BitDefender
Client-Server Anti-Malware SDK
ICAP Server Anti-Malware SDK

User Guide for OEM Partners

Last update: Nov 13, 2015

Table of Contents

1 Introduction

1.1 Legal

This documentation is proprietary and confidential. It contains trade secrets, and it is subject to a non-disclosure agreement with BitDefender. Commercial use is subject to a license agreement with BitDefender. The content of this document should not be disclosed to 3rd parties without the prior written permission of BitDefender.

All trademarks mentioned throughout the document belong to their respective owners.

1.2 Overview

The BitDefender Client-Server Software Development Kit (the SDK) is designed to provide the partner application with the functionality necessary to detect and clean malicious content.

1.3 What it can do

The SDK has the following features:

- Detects viruses in the partner-provided content. Can also optionally disinfect it, if possible. In case of detection, provides the infection name and type.
- Detects spyware, adware, dialers and other malicious application types. In case of detection, provides the malicious application's name and type.
- Native support for multiple operating systems and 32-bit and 64-bit platforms using the same API and protocol;
- Scans and detects malware in the following objects:
  - Files on physical or virtual disk using open protocol;
  - Data buffers in shared memory using open protocol;
  - Any content using the industry-standard ICAP protocol;
  - Any content using the industry-standard SpamAssassin / SPAMD protocol;
- Runs in a separate process providing an extra layer of independence and security to the partner application.
- Scans and disinfects all file formats known to be used to spread malware content, including archives, installers and various encodings;
- Full multithreading and concurrent scans support. All scans are run using either multiple threads or multiple processes mode (configured by the partner);
- Allows configuring scanning options individually, for each scan;
- Automatically updates the anti-malware database and anti-spam SDK;

- Provides optional callbacks to receive information about the scan in progress;
- Provides a functionality to enumerate the virus names present in the loaded anti-malware database using open protocol;
- Provides protocol-level and C/C++ interface bindings. The protocol and the client library sources are available for the partner, making multiple integration models possible.

The following ICAP features are implemented:

- Scanning HTTP client requests using REQMOD, with an automatic decoder for *multipart/form-data* MIME file upload forms.
- Scanning HTTP server responses using RESPMOD.
- Persistent connections support (keep-alive).
- 204 status code support (no changes needed) for the clients which support it.
- Preview support to automatically skip streaming video and audio, and skip specific content types (configurable)
- Configurable template information page if malware is detected in the request or response.

1.4 What it cannot do

- The SDK scans content in a separate process, and it therefore cannot scan the memory buffers of the partner application process. Shared memory scanning could be used to implement this functionality.
- The SDK cannot scan partial content. The complete object must be present when the scanning function is called. It does not support continuous stream scanning either.
- The SDK cannot scan or clean protocol-encoded content intercepted from the network, which contains network or protocol-specific information (TCP headers, SSL encryption, HTTP headers, SMTP commands and so on). The protocol-specific information must be stripped, and the object must be decoded before scanning. This includes decoding chunked encoding in HTTP streams, decoding TLS SMTP sessions down to a plain e-mail message, and so on.

1.5 Licensing

Like most commercial libraries, the SDK requires a run-time license in the form of a serial number received from your account manager. This serial number allows the partner to use the SDK until the license expiration date. The single serial number should be used for all installations of the partner product. There is no other license enforcement inside the SDK besides the expiration date check. For the trial license provided during the review and integration period the expiration date is checked against the computer clock, and the SDK will stop working as soon as the current time exceeds the license expiration date. For the

commercial license provided for inclusion into the released application the expiration date is checked against the release date built into the anti-malware database. This ensures that even if the user or malware application changes the computer date to a future date, the SDK will continue to function normally. However, if the anti-malware database was released after the commercial license expiration date, the SDK will not load it, and it will cease working.

There are two types of license. One type enables support for all server protocols, including ICAP. The second type does not allow using the ICAP protocol, but only internal protocols.

Only the server part of the SDK uses the license, which is checked during the server startup. The client part is provided in source code form, and it does not use a license.

The trial license is usually issued on a short-term base and it is typically valid for one to three months. The commercial license is usually valid for two or three years. Six months before the license expires, the account manager will provide a new serial number. The partner is responsible for the delivery of this serial number to all customer applications by means of their own update system. The license always expires on a specific date, and it does not depend on when the product was actually installed, which makes serial number maintenance easier.

The partner application should never use a trial license for commercial release.

1.6 SDK Components

The client-server SDK consists of three components, some of which are provided together with their source code. Please note that the source code is only provided for the integration purposes, and cannot be redistributed or delivered to the end user in the source code form.

1.6.1 Server Application

The server is the main SDK component. It is a separate, stand-alone daemon process without a user interface, which receives scan requests from clients, scans the files and provides the result. It can handle multiple clients simultaneously, and - depending on its configuration - process them using either multiple threads or multiple processes. The server communicates with clients using TCP/IP or Unix domain sockets, and is able to listen on multiple sockets. The server supports the BitDefender open protocol, and the ICAP protocol. For the sake of efficiency and simplicity, the BitDefender open protocol is used while to scan the local files and the shared memory buffers, and to enumerate the viruses. This protocol is described in the Server protocol section. If the scanned content needs to be transferred over the network, the ICAP protocol should be used.

The server should be started before any client can scan the files for viruses. It requires an anti-malware database and a license serial number to start. It supports multiple command-line options, and the configuration file as well as some signals to control its behavior.

The server application is provided in binary form only.

1.6.2 Client Library

The client library implements the server communication protocol, and it provides a high-level

API to access most of the server functionality through the C/C++ programming interface. The source code for the client library is available to the partner, and, therefore, the library can be used either via the public API, or modified to ease integration into the partner application. The client library API is described in the Client library API section. Only the internal server protocol is supported by the client library; the ICAP protocol is not supported.

The client library is provided in both binary and source forms.

1.6.3 Client Application

The client application is a sample application which uses the client library to communicate with the server in order to scan files for viruses. It uses the client library API to perform its tasks, and it can be used for the following purposes:

- To implement the easiest integration by running the client executable and analyzing its output in order to get the scan results;
- To study its sources and implement the custom integration based on them;
- To quickly test server installation (i.e. whether the server is up and running), or the latest update (i.e. whether the scan results are the same).

The documentation covers the client application command-line options and a few scenarios. The client application is provided in both binary and source forms. As a client library, the client application does not support the ICAP protocol.

1.7 Changes

1.7.1 Version 1.19

This version fixed excessive disk usage when checking for an update resulted in no update being downloaded, as well as added the following options:

- 2.6.5.2 AntiSpamScanRetry
- 2.6.5.3 AntiSpamScanRetryTimeout

Also three commands were added to the protocol, API and the command-line:

- Retrieve Anti-malware engine update information
- Retrieve Anti-spam engine update information
- Trigger an update

1.7.2 Version 1.18

This version added LogAppend option and support for logging ICAP requests and responses where infected content was found (see PathInfectedLogICAP)

1.7.3 Version 1.17

This version adds the client library in both static and shared format, and fixes two issues:

- The 32-bit versions of the SDK refused to scan for malware the files larger than 2Gb, which is fixed now. 64-bit versions were not affected by this issue
- The license expiration resulted in a cryptic error message which was hard to spot and decypher. Now a proper message is printed.

1.7.4 Version 1.16

The update logic changed. Prior versions of the SDK updated the database in-place, which in rare cases could result in update failures. This logic has been changed, and now the SDK downloads the updates into the different directory. As a result, the following configuration options have changed:

- *PathDatabase / PathAntimalwareDatabase* options are not valid anymore. Please use the PathAntimalwareRoot instead.
- *PathAntispamDatabase* option is not valid anymore. Please use the PathAntispamRoot instead;

Please read the documentation for those options before replacing them in the configuration file.

Other changes include several minor bugfixes; dropping command-line options in favor of the configuration file, which is now mandatory; and switching to downloadable *bdcore.so* shared library, which is not packaged with the SDK anymore.

1.7.5 Version 1.15

The SDK is rebuilt with the updated version of openssl without the heartbleed bug.

Fixed the issue when the SDK downloaded the updates but not used them, and kept using the old version of the anti-malware database.

Fixed the issue when the SDK hangs if PathDatabase was the same as PathAntispam.

1.7.6 Version 1.14

Anti-spam component now uses a different license. Added the AntispamAggressivityLevel setting.

1.7.7 Version 1.13

Added support for anti-spam scanning and SPAMD protocol.

2 SDK Integration

This section describes the steps necessary in order to set up the environment of, install, configure and run the application in such a way that it functions properly.

2.1 Requirements

2.1.1 Supported Operating Systems and Compilers

The SDK primarily supports Linux and FreeBSD platforms on x86 or x86_64 CPU. The following subsection describes the supported operating systems.

2.1.1.1 Linux

The 32-bit version of the SDK should work on any distribution which provides GLIBC 2.2 or higher. Since this version of glibc was introduced as far ago as RedHat 8.0, any modern Linux distribution should work as long as its kernel and dynamic linker are configured to support shared libraries.

The 64-bit version of the SDK supports GLIBC 2.3.3 and higher.

The SDK requires gcc 3.3.6 or higher. gcc4 is recommended. If gcc is custom-built, threading support (`--enable-threads` configure option) must be enabled during gcc configuration.

The pthreads library should be included in the list of linked libraries because the SDK is built using thread support and therefore depends on it.

2.1.1.2 FreeBSD

The SDK supports FreeBSD 6.3 and FreeBSD 7.

2.1.2 System Requirements

Besides CPU time, the SDK requires the system to provide some memory and disk storage.

2.1.2.1 Memory

The SDK uses memory to store an unpacked copy of the anti-malware database and for internal storage purposes. Unpacked files are also stored in memory until they reach the hardcoded memory consumption restrictions. Typically, the single engine instance will require about 5Mb of memory for its internal storage. The anti-malware database is shared among all engine instances.

2.1.2.2 File System

The SDK needs the file system to load the anti-malware database, and to store temporary files. Persistent storage should be used for the anti-malware database.

Temporary storage is used to store temporary files which come from archives. To speed up scanning, the SDK tries to keep temporary files in memory. However, if a temporary file is becoming too large, or there are too many such files, the disk space is used. The required amount depends on the types of files scanned, and it may range from 4Mb for a small HTTP scan server to up to 512Mb on a large e-mail scanning appliance, which scans hundreds of e-mails simultaneously.

2.1.3 Anti-Malware Database

The SDK needs the BitDefender anti-malware database to perform its functions. The database contains signatures to detect threats, and the code necessary for internal processing, heuristics and unpacking. The database is updated with new signatures approximately every hour, and new unpackers are added, if necessary. At this moment (Q3 2010) the anti-malware database consists of around 600 small files, and has a total size of 80Mb on disk, and approximately 150Mb in memory.

To maintain an adequate detection rate, the anti-malware database needs to be updated regularly. BitDefender recommends updating the anti-malware database every hour to ensure that there is a better chance of detecting the latest threats on time. Updates are usually downloaded from the BitDefender server to the mirror web server provided by a partner, and then to the client machines, using either the built-in updater or the updater provided by the partner application.

If updates are performed regularly, the approximate update size is 50-150Kb.

2.1.4 Anti-Spam SDK

The SDK uses the cloud-based Anti-spam, so the database updates are not necessary. Nevertheless the anti-spam SDK itself needs to be updated periodically to cope with new spam, so it should be checked for updates once a day.

2.1.5 Choosing the Integration Scenario

The SDK provides several components which can be integrated using one of the scenarios below. All scenarios only cover client integration, and assume that the server component is already started.

2.1.5.1 Integration by using bdamclient executable

In this integration scenario the partner executes the client application executable with the necessary command-line parameters, and parses its output to get the scanning results.

Pros: this is the easiest way to integrate the anti-malware scanning functionality. It does not depend on the programming language used by the partner application, and it does not suffer from binary compatibility issues. Being a separate program, it cannot crash the partner application, and there is no concern over things like resource leaks . Since the source code of the client executable is provided, the executable can be changed to output the result in any preferred format (i.e. XML), which allows the partner to integrate it into existing application

hook scripts without changing the application itself.

Cons: this method has the worst scanning performance possible, since it requires spawning a new process and establishing a new connection to the anti-malware server for each file scanned. It also provides only limited access to the server functionality. Writing error-proof output parsers is not an easy task either. Also the client executable based on the modified source code is not supported by BitDefender, which increases support efforts.

This integration is suggested to be used for proofs-of-concept, if performance is not a concern, and for test scripts.

2.1.5.2 Integration by using bdamclient source code or amclient library

In this integration scenario the partner implements anti-malware scanning by using the provided client library API to communicate with the server, scan the content and obtain the scan result.

Pros: this is the most common way to integrate 3rd party software, which every developer should be familiar with. It provides fine-grained control over the anti-malware scanning process, and it generally provides the best performance, if used reasonably. Since the source code of the client library is provided, the partner can easily review it to ensure stability or the absence of any undocumented functionality.

Cons: since the library has only C/C++ bindings, this method can only be used with applications written in C/C++. Other programming languages require putting extra efforts into using a library. Also, since the client code is now part of the partner code, the bugs in the library could affect the partner application.

This integration is suggested for the vast majority of partners both as proofs-of-concept and for production. It lasts longer than the integration using the client executable, but the performance and compatibility benefits usually justify the extra efforts.

2.1.5.3 Integration by implementing amserver protocol

In this integration scenario, the partners implement the server protocol themselves, instead of using the client library or application.

Pros: this method allows the best control over the server, and, in some cases, it can provide a performance increase compared to the library use (for example by pipelining scan commands). Since the protocol is open, it can be implemented in any language which has socket support, allowing the partner applications written in languages other than C/C++ to integrate with the server without a performance penalty.

Cons: this method requires the most efforts compared to all other methods. The protocol documentation is generally harder to follow than the API documentation, and debugging the implementation might be cumbersome.

This integration is suggested for the partners who require the best performance possible, cannot or do not want to use the client library, and are willing to spend extra efforts on implementation.

2.1.5.4 Integration by using ICAP client

In this integration scenario the partner implements or uses an industry-standard ICAP client. The SDK supports the ICAP 1.0 protocol, as well as REQMOD and RESPMOD commands, which allows scanning both HTTP requests and responses.

Pros: this method allows the easiest integration if the partner application (usually HTTP gateway) has already implemented the ICAP protocol, and it can work as an ICAP client. The ICAP protocol is industry-standard and supported by most major HTTP proxy vendors.

Cons: this method is the slowest of all methods , requiring everything to be transferred back and forth over TCP/IP. Also ICAP client implementation is more complex compared to using the library or even implementing the clear text protocol.

This integration is suggested for the partners who want to integrate with existing ICAP clients, or who will implement the ICAP client themselves.

2.1.5.5 Integration by using SpamAssassin / SPAMD client

In this integration scenario the partner implements or uses an industry-standard SPAMD client. The SDK supports the SPAMD 1.2 protocol, but only partially due to Bitdefender anti-spam being very different from SpamAssassin internally..

Pros: this method allows the easiest integration if the partner application (usually mail gateway) has already implemented the SPAMD protocol, and it can work as an SPAMD client. The SpamAssassin protocol is industry-standard and supported by most major mail servers.

Cons: this method is almost as slow as ICAP, requiring everything to be transferred back and forth over TCP/IP. Also the SPAMD client implementation is more complex compared to using the library or even implementing the clear text protocol.

This integration is suggested for the partners who want to integrate with existing SPAMD clients, or who will implement the SpamAssassin client themselves.

2.1.6 Choosing Server Mode

The server can handle clients in multi-threaded, worker or preforked mode. The mode is set in the configuration file, and can be changed any time (but requires full restart of the server). All modes are described below, together with their advantages and disadvantages. Note that the worker mode only supports ICAP protocol, and does not support client-server protocol.

2.1.6.1 Multi-threaded mode

In the multithreaded mode each connected client is handled by a separate thread. The thread is started when a new connection is established, and is destroyed when the connection is closed. Each thread proceeds a single client connection.

This mode has the following benefits:

- Low idle resource usage. If no clients are connected, the server consumes very little resources comparing to preforked mode.

- The maximum number of connected clients the server can handle simultaneously is only restricted by the number of threads which can be created. In a preforked more the maximum number of connected clients is specified in the configuration file, and cannot be increased in runtime.
- Overall system responsiveness is better in the multithreaded mode – the system limits the CPU time for the server process, and makes sure other processes have a fair chance to run too.
- Seamless update – the established connections will not be closed, and will use the new database once it is available. No scans are possible, however, while the database is being reloaded.

However this mode has the following drawbacks:

- With many clients overall memory consumption is larger comparing to preforked mode, as the application heavily uses memory allocation, and suffers from fragmentation issues.
- Overall performance is better in the preforked mode (which, however, consumes more system resources, especially CPU, so it affects the system responsiveness).
- Depending on KeepEngineMemory option setting, the first scan on an engine instance might take longer time as necessary memory needs to be allocated.
- If the server suddenly receives a lot of connections, it creates a lot of internal concurrency as a lot of threads now need to be spawned – which, in turn, creates synchronization issues, negatively affecting the performance.

This mode is recommended when the load on the server is irregular, and may vary from idle to very high, with the median is in the middle-high range. This mode is also useful if memory or system load is an issue, and performance expectations are not high.

This mode supports both the client-server and ICAP protocols. However it is not recommended to use this mode for ICAP protocol except for the scenarios when the ICAP client is using up to 8 persistent connections.

2.1.6.2 Preforked (multiprocess) mode

In the preforked mode the server spawns multiple children processes during start. Then each child process handles one connected client. Once the client finishes, the process is waiting for another connection, and handles another client, and so on.

This mode has the following benefits:

- Low overall memory consumption with numerous clients comparing to multithreaded mode. The memory consumption does not depend on KeepEngineMemory option setting, and it does not suffer from memory fragmentation issues.
- Overall performance is better, and scans are faster because there is no internal concurrency and CPU cores are fully utilized. This, however, increases system load.
- Since the children processes are ready to accept connections, the server does not have issues when a lot of connections is established instantly.

However this mode has the following drawbacks:

- Higher idle resource usage comparing to multithreaded mode. If no clients are connected, the server still runs extra processes which consume memory. They do not consume CPU though.
- The maximum number of connected clients the server can handle simultaneously is predefined in the configuration, and cannot be increased without server restart.
- Overall system responsiveness is worse comparing to multithreaded mode – the system spends more time in scanning processes, which provides better application performance, but leaves less CPU time for other processes.
- The update requires the client to reestablish the connection. While the server does not terminate connections in the middle of scans, the connection must be closed and the process handling the client must exit to complete the update.

This mode is recommended when the load on the server is expected to be constantly high, the system has enough resources to handle it, the number of connections the server will need to handle is known in advance, and the system is not expected to spend a lot of time in idle mode.

This mode supports both the client-server and ICAP protocols.

2.1.6.3 Worker mode (ICAP only)

In worker mode the server spawns the requested number of worker threads, and each thread handles multiple connections asynchronously. The connections are not tied to a specific thread, so the system resources are used effectively. This mode supports numerous simultaneous connections (up to 1300 have been successfully tested) while having relatively low memory impact. This mode is supported only on Linux with kernel 2.6.8 or higher, and with recent libc which has epoll support.

This mode has the following benefits:

- Low overall memory consumption with many clients.
- The threads are not spawned, so the connection handling is generally faster.
- Uses threads and does not clutter the process table.
- The performance should be similar to the preforked mode.
- Increasing the maximum number of supported connections has little impact on the system comparing to preforked mode.

However this mode has the following drawbacks:

- If there is a bug in the application and it crashes while processing a single connection, the whole process will crash.
- Only supports ICAP protocol, so if the partner application needs to use both ICAP and client-server protocols, this mode cannot be used.
- Linux only.

This mode is recommended for the highly loaded servers, but works well when the load goes down.

This mode supports only ICAP protocol, and does not support client-server or SPAMD protocols (the performance would likely suffer if it did).

2.1.7 Server Configuration

After choosing the integration method it is advised to choose the anti-malware server configuration and start it to make sure the environment supports the configuration.

2.1.7.1 Configuration file

The server should be configured using the configuration file. In past the configuration with command-line options only was supported, but due to the increased number of those options this configuration is not supported anymore.

2.1.7.2 Listening on sockets

The server can listen on IPv4 and Unix domain sockets. The number of sockets listened on generally does not affect the performance, so it is possible to listen on more than one socket.

Generally, Unix domain sockets provide better performance compared to IPv4 sockets, and they should be preferred if both the client and the server are running on the same machine. They don't use a TCP port either, so there is less chance of potential conflict with other applications. However, if the partner application is designed for a low-end appliance with limited resources, it might have support for Unix domain socket compiled out, so IPv4 socket has to be used instead.

The same listening socket is used for all enabled protocols. This means if the SDK is listening on the port 9100, it will handle both the internal protocol, the ICAP (if enabled) and SPAMD (if enabled) transparently.

If ICAP server mode is enabled, the suggestion is to use the standard listening port for ICAP protocol, which is 1344.

2.1.7.3 Log level

It is important to consider the logging level when designing the application. It is not recommended to enable ScanInfo and Debug logging on a production application because it wastes performance and fills storage with the logging data. However, it is important that the end user should be able to enable it, if necessary (in advanced settings or so), because BitDefender might require debug logs in order to be able to fix most of the issues found.

2.1.7.4 Chroot/Jail

The server can be run in a jailed/chrooted environment. While there is no built-in support for chroot, it is possible to provide it using standard Unix tools. It is highly recommended to use command-line options to test a new setup first, to make sure everything works.

There are several things which need to be watched out for:

- If the anti-malware database is copied to the chrooted environment, it should be copied again after the update, but before the server is restarted. Otherwise, the server will not pick up the updated database.
- All paths will be relative to the chrooted environment. This includes the log file location, anti-malware database path and Unix domain sockets listened on.
- The SCANFILE command and *BDAMClient_ScanFile* functions require the file path to be absolute. They specify the file path for the server to scan. If the server is run in a chrooted environment, the path must be provided as the server would see it. This means that the file must be stored somewhere in the chrooted environment, and that the server should receive an absolute path in this environment to be able to open it.

2.1.7.5 Configuring the anti-spam component

The anti-malware component is configured by default, but optionally the partner can enable the built-in anti-spam component. This component is designed to scan RFC822 format e-mails for spam using Bitdefender cloud. Therefore using this component only makes sense if the scanned object is an e-mail.

When using the internal protocol the partner can enable or disable scanning for spam using the BDAM_SCANOPT_SPAMCHECK scan option. When using the ICAP or SPAMD protocols the spam scanning is enabled by choosing the relevant configuration file option in ProtocolSupportICAP and ProtocolSupportSPAMD respectively.

To enable the anti-spam component in the SDK the partner needs to do the following:

- Obtain the anti-spam license serial number from the account manager (it is different from the regular client-server SDK license);
- Specify this license number in the LicenseAntispam configuration file option;
- Set the PathAntispam configuration file option which must point to the directory writable by the user ID the SDK is run under;
- Start the bdamserver component.

Upon start the server will download the latest anti-spam SDK into the directory specified by PathAntispam and will use it.

2.1.8 Choosing Scan Options

Depending on the application's purpose, the architect who integrates the SDK needs to choose the appropriate scanning options. Choosing an incorrect set of options might negatively affect performance or lead to missing viruses, so option selection must be done with care.

The full list of available options is included in the Scan Options section.

2.1.9 Scan Modes

No matter what integration scenario was chosen, the SDK supports three scan modes. Because choosing the appropriate scan mode leads to easier integration and might provide a performance boost, it is suggested that the partner application architect should get familiar with the available scanning modes and with their restrictions.

Generally, the best approach is to scan the content in its current form. If the content is stored in shared memory, shared memory scanning provides the best performance and easiness. Otherwise it should be scanned by file name. In the case of an ICAP, the content is available on a remote computer, and it is necessary to transfer it to the computer running the ICAP server.

2.1.9.1 Scan File by File Name

This is the typical and most familiar way to use the 3rd party anti-virus or anti-malware engine. The partner application has a file name pointed to the file on the disk, and calls the scanning function providing a file name. Of course, the file must be accessible to the server.

Pros: file scanning by file name is the simplest mode, easy to understand and implement. There are no restrictions on disinfection, and other actions are available which allow the partner application to tell the SDK to quarantine the file, move it or remove it.

Cons: since the server might be running using different credentials, it might not be able to open the file. The file path is required to be absolute, but it might not match if the server is being run in a chrooted environment, making the application more complex.

In most cases, this is also a slower mode comparing to shared memory scanning. Obviously, if the file name is all the partner application has, something will have to open this file anyway for scanning, so there are no other options. However, if the partner application has its content stored in memory, scanning shared memory buffer will be much faster.

2.1.9.2 Scan File stored in Shared Memory Buffer

In this mode the partner application stores the scanned file in the shared memory buffer, and then calls the scanning function providing the access information for this buffer and the object size.

Pros: this method is very fast and involves no disk access to the object. If the file was received from the network and needs to be scanned, this method will provide the best performance.

Cons: if disinfection is required, and the file size increases, the disinfection will fail (a very rare case, so there should be no concern about it). Requires more caution and maintenance efforts compared to file scanning.

2.1.9.3 Scan File stored on Remote Computer

In this mode the partner application stores the scanned file in the memory buffer or in a disk file, and then uses the standard ICAP protocol to send the object to the remote machine for

scanning.

Pros: this method uses the standard ICAP protocol, providing easy interoperability with other ICAP servers, if necessary. This is also the only method which allows scanning content on a remote machine.

Cons: scanning options and progress report are not available, since the ICAP protocol has no support for it. Also the scanning is much slower compared to all other methods, as extra time is spent to transfer the file to the remote server.

2.1.10 Interpreting Scan Results

When the scan is completed, the combined scan result is returned. It is very important for the application to handle the scan result properly. Simple one-line handling, such as “everything which is not infected is clean” or “everything which is not clean is infected” could lead to serious problems such as missing viruses or getting false positives.

The full list of possible scan result values is included in the API reference, Scan result section.

It is not possible to obtain the scan results using the ICAP protocol. Instead, the returned HTML template page will contain the verdict using the replaced keywords.

2.1.11 Updating the anti-malware database

To maintain good detection rate, the anti-malware database needs to be updated frequently. BitDefender releases updates 16-24 times a day, and it is important that the server downloads the new update as soon as it is available to reduce the detection time when new malware is available in the wild.

The anti-malware database can be updated either by using internal updater (recommended), or by using external updater.

The internal updater only updates the anti-malware database. The SDK itself, including bdcore shared library, is NOT updated.

2.1.11.1 Using internal updater

It is recommended to use the internal updater to update the anti-malware and anti-spam databases. Once configured, it runs automatically according to configuration, and keeps the anti-malware database up-to-date. The database is also automatically reloaded once updated.

2.1.11.2 Using external updater

For some reason the partners may want to use external updater. The reasons may include using partners' own technology to deliver updates, or maintaining more than one set of the anti-malware database on a machine. Using external updater is more difficult, prone to errors and not recommended. Bitdefender will not provide support for any issues related to using the external updater.

Please email the support for guidelines if you still decide to use the external updater.

2.2 Integration Using Client Library

This section describes the steps to be taken in order to use the client library for malware scanning. Since the library uses C language, the integration is done by writing the code. There are several steps, each of which covers a particular case together with sample code. Some of these steps are compulsory, which means that they need to be performed no matter what scenario is chosen. They are marked as [required]. Some steps are optional, namely they are only required if some specific functionality is to be used. These steps are marked accordingly, as well.

```
The code sample will be shown like this.
```

Generic string rule: all strings used in the SDK are encoded using UTF8 encoding.

The API interface is located in the *bdamclient.h* file ,which needs to be included in the source code calling the client library functions.

2.2.1 Creating New Client Instance [required]

The client SDK is based on a client instance, and all SDK functions, except the function to create this instance, require it as an argument. Therefore, before any other function can be used, the client instance must be created.

Each client instance can be connected to the server and used to scan content. Each instance represents a single connection, and it can only scan one object at a time. If the partner application needs to be able to scan multiple objects simultaneously, several client instances must be created, and connected separately.

The following code demonstrates how to create an instance:

```
BDAMClient * client = BDAMClient_Create();

if ( !client )
    // handle the error
```

If the function succeeds, it returns the instance. Otherwise (usually memory problems) it returns 0.

2.2.2 Connecting to Anti-Malware Server [required]

Once the client instance is created, it must be connected to the server to do any useful work. To connect to the server, you need to have the server up and running and know its IPv4 address and port or Unix domain socket path.

The following code demonstrates how to connect to the server:

```
//const char *  peer_addr = "/var/run/avsocket";
const char *  peer_addr = "127.0.0.1:5040";
```

```
int err = BDAMClient_Connect( client, peer_addr );

if ( err != 0 )
    // handle the error
```

This function requires a previously created client instance, and the argument which specifies either the full absolute path (starting with a slash) to the Unix domain socket, or the IPv4 address and port, separated by a colon. Since this function does not support domain names, the IP address must be provided in numeric form.

If the connection cannot be established, this function returns an error code. The function API reference contains the list of error codes which could be returned, together with their descriptions.

This function can also be used to reconnect to the anti-malware server, or cto a different server. It automatically closes the active connection, if already connected.

2.2.3 Setting Scan Options [optional]

Before scanning content, the allocated scanning instance can be configured if the default scanning options need to be changed, or callbacks need to be set. This configuration is optional as the SDK sets the default options automatically, during engine allocation, but it is recommended when scanning archives/e-mail databases, or when disinfection is required.

The following code demonstrates how to enable disinfection and to disable packed file scanning (not recommended):

```
if ( BDAMClient_SetOption( client, BDAM_SCANOPT_DISINFECT, 1 ) != 0 )
    // handle the error

if ( BDAMClient_SetOption( client, BDAM_SCANOPT_PACKED, 0 ) != 0 )
    // handle the error
```

Setting the option can only return an error if an invalid option value is provided, and it can be done either before or after establishing a connection to the server.

2.2.4 Setting in-progress callback [optional]

Before scanning content, the allocated scanning instance might be optionally configured to provide in-progress information during scan by calling the partner-specified callback. This callback will provide in-progress scan information for non-clean files, and only makes sense while scanning an archive as for regular files in-progress information just duplicates the scan result. It is not required and generally not recommended because it slows down scanning process.

The following code demonstrates how to set the callback:

```
void ClientCallback (const char * filename, int status, const char *
          threatname, int threattype, void * ctx)
```

```
{
    // this is the body of in-progress callback
}

...

if ( BDAMClient_SetCallback( client, ClientCallback, 0 ) )
    // handle the error
```

Setting the callback function address to 0 disables the in-progress callback. Setting the option can return an error only if an invalid option value is provided, and it can be done either before or after establishing a connection to the server. At this moment, the function does not return an error, but it might change in the future, so that a check is recommended.

2.2.5 Obtaining run-time information from server

This function should be used if the partner application needs to obtain the run-time information from the server. It is optional (not required for scanning) and it can provide the following information:

- License expiration date and time;
- Release date and time of the anti-malware database update;
- Number of virus records in the anti-malware database;

The following code demonstrates how to get this information using a connected client instance:

```
time_t license_expiration_time;
time_t amdb_update_time;
unsigned long amdb_records;

int err = BDAMClient_Info( client,
                           &license_expiration_time,
                           &amdb_update_time,
                           &amdb_records );

if ( err )
    // handle the error
```

The partner application must handle the cases where the function returned success, but one or both time parameters equal to -1, which means that this information is not available.

If this function fails, it returns an error code. The function API reference contains the list of error codes which could be returned, together with their descriptions.

2.2.6 Scanning File on Disk

To scan a file on the disk, the following code should be used:

```
int status, threattype;
const char * threatname;

// filepath must be absolute path to the file as server sees it
int err = BDAMClient_ScanFile( client,
                               filepath,
                               &status,
                               &threattype,
                               &threatname );

if ( err )
    // handle the error
```

The file path should meet the specific criteria explained in detail in the BDAMClient_ScanFile function API reference. In short, the path must be absolute, and the file needs to be a regular file which can be accessed by the server using the path provided.

If the function returns an error code, please refer to the function API reference for the list of possible error codes and their descriptions. The file should be treated as not scanned. The function output arguments are undefined.

If the function returns success, this only means the file was successfully scanned, and the scan result is stored in the *status* argument. It does not mean that the file is clean; the scan result should be interpreted according to the documentation to get a final verdict.

2.2.7 Scanning shared memory segment

To scan the file content in the SVr4 shared memory segment defined by the key *segmentKey* and size *segmentSize* the following code should be used:

```
int status, threattype;
const char * threatname;

int err = BDAMClient_ScanSharedMemory( client,
                               segmentKey,
                               segmentSize,
                               &status,
                               &threattype,
                               &threatname );

if ( err )
    // handle the error
```

To be scanned, the shared memory segment must be created by the partner application with permissions allowing remote process to attach to it. The segment should be created with a real key obtained by ftok(), not with IPC_PRIVATE.

If the function returns an error code, please refer to the function API reference for a list of possible error codes together with their descriptions. The shared memory buffer should be treated as not scanned. The function output arguments are undefined.

If the function returns success, this only means the shared memory buffer was successfully scanned, and the scan result is stored in the *status* argument. It does not mean that the file is clean; the scan result should be interpreted according to the documentation to get a final verdict.

2.2.8 Destroying Client Instance

Once the client instance is not used anymore, it should be destroyed to close the connection, and reclaim the memory. The following code shows how to do it:

```
int err = BDAMClient_Destroy ( client );

if ( err )
    // handle the error
```

Before destroying the instance, the partner application must make sure that the instance is not running any active scanning, or that it not being used by any other thread. Attempting to destroy an instance which is running an active scan will lead to unpredictable results.

After being destroyed, the client instance pointer becomes invalid and should not be used anymore.

2.2.9 Handling Connection Errors

If the anti-malware server is started with timeouts enabled (not a recommended configuration), the server might drop the established connection if there is no activity during a specified amount of time. This will lead to a CONNECTION_CLOSED error when the partner application tries to scan an object or to get the information from the server.

To work around this problem, the partner application code should close the connection (by destroying the client instance) once the instance has finished scanning the object queue.

2.3 Integration Using ICAP protocol

The server part of the SDK supports industry-standard ICAP protocol described in RFC3507. Generally ICAP protocol requires sending the whole file via TCP connection to the ICAP server, and (if 204 is not supported) back. Because of this transfer requirement the overall scan performance is worse comparing to using standard client-server protocol. There are, however, advantages of using ICAP protocol which are listed in the ICAP-specific section.

The anti-malware server bdamserver supports most parts of ICAP protocol as described in RFC3507, including but not limited to:

- OPTIONS and RESPMOD methods. The method URL is irrelevant and ignored by the server, as the functionality provided by all URLs would be the same.
- REQMOD is supported with extra support for uploading files through HTML forms (using multipart/form-data and base64 encoding). The server decodes MIME/base64, and properly scans the uploaded file.

- Allow: 204 is supported only if the client indicates its support in the ICAP header. Otherwise the whole request/response is sent back as described in RFC.
- The server defaults to persistent connections unless the client specifies “Connection: close” in the ICAP header.
- Preview is supported and used for content-type whitelisting and skipping streaming media (WMV, FLV and so on). Using the Preview: is optional, and controlled by the client.
- ISTag is generated from the update timestamp of the anti-malware database, and is changed every time the database is updated.
- ICAP server name returned is “BitDefender ICAP Server *version*”
- If sent by the client, the “X-Client-IP” ICAP header field is parsed and logged into scan log. The server does not perform any validation of this field, just writes into the log file, so the client may send any desirable information in this field, not just the IP address.

Please note that the client part of the SDK can only communicate via client-server protocol, and does not support ICAP protocol. Only server part supports ICAP.

2.3.1 Implementing ICAP client

If the partner wants to implement ICAP client, the RFC3507 contains all necessary information.

2.3.2 Using a third-party client

If a third-party ICAP client is available, it can be configured to use the ICAP server for anti-malware scanning using the settings described below. Some clients do not support all listed settings, in which case they may be ignored.

- The ICAP server is typically set up as an URL in a following form:

  ```
  icap://hostname:port/path
  ```

  where **hostname** is the machine the ICAP server is running on, the **port** is the port number the ICAP server is listens to. **path** is ignored by the BitDefender server, and can contain any reasonable value.
- If the client supports persistent connections, enable it to increase the performance.
- If the client supports “Allow: 204”, enable it to increase the performance. Most clients enable it by default. If there is no such setting, it is most likely enabled.
- If the client supports sending X-Client-IP header, enable it to make user tracking easier.
- If the client supports Preview, enable it and set the preview size to either “use the size provided by the server”, or 4096.

2.3.3 Using Squid proxy

Squid version 3.0 supports the ICAP protocol, and could be used as web filter to scan passing content for malware. The following configuration has been tested and worked on Squid versions below 3.19:

```
icap_enable on
icap_preview_enable on
icap_preview_size 4096
icap_persistent_connections on
icap_send_client_ip on
icap_service service_req reqmod_precache 0 icap://127.0.0.1:1344/reqmod
icap_service service_resp respmod_precache 0
           icap://127.0.0.1:1344/respmod
icap_class class_req service_req
icap_class class_resp service_resp
icap_access class_req allow all
icap_access class_resp allow all
```

Squid 3.19 and higher replaces icap_class and icap_access with adaptation_service_set and adaptation_access respectively, having the following configuration:

```
icap_enable on
icap_preview_enable on
icap_preview_size 4096
icap_persistent_connections on
icap_send_client_ip on
icap_service service_req reqmod_precache 0 icap://127.0.0.1:1344/reqmod
icap_service service_resp respmod_precache 0
           icap://127.0.0.1:1344/respmod
adaptation_service_set class_req service_req
adaptation_service_set class_resp service_resp
adaptation_access class_resp allow all
adaptation_access class_req allow all
```

Of course icap_service URL may be different if you run the ICAP server on a different machine.

2.4 Integration Using SPAMD protocol

The SDK supports the SPAMD protocol used by a popular SpamAssassin application. This protocol is public, and version 1.4 is supported. The full protocol description is available here: https://svn.apache.org/repos/asf/spamassassin/trunk/spamd/PROTOCOL

The SPAMD protocol support is not enabled by default. To enable it, the configuration option ProtocolSPAMD should be used.

2.4.1 How the protocol is implemented

Since the Bitdefender technology is different from SpamAssassin, some conventions are used to make the protocol output compatible. For example, for any command which requires the

spam score to be reported, the score is faked. As a result, the score for spam messages is always returned as “15 / 5”, and the score for clean messages is always returned as “0 / 5”.

Below is the description if the supported SPAMD commands as well as the restrictions

- **CHECK** is supported, and returns the proper verdict in *Spam:* header. The score is returned as described above;
- **SYMBOLS** is supported, but instead of the hit list returns the SDK verdict string describing why the message was marked spam or not. The score is returned as described above;
- **REPORT** and **REPORT_IFSPAM** are supported, and returns the proper verdict in *Spam:* header. The score is returned as described above. The follow-up string is also hard-coded.
- **PING** is supported.
- **SKIP** is supported, does nothing.
- **PROCESS** is supported, and returns the modified message with the added *X-Spam-Flag: YES* e-mail header for spam messages. No header is added for clean messages.
- **HEADERS** is supported, and returns the *X-Spam-Flag: YES* or *NO* e-mail header depending on whether the message is spam or not.
- **TELL** is not supported.

2.4.2 Scanning for malware

The SDK can scan the e-mail messages not only for spam but also for malware. This behavior could be enabled depending on the ProtocolSPAMD option value; if it includes *antimalware*, the e-mail message will be also scanned for malware.

Since the SPAMD protocol does not provide any way of reporting malware infections instead of spam, the e-mails with malware detected are reported back as spam.

2.4.3 Using spamc for integration

The spamc module from SpamAssassin could be used to interact with the SDK, either for the integration or for testing purposes.

For example, assuming the SDK listens on 127.0.0.1 port 1344, the following command-line would scan the e-mail for spam and print the score:

```
# spamc -c -d 127.0.0.1 -p 1344 < spam/spam-gtube.eml
15.0/5.0
```

2.5 Scan-related Information

2.5.1 Scanning Options

Options control the behavior of the SDK, and how it scans file content for viruses. All of the options are boolean. The options are provided to the server only during scanning. Therefore if the client library is used, it stores the options set internally, and only provides them to the server as part of the scan request.

If the partner uses the library for integration, the options are set using the BDAMClient_SetOption function call. If the partner implements the server protocol, the options should be calculated by combining the option values using bitwise OR, and provided together with SCAN commands (this is only relevant for the partners implementing the server protocol themselves).

Not all the options listed here can be set using BDAMClient_SetOption(). Some of them affect the internal library logic (for example, the progress option is only set when the partner sets the callback), while others are not exported for security reasons (heuristics).

The table below lists the available options. If the option has no name, it cannot be set using the BDAMClient_SetOption() function call.

Option name	Protocol value	Description
BDAM_SCANOPT_ARCHIVES	1	If set, it enables scanning and disinfection inside archives (and in archives inside archives and so on). If unset, it skips all archives (except ZIP archives in “store” mode) and returns the “clean” verdict. Please note that an archive is not just a “zip” or a “rar” file for the antimalware engine. This is any file type which could (but not necessarily does) contain one or more files. For example, an e-mail message with BASE64-encoded attachments is a typical archive. Not all archives can be disinfected – for example, the SDK cannot disinfect RAR archives. Archives are scanned recursively, which means that if there is another archive found inside the initial archive, it will also be entered, scanned, and so on. The easiest way to limit recursion is to handle it in the callback. In most cases, this option should be set. The only reason to disable it is to scan the files intercepted with the on-access scanner. It is relatively safe, since even if an infected file is unpacked from an archive, the on-access scanner will intercept access to it, and then it will be detected as infected and blocked.
BDAM_SCANOPT_PACKED	2	If set, it enables scanning and disinfection inside packed executables. If unset, it skips all packed executables and returns the “clean” verdict.

		Unlike archives, a packed file always contains one file. The SDK can disinfect viruses found inside the packed file. Since most Trojans are now packed, it is strongly recommended that this option should always be enabled.
BDAM_SCANOPT_EMAILS	4	If set, it enables scanning and disinfection inside e-mail databases. If unset, skips e-mail databases and returns the “clean” verdict. Scanning e-mail databases slows down the engine, so if the partner application does not need these databases to be scanned, this option may be disabled. Otherwise it should be enabled.
	8	If set, it enables using the heuristics scanner for virus detection. Since this option is required for proactive protection, which is necessary to detect unknown viruses, the partner is strongly encouraged to always set this option. It is always set automatically when using the library.
BDAM_SCANOPT_DISINFECT	16	If set, the engine will try to disinfect the file if a known infection was found. Not all files can be disinfected. By default, the engine just detects the viruses and does not attempt to disinfect them.
	32	If set, the server will return in-progress information during the scan, reporting every file which is not clean. This slows down the scan, so it should be used only when necessary. It is enabled by the library if the scan progress callback is set, and disabled otherwise.
BDAM_SCANOPT_SPAMCHECK	64	Scan the object not only for malware, but also for spam using the Anti-Spam SDK. This option requires the Anti-Spam SDK to be enabled in configuration with the proper license. This option should only be used when scanning e-mail messages in RFC822 format, as an attempt to scan any other object might result in false positives.

2.5.2 Scan Result

The scan result contains information about the scan, either in-progress or final.

Name	Description
BDAM_SCANRES_CLEAN	Scanning has been completed, and no known piece of malware was found in the file. This code is also returned for the files which were not scanned at all because the corresponding option was disabled.
BDAM_SCANRES_DISINFECTED	The file was found to be infected during scanning, the disinfection option was enabled, and the disinfection attempt succeed. The file is no longer infected.
BDAM_SCANRES_DISINFECTFAILED	The file was found to be infected during scanning, the disinfection option was enabled, and the disinfection attempt failed. The file is still infected

BDAM_SCANRES_INFECTED	The file was found to be infected or containing spam (if scanned for spam) during scanning, and either the disinfection option was disabled, or the disinfection attempt failed.
BDAM_SCANRES_SUSPICIOUS	Something similar to a previously known piece of malware was found inside the file during proactive scanning. Please send this file to the BitDefender antimalware laboratory for further classification. Because this malware type has not been added to the antimalware database yet, it is not possible to disinfect the file. Generally, the BitDefender proactive scanning method results in very few false positives. This means that, statistically, this file is most likely infected, and that it should be treated accordingly. It is still possible, however, that this is a legitimate file, so the suggested action would be to quarantine it and to check it for viruses again later on.
BDAM_SCANRES_ENCRYPTED	Scanning cannot be completed since the file is password-protected, and decryption is not possible without knowing the password. The file has not been not scanned.
BDAM_SCANRES_INCOMPLETE	Scanning failed. This code means that the file scan was not completed successfully, and therefore it is not known whether the file contains any viruses or not. This result code is caused either by an aborted scan,by a system error, such as memory allocation failure, or by an engine error. If a specific file triggers the same result again and again, and there does not seem to be any problem with the system, please send the file to your account manager for investigation.
BDAM_SCANRES_CORRUPTED	Scanning cannot be completed since the file is corrupted. Most likely the file is damaged to the point it cannot be unpacked or parsed. Therefore the engine cannot check the file and make sure the file is clean. This result code does not mean the file is safe to pass to the user. Some corrupted files could be fixed by application-specific tools (for example, some archivers can repair corrupted archives), and therefore an infected file could be extracted afterwards.

2.5.3 Threat Types

The threat type code allows the partner to classify the detected threat and to react accordingly. The library returns the threat type as a named value (define) described in the “Name” column. The server returns the threat type using a single uppercase character described in the “Protocol value” column (this is only relevant for the partners implementing the server protocol themselves).

Library definition	Protocol value	Description

BDAM_THREAT_TYPE_VIRUS	V	The object is a legitimate file infected by a virus. Sometimes, the infected file can be disinfected, and the legitimate file can be provided to the end user. However, not all types of infection can be disinfected – for example, the whole Trojan category cannot be disinfected. An important characteristic of a virus is that the virus replicates itself, ensuring its continuous spread. No other threat type replicates itself.
BDAM_THREAT_TYPE_SPYWARE	S	The object is a spyware application. Spyware is a large class of malicious applications with a huge range of malicious activity. This category includes applications which steal password and credit card information, online game account passwords, provide false security alerts giving the impression that the user's machine is in a critical state and demanding money for “fixes”, and so on. Usually, spyware gets installed without the user's informed consent. Since a spyware application is not a legitimate file infected by a virus, but a whole application coded with malicious intent, it cannot be disinfected.
BDAM_THREAT_TYPE_ADWARE	W	The object is an adware application. Adware is a class of malicious applications designed to display advertisements on the user's desktop, or in the web browser. Adware is also often used to monitor and report user browsing habits to the advertiser to bring more relevant ads. Some “free” applications available on the Web contain the adware payload, which is usually installed with user consent, while some other adware applications are installed without user consent. As with spyware, the adware application is not a legitimate infected file, and therefore it cannot be disinfected.
BDAM_THREAT_TYPE_DIALER	D	The object is a dialer. Dialers are applications which use the modem connected to the computer to dial premium-rate numbers. Usually they call either local pay-per-minute numbers or international numbers; per-minute costs have been known to reach several hundreds of dollars. Even if installed with user consent, they usually do not provide information about the real cost of the call.
BDAM_THREAT_TYPE_APP	A	The object is an application which is often installed and used for malicious purposes by 3rd parties. While the application itself is not malicious, experience shows that it poses a higher risk (compared to others) of being used for malicious purposes and of being installed without user consent. This category includes web or socks proxies, remote administration software and other types of software. Usually, the detected application is easy to install without user consent, and once installed, it has an option to be almost or completely hidden from the user. This object may not be malicious, and may be legitimately installed by a user, so it should not be quarantined or removed by default; the user should be asked instead. Obviously, since it's an application, it can only be removed, not disinfected.

2.6 Server

The BitDefender anti-malware server component is the core of this SDK. It is responsible for scanning the malware, loading and reloading the anti-malware database, processing the client requests and returning the scan results.

The server itself does not require root privileges, and, generally, the suggested way is to avoid running it under a root account. The best practice would be to run it under a dedicated user account. However, the partner application must either be run under the same account, or ensure that the temporary files it has written can be accessed by the server.

To handle multiple clients, the server supports two modes – multi-threaded mode and multi-process mode. In multi-threaded mode, the server starts a new thread for each connected client. In multi-process mode, the server forks a child process to handle each connected client. By default the server uses the multi-threaded mode, which is the preferred mode.

The server requires some information to start, which could be provided either by using the configuration file, or command-line options.

2.6.1 Environment

The server process tries to load the libraries included into the package. Those libraries are stored in the **bin/** directory. Therefore the system dynamic linker must be told where to find the server libraries. It could be done by setting the LD_LIBRARY_PATH environment variable to the full path to the server binary directory.

Example: assuming that the client-server SDK is unpacked into /opt directory, and the full path to the binary directory is /opt/clientserver/bin, the following bash command sets the LD_LIBRARY_PATH properly to run the server:

```
# export LD_LIBRARY_PATH=/opt/clientserver/bin
```

Avoid setting LD_LIBRARY_PATH to current directory (i.e. “.”) - not only this is insecure, but this also does not work if the server is configured to run in the Daemon mode, because it changes its working directory to the root directory, leading to “cannot load shared library” messages in the log file.

However if you install the SDK into the system directories (for example, you copied the binaries into /usr/bin and copied the libraries into /usr/lib), it is not necessary to set the LD_LIBRARY_PATH as those directories are typically included into system default library search path.

2.6.2 Command-line options

The server should be started using -c command line options with the configuration file.

Starting the server using configuration file */etc/bitdefender/server.conf*

```
#
# bdamserver -c /etc/bitdefender/server.conf
#
```

The following command-line options are supported:

-h
: Shows help message and exits

-v
: Shows program version and exits

-c configfile
: Loads the configuration from the *configfile* file

2.6.3 Configuration File (common options)

The server is configured using the configuration file. The file has a typical *option=value* format. Empty lines and the lines beginning with hash sign '#' are ignored.

This section lists all the options which are supported, together with allowed values.

2.6.3.1 PathAntimalwareRoot

This is a required option.

This option specifies the full path to the root of the anti-malware database directories. The SDK will create several subdirectories there (typically “1” and “2”) where it will download the actual anti-malware databases, as well as the file “active” indicating which database was updated the latest. The SDK will use one of them as working directory, and another one as update directory, so the actual directory the database is loaded from is never touched.

The path must be an absolute path to the directory. It must be writable and searchable.

The anti-malware database cannot be stored in the same directory as the anti-spam SDK, and hence this path cannot be the same as **PathAntispamRoot**.

2.6.3.2 PathAntispamRoot

If enabled, this option specifies the full path to the root of the anti-spam SDK directories. The SDK will create several subdirectories there (typically “1” and “2”) where it will download the actual anti-spam SDK, as well as the file “active” indicating which SDK was updated the latest. The Client-Server SDK will use one of them as working directory, and another one as update directory, so the actual directory the database is loaded from is never touched.

The path must be an absolute path to the directory. It must be writable and searchable. The server will attempt to load the anti-spam SDK from this directory, and if successful, will use it for scanning.

The anti-malware database cannot be stored in the same directory as the anti-spam SDK, and hence this path cannot be the same as **PathAntimalwareRoot**.

This option requires the LicenseAntispam option being set as well.

2.6.3.3 LicenseSerial

This is a required option.

This option specifies the BitDefender license serial number provided by your account manager.

2.6.3.4 LicenseAntispam

This option specifies the BitDefender anti-spam license serial number provided by your account manager. It is different from the LicenseSerial option above, and is only necessary if you want to use the anti-spam SDK to scan e-mails for spam.

2.6.3.5 Listen

At least one such option must be present.

This option specifies the TCP/IP port or Unix domain socket path to listen on. It is possible to specify multiple *Listen* parameters with different values. All supported and enabled protocols would be available on the listen socket(s).

Please note that by default only connections from 127.0.0.1 are accepted on the TCP port. If the server must accept remote connections, the HostAllow option needs to be used.

2.6.3.6 ServerMode

This is a required option.

This option specifies the server mode:

- If set to **preforked**, the server works in the preforking mode.
- If set to **threaded**, the server works in the multithreaded mode.
- If set to **worker**, the server uses the worker mode (requires Linux 2.6.8 or higher; only for ICAP)

2.6.3.7 MaxClients

This is a required option.

This option specifies the maximum number of simultaneously connected clients. Depending on the server mode this setting might not actually restrict any clients from connecting to the server, but rather allows the server to preallocate enough resources to handle the specified number of simultaneous connections. However since the resource allocation depends on the system, there is no guarantee that the server will be able to handle the requested number of clients.

One of the resources allocated with this setting is the file descriptor table. For each client connection the server needs to allocate up to two file descriptors. By default there are only 1024 file descriptors available, meaning the server will be able to handle maximum of roughly 500 concurrent connections. The system requires root privileges to raise this limit over 1024,

which means that if the server must handle more than 500 simultaneous connections, it must be run with root privileges (preferably in chroot).

Since for the preforked mode the server pre-forks the processes to handle the connections, in this mode this setting specifies how many processes will be preforked (i.e. for MaxClients=500 the server will prefork 500 processes). This also means that in this mode this setting actually limits the number of simultaneous connections the server will accept, meaning if this value is set to 100, the server will not accept more than 100 connections concurrently (however, as described above, it is not guaranteed the server may not be able to accept even 100 connections if the system is low on resources).

For the worker and threaded modes the number of simultaneously handled connections is not restricted by this setting, and the server will accept as many connections as allowed by the system.

2.6.3.8 HostAllow

If defined, this option specifies the IP address to accept connections from. 127.0.0.1 is always accepted. If “all” is used instead of the IP address, connections from any address are accepted. By default, only connections from 127.0.0.1 are accepted.

For Unix domain sockets this option is irrelevant as all connections are accepted.

2.6.3.9 PathTemp

If defined, this option specifies the temporary path the server will use. By default it uses an OS-specific temporary path.

2.6.3.10 PathPid

If defined, this option specifies the full path to the file where the application will write its process ID. In daemon mode the application will write the process ID after it forked into background. This file will be automatically removed when the server shuts down.

2.6.3.11 PathLog

If defined, this option specifies the full path of the file to write the log file to.

By default, no log file is written.

2.6.3.12 LogDebug

If defined, this option enables (1) or disables (0) the writing of debugging information to the log file. Disabled by default. Value 2 will also dump all network traffic between the server and the client, which is very useful for debugging the partner's own client implementation. Debug log grows extremely fast, and should be disabled in production mode.

By default debug information is not logged.

2.6.3.13 LogScanRequests

If defined, this option enables (1) or disables (0) the writing of scan result information to the log file, together with time measurements spent in various stages. Writes a single log line for each scanned object. Depending on the number of requests, it might be a good idea not to enable it in production mode.

The following time measurements are provided in the scan log line:

- total: total request processing time (sum of all fields below);
- recv: time spent on receiving the scanned content;
- corealloc: time spent waiting for the available engine (if the number of connections exceeds the value specified in MaxCores);
- scan: time spent scanning the object;
- send: time spent while sending the response back to the client.

If a field is missing, the time spent on this phase was less than 1 millisecond. Timing measurements are provided in milliseconds (1/1000 second) and only for ICAP requests.

By default scan requests are not logged.

2.6.3.14 LogAppend

If defined and set to true, this option makes the SDK opening the log files using the O_APPEND mode. In this mode it is possible to truncate the log file while the application is working (with possible data loss) without restarting the application.

However this mode is not compatible with NFS, and should not be enabled if the log files are stored at the NFS mounted directory.

By default this option is disabled.

2.6.3.15 Daemon

If defined, this option enables (1) or disables (0) daemonization, i.e. detaching from the terminal, and forking into background. The application tries to do as much as possible before actually forking into background, so its exit code status is still useful.

By default daemonization is enabled.

2.6.4 Configuration File (advanced options)

2.6.4.1 WorkerThreads

This option is required if ServerMode is set to worker.

If defined, this option specifies the number of worker threads spawned in worker server mode. The number which would give the best performance depends on the system and the objects scanned, and generally is in range 1 - 4 * the number of CPU cores, i.e. for quad core

machine the good number would be in 4-16 range.

2.6.4.2 MaxCores

This option cannot be used if ServerMode is set to preforked

If defined, this option specifies the maximum number of anti-virus engines server would preallocate to handle all the requests. Unlike MaxClients option, it does not restrict the number of connecting clients, only the number of anti-virus engines used internally. This means that if there are more threads processing the connections than available engines, some threads might wait until an engine is available for them. This setting only makes sense for the ICAP protocol, because threads tend to spend time doing network transfers, and limiting the number of the engines speeds up handling of request, and limits the memory consumed by the engines. It is not recommended to set it above 64 if the ICAP server is expected to handle more than70 simultaneous connections. The main use for this parameter is that it allows supporting more than 100 simultaneous threads without increasing memory requirements dramatically, as each engine instance could allocate up to 30Mb of memory in extreme cases.

2.6.4.3 ClientTimeout

If defined, this option specifies the amount in seconds the server will wait for client requests. When this amount is exceeded, if a client did not send any requests it is disconnected. If a client sent a request, the time is started again. It may be useful if KillIdleConnectionsOnUpdate option is not set, as otherwise a client hanging on a connection would prevent it from being updated in preforked mode.

By default there is no timeout, and all connections are kept permanently.

2.6.4.4 KillIdleConnectionsOnUpdate

This option only makes sense to use if ServerMode is set to preforked.

If defined and set to 1, this option instructs the server to close the connection right after the last request has been processed, so the child process can exit and a new process can be forked instead, which will contain a copy of new database. This option makes sense if ClientTimeout is not set, and the client uses persistent connections. In this case if the application performed the database update and needs to reload child processes, it will do it once the current request has been processed, the handling process will exit and a new process will be started, so the client can reconnect and use a new updated engine.

If this option is not set, the process will keep the copy of a new engine as long as the client keeps the connection. So if a client does not use persistent connections, or periodically reconnects, it may be ok not to use this option.

2.6.4.5 KeepEngineMemory

This option only makes sense to use if ServerMode is set to threaded or worker.

If defined, this option specifies the number of seconds engine internal memory buffers will be

kept after the last use, until they are freed. During request processing the engine tends to allocate memory for internal purposes (generated signature hash tables, loaded plugins, and so on), and it takes extra time (up to 10 seconds in extreme cases) to prepare and load all necessary data for the instance to work. This data, however, consumes memory, so if idle memory usage is important and the first scan on connection taking longer time is not important, this parameter should be set to something in 10-120 range. If idle usage memory is not important, or the application is not expected to spend a lot of time being idle, it is better to set this parameter to 0, which would mean this memory will not be freed. This will provide the best possible performance even on the first scan (as all required data is preloaded), but will keep much higher memory consumption when idle.

In preforked mode the application takes care of its memory automatically, and this option is not needed.

2.6.4.6 SocketSetNoDelay

If defined and set to 1, this option disables TCP Nagle algorithm on all connected sockets on server side. It might increase or decrease overall network performance, so it is hard to advice whether it should be set, as it depends on the network configuration, network equipment, traffic type and so on. By default Nagle is enabled.

This is advanced option, which should only be used by the partners who are experienced in fine-tuning networking applications.

2.6.4.7 SocketBufferSizeSend

If defined, this option specifies the socket send buffer size in bytes. If not set, the operating system default settings are used.

This is advanced option, which should only be used by the partners who are experienced in fine-tuning networking applications.

2.6.4.8 SocketBufferSizeRecv

If defined, this option specifies the socket receive buffer size in bytes. If not set, the operating system default settings are used.

This is advanced option, which should only be used by the partners who are experienced in fine-tuning networking applications.

2.6.5 Configuration File (AntiSpam options)

2.6.5.1 AntispamAggressivityLevel

If defined, this option overrides the default Anti-Spam SDK aggressivity level which is 5.

Increasing the aggressivity level increases the anti-spam detection rate but brings up the risk of false positives.

2.6.5.2 AntiSpamScanRetry

If defined and set to nonzero value, this option enables automatic retry for the anti-spam scans which resulted in the “CLOUD_NO_RESPONSE” error for the number of times specified. For example, if this option is set to 2, the SDK will automatically retry scanning the content for spam up to three times (first scan and two retries). A retry is only attempted for the “no response from the cloud” error, no retry will be attempted on success or any other error.

The SDK will wait some time between retries, specified in the AntiSpamScanRetryTimeout option.

By default the retry is disabled.

This option has no effect for the anti-malware scanning.

2.6.5.3 AntiSpamScanRetryTimeout

If defined, specifies the time **in milliseconds** the Anti-Spam SDK would wait between retries. Default timeout is 1000 milliseconds, i.e. 1 second.

This option has no effect for the anti-malware scanning nor when retry is disabled.

2.6.6 Configuration File (ICAP options)

Those options are only available when the ICAP server license is used.

2.6.6.1 ProtocolICAP

Specifies whether the ICAP protocol is supported, and which scan engines should be used.

If this option is commented out or set to *off,* the ICAP protocol support is disabled. This is default.

If this option is set to *antimalware*, the ICAP protocol support is enabled, and the content is scanned using the anti-malware engine.

If this option is set to *antispam*, the ICAP protocol support is enabled, and the content is scanned using the anti-spam engine.

If this option is set to *antimalware,antispam*, the ICAP protocol support is enabled, and the content is scanned using both the anti-malware and the anti-spam engines.

ONLY use the antispam option if you pass exclusively RFC822 e-mails via ICAP protocol, and not the regular web pages. Especially important is that you must not use this option if you connect Web gateway to the server, and scan the web pages. Doing so may trigger a significant amount of false positives.

2.6.6.2 ICAPMaxMemoryObjectSize

If set, this option specifies the maximum size in bytes of the processing object the ICAP server connection will keep in memory. If the object exceeds the specified size value, it will be dumped to the disk. This limit applies for a single connection, not for all connections as a whole. Setting this value too low will negatively affect performance. The default value is

1048576 (1Mb)

2.6.6.3 ICAPOutputChunkSize

If set, this option specifies the maximum chunk size in bytes of the ICAP reply to be sent back to the client. This is relevant only for clients which do not understand the "Allow: 204" option (squid does). Default value is 32768 bytes, values under 2048 are not supported.

2.6.6.4 ICAPHtmlTemplate

If set, this option specifies specifies the HTML template file which will be returned instead of the malicious object found. The file supports the following macros, which will be replaced by appropriate values before being sent:

%REASON%	A text message explaining why the file cannot be received. It will read like "Spyware Win32.Patched was detected" or "file is encrypted"
%MALWARENAME%	The name of a detected piece of malware. If no malware is detected, this will be an empty string.
%MALWARETYPE%	A type of detected malware. If no malware is detected, this will be an empty string.
%MALWARESTATUS%	A single word text description, such as "encryped" or "infected"
%REQUESTURL%	The URL requested by the client
%VERSION%	The ICAP server name and version

By default, the following template is used:

```
<html><head>
<title>The file you are trying to access cannot be accessed</title>
</head><body>
<h2>The file you are trying to access cannot be accessed
because %REASON%<br><br></h2>
<p>Malware name: %MALWARENAME%<br>
<p>Malware type: %MALWARETYPE%<br>
<p>Malware status: %MALWARESTATUS%<br>
```

2.6.6.5 ICAPSkipContentType

If set, this option defines a comma-separated list of content types which should be skipped. These content types will not be delayed and scanned, and they will be passed as soon as the header is parsed. Since relying on content type is not secure, this should be used as a temporary workaround for a reported issue until the fix is provided. The content type is matched against “Content-Type” header, and the ICAP server does not perform a check to ensure the actual content matches the content type reported by the web server.

By default, all content types are scanned.

2.6.6.6 ICAPBypassStreamMedia

If set to 1, this option instructs the server to skip known media streams during preview. This option provides a lower level of security, since media streams are bypassed without being scanned. However it might be necessary, as without it media streams are only played when they are received by the ICAP server completely, and then scanned. This might take a while for video streams, especially for streams such as live TV.

This options detects video streams using both content-type header, and the content data, by verifying that the video stream format matches the content type (if possible), so it provides more security than just adding the stream types into ICAPSkipContentType. However, it won't be able to protect against the vulnerabilities found in legitimate streams, as only the stream header is scanned.

By default, streams are not skipped.

2.6.6.7 ICAPScanArchives

If set, this option enables (1) or disables (0) archive scanning. If enabled, this option instructs the ICAP server to scan archives, which means the server will unpack them, and scan all the files inside it (and if it finds another archive inside it, it will unpack it and scan it and so on). The engine consider an archive any file which can contain zero or more files inside, and this includes not only traditional archives, such as RAR, ZIP or GZIP, but also other compound file types, such as MIME encoded email messages, CHM files, all Installers, and even OpenOffice documents. Therefore disabling this option offers less security, but better performance.

Even if archive scanning is disabled, the engine still scans some types of ZIP archives. This is not a bug, but intended behavior. This is done because some operating systems and common applications are able to open such archives directly, skipping the extracting phase which presupposes writing an unpacked content to the file system. Such scenarios have been exploited by malware writers, and they are common in independent industry tests. Not all ZIP archives are scanned when archive scanning is disabled; the engine determines heuristically whether the file should be scanned, using as criteria the number and size of files in the archive, whether the archive contains an executable file, and others. If the partner wants to make sure whether the option to disable archive scanning in their product works, it is recommended to use RAR archives instead of ZIP.

By default the archives are scanned.

2.6.6.8 ICAPScanEmaildb

If set, this option enables (1) or disables (0) e-mail databases scanning. Scanning e-mail databases slows down the engine, so if the partner application does not need them to be scanned, this option may be disabled. Otherwise it should be enabled.

The engine supports multiple e-mail database formats, including, but not limited to: DBX, MBX, PST, TNEF and other popular formats.

By default e-mail databases are **not** scanned.

2.6.6.9 PathInfectedLogICAP

If set, this option allows logging infected ICAP requests or responses into a separate log file. Only infected or suspicious verdicts are logged, together with the URL requested and the client IP (if available). This log is supplemental to the general bdamserver log, and does not depend on it.

This option only works when ProtocolICAP option is enabled.

2.6.7 Configuration File (Updater options)

The following options are responsible for the configuration of the internal updater.

2.6.7.1 UpdateURLAntivirus

If defined, this option specifies the URL the anti-malware database updates will be downloaded from. By default the update uses Bitdefender servers. This is fine for the SDK evaluation purposes, but for the commercial usage the partner may be required to use their own mirroring server. In this case this variable must point to the partner's mirroring server.

2.6.7.2 UpdateURLAntispam

If defined, this option specifies the URL the anti-spam SDK updates will be downloaded from. By default the update uses Bitdefender servers. This is fine for the SDK evaluation purposes, but for the commercial usage the partner may be required to use their own mirroring server. In this case this variable must point to the partner's mirroring server.

This option is only effective if the anti-spam SDK license is specified.

2.6.7.3 CheckUpdateInterval

If defined, this option specifies the interval in minutes between the anti-malware database update checks, meaning that if this option is set to 15 (default), the server will check for update every 15 minutes. If no update is available, the server will only download the index file.

If this option is set to zero the internal updater is disabled both for anti-spam and anti-virus components.

2.6.7.4 UpdateProxy

If defined, this option specifies the proxy server IP and port the updater should use. No prefixes should be used, just the IP and port separated by column, i.e. “192.168.0.1:3128”

This option only makes sense when the UpdateURL option is set.

2.6.7.5 UpdateProxyAuth

If defined, this option specifies the proxy authentication credentials. They must be specified in clean text either as *user:password* or *domain\user:password*. Both plain and NTLM authentication is supported.

This option only makes sense when the UpdateURL and UpdateProxy options are set.

2.6.8 Configuration File (SPAMD options)

2.6.8.1 ProtocolSPAMD

Specifies whether the SPAMD protocol is supported, and which scan engines should be used.

If this option is commented out or set to *off,* the SPAMD protocol support is disabled. This is default.

If this option is set to *antimalware*, the SPAMD protocol support is enabled, and the content is scanned using the anti-malware engine.

If this option is set to *antispam*, the SPAMD protocol support is enabled, and the content is scanned using the anti-spam engine.

If this option is set to *antimalware,antispam*, the SPAMD protocol support is enabled, and the content is scanned using both the anti-malware and the anti-spam engines.

2.6.9 Signals

By receiving SIGHUP the server process reloads the anti-malware database.

By receiving SIGTERM the server process shuts down.

2.6.10 Reloading Antimalware Database

The way server reloads the anti-malware database depends on the mode used.

2.6.10.1 If a multithreaded mode is used

If server is configured using multithreaded mode, the reload process uses the following sequence:

- The server holds all new connections, and all new scan requests. Active connections which do not have active scans are temporarily put on hold (but not closed), and new connections are placed in the listening queue, but not accepted.
- All active scans are completed as usual. Once a scan is completed and the scan result is reported, its connection is also being put on hold.
- Once all active scans are completed, the server reloads the database.
- When the database has been successfully reloaded, the server accepts all pending connections, and continues to handle all new scan requests.

2.6.10.2 If a preforked mode is used

If server is configured using preforked mode, the reload process uses the following sequence:

- The master process, which does not handle connections, reloads the database. At this

moment all active connections are processed, and new connections are accepted as usual.

- Once the database is successfully reloaded, the master process tells all children processes to terminate.
- Children processes finish the current scan, report the scan result, close the connection, and exit the process. If a child process is not currently running a scan but just keeps the active connection, the behavior depends on the KillIdleConnectionsOnUpdate option setting.
- For each exited child process the master process immediately forks a new child process, which will contain a new copy of the database. This copy accepts a new connection, and continues working. This continues until all “old” child processes are replaced by “new” processes.

2.7 Server Protocol

The server uses a text-based request-response protocol similar to HTTP. The client connects to the server, sends the request, receives the response, sends another request, receives another response, and then terminates the connection.

The client request is always a single line containing a text command with optional space-separated arguments. The client command should end in a CRLF, consisting of octals 015 012 or \r\n in C.

The server replies with one or more lines. Each line contains three digits representing the return code, the continuity symbol, and the optional parameters or text. The continuity symbol takes the fourth character position and could be either dash '-' or space. If the symbol is dash, the server reply does not end there, and at least one more line should be read and its continuity status checked. If the symbol is space, this is the last line.

The server also supports industry-standard ICAP protocol which is not described in this document.

2.7.1 Connecting to Server

The connection to the server is established using a stream socket, either AF_INET or AT_UNIX. Once the connection is established, the server is ready to receive requests. It is important to note that the server does not return any banners when the connection is established.

If a TCP/IP connection is used, the server might close the connection immediately after accepting it. This means the current IP address is not allowed by the server configuration to connect to the server. This possibility should be handled in the client code.

The following example shows a possible server session. Client requests are marked in red:

```
# telnet localhost 9000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

```
INFO
501 Incorrect info version
INFO 1
200 1 -1 2458088 1233296243
SCANFILE /etc/passwd
501 Incorrect syntax
SCANFILE 0 /etc/passwd
227
SCANFILE 63 /tmp/infected
322-V BD.TestSignature /tmp/infected=>single-infdis
326-V BD.TestSignature /tmp/infected=>single-infdis
322-V BD.TestSignature /tmp/infected=>single-infnondis
320-V BD.TestSignature /tmp/infected=>single-infnondis
220 V BD.TestSignature
```

2.7.2 Obtaining Server Runtime Information

2.7.2.1 Basic information (INFO 1)

To obtain the server basic runtime information, the INFO command should be issued by the client followed by the number 1:

```
CLIENT: INFO 1
SERVER: 200 1 expires amrecords amreleasetime
```

Field values:

expires

Represents the SDK license expiration time (as time_t - seconds after Epoch). If this information is not available, this field contains a question mark.

amrecords

Represents the total number of records in the antimalware database. If this information is not available, this field contains a question mark.

amreleasetime

Represents the antimalware database release time (as time_t - seconds after Epoch), meaning the time the database was released in the anti-malware Lab. This time may differ from the time the database was downloaded. If this information is not available, this field contains a question mark.

The following example shows how to obtain this information from the server:

```
INFO 1
200 1 1233256263 2459999 1233296263
```

If, irrespective of the reason, the information cannot be returned the server returns a 4XX or 5XX error code. The following table summarizes the possible errors:

Error Return Codes:

501 The requested block number is invalid.

2.7.2.2 Anti-malware engine update information (INFO 2)

To obtain the anti-malware engine update information, the INFO command should be issued by the client followed by the number 2:

```
CLIENT: INFO 2
SERVER: 200 2 attempt succeed perform errors upversion engversion
```

Field values:

attempt

Represents the last time (as time_t - seconds after Epoch) the SDK attempted to update the anti-malware database, even if the attempt failed, or did not download anything. Zero means no update was ever attempted.

succeed

Represents the last time (as time_t - seconds after Epoch) the SDK update attempt succeed. Success here means the update attempt did not result in error; an update which did not download anything because the SDK already has the latest version is considered successful. Zero means no update ever succeeded.

perform

Represents the last time (as time_t - seconds after Epoch) the SDK update successfully downloaded the new database version. Zero means no update was performed.

errors

Represents the number of errors the update attempt failed after the last successful update (see success above for the definition).

upversion

Represents the current version of the update available on the update server. This number typically increases by one or more after an update is performed.

engversion

Represents the anti-malware engine version as text string. This string may contain spaces and letters, so it should be parsed to the end of line. If this information is not available, this field contains a question mark.

The following example shows how to obtain this information from the server:

```
INFO 2
200 2 1447396892 1447396892 0 0 49607 AVCORE v2.1 Linux/x86_64 11.0.1.12 (Jan 21, 2015)
```

2.7.2.3 Anti-spam engine update information (INFO 3)

To obtain the anti-malware engine update information, the INFO command should be issued by the client followed by the number 3:

```
CLIENT: INFO 3
SERVER: 200 3 attempt succeed perform errors upversion engversion
```

Field values:

attempt

Represents the last time (as time_t - seconds after Epoch) the SDK attempted to update the anti-spam engine, even if the attempt failed, or did not download anything. Zero means no update was ever attempted.

succeed

Represents the last time (as time_t - seconds after Epoch) the SDK update attempt succeed. Success here means the update attempt did not result in error; an update which did not download anything because the SDK already has the latest version is considered successful. Zero means no update ever succeeded.

perform

Represents the last time (as time_t - seconds after Epoch) the SDK update successfully downloaded the new anti-spam engine version. Zero means no update was performed. This is a typical value as the engine updates are rare.

errors

Represents the number of errors the update attempt failed after the last successful update (see success above for the definition).

upversion

Represents the current version of the update available on the update server. This number typically increases by one or more after an update is performed.

engineversion

Represents the anti-spam engine version as text string. This string may contain digits and dots, but in future letters may be added, so it should be parsed to the end of line. If this information is not available, this field contains a question mark.

The following example shows how to obtain this information from the server:

```
INFO 3
200 3 1447396892 1447396893 0 0 30 2.4.4.205
```

2.7.3 Scanning File

To scan the local file, the SCANFILE command should be issued by the client:

```
CLIENT: SCANFILE options_number filename
SERVER: reply varies
```

Field values:

options_number

A decimal number representing the combination of the scanning options in the table. Options are specified by providing their number in the "protocol value" column. Multiple options should be combined using bitwise OR, and the resulting option should be provided as a single decimal number without leading zeros.

filename

An absolute pathname of the file to be scanned.

The SCANFILE command is used to scan a local file. The *options* parameter specifies how to scan the file, whether to attempt to disinfect it, if infected, and whether to report the scan progress.

If the scan progress option was selected, the server will send in-progress scan information followed by the scan result. If the scan progress option was not selected, only the scan result will be sent.

The following example shows how to scan a file using all scanning options set, except for in-progress information:

```
SCANFILE 31 /tmp/infected
220 V BD.TestSignature
```

The following example shows how to scan a file using all scanning options set, including in-progress information:

```
SCANFILE 63 /tmp/infected
322-V BD.TestSignature /tmp/infected=>single-infnondis
320-V BD.TestSignature /tmp/infected=>single-infnondis
220 V BD.TestSignature
```

If the file cannot be scanned for some reason, the server returns a 4XX or 5XX error code The following table summarizes the possible errors:

Error Return Codes:

501 The command syntax is incorrect (see server log for details)

405 The file path is not absolute (i.e. does not start with /)

401 The server cannot find the file

402 The server cannot access the file and gets the "access denied" error

403 The server cannot access the file but the error is not "access denied"

404 The scanned object is not a regular file

2.7.4 Scanning Shared Memory Segment

To scan the shared memory segment, the SCANSHM command should be issued by the client:

```
CLIENT: SCANSHM options_number shm_key shm_size object_size
SERVER: reply varies
```

Field values:

options_number

A decimal number representing the combination of the scanning options in the table. The options are specified by the number in the "protocol value" column. Multiple options should be combined using bitwise OR, and the resulting option should be provided as a single decimal number without leading zeros.

shm_key

A shared memory key obtained from the *ftok()* function, which was used to create the shared memory segment, represented in "pointer-style" format, i.e. using the %p format string.

shm_size

A shared memory segment size, which was used to create the shared memory segment, represented in decimal format.

object_size

The scanned object size (may be less than the shared memory segment size) represented in decimal format.

The SCANSHM command is used to scan a preallocated shared memory segment. The *options* parameter specifies how to scan the data, whether to attempt to disinfect it, if infected, and whether to report the scan progress. The segment is determined by key and size; the size of the object,helps in cases when a preallocated segment is used. The shared memory scan supports disinfection, but the disinfection attempt will fail in the unlikely event of the disinfection changing the file size.

If the scan progress option was selected, the server will send the in-progress scan information followed by the scan result. If the scan progress option was not selected, only the scan result will be sent.

The following example shows how to scan a segment using all scanning options set, except for in-progress information:

```
SCANSHM 31 8da3947 4096 898
220 V BD.TestSignature
```

The following example shows how to scan a file using all scanning options set, including in-progress information:

```
SCANSHM 63 8da3947 4096 898
```

```
322-V BD.TestSignature /tmp/infected=>single-infnondis
320-V BD.TestSignature /tmp/infected=>single-infnondis
220 V BD.TestSignature
```

If, irrespective of the reason, the file cannot be scanned , the server returns a 4XX or 5XX error code. The following table summarizes the possible errors:

Error Return Codes:

501 The command syntax is incorrect (see server log for details)

404 The shared memory segment cannot be attached

2.7.5 Retrieving List of Malware Names

To retrieve the list of known malware, the ENUMRECORDS command should be issued by the client:

```
CLIENT: ENUMRECORDS
SERVER: 310-virusname1
SERVER: 310-virusname2
…
SERVER: 310-virusnamelast
SERVER: 200 done
```

The ENUMRECORDS command forces the database record enumeration, and returns the list of all known viruses. Each record is returned as a separate line following the “310-” reply code and the continuity prefix. The list is ends in code 200, which does not include a virus name.

Because of internal restrictions the server can only run a single enumeration operation at a time. This means that, although it is very unlikely to be of any use, the enumeration cannot be run by two clients simultaneously. The amount of data transferred is also significant – with more than 2M records in the database and an average of 25 bytes per line the total traffic between client and server for the enumeration process will exceed 50Mbytes.

The following example shows how to use an enumeration with a cut-off middle:

```
ENUMRECORDS
310-Adware.007guard.A
310-Adware.007guard.B
…
310-trojan.peed.JUL
200 done
```

2.7.6 Triggering an update

To trigger an update the following command could be used:

```
CLIENT: UPDATE
SERVER: 200 Started
```

The UPDATE command forces the SDK to attempt the update for all enabled databases. If the new update is available, it will be downloaded and installed, and the server will reload the database, possibly closing the connection depending on settings.

The requested update happens asynchronously, no progress information is sent to the client and it is not informed when and whether the update succeeds. This information, if required, could be obtained by polling the INFO 2/INFO 3 output.

The update only starts if CheckUpdateInterval is set to nonzero value, and resets the CheckUpdateInterval timer (meaning if it is set to 15 minutes and UPDATE command was issued at 10th minutem the next automatic check for update will be performed in 15 minutes).

2.7.7 Closing Server Connection

To close the server connection the partner application needs to just close the socket. There is no QUIT command, and it is not necessary. The socket can be closed anytime, even in the middle of an enumeration or of an active scan.

2.7.8 In-progress Information

If the scan progress option bit in the scan options was set, the server provides in-progress information for the current scan. This information includes reporting the status of non-clean files. Each file is reported once, unless detection is enabled (in which case the same file may be reported when found infected and then disinfected or when disinfection failed). The in-progress information is temporary, and it should not be used as a final verdict.

Progress information uses codes 320-327. Each code represents a specific status, and it may have extra information, not just the filename. The table below summarizes the codes and the associated extra information. All extra parameters mentioned in the associated column are always present for a specific return code. The parameters are separated by a single space, except for the last parameter , which is filename, and which can contain spaces.

Table legend: **filename** – the reported file name in a custom form; **malwarename** – the malware name, **threattype** – a single character representing the threat in the “Protocol value” column in the threat types table.

Code	Extra parameters	Description
326	**malwarename threattype filename**	The file was found to be infected during scanning, the disinfection option was enabled, and the disinfection attempt succeed. The file is no longer infected.
320	**malwarename threattype filename**	The file was found to be infected during scanning, the disinfection option was enabled, and the disinfection attempt failed. The file is still infected
322	**malwarename threattype**	The file was found to be infected during scanning, and either the disinfection option was disabled, or the disinfection attempt failed.

	filename	
323	**malwarename filename**	Something similar to a previously known piece of malware was found inside the file during proactive scanning. Please send this file to the BitDefender anti-malware laboratory for further classification. Because this malware type has not been added to the anti-malware database yet, it is not possible to disinfect the file. Generally, the BitDefender proactive scanning method results in very few false positives. This means that, statistically, this file is most likely to be infected, and that it should be treated accordingly. It is still possible, however, that this is a legitimate file, so the suggested action would be to quarantine it and to check it for viruses again later on.
325	**filename**	Scanning cannot be completed since the file is password-protected, and decryption is not possible without knowing the password. The file has not been not scanned.
321	**filename**	Scanning failed. This code means that the file scan was not completed successfully, and therefore it is not known whether the file contains any viruses or not. This result code is caused either by an aborted scan,by a system error, such as memory allocation failure, or by an engine error. If a specific file triggers the same result again and again, and there does not seem to be any problem with the system, please send the file to your account manager for investigation.
321	**filename**	Scanning cannot be completed since the file is corrupted. Most likely, the file is damaged to the point it cannot be unpacked or parsed. Therefore, the engine cannot check the file and make sure the file is clean. This result code does not mean the file is safe to pass to the user. Some corrupted files could be fixed by application-specific tools (for example, some archivers can repair corrupted archives), and, therefore, an infected file could be extracted afterwards.

2.7.9 Scan Result

After the scan is completed, the server returns the scan result. Every scan command always returns the scan result if it didn't return an error (i.e. code 4XX or 5XX). If the in-progress scan option was enabled, the scan result follows the progress. Otherwise, only the result is returned.

The scan result is returned using the server code 220-227. Each code represents a specific status, and it may contain extra information. The table below summarizes the codes and the associated extra information. All extra parameters mentioned in the associated column are always present for a specific return code. The parameters are separated by a single space, except for the last parameter, which is filename and could contain spaces.

Table legend: **malwarename** – the malware name, **threattype** – a single character representing the threat from the “Protocol value” column in the threat types table.

Code	Extra parameters	Description
227		The scanning has been completed, and no known piece of malware was found

		in the file. This code is also returned for the files which were not scanned at all because the corresponding option was disabled.
226	**threattype malwarename**	The file was found to be infected during scanning, the disinfection option was enabled, and the disinfection attempt succeed. The file is no longer infected.
220	**threattype malwarename**	The file was found to be infected during scanning, the disinfection option was enabled, and the disinfection attempt failed. The file is still infected
222	**threattype malwarename**	The file was found to be infected during scanning, and either the disinfection option was disabled, or the disinfection attempt failed.
223	**malwarename**	Something similar to a previously known piece of malware was found inside the file during proactive scanning. Please send this file to the BitDefender anti-malware laboratory for further classification. Because this malware type has not been added to the anti-malware database yet, it is not possible to disinfect the file. Generally, the BitDefender proactive scanning method results in very few false positives. This means that, statistically, this file is most likely infected, and that it should be treated accordingly. It is still possible, however, for this to be a legitimate file, so the suggested action would be to quarantine it and to check it for viruses again later on.
225		Scanning cannot be completed since the file is password-protected, and decryption is not possible without knowing the password. The file has not been not scanned.
221		Scanning failed. This code means that the file scan was not completed successfully, and therefore it is not known whether the file contains any viruses or not. This result code is caused either by an aborted scan, by a system error, such as memory allocation failure, or by an engine error. If a specific file triggers the same result again and again, and there does not seem to be any problem with the system, please send the file to your account manager for investigation.
224		The scanning cannot be completed since the file is corrupted. Most likely the file is damaged to the point it cannot be unpacked or parsed. Therefore the engine cannot check the file and make sure the file is clean. This result code does not mean the file is safe to pass to user. Some corrupted files could be fixed by application-specific tools (for example, some archivers can repair corrupted archives), and therefore an infected file could be extracted afterwards.

2.7.10 Return Codes

The server returns the information using the return codes. This code is represented by the three-digit number each server reply starts with. Generally, codes are divided into categories by their first number, which can only be 2, 3, 4 or 5 for now :

- Code 2xx indicates that the command has been successfully completed, and that the

requested information has been provided.

- Code 3xx indicates that the command is being executed, and that in-progress information has been provided.
- Code 4xx indicates that the command is valid, but that it cannot be executed because of some problems (for example the scanned file does not exist). Once this problem is fixed, the command can be repeated.
- Code 5xx indicates that the command is not valid, and that it cannot be executed as requested. This includes syntax errors and invalid/unsupported options.

2.8 Client Library API Reference

This section describes the client library API.

2.8.1 Note on Thread Safety

Generally, a single client instance is not thread-safe, and should only be used by a single thread at any specific time. However, all the library API functions are thread-safe, and could be called simultaneously from multiple threads, assuming that each thread which calls the function uses a unique client instance.

An attempt to call the library function using the client instance which is currently used for another library function will result in undefined behavior.

2.8.2 BDAMClient_Create

Prototype:

```
BDAMClient * BDAMClient_Create();
```

Description:

This function creates the client instance, initializes it, and allocates the necessary memory. Each instance uses approximately 5Kb of memory.

Return Value:

This function returns the pointer to a new client on success, and zero on failure, in which case the error is related to memory allocation.

Server interaction:

This function does not interact with the server.

2.8.3 BDAMClient_Destroy

Prototype:

```
int BDAMClient_Destroy( BDAMClient * client );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

Description:

This function closes the established connection (if established), and frees the memory allocated by the client instance. After this call, the client instance is destroyed, and should not be used anymore.

Return Value:

This function returns zero on success, and a non-zero error code on failure. Currently, this function always returns success, but the partner application is encouraged to check the error value in case further changes return errors.

Server interaction:

This function closes the active server connection, if any.

2.8.4 BDAMClient_Connect

Prototype:

```
int BDAMClient_Connect( BDAMClient * client,
                        const char * peeraddr_or_path );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

peeraddr_or_path

[in] IPv4 IP address and port separated by colon, or a full absolute path to the Unix domain socket the server listens on.

Description:

This function establishes a connection to the BitDefender antimalware server. Depending on the argument, the connection is established either using the IPv4 socket (when the argument is in the form of ip:port) or the Unix domain socket (when the argument is in the form of an absolute path). If the client instance is already connected, the active connection will be closed and the function will try to connect again.

Return Value:

This function returns zero on success, and a non-zero error code on failure.

Common errors:

BDAM_ERROR_INVALIDPARAM	For Unix domain sockets: the path is not absolute, or it exceeds the system-allowed length. For IPv4 sockets: the peer address is not a valid ip:port string, an IP address is invalid (domain name cannot be used), or the port number is invalid.
BDAM_ERROR_SYSTEM	The socket cannot be created.
BDAM_ERROR_CONNECTION	The connection cannot be established. The system *errno* variable contains the error produced by connect() call.

Server interaction:

This function establishes a connection with the server.

2.8.5 BDAMClient_Info

Prototype:

```
int BDAMClient_Info( BDAMClient * client,
                     time_t * license_expiration_time,
                     time_t * amdb_update_time,
                     unsigned long * amdb_records );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

license_expiration_time

[out] A pointer to the time_t variable which will contain the license expiration time and date. If this information is not available, it reads -1.

amdb_update_time

[out] A pointer to the time_t variable which will contain the antimalware database release date and time. If this information is not available, it reads -1.

amdb_records

[out] A pointer to the variable which will contain the total number of records in the anti-malware database. If this information is not available, it reads 0.

Description:

This function retrieves current runtime information from the BitDefender antimalware server. The information returned contains the number of antimalware database records, the antimalware database release time/date and the license expiration date (both as time_t-like values representing a number of seconds since Jan 1st, 1970) in relation to the GMT time zone. The antimalware release time represents the time when the database entered the QA process, so it will always be at least 30 minutes past. It does not represent the time the

partner application performed the update or reloaded the database, as the partner application knows those times and can store them itself.

Return Value:

This function returns zero on success, and a non-zero error code on failure.

Common errors:

BDAM_ERROR_SYNTAXREPLY	The antimalware server reply was not recognized. Usually, this indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_OVERFLOW	The input buffer has been overflown. Usually, it indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_SYSTEM	An attempt to poll, read from or write to the client socket generated a system error.
BDAM_ERROR_CONNECTION	The connection was not established or has been closed by the antimalware server, usually because the server restarted or because of a timeout, if the server was configured to use timeouts.

Server interaction:

This function interacts with the server, and requires an established connection.

2.8.6 BDAMClient_InfoUpdatesAV

Prototype:

```
int BDAMClient_InfoUpdatesAV( BDAMClient * client,
                              time_t * update_attempted,
                              time_t * update_succeed,
                              time_t * update_performed,
                              unsigned int * failed,
                              unsigned int * currentver,
                              const char ** enginever );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

update_attempted

[out] Returns the last time the SDK attempted to update the anti-malware engine,

even if the attempt failed, or did not download anything. Zero means no update was ever attempted.

update_succeed

[out] Returns the last time the SDK update attempt succeed. Success here means the update attempt did not result in error; an update which did not download anything because the SDK already has the latest version is considered successful. Zero means no update ever succeeded.

update_performed

[out] Returns the last time the SDK update successfully downloaded the new anti-malware database version. Zero means no update was performed. This is a typical value as the engine updates are rare.

failed

[out] Returns the number of errors the update attempt failed after the last successful update (see success above for the definition).

currentver

[out] Returns the current version of the update available on the update server. This number typically increases by one or more after an update is performed.

enginever

[out] Returns the anti-malware engine version as text string. This string may contain letters, spaces, digits and dots. If this information is not available, this field contains a question mark.

Description:

This function retrieves the current runtime information from the BitDefender antimalware server about the status of the anti-malware database updates.

The difference between the update timestamps it the following:

- *update_attempted* changes as soon as the update process starts - even if it is aborted because of no network connectivity, or does not result in any data downloaded.
- *update_succeed* changes when the update process is finished without errors. It may download a new update, or just perform a check and download nothing if the current version is the same as on the server,
- *update_performed* only changes when a new update has been downloaded and installed.

The *enginever* value returned by this function is only valid until any other SDK function is called on the same client handle.

Return Value:

This function returns zero on success, and a non-zero error code on failure.

Common errors:

BDAM_ERROR_SYNTAXREPLY	The antimalware server reply was not recognized. Usually, this indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_CONNECTION	The connection was not established or has been closed by the antimalware server, usually because the server restarted or because of a timeout, if the server was configured to use timeouts.

Server interaction:

This function interacts with the server, and requires an established connection.

2.8.7 BDAMClient_InfoUpdatesAS

Prototype:

```
int BDAMClient_InfoUpdatesAS( BDAMClient * client,
                              time_t * update_attempted,
                              time_t * update_succeed,
                              time_t * update_performed,
                              unsigned int * failed,
                              unsigned int * currentver,
                              const char ** enginever );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

update_attempted

[out] Returns the last time the SDK attempted to update the anti-spam engine, even if the attempt failed, or did not download anything. Zero means no update was ever attempted.

update_succeed

[out] Returns the last time the SDK update attempt succeed. Success here means the update attempt did not result in error; an update which did not download anything because the SDK already has the latest version is considered successful. Zero means no update ever succeeded.

update_performed

[out] Returns the last time the SDK update successfully downloaded the new anti-spam engine version. Zero means no update was performed. This is a typical value as the engine updates are rare.

failed

[out] Returns the number of errors the update attempt failed after the last successful update (see success above for the definition).

currentver

[out] Returns the current version of the update available on the update server. This number typically increases by one or more after an update is performed.

enginever

[out] Returns the anti-spam engine version as text string. This string may contain digits and dots, but in future letters may be added, so it should be parsed to the end of line. If this information is not available, this field contains a question mark.

Description:

This function retrieves the current runtime information from the BitDefender anti-spam server about the status of the anti-spam engine updates.

The difference between the update timestamps it the following:

- *update_attempted* changes as soon as the update process starts - even if it is aborted because of no network connectivity, or does not result in any data downloaded.
- *update_succeed* changes when the update process is finished without errors. It may download a new update, or just perform a check and download nothing if the current version is the same as on the server,
- *update_performed* only changes when a new update has been downloaded and installed.

The *enginever* value returned by this function is only valid until any other SDK function is called on the same client handle.

Return Value:

This function returns zero on success, and a non-zero error code on failure.

Common errors:

BDAM_ERROR_SYNTAXREPLY	The antimalware server reply was not recognized. Usually, this indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_CONNECTION	The connection was not established or has been closed by the antimalware server, usually because the server restarted or because of a timeout, if the server was configured to use timeouts.

Server interaction:

This function interacts with the server, and requires an established connection.

2.8.8 BDAMClient_SetOption

Prototype:

```
int BDAMClient_SetOption( BDAMClient * client,
                          int option,
                          int value );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

option

[in] An option to change.

value

[in] The new value of the option.

Description:

This function changes the scanning options of a specified client instance. If they are boolean options, non-zero values will enable them, while zero values will disable them. The options are stored inside the client instance, and they are only used when the scanning function is called. Therefore, changing the options is very fast and does not require a server connection. The changed options will be preserved either until a new change is performed, or until the client instance is destroyed. Reconnecting to the server does not change the options.

The scanning options are instance-specific, which means that each client instance has its own scanning options. If the partner application wants all client instances to have the same options, they should be explicitly set for each client instance, separately.

Return Value:

This function returns zero on success, and a non-zero error code on failure.

Common errors:

BDAM_ERROR_INVALIDPARAM The *value* argument is invalid for this option.

BDAM_ERROR_INVALIDOPTION The *option* argument does not specify a valid option.

Server interaction:

This function does not interact with the server, and it does not require an established connection.

2.8.9 BDAMClient_SetCallback

Prototype:

```
int BDAMClient_SetCallback( BDAMClient * client,
                            BDAMClientCallback callback,
                            void * ctx );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

callback

[in] A pointer to the callback function or 0

ctx

[in] The callback context

Description:

This function changes the in-progress callback of a specified client instance. If the *callback* argument is non-zero, the callback is enabled, and it will be called from the scanning functions during the scan. If the *callback* argument is zero, the callback will not be called.

The callback pointer and context are stored inside the client instance, and they are only used when the scanning function is called. Once set, the callback will be kept until set again, or until the client instance is destroyed. Reconnecting to the server does not change the callback.

The callback context argument, *ctx*, is stored and provided directly to the callback as-is. It is not used by the library or server, and it can have any value.

The callback is instance-specific, which means that each client instance has its own callback pointer. If the partner application wants all client instances to call the same callback, it should explicitly set this callback for each client instance.

Return Value:

This function returns zero on success, and a non-zero error code on failure. Currently, this function always returns success, but the partner application is advised to check the return value in case the future implementation returns an error.

Server interaction:

This function does not interact with the server, and it does not require an established connection.

2.8.10 BDAMClient_ScanFile

Prototype:

```
int BDAMClient_ScanFile( BDAMClient * client,
                         const char * filename,
                         int *scanStatus,
```

```
int * threatType,
const char **threatName );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

filename

[in] An absolute pathname of the file to scan.

scanStatus

[out] A pointer to the variable which will contain the scan result, assuming the scan is completed successfully.

threatType

[out] A pointer to the variable which will contain the last detected threat type,assuming the scan is completed successfully and a malicious code was found. If more than one piece of malware was found, the the last one's type will be stored.

threatName

[out] A pointer to the variable which will contain the null-terminated UTF8-encoded char string on Unix representing the name of the last detected piece of malware, assuming the scan is completed successfully and malicious codes were found.

Description:

This function is used to scan files on disk. The scan is done synchronously, which means the function will not return until the scan is completed. If the scan is completed successfully, it returns the scan result. If the scanned file contains malware, it will also return information about the threat type and the names of the detected malware applications. It supports automatic disinfection, and calls in-progress callback.

To scan the file, it must be accessible to the server. This includes permissions and path access. Since the server's current directory is generally unknown, the function requires an absolute file path. However if the server is run in a chrooted environment, the path to the file as the server sees it and as the client sees it may not match. In this case, the absolute path must represent the way the server could access the file. For example, if the server is chrooted into path */var/lib/bdavserver*, and the client stores the file with the absolute path */var/lib/bdavserver/tmp/1.tmp*, the *filename* argument should contain */tmp/1.tmp* as this is how the server could access the file.

Automatic disinfection is possible if scanning option BDAM_SCANOPT_DISINFECT is set. The server needs to be able to write into the file if this option is enabled. If the disinfection succeeds, the partner application needs to reread the file content, flushing the buffers, if necessary. The file size might also change after disinfection, although rarely.

If a malicious object is found, its name will be returned in the *threatName* variable. The string is allocated internally and is guaranteed to be available until the next scanning function is called, or until the client instance is destroyed. The partner application should not deallocate the pointer returned in this variable.

Return Value:

This function returns zero on success, and a non-zero error code on failure. “Success” does not mean the file is clean – it merely means that the file was scanned successfully. The output variables contain the scan result.

Common errors:

BDAM_ERROR_SCAN_NOTFOUND	The antimalware server cannot find the specified file.
BDAM_ERROR_SCAN_NOACCESS	The antimalware server cannot access the specified file because of lacking permissions or other reasons, or the specified file is not a regular file.
BDAM_ERROR_SYNTAXREPLY	The anti-malware server reply was not recognized. Usually, tjis indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_OVERFLOW	The input buffer has been overflown. Usually, this indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_SYSTEM	An attempt to poll, read from or write to the client socket generated a system error.
BDAM_ERROR_CONNECTION	The connection was not established or has been closed by the antimalware server, usually because the server restarted or because of a timeout, if the server was configured to use timeouts.

Server interaction:

This function interacts with the server, and it requires an established connection.

2.8.11 BDAMClient_ScanSharedMemory

Prototype:

```
int BDAMClient_ScanSharedMemory( BDAMClient * client,
                                 unsigned long shmKey,
                                 unsigned long shmSize,
                                 unsigned long objectSize,
                                 int * scanStatus,
                                 int * threatType,
                                 const char **threatName );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

shmKey

[in] A shared memory key obtained from the *ftok()* function which was used to create the shared memory segment.

shmSize

[in] A shared memory segment size which was used to create the shared memory segment.

objSize

[in] The scanning object size (may be lower than the shared memory segment size).

scanStatus

[out] A pointer to the variable which will contain the scan result, assuming the scan is completed successfully.

threatType

[out] A pointer to the variable which will contain the last detected threat type,assuming the scan is completed successfully and a malicious code was found. If more than one piece of malware was found, the the last one's type will be stored.

threatName

[out] A pointer to the variable which will contain the null-terminated UTF8-encoded char string on Unix representing the name of the last detected piece of malware, assuming the scan is completed successfully and malicious codes were found.

Description:

This function is used to scan the files stored in the shared memory. The partner application should allocate the shared memory segment, copy the content there, and call the scanning function providing the necessary data for the segment to be accessed. The scan is done synchronously, which means the function will not return until the scan is completed. If the scan is completed successfully, it returns the scan result. If the scanned file contains malware, it will also return information about the threat type and the names of the detected malware applications. The shared memory scan supports disinfection, but the disinfection attempt will fail in the unlikely event the disinfection changes the file size. It could call the in-progress callback, but this is not recommended because of performance issues. Shared memory scanning tends to be up to twice faster than filename scanning.

If a malicious object is found, its name will be returned in the *threatName* variable. The string is allocated internally and it is guaranteed to be available until the next scanning function is called, or until the client instance is destroyed. The partner application should not deallocate the pointer returned in this variable.

Return Value:

This function returns zero on success, and a non-zero error code on failure. “Success” does not mean the file is clean – it merely means that the file was scanned successfully, and that the output variables contain scan result.

Common errors:

BDAM_ERROR_SCAN_NOACCESS	The antimalware server cannot attach the specified shared memory segment.

BDAM_ERROR_SYNTAXREPLY	The antimalware server reply was not recognized. Usually, this indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_OVERFLOW	The input buffer has been overflown. Usually, this indicates a bug in the library code or on the antimalware server. If the library source code was not modified, please report to BitDefender.
BDAM_ERROR_SYSTEM	An attempt to poll, read from or write to the client socket generated a system error.
BDAM_ERROR_CONNECTION	The connection was not established or has been closed by the antimalware server, usually because the server restarted or because of a timeout, if the server was configured to use timeouts.

Server interaction:

This function interacts with the server, and requires an established connection.

2.8.12 BDAMClient_StartUpdate

Prototype:

```
int BDAMClient_StartUpdate( BDAMClient * client );
```

Arguments:

client

[in] A client instance allocated by the BDAMClient_Create function call.

Description:

This function forces the SDK to attempt the update for all enabled databases. If the new update is available, it will be downloaded and installed, and the server will reload the database, possibly closing the connection depending on settings.

The requested update happens asynchronously, no progress information is sent to the client and it is not informed when and whether the update succeeds. This information, if required, could be obtained by polling the BDAMClient_InfoUpdatesAV output.

The update only starts if CheckUpdateInterval is set to nonzero value, and resets the CheckUpdateInterval timer (meaning if it is set to 15 minutes and UPDATE command was issued at 10th minutem the next automatic check for update will be performed in 15 minutes).

Return Value:

This function returns zero on success, and a non-zero error code on failure.

Server interaction:

This function interacts with the server, and requires an established connection.

2.8.13 In-progress Callback

It is possible to instruct the server to report the scanning progress by calling the callback function specified by the partner. It only makes sense during archive scanning, and the provided result is an in-progress result, which is temporary and could change. To avoid major slowdowns, in-progress callback only report non-clean files. However, it is still recommended to completely avoid using it for any application where performance expectations are high.

To use the callback, the partner application must declare the callback function with the following prototype:

```
void (*BDAMClientCallback) (const char * filename,
                            int status,
                            const char * threatname,
                            int threattype,
                            void * ctx);
```

Arguments:

filename

[in] The reported file. May be the original file or a specially formatted file in the case of a file in an archive. It could also be 0 for shared memory segments.

status

[in] The scan result, such as the status of the reported file.

threatname

[in] If the *status* reported detected threats, this argument will contain the threat name.

threattype

[in] [in] If the *status* reported detected threats, this argument will contain the threat type.

ctx

[in] The context provided by the partner application while setting the scan callback.

The callback is called in the context of the thread which called the scanning function. The server continues to scan the file while the callback is processed, and the scan is not stopped.

If the same callback is supposed to be called from different client instances, it must be thread-safe.

2.8.14 Error Codes

The following error codes might be returned by SDK. If the reason for the error is not clear, in most cases it is very helpful to enable the debugging log in the server configuration, and to check the server log file.

2.8.14.1 BDAM_ERROR_INVALIDPARAM

One or more function arguments are invalid. Usually, this means there is an error in the partner application.

2.8.14.2 BDAM_ERROR_SYSTEM

A socket-related system call returned an error. Might indicate problems with the system, such as extreme system conditions.

2.8.14.3 BDAM_ERROR_CONNECTION

Either the connection to the antimalware server cannot be established, was not established, or was terminated. Most likely, the anti-malware server was not started, or was shut down while there were active connections.

2.8.14.4 BDAM_ERROR_OVERFLOW

The library input buffer has overflown. If the library source code hasn't been changed, most likely, there is a bug in the library. Please report to BitDefender.

2.8.14.5 BDAM_ERROR_SYNTAXREPLY

The server issued a reply the library can't understand, or the command issued by the library was incorrect. If the library source code hasn't been changed, most likely, there is a bug in the library. Please report to BitDefender.

2.8.14.6 BDAM_ERROR_INVALIDOPTION

The partner application attempts to set an invalid option.

2.8.14.7 BDAM_ERROR_SCAN_NOTFOUND

The partner application attempts to scan the file which the server cannot access using the file name provided, because the system reported the file does not exist.

2.8.14.8 BDAM_ERROR_SCAN_NOACCESS

When scanning a file, the server cannot access the file using the file name provided because the file name is not absolute, the file is not a regular file, or the file exists, but cannot be accessed because of some other reason. The server log usually contains the necessary details.

When scanning shared memory, the server cannot access the requested shared memory segment using the provided credentials.

2.9 Client

The SDK package provides a client executable, which is based on the client library and uses the library for client-server communication. The client executable can query the server information, and scan files. Its output is designed to be easily parseable, and therefore it could be used for quick server integration, or for testing purposes, such as testing new database updates before providing them to the customers.

The client executable is also provided with the source code.

2.9.1 Command-line Options

The client should be started by providing required and optional parameters in the command line. The required parameters include the server Ipv4:port or Unix domain socket path, and either a file name to scan, or the -i switch used to get the runtime information from the server. The client can be started under any user as it does not need to access the to the files being scanned.

This is an example of starting the client to connect to the server running on localhost, port 9000 and scanning the */etc/passwd* file:

```
#
# bdamclient -p 127.0.0.1:9000 /etc/passwd
#
```

The following command-line options are supported:

-p path or ip:port
: Specifies the peer address (ip:port) or Unix domain socket path to the running bdamserver. Required.

-i
: Retrieves and shows run-time information about the server in a human-readable format.

-m
: Retrieves and shows run-time information about the server in a script-parseable format.

-h
: Prints help message and exist.

-u
: Tell the server to initiate the database update(s)

-oa
: Enables archives scanning

-op
: Enables packing files scanning

-od
: Enables disinfection

-oe
: Enables e-mail database scanning

-ov

Prints scan progress listing non-clean files

-on

Does not print the file name in progress and result messages, making it much easier to parse the output