FORTRA

beSTORM

UAC USB Peripheral Module Guide 12.4.19

202303010323

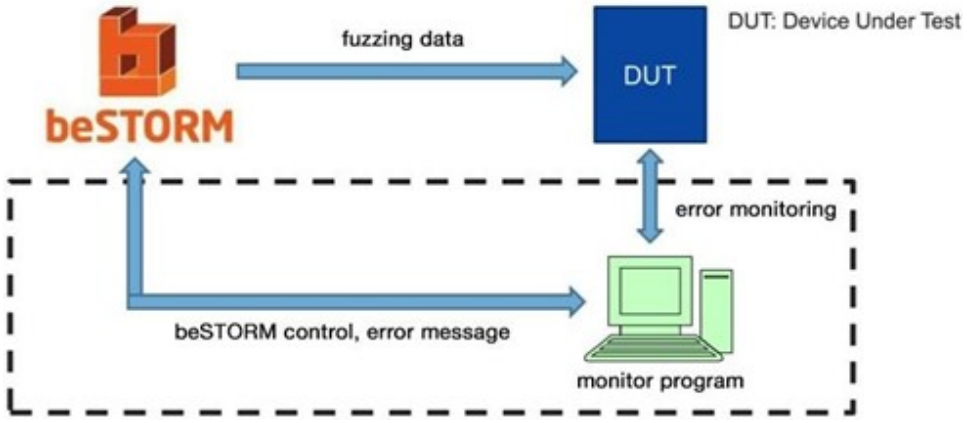
Table of Contents

beSTORM Overview

This section describes an overview of beSTORM. For more information, please refer to Beyond Security's Users Guide or our tutorial materials.

Overview

beSTORM is an intelligent fuzzing tool that performs black box testing, which sends a large number of test cases to various devices to detect vulnerabilities in the devices. Fuzzing can be done easily by creating user-defined files not only for pre-defined protocols, but also for user-specific protocols and arbitrary interfaces such as APIs on platforms and driver layers. The figure below shows a general image of beSTORM usage.



Test case generation

beSTORM generates test cases according to the data structure and procedures defined in advance in XML format, which can be customized by the user. User-defined "data structures" allow for efficient and exhaustive test case generation.

Data origination algorithm

beSTORM can specify a buffer type for each data to be tested, which is defined in the XML module. beSTORM generates the data to be sent according to the buffer type.

The following is an example of a typical case of a beSTORM standard buffer type.

Repeated A

beSTORM inserts the string A into the specified region. The test data is generated while increasing the length of the string A to a predetermined number of bytes. For example, in the case of this buffer type, it attempts to raise an exception related to a buffer overflow.

(Example of data change in the data area where Repeated A is set)

```
A
AA
AAA
AAA...A
```

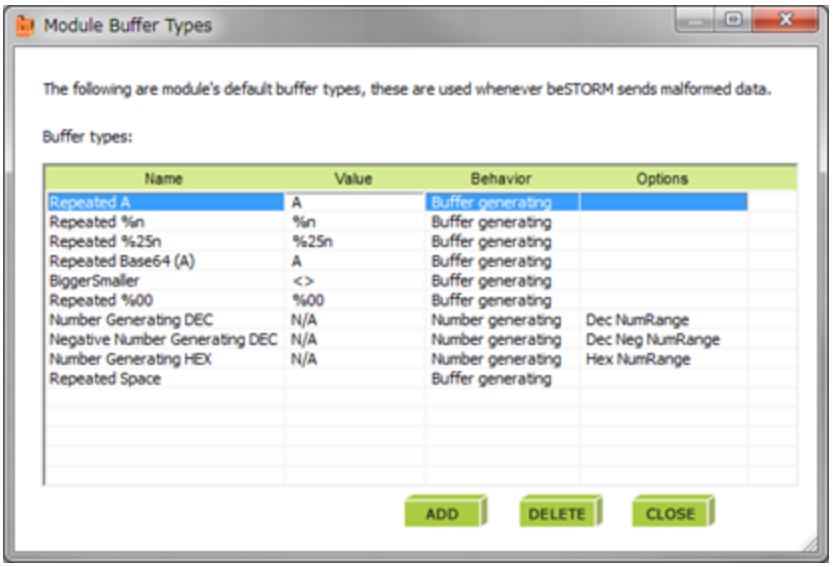
Number generation

beSTORM assigns an integer in the representable range of up to 4 bytes (0 to 4294967295) for the specified region. For example, in the case of this buffer type, it attempts to raise an exception related to an integer overflow.

(Example of data change in the data area where Number Generating is set)

```
0
1
2
...
4294967295
```

For reference, the following screen shows a list of standard buffer types.



Monitoring function

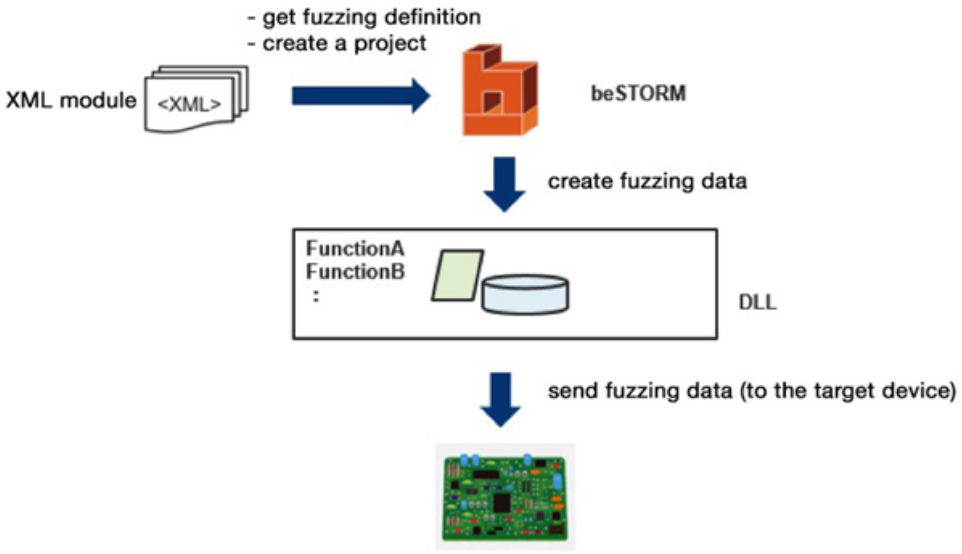
beSTORM has a monitor function to detect test cases in real-time when an exception occurs. When an error occurs, the monitor needs to send the error information to beSTORM's Exception Port (UDP Port) and beSTORM saves the received information. To use the monitoring function, it is necessary to implement a program that detects anomalies in the fuzzing target device and sends UDP to beSTORM. It is also possible to control the pause/resumption of beSTORM usage.

Customizing beSTORM

beSTORM can be customized for functions and communications that are not provided by the tool. This section describes the basics of customization. For more information, please refer to Beyond Security's Users Guide or our tutorial materials.

Overview

The basic behavior of beSTORM is shown below.



- beSTORM generates a project based on a scenario defined in XML.
- The XML defines how a module generates fuzzing data, what type of data it sends, and how it looks and behaves.
- beSTORM generates fuzzing data based on the description of the buffer data defined in XML.
- When select **Start** on the beSTORM screen to start fuzzing, beSTORM calls a DLL (library) function to output fuzzing data to the opposite device.
- 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).
- This call will be iterated multiple times (until all the generated fuzz data has been sent).
- The name of the DLL file to be called and the contents of the function are described in XML.

beSTORM works as described above. As for the standard protocols, beSTORM itself maintains XML data and uses internal libraries to communicate with it.

beSTORM, on the other hand, is characterized by a high degree of customization. You can prepare functions described by the defined interface and define their names in XML for your own control. For example, when a new serial communication standard comes out, you can develop a function that performs the serial communication as a DLL (Win32API based) and write XML to call the function.

Function 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 be exported.

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

The 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_sample1	Input Data • BIN=0x30(fixed) • TEXT=”abcd”(fixed) Output Data • RET_1ST(always returns ”1234”)
UAC_sample2	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.

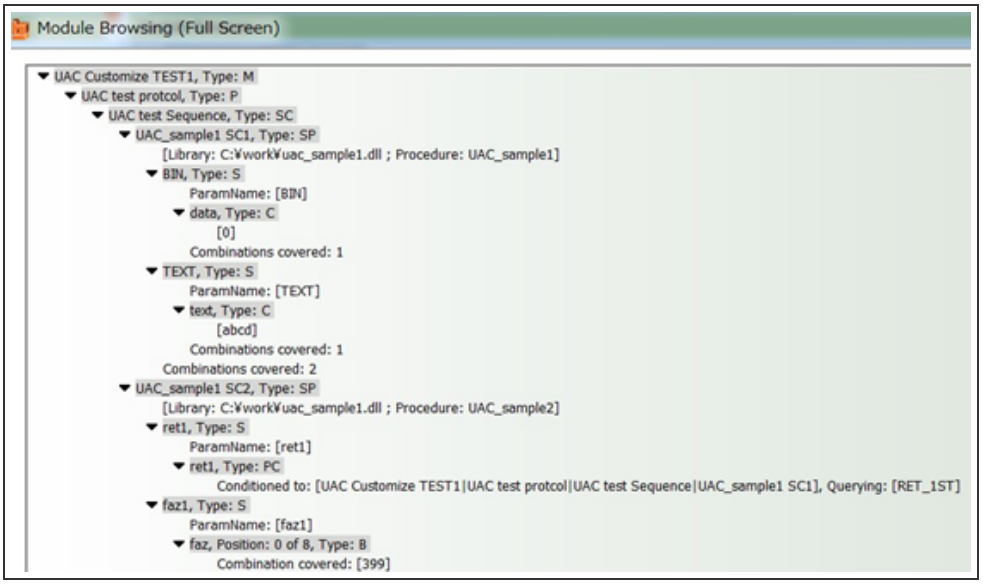
XML syntax

The following is the XML that calls the DLL defined in the previous section and performs fuzzing.

```
<?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">    Command to call the function 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"
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>
</SC>
</P>
</M>
</ModuleSettings>
</beSTORM>
```

The following figure shows the display of ModuleBrowser when a project is created using XML as described in the previous section.



The following is the description of the overview for each tag. For more information, please refer to the beSTORM Users Guide or our tutorial materials.

Configuration section

- 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 ***Buffer types* on page 10**.
- 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.

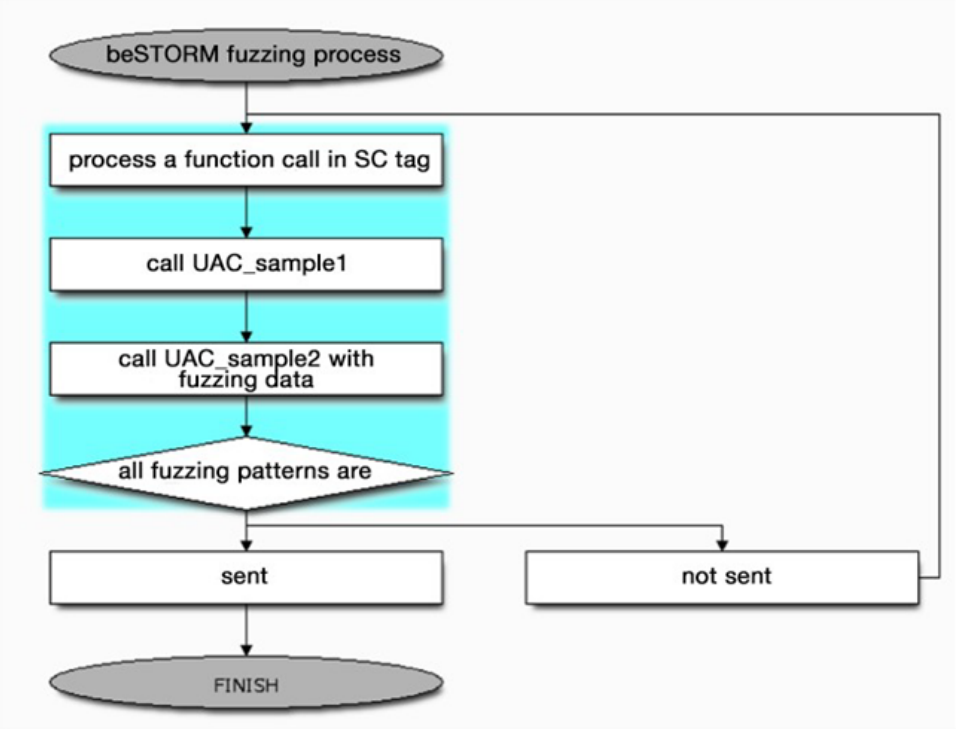
UAC_sample2 function

```
<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).

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 ...`

Buffer type	Data to be generated
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 FEFF	`0xfe 0xff, 0xfe 0xff 0xff 0xfe 0xff 0xff, 0xfe 0xff 0xff 0xfe 0xff 0xfe 0xfe 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 custom buffer type:

To define your own buffer type, use the T tag below the BT tag to enumerate the buffer type. Examples are given below (see the beSTORM User 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 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.

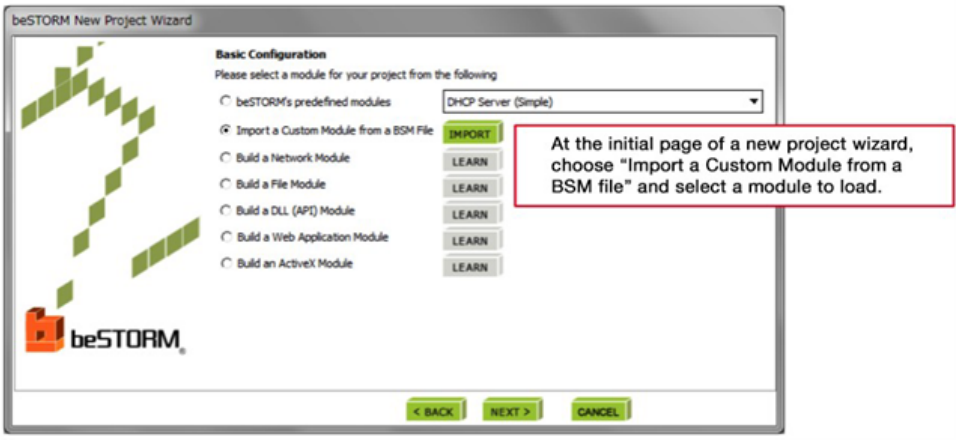
You can write like this when you need to mix the fixed value and fuzzing data according to the packet data specification of the target protocol.

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 the XML you have created as shown below.

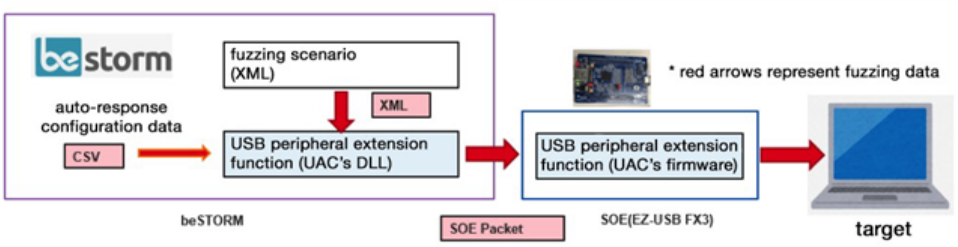


After making a selection, select Start to begin fuzzing.

Overview of the UAC USB Peripheral Module

USB peripheral module architecture

To send fuzzing data to a USB host, the USB peripheral function is required. Therefore, the USB Peripheral module requires a separate device that provides the peripheral function. The figure below shows a configuration diagram including a separate Serial Offload Engine (SOE) device.



- The beSTORM on the left side of the figure above is a Windows PC with beSTORM installed.
- An SOE is a USB peripheral device that can be controlled from Windows by way of USB serial communication. Currently, beSTORM uses the **EZ-USB FX3** board from Cypress (https://www.infineon.com/cms/en/product/evaluation-boards/cyusb3kit-003/).
- Target refers to a device that operates as a USB host, such as a Windows PC.
- The fuzzing data created by beSTORM is sent to the SOE by way of a DLL and then sent to the target as data from the USB peripheral.
- Data is exchanged between DLLs and SOEs (USB serials) via an original communication format called SOE packets. See the Communication Protocol for SOE (Serial Offload Engine) chapter for more information.
- We provide the following resources as a USB Peripheral Module (refer to the Resources chapter for more information):
  - DLL (including source code)
  - SOE firmware (including source code)
  - An XML or CSV file containing sample fuzzing scenarios

USB peripheral module functions

The USB Peripheral Module is provided by `best_uac_usbd.dll`, which provides the following features from XML.

List of USB peripheral module functions

Function	Description
Start USB fuzzing protocol	Start a series of fuzzing processes.
Wait for the SOE to be ready	Waits for the SOE device to become ready to communicate. At this point, there is still no USB connection to the target.
Set Descriptor	Sets a USB descriptor for the SOE. By changing this setting, the target can be recognized as an arbitrary device. The following descriptors can be set from XML: • Device qualifier • Descriptor configuration descriptor (including interfaces, endpoints, etc.) • String descriptor (language ID) • String descriptor (manufacturer) • String descriptor (product)
Start Fuzzing sequence	Indicates the start of the fuzzing data processing sequence. Enumerates a USB connection with the SOE.
Wait for OUT transfer reception	Waits for data to arrive from the specified endpoint from the target.
Process IN transfer transmission	Sends data from the specified endpoint to the target.
End fuzzing sequence	Indicates the end of the fuzzing data processing sequence. Performs the USB disconnect process at the SOE.
Sleep	Sleeps for a specified amount of time.
End USB fuzzing protocol	Ends the fuzzing process.

The basic flow of fuzzing using the USB Peripheral Module is shown below.

*1: In this example, we wait for reception before sending, but the description should be adapted to the actual communication specification. This part of the procedure describes the XML according to the actual target.

In the field labeled by ※1, you should specify an XML tailored for the specific target as shown above. Please code the XML using multiple sequences (SC tags) as it is assumed that the USB Peripheral Module repeats only the blue part in the figure. See *XML* section of the Sample for Mass Storage chapter for a practical example.

By defining arbitrary communication in XML, you can perform fuzzing according to the target protocol.

The USB Peripheral Module supports the auto-responding function. When communication other than fuzzing is performed in parallel, it is possible to set up this automatic response

function in conjunction with the communication. For details of the auto-responder function, please refer to ***Auto-responding function* on page 26**.

How to use functions

Here's an XML example of how to use each feature: in the Library attribute, you need to specify the DLL path, in this case `c:\Work`. This can be rewritten and used according to the environment.

Begin USB fuzzing protocol (UAC_usb_start)

This is to start fuzzing process (`UAC_usb_start` function). Make sure to call it only once for the first time.

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_usb_start SC"
Procedure="UAC_usb_start">
<S Name="COM_PORT" ParamName="COM_PORT">
<EV ASCIIValue="4" Description="COM" Name="COM" Required="1"/>
</S>
<S Name="CSV_FILE" ParamName="CSV_FILE">
<EV ASCIIValue="C:\TEMP\bestorm.csv" Description="CSV_ATH"
Name="CSV_ATH" Required="1"/>
</S>
<S Name="LOG_FILE" ParamName="LOG_FILE">
<EV ASCIIValue="C:\TEMP\bestorm.log" Description="LOG_PATH"
Name="LOG_PATH" Required="1"/>
</S>
<S Name="RECORD_FILE" ParamName="RECORD_FILE">
<EV ASCIIValue="C:\TEMP\best_record.dat" Description="RECORD_PATH"
Name="RECORD_PATH" Required="1"/>
</S>
<S Name="ILLEGAL_TIMER" ParamName="ILLEGAL_TIMER">
<EV ASCIIValue="3000" Description="timer" Name="timer"
Required="1"/>
</S>
<S Name="SER_ERR_RETRY_COUNT" ParamName="SER_ERR_RETRY_COUNT">
<EV ASCIIValue="3" Description="ser_err_retry_count" Name="ser_err_
retry_count" Required="1"/>
</S>
<S Name="USB_DATA_TIMEOUT" ParamName="USB_DATA_TIMEOUT">
<EV ASCIIValue="600" Description="usb_data_timeout" Name="usb_data_
timeout" Required="1"/>
</S>
</SP>
```

The XML requires the following arguments.

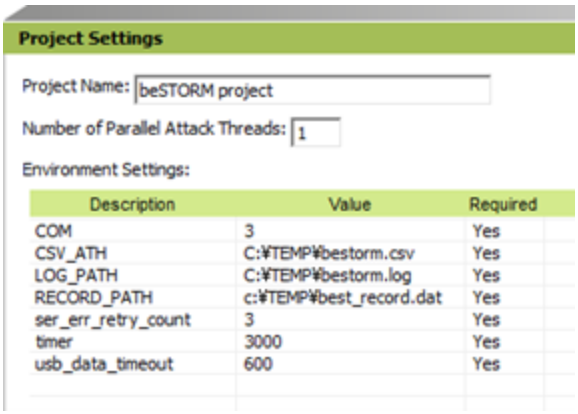
- `COM_PORT` Serial port used to communicate with the SOE.
- `CSV_FILE` Path to the CSV file containing the rules for the automatic response.
- `LOG_FILE` Path to the log file. This DLL outputs its own log separately from the beSTORM.
- `RECORD_FILE` Path to the record data for the recovery tool.
- `ILLEGAL_TIMER` Time to wait in the event of a communication failure with the SOE, usually this value is used.
- `SER_ERR_RETRY_COUNT` Number of retries in case of serial communication error. Normally, this value in the example is used.
- `USB_DATA_TIMEOUT` IN transfer/OUT transfer timeout (interruption. The unit is a second. Normally, the value in the example is used.

`COM_PORT` specifies the serial port to use for communication with the `SOE (EZ-USB3)`. You can also specify the path of CSV file and log file for this fuzzing, which will be explained later. `ILLEGAL_TIMER` is supposed to be used for adjustment in case of abnormalities, but usually this value is used as it is.

`SER_ERR_RETRY_COUNT` sets the number of retries to be performed when a serial communication error occurs. Normally, use the default 3 as is (in most cases, a single retry is enough).

`USB_DATA_TIMEOUT` is a communication timeout. The default setting is 10 minutes, which might be long. If the host issues a USB bus reset for an inappropriate device, you do not need to use this function (leave it as the default). It is used to determine the error when the bus is stopped without issuing a bus reset.

It is expected that these values will be different for each environment. Therefore, EV tags can be used to customize from beSTORM's GUI. If you use the example, you can modify it from the SETTING screen as shown below.



Also, when this function is called, the following message will appear to prompt you to initialize the SOE (EZ-USB FX3). In that case, initialize the SOE (EZ-USB FX3) and select **OK**.

Wait for the SOE to be ready (`UAC_wait_for_start`)

Wait for the EZ-USB side to be able to communicate. Call the `UAC_wait_for_start` function as shown below. You don't need to specify the data.

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_wait_for_start SC"
Procedure="UAC_wait_for_start"/>
```

Set descriptor for USB2.0 (`UAC_set_dev_desc20`)

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_set_dev_desc20 SC"
Procedure="UAC_set_dev_desc20">
<S Name="DevDesc20" ParamName="DevDesc20">
<C Name="bLength" Value="0x12"/>
<C Name="bDescriptorType" Value="0x01"/>
<C Name="bcdUSB" Value="0x10 0x02"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
<C Name="bMaxPacketSize0" Value="0x40"/>
<C Name="idVendor" Value="0x45 0x04"/>
<C Name="idProduct" Value="0xf1 0x00"/>
<C Name="bcdDevice" Value="0x00 0x00"/>
<C Name="iManufacturer" Value="0x01"/>
<C Name="iProduct" Value="0x02"/>
<C Name="iSerialNumber" Value="0x00"/>
<C Name="bNumConfigurations" Value="0x01"/>
</S>
</SP>
```

Set the device descriptor with a C tag (fixed value) in compliance with the USB 2.0 specification in the data `DevDesc20`. The above is an example and can be set to any value.

Set descriptor for USB3.0 (`UAC_set_dev_desc30`)

Set the device descriptor for with a C tag (fixed value) in compliance with the USB 3.0 specification in the data `DevDesc30`. The above is an example and can be set to any value.

Set device qualifier descriptor (`UAC_set_qual_desc`)

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_set_qual_desc SC"
Procedure="UAC_set_qual_desc">
<S Name="QualDesc" ParamName="QualDesc">
<C Name="bLength" Value="0x0a"/>
<C Name="bDescriptorType" Value="0x06"/>
<C Name="bcdUSB" Value="0x00 0x02"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
<C Name="bMaxPacketSize0" Value="0x40"/>
<C Name="bNumConfigurations" Value="0x01"/>
<C Name="bReserved" Value="0x00"/>
</S>
</SP>
```

Set the device qualifier descriptor in compliance with USB specification with C tag (fixed value) in data `QualDesc`. The above is an example and can be set to any value.

Configuration descriptor for full speed (`UAC_set_fs_config`)

```
<sp library="c:\work\best_uac_usbd.dll" name="uac_set_fs_config sc"
procedure="uac_set_fs_config">
<s name="fsconfig" paramname="fsconfig">
<c name="blength" value="0x09"/>
<c name="bdescriptortype" value="0x02"/>
<c name="wtotallength" value="0x20 0x00"/>
<c name="bnuminterface" value="0x01"/>
<c name="bconfigurationvalue" value="0x01"/>
<c name="iconfiguration" value="0x00"/>
<c name="bmattributes" value="0x80"/>
<c name="bmaxpower" value="0x32"/>
<c name="blength" value="0x09"/>
<c name="bdescriptortype" value="0x04"/>
<c name="binterfacenumber" value="0x00"/>
<c name="balternatesetting" value="0x00"/>
<c name="bnumendpoints" value="0x02"/>
```

```
<c name="binterfaceclass" value="0x08"/>
<c name="binterfacesubclass" value="0x06"/>
<c name="binterfaceprotocol" value="0x50"/>
<c name="iinterfaceprotocol" value="0x00"/>
<c name="blength" value="0x07"/>
<c name="bdescriptortype" value="0x05"/>
<c name="bendpointaddress" value="0x81"/>
<c name="bmattributes" value="0x02"/>
<c name="wmaxpacketsize" value="0x40 0x00"/>
<c name="binterval" value="0x00"/>
<c name="blength" value="0x07"/>
<c name="bdescriptortype" value="0x05"/>
<c name="bendpointaddress" value="0x02"/>
<c name="bmattributes" value="0x02"/>
<c name="wmaxpacketsize" value="0x40 0x00"/>
<c name="binterval" value="0x00"/>
</s>
</sp>
```

- Set the descriptor in compliance with USB specification with C tag (fixed value) in the data `FSCONFIG`. The above is an example and can be set to any value.
- Set each descriptor that follows a configuration descriptor here. The above example sets the interface descriptor and the endpoint descriptor.
- The SOE (EZ-USB FX3) will configure the endpoint based on this endpoint descriptor.
- Endpoint numbers from 1 to 8 can be specified (`0x81 - 0x88` in the case of IN transfer).

Configuration descriptor for Super Speed (`UAC_set_ss_config`)

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_set_ss_config SC"
Procedure="UAC_set_ss_config">
<S Name="SSCONFIG" ParamName="SSCONFIG">
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x02"/>
<C Name="wTotalLength" Value="0x20 0x00"/>
<C Name="bNumInterface" Value="0x01"/>
<C Name="bConfigurationValue" Value="0x01"/>
<C Name="iConfiguration" Value="0x00"/>
<C Name="bmAttributes" Value="0x80"/>
<C Name="bMaxPower" Value="0x32"/>
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x04"/>
<C Name="bInterfaceNumber" Value="0x00"/>
<C Name="bAlternateSetting" Value="0x00"/>
<C Name="bNumEndpoints" Value="0x02"/>
```

```
<C Name="bInterfaceClass" Value="0x08"/>
<C Name="bInterfaceSubClass" Value="0x06"/>
<C Name="bInterfaceProtocol" Value="0x50"/>
<C Name="iInterfaceProtocol" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x81"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x04"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x06"/>
<C Name="bDescriptorType" Value="0x30"/>
<C Name="bMaxBurst" Value="0x0f"/>
<C Name="bmAttributes" Value="0x00"/>
<C Name="wBytesPerInterval" Value="0x00 0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x02"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x04"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x06"/>
<C Name="bDescriptorType" Value="0x30"/>
<C Name="bMaxBurst" Value="0x0f"/>
<C Name="bmAttributes" Value="0x00"/>
<C Name="wBytesPerInterval" Value="0x00 0x00"/>
</S>
</SP>
```

- Set the descriptor in compliance with USB specification with C tag (fixed value) in the data `SSCONFIG`. The above is an example and can be set to any value.
- Set each descriptor that follows a configuration descriptor here. The above example sets the interface descriptor, the endpoint descriptor, and the maximum length descriptor for Burst transfer as it is USB 3.0.
- The SOE (EZ-USB FX3) will configure the endpoint based on this endpoint descriptor.
- Endpoint numbers from 1 to 8 can be specified (`0x81 - 0x88` in the case of IN transfer).

Set string descriptor (Language ID) (`UAC_set_str_lang`)

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_set_str_lang SC"
Procedure="UAC_set_str_lang">
<S Name="StrLang" ParamName="StrLang">
<C Name="bLength" Value="0x04"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="wLANGID" Value="0x09 0x04"/>
```

```
</S>
</SP>
```

Set the string descriptor in compliance with USB specification with C tag (fixed value) in the data `StrLang`. The above is an example and can be set to any value.

Set string descriptor (Manufacturer) (`UAC_set_str_manufacture`)

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_set_str_
manufacture SC" Procedure="UAC_set_str_manufacture">
<S Name="StrManufacture" ParamName="StrManufacture">
<C Name="bLength" Value="0x10"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="bString" Value="0x43 0x00 0x79 0x00 0x70 0x00 0x72 0x00
0x65 0x00 0x73 0x00 0x73 0x00"/>
</S>
</SP>
```

Set the string descriptor in compliance with USB specification with C tag (fixed value) in the data `StrManufacture`. The above is an example and can be set to any value.

Set string descriptor (Product) (`UAC_set_str_product`)

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_set_str_product
SC" Procedure="UAC_set_str_product">
<S Name="StrProduct" ParamName="StrProduct">
<C Name="bLength" Value="0x08"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="bString" Value="0x55, 0x00, 0x41, 0x00, 0x43, 0x00"/>
</S>
</SP>
```

Set the string descriptor in compliance with USB specification with C tag (fixed value) in the data `StrProduct`. The above is an example and can be set to any value.

Start fuzzing sequence (`UAC_sequence_start`)

When the UAC_sequence_start function is called, the SOE (EZ-USB FX3) performs USB connection processing and enumeration with the target device. `UAC_sequence_start` must be placed immediately after the SC tag, and no data need to be specified.

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_sequence_start SC"
Procedure="UAC_sequence_start"/>
```

Wait for OUT transfer reception (UAC_recv_outep)

When the UAC_recv_outep function is called, it waits until the data of the specified endpoint is received. The USB IN/OUT direction is seen from the host's point of view, so from the peripheral's point of view, the OUT transfer means the reception direction.

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_recv_outep SC1"
Procedure="UAC_recv_outep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x02"/>
</S>
<S Name="BUS_RESET" ParamName="BUS_RESET">
<C ASCIIValue="SKIP" Name="bus_reset"/>
</S>
</SP>
```

- Set the endpoint number to be received by the EP with a C tag (fixed value). The above example waits for data from endpoint 0x02.
- Set the behavior when the bus reset is issued from the target (USB host) in BUS_RESET by selecting from the following. The above is an example of a SKIP.
    - SKIP Ignores the SKIP bus reset and continues to wait for the OUT transfer.
    - ERROR Treat the BUS_RESET as an error (warning) and stops the processing.
    - BREAK Does not cause an error but terminates UAC_recv_outep.

Process IN transfer transmission(UAC_send_inep)

When the UAC_recv_inep function is called, the data is sent from an arbitrary endpoint. The USB IN/OUT direction is seen from the host's point of view, so from the peripheral's point of view, the IN transfer means the transmission direction.

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_send_inep response
SC" Procedure="UAC_send_inep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x81"/>
</S>
<S Name="data" ParamName="data">
<B Name="faz" Value="0x30"/>
</S>
</SP>
```

- Set the endpoint number to be sent to the EP with a C tag (fixed value). The above example sends data to `0x81`.
- Set the payload data to be sent. In the above example, one fuzzing data (default `0x30`) is specified.

In the USB Peripheral Module, fuzzing data is sent from the `UAC_send_inep` function.

End fuzzing sequence (`UAC_sequence_end`)

When the UAC_sequence_end function is called, the SOE (EZ-USB FX3) performs USB disconnection and USB away from the target device in a USB-like manner. You don't need to specify the data.

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_sequence_end SC"
Procedure="UAC_sequence_end"/>
```

Sleep (`UAC_usb_sleep`)

Calling the `UAC_usb_sleep` function can make the device sleep for a certain amount of time, which is supposed to be used when you want to put some time after `UAC_sequence_end`, for example..

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_usb_sleep csw SC"
Procedure="UAC_usb_sleep">
<S Name="msec" ParamName="msec">
<C ASCIIValue="2000" Name="value"/>
</S>
</SP>
```

Set the sleep time (in `msec`) with a C tag (fixed value) in msec. In the above example, it waits for 2 seconds.

End USB fuzzing protocol (UAC_usb_stop)

This is to end fuzzing processing (`UAC_usb_stop` function). Make sure to call only once at the end. You don't need to specify the data.

```
<SP Library="C:\work\best_uac_usbd.dll" Name="UAC_usb_stop SC"
Procedure="UAC_usb_stop"/>
```

The DLL releases the source release etc. internally.

Auto-responding function

The USB Peripheral Module can automatically return up to 256-bytes of data for communications from non-fuzzed endpoints. The data returned automatically in the form of CSV, described as follows:

```
Number of data records,Target endpoint comparison size, comparison
data (multiple),transmission endpoint number,data size,transmission
data (multiple)
```

(Repeat the above for the number of records)

Also, when there is a # at the beginning of a line, it is considered a comment. The following is an example of two automated responses:

```
#Number of auto response data
0x2,
#EP,  len,  data ....................................., EP,  len,  data
0x00, 0x08, 0xa1, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x01, 0x01
#EP,  len,  data .................  EP,  len,  data .................
0x07, 0x04, 0x01, 0x02, 0x03, 0x04, 0x86, 0x04, 0xf1, 0xf2, 0xf3, 0xf4
```

- Red characters represent the endpoint number. Green characters represent the length of the data
- Gray characters represent that it is a comment line
- One data should be defined in one row, followed by a line-break.

In the above example, the response will be returned as follows.

- When the data `0xa1, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00` is received from `Endpoint 0` (control transfer), the data `0x01` is returned to `Endpoint 0` (control transfer).
- When the data `0x01,0x02,0x03,0x04` is received from `Endpoint 0x07`, the data `0xF1, 0xF2, 0xF3, 0xF4` is returned to `Endpoint 0x86`.

This behavior is triggered in the DLL regardless of `UAC_recv_outep` or `UAC_send_inep`, and the CSV file uses the data set at `UAC_usb_start` and the path defined in `CSV_FILE`.

Logging

The USB Peripheral Module has a function to output the status of communication with the SOE (EZ-USB FX3) to a separate log file, apart from beSTORM. By looking at this log, you can see what USB communication was done. The following is an excerpt from the actual log:

```
[2019/08/23 16:46:14.0624][beSTORM][enter UAC_sequence_start : 0-10
] [2019/08/23 16:46:14.0624]UAC_USBD_SequenceStart
[2019/08/23 16:46:14.0624]#--->UAC_SOE_OP_START_USB len 0
[2019/08/23 16:46:14.0624]
[2019/08/23 16:46:14.0827]#<--UAC_SOE_OP_START_USB len=4 [2019/08/23
16:46:14.0827] 00 00 00 00
[2019/08/23 16:46:14.0827] * SUCCESS
[2019/08/23 16:46:14.0827][beSTORM][enter UAC_recv_outep : 0-11 ]
[2019/08/23 16:46:15.0312]#<--UAC_SOE_OP_BUS_RESET len=0 [2019/08/23
16:46:15.0553]#<--UAC_SOE_OP_BUS_RESET len=0 [2019/08/23
16:46:15.0803]#<--UAC_SOE_OP_SETCONFIG len=4 [2019/08/23
16:46:15.0803] 00 00 00 00
[2019/08/23 16:46:15.0803] * ConfigValue:0x0
[2019/08/23 16:46:16.0047]#<--UAC_SOE_OP_CTRL_REQ_RECV len=8
[2019/08/23 16:46:16.0047] a1 fe 00 00 00 00 01 00
[2019/08/23 16:46:16.0047]#--->UAC_SOE_OP_CTRL_RESPONSE len 1
[2019/08/23 16:46:16.0047] 0x01
[2019/08/23 16:46:16.0066]#<--UAC_SOE_OP_OUT_RECV len=35
[2019/08/23 16:46:16.0066]  02 00 00 00 55 53 42 43 d0 34 5d 0f 24
00 00 00 80 00 06 12 00 00 00 24 00 00 00 00 00 00 00 00 0000 00
[2019/08/23 16:46:16.0066] * Endpoint:0x02 [2019/08/23
16:46:16.0066] * length=31
```

The log file is output to the path defined in the data `LOG_FILE` at `UAC_usb_start`.

It is also output to Windows debug messages during fuzzing to check the operation. Therefore, you can use tools such as DbgView to check the running status in real-time.

For details of the log contents and debug messages, please refer to the Logging chapter.

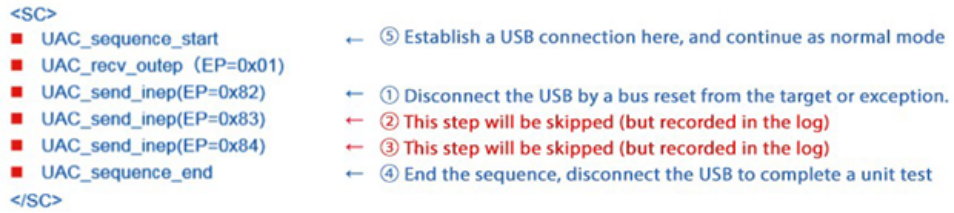
Behavior in case of abnormalities

Behavior when an error occurs

The designed behavior for exceptions such as `UAC_recv_outep` (when `BUS_RESET` is `ERROR`) or bus reset is issued from the target during the processing of `UAC_send_inep` is as follows:

- Sleeps for the time set by the data `ILLEGAL_TIMER` of `UAC_usb_start`.
- Indicates a USB disconnect to the `SOE (EZ-USB FX3)`.
- Skips all DLL requests without processing until the next `UAC_sequence_end` process.
- Logs the skip.

The following is an example of the behavior in case of an abnormality:



`UAC_sequence_end` process disconnects the USB and waits until the USB is disconnected. The system is also implemented to retry when the disconnect cannot be confirmed after a certain period of time (the retry process is performed by the monitor described in ***Monitoring function* on page 28**)>

`UAC_sequence_end` process is dedicated only to disconnecting the USB. Therefore, if an error occurs during the `UAC_sequence_end` process, it is left in the log, and if necessary, a USB disconnection request is issued again.

When a serial communication error occurs, the DLL retries the last packet it tried to send. The maximum number of retries is the number of times set by `SER_ERR_RETRY_COUNT` in ***Begin USB fuzzing protocol (UAC_usb_start)* on page 17**.

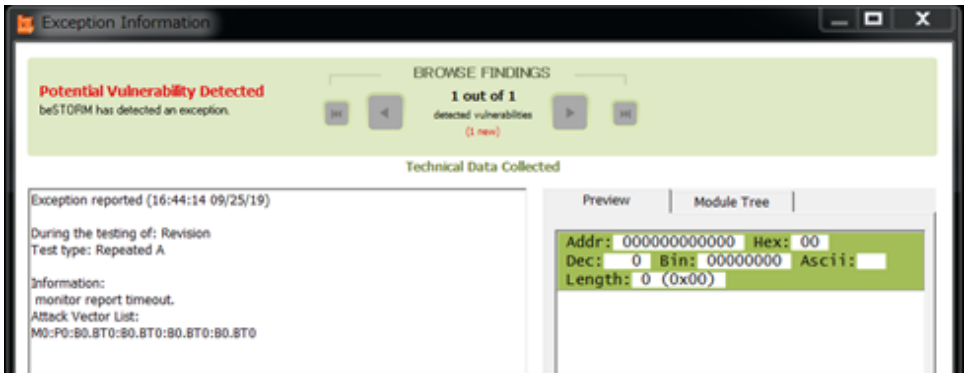
When the completion of the IN and OUT transfer of USB does not return to the DLL until the time specified by `USB_DATA_TIMEOUT` in ***Begin USB fuzzing protocol (UAC_usb_start)* on page 17**, it skips all DLL requests without processing until the next `UAC_sequence_end` processing as an error.

Monitoring function

The USB Peripheral Module has a monitor function that performs the following monitoring processes:

1. When there is no communication with the SOE (EZ-USB FX3) for a certain period of time (about 3 minutes), an `EXCEPTION` is issued to beSTORM to notify beSTORM that it is in an abnormal condition.
2. When a USB disconnection is issued in `UAC_sequence_end` process, it monitors the completion of the disconnection every 10 seconds. If the completion of the disconnection cannot be observed, it will be disconnected again. The purpose of this design is to ensure a secure transition to the disconnected state at the end of each test.

Therefore, if serial communication with the SOE (EZ-USB FX3) does not occur at all for some reason, the following error notification screen will appear on the beSTORM side.



Limitations and constraints

The restrictions are as follows:

- Currently, only support bulk transfer fuzzing is supported.
- Currently, only the fuzzing described in the Sample for Mass Storage and Customize the USB Peripherial Module chapters has been tested.
- Fuzzing of control transfer is not supported (vendor and class requests will be supported in the future).
- Fuzzing for interrupt transfer is not supported (to be supported in the future).
- Automatic responses are now only available for control transfers (other transfers will be supported in the future).

As for the isochronous transfer, there is no plan to support it for now (please contact us if necessary). The maximum transfer size for endpoint data is 512-bytes.

Overview USB Mass Storage Class

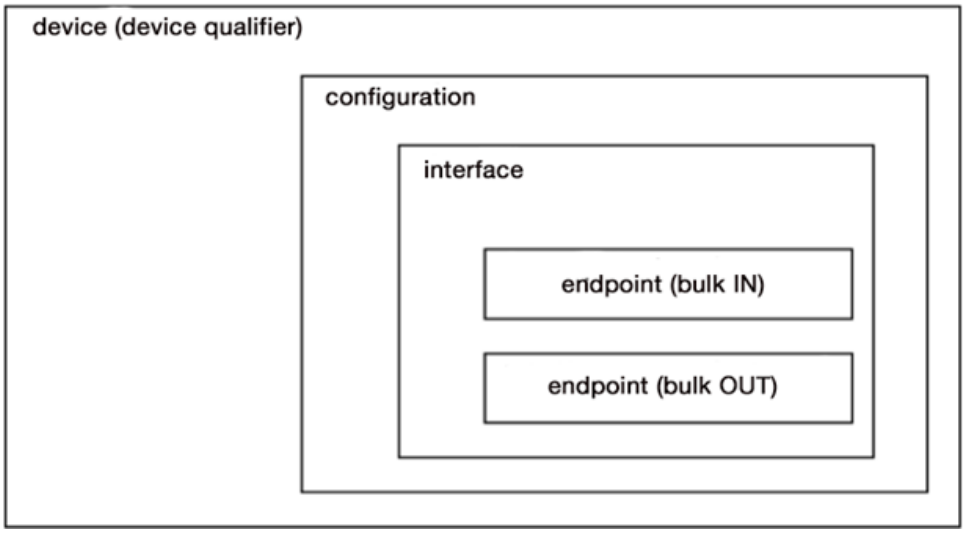
Storage devices such as USB memory conforms to the USB mass storage class specifications. In the mass storage class, the USB host issues SCSI/ATAPI commands to control the storage device. This section describes the Overview of the USB mass storage class (Bulk Only protocol).

Device configuration

USB mass storage class (Bulk Only protocol) requires following descriptors:

- Device descriptor (device qualifier descriptor).
- Configuration descriptor
    - Interface descriptor
    - Endpoint descriptor (bulk IN and bulk OUT)

The interface descriptor belongs to the configuration. The figure below shows the descriptor configuration of a typical USB storage device.



In addition to those above, string descriptors containing texts (such as a device name) can be defined.

Device (device qualifier) descriptors and configuration descriptors are described in compliance to the general USB specification. Interface descriptors are defined as follows (it will be noted here that it is mass storage):

Interface descriptors

Field name	Description
bLength	Descriptor Size (9).
bDescriptor	Descriptor Type (4)..
bInterFaceNumber	Identification number of this interface (optional).
bAlternateSetting	Setting value to select an alternative setting (default 0).
bNumEndpoints	The number of endpoints an interface can have.
bInterfaceClass	Class code - Storage class is 8.
bInterfaceSubClass	Subclass code - if it is a storage device with SCSI command specifications, it is set to 6.
bInterfaceProtocol	Protocol code (0x50 for bulk-only).
iInterface	Interface string descriptor index.

It is necessary to have two endpoint descriptors - bulk IN and OUT. The content of the descriptor itself is set in compliance to the USB standard specification. Endpoint descriptors are defined as follows:

Bulk IN

Field name	Description
bLength	Descriptor Size (7).
bDescriptor	Descriptor Type (5).
bEndpointAddress	Endpoint number 0x8n: n is an arbitrary endpoint number.
bmAttributes	Attribute - 0x02 Bulk.
wMaxPacketSize	Maximum packet size Default 64 for Full, 512 for high, and 1024 for SS.
bInterval	The polling interval is 0.

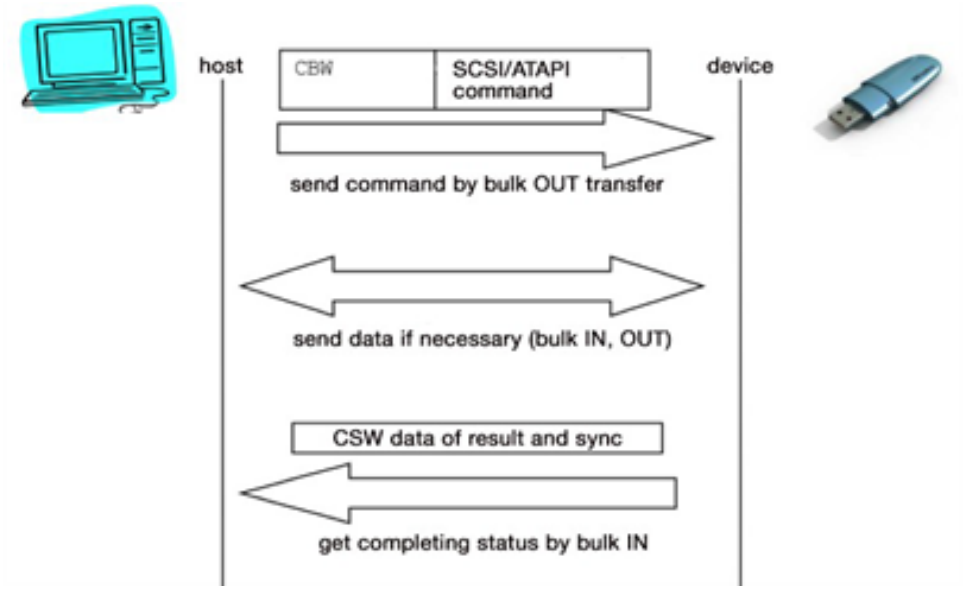
Bulk OUT

Field name	Description
bLength	Descriptor Size (7).
bDescriptor	Descriptor Type (5).
bEndpointAddress	Endpoint number 0x0 : is an arbitrary endpoint number.
bmAttributes	Attribute - 0x02 Bulk.
wMaxPacketSize	Maximum packet size Default 64 for Full, 512 for high, and 1024 for SS.
bInterval	The polling interval is 0.

Data format

Control of the device is done using bulk transfer as follows.

1. Send the structured data called CBW (CommandBlockWrapper) to the device in a bulk OUT transfer. The SCSI/ATAPI command is also included in this CBW.
2. When involving data transfer. it is done as using the bulk endpoint.
3. After the command processing is completed, the device sends the structured data called CSW (CommandStatusWrapper) as a status to the bulk IN transfer and notifies the USB host.



CBW (CommandBlockWrapper)

CBW is defined as follows:

Common block wrapper

Bit-Byte	7	6	5	4	3	2	1	0
0-3	dCBWSignature							
4-7	dCBWTag							
8-11	dCBWDataTransferLength							
12	bmCBWFlags							
13	Reserved (0)				bCBWLUN			
14	Reserved (0)				bCBWCBLength			
15-30	CBWCB							

- Set 43425355h (that is, USBC) in Little Endian to `dCBWSignature`.
- Set any number to `dCBWTag`, as it is used as a tag. The device keeps this value and puts the value given by this `dCBWTag` into the status data (CSW) returned to the host at the end of command processing. Hosts and devices can use this tag to check if they are synchronized or not.
- Set the length of the data to `dCBWDataTransferLength`. Set this to 0 if you want to issue a command with no data.
- Set the transfer direction to `bmCBWFlags: D7` is 0 for data output, and D7 is 1 for data input.
- Set a drive number specification to `bCBWLUN`.
- Set the length of the command packet to `bCBWCBLength`.
- This means that the SCSI/ATAPI command data itself will be placed in the `CBWCB` location. Since CBW is fixed at 31 bytes, the rest of the SCSI/ATAPI command data is padded with 0.

CSW (Command Status Wrapper)

CSW is defined as follows:

Command status wrapper

Bit-Byte	7	6	5	4	3	2	1	0
0-3	dCBWSignature							
4-7	dCBWTag							
8-11	dCSWDataResidue							
12	bCSWStatus							

- Set 53425355h (that is, "USBS") in Little Endian to `dCSWSignature`.
- Set the same number as CBW's `dCBWTag` to `dCBWTag`.
- `dCSWDataResidue` means "leftover data." If the length of data the host has processed differs from the length of data specified by `dCBWDataTransferLength` in the CBW and the actual length of data processed, the device returns the difference by setting `dCSWDataResidue`. The host can look at this to see if all the data has been successfully processed or not.
- `bCSWStatus` is the state when the command processing is terminated. The device sets the appropriate value here from the following:

Value of bCSWStatus

Value	Description
0x00	Command processing succeeded.
0x01	Command processing failed.
0x02	Command processing was out of sync between host and device, so processing failed.
0x03 or larger	Reserved.

Get Max LUN

USB mass storage class (Bulk Only protocol) defines the `Get Max Lun` with a class request. The definition of a request is as follows:

Get Max LUN

Field name	Description
bmRequest	0xa1 (to interface)
bRequest	0xfe (Get Max LUN))
wValue	0
wIndex	Set the bInterfaceNumber of the control target to the lower byte
wLength	1

The device receives the above and returns the maximum LUN of 1 byte. Typically, the number of drives is set to -1.

Inquiry command

This section explains the definition of the Inquiry command that is a fuzzing target in Sample for Mass Storage chapter.

The INQUIRY command is used to obtain information and functions of the device. The information is defined to include the type of device, whether it is removable or not, as well as a string of characters such as the name of the product or the manufacturer.

Command (host to device)

Inquiry command

Bit-Byte	7	6	5	4	3	2	1	0
0	Operation Code (12h)							
1	Reserved							EPVD
2	Page Code							
3	Reserved							
4	Allocation Length							
5	Control							

- EPVD and Page Code select the data to be retrieved, where 0 means the default INQUIRY information; anything other than 0 will allow you to retrieve product data called Vital Product Data (VPD). By default, 0 is specified.
- Allocation length is the host's buffer size. If the data to be returned is greater than or equal to Allocation Length, the device will try to return the data in Allocation Length bytes.
- Control is a configuration item for issuing commands in batch. By default, it is set to 0.

Response (device to host)

Response of inquiry command

Bit-Byte	7	6	5	4	3	2	1	0
0	Peripheral Qualifier			Peripheral Device Type				
1	RMB	Reserved						

Bit-Byte	7	6	5	4	3	2	1	0
2	Version							
3	Reserved		NORMACA	HISUP	Response Data Format			
4	Additional Length(n – 4)							
5	SCCS	ACC	TPGS		3PC	Reserved		PROTECT
6	Reserved	ENCSERV	VS	MULTIP	Reserved			
7	Reserved						CmdQue	VS
8	(MSB)	Vendor Information						(LSB)
15								
16	(MSB)	Product Identification						(LSB)
31								
32	(MSB)	Product Revision Information						(LSB)
35								

In the above example, it is 36 bytes, but it is possible to expand it further and add the vendor's own data. *VS* means *Vendor Specific* and can be defined by the vendor, but by default it should be set to 0.

Peripheral qualifier

- 0 The specified LUN drive is an input/output device defined by the Peripheral Device Type.
- 1 The specified LUN drive is an input/output device defined by the Peripheral Device Type, but it is not actually connected.
- 3 The specified LUN drive is not supported.

Peripheral device type (excerpts of key items)

- 0 Direct access device (e.g., magnetic disks such as HDDs) 1 Sequential access device (for example, magnetic tape).
- 2 Printer.
- 3 Processor.
- 4 Write-once device (device that can be written only once) 5 CD/DVD device.
- 6 Scanner.

- 7 Optical memory device.
- 8 Media Changer device (device like a jukebox).
- 9 Communication device (device that can be used for communication, etc.) 12 RAID device.

RMB

- 0 Devices that cannot remove the media from the drive (such as fixed HDDs).
- 1 Devices that can remove media from the drive (such as CDs and DVDs).

Version: Compatible standard (extracts of key items)

- 0 Not compatible with any specific standard.
- 3 ANSI INCITS 301-1997 (SPC).
- 4 ANSI INCITS 351-2001 (SPC-2).
- 5 ANSI INCITS 408-2005 (SPC-3).
- 6 ANSI INCITS 513-2015 (SPC-4).
- 7 T10/BSR INCITS 503 (SPC-5).

NORMACA

- 0 NACA (Normal ACA) is not supported.
- 1 NACA (Normal ACA) is supported.

HISUP

- 0 Hierarchical Support Addressing is not used.
- 1 Hierarchical Support Addressing is used.

Response data format

Specify the version of the INQUIRY information. By default, the value is set to 2.

- 0 Information defined by the SCSI-1 standard.
- 1 Information defined in CCS (ATAPI X3T9.2/85-92) 2 Information defined by the SCSI-2 standard.

Additional Length(n-4)

It refers to an additional data length. Specifically, it is the length from Byte5 as mentioned in the previous section. If there is no additional data in the standard INQUIRY, it will be set to "31."

SCCS

- 0 SCC is not supported.
- 1 SCC is supported.

ACC

- 0 ACC (Access Controls Coordinator) is not supported.
- 1 ACC (Access Controls Coordinator) is supported.

TPGS

- 0 Asymmetric logical unit access is not supported.
- 1 Implicit asymmetric logical unit access is supported.
- 2 Explicit asymmetric logical unit access is supported.
- 3 Both Implicit and Explicit are supported.

3PC

- 0 3PC (Third-Party Copy) function is not supported
- 1 3PC (Third-Party Copy) function is supported

PROTECT

- 0 protection feature is not supported
- 1 protection function is supported

ENCSERV

- 0 Enclosure Services feature is not supported.
- 1 Enclosure Services feature is supported

MULTIP

- 0 Multi Port function is not supported.
- 1 Multi Port function is supported.

CmdQue

- 0 Command queuing is not supported.
- 1 Command queuing is supported.

Vendor information

Manufacturer information/ASCII code string

Product identification

Product Information/ASCII Code String

Product revision level

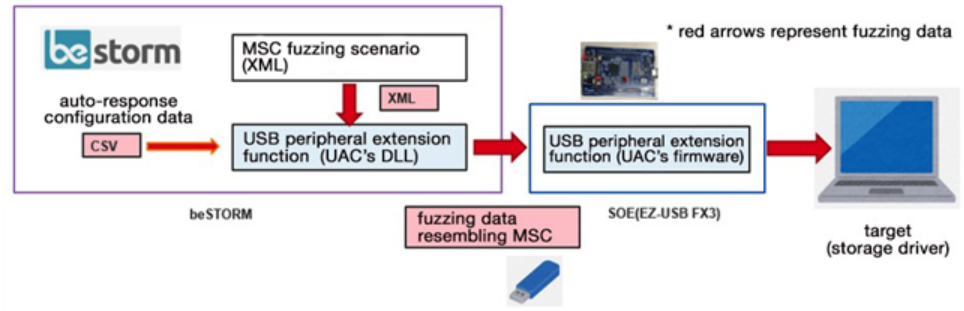
Version information/ASCII code string

Sample for Mass Storage

The USB Peripheral Module provides XML and CSV fuzzing samples for mass storage of the USB host. This section describes the fuzzing of mass storage using them.

System overview

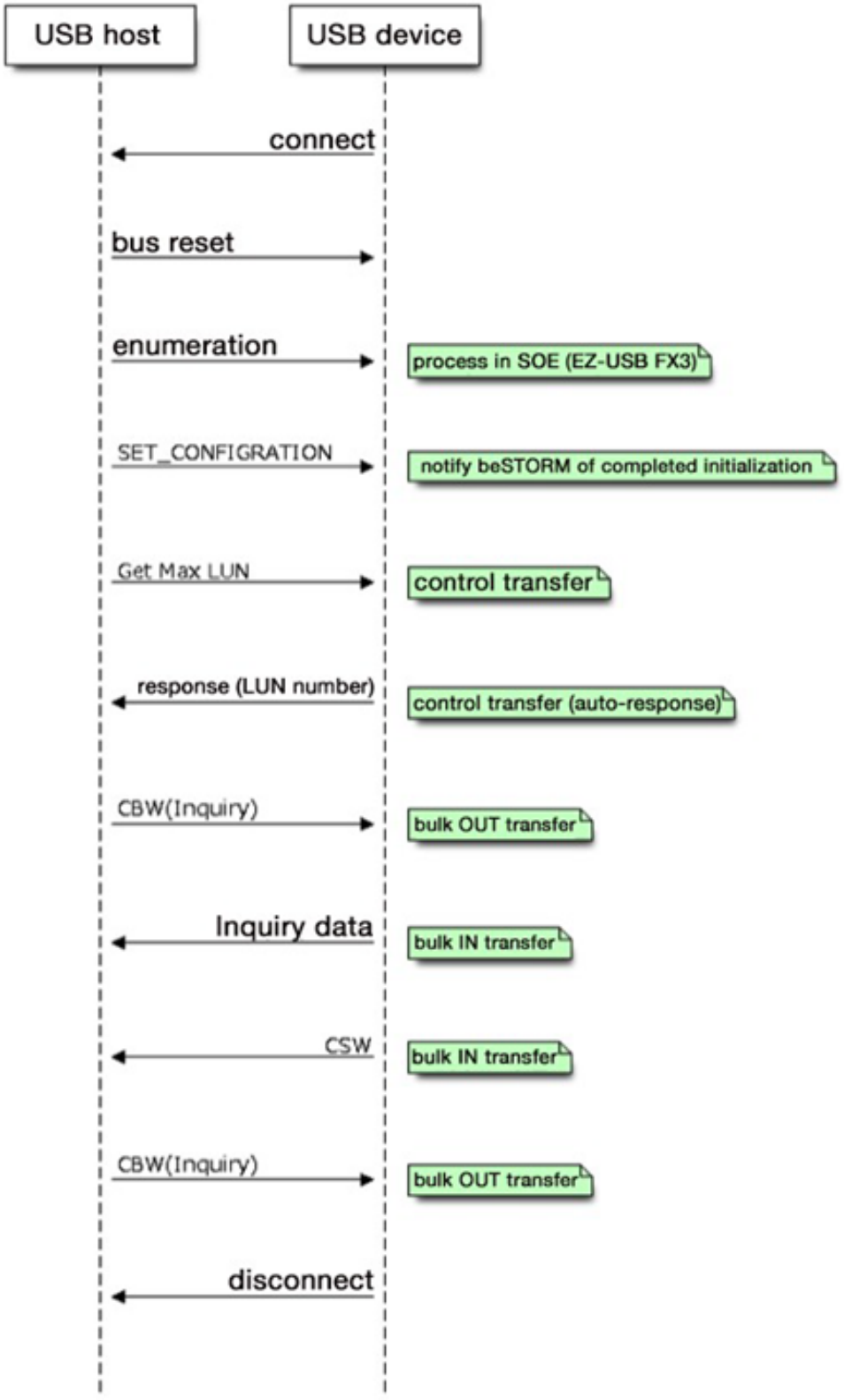
The basic configuration of the mass storage fuzzing system is shown below:



Using the USB Peripheral Module (`best_uac_usbd.dll`) described in the Overview of UAC USB Peripheral Module chapter, we have prepared XML and CSV files for sending mass storage device data. These can be used to check fuzzing behavior against mass storage hosts.

Content for mass storage class fuzzing

The following figure shows the content of fuzzing (usually assumed sequence) and the insertion point of fuzzing data in the sample.


USB host
USB device
connect
bus reset
enumeration
process in SOE (EZ-USB FX3)
SET_CONFIGRATION
notify beSTORM of completed initialization
Get Max LUN
control transfer
response (LUN number)
control transfer (auto-response)
CBW(Inquiry)
bulk OUT transfer
Inquiry data
bulk IN transfer
CSW
bulk IN transfer
CBW(Inquiry)
bulk OUT transfer
disconnect

Response data and CSW includes fuzzing data created by beSTORM. Wait for the next command.

- Usually, the USB mass storage host issues an Inquiry command first to learn about the device's characteristics.
- Fuzzing is performed on the response data and status response data (CSW) of the command (red part in the above figure).
- After responding to the data, it waits for the next command (to stop here if the host is defective and not able to respond).
- Then, the USB is disconnected by beSTORM (USB peripheral), reconnected, and the test is repeated.
- It returns 1 back to Get Max Lun as an auto-response.

XML

The XML for this fuzzing is shown below:

```
<?xml version="1.0" ?>
<!DOCTYPE beSTORM SYSTEM '\Program Files\beSTORM\beSTORM.dtd'>
<beSTORM Revision="$Revision: 7298 $" Version="1.2">
<Global/>
<GeneratorOptSettings>
<BT FactoryDefined="1" FactoryType="Binary"/>
</GeneratorOptSettings>
<ModuleSettings>
<M Name="UAC USB peripheral1">
<P Name="UAC USB peripheral1 protocol">
<SC Name="Initialize">
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_usb_start SC"
Procedure="UAC_usb_start">
<S Name="COM_PORT" ParamName="COM_PORT">
<EV ASCIIValue="4" Description="COM" Name="COM" Required="1"/>
</S>
<S Name="CSV_FILE" ParamName="CSV_FILE">
<EV ASCIIValue="C:\TEMP\bestorm.csv" Description="CSV_ATH"
Name="CSV_ATH" Required="1"/>
</S>
<S Name="LOG_FILE" ParamName="LOG_FILE">
<EV ASCIIValue="C:\TEMP\bestorm.log" Description="LOG_PATH"
Name="LOG_PATH" Required="1"/>
</S>
<S Name="RECORD_FILE" ParamName="RECORD_FILE">
<EV ASCIIValue="c:\TEMP\best_record.dat" Description="RECORD_
```

```
PATH" Name="RECORD_PATH" Required="1"/>
</S>
<S Name="ILLEGAL_TIMER" ParamName="ILLEGAL_TIMER">
<EV ASCIIValue="3000" Description="timer" Name="timer"
Required="1"/>
</S>
<S Name="SER_ERR_RETRY_COUNT" ParamName="SER_ERR_RETRY_COUNT">
<EV ASCIIValue="3" Description="ser_err_retry_count" Name="ser_
err_retry_count" Required="1"/>
</S>
<S Name="USB_DATA_TIMEOUT" ParamName="USB_DATA_TIMEOUT">
<EV ASCIIValue="600" Description="usb_data_timeout" Name="usb_
data_timeout" Required="1"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_wait_for_
start SC" Procedure="UAC_wait_for_start"/>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_dev_
desc20 SC" Procedure="UAC_set_dev_desc20">
<S Name="DevDesc20" ParamName="DevDesc20">
<C Name="bLength" Value="0x12"/>
<C Name="bDescriptorType" Value="0x01"/>
<C Name="bcdUSB" Value="0x10 0x02"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
<C Name="bMaxPacketSize0" Value="0x40"/>
<C Name="idVendor" Value="0x45 0x04"/>
<C Name="idProduct" Value="0xf1 0x00"/>
<C Name="bcdDevice" Value="0x00 0x00"/>
<C Name="iManufacturer" Value="0x01"/>
<C Name="iProduct" Value="0x02"/>
<C Name="iSerialNumber" Value="0x00"/>
<C Name="bNumConfigurations" Value="0x01"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_dev_
desc30 SC" Procedure="UAC_set_dev_desc30">
<S Name="DevDesc30" ParamName="DevDesc30">
<C Name="bLength" Value="0x12"/>
<C Name="bDescriptorType" Value="0x01"/>
<C Name="bcdUSB" Value="0x00 0x03"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
<C Name="bMaxPacketSize0" Value="0x09"/>
<C Name="idVendor" Value="0x45 0x04"/>
<C Name="idProduct" Value="0xf1 0x00"/>
```

```
<C Name="bcdDevice" Value="0x00 0x00"/>
<C Name="iManufacturer" Value="0x01"/>
<C Name="iProduct" Value="0x02"/>
<C Name="iSerialNumber" Value="0x00"/>
<C Name="bNumConfigurations" Value="0x01"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_qual_desc
SC" Procedure="UAC_set_qual_desc">
<S Name="QualDesc" ParamName="QualDesc">
<C Name="bLength" Value="0x0a"/>
<C Name="bDescriptorType" Value="0x06"/>
<C Name="bcdUSB" Value="0x00 0x02"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
<C Name="bMaxPacketSize0" Value="0x40"/>
<C Name="bNumConfigurations" Value="0x01"/>
<C Name="bReserved" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_fs_config
SC" Procedure="UAC_set_fs_config">
<S Name="FSCONFIG" ParamName="FSCONFIG">
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x02"/>
<C Name="wTotalLength" Value="0x20 0x00"/>
<C Name="bNumInterface" Value="0x01"/>
<C Name="bConfigurationValue" Value="0x01"/>
<C Name="iConfiguration" Value="0x00"/>
<C Name="bmAttributes" Value="0x80"/>
<C Name="bMaxPower" Value="0x32"/>
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x04"/>
<C Name="bInterfaceNumber" Value="0x00"/>
<C Name="bAlternateSetting" Value="0x00"/>
<C Name="bNumEndpoints" Value="0x02"/>
<C Name="bInterfaceClass" Value="0x08"/>
<C Name="bInterfaceSubClass" Value="0x06"/>
<C Name="bInterfaceProtocol" Value="0x50"/>
<C Name="iInterfaceProtocol" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x81"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x40 0x00"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
```

```
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x02"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x40 0x00"/>
<C Name="bInterval" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_hs_config
SC" Procedure="UAC_set_hs_config">
<S Name="HSCONFIG" ParamName="HSCONFIG">
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x02"/>
<C Name="wTotalLength" Value="0x20 0x00"/>
<C Name="bNumInterface" Value="0x01"/>
<C Name="bConfigurationValue" Value="0x01"/>
<C Name="iConfiguration" Value="0x00"/>
<C Name="bmAttributes" Value="0x80"/>
<C Name="bMaxPower" Value="0x32"/>
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x04"/>
<C Name="bInterfaceNumber" Value="0x00"/>
<C Name="bAlternateSetting" Value="0x00"/>
<C Name="bNumEndpoints" Value="0x02"/>
<C Name="bInterfaceClass" Value="0x08"/>
<C Name="bInterfaceSubClass" Value="0x06"/>
<C Name="bInterfaceProtocol" Value="0x50"/>
<C Name="iInterfaceProtocol" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x81"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x02"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x02"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x02"/>
<C Name="bInterval" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_ss_config
SC" Procedure="UAC_set_ss_config">
<S Name="SSCONFIG" ParamName="SSCONFIG">
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x02"/>
<C Name="wTotalLength" Value="0x20 0x00"/>
<C Name="bNumInterface" Value="0x01"/>
```

```
<C Name="bConfigurationValue" Value="0x01"/>
<C Name="iConfiguration" Value="0x00"/>
<C Name="bmAttributes" Value="0x80"/>
<C Name="bMaxPower" Value="0x32"/>
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x04"/>
<C Name="bInterfaceNumber" Value="0x00"/>
<C Name="bAlternateSetting" Value="0x00"/>
<C Name="bNumEndpoints" Value="0x02"/>
<C Name="bInterfaceClass" Value="0x08"/>
<C Name="bInterfaceSubClass" Value="0x06"/>
<C Name="bInterfaceProtocol" Value="0x50"/>
<C Name="iInterfaceProtocol" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x81"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x04"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x06"/>
<C Name="bDescriptorType" Value="0x30"/>
<C Name="bMaxBurst" Value="0x0f"/>
<C Name="bmAttributes" Value="0x00"/>
<C Name="wBytesPerInterval" Value="0x00 0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x02"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x04"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x06"/>
<C Name="bDescriptorType" Value="0x30"/>
<C Name="bMaxBurst" Value="0x0f"/>
<C Name="bmAttributes" Value="0x00"/>
<C Name="wBytesPerInterval" Value="0x00 0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_str_lang
SC" Procedure="UAC_set_str_lang">
<S Name="StrLang" ParamName="StrLang">
<C Name="bLength" Value="0x04"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="wLANGID" Value="0x09 0x04"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_str_
manufacture SC" Procedure="UAC_set_str_manufacture">
<S Name="StrManufacture" ParamName="StrManufacture">
```

```
<C Name="bLength" Value="0x10"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="bString" Value="0x43 0x00 0x79 0x00 0x70 0x00 0x72
0x00 0x65 0x00 0x73 0x00 0x73 0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_str_
product SC" Procedure="UAC_set_str_product">
<S Name="StrProduct" ParamName="StrProduct">
<C Name="bLength" Value="0x08"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="bString" Value="0x55, 0x00, 0x41, 0x00, 0x43, 0x00"/>
</S>
</SP>
</SC>
<SC Name="Fuzzing">
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_sequence_
start SC" Procedure="UAC_sequence_start"/>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_recv_outep
SC1" Procedure="UAC_recv_outep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x02"/>
</S>
<S Name="BUS_RESET" ParamName="BUS_RESET">
<C ASCIIValue="SKIP" Name="bus_reset"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_send_inep
response SC" Procedure="UAC_send_inep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x81"/>
</S>
<S Name="data" ParamName="data">
<B MaxBytes="1" MinBytes="1" Name="PeripheralQualifier"
Value="0x00"/>
<B MaxBytes="1" MinBytes="1" Name="RMB" Value="0x80"/>
<B MaxBytes="1" MinBytes="1" Name="Versions" Value="0x06"/>
<B MaxBytes="1" MinBytes="1" Name="Response Data Format"
Value="0x02"/>
<B MaxBytes="4" Name="Length" Value="0x3b"/>
<B MaxBytes="2" MinBytes="2" Name="Reserve" Value="0x00 0x00"/>
<B MaxBytes="1" MinBytes="1" Name="RelADR plus" Value="0x00"/>
<B MaxBytes="16" Name="Vendor" Value="0x30 0x31 0x32 0x33 0x34
0x35 0x36 0x37"/>
<B MaxBytes="32" Name="Product" Value="0x61 0x62 0x63 0x64 0x65
0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70"/>
<B MaxBytes="8" Name="Revision" Value="0x30 0x31 0x32 0x33"/>
</S>
```

```
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_send_inep csw
SC" Procedure="UAC_send_inep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x81"/>
</S>
<S Name="data" ParamName="data">
<B MaxBytes="4" Name="dCSWSignature" Value="0x55 0x53 0x42
0x53"/>
<B MaxBytes="4" Name="dCSWTag" Value="0x80 0x00 0x00 0x00"/>
<B MaxBytes="4" Name="dCSWDataResidue" Value="0x00 0x00 0x00
0x00"/>
<B MaxBytes="1" Name="bCSWStatus" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_recv_outep
SC2" Procedure="UAC_recv_outep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x02"/>
</S>
<S Name="BUS_RESET" ParamName="BUS_RESET">
<C ASCIIValue="BREAK" Name="bus_reset"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_sequence_end
SC" Procedure="UAC_sequence_end"/>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_usb_sleep csw
SC" Procedure="UAC_usb_sleep">
<S Name="msec" ParamName="msec">
<C ASCIIValue="2000" Name="value"/>
</S>
</SP>
</SC>
<SC Name="Deinitialize">
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_usb_stop SC"
Procedure="UAC_usb_stop"/>
</SC>
</P>
</M>
</ModuleSettings>
</beSTORM>
```

- The code consists of 3 sequences (SC tags). Not only mass storage, but also the UAC USB Peripheral Module requires such a structure to be used.

- Initialize sequence containing procedures for setting descriptor and called only once at the beginning.
- Fuzzing sequence (that is, this part is iterated).
- Deinitialize sequence containing procedures for termination and called only once at the end.

Initialize sequence (called only once at the beginning)

- `UAC_usb_start` is prepared as described in the *Begin USB fuzzing protocol (UAC_usb_start)* section of the Overview of UAC USB Peripheral Module chapter.
- Wait for connection from the SOE (EZ-USB FX3) with `UAC_wait_for_start`.
- Set descriptors for bulk-only USB mass storage devices, which are often used in USB memory, by calling following functions of the DLL. Please refer to the *How to use functions* section of the Overview of UAC USB Peripheral Module chapter for the specification of each function.
  - `UAC_set_dev_desc20`
  - `UAC_set_dev_desc30`
  - `UAC_set_qual_desc`
  - `UAC_set_fs_config`
  - `UAC_set_hs_config`
  - `UAC_set_ss_config`
  - `UAC_set_str_lang`
  - `UAC_set_str_manufacture`
  - `UAC_set_str_product`

Fuzzing sequence (this part will be iterated)

- Establish a USB connection with `UAC_sequence_start`.
- Receive data from the mass storage host with `UAC_recv_outp` (that is, receive command data called CBW, in which the Inquiry command should be found)
  - Put the endpoint number in the EP. This is where the bulk OUT endpoint number is specified. This value is supposed to be the same value set by the descriptor.
- Send fuzzed inquiry data with `UAC_send_inep`.

    - A B tag is used here to write XML to use the fuzzing data of beSTORM.
    - Put the endpoint number in the EP. This is where the bulk IN endpoint number is specified. This value is supposed to be the same value set by the descriptor.
- Again, send CSW status data to the host from the peripheral side with `UAC_send_inep`.
    - A B tag is used here to write XML to use the fuzzing data of beSTORM.
    - Put the endpoint number in the EP. This is where the bulk IN endpoint number is specified. This value is supposed to be the same value set by the descriptor.
- Receive data from the mass storage host with `UAC_recv_outp` (that is, receive command data called CBW - It doesn't matter what the content of the command is, as it's just a readout.).
- Disconnect the USB with `UAC_sequence_end`.
- Wait for 2 seconds with `UAC_usb_sleep`. This is to ensure the disconnection.

Deinitialize sequence (called only once at the end)

- `UAC_usb_stop` - to call the end process on the DLL side.

CSV

The following is a CSV that automatically responds to the Get Max LUN control transfer.

```
#Number of auto response data
0x1,
#EP,  len,  data ..........................................,  EP,  len,  data
0x00, 0x08, 0xa1, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x01, 0x00
```

■ Red characters represent the endpoint number. Green characters represent the length of the data
■ Gray characters represent that it is a comment line

Since the class request packet of `Get Max LUN` is `0xa1, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00`, it is set to automatically respond to 1 byte of data when receiving it.

USB Fuzzing Setup

To perform USB fuzzing in beSTORM, do the following:

Acquire the hardware

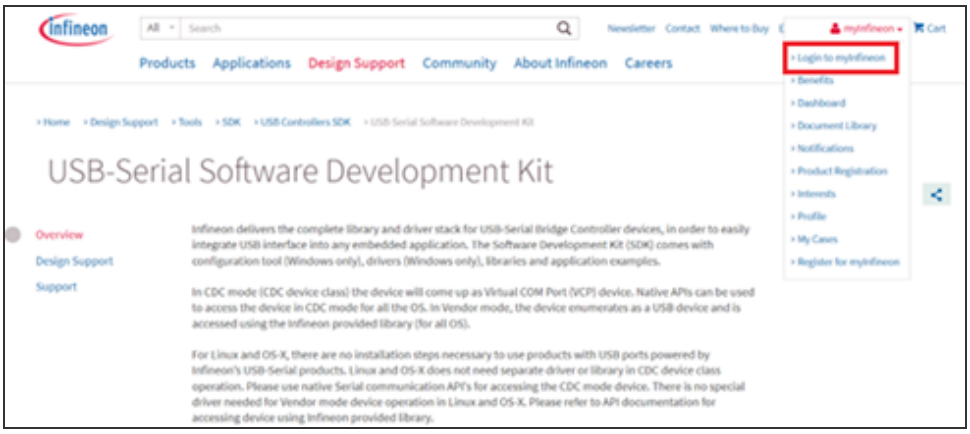
Before you can fuzz with beSTORM, you will need to supply the following third-party hardware:

- (1) **CYUSB3KIT-003 EZ-USB FX3 SuperSpeed Explorer Kit** (to purchase, go to the EZ-USB FX3 SuperSpeed Explorer Kit product page athttps://www.infineon.com/cms/en/product/evaluation-boards/cyusb3kit-003/)
    - This kit includes:
        - (1) EZ-USB FX3 board
        - (1) USB 3.0 A to B cable
        - (4) plastic board jumpers (two jumpers will come preinstalled on the board)
        - (1) EZ-USB FX3 SuperSpeed Explorer Kit Quick Start Guide (you will use this to identify parts of the board in later sections)
- (1) **USB 2.0 A to 5-pin Micro-USB Type B cable**

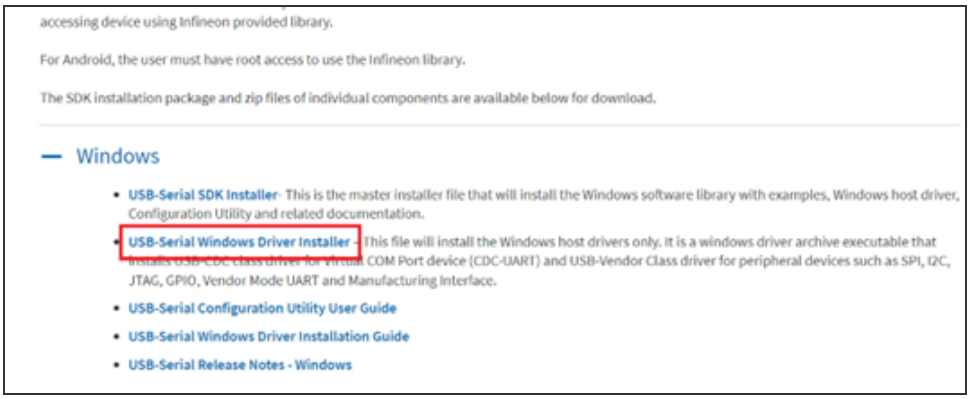
Install the EZ-USB FX3 board

After receiving your EZ-USB FX3 SuperSpeed Explorer Kit, do the following to install the EZ-USB FX3 board on the beSTORM computer:

1. Download and install the Windows driver:
    a. Go to the Infineon USB-Serial Software Development Kit page at https://www.infineon.com/cms/en/design-support/tools/sdk/usb-controllers-sdk/usb-serial-software-development-kit/.
    b. Sign in or register an account on the Infineon site.



c. Under the **Windows** section, select the **USB-Serial Windows Driver Installer** link (the driver will download as `CypressDriverInstaller_1.exe`).



d. Double-click **`CypressDriverInstaller_1.exe`**.

e. Once the driver installation process is complete, proceed to step 2.

2. Install the **EZ-USB FX3 board** on the computer:

   a. Place one of the plastic jumpers included with the kit over the **PMODE Jumper (J4)** pins on the board (this sets the jumper to the closed state and configures the board for firmware flashing refer to ***Flash the EZ-USB FX3 board's firmware on page 53***).

   b. Connect the **USB 3.0 cable** included with the kit to the **USB 3.0 port** on the board, and then connect the other end of the USB cable to the computer.

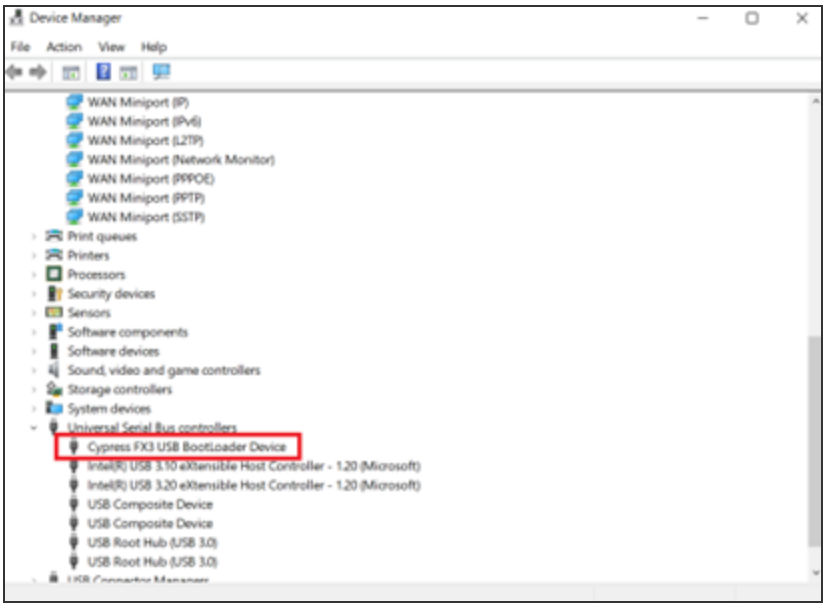
   > **NOTE:** For the locations of the **PMODE Jumper (J4)** and **USB 3.0 port** on the board, refer to the *EZ-USB FX3 SuperSpeed Explorer Kit Quick Start Guide* included with your kit.

3. Verify your EZ-USB FX3 board is properly detected by the computer:

   a. In **Windows**, open **Device Manager**.

   b. Expand **Universal Serial Bus controllers**.

   c. Verify **Cypress FX3 USB Bootloader Device** appears in the list.



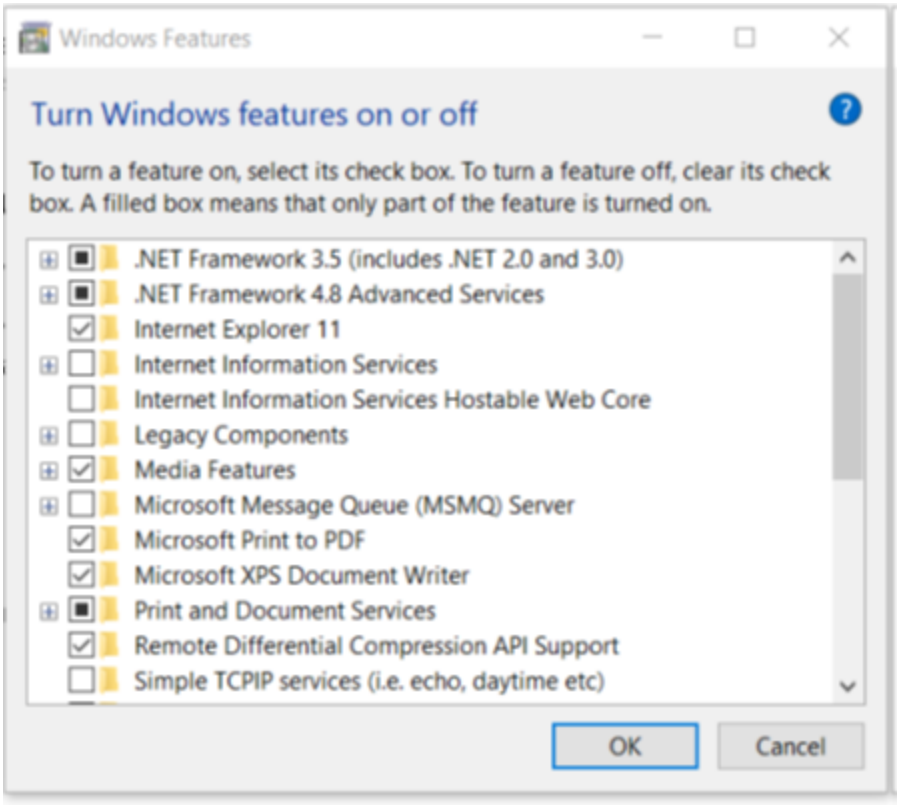
NOTE: If you do not see **Cypress FX3 USB Bootloader Device** in your list of USB devices, try reinstalling the Windows driver, or contact *Cypress technical support* from the *Support* section of the https://www.infineon.com/cms/en/product/evaluation-boards/cyusb3kit-003/ web page.

You can now proceed with flashing EZ-USB FX3 board's firmware.

Flash the EZ-USB FX3 board's firmware

After installing the EZ-USB FX3 board on the beSTORM computer, do the following to flash the board's firmware:

1. Verify .NET Framework 3.5 is selected:
    a. In the Windows search box, enter *Windows Features*.
    b. From the search results, select **Turn Windows features on or off**.
    c. In the dialog, select **.NET Framework 3.5 (includes .NET 2.0 and 3.0)**.

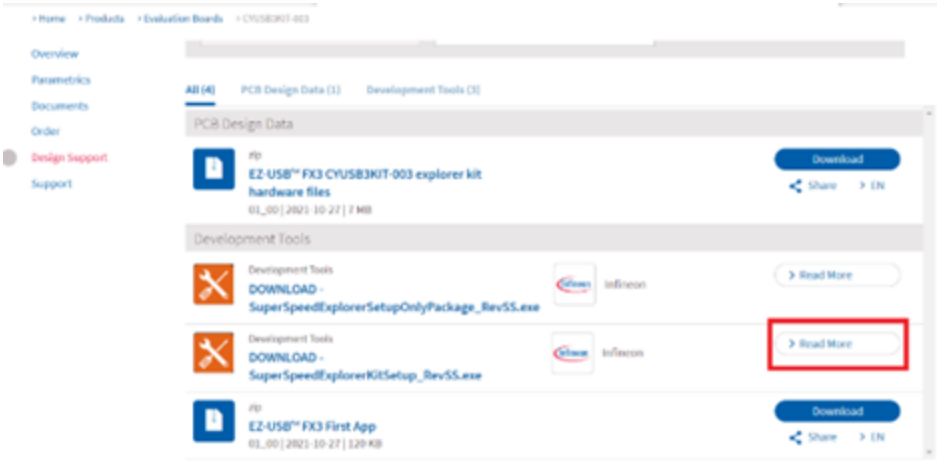


d. Select **OK**.

2. Download and unzip the firmware file:
   a. Go to the Fuzzing a USB device in beSTORM FAQ page at https://beyondsecurity.freshdesk.com/a/solutions/articles/44002334275.
   b. Download the **`UAC_USB_Peripheral_Module.zip`** file.
   c. Unzip the **`UAC_USB_Peripheral_Module.zip`** file to extract the `UAC_USB_Peripheral_Module` folder.

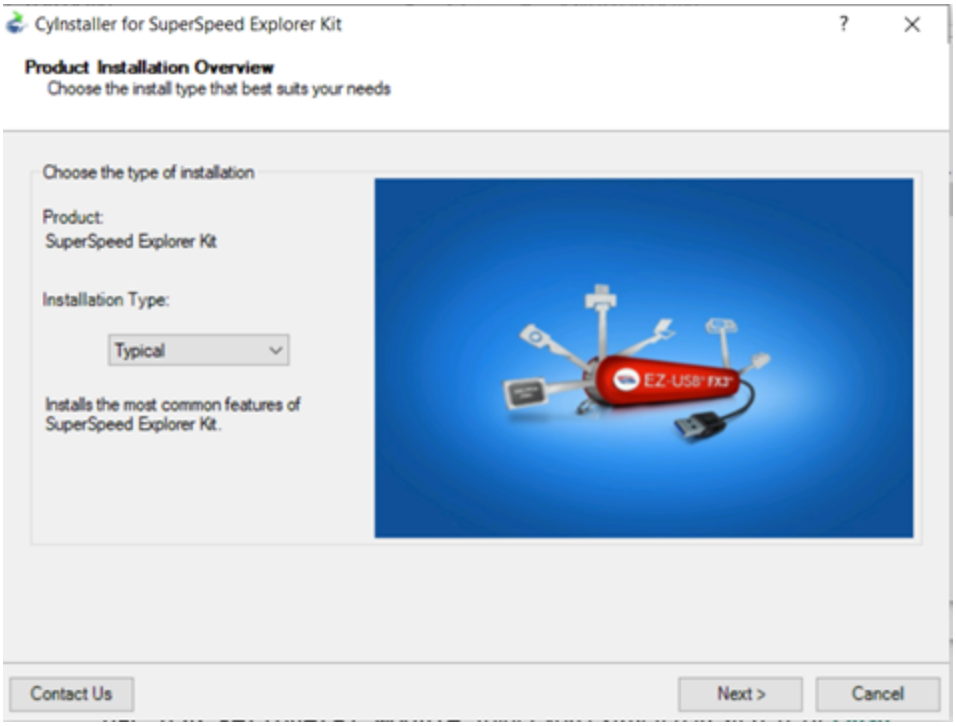
> **NOTE:** In addition to the firmware file, this folder also includes files you will use for fuzzing a USB device in ***How to fuzz a USB device with the EZ-USB FX3 board* on page 58**.

3. Download the flashing tool:
   a. Go to the EZ-USB FX3 SuperSpeed Explorer Kit product page https://www.infineon.com/cms/en/product/evaluation-boards/cyusb3kit-003/.
   b. Expand **Development Tools**, and then select **DOWNLOAD - SuperSpeedExplorerKitSetup_RevSS.exe**.



c. Double-click **`SuperSpeedExplorerKitSetup_RevSS.exe`**.

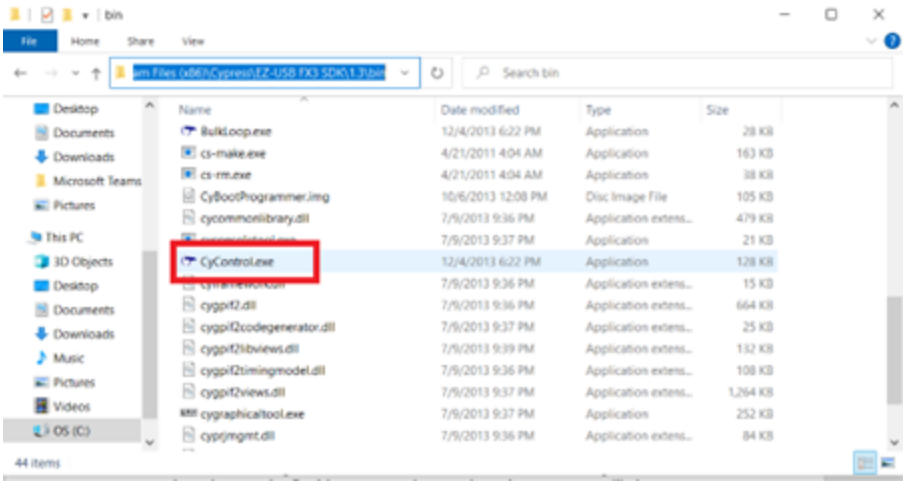
d. During the installation process, select **Typical** as the **Installation Type**.



e. Once the installation process is complete, proceed to step 3.

4. Flash the EZ-USB FX3 board:

   a. In the `C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\bin` folder, double-click **`CyControl.exe`**.

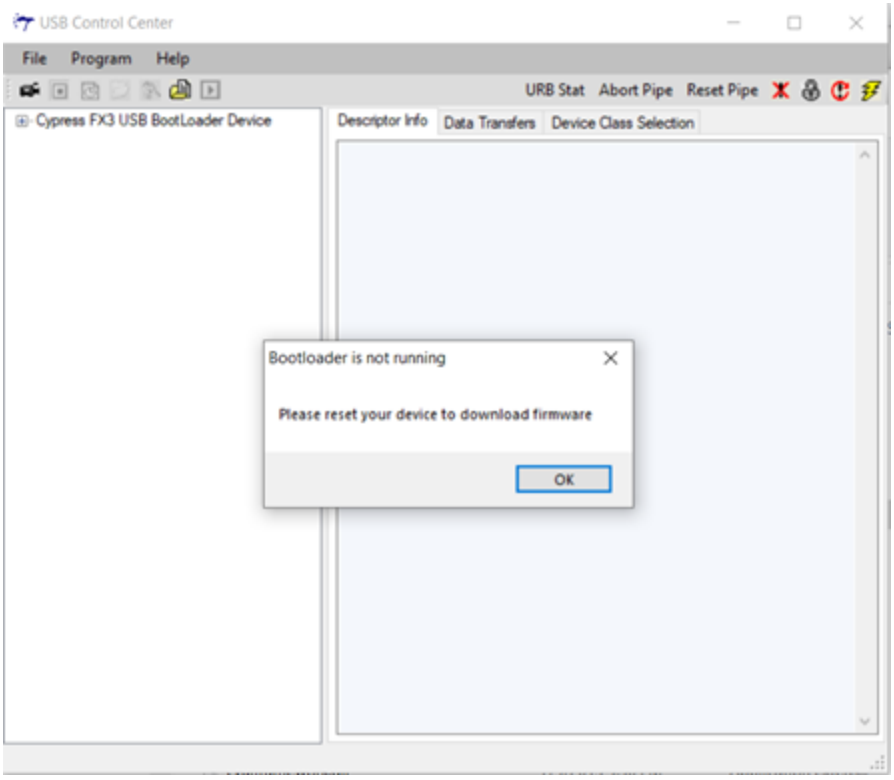


b. From the top menu of the **USB Control Center**, select **Program** > **FX3** > **I2C EEPROM**.

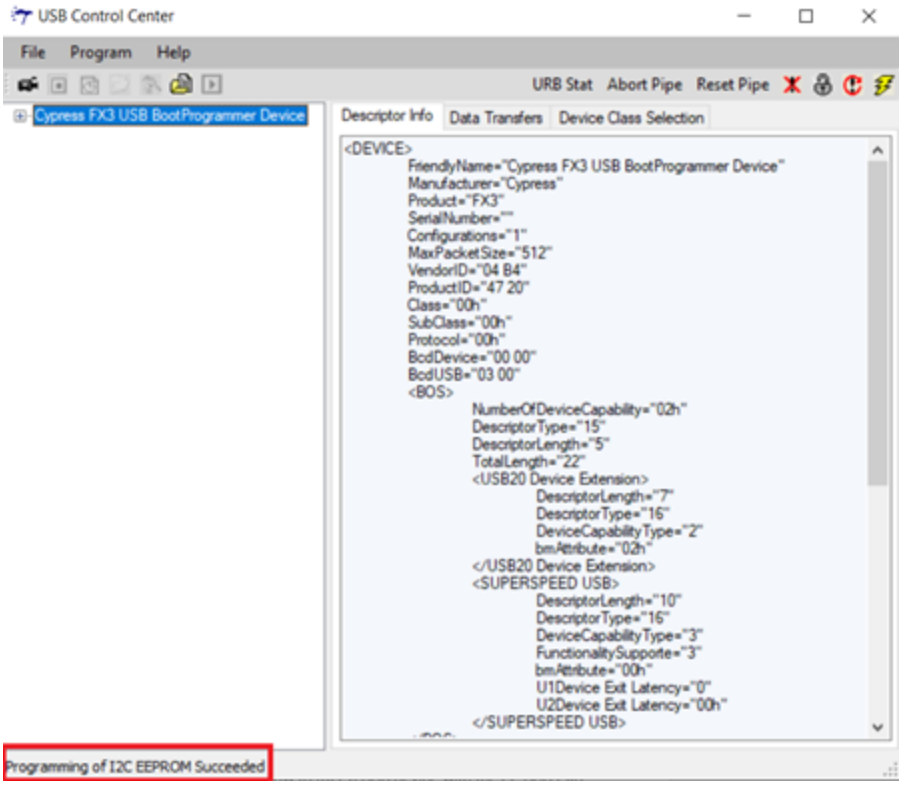
c. From the dialog that appears, open the **`UAC_USB_Peripheral_Module`** folder you extracted in step *2c* of ***Flash the EZ-USB FX3 board's firmware* on page 53**.

d. Select the **`best_uac_usbd.img`** firmware file, and then select **Open** to start flashing the firmware.

> **IMPORTANT:** If a dialog appears instructing you to *Please reset your device to download firmware*, do not select **OK**. Instead, press the **Reset Switch** on the **EZ-USB FX3 board**, and then retry step 4c. For the location of the **Reset Switch** on the board, refer to the EZ-USB FX3 SuperSpeed Explorer Kit Quick Start Guide included with your kit.



e. The bottom-left corner of USB Control Center will display *Programming of I2C EEPROM in Progress* while the USB Control Center is flashing the EZ-USB FX3 board. Once the flashing process is complete, the message will change to *Programming of I2C EEPROM Succeded*, indicating the flashing process is complete.



f. Disconnect the **USB 3.0 cable** from the **EZ-USB FX3 board**.

g. Remove the plastic jumper from the **PMODE Jumper (J4)** on EZ-USB FX3 board to set it back to the open state.

Your EZ-USB FX3 board is now ready to fuzz a USB device with beSTORM.

How to fuzz a USB device with the EZ-USB FX3 board

After installing the EZ-USB FX3 board on the beSTORM computer and then flashing its firmware, do the following to fuzz a USB device:

1. Connect the EZ-USB FX3 board to the beSTORM computer and the target USB device:
   a. Insert the **USB 3.0 cable** included with the EZ-USB FX3 SuperSpeed Explorer Kit into the **USB 3.0 port** on the board, and then connect the other end of the cable to the **target USB device** you want to fuzz.
   b. Insert the **USB 2.0 Micro USB cable** you supplied separately from the kit into the **USB 2.0 port** on the board, and then connect the other end of the cable to the **beSTORM computer**.

> **NOTE:** For the locations of the **USB 3.0 port** and **USB 2.0 port** on the board, refer to the *EZ-USB FX3 SuperSpeed Explorer Kit Quick Start Guide* included with your kit.

2. Verify your EZ-USB FX3 board and the target USB device is properly detected by the computer:
   a. In Windows, open **Device Manager**.
   b. Expand **Universal Serial Bus controllers**.
   c. Verify **USB-Serial (Dual Channel) Vendor 1** and **USB-Serial (Dual Channel) Vendor MFG** appear in the list of devices.

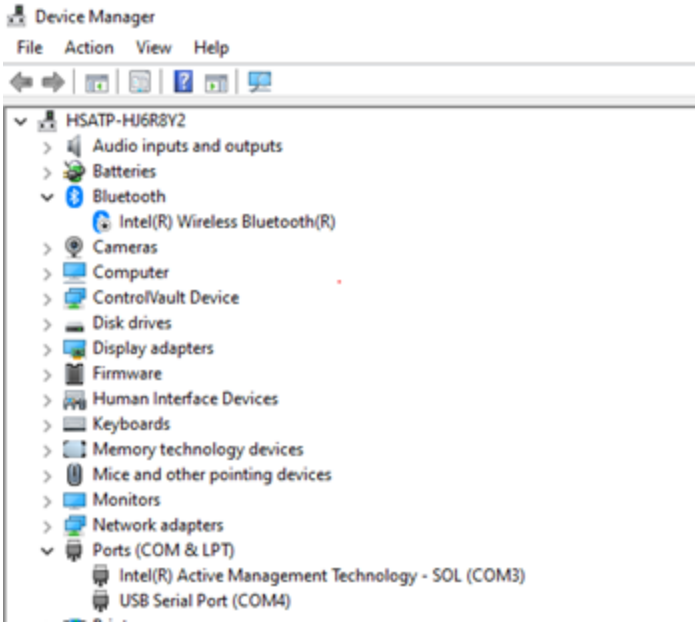
   > **NOTE:** If you do not see **USB-Serial (Dual Channel) Vendor 1** and/or **USB-Serial (Dual Channel) Vendor MFG** in your list of USB devices, contact Beyond Security Technical Support at support@beyondsecurity.com.

3. Prepare the project fuzzing files:
   a. On the beSTORM computer, create the following folder: `C:\TEMP`.
   b. Copy the **`best_uac_usbd.dll`**, **`bestorm.csv`**, and **`best_uac_usbd.xml`** files from the **`UAC_USB_Peripheral_Module`** folder (steps *2a-2c* of ***Flash the EZ-USB FX3 board's firmware* on page 53**) into the **`C:\TEMP` folder**.
4. Create a new beSTORM project and start fuzzing:
   a. Start the **beSTORM client**.
   b. Select **New Project**.
   c. On the **Welcome** page, do the following:
      i. Enter a name for this project in the **Project Name** box.
      ii. Proceed with the default values for the remaining settings.
   d. Select **Next**.
   e. On the **Basic Configuration** page, select **Import a Custom Module from a BSM file**, and then select **Import**.
   f. In the dialog, browse to the `C:\TEMP` folder, and then select the **`best_uac_usbd.xml`** file.
   g. Select **Open**.
   h. On the **Module Environment** page, set COM to the correct COM port for the board. To determine the COM port, do the following:

i. In Windows, open **Device Manager**.

ii. Expand **Ports (COM & LPT)**.

iii. Disconnect, and then reconnect the **USB 2.0 Micro USB cable** from the computer. Make note of the *(COM)* number. For example, USB Serial Port (COM4) (see image).



i. Select **Next**.

j. On the **Test Selection** page, leave the **Initialize**, **Fuzzing**, and **Deinitialize** check boxes selected and select **Next**.

k. On the **Extra Configuration** page, proceed with the default values for all settings and select **Next**.

l. On the **Complete beSTORM wizard** page, clear the **Auto-start beSTORM scan now** check box to disable this option.

m. Select **Finish**.

n. In the **Project Settings** pane of the main beSTORM client window, select **Settings**.

o. In the **Project Settings** pane of the **beSTORM Settings** dialog, review the following settings for your environment:

i. **LOG_PATH** - The default path and file name for the project's log file is `C:\TEMP\bestorm.log`. Optionally, you can change either by using alphanumeric characters and entering a different file name (must end with `.log`) and path to store the log file.

ii. **RECORD_PATH** - The default path and file name for the project's recovery data file is `c:\TEMP\best_record.dat`. Optionally, you can change either by using alphanumeric characters and entering a different file name

(must end with .dat) and path to store the recovery data file for the recovery tool in your environment. For more information, refer to the Recovery Tool chapter.

iii. Proceed with the default values for the remaining settings.

p. Select **Apply**.

q. From the left pane on the **beSTORM Settings** dialog, select **Monitor**.

r. Under **Monitor Settings**, clear the **Enable Batch Mode** checkbox to disable this option.

s. Select **Apply**.

t. Select **OK**.

u. In the **Project Settings** pane on the main beSTORM client window, select **Start**.

> **NOTE:** An error message will appear if a serial port or a file cannot be opened. If this occurs, check the settings for your environment again.

v. A dialog appears requesting you to *Please reset the device*. Do not select OK until *after* the next step.

w. Press the **Reset Switch** on the **EZ-USB FX3 board**.

> **NOTE:** For the location of the **Reset Switch** on the board, refer to the *EZ-USB FX3 SuperSpeed Explorer Kit Quick Start Guide* included with your kit.

x. On the *Please reset the device* dialog, select **OK** to start fuzzing the target USB device.

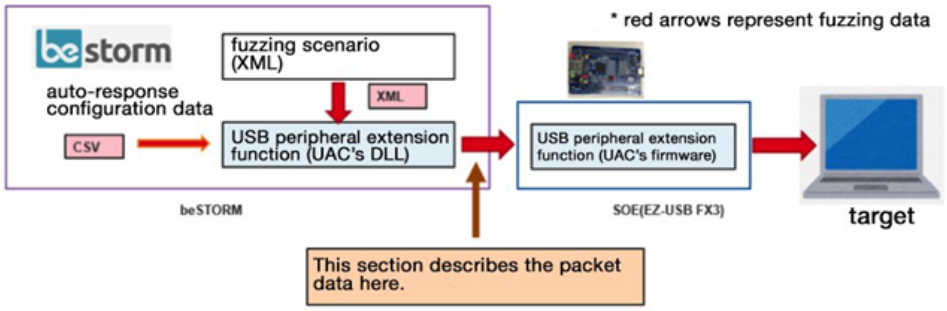
When the ScaleType is standard (Base2+/-1), it sends ~1200 fuzz data. Fuzzing may take 3 to 4 hours, depending on the environment.

When the test is run, a log file and a record file for the recovery tool are created separately from the standard beSTORM log file.

> **NOTE:** The fuzzing process is slow and on start it might take a minute or two to start showing any signs progress. The window might appear as if it is not responding. This is not a sign of the program failing. Wait until you hear the USB disconnect/reconnect chime before attempting to interact with the Monitor's controls.

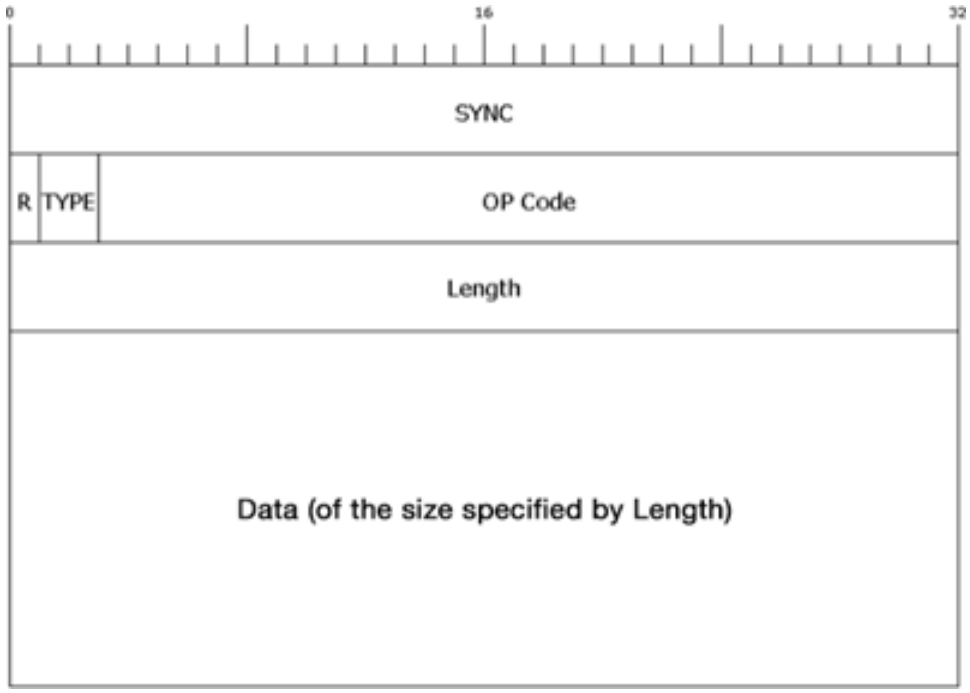
Communication Protocol for the Serial Offload Engine (SOE)

This chapter describes the specifications of the DLL of the UAC USB Peripheral Module and the SOE packet, which is the communication data between the DLL of the UAC USB Peripheral Module and the SOE (EZ-USB FX3). As described below, the log file will record this communication exchange.



Overview of SOE packet

An SOE packet is composed of 12 bytes of header data, followed by payload data if necessary (see below):



- The header consists of 12 bytes (up to the Length in the figure above). This is constant for all packets.
- The header is sometimes followed by the data. Its length is specified in Length.

Data in headers

The definition of each header is shown below. All of these data are in Little Endian:

Contents in header

Element	Size	Description
SYNC	32 bits	Set to `0xaaaaaaaa`, which indicates that it is the first data.
R	1 bit	Reserved (not currently in use).
TYPE	3 bit	Indicates the type of packet • 100 used by the SOE to notify the DLL that an event has occurred . • 010 used for a response packet from the SOE to the DLL. • 001 used for a request packet from the DLL to the SOE.
OP Code	28 bits	Contains a code indicating the operation of the packet.
Length	32 bits	If the header is followed by data, the length of the data is specified here. If there is no data followed, this element should be zero.

OP codes for request/response types

OP codes used in a request from the DLL and its response are defined as follows:

OP Codes for event type

OP Code	Value	Description
UAC_SOE_OP_GENERAL_PING	0x0001	Request to send back the sent data as it is (for check).
UAC_SOE_OP_START_USB	0x0004	Activate the USB peripheral function.
UAC_SOE_OP_SET_CONN_STATE	0x0006	Connect and Disconnect USB.
UAC_SOE_OP_STALL_PIPE	0x0007	Set the specified endpoint to STALL state.

OP Code	Value	Description
UAC_SOE_OP_RESET_PIPE	0x0009	Release the STALL state of the specified endpoint.
UAC_SOR_OP_SET_DESC	0x000a	Set Descriptor.
UAC_SOE_OP_CTRL_ RESPONSE	0x000f	Send control transfer response data.
UAC_SOE_OP_IN_SEND	0x1002	Send IN transfer data.

OP codes for event type

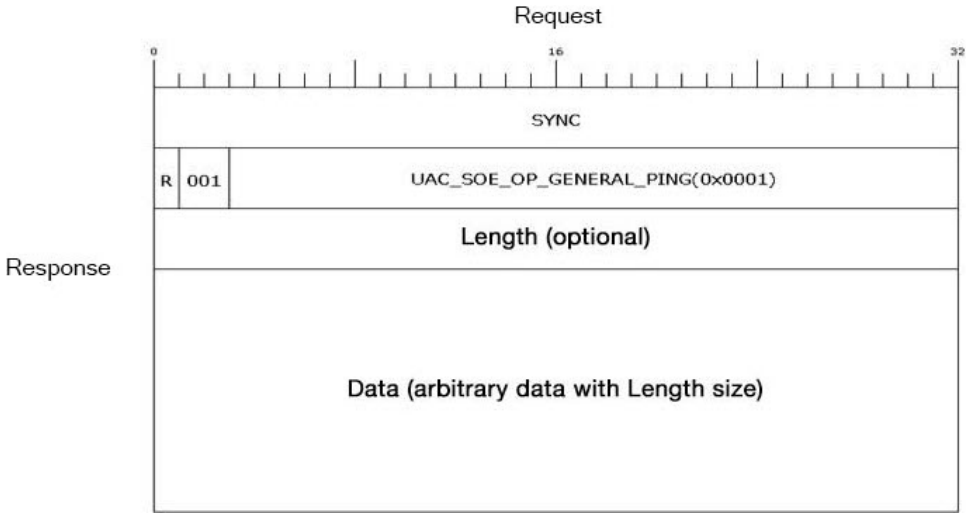
OP codes that notifies the DLL of information from the SOE is defined as follows:

OP Code	Value	Description
UAC_SOE_OP_START_SOE	0x0005	Notification that communication with the SOE is now available.
UAC_SOE_OP_GENERAL_ DBGMSG	0x00ff	Notification of debug messages (strings): used for debug.
UAC_SOE_OP_SETCONFIG	0x0003	A SET_CONFIGURATION is issued by the target.
UAC_SOE_OP_CLEAR_ FEATURE	0x0008	A CLEAR_FEATURE is issued by the target.
UAC_SOE_OP_BUS_RESET	0x000b	A USB bus reset is issued by the target.
UAC_SOE_OP_RECV_ ERROR	0x000c	Anomaly occurs during the communication process.
UAC_SOE_OP_CTRL_REQ_ RECV	0x000e	A control transfer is received from the target.
UAC_SOE_OP_OUT_RECV	0x1001	Data is received from the target's OUT endpoint.
UAC_SOE_OP_IN_ COMPLETE	0x1003	IN transfer with the target is completed.

Specification of OP codes

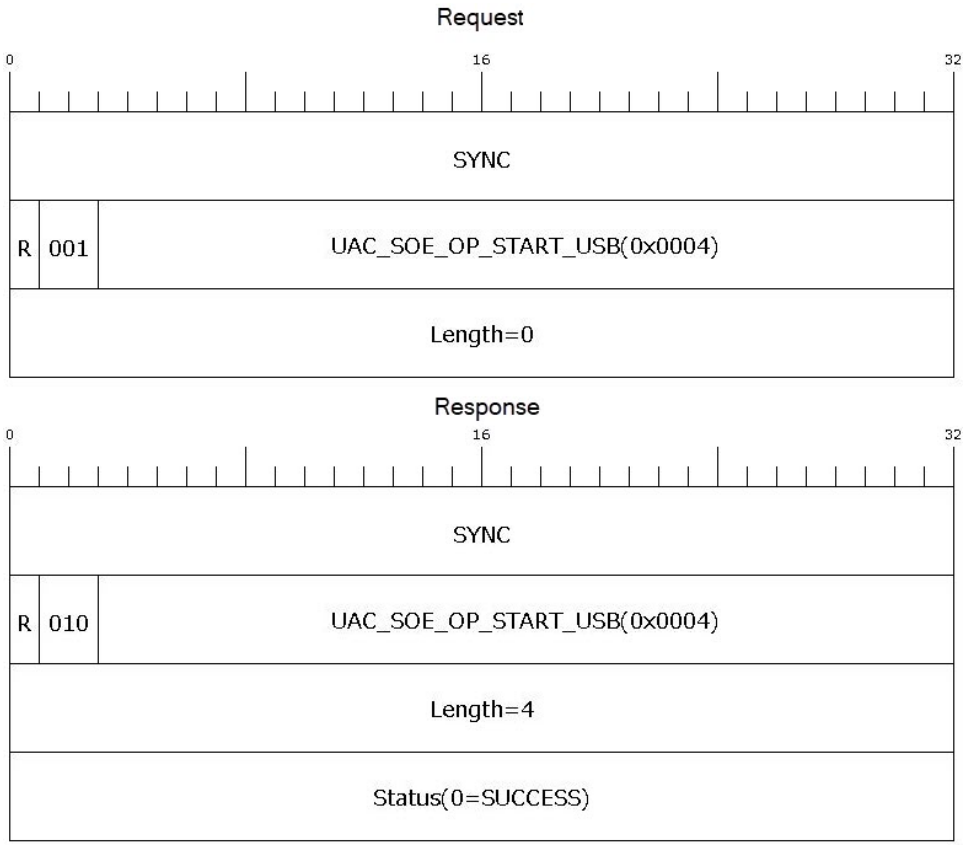
UAC_SOE_OP_GENERAL_PING

`UAC_SOE_OP_GENERAL_PING` is used to check the communication status during debugging, etc. When data is sent to the SOE, the same data is returned. This communication is mainly used for debugging purposes.



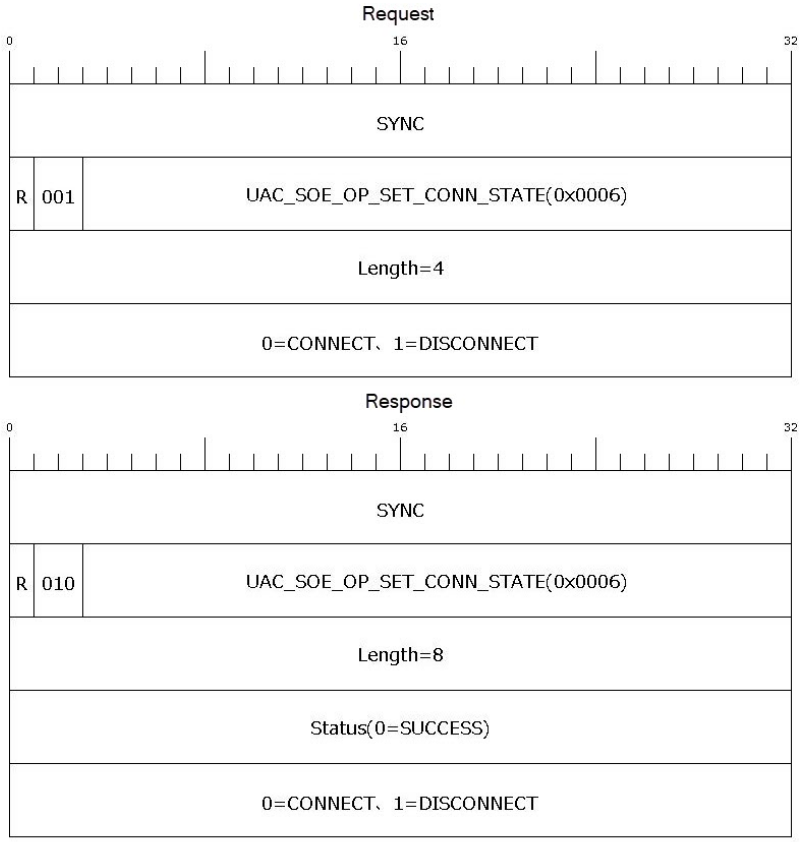
UAC_SOE_OP_START_USB

`UAC_SOE_OP_START_USB` instructs the SOE to start as a USB peripheral, and when the SOE receives this OP code, it initializes the peripheral function and creates a USB connection with the target.



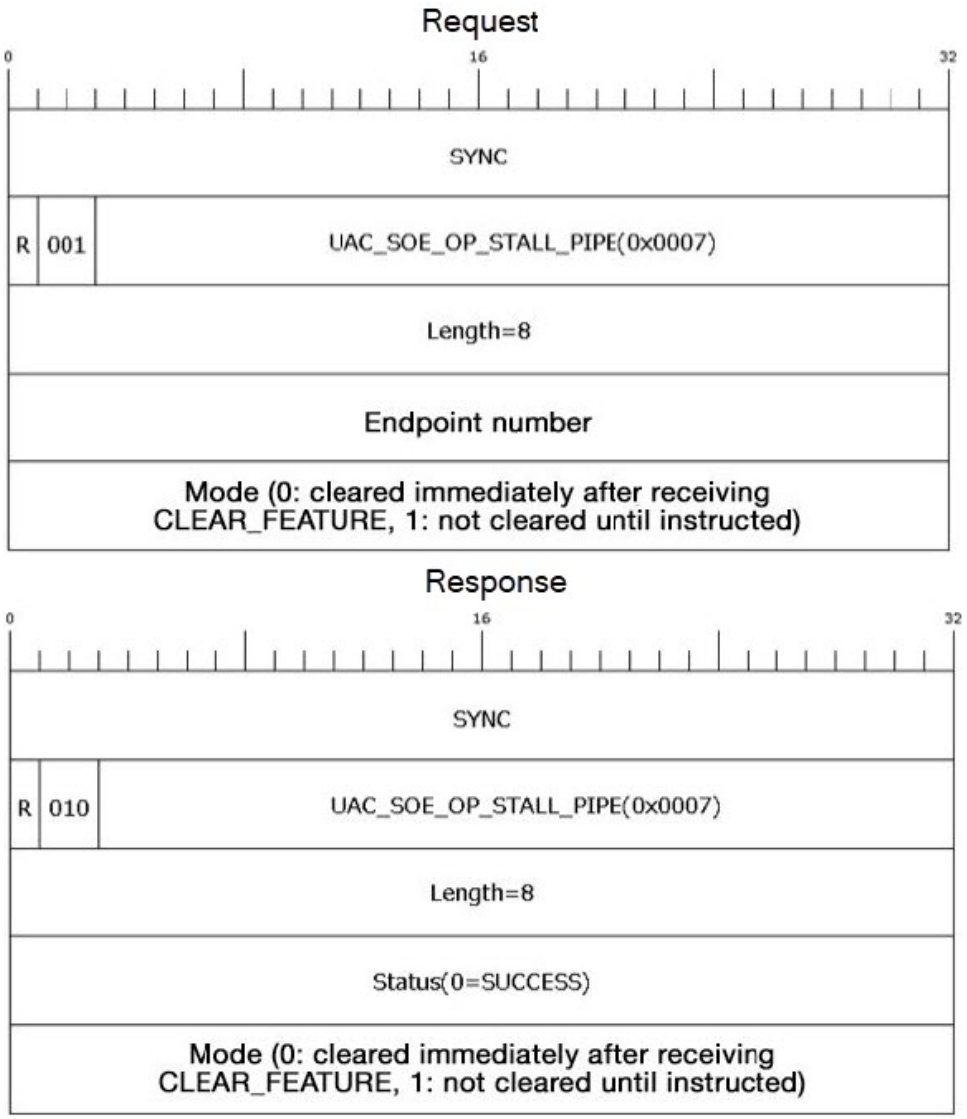
UAC_SOE_OP_SET_CONN_STATE

UAC_SOE_OP_SET_CONN_STATE instructs the USB connection/disconnection.



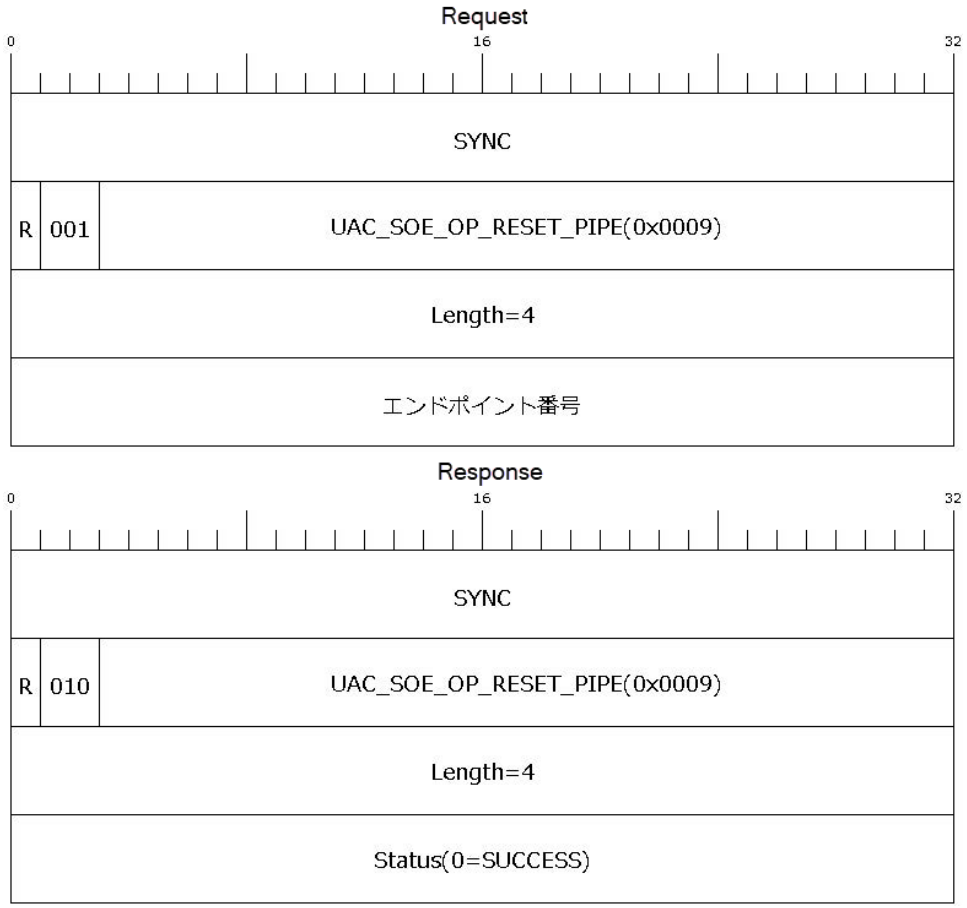
UAC_SOE_OP_STALL_PIPE

UAC_SOE_OP_STALL_PIPE specifies the STALL of the specified endpoint. If the mode is set to 0, the STALL state is immediately released by CLEAR_FEATURE from the target; if the mode is set to 1, the STALL state will not be released until the instruction in UAC_SOE_OP_RESET_PIPE.



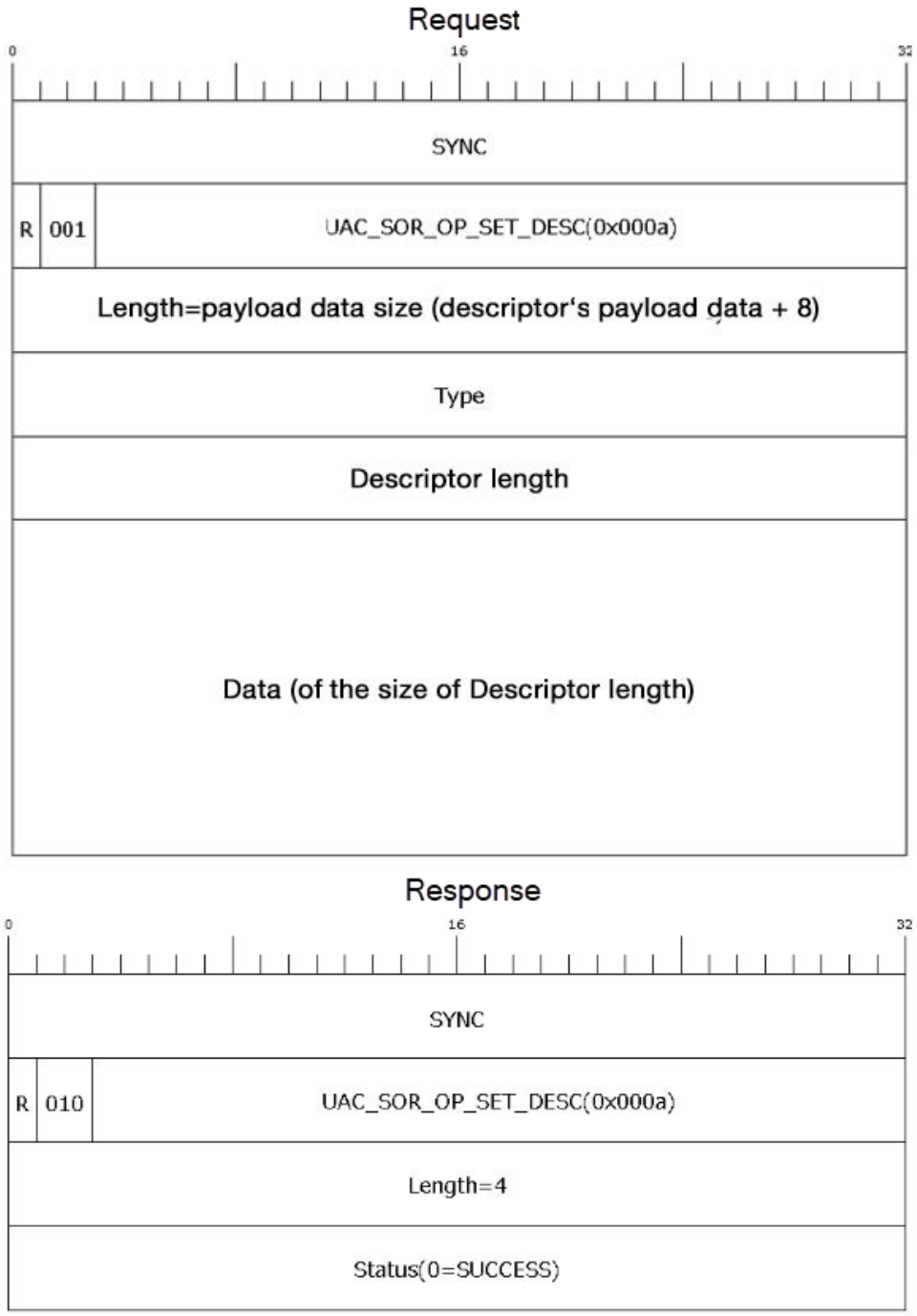
UAC_SOE_OP_RESET_PIPE

`UAC_SOE_OP_RESET_PIPE` releases the STALL state of the specified endpoint.



UAC_SOR_OP_SET_DESC

`UAC_SOR_OP_SET_DESC` instructs the SOE to set the descriptor.



The following descriptors can be specified in Type:

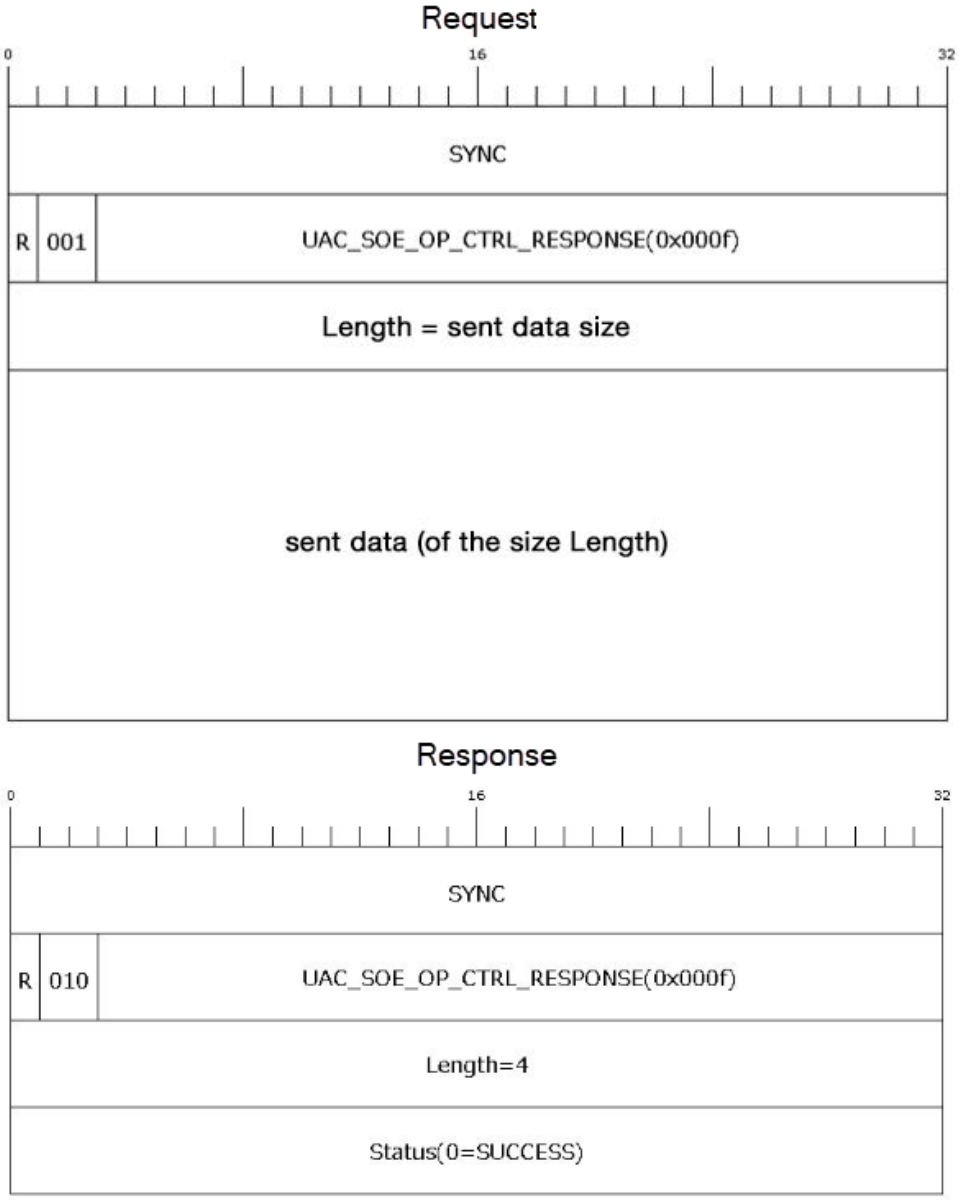
List of descriptors

Value	Description
1	USB 2.0 compliant device descriptor.

Value	Description
2	USB 3.0 compliant device descriptor.
3	Device qualifier descriptor.
4	Configuration descriptor at full speed (including attached descriptors).
5	Configuration descriptor at high speed (including attached descriptors).
6	Configuration descriptor for super speed (including attached descriptors).
7	String descriptor (language ID).
8	String descriptor (manufacturer).
9	String descriptor (product).

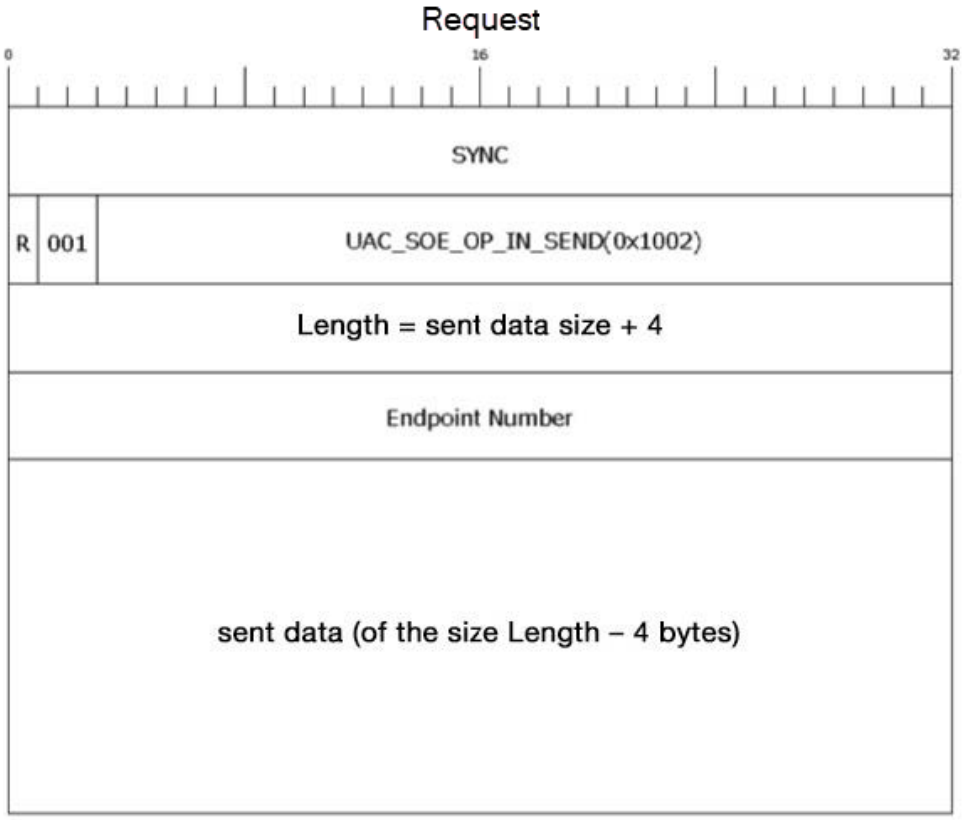
UAC_SOE_OP_CTRL_RESPONSE

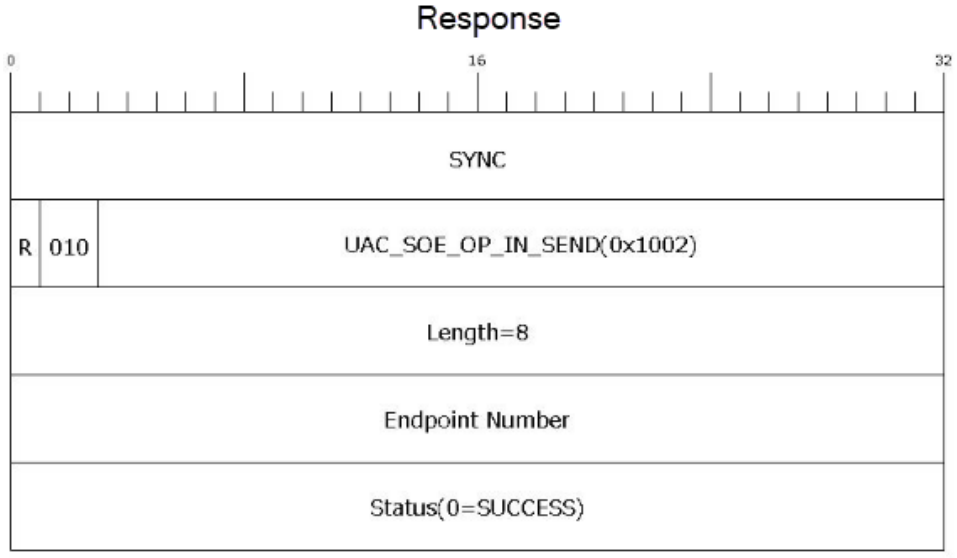
`UAC_SOE_OP_CTRL_RESPONSE` instructs the SOE to send the control transfer response data (valid only for vendor requests or class requests).



UAC_SOE_OP_IN_SEND

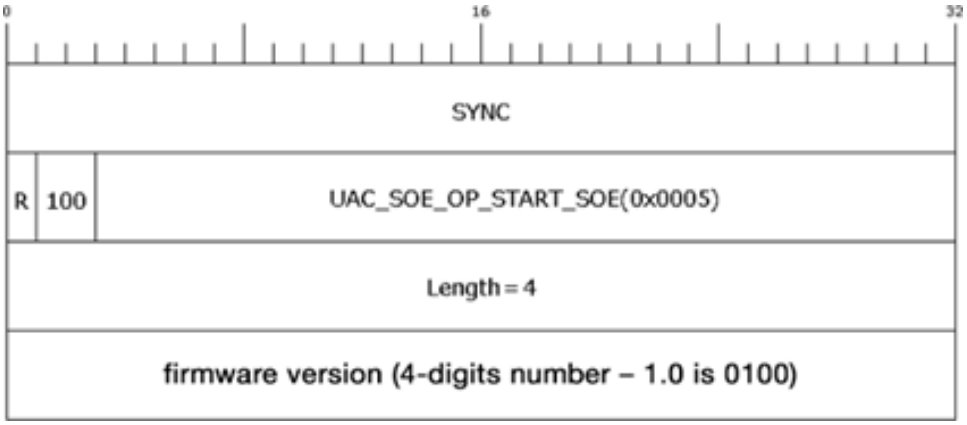
`UAC_SOE_OP_IN_SEND` instructs the SOE to send data to the IN endpoint.





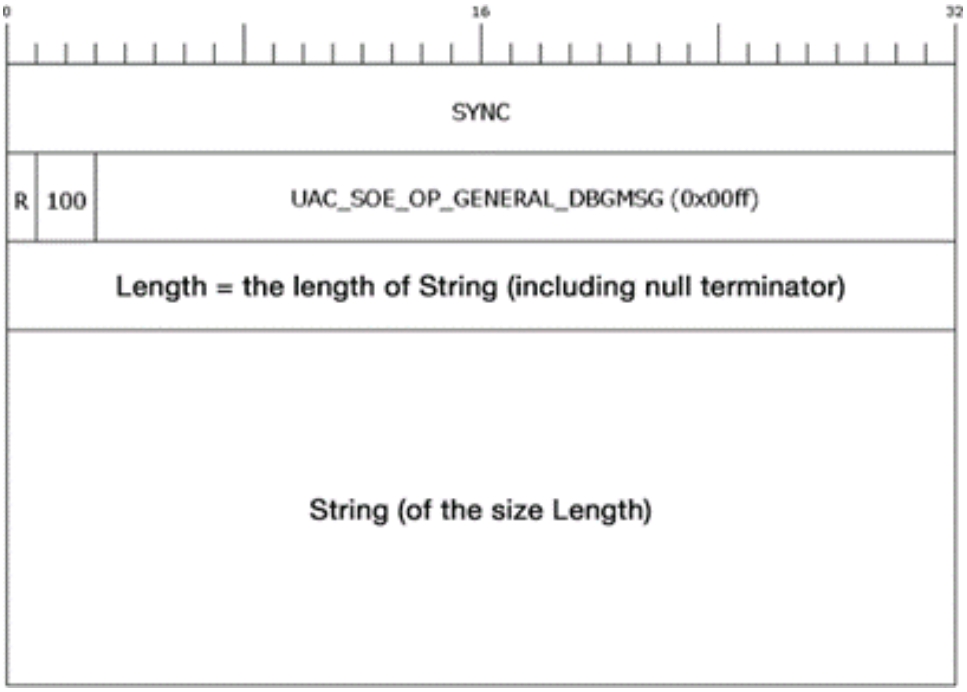
UAC_SOE_OP_START_SOE

Once the SOE's firmware has been booted and initialized and serial communication is available, the SOE issues a `UAC_SOE_OP_START_SOE` event to the DLL. The firmware version information will be provided here.



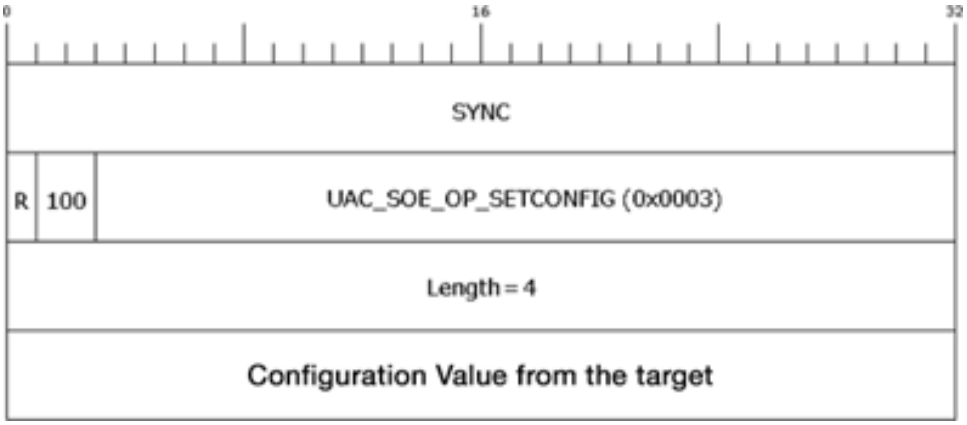
UAC_SOE_OP_GENERAL_DBGMSG

This event is used for debugging, and the SOE can put any string on the event packet of `UAC_SOE_OP_GENERAL_DBGMSG` and notify the DLL. It's not supposed to be used for normal fuzzing.



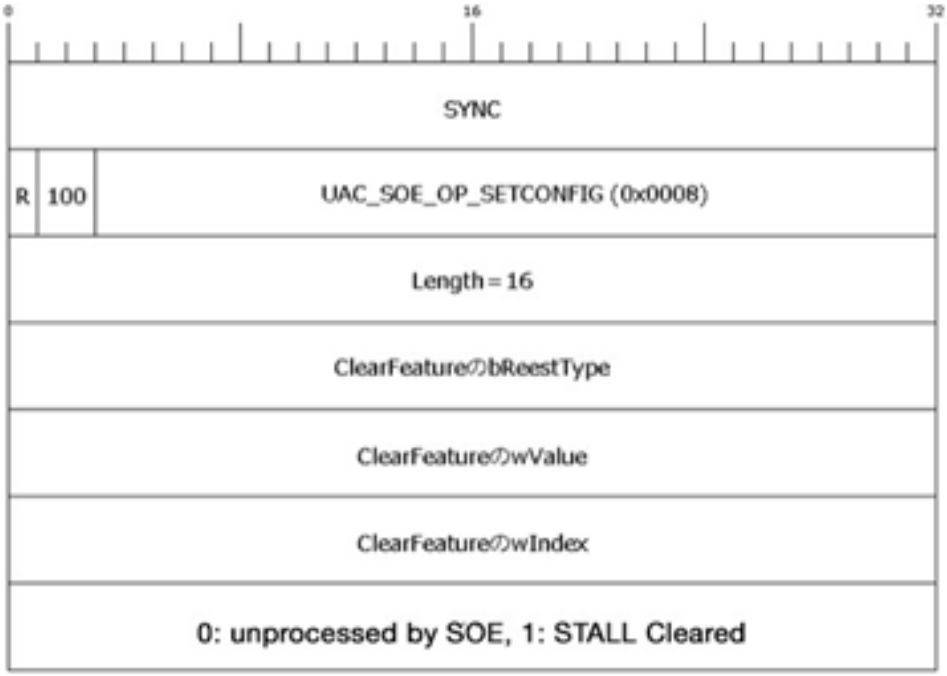
UAC_SOE_OP_SETCONFIG

Upon receiving a `SET_CONFIGURATION` from the target, the SOE issues a `UAC_SOE_OP_SETCONFIG` event to notify the DLL.



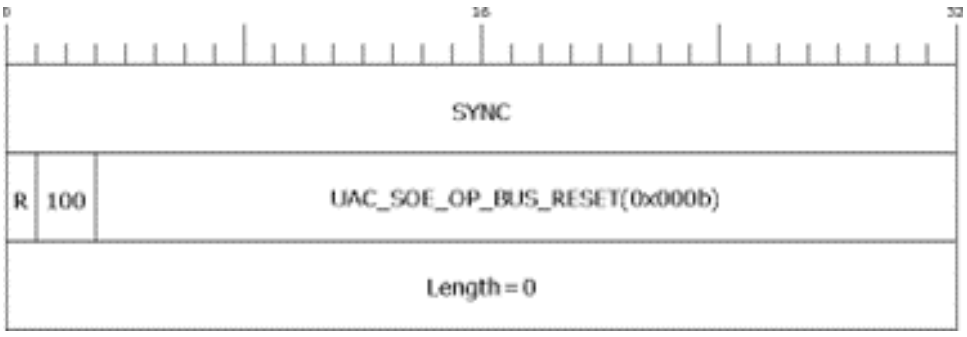
UAC_SOE_OP_CLEAR_FEATURE

Upon receiving a `CLEAR_FEATURE` from the target, the SOE issues a `UAC_SOE_OP_CLEAR_FEATURE` event to notify the DLL.



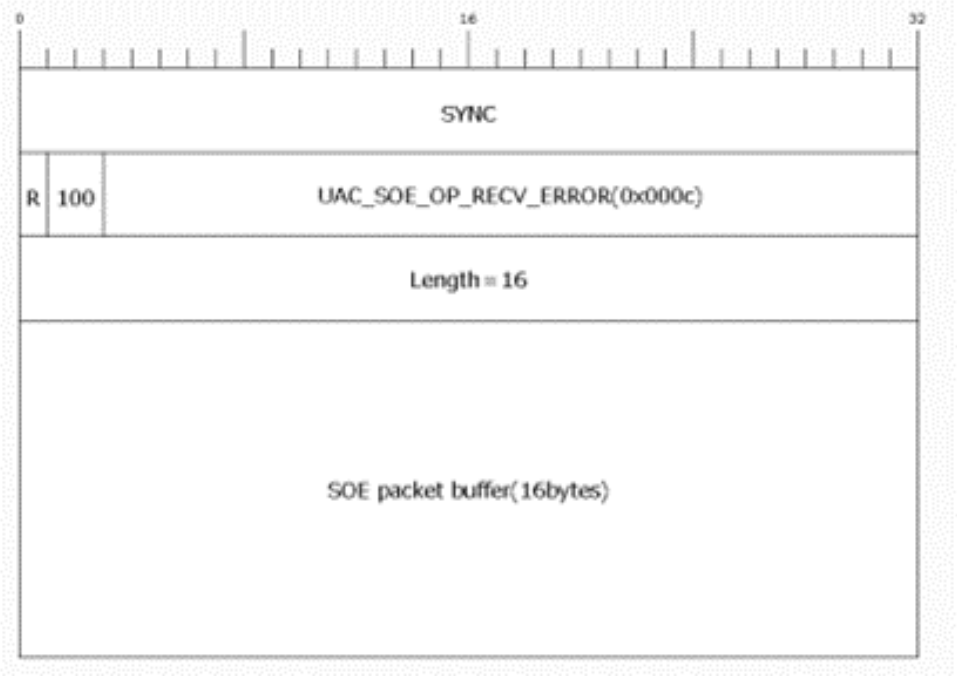
UAC_SOE_OP_BUS_RESET

When the target issues a USB bus reset (USB device reinitialization instruction) to the SOE, the SOE issues a `UAC_SOE_OP_BUS_RESET` event to notify the DLL.



UAC_SOE_OP_RECV_ERROR

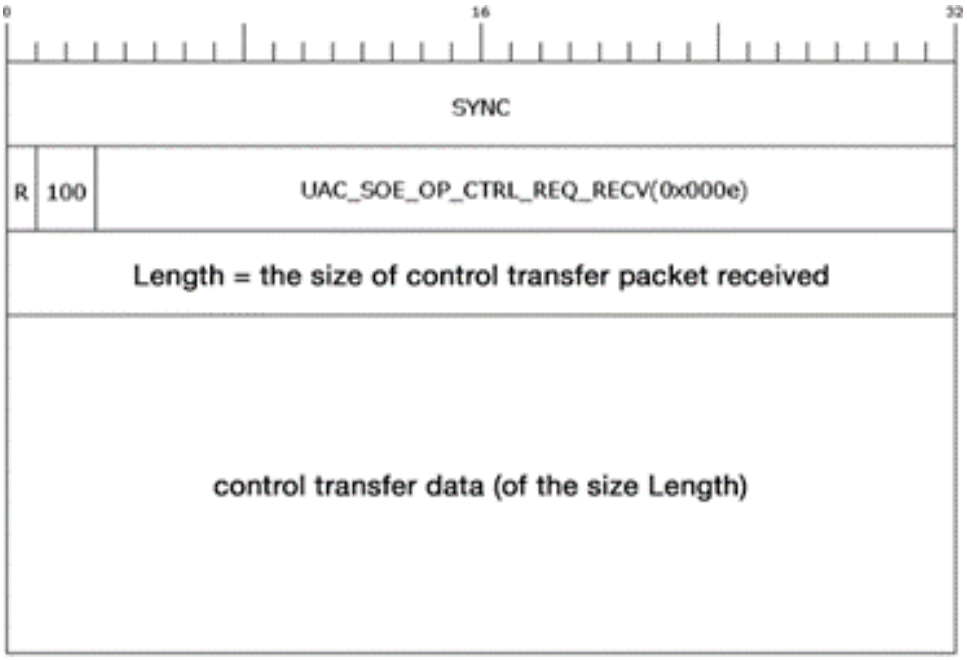
If any abnormality is detected by serial communication, the SOE issues a `UAC_SOE_OP_RECV_ERROR` event to notify DLL.



The EZ-USB FX3 board's USB serial communication can result in lost serial data if there is a conflict between the USB communication with the target or the connect/disconnect process and the serial communication. This event type has been implemented to mitigate this issue. The data contains the received buffer (the first 16 bytes) of the serial data, You may see the buffer here.

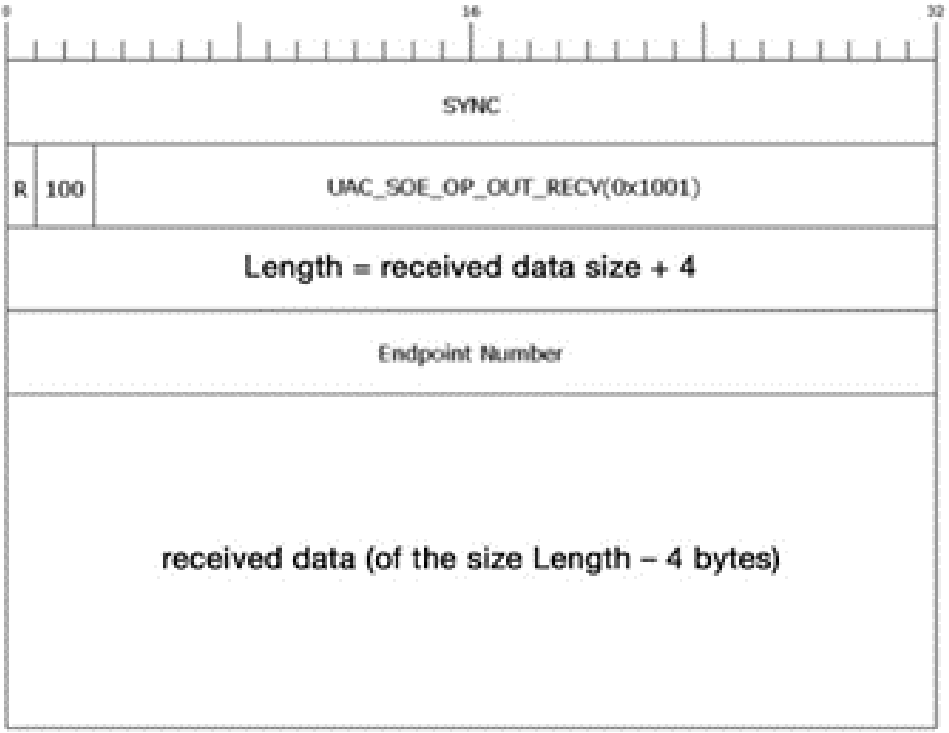
UAC_SOE_OP_CTRL_REQ_RECV

Upon receiving a control transfer (class request or vendor request) from the target, the SOE issues a `UAC_SOE_OP_CTRL_REQ_RECV` event to notify the DLL.



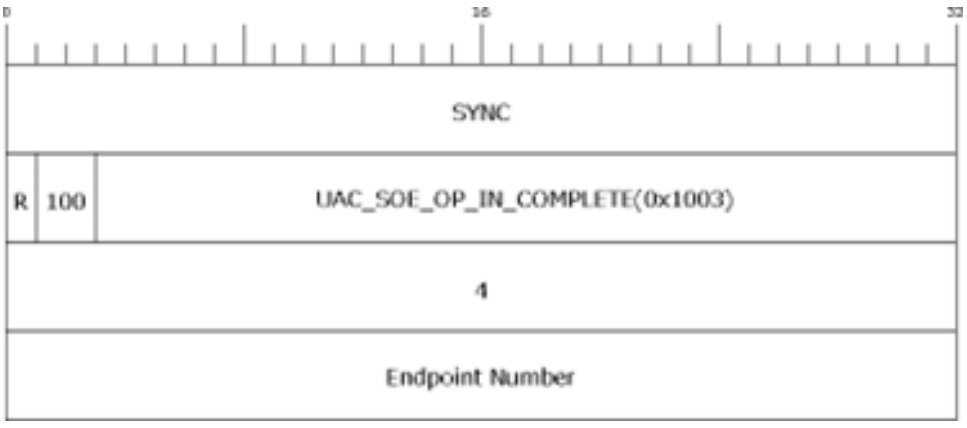
UAC_SOE_OP_OUT_RECV

Upon receiving the data of the OUT endpoint from the target, the SOE issues a `UAC_SOE_OP_CTRL_REQ_RECV` event to notify the DLL with the data.



UAC_SOE_OP_IN_COMPLETE

Upon detecting that the ACK is returned from the target and the communication is completed after the IN transfer, the SOE issues an `UAC_SOE_OP_IN_COMPLETE` event to the DLL.

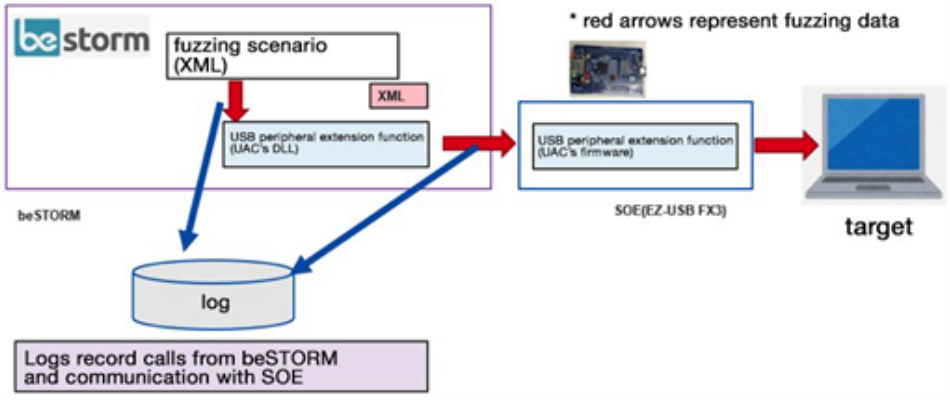


Logging

Overview

- The USB Peripheral Module creates its own log file when fuzzing, which is separate from beSTORM.
- The log file is output to the specified path of LOG_FILE (LOG_PATH from beSTORM GUI setting page) described in the *Begin USB fuzzing protocol* (`UAC_usb_start`) section of the Overview of UAC USB Peripheral Module.
- The log file consists of the following:
  - A message stating that each DLL function described in *How to use functions* section of the Overview of UAC USB Peripheral Module chapter is called.
  - The contents of the data sent and received for the SOE packets described in the Communication Protocol for the Serial Offload Engine (SOE) chapter.
  - Statistical information about fuzzing and other tests, provided at the end of fuzzing.
- At the same time, it outputs the message when each DLL function is called to Windows debug message. Therefore, you can check the execution status in real-time with tools such as DebugView.

As shown in the figure below, the log shows which function of the DLL was called and the content of the SOE packet. This enables you to check the communication status with the target from the log to some extent even without a USB analyzer.

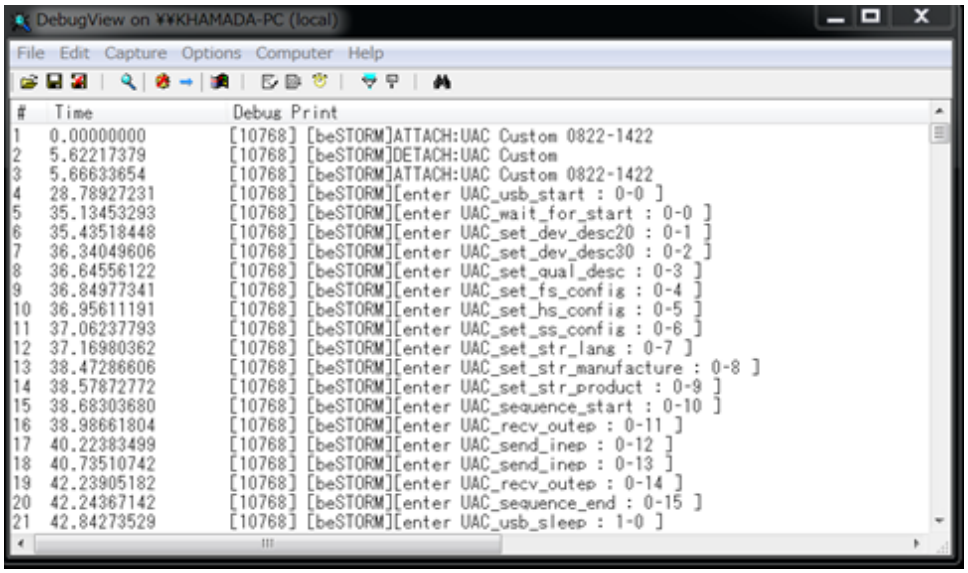


Debug message

The USB Peripheral Module outputs information about each DLL function called from beSTORM to both the log file and Windows debug messages. This enables you to use a tool like DebugView that allows you to check debug messages in real-time to see what's going on with your fuzzing at hand.

DebugView is available from the following website: https://docs.microsoft.com/en-us/sysinternals/downloads/debugview

- Unzip the downloaded ZIP file to run `Dbgview.exe`.
- Double-click `Dbgview.exe` to start DebugView as shown below.
- When performing fuzzing with beSTORM with the DebugView open, messages are displayed as shown below, so that you can grasp the execution in real-time.



Content of Debug Message

Windows debug messages displayed in DebugView are formatted with the pre-fixed [beSTORM], which makes it easy to identify which messages are from the USB Peripheral Module.

The following is an excerpt of a message that are observed on DbgView:

```
00000049 66.22557068 [10768] [beSTORM][enter UAC_usb_sleep : 5-0 ]
00000050 68.22835541 [10768] [beSTORM][enter UAC_sequence_start : 5-
1 ]
00000051 68.33224487 [10768] [beSTORM][enter UAC_recv_outep : 5-2 ]
00000052 69.81584930 [10768] [beSTORM][enter UAC_send_inep : 5-3 ]
00000053 70.32501984 [10768] [beSTORM][enter UAC_send_inep : 5-4 ]
00000054 70.43362427 [10768] [beSTORM][enter UAC_recv_outep : 5-5 ]
00000055 70.43775177 [10768] [beSTORM][enter UAC_sequence_end : 5-6
```

```
]
00000056 71.13491058 [10768] [beSTORM][enter UAC_usb_sleep : 6-0 ]
00000057 73.13673401 [10768] [beSTORM][enter UAC_sequence_start : 6-
1 ]
00000058 73.34024811 [10768] [beSTORM][enter UAC_recv_outep : 6-2 ]
00000059 74.82383728 [10768] [beSTORM][enter UAC_send_inep : 6-3 ]
00000060 75.43627930 [10768] [beSTORM][enter UAC_send_inep : 6-4 ]
00000061 75.54264832 [10768] [beSTORM][enter UAC_recv_outep : 6-5 ]
00000062 76.28384399 [10768] [beSTORM][enter UAC_sequence_end : 6-6
]
```

The basic format of DbgView is as follows. Anything other than the "message payload" is prepared by DbgView:

```
Message number Timestamp [Process number] Message payload
```

- The message number is assigned to a sequential number by DbgView.
- The timestamps is used to see the time. The unit is a second.
- The process number of the program is also displayed.

The format of the message payload by the UAC USB Peripheral Module is as follows:

```
[beSTORM][enter DLL function name:number X - number Y]
```

- [beSTORM] is a prefix, to determine if the message is sent from beSTORM or not.
- The function name is displayed on order to see which function in the DLL is currently being called.
- The number X indicates the number of fuzzing data being processed.
  - The number 10000 represents the processing of the Initialize sequence.
  - The number 20000s represents the processing of a fuzzing sequence.
  - The number 30000 represents the processing of the deinitialize sequence.
    - The above numbers are counted up every time a `UAC_sequence_end` is processed. This enables you to see which fuzzing data is being processed.
- The number Y is counted up every time a DLL function is called. This will be reset when processing a `UAC_sequence_end`. This enables you to see which function is being called in the sequence.

Description of the log file

The log file contains the following information:

- Information about called DLL functions as described in ***Debug message* on page 78**.
- Contents of SOE packets described in the Communication Protocol for the Serial Offload Engine (SOE) chapter.
- Statistics about fuzzing and other tests (at the end of the log).
- Other supplementary messages from DLLs.

The following is an excerpt from the actual log as an example:

```
[2019/08/22 20:51:12.0028][beSTORM][enter UAC_usb_sleep : 259-0 ]
[2019/08/22 20:51:14.0029][beSTORM][enter UAC_sequence_start : 259-1
]
[2019/08/22 20:51:14.0029]UAC_USBD_SequenceStart
[2019/08/22 20:51:14.0029]#--->UAC_SOE_OP_SET_CONN_STATE len 4
[2019/08/22 20:51:14.0029] 0x01 0x00 0x00 0x00
[2019/08/22 20:51:14.0029] * CONNECT
[2019/08/22 20:51:14.0136]#<--UAC_SOE_OP_SET_CONN_STATE len=8
[2019/08/22 20:51:14.0136] 00 00 00 00 01 00 00 00
[2019/08/22 20:51:14.0136] * SUCCESS
[2019/08/22 20:51:14.0136] * CONNECT
[2019/08/22 20:51:14.0137][beSTORM][enter UAC_recv_outep : 259-2 ]
[2019/08/22 20:51:14.0866]#<--UAC_SOE_OP_BUS_RESET len=0
[2019/08/22 20:51:15.0114]#<--UAC_SOE_OP_SETCONFIG len=4
[2019/08/22 20:51:15.0114] 00 00 00 00
[2019/08/22 20:51:15.0114] * ConfigValue:0x0
[2019/08/22 20:51:15.0356]#<--UAC_SOE_OP_CTRL_REQ_RECV len=8
[2019/08/22 20:51:15.0356] a1 fe 00 00 00 00 01 00
[2019/08/22 20:51:15.0356]#--->UAC_SOE_OP_CTRL_RESPONSE len 1
[2019/08/22 20:51:15.0356] 0x01
[2019/08/22 20:51:15.0372]#<--UAC_SOE_OP_CTRL_RESPONSE len=4
[2019/08/22 20:51:15.0372] 00 00 00 00
[2019/08/22 20:51:15.0372] * SUCCESS
[2019/08/22 20:51:15.0376]#<--UAC_SOE_OP_OUT_RECV len=35
[2019/08/22 20:51:15.0376]  02 00 00 00 55 53 42 43 40 4b de 5b 24
00 00 00 80 00 06 12 00 00 00 24 00 00 00 00 00 00 00 0000 00 00
[2019/08/22 20:51:15.0377] * Endpoint:0x02
[2019/08/22 20:51:15.0377] * length=31
[2019/08/22 20:51:15.0378][beSTORM][enter UAC_send_inep : 259-3 ]
[2019/08/22 20:51:15.0378]#--->UAC_SOE_OP_IN_SEND len 40
[2019/08/22 20:51:15.0378]  0x81 0x00 0x00 0x00 0x00 0x80 0x06 0x02
0x3b 0x00 0x00 0x00 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x61
0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e
0x6f 0x70 0x30 0x31 0x32 0x33
[2019/08/22 20:51:15.0378] * Endpoint:0x81
[2019/08/22 20:51:15.0379] * DataLength:36
[2019/08/22 20:51:15.0484]#<--UAC_SOE_OP_IN_SEND len=8
[2019/08/22 20:51:15.0484] 81 00 00 00 00 00 00 00
[2019/08/22 20:51:15.0484] * Endpoint:0x81
```

```
[2019/08/22 20:51:15.0484] * SUCCESS
[2019/08/22 20:51:15.0486]#<--UAC_SOE_OP_IN_COMPLETE len=4
[2019/08/22 20:51:15.0486] 81 00 00 00
[2019/08/22 20:51:15.0486] * Endpoint:0x81
```

The format of the log is as follows:

```
[Timestamp (date, time)] Message
```

Information about DLL function calls

As you can see in the above example, the information of the called DLL function is written in the log as well as the Windows debug message. The contents are as explained in ***Content of Debug Message* on page 79**.

Information about SOE packets

The USB Peripheral Module writes the contents of every packet sent and received by the SOE to a log file. A line beginning with #--> means that the information of SOE packets is sent from the DLL to the SOE. A line beginning with #<-- means that the information of SOE packets is sent from the SOE to the DLL.

Request/response packets

The following is an example of `UAC_SOE_OP_IN_SEND` (sending data to the IN endpoint):

```
[2019/08/22 20:51:15.0487]#--->UAC_SOE_OP_IN_SEND len 17
[2019/08/22 20:51:15.0487] 0x81 0x00 0x00 0x00 0x55 0x53 0x42 0x53
0x00 0x00 0x00 0x80 0x00 0x00 0x00 0x00 0x00
[2019/08/22 20:51:15.0487] * Endpoint:0x81
[2019/08/22 20:51:15.0487] * DataLength:13
[2019/08/22 20:51:16.0091]#<--UAC_SOE_OP_IN_SEND len=8
[2019/08/22 20:51:16.0091] 81 00 00 00 00 00 00 00
[2019/08/22 20:51:16.0091] * Endpoint:0x81
[2019/08/22 20:51:16.0091] * SUCCESS
```

- The OP Code of each SOE packet is shown after `#-->` or `#<--`, followed by the `len` and the value of the Length of the SOE header.
- Below the OP code line, the substantial data is shown in a binary dump.
- Supplementary information is also provided. In the above example, the content of the Endpoint number and Status are displayed.

- In the above example, this means that "it made an endpoint send request to `0x81` with 14 bytes of data and got a successful response back".

Event

The following is an example of the `UAC_SOE_OP_OUT_RECV` (data transmission to the IN endpoint):

```
[2019/08/22 16:41:10.0689]#<--UAC_SOE_OP_OUT_RECV len=35
[2019/08/22 16:41:10.0689]  02 00 00 00 55 53 42 43 40 bb 7c 5c 24
00 00 00 80 00 06 12 00 00 00 24 00 00 00 00 00 00 00 00 00 00 00
[2019/08/22 16:41:10.0689] * Endpoint:0x02
[2019/08/22 16:41:10.0689] * length=31
```

- You may read the logs in the same way as Request/Response packets.
- In the above example, this means "it received 31 bytes of data from endpoint 02".

Please refer to the Communication Protocol for the Serial Offload Engine (SOE) chapter for details of each SOE packet.

The following event information is issued in conjunction with a USB system event (that is, interrupt) within the SOE (EZ-USB FX3):

- `UAC_SOE_OP_SETCONFIG`
- `UAC_SOE_OP_BUS_RESET`
- `UAC_SOE_OP_CTRL_REQ_RECV`
- `UAC_SOE_OP_OUT_RECV`
- `UAC_SOE_OP_IN_COMPLETE`

Therefore, it can happen that the order of the events or communications are reversed when they occur almost simultaneously with other events or communications (for example, `UAC_SOE_OP_OP_BUS_RESET` is followed by `UAC_SOE_OP_IN_COMPLETE`).

Statistics

The USB Peripheral Module adds statistics as shown below at the end of log.

```
[2019/11/05 23:34:52.0039]----------------------------------------
-----------------
[2019/11/05 23:34:52.0039] [INFO]Number of test = 1197
[2019/11/05 23:34:52.0039] [INFO]Number of OUT Transaction      = 2205
[2019/11/05 23:34:52.0039] [INFO]Number of IN Transaction       = 2394
[2019/11/05 23:34:52.0039] [INFO]Number of retry packet = 6
[2019/11/05 23:34:52.0039] [INFO]Number of BUS RESET    = 1196
```

```
[2019/11/05 23:34:52.0039] [WARN]Number of USB data Timeout = 0
[2019/11/05 23:34:52.0039] [WARN]Number of BUS RESET    = 1
[2019/11/05 23:34:52.0039] [ERR ]Number of RECV ERROR   = 0
[2019/11/05 23:34:52.0039]----------------------------------------
----------------
```

- Each item is categorized into three types: [INFO], [WARN] and [ERR].
  - [INFO] is not a problem in terms of execution, but it is provided as information.
  - [WARN] is an item related to unexpected behavior. However, since it does not interfere with fuzzing, there is no problem as long as there are no abnormalities in the equipment.
  - [ERR] item indicates a situation that makes it difficult to continue fuzzing.
- [INFO]Number of test represents the number of fuzzed data processed (counted in the number of times `UAC_sequence_start` is called).
- [INFO]Number of OUT Transaction represents the number of times data has been received from the OUT endpoint.
- [INFO]Number of IN Transaction represents the number of times the data transfer to the IN endpoint has been completed.
- [INFO]Number of BUS RESET represents the total number of unexpected bus resets. However, bus resets that do not interfere with the test, such as those that occur during the disconnection process after sending fuzz data, are counted here.
- [WARN]Number of USB data Timeout represents the number of times the transfer is not completed within the time set in `USB_DATA_TIMEOUT` in the *Begin USB fuzzing protocol (`UAC_usb_start`)* section of the Overview of UAC USB Peripheral Module chapter. Incomplete transfers are exceptional and it depends on the specification of the target (USB host) whether it will be a problem or not. If the target (USB host) is Windows, this phenomenon will not occur (bus reset will be issued).
- [WARN]Number of BUS RESET represents the number of bus resets that occurred during the IN transfer. This is an unexpected behavior, but it doesn't interfere with fuzzing. If you are able to issue a bus reset for an unusual situation/data, it will not be a problem.
- [ERR]Number of RECV ERROR counts the number of failed serial communication retries. Therefore, if this value is non-zero, it means that the fuzzing failed.

In the case of fuzzing, the test is generally considered as OK if the target does not become abnormal and data communication is normal even after the test is completed.

Others

In addition to the SOE packets, the USB Peripheral Module logs supplementary information as needed. Here are a few examples:

Retry serial error

```
[2019/11/06 15:15:14.0494][beSTORM][enter UAC_recv_outep : 1113-2 ]
[2019/11/06 15:15:14.0694]#<--UAC_SOE_OP_BUS_RESET len=0
[2019/11/06 15:15:14.0895]#<--UAC_SOE_OP_SETCONFIG len=4
[2019/11/06 15:15:14.0895] 00 00 00 00 :
[2019/11/06 15:15:14.0895] * ConfigValue:0x0
[2019/11/06 15:15:15.0295]#<--UAC_SOE_OP_CTRL_REQ_RECV len=8
[2019/11/06 15:15:15.0295] a1 fe 00 00 00 00 01 00 :
[2019/11/06 15:15:15.0295]#--->UAC_SOE_OP_CTRL_RESPONSE len 1
[2019/11/06 15:15:15.0295] 0x00 :
[2019/11/06 15:15:17.0096]#<--UAC_SOE_OP_RECV_ERROR len=16
[2019/11/06 15:15:17.0096]  aa aa aa aa 06 00 00 10 04 00 00 00 01
00 00 00 :
[2019/11/06 15:15:17.0096]Retry latest sent packet.
[2019/11/06 15:15:17.0096]#--->UAC_SOE_OP_CTRL_RESPONSE len 1
[2019/11/06 15:15:17.0096] 0x00 :
[2019/11/06 15:15:17.0297]#<--UAC_SOE_OP_CTRL_RESPONSE len=4
[2019/11/06 15:15:17.0297] 00 00 00 00 :
[2019/11/06 15:15:17.0297] * SUCCESS
```

When a retry is performed, the message is recorded as shown in red above.

Bus reset during IN transfer

```
[2019/11/06 12:43:33.0023][beSTORM][enter UAC_send_inep : 2-4 ]
[2019/11/06 12:43:33.0023]#--->UAC_SOE_OP_IN_SEND len 17
[2019/11/06 12:43:33.0023] 0x81 0x00 0x00 0x00 0x55 0x53 0x42 0x53
0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x41 :
[2019/11/06 12:43:33.0023] * Endpoint:0x81
[2019/11/06 12:43:33.0023] * DataLength:13
[2019/11/06 12:43:42.0224]#<--UAC_SOE_OP_IN_SEND len=8
[2019/11/06 12:43:42.0224] 81 00 00 00 00 00 00 00 :
[2019/11/06 12:43:42.0224] * Endpoint:0x81 [2019/11/06
12:43:42.0224] * SUCCESS
[2019/11/06 12:43:42.0424]#<--UAC_SOE_OP_BUS_RESET len=0
[2019/11/06 12:43:42.0424][WARN]You may check this UAC_SOE_OP_BUS_
RESET.
[2019/11/06 12:43:42.0424]USB Error in transmit with SOE.
[2019/11/06 12:43:42.0425][beSTORM][enter UAC_recv_outep : 2-5 ]
[2019/11/06 12:43:42.0425]UAC_recv_outep: SKIP this sequence because
illegal status.
[2019/11/06 12:43:42.0425][beSTORM][enter UAC_sequence_end : 2-6 ]
[2019/11/06 12:43:42.0425]#--->UAC_SOE_OP_SET_CONN_STATE len 4
```

```
[2019/11/06 12:43:42.0425] 0x00 0x00 0x00 0x00 :
[2019/11/06 12:43:42.0425] * DISCONNECT
```

When IN transfer fails (that is, the USB host refuses to accept data, which is considered as compliant to the protocol), the message `[WARN]You may check this UAC_SOE_OP_BUS_RESET` is recorded as shown in red above.

Data transfer time out

```
[2019/11/06 12:43:33.0023][beSTORM][enter UAC_send_inep : 2-4 ]
[2019/11/06 12:43:33.0023]#--->UAC_SOE_OP_IN_SEND len 17
[2019/11/06 12:43:33.0023] 0x81 0x00 0x00 0x00 0x55 0x53 0x42 0x53
0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x41 :
[2019/11/06 12:43:33.0023] * Endpoint:0x81
[2019/11/06 12:43:33.0023] * DataLength:13
[2019/11/06 12:43:42.0224]#<--UAC_SOE_OP_IN_SEND len=8
[2019/11/06 12:43:42.0224] 81 00 00 00 00 00 00 00 :
[2019/11/06 12:43:42.0224] * Endpoint:0x81
[2019/11/06 12:43:42.0224] * SUCCESS
[2019/11/06 12:53:49.0232][WARN]Timeout of USB transmit. msec=600000
```

When the data transfer times out, the message `[WARN]Timeout of USB transmit` is recorded as shown in red above. If the target (USB host) is Windows, this phenomenon will not occur (bus reset will be issued).

Retry communication failure

```
[2019/11/06 15:15:15.0295]#--->UAC_SOE_OP_CTRL_RESPONSE len 1
[2019/11/06 15:15:15.0295] 0x00 :
[2019/11/06 15:15:17.0096]#<--UAC_SOE_OP_RECV_ERROR len=16
[2019/11/06 15:15:17.0096]  aa aa aa aa 06 00 00 10 04 00 00 00 01
00 00 00 :
[2019/11/06 15:15:17.0096] [ERR ]Can't retry -> error occured
```

If the number of retries specified by `SER_ERR_RETRY_COUNT` in "" fails, the message `[ERR ]Can't retry -> error occured` is recorded as shown in red above.

Recovery Tool

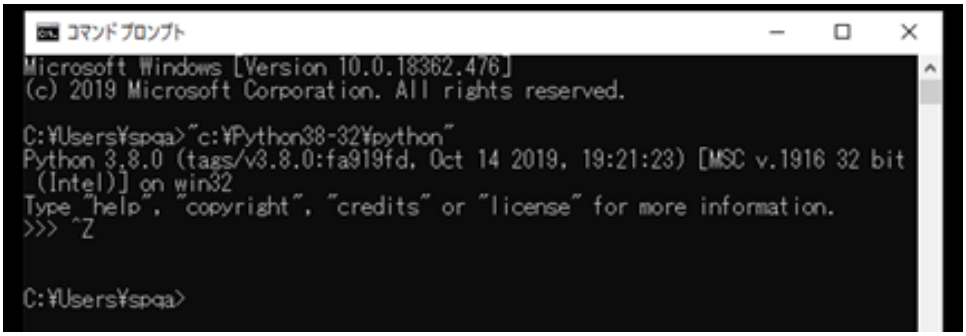
By using the recovery tool, you can manage the fuzzing done in beSTORM individually and retest it if necessary.

- You must run the recovery tool in the same environment as beSTORM.
- Python 3 (32-bit variant) is required.
- It uses the same DLL as beSTORM, so you may not use the recovery tool and beSTORM at the time.

Installing Python 3 (32-bit variant)

beSTORM is a 32-bit application, so a 32-bit variant of Python 3 is required. This section describes how to perform the installation without setting the path, etc. Follow the steps below to install the software:

1. Download the latest version of the Python Windows x86 executable installer at https://www.python.org/downloads/release/python-380/.
2. After the installer file finishes downloading, double-click on it (the file name may be slightly different depending on the version).
3. Clear the **Install launcher for all users** check box, and then select **Customize installation**.
4. Select **Next**. The **Advanced Options** dialog is displayed.
5. Clear the **Associate files with Python (requires the py launcher)** and **Create shortcuts for installed applications** check boxes. Do not change the **Customize install location** default location.
6. Select **Install**.
7. Start a command prompt and check if Python is using `< installation path >\python`.



You can exit from Pyhthon by pressing CTRL-Z ENTER. The figure below shows an example of the installation on `c:\Python38-32`. The rest of this document is written as assuming this path. If you install it in a different location, please replace the path location by yours. The following instructions are executed on the command prompt.

8. Update the Python package controller (pip) with the following command:

   ```
   "c:\Python38-32\pyton" -m pip install --upgrade pip
   ```

9. Install packages for the recovery tool

   ```
   "c:\Python38-32\pyton" -m pip install hexdump
   ```

Overview of the recovery tool

The recovery tool consists of Python scripts, DLLs and batch files in the `best_dataplayer` folder.

The USB Peripheral Module (DLL) can perform analysis and re-fuzzing using the record data (file specified by `LOG_FILE`) and log (file specified by `RECORD_FILE`) created by the USB Peripheral Module (DLL).

The USB Peripheral Module (DLL) creates record data (the file specified by `LOG_FILE`) separately from the log at the time of fuzzing. This is a record of the actual fuzzing data. The recovery tool can use this data to perform fuzzing again.

The recovery tool uses the settings made in beSTORM as they are and re-fuzzes them using the same DLL as beSTORM. Therefore, the record data (the file specified by `LOG_FILE`) and the log (the file specified by `RECORD_FILE`) may be overwritten, so make a backup if necessary.

Please note that Timed is not supported as a limitation.

List of commands

The recovery tool can be executed from the command prompt. The following is a list of commands.

```
"c:\Python38-32\pyton" usbd_player.py init < project name> < RECORD_
FILE path > < path to the dll> < LOG_FILE path >
```

Creates a project for the recovery tool (the above example breaks one long line into two for better typesetting, type them without breaking lines).

```
"c:\Python38-32\pyton” usbd_player.py load < project name >
```

Loads a project previously created with the recovery tool for reuse.

```
"c:\Python38-32\pyton” usbd_player.py delete < project name >
```

Deletes a project previously created with the recovery tool.

```
"c:\Python38-32\pyton” usbd_player.py parse < path to CSV output >
```

Analyzes the fuzzing content from log and record data and generates a CSV file. At the same time, it also creates a list of sequences that have failed to communicate.

```
"c:\Python38-32\pyton” usbd_player.py playseq < all or numbers like
20001, 2002 >
```

If "all" is specified, all packets that failed to communicate will be fuzzed again. Or you can specify a sequence number to be fuzzed.

Instruction

The recovery tool is provided by `firmware/best_uac_usbd.img` in the release file. Copy this folder to any location, open a command prompt and move the working directory to that folder.

Initializing/creating a project

Use the init command in ***List of commands* on page 88** to initialize the program. Because there are many arguments, it can be done by using the batch file `usbd_init.bat`. The contents of `usbd_init`.bat are shown below.

```
@echo off

rem Don’t use a space in the following configuration ( Write not set
xxx = yyy, but set xxx=yyy ) rem Specify the path to pyhon.exe

rem Specify the path to pyhon.exe
set pythonPath="c:\Python38-32\python.exe"

rem Specify the project name
set projectName=msc_test1
```

```
rem Specify the path to the record file
set recordFile="C:\TEMP\best_record.dat"

rem Specify the pass to the DLL
set dll="C:\TEMP\best_uac_usbd.dll"

rem Specify the file name of the log file created by best_uac_
usbd.dll
set log="C:\TEMP\bestorm.log"

@echo on

%pythonPath% usbd_player.py init %projectName% %recordFile% %dll%
%log%
```

The above `pythonPath`, `projectName`, `recordFile, dll`, and log should be customized for your environment. The initialization is completed by running `usbd_init.bat` from the command prompt.

Once the initialization is complete, the recorder will generate project files from project name `.bs1` to project name `.bs4`. Based on these files, parse and `playseq` operations are performed. Make a backup of these files if necessary. The recorder stores the name of this project in the file: `best_player.ini`. So you can use this project to parse and `playseq` until the changes are made.

The contents of each project file are as follows.

- `< project name >.bs1` - Same as the contents of the recorder file
- `< project name >.bs2` - The pass to the DLL
- `< project name >.bs3` - Same as the log created by the DLL
- `< project name >.bs4` - List that recovery tool creates (list of data to be recovered)

The actual list is created by the operation of ***Analyzing fuzzing* on page 91**.

Removing a project

To destroy a project file, run the following:

```
"c:\Python38-32\pyton” usbd_player.py delete < project name >
```

Please note that the file will be deleted.

Loading a project

To load a project, run the following.

```
"c:\Python38-32\pyton” usbd_player.py load < project name >
```

It will not work properly if any file from project name `.bs1` to project name `.bs4` does not exist. Once loaded, the recorder records the project name in the file: `best_player.ini`. So you can use this project to parse and `playseq` until the changes are made.

Analyzing fuzzing

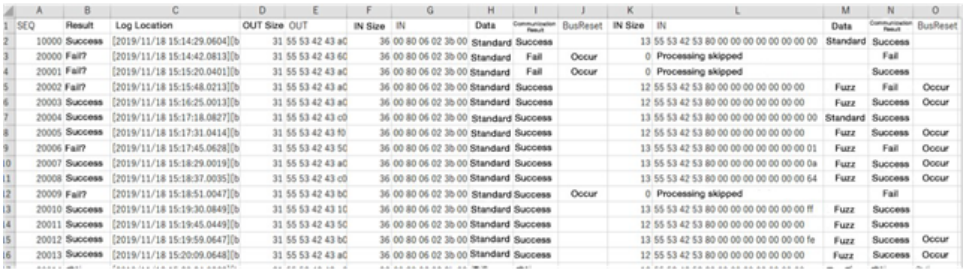
To analyze fuzzing with the recovery tool, run the following

```
"c:\Python38-32\pyton” usbd_player.py parse < path to CSV output >
```

It creates a CSV file with the specified name. In it, the following data is listed:

Column	Description
SEQ	The sequence number, which you can use to re-fuzz with the play command.
Result	When two `UAC_SOE_OP_IN_COMPLETE` of the following IN data are present, it outputs "Success."
Log Location	If you want to see this data in the log, please search for it in the text provided here.
OUT Size	The size of the CBW (Inquiry command) received.
OUT Data	The data of CBW (Inquiry command) received.
IN Size	The size of the Inquiry response sent.
IN Data	The data of the Inquiry response sent.
Communication Result	If `UAC_SOE_OP_IN_COMPLETE` is present, it outputs "Success."
BusReset	If `UAC_SOE_OP_BUS_RESET` is present, it outputs "Occurred."
IN Size	The size of the CSW sent.
IN Data	The data of the CSW sent.
Communication Result	If `UAC_SOE_OP_IN_COMPLETE` is present, it outputs "Success."
BusReset	If `UAC_SOE_OP_BUS_RESET` is present, it outputs "Occurred."

The following figure shows an image of the data in Excel (slightly modified in Excel).



SEQ	Result	Log Location	OUT Size	OUT	IN Size	IN	Data	Communication Result	BusReset	IN Size	IN	Data	Communication Result	BusReset
10000	Success	[2019/11/18 15:14:29.0604][b	31	55 53 42 43 a0	36	00 80 06 02 3b 00	Standard	Success		13	55 53 42 53 80 00 00 00 00 00 00 00 00	Standard	Success	
20000	Fail?	[2019/11/18 15:14:42.0813][b	31	55 53 42 43 60	36	00 80 06 02 3b 00	Standard	Fail	Occur	0	Processing skipped		Fail	
20001	Fail?	[2019/11/18 15:15:20.0401][b	31	55 53 42 43 a0	36	00 80 06 02 3b 00	Standard	Fail	Occur	0	Processing skipped		Success	
20002	Fail?	[2019/11/18 15:15:48.0213][b	31	55 53 42 43 a0	36	00 80 06 02 3b 00	Standard	Success		12	55 53 42 53 80 00 00 00 00 00 00 00	Fuzz	Fail	Occur
20003	Success	[2019/11/18 15:16:25.0013][b	31	55 53 42 43 a0	36	00 80 06 02 3b 00	Standard	Success		12	55 53 42 53 80 00 00 00 00 00 00 00	Fuzz	Success	Occur
20004	Success	[2019/11/18 15:17:18.0827][b	31	55 53 42 43 c0	36	00 80 06 02 3b 00	Standard	Success		13	55 53 42 53 80 00 00 00 00 00 00 00 00	Standard	Success	
20005	Success	[2019/11/18 15:17:31.0414][b	31	55 53 42 43 f0	36	00 80 06 02 3b 00	Standard	Success		12	55 53 42 53 80 00 00 00 00 00 00 00	Fuzz	Success	Occur
20006	Fail?	[2019/11/18 15:17:45.0628][b	31	55 53 42 43 50	36	00 80 06 02 3b 00	Standard	Success		13	55 53 42 53 80 00 00 00 00 00 00 00 01	Fuzz	Fail	Occur
20007	Success	[2019/11/18 15:18:29.0019][b	31	55 53 42 43 a0	36	00 80 06 02 3b 00	Standard	Success		13	55 53 42 53 80 00 00 00 00 00 00 00 0a	Fuzz	Success	Occur
20008	Success	[2019/11/18 15:18:37.0035][b	31	55 53 42 43 c0	36	00 80 06 02 3b 00	Standard	Success		13	55 53 42 53 80 00 00 00 00 00 00 00 64	Fuzz	Success	Occur
20009	Fail?	[2019/11/18 15:18:51.0047][b	31	55 53 42 43 b0	36	00 80 06 02 3b 00	Standard	Success	Occur	0	Processing skipped		Fail	
20010	Success	[2019/11/18 15:19:30.0849][b	31	55 53 42 43 10	36	00 80 06 02 3b 00	Standard	Success		13	55 53 42 53 80 00 00 00 00 00 00 00 ff	Fuzz	Success	
20011	Success	[2019/11/18 15:19:45.0449][b	31	55 53 42 43 50	36	00 80 06 02 3b 00	Standard	Success		12	55 53 42 53 80 00 00 00 00 00 00 00	Fuzz	Success	
20012	Success	[2019/11/18 15:19:59.0647][b	31	55 53 42 43 b0	36	00 80 06 02 3b 00	Standard	Success		13	55 53 42 53 80 00 00 00 00 00 00 00 fe	Fuzz	Success	Occur
20013	Success	[2019/11/18 15:20:09.0648][b	31	55 53 42 43 a0	36	00 80 06 02 3b 00	Standard	Success		12	55 53 42 53 80 00 00 00 00 00 00 00	Fuzz	Success	Occur

Supplementary information about sequence numbers: The sequence numbers are determined as follows.

- 10000 The data of setting of beSTORM and USB communication of the first time (default value).
- 20000s Processing of fuzzing data communication for and after the second time.

Replaying fuzzing

To replay fuzzing, you can use the `playseq` command, but note the following:

- Analyze it with parse command (you only need to do it once).
- The recovery tool and beSTORM must be in the same environment.
- beSTORM must be shutdown.
- Log files will be overwritten, so please make a backup if necessary.

To replay all failed communications, use all as follows

```
"c:\Python38-32\pyton” usbd_player.py playseq all
```

The recovery tool resends all the failed data. Specifically, it resends all data with the column of Result in ***Analyzing fuzzing* on page 91** is “Failure?”.

It is also possible to specify individual numbers and perform fuzzing again. Here's how to do it when you have three data streams, 20010, 20015, and 20019.

```
"c:\Python38-32\pyton” usbd_player.py playseq 20010,20015,20019
```

> **NOTE:** There should be no spaces in 20010,20015,20019. Otherwise, it will no longer work properly. Also, the same number cannot be specified more than once.

The sequence number 10000 (initialization process, etc.) is always executed first in the recovery tool, so there is no need to specify it.

Analyzing the result of replay fuzzing

You can use reparse to reanalyze the results of `playseq`. Because beSTORM assigns sequence numbers in the order in which they are executed, they are different from the sequence numbers that were assigned when the project was created.

Here's how it works:

```
"c:\Python38-32\pyton” usbh_player.py reparse < path to input log
data > < pass to output log data > < path to the result CSV >
"c:\Python38-32\pyton” usbh_player.py reparse up < path to input log
data > < pass to output log data > < path to the result CSV >
```

For example, type the following.

```
"c:\Python38-32\pyton” usbh_player.py reparse
\\192.168.0.23\beSTORM\bestorm_log\bestorm.log out.log out.csv
```

It will generate out.log and out.csv from the log at `playseq` on the virtual machine (192.168.0.23). The .log and .csv formats are the same, but the sequence numbers are changed to the numbers used in project generation.

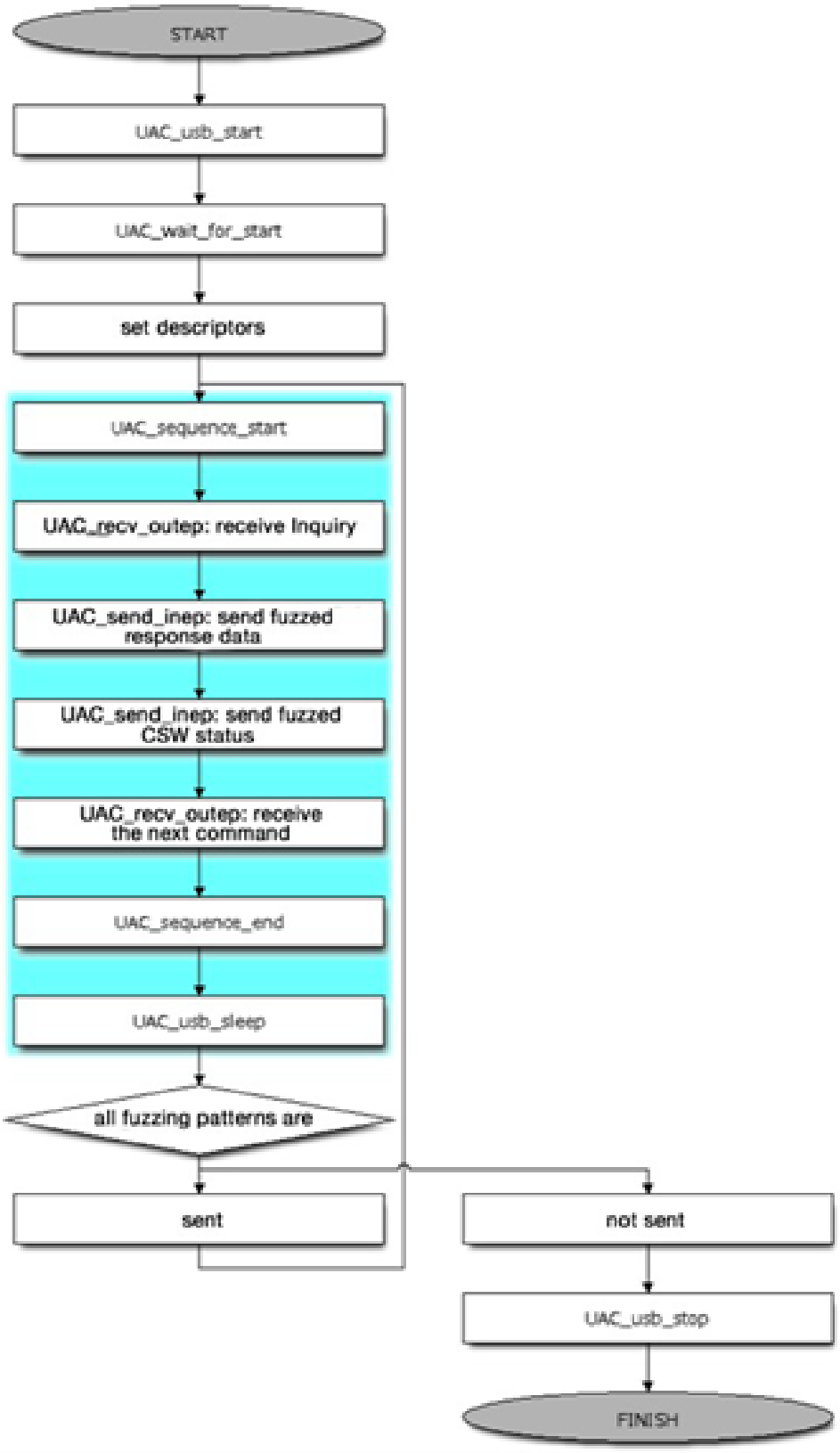
Moreover, the test list is also updated by specifying *up* before the argument. Therefore, only the data that fails in the input log data path can be replayed by the next `playseq` all.

Customize the UAC USB Peripheral Module

You can change the fuzzing scenario by changing the content of the XML. This section describes how to modify the sample in the Sample for Mass Storage chapter for fuzzing.

Modifying mass storage sample

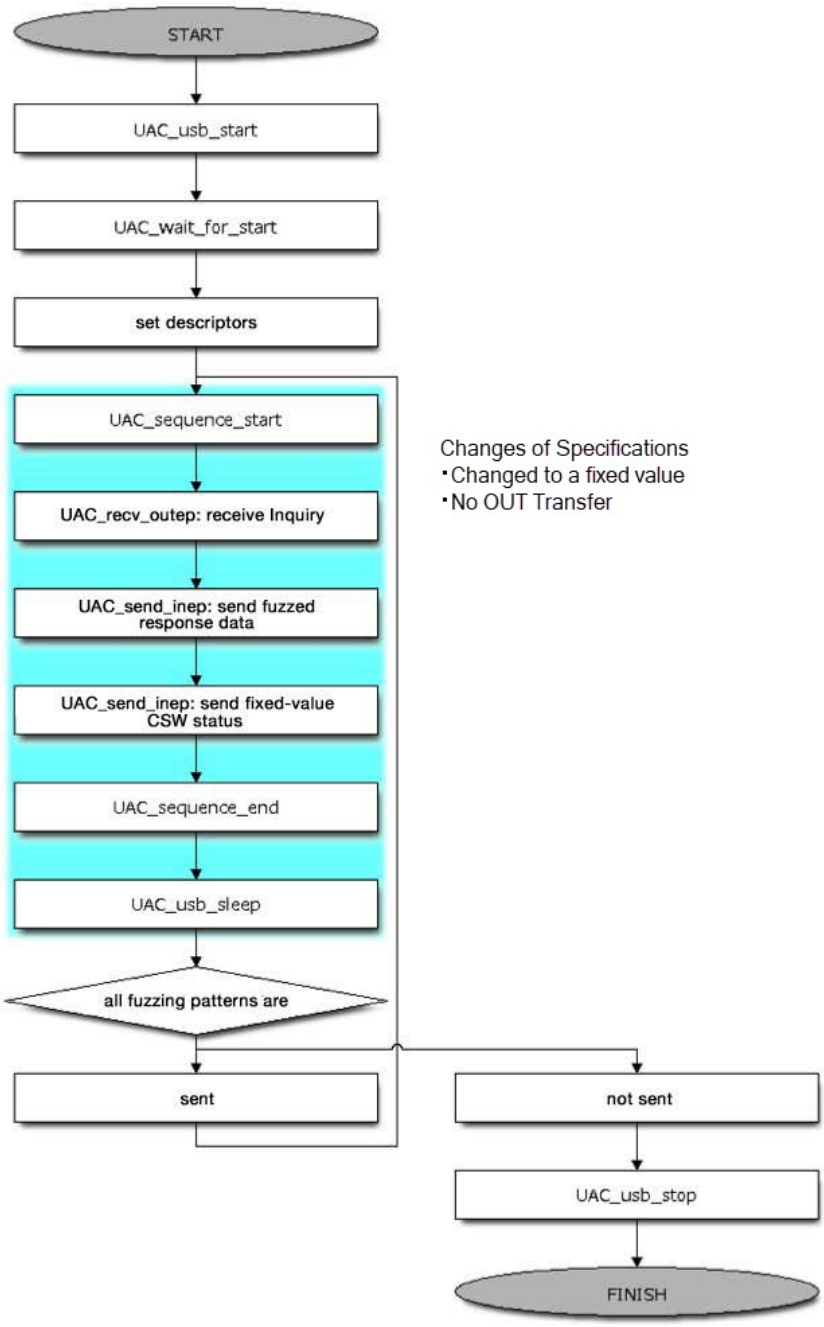
With the XML described in the *XML* section of the Sample for Mass Storage chapter, fuzzing is done as shown below:



In this section, we will make the following modifications to reduce the number of tests.

- CSW returns a fixed value.
- Since CSW is a fixed value, the second OUT transmission/reception processing is not performed.

The sequence is shown below:



Modifying XML

Here is the XML content modified according to the previous section. The changes are shown in red.

```
<?xml version="1.0" ?>
<!DOCTYPE beSTORM SYSTEM '\Program Files\beSTORM\beSTORM.dtd'>
<beSTORM Revision="$Revision: 7298 $" Version="1.2">
<Global/>
<GeneratorOptSettings>
<BT FactoryDefined="1" FactoryType="Binary"/>
</GeneratorOptSettings>
<ModuleSettings>
<M Name="UAC USB peripheral1">
<P Name="UAC USB peripheral1 protocol">
<SC Name="Initialize">
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_usb_start SC"
Procedure="UAC_usb_start">
<S Name="COM_PORT" ParamName="COM_PORT">
<EV ASCIIValue="4" Description="COM" Name="COM" Required="1"/>
</S>
<S Name="CSV_FILE" ParamName="CSV_FILE">
<EV ASCIIValue="C:\TEMP\bestorm.csv" Description="CSV_ATH"
Name="CSV_ATH" Required="1"/>
</S>
<S Name="LOG_FILE" ParamName="LOG_FILE">
<EV ASCIIValue="C:\TEMP\bestorm.log" Description="LOG_PATH"
Name="LOG_PATH" Required="1"/>
</S>
<S Name="RECORD_FILE" ParamName="RECORD_FILE">
<EV ASCIIValue="c:\TEMP\best_record.dat" Description="RECORD_PATH"
Name="RECORD_PATH" Required="1"/>
</S>
<S Name="ILLEGAL_TIMER" ParamName="ILLEGAL_TIMER">
<EV ASCIIValue="3000" Description="timer" Name="timer"
Required="1"/>
</S>
<S Name="SER_ERR_RETRY_COUNT" ParamName="SER_ERR_RETRY_COUNT">
<EV ASCIIValue="3" Description="ser_err_retry_count" Name="ser_err_
retry_count" Required="1"/>
</S>
<S Name="USB_DATA_TIMEOUT" ParamName="USB_DATA_TIMEOUT">
<EV ASCIIValue="600" Description="usb_data_timeout" Name="usb_data_
timeout" Required="1"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_wait_for_start SC"
```

```
Procedure="UAC_wait_for_start"/>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_dev_desc20 SC"
Procedure="UAC_set_dev_desc20">
<S Name="DevDesc20" ParamName="DevDesc20">
<C Name="bLength" Value="0x12"/>
<C Name="bDescriptorType" Value="0x01"/>
<C Name="bcdUSB" Value="0x10 0x02"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
<C Name="bMaxPacketSize0" Value="0x40"/>
<C Name="idVendor" Value="0x45 0x04"/>
<C Name="idProduct" Value="0xf1 0x00"/>
<C Name="bcdDevice" Value="0x00 0x00"/>
<C Name="iManufacturer" Value="0x01"/>
<C Name="iProduct" Value="0x02"/>
<C Name="iSerialNumber" Value="0x00"/>
<C Name="bNumConfigurations" Value="0x01"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_dev_desc30 SC"
Procedure="UAC_set_dev_desc30">
<S Name="DevDesc30" ParamName="DevDesc30">
<C Name="bLength" Value="0x12"/>
<C Name="bDescriptorType" Value="0x01"/>
<C Name="bcdUSB" Value="0x00 0x03"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
<C Name="bMaxPacketSize0" Value="0x09"/>
<C Name="idVendor" Value="0x45 0x04"/>
<C Name="idProduct" Value="0xf1 0x00"/>
<C Name="bcdDevice" Value="0x00 0x00"/>
<C Name="iManufacturer" Value="0x01"/>
<C Name="iProduct" Value="0x02"/>
<C Name="iSerialNumber" Value="0x00"/>
<C Name="bNumConfigurations" Value="0x01"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_qual_desc SC"
Procedure="UAC_set_qual_desc">
<S Name="QualDesc" ParamName="QualDesc">
<C Name="bLength" Value="0x0a"/>
<C Name="bDescriptorType" Value="0x06"/>
<C Name="bcdUSB" Value="0x00 0x02"/>
<C Name="bDeviceClass" Value="0x00"/>
<C Name="bDeviceSubClass" Value="0x00"/>
<C Name="bDeviceProtocol" Value="0x00"/>
```

```
<C Name="bMaxPacketSize0" Value="0x40"/>
<C Name="bNumConfigurations" Value="0x01"/>
<C Name="bReserved" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_fs_config SC"
Procedure="UAC_set_fs_config">
<S Name="FSCONFIG" ParamName="FSCONFIG">
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x02"/>
<C Name="wTotalLength" Value="0x20 0x00"/>
<C Name="bNumInterface" Value="0x01"/>
<C Name="bConfigurationValue" Value="0x01"/>
<C Name="iConfiguration" Value="0x00"/>
<C Name="bmAttributes" Value="0x80"/>
<C Name="bMaxPower" Value="0x32"/>
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x04"/>
<C Name="bInterfaceNumber" Value="0x00"/>
<C Name="bAlternateSetting" Value="0x00"/>
<C Name="bNumEndpoints" Value="0x02"/>
<C Name="bInterfaceClass" Value="0x08"/>
<C Name="bInterfaceSubClass" Value="0x06"/>
<C Name="bInterfaceProtocol" Value="0x50"/>
<C Name="iInterfaceProtocol" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x81"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x40 0x00"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x02"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x40 0x00"/>
<C Name="bInterval" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_hs_config SC"
Procedure="UAC_set_hs_config">
<S Name="HSCONFIG" ParamName="HSCONFIG">
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x02"/>
<C Name="wTotalLength" Value="0x20 0x00"/>
<C Name="bNumInterface" Value="0x01"/>
<C Name="bConfigurationValue" Value="0x01"/>
<C Name="iConfiguration" Value="0x00"/>
```

```
<C Name="bmAttributes" Value="0x80"/>
<C Name="bMaxPower" Value="0x32"/>
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x04"/>
<C Name="bInterfaceNumber" Value="0x00"/>
<C Name="bAlternateSetting" Value="0x00"/>
<C Name="bNumEndpoints" Value="0x02"/>
<C Name="bInterfaceClass" Value="0x08"/>
<C Name="bInterfaceSubClass" Value="0x06"/>
<C Name="bInterfaceProtocol" Value="0x50"/>
<C Name="iInterfaceProtocol" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x81"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x02"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x02"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x02"/>
<C Name="bInterval" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_ss_config SC"
Procedure="UAC_set_ss_config">
<S Name="SSCONFIG" ParamName="SSCONFIG">
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x02"/>
<C Name="wTotalLength" Value="0x20 0x00"/>
<C Name="bNumInterface" Value="0x01"/>
<C Name="bConfigurationValue" Value="0x01"/>
<C Name="iConfiguration" Value="0x00"/>
<C Name="bmAttributes" Value="0x80"/>
<C Name="bMaxPower" Value="0x32"/>
<C Name="bLength" Value="0x09"/>
<C Name="bDescriptorType" Value="0x04"/>
<C Name="bInterfaceNumber" Value="0x00"/>
<C Name="bAlternateSetting" Value="0x00"/>
<C Name="bNumEndpoints" Value="0x02"/>
<C Name="bInterfaceClass" Value="0x08"/>
<C Name="bInterfaceSubClass" Value="0x06"/>
<C Name="bInterfaceProtocol" Value="0x50"/>
<C Name="iInterfaceProtocol" Value="0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x81"/>
```

```
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x04"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x06"/>
<C Name="bDescriptorType" Value="0x30"/>
<C Name="bMaxBurst" Value="0x0f"/>
<C Name="bmAttributes" Value="0x00"/>
<C Name="wBytesPerInterval" Value="0x00 0x00"/>
<C Name="bLength" Value="0x07"/>
<C Name="bDescriptorType" Value="0x05"/>
<C Name="bEndpointAddress" Value="0x02"/>
<C Name="bmAttributes" Value="0x02"/>
<C Name="wMaxPacketSize" Value="0x00 0x04"/>
<C Name="bInterval" Value="0x00"/>
<C Name="bLength" Value="0x06"/>
<C Name="bDescriptorType" Value="0x30"/>
<C Name="bMaxBurst" Value="0x0f"/>
<C Name="bmAttributes" Value="0x00"/>
<C Name="wBytesPerInterval" Value="0x00 0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_str_lang SC"
Procedure="UAC_set_str_lang">
<S Name="StrLang" ParamName="StrLang">
<C Name="bLength" Value="0x04"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="wLANGID" Value="0x09 0x04"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_str_
manufacture SC" Procedure="UAC_set_str_manufacture">
<S Name="StrManufacture" ParamName="StrManufacture">
<C Name="bLength" Value="0x10"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="bString" Value="0x43 0x00 0x79 0x00 0x70 0x00 0x72 0x00
0x65 0x00 0x73 0x00 0x73 0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_set_str_product
SC" Procedure="UAC_set_str_product">
<S Name="StrProduct" ParamName="StrProduct">
<C Name="bLength" Value="0x08"/>
<C Name="bDescriptorType" Value="0x03"/>
<C Name="bString" Value="0x55, 0x00, 0x41, 0x00, 0x43, 0x00"/>
</S>
</SP>
</SC>
<SC Name="Fuzzing">
```

```
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_sequence_start SC"
Procedure="UAC_sequence_start"/>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_recv_outep SC1"
Procedure="UAC_recv_outep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x02"/>
</S>
<S Name="BUS_RESET" ParamName="BUS_RESET">
<C ASCIIValue="SKIP" Name="bus_reset"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_send_inep response
SC" Procedure="UAC_send_inep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x81"/>
</S>
<S Name="data" ParamName="data">
<B MaxBytes="1" MinBytes="1" Name="PeripheralQualifier"
Value="0x00"/>
<B MaxBytes="1" MinBytes="1" Name="RMB" Value="0x80"/>
<B MaxBytes="1" MinBytes="1" Name="Versions" Value="0x06"/>
<B MaxBytes="1" MinBytes="1" Name="Response Data Format"
Value="0x02"/>
<B MaxBytes="4" Name="Length" Value="0x3b"/>
<B MaxBytes="2" MinBytes="2" Name="Reserve" Value="0x00 0x00"/>
<B MaxBytes="1" MinBytes="1" Name="RelADR plus" Value="0x00"/>
<B MaxBytes="16" Name="Vendor" Value="0x30 0x31 0x32 0x33 0x34 0x35
0x36 0x37"/>
<B MaxBytes="32" Name="Product" Value="0x61 0x62 0x63 0x64 0x65 0x66
0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70"/>
<B MaxBytes="8" Name="Revision" Value="0x30 0x31 0x32 0x33"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_send_inep csw SC"
Procedure="UAC_send_inep">
<S Name="EP" ParamName="EP">
<C Name="addr" Value="0x81"/>
</S>
<S Name="data" ParamName="data">
<C Name="dCSWSignature" Value="0x55 0x53 0x42 0x53"/>
<C Name="dCSWTag" Value="0x80 0x00 0x00 0x00"/>
<C Name="dCSWDataResidue" Value="0x00 0x00 0x00 0x00"/>
<C Name="bCSWStatus" Value="0x00"/>
</S>
</SP>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_sequence_end SC"
Procedure="UAC_sequence_end"/>
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_usb_sleep csw SC"
```

```
Procedure="UAC_usb_sleep">
<S Name="msec" ParamName="msec">
<C ASCIIValue="2000" Name="value"/>
</S>
</SP>
</SC>
<SC Name="Deinitialize">
<SP Library="c:\work\best_uac_usbd.dll" Name="UAC_usb_stop SC"
Procedure="UAC_usb_stop"/>
</SC>
</P>
</M>
</ModuleSettings>
</beSTORM>
```

- The data definition part of the SP tag (in the S tag) of `UAC_send_inep csw SC` has been modified. As the difference is shown below, the B tag is replaced by the C tag and set to a fixed value. You may change the role of the data by changing the tag name likewise.
  - Before changes

    ```
    <S Name="data" ParamName="data">
    <B MaxBytes="4" Name="dCSWSignature" Value="0x55 0x53 0x42
    0x53"/>
    <B MaxBytes="4" Name="dCSWTag" Value="0x80 0x00 0x00
    0x00"/>
    <B MaxBytes="4" Name="dCSWDataResidue" Value="0x00 0x00
    0x00 0x00"/>
    <B MaxBytes="1" Name="bCSWStatus" Value="0x00"/>
    </S>
    ```

  - After changes

    ```
    <S Name="data" ParamName="data">
    <C Name="dCSWSignature" Value="0x55 0x53 0x42 0x53"/>
    <C Name="dCSWTag" Value="0x80 0x00 0x00 0x00"/>
    <C Name="dCSWDataResidue" Value="0x00 0x00 0x00 0x00"/>
    <C Name="bCSWStatus" Value="0x00"/>
    </S>
    ```

- The SP tag of `UAC_recv_outep SC2` has been removed. Therefore, it has been changed not to wait for the second OUT transfer.
  - Before changes

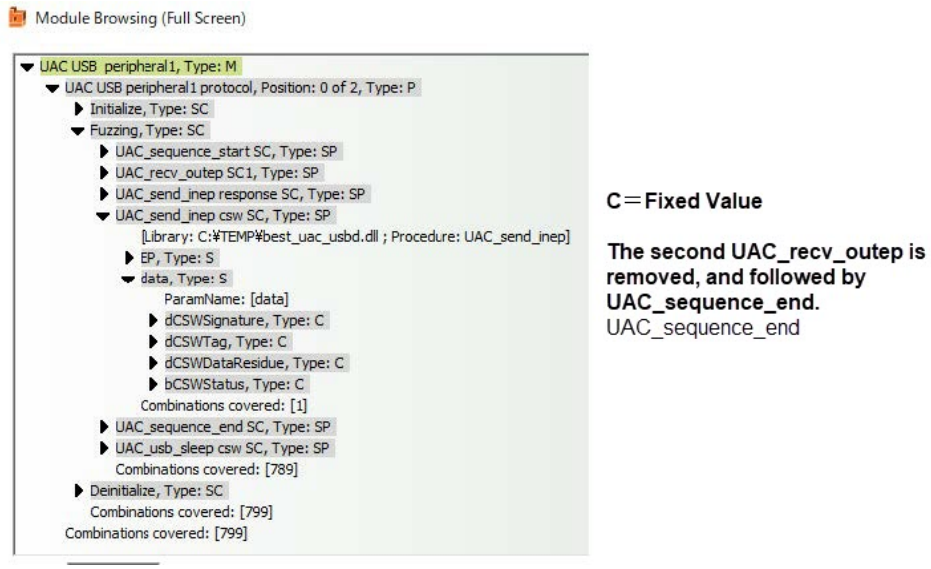
    ```
    <SP Library="c:\work\best_uac_usbd.dll" Name="UAC_recv_
    outep SC2" Procedure="UAC_recv_outep">
    <S Name="EP" ParamName="EP">
    ```

```
<C Name="addr" Value="0x02"/>
</S>
<S Name="BUS_RESET" ParamName="BUS_RESET">
<C ASCIIValue="BREAK" Name="bus_reset"/>
</S>
</SP>
```

- After changes

  Nothing (deleted)

The following is the ModuleBrowsing screen when this XML is applied to beSTORM. The changed state is displayed. The number of tests is also around 800.



Fuzzing with the modified XML

- When you change the XML, you need to create a new beSTORM project.
- Replace the XML and create a project following the steps in the *How to fuzz a USB device with the EZ-USB FX3 board* section of the USB Fuzzing Setup chapter. After that, you can follow the same procedure for fuzzing.

In this way, you can perform fuzzing with different specifications by rewriting the XML.

Resources

The release files for the UAC USB Peripheral Module are attached to the *UAC USB Module Guide* page on the Beyond Security portal) at https://beyondsecurity.freshdesk.com/a/solutions/articles/44002334275 as `UAC_USB_Peripheral_Module.zip`. The files with in the zip file are organized as shown below:

Basically, fuzzing is done by using beSTORM in the firmware folder and the EZ-USB FX3 firmware in the modules folder, and if necessary, recovery is done by using the best_dataplayer folder. Customization is supposed to be done by changing the XML. Normally, you do not need to build the source code, just use `best_uac_usbd.img` in the firmware and `best_uac_usbd.dll` in the modules folder.

Building DLL

1. Unzip `best_uac_usbd.zip`.
2. Double-click `best_uac_usbd.sln` to start VisualStudio.
3. Set the configuration to Debug, and the platform to x86 in VisualStudio.
4. Once build is completed, `best_uac_usbd.dll` is created under the Debug folder.

An overview of the included source code is shown below:

List of resources

File name	Description
bestorm_dll.cpp	Tool functions used in customized implementations of beSTORM.
bestorm_dll.h	DLL functions called by beSTORM.
best_uac_usbd.cpp	Procedures to launching and terminating the DLL itself.
dllmain.cpp	Tools for beSTORM provided by Beyond Security.
helper.cpp	Header file.
helper.h	Serial Communications Module.
serial_if.cpp	Files automatically generated by VisualStudio (basic DLL description).
serial_if.h	Files automatically generated by VisualStudio (basic DLL description).
stdafx.cpp	Files automatically generated by VisualStudio (basic DLL description).
stdafx.h	SOE Communications Processing Module.
targetver.h	Header file.
uac_bestorm_soe.cpp	Analyzing and writing SOE Packets to the log.
uac_bestorm_soe.h	Buffer processing for serial reception.
uac_bestorm_soe_ logger.cpp	Header file.
uac_bestorm_recv.cpp	CSV file related processing of automatic response.
uac_dll_usb_soe.h	Header file.
UAC_SOE_csv.cpp	Overhead for each DLL process.
UAC_SOE_csv.h	Header file.
UAC_SOE_dllfunc.cpp	Windows Exclusion Handling.
UAC_SOE_dllfunc.h	Header file.
UAC_SOE_event.cpp	Log file manipulation.
UAC_SOE_event.h	Header file.
UAC_SOE_LogFie.cpp	USB monitor processing.
UAC_SOE_LogFie.h	Header file.
UAC_SOE_monitor.cpp	Reception Processing from SOE.
UAC_SOE_monitor.h	Header file.

File name	Description
UAC_SOE_recv.cpp	Data transfer timeout management.
UAC_SOE_recv.h	Header file.
UAC_SOE_transmit_ timeout.cpp	Main processing of USB Peripheral Module.
UAC_SOE_transmit_ timeout.h	Header file.
UAC_SOE_usb_manager .cpp	Processing of recovery data writing.
UAC_SOE_usb_manager .h	Header file.
UAC_BEST_ DataRecorder.cpp	Tool functions used in customized implementations of beSTORM.
UAC_BEST_DataRecorder.h	Header file.

Building EZ-USB FX3 Firmware

The FX-USB 1.3 SDK from Cypress is required to build the firmware. Go to https://www.infineon.com/cms/en/product/evaluation-boards/cyusb3kit-003/ to download and install the **DOWNLOAD - SuperSpeedExplorerKitSetup_RevSS.exe** under the Development Tools section of the web page.

The `Makefile` needs to be customized accordingly. The following is an excerpt from `Makefile`:

```
# Settings
SDK_CORE_PATH = C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3
LIBC_A_PATH = "$(SDK_CORE_PATH)\ARM GCC\arm-none-eabi\lib\libc.a"
LIBGCC_A_PATH = "$(SDK_CORE_PATH)\ARM GCC\lib\gcc\arm-none-
eabi\4.8.1\libgcc.a"

IMG_TOOL_PATH = "$(SDK_CORE_PATH)\util\elf2img\elf2img.exe"
EZUSB_SDK_INCPATH = "$(SDK_CORE_PATH)\firmware\u3p_firmware\inc"
FX3_LD_PATH = "$(SDK_CORE_PATH)\firmware\common\fx3.ld"
CYFXAPI_A_PATH = "$(SDK_CORE_PATH)\firmware\u3p_firmware\lib\fx3_
debug\cyfxapi.a"
CYU3LPP_A_PATH = "$(SDK_CORE_PATH)\firmware\u3p_firmware\lib\fx3_
debug\cyu3lpp.a"
CYU3THREADX_A_PATH = "$(SDK_CORE_PATH)\firmware\u3p_
firmware\lib\fx3_debug\cyu3threadx.a"
```

- For the above paths, set the paths for the installed environment.
- In particular, the `libgcc.a` part often needs to be checked (because it contains detailed version numbers in the path).

1. Unzip the `best_ez_usb3.zip` to a location of your choice.
2. Modify the `Makefile`.
3. Start a command prompt (DOS screen) and move the current directory to `best_ez_usb3`, which was unpacked above.
4. Build the source by `cs-make` ENTER. When the build is finished successfully, `best_uac_usbd.img` is created. This can be written to the EX-USB FX3 to be used as the USB Peripheral Module device.

You may clear related objects with cs-make clean <enter>.

An overview of the included source code is shown below:

List of source codes

File name	Description
cyfx_gcc_startup.S	Start-up routine (using code from Cypress EZ-USB SDK).
cyfxbulkdscr.c	Processing related to descriptors.
cyfxtx.c	Tools related to EZ-USB (using code from Cypress EZ-USB SDK).
cyfx_usb_peripheral.c	Descripting the USB peripheral behavior of EZ-USB.
cyfx_usb_peripheral.h	Header file.
uac_bestorm_soe.c	SOE communication processing module.
uac_bestorm_soe.h	Header file.
uac_ezusb_lib.c	EZUSB implementation dependencies (mainly endpoint settings).
uac_ezusb_lib.h	Header file
uac_ezusb_ser_recv.c	Buffer processing for serial reception.
uac_ezusb_ser_recv.h	Header file.