

FORTRA

beSTORM
13.2.0
User Guide

Copyright Terms and Conditions

Copyright © Fortra, LLC and its group of companies. All trademarks and registered trademarks are the property of their respective owners.

The content in this document is protected by the Copyright Laws of the United States of America and other countries worldwide. The unauthorized use and/or duplication of this material without express and written permission from Fortra is strictly prohibited. Excerpts and links may be used, provided that full and clear credit is given to Fortra with appropriate and specific direction to the original content.

202403040740

Table of Contents

Introduction	1
System Requirements	2
Hardware	2
Software	2
Welcome to beSTORM Screen	3
Getting Started with beSTORM	4
Fuzzing	8
beSTORM Walkthrough	9
Interface Overview	12
Menu options	12
Project Settings	19
Module Browser	20
Preview	20
Test Information	21
Test Progress	22
Exception Information	22
Conclusion Screen	23
Auto Learn	25
Network File Specification Auto Learn	25
Generation (Editing the output into a beSTORM module):	28
Environment Variables	32

Module Buffer Types	33
Custom Modules	35
Configuration Elements	36
Words	37
Bits	54
Sentences	55
Internal functionality	60
External functionality	65
Load a custom module	116
beSTORM Monitoring	118
Overview	118
Microsoft Windows monitoring	127
Linux monitoring	129

Introduction

beSTORM represents a new approach to security auditing. It's essentially a fuzzing framework that can be used for securing in-house developed applications and devices, as well as applications and devices of external vendors. Vendors can use beSTORM to test their products in a certification test or as part of the development life cycle. It provides the ability to customize existing modules and add new modules for testing all in an intuitive and easy to use environment.

Although beSTORM is a generic fuzzing framework, no programming skills are necessary to use it. It is especially useful for testing standard protocols – HTTP, POP3, SMTP, SIP and similar protocols with an RFC definition as well as standard file types – BMP, TGA and similar. Of course, you can always define your own testing modules to test proprietary protocols.

NOTE: Although most of the examples in this document will refer to network protocols, beSTORM is certainly not limited to testing network protocols alone. Network protocols can be seen as “recipes” to building anything from an HTTP protocol description traffic to a JPEG file description

System Requirements

The following are the hardware and software system requirements for beSTORM:

Hardware

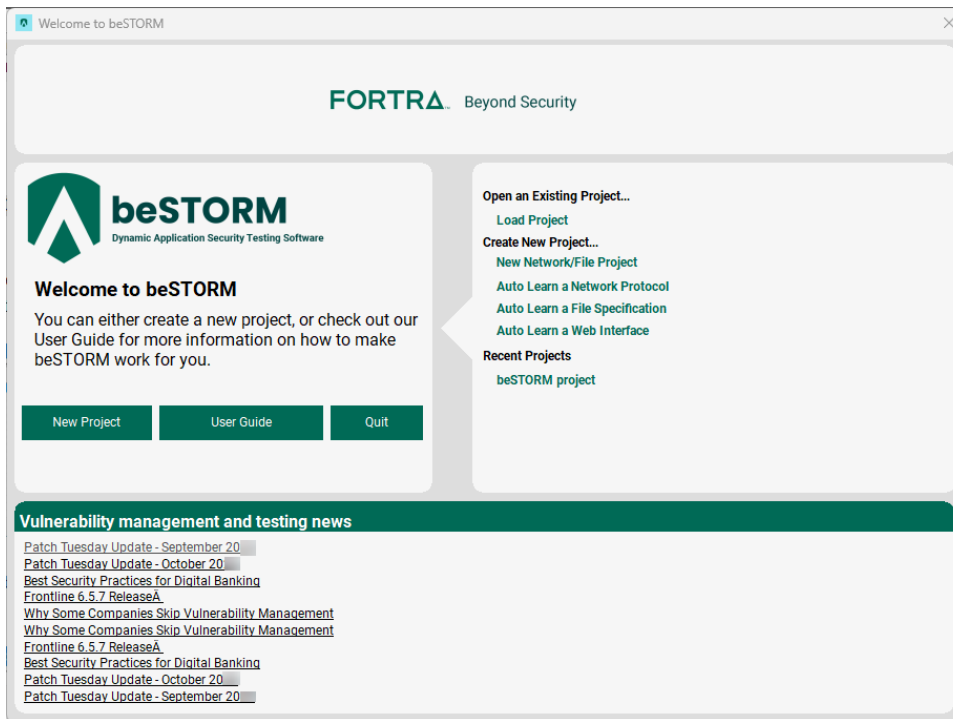
Hardware component	Minimum requirements	Recommended requirements
Processor	x86-64 processor	Quad-core processor (Intel i5+ or equivalent)
Memory	1 GB RAM (Linux, Docker, Embedded application)	8 GB RAM (Windows 10)
Hard drive	250 MB available hard drive space	1 GB available hard drive space

Software

Operating system	beSTORM version(s)
Windows 10 or later	13.1.0 or later
Kali (rolling)	11.6.27, 10.9.16, 10.7.23
Ubuntu 20.04	11.5.23
Ubuntu 18.04	10.10.19

Welcome to beSTORM Screen

When you first launch beSTORM, you are presented with a Welcome to beSTORM screen, from which you can begin working with beSTORM and utilize its different features.



The welcome screen allows you to either open up an existing project, by way of Load Project, create new projects, by way of New Network/File Project, teach beSTORM about new standards, by way of Auto Learn a Network Protocol or by way of Auto Learn a File Specification and finally use beSTORM to test your product's API, by way of Auto Learn a Web Interface.

In addition, the welcome screen provides shortcuts to previously loaded beSTORM projects.

Getting Started with beSTORM

beSTORM utilizes a wizard interface that guides you through the process of configuring beSTORM's testing session, fuzzing optimizations, and monitoring capabilities.

To begin your fuzzing session:

1. Open **beSTORM Client**.
2. Select **New Project**.
3. On the **Welcome** page, configure the following:
 - a. **Project Name** - Enter a name for the project, or use the default name provided.
 - b. **Location** - Browse to a location to store the project and its files, or use the default location provided.
 - c. **Wizard level** - Select **Simple** to use pre-configured settings, or select **Advanced** to manually configure those settings yourself.
 - d. **Perform a port scan, and service detection and assist me in choosing the relevant module** - Select or clear this setting, based on your preference.
4. Select **Next**.
5. On the **Basic Configuration** page, select a module for your project:
 - a. **beSTORM's predefined modules** - Predefined modules such as; BMP, GIF, FTP, HTTP, POP3, etc.
 - b. **Import a Custom Module from a BSM File** - A custom module you can import.
 - c. **Build a Network Module** - Create a new Network-based module utilizing beSTORM's network auto learning capabilities.
 - d. **Build a File Module** - Create a new File-based module utilizing beSTORM's file auto learning capabilities.
 - e. **Build a Web Application Module** - Create a new Web Application module utilizing beSTORM's web testing capabilities.
 - f. **Build a CANBUS Module** - Create a new Controller Area Network (CAN bus) module utilizing beSTORM's ability to read and process CAN DBC files.

Depending on your selection, the **Hostname or IP address** (default is **127.0.0.1**), **Protocol** (default is **udp**), and **Local Port** (default is **67**) parameters are preset. If you select a predefined module (which do not require network configuration) or a new File, Web Application, or CANBUS module, the Target Host Settings section of the wizard will not appear.
6. Select **Next**.

7. If you selected **Advanced** in step 3c, the **Advanced Configuration** page will appear providing optimizations and options based on the module selected in step 5 (*if you selected **Simple** in step 3c skip to step 11*). The available options are:
 - a. **Optimizations** - Different modules support different settings; for example HTTP testing do not run multiple Parallel Attack Threads, while modules such as SMTP do. This depends on the type of server being tested, the robustness of the protocol to parallel testing, and other considerations.
 - b. **Run in batch mode** - If selected, beSTORM continues running after the first fault is found.

NOTE: In this case, the product being tested should recover from the previous fault, either by being restarted or by some other way.

- c. **Make sure monitor is up before starting** - If selected, beSTORM waits for an agreed signal from the tested environment before beginning the test. This allows beSTORM to be certain that the tested environment is running properly.
- d. **Report connectivity issues as exceptions** - If selected, beSTORM treats a case of receiving no network traffic from the tested environment as potential problems or vulnerabilities. This feature is useful while testing an environment that is not easily monitored (such as a proprietary hardware device) as it marks all network problems as a potential vulnerability. By doing so, it allows to easily reproduce the issue at a later time and discover the cause.
- e. **Periodically test connection and report vulnerability upon failure** - If selected, this option tests the behavior of the product being tested and expects it to answer traffic that is not malformed in a normal manner. If the product does not respond, beSTORM reports an exception.

Certain modules require additional information to performing proper testing. One such example is in the case of the FTP module, a username and password are needed for FTP login, if you want to test all the available commands.

NOTE: beSTORM can not log in without a proper username and password.

8. Select **Next**.
9. On the **Module Environment** page, some of the fields appearing here are automatically populated by values that were previously defined (for example, Remote Hostname, Remote Port, and Remote Protocol Type). Other items to note:
 - a. To change the username or password value, double-click on the **Value** field just right of the Descriptive text Username for FTP login and Password for FTP login respectively.

- b. The **Required** column indicates which parameters must contain an actual value. If the protocol contains such parameters, beSTORM prompts you to supply values, otherwise values are assigned by default.

10. Select **Next**.

11. On the **Extra Configuration** page, configure these test settings:

- a. To adjust the speed of your test to be a fixed number of sessions per seconds, select a value for **Saturation Rate Threshold** and leave **Fixed Saturation Rate Threshold** selected.
- b. To allow your test's speed to be automatically adjusted according to reports from the beSTORM monitor, select **Auto Adjust - Optimize CPU usage**.
- c. To configure the monitoring communication settings, optionally select from any of the following monitor options:

- i. **ARP Echo** – Attempts to resolve the IP address of the machine tested into a MAC address.

NOTE: ARP Echo properly works on LAN in a WAN environment where the target is not on the same network/subnet class. An ARP response will be received from the Router that connects the two networks, thus causing a false status.

- ii. **ICMP Echo** – Attempts to perform an ICMP Echo/ICMP Response test on the remote IP address.
- iii. **UDP Echo** – Attempts to verify whether the remote UDP port is open.

NOTE: For UDP to be properly detected as non-responsive/closed, the Windows Firewall has to allow ICMP Destination Unreachable packets to arrive. By default, Windows Firewall blocks such packets from arriving.

- iv. **TCP Echo** – Attempts to verify whether the remote TCP port is open.
- v. **External Monitor**- The Fortra's Beyond Security provided monitor, or your own custom monitoring device/program.
Defined what is the IP address of the machine you would like to perform ARP, ICMP, UDP, or TCP monitoring, by providing a value to the Monitored IP address field, and if you are utilizing UDP or TCP, define what port you would like to monitor by providing a value to the Port field just right from the Monitored IP address field.
- d. Enter a port number for the **Incoming Command Port** parameter. This parameter is for receiving commands from the monitor (such as reports on the tested machine's load and status).

- e. Enter a port number for the **Incoming Exception Port** parameter. This parameter is for Exception Data being sent from the monitor to beSTORM.
 - f. Enter a port number for the **Outgoing Command Port** parameter. This parameter tells beSTORM which port to use to establish that communication.
12. Select **Next**.
 13. To prevent beSTORM from automatically scanning after completing the wizard, clear the **Auto-start beSTORM scan now** check box.
 14. Select **Finish** to complete the wizard.

Fuzzing

beSTORM starts its fuzzing as soon as you select **Start**.

The fuzzing sequences are deterministic and can be replayed by telling beSTORM to start from the beginning, or from any other particular attack vector (position) you provide to beSTORM.

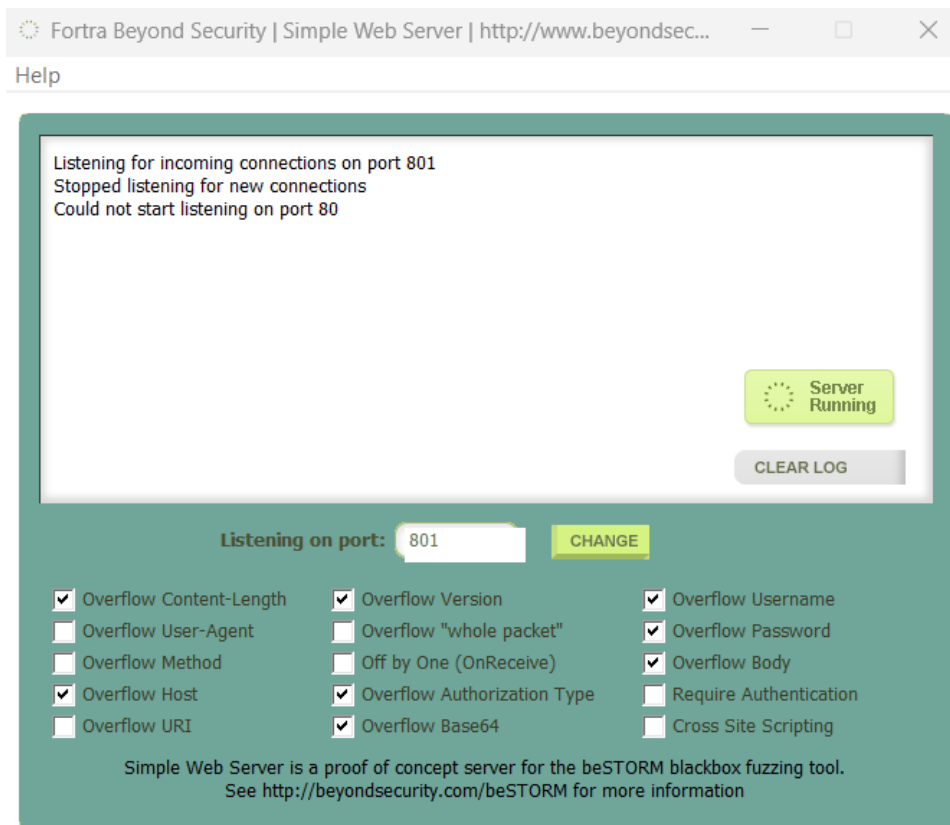
You can monitor beSTORM's progress by viewing the Progress Information section by selecting **Preview**. This displays the dataset currently being sent by beSTORM to the tested product, or you can look up the Detailed Log to view the current speed beSTORM is testing the product.

Even though beSTORM has predefined buffers which it fuzzes, you have complete control over the types of data it fuzzes and the type of data it generates (for example, long buffers, overflowing integers, etc.). Changing these predefined buffers, or even adding additional buffers, can greatly enhance the performance and the usability of beSTORM as it allows it to find more exceptions quickly, as well as find exceptions that might be specifically relevant to your product.

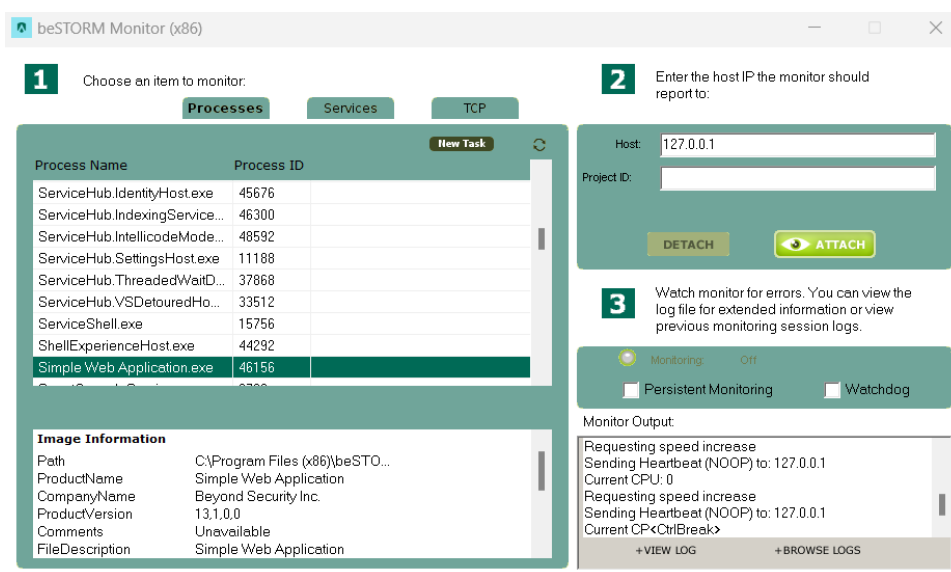
beSTORM Walkthrough

To demonstrate the initial stages of running beSTORM, the example below has a web (HTTP) server that has been tainted with numerous vulnerabilities such as: Overflow by way of Method, Overflow by way of URI, Overflow by way of User-Agent, Off-by-One in Receive, Overflow in Base64 decoded content, and others.

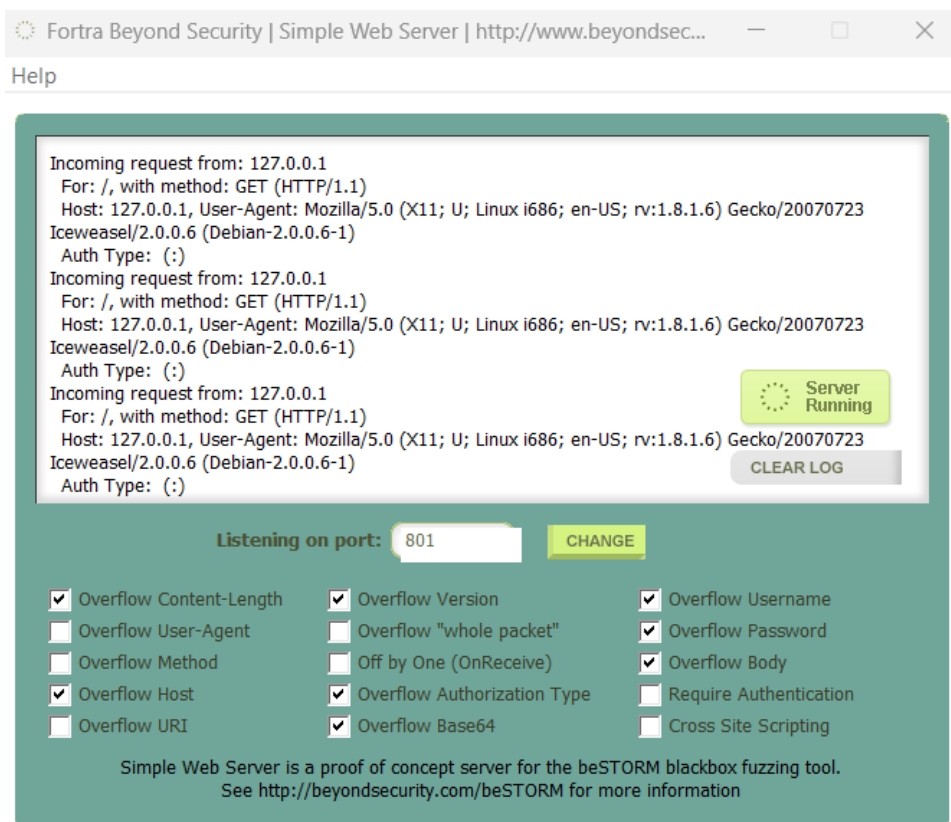
To use the web server, launch the executable and the program automatically starts to listen on the desired port.



To monitor the port and its status, use the GUI version of the monitor, shown below:



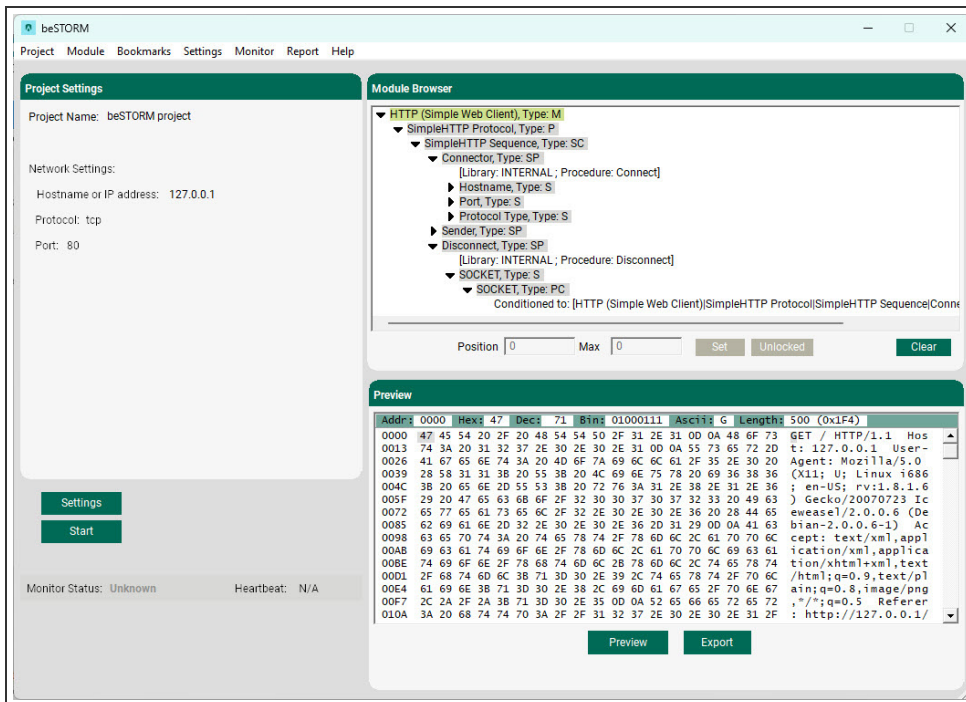
The Simple Web Server process (Simple Web Server.exe) has been selected from the process list on the left side, and a host to report to on the right top side (127.0.0.1). To begin monitoring, click **Attach**. The Simple Web Server automatically begins answering incoming requests:



You can quickly test whether the program crashes by sending the following request:

Interface Overview

The beSTORM user interface consists of three sections. Initially, the sections shown are; Project Settings, Module Browser and Preview. When a test is in progress two sections change. The Project Settings is replaced with Project Information and Module Browser with Progress Information.



Menu options

Project

- **New** - Starts a new project by way of beSTORM's wizard.

Under **New**, you can see the different types of projects that can be created using the wizard. Choose the one relevant to your test scenario.

NOTE: Learn more about beSTORM's wizard from the [Getting Started with beSTORM on page 4](#) section.

- **Open** - Loads an existing beSTORM project.
- **Save/Save as** - Saves your currently loaded beSTORM project.
- **Recent Projects** - Loads one of the recently loaded projects.

- **Load Last Saved** - Reverts any changes done for the current module to the last saved settings. The settings include changes done to the module configuration, as well as changes made to the beSTORM environment.
- **Auto Learn** - Selecting **Network Protocol** or **File Specification** opens the **Auto Learn** window.

NOTE: Learn more about beSTORM's Auto Learn feature from the [Auto Learn on page 25](#) section.

Module

- **Load from Attack Vector** - Manually instructs beSTORM to begin from a different starting position than the one beSTORM is currently at. In the case that the attack vector provided to beSTORM is invalid, an error message will be shown.
Attack vectors can be obtained from the Preview Dialog, beSTORM's running screen, beSTORM's log files, exported test cases or the Exception Information.
- **Find Attack Vector** - Clicking on this menu item allows you to browse through previously ran sessions of beSTORM and locate a specific Attack Vector based on a certain date value. For example if you have been running beSTORM for the past 3 days, and would like to know which Attack Vector (position) it was running 2 days ago at midnight, opening this menu will give you a list of all the log files generated by beSTORM. Simply locate the log file relevant to that time period and select the Attack Vector of interest.

NOTE: Attack vectors are the textual representation of the state at which the beSTORM's testing is currently at. An attack vector shared between beSTORM copies that test the same protocol will bring beSTORM to the same testing position.

- **Browse Full Screen** - Clicking on this menu item opens up a new window, similar in its content to the Module Browser section shown in the main window. The only difference between this new window and the one found in the main window is the fact that the new one's size can be altered.
- **Show Graphical Representation** - Clicking on this menu item instructs beSTORM to generate a graphical view of the module currently being used. The graphical representation shows how each of the module's elements are interconnected and in addition what elements are currently active.

NOTE: beSTORM saves the graphical representation as a Graphviz file (.dot extension). You will need an appropriate viewer in order to view the graphs, for example: <http://www.webgraphviz.com/>.

- **Edit Module Environment Variables** - Clicking on this menu item allows you to change the current environment settings of beSTORM.

NOTE: Learn more about beSTORM's Environment Variables feature from the [Environment Variables on page 32](#) section.

- **View Module Buffer Types** - Clicking on this menu item allows you to view the current buffers being used by beSTORM to detect exceptions.

NOTE: Learn more about beSTORM's Module Buffer Types feature from the [Module Buffer Types on page 33](#) section.

- **Increment Module's Position** - Clicking this menu opens it's sub items, allows you to cause the module currently being set in beSTORM to move forward by a set number of positions. This can be used both for testing purposes as well as skip on undesired attack vectors.
- **Simulate** - Clicking on this menu item tell beSTORM to perform one test, according the current position of the module. For example, in Network Protocol based project, beSTORM would send one packet, representing the current position of the beSTORM module.

Bookmarks

This menu allows you to tell beSTORM to set the module to a previous, recent configuration. This allows you to tell beSTORM to go back an amount of time and replay an entire attack sequence to assist you in reproducing issues discovered by beSTORM.

Settings

This menu provides access to beSTORM's settings.

Configure beSTORM

- **Project Name** - Specifies the name of the current project. The project name is editable.
- **Number of Parallel Attack Threads** - Specifies the number of threads to use during a test. Running a test with multiple threads increases its speed, especially when beSTORM modules wait for a response.
- **Environment Settings** - Dynamically displays environment settings for the current module. Settings can vary, depending on the module.

Configure Advanced Settings

- **Starting Saturation Rate Threshold** - Specifies the starting saturation rate threshold, which determines the number of tests sent per second. Slide the control to increase the value. The default value is 100.
- **Scale Type** - Optimizes testing by specifying the number of combinations sent per [Module Buffer Type](#). Each Scale Type alters the **Estimated combination count per Buffer** number. The available options are:
 - **Base2+/-2** - Sends buffer combinations by +/-2: 2, 4, 6, 8, 10, 12, 14, 16, etc.
 - **Base2+/-1** - Sends buffer combinations by +/-1: 0, 1, 2, 3, 4, 5, 7, 8, 9, 15, 16, 17, etc.
 - **Base2** - Sends buffer combinations by 2, 4, 8, 16, 32, etc.
 - **Base10+/-2** - Sends buffer combinations by +/-2: 10, 100, 1000, 10000, etc.
 - **Base10+/-1** - Sends buffer combinations by +/-1: 10, 100, 1000, 10000, etc.
 - **Base10** - Sends buffer combinations by 10, 100, 1000, 10000, etc.
 - **Serial** - Sends buffer combinations by 1, 2, 3, 4, etc.

NOTE: The Serial type is extremely time consuming. Only use this type if your test has no time constraints.

- **Timed** - Select this type if you have time constraints for your test to run (that is, you can only run beSTORM for 1 hour, 10 hours, 1 day, etc.), but want to test all fields regardless. The Timed type spends one second on each field in the first loop (changing the buffer types as usual, but stopping after one second) covering the entire protocol quickly. Then, on the second loop, it spends two seconds on each field, then four seconds, eight seconds, etc., until the allotted time expires. beSTORM will incrementally test more and more of each field until you stop the test manually.
- **Serial/Base2** - Combines the Serial and Base2 types, providing an intermediate option that generates more combinations than Base2, but less than Serial. Sends buffer combinations by 1, 2,...4095, 4096, 8192, etc.
- **Increment Order** - Determines the order the module will use to test buffer sizes. The order does not affect the combination count or speed of the test.
 - **Normal** - Starts with small buffer sizes (for example, 2, 4, 8, 16, etc.) and increases in size as the test runs. This order can possibly find vulnerabilities more precisely as the smallest attack will trigger an issue.
 - **Reverse** - Starts with larger buffer sizes (for example, 2,000,000) and decreases in size as the test runs. This order can possibly find vulnerabilities earlier in the test.

- **Distributed Testing** - Combines the **Number of beSTORM copies available** and **beSTORM copy number** settings to allow multiple copies of beSTORM to be in use and testing against the device under test (DUT). While working together, each copy can do 1/n of the tests.

For example, if you run two copies of beSTORM in parallel, one copy will do half of the test, and the other copy will do the other half. The two values in this case would show **Number of beSTORM copies available** as **2** and **beSTORM copy number** as **1** in one copy of beSTORM, and **beSTORM copies available** as **2** and **beSTORM copy number** as **2** in the other copy of beSTORM.

- **Overflow buffers only once** - Prevents testing a field in more than one combination. Selecting this setting can reduce testing time. This setting is disabled by default.
- **Allow Fuzzing of conditioned values** - Fuzzes conditioned values (for example, length) as regular fields. Disabling this option only tests these values for logical issues (that is, too large length, too small length, negative length, and zero length) and reduces testing time. This setting is selected by default.
- **Debug function in/out to log files** - Selecting this setting instructs beSTORM to log additional debug information into a file (for example, received and sent data, function calls (that process the data), etc.), but doing so will severely impact its performance. This setting is disabled by default.

Configure Behavior Settings

- **Interface refresh rate (seconds)** - Specifies the user interface refresh rate. The default value is 1 as this is sometimes a labor intensive process, but increasing the value slows down the refresh rate of user interface, which is ideal when beSTORM is run in batch mode and user interaction is expected.
- **Saturation Rate Threshold Optimization** - Specifies how your testing speed is determined.
 - **Auto Adjust - Optimize CPU usage** - Runs your test as quickly as possible, utilizing up to 75% of available CPU bandwidth on the local machine, based on reports from the beSTORM monitor.
 - **Fixed Saturation Rate Threshold** - Sends a fixed number of tests per second, based on the **Starting Saturation Rate Threshold** setting under [Configure Advanced Settings on page 14](#).

NOTE: beSTORM will attempt to reach and stay at this speed during the test, but the speed may fluctuate at times.

- **Send SMTP (Email) Notifications** - To send email notifications to contacts when an event in beSTORM occurs during a fuzzing session, enter the following email information:

- **From** - The sender's email address to use with email notifications.
- **To** - The email address(es) to send email notifications to (use a comma to separate multiple email addresses).
- **SMTP Server** - The IP address of the SMTP server.
- **SMTP Port** - The port number of the SMTP server boxes.
- **Notification Types** - After entering email addresses and SMTP information, select which types of notifications to send when the corresponding event occurs:
 - **Test Started** - When fuzzing starts.
 - **Test Paused** - When fuzzing is paused.
 - **Tested Ended** - When fuzzing ends.
 - **Test Error** - When fuzzing experiences an error.
 - **Test Failure** - When fuzzing fails.
 - **Exception Found** - When an exception is found during fuzzing.

Configure Monitor Settings

- **Enable Batch Mode** - Instructs beSTORM to run in non-interactive mode. In this mode, beSTORM will automatically start, run a test, if an exception is found the test will automatically resume as soon as the device under test responds, and then automatically closes beSTORM once testing is done. This setting is selected by default.
- **Monitor Port Assignment** - The beSTORM counterpart for testing is a monitor that either resides on the same computer as the beSTORM Client, or on a different server. Change the default port numbers, if necessary.
 - **Hostname or IP address** - The hostname or IP address of the monitor.
 - **Incoming Command Port** - Receives responses from the monitor to the beSTORM Client. The default port number is 6970.
 - **Outgoing Command Port** - Sends information from the beSTORM Client to the device under test. The default port number is 6971.
 - **Incoming Exception Port** - Sends exceptions received by the monitor to the beSTORM Client. The default port number is 6969.
- **Enable Monitor Enforcement** - Instructs beSTORM to not start or conduct any test until the monitor counterpart reports that it can monitor the device under test.
- **Monitor Type(s)** - Specifies the provided monitor type(s)/external monitor to use to verify the remote device under test is functioning by communicating with it using the respective protocol (ARP, ICMP, UDP, and TCP). The available options are:

- **ARP Echo** – Attempts to resolve the IP address of the machine tested into a MAC address.

NOTE: ARP Echo works on LAN in a WAN environment where the target is not on the same network/subnet class. An ARP response is received from the Router that connects the two networks, thus causing a false status.

- **ICMP Echo** – Attempts to perform an ICMP Echo/ICMP Response test on the remote IP address.
- **UDP Echo** – Attempts to verify whether the remote UDP port is open.

NOTE: To properly detect UDP as non-responsive/closed, the Windows Firewall must allow ICMP Destination Unreachable packets to arrive. By default, Windows Firewall blocks such packets.

- **TCP Echo** – Attempts to verify whether the remote TCP port is open.
- **External Monitor**- The Fortra's Beyond Security provided monitor, or your own custom monitoring device/program.
- **Monitored IP address** - The IP address of the machine to perform monitoring on.
- **Port** - The port number of the external monitor (UDP Echo and TCP Echo only). The default value is 1.
- **Interval** - The interval to verify the remote device in milliseconds. The default value is 5000.
- **When exception is detected, stop the test for <#> seconds** - Specifies the number of seconds to stop the test when an exception is detected. allowing you to take note of it. The default value is 10.
- **Report Connectivity Issues as Exceptions** - Select this setting to report connectivity issues with the remote device as an exception. This setting is disabled by default.
 - **Number of connectivity failures before reporting back** - Specifies the number of failures that need to occur before connectivity issues are reported while **Report Connectivity Issues as Exceptions** is selected. The default value is 10.
- **Test Fuzzed files by calling beSTORM's Minion** - Select this setting to use the beSTORM Minion to test files (for example, DLLs). Refer to <https://www.beyondsecurity.com/testing-dll-api-fuzzing-with-bestorm> for more information. This setting is disabled by default.

The Minion requires the following:

- **beSTORM Minion IP address** - The IP address of the Minion.
- **Port** - The port number to use with the Minion.
- **beSTORM Minion Password** - The password to use with the Minion.

- **Process to Launch (Full Path)** - The full path of the process to launch (for example, when testing files are part of an application).

Monitor

- **Check Monitor Status** - Tells beSTORM to actively connect to the monitor and check its status. This can be used to determine whether the tested environment is running properly.
- **Configure Monitor Settings** - Configures the beSTORM monitor settings. See separate section on beSTORM settings for more details.

Report

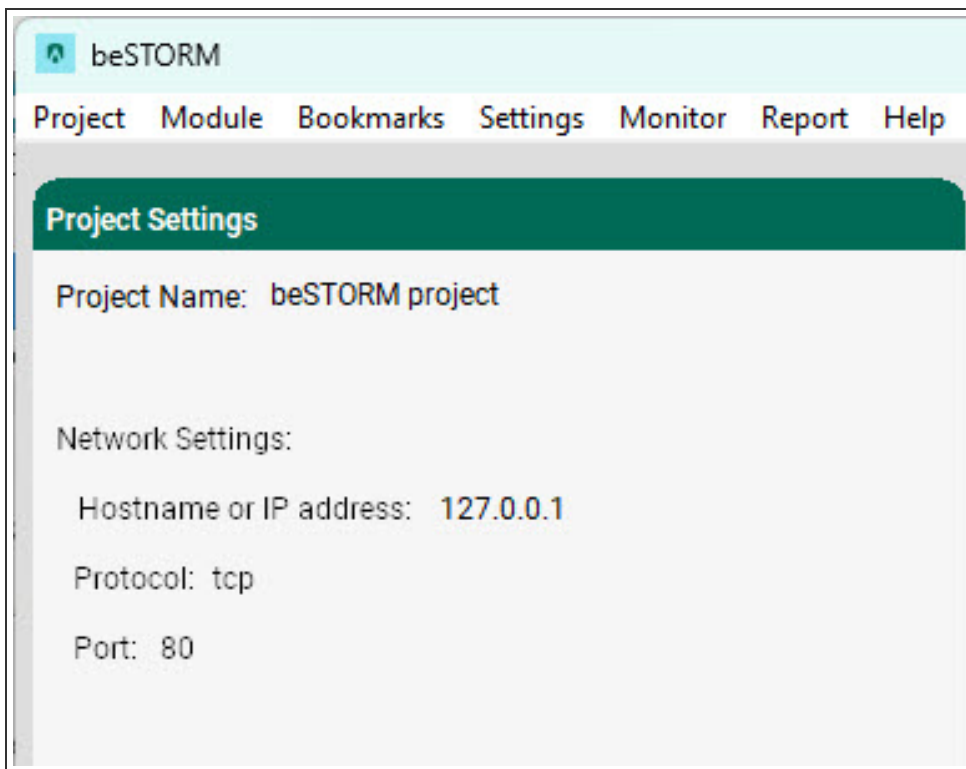
- **Generate Report** - Generates a report in HTML, PDF, or CSV format on the currently loaded beSTORM project, including test and result information.

Help

- **User Guide** - Opens the beSTORM User Guide.
- **beSTORM Architecture** - Displays the current beSTORM project's testing architecture.
- **About** - Displays the current beSTORM version number and benchmark information.
- **License** - Displays information regarding your beSTORM license.

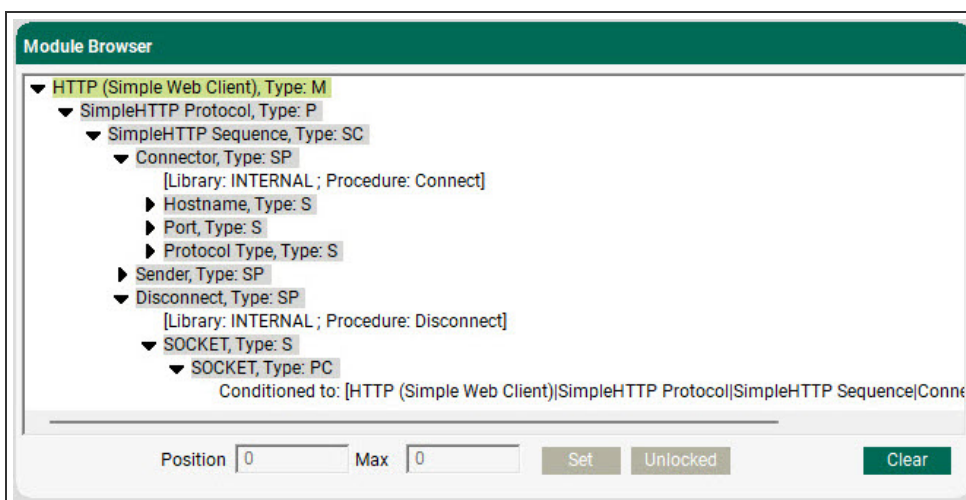
Project Settings

The Project Settings displays the current project's settings; project name, number of parallel processes used when running the test, and when the project is network related, the Hostname or IP address, Port and transport Protocol being used for testing:



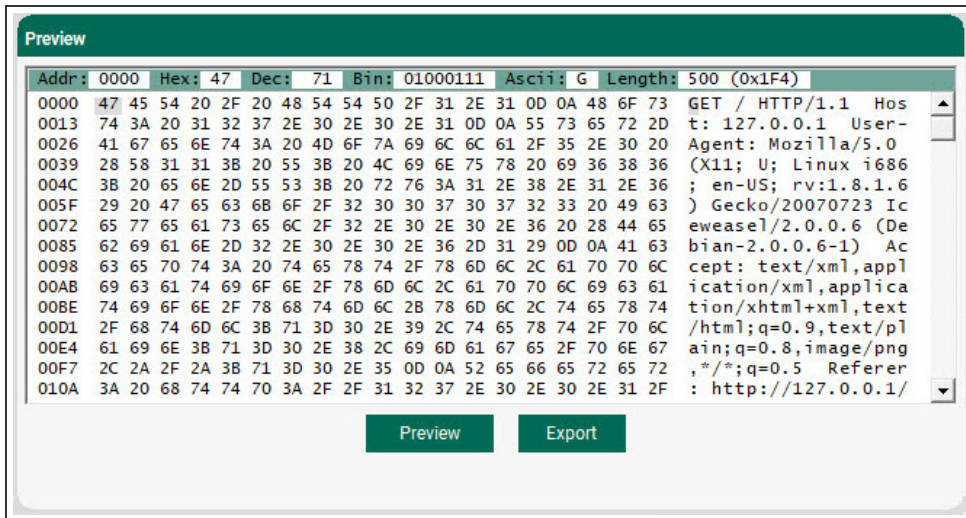
Module Browser

The Module Browser allows you to view the current module configuration, as well as control the testing behavior. By locking and setting the position of the module, you can direct beSTORM to test certain sections, while ignoring others.



Preview

The Preview screen allows you to see the current position of the module. You can also export the current content of the Preview screen into a file for further manual testing:



Clicking the **Export** button generates a platform-independent Perl script that imitates the behavior of beSTORM, and provide the same sequence of events that have caused the shown data to be sent to the remote server.

Test Information

After starting beSTORM by clicking **Start**, the Test Information section is shown which allows you to monitor the progress of the current test, identify what is being tested, and what is currently being fuzzed. In addition, as soon as a vulnerability is detected, the counter increases and clicking on it displays the attack vector that has triggered the problem, as well as the outcome of the problem:

Test Information

Host: 127.0.0.1 Port: 80

Started: 13:10:23 12/11/

Elapsed: 0 days 0 hours 0 minutes 19 seconds

Total elapsed: 0 days 0 hours 1 minutes 30 seconds

Time remaining (estimate): N/A

Session combinations: 0

Total combinations: 0

Testing: Simple GET HTTP Packet

Fuzzing: If-Modified-Since value (Format (textual))

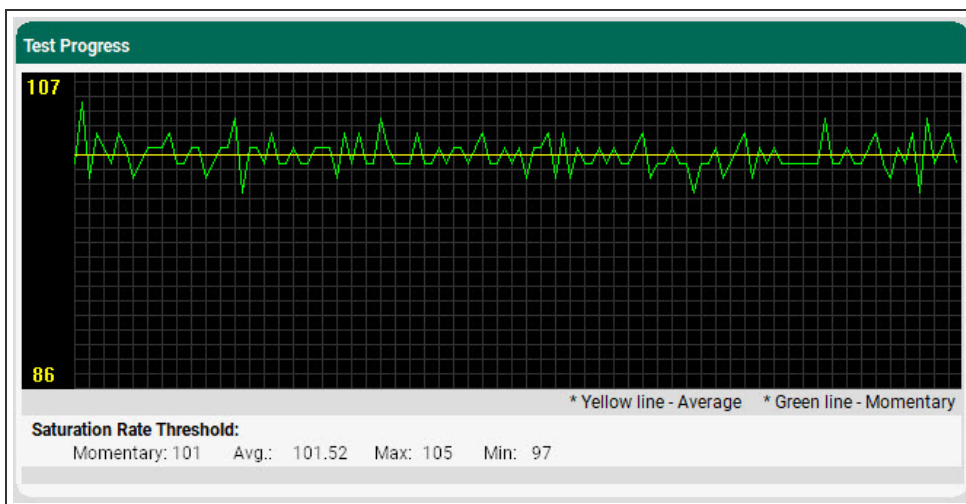
Suspected vulnerabilities detected: 0

Attack vector: [\(Preview...\)](#) Thread count: 1 of 1

M0:P0:B0.BT0:B0.BT0:B0.BT0:SE0:B0.BT0:B0.BT0:B0.BT0:B0.BT0:
 B0.BT0:B0.BT0:B0.BT0:B0.BT0:B0.BT0:B0.BT0:B0.BT0:B0.BT0:B0.
 BT0:B0.BT0:B0.BT0:B1.BT1

Test Progress

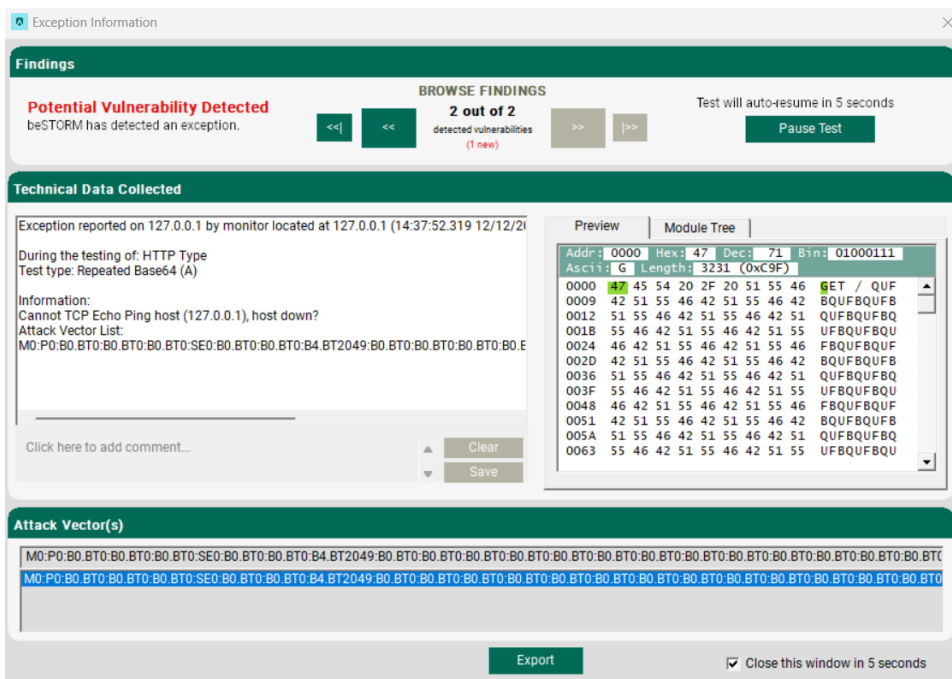
After starting beSTORM by clicking **Start**, a Test Progress section is shown which allows you to monitor the progress of the current test, see the momentary and average speed of the test (measured as "saturation rate threshold" meaning number of tests per second):



Exception Information

beSTORM is able to detect a variety of issues in products, the issues or exceptions, are referred to as such as they are unexpected behavior being manifested by the product being tested. Whenever beSTORM tests a network protocol, file specification or API calls, the exceptions beSTORM looks for are any behavior that caused the software being tested to crash, overflow an internal buffer, access information that it would otherwise not be able to access, manipulate or change items it shouldn't have access to, etc.

For other types of testing, such as in the case of Web Applications, exceptions are not limited to those just mentioned, but also include problems which manifest as cross site scripting, SQL injection, code injection and command execution vulnerabilities. The following example of the Exception Information screen displays information relevant to the exception related to a network protocol.



The Exception Information screen displays information relevant to the exception that was detected. Whenever possible, a stack dump as well as the trigger for the exception is displayed at the Exception data section of the dialog box, and in any case an attack vector is displayed at the bottom of the screen. Once the product being tested recovers, beSTORM will continue its testing until it has exhausted all possible attack vectors, if the product does not recover beSTORM will halt and await further instructions.

Conclusion Screen

Selecting **Report** causes beSTORM to pause the test and show a temporary conclusion of the test up to this point.

The screenshot displays the 'beSTORM - Report' application window. The interface is divided into several sections:

- Test Information:** Shows 'Current state: Test Paused', 'Connectivity: OK!', 'Total Combinations: 9609', 'Suspected vulnerabilities detected: 4', and 'Elapsed Time: 0 days 0 hours 2 minutes 24 seconds'. It includes buttons for 'Settings', 'Export', 'Finish', and 'Resume'.
- Test Progress:** A tree view showing the 'SimpleHTTP Protocol' and 'SimpleHTTP Sequence'. Under 'Data', it lists various test cases with checkboxes: 'Method', 'URI', 'HTTP Type' (checked), 'HTTP Major' (checked), 'HTTP Minor' (checked), 'Host value' (checked), 'User-Agent value' (checked), 'Accept value' (checked), 'Referer value' (checked), 'Accept-Language value' (checked), 'Accept-Encoding value' (checked), and 'Accept-Charset value' (checked).
- Preview:** A table showing test data with columns for Address, Hex, Dec, Bin, Ascii, and Length. The data is mostly zeros, indicating a paused or failed test. Below the table are 'Preview' and 'Export' buttons.
- Monitor Status:** Shows 'Stopped!' and 'Heartbeat: N/A'. A log window below it contains timestamps and messages: '[14:36:36 12/12/23] Pausing ...', '[14:36:36 12/12/23] Starting ...', '[14:35:45 12/12/23] Exhausted waiting for monitor to restart. Did not resume test automatically ...', and '[14:35:34 666 12/12/2023] Exception caught...'. A 'Detailed...' link is at the bottom.
- Attack Vector:** A string of hexadecimal characters: 'M0:P0:B0:BT0:B0:BT0:B0:BT0:SE0:B0:BT0:B0:BT0:B6:BT2048:B0:BT0:B0:BT0:B0:BT0:B0:BT0:B0:BT0:B0:E'.

This screen includes general information about the test, including: reason for pause (error, user request, etc.), connectivity status, number of vulnerabilities detected and the number of combinations ran in the last session. On the right side of the screen, a list of the test cases included in the tested module, which were exhausted, not in process, and not started yet is displayed.

Select **Export** to generate a fully detailed HTML report that includes all data collected throughout the test process.

Select **Finish** to close the project and load the last saved status. You can resume the scan by selecting **Resume**.

Auto Learn

One of beSTORM's main strengths is the ability of the user to construct a complex protocol and have beSTORM fuzz all of its variations. Therefore, it is imperative that beSTORM provides the ability to create a module, or at least a basic module structure.

While it remains true that creating/extending a beSTORM module may be done using any XML editor and following the beSTORM grammar which is described later on in this manual (Custom Module section), this may become difficult if, for example, if the complete protocol specifications are missing, or if the module contains unknown/uninteresting sections).

beSTORM provides functionality that tries to resolve this difficulty. This by providing a simple to use graphic interface that allows the user to create beSTORM modules from provided data blocks.

This interface allows the user to load different types of data that describes the data sets we are interested in analyzing with beSTORM. beSTORM uses these data sets to determine how the protocol or file is built.

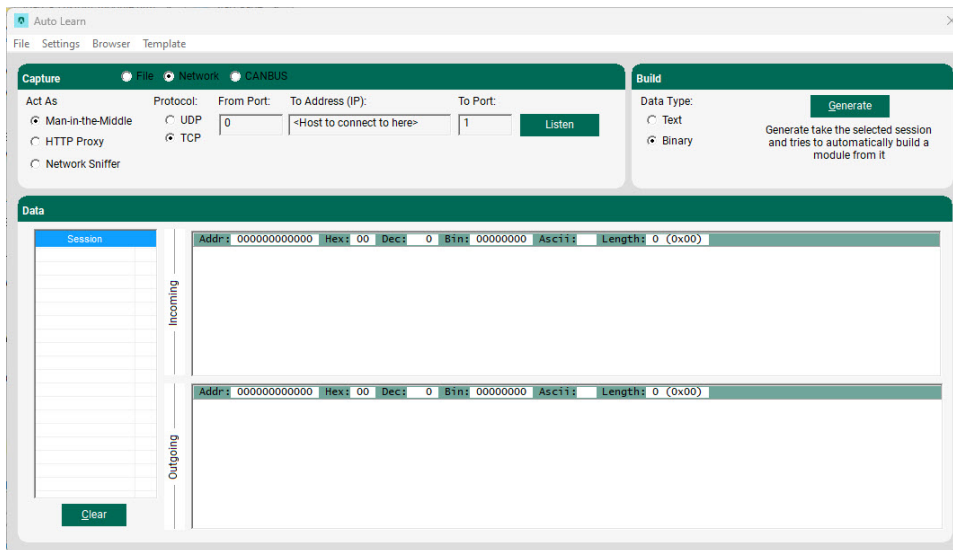
These input types can be network traffic that beSTORM captured, or a file sample containing the network traffic, which was captured using other means or a syntax describing an API you want to fuzz.

Once the input data is being loaded into beSTORM, you can analyze it in the relevant screen, depending on the type of data being used (Binary / Textual / API syntax).

Network File Specification Auto Learn

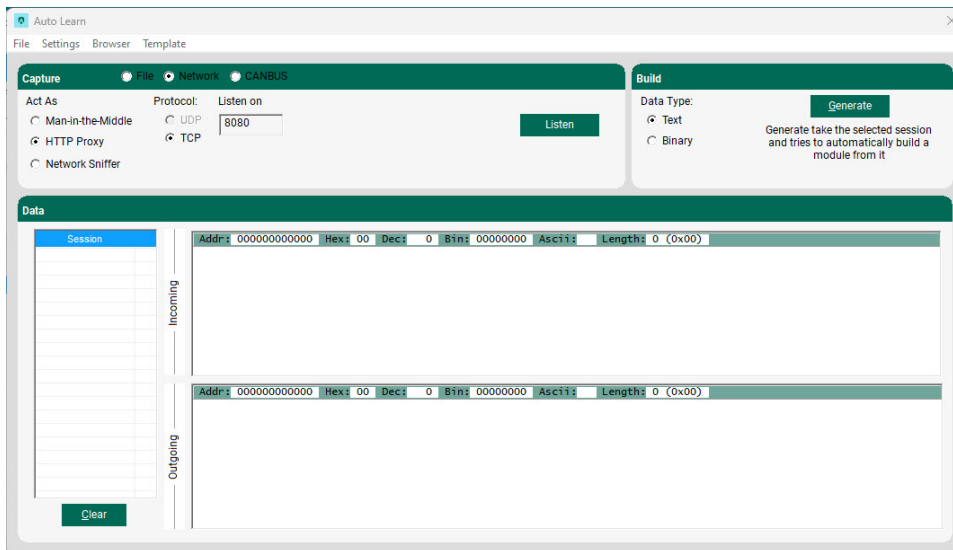
Upon choosing Network/File Auto Learn from the beSTORM menu, or by choosing Learn in the wizard, you arrive to a screen with three action options:

Capture network traffic using the Man-in-the-Middle option



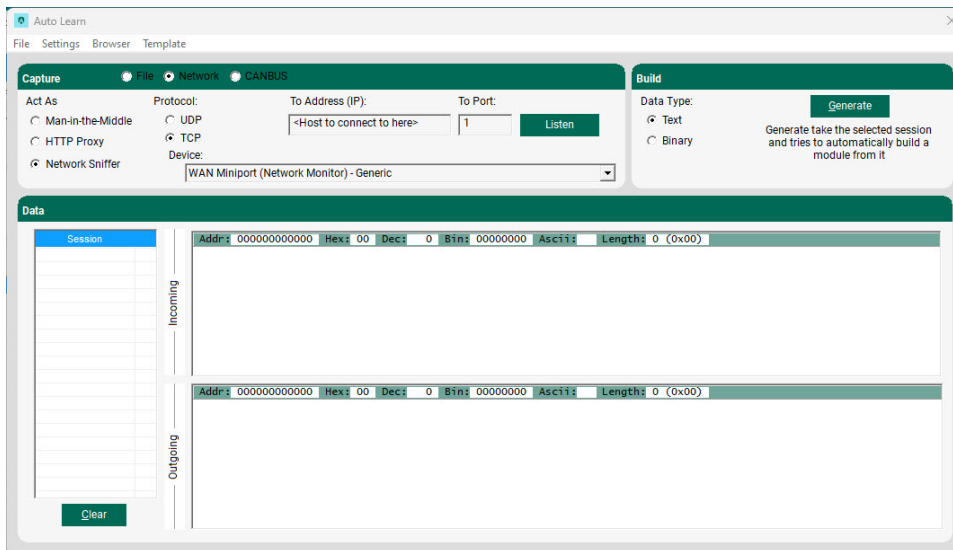
1. To set up this option, specify the incoming port that beSTORM will listen to and act as a server on, the protocol type (TCP or UDP), the IP address beSTORM and the destination port to relay to. Configure your 'Client'/Script to transfer network to the beSTORM machine's IP address and incoming port, as specified.
2. After selecting **Listen**, beSTORM starts to capture network traffic. Once data is being sent, beSTORM captures both incoming network traffic (from client/script) and the server response (referred to as outgoing traffic).
3. Once the collecting process is done, select **Stop**. At this point you can go over the captured data (divided into sessions) and choose which part of the traffic (incoming or outgoing) interests you.
4. Even though beSTORM attempts to determine whether the data it collected is binary or textual, make sure the correct option is selected and then click **Generate** to proceed to the editing stage.

Capture network traffic using beSTORM as an HTTP proxy of your web browser



1. To set up beSTORM as an HTTP proxy you need to specify the listening port, and then select **Listen**.
2. After configuring your web browser to connect by way of HTTP proxy with beSTORM's IP address and listening port (as specified earlier), start navigating the web. beSTORM will capture both requests and responses, which are divided into sessions.
3. Once you are satisfied with the captured session, select **Stop** and then select the relevant data.
4. Even though beSTORM attempts to determine whether the data it collected is binary or textual, make sure the correct option is selected and then select **Generate** to proceed to the editing stage.

Capture Network data using a Network Sniffer



1. To set up beSTORM as a Network Sniffer, specify the desired hostname/IP and port number to capture data from/to, and then select **Listen**. beSTORM captures both requests and responses, which are divided into sessions.
2. Once you are satisfied with the captured session, select **Stop** and then select the relevant data.
3. Even though beSTORM attempts to determine whether the data it collected is binary or textual, make sure the correct option is selected and select **Generate** to proceed to the editing stage.

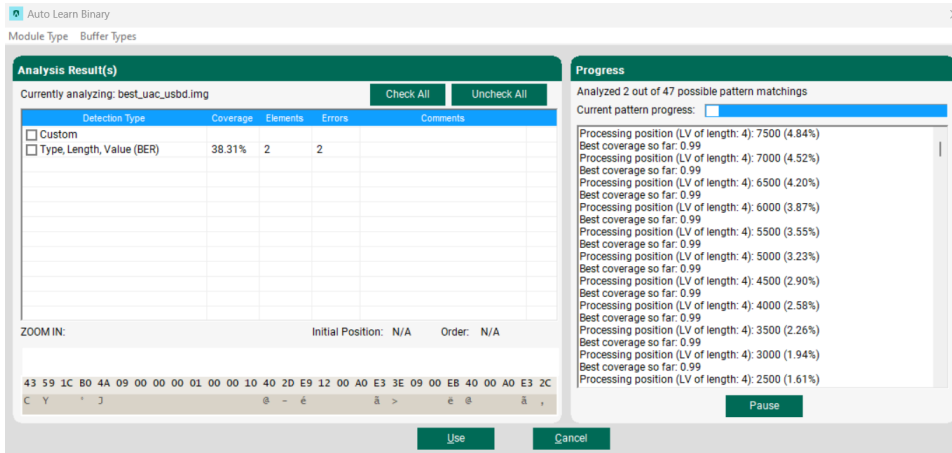
Import a file (PCAP Capture, packet dump, or file sample)

1. Choose the file you wish to edit. After loading the file you can view it in the hex editor on the right side of the screen.
2. Even though beSTORM attempts to determine whether the data it collected is binary or textual, make sure the correct option is selected and press the Generate button to go on to the editing stage

Generation (Editing the output into a beSTORM module):

Once you select **Generate**, beSTORM checks whether you treat the data as binary or textual. For binary data, beSTORM begins an automatic analysis process in attempt to determine the module's structure. In the end of this report, beSTORM suggests a list of relevant options. For textual data, beSTORM launches the Textual editor process which allows you to easily break down a sample into fuzzing parts, according to the text's structure.

Binary data

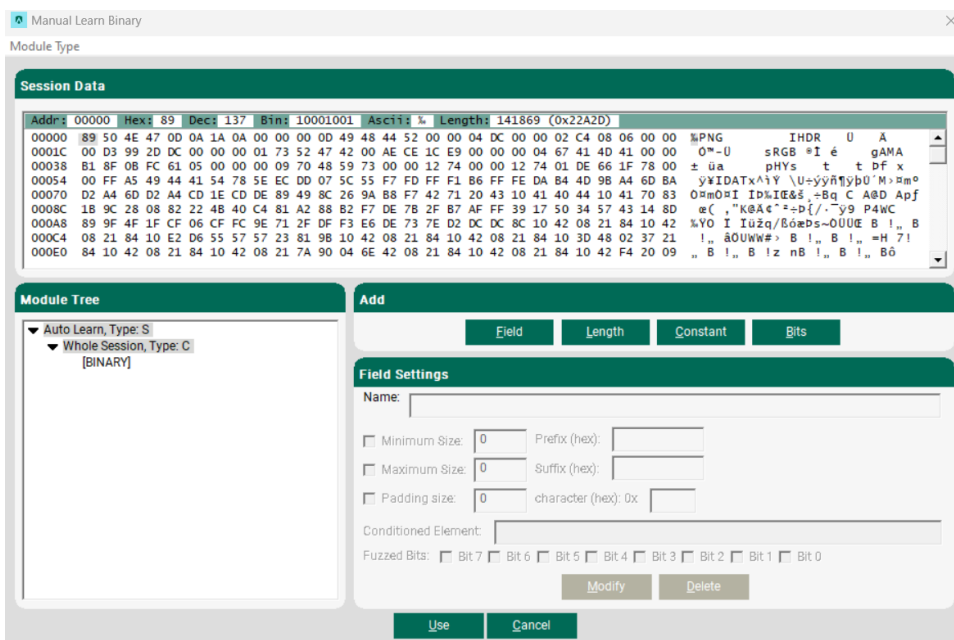


You may select any of the options, or all of them and proceed by clicking **Use**.

beSTORM gives some statistical information about each analysis output to aid you in selecting the most relevant options. The information consists of the total coverage of the packet beSTORM was able to match, the number of different elements found and the number of blocks that strictly differ from the tested pattern.

If you choose to select more than one option, beSTORM would perform all relevant options, in a prioritized order. This allows you to have beSTORM begin working on a protocol, even if you have no hint on its structure.

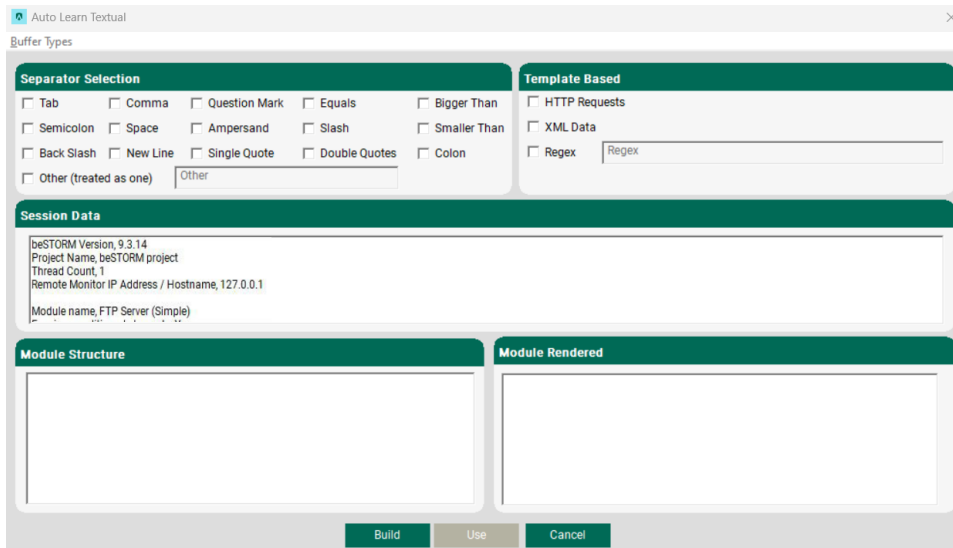
If you own some knowledge on the protocol structure, and it seems that beSTORM is unable to cover its specific unique form, it is possible to manually assist beSTORM in building the module. To do so, select **Custom** from the **Detection Type** list.



1. At the binary data editor, you can see a list of sessions on the top right of the screen (in case you wish to edit a different session), a hex and textual representation of the data in the top right side of the screen and the editing options/results on the bottom part of the screen.
2. Notice the Module Tree on the bottom left side of the screen. This part consists of your new module. Pressing on any of the branches would show its specific data on the Hex/Textual representation at the top right of the screen. Clicking the root would show the basic packet (no fuzzing of the data is shown) the module represents.
3. To mark a part of the packet to be Fuzzed, you need to select the relevant part in the Hex/Textual view (top right side of the screen), and then select **Add Buffer**. You can assign an indicative name for this part and configure the maximum/minimum size in bytes of this part and padding information (if needed).
4. Often, parts of the protocol are influenced by other parts, such as length of a buffer, number of appearances or actual content. This interface allows you to create element types that are affected by length. To do so, choose the location of the length part inside the module, and then select **Add Length**. A new dialog appears with the list of Buffers already added. Choose the buffer you want to bind the length to, and then select **OK**. Notice that the Buffer must be created before the Length.
5. You can also add constant data outside of the packet if necessary by selecting **Add Constant**.
6. After you finish building your module, select **Use** to use this module with beSTORM.

Textual data

The textual editor can be used by selecting the data as **Textual** in the Auto Learn screen (the one where the data input was selected) and then select **Generate**.



1. In the textual data editor you can see the list of sessions on the top right of the screen (in case you wish to edit a different session) and a list of delimiters. choose the relevant delimiters or choose 'HTTP Request' if this is the case and use our predefined delimiters for HTTP.
2. Select **Build** to automatically create your module.
3. After your module finishes building, select **Use** to use this module with beSTORM.

Environment Variables

Certain modules require additional information to perform proper testing. One such example is in the case of FTP, a Username and Password are needed for FTP log in, if you want to test all the available commands. This additional information can be changed by accessing the Environment Variables window:

This module's configuration can be further tweaked by altering certain parameters that depend on the testing environment.

Please review the following parameters and change their value if necessary:

Description	Value	Required
Destination MAC Address	00:00:00:00:00:00	Yes
First VLAN ID (default value 0)	00	No
Interface Name	\Device\NPF_{0B9C98BF-FC4D-4C...}	Yes
KiloBits per second (Maximum)	999999	No
Max Jitter Confidence (in percentage larger than or equa...)	95	No
Max Jitter Tolerance (in millisecond larger than 0)	100	No
Packets per second (Maximum)	999999	No
Second VLAN ID (default value 3)	03	No
Source MAC Address	00:02:03:04:05:06	Yes
Third VLAN ID (default value 4)	04	No

Save Cancel

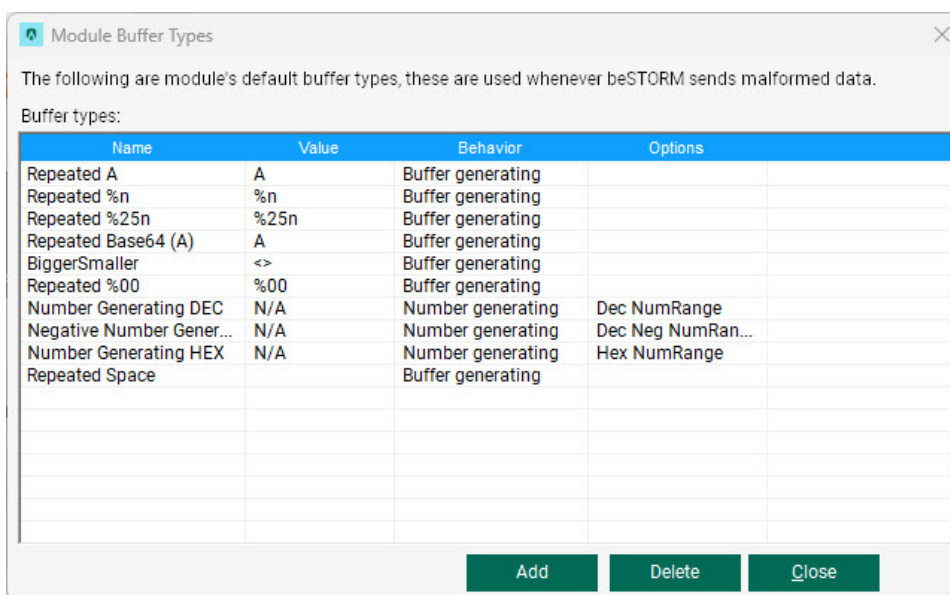
By double-clicking on the **Value** column, you can modify the values written in the field. If you input a value that starts with the "0x" characters and is followed by hexadecimal representation (if it is multiple bytes it needs to be comma separated) the value written is considered as binary.

For example the value, "anonymous", without the quotes will be regarded as ASCII while, "0x61, 0x6e, 0x6f, 0x6e, 0x79, 0x6d, 0x6f, 0x75, 0x73", without the quotes will be regarded as Binary, even though their value is the same, in both cases "anonymous" without the quotes.

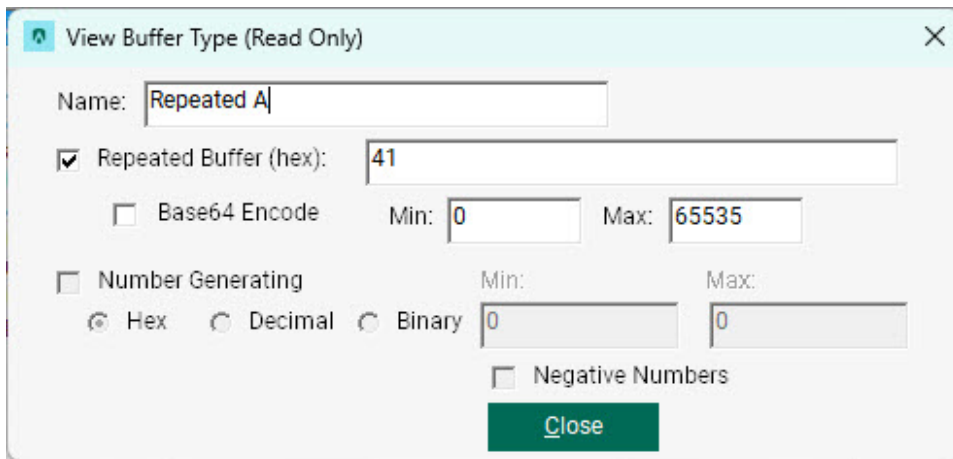
Module Buffer Types

beSTORM's different predefined modules have their own built-in Buffer Types. These Buffer Types are used by beSTORM whenever it tests the buffers found inside the module. Each of the Buffer Types checks for different types of vulnerabilities.

For example, the Repeated A buffer type attempts to trigger an exception that is usually associated with Buffer Overflows or Heap Overflows, while the Bigger Smaller buffer type attempts to trigger exceptions related to email address (SMTP) or hostname designators (SIP).



The settings of each buffer type is configurable, and you can add additional types to the existing set of buffer types. Buffer Types have several behavioral aspects, they can be either be buffer generating (that is, they generate a buffer that increases in length), or number generating (that is, their value increases).



View Buffer Type (Read Only)

Name:

Repeated Buffer (hex):

Base64 Encode Min: Max:

Number Generating Min: Max:

Hex Decimal Binary

Negative Numbers

In addition, each Buffer Type can have other behavioral aspects such as decimal (that is, the value is represented as a number between 1 and 4,294,967,296), as a binary value (that is, the value is represented by the byte value between 0x00000000 and 0xFFFFFFFF), or a hexadecimal value (that is, the value is represented by the string representation between 0 and FFFFFFFF). Further, a buffer type's behavior is extendable by limiting its range or even adding additional encoding to it, such as in the case of Base64, in accordance with the BASE64 encoding standard, and URL encoding, where we convert such characters as % to %25.

Custom Modules

Before we can start building custom modules, we need to understand how the language with which the modules are defined is structured. beSTORM modules are constructed using an XML syntax. The beSTORM's XML syntax allows both extensibility as well as a structured session to be built.

Like most XML-based syntax, refer to the Element Type as the string without any spaces within it that follows just after the lower than character (<).

- Element Attribute as the string without any spaces within it that provides additional parameters to the Element Type.
- Element Value as the string encompassed within double quotes that provide a value to the Element Attribute.

By definition, no Element Attribute can appear without its Element Value. The Element Type is either closed with a forward slash character and greater-than character combination (/>), meaning that no closer tag follows, or only with a greater-than character (>), meaning that the element contains additional elements which are defined as its children (also referred to as nodes). For example:

```
<ElementType ElementAttribute="ElementValue"  
ElementAttribute="ElementValue" ... />
```

NOTE: All of the Element Values are contained within double quotes.

A beSTORM module's XML syntax is read from top to bottom, with some exceptions where elements that can be introduced into the module change this behavior. Start off by providing the most basic element; Words. Words define elements whose smallest size is 8 bits (1 byte) for one word (for smaller elements refer to Bits).

All of beSTORM's elements have an Element Attribute called Name, whose value must be unique within the same module.

Configuration Elements

The XML file defines both the way the module looks like, acts and what types of data it sends as well as how it should generate the fuzzed data. Fuzz data can either be generated from a predefined list of buffer types, Textual and Binary, or from a list the builder of the module provides. The method to define what type of buffer types beSTORM will generate is done by placing a `GeneratorOptSettings` element under the top `beSTORM` element.

For example, the following defines to beSTORM that you would like to utilize the factory defined buffer types of Binary form.

```
<beSTORM Version="1.2" ><GeneratorOptSettings FactoryDefined="1"
FactoryType="Binary" />
<ModuleSettings>
<M Name="ASN1 Samples" >
<P Name="ASN1 Samples" >
<S Name="ASN1 Sentence" >
<C Name="Identifier Octet (Bit stream)" Value="0x04" />
<L Name="Length of Element" Split="128" ConditionedName="Buffer of
Element" />
<B Name="Buffer of Element" ASCIIIDefault="Data of Element" />
</S>
</P>
</M>
</ModuleSettings>
</beSTORM>
```

If you want to define your own buffer types you could place `BT` elements under the `GeneratorOptSettings` element, for example:

```
<GeneratorOptSettings>
<BT>
<BT Name="Repeated A" Max="65536" ASCIIValue="A" />
<BT Name="Repeated %n" Max="512" ASCIIValue="%n" />
<BT Name="Repeated %25n" Max="256" ASCIIValue="%25n" />
<BT Name="Repeated Base64A" Max="16384" Type="Base64" ASCIIValue="A"
/>
<BT Name="BiggerSmaller" Max="32768" ASCIIValue="&lt;&gt;" />
<BT Name="Repeated %00" Max="21846" ASCIIValue="%00" />
<BT Name="Number Generating DEC" Max="4294967295"
Type="DecimalPositive" />
<BT Name="Negative Number Generating DEC" Max="2147483648"
Type="DecimalNegative" />
<BT Name="Number Generating HEX" Max="4294967295"
```



```
Type="DecimalPositive" />
<BT Name="Repeated Space" Max="65536" ASCIIValue=" " />
</BT>
</GeneratorOptSettings>
```

Each BT element has a child T element which defines the type of data that is generated.

- The Name attribute is a user provided description of what this attack would generate.
- The Max attribute defines how many times will this data be either repeated or incremented (depending on the attributes ASCIIValue or Type appearing).
- The ASCIIValue (or Value for binary data) attribute tell beSTORM it should generate data that is repeated from 0 up to and including Max.
- The Type attribute defines what sort of incrementing should be performed on the data, default value of Repeat is set to the attribute if no value is provided.
- If Base64 is provided the data will be encoded (after it is generated) with the Base64 encoding scheme.
- If DecimalPositive value is provided the number will be incremented and it is assumed that no negative number will ever be generated (due to overflowing of the number when incremented above the maximum allowed value).
- If DecimalNegative value is provided the number will act as the DecimalPositive behaves, the only difference is that when an overflow occurs negative numbers will be used.
- If Hex value is provided the DecimalPositive type will be used but the returned value would be presented in 0000000-FFFFFF form (textual).
- If Binary value is provided the DecimalPositive type will be used but the returned value would be presented in 0-4294967296 form (textual).

Words

Words are elements that contain data, but in most cases cannot contain other words or elements – the exception is the Enumerate element. In most cases Words are defined by a single character when they are written in the XML file, for example to place a Buffer, you write the element like so: <B . . . />

There are two types of Words, conditioned and non-conditioned.

- Words that are conditioned are words whose value depends on other word's or sentence's value for their own value, while Words that are non-conditioned are words whose value does not depend on other word's or sentence's value for their own value.

- Non conditioned Words have either a Default element attribute if beSTORM fuzzes them, such as in the case of Buffers and Variable Buffers, or a Value element attribute if beSTORM does not fuzz them, such as in the case of Constant and Environment Variables.

Each of these have an equivalent ASCIIDefault and ASCIIValue respectively which unlike Default and Value receives the data in ASCII form, whereas the other receives it in hexadecimal form (for example, "0x40, 0x00,0x00,0x00,0x00" (without the quotes).

Conditioned Words have a ConditionedName attribute which instructs beSTORM from where the Word's value is dependent upon. Most conditioned elements have also a Size attribute which instructs beSTORM how big, in bytes, the return value should be, such as in the case of Length Conditioned and Count Conditioned. The conditioned element called Procedure Conditioned has an additional attribute, named Parameter, which instructs beSTORM which value should this element take.

Buffer

- **Type** - B
- **Usage** - Defines a value that is manipulated by beSTORM's fuzzing mechanism.

```
<B Name="A Buffer" ASCIIDefault="The ASCII Default" />
```

Attribute name	Attribute function
Value	The system default value of the buffer, which is used when the user requests that the module be reverted to its factory default. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIIValue	Same as Value but the provided data is in ASCII form
Prefix	The prefix that will be attached to the beginning of the buffer. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIIPrefix	Same as Prefix but the provided data is in ASCII form.
PrefixOnlyIfNonEmpty	Sets whether to place the Prefix even if the buffer value is empty, due to fuzzing, valid values are either '1' (the number one) for place prefix only if non empty or '0' (the number zero) for always place prefix. By default we place the Prefix even if the buffer is empty.
Suffix	The suffix that will be attached to the end of the buffer. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIISuffix	Same as Suffix but the provided data is in ASCII form.

Attribute name	Attribute function
SuffixOnlyIfNonEmpty	Sets whether to place the Suffix even if the buffer value is empty, due to fuzzing, valid values are either '1' (the number one) for place suffix only if non empty or '0' (the number zero) for always place suffix. By default we place the Suffix even if the buffer is empty.
MaxBytes	Sets the maximum number of allowed bytes that this buffer can fuzz to – usually used for binary protocols. By default the value is not set, no restriction.
MinBytes	Sets the minimum number of allowed bytes that this buffer can fuzz to – usually used for binary protocols. By default the value is unset, no restriction.
Blocked	Sets whether this buffer is currently being fuzzed, as one buffer or buffers depending on the settings blocks other buffers from fuzzing until they are finished. By default one buffer blocks others from fuzzing, that buffer will be marked as blocked while the rest will not be marked as blocked. Valid values are either '1' (the number one) for blocked or '0' (the number zero) for not blocked.
Default	The current value being used by the Buffer, unlike Value, Default sets the current value of the Buffer, which is lost if the buffer is reset to factory default.
ASCIIDefault	Same as Default but the provided data is in ASCII form.
NoDefaultTypes	This sets whether we wish or not to fuzz this buffer with our set of Buffer Types. Valid values are either '1' (the number one) for no default buffer types (that is, do not fuzz) or '0' (the number zero) for set default buffer types (that is, fuzz). By default the buffer is fuzzed using the default buffer types.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either "Suffix" for suffix padding or "Prefix" for prefix padding. By default the element is Suffix padded.

Attribute name	Attribute function
Locked	This sets whether the element is changed or not, a locked element's value is never changed, while an unlocked one will be modified. Valid values are either '1' (element locked) and '0' (element is unlocked). By default the element is not locked.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.
Position	The position at which the current Element is at, this is used by beSTORM to allow restoring the state of beSTORM to how it was when the module settings were saved.
Max	The maximum position of the current Element, for Buffer it is the number of Buffer Types for this element. It is recommended to not modify this value unless you are removing Buffer Types from this Buffers.
NoDefaultTypes	This sets whether we wish or not to fuzz this buffer with our set of Buffer Types, valid values are either '1' (the number one) for no default buffer types – i.e. do not fuzz – or '0' (the number zero) for set default buffer types – i.e. fuzz. By default the buffer IS fuzzed using the default buffer types.
CalculateScenarios	An attribute that saves the number of scenarios this element has, allows beSTORM to load the element more quickly.
Priority	An attribute that defines when will this element get fuzzed, the default value of 5 means neither higher nor lower priority. A higher number means this will be tested first, while a lower priority means this will be tested last. For priority to work properly the path leading to this element has to have a priority higher or equal to this element's priority.
IterateOnce	An attribute that defines that this Buffer will be tested ONLY once, after it is tested, it will be tested it will start sending its default value.
Disabled	An attribute that defined that this Buffer will become inactive and will not be fuzzed, moreover, it will NOT return any data when beSTORM will attempt to generate this element. This attribute should not be used to prevent beSTORM from fuzzing this element.

Attribute name	Attribute function
Type	An attribute that defines Buffer Types for this Buffer, the following Types are currently supported: <ul style="list-style-type: none"> • Unicode • Timestamp (textual) • Integer (textual) • Integer (binary) • MAC Address • Hostname (textual) • IP Address (binary) • IP Address (textual) • IP Address v6 (binary)
Format	An attribute that defines a Buffer Type that will try to overflow a scan function, by looking into creating a complex buffer that support %s and %d field types.

Buffer type

- **Type** - BT
- **Usage** - Defines a value that is manipulated by beSTORM's fuzzing mechanism.

```
<BT Name="A Buffer" Max="65535" ASCIIValue="A" />
```

Attribute name	Attribute function
Value	The value that is repeated by the buffer. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...)
ASCIIValue	Same as Value but the provided data is in ASCII form
Prefix	The prefix that will be attached to the beginning of the buffer type. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...)
ASCIIPrefix	Same as Prefix but the provided data is in ASCII form
PrefixOnlyIfNonEmpty	Sets whether to place the Prefix even if the buffer type value is empty, due to fuzzing, valid values are either '1' (the number one) for place prefix only if non empty or '0' (the number zero) for always place prefix. By default we place the Prefix even if the buffer type is empty

Attribute name	Attribute function
Suffix	The suffix that will be attached to the end of the buffer type. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...)
ASCIISuffix	Same as Suffix but the provided data is in ASCII form
SuffixOnlyIfNonEmpty	Sets whether to place the Suffix even if the buffer type value is empty, due to fuzzing, valid values are either '1' (the number one) for place suffix only if non empty or '0' (the number zero) for always place suffix. By default we place the Suffix even if the buffer type is empty
Appender	Sets whether this buffer type is an appender (that is, adds to the existing value of the Buffer under which the Buffer Type sits under) or whether it replaces it. For example, an appender adds to 'Data' a set of 'A' resulting in 'DataA', 'DataAA', 'DataAAA', etc., while a replacer replaces 'Data' with 'A', 'AA', 'AAA' etc. Valid values are either '0' (not an appender) and '1' (an appender). By default the buffer type is a replacer not an appender.
AppenderType	Sets what type of appender is being used to append to the beginning of the data or to its end. Valid values are either 'Prefix' (at the beginning) and 'Suffix' (at the end). By default the buffer type appends at the beginning.
Type	Sets what type of manipulation is performed on this Buffer Type. Valid values are: <ul style="list-style-type: none"> • Base64 – Repeat the data then base64 encode it. • Repeat – Repeat the data. • DecimalPositive – Increment the value from the number 0 up to the maximum set for this buffer type. • DecimalNegative – Increment the value from the number 0 up to the maximum set for this buffer type divided by 2 and from there start providing negative values. • Hex – Like DecimalPositive, but in hex form. • Binary - Like DecimalPositive, but in binary form.
Enable	Sets whether this Buffer Type is used or not, setting a value of '1' will enable the buffer type, while setting a value of '0' will disabled it. By default buffer types are enabled.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.

Attribute name	Attribute function
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either "Suffix" for suffix padding or "Prefix" for prefix padding. By default the element is Suffix padded.
Locked	This sets whether the element is changed or not, a locked element's value is never changed, while an unlocked one will be modified. Valid values are either '1' (element locked) and '0' (element is unlocked). By default the element is not locked.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.
Max	The maximum position of the Buffer Type, for generating data Buffer Type, this is the number of times the value is replicated. For example, 1024, of "AA" replicated data, will generate 2048 bytes of data at its Maximum position. While a Max value of 1024 for a Number generating will mean it will generate the number 1024 at its Maximum position.
MinPos	An attribute that defines what will be the minimal position for this Buffer Type. For a replicating Buffer Type, a value of 10 will mean that this data will be generated at its minimum value 10 times.
FactoryDefined	An attribute that defines whether or not the Buffer Type will be using beSTORM shipped buffer types, or a custom one. valid values for this field is either '0' or '1' (ASCII value zero or one).
FactoryType	An attributes that defined what type of Buffer Type is being used, valid values are: <ul style="list-style-type: none"> • Text • Binary • WSSA (for Windows only)
CalculateScenarios	An attribute that saves the number of scenarios this element has, allows beSTORM to load the element more quickly.

Attribute name	Attribute function
Priority	An attribute that defines when will this element get fuzzed, the default value of 5 means neither higher nor lower priority. A higher number means this will be tested first, while a lower priority means this will be tested last. For priority to work properly the path leading to this element has to have a priority higher or equal to this element's priority.
IterateOnce	An attribute that defines that this Buffer will be tested ONLY once, after it is tested, it will be tested it will start sending its default value.
Disabled	An attribute that defined that this Buffer will become inactive and will not be fuzzed, moreover, it will NOT return any data when beSTORM will attempt to generate this element. This attribute should not be used to prevent beSTORM from fuzzing this element.
MaxBytesToGenerate	An attribute that allows you to define how many bytes this Buffer Type will send, unlike Max which you cannot directly know how many bytes will be generated as it is based on a formula of "Position" * "Number of Bytes Replicated", with this you can directly define a value of 1024, so that no more than 1024 bytes are ever sent.
Position	The position at which the current Element is at, this is used by beSTORM to allow restoring the state of beSTORM to how it was when the module settings were saved.
ReturnDefault	An attribute that defines whether this Buffer Type will return the default value at position 0, the default behavior is that at position 0 is that nothing is being sent (empty buffer). Valid values for this field is either '0' or '1' (ASCII value zero or one).

Constant

- **Type** - C
- **Usage** - Defines a value that is not manipulated by beSTORM's fuzzing mechanism.

```
<C Name="A Const" ASCIIValue="The ASCII Value" />
```

Attribute name	Attribute function
Value	The system default value of the constant. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIIValue	Same as Value, but the provided data is in ASCII form.

Attribute name	Attribute function
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either "Suffix" for suffix padding or "Prefix" for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.

Enumerate

- **Type** - E
- **Usage** - Defines a group of Constant elements that will be used in sequence – one after the other – similar in behavior to an OR between element, either one is used or the other.

```
<E Name="Aset of constants">
  <C Name="Const#1"ASCIIValue="First constant sent"/>
  <C Name="Const #2" ASCIIValue="Second constant sent"/>
  <C Name="Const#3"ASCIIValue="Thirdconstant sent"/>
</E>
```

Attribute name	Attribute function
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either "Suffix" for suffix padding or "Prefix" for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.

Length conditioned

- **Type** - L
- **Usage** - Defines an element whose value is based on the length –size – of the element it is bound/conditioned to.

```
<L Name="Length of the Buffer" ConditionedName="A Buffer" />
<B Name="ABuffer"ASCIIDefault="The ASCIIDefault"/>
```

The B element's value will be the length of "The ASCIIDefault" without the quotes (that is, 17).

Attribute name	Attribute function
Size	Length conditioned elements can either return ASCII form values, the string representation of the value of 10221 or binary form values, the binary representation of the value of 10221 – 0x10221. If this attribute is set, the binary form is used. Valid values are between 0 (don't use binary form) and 4 (4 bytes of data will be used to represent the number).
Split	This sets at what value should we split the binary form of the length into more than one byte, this is relevant only to ASN.1 protocols where the length values greater than 0x7F are split into two bytes or more. Valid values are between 0x00 and 0xFF, for ASN.1 use the value of 0x7F.
ForceSplitSize	In some cases we want to force the split to occur even if the length is smaller than the provided split size. This is useful when protocols are strict to how many bytes are used in each of the length fields. Valid values are between 0x01 and 0x04, which in turn will force the size to be 1 up to 4 bytes even if more or less is necessary. If a size smaller than required is provided the length is trimmed and will be invalid.
NetworkOrder	This value is only relevant for binary length values, this sets whether a Network order (big endian) is used or Host order (small endian) is used. Valid values are either '1' (for network order) or '0' (for host order). By default the length is set to Host order.
Base	For decimal representation of the length, you can decide whether you want the value to be generated in decimal form or hexadecimal form. Valid values are either 'Hex' (for hexadecimal) or 'Decimal' (for decimal form). By default the representation is done in decimal form.

Attribute name	Attribute function
ConditionedName	The value of the Length Conditioned element depends not on its own value, but rather on the value on which it is depended on. In the case of Length Conditioned elements, the size of the elements its conditioned to is calculated and returned as the results of this element. Valid values are either short name form – the name of the element it is bound to without the full path to the element, or full name form – the name of all the elements up to the element we are bound to, separated by a pipe (“ ”) character.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either “Suffix” for suffix padding or “Prefix” for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.

Count conditioned

- **Type** - CC
- **Usage** - Defines an element whose value is based on the number of elements the element it is bound/conditioned to has.

```
<CC Name="Count of Elements "Conditioned Name="Contains a
Number of Elements"/>
<S Name="Contains a Number of Elements">
<B Name="1 Buffer"ASCIIDefault="TheASCIIDefault" />
<B Name="2 Buffer "ASCIIDefault="TheASCIIDefault" />
</S>
```

The CC element will return 2.

Attribute name	Attribute function
Size	Count conditioned elements can either return ASCII form values, the string representation of the value of 10221 or binary form values, the binary representation of the value of 10221 – 0x10221. If this attribute is set, the binary form is used. Valid values are between 0 (don't use binary form) and 4 (4 bytes of data will be used to represent the number).

Attribute name	Attribute function
NetworkOrder	This value is only relevant for binary counting values, this sets whether a Network order (big endian) is used or Host order (small endian) is used. Valid values are either '1' (for network order) or '0' (for host order). By default the counting is set to Host order.
Base	For decimal representation of the counting, you can decide whether you want the value to be generated in decimal form or hexadecimal form. Valid values are either 'Hex' (for hexadecimal) or 'Decimal' (for decimal form). By default the representation is done in decimal form.
ConditionedName	The value of the Count Conditioned element depends not on its own value, but rather on the value on which it is depended on. In the case of Count Conditioned elements, the number of the repeated elements (of the repeater) its conditioned to is calculated and returned as the results of this element. Valid values are either short name form – the name of the element it is bound to without the full path to the element, or full name form – the name of all the elements up to the element we are bound to, separated by a pipe (“ ”) character.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either “Suffix” for suffix padding or “Prefix” for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.

Duplicate conditioned

- **Type** - DC
- **Usage** - Defines an element whose value is based on the copy of the value of the element it is bound/conditioned to.

```
<DC Name="Duplicate of Element" ConditionedName="A Buffer">
<B Name="A Buffer "ASCIIIDefault="The ASCIIIDefault"/>
```

The DC element will return “The ASCII Default” without the quotes.

Attribute name	Attribute function
ConditionedName	The value of the Duplicate Conditioned element depends not on its own value, but rather on the value on which it is depended on. In the case of Duplicate Conditioned elements, the value of the element it is depended on is returned as the results of this element. Valid values are either short name form – the name of the element it is bound to without the full path to the element, or full name form – the name of all the elements up to the element we are bound to, separated by a pipe (“ ”) character.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either “Suffix” for suffix padding or “Prefix” for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element

Procedure conditioned

- **Type** - PC
- **Usage** - Defines and element whose value is based on the returnvalue of a Sentence Procedure of the element it is bound/conditioned to.

```
<PC Name="SOCKET" ConditionedName="Connector"
Parameter="SOCKET"/>
```

See below for Sentence Procedure for further details on how to use the Procedure Conditioned element.

Attribute name	Attribute function
Parameter	The name of the parameter we query the dependent element for, most elements support a 'Output' parameter by default.

Attribute name	Attribute function
ConditionedName	The value of the Procedure Conditioned element depends not on its own value, but rather on the value on which it is depended on. In the case of Procedure Conditioned elements, the element it is depended on is queried for a specific parameter, the value of this parameter is returned as the value for this element. Valid values are either short name form – the name of the element it is bound to without the full path to the element, or full name form – the name of all the elements up to the element we are bound to, separated by a pipe (“ ”) character.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either “Suffix” for suffix padding or “Prefix” for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.

Variable buffer

- **Type** - VB
- **Usage** - Defines an element whose value can be easily changed by the user by way of the GUI's Environment Variables but beSTORM regards them like Buffers (that is, beSTORM will fuzz its value).

```
<VB Name="Variable Buffer" ASCIIIDefault="This is the default value of the variable buffer" Description="Place here the value you want for the Variable Buffer"/>
```

The “Place here the value you want for the Variable Buffer” string will be displayed to the user under the Environment Variables screen.

Attribute name	Attribute function
Description	This sets the string that is displayed to the user under the “ Edit Module Environment Variables” of the project he has loaded the module into. Using the same description will bound the values of all the elements together, modifying one in the “Edit Module Environment Variables” screen will modify all of them.

Attribute name	Attribute function
Value	The system default value of the buffer, which is used when the user requests that the module be reverted to its factory default. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIIValue	Same as Value, but the provided data is in ASCII form.
Prefix	The prefix that will be attached to the beginning of the buffer. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIIPrefix	Same as Prefix, but the provided data is in ASCII form.
PrefixOnlyIfNonEmpty	Sets whether to place the Prefix even if the buffer value is empty, due to fuzzing, valid values are either '1' (the number one) for place prefix only if non empty or '0' (the number zero) for always place prefix. By default we place the Prefix even if the buffer is empty.
Suffix	The suffix that will be attached to the end of the buffer. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIISuffix	Same as Suffix but the provided data is in ASCII form.
SuffixOnlyIfNonEmpty	Sets whether to place the Suffix even if the buffer value is empty, due to fuzzing, valid values are either '1' (the number one) for place suffix only if non empty or '0' (the number zero) for always place suffix. By default we place the Suffix even if the buffer is empty.
MaxBytes	Sets the maximum number of allowed bytes that this buffer can fuzz to – usually used for binary protocols. By default the value is unset, no restriction.
MinBytes	Sets the minimum number of allowed bytes that this buffer can fuzz to – usually used for binary protocols. By default the value is unset, no restriction.
Blocked	Sets whether this buffer is currently being fuzzed, as one buffer or buffers depending on the settings blocks other buffers from fuzzing until they are finished. By default one buffer blocks others from fuzzing, that buffer will be marked as blocked while the rest will not be marked as blocked. Valid values are either '1' (the number one) for blocked or '0' (the number zero) for not blocked.
Default	The current value being used by the Buffer, unlike Value, Default sets the current value of the Buffer, which is lost if the buffer is reset to factory default.

Attribute name	Attribute function
ASCIIDefault	Same as Default but the provided data is in ASCII form.
NoDefaultTypes	This sets whether we wish or not to fuzz this buffer with our set of BufferTypes, valid values are either '1' (the number one) for no default buffer types (that is, do not fuzz) – or '0' (the number zero) for set default buffer types (that is, fuzz). By default the buffer is fuzzed using the default buffer types.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either "Suffix" for suffix padding or "Prefix" for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.
Required	Whether the element must contain a default value.

Environment variables

- **Type** - EV
- **Usage** - Defines an element whose value can be easily changed by the user by way of the GUI's Environment Variables, but beSTORM regards them like Constants (that is, beSTORM will not fuzz its value).

```
<EV Name="Environment Buffer" ASCIIDefault="This is the default
value of the environment buffer" Description="Place here the
value you want for the Environment Buffer" />
```

The "Place here the value you want for the Environment Buffer " string will be displayed to the user under the Environment Variables screen.

Attribute name	Attribute function
Description	This sets the string that is displayed to the user under the “ Edit Module Environment Variables” of the project he has loaded the module into. Using the same description will bound the values of all the elements together, modifying one in the “Edit Module Environment Variables” screen will modify all of them.
Value	The system default value of the buffer, which is used when the user requests that the module be reverted to its factory default. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIIValue	Same as Value but the provided data is in ASCII form.
Prefix	The prefix that will be attached to the beginning of the buffer. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIIPrefix	Same as Prefix but the provided data is in ASCII form.
PrefixOnlyIfNonEmpty	Sets whether to place the Prefix even if the buffer value is empty, due to fuzzing, valid values are either '1' (the number one) for place prefix only if non empty or '0' (the number zero) for always place prefix. By default we place the Prefix even if the buffer is empty.
Suffix	The suffix that will be attached to the end of the buffer. The value provided is in hexadecimal form and separated by comma (0xXX, 0xXX...).
ASCIISuffix	Same as Suffix but the provided data is in ASCII form.
SuffixOnlyIfNonEmpty	Sets whether to place the Suffix even if the buffer value is empty, due to fuzzing, valid values are either '1' (the number one) for place suffix only if non empty or '0' (the number zero) for always place suffix. By default we place the Suffix even if the buffer is empty.
MaxBytes	Sets the maximum number of allowed bytes that this buffer can fuzz to – usually used for binary protocols. By default the value is unset, no restriction.
MinBytes	Sets the minimum number of allowed bytes that this buffer can fuzz to – usually used for binary protocols. By default the value is unset, no restriction.

Attribute name	Attribute function
Blocked	Sets whether this buffer is currently being fuzzed, as one buffer or buffers depending on the settings blocks other buffers from fuzzing until they are finished. By default one buffer blocks others from fuzzing, that buffer will be marked as blocked while the rest will not be marked as blocked. Valid values are either '1' (the number one) for blocked or '0' (the number zero) for not blocked.
Default	The current value being used by the Buffer, unlike Value, Default sets the current value of the Buffer, which is lost if the buffer is reset to factory default.
ASCIIDefault	Same as Default but the provided data is in ASCII form.
NoDefaultTypes	This sets whether we wish or not to fuzz this buffer with our set of BufferTypes, valid values are either '1' (the number one) for no default buffer types (that is, do not fuzz) – or '0' (the number zero) for set default buffer types (that is, fuzz). By default the buffer IS NOT fuzzed using the default buffer types.
PaddingChar	The character used to pad this element. Valid values are between 0x00 and 0xFF, the NULL character and EOF character respectively.
PaddingSize	The size of the padding to do for the given element. Valid values are between 0 (do not pad) and 65536 (pad up to a length of 65536 bytes). By default the element is not padded.
PaddingType	The type of padding to conduct on the element, either Prefix or Suffix. Valid values are either "Suffix" for suffix padding or "Prefix" for prefix padding. By default the element is Suffix padded.
Comment	The element's comment, this is a free text that can be used by the module build to provide meaning and context to the element.
Required	Whether the element must contain a default value.

Bits

In some cases the modules' builder needs elements that are smaller than 8 bits – 1 byte – 1 word, in such cases Bit elements come into play. However, as all data on the computer is stored in bytes, 1 or more, we need to place the Bit elements inside an element whose smallest size is 8 bits – 1 byte – 1 word.

Name	Type	Usage
Bit	BB	<p>Defines an element whose value is 0 (zero) and 1 (one), and can grow up to a size of 8 bits – 1 byte – 1 word. The Bit element behaves like a Constant (that is, beSTORM will not fuzz its value). As mentioned all elements need to be at least of the size of a round byte, 1, 2, 3, etc bytes, thus a Bit element is always surrounded by a BitsContainer element which makes sure the internal bits found inside it sums up to a whole byte/word.</p> <pre><BB Name="A list of Bits" Size="2" MultiBits="0,1" /></pre>
BitsEnum	BE	<p>Defines a group of Bit elements that will be used in sequence – one after the other – similar in behavior to an OR between elements, either one is used or the other.</p> <pre><BE Name="A set of bits"> <BB Name="Bits #1" Size="3" MultiBits="0,0,0" /> <BB Name="Bits #2" Size="3" MultiBits="1,0,0" /> <BB Name="Bits #3" Size="3" MultiBits="1,0,0" /> </BE></pre> <p>The MultiBits attribute is written with the MSB as the leftmost digit, and the LSB as the rightmost digit.</p>
BC	BC	<p>Defines a group of Bit elements that will be used in concatenation – all at the same time – similar in behavior to an AND between elements, use all of them together.</p> <pre><BC Name="All at once"> <BB Name="Bits #1" Size="2" MultiBits="0,0" /> <BB Name="Bits #2" Size="2" MultiBits="1,0" /> <BB Name="Bits #3" Size="4" MultiBits="0,1,0,0" /> </BC></pre> <p>The bits value returned will be (MSB first) 0,0 followed by 1,0 followed by 0,1,0,0.</p>

Sentences

Sentences bind things together in the simplest form they bind together Words and Bits, in more complex forms they allow the module to express decision taking behavior – such as in the case of Session based modules (that is, the FTP module).

Name	Type	Usage
Simple Sentence	S	<p>Defines a simple sentence that can encompass under it one or more Words and BitsContainer. The elements under this the Simple Sentence will be used in concatenation – all at the same time – similar in behavior to an AND between elements, use them all together.</p> <pre data-bbox="513 432 1230 638"><S Name="A very simple sentence"> <B Name="First buffer in the block" ASCIIIDefault=" This is my default" /> <C Name="Second constant in the block "ASCIIValue="This is my value" /> </S></pre> <p>The value of the S element will be "This is my default This is my value " without the quotes.</p>
Sentence Enumerate	SE	<p>Defines a group of Sentence elements that will be used in sequence – one after the other – similar in behavior to an OR between elements, either one is used or the other.</p> <pre data-bbox="513 926 1247 1268"><SE Name="A set of sentences"> <S Name="Sentence #1"> <B Name="First buffer in the block" ASCIIIDefault="This is my default" /> </S> <S Name="Sentence #2"> <C Name="Second constant in the block" ASCIIValue="This is my value" /> </S> </SE></pre> <p>The value of the SE element will be first " This is my default" without the quotes, followed by "This is my value" without the quotes.</p>

Name	Type	Usage
Sentence Container	SC	<p>Defines a group of Sentence elements that will be used in concatenation – all at the same time – similar in behavior to an AND between elements, use all of them together.</p> <pre data-bbox="513 363 1247 709"><SC Name="All at once"> <S Name="Sentence #1"> <B Name="First buffer in the block" ASCIIIDefault="This is my default" /> </S> <S Name="Sentence #2"> <C Name="Second constant in the block" ASCIIValue="This is my value" /> </S> </SC></pre> <p>The value of the SC element will be "This is my default This is my value" without the quotes.</p>
Sentence Repeater	R	<p>Defines that the element under this element will be repeated by default from 0 (zero) times up to 8 (eight) times. The maximum number of repeated sentences can be controlled by setting of the Max attribute. Any repeated element will be created from the original provided element. The elements will interact with their siblings – other repeated elements under the same elements – as if they were placed under a Sentence Container.</p> <pre data-bbox="513 1140 1211 1339"><R Name="All at once" Max="3"> <S Name="Sentence #1"> <B Name="First buffer in the block" ASCIIIDefault="This is my default" /> </S> </R></pre> <p>At first the R element will contain no Sentence #1 element, then there will be 1 such element, 2 such elements, up to 3 such elements as defined by the Max attribute. In each case they will be used in concatenation, similar to how a Sentence Container having multiple elements would behave.</p>

Name	Type	Usage
Sentence Procedure	SP	<p>Defines an element which calls some functionality that is provided outside beSTORM's main programming code. This allows beSTORM to use the capabilities of external libraries such as OpenSSL, Mathematical Algorithm, etc., as well as provide the builders of custom modules the ability to introduce into beSTORM additional proprietary functionality. The value of a Sentence Procedure can be retrieved by using a Procedure Conditioned element and querying the value returned by the Sentence Procedure's execution. See more details on external functionality provided by beSTORM at the External Functionality section.</p> <pre data-bbox="516 646 1369 1213"> <SP Name="Add Function" Library=".dll" Procedure="Add"> <S Name="First Parameter" ParamName="A"> <C Name="Fixed Header Size" Value="0x10"/> </S> <S Name="Second Parameter" ParamName="B"> <C Name="Fixed Body Size" Value="0x40"/> </S> <S Name="Size Parameter" ParamName="Size"> <C Name="Return Size" ASCIIValue="4" /> </S> </SP> <S Name="Use return value"> <PC Name="Value of Add" ConditionedName="Add Function" Parameter="Output"/> </S> </pre> <p>The value returned by the SP element in this case, as we are using the mathematical function of Add will be in hexadecimal form 0x0050, the size of 4 indicates that we are interested in receiving 4 bytes in the result.</p>

Name	Type	Usage
Protocol	P	<p>The element just under the Module sentence, protocols define one or more state at which the module can work at. In similar fashion to the Sentence Enumerate the Protocol element defines a group of Sentence elements that will be used in sequence – one after the other – similar in behavior to an or between elements, either one is used or the other.</p> <pre data-bbox="513 470 1442 957"> <P Name="Simple Protocol"><S Name="A very simple sentence"> <B Name="First buffer in the block" ASCIIIDefault="This is my default" /> <C Name="Second constant in the block" ASCIIValue="This is my value" /> </S> <S Name="A more complex sentence"> <B Name="First buffer in the block" ASCIIIDefault="This is my default" /> <B Name="Second buffer in the block"ASCIIIDefault="This is my default #2" /> </S> </P> </pre>
Module	M	<p>The root element under which the module is built, only one occurrence can appear in any one module, in addition only Protocol elements can appear under it. In similar fashion to the Sentence Enumerate the Module element defines a group of Protocol elements that will be used in sequence – one after the other – similar in behavior to an or between elements, either one is used or the other.</p> <pre data-bbox="513 1278 1268 1839"> <M Name="BasicModule"> <P Name="SimpleProtocol"> <S Name="Avery simplesentence"> <B Name="Firstbuffer intheblock" ASCIIIDefault="This is my default" /> <C Name="Secondconstant inthe block" ASCIIValue="This is my value"/> </S> <S Name="A more complexsentence"> <B Name="First buffer in the block" ASCIIIDefault="This is my default" /> <B Name="Second buffer in the block" ASCIIIDefault="This is my default #2" /> </S> </P> </M> </pre>

Internal functionality

The Sentence Procedure can be used to call functions that are provided by beSTORM's internal mechanisms. Currently, these mechanisms include support for SOCKET operations, including Connect, Receive, Send, Listen and Accept.

Accept – Allows accepting an incoming connection.

Parameter	Description
SOCKET [required]	<p>The previously created SOCKET parameter (should be created by a Listen call).</p> <pre><S Name="SOCKET"> <PC Name="Socket" ConditionedName="Parameters for Connect" Parameter="SOCKET"/> </S></pre>
Timeout	<p>The timeout period between an Accept call and the time beSTORM will report a timeout. By default the value is 5 seconds (values are given in whole seconds), a value of "-1" defines an infinite timeout value (never times out).</p> <pre><S Name="Timeout"> <C Name="Timeout value "ASCIIValue="3" /> </S></pre>
ReceiveOnAccept	<p>Tells beSTORM whether to issue a receive as soon as the connection is accepted or not, valid values are '0' or '1'. By default the value is set to '1'.</p> <pre><S Name="My ReceiveOnAccept" ParamName="ReceiveOnAccept"> <C Name="ReceiveOnAccept" ASCIIValue="0"/> </S></pre>

Parameter	Description
[Return Value]	<p>Five return values are sent back, "ASOCKET" which is passed to any follow up SOCKET operation, "Received" which returns the amount of data returned if the ReceiveOnAccept has been enabled, "Data" which returns the data returned if the ReceviedOnAccept has been enabled, "RETRY" which returns whether an accept operation has timed out and "CounterDown" which reports how many non-open Accept sockets we currently have.</p> <pre><SP Name="Listener" Library="INTERNAL" Procedure="Listen"> <S Name="Protocol Type"> <EV Name="Local Protocol Type" Description="Local Protocol Type" ASCIIValue="udp" /> </S> <S Name="Local Port"></pre>

Connect – Connect to a remote host.

Parameter	Description
Hostname [required]	<p>The hostname, IPv4 address, or IPv6 address to connect to.</p> <pre><S Name="Hostname"> <C Name="Hostname value"<u>ASCIIValue="www.beyondsecurity.com"/> </S></u></pre>
Port [required]	<p>The port with which the Hostname parameter will be accessed, a numerical value should be provided.</p> <pre><S Name="Port"> <C Name="Port value" ASCIIValue="80" /> </S></pre>
Source Port	<p>Some protocols require the source port to be defined (SIP is an example), for those scenarios you can define the Source Port, the ability to define which Source Port will be used only works in a UDP connection, numerical value should be provided.</p> <pre><S Name="My Source Port" ParamName="Source Port"> <C Name="Source Port" ASCIIValue="5060"/> </S></pre>

Parameter	Description
Protocol Type [required]	<p>The protocol type to which beSTORM will listen to, supported types are TCP and UDP.</p> <pre><S Name="My Protocol Type" ParamName="Protocol Type"> <C Name="Protocol Type" ASCIIValue="udp"/> </S></pre>
[Return Value]	<p>Four return values are sent back, "SOCKET" which is passed to any followup SOCKET operation, "RETRY" which returns whether a connection operation has timed out, "Status" which returns whether we failed or were successful in connecting, "Source Port" what source port was used for the connection, and "CounterUp" which reports how many open Connect sockets we currently have.</p> <pre><SP Name="Parameters for Connect" Library="INTERNAL"Procedure="Connect"> <S Name="Protocol Type"> <EV Name="Protocol Type" ASCIIValue="tcp" Description="Remote Protocol Type" Required="1"/> </S> <S Name="Port"> <EV Name="Port" ASCIIValue="179" Description="Remote Port" Required="1"/> </S> <S Name="Hostname"> <EV Name="Hostname" ASCIIValue="192.168.1.1" Description="Remote Hostname" Required="1"/> </S> </SP></pre>

Send/SendEx – Send content to the server (with a 65,535 size limit or 16,777,216 size limit).

Parameter	Description
SOCKET [required]	<p>The pre-created SOCKET parameter (should be created by an Listen call or Connect call).</p> <pre><S Name="SOCKET"> <PC Name="Socket" ConditionedName="Parameters for Connect" Parameter="SOCKET"/> </S></pre>

Parameter	Description
Data [required]	<p>The data that will be sent by way of the pre-created socket.</p> <pre><S Name="Data"> <C Name="Data value"ASCIIValue="Our data"/> </S></pre>
[Return Value]	<p>One return value is sent back, "Sent" which contains how much data was successfully sent.</p> <pre><SP Name="BGP First Packet" Library="INTERNAL" Procedure="SendEx"> <S Name="SOCKET"> <PC Name="Socket" ConditionedName="Parameters for Connect" Parameter="SOCKET"/> </S> <S Name="Data"/> </SP></pre>

Receive/ReceiveEx – Receive content from the server (with a 65,535 size limit or 16,777,216 size limit).

Parameter	Description
SOCKET [required]	<p>The pre-created SOCKET parameter (should be created by an Listen call or Connect call).</p> <pre><S Name="SOCKET"> <PC Name="Socket" ConditionedName="Parameters for Connect" Parameter="SOCKET"/> </S></pre>
Timeout	<p>This defines how long will beSTORM wait for data to come in, value is provided in milliseconds. By default beSTORM will wait for a period of 100 milliseconds.</p> <pre><S Name="Timeout"> <C Name="Timeout value" ASCIIValue="1500"/> </S></pre>
ReceiveDoneOn	<p>This defines for beSTORM whether to continue receiving until this characters are found in the response or a timeout has occurred, or just do a single cycle of receive / wait for a timeout.</p> <pre><S Name="ReceiveDoneOn"> <C Name="220 Marker" ASCIIValue="220"/> </S></pre>

Parameter	Description
ForceReceive	<p>This defines for beSTORM whether a failure to receive will be considered an error and force beSTORM to stop, or whether it will be ignored. Value should be either '0' or '1'. By default beSTORM will ignore failed receives.</p> <pre><S Name="ForceReceive"> <EV Name="ForceReceive" Description="Force receiving data from remote host" ASCIIValue="0" /> </S></pre>
[Return Value]	<p>Three return values are sent back, "RETRY" which returns whether a receive operation has timed out, "CounterDown" which reports how many sockets we currently have open, "Received" which contains the data received from the socket.</p> <pre><SP Name="Parameters for Receive" Library="INTERNAL" Procedure="Receive"> <S Name="SOCKET"> <PC Name="Socket Value" ConditionedName="Parameters for Connect" Parameter="SOCKET" /> </S> <S Name="Timeout"> <EV Name="Timeout Value" ASCIIValue="500" Description="Remote Receive Timeout" /> </S> </SP></pre>

Disconnect – Close an open socket.

Parameter	Description
SOCKET (required)	<p>The pre-created SOCKET parameter (should be created by a Listen call).</p> <pre><S Name="SOCKET"> <PC Name="Socket" ConditionedName="Parameters for Connect" Parameter="SOCKET"/> </S></pre>
[Return Value]	<p>One return value is sent back, "CounterDown" which reports how many non-open Accept sockets we currently have.</p> <pre><S Name="SOCKET"> <PC Name="Socket" ConditionedName="Parameters for Connect" Parameter="SOCKET"/> </S> </SP></pre>

External functionality

As mentioned, the Sentence Procedure can be used to call functions that are provided by code provided outside beSTORM's main source code. Some functionality is shipped with beSTORM's setup file, while others can be provided by the builder of the modules or 3rd parties.

The following external functionality is provided by beSTORM and can be incorporated into custom modules you build:

FILE UTILS.DLL

Using the File Utils.dll's Write and WriteEx procedure you can instruct beSTORM to write the content fuzzed by it to a file. As beSTORM can generate hundreds of thousands of files and more, one of beSTORM's file writer functionalities allows it to create sub-directories under which it will place the files, this to prevent any one directory from containing too many files which might put a burden on the operating system. In addition, beSTORM generates the file names of the files it writes based on the SHA Digest value of the data written, this provides an added performance gain as no duplicate data sets are saved.

Write – Allows writing of fuzzed content into a file (files up to 65530 bytes long).

Parameter	Description
Data [required]	<p>The data that is written by beSTORM's file writer.</p> <pre><S Name="My Data" ParamName="Data"> <C Name="Identify the bitmap file" ASCIIValue="BM" /> <L Name="Size of the bitmap" ConditionedName="Image Data" Size="4" /> <B Name="Reserved" Default="0x00,0x00,0x00,0x00" MaxBytes="4" MinBytes="4"/> </S></pre>

Parameter	Description
Directory Splitter	<p>Defines how many characters will be used to create a sub directory hierarchy.</p> <pre><S Name="Directory Splitter" ParamName="Directory Splitter" > <EV Name="Directory Splitter" Description="Directory Splitter size" ASCIIValue="2" /> </S></pre> <p>NOTE: The sub directory structure is automatically generated by beSTORM.</p>
Path	<p>Defines under which root directory will beSTORM place the files it is about to write.</p> <pre><S Name="Path to save files to" ParamName="Path" > <EV Name="Path" Description="Path to store files" ASCIIValue="C:\Temp\" /> </S></pre> <p>NOTE: The files are not overwritten if they are already present, nor is the root directory created if it does not exist.</p>
Extension	<p>Defines what file extension will be given to the files beSTORM creates.</p> <pre><S Name="File Extension" ParamName="Extension"> <EV Name="Extension" Description="Extension" ASCIIValue="bmp" /> </S></pre>
[Limitations]	<p>This function can only write up to 65,330 bytes of data, use WriteEx for larger files.</p>

Parameter	Description
[Return Value]	<p>No return value.</p> <pre> <SP Name="Writer" Library="File Utils.dll" Procedure="Write"> <S Name="Path to save files to "ParamName="Path" > <EV Name="Path" Description="Path to store files" ASCIIValue="C:\Temp\"/> </S> <S Name="Directory Splitter" ParamName="Directory Splitter"> <EV Name="Directory Splitter" Description="Directory Splitter size" ASCIIValue="2" /> </S>\ <S Name="File Extension" ParamName="Extension" > <EV Name="Extension" Description="Extension" ASCIIValue="bmp"/> </S> <SC Name="My Data" ParamName="Data" > <S Name="Color-mapped images" > <C Name="Identify the bitmap file" ASCIIValue="BM" /> <C Name="Size of the bitmap" Value="0x00,0x00,0x00,0x01" /> <B Name="Reserved" Default="0x00,0x00,0x00,0x00" MaxBytes="4" MinBytes="4"/> </S> </SC> </SP> </pre>

WriteEx – Allows writing of fuzzed content into a file (files up to 16,777,210 bytes long).

Parameter	Description
Data [required]	<p>The data that is written by beSTORM's file writer.</p> <pre> <S Name="My Data" ParamName="Data"> <C Name="Identify the bitmap file" ASCIIValue="BM" /> <L Name="Size of the bitmap" ConditionedName="Image Data" Size="4" /> <B Name="Reserved" Default="0x00,0x00,0x00,0x00" MaxBytes="4" MinBytes="4"/> </S> </pre>

Parameter	Description
Directory Splitter	<p>Defines how many characters will be used to create a sub-directory hierarchy.</p> <pre><S Name="Directory Splitter" ParamName="Directory Splitter"> <EV Name="Directory Splitter" Description="Directory Splitter size" ASCIIValue="2" /> </S></pre> <p>NOTE: The sub directory structure is automatically generated by beSTORM.</p>
Path	<p>Defines under which root directory will beSTORM place the files it is about to write.</p> <pre><S Name="Path" ParamName="Path" > <EV Name="Path" Description="Path to store files" ASCIIValue="C:\Temp\" /> </S></pre> <p>NOTE: The files are not overwritten if they are already present, nor is the root directory created if it does not exist.</p>
Extension	<p>Defines what file extension will be given to the files beSTORM creates.</p> <pre><S Name="File Extension" ParamName="Extension" > <EV Name="Extension" Description="Extension" ASCIIValue="bmp" /> </S></pre>
[Limitations]	This function can only write up to 16,777,210 bytes of data.

Parameter	Description
[Return Value]	<p>No return value.</p> <pre> <SP Name="Writer" Library="File Utils.dll" Procedure="WriteEx"> <S Name="Path" ParamName="Path"> <EV Name="Path" Description="Path to store files"ASCIIValue="C:\Temp\" /> </S> <S Name="Directory Splitter" ParamName="Directory Splitter"> <EV Name="Directory Splitter" Description="Directory Splitter size" ASCIIValue="2" /> </S>\ <S Name="Extension" ParamName="Extension"> <EV Name="Extension" Description="Extension" ASCIIValue="bmp" /> </S> <SC Name="My Data" ParamName="Data"> <S Name="Color-mapped images" > <C Name="Identify the bitmap file" ASCIIValue="BM" /> <C Name="Size of the bitmap" Value="0x00,0x00,0x00,0x01" /> <B Name="Reserved" Default="0x00,0x00,0x00,0x00" MaxBytes="4" MinBytes="4"/> </S> </SC> </SP> </pre>

Using File Utils.dll's **Read** function you can read external content, such as an RSA public key file and use it for beSTORM's internal workings.

Read – Read a file as input for beSTORM.

Parameter	Description
Filename [required]	<p>The name of the file to open, the filename should be provided in ASCII with a full path leading to the file.</p> <pre> <S Name="Filename" ParamName="Filename"> <C Name="Filename to open" ASCIIValue="C:\boot.ini" /> </S> </pre>
[Limitations]	This function can only read up to 65530 bytes of data.

Parameter	Description
[Return Value]	<p>The content of the file that was read.</p> <pre><SP Name="Reader" Library="File Utils.dll" Procedure="Read"> <S Name="Filename" ParamName="Filename"> <EV Name="File to read" Description="Path to read the file from" ASCIIValue="C:\boot.ini" /> </S> </SP></pre>

File Utils.dll provides you access to Device Drivers by way of the functions **Win32CreateFile**, **CloseHandle**, **CtlCode**, and **DeviceIoControl**.

Win32CreateFile – Open a device driver for access (this allows access to Win32CreateFile function).

Parameter	Description
Filename [required]	<p>The name of the file to open, the filename should be provided in ASCII with a full path leading to the file.</p> <pre><S Name="Filename" ParamName="Filename"> <EV Name="Filename value" ASCIIValue="\.\DVWD" Description="CreateFile Filename" /> </S></pre>
DesiredAccess [required]	<p>This provides the CreateFile function with the argument of dwDesiredAccess. See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <pre><S Name="DesiredAccess"> <C Name="DesiredAccess value" Value="C0 00 00 00" /> </S></pre> <p>Example values:</p> <ul style="list-style-type: none"> • GENERIC_READ 80 00 00 00 • GENERIC_WRITE 40 00 00 00 • GENERIC_EXECUTE 20 00 00 00 • GENERIC_ALL 10 00 00 00

Parameter	Description
ShareMode [required]	<p>This provides the CreateFile function with the argument of dwShareMode. See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <pre data-bbox="505 363 1377 457"><SName="ShareMode"> <C Name="ShareModevalue" Value="0000 0007" /> </S></pre> <p>Example values:</p> <ul data-bbox="553 583 1114 709" style="list-style-type: none"> • FILE_SHARE_READ00000001 • FILE_SHARE_WRITE 0000 0002 • FILE_SHARE_DELETE 0000 0004
CreationDisposition [required]	<p>This provides the CreateFile function with the argument of dwShareMode. See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <pre data-bbox="505 886 1435 1020"><SName="CreationDisposition"> <C Name="CreationDispositionvalue" Value="000000 03"/> </S></pre> <p>Example values:</p> <ul data-bbox="553 1142 1114 1367" style="list-style-type: none"> • CREATE_NEW00000001 • CREATE_ALWAYS 0000 0002 • OPEN_EXISTING 00 0000 03 • OPEN_ALWAYS 00 0000 04 • TRUNCATE_EXISTING 0000 0005

Parameter	Description
[Return Value]	<p>A handle to the file that was created.</p> <pre> <SP Name="Win32CreateFile" Procedure="Win32CreateFile" Library="File Utils.dll"> <S Name="Filename"> <EV Name="Filename value" ASCIIValue="\.\DVWD" Description="CreateFile Filename" /> </S> <S Name="DesiredAccess"> <C Name="DesiredAccess value" Value="C0 00 00 00" /> </S> <S Name="ShareMode"> <C Name="ShareMode value" Value="00 00 00 07" /> </S> <S Name="CreationDisposition"> <C Name="CreationDisposition value" Value="00 00 00 03" /> </S> </SP> </pre>

Win32DeviceloControl – Send a control IO to a device driver (this allows access to Win32 DeviceloControl function).

Parameter	Description
HANDLE [required]	<p>The handle of the previously called Win32CreateFile function.</p> <pre> <S Name="HANDLE"> <PC Name="HANDLE" ConditionedName="Win32CreateFile" Parameter="HANDLE"/> </S> </pre>

Parameter	Description
IoControlCode [required]	<p>This provides the DeviceIoControl function with the argument of dwIoControlCode See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>NOTE: The best way to return this generate this value is to use the CtlCode function</p> </div> <pre> <SP Name="IoControlCode" Procedure="Win32CtlCode" Library="File Utils.dll"> <S Name="DeviceType"> <C Name="DeviceType value" Value="00000022" Comment="FILE_DEVICE_UNKNOWN" /> </S> <S Name="Function"> <C Name="Function value" Value="00 00 08 01" /> </S> <S Name="Method"> <C Name="Method value" Value="00 00 00 03" Comment="METHOD_NEITHER" /> </S> <S Name="Access"> <C Name="Access value" Value="00 00 00 03" Comment="FILE_READ_DATA FILE_WRITE_DATA" /> </S> </SP> </pre>
InBuffer [required]	The data you would like to send the device driver,

Win32CtlCode – Send a control IO to a device driver (this allows access to Win32CTL_CODE macro).

Parameter	Description
DeviceType [required]	<p>This provides the CTL_CODE macro with the argument of dwDeviceType. See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <pre> <S Name="DeviceType"> <C Name="DeviceType value" Value="00000022" Comment="FILE_DEVICE_UNKNOWN" /> </S> </pre>

Parameter	Description
Function [required]	<p>This provides the CTL_CODE macro with the argument of dwFunction. See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <pre data-bbox="386 363 1300 457"><S Name="Function"> <C Name="Function value" Value="00 00 08 01" /> </S></pre>
Method [required]	<p>This provides the CTL_CODE macro with the argument of dwMethod. See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <pre data-bbox="386 636 1198 768"><S Name="Method"> <C Name="Method value" Value="00 00 00 03" Comment="METHOD_NEITHER" /> </S></pre>
Access [required]	<p>This provides the CTL_CODE macro with the argument of dwAccess. See Win32 documentation for valid values, the expected size of this value is exactly 4 bytes.</p> <pre data-bbox="386 945 1260 1073"><S Name="Access"> <C Name="Access value" Value="00 00 00 03" Comment="FILE_READ_DATA FILE_WRITE_DATA" /> </S></pre>

Parameter	Description
[Return Value]	<p>Returns the ControlCode that should be supplied to the Win32DeviceIoControl function.</p> <pre> <SP Name="Win32DeviceIoControl" Procedure="Win32DeviceIoControl" Library="File Utils.dll"> <S Name="HANDLE"> <PC Name="HANDLE" ConditionedName="Win32CreateFile" Parameter="HANDLE"/> </S> <S Name="InBuffer"> <B Name="InBuffer value" /> </S> <SP Name="IoControlCode" Procedure="Win32CtlCode" Library="File Utils.dll"> <S Name="DeviceType"> <C Name="DeviceType value" Value="00000022" Comment="FILE_DEVICE_UNKNOWN" /> </S> <S Name="Function"> <C Name="Function value" Value="00 00 08 01" /> </S> <S Name="Method"> <C Name="Method value" Value="00 00 00 03" Comment="METHOD_NEITHER" /> </S> <S Name="Access"> <C Name="Access value" Value="00 00 00 03" Comment="FILE_READ_DATA FILE_WRITE_DATA" /> </S> </SP> </SP> </pre>

Win32CloseHandle – Closes a previously opened device driver (this allows access to Win32 CloseHandle function).

Parameter	Description
HANDLE [required]	<p>The handle of the device driver that you would like to close.</p> <pre> <S Name="HANDLE"> <PC Name="HANDLE" ConditionedName="Win32CreateFile" Parameter="HANDLE"/> </S> </pre>

MATH UTILS.DLL

The Math Utils.dll provides several functions that can be incorporated into beSTORM's modules, this includes:

- Binary2Dec – converts a binary number into its decimal form, i.e. from 0x1020 to 4128
- Dec2Binary – converts a decimal form number to its binary equivalent, i.e. from 4128 to 0x1020
- Random – returns a random number Add – adds two or more numbers Sub – subtracts two numbers
- Multi – multiples two or more numbers Divide – divides two numbers
- ReverseBytes – reverses the order of the bytes it is provided with, i.e. from 0x1021 to 0x1201
- Adler32 – calculates the Adler checksum algorithm of a provided data set
- CRC32 – calculates the cyclic redundancy check algorithm value of a provided data set
- CRC16 – calculates the cyclic redundancy check algorithm value of a provided data set in two forms IBM and CCITT
- ChecksumFletcher – calculates the checksum of a provided data using Fletcher's algorithm
- OR, AND, XOR and NOT – calculates the result of two values of 4 bytes with the logical operator OR, AND, XOR and NOT respectively

Binary2Dec – Allows conversion of a binary value into its decimal form.

Parameter	Description
Data [required]	<p>The data that is converted by beSTORM's binary to decimal function, the largest binary value that can be converted is 0xFFFFFFFF (2³²-1).</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" Value="0x01,0x21" /> </S></pre>

Parameter	Description
Size [required]	<p>The length of the string to return by the function, valid values are based on the decimal conversion that return anything from 1 to 10 characters (for values $2^{32}-1$) long, the function treats the Size parameter it receives as decimal data.</p> <pre><S Name="Size" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S></pre>
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Calculate Binary2Dec" Procedure="Binary2Dec" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="binary value" Value="0x01,0x21"/> </S> <S Name="Size" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S> </SP></pre> <p>Execution of the above sample will return "8449" without the quotes.</p>

Dec2Binary – Allows conversion of a decimal value into its binary form.

Parameter	Description
Data [required]	<p>The data that is converted by beSTORM's decimal to binary function, the largest number that can be converted is $2^{32}-1$ (4,294,967,295).</p> <pre><S Name="Data" ParamName="Data"> <C Name="decimal value" ASCIIValue="289" /> </S></pre>
Size [required]	<p>The length of the binary data to return by the function, valid values are based on the binary conversion that return anything from 1 to 4 bytes (for values $2^{32}-1$) long, the function treats the Size parameter it receives as decimal data.</p> <pre><S Name="Size" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S></pre>

Parameter	Description
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Calculate Dec2Binary" Procedure="Dec2Binary" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="decimal value" ASCIIValue="289" /> </S> <S Name="Size" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S> </SP></pre> <p>Execution of the above sample will return "0x21,0x1,0x00,x00" without the quotes in binary form – the DWORD value of 0x00000121.</p>

Random – Returns a random number.

Parameter	Description
Size [required]	<p>The length of the binary data to be returned by the function, valid values are from 1 to 4 bytes long, the function treats the Size parameter it receives as decimal data.</p> <pre><S Name="Size" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S></pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>NOTE: You cannot control the seed value of the random function.</p> </div>
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Return Random" Procedure="Random" Library="Math Utils.dll"> <S Name="Size" ParamName="Size"> <C Name="size value" ASCIIValue="3" /> </S> </SP></pre> <p>Execution of the above sample will return (in our case) " 0x33,0x12,0x01 " without the quotes in binary form – the DWORD value of 0x011233.</p>

Add – Adds two or more numbers together (equivalent to A+B+C...).

Parameter	Description
A [required]	<p>The data that is added by beSTORM's add function, the function treats the A parameter it receives as binary data, only up to 4 bytes can be added.</p> <pre><S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x12,0x03" /> </S></pre>
B [required]	<p>The data that is added by beSTORM's add function, the function treats the B parameter it receives as binary data, only up to 4 bytes can be added.</p> <pre><S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x30,0x21" /> </S></pre>
Size [required]	<p>The length of the binary data to return by the function, valid values are based on the binary conversion that return anything from 1 to 4 bytes (for values 2 32 -1) long, the function treats the Size parameter it receives as decimal data.</p> <pre><S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S></pre>
C	<p>The data that is added by beSTORM's add function, the function treats the C parameter it receives as binary data, only up to 4 bytes can be added.</p> <pre><S Name="Third Parameter" ParamName="C"> <C Name="C's value" Value="0x11,0x12" /> </S></pre>
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Add Numbers" Procedure="Add" Library="Math Utils.dll"> <S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x12,0x03" /> </S> <S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x30,0x21" /> </S> <S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="3" /> </S> </SP></pre> <p>Execution of the above sample will return " 0x42,0x24,0x00 " without the quotes in binary form – the 3 BYTES value of 0x002442.</p>

Sub – Subtracts two numbers (equivalent to A-B).

Parameter	Description
A [required]	<p>The data that is subtracted by beSTORM's sub function, the function treats the A parameter it receives as binary data, only up to 4 bytes can be subtracted.</p> <pre><S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x01,0x23" /> </S></pre>
B [required]	<p>The data that is subtracted by beSTORM's sub function, the function treats the B parameter it receives as binary data, only up to 4 bytes can be subtracted.</p> <pre><S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x03,0x21" /> </S></pre>
Size [required]	<p>The length of the binary data to return by the function, valid values are based on the binary conversion that return anything from 1 to 4 bytes (for values 2 32 -1) long, the function treats the Size parameter it receives as decimal data.</p> <pre><S Name="Size"> <C Name="size value" ASCIIValue="4" /> </S></pre>
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Subtracts Numbers" Procedure="Sub" Library="Math Utils.dll"> <S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x12,0x03" /> </S> <S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x30,0x21" /> </S> <S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="3" /> </S> </SP></pre> <p>Execution of the above sample will return "0xE2,0xE1,0xFF" without the quotes in binary form – the 3 BYTES value of 0xFFE1E2, note that we have underflowed causing the number to appear with a 0xFF at the rightmost byte.</p>

Multi – Multiplies two or more numbers (equivalent to $A*B*C..$).

Parameter	Description
A [required]	<p>The data that is multiplied by beSTORM's multiplier function, the function treats the A parameter it receives as binary data, only up to 4 bytes can be multiplied.</p> <pre><S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x01,0x23" /> </S></pre>
B [required]	<p>The data that is multiplied by beSTORM's multiplier function, the function treats the B parameter it receives as binary data, only up to 4 bytes can be multiplied.</p> <pre><S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x03,0x21" /> </S></pre>
Size [required]	<p>The length of the multiplied data to return by the function, valid values are based on the binary conversion that return anything from 1 to 4 bytes (for values 2 32 -1) long, the function treats the Size parameter it receives as decimal data.</p> <pre><S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S></pre>
C	<p>The data that is multiplied by beSTORM's multiplier function, the function treats the C parameter it receives as binary data, only up to 4 bytes can be multiplied.</p> <pre><S Name="Third Parameter" ParamName="C"> <C Name="C's value" Value="0x03,0x21" /> </S></pre>

Parameter	Description
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Multiple Numbers" Procedure="Multi" Library="Math Utils.dll"> <S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x12,0x03" /> </S> <S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x30,0x21" /> </S> <S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="3" /> </S> </SP></pre> <p>Execution of the above sample will return "0x60,0xE5,0x65" without the quotes in binary form – the 3 BYTES value of 0x65E560.</p>

Divide – Divides two numbers (equivalent to A/B).

Parameter	Description
A [required]	<p>The data that is divided by beSTORM's divider function, the function treats the A parameter it receives as binary data, only up to 4 bytes can be divided.</p> <pre><S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x01,0x23" /> </S></pre>
B [required]	<p>The data that is divided by beSTORM's divider function, the function treats the B parameter it receives as binary data, only up to 4 bytes can be divided.</p> <pre><S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x03,0x21" /> </S></pre>
Size [required]	<p>The length of the binary data to return by the function, valid values are based on the binary conversion that return anything from 1 to 4 bytes (for values 2 32 -1) long, the function treats the Size parameter it receives as decimal data.</p> <pre><S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="4" /> </S></pre>

Parameter	Description
Zero	<p>The data that is returned by beSTORM's divider function whenever a divide by zero has happened, the function treats the Zero parameter it receives as binary data, only up to 4 bytes can be placed in it. By default the Zero value is returned as -1 (that is, all the returned bits are marked with the bit 1 (one)).</p> <pre data-bbox="386 432 1219 531"><S Name="Zero" ParamName="Zero"> <C Name="Zero's value" Value="0xFF,0xFF" /> </S></pre>
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre data-bbox="386 632 1235 1052"><SP Name="Divide Numbers" Procedure="Divide" Library="Math Utils.dll"> <S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x12,0x03" /> </S> <S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x30,0x21" /> </S> <S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="3" /> </S> </SP></pre> <p>Execution of this sample will return " 0x00,0x00,0x00 " without the quotes in binary form – the 3 BYTES value of 0x000000.</p> <pre data-bbox="386 1209 1295 1629"><SP Name="Divide Numbers #2" Procedure="Divide" Library="Math Utils.dll"> <S Name="Second Parameter" ParamName="B"> <C Name="B's value" Value="0x12,0x03" /> </S> <S Name="First Parameter" ParamName="A"> <C Name="A's value" Value="0x30,0x21" /> </S> <S Name="Size Parameter" ParamName="Size"> <C Name="size value" ASCIIValue="3" /> </S> </SP></pre> <p>Execution of the above sample will return " 0x0A,0x00,0x00 " without the quotes in binary form – the 3 BYTES value of 0x00000A.</p>

ReverseBytes – Reverse the bytes of one or more bytes (equivalent to little endian to big endian conversion and visa versa).

Parameter	Description
Data [required]	<p>The data that is reversed by beSTORM's reverse bytes function, the function treats the Data parameter it receives as binary data, up to 4 bytes can be reversed.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Data's value" Value="0x01,0x02,0x03" /> </S></pre>
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Reverse the Bytes" Procedure="ReverseBytes" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="Data's value" Value="0x01,0x02,0x03" /> </S> </SP></pre> <p>Execution of the above sample will return " 0x03,0x02,0x01 " without the quotes in binary form – the 3 BYTES value of 0x010203.</p>

Adler32 – Calculates the Adler32 value of a provided data set (Based on <http://en.wikipedia.org/wiki/Adler-32>).

Parameter	Description
Data [required]	<p>The data that is provided to the Adler32 algorithm, the function treats the Data parameter it receives as binary data, there is no limit to the incoming data set size.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="Return in binary form the Adler32 value (4 bytes)" /> </S></pre>

Parameter	Description
[Return value]	<p>The ' Output ' value contains the content returned by the function.</p> <pre><SP Name="Adler32 Calculation" Procedure="Adler32" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="Return in binary form the Adler32 value (4 bytes)" /> </S> </SP></pre> <p>Execution of the above sample will return " 0xE8,0x10,0x36,0xB6 " without the quotes in binary form – the DWORD value of 0xB63610E8.</p>

CRC32 – Calculates the CRC32 value of a provided data set (Based on <http://en.wikipedia.org/wiki/CRC-32>).

Parameter	Description
Data [required]	<p>The data that is provided to the CRC32 algorithm, the function treats the Data parameter it receives as binary data, there is no limit to the incoming data set size.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="Return in binary form the CRC32 value (4 bytes)" /> </S></pre>
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="CRC32 Calculation" Procedure="CRC32" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="Return in binary form the CRC32 value (4 bytes)" /> </S> </SP></pre> <p>Execution of the above sample will return "0x4F,0xDA,0xA8,0xE8" without the quotes in binary form – the DWORD value of 0xE8A8DA4F.</p>

CRC16CCITT – CRC16 CCITT calculates the value of CRC16 using the following calculation:
 $x^{16} + x^{12} + x^5 + 1$.

Parameter	Description
Data [required]	<p>The data that is provided to the CRC16 CCITT algorithm, the function treats the Data parameter it receives as binary data, there is no limit to the incoming data set size.</p> <pre><SName="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="Return in binary form the CRC16 value (2 bytes)" /> </S></pre>
Initial	<p>CRC16 supports a variety of initializing values, by default CRC16 CCITT sets the initializing value to 0x0000 (zero). There are several commonly used initializing values, 0x0000, 0x1D0F and 0xFFFF.</p> <pre><S Name="Initial" ParamName="Initial"> <C Name="Initial's value" Value="0x1D, 0x0F" /> </S></pre>
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="CRC16 CCITT Calculation" Procedure="CRC16CCITT" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="123456789" /> </S> </SP></pre> <p>Execution of the above sample will return "0x31,0xC3" without the quotes in binary form – the WORD value of 0xC331.</p>

CRC16IBM – CRC16 IBM calculates the value of CRC16 using the following calculation: $x^{16} + x^{15} + x^2 + 1$.

Parameter	Description
Data [required]	<p>The data that is provided to the CRC16 IBM algorithm, the function treats the Data parameter it receives as binary data, there is no limit to the incoming data set size.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="Return in binary form the CRC16 value (2 bytes)" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="CRC16 IBM Calculation" Procedure="CRC16IBM" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="123456789" /> </S> </SP></pre> <p>Execution of the above sample will return "0xBB,0x3D" without the quotes in binary form – the WORD value of 0x3DBB.</p>

ChecksumFletcher – Fletcher checksum calculates the checksum value of the data using the Fletcher algorithm.

Parameter	Description
Data [required]	<p>The data that is provided to the Fletcher checksum algorithm, the function treats the Data parameter it receives as binary data, there is no limit to the incoming data set size.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="Return in binary form the Checksum result using the Fletcher algorithm (2 bytes)" /> </S></pre>
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="Fletcher Calculation" Procedure="ChecksumFletcher" Library="Math Utils.dll"> <S Name="Data" ParamName="Data"> <C Name="Data's value" ASCIIValue="123456789" /> </S> </SP></pre> <p>Execution of the above sample will return " 0xDD,0x01 " without the quotes in binary form – the WORD value of 0x01DD.</p>

OPENSSL UTILS.DLL

The OpenSSL Utils.dll provides several functions that can be incorporated into beSTORM's modules, this includes:

- MD2Hash – Calculate the MD2 value of a data set
- MD4Hash – Calculate the MD4 value of a data set
- MD5Hash – Calculate the MD5 value of a data set
- SHA1Hash – Calculate the SHA1 value of a data set
- SHA224Hash – Calculate the SHA224 value of a data set
- SHA256hash – Calculate the SHA256 value of a data set
- SHA384Hash – Calculate the SHA384 value of a data set
- SHA512Hash – Calculate the SHA384 value of a data set
- Base64Encode – Return the base64 encoded form of a data set
- Base64Decode – Return the plain text form from a base64 encoded data set
- HexConvert – Convert a given data set's binary form to hexadecimal string form
- AESECBEncrypt – Encrypts the data provided to it using the AES (Advanced Encryption Standard) ECB mode (Electronic Cookbook)
- AESECBDecrypt – Decrypts the data provided to it using the AES (Advanced Encryption Standard) ECB mode (Electronic Cookbook)
- AESCBCEncrypt – Encrypts the data provided to it using the AES (Advanced Encryption Standard) CBC mode (Cipher-block chaining)
- AESCBCDecrypt – Decrypts the data provided to it using the AES (Advanced Encryption Standard) CBC mode (Cipher-block chaining)
- AESCFBEncrypt – Encrypts the data provided to it using the AES (Advanced Encryption Standard) CFB mode (Cipher feedback)
- AESCFBDecrypt – Decrypts the data provided to it using the AES (Advanced Encryption Standard) CFB mode (Cipher feedback)
- AESOFBEncrypt – Encrypts the data provided to it using the AES (Advanced Encryption Standard) OFB mode (Output feedback)
- AESOFBDecrypt – Decrypts the data provided to it using the AES (Advanced Encryption Standard) OFB mode (Output feedback)
- RSAPublicEncrypt – Encrypts the data provided to it using the standard RSA (Ron Rivest, Adi Shamir and Leonard Adleman) proposed algorithm
- RSAPrivateDecrypt – Decrypts the data provided to it using the standard RSA (RonRivest, Adi Shamir and Leonard Adleman) proposed algorithm
- HMAC_MD5 – Hashes the provided data with the HMAC MD5 algorithm
- HMAC_SHA1 – Hashes the provided data with the HMAC SHA1 algorithm

MD2Hash, MD4Hash, and MD5 – Calculate the MD2, MD4, and, MD5 value of a dataset.

Parameter	Description
Data [required]	<p>The data that is calculated for its MD2, MD4 and MD5 value by beSTORM's MD2, MD4, and MD5 hashing function.</p> <pre><SName="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="There is no limit to the incoming data, outgoing data is always 16 bytes long" /> </S></pre>
[Return value]	The 'Output' value contains the content returned by the function.

SHA1Hash, SHA224Hash, SHA256Hash, SHA384Hash, and SHA512Hash – Calculate the SHA1, SHA224, SHA256, SHA384, and SHA512 value of a data set.

Parameter	Description
Data [required]	<p>The data that is calculated for its SHA1, SHA224, SHA256, SHA384, and SHA512 value by beSTORM's SHA1, SHA224, SHA256, SHA384, and SHA512 hashing function, the return value is receptively 20, 28,32, 48, and 64 bytes long.</p> <pre><SName="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="There is no limit to the incoming data, outgoing data is depended on the hashing function called" /> </S></pre>
[Return value]	The 'Output' value contains the content returned by the function.

Base64Encode – Encode a given dataset into its base64 encoded form.

Parameter	Description
Data [required]	<p>The data that is encoded with the Base64 algorithm.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="There is no limit to the incoming data, outgoing data is dependent on the type of data that comes in" /> </S></pre>
[Return value]	The 'Output' value contains the content returned by the function.

HexConvert – Converts a binary stream into its hexadecimal equivalent.

Parameter	Description
Data [required]	<p>The data that is converted into its hexadecimal form, binary data in the form of "0x02,0x30,0x10,0x20" (as it is provided to beSTORM) will return as 02301020.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="1234" /> </S></pre> <p>Will return as 31323334.</p>
[Return value]	The 'Output' value contains the content returned by the function.

AESECBEncrypt and **AESECBDecrypt** – Encrypts and decrypts the provided data using the key respectively. Key length can be either 16 (128), 24 (192) or 32 (256) bytes (bits) long.

Parameter	Description
Data [required]	<p>The provided data is either encrypted or decrypted using the key value given.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Value to use the AES ECB algorithm on" ASCIIValue="Protect me" /> </S></pre>
Key [required]	<p>The provided key used for both encrypting and decrypting. The Key value needs to be either 16, 24 or 32 bytes long depending on the value of KeySize given to the function. If the provided Key value shorter than required, the Key is padded with NULL (0x00) characters.</p> <pre><S Name="Key" ParamName="Key"> <C Name="The Key that is used by the AES EBC algorithm (16 bytes in this case)" ASCIIValue="1234567890123456" /> </S></pre>

Parameter	Description
KeySize [required]	<p>The key size that is to be used with the AES ECB algorithm. Valid values are 16, 24 or 32 in their ASCII form. If the value of KeySize does not match 16, 24 or 32 the value of 16 is used.</p> <pre><S Name="Key Size" ParamName="KeySize"> <C Name="The key size we desire" ASCIIValue="16" /> </S></pre>
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="AESECB Round" Procedure="AESECBDecrypt" Library="OpenSSL Interface.dll"> <SP Name="Data" Procedure="AESECBEncrypt" Library="OpenSSL Interface.dll"> <S Name="Data" ParamName="Data"> <C Name="ascii value" ASCIIValue="Encrypt Me" /> </S> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"></pre>

AESCBCEncrypt and **AESCBCDecrypt** – Encrypts and decrypts the provided data using the key respectively. Key length can be either 16 (128), 24 (192) or 32 (256) bytes (bits) long.

Parameter	Description
Data [required]	<p>The provided data is either encrypted or decrypted using the key value given.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Value to use the AES CBC algorithm on" ASCIIValue="Protect me" /> </S></pre>
Key [required]	<p>The provided key used for both encrypting and decrypting. The Key value needs to be either 16, 24 or 32 bytes long depending on the value of KeySize given to the function. If the provided Key value shorter than required, the Key is padded with NULL (0x00) characters.</p> <pre><S Name="Key" ParamName="Key"> <C Name="The Key that is used by the AES CBC algorithm (16 bytes in this case)" ASCIIValue="1234567890123456" /> </S></pre>

Parameter	Description
KeySize [required]	<p>The key size that is to be used with the AES CBC algorithm. Valid values are 16, 24 or 32 in their ASCII form. If the value of KeySize does not match 16, 24 or 32 the value of 16 is used.</p> <pre><S Name="Key Size" ParamName="KeySize"> <C Name="The key size we desire" ASCIIValue="16" /> </S></pre>
IV	<p>The initial vector used for encryption. The length of the initial vector can be up to 16 bytes long, if no value is provided for it the initial vector of 0 (zero) is used. If the IV value provided is shorter than 16 bytes, the IV is padded with NULL (0x00) characters.</p> <pre><S Name="IV" ParamName="IV"> <C Name="The IV value we want to use" ASCIIValue="1234567890123456" /> </S></pre>
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="AESCBC Round" Procedure="AESCBCDecrypt" Library="OpenSSL Interface.dll"> <SP Name="Data" Procedure="AESCBCEncrypt" Library="OpenSSL Interface.dll"> <S Name="Data" ParamName="Data"> <C Name="ascii value" ASCIIValue="Encrypt Me" /> </S> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP></pre>

AESCFBEncrypt and **AESCFBDecrypt** – Encrypts and decrypts the provided data using the key respectively. Key length can be either 16 (128), 24 (192) or 32(256) bytes (bits) long.

Parameter	Description
Data [required]	<p>The provided data is either encrypted or decrypted using the key value given.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Value to use the AES CFB algorithm on" ASCIIValue="Protect me" /> </S></pre>
Key [required]	<p>The provided key used for both encrypting and decrypting. The Key value needs to be either 16, 24 or 32 bytes long depending on the value of KeySize given to the function. If the provided Key value shorter than required, the Key is padded with NULL (0x00) characters.</p> <pre><S Name="Key" ParamName="Key"> <C Name="The Key that is used by the AES CFB algorithm (16 bytes in this case)" ASCIIValue="1234567890123456" /> </S></pre>
KeySize [required]	<p>The key size that is to be used with the AES CFB algorithm. Valid values are 16, 24 or 32 in their ASCII form. If the value of KeySize does not match 16, 24 or 32 the value of 16 is used.</p> <pre><S Name="Key Size" ParamName="KeySize"> <C Name="The key size we desire" ASCIIValue="16" /> </S></pre>
IV	<p>The initial vector used for encryption. The length of the initial vector can be up to 16 bytes long, if no value is provided for it the initial vector of 0 (zero) is used. If the IV value provided is shorter than 16 bytes, the IV is padded with NULL (0x00) characters.</p> <pre><S Name="IV" ParamName="IV"> <C Name="The IV value we want to use" ASCIIValue="1234567890123456" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre> <SP Name="AESCFB Round" Procedure="AESCFBDecrypt" Library="OpenSSL Interface.dll"> <SP Name="Data" Procedure="AESCFBEncrypt" Library="OpenSSL Interface.dll"> <S Name="Data" ParamName="Data"> <C Name="ascii value" ASCIIValue="Encrypt Me" /> </S> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP> </pre>

AESCBCEncrypt and **AESCBCDecrypt** – Encrypts and decrypts the provided data using the key respectively. Key length can be either 16 (128), 24 (192) or 32 (256) bytes (bits) long.

Parameter	Description
Data [required]	<p>The provided data is either encrypted or decrypted using the key value given.</p> <pre> <S Name="Data" ParamName="Data"> <C Name="Value to use the AES CBC algorithm on" ASCIIValue="Protect me" /> </S> </pre>

Parameter	Description
Key [required]	<p>The provided key used for both encrypting and decrypting. The Key value needs to be either 16, 24 or 32 bytes long depending on the value of KeySize given to the function. If the provided Key value shorter than required, the Key is padded with NULL (0x00) characters.</p> <pre><S Name="Key" ParamName="Key"> <C Name="The Key that is used by the AES CBC algorithm (16 bytes in this case)" ASCIIValue="1234567890123456" /> </S></pre>
KeySize [required]	<p>The key size that is to be used with the AES CBC algorithm. Valid values are 16, 24 or 32 in their ASCII form. If the value of KeySize does not match 16, 24 or 32 the value of 16 is used.</p> <pre><S Name="Key Size" ParamName="KeySize"> <C Name="The key size we desire" ASCIIValue="16" /> </S></pre>
IV	<p>The initial vector used for encryption. The length of the initial vector can be up to 16 bytes long, if no value is provided for it the initial vector of 0 (zero) is used. If the IV value provided is shorter than 16 bytes, the IV is padded with NULL (0x00) characters.</p> <pre><S Name="IV" ParamName="IV"> <C Name="The IV value we want to use" ASCIIValue="1234567890123456" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre> <SP Name="AESCFB Round" Procedure="AESCBCDecrypt" Library="OpenSSL Interface.dll"> <SP Name="Data" Procedure="AESCBCEncrypt" Library="OpenSSL Interface.dll"> <S Name="Data" ParamName="Data"> <C Name="ascii value" ASCIIValue="Encrypt Me" /> </S> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP> </pre>

AESOFBEncrypt and **AESOFBDecrypt** – Encrypts and decrypts the provided data using the key respectively. Key length can be either 16 (128), 24 (192) or 32 (256) bytes (bits) long.

Parameter	Description
Data [required]	<p>The provided data is either encrypted or decrypted using the key value given.</p> <pre> <S Name="Data" ParamName="Data"> <C Name="Value to use the AES OFB algorithm on" ASCIIValue="Protect me" /> </S> </pre>

Parameter	Description
Key [required]	<p>The provided key used for both encrypting and decrypting. The Key value needs to be either 16, 24 or 32 bytes long depending on the value of KeySize given to the function. If the provided Key value shorter than required, the Key is padded with NULL (0x00) characters.</p> <pre data-bbox="386 401 1443 569"><S Name="Key" ParamName="Key"> <C Name="The Key that is used by the AES OFB algorithm (16 bytes in this case)" ASCIIValue="1234567890123456" /> </S></pre>
KeySize [required]	<p>The key size that is to be used with the AES OFB algorithm. Valid values are 16, 24 or 32 in their ASCII form. If the value of KeySize does not match 16, 24 or 32 the value of 16 is used.</p> <pre data-bbox="386 741 1378 842"><S Name="Key Size"ParamName="KeySize"> <C Name="The key size we desire" ASCIIValue="16" /> </S></pre>
IV	<p>The initial vector used for encryption. The length of the initial vector can be up to 16 bytes long, if no value is provided for it the initial vector of 0 (zero) is used. If the IV value provided is shorter than 16 bytes, the IV is padded with NULL (0x00) characters.</p> <pre data-bbox="386 1052 1086 1184"><S Name="IV" ParamName="IV"> <C Name="The IV value we want to use "ASCIIValue="1234567890123456" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre> <SP Name="AESCFB Round" Procedure="AESOFBDecrypt" Library="OpenSSL Interface.dll"> <SP Name="Data" Procedure="AESOFBEncrypt" Library="OpenSSL Interface.dll"> <S Name="Data" ParamName="Data"> <C Name="ascii value" ASCIIValue="Encrypt Me" /> </S> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP> <S Name="Key" ParamName="Key"> <C Name="key value" ASCIIValue="This is a key 16" /> </S> <S Name="Key Size" ParamName="KeySize"> <C Name="keySize value" ASCIIValue="24" /> </S> </SP> </pre>

RSAPublicEncrypt – Encrypts the provided data using the public key. RSA encryption's padding method is set to RSA_PKCS1_OAEP_PADDING.

Parameter	Description
Data [required]	<p>The provided data is either encrypted using the key value given.</p> <pre> <S Name="Data" ParamName="Data"> <C Name="Value to use the RSA Public Key encryption" ASCIIValue="Protect me" /> </S> </pre>

Parameter	Description
PublicKey [required]	<p>The provided public key used for encryption. The key is either what is referred to by OpenPGP as armored key, base 64 encoded form of the key, or its binary form (BER encoded).</p> <pre data-bbox="370 363 1438 709"> <S Name="PublicKey" ParamName="Public-Key"> <C Name="Key" ASCIIValue="-----BEGIN RSA PUBLIC KEY----- - MIGHAoGBALPLeJAQ7+Rl5ZL3Bb1USsLfgEoT/5eeQkpTO6rL0is3m5D zr+oeUrIQ+ep8ZxNC k041Bcp81JJ+OYXY8sRYoirAP2qbGie6SxrasFsK0jhJN/z53tHYnRr w8Ry9wlVDh7v06IqRbRL +byHIJJyDWF7AQcFoXgMbc6zCLg08+9nzAgED-----END RSA PUBLICKEY-----" /> </S> </pre>

Parameter	Description
PublicKeyMode	<p>The public key mode sets whether the Public Key is in its base64 encoded form or in its binary form. A value of 0x00 (zero) tells beSTORM that we are going to provide a binary form public key, while a value of 0x01 (one) tells beSTORM that we are going to provide a base 64 encoded form of the public key. Base64 encoded form supports unlimited length of public keys, while the binary form supports up to 128 bytes long key. By default the Public Key Mode is textual (base64 encoded).</p> <pre> <S Name="PublicKeyMode" ParamName="Public-Key-Mode" > <C Name="Value" Value="0x00" /> </S> <S Name="PublicKey" ParamName="Public-Key" > <C Name="Key" Value="0xb3, 0xcb, 0x78, 0x90, 0x10, 0xef, 0xe4, 0x65, 0xe5, 0x92, 0xf7, 0x05, 0xbd, 0x54, 0x4a, 0xc2, 0xdf, 0x80, 0x4a, 0x13, 0xff, 0x97, 0x9e, 0x42, 0x4a, 0x53, 0x3b, 0xaa, 0xcb, 0xd2, 0x2b, 0x37, 0x9b, 0x90, 0xf3, 0xaf, 0xea, 0x1e, 0x52, 0xb2, 0x10, 0xf9, 0xea, 0x7c, 0x67, 0x13, 0x42, 0x93, 0x4e, 0x35, 0x05, 0xca, 0x7c, 0xd4, 0x92, 0x7e, 0x39, 0x85, 0xd8, 0xf2, 0xc4, 0x58, 0xa2, 0x2a, 0xc0, 0x3f, 0x6a, 0x9b, 0x18, 0x87, 0xba, 0x4b, 0x1a, 0xda, 0xb0, 0x5b, 0x0a, 0xd2, 0x38, 0x49, 0x37, 0xfc, 0xf9, 0xde, 0xd1, 0xd8, 0x9d, 0x1a, 0xf0, 0xf1, 0x1c, 0xbd, 0xc2, 0x55, 0x43, 0x87, 0xbb, 0xf4, 0xe8, 0x8a, 0x91, 0x6d, 0x12, 0xfe, 0x6f, 0x21, 0xc8, 0x24, 0x9c, 0x83, 0x58, 0x5e, 0xc0, 0x41, 0xc1, 0x68, 0x5e, 0x03, 0x1b, 0x73, 0xac, 0xc2, 0x2e, 0x0d, 0x3c, 0xfb, 0xd9, 0xf3" /> </S> </pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre> <SP Name="Data" Procedure="RSAPublicEncrypt" Library="OpenSSL Interface.dll"> <S Name="PublicKeyMode" ParamName="Public-Key-Mode"> <C Name="Value" Value="0x00" /> </S> <S Name="PublicKey" ParamName="Public-Key"> <C Name="Key" Value="0xb3, 0xcb, 0x78, 0x90, 0x10, 0xef, 0xe4, 0x65, 0xe5, 0x92, 0xf7, 0x05, 0xbd, 0x54, 0x4a, 0xc2, 0xdf, 0x80, 0x4a, 0x13, 0xff, 0x97, 0x9e, 0x42, 0x4a, 0x53, 0x3b, 0xaa, 0xcb, 0xd2, 0x2b, 0x37, 0x9b, 0x90, 0xf3, 0xaf, 0xea, 0x1e, 0x52, 0xb2, 0x10, 0xf9, 0xea, 0x7c, 0x67, 0x13, 0x42, 0x93, 0x4e, 0x35, 0x05, 0xca, 0x7c, 0xd4, 0x92, 0x7e, 0x39, 0x85, 0xd8, 0xf2, 0xc4, 0x58, 0xa2, 0x2a, 0xc0, 0x3f, 0x6a, 0x9b, 0x18, 0x87, 0xba, 0x4b, 0x1a, 0xda, 0xb0, 0x5b, 0x0a, 0xd2, 0x38, 0x49, 0x37, 0xfc, 0xf9, 0xde, 0xd1, 0xd8, 0x9d, 0x1a, 0xf0, 0xf1, 0x1c, 0xbd, 0xc2, 0x55, 0x43, 0x87, 0xbb, 0xf4, 0xe8, 0x8a, 0x91, 0x6d, 0x12, 0xfe, 0x6f, 0x21, 0xc8, 0x24, 0x9c, 0x83, 0x58, 0x5e, 0xc0, 0x41, 0xc1, 0x68, 0x5e, 0x03, 0x1b, 0x73, 0xac, 0xc2, 0x2e, 0x0d, 0x3c, 0xfb, 0xd9, 0xf3" /> </S> <S Name="Data" ParamName="Data"> <C Name="Encrypt me" ASCIIValue="Please encrypt this data for me and show me it in plaintext" /> </SP> </S> </pre>

RSAPrivateDecrypt – Decrypts the provided data using the private key. RSAdecryption's padding method is set to RSA_PKCS1_PADDING.

Parameter	Description
Data [required]	<p>The provided data is either decrypted using the key value given.</p> <pre> <S Name="Data" ParamName="Data"> <C Name="Value to use the RSA Private Key decryption" ASCIIValue="Unscarmble me" /> </S> </pre>

Parameter	Description
Private Key [required]	<p>The provided private key used for decryption. The key is either what is referred to by OpenPGP as armored key, base64 encoded form of the key, or its binary form (BER encoded).</p> <pre> <S Name="PrivateKey" ParamName="Private-Key"> <C Name="Key" ASCIIValue="-----BEGIN RSA PRIVATE KEY----- MIICWwIBAAKBgQCzy3iQE0/kZeWS9wW9VErC34BKE/+XnkJKUzuqy9IrN5u Q86/qHlK yEPnqfGcTQpNONQXKfNSSfjmF2PLEWKIqwD9qmxihuksa2rBbCtI4STf8+d 7R2J0a8PE cvcJVQ4e790iKkW0S/m8hyCScg1hewEHBaF4DG30swi4NPPv Z8wIBAwKBgHfc+wq19ULumQykrn44Mdc/qtwNVQ++1tw3fRyH4XIlEmCidU a+4cwLUUb 9mgzXDN70A9xTOGGpe66Qodg7FsYMQVSedzkxUjUj+mawv JU1Rr917i8NRINDmpQD1rV8l2xAPMzxpP9Bx8f6hSx71bLTMty5Ft83bULO 9yZiDGp7AkeA 4+m1PcsgwI3+UGxwab348dVzxJzEGC+JyofOc8+kWJni7krDw ETkGgcg5euOX9L5spf/Wzd5FIQBkmdnrmoxwJBAMnztm+akS/BzNRMUJf5 X594agRcxxy GzBoBcEQwoLAKtqZCk2bmsAG8VOpXS2nE2NDddveEOzPLx v1T7AsvkXUCQQCX8SN+h2srCVQ1naBGfqX2jk0tvdgQH7Exr97338Llu+ye 3IKALelmr2tD8l 7qjKZ3D/+SJPtjAqu275pp0RsvAkEAhqJ5n7xgyoEzODLgZVDqa lBGrZMvcwSIEVZK2CBriAckbtcm70R1Vn2N8aDc8S3l4JOkpQLSIofZ+41I B3ULowJAGz9t LqbixQteNwtGJ3v1Gy5wE7XNU6SbWSC0lbfWj+i6d9x/JmgMFX0X crnj/4ExhQK8iUZL2JgDw0XA+JvEpw== -----END RSA PRIVATE KEY " /> </S> </pre>

Parameter	Description
Private Key Mode	<p>The private key mode sets whether the Private Key is in its base 64 encoded form or in its binary form. A value of 0x00 (zero) tells beSTORM that we are going to provide a binary form private key, while a value of 0x01 (one) tells beSTORM that we are going to provide a base 64 encoded form of the private key. Base64 encoded form supports unlimited length of private keys, while the binary form supports up to 128 bytes long key. By default the Private Key Mode is textual (base64 encoded).</p> <pre> <S Name="PrivateKeyMode" ParamName="Private-Key-Mode" > <C Name="Value" Value="0x01" /> </S> <S Name="PrivateKey" ParamName="Private-Key"> <C Name="Key" ASCIIValue="-----BEGIN RSA PRIVATE KEY----- MIICWwIBAAKBgQCzy3iQEO/kZeWS9wW9VErC34BKE/+XnkJKUzu qy9IrN5uQ86/qHlKyEPnqfGcTQpNONQXKfNSSfjmF2PLEWKIqwD9qmxiHuk sa2rBbCtI4STf8+d7R 2J0a8PEcvcJVQ4e79OiKkW0S/m8hyCScg1hewEHBaF4DG3Oswi4NPPvZ8wI BAwKB gHfc+wq19ULumQykrn44Mdc/qtwnVQ++1tw3fRyH4XIleMcidUa+4cwLUUb 9mgzXDN7OA9xTOG Gpe66Qodg7FsYMQVSedzkxUjUj+mawvJU1Rr917i8NRINDmpQD1rV8l2xAP MzxgP9Bx8f6hSx71bL TMty5Ft83bULO9yZiDGp7AkEA4+m1PcsgwI3+UGxwab348dVzxJzEGC+Jyo fOc8+kWJni7krDwETk Ggcg5euOX9L5spf/Wzd5FIQBkmdnrmoxwJBAMnztm+akS/BzNRMUJf5X59 4agRcxxyGzBoBcEQw oLAKtqZCk2bmsAG8VOpXS2nE2NDddveEOzPLxv1T7AsvkXUCQQCX8SN+h2s rCVQ1naBGfqX2jk0t vdgQH7Exr97338Llu+ye3IKALe1mr2tD817qjKZ3D/+SJPtjAq275pp0Rs vAkeAhqJ5n7xgyoEzODLgZV DqalBGrZMvcwSIEVZK2CBRIAckbtcm70R1Vn2N8aDc8S314JOkpQLSIofZ+ 41IB3ULowJAGz9tLqbixQ teNwtGJ3v1Gy5wE7XNU6SbWSc0lbfWj+i6d9x/JmgMFX0Xcrnj/4ExhQK8i UZL2JgDw0XA+JvEpw== -----END RSA PRIVATE KEY " /> </S> </pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre> <SP Name="RSAEncrypt Test" Procedure="RSAPrivateDecrypt" Library="OpenSSL Interface.dll"> <S Name="PrivateKey" ParamName="Private-Key"> <C Name="Key" ASCIIValue="-----BEGIN RSA PRIVATE KEY----- MIICWwIBAAKBgQCzy3iQEO/kZeWS9wW9VErC34BKE/+XnkJKUzuqy9IrN5u Q86/qHlKyEPnqfGcTQpNONQXKfNSSfjmF2PLEWKIqwD9qmxihuksa2rBbCt I4STf8+d7R 2J0a8PEcvcJVQ4e790iKkW0S/m8hyCScglhewEHBaF4DG3Oswi4NPPvZ8wI BAwKB gHfc+wq19ULumQykrn44Mdc/qtwNVQ++1tw3fRyH4XI1EmCidUa+4cwLUUb 9mgzXDN70A9xTO GGpe66Qodg7FsYMQVSedzkxUjUj+mawvJU1Rr917i8NRINdmpQD1rV8l2xA PMzxpP9Bx8f6hSx71 bLTMTy5Ft83bUL09yZiDGp7AkEA4+m1PcsgwI3+UGxwab348dVzxJzEGC+J yofOc8+kWJni7krDw ETkGgcg5euOX9L5spf/Wzd5FIQBkmdnrmoxwJBAMnztm+akS/BzNRMUJf5 X594agRcxxyGzBoBc EQwoLAKtqZCk2bmsAG8VOpXS2nE2NDddveEOzPLxv1T7AsvkXUCQQCX8SN+ h2srCVQ1naBGfqX2 jk0tvdgQH7Exr97338Llu+ye3IKALelmr2tD8l7qjKZ3D/+SJPtjAqu275p p0RsvAkeAhqJ5n7xgyoEzODLgZ VDqalBGrZMvcsIEVZK2CBRIAckbtcm70R1Vn2N8aDc8S3l4JOkpQLSIofZ +4lIB3ULowJAGz9tLqbixQt eNwtGJ3v1Gy5wE7XNU6SbWSc0lbfWj+i6d9x/JmgMFX0Xcrnj/4ExhQK8iU ZL2JgDw0XA+JvEpw== -----END RSA PRIVATE KEY " /> </S> <SP Name="Data" Procedure="RSAPublicEncrypt" Library="OpenSSL Interface.dll"> <S Name="PublicKey" ParamName="Public-Key"> <C Name="Key" ASCIIValue="-----BEGIN RSA PUBLIC KEY----- MIGHAoGBALPLeJAQ7+Rl5ZL3Bb1USsLfgEoT/5eeQkpTO6rL0is3m5Dzr+o eUrIQ +ep8ZxNck041Bcp81JJ+OYXY8sRYoirAP2qbGIE6SxrasFsK0jhJN/z53tH YnRrw8Ry 9wlVDh7v06IqRbRL+byHIJjyDWF7AQcFoXgMbc6zCLg08+9nzAgED -----END RSA PUBLIC KEY " /></S> <S Name="Data" ParamName="Data"> <C Name="Encrypt me" ASCIIValue="Please encrypt this data for me and show me it in plaintext" /> </S> </SP> </SP> </pre>

HMAC_MD5 and **HMAC_SHA1** – Generates a keyed-hash message authentication code using MD5 or SHA1 respectively as the supporting algorithm.

Parameter	Description
Data [required]	<p>The provided data is either decrypted using the key value given.</p> <pre><S Name="Data" ParamName="Data"> <C Name="Value to use the RSA Public Key encryption" ASCIIValue="Protect me" /> </S></pre>
Key [required]	<p>The key to be used by the HMAC algorithm for calculation of the keyed-hash message value.</p> <pre><S Name="Key" ParamName="Key"> <C Name="The key to use with HMAC" ASCIIValue="My key is my password" /> </S></pre>
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="HMAC_SHA1 Compute" Library="OpenSSL Interface.dll" Procedure="HMAC_SHA1"> <S Name="Key" ParamName="Key"> <C Name="The Key is the password" ASCIIValue="MyPassword" /> </S> <S Name="Data" ParamName="Data"> <C Name="The data to compute for" ASCIIValue="MyUsername" /> </S> </SP></pre>

STREAM UTILS.DLL

The Stream Utils.dll provides several functions that can be incorporated into beSTORM's modules, this includes:

- Left – Extracts up to the rightmost location of a provided data set
- Right – Extracts from the leftmost location of a provided data set
- Mid – Extracts the middle part of a data set
- Copy – Copies from a location a set number of characters from a provided data set

- CopyFromChar – Finds a defined character and copies from there up to the end of a provided data set
- CopyUntilChar – Copies until a defined character is found inside a provided data set
- LongToString and UnsignedLongToString – Converts up to 4 bytes of data into their string form. These are ltoa() and ultoa() respectively equivalents
- NetBIOSEncode and NetBIOSDecode – either encodes or decodes respectively the data provided to it into NetBIOS equivalent strings
- CopyFromCharReverse and CopyFromCharReverseInclusive – Finds in reverse (goes backwards) a defined character and copies from there up to the end of a provided data set, while the inclusive version copies also the character we looked for
- Find – returns the position of a provided string
- Patch – modifies an existing data set and replaces a given data in it
- Compare – compares to given data sets and returns either 0 for no match or 1 for match
- ProcessLV – processes a provided data set and returns its data in accordance to its Length Value structure

Left – Extracts up to the rightmost location of a provided data set.

Parameter	Description
Data [required]	<p>The data that is used as the source from which beSTORM copies the data from.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="Copy from here [] to the end"/> </S></pre>
To [required]	<p>The rightmost location that beSTORM will copy the data to.</p> <pre><S Name="To" ParamName="To"> <C Name="binary value" Value="0x10" /> </S></pre> <p>This will copy from the 16 th position up to the end of the string. An error is returned if we try to copy from outside the data set.</p>
TooShortCopy	<p>If there is insufficient data to copy, the function will copy as much as is available instead of giving out an error.</p> <pre><S Name="TooShortCopy"> <C Name="TooShortCopy value" ASCIIValue="1" /> </S></pre>

Right – Extracts from the leftmost location of a provided data set.

Parameter	Description
Data [required]	<p>The data that is used as the source from which beSTORM copies the data from.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="Copy from the beginning until here" /> </S></pre>
From [required]	<p>The leftmost location that beSTORM will copy the data from.</p> <pre><S Name="From" ParamName="From"> <C Name="binary value" Value="0x10" /> </S></pre> <p>This will copy from beginning up to the 16 th position up. An error is returned if we try to copy from outside the data set.</p>

Mid – Extracts the middle part of a data set.

Parameter	Description
Data [required]	<p>The data that is used as the source from which beSTORM copies the data from.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="Copy from a certain location up to another location" /> </S></pre>
From [required]	<p>The leftmost location that beSTORM will copy the data from.</p> <pre><S Name="From" ParamName="From"> <C Name="binary value" Value="0x0B" /> </S></pre> <p>This will copy from the 12th position. An error is returned if we try to copy from outside the data set.</p>

Parameter	Description
To [required]	<p>The rightmost location that beSTORM will copy the data from.</p> <pre><S Name="To" ParamName="To"> <C Name="binary value" Value="0x10" /> </S></pre> <p>This will copy up to the 16th position. An error is returned if we try to copy from outside the data set.</p>

Copy – Copies from a location a set number of characters from a provided data set.

Parameter	Description
Data [required]	<p>The data that is used as the source from which beSTORM copies the data from.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="Copy from a certain location up to a given length" /> </S></pre>
From [required]	<p>The leftmost location that beSTORM will copy the data from.</p> <pre><S Name="From" ParamName="From"> <C Name="binary value" Value="0x0B" /> </S></pre> <p>This will copy from the 12th position. An error is returned if we try to copy from outside the data set.</p>
Length [required]	<p>The number of characters that beSTORM will copy the data from.</p> <pre><S Name="Length" ParamName="Length"> <C Name="binary value" Value="0x10" /> </S></pre> <p>This will copy 10 bytes. An error is returned if we try to copy from outside the data set.</p>

CopyFromChar – Finds a defined character and copies from there up to the end of a provided data set.

Parameter	Description
Data [required]	<p>The data that is used as the source from which beSTORM copies the data from.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="Copy from a certain location found by looking for a certain character" /> </S></pre>
From [required]	<p>The leftmost location that beSTORM will start looking for the character.</p> <pre><S Name="From" ParamName="From"> <C Name="binary value" Value="0x0B" /> </S></pre>
Char [required]	<p>The character you seek to copy from.</p> <pre><S Name="Char" ParamName="Char"> <C Name="binary value" Value="0x0D" /> </S></pre> <p>This will look for the 0x0D character in a stream and return any content found after it, or nothing if it is not found.</p>

Find – Returns the position of a sub-string inside a provided string.

Parameter	Description
Data [required]	<p>The data that is used as the source from which beSTORM tries to locate the data inside.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="Copy from a certain location found by looking for a string" /> </S></pre>
From [required]	<p>The leftmost location that beSTORM will start looking for the sub string.</p> <pre><S Name="From" ParamName="From"> <C Name="binary value" Value="0x0B" /> </S></pre>

Parameter	Description
Value [required]	<p>The value string you want to seek inside the Data set.</p> <pre><S Name="Value" ParamName="Value"> <C Name="The string weare looking for" ASCIIValue="Cookie: " /> </S></pre> <p>This will look for the "Cookie: " string (without the quotes) returns the position if found, or 0 if not found, if the position of the sub-string is at the beginning (at position 0) of the provided Data there is no way to use this function's return value to determine whether we found or not found the provided string.</p>
PosAfterValue	<p>By setting this value you can tell the function whether to return the position just after the provided Value or at the beginning of it, valid values are 0x00 (for position of value) and 0x01 (for position after value).</p> <pre><S Name="PosAfterValue" ParamName="PosAfterValue"> <C Name="Flag of whether to return the position of Value or just after it" Value="0x01" /> </S></pre>

CopyUntilChar – Copies until a defined character is found inside a provided data set.

Parameter	Description
Data [required]	<p>The data that is used as the source from which beSTORM copies the data from.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" ASCIIValue="Copy until a certain location found by looking for a certain character" /> </S></pre>
From [required]	<p>The leftmost location that beSTORM will start looking for the character.</p> <pre><S Name="From" ParamName="From"> <C Name="binary value" Value="0x0B" /> </S></pre>

Parameter	Description
Char [required]	<p>The character you seek to copy from.</p> <pre><S Name="Char" ParamName="Char"> <C Name="binary value" Value="0x0D" /> </S></pre> <p>This will look for the 0x0D character in a stream and return any content found before it, or nothing if it is not found.</p>

LongToString and **UnsignedLongToString** – Converts a provide long value (up to 4 bytes) into its string form by using the ltoa and ultoa function respectively. For the LongToString function values larger than 2^{31} will be regarded as negative values and the string returned by the function will include a minus sign, while for the UnsignedLongToString function values larger than 2^{31} will not be regarded as negative.

Parameter	Description
Data [required]	<p>The data that is used to provide the long value or unsigned long value respectively.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" Value="0x01, 0x02, 0x03, 0x04" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="LongToString convert" Library="Stream Utils.dll" Procedure="LongToString"> <S Name="Data" ParamName="Data"> <C Name="Value" Value="0x01, 0x02, 0x03, 0xF4" /> </S> </SP></pre> <p>Returns the value of -2011292471 (0x2D, 0x32, 0x30, 0x31, 0x31, 0x32, 0x39, 0x32, 0x34, 0x37, 0x31), as the DWORD value provided is 0xF4030201.</p> <pre><SP Name="UnsignedLongToString convert" Library="Stream Utils.dll" Procedure="UnsignedLongToString"> <S Name="Data" ParamName="Data"> <C Name="Value" Value="0x01, 0x02, 0x03, 0xF4" /> </S> </SP></pre> <p>Returns the value of 4093837825 (0x34, 0x30, 0x39, 0x33, 0x38, 0x33, 0x37, 0x38, 0x32, 0x35), as the DWORD value provided is 0xF4030201.</p>

NetBIOSEncode and **NetBIOSDecode** – NetBIOS has its own encoding mechanism, this is supported by way of these two functions, either to encode or decode respectively.

Parameter	Description
Data [required]	<p>The data that is used to provide the data to either encode or decode.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" Value="0x01, 0x02, 0x03, 0x04" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="NetBIOSDecode data" Library="Stream Utils.dll" Procedure="NetBIOSDecode"> <SP Name="Data" Comment="NetBIOSEncode data" Library="Stream Utils.dll" Procedure="NetBIOSEncode"> <S Name="Data" ParamName="Data"> <C Name="Value" ASCIIValue="Encode this" /> </S> </SP> </SP></pre>

Patch – Modifies an existing data set and places another data set inside it. Patch cannot insert new bytes into the original data set, rather it can only modify it.

Parameter	Description
Data [required]	<p>The data that is used to provide the data to patch.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" Value="0x01, 0x02, 0x03, 0x04" /> </S></pre>
From [required]	<p>The location where to start the patch work.</p> <pre><S Name="From" ParamName="From"> <C Name="From location" Value="0x01" /> </S></pre>
Patch [required]	<p>The data to patch with.</p> <pre><S Name="Patch" ParamName="Patch"> <C Name="Patch value" Value="0x03, 0x02" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="Patch call" Library="Stream Utils.dll" Procedure="Patch"> <S Name="Data" ParamName="Data"> <C Name="binary value" Value="0x01, 0x02, 0x03, 0x04" /> </S> <S Name="From" ParamName="From"> <C Name="From location" Value="0x01 /> </S> <S Name="Patch" ParamName="Patch"> <C Name="Patch value" Value="0x03, 0x02" /> </S> </SP></pre> <p>Would result in the following output: 0x01, 0x03, 0x02 , 0x04.</p>

Compare – Compares two given sets of data and returns whether they match (0x01) or not (0x00).

Parameter	Description
A [required]	<p>The data that is used to provide the data to compare.</p> <pre><S Name="Data A" ParamName="A"> <C Name="binary value" Value="0x01, 0x02, 0x03, 0x04" /> </S></pre>
B [required]	<p>The data that is used to provide the data to compare.</p> <pre><S Name="Data A" ParamName="A"> <C Name="binary value" Value="0x01,0x02, 0x03, 0x04" /> </S></pre>

Parameter	Description
[Return value]	<p>The 'Output' value contains the content returned by the function.</p> <pre><SP Name="Compare call" Library="Stream Utils.dll" Procedure="Compare"> <S Name="Data" ParamName="A"> <C Name="binary value" Value="0x01, 0x02, 0x03, 0x04" /> </S> <S Name="Data 2" ParamName="B"> <C Name="binary value" Value="0x01, 0x02, 0x03, 0x04" /> </S> </SP></pre> <p>Would result in the following output: 0x01.</p>

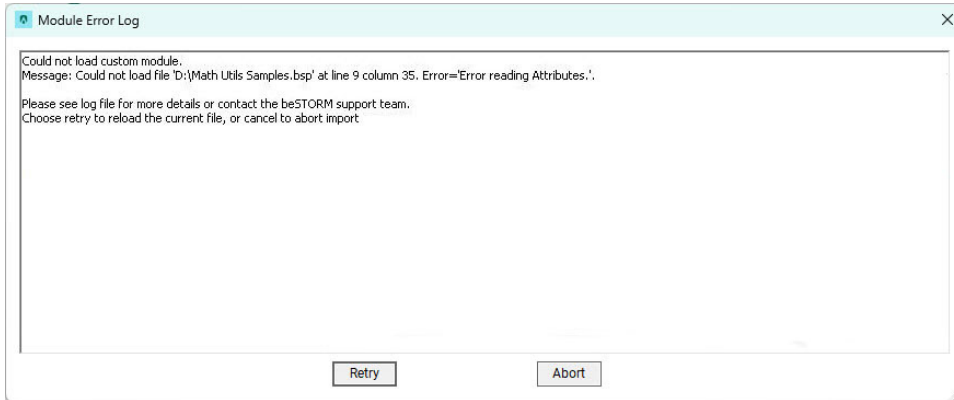
ProcessLV – Process a given data set for a structure of Length Value and returns the LV structure's components.

Parameter	Description
Data [required]	<p>The data that is used to provide the data to compare.</p> <pre><S Name="Data" ParamName="Data"> <C Name="binary value" Value="0x03, 0x02, 0x03, 0x04" /> </S></pre>
Size [required]	<p>The size of the length value (between 0x01 and 0x04).</p> <pre><S Name="Size" ParamName="Size"> <C Name="binary value" Value="0x01" /> </S></pre>
From [required]	<p>The starting location from which the function will start processing.</p> <pre><S Name="From" ParamName="From"> <C Name="binary value" Value="0x00" /> </S></pre>

Parameter	Description
NetworkOrder	<p>Whether to expect the length value to be in big-endian (host) or little-endian (network). Big-endian 0x010203 stored in memory like so 0x01, 0x02, 0x03 while little-endian which is stored in memory like so 0x03, 0x02, 0x01. By default we length is expected to be in big-endian.</p> <pre><S Name="NetworkOrder" ParamName="NetworkOrder"> <C Name="binary value" Value="0x00" /> </S></pre>
[Return value]	<p>Several output values return 'Data' – containing the data found inside the structure – 'Length' – the length found inside the structure and 'To' – the position just after the 'Data', which can then be used for any follow-up functions that need to know where the LV structure ended.</p> <pre><SP Name="SP Process LV" Library="Stream Utils.dll" Procedure="ProcessLV"> <S Name="NetworkOrder"> <C Name="Network Order" ASCIIValue="0" /> </S> <S Name="From"> <C Name="From value" Value="0x00"/> </S> <S Name="Size"> <C Name="Size value" Value="0x02"/> </S> <S Name="Data"> <C Name="Length" Value="0x00, 0x04"/> <C Name="Value" ASCIIValue="Noam"/> </S> </SP> <S Name="Results"> <C Name="1" ASCIIValue="Length: " /> <PC Name="V1" ConditionedName="SP Process LV" Parameter="Length" /> <C Name="2" ASCIIValue="Data: " /> <PC Name="V2" ConditionedName="SP Process LV" Parameter="Data" /> <C Name="3" ASCIIValue="To: " /> <PC Name="V3" ConditionedName="SP Process LV" Parameter="To" /> </S></pre>

Load a custom module

You can load a Custom Module by selecting **New Project** from the Welcome page, and then selecting **Import a Custom Module from a BSM File**. In some occasions the module you are trying to write is malformed which in turn will trigger a Module Error Log dialog.



As shown in the above image, we tried to load the 'Math Utils Samples.bsp' but the XML parser has detected an error at line 9, column 35. Proceed by fixing the issue reported, and then selecting **Retry**, or select **Abort** and select another file.

beSTORM Monitoring

Overview

beSTORM comes bundled with several samples of monitors:

- The `SNMPMonitor.java` monitor which utilizes the SNMP protocol to determine whether the remote host's SNMP agent is alive.
- The `gdb_monitor.pl` utilizes the GNU Debugger to attach to the process and sends a notification whenever the debugger detects that an error has occurred.
- The `bestorm_tail.pl` and `bash_monitor.sh` which tail a log file and based on detected strings inform beSTORM whenever something out of the ordinary has occurred.

Monitoring consists of sending traffic to beSTORM traffic on two ports by way of UDP packets, 6969 and 6970:

- 6969 allows you to notify beSTORM that an exception/vulnerability/a problem with the product has been detected - whatever you send to this port will get documented, up to 64K.
- Via communication on port 6970 (UDP) you can control beSTORM, all commands return either the command issued followed by a FAILURE or SUCCESSFUL string, except NOOP which returns a timestamp.

In most cases it is sufficient for the monitor to just send the NOOP command (which means sending N O O P without spaces) to the remote beSTORM installation and letting beSTORM know you are alive, and stop sending them when it wants to inform that something needs to be reported to the user.

Available commands

Command	Description	Response
NOOP	Allows you to tell beSTORM that the monitor is still alive.	Returns NOOP followed by a timestamp.
EXCEPTION	Any text that follows this string will be handled as an exception data that needs to be reported to the user as a possible vulnerability.	Returns either EXCEPTION FAILURE if it failed to log the exception or EXCEPTION SUCCESS if it was successful.

Command	Description	Response
ERROR	Any text that follows this string will be handled as an error data that needs to be reported to the user as a possible problem with the testing process.	Returns either ERROR FAILURE if it failed to log the exception or ERROR SUCCESS if it was successful.
UP	Tell beSTORM to speed up.	Returns either UP FAILURE if it failed to log the exception or UP SUCCESS if it was successful.
DOWN	Tell beSTORM to speed down.	Returns either DOWN FAILURE if it failed to log the exception or DOWN SUCCESS if it was successful.
FINISH	If beSTORM is currently in PAUSED mode, beSTORM will move to FINISH mode which means that it will allow you load another project if required.	Returns either FINISH STOP FIRST if beSTORM is still running and cannot be put into FINISH mode or FINISH SUCCESS if it was successful. <ul style="list-style-type: none"> • Code value of 0 means internal error while trying to process the FINISH command. • Code value of 1 means successfully processed the FINISH command. • Code value of 2 means that we were already in FINISH state.
STOP	Request that beSTORM to move to PAUSED mode.	Returns either STOP FAILURE if it failed to log the exception or STOP SUCCESS if it was successful.

Command	Description	Response
START	Request that beSTORM to move to RUNNING mode.	Returns either START FAILURE if it failed to log the exception or START SUCCESS if it was successful.
STATUS	Ask beSTORM what is its running status, possible status values are PRE_RUN, RUNNING, CONCLUSION, FAILURE, PAUSED, UNKNOWN and UNSET, the last two status values should be regarded as a failure of the beSTORM program and it is recommended that beSTORM is restarted.	---
EXIT	Ask beSTORM to shutdown/exit.	Returns either EXIT STOP FIRST if beSTORM is still running and cannot be put into FINISH mode or EXIT SUCCESS if it was successful.

Command	Description	Response
LOAD	Ask beSTORM to load another beSTORM project (settings.bsp file), any text the follows this string will be used as the file to load (for example, LOADc:\project\settings.bsp).	<p>Returns either one of these:</p> <ul style="list-style-type: none"> • LOAD FAILURE STOP FIRST if beSTORM is still running and cannot load a new project. • LOAD FAILURE FINISH FIRST if beSTORM is in PAUSED mode and cannot load a new project. • LOAD FAILURE NO FILENAME if no filename has been provided with the command. • LOAD FAILURE CANNOT OPEN (Code: ...) if opening a settings.bsp file that was provided failed due to some error (open error codes). • LOAD FAILURE CHANGES DISCARDED if loading another project caused the settings to be lost. • LOAD SUCCESS if loading of another project was successful.
SAVE	Ask beSTORM to save the current beSTORM project.	Returns either SAVE STOP FIRST if beSTORM is still running and cannot save, or SAVE SUCCESS if it was successful.

Command	Description	Response
VECTOR	Ask beSTORM to load a specific Attack Vector (for example, VECTORM0:P0:B0.BT0:B0.BT0:B0.BT0:SE0.CC0:SR3 (L0:B0.BT0.L5:B0.BT0.L10:B0.BT0).E2.E3).	Returns either one of these: <ul style="list-style-type: none"> • VECTOR STOP FIRST if beSTORM is still running and cannot load a new attack vector. • VECTOR CHANGES DISCARDED if loading another attack vector caused the settings to be lost. • VECTOR FAILURE if the provided attack vector was invalid to the current beSTORM configuration. • VECTOR SUCCESS if loading of a new attack vector was successful.
COUNTEXCEPTION	Returns the number of exceptions found so far.	Returns either COUNTEXCEPTION SUCCESS (Count: ...) with the number of exceptions recorded, or COUNTEXCEPTION FAILURE if it was not possible to return the number of exceptions.

Command	Description	Response
RETURNEXCEPTION	Returns the data captured for a certain exception.	<p>Returns either one of these:</p> <ul style="list-style-type: none"> • Returns RETURNEXCEPTION FAILURE (Code: 1), if no exceptions have been recorded – usually means that beSTORM is currently in an error state. • Returns RETURNEXCEPTION FAILURE (Code: 2), if the provided exception number is invalid (cannot be parsed). • Returns RETURNEXCEPTION FAILURE (Code: 3), if the provided exception number is larger than the number of exceptions present (the first exception is found at position number zero). • Returns the exception information in the following structure: RETURNEXCEPTION SUCCESS New: %d, CommentSize: %d, Comment: %s, ExceptionInformationSize: %d, ExceptionInformation: %s, ExceptionTypeSize: %d,

Command	Description	Response
		<pre>ExceptionType: %s, Time: %I64d, AttackVectorCount: %d, AttackVector [%d]: %s</pre>
SETTINGSCHANGED	Returns whether the current project's settings have changed (due to beSTORM running or user intervention).	Returns SETTINGSCHANGED SUCCESS (Code: %d) either the value of 0 or 1 – either false or true respectively.
STATS	Returns the total number of sessions generated up to this point as well as the current SPS (momentary value, not average).	---
CURRENTVECTOR	Returns the current attack vector beSTORM is at.	---
INCREMENTVECTOR	The command is followed by a positive number from 1 and up. The command will move the module's position by the provided number. Every 5000 increments a status response is sent to provide feedback to the progress (as increments might take a few seconds to complete). For example, INCREMENTVECTOR1000, This will increment the module by 1000 attacks.	---

Command	Description	Response
RETURNVECTORS	<p>The command is followed by a full path and filename, the tab character (0x09) and the number of vectors to dump. The full path and filename will be used as the storing place for dumping the vectors.</p> <p>For example: RETURNVECTORSC:\TEMP\DUMP.TXT\t1000.</p> <p>(In the above sample the \t was used to distinguish that we are supposed to put a TAB character there)</p> <p>After the command finishes, the file, C:\TEMP\DUMP.TXT will contain 1000 attack vectors from the current position of the module up to the 1000th position</p>	---

Command	Description	Response
RUNFOR	<p>The command is followed by a number. The number tells beSTORM how many combinations (attack vectors) to go through. If used in combination with VECTOR and RETURNVECTORS you can beSTORM run in a distributed manner by combining a VECTOR to load a pre-fetched vector returned by the RETURNVECTORS command, RUNFOR 1000 combinations to prevent it from testing more the pre-fetched interval.</p> <ul style="list-style-type: none"> • For example: RUNFOR1000 will make beSTORM run for 1000 combinations. • To run beSTORM in parallel the following would be required: <ul style="list-style-type: none"> • Launch one instance of beSTORM, generate the module you desire to utilize for testing, configure it and save the settings file • Distribute this settings file between the beSTORM clients • Relaunch the beSTORM and issue the following command by way of the command interface: RETURNVECTORSc:\temp\list.txt\t1000 (NOTE the \t character which needs to replace with a real tab character, 0x09) • Take the list of attack vectors found inside c:\temp\list.txt and for each of your beSTORM instances open them with the same settings file and issue these commands VECTOR... 	---

Command	Description	Response
	(Take from the list.txt file) RUNFOR1000 (So it matches the RETURNVECTORS command)	

Microsoft Windows monitoring

One of the complementary tools beSTORM provides is a monitoring tool that attaches to the tested application or service and reports back to beSTORM all relevant information concerning exceptions, crashes, and various errors. This allows both the user to have more relevant information for reproducing the problem and detecting its cause, and beSTORM to control the test progress.

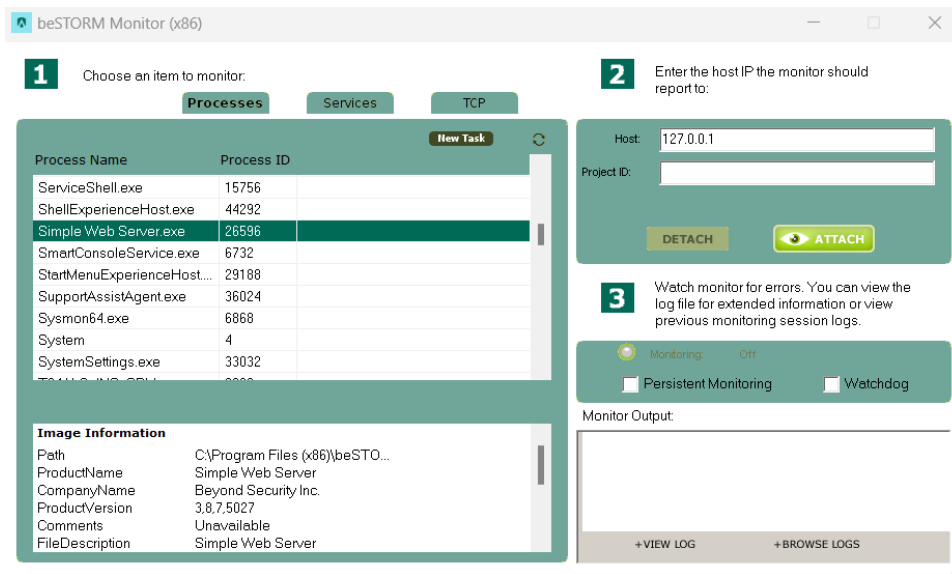
The beSTORM monitor communication allows beSTORM to log vulnerability information, stop test if necessary, wait while tested server is on high load, increase test speed if possible and auto-start tested server in case of crash to automatically resume the beSTORM test.

beSTORM monitor interfaces for Windows

beSTORM monitor has two interfaces for monitoring Windows:

To monitor Windows using the GUI interface

1. Open **beSTORM Monitor**.
2. Select the desired process/service from the **Process Name** list.
3. In **Host**, enter the desired host IP address.
4. Select **Attach**.



To monitor Windows using the Console interface

- To monitor a process, run `Monitor.exe --attach_pid <Process ID of the process we wish to monitor>--host <Host to report to>`.
- To monitor a service, run `Monitor.exe --host 127.0.0.1 --register_debugger <Service executable name here, with extension>`, and then restart the service.
- To stop monitoring a service, run `Monitor.exe --unregister_debugger <Service executable name here, with extension>`, and then restart the service.

To view the full usage, simply run the application with no parameters and it would be displayed on screen.

In addition, beSTORM provides use of the same functionality with different monitors/debuggers such as gdb, OllyDbg or even your own proprietary monitoring tool. Since the communication is through an easy to use UDP protocol, this allows you to best suit the monitoring process to your test scenario or setup lab.

For Windows two additional components have been provided, one that allows utilizing of OllyDbg as the debugger and the other to allow usage of WinDBG as the debugger.

For OllyDbg support, copy the enclosed `OllyDbgbeSTORMPlugin.dll` file to your OllyDbg installation folder and start OllyDbg as usual. beSTORM is notified for every pause event triggered in OllyDbg. See the `OllyDbg.ini` file for extra configurations or use the added beSTORM configuration GUI.

For WinDBG support, copy the enclosed MSEC.dll file to your WinDBG winext directory under the WinDBG installation directory, modify the beSTORM supplied windbg.txt script to correctly point to your beSTORM's client IP address and call the script from within windbg.txt

Linux monitoring

For Linux monitoring you have several options to monitor your process, the first one is to use GnuDB (debugger). This open source and widely available debugger can be instructed to inform beSTORM whenever an exception occurs. A sample script named gdb_monitor.pl, that performs this for you can be found under the beSTORM installation folder.

In some cases debugging is not possible as the process spawns new child processes, is a kernel module, or any other reason, in those cases you can use a non-debugger beSTORM monitor agent.

For example, one of the monitors that comes with beSTORM is the bestorm_tail.pl, this script tails a provided file and looks for specific strings that it will then pass them to beSTORM to inform him that some failure/exception has occurred. One of the most common strings you can look for is SEGFALT, this string if found inside a log file generated by a program indicates that a segmentation fault (a crash, usually unhandled) has occurred.