FORTRA

beSTORM
13.1.0

API Command List Guide (Network/UDP)

202307130926

Table of Contents

Overview

The beSTORM API allows you to manage a beSTORM project remotely on the beSTORM computer using port 6970 (you can change this port in the project setup phase).

The API works by sending a command as a string to the given port. The string is then received by the beSTORM computer and will trigger the adequate action. The API will also respond with a with either a success or error message based on the state of the request. This guide lists all the possible beSTORM API commands along with their associated behaviors and possible responses.

Exception Counter

Description

Returns the number of exceptions caught so far.

Send

```
COUNTEXCEPTION
```

Receive

```
COUNTEXCEPTION SUCCESS (Count: %d)
```
Returns the number of exceptions caught so far.

```
COUNTEXCEPTION FAILURE
```
The beSTORM module has not been loaded. Cannot return the number of exceptions.

Exception Content

Description

Returns the number of exceptions found.

Send

```
RETURNEXCEPTION%d
```

Receive

```
RETURNEXCEPTION SUCCESS
New: %d, CommentSize: %d, Comment: %s, ExceptionInformationSize: %d,
ExceptionInformation: %s, ExceptionTypeSize: %d, ExceptionType: %s,
Time: %I64d, AttackVectorCount: %d, AttackVector[%d]: %s

RETURNEXCEPTION FAILURE (Code: %d)
```
beSTORM failed to return the number of exceptions and returned one of the following codes:

- Code 3 = There is no such exception number.
- Code 2 = The provided exception number is missing or invalid and cannot be loaded.
- Code 1 = There are no recorded exceptions or beSTORM modules loaded.

Example

This example demonstrates returning exception number 2.

Send

```
RETURNEXCEPTION2
```

Report Exception

Description

Reports the specified exception message.

Send

```
EXCEPTION%s
```

Receive

```
EXCEPTION FAILURE
```
No beSTORM module has been loaded.

```
EXCEPTION FAILURE NOT RUNNING
```
beSTORM is not currently running/testing.

```
EXCEPTION SUCCESS
```
beSTORM successfully recorded the exception sent to it.

Exit

Description

Causes beSTORM to exit.

Send

```
EXIT
```

Receive

```
EXIT FAILURE STOP FIRST
```
beSTORM is currently running. Stop beSTORM to run the command.

```
EXIT SUCCESS
```
beSTORM was able to exit without errors.

Return Status

Description

Returns the current state/status of a beSTORM module.

Send

```
STATUS
```

Receive

`STATUS SUCCESS PRE_RUN`
The module is loaded and ready to run.

`STATUS SUCCESS RUNNING`
The module is under test.

`STATUS SUCCESS CONCLUSION`
The module has finished testing and is currently waiting.

`STATUS SUCCESS FAILURE`
An error has occurred, causing the module to stop running.

`STATUS SUCCESS PAUSED`
The module has been paused and is currently not running.

`STATUS SUCCESS UNKNOWN`
An error has occurred while the module was loading or no module is loaded.

`STATUS SUCCESS UNSET`
The beSTORM engine has not yet started.

Current Running Vector

Description

Retrieves the attack vector currently running in beSTORM.

Send

```
CURRENTVECTOR
```

Receive

```
CURRENTVECTOR FAILURE STOP FIRST
```
beSTORM is currently running. Stop beSTORM to run the command.

```
CURRENTVECTOR SUCCESS %s
```
The attack vector is returned.

Statistics and Performance

Description

Returns the total number of iterations the module has performed and sessions per second (SPS).

Send

```
STATS
```

Receive

```
STATS SUCCESS TotalCount: %u, SPS: %u
```

Save Perl Exploitation Script

Description

Saves the Perl script for the current position, based on the provided full path and filename.

Send

```
RETURNPERL%s
```

Receive

```
RETURNPERL FAILURE STOP FIRST
```
beSTORM is currently running. Stop beSTORM to run the command.

```
RETURNPERL MISSING OR INVALID VALUE
RETURNPERL FAILURE
```
The provided path is either invalid or not provided.

```
RETURNPERL INVALID PATH PROVIDED
RETURNPERL FAILURE
```
The provided path is inaccessible.

```
RETURNPERL FILE EXISTS WILL NOT OVERWRITE
RETURNPERL FAILURE
```
The provided filename already exists. beSTORM will not overwrite the file.

```
RETURNPERL FAILED TO OPEN FILE FOR WRITING
RETURNPERL FAILURE
```
The provided file could not be opened. This usually indicates a permission problem.

```
RETURN SUCCESSFUL
```
The Perl script has been successfully written.

```
RETURN FAILURE
```
beSTORM was unable to retrieve the Perl script from the current module state and configuration.

Example

This example demonstrates how to save a Perl script in c:\temp\script.pl.

Send

```
RETURNPERLc:\temp\script.pl
```

Run for a Certain Number of Combinations

Description

Informs beSTORM to run for a certain amount of combinations.

Send

```
RUNFOR%d
```

Receive

```
RUNFOR MISSING OR INVALID AMOUNT VALUE
RUNFOR FAILURE
```
beSTORM could not parse the provided value.

```
RUNFOR FAILURE STOP FIRST
```
beSTORM has to be in STOP state to set this value.

```
RUNFOR SUCCESS
```
beSTORM was successful setting the amount to run for.

```
RUNFOR FAILURE
```
beSTORM is unable to set the value or process the data provided.

Example

Description

This example demonstrates how to run beSTORM for 60 combinations.

Send

```
RUNFOR60
```

Move a Certain Number of Combinations

Description

beSTORM moves forward by X amount of vectors without running them.

Send

```
INCREMENTVECTOR%d
```

Receive

```
INCREMENTVECTOR FAILURE STOP FIRST
```
beSTORM is currently running a module. Stop beSTORM to move the vector forward.

```
INCREMENTVECTOR MISSING OR INVALID AMOUNT VALUE
INCREMENTVECTOR FAILURE
```
beSTORM could not parse the provided value.

```
INCREMENTVECTOR INVALID AMOUNT VALUE
INCREMENTVECTOR FAILURE
```
The provided value cannot equal zero or a value less than zero.

```
INCREMENTVECTOR CONTINUE (Count: %ld)
```
beSTORM will return progress every 5000 vector increments.

```
INCREMENTVECTOR SUCCESS (Count: %ld)
```
beSTORM will return the number of vectors that have progressed.

```
INCREMENTVECTOR FAILURE (Count: %ld)
```
beSTORM is unable to progress the amount of vectors requested. The actual number of vectors progressed are provided.

Example

This example demonstrates how beSTORM moves forward 150 combinations.

Send

```
INCREMENTVECTOR150
```

Save the Current Configuration and Settings

Description

beSTORM saves the current settings into the settings file of the project.

Send

```
SAVE
```

Recieve

```
SAVE FAILURE STOP FIRST
```
beSTORM is currently running. Stop beSTORM to run the command.

```
SAVE SUCCESS (Code: %d)
```
beSTORM has saved the settings and returned one of the following codes:

- Code 1 = No errors occurred.
- Code 0 = Errors occurred while saving.

Load a Provided Configuration File and Settings

Description

Instructs beSTORM to load a project file from the provided path

Send

```
LOAD%s
```

Receive

`LOAD FAILURE STOP FIRST`
beSTORM is currently running. Stop beSTORM to run the command.

`LOAD FAILURE FINISH FIRST`
beSTORM is in the Conclusion/Finish state. You must put beSTORM in the STOP state to run the command.

`LOAD FAILURE NO FILENAME`
File name was not provided.

`LOAD FAILURE CANNOT OPEN (Code: %d)`
The provided file could not be opened. The code returned is the WinAPI error code for fopen.

`LOAD CHANGES DISCARDED`
beSTORM discarded any unsaved changes prior to loading the project.

`LOAD SUCCESS (Code: %d)`
beSTORM has loaded or attempted to load the project and returned one of the following codes:

- Code 1 = No errors occurred.
- Code 0 = Errors have occurred.

Example

This example demonstrates how to load a project located in c:\temp\project.bsp.

Send

```
LOAD%c:\temp\project.bsp
```

Check Settings Status

Description

Reports if settings were changed after the project was loaded.

Send

```
SETTINGSCHANGED
```

Receive

```
SETTINGSCHANGED SUCCESS (Code: %d)
```
beSTORM has checked the settings' status and returned one of the following codes:

- Code 1 = Settings have changed.
- Code 0 = Settings have not changed.

Change to Finished Running State

Description

Allows you to return beSTORM to its FINISH state or just after the module has been loaded.

Send

```
FINISH
```

Receive

```
FINISH FAILURE STOP FIRST
```
beSTORM is currently running. Stop beSTORM to run the command.

```
FINISH SUCCESS (Code: %d)
```
beSTORM was returned to the FINISH state and returned one of the following codes:

- Code 2 = beSTORM is already in the FINISH state.
- Code 1 = Finished without errors.
- Code 0 = Finished with errors.

Change to Stop Running State

Description

Instructs beSTORM to stop running.

Send

```
STOP
```

Receive

`STOP SUCCESS`
beSTORM has stopped the running module.

`STOP FAILURE`
beSTORM was unable to stop the running module or the module was not running.

Change to Start Running State

Description

Instructs beSTORM to start.

Send

START

Receive

`START FAILURE`
Could not start beSTORM. beSTORM must be in one of the following states:, PRE_RUN (after the module has been loaded), CONCLUSION (when all tests have ran) or PAUSED.

`START SUCCESS`
beSTORM has started testing.

Report and Log Errors

Description

Allows you to inform thebeSTORM client of an error, unlike exceptions, these are not logged as vulnerabilities found.

Send

```
ERROR%s
```

Receive

`ERROR FAILURE`
beSTORM is not currently running, no errors can be logged.

`ERROR SUCCESS`
beSTORM has successfully logged the error.

Example

This example demonstrates informing the client of the following message: "System not ready."

Send

```
ERRORSystem not ready
```

No Operation Heartbeat

Description

Returns the number of seconds since the year 1970 in 64-bit form.

Send

```
NOOP
```

Receive

```
NOOP%I64d
```

Code Sample

This code sample demonstrates how to create a console program to send commands to beSTORM (C++ 14 x86/x64).

```
#include <WinSock2.h>
#include "WS2tcpip.h"
#pragma comment(lib, "ws2_32.lib")

#include <iostream>
#include <thread>

#define BUFFER_SIZE 1024u

void Receive(std::atomic<SOCKET>* const mySocket, std::atomic<bool>* const loop)
{
    while (*loop)
    {
        char response[BUFFER_SIZE];
        memset(response, 0, BUFFER_SIZE);

        sockaddr fromAddress;
        int addrLen = sizeof(sockaddr);
        int error = recvfrom(*mySocket, response, BUFFER_SIZE, 0, &fromAddress, &addrLen);

        if (error != SOCKET_ERROR)
            printf_s("Received: %s", response);
    }
}

int main()
{
    // WinSock set up
    WORD version = MAKEWORD(2, 2);
    WSADATA data;
```

```
if (WSAStartup(version, &data) == SOCKET_ERROR)
    return 0;

if (data.wVersion != version)
{
    if (WSACleanup() == SOCKET_ERROR)
        return 0;
    return 0;
}

// Create and bind socket
std::atomic<SOCKET> mySocket;

mySocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (INVALID_SOCKET == mySocket)
{
    int error = WSAGetLastError();
    return 0;
}

sockaddr_in* myAddress = new sockaddr_in();
myAddress->sin_family = AF_INET;
myAddress->sin_port = htons(1234u);
InetPton(AF_INET, L"127.0.0.1", &myAddress->sin_addr);

unsigned long mode = 1;
ioctlsocket(mySocket, FIONBIO, &mode);

int bindRet = bind(mySocket, (const sockaddr*)myAddress, sizeof(sockaddr));
if (bindRet < 0)
{
    std::cout << "Bind Error: " << WSAGetLastError() << '\n';

    closesocket(mySocket);

    delete myAddress;
```

```
    if (WSACleanup() == SOCKET_ERROR)
        std::cout << "Clean Up Error: " << WSAGetLastError() << '\n';

    std::cout << "Exiting\n";
    return 0;
}

// Create the destination address (IP of the beSTORM compter and port 6970)
sockaddr_in* otherAddress = new sockaddr_in();
otherAddress->sin_family = AF_INET;
otherAddress->sin_port = htons(6970u);
InetPton(AF_INET, L"127.0.0.1", &otherAddress->sin_addr);


// Launch the thread that will receive the beSTORM responses and print them
std::atomic<bool> loop = true;
std::thread receiver(Receive, &mySocket, &loop);

// Send to the beSTORM address anything writen in the console
while (loop)
{
    char command[BUFFER_SIZE];
    scanf_s("%s", command, BUFFER_SIZE);
    if (command[0] == 0)
        continue;

    if (strcmp("q", command) == 0 || strcmp("Q", command) == 0)
        loop = false;

    int size = 0;
    for (int i = 0; i < BUFFER_SIZE; ++i)
        if (command[i] == 0)
        {
            size = i;
            break;
        }
```

```
        sendto(mySocket, command, size, 0, (const sockaddr*)otherAddress, sizeof(sockaddr));
    }

    // Clean up
    receiver.join();

    closesocket(mySocket);

    delete myAddress;
    delete otherAddress;

    std::cout << "Exiting\n";
    if (WSACleanup() == SOCKET_ERROR)
        return 0;

    return 0;
}
```