Tizen Content Screening

Test Specification

Document version 1.1.2

1 Contents

1.1 Document History

Version	Date	Reason
1.0.0	11/05/2012	First draft from McAfee
1.0.1	11/07/2012	Added more test cases for stub funtions
1.0.2	11/08/2012	Correct some test statement and wording
1.0.3	11/12/2012	Add library replacement test cases, add test contents and test guide.
1.0.4	01/26/2013	Add license
1.0.5	07/23/2013	Add multiple bytes test case
1.1.0	03/28/2014	Add test cases for TCSGetversion API
1.1.1	06/24/2014	Add testcases for scan cancel, async scan and TCSGetInfo API changes
1.1.2	07/30/2014	Fix backward compatible issues for scan API

1.2 References

Ref	Document	Issue	Title
[1]	Tizen Content Screening API Specification	1.0.2	Tizen Content Screening API Specification

1.3 Glossary and definitions

API Application Programming Interface

TCS Tizen Content Screening

2 Purpose and Scope

The overall purpose of this document is to describe the conformance test cases for the Tizen Content Screening framework.

This document shall include:

1. Tizen Content Screening Test Configuration
2. Test Case procedures

The scope of this document is the Tizen Content Screening Foundation API functions that are common to all Content Screening implementations. Specific functions of the Content Screening plug-in are not tested. All TCS implementations must include and meet the test cases defined in this document.

TCS validation plug-in

- A security plug-in for Tizen Content Screening Framework validation. Includes the functionalities required for the validation, including scanning, and conforms to the TCS framework API specification.

3 Component Description

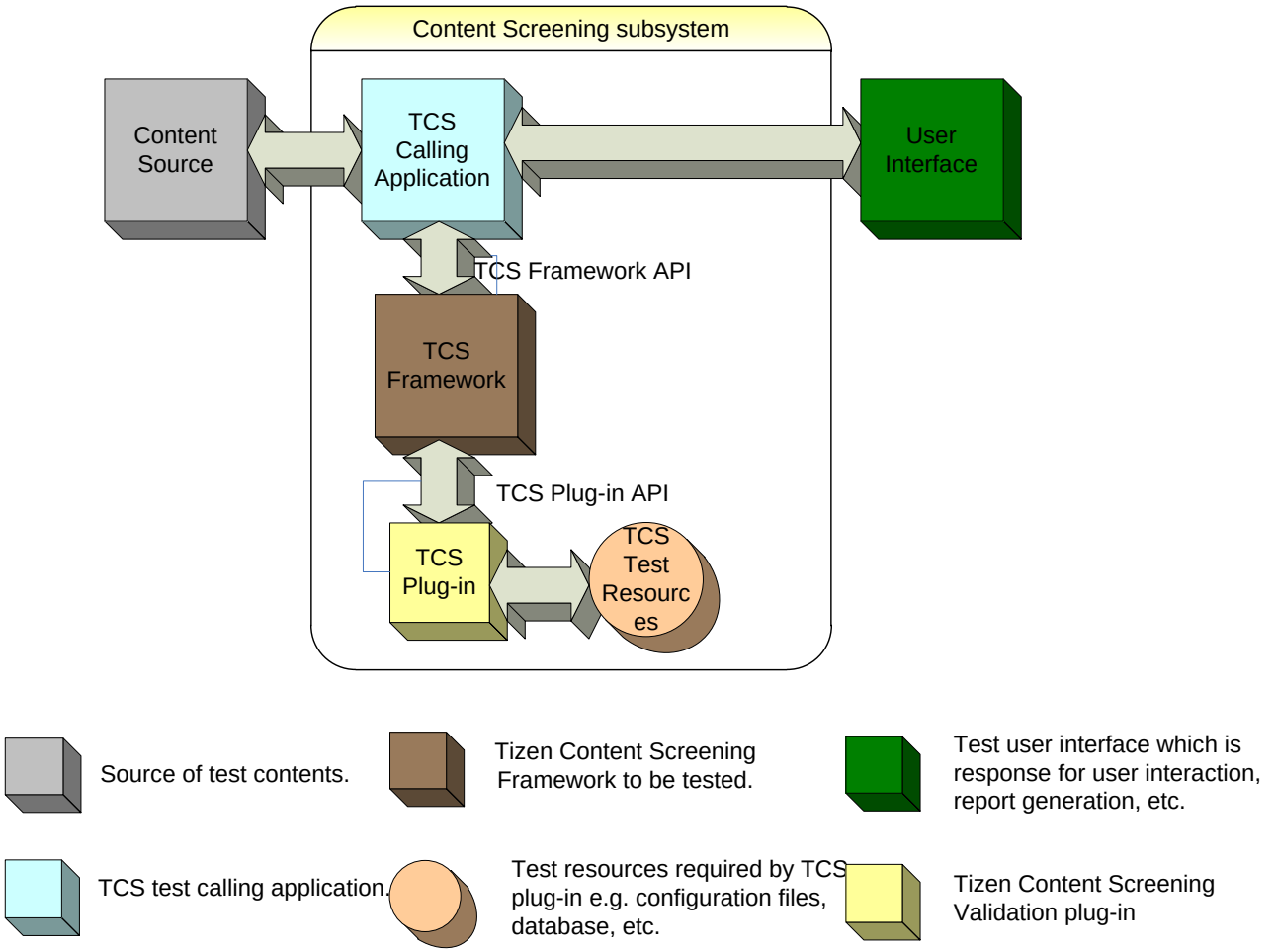

Figure 1: Tiezn Content Screening Architecture

The TCS framework (here on will be referenced as “tizen content screening library”, “TCS library”) works (interacts) with the calling application through an interface identified as one of the main elements to be tested in this test specification.

TCS plug-in is the content screening function implementation interfacing the TCS framework via Tizen Content screening Framework API functions.

“TCS Test Resources” is the resource data used by the TCS plug-in for test purposes (e.g. configurations, signatures for test content, etc.).

For testing purposes, the TCS library can be interchanged with a test tool. Rather than using software to analyze the content from the calling application and return the result of the scanning, a test tool is used to return the desired result matching the input content and the test case under execution. The test tool should also analyze the request from the calling application implementation to check that the process and the implementation is successful in both of the following ways:

1. The input content received from the calling application triggers the scanning process according to the content type (the request to the engine/test tool could be different if the content is an e-mail, a HTML document, a binary file, etc.).
2. The result of the scanning APIs must be understood by the calling application which should take an action with the received content:
   a) Do nothing if the content is correct, or
   b) Request more information from the TCS library (by the test tool).

This test tool can generate a log file with the result of the performed tests for checking purposes.

4 Test Environment Description

The test environment used is on Tizen platform.

The following requirements apply to all test cases defined in this document:

1. Any resources required by Tizen Content Screening subsystem in runtime should be installed in the test environment.
2. Test samples required by test suite should be installed in the test environment.

5 Test Cases Specifications

5.1 Test Case TC_SEC_CS_TCSLibraryOpen_0001

TC_SEC_CS_TCSLibraryOpen_0001	TCS library interface initialization test.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSLibraryClose(TCSLIB_HANDLE hLib);`	

Test Objectives: This test case verifies that the calling application can correctly initialize the TCS library handle.	
Test pre-conditions: validation plug-in	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Verify the API return `value`.	
Test PASS Condition: Step 2 should return valid TCSLIB_HANDLE instead of INVALID_TCSLIB_HANDLE.	
Test Clean-up procedure: Call TCSLibraryClose() with the TCS library handle returned by TCSLibraryOpen().	

5.2 Test Case TC_SEC_CS_TCSLibraryOpen_0002

TC_SEC_CS_TCSLibraryOpen_0002	TCS library interface initialization test.
API Function(s) covered: `TCSLIB_HANDLE TCSLibraryOpen(void);`	
Test Objectives: This test case verifies that the calling application can get proper error when there is no TCS plugin found in system.	
Test pre-conditions: Stub functions	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Verify it returns INVALID_TCSLIB_HANDLE.	

TC_SEC_CS_TCSLibraryOpen_0002	TCS library interface initialization test.
Test PASS Condition: Step 2 should return valid INVALID_TCSLIB_HANDLE.	
Test Clean-up procedure: None.	

5.3 Test Case TC_SEC_CS_TCSLibraryOpen_0003

TC_SEC_CS_TCSLibraryOpen_0003	TCS library replacement test.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSLibraryClose(void);`	

Test Objectives: This test case verifies that the calling application can get always get the latest TCS library API call after close/open.	
Test pre-conditions: Stub functions	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Verify it returns INVALID_TCSLIB_HANDLE. 3. Copy validation plug-in to "/opt/usr/share/sec_plugin" 4. Call `TCSLibraryOpen()`. 5. Verify it returns valid TCS library handle. 6. Call `TCSLibraryClose()`.	
Test PASS Condition: Step 2 should pass. Step 5 should pass.	
Test Clean-up procedure: None.	

5.4 Test Case TC_SEC_CS_TCSLibraryOpen_0004

TC_SEC_CS_TCSLibraryOpen_0004	TCS library replacement test.

TC_SEC_CS_TCSLibraryOpen_0004	TCS library replacement test.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSLibraryClose(void);`	

Test Objectives: This test case verifies that the calling application can get always get the latest TCS library API call after close/open.	
Test pre-conditions: validation plug-in	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Verify it returns valid TCS library handle. 3. Delete validation plug-in from "/opt/usr/share/sec_plugin" 4. Call `TCSLibraryClose()`. 5. Call `TCSLibraryOpen()`. 6. Verify it returns `INVALID_TCSLIB_HANDLE`.	
Test PASS Condition: Step 2 should pass. Step 6 should pass.	
Test Clean-up procedure: None.	

5.5 Test Case TC_SEC_CS_TCSGetLastError_0001

TC_SEC_CS_TCSGetLastError_0001	Stub TCS function error return.
API Function(s) covered: `int TCSGetLastError(TCSLIB_HANDLE hLib);`	
Test Objectives: This test case verifies that the calling application can get proper error code from TCS stub functions.	
Test pre-conditions: Stub functions	
Test Procedure:	

TC_SEC_CS_TCSGetLastError_0001	Stub TCS function error return.
1. Call `TCSGetLastError()` with INVALID_TCSLIB_HANDLE. 2. Verify it returns TCS_ERROR_NOT_IMPLEMENTED.	
Test PASS Condition: Step 2 should passed.	
Test Clean-up procedure: None.	

5.6 Test Case TC_SEC_CS_TCSLibraryClose_0001

TC_SEC_CS_TCSLibraryClose_0001	TCS library interface finalization.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSLibraryClose(TCSLIB_HANDLE hLib);`	

Test Objectives: This test case verifies that the calling application can close the TCS library handle.	
Test pre-conditions: validation plug-in.	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Verify that the API returns valid `TCSLIB_HANDLE` instead of `INVALID_TCSLIB_HANDLE`. 3. Call `TCSLibraryClose()` with the TCS library handle returned by `TCSLibraryOpen()`. 4. Verify that the return value of the `TCSLibraryClose()` is `0`.	
Test PASS Condition: Step 2 should pass verification. Step 4 should pass verification.	
Test Clean-up procedure: No specific cleanup required.	

5.7 Test Case TC_SEC_CS_TCSScanData_0001

TC_SEC_CS_TCSScanData_0001	Call TCS interface to scan benign content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case tests the scan request interface and verifies that the TCS interface returns the expected return value in the case of benign content data.

Test pre-conditions:

validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier and set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.8 Test Case TC_SEC_CS_TCSScanData_0002

TC_SEC_CS_TCSScanData_0002	Call TCS interface to scan benign content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case tests the scan request interface and verifies that the TCS interface returns the expected return value in the case of benign content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign, TCS_SA_SCANONLY as the scan action ID, TCS_DTYPE_UNKNOWN as the data type identifier and `pfCallback` is not `NULL`.
3. Verify that the `pfCallback` is not called.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.9 Test Case TC_SEC_CS_TCSScanData_0003

TC_SEC_CS_TCSScanData_0003	Call TCS interface to scan infected content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to scan infected content data

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected data, `TCS_SA_SCANONLY` as the scan action ID, TCS_DTYPE_UNKNOWN as the data type identifier and set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.10 Test Case TC_SEC_CS_TCSScanData_0004

TC_SEC_CS_TCSScanData_0004	Call TCS interface to scan infected content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to scan infected content data

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called and that the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.11 Test Case TC_SEC_CS_TCSScanData_0005

TC_SEC_CS_TCSScanData_0005	Call TCS interface to scan benign HTML formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the TCS interface returns the expected return value when it is called to scan benign HTML formatted content data

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign HTML formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_HTML` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.12 Test Case TC_SEC_CS_TCSScanData_0006

TC_SEC_CS_TCSScanData_0006	Call TCS interface to scan benign HTML formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the TCS interface returns the expected return value when it is called to scan benign HTML formatted content data

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call TCSLibraryOpen `()`.
2. Call `TCSScanData()` with a buffer filled with benign HTML formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_HTML` as the data type identifier and `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is not called.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.13 Test Case TC_SEC_CS_TCSScanData_0007

TC_SEC_CS_TCSScanData_0007	Call TCS interface to scan infected HTML formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to scan infected HTML formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected HTML formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_HTML` as the data type identifier and set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.14 Test Case TC_SEC_CS_TCSScanData_0008

TC_SEC_CS_TCSScanData_0008	Call TCS interface to scan infected HTML formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to scan infected HTML formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected HTML formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_HTML` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called, the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.15 Test Case TC_SEC_CS_TCSScanData_0009

TC_SEC_CS_TCSScanData_0009	Call TCS interface to scan benign URL formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned from the interface when it is called to scan benign URL formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign URL formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_URL` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.16 Test Case TC_SEC_CS_TCSScanData_0010

TC_SEC_CS_TCSScanData_0010	Call TCS interface to scan benign URL formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned from the interface when it is called to scan benign URL formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign URL formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_URL` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is not called.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.17 Test Case TC_SEC_CS_TCSScanData_0011

TC_SEC_CS_TCSScanData_0011	Call TCS interface to scan infected URL formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan infected URL formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected URL formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_URL` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.18 Test Case TC_SEC_CS_TCSScanData_0012

TC_SEC_CS_TCSScanData_0012	Call TCS interface to scan infected URL formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan infected URL formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected URL formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_URL` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called, the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.19 Test Case TC_SEC_CS_TCSScanData_0013

TC_SEC_CS_TCSScanData_0013	Call TCS interface to scan benign Email formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign Email formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign Email formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_EMAIL` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.20 Test Case TC_SEC_CS_TCSScanData_0014

TC_SEC_CS_TCSScanData_0014	Call TCS interface to scan benign Email formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign Email formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign Email formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_EMAIL` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is not called.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.21 Test Case TC_SEC_CS_TCSScanData_0015

TC_SEC_CS_TCSScanData_0015	Call TCS interface to scan infected Email formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan infected Email formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected Email formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_EMAIL` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.22 Test Case TC_SEC_CS_TCSScanData_0016

TC_SEC_CS_TCSScanData_0016	Call TCS interface to scan infected Email formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan infected Email formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected Email formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_EMAIL` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called, the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.23 Test Case TC_SEC_CS_TCSScanData_0017

TC_SEC_CS_TCSScanData_0017	Call TCS interface to scan benign phone number formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign phone number formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign phone number formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_PHONE` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.24 Test Case TC_SEC_CS_TCSScanData_0018

TC_SEC_CS_TCSScanData_0018	Call TCS interface to scan benign phone number formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign phone number formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign phone number formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_PHONE` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is not called.
4. Verify that the return value of TCSScanData`()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.25 Test Case TC_SEC_CS_TCSScanData_0019

TC_SEC_CS_TCSScanData_0019	Call TCS interface to scan infected phone number formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan infected phone number formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected phone number formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_PHONE` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.26 Test Case TC_SEC_CS_TCSScanData_0020

TC_SEC_CS_TCSScanData_0020	Call TCS interface to scan infected phone number formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan infected phone number formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected phone number formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_PHONE` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called, the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.27 Test Case TC_SEC_CS_TCSScanData_0021

TC_SEC_CS_TCSScanData_0021	Call TCS interface to scan benign Java code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign Java code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign Java code formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_JAVA` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.28 Test Case TC_SEC_CS_TCSScanData_0022

TC_SEC_CS_TCSScanData_0022	Call TCS interface to scan benign Java code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign Java code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign Java code formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_JAVA` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is not called.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.29 Test Case TC_SEC_CS_TCSScanData_0023

TC_SEC_CS_TCSScanData_0023	Call TCS interface to scan infected Java code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan infected Java code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected Java code formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_JAVA` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.30 Test Case TC_SEC_CS_TCSScanData_0024

TC_SEC_CS_TCSScanData_0024	Call TCS interface to scan infected Java code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan infected Java code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected Java code formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_JAVA` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called, the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.31 Test Case TC_SEC_CS_TCSScanData_0025

TC_SEC_CS_TCSScanData_0025	Call TCS interface to scan benign JavaScript code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign JavaScript code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign JavaScript code formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_JAVAS` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.32 Test Case TC_SEC_CS_TCSScanData_0026

TC_SEC_CS_TCSScanData_0026	Call TCS interface to scan benign JavaScript code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan benign JavaScript code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign Java code formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_JAVAS` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is not called.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.33 Test Case TC_SEC_CS_TCSScanData_0027

TC_SEC_CS_TCSScanData_0027	Call TCS interface to scan infected JavaScript code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan infected JavaScript code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected JavaScript code formatted data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_JAVAS` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.34 Test Case TC_SEC_CS_TCSScanData_0028

TC_SEC_CS_TCSScanData_0028	Call TCS interface to scan infected JavaScript code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan infected JavaScript code formatted content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected JavaScript code formatted data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_JAVAS` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called, the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.35 Test Case TC_SEC_CS_TCSScanData_0029

TC_SEC_CS_TCSScanData_0029	Call TCS interface to scan benign text content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when interface is called to scan benign text content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign text data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_TEXT` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.36 Test Case TC_SEC_CS_TCSScanData_0030

TC_SEC_CS_TCSScanData_0030	Call TCS interface to scan benign text content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when interface is called to scan benign text content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with benign text data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_TEXT` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is not called.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is `0`.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.37 Test Case TC_SEC_CS_TCSScanData_0031

TC_SEC_CS_TCSScanData_0031	Call TCS interface to scan infected text content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan infected text content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected text data, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_TEXT` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.38 Test Case TC_SEC_CS_TCSScanData_0032

TC_SEC_CS_TCSScanData_0032	Call TCS interface to scan infected text content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan infected text content data.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected text data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_TEXT` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called. The malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.39 Test Case TC_SEC_CS_TCSScanData_0033

TC_SEC_CS_TCSScanData_0033	Call TCS interface to scan content data infected by multiple malware.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan content data infected by multiple malware.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with data infected by multiple malwares, `TCS_SA_SCANONLY` as the scan action ID, and `TCS_DTYPE_UNKNOWN` as the data type identifier. Set `pfCallback` to `NULL`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.40 Test Case TC_SEC_CS_TCSScanData_0034

TC_SEC_CS_TCSScanData_0034	Call TCS interface to scan content data infected by multiple malware.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan content data infected by multiple malware.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with data infected by multiple malwares, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier and where `pfCallback` is not `NULL`.
3. Verify that `pfCallback` is called, the malware name or variant name is as expected and the severity/behaviour is as expected.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.41 Test Case TC_SEC_CS_TCSScanData_0035

TC_SEC_CS_TCSScanData_0035	Call TCS interface to repair infected content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies the expected return value is returned when TCS interface is called to repair infected content data

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.42 Test Case TC_SEC_CS_TCSScanData_0036

TC_SEC_CS_TCSScanData_0036	Call TCS interface to repair infected HTML formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to repair infected HTML formatted content data.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected HTML formatted data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_HTML` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.43 Test Case TC_SEC_CS_TCSScanData_0037

TC_SEC_CS_TCSScanData_0037	Call TCS interface to repair infected URL formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair infected URL formatted content data.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected URL formatted data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_URL` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.44 Test Case TC_SEC_CS_TCSScanData_0038

TC_SEC_CS_TCSScanData_0038	Call TCS interface to repair infected Email formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair infected Email formatted content data.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected Email formatted data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_EMAIL` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.45 Test Case TC_SEC_CS_TCSScanData_0039

TC_SEC_CS_TCSScanData_0039	Call TCS interface to repair infected phone number formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to repair infected phone number formatted content data.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected phone number formatted data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_PHONE` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.46 Test Case TC_SEC_CS_TCSScanData_0040

TC_SEC_CS_TCSScanData_0040	Call TCS interface to repair infected Java code formatted content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to repair infected Java code formatted content data.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected Java code formatted data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_JAVA` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.47 Test Case TC_SEC_CS_TCSScanData_0041

TC_SEC_CS_TCSScanData_0041	Call TCS interface to repair infected text content data.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair infected text content data.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected text data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_TEXT` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is 0.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.48 Test Case TC_SEC_CS_TCSScanData_0042

TC_SEC_CS_TCSScanData_0042	Call TCS interface to repair content data infected by multiple malware.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair content data infected by multiple malware.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with test multiple malware data, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the content data is repaired by comparing with prepared clean data.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.49 Test Case TC_SEC_CS_TCSScanData_0043

TC_SEC_CS_TCSScanData_0043	Return -1 in pfCallback.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when pfCallback returns -1 to the TCS library.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with test malware data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier and where `pfCallback` is not `NULL`.
3. Return `-1` in `pfCallback` when the detection notify occurrs.
4. Verify that the return value of `TCSScanData()` is `-1`.
5. Call `TCSGetLastError()`.
6. Verify that the error code returned from `TCSGetLastError()` is `TCS_ERROR_CANCELLED`.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 4 should pass verification.

Step 6 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.50 Test Case TC_SEC_CS_TCSScanData_0044

TC_SEC_CS_TCSScanData_0044	Call TCS interface to repair infected content data when repair functionality is not implemented in TCS library.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when calling the TCS interface to repair infected content data where the repair functionality is not implemented in the TCS library.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required to be not implemented in validation plug-in for this test case.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled with infected data, `TCS_SA_SCANREPAIR` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanData()` is `-1`.
4. Call `TCSGetLastError()` to get error code.
5. Verify that the error code returned by `TCSGetLastError()` is `TCS_ERROR_NOT_IMPLEMENTED`.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.51 Test Case TC_SEC_CS_TCSScanData_0045

TC_SEC_CS_TCSScanData_0045	Call TCS data scan interface with invalid library instance handle.

API Function(s) covered:

```
int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `-1` is returned when an invalid scanner instance handle is passed to data scan interface.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSScanData()` with an invalid library instance handle `INVALID_TCSLIB_HANDLE`.
2. Verify that the return value of `TCSScanData()` is `-1`.

Test PASS Condition:

Step 2 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.52 Test Case TC_SEC_CS_TCSScanData_0046

TC_SEC_CS_TCSScanData_0046	Concurrency TCS data scan test.

API Function(s) covered:

```
int TCSScanData(TCSSCAN_HANDLE hScan, TCSScanParam *pParam,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `TCSScanData()` can be correctly handled by multiple scanner instance handles in multiple threads.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Create multiple threads to execute from 2 to 10.
2. Call `TCSLibraryOpen()`.

TC_SEC_CS_TCSScanData_0046	Concurrency TCS data scan test.

3. Call `TCSScanData()` with an infected buffer with test malware data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier.
4. Verify that the return value of `TCSScanData()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.
8. Repeat 2 ~ 9 with different parameter for `TCSScanData()`, other test samples: (html, url, email, phone number, Java code, text) and respective data type identifier.

Test PASS Condition:

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.53 Test Case TC_SEC_CS_TCSScanData_0047

TC_SEC_CS_TCSScanData_0047	Concurrency TCS data clean test.

API Function(s) covered:

```
int TCSScanData(TCSSCAN_HANDLE hScan, TCSScanParam *pParam,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `TCSScanData()` can be correctly handled by multiple scanner instance handles in multiple threads.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Create multiple threads to execute from 2 to 10.
2. Call `TCSLibraryOpen()`.
3. Call `TCSScanData()` with an infected buffer with test malware data, `TCS_SA_SCANREPAIR` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier.
4. Verify that the return value of `TCSScanData()` is `0`.

TC_SEC_CS_TCSScanData_0047	Concurrency TCS data clean test.

5. Verify that the infected data is repaired by comparing with the respective clean buffer data if the input data is supposed to be infected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.
8. Repeat 2 ~ 9 with different parameter for `TCSScanData()`, other test samples: (html, url, email, phone number, java code, text) and respective data type identifier.

Test PASS Condition:

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.54 Test Case TC_SEC_CS_TCSScanData_0048

TC_SEC_CS_TCSScanData_0048	Compress flag TCS data clean test.

API Function(s) covered:

```
int TCSScanData(TCSSCAN_HANDLE hScan, TCSScanParam *pParam,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `TCSScanData()` can correctly scan clean data with compress flag enabled.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled by clean data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier, set compress flag to `1`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the no malware found.
5. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

TC_SEC_CS_TCSScanData_0048	Compress flag TCS data clean test.
Test Clean-up procedure: No specific cleanup required.	

5.55 Test Case TC_SEC_CS_TCSScanData_0049

TC_SEC_CS_TCSScanData_0049	Compress flag TCS data clean test.

API Function(s) covered:
`int TCSScanData(TCSSCAN_HANDLE hScan, TCSScanParam *pParam,`
`TCSScanResult *pResult);`	

Test Objectives: This test case verifies that `TCSScanData()` can correctly scan clean data with compress flag disabled.	
Test pre-conditions: For validation plug-in only. Repairing functionality is required in validation plug-in.	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Call `TCSScanData()` with a buffer filled by clean data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier, set compress flag to `0`. 3. Verify that the return value of `TCSScanData()` is `0`. 4. Verify that the no malware found. 5. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.	
Test PASS Condition: Step 3 should pass verification. Step 4 should pass verification.	
Test Clean-up procedure: No specific cleanup required.	

5.56 Test Case TC_SEC_CS_TCSScanData_0050

TC_SEC_CS_TCSScanData_0050	Compress flag TCS data test.

TC_SEC_CS_TCSScanData_0050	Compress flag TCS data test.

API Function(s) covered:

```
int TCSScanData(TCSSCAN_HANDLE hScan, TCSScanParam *pParam,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `TCSScanData()` can correctly detect malware with compress flag enabled.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled by test malware, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier, set compress flag to `1`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that the infected data is repaired by comparing with the respective clean buffer data if the input data is supposed to be infected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.57 Test Case TC_SEC_CS_TCSScanData_0051

TC_SEC_CS_TCSScanData_0051	Compress flag TCS data test.

API Function(s) covered:

```
int TCSScanData(TCSSCAN_HANDLE hScan, TCSScanParam *pParam,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `TCSScanData()` cannot correctly detect malware without compress flag enabled.

Test pre-conditions:

TC_SEC_CS_TCSScanData_0051	Compress flag TCS data test.

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanData()` with a buffer filled by test malware, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier, set compress flag to `0`.
3. Verify that the return value of `TCSScanData()` is `0`.
4. Verify that no malware found.
5. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.58 Test Case TC_SEC_CS_TCSScanData_0052

TC_SEC_CSSTUB_TCSScanData_0052	Stub TCS function error return.

API Function(s) covered:

```
int TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that the calling application can get proper error code from TCS stub functions.

Test pre-conditions:

Stub functions

Test Procedure:

1. Call `TCSScanData()` with INVALID_TCSLIB_HANDLE.
2. Verify it returns -1.

Test PASS Condition:

Step 2 should passed.

Test Clean-up procedure:

None.

5.59 Test Case TC_SEC_CS_TCSScanDataAsync_0053

TC_SEC_CS_TCSScanDataAsync_0053	Concurrency Async TCS data scan test.

API Function(s) covered:

```
int TCSScanDataAsync(TCSSCAN_HANDLE hScan, TCSScanParam *pParam);
```

Test Objectives:

This test case verifies that `TCSScanDataAsync()` can be correctly handled by multiple scanner instance handles in multiple threads.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Create multiple threads to execute below steps from 3 to 6.
2. Call `TCSLibraryOpen()`.
3. Call `TCSScanDataAsync()` with an infected buffer with test malware data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier.
4. Verify that the return value of `TCSScanDataAsync()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.60 Test Case TC_SEC_CS_TCSScanDataAsync_0054

TC_SEC_CS_TCSScanDataAsync_54	Return -1 in pfCallback.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
```

```
int TCSScanDataAsync(TCSLIB_HANDLE hLib, TCSScanParam *pParam);
```

TC_SEC_CS_TCSScanDataAsync_54	Return -1 in pfCallback.

```
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when pfCallback returns -1 to the TCS library.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Create multiple threads to execute below steps from 3 to 7.
2. Call `TCSLibraryOpen()`.
3. Call `TCSScanDataAsync()` with a buffer filled with test malware data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier and where `pfCallback` is not `NULL`.
4. Return `-1` in `pfCallback` when the detection notify occurrs.
5. Verify that the return value of `TCSScanDataAsync()` is `0`.
6. Verify the callback pParam is set to NULL.
7. Call `TCSGetLastError()`.
8. Verify that the error code returned from `TCSGetLastError()` is `TCS_ERROR_CANCELLED`.
9. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 5, 6 and 8 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.61 Test Case TC_SEC_CS_TCSScanFile_0001

TC_SEC_CS_TCSScanFile_0001	Call TCS interface to scan a benign file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case tests the scan request interface and verifies that the TCS interface returns the expected return value in the case of a benign file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.62 Test Case TC_SEC_CS_TCSScanFile_0002

TC_SEC_CS_TCSScanFile_0002	Call TCS interface to scan an infected file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to scan an infected file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected file, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.63 Test Case TC_SEC_CS_TCSScanFile_0003

TC_SEC_CS_TCSScanFile_0003	Call TCS interface to scan a benign HTML file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the TCS interface returns the expected return value when it is called to scan a benign HTML file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign HTML file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_HTML` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.64 Test Case TC_SEC_CS_TCSScanFile_0004

TC_SEC_CS_TCSScanData_0004	Call TCS interface to scan an infected HTML file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to scan an infected HTML file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected HTML file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_HTML` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.65 Test Case TC_SEC_CS_TCSScanFile_0005

TC_SEC_CS_TCSScanFile_0005	Call TCS interface to scan a benign URL within a file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned from the interface when it is called to scan a benign URL within a file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign URL file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_URL` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.66 Test Case TC_SEC_CS_TCSScanFile_0006

TC_SEC_CS_TCSScanFile_0006	Call TCS interface to scan an infected URL within a file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan an infected URL within a file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected URL file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_URL` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.67 Test Case TC_SEC_CS_TCSScanFile_0007

TC_SEC_CS_TCSScanFile_0007	Call TCS interface to scan a benign Email file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a benign Email file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign Email file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_EMAIL` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.68 Test Case TC_SEC_CS_TCSScanFile_0008

TC_SEC_CS_TCSScanFile_0008	Call TCS interface to scan an infected Email file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan an infected Email file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected Email file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_EMAIL` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.69 Test Case TC_SEC_CS_TCSScanFile_0009

TC_SEC_CS_TCSScanFile_0009	Call TCS interface to scan a benign phone number within a file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a benign phone number within a file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign phone number file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_PHONE` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.70 Test Case TC_SEC_CS_TCSScanFile_0010

TC_SEC_CS_TCSScanFile_0010	Call TCS interface to scan an infected phone number within a file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan an infected phone number within a file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected phone number file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_PHONE` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.71 Test Case TC_SEC_CS_TCSScanFile_0011

TC_SEC_CS_TCSScanFile_0011	Call TCS interface to scan a benign Java file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a benign Java file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_JAVA` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.72 Test Case TC_SEC_CS_TCSScanFile_0012

TC_SEC_CS_TCSScanFile_0012	Call TCS interface to scan an infected Java file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan an infected Java file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_JAVA` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.73 Test Case TC_SEC_CS_TCSScanFile_0013

TC_SEC_CS_TCSScanFile_0013	Call TCS interface to scan a benign text file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when interface is called to scan a benign text file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign text file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_TEXT` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.74 Test Case TC_SEC_CS_TCSScanFile_0014

TC_SEC_CS_TCSScanFile_0014	Call TCS interface to scan an infected text file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan an infected text file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call TCSLibraryOpen().
2. Call `TCSScanFile()` with an infected text file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_TEXT` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.75 Test Case TC_SEC_CS_TCSScanFile_0015

TC_SEC_CS_TCSScanFile_0015	Call TCS interface to scan a file infected by multiple malware.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a file infected by multiple malware.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a file path of a file infected by multiple malware, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.76 Test Case TC_SEC_CS_TCSScanFile_0016

TC_SEC_CS_TCSScanFile_0016	Call TCS interface to repair an infected file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to repair an infected file.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected file path, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.77 Test Case TC_SEC_CS_TCSScanFile_0017

TC_SEC_CS_TCSScanFile_0017	Call TCS interface to repair an infected HTML file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the TCS interface is called to repair an infected HTML file.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected HTML file path, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_HTML` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.78 Test Case TC_SEC_CS_TCSScanFile_0018

TC_SEC_CS_TCSScanFile_0018	Call TCS interface to repair an infected URL within a file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair an infected URL within a file.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected URL file path, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_URL` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.79 Test Case TC_SEC_CS_TCSScanFile_0019

TC_SEC_CS_TCSScanFile_0019	Call TCS interface to repair an infected Email file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair an infected Email file.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected Email file path, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_EMAIL` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.80 Test Case TC_SEC_CS_TCSScanFile_0020

TC_SEC_CS_TCSScanFile_0020	Call TCS interface to repair an infected phone number within a file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to repair an infected phone number within a file.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected phone number file path, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_PHONE` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.81 Test Case TC_SEC_CS_TCSScanFile_0021

TC_SEC_CS_TCSScanFile_0021	Call TCS interface to repair an infected Java file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to repair an infected Java file.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected Java file path, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_JAVA` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.82 Test Case TC_SEC_CS_TCSScanFile_0022

TC_SEC_CS_TCSScanFile_0022	Call TCS interface to repair an infected text file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair an infected text file.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected text file path, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_TEXT` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.83 Test Case TC_SEC_CS_TCSScanFile_0023

TC_SEC_CS_TCSScanFile_0023	Call TCS interface to repair a file infected by multiple malware.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to repair a file infected by multiple malware.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected file path of the file infected by multiple malware, `TCS_SA_SCANREPAIR` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is 0.
4. Verify that the content file is repaired by comparing with prepared clean file.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.84 Test Case TC_SEC_CS_TCSScanFile_0024

TC_SEC_CS_TCSScanFile_0024	Call TCS interface to repair an infected file where the repair functionality is not implemented in the TCS library.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when calling the TCS interface to repair an infected file where the repair functionality is not implemented in the TCS library.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required to be not implemented in validation plug-in for this test case.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected file path and `TCS_DTYPE_TEXT` as the data type identifier, and `TCS_SA_SCANREPAIR` as the scan action ID.
3. Verify that the return value of `TCSScanFile()` is `-1`.
4. Call `TCSGetLastError()` to get error code.
5. Verify that the error code returned by `TCSGetLastError()` is `TCS_ERROR_NOT_IMPLEMENTED`.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.85 Test Case TC_SEC_CS_TCSScanFile_0025

TC_SEC_CS_TCSScanFile_0025	Call TCS file scan interface with an invalid library instance handle.

API Function(s) covered:

```
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult );
```

Test Objectives:

This test case verifies that `-1` is returned when an invalid scanner instance handle is passed to the TCS file scan interface.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSScanFile()` with an invalid TCS scanner instance handle `INVALID_TCSLIB_HANDLE`.
2. Verify that the return value of `TCSScanFile()` is `-1`.

Test PASS Condition:

Step 2 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.86 Test Case TC_SEC_CS_TCSScanFile_0026

TC_SEC_CS_TCSScanFile_0026	Concurrency TCS file scan test.

API Function(s) covered:

```
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `TCSScanFile()` can be correctly handled by multiple scanner instance handles in multiple threads.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Create multiple threads to execute from 2 to 10.
2. Call `TCSLibraryOpen()`.
3. Call `TCSScanFile()` with an infected file, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier.
4. Verify that the return value of `TCSScanFile()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.
8. Repeat 2 ~ 9 with different parameter for `TCSScanFile()`, other test samples: (html, url, email, phone number, Java code, text) and respective data type identifier.

Test PASS Condition:

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.87 Test Case TC_SEC_CS_TCSScanFile_0027

TC_SEC_CS_TCSScanFile_0027	Concurrency TCS file clean test.

API Function(s) covered:

```
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that `TCSScanFile()` can be correctly handled by multiple scanner instance handles in multiple threads.

Test pre-conditions:

For validation plug-in only.

Repairing functionality is required in validation plug-in.

Test Procedure:

1. Create multiple threads to execute from 2 to 10.
2. Call `TCSLibraryOpen()`.
3. Call `TCSScanFile()` with an infected file, `TCS_SA_SCANREPAIR` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier.
4. Verify that the return value of `TCSScanFile()` is `0`.
5. Verify that the file is repaired by comparing with the respective clean file if the input file is supposed to be infected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.
8. Repeat 2 ~ 9 with different parameter for `TCSScanFile()`, other test samples: (html, url, email, phone number, java code, text) and respective data type identifier.

Test PASS Condition:

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.88 Test Case TC_SEC_CS_TCSScanFile_0028

TC_SEC_CS_TCSScanFile_0028	Call TCS interface to scan a benign JavaScript file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a benign JavaScript file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_JAVAS` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.89 Test Case TC_SEC_CS_TCSScanFile_0029

TC_SEC_CS_TCSScanFile_0029	Call TCS interface to scan an infected JavaScript file.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan an infected JavaScript file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_JAVAS` as the data type identifier.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.90 Test Case TC_SEC_CS_TCSScanFile_0030

TC_SEC_CS_TCSScanFile_0030	Call TCS interface to scan a benign file with compress flag.

TC_SEC_CS_TCSScanFile_0030	Call TCS interface to scan a benign file with compress flag.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a benign file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKOWN` as the data type identifier, and compress flag to `1`.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.91 Test Case TC_SEC_CS_TCSScanFile_0031

TC_SEC_CS_TCSScanFile_0031	Call TCS interface to scan an infected file with compress flag.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);

int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);

int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected value is returned when the interface is called to scan an infected file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with an infected Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKNOWN` as the data type identifier, and compress flag to `1`.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.92 Test Case TC_SEC_CS_TCSScanFile_0032

TC_SEC_CS_TCSScanFile_0032	Call TCS interface to scan a benign file with compress flag.

TC_SEC_CS_TCSScanFile_0032	Call TCS interface to scan a benign file with compress flag.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a benign file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKOWN` as the data type identifier, and compress flag to `0`.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.93 Test Case TC_SEC_CS_TCSScanFile_0033

TC_SEC_CS_TCSScanFile_0033	Call TCS interface to scan an infected file with compress flag.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
```

TC_SEC_CS_TCSScanFile_0033	Call TCS interface to scan an infected file with compress flag.

```
                TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when the interface is called to scan a infected file.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Call `TCSLibraryOpen()`.
2. Call `TCSScanFile()` with a benign Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKOWN` as the data type identifier, and compress flag to `0`.
3. Verify that the return value of `TCSScanFile()` is `0`.
4. Verify that the number of the detected malware is `0`.
5. Call `pfFreeResult()` to release the resource returned by TCS library.
6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 3 should pass verification.

Step 4 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.94 Test Case TC_SEC_CS_TCSScanFile_0034

TC_SEC_CS_TCSScanFile_0034	Stub TCS function error return.

API Function(s) covered:

```
int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,
                int iDataType, int iAction, int iCompressFlag,
                TCSScanResult *pResult);
```

Test Objectives:

This test case verifies that the calling application can get proper error code from TCS stub functions.

TC_SEC_CS_TCSScanFile_0034	Stub TCS function error return.
Test pre-conditions: Stub functions	
Test Procedure: 1. Call `TCSScanFile()` with INVALID_TCSLIB_HANDLE. 2. Verify it returns -1.	
Test PASS Condition: Step 2 should passed.	
Test Clean-up procedure: None.	

5.95 Test Case TC_SEC_CS_TCSScanFile_0035

TC_SEC_CS_TCSScanFile_0035	Scan multiple bytes infected file.

API Function(s) covered:
`int TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName,`
`int iDataType, int iAction, int iCompressFlag,`
`TCSScanResult *pResult);`	

Test Objectives: This test case verifies that the calling application can get proper result from TCS plugin when infected file is encoded by multiple bytes encoding.	
Test pre-conditions: For validation plug-in only.	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Call `TCSScanFile()` with a benign Java file path, `TCS_SA_SCANONLY` as the scan action ID and `TCS_DTYPE_UNKOWN` as the data type identifier, and compress flag to `0`. 3. Verify that the return value of `TCSScanFile()` is `0`. 4. Verify that the number of the detected malware is `1`. 5. Call `pfFreeResult()` to release the resource returned by TCS library. 6. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.	
Test PASS Condition: Step 3 should passed.	

TC_SEC_CS_TCSScanFile_0035	Scan multiple bytes infected file.
Step 4 should passed.	
Test Clean-up procedure: None.	

5.96 Test Case TC_SEC_CS_TCSScanFileEx_0036

TC_SEC_CS_TCSScanFileEx_005336	Return -1 in pfCallback.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFileEx(TCSLIB_HANDLE hLib, char const *pszFileName, int iDataType,
                int iAction, int iCompressFlag, void *pPrivate, TCSCbFunc
                pfCallBack, TCSScanResult *pResult);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when pfCallback returns -1 to the TCS library.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

8. Call `TCSLibraryOpen()`.
9. Call `TCSScanFileEx()` with a buffer filled with test malware data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier and where `pfCallback` is not `NULL`.
10. Return `-1` in `pfCallback` when the detection notify occurrs.
11. Verify that the return value of `TCSScanFileEx()` is `-1`.
12. Call `TCSGetLastError()`.
13. Verify that the error code returned from `TCSGetLastError()` is `TCS_ERROR_CANCELLED`.
14. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 4 should pass verification.

Step 6 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.97 Test Case TC_SEC_CS_TCSScanFileAsync_0037

TC_SEC_CS_TCSScanFileAsync_0037	Concurrency Async TCS file scan test.

API Function(s) covered:

```
int TCSScanFileAsync(TCSLIB_HANDLE hLib, char const *pszFileName,
                     int iDataType, int iAction, int iCompressFlag,
                     void *pPrivate, TCSCbFunc pfCallBack);
```

Test Objectives:

This test case verifies that `TCSScanFileAsync()` can be correctly handled by multiple scanner instance handles in multiple threads.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Create multiple threads to execute below steps from 3 to 6.
2. Call `TCSLibraryOpen()`.
3. Call `TCSScanFileAsync()` with an infected buffer with test malware data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier.
4. Verify that the return value of `TCSScanFileAsync()` is `0`.
5. Verify that the number of the detected malware is as expected, the malware name or variant name is as expected and the severity/behaviour is as expected.
6. Call `pfFreeResult()` to release the resource returned by TCS library.
7. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 4 should pass verification.

Step 5 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.98 Test Case TC_SEC_CS_TCSScanFileAsync_0038

TC_SEC_CS_TCSScanFileAsync_38	Return -1 in pfCallback.

TC_SEC_CS_TCSScanFileAsync_38	Return -1 in pfCallback.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSScanFileAsync(TCSLIB_HANDLE hLib, char const *pszFileName,
                     int iDataType, int iAction, int iCompressFlag,
                     void *pPrivate, TCSCbFunc pfCallBack);
int TCSLibraryClose(TCSLIB_HANDLE hLib);
```

Test Objectives:

This test case verifies that the expected return value is returned when pfCallback returns -1 to the TCS library.

Test pre-conditions:

For validation plug-in only.

Test Procedure:

1. Create multiple threads to execute below steps from 3 to 8.
2. Call `TCSLibraryOpen()`.
3. Call `TCSScanFileAsync()` with a buffer filled with test malware data, `TCS_SA_SCANONLY` as the scan action ID, `TCS_DTYPE_UNKNOWN` as the data type identifier and where `pfCallback` is not `NULL`.
4. Return `-1` in `pfCallback` when the detection notify occurs.
5. Verify that the return value of `TCSScanFileAsync()` is `0`.
6. Verify the callback pParam is set to NULL.
7. Call `TCSGetLastError()`.
8. Verify that the error code returned from `TCSGetLastError()` is `TCS_ERROR_CANCELLED`.
9. Call `TCSLibraryClose()` with the TCS library handle returned by the `TCSLibraryOpen()`.

Test PASS Condition:

Step 5, 6 and 8 should pass verification.

Test Clean-up procedure:

No specific cleanup required.

5.99 Test Case TC_SEC_CS_TCSGetVersion_0001

TC_SEC_CS_TCSGetVersion_0001	Get framework and plugin version info.

API Function(s) covered:

```
TCSLIB_HANDLE TCSLibraryOpen(void);
int TCSGetVersion(TCSLIB_HANDLE hLib, TCSVerInfo *pVerInfo);
```

TC_SEC_CS_TCSGetVersion_0001	Get framework and plugin version info.
Test Objectives: This test case verifies that the calling application can get version information from the library.	
Test pre-conditions: For validation plug-in only.	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Call `TCSGetVersion()with a valid handle and TCSVerInfo pointer variable`. 3. Verify that the return value of `TCSGetVersion()` is `0`. 4. Verify string length of plugin version is greater than zero. 5. Verify string length of framework version is greater than zero. 6. Verify the framework version matches in format and value with TCS_FRAMEWORK_VERSION.	
Test PASS Condition: Step 3 should pass. Step 4 should pass. Step 5 should pass. Step 6 should pass.	
Test Clean-up procedure: None.	

5.100 Test Case TC_SEC_CS_TCSGetVersion_0002

TC_SEC_CS_TCSGetVersion_0002	Get framework and plugin version info.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSGetVersion(TCSLIB_HANDLE hLib, TCSVerInfo *pVerInfo);`	

Test Objectives: This test case verifies that the calling application can get error from library.	
Test pre-conditions: For validation plug-in only.	
Test Procedure: 1. Call `TCSGetVersion()with a invalid handle and TCSVerInfo pointer variable`.	

TC_SEC_CS_TCSGetVersion_0002	Get framework and plugin version info.
2. Verify that the return value of `TCSGetVersion()` is `-1`.	
Test PASS Condition: Step 2 should pass.	
Test Clean-up procedure: None.	

5.101 Test Case TC_SEC_CS_TCSGetVersion_0003

TC_SEC_CS_TCSGetVersion_0003	Get framework and plugin version info.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSGetVersion(TCSLIB_HANDLE hLib, TCSVerInfo *pVerInfo);`	

Test Objectives: This test case verifies that the calling application can get error from the library.	
Test pre-conditions: For validation plug-in only.	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Call `TCSGetVersion()with a valid handle and NULL for TCSVerInfo`. 3. Verify that the return value of `TCSGetVersion()` is `-1`.	
Test PASS Condition: Step 3 should pass.	
Test Clean-up procedure: None.	

5.102 Test Case TC_SEC_CS_TCSGetInfo_0001

TC_SEC_CS_TCSGetInfo_0001	Get Meta info.

TC_SEC_CS_TCSGetInfo_0001	Get Meta info.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSGetInfo(TCSLIB_HANDLE hLib, char *pszInfo);`	

Test Objectives: This test case verifies that the calling application can get version information from the library.	
Test pre-conditions: For validation plug-in only.	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Call `TCSGetInfo()with a valid handle and pszInfo pointer variable`. 3. Verify that the return value of `TCSGetInfo()` is `0`. 4. Verify string length of pszInfo is greater than zero.	
Test PASS Condition: Step 3 should pass. Step 4 should pass.	
Test Clean-up procedure: None.	

5.103Test Case TC_SEC_CS_TCSGetInfo_0002

TC_SEC_CS_TCSGetInfo_0002	Check if API handles invalid input

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSGetInfo(TCSLIB_HANDLE hLib, char *pszInfo);`	

Test Objectives: This test case verifies that the calling application can get error from library.	
Test pre-conditions: For validation plug-in only.	
Test Procedure: 1. Call `TCSGetInfo()with a invalid handle and pszInfo pointer variable`. 2. Verify that the return value of `TCSGetInfo()` is `-1`.	

TC_SEC_CS_TCSGetInfo_0002	Check if API handles invalid input
Test PASS Condition: Step 2 should pass.	
Test Clean-up procedure: None.	

5.104Test Case TC_SEC_CS_TCSGetInfo_0003

TC_SEC_CS_TCSGetInfo_0003	Check if API handles invalid input.

API Function(s) covered:
`TCSLIB_HANDLE TCSLibraryOpen(void);`
`int TCSGetInfo(TCSLIB_HANDLE hLib, char *pszInfo);`	

Test Objectives: This test case verifies that the calling application can get error from the library.	
Test pre-conditions: For validation plug-in only.	
Test Procedure: 1. Call `TCSLibraryOpen()`. 2. Call `TCSGetInfo()with a valid handle and NULL for pszInfo`. 3. Verify that the return value of `TCSGetInfo()` is `-1`.	
Test PASS Condition: Step 3 should pass.	
Test Clean-up procedure: None.	

6 Test Guide

To run test cases, we need to have:

- TCS plug-in for test purpose
- Test contents
- Test cases
- TCS security framework

Test cases need to be compiled with TCS security framework. A TCS plug-in need to be created which can detect the test contents as expected. All test contents, test cases and test TCS plug-in will be provided as a test suite along with accordinate script file which will automate the test process.

7 Test Contents

Sample Name	Status	Content Type	Malware Name	Variant Name	Severity Class	Behavior Class
tcs-testfile-0.buf	clean	Unknown	n/a	n/a	n/a	n/a
tcs-testfile-0.class	clean	Java	n/a	n/a	n/a	n/a
tcs-testfile-0.email	clean	Email	n/a	n/a	n/a	n/a
tcs-testfile-0.html	clean	HTML	n/a	n/a	n/a	n/a
tcs-testfile-0.js	clean	JavaScript	n/a	n/a	n/a	n/a
tcs-testfile-0.phone	clean	Phone Number	n/a	n/a	n/a	n/a
tcs-testfile-0.txt	clean	Text	n/a	n/a	n/a	n/a
tcs-testfile-0.url	clean	URL	n/a	n/a	n/a	n/a
tcs-testfile-0.z	clean	Archived	n/a	n/a	n/a	n/a
tcs-testfile-0.multiple	clean	Unknown	n/a	n/a	n/a	n/a
tcs-testfile-1.buf	infected	unknown	Malware-fortest-1.6.0	Variant-fortest-1.6.0	TCS_SC_USER	TCS_BC_LEVEL1
tcs-testfile-1.class	infected	Java	Malware-fortest-1.7.0	Variant-fortest-1.7.0	TCS_SC_USER	TCS_BC_LEVEL0
tcs-testfile-1.email	infected	Email	Malware-fortest-1.2.0	Variant-fortest-1.2.0	TCS_SC_TERMINAL	TCS_BC_LEVEL2
tcs-testfile-1.html	infected	HTML	Malware-fortest-1.0.0	Variant-fortest-1.0.0	TCS_SC_USER	TCS_BC_LEVEL0
tcs-testfile-1.js	infected	JavaScript	Malware-fortest-1.8.0	Variant-fortest-1.8.0	TCS_SC_USER	TCS_BC_LEVEL2
tcs-testfile-1.phone	infected	Phone Number	Malware-fortest-1.3.0	Variant-fortest-1.3.0	TCS_SC_TERMINAL	TCS_BC_LEVEL3
tcs-testfile-1.txt	infected	Text	Malware-fortest-1.4.0	Variant-fortest-1.4.0	TCS_SC_TERMINAL	TCS_BC_LEVEL4
tcs-testfile-1.url	infected	URL	Malware-fortest-	Variant-fortest-	TCS_SC_USER	TCS_BC_LEVEL1

			1.1.0	1.1.0		
tcs-testfile-1.z	infected	Archived	Malware-fortest-1.9.0	Variant-fortest-1.9.0	TCS_SC_USER	TCS_BC_LEVEL2
tcs-testfile-1.multiple	infected	Unknown	Malware-fortest-1.6.0	Variant-fortest-1.6.0	TCS_SC_USER	TCS_BC_LEVEL1
			Malware-fortest-1.5.0	Variant-fortest-1.5.0	TCS_SC_USER	TCS_BC_LEVEL0