FORTRA

beSTORM

13.1.0

Fuzzing an Imported File Guide

Table of Contents

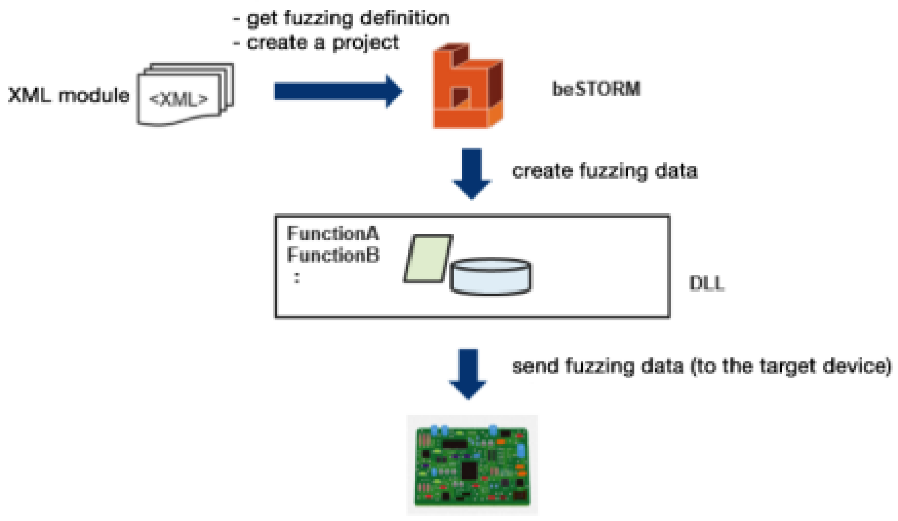
Overview

You can customize beSTORM for functions and communications not provided by the tool. This section describes the basics of customization. For more information, please refer to the beSTORM User Guide or knowledge articles.

This image displays the basic behavior of beSTORM:

The basic behavior of beSTORM is as follows:

1. beSTORM generates a project based on a scenario defined in the XML module.
2. The XML module defines how a module generates fuzzing data, what type of data it sends, and how it looks and behaves.
3. beSTORM generates fuzzing data based on the description of the buffer data defined in XML module.
4. Once fuzzing begins in beSTORM, it calls a DLL (library) function to output fuzzing data to the opposite device.
5. Each function defined in the DLL generates the payload data output (in addition to the data output, it is possible to initialize and receive data).
6. This call is iterated multiple times until all the generated fuzz data has been sent.
7. The name of the DLL file to be called and the contents of the function are described in XML.



As for standard protocols, beSTORM maintains XML data and uses internal libraries to communicate with it.

With beSTORM's customization options, you can prepare functions and define their names in XML for your own control. For example, for a new serial communication standard, you can develop a function that performs the serial communication as a DLL (Win32API based) and write XML to call the function.

Functions Defined by DLL

beSTORM loads the customized DLL with dynamic links and executes each function. Therefore, it is necessary to implement a function according to the beSTORM specification and export the function. The following is a prototype of the function to export:

```
extern "C"   declspec(dllexport) bool FUNCTION
(int inSize, unsigned char *in, int *outSize, unsigned char
*out, Function_Modes eMode)
```

beSTORM calls the function using the function name defined in XML, so the name can be set arbitrarily. Multiple settings can also be made.

The arguments consist of input data and its size (defined by beSTORM), output data and its size (defined by DLL), and the mode of operation.

In beSTORM, input and output data are handled by packing multiple data (consisting of names and values). Therefore, it is composed of in and out as arguments.

Mode_Preview is specified when it is called to perform analysis in order to display on Module Browser, etc., and Mode_Execute is specified when it actually runs the test. Therefore, the function implemented in DLL implements the actual communication when this argument is Mode_Execute.

The return value is bool. Normally, it returns true. It returns false to notify beSTORM of anomalies.

When you build a customized fuzzing system, you need to prepare a DLL program to communicate with the target device, and build a program to send or receive arbitrary data if necessary. You include the fuzzing data in this arbitrary data.

Specification of Sample DLL

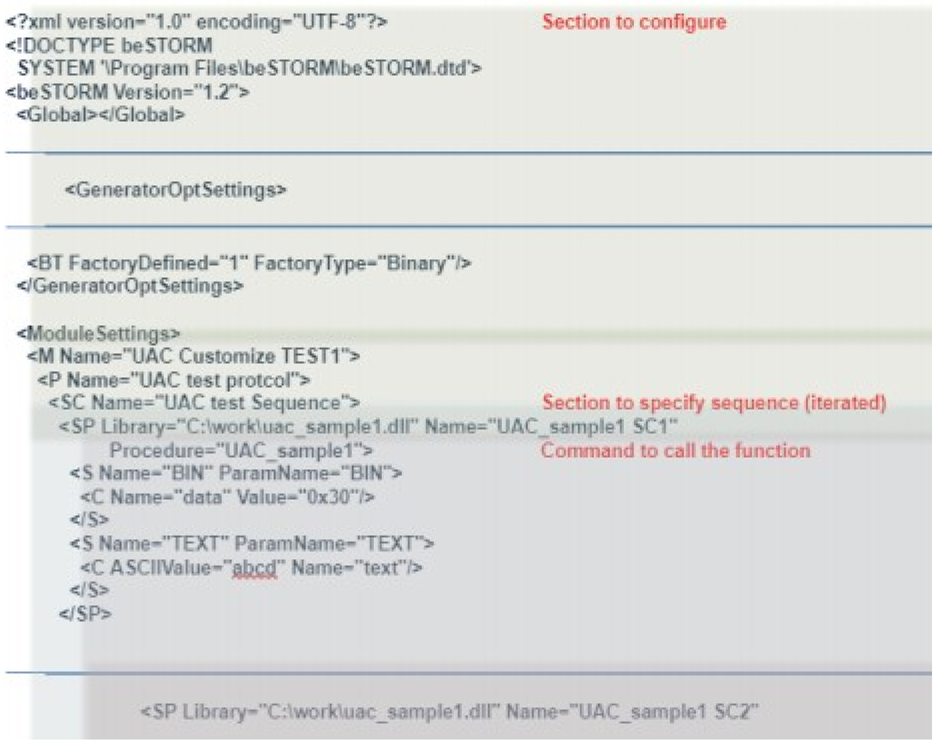
This section explains how to customize the functions in beSTORM with practical examples. Here are sample specifications, assuming that the following two functions exist in the DLL located at c:¥work\uac_sample1.dll for running fuzzing.

Functions Defined in Sample DLL

Function Name	Description
UAC_sample 1	Input Data • BIN=0x30(fixed) • TEXT=”abcd”(fixed) Output Data • RET_1ST(always returns ”1234”)
UAC_sample 2	Input Data • faz1(fuzzing data set by beSTORM) • ret1(set the RET_1ST of uac_sample1 here) Output Data • None

- This example consists of setting in UAC_sample1 and fuzzing in UAC_sample2.
- It is assumed that the DLL performs fuzzing by sending data from faz1 (USB, Ethernet, etc.). • In beSTORM, it is possible to set the return value of the previously called function to the argument of the next function as shown in ret1 above. This is an effective mechanism for controlling devices (such as sockets/device drivers) using handle values.

XML Syntax

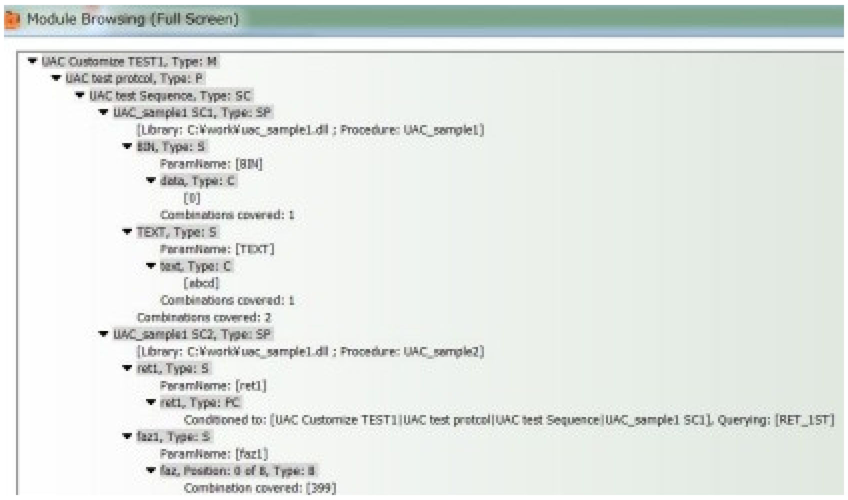


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beSTORM
 SYSTEM '\Program Files\beSTORM\beSTORM.dtd'>
<beSTORM Version="1.2">
 <Global></Global>

    <GeneratorOptSettings>

  <BT FactoryDefined="1" FactoryType="Binary"/>
 </GeneratorOptSettings>

 <ModuleSettings>
  <M Name="UAC Customize TEST1">
   <P Name="UAC test protcol">
    <SC Name="UAC test Sequence">
     <SP Library="C:\work\uac_sample1.dll" Name="UAC_sample1 SC1"
          Procedure="UAC_sample1">
      <S Name="BIN" ParamName="BIN">
       <C Name="data" Value="0x30"/>
      </S>
      <S Name="TEXT" ParamName="TEXT">
       <C ASCIIValue="abcd" Name="text"/>
      </S>
     </SP>

          <SP Library="C:\work\uac_sample1.dll" Name="UAC_sample1 SC2"
```

Section to configure
Section to specify sequence (iterated)
Command to call the function

```
<SP Library="C:\work\uac_sample2.dll" Name="UAC_sample SC1
"Procedure="UAC_sample2"> Command to call the function
<S Name="ret1" ParamName="ret1">
<PC ConditionedName="UAC_sample1 SC1" Name="ret1" Parameter="RET_
1ST"/>
</S>
<S Name="faz1" ParamName="faz1">
<B Name="faz" Value="0x30"/>
</S>
</SP>
</SC>
</P>
</M>
</ModuleSettings>
</beSTORM>
```

This image shows the display of ModuleBrowser when a project is created using XML as described in the previous section.



```
▼ UAC Customize TEST1, Type: M
  ▼ UAC test protcol, Type: P
    ▼ UAC test Sequence, Type: SC
      ▼ UAC_sample1 SC1, Type: SP
          [Library: C:¥work¥uac_sample1.dll ; Procedure: UAC_sample1]
        ▼ BIN, Type: S
            ParamName: [BIN]
          ▼ data, Type: C
              [0]
            Combinations covered: 1
        ▼ TEXT, Type: S
            ParamName: [TEXT]
          ▼ text, Type: C
              [abcd]
            Combinations covered: 1
          Combinations covered: 2
      ▼ UAC_sample1 SC2, Type: SP
          [Library: C:¥work¥uac_sample1.dll ; Procedure: UAC_sample2]
        ▼ ret1, Type: S
            ParamName: [ret1]
          ▼ ret1, Type: PC
              Conditioned to: [UAC Customize TEST1|UAC test protcol|UAC test Sequence|UAC_sample1 SC1], Querying: [RET_1ST]
        ▼ faz1, Type: S
            ParamName: [faz1]
          ▼ faz, Position: 0 of 8, Type: B
              Combination covered: [399]
```

Configurations

- xml tag, !DOCTYPE, and beSTORM tag are specified as the settings above. (This part does not change depending on the specification.)
- GeneratorOptSettings specifies the beSTORM default fuzzing buffer specification. FactoryDefined="1" FactoryType="Binary" applies the beSTORM default fuzzing buffer specification The buffer specifications are described later in "2.4.4 Buffer Type".
- The M tag can be used to specify an arbitrary module name.
- The P tag is an arbitrary protocol name.

Section for Sequence/Function Call

The SC tag is an arbitrary sequence name, and if you put SP tags after the SC tag, beSTORM behaves as if it repeats those processes.

It is possible to describe the function call by using SP tag. I'll elaborate on each one below.

UAC_sample1 Function

```
<SP Library="C:\work\uac_sample1.dll" Name="UAC_sample1 SC1"
Procedure="UAC_sample1"> <S Name="BIN" ParamName="BIN">
<C Name="data" Value="0x30"/>
</S>
<S Name="TEXT" ParamName="TEXT">
<C ASCIIValue="abcd" Name="text"/>
</S>
</SP>
```

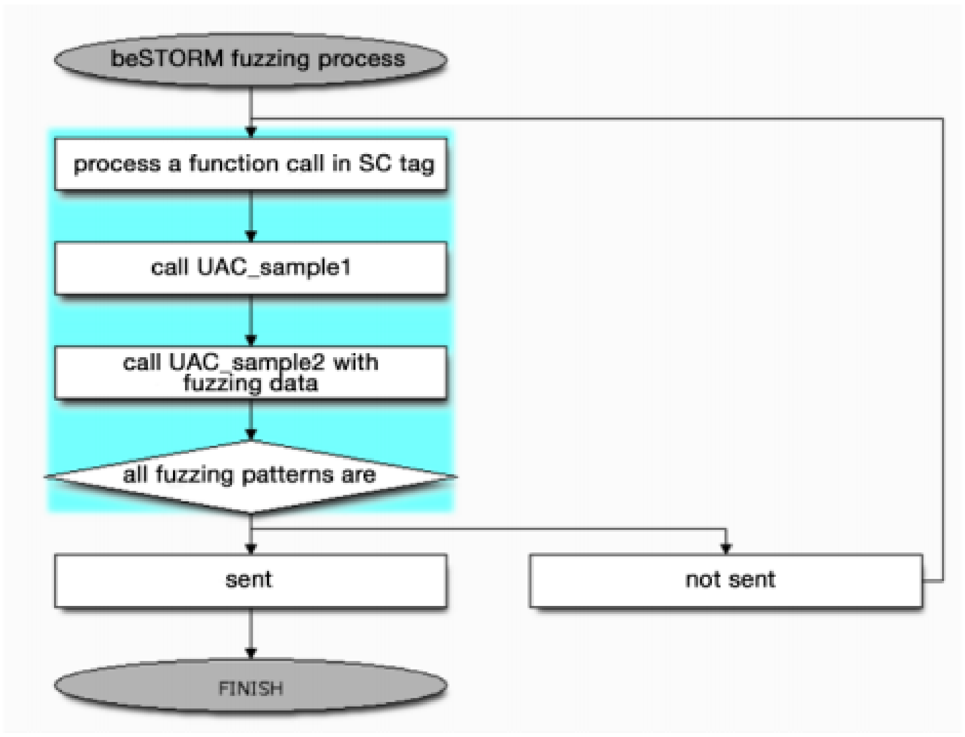
- The Library attribute specifies the path of the DLL to be used.
- The Name attribute will be displayed in the Module Browser, so you can enter any character string.
- The Procedure attribute specifies the name of the function. Therefore, it needs to be consistent with DLL implementation.
- S tag contains settings of the arguments. The Name attribute of S tag specifies the string displayed in Module Browser. The ParamName attribute specifies the name of the variable notified to the DLL, so you need to match it with the DLL implementation.
- The data column to be set is listed under the S tag. As the example above uses a C tag, it means a constant. The constant data is not used as fuzzing data, but passed to the DLL as it is. In the above example, the data "BIN=0x30, TEXT="abcd"" is passed to the DLL.
- Calling UAC_sample1 returns the return value of RET_1ST, which beSTORM keeps. The data is available in XML. We actually use it in the following UAC_sample2. The name of this return value is dependent on the DLL implementation.

UAC_sample2 Function

```
<SP Library="C:\work\uac_sample1.dll" Name="UAC_sample1 SC2"
Procedure="UAC_sample2"> <S Name="ret1" ParamName="ret1">
<PC ConditionedName="UAC_sample1 SC1" Name="ret1" Parameter="RET_
1ST"/>
</S>
<S Name="faz1" ParamName="faz1">
<B Name="faz" Value="0x30"/>
</S>
</SP>
```

- The Library attribute specifies the path of the DLL to be used.
- The Name attribute will be displayed in the Module Browser, so you can enter any character string.
- The Procedure attribute specifies the name of the function. Therefore, it needs to be consistent with DLL implementation.
- S tag contains settings of the arguments.
- By using PC tags, it is possible to use the return value of a previously called function. In this case, ConditionName is the value of Name of SP tag of the corresponding function, Parameter is the name of return value (this is defined in the DLL implementation specification), and Name tag is the string to be displayed by Module Browser.
- In the example above, the return value RET_1 ST of UAC_sample1 is specified as the argument RET1 of the UAC_sample2 function.
- In the example above, we have the data faz1, which is specified by using the B tag under the S tag, where B tag means the data to be fuzzed. The default value is set to 0x30. Because FAZ1 is set to fuzzing data, which is actually set to various contents of various lengths.
- To summarize, this XML passes the two values "ret1=ret1=return value of UAC_ sample1, fuzz=fuzz data" to UAC_sample2.

Sequence



If the SC tag contains the fuzzing data, beSTORM repeats the process for the number of fuzz data. In the case of XML described in this section, it works as shown below.

The UAC_sample1 function does not contain fuzzing data, but it is called repeatedly because it is in the SC tag. Then, various fuzzing data are set to faz1 of the UAC_sample2 function, and it is repeated. What kind of fuzzing data is set up will be explained in the next section.

Buffer Types

If FactoryDefined="1" and FactoryType="Binary", the beSTORM default fuzzing buffer specification will be applied. Concretely, the following data is generated and fuzzed (the function defined in the DLL is called for the number of generated patterns).

Available buffer types

Buffer Type	Data to be Generated
Repeated A	A, AA, AAA, ... and so on, increasing the string A up to 65535 bytes (*1).
Repeated %n	Generate the string %n, %n%n, %n%n%n ... while increasing the string %n to a maximum of 65535 bytes (*1), such as %n, %n%n%n, %n%n%n ...
Repeated NULL	0x00, 0x00 0x00 0x00, 0x00 0x00 0x00 ... and so on, while increasing the binary data of 0x00 up to 65535 bytes (*1).
Number Generating	The expression range of up to 4 bytes is set as unsigned for the specified area (0x00 to 0xffffffffff).
Repeated FF	0xff, 0xff 0xff 0xff, 0xff 0xff 0xff ... and so on, while increasing the binary data of 0xfe up to 65535 bytes (*1).
Repeated FE	0xfe, 0xfe 0xfe 0xfe, 0xfe 0xfe 0xfe ... and so on, while increasing 0xff binary data up to 65535 bytes (*1).
Repeated FFFE	0xff 0xfe, 0xff 0xfe 0xfe 0xfe 0xff 0xfe, 0xff 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe 0xfe ... and so on, while increasing the binary data of 0xff 0xfe to a maximum of 32766 bytes (*1).
Repeated	0xfe 0xff, 0xfe 0xff 0xff 0xfe 0xff 0xff, 0xfe 0xff 0xff 0xfe 0xff 0xfe 0xfe
FEFF	0xff ... and so on, while increasing the binary data of 0xfe 0xff to a maximum of 32766 bytes (*1).

- `*1:` Because of the relationship with other data used at the same time, a value lower than the maximum written above may be required.
- The number of data patterns can be changed by specifying a ScaleType, which can be changed from SETTINGS in beSTORM.

How to specify a custom buffer type

To specify your own buffer type, use the T tag below the BT tag to enumerate the buffer type. Examples are given below (see Beyond Security's Users Guide for more information).

```
<GeneratorOptSettings>
<T Name="Repeated A" Max="65536" ASCIIValue="A" />
<T Name="Repeated %n" Max="65536" ASCIIValue="%n" />
<T Name="Repeated NULL" Max="65536" Value="00" />
<T Name="BiggerSmaller" Max="32768" ASCIIValue="&lt;&gt;" />
<T Name="Repeated Space" Max="65536" ASCIIValue=" " />
</GeneratorOptSettings>
```

In this example, no binary fuzzing data is generated, only string repetition is performed: BiggerSmaller repeats the <> string and Repeated Space repeats the space (ASCII code 0x20). Including them, the XML defines five different repetition patterns.

How to specify a buffer type more precisely

The data passed to the DLL can be specified in detail. The faz1 data shown so far only has a single buffer, but it is also possible to arrange multiple data as shown below.

```
<S Name="faz1" ParamName="faz1">
<B MaxBytes="1" Name="TEST1" Value="0x00"/>
<C Name="TEST2" Value="0x88"/>
<B MaxBytes="1" Name="TEST3" Value="0xff"/>
<C Name="TEST4" Value="0xaa"/>
</S>
```

In this example, the faz1 data contains the following four data.

- Up to 1 byte of fuzzing data (default 0x00)
- Fixed value 0x88
- Up to 1 byte of fuzzing data (default 0xff)
- Fixed value 0xff

Therefore, the function in the DLL is passed as a data sequence of the form "Fuzzing data (TEST1), 0x88, Fuzzing data (TEST2), 0xff".

If multiple fuzzing targets are defined in the same data, only one of them is actually fuzzed. For example, in the above example, any data of "indefinite value, 0x88, 0xff, 0xaa" or "0x00, 0x88, indefinite value, 0xaa" is passed to the DLL.

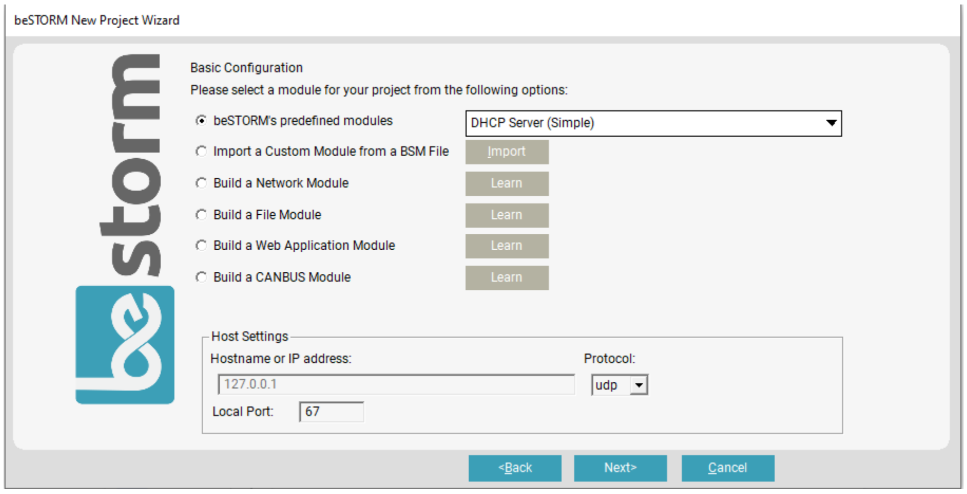
You can write like this when you need to mix the fixed value and fuzzing data according to the packet data specification of the target protocol.

Customized Fuzzing (Summary)

This is a summary of what you need to do for fuzzing customization.

- Prepare the DLL that performs the relevant communication.
- Describes the XML (or modifies the provided XML) according to the DLL specification.
- ML describes the path to the DLL to be used, the definition of fuzzing data, other necessary settings, functions to be called, etc.
    - It is also possible to change the structure of the fuzzing data by changing the definition below the S tag.

Once the XML is created, generate the project with beSTORM, select your XML, and then select **Start** to begin fuzzing.


beSTORM New Project Wizard

Basic Configuration

Please select a module for your project from the following options:

- (•) beSTORM's predefined modules — DHCP Server (Simple)
- ( ) Import a Custom Module from a BSM File
- ( ) Build a Network Module
- ( ) Build a File Module
- ( ) Build a Web Application Module
- ( ) Build a CANBUS Module

Host Settings

Hostname or IP address: 127.0.0.1

Protocol: udp

Local Port: 67