monitor is available for Windows, Linux, and OS X operating systems.]
Android	X		X	
Android Emulator	X		X	
APC Power			X	
ButtonClicker			X	Windows
CanaKit Relay			X	
CAN Capture		X		
CAN Error Frame	X			
CAN Send Frame			X	
CAN Timing	X			
CAN Threshold	X			
Cleanup Folder			X	
Cleanup Registry			X	Windows
Crash Reporter	X	X		OS X
Crash Wrangler	X	X	X	OS X
Gdb	X	X	X	Linux, OS X
GdbServer	X	X	X	Linux, OS X

Monitor	Fault Detection	Data Collection	Automation	Operating Systems [1: When an operating system is listed, the monitor is available only for the listed operating system. A blank entry indicates the monitor is available for Windows, Linux, and OS X operating systems.]
IpPower9258			X	
LinuxCoreFile	X	X		Linux
Memory	X			
NetworkCapture	X	X		
Page Heap			X	Windows
Ping	X			
Popup Watcher	X		X	Windows
Process Launcher	X		X	
Process Killer			X	
Run Command	X	X	X	
Save File		X		
Serial Port	X	X	X	
SNMP Power			X	
Socket Listener	X	X		
SSH Command	X	X	X	
SSH Downloader		X		
Syslog	X	X	X	
TcpPort	X	X	X	
Vmware Control			X	
Windows Debugger	X	X	X	Windows
Windows Service	X		X	Windows

9.1. Android Monitor

Monitor Categories: Automation, Fault detection

The *Android* monitor examines both targeted Android applications and the state of the Android OS. Peach supports Android OS versions 4.0 to 5.1, inclusively.

Two expected use cases are:

- Maintaining state while fuzzing native code
- Launching and monitoring Android Java applications.

The *Android* monitor can start or restart the device at the following times:

- The start of a fuzzing run
- The start of each test iteration
- The start of an iteration that immediately follows a fault
- When called from the state model

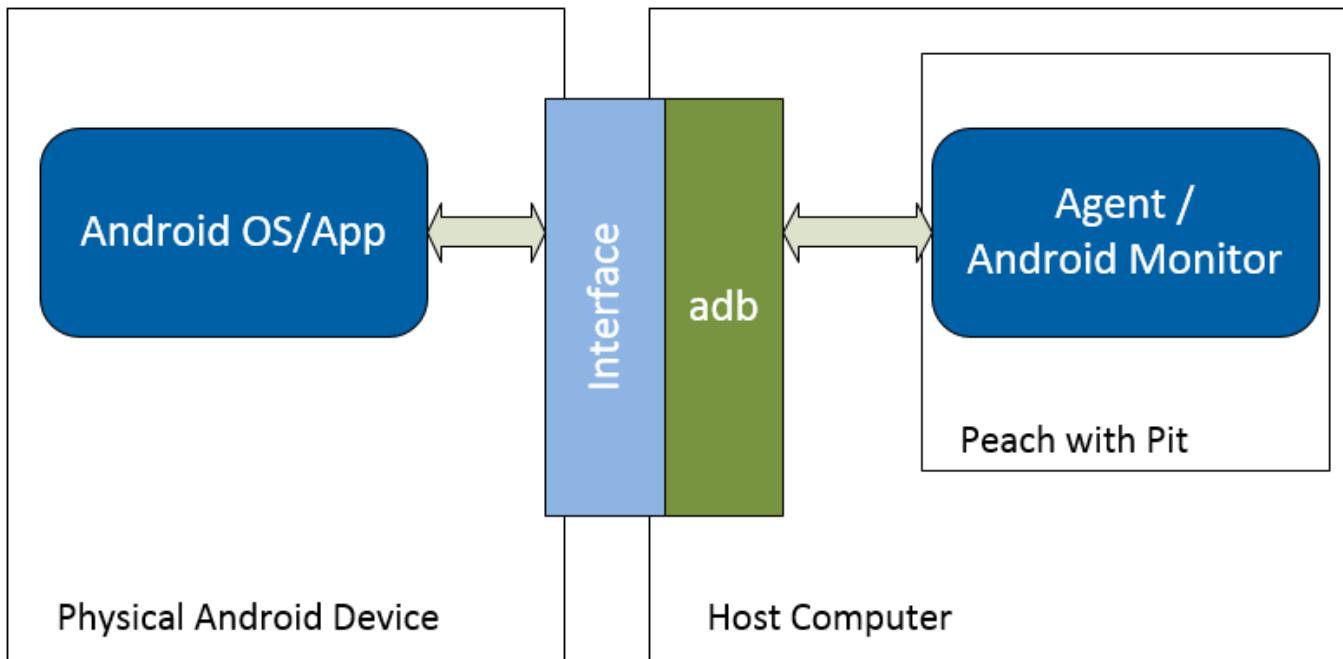
The Android monitor watches the message logs and the device, and can generate faults for the following conditions:

- A logging message matches the fault search criteria, Peach logs a fault.
- A logging message matches the fault search criteria, Peach logs a fault and stops fuzzing.
- A physical device becomes non-responsive.
- A virtual device becomes non-responsive or lost.

After detecting a fault, the monitor collects data from the device log files and crash dumps.

Additionally, the monitor logs exceptions, and updates fault bucket information. For bucketing, Peach uses the text from the fault to determine the major bucket level. The minor bucket level is not used. The risk evaluation looks for error, fatal error, or unknown.

The *Android* monitor uses the Android Debugging Bridge (adb) to communicate with a device. This monitor can target both emulated and physical devices. The *Android* monitor requires [Android Platform Tools](#) and either an emulator or a physical Android device. The configuration for a physical device follows. For a configuration using a virtual device, see the [Android Emulator Monitor](#).



Connecting to a physical device requires the device serial number. You can obtain this from a connected device by using the following adb command: "adb devices". The result is a list of devices that adb found. The information for each device consists of two parts: the device number and the connection status between adb and the device. The list includes physical and virtual devices.



For more information about debugging Android devices, see the following:

- [Android Debug Bridge](#).
- [How To Install and Use ADB, the Android Debug Bridge Utility](#)

9.1.1. Parameters

Required:

ApplicationName

Name of the Android application.

Optional:

ActivityName

Name of the application activity, defaults to "".

AdbPath

Directory path to adb, defaults to "".

ClearAppData

Removes the application data and cache every iteration, defaults to false.

ClearAppDataOnFault

Removes the application data and cache on faulting iterations, defaults to false.

CommandTimeout

Sets the maximum number of seconds to wait for the adb command to complete, defaults to 10 seconds.

ConnectTimeout

Sets the maximum number of seconds to wait to establish an adb connection, defaults to 5 seconds.

DeviceMonitor

Identifies the Android monitor that supplies the device serial number, defaults to "". Used when monitoring a virtual device.

DeviceSerial

The serial number of the device to monitor, defaults to "". Used when monitoring a physical device.

FaultRegex

Specifies a regular expression; when matched from a log entry, triggers a fault. The default pattern is (^E/ActivityMonitor)|(^E/AndroidRuntime)|(^F/.*)

FaultWaitTime

Sets the time period, in milliseconds, to wait when checking for a fault, defaults to 0 ms.

IgnoreRegex

Specifies a regular expression; when matched, the monitor ignores potential false positive faults, defaults to "".

MustStopRegex

Specifies a regular expression; when a match occurs, the monitor triggers a fault and stops fuzzing, defaults to "".

ReadyTimeout

Sets the maximum number of seconds to wait for the device to reach readiness—able to respond to inputs, defaults to 600 seconds.

RebootEveryN

Specifies the number of iterations between successive device reboots, defaults to 0.

RebootOnFault

Reboots the device when a fault occurs, defaults to false.

RestartEveryIteration

Restarts the application every iteration, defaults to false.

StartOnCall

Starts the application when notified by the state machine. The string value used here must match the Call Action statement of the state model. The default string is "".

WaitForReadyOnCall

Waits for the device to be ready when notified by the state machine. The string used here must match the corresponding Call Action statement of the state model. the default string is "".



The DeviceMonitor and the DeviceSerial parameters are mutually exclusive. Use DeviceSerial to provide the serial number of a physical device. Use DeviceMonitor when using the Android Emulator, as the Emulator will provide the serial number of the virtual device.

9.1.2. Examples

Example 11. Basic Usage with a Physical Device

This parameter example is from a setup that the BadBehaviorActivity, sending random taps to generate different types of exceptions and crashes. The setup is for a physical Android device.

Android Monitor (App) Parameters

Parameter	Value
ApplicationName	<code>com.android.development</code>
ActivityName	<code>.BadBehaviorActivity</code>
AdbPath	<code>C:\adt-bundle-windows-x86_64-20131030\sdk\platform-tools</code>
DeviceSerial	<code>emulator-5554</code>

Example 12. Basic Usage with a Virtual Device

This parameter example is from a setup that the BadBehaviorActivity, sending random taps to generate different types of exceptions and crashes. The setup is for a virtual Android device, and uses the Android monitor, as well as the Android Emulator monitor.

If you want to run the Android emulator, set your AdbPath to the directory containing the adb (Android Debug Bridge) platform-tools directory and point the EmulatorPath in the Android Emulator Monitor to the adb tools directory.

The Avd parameter in the Android Emulator Monitor must also be the name of a valid AVD (Android Virtual Device). Use the following steps to create a new AVD:

1. Open the *android.bat* file located in the adb SDK tools directory.
2. From the GUI that opens, click on *Tools* in the menu bar, then *Manage AVDs*....
3. From the window that opens, click *New...* and create a new AVD.

Android Emulator (Emu) Monitor Parameters

Parameter	Value
Avd	Nexus4
EmulatorPath	C:\adt-bundle-windows-x86_64-20131030\sdk\tools

Android Monitor (App) Parameters

Parameter	Value
ApplicationName	com.android.development
ActivityName	.BadBehaviorActivity
AdbPath	C:\adt-bundle-windows-x86_64-20131030\sdk\platform-tools
DeviceMonitor	Emu

9.2. AndroidEmulator Monitor

Monitor Categories: Automation, Fault detection

The *AndroidEmulator* monitor handles setup and teardown for Android Virtual Devices (AVDs) running within the Android Emulator. This monitor detects faults related to the emulator operation, not fuzzing results.

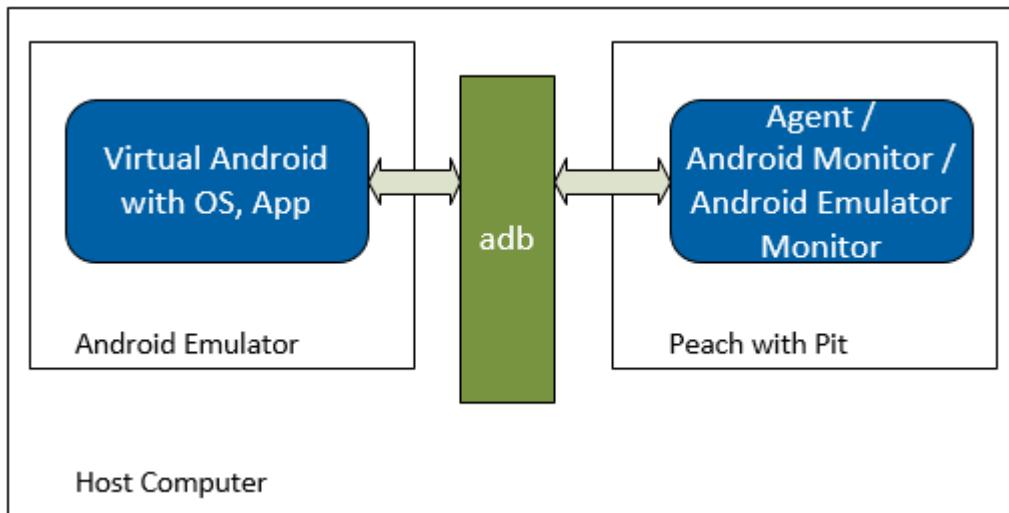
The monitor provides the following functionality:

- Start a virtual device at the start of a fuzzing run.
- Start a virtual device at the start of each test iteration.
- Start a virtual device when called from the state model.
- Start a virtual device at the beginning of an iteration that immediately follows a fault.
- Shuts down at the end of a fuzzing session.
- Logs timeout messages when querying the emulator for the device serial number.
- Logs timeout messages when shutting down the emulator.
- logs messages sent to StdErr.
- Logs messages sent to StdOut.

This monitor requires the following items to run:

- The Android monitor that watches the OS and application being fuzzed.
- The [Android Emulator](#)
- The [Android Platform Tools](#).

The fuzzing configuration for a virtual or emulated device follows. For a configuration using a physical device, see the [Android Monitor](#).



9.2.1. Parameters

Required:

Avd

Android virtual device.

Optional:

EmulatorPath

Directory containing the Android emulator. If not provided, Peach searches the directories in the PATH variable for the installed emulator.

Headless

If true, runs the emulator **without a display**. Defaults to false.

RestartAfterFault

If **true**, restarts the emulator when any monitor detects a fault. If **false**, restarts the emulator only if the emulator exits or crashes. This argument defaults to **true**.

RestartEveryIteration

Restart emulator on every iteration. Defaults to false.

StartOnCall

Start the emulator when notified by the state machine.

StartTimeout

How many seconds to wait for emulator to start running. Defaults to 30.

StopTimeout

How many seconds to wait for emulator to exit. Defaults to 30.

9.2.2. Examples

Example 13. Basic Usage Example

This parameter example is from a setup that the BadBehaviorActivity, sending random taps to generate different types of exceptions and crashes. The setup is for a virtual device that uses the Android Emulator Monitor, as well as the Android monitor.

In order to run the Android emulator, set the EmulatorPath in the Android Emulator Monitor to the adb tools directory, and set the Avd parameter to the name of an Android virtual device. Here the name of the virtual device is "Nexus4".

In the Android monitor, set the AdbPath to the platform-tools directory containing the adb (Android Debug Bridge).

Android Emulator (Emu) Monitor Parameters

Parameter	Value
Avd	Nexus4
EmulatorPath	C:\adt-bundle-windows-x86_64-20131030\sdk\tools

Android Monitor (App) Parameters

Parameter	Value
ApplicationName	com.android.development
ActivityName	.BadBehaviorActivity
AdbPath	C:\adt-bundle-windows-x86_64-20131030\sdk\platform-tools
DeviceMonitor	Emu



Position the Android Emulator Monitor before (above) the Android monitor in your Pit, so that, at run time, the virtual device exists when adb tries to connect to it. Peach executes the monitors in the order that they are listed in the fuzzing definition.

9.3. APC Power Monitor

Monitor Categories: Automation

The *APC Power* monitor switches outlets on an APC power distribution unit (PDU) on and off via SNMPv1. This monitor is useful for automatically power cycling devices during a fuzzing session. APC's Switched Rack Power Distribution Unit (AC7900) is known to work with this monitor.

Each *APC Power* monitor switches one or more of a PDU's outlets, according to the configuration. All affected outlets are given the same commands, so turning some outlets on and others off would require another monitor. The monitor can reset the power outlets at the following points in time:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model



The [IpPower9258 Monitor](#) provides similar features, specific to the IP Power 9258 devices. The [SnmpPower Monitor](#) is designed to work with non-APC PDUs that can be controlled via SNMPv1. For controlling power to a device by wiring through a relay, Peach provides a monitor for the [CanaKit 4-Port USB Relay Controller](#).

9.3.1. Parameters

Required:

Host

IP address of the switched power distribution unit.

OutletGrouping

Whether outlets on the PDU are identified individually ([Outlet](#)) or in groups ([OutletGroup](#)). Default is [Outlet](#).

Outlets

Comma-separated list of numeric identifiers for outlets or outlet groups to control.

Optional:

Port

SNMP port on the switched power distribution unit. Default is [161](#).

ReadCommunity

SNMP community string to use when reading the state of the outlets. Default is [public](#).

WriteCommunity

SNMP community string to use when modifying the state of the outlets. Default is **private**.

RequestTimeout

Maximum duration in milliseconds to block when sending an SNMP request to the PDU. Default is **1000**.

SanityCheckOnStart

On startup, ensure switch state changes persist. Default is **true**.

SanityCheckWaitTimeout

Maximum duration to wait for state change to take effect during startup sanity check. Default is **3000**.

ResetOnCall

Reset power when the specified call is received from the state model. This value is used only when the *When* parameter is set to **OnCall**.

PowerOffOnEnd

Power off when the fuzzing session completes, default is **false**.

PowerOnOffPause

Pause in milliseconds between power off/power on, default is **500**.

When

When to reset power on the specified outlets or outlet groups. Default is **OnFault**.

"When" Setting	Description
DetectFault	Reset power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Reset power when the fuzzing session starts. This occurs once per session.
OnEnd	Reset power when the fuzzing session stops. This occurs once per session.
OnIterationStart	Reset power at the start of each iteration.
OnIterationEnd	Reset power at the end of each iteration.
OnFault	Reset power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Reset power at the start of an iteration that immediately follows a fault detection.

"When" Setting	Description
OnCall	Reset power when the call specified by the <i>ResetOnCall</i> parameter is received from the state model.

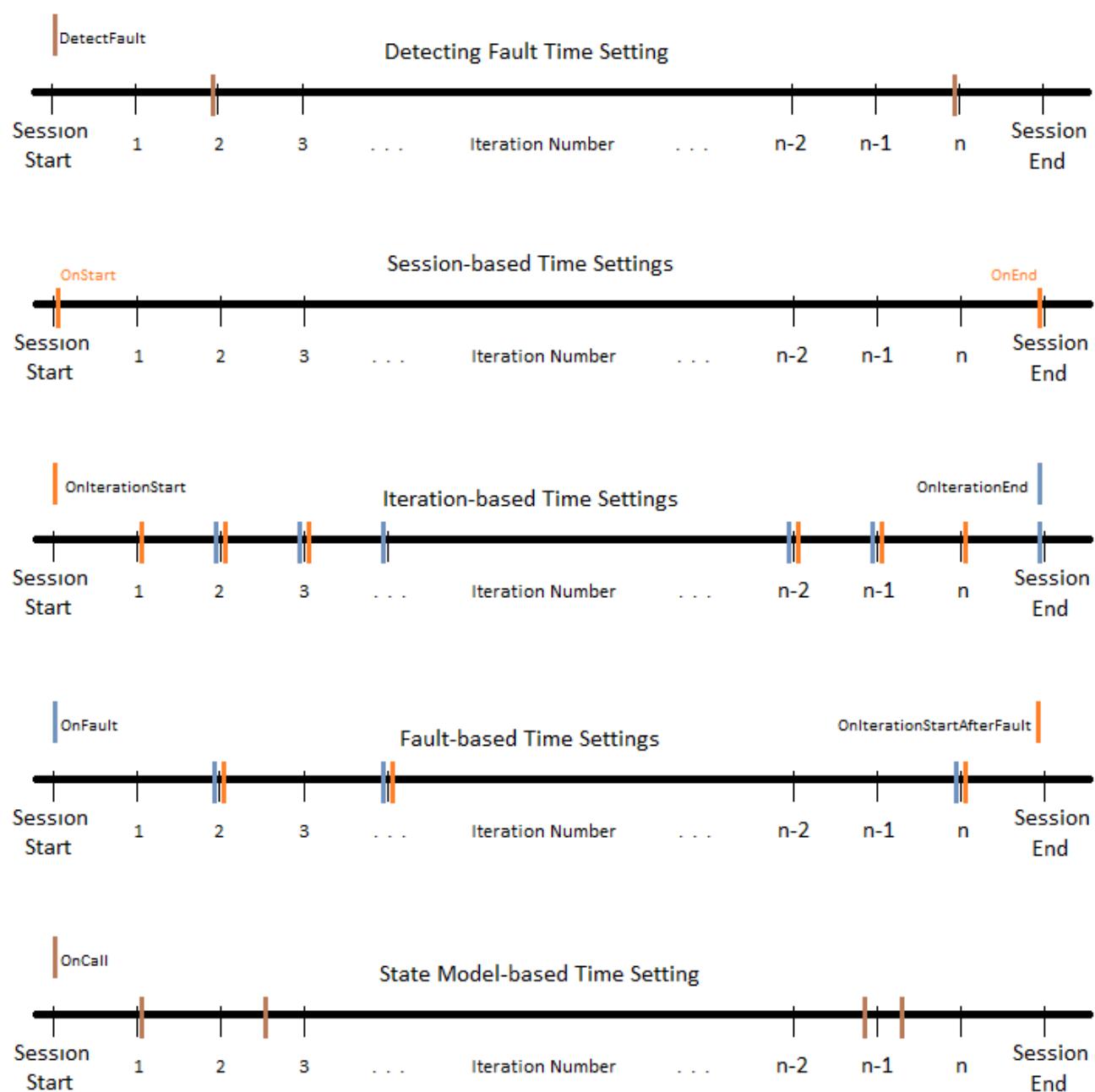


Figure 2. When Choices for Performing an Action

9.3.2. Examples

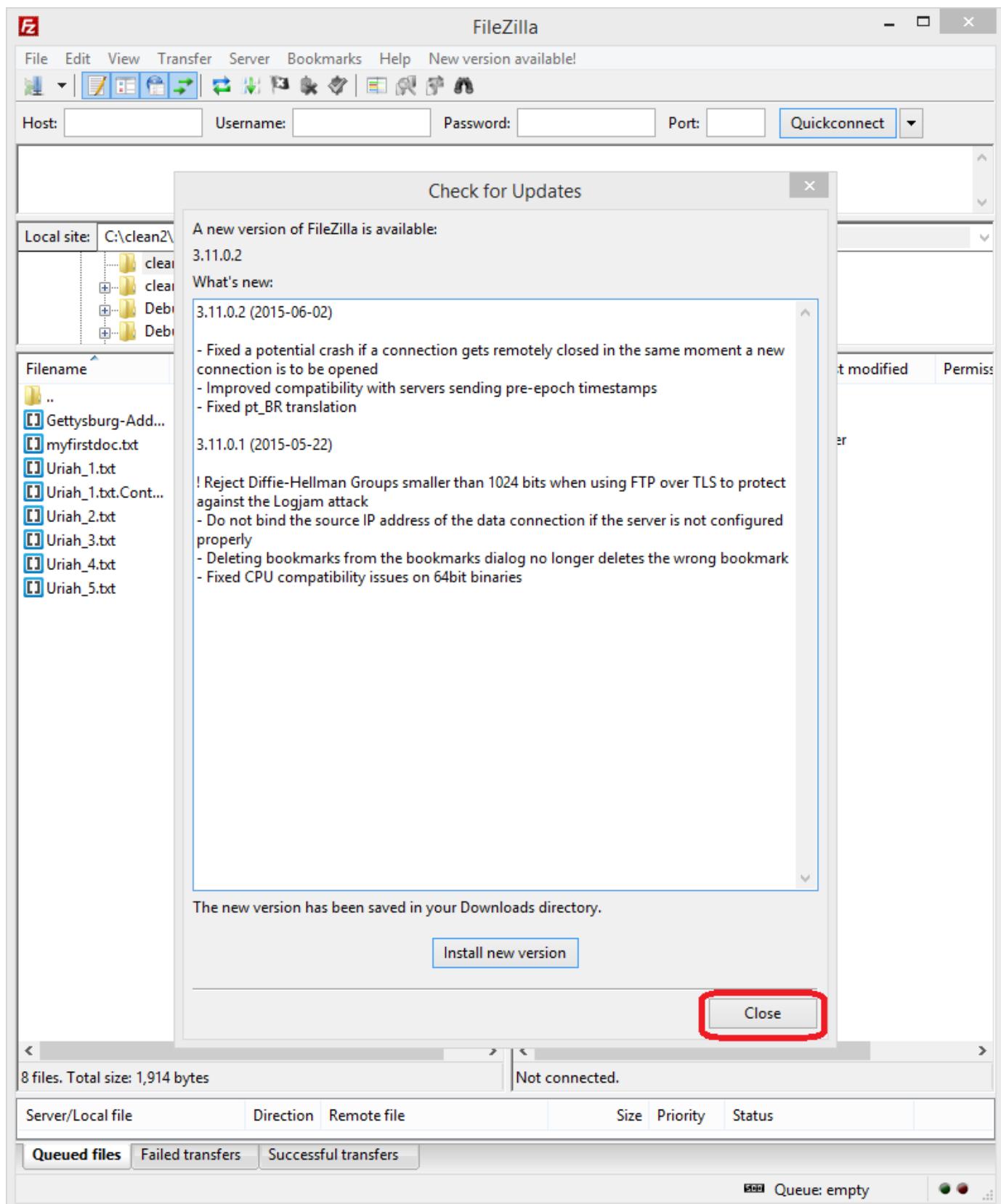
9.4. ButtonClicker Monitor

Monitor Category: Automation

The *ButtonClicker* monitor provides automation functionality by clicking buttons in the Windows GUI. This monitor runs in the Windows environment. *ButtonClicker* watches and clicks the appropriate button within the specified window. You can use *ButtonClicker* to click a button in the window, such as "OK" or "Close".

The intent of this monitor is to initiate an action or to close a window to keep a fuzzing session active. *Buttonclicker* runs from the beginning of a fuzzing session to the end of the session.

The following example uses an FTP client, FileZilla, that sometimes opens a popup window asking whether to install an update. Here, *ButtonClicker* monitor provides a mouse click to the "Close" button to close the popup window. With the popup window out of the way, fuzzing continues without delay.



Another monitor to consider for watching popup windows is [PopupWatcher](#), that can monitor several popup windows.

9.4.1. Parameters

Required:

WindowText

Text from the window title that identifies the window to receive the button click. The text string can be part or all of the window title.

ButtonName

Text label of the button to click. The label is displayed to the user, and is on or near the button that will receive the click.

Optional:

None.

If Peach has trouble clicking a button, the button might have a link to a shortcut key.

The shortcut key is displayed in the label using an underlined character. Windows does this by inserting an ampersand "&" immediately before the shortcut key within the button label. Further, not all underlining shows in the initial display of the window.

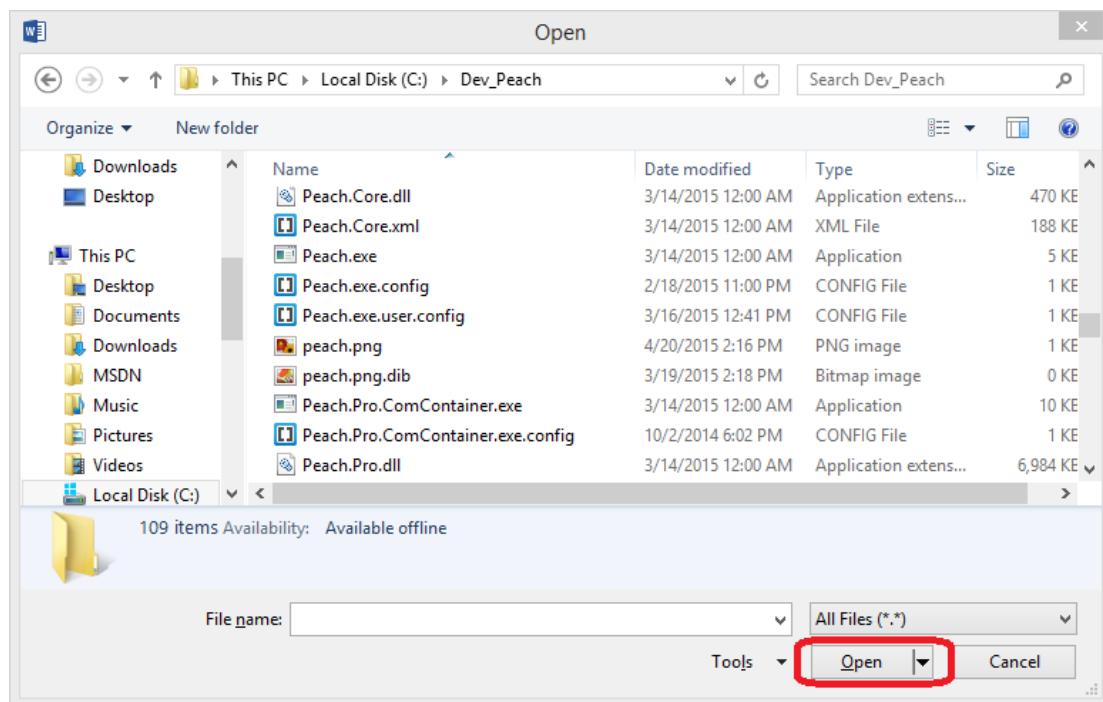
In an application, you can manually force underlining to display in window buttons by pressing the <CTRL> or <ALT> key. Once you find the underlining in the application, you can adjust the value of the **ButtonName** parameter for *ButtonClick* by inserting an ampersand (&) immediately before any underlined character. Then, Peach will find the button to click.

For example, in Microsoft Word, the Browsing dialog box used to open a document does not immediately display underlining in its command buttons. You can see this by following the sequence that opens a file:

1. Click **File** on the ribbon.
2. Click **Open** on the vertical menu.
3. Click **Computer** in the Open column.
4. Click the **Browse** button.



The dialog opens and the "Open" button in the lower right corner is not underlined. Press the <ALT> or <CTRL> key to see the underlining as in the following illustration.



9.4.2. Examples

Basic Usage Example

This parameter example is from the FTP client previous listed. The monitor will respond to the popup window that asks whether to install an update.

The ButtonClicker monitor uses the following parameter settings to click the "Close" button in the popup window :

Parameter	Value
WindowText	for Updates
ButtonName	Close

9.5. CanaKitRelay Monitor

Monitor category: Automation

The *CanaKitRelay* monitor provides automation to a number of test configurations:

- Control power to an external device that is the fuzzing target. You can turn it on at the start of the test, then toggle power after a fault occurs to return the device to a known, stable state.
- Control a supporting device in a fuzzing test configuration. Supporting devices can include recording devices, lighting, heating devices, sound or motion generators. An example is turning on an espresso machine after every 10,000th test iteration.
- Emulate a button push by inserting the device into a circuit containing the button. The idea is to automate the press of a button, such as **NumLock**, or the play button on a surveillance device.
- For simulating the attachment or removal of a cable, such as USB, by routing the power line (VCC) through the relay.

The CanaKit is an external product that you can add as part of your test configuration. The kit consists of 4 relays. A relay is an electrically operated switch that uses an electromagnet to operate a switching mechanism. Each relay in the kit is capable of controlling a 5-amp, 110V AC or a 24V DC circuit. Communication with the kit occurs over USB.

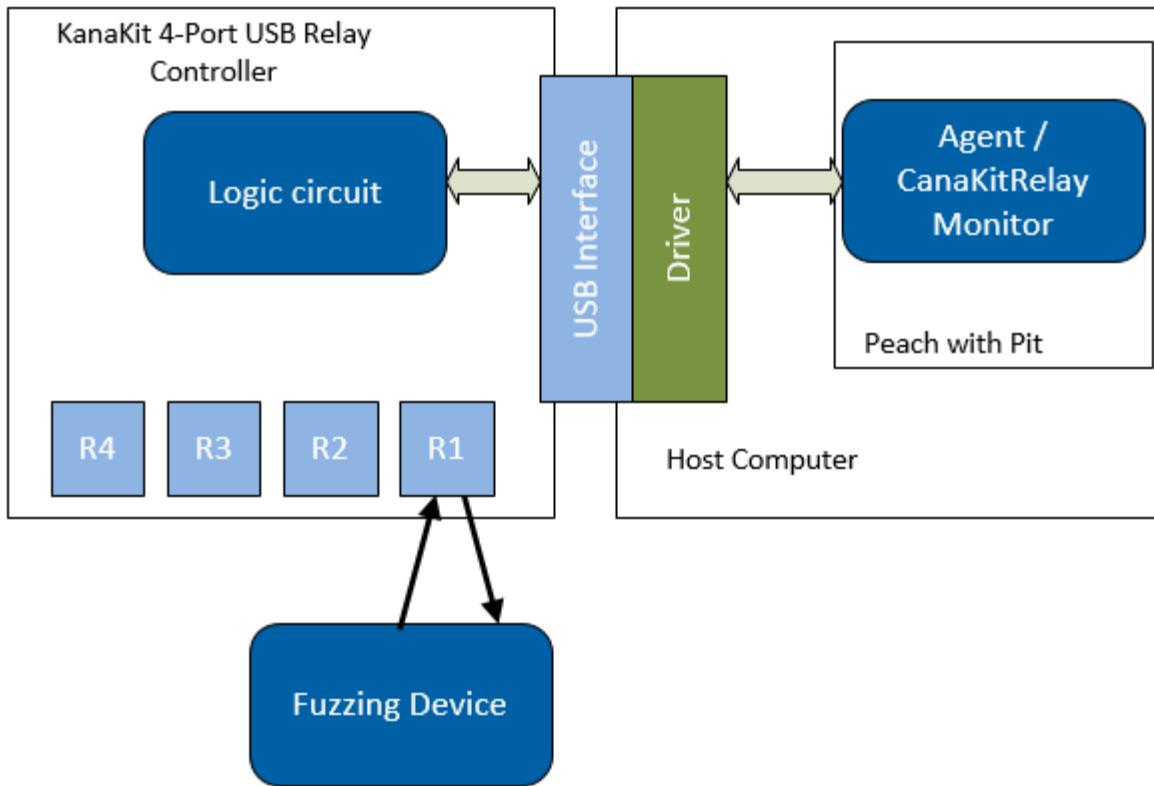
For more information on the kit, including configurations, installation, and the relay command set, see [CanaKit 4-Port USB Relay Controller](#).

You can purchase the CanaKit from the manufacturer at the previous web site or from Amazon.com. In each case, the kit price is about \$60.00 U.S. plus shipping costs.

The *CanaKitRelay* monitor controls one relay in a kit. Use one monitor per relay to control multiple relays concurrently. Within a fuzzing session, the monitor can trigger the relay at the following times:

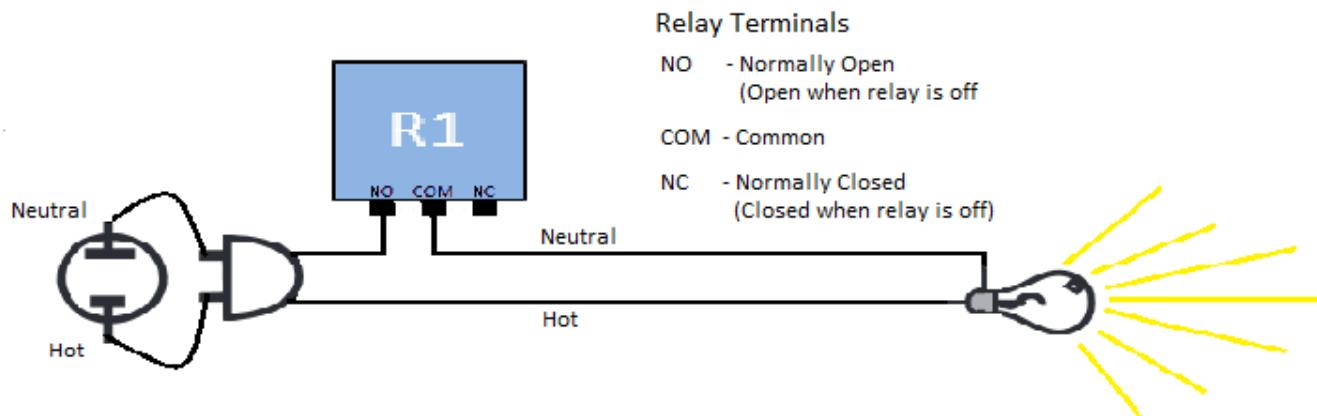
- At the start or end of a fuzzing run
- At the start or end of each test iteration
- During the detection of a fault
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

The following diagram shows a sample configuration that controls power through Relay 1.



Each relay supplies three terminals: Normally Open (NO), Common (COM), and Normally Closed (NC). Basic configurations will connect the hot wire to Common and the other wire to either NO or NC. For DC connections, attach the anode (+) to the Common terminal.

NO provides an open circuit when the relay is off. NC provides a closed circuit when the relay is off, giving opposite on/off states from NO. The following diagram shows the terminal layout and imagines that Relay 1 is On.



 The CanaKit Relay requires a driver that is available at the supplier's website. At the time of this writing, the driver is unsigned, which forces you to turn off Driver Signing Enforcement when installing the driver in 64-bit Windows 8. For instructions on how to install unsigned drivers in this environment, see [How to Disable Driver Signature Verification on 64-Bit Windows 8.1](#).

After installing the CanaKit device driver, connect the unit to your PC. Windows dynamically assigns a serial port to the USB channel for the connection. You can see the port assignment by looking at the [Ports\(COM and LPT\)](#) entry in the Device Manager. The Device Manager is available from the System applet in the Control Panel.



For controlling power to devices using 3-prong outlets, Peach provides the [IpPower9258](#), [ApcPower](#), and [SnmpPower](#) monitors.

9.5.1. Parameters

Required:

SerialPort

Serial port for the board (such as COM2).

RelayNumber

Relay to trigger (1, 2, 3, or 4). Each relay number corresponds to a single relay in the kit, as shown in the first diagram.

Optional:

Action

Perform an action on the specified relay, defaults to ToggleOff. Valid actions include the following:

Action	Description
ToggleOff	Sets the relay to the OFF position, then sets the relay to the ON position.
ToggleOn	Sets the relay to the ON position, then sets the relay to the OFF position.
SetOn	Sets the relay to the ON position.
SetOff	Sets the relay to the OFF position.

StartOnCall

Toggle power when the specified call is received from the state model. This value is used only when the *When* parameter is set to [OnCall](#).

ToggleDelay

Pause in milliseconds between off/on, defaults to **500**. Formerly named OnOffPause.

When

Specify one of the following values to determine when a relay should be toggled:

"When" Setting	Description
DetectFault	Toggle power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Toggle power when the fuzzing session starts. This occurs once per session.
OnEnd	Toggle power when the fuzzing session stops. This occurs once per session.
OnIterationStart	Toggle power at the start of each iteration.
OnIterationEnd	Toggle power at the end of each iteration.
OnFault	Toggle power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Toggle power at the start of an iteration that immediately follows a fault detection.
OnCall	Toggle power when the call specified by the <i>StartOnCall</i> parameter is received from the state model.

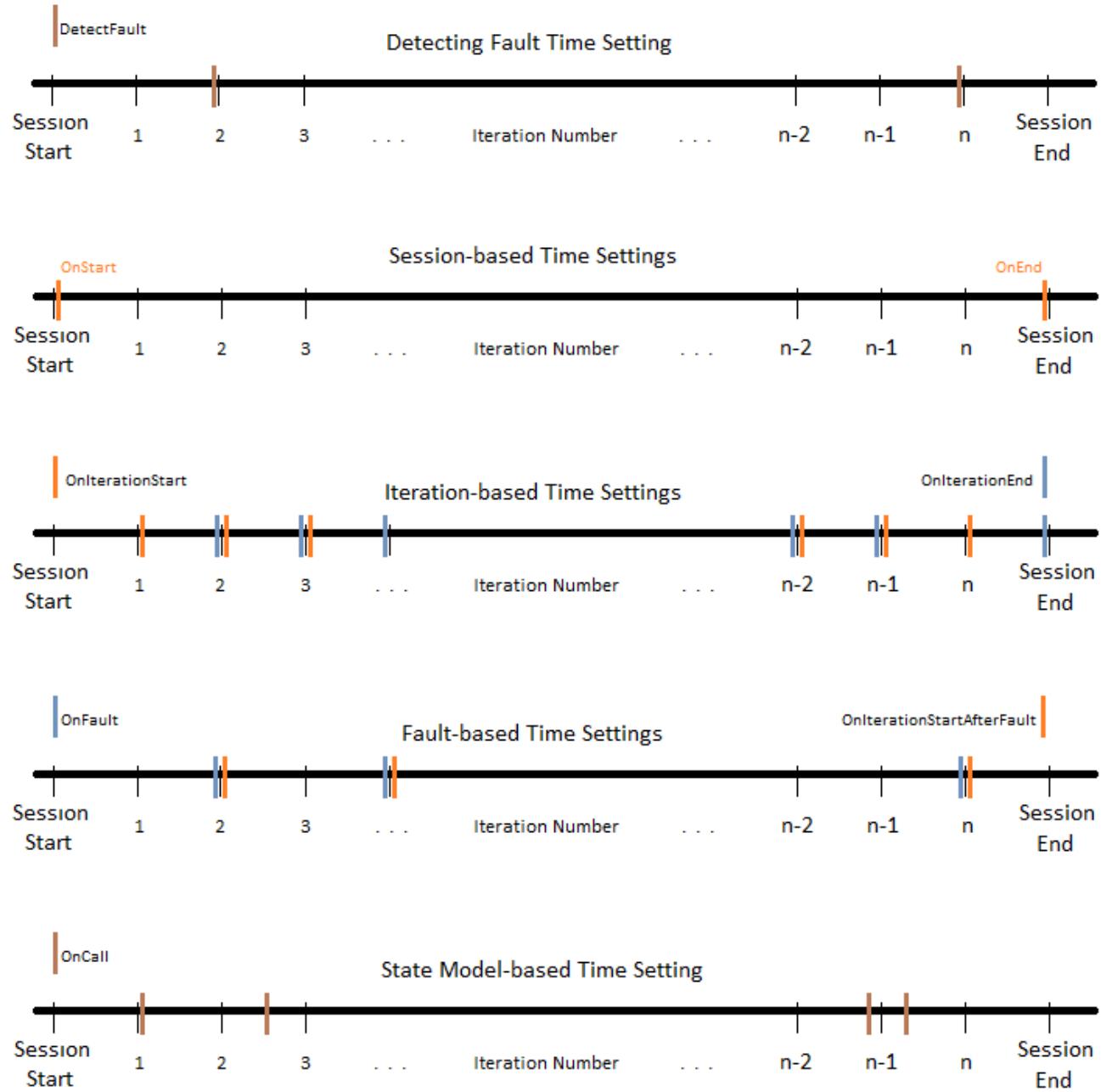


Figure 3. When Choices for Performing an Action

9.5.2. Examples

Reset power on Relay 1

This uses the CanaKitRelay monitor to reset relay 1, which toggles the power off, then back on. A device attached to this relay will restart when the relay resets. The default setting for the *When* parameter is **OnFault**, so the relay will be toggled after a fault is detected.

Parameter	Value
SerialPort	COM5

Parameter	Value
RelayNumber	1

9.6. CAN Capture Monitor

Monitor Category: Data Collection

The *CAN Capture* monitor collects all frames received (but not transmitted) on the specified interface. If a fault occurs, the capture is saved in the *PCAP* format, loadable into Wireshark for analysis.



Not all information from the CAN frame is visible in Wireshark. Basic flags and data are available, but not the full frame. This is a limitation of the format and not Peach Fuzzer.

9.6.1. Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet transmission (default 500,000)

Optional:

None

9.7. CAN Error Frame Monitor

Monitor Category: Fault detection

The *CAN Error Frame* monitor is will trigger a fault when an error frame is received on the specified device/channel. Multiple *CAN Error Frame* monitors can be configured to monitor different devices/channel combinations.

9.7.1. Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet reception (default 500,000)

Optional:

No optional parameters supported on this monitor.

9.7.2. Examples

Example 14. Fault if an error frame is received+

This parameter example is from a setup that uses the CAN Error Frame Monitor to fault if an error frame is received.

Parameter	Value
CanDriver	<i>Vector XL</i>
CanChannel	1
CanBitrate	(default)

9.8. CAN Send Frame Monitor

Monitor Category: Automation

The *CAN Send Frame* monitor is used to send CAN frames that are not fuzzed. Frames can be sent at specific points during a test case, every N milliseconds, or both. This can be useful to simulate messages required by a target.

The CAN frames being sent can share the same driver and channel as the fuzzing, or use a different driver/channel.



The DLC is auto set based on the data size provided.

9.8.1. Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet transmission (default 500,000)

Id

CAN Frame ID field in hex.

Data

CAN data to transmit in hex.

Optional:

SendEvery

Send frame every N milliseconds. Disable by setting to zero. Defaults to 0.

When

Specify one of the following values to determine when a CAN frame should be sent (defaults to **OnStart**):

"When" Setting	Description
DetectFault	Send CAN frame when checking for a fault. This occurs after OnIterationEnd.

"When" Setting	Description
OnStart	Send CAN frame when the fuzzing session starts. This occurs once per session.
OnEnd	Send CAN frame when the fuzzing session stops. This occurs once per session.
OnIterationStart	Send CAN frame at the start of each iteration.
OnIterationEnd	Send CAN frame at the end of each iteration.
OnFault	Send CAN frame when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Send CAN frame at the start of an iteration that immediately follows a fault detection.
OnCall	Send CAN frame when the call specified by the <i>StartOnCall</i> parameter is received from the state model.

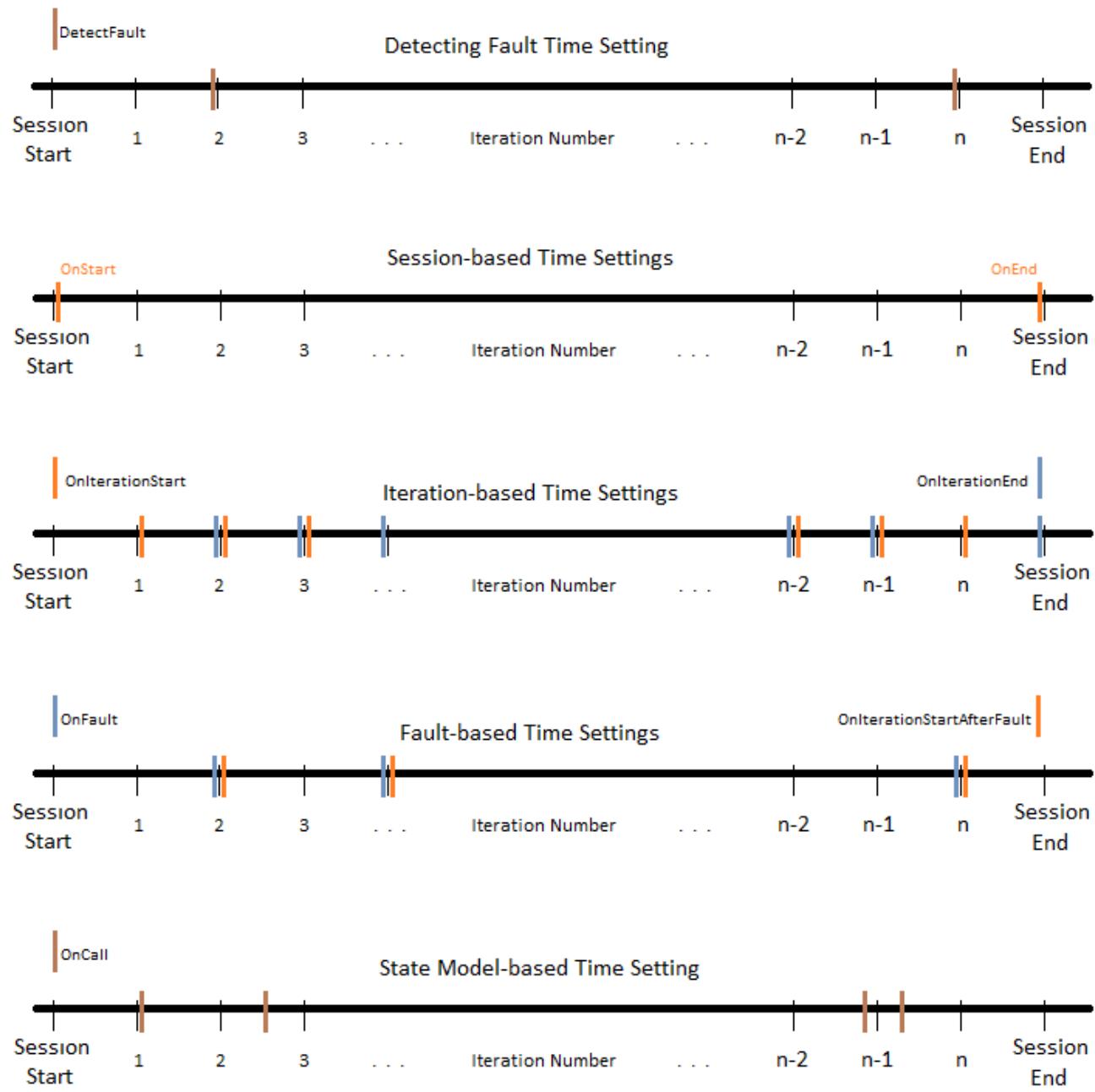


Figure 4. When Choices for Performing an Action

StartOnCall

Send frame when the specified event is received from the state model. This value is used only when the *When* parameter is set to **OnCall**.

9.8.2. Examples

Example 15. Send CAN Frame Every 500 ms

This parameter example is from a setup that uses the CAN Send Frame monitor to send a CAN frame every 500 ms.

Parameter	Value
CanDriver	Vector XL
CanChannel	1
CanBitrate	(default)
Id	0x07DC
Data	AA BB CC DD EE FF
SendEvery	500
When	(default)
StartOnCall	(default)

9.9. CAN Timing Monitor

Monitor Category: Fault detection

The *CAN Timing* monitor will trigger a fault when a frame is not received within a specified time window. Many CAN targets are designed to send one or more CAN frames every N milliseconds. When transmission of these frames stops or is out of spec, this can indicate a failure in the target device.

The CAN frames being sent can share the same driver and channel as the fuzzing, or use a different driver/channel.

9.9.1. Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet reception (default 500,000)

Id

CAN Frame ID field in hex to expect

Window

Reception window (how often frame should be received) in milliseconds. If frame with **Id** is not received in this window of time, a fault will be raised.

Optional:

9.9.2. Examples

Example 16. Expect frame every half second

This parameter example is from a setup that uses the CAN Timing Monitor to fault if a specific CAN frame is not received every half second.

Parameter	Value
CanDriver	Vector XL
CanChannel	1
CanBitrate	(default)
Id	0x07DC
Window	500

9.10. CAN Threshold Monitor

Monitor Category: Fault detection

The *CAN Threshold* monitor triggers a fault when a CAN signal is outside a specified threshold. The threshold is provided as a python expression that must evaluate to bool true or false.

This is one of several ways to monitor a CAN target during fuzzing to determine if testing has adversely affected the target.

The CAN frames being sent can share the same driver and channel as the fuzzing, or use a different driver/channel.

9.10.1. Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet reception (default 500,000)

Id

CAN Frame ID field in hex to expect

SignalEndian

Endianness of signal (if needed). Defaults to *little*. Options are *little* or *big*.

SignalOffset

Bit offset to signal start

SignalSize

Length of signal field in bits

SignalType

Data type of signal. This is used to convert signal data into a usable type in the python threshold expression.

Signal Types	
Int	SignalSize can be any value 32 or lower

Signal Types	
Float	SignalSize must be 32 or 64
Long	SignalSize between 32 and 64
String	Interpreted as UTF-8
Binary	SignalSize must be factor of 8

Expression

Stateless threshold expression used to determine if the signal value is within spec. Expression results are cached for speed, and must not be stateful. The threshold expression is specified as a Python 2 expression. The provided expression must evaluate to a bool true/false. Python expressions are single line code statements. The following is an example of a code snippet that verifies a signal value is between 10 and 20 inclusive: `signal >= 10 and signal <= 20`

The Python expressions have access to the following local variables:

Local Variables	Description
id	CAN Frame ID as a python int
logger	Logging interface output stored in debug.log in run folder and test output. logger.Debug(msg)
signal	Signal value decoded based on provided type

Optional:

No optional parameters.

9.10.2. Examples

Example 17. Expect frame every half second

This parameter example is from a setup that uses the CAN Threshold Monitor to fault if a signal in the CAN frame with id 0x7DE is not between 10 and 100.

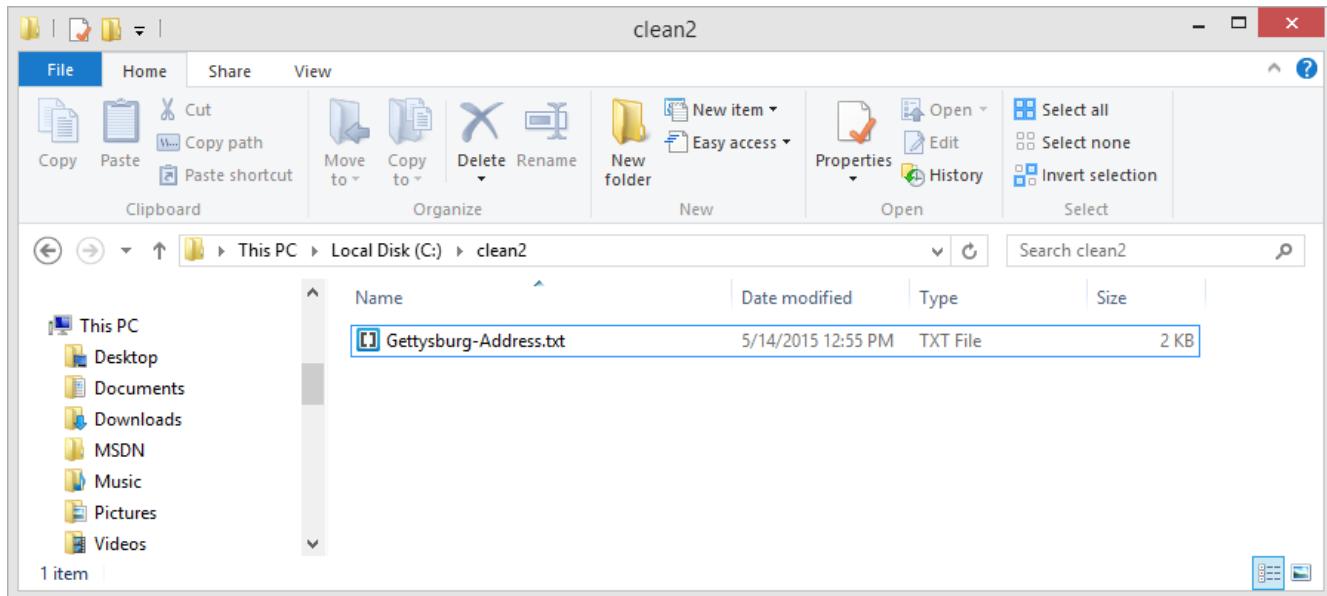
Parameter	Value
CanDriver	Vector XL
CanChannel	1
CanBitrate	(default)
Id	0x07DE
SignalEndian	(default)
SignalOffset	0
SignalSize	8
SignalType	int
Expression	signal > 10 and signal < 100

9.11. CleanupFolder Monitor

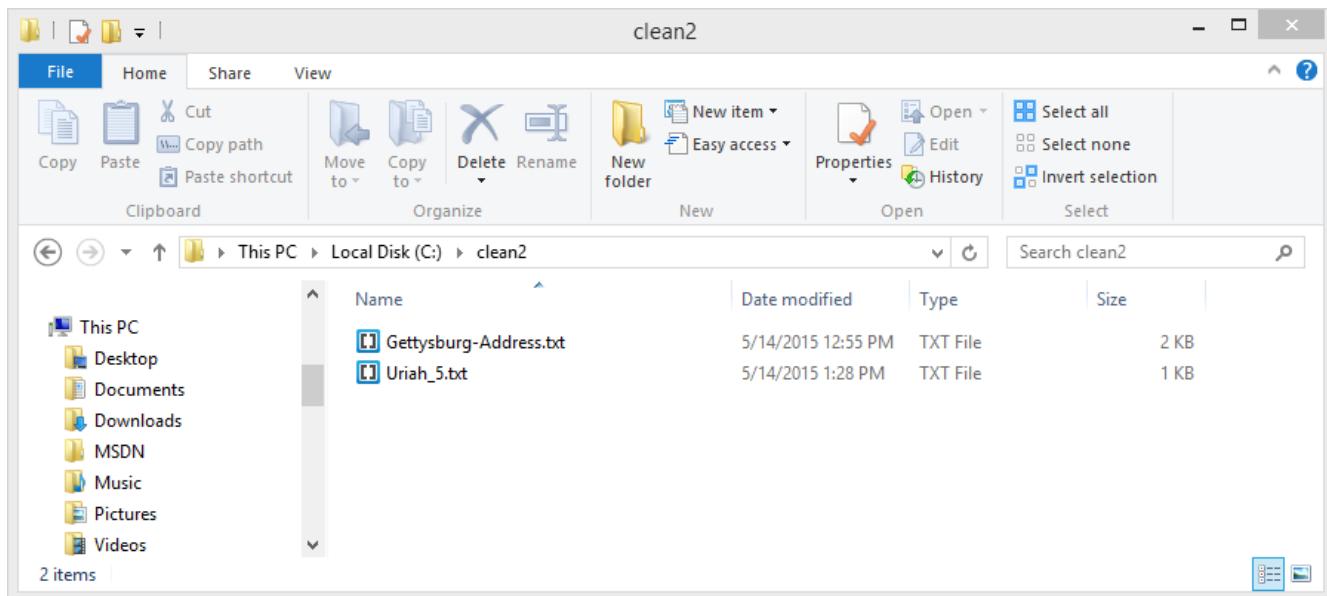
Monitor Category: Automation

The *CleanupFolder* monitor provides automated cleanup of files created during a fuzzing session. This monitor purges files produced during an iteration before the start of the following iteration. *CleanupFolder* acts upon a specified directory, and does not affect any files or directories that predate the fuzzing session.

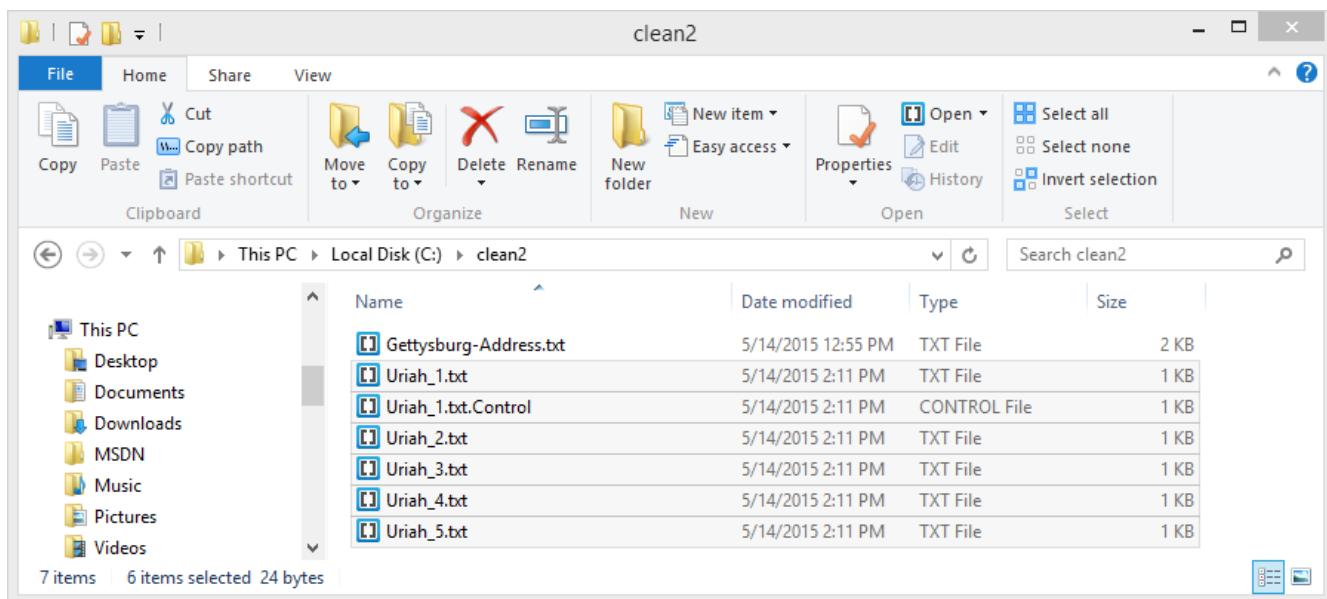
1. The directory, `clean2`, initially contains a single file.



2. Before each iteration, the *CleanupFolder* monitor deletes from the specified folder all files that were generated in the previous fuzzing iteration.
3. During each iteration, the File publisher creates a fuzzed file.
4. The following image shows the output from a file fuzzing session that uses *CleanupFolder* on the `clean2` directory. The session duration is 5 iterations.



5. The following image shows the `clean2` directory with output from a file fuzzing session that does *not* use `CleanupFolder`, but is otherwise identical to the pit used for the previous diagram. Files generated in the fuzzing session are highlighted.



The `FilePerIteration` publisher, that generates a new filename for each fuzzed file, produced the fuzzed output files for both of the previous images.

9.11.1. Parameters

Required:

Folder

Folder to clean.

Optional:

None.

9.11.2. Examples

Example 18. Remove contents of a folder

This parameter example is from a setup that uses the CleanupFolder monitor to remove a fuzzed file created by the File publisher.

Parameter	Value
Folder	C:\clean2

9.12. CleanupRegistry Monitor (Windows)

Monitor Category: Automation

The *CleanupRegistry* monitor provides automated cleanup of Windows registry entries created during a fuzzing session. Cleanup occurs before every iteration.

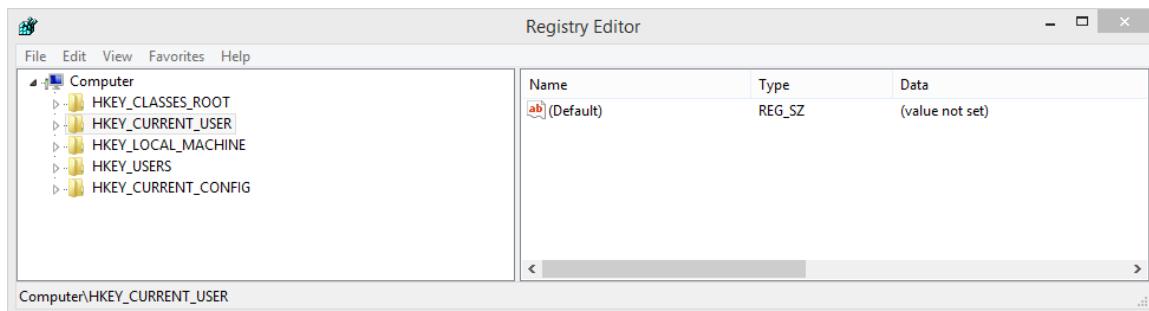
This monitor uses a specified registry key (a parent key) as a reference point and deletes all child keys or descendants of the parent key. All child keys at all levels beneath the parent key are removed. Values attached to child keys are removed as well.

Optionally, *CleanupRegistry* can include or exclude the parent key and values attached to the parent key in its purge.

The Windows Registry is a hierarchical database that stores information about the computer system, configuration, applications, current user settings, and performance data. The contents and use of the registry has evolved since its inception. Originally, the registry stored installation and initialization settings for applications. On current versions of Windows, the registry contains information for the current user, and much more, such as a list of external serial ports that the machine has allocated.

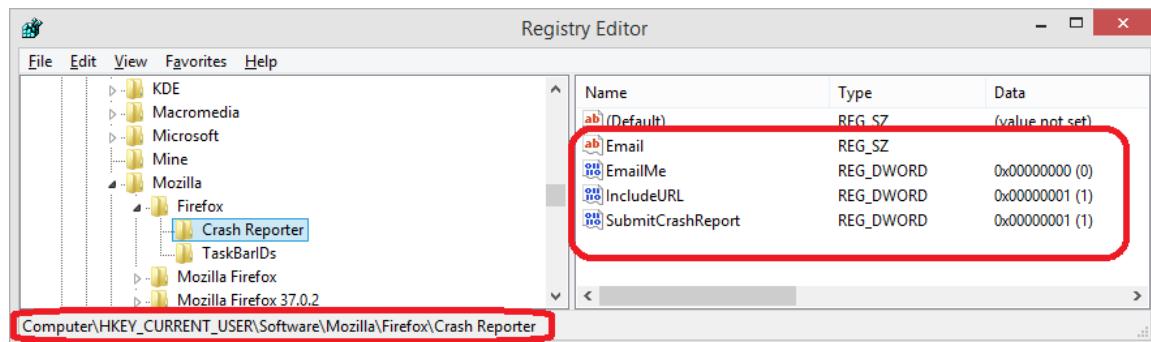
One common use of the registry is to store a list of the most recently used files used in an application. This list is typically displayed to a user to improve the experience of selecting a file for the application to open.

The Windows registry consists of five hierarchies called hives, as shown in the following illustration. Each hive consists of keys and values.



- A key is a container that can contain other keys and values. A key has a name, a type, and optionally, a data value. The following illustration shows a selected key. The complete name for this key is reported in the lower left corner.
- A value is an item associated with a key that has a type, a data value, and optionally, a name. In the following illustration, the named values defined for the selected key are listed in the right side of the window.





Each hive has one root key that is the entry point to the hive. These root keys are the prefixes described for the *CleanupRegistry Key* parameter. The hives used most often are **HKCU** and **HKLM**, but keys from other hives are accessible as well.

Locating a key in the registry is similar to locating a folder on disk. Start with the root key and specify the sequence of keys you need to reach the desired key. Separate keys with the back slash character "\", just like the folders in a path.

A root key, such as **HKCU**, has child keys; that is, the keys that it contains. A non-root key, such as **HKCU\Software**, has a parent key (**HKCU**) and, optionally, child keys. Note that **HKCU** is the parent key of **Software**. The key **HKCU\Software\Mozilla\FireFox** also has two child keys or children (**Crash Reporter** and **TaskBarIDs**).

For more information, see [Windows Registry](#). Note the rich bibliography, which can serve as a foundation for additional reading. Also, the Microsoft Developer Network (MSDN) has several articles on the registry.

In *CleanupRegistry*, you do not have control over individual values. You simply identify the parent key, and specify whether you need to zap the parent key; then let the monitor clean up around that registry key after each iteration.



Sometimes, applications keep lists of recently used files in the registry, typically in the **HKCU** or **HKLM** hives. If you receive an error about not being able to create a key in the registry, you might need to clear the children of a specified key.

9.12.1. Parameters

Required:

Key

Registry key to remove. The following **key prefixes** are used:

HKCU - Current user

HKCC - Current configuration

HKLM - Local machine

HKPD - Performance data

HKU - Users

Optional:

ChildrenOnly

If true, omits the parent key from the purge, thereby deleting all descendants (sub-keys) of the specified key.

If false, the parent key and all descendants are deleted. Defaults to false.

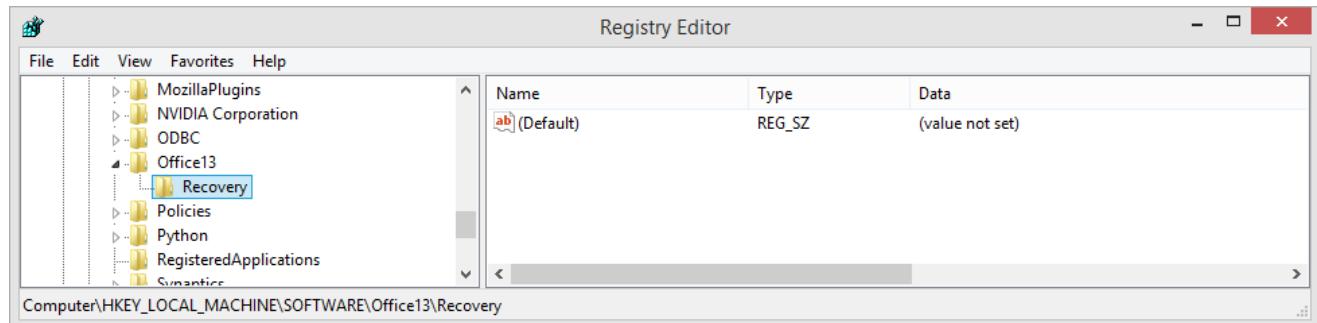
9.12.2. Examples

Cleanup for Office

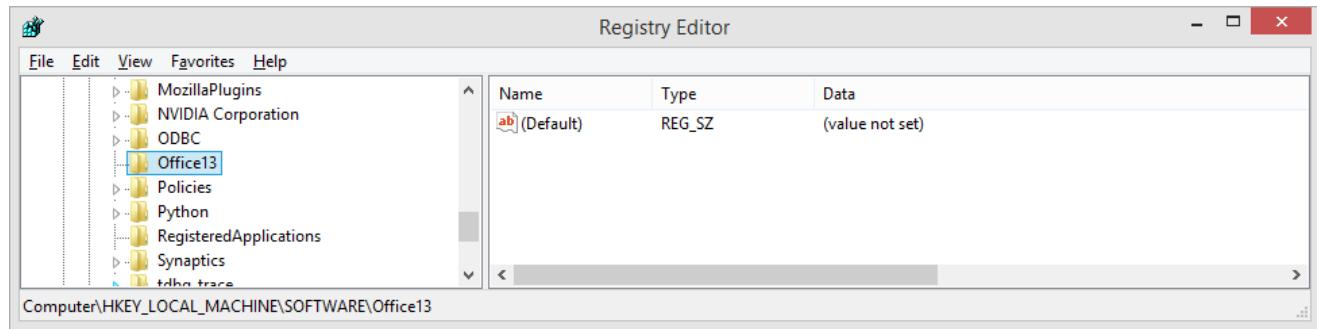
This parameter example is from a setup that uses the CleanupRegistry monitor to remove a specified key from the Windows Registry.

Parameter	Value
Key	HKLM\SOFTWARE\Office13\Recovery

The following image shows the portion of the registry that contains the key to delete.



This monitor deleted the key. The following image shows the same area of the registry after fuzzing and using the CleanupRegistry monitor.



9.13. CrashReporter Monitor (OS X)

Monitor categories: Data collection, Fault detection

The *CrashReporter* monitor collects and logs core dump information from crashes detected by the OS X System Crash Reporter. Use *CrashReporter* when crashes can occur and you cannot use [CrashWrangler](#).

The monitoring scope can focus on a single executable, specified by setting the **ProcessName** parameter. Or, the monitoring scope can include all processes. By default, all processes are monitored.

Before fuzzing, make sure to disable the Apple crash report dialog window. When the fuzzing session ends, reenable the crash report dialog window. Commands to disable and enable the crash report dialog window follow:

- Disable the crash report dialog:



```
defaults write com.apple.CrashReporter DialogType none
```

- Enable the crash report dialog after fuzzing:

```
defaults write com.apple.CrashReporter DialogType crashreport
```

9.13.1. Parameters

Required:

None.

Optional:

ProcessName

Name of the process to watch (optional, defaults to all)

9.13.2. Examples

Example 19. Catch crashes from Movie Player

This parameter example is from a setup that monitors a movie player for OS X.

Parameter	Value
ProcessName	mplayer

9.14. CrashWrangler Monitor (OS X)

Monitor Categories: Automation, Data collection, Fault detection



The gdb debugger and `gdb` monitor are the preferred tools to detect crashes and to collect core files on OS X.

The *CrashWrangler* monitor launches a process attached to the CrashWrangler debugger and monitors the process for crashes and other faults. This monitor runs only on OS X systems. Use this monitor when gdb is not an option, such as when anti-debugging mechanisms are used on the test target.

CrashWrangler monitor detects crashes, exceptions, access violations, and faults. This monitor can generate faults for an application that exits early or that fails to exit.

After detecting a fault, the *CrashWrangler* monitor collects a stack trace, data from the device log files, and crash dumps. Additionally, the monitor logs exceptions, and updates fault bucket information. For bucketing, Peach uses the text from the fault to determine the major bucket level. The minor bucket level is not used. The risk evaluation provides the following levels: exploitable, not exploitable, or unknown.

This monitor uses Apple's Crash Wrangler tool that can be downloaded from the developer website. Crash Wrangler must be compiled on each machine it is used.



When using more than one instance of CrashWrangler in the same fuzzing session, assign unique names for the CwLockFile, CwLogFile, and CwPidFile files used with each instance of CrashWrangler. This practice will avoid contention issues involving these files.

9.14.1. Parameters

Required:

Executable

Command or application to launch. This parameter name is preferred over Command.

Command

Command to execute. Alias with Executable.



The Command parameter is supported, but is being deprecated. Instead, use the Executable parameter.

Optional:

Arguments

Command line arguments for the application that CrashWrangler launches, defaults to none.

CwLockFile

CrashWrangler Lock file, defaults to `cw.lock`.

CwLogFile

CrashWrangler Log file, defaults to `cw.log`.

CsPidFile

CrashWrangler PID file, defaults to `cw.pid`.

ExecHandler

Crash Wrangler execution handler program, defaults to `exc_handler`.

ExploitableReads

Are read a/v's considered exploitable? Defaults to false.

FaultOnEarlyExit

Trigger a fault if the process exits prematurely, defaults to false.

NoCpuKill

Disable process killing by CPU usage, Defaults to false.

RestartAfterFault

If "true", restarts the target when any monitor detects a fault. If "false", restarts the target only if the process exits or crashes.

This argument defaults to true.

RestartOnEachTest

Restarts the process for each iteration, defaults to false.

StartOnCall

Start the executable on a state model call, defaults to none.

UseDebugMalloc

Use the OS X Debug Malloc (slower), defaults to false.

WaitForExitOnCall

Wait for the process to exit on a state model call and fault if a timeout occurs, defaults to none.

WaitForExitTimeout

WaitForExitOnCall timeout value, expressed in milliseconds. -1 is infinite, defaults to 10000.

9.14.2. Examples

Example 20. Fuzzing Safari

This parameter example is from a setup that fuzzes the Safari browser on OS X.

Parameter	Value
Executable	<code>/Applications/Safari.app/Contents/MacOS/Safari</code>
Arguments	<code>fuzzed.bin</code>
UseDebugMalloc	<code>true</code>
ExploitableReads	<code>true</code>
ExecHandler	<code>./exc_handler</code>
StartOnCall	<code>ScoobySnacks</code>

9.15. Gdb Monitor (Linux, OS X)

Monitor Categories: Automation, Data collection, Fault detection

The *Gdb* monitor uses GDB to launch an executable. It then monitors the executing process for specific signals/exceptions. *Gdb* is the main debugger for Linux and OS X operating systems.

When a configured signal/exception is handled, *Gdb* collects and logs information about the crash including frame information, registers and backtrace.

Gdb supports bucketing of crashes and basic risk ranking that is based on the exploitability of the fault. Bucketing uses categories for major and minor issues. The exploitability evaluation is similar to the !exploitable debugging extension for .NET.

For GDB Server support see the [GdbServer](#) monitor.



The *Gdb* monitor requires a recent version of GDB.

9.15.1. Parameters

Required:

Executable

Executable to launch. This should be just the executable. Command line arguments can be provided using the optional *Arguments* parameter.

Optional:

Arguments

Command line arguments for the executable.

FaultOnEarlyExit

Trigger a fault if the process exits prematurely, defaults to `false`.

GdbPath

Path to `gdb`, defaults to `/usr/bin/gdb`.

HandleSignals

Signals to consider faults. Space separated list of signals/exceptions to handle as faults. Defaults to: SIGSEGV SIGFPE SIGABRT SIGILL SIGPIPE SIGBUS SIGSYS SIGXCPU SIGXFSZ EXC_BAD_ACCESS EXC_BAD_INSTRUCTION EXC_ARITHMETIC

NoCpuKill

Disable process killing when the CPU usage nears zero, defaults to `false`.

RestartAfterFault

If `true`, restarts the target when any monitor detects a fault. If `false`, restarts the target only if the process exits or crashes. This argument defaults to `true`.

RestartOnEachTest

Restarts the process for each iteration, defaults to `false`.

Script

Script file used to drive GDB and perform crash analysis. This script sets up GDB to run the target application, report back information such as child pid, and handle any signals. This script also performs crash analysis and bucketing. An example script is provided in the examples section.

The script is a Mushtache style template with the following available parameters:

executable

Maps to the Executable parameter

arguments

Maps to the Arguments parameter

exploitableScript

Default crash analysis and bucketing script

faultOnEarlyExit

Maps to the FaultOnEarlyExit parameter

gdbLog

Log file used to record information that will be logged with any fault

gdbCmd

This script file after being processed by Mushtache

gdbPath

Maps to the GdbPath parameter

gdbPid

Pid file generated by script at start of executable

gdbTempDir

Temporary directory to hold generated data

handleSignals

Maps to the HandleSignals parameter

noCpuKill

Maps to the NoCpuKill parameter

restartOnEachTest

Maps to the RestartOnEachTest parameter

restartAfterFault

Maps to the RestartAfterFault parameter

startOnCall

Maps to the StartOnCall parameter

waitForExitOnCall

Maps to the WaitForExitOnCall parameter

waitForExitTimeout

Maps to the WaitForExitTimeout parameter

StartOnCall

Start the executable on a state model call.

WaitForExitOnCall

Wait for the process to exit on a state model call and fault if a timeout is reached.

WaitForExitTimeout

WaitForExitOnCall timeout value, expressed in milliseconds. -1 is infinite, defaults to 10000.

9.15.2. Gdb Installation

Installing `gdb` on Linux consists of using the package manager to download and install the `gdb` debugger and the needed support files and libraries.

- For Ubuntu systems, run the following command install `gdb`:

```
sudo apt-get install gdb
```

- For CentOS and RedHat Enterprise Linux installations, run the following command to install `gdb`:

```
sudo yum install gdb
```

- For SUSE Enterprise Linux installations, run the following command to install `gdb`:

```
sudo zypper install gdb
```

You can also install `gdb` on OS X using a package manager. See the [Installing GDB on OS X Mavericks](#) article for more information.

9.15.3. Examples

Example 21. Script Example

This is an example `Script` for driving GDB.

The `Gdb` monitor expects specific output that the included exploit analysis to generate the fault title, risk and bucket hashes.

The following regular expressions are used to extract information from log. If replacing the crash analysis portion of the script make sure to output to allow the first three regexes to pass.

```
"^Hash: (\w+)\.(\w+)$" -- Major and Minor bucket
"^Exploitability Classification: (.*)$" -- Risk
"^Short description: (.*)$" -- Title
"^Other tags: (.*)$" -- If found, added to Title
```

Script:

```

define log_if_crash
if ($_thread != 0x00)
printf "Crash detected, running exploitable.\n"
set logging overwrite on
set logging redirect on
set logging on {{gdbLog}}          ②
exploitable -v                   ③
printf "\n--- Info Frame ---\n\n"
info frame
printf "\n--- Info Registers ---\n\n"
info registers
printf "\n--- Backtrace ---\n\n"
thread apply all bt full
set logging off
end
end

handle all nostop noprint
handle {{handleSignals}} stop print

file {{executable}}
set args {{arguments}}
source {{exploitableScript}}


python
def on_start(evt):
    import tempfile, os
    h,tmp = tempfile.mkstemp()
    os.close(h)
    with open(tmp, 'w') as f:
        f.write(str(gdb.inferiors()[0].pid))
    os.renames(tmp, '{{gdbPid}}')      ①
    gdb.events.cont.disconnect(on_start)
gdb.events.cont.connect(on_start)
end

printf "starting inferior: '{{executable}} {{arguments}}'\n"

run
log_if_crash
quit

```

① Must generate *gdbPid* when target has been started

② Must generate *gdbLog* when a fault has been handled

③ Generates output compatible with regexes

Example 22. Base Usage Example+

This parameter example is from a typical setup where a state model call triggers launching of the executable.

Parameter	Value
Executable	/usr/bin/curl
Arguments	http://localhost
StartOnCall	ScoobySnacks

9.16. GdbServer Monitor (Linux, OS X)

Monitor Categories: Automation, Data collection, Fault detection

The *GdbServer* monitor uses GDB to access a remote debugger that implements the gdb server protocol. This can be `gdbserver` or any other implementation. This monitor also supports the multi mode of the extended `gdbserver` protocol via the *RemoteExecutable* parameter.

When a configured signal/exception is handled, *GdbServer* collects and logs information about the crash including frame information, registers and backtrace.

GdbServer supports bucketing of crashes and basic risk ranking that is based on the exploitability of the fault. Bucketing uses categories for major and minor issues. The exploitability evaluation is similar to the !exploitable debugging extension for .NET.

For normal GDB support see the [Gdb](#) monitor.



The *GdbServer* monitor requires a recent version of GDB.

9.16.1. Parameters

Required:

Target

GDB target command arguments. Example: `remote 192.168.1.2:6000`.

When possible use the extended-remote version: `extended-remote IP:PORT`.

LocalExecutable

Local copy of executable remote target is running.

Optional:

RemoteExecutable

Enables multi-process capabilities in the extended gdb server protocol. Provides the remote executable name to load and run. This is the remote exec file-name.

FaultOnEarlyExit

Trigger a fault if the process exits prematurely, defaults to `false`.

GdbPath

Path to `gdb`, defaults to `/usr/bin/gdb`.

HandleSignals

Signals to consider faults. Space separated list of signals/exceptions to handle as faults. Defaults to: `SIGSEGV SIGFPE SIGABRT SIGILL SIGPIPE SIGBUS SIGSYS SIGXCPU SIGXFSZ EXC_BAD_ACCESS`

EXC_BAD_INSTRUCTION EXC_ARITHMETIC

NoCpuKill

Disable process killing when the CPU usage nears zero, defaults to `false`.

RestartAfterFault

If `true`, restarts the debugger when any monitor detects a fault. If `false`, restarts the debugger only if the process exits or crashes. This argument defaults to `true`.

RestartOnEachTest

Restarts the process for each iteration, defaults to `false`.

Script

Script file used to drive GDB and perform crash analysis. This script sets up GDB to run the target application, report back information such as child pid, and handle any signals. This script also performs crash analysis and bucketing. An example script is provided in the examples section.

The script is a Mustache style template with the following available parameters:

exploitableScript

Default crash analysis and bucketing script

faultOnEarlyExit

Maps to the FaultOnEarlyExit parameter

gdbLog

Log file used to record information that will be logged with any fault

gdbCmd

This script file after being processed by Mustache

gdbPath

Maps to the GdbPath parameter

gdbPid

Pid file generated by script at start of executable

gdbTempDir

Temporary directory to hold generated data

handleSignals

Maps to the HandleSignals parameter

localExecutable

Maps to the LocalExecutableParameter

noCpuKill

Maps to the NoCpuKill parameter

remoteExecutable

Maps to the RemoteExecutable parameter

restartOnEachTest

Maps to the RestartOnEachTest parameter

restartAfterFault

Maps to the RestartAfterFault parameter

startOnCall

Maps to the StartOnCall parameter

target

Maps to the Target parameter

waitForExitOnCall

Maps to the WaitForExitOnCall parameter

waitForExitTimeout

Maps to the WaitForExitTimeout parameter

StartOnCall

Start the executable on a state model call.

WaitForExitOnCall

Wait for the process to exit on a state model call and fault if a timeout is reached.

WaitForExitTimeout

WaitForExitOnCall timeout value, expressed in milliseconds. -1 is infinite, defaults to 10000.

9.16.2. Gdb Installation

Installing `gdb` on Linux consists of using the package manager to download and install the `gdb` debugger and the needed support files and libraries.

- For Ubuntu systems, run the following command install `gdb`:

```
sudo apt-get install gdb
```

- For CentOS and RedHat Enterprise Linux installations, run the following command to install `gdb`:

```
sudo yum install gdb
```

- For SUSE Enterprise Linux installations, run the following command to install `gdb`:

```
sudo zypper install gdb
```

You can also install `gdb` on OS X using a package manager. See the [Installing GDB on OS X Mavericks](#) article for more information.

9.16.3. Examples

Example 23. Script Example

This is an example `Script` for driving GDB.

The `GdbServer` monitor expects specific output that the included exploit analysis to generate the fault title, risk and bucket hashes.

The following regular expressions are used to extract information from log. If replacing the crash analysis portion of the script make sure to output to allow the first three regexes to pass.

```
"^Hash: (\w+)\.(\w+)$" -- Major and Minor bucket
"^Exploitability Classification: (.*)$" -- Risk
"^Short description: (.*)$" -- Title
"^Other tags: (.*)$" -- If found, added to Title
```

Script:

```
define log_if_crash
if ($_thread != 0x00)
printf "Crash detected, running exploitable.\n"
set logging overwrite on
set logging redirect on
set logging on {{gdbLog}}          ②
exploitable -v                    ④
printf "\n--- Info Frame ---\n\n"
info frame
printf "\n--- Info Registers ---\n\n"
info registers
printf "\n--- Backtrace ---\n\n"
thread apply all bt full
set logging off
end
```

```

end

handle all nostop noprint
handle {{handleSignals}} stop print

file {{executable}}
source {{exploitableScript}}


python

import sys

def on_start(evt):
    import tempfile, os
    h,tmp = tempfile.mkstemp()
    os.close(h)
    with open(tmp, 'w') as f:
        f.write(str(gdb.inferiors()[0].pid))
    os.renames(tmp, '{{gdbPid}}')          ①
    gdb.events.cont.disconnect(on_start)

gdb.events.cont.connect(on_start)

print("starting inferior: '{{target}} {{remoteExecutable}}'")

try:
    if len('{{remoteExecutable}}') > 1:
        print('starting in extended-remote, multi-process mode')
        gdb.execute('set remote exec-file {{remoteExecutable}}')
        gdb.execute('target {{target}}')
        gdb.execute('run')
    else:
        gdb.execute('target {{target}}')
        gdb.execute('continue')

except:
    e = sys.exc_info()[1]
    print('Exception starting target: ' + str(e))
    import tempfile, os
    h,tmp = tempfile.mkstemp()
    os.close(h)
    with open(tmp, 'w') as f:
        f.write(str(e))
    os.renames(tmp, '{{gdbConnectError}}')  ③
    gdb.execute('quit')

end

```

```
log_if_crash  
quit
```

- ① Must generate *gdbPid* when target has been started
- ② Must generate *gdbLog* when a fault has been handled
- ③ On remote connection error, output the error message to *gdbConnectError*
- ④ Generates output compatible with regexes

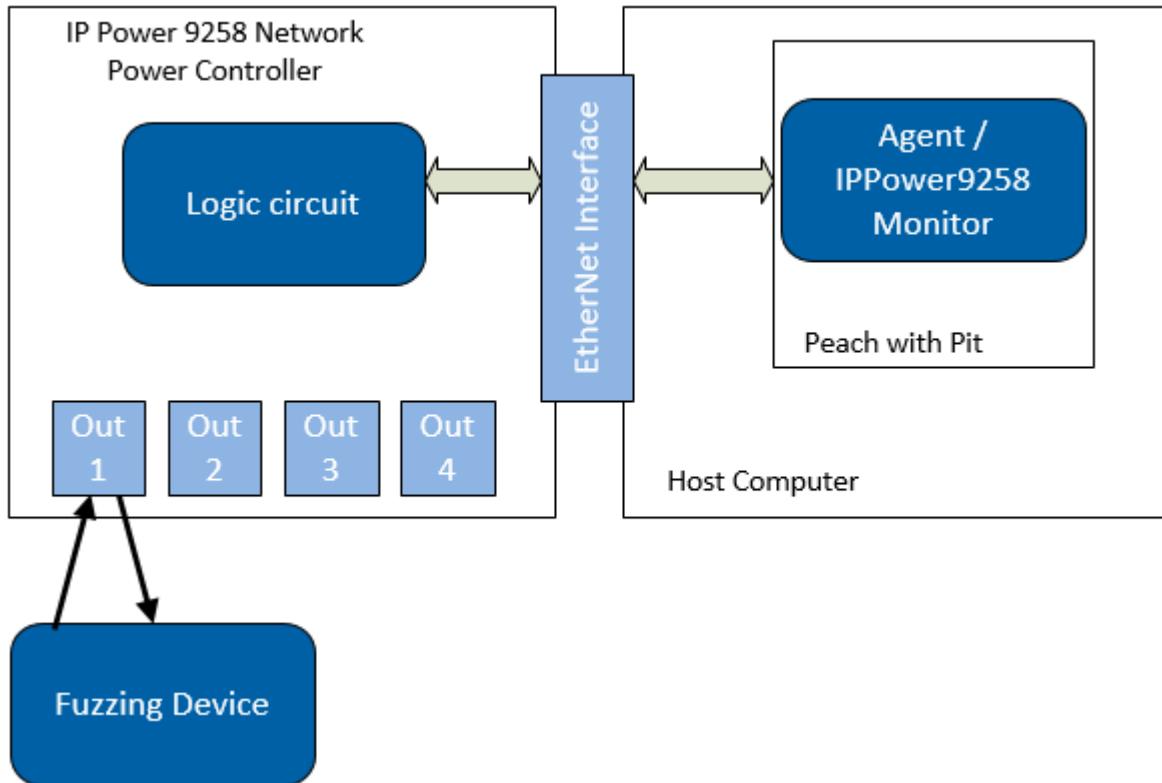
9.17. IpPower9258 Monitor

Monitor Category: Automation

The *IpPower9258* monitor controls an IP Power 9258 Network Power Controller (IP9258). The IP9258 consists of four three-prong outlets that support electrical loads of 6 amps at 110 or 240 VAC. Communications between Peach and the IP9258 uses Ethernet cabling. This monitor allows devices plugged into the IP Power 9258 switch to be powered on/off during fuzzing.

Each *IpPower9258* monitor switches an individual outlet. Use one monitor per outlet to control multiple outlets concurrently. The monitor can toggle power on an outlet at the following points in time:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- During the detection of a fault
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model





Peach also supports power distribution units that can be controlled using SNMPv1. For APC PDUs use the [APC Power Monitor](#), and for others try the [SNMP Power Monitor](#). To control power to a device by wiring through a relay, Peach provides the [CanaKit 4-Port USB Relay Monitor](#).

9.17.1. Parameters

Required:

Host

Host or IP address (can include HTTP interface port e.g. :8080)

Port

Port/Outlet to reset (1, 2, 3, 4)

User

Username to be used when connecting to the IP Power 9258 device.

Password

Password to be used when connecting to the IP Power 9258 device.

Optional:

StartOnCall

Toggle power when the specified call is received from the state model. This value is used only when the *When* parameter is set to [OnCall](#).

PowerOnOffPause

Pause in milliseconds between power off/power on, default is [500](#).

PowerOffOnEnd

Power off when the fuzzing session completes, default is [false](#).

When

Specify one of the following values to determine when a port should be toggled:

"When" Setting	Description
DetectFault	Toggle power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Toggle power when the fuzzing session starts. This occurs once per session.
OnEnd	Toggle power when the fuzzing session stops. This occurs once per session.

"When" Setting	Description
OnIterationStart	Toggle power at the start of each iteration.
OnIterationEnd	Toggle power at the end of each iteration.
OnFault	Toggle power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Toggle power at the start of an iteration that immediately follows a fault detection.
OnCall	Toggle power when the call specified by the <i>StartOnCall</i> parameter is received from the state model.

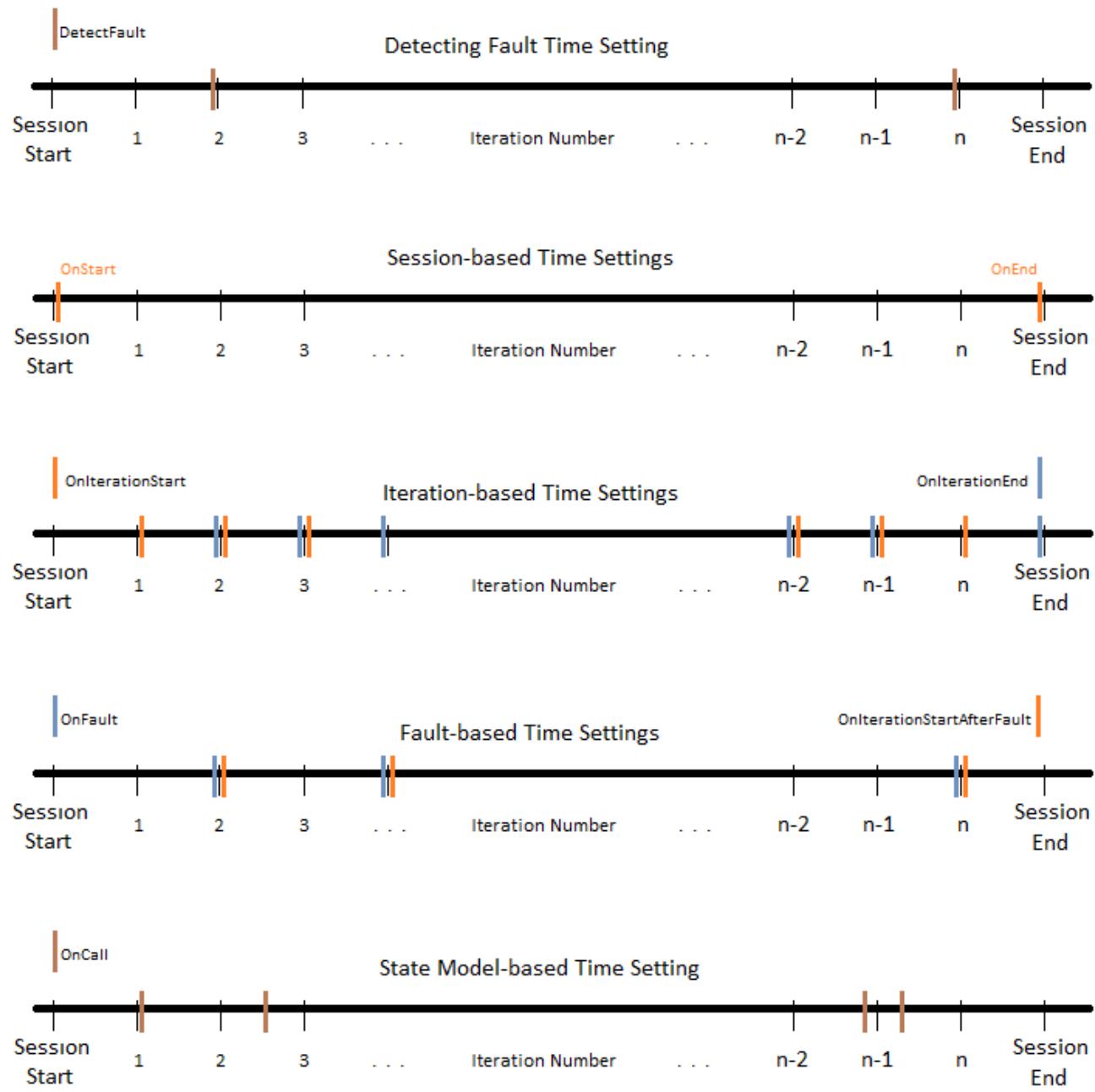


Figure 5. When Choices for Performing an Action

9.17.2. Examples

Example 24. Catch crashes from Movie Player

This parameter example is from a setup that controls port 1 of a IP Power 9258 Network Power Controller. After a fault is detected, the monitor toggles the power on port 1.

Parameter	Value
Host	192.168.1.1:8080
Port	1
User	peach
Password	power

9.18. LinuxCoreFile Monitor (Linux)

Monitor Categories: Data collection, Fault detection

The `gdb` debugger and [Gdb Monitor](#) are the preferred tools to detect crashes and to collect core files on Linux.



This monitor runs only on Linux systems. Use this monitor when `gdb` is not an option, such as when anti-debugging mechanisms are used on the test target.

The *LinuxCoreFile* monitor detects when a process crashes and collects the resulting core file. The monitoring scope includes all processes by default, but can focus on a single executable.

This monitor runs for the entire fuzzing session and uses the Linux crash recording facility. When a crash occurs, the *LinuxCoreFile* monitor pulls the logging information and available core files, and sets the bucketing information. The major bucket is based on the name of the crashed executable, and the minor bucket is a constant value based on the string "CORE".

At the end of the session, *LinuxCoreFile* restores the saved state and removes the logging folder.

Because the *LinuxCoreFile* monitor registers a custom core file handler with the Linux kernel, only one instance of the monitor is allowed to run on a host at any given time.

Setup Requirements

This monitor has the following setup requirements:

- *LinuxCoreFile* requires root or equivalent privileges to run. The *LinuxCoreFile* monitor registers a script with the kernel to catch core dumps of faulting processes.
- Core files must be enabled. If the maximum core file size in the system is zero, no core files are created. You can enable core file generation using the following process:
 1. Find the current core file hard and soft limits in use by the operating system. The hard limit is an absolute maximum that, once set, cannot be increased during a session. The soft limit is the current maximum file size that you can adjust up to the value of the hard limit. If the hard limit or the soft limit is set to zero, core files are disabled.
 - You can display the core file hard limit using the following command:

```
ulimit -Hc
```

- You can display the current core file soft limit using the following command:

```
ulimit -Sc
```

- Set the core file hard limit using the following command. You can specify **unlimited** or a numeric value that represents the number of 512-byte blocks to allow in a core file:

```
ulimit -Hc unlimited
```

- Set the core file soft limit using the following command. You can specify any value less than or equal to the hard limit.

```
ulimit -Sc unlimited
```

The hard and soft limits can be added to **/etc/sysctl.conf**. Then, whenever this file loads, appropriate core file limits are specified.

- gdb** must be installed to analyze the resulting core files. For information on installing **gdb**, see the [Gdb Monitor](#).

9.18.1. Parameters

Required:

None.

Optional:

Executable

Target executable process, used to filter crashes, defaults to all processes.

LogFolder

Folder with log files, defaults to **/var/peachcrash**.

9.18.2. Examples

Example 25. Catch crashes from Movie Player

This parameter example is from a setup that monitors a movie player in Linux.

Parameter	Value
Executable	mplayer

9.19. Memory Monitor

Monitor Category: Data collection, Fault detection

The *Memory* monitor provides two modes of operation:

1. Data collection

When the *MemoryLimit* is `0`, the monitor will collect memory metrics at the end of each iteration. If a fault is triggered by another monitor, the collected memory metrics will be stored as `Usage.txt` in the fault's data bundle.

2. Fault detection

When the *MemoryLimit* is `> 0`, the monitor will initiate a fault when the memory usage for the process being monitored exceeds the specified limit.

The reported metrics for this monitor include the following items:

- **Private memory size** - Number of bytes allocated for the process. An approximate number of bytes a process is using.
- **Working set memory** - Total physical memory used by the process, consisting of in-memory private bytes plus memory-mapped files, such as DLLs.
- **Peak working set** - Largest working set used by the process.
- **Virtual memory size** - Total address space occupied by the entire process, including the working set plus paged private bytes and the standby list.



This monitor requires that you set a memory limit and identify a process to monitor, specifying the process by name or by process id.

9.19.1. Parameters

Required:

Either one of *Pid* or *ProcessName* is required. It is an error to specify both.

Pid

Process ID to monitor.

ProcessName

Name of the process to monitor.

Optional:

MemoryLimit

A value specified in bytes.

When `0` is specified, enable data collection mode, which causes memory metrics to be collected at the end of every iteration.

When a value greater than `0` is specified, enable fault detection mode, which causes a fault to occur if the memory usage of the monitored process exceeds the specified limit.

Defaults to `0`.

StopOnFault

Stop fuzzing if a fault is triggered, defaults to `false`.

9.19.2. Examples

Example 26. Monitor memory usage of Notepad

This parameter example is from a setup that monitors memory usage of Notepad. It is configured to generate a fault if memory usage exceeds 10MB.

Parameter	Value
MemoryLimit	<code>10000000</code>
ProcessName	<code>Notepad</code>

9.20. NetworkCapture Monitor

Monitor Categories: Data collection, Fault detection

The *NetworkCapture* monitor performs network captures during the fuzzing iteration. When a packet arrives, this monitor writes the content into a file, increments the received packet count, and waits for the next packet to arrive. If a filter is used, the captured packets and associated packet count are for packets that pass the filtering criteria.

The captured data begins afresh for each iteration. If a fault occurs, the captured data is logged as a `.pcap` file and returned with the fault data. The `.pcap` file is compatible with Wireshark and `tcpdump`. A sample capture follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.146.128	10.71.0.126	QUAKEWC	59	Client to Server Connectionless[Malformed Packet]
2	0.004981	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Connectionless[Malformed Packet]
3	0.025181	192.168.146.128	10.71.0.126	QUAKEWC	139	Client to Server Connectionless[Malformed Packet]
4	0.027660	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Connectionless[Malformed Packet]
5	0.058870	192.168.146.128	10.71.0.126	QUAKEWC	69	Client to Server Game
6	0.085727	192.168.146.128	10.71.0.126	QUAKEWC	64	Client to Server Game
7	0.087455	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Game
8	0.118592	192.168.146.128	10.71.0.126	QUAKEWC	66	Client to Server Game
9	0.120024	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Game
10	0.149570	192.168.146.128	10.71.0.126	QUAKEWC	71	Client to Server Game
11	0.151014	10.71.0.126	192.168.146.128	QUAKEWC	323	Server to Client Game
12	0.188515	192.168.146.128	10.71.0.126	QUAKEWC	78	Client to Server Game
13	0.215186	192.168.146.128	10.71.0.126	QUAKEWC	63	Client to Server Game
14	0.217020	10.71.0.126	192.168.146.128	QUAKEWC	787	Server to Client Game
15	0.247285	192.168.146.128	10.71.0.126	QUAKEWC	80	Client to Server Game
16	0.277527	192.168.146.128	10.71.0.126	QUAKEWC	65	Client to Server Game
17	0.309523	192.168.146.128	10.71.0.126	QUAKEWC	66	Client to Server Game

Frame 1: 59 bytes on wire (472 bits), 59 bytes captured (472 bits)
Ethernet II, Src: Vmware_33:39:7d (00:0c:29:33:39:7d), Dst: Vmware_eb:aa:b1 (00:50:56:eb:aa:b1)
Internet Protocol Version 4, Src: 192.168.146.128 (192.168.146.128), Dst: 10.71.0.126 (10.71.0.126)
User Datagram Protocol, Src Port: 27001 (27001), Dst Port: 27500 (27500)
Quakeworld Network Protocol
[Malformed Packet: QUAKEWORLD]

```
0000 00 50 56 eb aa b1 00 0c 29 33 39 7d 08 00 45 00 .PV..... )39}..E.  
0010 00 2d 26 2b 00 00 80 11 b6 a7 c0 a8 92 80 0a 47 .-&+.... .....G  
0020 00 7e 69 79 6b 6c 00 19 45 7e ff ff ff ff 67 65 .~iyk1.. E-....ge  
0030 74 63 68 61 6c 6c 65 6e 67 65 0a tchallen ge.
```

Peach supports multiple *NetworkCapture* monitors in the same pit, as well as simple and compound packet filters in a single monitor. A compound packet filter consists of more than one packet filter joined by AND or OR.

A packet filter is a boolean value applied to a packet. If the result of the operation is true, the packet is accepted. If the result is false, the packet is ignored.

The main benefit of using filters is performance. Applying a filter to a packet produces a smaller dataset, resulting in faster processing time for each iteration.

Two strategies for developing effective filters:

1. Develop a filter that pinpoints the packets to process.
2. Develop a filter that prunes unwanted packets using negative logic.



Generally, using one monitor with a complex filter is better than using two monitors with simpler filters. Using one monitor places all the packets in a single file with an order of arrival. Using multiple monitors makes correlating arrival more difficult because each monitor has its own file to keep the processed packets.

For information on filter strings, see the following:

- [Berkely Packet Filters - The Basics](#)
- [tcpdump man page](#)

9.20.1. Parameters

Required:

Device

Device name or port where the packet capture takes place.

The Peach command line option `--showdevices` causes Peach to generate a list of all available network interfaces.



- On Windows platforms, the `ipconfig` utility lists the network devices.
- On Unix platforms, the `ifconfig` utility lists the network devices.

Optional:

Filter

PCAP style filter string. If present, the filter restricts capture to packets that match the filter string.

9.20.2. Examples

Example 27. Show the network devices from ipconfig

This example uses `ipconfig` from the Windows command line to list the available network devices on the system. The device names follow:

- Local Area Connection
- Wi-Fi
- Ethernet
- VMware Network Adapter VMnet1
- VMware Network Adapter VMnet8

Type the following command and press ENTER.

```
>ipconfig
```

The list of devices follows:

Windows IP Configuration

Wireless LAN adapter Local Area Connection:

```
Media State . . . . . : Media disconnected  
Connection-specific DNS Suffix . :
```

Wireless LAN adapter Wi-Fi:

```
Media State . . . . . : Media disconnected  
Connection-specific DNS Suffix . :
```

Ethernet adapter Ethernet:

```
Connection-specific DNS Suffix . :  
Link-local IPv6 Address . . . . . : fe80::d0ef:e30b:2d5c:12c5%3  
IPv4 Address. . . . . : 10.0.1.47  
Subnet Mask . . . . . : 255.255.255.0  
Default Gateway . . . . . : 10.0.1.1
```

Ethernet adapter VMware Network Adapter VMnet1:

```
Connection-specific DNS Suffix . :  
Link-local IPv6 Address . . . . . : fe80::7859:6e2f:6816:4c38%14  
IPv4 Address. . . . . : 192.168.47.1  
Subnet Mask . . . . . : 255.255.255.0  
Default Gateway . . . . . :
```

Ethernet adapter VMware Network Adapter VMnet8:

```
Connection-specific DNS Suffix . :  
Link-local IPv6 Address . . . . . : fe80::9185:c8de:2e72:1855%15  
IPv4 Address. . . . . : 192.168.127.1  
Subnet Mask . . . . . : 255.255.255.0  
Default Gateway . . . . . :
```

Example 28. Capture output to CrashableServer on port 4244

This parameter example is from a setup that captures all network traffic using the NetworkCapture monitor when a fault occurs. When running the fuzzing definition for this example, a crash occurs after a few iterations. When Peach logs the fault, a `.pcap` file is created inside the fault record.

NetworkCapture monitor settings

Parameter	Value
Device	Local Area Connection
Filter	port 4244

The setup for this example uses a second monitor, the [Windows Debugger](#) monitor, to launch the `CrashableServer` executable, normally located in the Peach directory. The following table lists the parameters for that monitor.

Windows Debugger monitor settings

Parameter	Value
Executable	<code>CrashableServer.exe</code>
Arguments	<code>127.0.0.1 4244</code>

9.21. PageHeap Monitor (Windows)

Monitor Category: Automation

The *PageHeap* monitor enables heap allocation monitoring for an executable through the Windows debugger. Peach sets and clears the parameters used for monitoring heap allocation at the beginning and end of the fuzzing session.



The *PageHeap* monitor requires heightened or administrative permissions to run.

9.21.1. Parameters

Required:

Executable

Executable name (no path)

Optional:

WinDbgPath

Path to the Windows Debugger installation. If not provided, Peach attempts to locate the directory.

9.21.2. Examples

Example 29. Enable for Notepad

This parameter example is from a setup that monitors heap allocation in Notepad. The example is a common setup in which both the *PageHeap* and the [Windows Debugger](#) monitors are configured for the fuzzing run.

PageHeap Monitor Parameters

Parameter	Value
Executable	notepad.exe

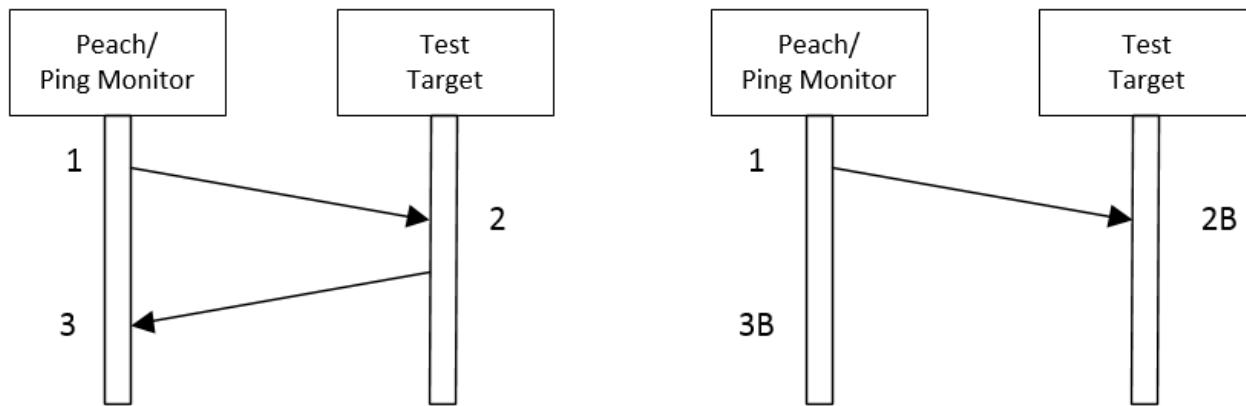
Windows Debugger Monitor Parameters

Parameter	Value
Executable	notepad.exe
Arguments	fuzzed.txt
StartOnCall	launchProgram

9.22. Ping Monitor

Monitor Category: Fault detection

The *Ping* monitor verifies whether a device is functioning by sending a packet to a target location, and waiting for a response from the device. *Ping* continues to monitor until either a *Timeout* occurs or a response from the target device reaches the *Ping* monitor. *Ping* runs at the end of each iteration.



In the first scenario (left), Ping successfully interacts with the test target.

1. At the end of each iteration, the Ping monitor sends a message to the test target.
2. In this case, the target receives the message and sends a response.
3. The monitor waits a specified time period for a response. In this case, the response within the time period, and gives the "OK" response. The Ping Monitor resets for the sequence to repeat at the end of the next iteration.

In the second scenario (right), the test target is non-responsive.

1. At the end of the iteration, the Ping monitor sends a message to the test target.
2. (2B) In this case, the target is non-responsive or no longer exists. No response is sent.
3. (3B) The monitor waits a specified time period for a response. In this case, a timeout occurs. The Ping monitor issues a fault and collects data around the fault. Ping resets for the next iteration.

Ping can validate that a target is still up or is waiting to restart. This is useful when fuzzing embedded devices that crash surreptitiously.

Additionally, by using the *FaultOnSuccess* parameter, *Ping* can help you to verify whether or not a new device at a specific address becomes available, or that a non-functioning device starts to function. For example, if you are fuzzing a device that programmatically turns other computers on, you can use *Ping* with *FaultOnSuccess* set to `true` to determine whether the computers are off (an uninteresting result) or on (an interesting result).

When running under Unix, the following restrictions apply:



- Root privileges are required
- *Data* parameter is limited to 72 bytes

9.22.1. Parameters

Required:

Host

Host name or IP address to ping.

Optional:

Data

Data to send in the ping packet payload. If this value is left blank, the OS will use default data to send.

FaultOnSuccess

Fault if ping is successful, defaults to `false`.

Timeout

Timeout value expressed in milliseconds, defaults to `1000`.

9.22.2. Examples

Example 30. Ping Host

This parameter example is from a setup that pings the Peach Fuzzer, LLC, website.

Parameter	Value
Host	www.peachfuzzer.com
Timeout	<code>10000</code>

9.23. PopupWatcher Monitor (Windows)

Monitor Categories: Automation, Fault detection

The *PopupWatcher* monitor closes pop-up windows based on a window title. *PopupWatcher* monitors the test target for a list of windows. When a window opens whose name is in the list, *PopupWatcher* closes the pop-up window, and if specified in the configuration, initiates a fault.

This monitor starts at the beginning of the session and runs to the session end.



Some applications re-use a pop-up window for many purposes. The window has one title, but the main area of the window can display several different messages depending on the context of the application. If you're interested in monitoring a pop-up window based on content rather than the window title, consider using the [ButtonClicker Monitor](#).

9.23.1. Parameters

Required:

WindowNames

One or more Window names separated by commas.



The comma-delimited list should not contain any white-space characters.

Optional:

Fault

Trigger a fault when a pop-up window is found, default `false`.

9.23.2. Examples

Example 31. Fault on Assert

This parameter example is from a setup that initiates a fault when a window titled `Assertion` is found.

Parameter	Value
WindowNames	<code>Assertion</code>
Fault	<code>true</code>

9.24. Process Monitor

Monitor Categories: Automation, Fault detection

The *Process* monitor controls a process during a fuzzing run. This monitor provides automation by controlling when the process starts, restarts and ends, and whether the process should be killed. This monitor also provides fault detection for early exit and failure to exit. Finally, the *Process* monitor provides data collection by copying messages from standard out and standard error.

The *Process* monitor provides the following functionality:

- Start a process at the session start.
- Start or restart a process on every iteration.
- Start a process in response to a call from the state model.
- Wait for a process to exit in response to a call from the state model.
- Restart a process when it exits.
- Terminates a process if the CPU usage is low.
- Logs a fault if a process exits early.
- Logs a fault if a process fails to exit.

The *Process* monitor initiates a fault when a process being monitored experiences the following conditions:

- Early exit
- Timeout while waiting for exit
- [Address Sanitizer](#) detection

9.24.1. Parameters

Required:

Executable

Executable to launch

Optional:

Arguments

Command line arguments

FaultOnEarlyExit

Trigger fault if process exits, defaults to `false`.

NoCpuKill

Disable process killing when the CPU usage nears zero, defaults to `false`.

RestartOnEachTest

Restart process on every iteration.

RestartAfterFault

If `true`, restarts the target when any monitor detects a fault. If `false`, restarts the target only if the process exits or crashes. This argument defaults to `true`.

StartOnCall

Start process when the specified call is received from the state model.

WaitForExitOnCall

Wait for process to exit when the specified call is received from the state model.

WaitForExitTimeout

Wait timeout value, expressed in milliseconds. Triggers a fault when the timeout period expires. Defaults to `10000`. Use `-1` for infinite, no timeout.

9.24.2. Examples

The following parameter examples are from different uses of the *Process* monitor. While not exhaustive, the examples provide a good base for comparing and contrasting parameter settings for the various uses of this monitor.

Example 32. Start a Process at the Start of a Session

The following parameter example is from a setup that starts a process at the beginning of a fuzzing session.

Parameter	Value
Executable	<code>notepad.exe</code>
Arguments	<code>fuzzed.txt</code>

Example 33. Start a Process at the Start of each Iteration

The following parameter example is from a setup that starts a process at the beginning of each test iteration.

Parameter	Value
Executable	notepad.exe
Arguments	fuzzed.txt
RestartOnEachTest	true

Example 34. Start a Process When Called from the State Model (Delayed Start)

The following parameter example is from a setup that starts a process when called from the state model, amid a test iteration.

Parameter	Value
Executable	notepad.exe
Arguments	fuzzed.txt
StartOnCall	LaunchProgram

Example 35. Suspend Fuzzing Until a Process Closes

The following parameter example is from a setup that interrupts fuzzing and waits for a process to close before resuming.

Parameter	Value
Executable	notepad.exe
Arguments	fuzzed.txt
WaitForExitOnCall	WaitForExit

9.25. ProcessKiller Monitor

Monitor Category: Automation

The *ProcessKiller* monitor kills (terminates) specified processes after each iteration.

9.25.1. Parameters

Required:

ProcessNames

Comma separated list of the processes to kill.



The process name is usually the executable filename without the extension (`.exe`). For example, `notepad.exe` will be `Notepad` or `notepad`. For Windows operating systems, the process name can be found by using the `tasklist.exe` command.

The comma-delimited list should not contain any white-space characters.

Optional:

None.

9.25.2. Examples

Example 36. Terminate `notepad` and `mspaint`

This parameter example is from a setup that terminates the `notepad` and the `paint` processes whenever they run.

Parameter	Value
ProcessNames	<code>notepad,mspaint</code>

9.26. Run Command

Monitor Categories: Automation, Data collection, Fault detection

The *RunCommand* monitor can be used to launch a command at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

If a fault occurs, the monitor captures and logs the console output from `stdout` and `stderr`. Additionally, this monitor can initiate a fault under the following conditions (in-order of evaluation):

1. An [Address Sanitizer](#) message appears in `stderr`.
2. The specified regular expression matches messages in `stdout` or `stderr`.
3. The command takes longer to finish than the specified timeout duration.
4. The command exits with a specified exit code.
5. The command exits with a nonzero exit code.

9.26.1. Parameters

Required:

Command

The command or application to launch.

Optional:

Arguments

Command line arguments

FaultOnNonZeroExit

When this value is set to `true`, generate a fault if the exit code is non-zero. The default value is `false`.

FaultOnExitCode

When this value is set to `true`, generate a fault if the exit code matches the specified `FaultExitCode` parameter. The default value is `false`.

FaultExitCode

When `FaultOnExitCode` is set to `true`, generate a fault if the specified exit code occurs. The default

value is 1.

StartOnCall

Launch the command when the monitor receives the specified call from the state machine. This value is used only when the *When* parameter is set to **OnCall**.

FaultOnRegex

If this value is specified, generate a fault if the specified regular expression matches the command output. The default value is unspecified.

Timeout

Maximum time period, in milliseconds, for the process to run. Generate a fault if the command runs longer than the specified value. This feature is disabled by specifying -1 for the time period. The default value is -1.

WorkingDirectory

Set the current working directory for the command launched by this monitor. The default value is the Peach current working directory. The current working directory for the command is valid until the command changes the directory or ends.

When

Specify one of the following values to determine when to launch the command:

"When" Setting	Description
OnStart	Run command when the fuzzing session starts. This occurs once per session.
OnEnd	Run command when the fuzzing session stops. This occurs once per session.
OnIterationStart	Run command at the start of each iteration.
OnIterationEnd	Run command at the end of each iteration.
OnFault	Run command when any monitor detects a fault.
OnIterationStartAfterFault	Run command at the start of an iteration that immediately follows a fault detection.
OnCall	Run command when the call specified by the <i>StartOnCall</i> parameter is received from the state model. This is the default setting.

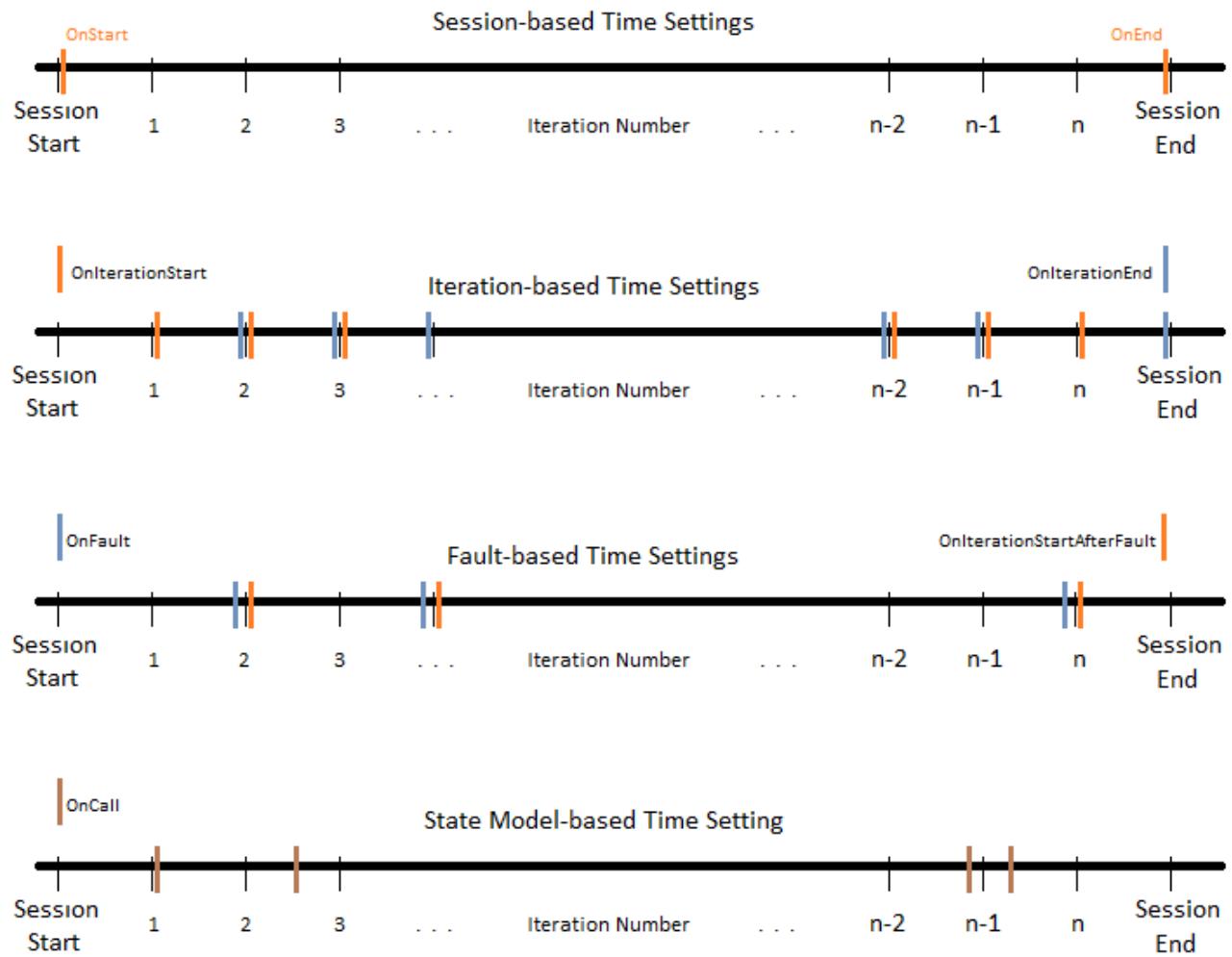


Figure 6. When Choices

9.26.2. Examples

Example 37. Using RunCommand for Fault Detection

This parameter example detects a fault by checking for any lines that begin with `ERROR_`.

Parameter	Value
Command	<code>python</code>
Arguments	<code>check_for_fault.py</code>
WorkingDirectory	<code>C:\MyScripts</code>
FaultOnRegex	<code>^ERROR_</code>
When	<code>OnIterationEnd</code>

Example 38. Using *RunCommand* for Data Collection

This parameter example captures `stderr` and `stdout` for data collection when another monitor detects a fault.

Parameter	Value
Command	<code>python</code>
Arguments	<code>collect_log.py</code>
WorkingDirectory	<code>C:\MyScripts</code>
When	<code>OnFault</code>

Example 39. Using *RunCommand* for Automation

This parameter example runs the `clear_state.py` python script on the next iteration start after a fault is detected. This can be used to get the target back into a working state so that fuzzing can continue.

Parameter	Value
Command	<code>python</code>
Arguments	<code>clear_state.py</code>
WorkingDirectory	<code>C:\MyScripts</code>
When	<code>OnIterationStartAfterFault</code>

9.27. SaveFile Monitor

Monitor Categories: Data collection

The *SaveFile* monitor saves a specified file as part of the logged data when a fault occurs. A copy of the file is placed in the log folder.

9.27.1. Parameters

Required:

Filename

File to save when a fault is detected by another monitor.

Optional:

None.

9.27.2. Examples

Example 40. Collect logs when a fault occurs

In this example, the [Process Monitor](#) is used to launch `nginx` as the target. When a fault is detected by this monitor, the *SaveFile* monitor is configured to collect logs from `nginx`. These logs will be available as part of the data collected for the fault.

Process (Launch `nginx`) Parameters

Parameter	Value
Executable	<code>/usr/sbin/nginx</code>

SaveFile (Save `error.log`) Parameters

Parameter	Value
Filename	<code>/var/log/nginx/error.log</code>

SaveFile (Save `access.log`) Parameters

Parameter	Value
Filename	<code>/var/log/nginx/access.log</code>

9.28. Serial Port Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *Serial Port* monitor can be used to perform data collection, fault detection, or automation, based on specified parameters.

The default usage of the *Serial Port* monitor is data collection. The data received via the serial port is logged when a fault occurs.

To perform fault detection, specify a regular expression using the *FaultRegex* parameter. When the regular expression matches, Peach generates a fault.

For automation tasks, use the *WaitForRegex* and *WaitWhen* parameters. These automation parameters cause Peach to wait for matching input before continuing. The *Serial Port* monitor can wait at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

Additionally, Peach supports multiple *Serial Port* monitors in a pit, allowing for more complex configurations. This can be used to monitor multiple serial ports. Multiple monitors may also be configured to use the same port, allowing for fault detection, automation, and/or data collection to occur on a single port.

9.28.1. Parameters

Required:

Port

The port to use (for example, `COM1` or `/dev/ttyS0`)

Optional:

BaudRate

The baud rate (only standard values are allowed). Defaults to `115200`.

DataBits

The data bits value. Defaults to `8`.

Parity

Specifies the parity bit. Defaults to `None`. Available options for this parameter are:

Even

Mark

None

Odd

Space

StopBits

Specifies the number of stop bits used. Defaults to **One**. Available options for this parameter are:

One

OnePointFive

Two

Handshake

Specifies the control protocol used in establishing a serial port communication. Defaults to **None**.

Available options for this parameter are:

None

RequestToSend

RequestToSendXOnXOff

XOnXOff

DtrEnable

Enables the Data Terminal Ready (DTR) signal during serial communication. Defaults to **false**.

RtsEnable

Enables the Request To Transmit (RTS) signal during serial communication. Defaults to **false**.

MaxBufferSize

Maximum amount of serial data to store in bytes. Defaults to **1048576**.

FaultRegex

Generate a fault when the specified regular expression matches received data. This causes the *Serial Port* monitor to be used for fault detection.

WaitRegex

Wait until the specified regular expression matches received data. This causes the *Serial Port* monitor to be used for automation.

WaitOnCall

Begin waiting for the regular expression specified in the *WaitRegex* parameter after the monitor receives the specified call from the state machine. This value is used only when the *WaitWhen* parameter is set to **OnCall**.

WaitWhen

Specify one of the following values to determine when to begin waiting for the regular expression specified in the *WaitRegex* parameter to match received data:

"WaitWhen" Setting	Description
OnStart	Waits when the fuzzing session starts. This occurs once per session. This is the default setting.
OnEnd	Waits when the fuzzing session stops. This occurs once per session.
OnIterationStart	Waits at the start of each iteration.
OnIterationEnd	Waits at the end of each iteration.
OnFault	Waits when any monitor detects a fault.
OnIterationStartAfterFault	Waits at the start of the iteration that immediately follows a fault detection.
OnCall	Waits upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model.

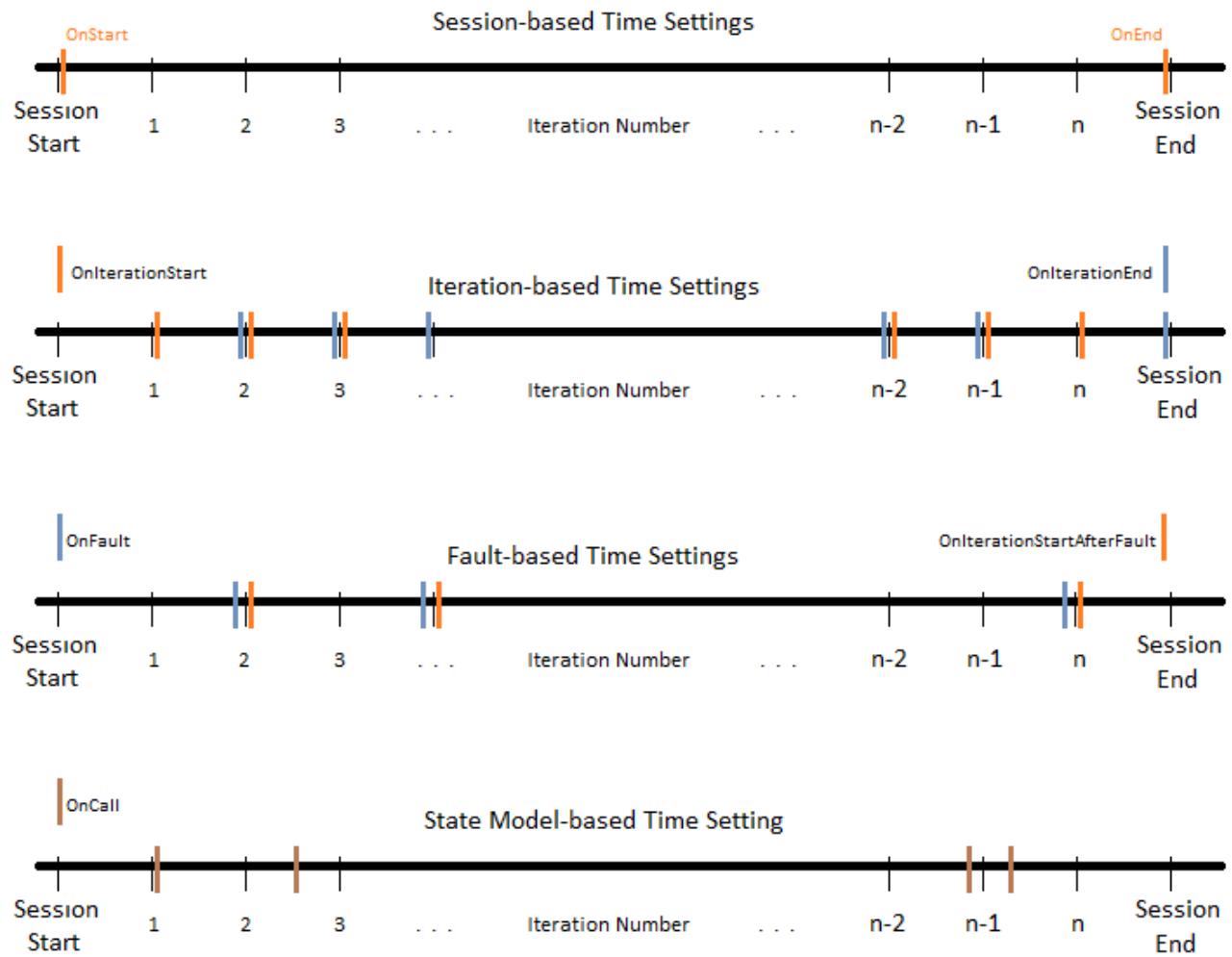


Figure 7. WaitWhen Choices

9.28.2. Examples

Example 41. Data Collection example

This parameter example is from a setup that uses the default settings for the *Serial Port* monitor, resulting in the monitor performing data collection of the data received over **COM1** when a fault is detected by another monitor. Other than the mandatory *Port* declaration, the setup uses default values.

Parameter	Value
Port	COM1

Example 42. Fault Detection example

This parameter example is from a setup that detects a fault on **COM1**. In addition to normal data collection, this setup generates a fault when the text **ERROR** is received over the serial port.

Parameter	Value
Port	COM1
FaultRegex	ERROR

Example 43. Combined Automation and Fault Detection example

This example might be used when fuzzing a network device such as a router. One *Serial Port* monitor is configured to wait until the router has booted before starting the fuzzing session. Another *Serial Port* monitor is configured to detect faults and also to wait for the router to finish rebooting after a fault is detected. The [IpPower9258 Monitor](#) is configured to reboot the router after a fault is detected.

Serial Port (Wait for boot) parameters

Parameter	Value
Monitor	SerialPort
Name	Wait for boot
Port	COM1
WaitRegex	Boot up completed

Serial Port (Detect fault) parameters

Parameter	Value
Monitor	SerialPort
Name	Detect fault
Port	COM1
FaultRegex	ERROR
WaitRegex	Boot up completed
WaitWhen	OnIterationAfterFault

IpPower9258 (Reboot router) parameters

Parameter	Value
Monitor	IpPower9258
Name	Reboot router
Host	192.168.1.1:8080
Port	1
User	peach
Password	PeachPower
When	OnFault

Example 44. Multiple Serial Port monitors for different ports

This example connects Peach to the console port and also the debug port of a target device. The monitor on the console port is set up for fault detection, data collection, and automation. The monitor on the debug port is set up for data collection.

Console Port

Parameter	Value
Monitor	SerialPort
Name	Console Port
Port	COM1
FaultRegex	ERROR
WaitRegex	Boot up completed
WaitWhen	OnIterationAfterFault

Debug Port

Parameter	Value
Monitor	SerialPort
Name	Debug Port
Port	COM2

9.29. SNMP Power Monitor

Monitor Categories: Automation

The *SNMP Power* monitor switches outlets on a power distribution unit (PDU) on and off via SNMPv1. This monitor is useful for automatically power cycling devices during a fuzzing session.

Each *SNMP Power* monitor switches one or more of a PDU's outlets, according to the configuration. All affected outlets are given the same commands, so turning some outlets on and others off would require another monitor. The monitor can reset the power outlets at the following points in time:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model



The [IpPower9258 Monitor](#) and [ApcPower Monitor](#) provide similar features, for IP Power 9258 and APC devices, respectively. For controlling power to a device by wiring through a relay, Peach provides a monitor for the [CanaKit 4-Port USB Relay Controller](#).

9.29.1. Parameters

Required:

Host

IP address of the switched power distribution unit.

OIDs

Comma-separated list of OIDs for controlling the power outlets. To determine the OIDs, start by installing the SNMP MIB provided by the device manufacturer. Use a utility like `snmptranslate` to lookup numeric OID associate with the OID name of an outlet. For example, `snmptranslate -On PowerNet-MIB::sPDUOutletCtl.1` indicates `.1.3.6.1.4.1.318.1.1.4.4.2.1.3.1` is the OID to use for outlet 1 on an APC Switched Power Distribution Unit (AC7900).

Optional:

OnCode

On indicator code used by outlet OIDs. Default is `1`.

OffCode

Off indicator code used by outlet OIDs. Default is `2`.

Port

SNMP port on the switched power distribution unit. Default is **161**.

ReadCommunity

SNMP community string to use when reading the state of the outlets. Default is **public**.

WriteCommunity

SNMP community string to use when modifying the state of the outlets. Default is **private**.

RequestTimeout

Maximum duration in milliseconds to block when sending an SNMP request to the PDU. Default is **1000**.

SanityCheckOnStart

On startup, ensure switch state changes persist. Default is **true**.

SanityCheckWaitTimeout

Maximum duration to wait for state change to take effect during startup sanity check. Default is **3000**.

ResetOnCall

Reset power when the specified call is received from the state model. This value is used only when the *When* parameter is set to **OnCall**.

PowerOffOnEnd

Power off when the fuzzing session completes, default is **false**.

PowerOnOffPause

Pause in milliseconds between power off/power on, default is **500**.

When

When to reset power on the specified outlets. Default is **OnFault**.

"When" Setting	Description
DetectFault	Reset power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Reset power when the fuzzing session starts. This occurs once per session.
OnEnd	Reset power when the fuzzing session stops. This occurs once per session.
OnIterationStart	Reset power at the start of each iteration.
OnIterationEnd	Reset power at the end of each iteration.

"When" Setting	Description
OnFault	Reset power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Reset power at the start of an iteration that immediately follows a fault detection.
OnCall	Reset power when the call specified by the <i>ResetOnCall</i> parameter is received from the state model.

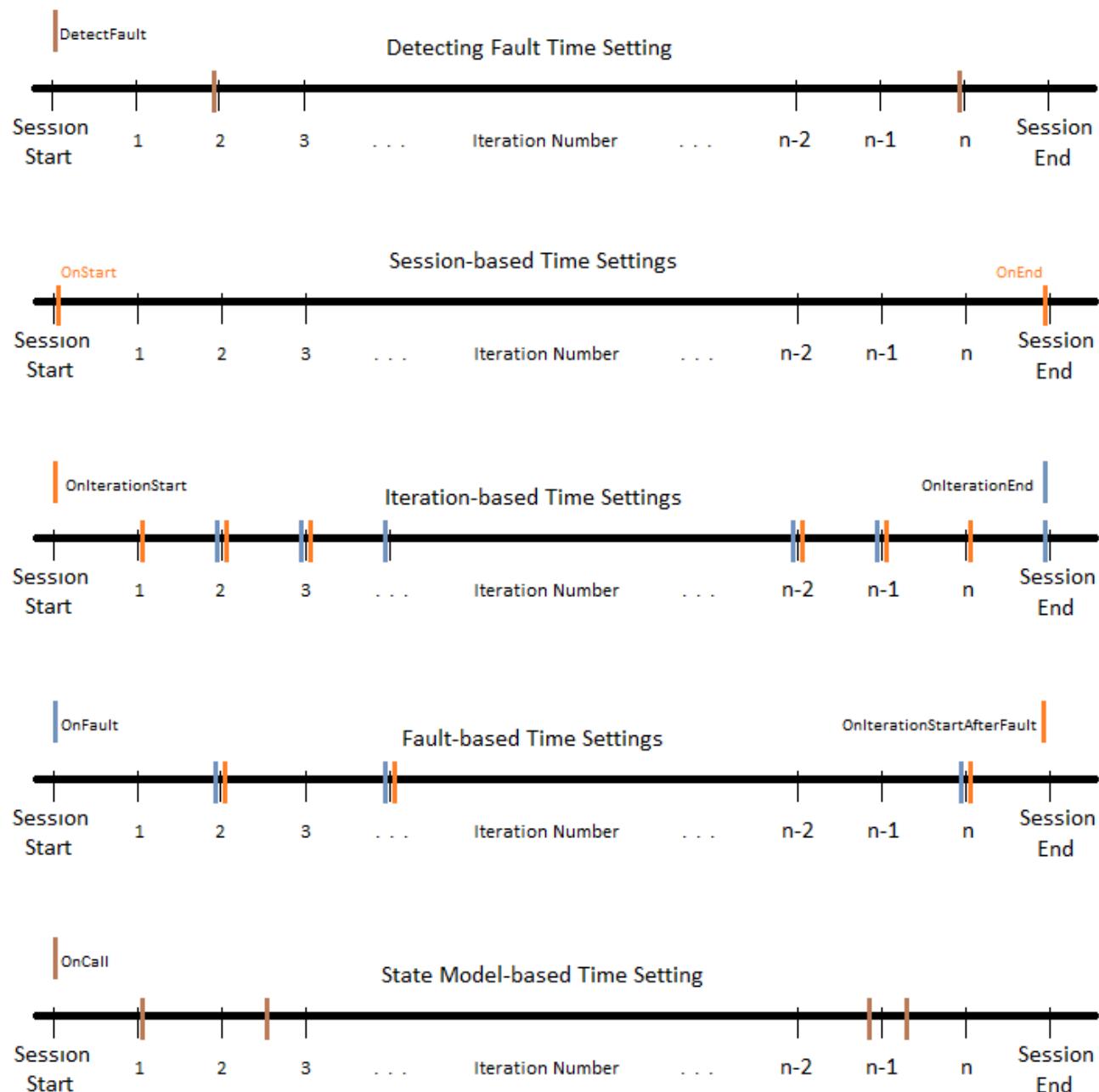


Figure 8. When Choices for Performing an Action

9.29.2. Examples

9.30. Socket Monitor

Monitor Category: Data collection, Fault detection

The *Socket* monitor waits for an incoming TCP or UDP connection at the end of a test iteration. This monitor accepts a point-to-point connection to a single host or a multicast connection where the host broadcasts to one or more clients. Multicast connections are not supported when using the TCP protocol.

The monitor can be configured to be used for data collection or fault detection depending on the *FaultOnSuccess* parameter value and whether or not data is received within the specified timeout. The following table provides the available options:

FaultOnSuccess	Data Received	Behavior
true	yes	Data collection
true	no	Fault detection
false	yes	Fault detection
false	no	Data collection

9.30.1. Parameters

Required:

None.

Optional:

Host

IP address of the remote host to receive data from. Defaults to "", which means accept data from any host.

Interface

IP address of the interface to listen on. Defaults to **0.0.0.0**, which means listen to all interfaces on the host.

Port

Port to listen on. Defaults to **8080**.

Protocol

Protocol type to listen for. Defaults to **tcp**. Available options for this parameter are **tcp** and **udp**.

Timeout

Length of time in milliseconds to wait for an incoming connection. Defaults to **1000**.

FaultOnSuccess

Generate a fault if no data is received. Defaults to `false`.

9.30.2. Examples

Example 45. Fault Detection example

This example generates a fault if data from a tcp connection on port `53` is received at the end of a test iteration.

Parameter	Value
Port	<code>53</code>

9.31. SshCommand Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *SshCommand* monitor connects to a remote host over SSH (Secure Shell), runs a command, and waits for the command to complete. The output from the process is logged when a fault is detected. This monitor can operate as a fault detector, data collector, and automation module depending on configuration.

SshCommand supports password, keyboard, and private key authentication methods.

To increase the speed of operation, the monitor holds open the SSH connection to the remote machine across test iterations. This removes the cost of authenticating every time the command is executed. If multiple *SshCommand* monitors are configured against the same remote host, multiple SSH connections are created and held open.

Fault Detection

This monitor can perform fault detection depending on the configuration of the *FaultOnMatch* and *CheckValue* parameters. The following table describes the behavior of these parameters:

FaultOnMatch	CheckValue match	Behavior
true	yes	Fault detection
true	no	Data collection
false	yes	Data collection
false	no	Fault detection

This monitor will also automatically detect [AddressSanitizer](#) crash information and generate a fault if found.

Data Collection

The monitor always collects the output from the executed command and reports it for logging when a fault is detected.

Automation

The *SshCommand* monitor can run the specified command at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- While detecting a fault

- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

9.31.1. Parameters

Required:

Host

Remote hostname or IP address for the SSH connection.

Username

Username for authentication with the remote host.

Command

The command to execute on the remote host.

Optional:

Password

Password for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

KeyPath

A local path to the private part of an SSH key-pair to be used for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

CheckValue

A regular expression to match the command output. Defaults to `""`.

FaultOnMatch

Trigger a fault if *FaultOnMatch* is `true` and the *CheckValue* regular expression matches, or *FaultOnMatch* is `false` and the *CheckValue* regular expression does not match. Defaults to `false`.

StartOnCall

Run the specified command after the monitor receives the specified call from the state machine. This value is used only when the *When* parameter is set to `OnCall`.

When

Specify one of the following values to determine when to run the specified command:

When Value	Description
<code>DetectFault</code>	Run the command to perform fault detection. Requires a regular expression to be specified in the <i>CheckValue</i> parameter. This is the default setting.
<code>OnStart</code>	Run the command when fuzzing session starts. This occurs once per session.
<code>OnEnd</code>	Run the command when fuzzing session stops. This occurs once per session.
<code>OnIterationStart</code>	Run the command at the start of each iteration.
<code>OnIterationEnd</code>	Run the command at the end of each iteration.
<code>OnFault</code>	Run the command when any monitor detects a fault.
<code>OnIterationStartAfterFault</code>	Run the command at the start of an iteration that immediately follows a fault detection.
<code>OnCall</code>	Run the command upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model.

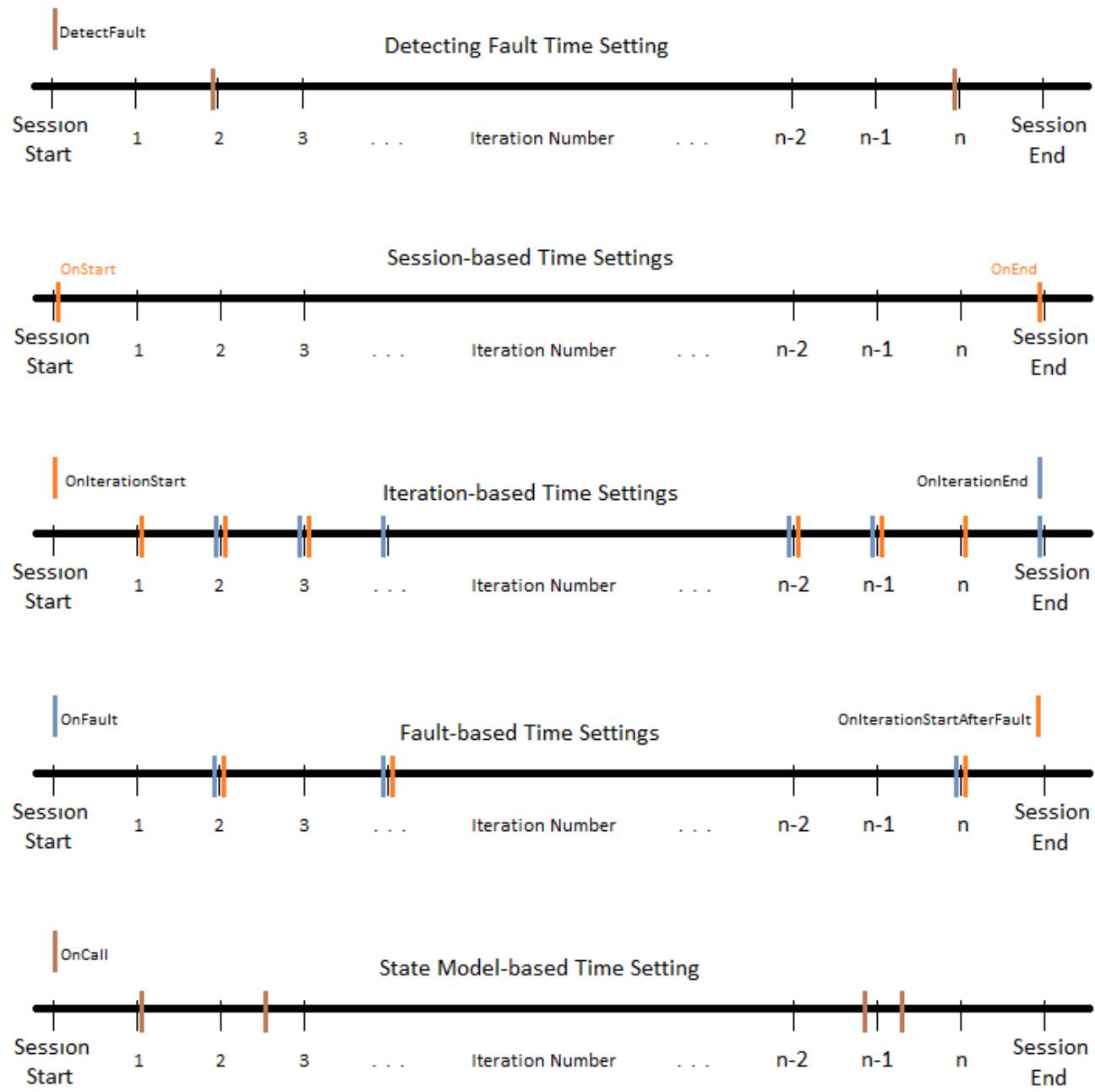


Figure 9. When Choices

9.31.2. Examples

Example 46. Fault Detection example

This example connects to the target machine using SSH during the fault detection phase of a test iteration. A fault occurs if any core files exist.

Parameter	Value
Host	my.target.com
Username	tester
Password	Password!
Command	ls /var/cores/*.core
CheckValue	target.*?.core
FaultOnMatch	true

9.32. SshDownloader Monitor

Monitor Categories: Data collection

The *SshDownloader* monitor downloads a file or folder from a remote host over SSH SFTP (Secure Shell File Transfer Protocol) after any other monitor detects a fault.

SshDownloader supports password, keyboard, and private key authentication methods.

SshDownloader can be configured to delete files from the source after they have been downloaded.

9.32.1. Parameters

Required:

Host

Remote hostname or IP address for the SSH connection.

Username

Username for authentication with the remote host.

Optional:

Password

Password for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

KeyPath

A local path to the private part of an SSH key-pair to be used for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

File

Path of the remote file to download. Defaults to `""`. Either the *File* or the *Folder* parameter must be set.

Folder

Path of the remote folder to download. Defaults to `""`. Either the *File* or the *Folder* parameter must be set.

Remove

When this value is set to `true`, remove the remote file after the download completes. Defaults to `true`.

9.32.2. Examples

Example 47. Download a log file

This example downloads a log file when a fault is detected by any other monitor.

Parameter	Value
Host	<code>my.target.com</code>
Username	<code>tester</code>
Password	<code>Password!</code>
File	<code>/var/log/syslog</code>
Remove	<code>false</code>

9.33. Syslog Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *Syslog* monitor listens on a specified port for incoming syslog messages. This monitor is capable of performing automation, data collection, and fault detection.

The default usage of the *Syslog* monitor is data collection. The syslog messages received are logged when a fault occurs.

To perform fault detection, specify a regular expression using the *FaultRegex* parameter. When the regular expression matches an incoming syslog message, Peach generates a fault.

For automation tasks, use the *WaitForRegex* and *WaitWhen* parameters. These automation parameters cause Peach to wait for matching input before continuing. The *Syslog* monitor can wait at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

9.33.1. Parameters

Required:

None.

Optional:

Port

Port number to listen on. The default value is [514](#).

Interface

IP address of the interface to listen on. The default value is [0.0.0.0](#), which listens on all interfaces.

FaultRegex

Generate a fault when the specified regular expression matches received data. This causes the *Syslog* monitor to be used for fault detection.

WaitRegex

Wait until the specified regular expression matches received data. This causes the *Syslog* monitor to be used for automation.

WaitOnCall

Begin waiting for the regular expression specified in the *WaitRegex* parameter after the monitor receives the specified call from the state machine. This value is used only when the *WaitWhen* parameter is set to **OnCall**.

WaitWhen

Specify one of the following values to determine when to begin waiting for the regular expression specified in the *WaitRegex* parameter to match received data:

"WaitWhen" Setting	Description
OnStart	Waits when the fuzzing session starts. This occurs once per session. This is the default setting.
OnEnd	Waits when the fuzzing session stops. This occurs once per session.
OnIterationStart	Waits at the start of each iteration.
OnIterationEnd	Waits at the end of each iteration.
OnFault	Waits when any monitor detects a fault.
OnIterationStartAfterFault	Waits at the start of the iteration that immediately follows a fault detection.
OnCall	Waits upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model.

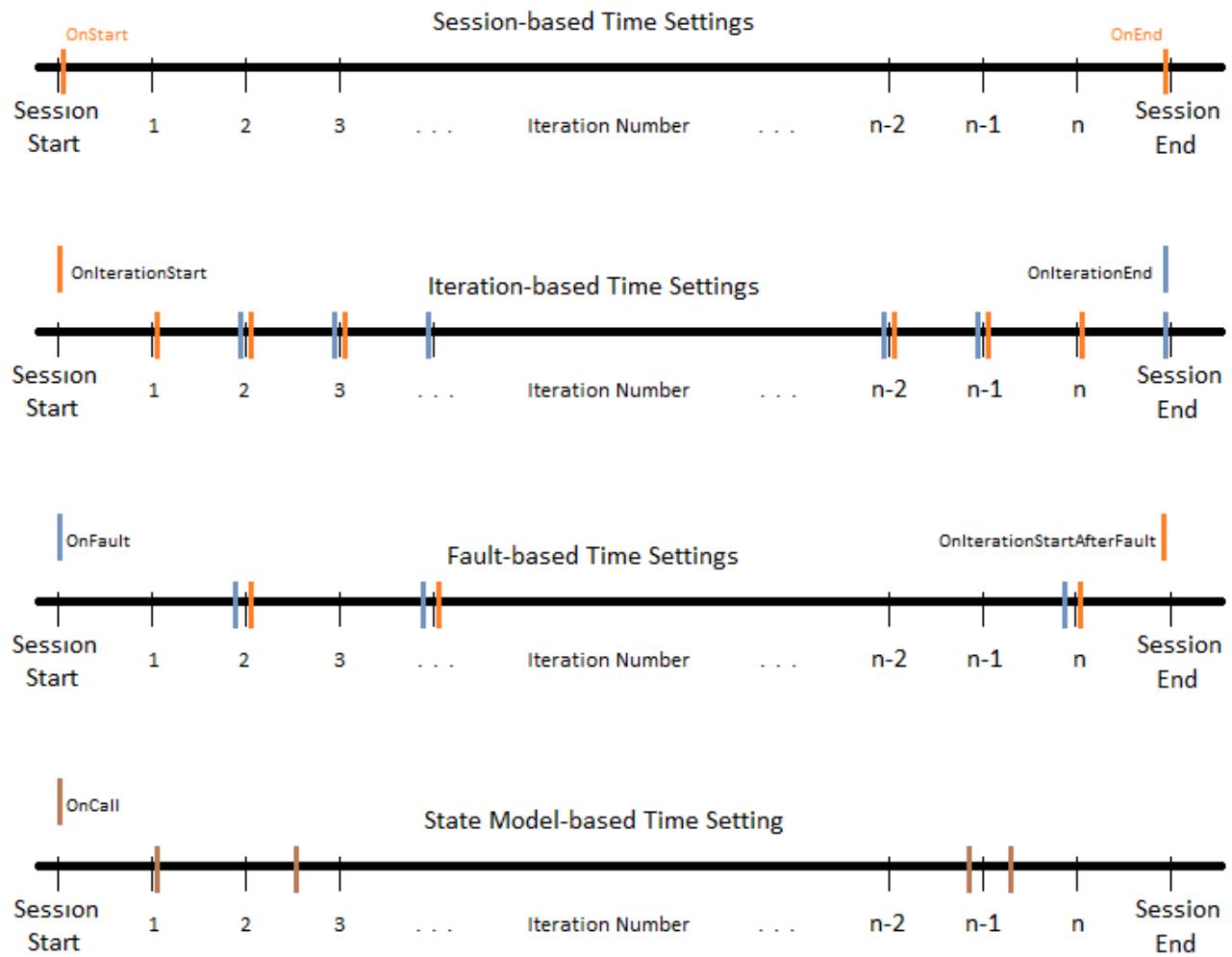


Figure 10. WaitWhen Choices

9.33.2. Examples

Example 48. Data Collection example

This parameter example is from a setup that uses the `Syslog` monitor to listen on the default port for incoming messages. When a fault occurs, all messages are saved. Default values are used; no values are specified.

Example 49. Fault Detection example

This parameter example is from a setup that detects a fault from incoming syslog messages. In addition to normal data collection, this setup generates a fault when the text `ERROR` is received.

Parameter	Value
<code>FaultRegex</code>	<code>ERROR</code>

Example 50. Combined Automation and Fault Detection example

This example might be used when fuzzing a network device such as a router. One *Syslog* monitor is configured to wait until the router has booted before starting the fuzzing session. Another *Syslog* monitor is configured to detect faults and also to wait for the router to finish rebooting after a fault is detected. The [IpPower9258 Monitor](#) is configured to reboot the router after a fault is detected.

Syslog (Wait for boot) parameters

Parameter	Value
Monitor	Syslog
Name	Wait for boot
WaitRegex	Boot up completed

Syslog (Detect fault) parameters

Parameter	Value
Monitor	Syslog
Name	Detect fault
FaultRegex	ERROR
WaitRegex	Boot up completed
WaitWhen	OnIterationAfterFault

IpPower9258 (Reboot router) parameters

Parameter	Value
Monitor	IpPower9258
Name	Reboot router
Host	192.168.1.1:8080
Port	1
User	peach
Password	PeachPower
When	OnFault

9.34. TcpPort Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *TcpPort* monitor detects state changes on TCP ports. A state change is a transition in port status from *Open* to *Closed* or from *Closed* to *Open*. The *TcpPort* monitor can be configured in the following ways:

- as an automation task (wait until a specified state occurs)
- fault detection (fault on a specific state)
- data collection (report the current state)

For automation tasks, the *When* parameter can be configured to detect the state of a TCP port at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

9.34.1. Parameters

Required:

Host

Hostname or IP address of the remote host

Port

Port number to monitor

Optional:

Action

Action to take (Automation, Data, Fault). Defaults to [Automation](#).

Automation

Wait for the port to reach a specified state. The *When* parameter determines when the monitor should detect the state of the specified port.

Data

Report the state of the port immediately following a fault that is detected by any other monitor.

Fault

Generate a fault if the port is in a specified state at the end of an iteration.

State

Port state to monitor. The default value is **Open**.

Open

The port is available for use.

Closed

The port is not available.

Timeout

The amount of time in milliseconds to wait for the specified TCP state to occur. Specify **-1** to indicate an infinite timeout.

WaitOnCall

Detect port state upon receipt of the specified call from the state model. This value is used only when the *When* parameter is set to **OnCall**.

When

Specify one of the following values to control when port state detection should occur during a fuzzing session:

"When" Setting	Description
OnStart	Detect port state when the fuzzing session starts. This occurs once per session.
OnEnd	Detect port state when the fuzzing session stops. This occurs once per session.
OnIterationStart	Detect port state at the start of each iteration.
OnIterationEnd	Detect port state at the end of each iteration.
OnFault	Detect port state when any monitor detects a fault.
OnIterationStartAfterFault	Detect port state at the start of the iteration that immediately follows a fault detection.
OnCall	Detect port state upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model. This is the default setting.

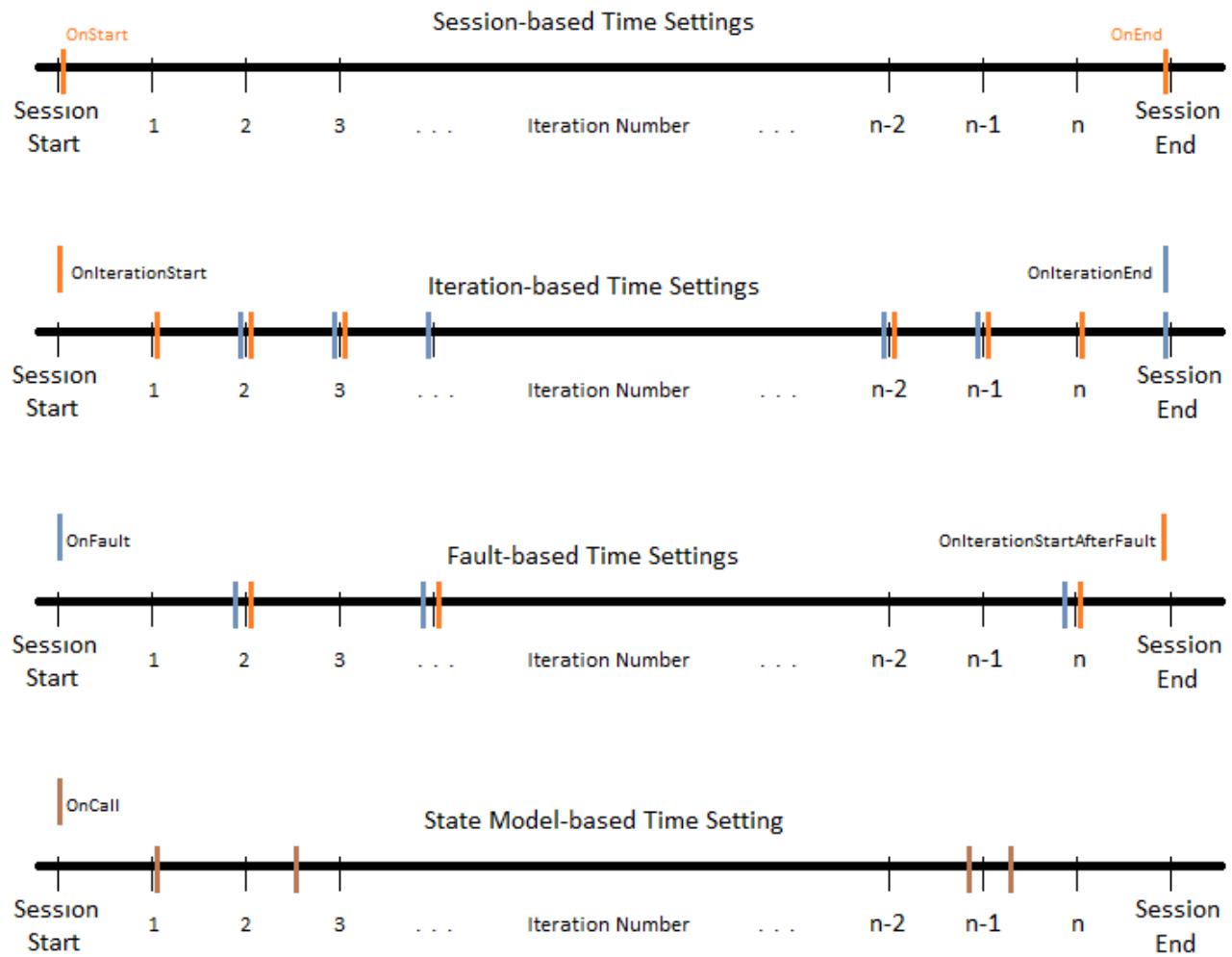


Figure 11. When Choices

9.34.2. Examples

Example 51. Automation example: Open

This example causes Peach to wait until the remote port is in an open state after the `WaitForPort` call is received from the state model. No timeout interval is provided, so Peach will wait forever.

Parameter	Value
Host	192.168.133.4
Port	502
WaitOnCall	<code>WaitForPort</code>

Example 52. Automation example: Closed

This example causes Peach to wait until the remote port is in a closed state after the `WaitForPort` call is received from the state model. No timeout interval is provided, so Peach will wait forever.

Parameter	Value
Host	192.168.133.4
Port	502
State	Closed
WaitOnCall	WaitForPort

Example 53. Fault Detection example

This example inspects the state of the remote port at the end of an iteration. A fault is generated if the port is closed at the end of an iteration.

Parameter	Value
Host	192.168.133.4
Port	502
Action	Fault
State	Closed

Example 54. Data Collection example

This example causes the TcpPort monitor to report the state of a port after a fault is detected by any other monitor.

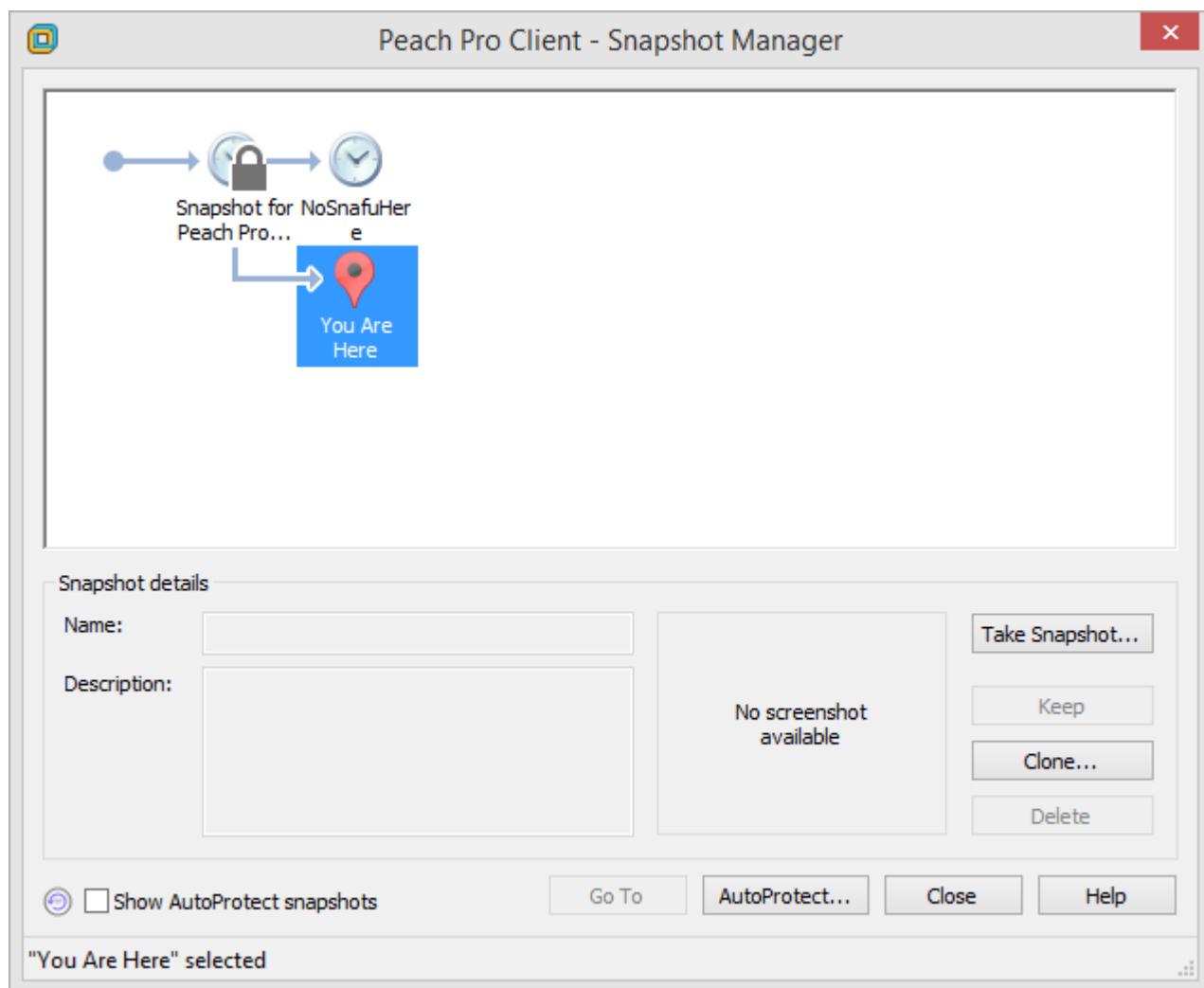
Parameter	Value
Host	192.168.133.4
Port	502
Action	Data

9.35. Vmware Monitor

Monitor Categories: Automation

The *Vmware* monitor can control a VMware virtual machine (VM). This monitor works with a snapshot of the VM that provides a consistent VM environment throughout a fuzzing session. The *Vmware* monitor can start a VM snapshot and, optionally, reset the VM to a snapshot configuration each iteration.

The following illustration shows the VMWare Snapshot Manager, which is used for managing snapshots for a particular VM.



9.35.1. Parameters

Required:

Vmx

Path to the virtual machine.



When using with vSphere/ESXi, prefix the VM image name with the storage location in brackets. For example, **[ha-datacenter/datastore1] guest/guest.vmx**.

Optional:

SnapshotName

VM snapshot name.

Either *SnapshotName* or *SnapshotIndex* must be specified, but it is an error to specify both.

SnapshotIndex

VM snapshot index specification.

Either *SnapshotName* or *SnapshotIndex* must be specified, but it is an error to specify both.

The index specification is a list of zero-based index values delimited by a period (.). The specification resolves which leaf in a tree of snapshots should be used.

For example, in the following tree of snapshots, the snapshot named **Snapshot 1.2.1** would be used when **0.1.0** is specified.

- Snapshot 1
 - Snapshot 1.1
 - Snapshot 1.2
 - **Snapshot 1.2.1**
 - Snapshot 1.2.2
- Snapshot 2
 - Snapshot 2.1
 - Snapshot 2.2

Host

Hostname or IP address the VMware host.

HostPort

TCP/IP port of the VMware host.

Login

Username for authentication with the VMware host.

Password

Password for authentication with the VMware host.

HostType

Type of remote host, defaults to **Default**

VM Product	Description
VIserver	vCenter Server, ESX/ESXi hosts, VMWare Server 2.0
Workstation	VMWare Workstation
WorkstationShared	VMWare Workstation (Shared Mode)
Player	VMWare Player
Server	VMWare Server 1.0.x
Default	Default

ResetEveryIteration

If **true** is specified, reset the VM on every iteration. Defaults to **false**.

ResetOnFaultBeforeCollection

If **true** is specified, reset the VM after a fault is detected by any other monitor. Defaults to **false**.

StopOnFaultBeforeCollection

If **true** is specified, stop the VM after a fault is detected by any other monitor. Defaults to **false**.

WaitForToolsInGuest

If **true** is specified, wait for VMware tools to start within the guest whenever a VM is restarted. Defaults to **true**.

WaitTimeout

The number of seconds to wait for VMware tools to start within a guest. Defaults to **600**.

Headless

Run a VM without a GUI. Using this parameter can improve performance but may cause issues if the target interacts with the desktop. Defaults to **true**.

9.35.2. Examples

Example 55. Start Virtual Machine

This parameter example is from a setup that programmatically starts a VM.

- The monitor requires both the physical filename with path of the VM and the `SnapshotName` of the VM.
- The `HostType` identifies the VMWare product that is hosting the VM.
- The `Headless` parameter provides visual feedback while configuring the test setup.

Parameter	Value
Vmx	D:\VirtualMachines\OfficeWebTest\OfficeWebTest.vmx
HostType	Workstation
SnapshotName	Fuzzing
Headless	false

Example 56. Start Virtual Machine hosted on ESXi

This parameter example is from a setup that programmatically starts a VM.

Parameter	Value
Vmx	[ha-datacenter/datastore1] guest/guest.vmx
SnapshotName	Fuzzing

9.36. WindowsDebugger Monitor (Windows)

Monitor Categories: Automation, Data collection, Fault detection

The *WindowsDebugger* monitor controls a windows debugger instance.

This monitor launches an executable file, a process, or a service with the debugger attached; or, this monitor can attach the debugger to a running executable, process, or service.

The *WindowsDebugger* performs automation, fault detection, and data collection.

- Automation manages when the fuzzing target (process, service, etc) starts and restarts. This can occur at the start of a fuzzing session, at the start of each iteration, after detecting a fault, or upon receiving a call from the state model.
- Fault detection watches and reports crashes and faults—when the target exits prematurely, and when the target fails to exit.
- Data collection retrieves stack traces, logs, and other information provided by the debugger. Note that this monitor provides bucket information—major and minor hash values—as part of data collected on faults. For more information on bucketing provided by this monitor, see [!exploitable](#).

9.36.1. Parameters

Exactly one of the three required parameters is needed for each instance of this monitor. The other two required parameters are not used.

Executable

Executable to launch via the debugger. If the executable has command-line arguments, specify these in the "Arguments" parameter.

ProcessName

Name of the process that the debugger attaches.

Service

Name of the Windows process or service to attach to the debugger. If the service crashes or is stopped, this monitor will restart the service.

Optional:

Arguments

Command-line arguments for the executable file specified with the "Executable" parameter.

SymbolsPath

Path to the debugging symbol files. The default value is Microsoft public symbols server,SRV*<http://msdl.microsoft.com/download/symbols>.

WinDbgPath

Path to the Windows Debugger installation. If undeclared, Peach attempts to locate a local installation of the debugger.

StartOnCall

Defers launching the target until the state model issues a call to the monitor to begin. Upon receiving the call, the debugger attaches to the process, or starts the process or executable.

IgnoreFirstChanceGuardPage

Ignores faults from the first chance guard page. These faults are sometimes false positives or anti-debugging faults, defaults to false.

IgnoreSecondChanceGuardPage

Ignores faults from the second chance guard page. These faults are sometimes false positives or anti-debugging faults, defaults to false.

IgnoreFirstChanceReadAv

Ignores faults from the first chance read access violations. These faults are sometimes false positives, defaults to false. When monitoring a Java process, it is recommended to set this parameter to true.

NoCpuKill

Allows or disallows the CPU to idle. If true, the CPU can idle without terminating the target. If false, Peach polls and then terminates the target if it is caught idling. The default value is false.

CpuPollInterval

Specifies the time interval, expressed in milliseconds (ms), that the monitor waits between successive polls of the target. This argument is used when NoCpuKill is false. The default value is 200 ms.

FaultOnEarlyExit

Triggers a fault if the target exits prematurely, defaults to false.

RestartAfterFault

If "true", restarts the target when any monitor detects a fault. If "false", restarts the target only if the process exits or crashes.

This argument defaults to true.

RestartOnEachTest

Restarts the process for each iteration, defaults to false.

ServiceStartTimeout

When debugging a windows service, this specifies how long the monitor should wait for the service to start. The default value is 60 seconds.

WaitForExitOnCall

Exits the target upon receiving a call from the state model. If the call fails to occur within an acceptable waiting period, issue a fault and then exit. The WaitForExitTimeout parameter specifies the waiting period.

WaitForExitTimeout

Specifies the WaitForExitOnCall timeout value, expressed in milliseconds, defaults to 10000 ms (10 sec). The value -1 specifies an infinite waiting period.

9.36.2. Examples

Command Line Configuration

This parameter example is from a setup that launches an application with command-line arguments from the Windows Debugger. The setup also supplies the path where the Windows Debugger resides.

Parameter	Value
Executable	CrashableServer.exe
Arguments	127.0.0.1 4244
WinDbgPath	C:\Program Files (x86)\Debugging Tools for Windows (x86)

Service Configuration

This parameter example attaches the debugger to a service.

Parameter	Value
Service	WinDefend

Process Configuration

This parameter example attaches the debugger to a process name.

Parameter	Value
ProcessName	CrashableServer.exe

StartOnCall Configuration

This parameter example uses the debugger to launch an application with command-line arguments. Further, the launch starts after the monitor receives a call request from the state model to initiate the launch.

Parameter	Value
Executable	CrashableServer.exe
Arguments	127.0.0.1 4244
StartOnCall	launchProgram

Exit Configurations

This parameter example uses the debugger to launch an application with command-line arguments. Further, the monitor polls the application for idleness, and terminates the application if it finds an idle CPU. At the end of each iteration, Peach waits a maximum of 250ms for the application to close of its own accord before terminating the application.

Parameter	Value
Executable	CrashableServer.exe
Arguments	127.0.0.1 4244
NoCpuKill	true
FaultOnEarlyExit	false
WaitForExitTimeout	250

WaitForExitOnCall Configuration

This parameter example uses the debugger to launch an application with command-line arguments. Further, the monitor defers closing the application until receiving the notice from the state model.

Parameter	Value
Executable	CrashableServer.exe
Arguments	127.0.0.1 4244
WaitForExitOnCall	exitProgram

9.37. WindowsKernelDebugger Monitor (Windows)

Monitor Categories: Data collection, Fault detection

The *WindowsKernelDebugger* monitor controls debugging of a remote windows kernel debugging instance. At least two machines are involved:

- The target machine that receives fuzzing data.
- The host machine that controls the debugging session. Peach resides on this machine

The *WindowsKernelDebugger* performs fault detection, and data collection.

- Fault detection watches and reports crashes and faults—when the target exits prematurely, and when the target fails to exit.
- Data collection retrieves stack traces, logs, and other information provided by the debugger. Note that this monitor provides bucket information—major and minor hash values—as part of data collected on faults. For more information on bucketing provided by this monitor, see [!Exploitable](#).

The workflow for this configuration follows:

1. On the host machine, start Peach with the WindowsKernelDebugger Monitor.
This action starts the fuzzing and loads a copy of Windbg in kernel mode. The host machine enters a waiting state because the target machine will initiate and secure the connection for the debugging session.
2. On the target machine, open Windbg, and choose "Connect to Remote Session" from the File menu.
 - For the "Connection String", specify the "transport:port" and the server for the debugging session, as in this example: `net:port=5000,key=1.2.3.4`.
 - Click OK.

The debug session starts on the target machine. The target machine initiates contact with the host to establish the connection. Once the connection is secure, the host machine (with Peach) drives the activities.



Instrumenting this configuration requires additional monitors external to the target. The monitors need to be able to restart the target after a crash, and to have the host resume fuzzing.

9.37.1. Parameters

KernelConnectionString

Connection string for attaching the debugger to a kernel-mode process (e.g. a driver). The connection string `net:port=5000,key=1.2.3.4` indicates this is the debug server and uses the tcp transport on port 5000. The Windows debugger offers a choice of transports; and, the port is arbitrary. For more information on using other values, see MSDN articles on live kernel mode,

debugging, remote debugging, and kernel connection strings.

Optional:

SymbolsPath

Path to the debugging symbol files. The default value is Microsoft public symbols server
`SRV*http://msdl.microsoft.com/download/symbols`.

WinDbgPath

Path to the windbg installation. If undeclared, Peach attempts to locate a local installation of the debugger.

ConnectTimeout

Duration, in milliseconds, to wait to receive a kernel connection. The default value is 3000 ms.

9.38. WindowsService Monitor (Windows)

Monitor Categories: Automation, Fault detection

The *WindowsService* monitor controls a windows service. When the monitor runs, it checks the state of the service. If the service exits prematurely, *WindowsService* generates a fault. If a fault is detected by from *windowsService* or from another monitor, the monitor collects status of the service and continues.

If the service is not running, this monitor attempts to restore the service to a running state, whether starting, restarting, or resuming from a paused state.

9.38.1. Parameters

Required:

Service

Name that identifies the service to the system, such as the display name of the service.

Optional:

FaultOnEarlyExit

Fault if the service exits early, defaults to false.

MachineName

Name of the computer on which the service resides, defaults to local machine.

Restart

Specifies whether to start the service on each iteration, defaults to false.

StartTimeout

Specifies the duration, in minutes, to wait for the service to start, defaults to 1 minute.

9.38.2. Examples

Example 57. Start IIS

This parameter example is from a minimal setup that monitors the Internet Information Service (IIS) at the beginning of the fuzzing run. The example runs on the local machine with a startup timeout period of 1 minute. In this case, the monitor does not generate a fault if the service exits early, nor restarts the service for each test iteration.

Parameter	Value
Service	World Wide Web Publishing Service

10. Reproducing Faults

A fault that occurs in a fuzzing session needs to be reproducible so that it can be investigated, understood, and mitigated. Peach has the following features that aid in reproducing faults that occur during fuzzing:

- Automatic fault reproduction
- Replay the fuzzing session

When a fault is detected at the end of a test case, Peach automatically enters reproduction mode. How Peach implements fault reproduction depends on the type of target being fuzzed.

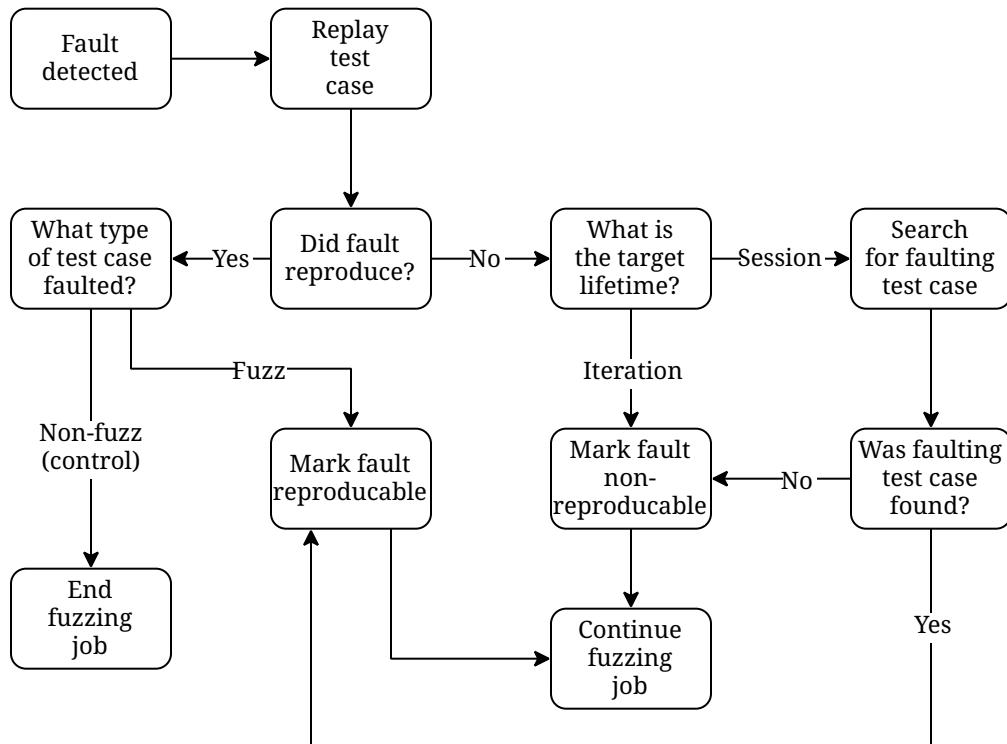


Figure 12. Fault Reproduction Flow Chart

10.1. Iteration Based Targets

Iteration based test targets are targets that restart with every test case. File fuzzing targets are typically iteration based targets. For example, an image viewer that is started during every test case would be considered an iteration target. When reproducing a fault found on an iteration target, Peach will only try the initial faulting test case. Peach will not try and reproduce the fault using previous test cases.



There are rare cases in which an iteration based target retains enough state through a data store that a prior test case may impact the crash. As faults are triaged if you find a significant number of faults that fall into this category it may make sense to reconfigure the Pit to be session based. Currently this is only possible if you have developer access to the pit in question.

10.2. Session Based Targets

Session based test targets are targets that do not restart with every test case. Server or service targets are typically session based targets. For example, an FTP server that is started at the beginning of a fuzzing session would be considered a session target. With session targets, it is possible that a detected fault is the byproduct of previously executed test cases. Therefore, when reproducing a fault found on a session target, Peach will replay sequences of previous test cases. Peach will then locate the exact test case or range of test cases required to reproduce the fault. If a fault is reproduced on a non-fuzz (control) test case that runs after one or more fuzz test cases, Peach will record the range of test cases executed and consider the fault to be reproducible.

10.2.1. Searching For Reproduction

When Peach detects a fault, it always attempts to reproduce the fault. This reproduction involves searching through previously executed test cases. The following steps describe the major elements of the search:

1. Replay the most recent test. This typically is where the fault surfaces.
If the fault reproduces, the system logs the information and the search finishes.
If the test target restarts with each fuzzing iteration, the search finishes because the test started in a known, clean state. Otherwise, continue with step 2.
2. Replay the last 10 iterations, running them in the same sequence as in the fuzzing session.
If the fault reproduces, the system logs the information and the search finishes. Otherwise, continue with step 3.
3. Double the number of iterations, and run them in the same sequence as in the fuzzing session.
If the fault reproduces, the system logs the information and the search finishes. If not, continue the search, each time doubling the number of iterations to run. The criteria for stopping follows:
 - Reproduce the fault.
 - Encounter a critical point in the data; the effort to recreate the fault encountered another fault that peach found and logged during the test session.
 - Encounter a user-specified limit for the search.

Using a search limit of 200, the following sets of iterations would run until the limit is reached:

- 1 iteration
- 10 iterations

- 20 iterations
- 40 iterations
- 80 iterations
- 160 iterations
- 200 iterations (greater than 160 iterations and less than 320 iterations, which is the next cutoff point)

All told, a maximum of 511 iterations would run ($1 + 10 + 20 + \dots + 200$) without human intervention; and some of the iterations would be repeated on subsequent passes.

Some additional considerations about automated fault reproduction include the following:

- Control iterations are treated as in a normal fuzzing session. That is, the results of a control iteration serves as a standard of comparison for the results of record iterations. Also, the frequency of performing a control iteration is determined by the *controlIteration* attribute of the *Test* element.
- Record iterations are still compared to control iterations. If the results of a record iteration matches the results of a control iteration, all is well and good, and processing advances to the next iterations. If the results of a record iteration do not match the results of a control iteration, the results are logged and the search to reproduce the fault ends.

10.3. How to Control the Automated Fault Reproduction

If you use a licensed Peach Pit, the appropriate automated settings are already included in the fuzzing definition.

10.4. Replay the Fuzzing Session

Any test case generated by Peach can be replayed. This allows an engineer to more easily perform root cause analysis of the fault discovered in the device under test. In order for Peach to replay a test case, you need:

- The exact version of Peach
- The pit used in the fuzzing session (the DataModel and the StateModel must be identical to those used in the fuzzing session)
- The seed value used in the fuzzing job
- The test case that caused the fault

With this information, you can re-run the appropriate part of the fuzzing session. For an example, see the [example](#) listed in the [The Peach Command Line Interface](#).

If the command line does not specify an iteration range or a starting iteration, the entire fuzzing session runs.

11. Appendixes

11.1. Using Virtual Machines

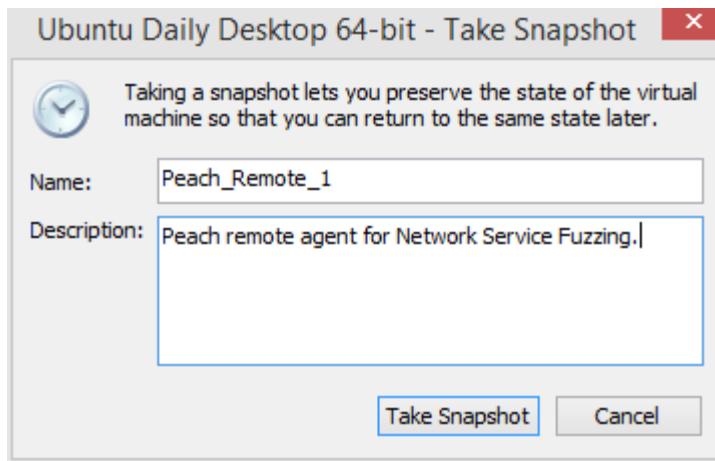
When a configuration requires two or more machines, one machine runs Peach to generate the fuzzing test cases and to manage the fuzzing contributions of the other, remote machines. Within each remote machine, an agent process runs that handles monitoring of the test target and communications with Peach.

A remote machine can be either a physical machine or a virtual machine (VM). Peach includes monitors for automating VMs that use VMware products. Further, many of the recipes assume a VM is used and that it has a saved state called a snapshot. Peach can use the snapshot for initially loading the VM in a fuzzing job and for resetting the VM after detecting a fault.

Since a Peach agent always runs in the remote machines, it helps the automation effort to create the snapshot with the Peach agent running. The resulting benefit is that whenever Peach reverts to the snapshot, the VM starts with the Peach agent loaded and running.

The steps to create a VM snapshot for Peach follows:

1. Start the VM to house the test target (such as a network service).
2. Start a Peach Agent in the VM.
 - a. For Windows VMs,
 - i. Open a command Processor with administrative privileges.
 - ii. Navigate to the folder containing Peach, such as: `cd C:\peach`
 - iii. Launch the Peach agent: `peach -a tcp`
 - b. For Linux and OS X VMs,
 - i. Open a terminal.
 - ii. Navigate to the folder containing peach, such as: `cd ~/peach`
 - iii. Start the script for running the Peach agent: `./peach -a tcp`
3. Using the VMware menu on the host OS, select VM → Snapshot → Take Snapshot. The following dialog appears. The title bar identifies image used in the snapshot.



4. Fill in the following information in the dialog:

- Name: The name must be unique.
- Description: This is an optional field.

5. Click <Take Snapshot> to record the VM state.

IMPORTANT: Save the following pieces of information. They are needed for automating the VM.



- Full path of the VM image used in the snapshot.
The VM image is stored on the file system. The VM image file extension is **.vmx**.
- Snapshot name.

11.2. Setting a Static IPv4 Address

While running automated fuzzing sessions, use this to ensure the IP address on the target system is the same every time it runs. Different operating systems have different ways of implementing it.

The information for Windows and Ubuntu Linux systems is provided. The workflow for other Unix-based systems, such as OS X, CentOS, and SUSE Linux, are similar.

11.2.1. Windows

This consists of changing the Internet Protocol Version 4 settings for the network adapter used in the test.

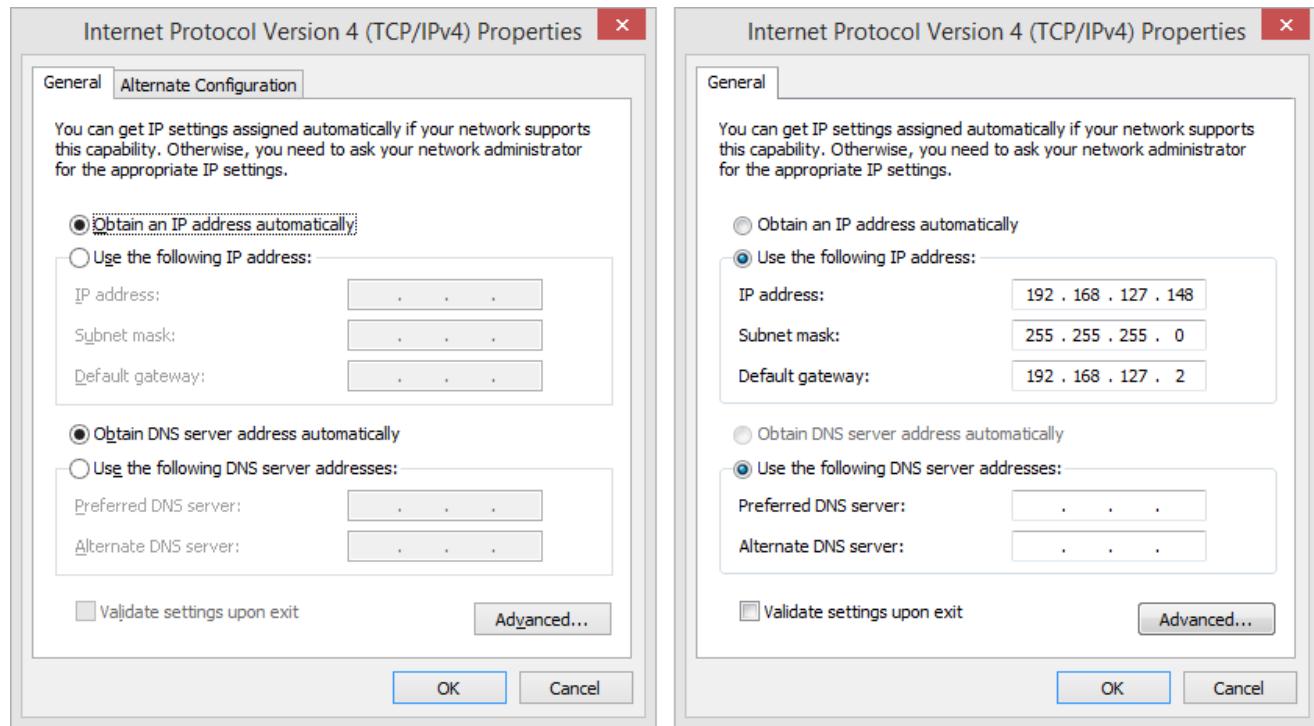
1. You can navigate to the IPv4 Settings using the following menu entries:

Control Panel → Network and Internet → Network Connections

- Select the appropriate network adapter.
- Right-click and select Properties from the shortcut menu.

4. Select the Internet Protocol Version 4 (IPv4) entry and click "Properties".

The following properties dialogs for the IPv4 protocol display settings for a dynamic IP assignment on the left. The static values are displayed on the right. Values specified for the static IPv4 address are fictitious. Use the values provided by running the `ipconfig -all` command in a command processor.



Click "Use the following IP address", and fill in the values for the IP address, submask, and gateway, as well as the entries for DNS servers.

5. When finished, click OK.

Linux

The change to use a static IPv4 address is similar to the change made to Windows systems. The biggest difference is adding a step to restart the network service after the changes.

The changes are made to entries in the `/etc/network/interfaces` file. Note that root privileges are needed to update this file.

1. Configure the network interface

Initial settings (Before change)

```
auto lo
iface lo inet loopback
iface eth0 inet dynamic
```

- a. Change the first line of the interface entry (here, the `eth0` interface) by replacing `dynamic` with `static`.
- b. Supply the address of the interface to the entry listed by `ifconfig`.
- c. Supply the gateway address that is the address of the router.

Edited settings (Changes applied)

```
auto lo
iface lo inet loopback
iface eth0 inet static
```

```
    address 192.168.1.101 +
    netmask 255.255.255.0 +
    gateway 192.168.1.1 +
```

2. Save the changes.
3. Configure the DNS server.

The DNS name servers need to be updated.

- If your Ubuntu system is 14.04 or greater, add the following line to `/etc/network/interfaces`:

```
dns-nameservers 8.8.8.8
```

- For Ubuntu systems earlier than 14.04, the update DNS server update occurs in the `/etc/resolv.conf` file. Add the IP addresses for the servers using the following syntax.

```
nameserver xxx.xxx.xxx.xxx
nameserver xxx.xxx.xxx.xxx
```

+ Example: Using the Google DNS servers, these lines would be: `nameserver 8.8.8.8`
`nameserver 8.8.4.4`

4. Save the change.
5. Restart the network service.

As with the DNS servers, 2 permutations exist. **For Ubuntu systems version 14.04 and greater, use the following command:**

`systemctl restart ifup@eth0` For Ubuntu systems with a version less than 14.04, use the following

command:

```
sudo /etc/init.d/networking restart
```