AUTHORIZED DOCUMENTATION

NOVELL® NSS AUDIT SDK

Novell®
Developer Kit

July 18, 2009

www.novell.com

Legal Notices

Novell, Inc., makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc., makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the Novell International Trade Services Web page (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc., has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed on the Novell Legal Patents Web page (http://www.novell.com/company/legal/patents/) and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see the Novell Documentation Web page (http://www.novell.com/documentation).

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

This guide describes the NSS Audit functions.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to www.novell.com/documentation/feedback.html and enter your comments there.

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

Overview

1

The NSS file system audit framework enables application to be tracked and log all file and directory access events. Auditing facilitates the process of monitoring NSS file system activities to ensure that transactions are accurate and that confidential information is secure. It monitors events from all protocols like NCP, Novell AFP, and Novell CIFS.

The NSS Auditing engine collects information about any security-relevant events. The audit records can be examined to determine whether any violation of the security policies has been committed and by whom.

1.1 Components of NSS Audit

The NSS audit includes Linux kernel components and user-space components.

1.1.1 Kernel Components

vigil.ko

This module is the core framework of the NSS auditing engine.

User-space applications interface with this module to create and maintain auditing clients. This module writes audit records to audit trail files and and also provides event filtering methods.

vigil_nss.ko

This module provides a connection for internal NSS events to communicate to vigil.ko.

This module interfaces with pmd.ko to obtain information about the NDS client that is performing the audit operation.

vigil_ncp.ko

This module provides a connection for internal NCP engine events to communicate to vigil.ko.

Currently, these events are limited to “file open” and “file close” events; which the NCP engine handle internally without initiating calls to the NSS engine. This module interfaces with pmd.ko to obtain information about the NDS client that is performing the audit operation.

vigil_cifs.ko

This module provides a connection for internal CIFS engine events to communicate to vigil.ko.

Currently, these events are limited to “file open” and “file close” events; which the CIFS engine handle internally without initiating calls to the NSS engine. This module interfaces with pmd.ko to obtain information about the NDS client that is performing the audit operation.

pmd.ko

This module is used by the NCP engine and the CIFS engine to pass auditing data elements to kernel components including vigil_nss, vigil_cifs, and vigil_ncp.

1.1.2 User Space Components

The user space components are:

libvigil.so and libvigil.a

A dynamic shared library and a static-linked library are provided to facilitate the construction of user-space applications which interface with vigil.ko to create and maintain auditing clients.

The library shows the proper method of interfacing with vigil.ko (from a 'C' perspective) when porting those methods to other languages.

v2ld

This is a vigil to LAF daemon. It is an auditing client which parses NSS Auditing events to an XML format, and forwards them through the “Linux Auditing Framwork” (LAF). The v2ld deamon is not recommended for moderate or high volume OES2 server production systems. It is not installed, by default.

vdump

This is an auditing client application which parses NSS duditing events and outputs them to the console, or to a file, in several formats. A sample application intended for developers to write their own auditing clients. By default, it is not automatically installed. It is included with source code in the NSS Auditing SDK package.

1.2 Auditing Client

The NSS Auditing engine allows creation of auditing clients. List of NSS operations which generate the audit records.

Table 1-1 *Auditing Client Operations*

Operation	Files	Directories
Create	Yes	Yes
Close	Yes	Yes
Delete	Yes	Yes
Link	Yes	Yes
Metadata Modification	Yes	Yes
Open	Yes	Yes

Operation	Files	Directories
Rename	Yes	Yes
Add Trustee	Yes	Yes
Remove Trustee	Yes	Yes
Set Inherited Rights	Yes	Yes

On enabling the auditing engine on one NSS volume, it becomes active for all the NSS volumes on the system.

NOTE: The auditing engine is enabled automatically by applications which implement auditing clients.

When an auditing client is defined by an auditing application, audit trail files are automatically created by the auditing engine. As an audited event occurs, the auditing engine creates an auditing record, and writes that record to the current audit file of each auditing client.

The auditing engine continues to write audit records to audit trail file until the associated auditing client is terminated, or the audit file becomes full. The auditing client can be configured to "roll" to a new audit trail file, if the current auditing trail file becomes full. The maximum size of an audit trail file is specified by the application which creates the auditing client.

1.2.1 Audit Record Header

Each audit record has a header which includes:

Table 1-2 *Header of Audit Record*

Field	Description
sign	Tag that indicates the beginning of the audit record. Currently, records are signed with a newline, followed by the characters "VIGIL". (ie: "\nVIGIL") See VIGIL_SIGN in vigil.h.
length	Length of the audit record in bytes, including the header.
type	Audit Record Type.
recNo	Record number specific to the auditing client.
pid	Process ID of the (Linux) thread performing the audited operation.
time	System time-stamp of when the event occurred.

Audit Record Type

Audit records are produced due to internal audit engine activities, as well as by the NSS, NCP and CIFS engines. Each engine has been assigned an audit record type, as shown in the table below:

Table 1-3 *Record Type Description*

Type	vigil.h Macro	Description
0	VIGIL_T_VIGIL	Internal auditing engine event. Auditing engine management and configuration.
1	(DEPRECATED) VIGIL_T_NEB	NEB (Novell Event Bus) event.
2	VIGIL_T_NCP	Specific NCP events which are handled internally by the NCP engine (and would otherwise go un-audited by the NSS engine). These include “Open File” and “Close File” events.Events reported by the NSS engine. These events include delete, create, open, close, rename, link, modify(attributes), add trustee, remove trustee and set inherited rights. These events are logged for files, directories, volumes, pools, etc.
3	VIGIL_T_NSS	Events reported by the NSS engine. These events include delete, create, open, close, rename, link, modify(attributes), add trustee, remove trustee and set inherited rights. These events are logged for files, directories, volumes, pools, etc.
4	VIGIL_T_CIFS	Specific CIFS events which are handled internally by the CIFS engine (and would otherwise go un-audited by the NSS engine). These include “Open File” and “Close File” events.

1.3 Planning

An auditing client is created by an application with Linux root privleges, which calls the libvigil function LIBVIGIL_SYS_ClientOpen.

- On calling the libvigil function, the application provides a unique client name and key string. It also supplies the following bit flags:

Table 1-4 *Bit Flags*

Bit	vigil.h Macro	Description
1	VIGIL_CLIENT_F_ROLLOUTPUTFILE	Indicates that when an audit trail file becomes full, a new audit trail file is created, where new audit records are captured.
2	VIGIL_CLIENT_F_EXCLUDE_SIGIO_PIDS (DEPRECATED)	
3	VIGIL_CLIENT_F_SHARE	Allows multiple user-space applications to share the same auditing client.
4	VIGIL_CLIENT_F_KEEP	The application runs in order to maintain, and eventually close, the auditing client. This bit allows the auditing client to be orphaned, while it continues to log auditing events to audit trail files.

- The application creates a Linux file system directory where the audit trail files are stored. Maximum file size, bytes, or of records can be indicated. The Linux file system directory must not be stored on an NSS volume.

- Each auditing client can have multiple user applications associated with it. Each of these auditing client users must have a unique name (unique within the client). The initial auditing client user name is also specified when calling this function, along with a userKey.

  When calling this function, userFlags should be set to VIGIL_USER_F_NONE, as they are not yet implemented.
- A Linux process ID is the process that parses the audit trail file. The pid sents SIGIO signals by vigil.ko each time new audit records are written to the client's audit trail file. This allows the process to perform real-time parsing of the audit trail file.

1.3.1 Audit Trail File

The auditing client creates an audit trail file. The name of the audit trial file is generated by `vigil.ko` and sends SIGIO signals to the pid as new records are written to the audit trail file. The auditing client opens the audit trail file and parses the records.

Calling the function LIBVIGIL_SYS_CurrentFileAndOffset returns the name of the current audit trail file and the location of next audit trail file. When the audit file reaches the maximum file size, records are written to the new file. Set the flag VIGIL_CLIENT_F_ROLLOUTPUTFILE to store audit trail files in the specified directory and you must delete the trail files when not required.

The auditing client notifies vigil.ko before it terminates.If the VIGIL_CLIENT_F_SHARE flag was set when the client was created, it is possible that there are multiple applications sharing the same auditing client. When the user use-count decrements to zero, (kept by vigil.ko) the auditing client is terminated and all associated structures in vigil.ko are deleted. In order to terminate the client with user count zero, LIBVIGIL_SYS_ClientClose function must call proper auditing client and key credentials.

1.4 Auditing with NCP

The auditing client audits the NCP requests for file-system operations on an NSS volume.

The NCP engine verifies that the calling NCP or NDS client has proper authorization to perform the requested file system function. On authorization, the request is passed from the NCP engine to a POSIX function call as the Linux root user. Before making the POSIX call, the NCP engine writes the NDS GUID in the Process metadata table (as supplied by pmd.ko). NCP also writes the client's connection number and task number into the metadata table. NCP makes POSIX calls representing the NCP requested operation. The process makes the POSIX call in the kernel space, and sent to the Linux kernel's “Virtual File System” (VFS). VFS dispatches the call to NSS file system.

NSS processes the request and returns the reply back to the call stack of the NCP engine, which formulates the NCP_REPLY. On replying to the request, NSS notifies vigil_nss.ko. The vigil_nss.ko module gets the associated data from pmd.ko, and then formulates an audit record of the event, and sends the record to vigil.ko. The vigil.ko module writes the record to each registered client's audit trail file (for client's filtering criteria). On writing to the various client audit trail files, vigil.ko sends SIGIO signals to each of the pids associated with clients with updated audit trail files.

In user-space, the auditing client application is invoked by the SIGIO signal. It then reads and processes the new records that have been placed in the audit trail file, and goes back to sleep.

Auditing application developers must be aware that not all (file system related) NCP requests generate corresponding POSIX requests. Specifically, for file open and file close requests. This is due to the NCP engine caching (POSIX) the file handles to frequently accessed files. If a file open

request NCP is serviced for which a cached handle exists, NCP uses the cached handle rather than making an unnecessary POSIX call. Hence, NCP may not call through to POSIX on a file close request, until the associated handle is no longer profitable to the NCP file handle cache. Because of this, there may be many file open, and file close requests that cannot be reported by the NSS engine. The purpose of the vigil_ncp.ko module is to receive events directly from the NCP engine. As the NCP engine processes file open and file close events from NCP clients, it reports this activity directly to the vigil_ncp.ko module, which constructs audit records indicating the NCP engine specific event and sends the record off to vigil.ko to be written into the client audit trail logs.

CIFS has very similar issues as NCP, and uses vigil_cifs.ko to log file open and file close events handled internally.

The AFP protocol, as implemented with OES, does not make POSIX calls. It calls directly to NSS (via zAPIs). AFP does not cache file handles (as does NCP and CIFS), so there are no AFP specific record types, as there are for NCP and CIFS. In addition, AFP passes the AFP/NDS client GUID in through the zAPI interface, where it is available to the vigil_nss.ko module. Hence, AFP auditing does not interface with pmd.ko.

Functions

2

This section contains the API reference for the Novell® NSS Audit SDK.

LIBVIGIL_FILE_WriteString

Writes a string into the offset of a file.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_FILE_WriteString(

        const char *filePath,
        long        offset,
        const char *string
      )
```

Parameters

filepath

(IN) Path of the target file.

offset

(IN) Specify the file offset value to write.

string

Specify the string to write to the target file

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	Parameter passed is NULL or invalid.
errno		When writing a string to a file, this function calls: ◆ fopen ◆ fseek ◆ fputs ◆ fflush ◆ fclose When any of the above calls fail, error codes are returned.

LIBVIGIL_FILE_ReadString

Reads a offset of a file.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_FILE_ReadString(

        const char *filePath,
        long        offset,
        char        *buff.
        size_t      bugsize,
        size_t      *buflen
      )
```

Parameters

filePath

(IN) Specify path of the target file.

offset

(IN) Specify the file offset value to read.

buf

(OUT) Location of the buffer to store the string.

bufsize

(IN) Specifies the buffer size in bytes.

bufLen

(OUT) Specifies the number of bytes to read from the buffer.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	Parameter passed is NULL or invalid.
errno		When reading a string from a file, this LIBVIGIL_FILE_ReadString calls: ◆ fopen ◆ fseek ◆ fread ◆ fclose When any of the above calls fail, error codes are returned.

LIBVIGIL_FILE_Size

Returns datastream size of the file.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_FILE_Size(

        const char *filePath,
        off_t      *size,
        )
```

Parameters

filePath

(IN) Specify path of the target file.

size

[OUT] Size of file in bytes.

NULL value can be passed, when only testing file access.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameter passed is NULL or invalid.
errno		LIBVIGIL_FILE_Size calls stat function.
		If the stat call fails, error codes are returned.

LIBVIGIL_LIST_Unlock

Unlock the list of files, previously locked by the LIBVIGIL_LIST_Lock function.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_Unlock(

        LIBVIGIL_LIST_T *list
        )
```

Parameters

list

(IN) List of files to unlock.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameter passed is NULL or invalid.

LIBVIGIL_LIST_Lock

List of files to be locked.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_Lock(

        LIBVIGIL_LIST_T *list
        )
```

Parameters

list

(IN) List of files to lock.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	Parameter passed is NULL or invalid.

LIBVIGIL_LIST_GetNodeEqualTo

Searches for a node in the list that is equal to the supplied data.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_GetNodeEqualTo(

        LIBVIGIL_LIST_T *list,
        void * data,
        LIBVIGIL_LIST_NODE_T **node
        )
```

Parameters

list

(IN) List objectto search .

data

(IN) Data to match

node

(OUT) Returns the matched node.

Return Values

0	VIGIL_SUCCESS	Success.
2	ENOENT	No matching node found. The reasons might be: ◆ No nodes exist in the specified list. ◆ The node compare function NodeCmpFn is NULL. ◆ No nodes are equal to the specified data. ◆ Node parameter is set to NULL.
22	EINVAL	Parameters passed are NULL or invalid.
	rCode	LIBVIGIL_LIST_GetNodeEqualTo calls NodeCmpFn function, which is associated with the specified list. If the NodeCmpFn fails, error codes are returned.

Remarks

LIBVIGIL_LIST_GetNodeEqualTo must be called only after the caller has acquired a spinlock on the specified list.

LIBVIGIL_LIST_GetNodeGreaterThan

Returns a node in the list that is greater than the supplied data.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_GetNodeGreaterThan(

        LIBVIGIL_LIST_T *list,
        LIBVIGIL_LIST_NODE_T *node,
        LIBVIGIL_LIST_NODE_T **cur
        )
```

Parameters

list

(IN) List to search

node

(IN) Reference node to compare.

cur

(OUT) Node which is greater than the reference node.

Return Values

0	VIGIL_SUCCESS	Success.
2	ENOENT	No greater node found. The reasons might be: ◆ No nodes exist in the specified list. ◆ The node compare function NodeCmpFn is NULL. ◆ No nodes are greater than the reference node. ◆ *cur parameter is set to NULL.
22	EINVAL	Parameters passed are NULL or invalid.
114	EALREADY	The reference node matches an existing node and the no duplicate records LIBVIGIL_LIST_F_NODUPS flag is set for the specified list. The *cur parameter is set to NULL.
	rCode	LIBVIGIL_LIST_GetNodeGreaterThan calls NodeCmpFn function, which is associated with the specified list. If the NodeCmpFn function fails, error codes are returned.

Remarks

LIBVIGIL_LIST_GetNodeGreaterThan must be called only after acquiring a spinlock on the specified list.This function assumes that the caller has acquired a spinlock on the specified list prior to calling this function.

LIBVIGIL_LIST_Iter

The nodes of a list are iterated by calling the callback_Fn function for each node in the specified list.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_Iter(

        LIBVIGIL_LIST_T   *list,
        int              (*callBack_Fn) (LIBVIGIL_LIST_NODE_T *, void *),
        void              *callBack_arg,
        int               *callBack_rcode
        )
```

Parameters

list

(IN) List to iterate.

callBack_Fn

(IN) Function to call for each node on the list. NULL can be specified.

callBack_arg

(IN) Auxiliary metadata to pass to the callBack_Fn. If not required, NULL can be specified.

callBack_rCode

(OUT) The last return code from the callBack_Fn function. If not required, NULL can be specified.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	Parameters passed are NULL or invalid.
	rCode	This function calls LIBVIGIL_LIST_Lock, which if returns an error, the LIBVIGIL_LIST_Iter immediately returns with the same error. This function calls the caller specified callBack_Fn. If the callBack_Fn reports an error, the error is also returned by this function.

Remarks

This function acquires a spinlock on the specified list. The callBack_Fn is called for each node in this "spinlock acquired" state.

The callBack_Fn might return ELOOP to stop the iteration of nodes, and cause this function to return VIGIL_SUCCESS. ELOOP will be returned via the *callBack_rCode parameter.

LIBVIGIL_LIST_NodeUnlink

A specified node is unlinked from its list.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_NodeUnlink(

        LIBVIGIL_LIST_NODE_T *node
        )
```

Parameters

node

(IN) List node to unlink.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
	rCode	This function calls LIBVIGIL_LIST_Lock, which if returns an error, the LIBVIGIL_LIST_NodeUnlink immediately returns with the same error.

LIBVIGIL_LIST_NodeLink

Links a node into the specified list.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_NodeLink(

        LIBVIGIL_LIST_T      *list,
        LIBVIGIL_LIST_NODE_T *node
        )
```

Parameters

list

(IN) List to which the node is linked.

node

(IN) Node to link.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
	rCode	This function calls the following functions: ◆ LIBVIGIL_LIST_Lock ◆ LIBVIGIL_LIST_GetNodeGreaterThan If any of the above function returns an error that cannot be handled, then the LIBVIGIL_LIST_NodeLink function returns with the same error.

LIBVIGIL_LIST_NodeCloseAll

Closes all listed nodes. All the nodes are unlinked, then closed.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_NodeCloseAll(

        LIBVIGIL_LIST_T *list,
        )
```

Parameters

list

(IN) List of nodes to close.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
	rCode	This function invokes the LIBVIGIL_LIST_NodeClose function. If LIBVIGIL_LIST_NodeClose returns an error that cannot be handled, then the LIBVIGIL_LIST_NodeCloseAll function returns with the same error.

LIBVIGIL_LIST_NodeClose

Closes the specified node. The node is unlinked, then closed.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_NodeClose(

        LIBVIGIL_LIST_NODE_T **node,
        )
```

Parameters

Node

(IN) Node to close.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
114	EALREADY	The specified node is already closed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_LIST_NodeUnlink ◆ LIBVIGIL_MEM_Free This function calls data_FreeFn to free data associated with the specified node. If any of the above function returns an error that cannot be handled, then LIBVIGIL_LIST_NodeClose function returns with the same error.

See Also

LIBVIGIL_LIST_NodeOpen

LIBVIGIL_LIST_NodeOpen

Creates and links a new linked-list node.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_NodeOpen(

        LIBVIGIL_LIST_T       *list,
        void                  *data,
        int                  (*data_FreeFn)(void **memPtr),
        LIBVIGIL_LIST_NODE_T  **node_OUT
        )
```

Parameters

list

(IN) Target list object to which this new node is associated.

data

(IN) Node metadata (payload). NULL can be specified.

data_FreeFn

(IN) Function call to free data when node is closed. NULL can be specified.

node

(OUT) Pointer to a newly created list node. NULL can be specified.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
114	EALREADY	The specified node is already closed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_LIST_NodeLink If any of the above function returns an error that cannot be handled, then LIBVIGIL_LIST_NodeOpen function returns with the same error.

See Also

LIBVIGIL_LIST_NodeClose

LIBVIGIL_LIST_Open

Creates a new linked list.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LIST_Open(

        LIBVIGIL_LIST_T  **list,
        int              (*NodeCmpFn)(
                                      LIBVIGIL_LIST_NODE_T *a,
                                      LIBVIGIL_LIST_NODE_T *b,
                                      int *result),
        uint32_t         flags,
        const char       *listName
        )
```

Parameters

list

(OUT) Handle to the list object.

NodeCmpFn

(IN) Node Compare Function to create list of nodes in order. NULL can be specified.

flags

(IN) List flags.

LIBVIGIL_LIST_F_NONE	No flags set
LIBVIGIL_LIST_F_NODUPS	No duplicate records allowed

listName

(IN) Name of list of nodes. NULL can be specified.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
114	EALREADY	The specified list is already opened.

rCode	This function calls the following functions: ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_MEM_StrDup If any of the above function returns an error that cannot be handled, then LIBVIGIL_LIST_Open function returns with the same error.

See Also

LIBVIGIL_LIST_Close

LIBVIGIL_ERR

A macro function used to report errors to the user.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_ERR(

        const char *fmtStr,
        ...
        )
```

Parameters

fmStr

(IN) Formatted string for an unformatted error report.

...

(IN) Variable argument list for fmStr.

Return Values

0	VIGIL_SUCCESS	Success.
	errno	This function calls fprintf, which if fails, LIBVIGIL_ERR returns an error.

LIBVIGIL_LOG_Err

Indicates an error condition.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LOG_Err(

        const char *file,
        const char *functionName,
        int         srcFileLine,
        const char *fmtStr,
        ...
        )
```

Parameters

file

(IN) File name of the C source code file. NULL can be specified.

functionName

(IN) Function name of the C source code in which error is reported. NULL can be specified.

srcFileLine

(IN) Line number in the C source code file in which error is reported. Zero value can be specified.

fmStr

(IN) Formatted string for an unformatted error report.

...

(IN) Variable argument list for fmStr.

Return Values

0	VIGIL_SUCCESS	Success.
	errno	This function calls fprintf, which if fails, LIBVIGIL_LOG_Err returns an error.

LIBVIGIL_LOG_Note

Indicates status or condition.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_LOG_Note(

        const char *fmtStr,
        ...
        )
```

Parameters

fmStr

(IN) Formatted string for an unformatted node report. NULL can be specified.

...

(IN) Variable argument list for fmStr.

Return Values

0	VIGIL_SUCCESS	Success.
	errno	This function calls fprintf, which if fails, LIBVIGIL_LOG_Note returns an error.

LIBVIGIL_MEM_Free

Frees memory and NULL pointer reference of that memory.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_MEM_Free(

        void **addr,
        )
```

Parameters

addr

(IN) Address of pointer which points to free memory.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.

LIBVIGIL_MEM_Alloc

Allocates and clears memory.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_MEM_Alloc(

        void   **addr,
        size_t size
        )
```

Parameters

addr

(OUT) Allocated memory reference.

size

(IN) Memory to allocate in bytes.

Return Values

0	VIGIL_SUCCESS	Success.
12	ENOMEM	Malloc function failed.
22	EINVAL	The parameters passed are NULL or invalid.

LIBVIGIL_MEM_StrDup

Duplicates source string and displays the location of the copied string.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_MEM_StrDup(

        const char *src,
        char       **dup
        )
```

Parameters

src

(IN) String to copy.

dup

(OUT) Location of the allocated or duplicate string.

Return Values

0	VIGIL_SUCCESS	Success.
12	ENOMEM	Malloc function failed.
22	EINVAL	The parameters passed are NULL or invalid.

LIBVIGIL_PARSE_FileValidate

Verifies the specified path and checks if it points to a valid Vigil audit file.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_PARSE_FileValidate(

        const char *filePath,
        )
```

Parameters

filePath

(IN) Path to the audit file.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
61	ENODATA	The specified path points to an empty file. Valid audit file must contain at least one audit record.
77	EBADFD	The path is not prefixed with a '/' character.
126	ENOKEY	The file is small in size and cannot contain a audit record.
129	EKEYREJECTED	The first record in the audit file does not contain valid audit record signature.
	rCode	This function calls the following functions: ◆ LIBVIGIL_FILE_Size ◆ LIBVIGIL_FILE_ReadString If any of the above function returns an error that cannot be handled, then LIBVIGIL_PARSE_FileValidate function returns with the same error.

LIBVIGIL_PARSE_RecValidate

Verifies that the specified record meets the minimal requirement to be a valid Vigil audit record. The minimal requirements are it verifies that the recVoid parameter is not NULL, the length of the record is large enough to hold the record header, or verifies that the audit record has a valid record signature.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_PARSE_RecValidate(

        void   *recVoid,
        size_t recLen
        )
```

Parameters

recVoid

(IN) Points to record buffer to be validated.

recLen

(IN) Length of recVoid buffer.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
53	EBADR	The record length is short and cannot contain the required header.
129	EKEYREJECTED	The first record in the audit file does not contain valid audit record signature.

LIBVIGIL_PARSE_RecElement_ByTypeInstance

Gets a reference to a record element of a instance of a element type.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_PARSE_RecElement_ByTypeInstance(

        void      *recData,
        uint16_t   recDataElements,
        uint16_t   elementType,
        uint16_t   instance,
        void     **elementPtr_out
        )
```

Parameters

recData

(IN) Pointer to the data portion of an audit record.

recDataElements

(IN) Number of elements in recData.

elementType

(IN) Type of element to return.

instance

(IN) The instance of the element (of the specified type) to reference.

elementPtr

(OUT) On success returns a pointer to the element which matches the above criteria.

Return Values

0	VIGIL_SUCCESS	Success.
2	ENOENT	Matching element is not found in the record.
22	EINVAL	The recData parameter passed is NULL or invalid.

LIBVIGIL_PARSE_Rec2TypeEventSeq

Parses the record type, event type, and sequence number from a vigil record.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_PARSE_Rec2TypeEventSeq(

        VIGIL_AUDIT_REC_T *rec,
        uint32_t           auditType_OUT,
        uint32_t           auditEvent_OUT,
        uint64_t           auditSequenceNo_OUT,
        )
```

Parameters

rec

(IN) Vigil record to parse.

auditType

(OUT) Vigil record type.

auditEvent

(OUT) Vigil record type event.

auditSequenceNo

(OUT) Vigil record type sequence number.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	Vigil record parsed is NULL or invalid.
34	ERANGE	Vigil record type is unknown.

LIBVIGIL_PARSE_AuditTypeStr

Returns string representing the audit record type.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_PARSE_AuditTypeStr(

        uint32_t  auditType,
        char      **auditTypeStr_OUT
        )
```

Parameters

auditType

(IN) Vigil record type.

auditTypeStr

(OUT) String representation of audit record type.

Return Values

0	VIGIL_SUCCESS	Success.
2	ENOENT	Vigil record type is unknown.

LIBVIGIL_PARSE_AuditTypeEventStateStr

Returns string representing the audit record type, event, and state.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_PARSE_AuditTypeEventStateStr(

        uint32_t  auditType,
        uint32_t  auditEvent,
        char      **auditTypeStr,
        char      **auditEventStr,
        char      **auditStateStr
        )
```

Parameters

auditType

(IN) Vigil record type.

auditEvent

(IN) Vigil record type event.

auditTypeStr

(OUT) String representation of audit record type.

auditEventStr

(OUT) String representation of audit record type event.

auditStateStr

(OUT) String representation of state of a audit record type. Strings returned by this functions include:

- SINGULAR - Indicates that the audit record is complete.
- [unknown] - Indicates that the audit record cannot be fully parsed by this function.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
2	ENOENT	Vigil record type or event is unknown.
	rCode	LIBVIGIL_PARSE_AuditTypeEventStateStr calls the LIBVIGIL_PARSE_EnterExitCheck function. If LIBVIGIL_PARSE_EnterExitCheck fails, error codes are returned.

Note

This function is deprecated, due to consolidation of pre and post events by the auditing engine.

LIBVIGIL_PROC_Pause

Pauses the process.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_PROC_Pause(void)
```

Return Values

0	VIGIL_SUCCESS	Success.
	rCode	LIBVIGIL_PROC_Pause calls pause function. If pause function fails, error codes are returned.

LIBVIGIL_SYS_UserPathAssemble

Gets the path of the user CONTROL file of the auditing client and user.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_UserPathAssemble(
              const char *clientName,
              const char *userName,
                    char **path_ALLOC
               )
```

Parameters

clientName

(IN) The auditing client as specified by LIBVIGIL_SYS_ClientOpen function.

userName

(IN) User name as specified by LIBVIGIL_SYS_ClientOpen function.

path_ALLOC

(OUT) Path to CONTROL file. The caller is responsible to free the allocated memory.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
	rCode	This function calls LIBVIGIL_MEM_Alloc. If LIBVIGIL_MEM_Alloc returns an error, this function returns the error codes.

Remarks

This is an example of how the CONTROL file path is created in newly allocated memory:

`/sys/audit/vigil/CLIENT_vigil_dump/USER_fred/CONTROL`

where the value of attributes clientName is *vigil_dump* and userName is *fred*.

SEE ALSO

- LIBVIGIL_SYS_ClientOpen
- LIBVIGIL_SYS_UserClose

LIBVIGIL_SYS_ClientPathAssemble

Gets the path of the CONTROL file of the auditing client.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_ClientPathAssemble(
              const char *clientName,
                    char **path_ALLOC
               )
```

Parameters

clientName

(IN) The auditing client as specified by LIBVIGIL_SYS_ClientOpen function.

path_ALLOC

(OUT) Path to the CONTROL file. The caller is responsible to free the allocated memory.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
	rCode	This function calls LIBVIGIL_MEM_Alloc. If LIBVIGIL_MEM_Alloc returns an error, this function returns the error codes.

Remarks

This is an example of how the CONTROL file path is created in newly allocated memory:

`/sys/audit/vigil/CLIENT_vigil_dump/CONTROL`

where the value of attributes clientName is *vigil_dump*.

SEE ALSO

LIBVIGIL_SYS_ClientOpen

LIBVIGIL_SYS_ClientPathVerify

Verifies the CONTROL file for the auditing client exists.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_ClientPathVerify(
              const char *clientName
              )
```

Parameters

clientName

(IN) The auditing client as specified by LIBVIGIL_SYS_ClientOpen function.

Return Values

0	VIGIL_SUCCESS	Success.
2	ENOENT	Client path is invalid.
22	EINVAL	The parameters passed are NULL or invalid.
	rCode	This function calls the following functions: ◆ LIBVIGIL_SYS_ClientPathAssemble ◆ LIBVIGIL_FILE_Size If any of the above function returns an error, then LIBVIGIL_SYS_ClientPathVerify returns with the same error.

SEE ALSO

LIBVIGIL_SYS_ClientOpen

LIBVIGIL_SYS_UserClose

Closes a vigil auditing client user.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_UserClose(
              const char *clientName,
              const char *userName,
              const char *userKey
              )
```

Parameters

clientName

(IN) Auditing client as specified by LIBVIGIL_SYS_ClientOpen function.

userName

(IN) Auditing client user as specified by LIBVIGIL_SYS_ClientOpen function.

userKey

(IN) User key as specified by LIBVIGIL_SYS_ClientOpen function. NULL can be specified if the user has no userKey, or this function is called by the same PID that called LIBVIGIL_SYS_ClientOpen to create the user.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	clientName and userName parameters passed are NULL or any invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_SYS_UserPathAssemble ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_FILE_WriteString If any of the above function returns an error, then LIBVIGIL_SYS_UserClose returns with the same error.

SEE ALSO

LIBVIGIL_SYS_ClientOpen

LIBVIGIL_SYS_ClientClose

Closes a vigil auditing client.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_ClientClose(
              const char *clientName,
              const char *clientKey
              )
```

Parameters

clientName

(IN) Auditing client as specified by LIBVIGIL_SYS_ClientOpen function.

userKey

(IN) Client key as specified by LIBVIGIL_SYS_ClientOpen function. NULL can be specified if the client has no clientKey, or this function is called by the same PID that called LIBVIGIL_SYS_ClientOpen to create the client.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	clientName parameter passed is NULL or any invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_SYS_ClientPathAssemble ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_FILE_WriteString If any of the above function returns an error, then LIBVIGIL_SYS_ClientClose returns with the same error.

SEE ALSO

LIBVIGIL_SYS_ClientOpen

LIBVIGIL_SYS_ClientOpen

Opens a vigil auditing client.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_SYS_ClientOpen(
              const char *clientName,
              const char *clientKey,
              unit32_t    clientFlags,
              const char *outputDirPath,
              size_t      outputSizeMax,
              unit64_t    outputRecordsLimit,
              const char *userName,
              const char *userKey,
              unit32_t    userFlags,
              int         outputSigioPid
              )
```

Parameters

clientName

(IN) Unique auditing client name.

clientKey

(IN) Client key used to validate furture client maintenance privileges. NULL can be specified if client key is not necessary.

clientFlags

(IN) Specifies various bit-value attributes for this auditing client. The bit value attributes are explained in the Notes section.

outputDirPath

(IN) Path to a directory where this client will place audit files.

outputSizeMax

(IN) Specifies the maximum size (in bytes) of the audit log files. Zero value can be specified to indicate no size limit.

outputRecordsLimit

(IN) Specifies the maximum number of records of audit log files. Zero value can be specified to indicate no size limit.

userName

(IN) A unique (per client) string used to identify a audit client user. NULL can be specified if the client has no initial users.

userKey

(IN) A string is used to validate audit client user maintenance privileges. Null can be specified if a user key is not required.

userFlags

(IN) Specifies various bit-value attributes for this auditing client-user. Currently, this field is not implemented. Callers must specify the value as VIGIL_USER_F_NONE or 0.

outputSigioPid

(IN) Specifies a PID to which SIGIO signals will be sent when vigil writes one or more records to the client's auditing log files.

This parameter is associated with the auditing client-user, if userName parameter is NULL, this parameter is ignored.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	clientName parameter passed is NULL or any invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_FILE_WriteString If any of the above function returns an error, then LIBVIGIL_SYS_ClientOpen returns with the same error.

Remarks

Bit values for the clientFlags parameter are defined as follows:

Bit	Macro Value	Description
1	VIGIL_CLIENT_F_ROLLOUTPUTFILE	Causes the audit engine to write records to a new audit log file when the current audit log file reaches its maximum size.
2	VIGIL_CLIENT_F_EXCLUDE_SIGIO_PIDS (DEPRECATED)	This bit must always be clear, set it to zero.
3	VIGIL_CLIENT_F_SHARE	Allow multiple auditing users to share the same auditing client.
4	VIGIL_CLIENT_F_KEEP	Override the automatic auditing client destruction when the last auditing client user terminates.

SEE ALSO

LIBVIGIL_SYS_ClientClose, LIBVIGIL_SYS_UserClose

LIBVIGIL_SYS_ParseTagValue

Parses 32-bit integers, 64-bit integers or string values which immediately follow a tag string from a memory buffer.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_ParseTagValue(
              const char *buf,
              const char *tag,
              unit32_t   *value32,
              unit64_t   *value64,
                    char **valueStr_ALLOC
               )
```

Parameters

buf

(IN) Buffer contains tags and values.

tag

(IN) Target tag to which a value is sought.

value32

(OUT) 32-bit unsigned integer representation of the tag value. NULL can be specified if this value is not required.

value64

(OUT) 64-bit unsigned integer representation of the tag value. NULL can be specified if this value is not needed.

valueStr_ALLOC

(OUT) String representation of the tag value. Null can be specified if this value is not needed. If value is specified, the caller is responsible to free this allocated memory.

Return Values

0	VIGIL_SUCCESS	Success.
2	ENOENT	Specified tag is not found
22	EINVAL	buf or tag parameter is NULL or any invalid parameters are passed.
	rCode	This function calls LIBVIGIL_MEM_Alloc function.
		If LIBVIGIL_MEM_Alloc returns an error, then LIBVIGIL_SYS_ParseTagValue returns with the same error.

LIBVIGIL_SYS_ClientInfo_Alloc

Returns information in XML format of the auditing client in the allocated memory.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_SYS_ClientInfo_Alloc(
              const char *clientName,
                    char **clientInfo_ALLOC,
                  size_t  *clientInfoLen
               )
```

Parameters

clientName

(IN) Specifies the name of the target vigil auditing client.

clientInfo_ALLOC

(OUT) Allocated memory which contains a snapshot of the specified auditing client's information. The caller is responsible to free this allocated memory.

clientInfoLen

(OUT) Length in bytes of the value specified in the clientInfo_ALLOC parameter. NULL can be specified if this value is not required.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	clientName parameter passed is NULL or invalid parameters are passed.
61	ENODATA	Client control file is empty.
	rCode	This function calls the following functions: ◆ LIBVIGIL_SYS_ClientPathAssemble ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_FILE_ReadString If any of the above function returns an error, then LIBVIGIL_SYS_ClientInfo_Alloc returns with the same error.

LIBVIGIL_SYS_CurrentFileAndOffset

Returns the file path and file offset where the auditing engine writes the next audit record for the auditing client.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_CurrentFileAndOffset(
              const char *clientName,
                    char **outputFile_ALLOC,
                 size_t  *outputFileOffset
               )
```

Parameters

clientName

(IN) Name of a vigil audit client, as per call to the LIBVIGIL_ClientOpen function.

outputFile_ALLOC

(OUT) Audit log file path for this client. NULL can be specified. If value is specified, the caller is responsible to free this allocated memory.

outputFileOffset

(OUT) Returns the client's current outputFile and the current (output) offset within that file.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	clientName parameter is NULL or invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_SYS_ClientInfo_Alloc ◆ LIBVIGIL_SYS_ParseTagValue If any of the above function returns an error, then LIBVIGIL_SYS_CurrentFileAndOffset returns with the same error.

LIBVIGIL_SYS_InitialFile

Returns the client's initial outputFile.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_SYS_InitialFile(
              const char *clientName,
                    char **outputFile_ALLOC
               )
```

Parameters

clientName

(IN) Name of a vigil audit client, as per call to the LIBVIGIL_ClientOpen function.

outputFile_ALLOC

(OUT) Initial audit log file path for this client. If value is specified, the caller is responsible to free this allocated memory.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	clientName parameter is NULL or invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_SYS_ClientInfo_Alloc ◆ LIBVIGIL_SYS_ParseTagValue If any of the above function returns an error, then LIBVIGIL_SYS_InitialFile returns with the same error.

LIBVIGIL_SYS_FilterFileImport

Causes vigil.ko to read-in a filter file, and apply it to the specified clientName.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_SYS_FilterFileImport(
              const char *clientName,
              const char *clientKey,
              const char *filterFilePath
              )
```

Parameters

clientName

(IN) Name of a vigil audit client, as per call to the LIBVIGIL_ClientOpen function.

clientKey

(IN) Client key as specified by LIBVIGIL_SYS_ClientOpen. NULL can be specified if the client has no clientKey parameter set, or this function is called by the same PID that called LIBVIGIL_SYS_ClientOpen to create the client.

filterFilePath

(IN) Path to filter file to be imported.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	clientName or filterFilePath parameter is NULL or invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_SYS_ClientPathAssemble ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_File_WriteString If any of the above function returns an error, then LIBVIGIL_SYS_FilterFileImport returns with the same error.

LIBVIGIL_TIME_TimeValCmp

Compares two time references.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_TIME_TimeValCmp(
              struct timeval *timeRef1,
              struct timeval *timeRef2,
              int            *result
              )
```

Parameters

timeRef1

(IN) First time reference to compare.

timeRef2

(IN) Second time reference to compare.

result

(OUT) Result of compare:First time reference to compare.

Equation	Description
result = 0	The time references are equal.
result > 0	The value in time reference 1 is greater than time reference 2.
result < 0	The value in time reference 1 is less than time reference 2.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	timeRef1 or timeRef2 parameter is NULL or invalid parameters are passed.

LIBVIGIL_GUID_GUIDStrToGUID

Convert a UTF GUID string into a VIGIL_GUID_T type.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_GUID_GUIDStrToGUID(
              const uint8_t *utfStr,
               VIGIL_GUID_T *guid
              );
```

Parameters

utfstr

(IN) utf GUID string.

guid

(OUT) 16-byte GUID value.

Remarks

The UTF GUID string must be in one of the following (BASE 16) formats:

- Traditional NetWare: "aaaaaaaa-bbbb-cccc-dd-ee-nnnnnnnnnnnn"
- Traditional Unix/Linux: "aaaaaaaa-bbbb-cccc-ddee-nnnnnnnnnnnn"

This function doesn't validate guid's value. It only checks its string format and converts it.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	utfStr or o_guid parameter is NULL or invalid parameters are passed.
34	ERANGE	Length of utfStr is not 36 or 37 characters long.
42	ENOMSG	Non-hex character found in utfStr, where a hex character was expected.
61	ENODATA	Non-dash '-' character found in utfStr parameter, where a dash was expected.
90	EMSGSIZE	Number of hex characters, between dashes, does not match expected utfStr format.

LIBVIGIL_GUID_StrGUIDToGuidMatch

Compares a UTF GUID string to a VIGIG_GUID_T object. The result is returned through the match parameter.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_GUID_StrGUIDToGuidMatch(
              const uint8_t       *guidStr,
              const VIGIL_GUID_T  *guidObj,
              int                 *match
              );
```

Parameters

guidStr

(IN) utf GUID string.

guidObj

(IN) 16-byte GUID value.

match

(OUT) VIGIL_TRUE or VIGIL_FALSE is returned through this parameter.

VIGIL_TRUE is returned if the guidStr matches the guidObj, VIGIL_FALSE if not.

Remarks

The UTF GUID string must be in one of the following (BASE 16) formats:

- Traditional NetWare: "aaaaaaaa-bbbb-cccc-dd-ee-nnnnnnnnnnnn"
- Traditional Unix/Linux: "aaaaaaaa-bbbb-cccc-ddee-nnnnnnnnnnnn"

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	guidStr or guidObj parameter is NULL or invalid parameters are passed.
34	ERANGE	Length of utfStr is not 36 or 37 characters long.
	rCode	This function calls LIBVIGIL_GUID_GUIDStrToGUID. If LIBVIGIL_GUID_GUIDStrToGUID returns an error, then LIBVIGIL_GUID_StrGUIDToGuidMatch returns with the same error.

LIBVIGIL_FILE_Size_2

Get the datastream size of the specified file.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_FILE_Size_2(
              const char    *filePath,
              off_t         *size
             );
```

Parameters

filePath

(IN) Filesystem path to target file.

size

(OUT) File size in bytes. NULL can be specified if value is not required (such as when testing file access only).

Remarks

This function differs from LIBVIGIL_FILE_Size, it opens the target file and counts each byte. LIBVIGIL_FILE_Size calls the stat function to get the file size.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	The parameters passed are NULL or invalid.
	rCode	This function calls the following functions: ◆ fopen ◆ fread ◆ fclose If any of the above function returns an error, then LIBVIGIL_FILE_Size_2 returns with the same error.

LIBVIGIL_FILE_ToMem

Places a snapshot image of a file into allocated memory. Caller must free the image using LIBVIGIL_MEM_Free.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_FILE_ToMem(
              const uint8_t   *filePath,
              unit8_t         *file_ALLOC,
              size_t          *fileSize
             )
```

Parameters

filePath

(IN) Filesystem path to target file.

file_ALLOC

(OUT) Memory image of file. Caller must free.

fileSize

(OUT) Size of memory image in bytes. NULL can be specified if value is not required by caller.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	filePath or file_ALLOC parameter passed is NULL or invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_FILE_Size_2 ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_FILE_ReadString If any of the above function returns an error, then LIBVIGIL_FILE_ToMem returns with the same error.

LIBVIGIL_STR_Stat

Returns various attributes of a specified ('C') '\0' terminated string.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_STR_Stat(
              const uint8_t  *str,
              size_t         *strLen,
              size_t         *strSize,
              uint8_t       **lastCharPtr,
              uint8_t       **termPtr
             )
```

Parameters

str

(IN) Target character string to analize.

strLen

(OUT) Length of string, not including the string termination character. NULL may be specified if value is not required by caller.

strSize

(OUT) The memory footprint size of the string (including the termination character).

lastCharPtr

(OUT) Returns the address of the last character in the string, does not include the string termination character. If the string is empty, NULL is returned.

termPtr

(OUT) Returns address of the string termination character. NULL can be specified if value is not required by caller.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	str parameter passed is NULL or invalid parameters are passed.

LIBVIGIL_DIR_PathPaste

Compiles a file system path string in allocated memory from two path fragments.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_DIR_PathPaste(
              const uint8_t  *pathSeg1,
              const uint8_t  *pathSeg2,
              uint8_t       **path_ALLOC
              )
```

Parameters

pathSeg1

(IN) Left or prefix portion of the segment.

pathSeg2

(IN) Right or postfix portion of the segment.

path_ALLOC

(OUT) Returns address of composit path in the allocated storage.Caller is responsible to free the allocated memory.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	str parameter passed is NULL or invalid parameters are passed.
	rCode	This function calls the following functions: ◆ LIBVIGIL_STR_Stat ◆ LIBVIGIL_MEM_Alloc If any of the above function returns an error, then LIBVIGIL_DIR_PathPaste returns with the same error.

LIBVIGIL_DIR_Iter

This function scans the specified directory for entries. For each entry found in the directory, the specified callBackFn is called. The directory entry and argVoid are passed to the callBackFunction as parameters.

If the callbackFn returns a non-zero value, directory iteration is terminated.Generally, the value returned by the callBackFn is also returned by LIBVIGIL_DIR_Iter. However, if the callBackFn returns ELOOP, LIBVIGIL_DIR_Iter returns VIGIL_SUCCESS.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_DIR_Iter(
              const char *dirPath,
              int       (*callBackFn)(struct dirent *dirEnt, void *argVoid),
              void       *argVoid,
              int        *callBackRCode
              );
```

Parameters

dirPath

(IN) Filesystem path to target directory to iterate.

callBackFn

(IN) Specifies a function to call, once for each entry in the directory.

argVoid

(IN) A void pointer to pass to the callBackFn. NULL can be specified if no data is passed.

callBackRCode

(OUT) Specifies the return value from the last call to the callBackFn. NULL can be specified if the caller does not require this value.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	dirPath or callBackFn parameter passed is NULL or invalid parameters are passed.

rCode	This function calls the following functions: ◆ opendir ◆ readdir ◆ closedir If any of these calls fail, this function returns the resulting errno. This function also calls the specified callBackFn. If the callBackFn returns a non-zero value, directory iteration halts. If that non-zero value is not ELOOP, LIBVIGIL_DIR_Iter will also return the same value.

LIBVIGIL_NSS_MapNssGUIDToNssObjectName

This function resolves an NSS volumeID to an NSS volumeName.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_NSS_MapNssGUIDToNssObjectName(
              VIGIL_GUID_T  *objectGuid,
              uint8_t      **objectName_ALLOC
              )
```

Parameters

objectGuid

(IN) Specifies a 16-byte volumeID GUID value.

objectName_ALLOC

(OUT) Returns the volumeName of the specified guidObject.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	objectGuid parameter passed is NULL or invalid parameters are passed.
	rCode	This function calls lsata. If stat fails this function returns the resulting errno. This function calls the following functions: ◆ LIBVIGIL_DIR_Iter ◆ LIBVIGIL_MEM_Alloc ◆ LIBVIGIL_DIR_PathPaste ◆ LIBVIGIL_XML_GetContentBySimpleRootLabel ◆ LIBVIGIL_GUID_StrGUIDToGuidMatch If any of the above function returns an error, then LIBVIGIL_DIR_PathPaste returns with the same error.

LIBVIGIL_XML_GetContentBySimpleRootLabel

This function returns the value found between a xml tag label and its closing tag.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

  int LIBVIGIL_XML_GetContentBySimpleRootLabel(
              uint8_t       *filePath,
              const uint8_t *label,
              uint8_t       **content_ALLOC
              )
```

Parameters

filePath

(IN) File system path to the target xml file.

label

(IN) XML label identifying sought content.

content_ALLOC

(OUT) Returns the sought content in allocated storage.

Remarks

This function is designed only for very simple xml files which do not imbed xml tags within tag content. This function can be used for examining xml files under the NSS _admin volume.

Return Values

0	VIGIL_SUCCESS	Success.
22	EINVAL	Parameter passed is NULL or invalid.
	ENOENT	The specified label was not found in the XML file or a closing tag was not included after the content. This function calls the following functions: ◆ LIBVIGIL_FILE_ToMem ◆ LIBVIGIL_MEM_Alloc If any of the above function returns an error, then LIBVIGIL_XML_GetContentBySimpleRootLabel returns with the same error.

Deprecated Functions

A

This section lists the deprecated functions and will be removed in the next release of the code.

LIBVIGIL_PARSE_EnterExitCheck

This function is deprecated due to consolidation of pre and post events by the auditing engine. The function will be removed from the code in the next release.

Syntax

```
#include <vigil.h>
#include <libvigil.h>

int LIBVIGIL_PARSE_EnterExitCheck(

        uint32_t  auditType,
        uint32_t  auditEvent,
        int       *state_OUT
        )
```

Parameters

auditType

(IN) Vigil record type.

auditEvent

(IN) Vigil record type event.

state

(OUT) Type event state, including:

(-1) LIBVIGIL_PARSE_STATE_UNKNOWN

0 LIBVIGIL_PARSE_STATE_SINGULAR

1 LIBVIGIL_PARSE_STATE_ENTER

2 LIBVIGIL_PARSE_STATE_EXIT

Return Values

0	VIGIL_SUCCESS	Success.
2	ENOENT	Vigil record type or event is unknown.