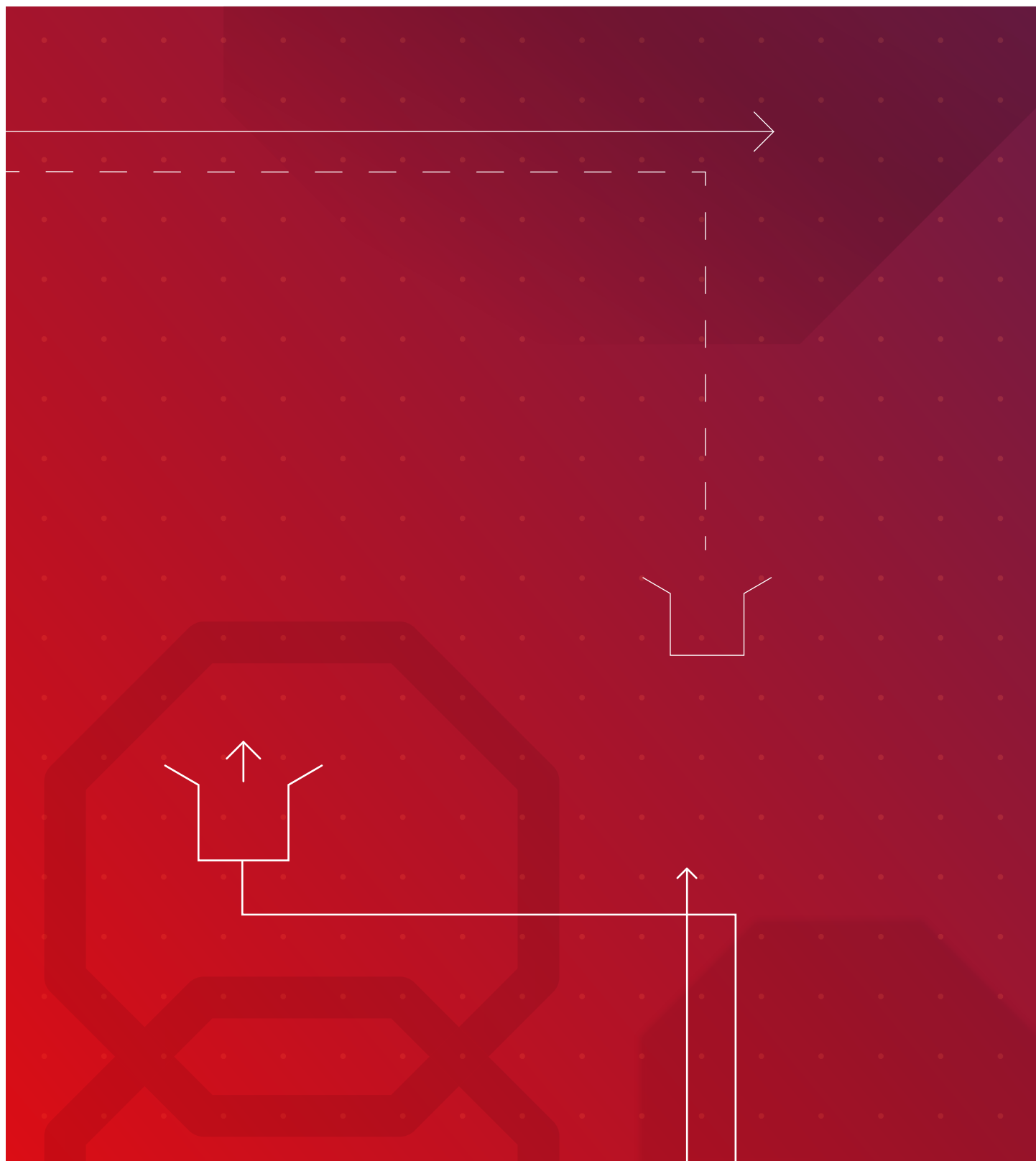


Anti-malware SDK v.4.15

Cross Platform Documentation (SAVAPI)



Contents

1 General considerations.....	5
1.1 Introduction.....	5
1.1.1 General features.....	5
1.2 System Requirements.....	6
1.2.1 Limitations.....	7
1.2.2 Binary integrity.....	8
1.3 Integrating Anti-malware SDK (SAVAPI).....	8
1.4 Third-party libraries.....	8
1.5 Third-party license duties.....	8
1.6 Handling of containers/archives.....	9
2 Anti-malware SDK (SAVAPI) Library.....	9
2.1 General description.....	9
2.2 Library integration.....	9
2.2.1 OnAccess file scanning.....	9
2.3 Library configuration.....	10
2.3.1 File Operation Structure (FOPS).....	10
2.3.2 What is FOPS?.....	10
2.3.3 How to use FOPS: The SAVAPI_scan call.....	10
2.3.4 Library callbacks.....	12
2.4 Logging.....	14
2.4.1 Initialization.....	14
2.4.2 Configuration.....	14
2.4.3 Logging guidelines.....	15
2.4.4 Malware name generation rules.....	16
2.5 Selective file repair.....	16
2.5.1 Steps.....	16
2.6 Extracting malware names.....	17
2.6.1 Troubleshooting: "350 Failed to read VDF file".....	17
2.7 File reputation API support.....	18
2.7.1 File Reputation extension caching.....	19
2.7.2 File Reputation extension blackout mechanism.....	19
2.7.3 File Reputation extension hash scanning.....	19
2.7.4 Computing the File Reputation extension hash using the apchash library.....	19
2.7.5 Scanning the hash with Anti-malware SDK (SAVAPI).....	20
2.7.6 Updating the File Reputation extension hash library.....	20
2.7.7 File Reputation extension scan callback.....	21
2.7.8 File Reputation extension quota.....	23
2.8 Anti-malware SDK (SAVAPI) OnAccess.....	24
2.8.1 Dependencies.....	24
2.8.2 Object exclusions.....	25
2.8.3 Updating OnAccess.....	26
2.9 FPC support.....	28
2.9.1 FPC blackout mechanism.....	29
3 Anti-malware SDK (SAVAPI) Service.....	30
3.1 General description.....	30
3.2 Integration.....	31
3.2.1 On-demand file scanning.....	31
3.2.2 OnAccess file scanning.....	31
3.3 Configuration.....	31

3.3.1 Command line parameters.....	32
3.3.2 Configuration file options.....	37
3.3.3 Protocol.....	55
3.4 Exit codes.....	70
3.5 Logging.....	71
3.5.1 Initialization.....	71
3.5.2 Configuration.....	71
3.6 Non-disruptive service update.....	74
3.7 Fops plug-in.....	74
3.8 Cloud component.....	75
3.9 OnAccess file scanning.....	75
4 Anti-malware SDK (SAVAPI) Client Library.....	75
4.1 General description of Anti-malware SDK (SAVAPI) Client Library.....	75
4.2 Integration of Anti-malware SDK (SAVAPI) Client Library.....	76
4.3 Configuration of Anti-malware SDK (SAVAPI) Client Library.....	76
4.4 Logging in Anti-malware SDK (SAVAPI) Client Library.....	76
4.5 Extracting malware names.....	76
5 Installation.....	76
5.1 Installation on Windows.....	76
5.1.1 Installing the OnAccess driver.....	76
5.1.2 Installing the Anti-malware SDK (SAVAPI) Service.....	77
5.2 Installation on UNIX.....	77
6 Licensing.....	77
7 Updating Anti-malware SDK (SAVAPI).....	78
7.1 Mirroring the Updater's server structure.....	79
7.2 Anti-malware SDK (SAVAPI) update structure and modules.....	79
7.2.1 Updater related files.....	79
7.2.2 Introduction to Anti-malware SDK (SAVAPI) update.....	80
7.2.3 Modules of the Anti-malware SDK (SAVAPI) update.....	80
7.2.4 Anti-malware SDK (SAVAPI) update script details.....	80
7.2.5 How to use Anti-malware SDK (SAVAPI) update scripts.....	81
7.2.6 Limitations.....	82
7.3 Avira Updater's configuration parameters.....	82
7.3.1 General parameters.....	82
7.3.2 Update mode.....	86
7.3.3 Connection settings.....	88
7.3.4 Notification emails.....	90
7.4 Avira Updater's logging.....	91
7.5 Avira Updater's return codes.....	92
7.6 xVDF files merging.....	93
7.6.1 Merging the xVDF files using the xvdfmerge library.....	93
7.6.2 Updating the xVDF merge library.....	93
8 Contact information.....	93
8.1 Support services.....	93
8.2 Contact.....	94
9 Appendix.....	94
9.1 Anti-malware SDK (SAVAPI) binaries.....	94
9.1.1 The files for the Avira Updater.....	94



1 General considerations

1.1 Introduction

The Anti-malware SDK cross-platform (Codename "SAVAPI") from Avira provides an interface to detect malware and repair infected files. The cross-platform engine behind the interface is based on the technology of Avira Operations GmbH.


The SAVAPI interface is written in C and can be compiled by any common C/C++ compiler on many operating systems. Our goal is to give third-party developers an opportunity to easily add antivirus functionality to their products. Typical applications include:


- Email gateways
- Desktop and server solutions
- www and ftp proxies
- Firewalls
- Backup applications

1.1.1 General features

Features	Library	Service	Client Library (combo)
Communication over network sockets	no	yes	yes
Configure Avira Engine location	yes	yes	yes
Configure licence file location	yes	yes	yes
Configure logging options	yes (via API)	yes (via command line or configuration file)	yes
Configure network connection properties	no	yes (via command line or configuration file)	yes (via API)
Configure scan options	yes (via API)	yes (via command line or configuration file)	yes (via API)
Configure VDF location	yes	yes	yes
Repair files in memory	possible with user-implemented FOPS	no	no
Repair files on disk	yes	yes	yes
Scan files in memory*	yes	no	yes
Scan files on disk	yes	yes	yes
Scan emails and mailboxes	yes	yes	yes
Scan regular archives	yes (disabled by default)	yes (disabled by default)	yes (disabled by default)
Scan threads management	yes (must be implemented by the integrator)	limited (only the number of working threads can be configured)	yes (must be implemented by the integrator)
Report scan errors	yes (via API)	yes (via text based protocol)	yes (via API)
Report scan information (warnings, info, progress)	yes	yes	yes
Report scan results	yes (via API)	yes (via text based protocol)	yes (via API)
Extract the malware names from memory to disk	yes	yes	no
Supervisor	no	yes	yes (only for SAVAPI Service)
UNICODE support	yes	yes	yes
User-implemented FOPS	yes	no	no
Cloud file scanning	yes	yes	no
OnAccess file scanning	yes	no	no



 **Note** The structure used for scanning objects from memory uses the type: `unsigned int` for the size of the object in memory. This limits the maximum size for a scanned object to approximately 4GB (2^{32}) on most of the 64-bit systems.

 **Note** Even if it is enabled, APC will not be used when scanning objects from memory.

1.2 System Requirements


Supported OS and hardware platforms


The **32-bit SAVAPI version** runs on the following platforms:

- Linux x86 32-bit (starting with glibc 2.12)
- Linux x86 64-bit (starting with glibc 2.12, if the 32-bit compatibility libs are installed on the system)
- Windows


The **64-bit SAVAPI version** runs on:


- Linux x86 64-bit (starting with glibc 2.12)
- Linux arm64 (starting with glibc 2.17)
- macOS Universal binaries
- Windows x86 64-bit
- Windows arm64

 **Note** Proper functioning of SAVAPI is only guaranteed for operating systems versions that are in official support of the corresponding platform vendor. This does not necessarily imply that SAVAPI does not work on operating systems that are already end of life.

 **Note** On Windows systems, SAVAPI is statically linked with Visual Studio C runtime (CRT), so there is no need of Visual C++ redistributable packages to run SAVAPI.

 **Note** On Windows systems, the SAVAPI OnAccess functionality is provided for both x86 32-bit and 64-bit versions.

 **Note** On the Linux x86 32-bit platform, SAVAPI binaries can also run on systems with 64-bit inode support.

 **Note** On the macOS platform, starting with version 4.12.0, all binaries are codesigned with Apple Code Sign. This is a prerequisite for supporting the Apple app notarization requirements. The integrator has to codesign SAVAPI binary files with his own certificate, before submitting the final package to Apple's app notary service. As a consequence, the engine and VDF files are not delivered as part of the SAVAPI package.

Until SAVAPI version 4.13 (included), the integrator must not try to codesign engine files with his own certificate and must not include engine and VDF files into the final package.

Starting with macOS 10.15, Apple enforced library validation which requires that all libraries loaded into the address space of the caller process are signed with the same team ID. Therefore, starting with the SAVAPI version 4.14, the integrator must codesign SAVAPI and related libraries, as well as the engine files to meet these requirements.

Additionally, a predefined `avira.entitlements` file is provided in SAVAPI package on macOS platform. This entitlement file was used to codesign the SAVAPI daemon executable. It can be used by the integrator to codesign the SAVAPI daemon or other executables with his own certificate. It is important



to note that `com.apple.security.cs.disable-library-validation` key must always be defined in `avira.entitlements`. This is because the engine binary files will always be codesigned with Avira's certificate. For more details on Hardened Runtime Entitlements, please visit: https://developer.apple.com/documentation/security/hardened_runtime_entitlements.

Minimum system requirements (for SAVAPI Service)

- 32-bit or 64-bit CPU, min. 1.6 GHz
- 512 MB RAM (exclusively for SAVAPI)
- 1 GB HDD space (needed for unpacking archives)



Note The system requirements depend on how your application is using SAVAPI. The suggested values are computed for the service default values. If the application that uses the SAVAPI Service modifies the default configuration, the requirements will be different.

On UNIX systems, the *gnu-make* file is required in order to use makefiles and to build the example solution included in the package.

On Windows systems, Visual Studio 2017 is required in order to build the example solution included in the package.

1.2.1 Limitations

- SAVAPI imposes a limit of 300 scan threads that can run in parallel. The best performance is generally reached when 1 or maximum 2 threads per processing unit (CPU core) are used. However, multiple threads per processing can be used in most of the situations, because the scan performance is mostly influenced not by the processing power but by the I/O operations. As there is no formula to calculate the optimal threads per processing unit number for all possible scenarios, SAVAPI integrators should try different setups and decide which one fits the environment better. The default values were chosen based on generic tests and they might not always be the best option.
- Based on stress tests under several versions of Windows, **we do not recommend creating more than 10 connections per second** in parallel to SAVAPI. This limitation has been observed while creating and destroying many connections (each connection created to scan a single file). Microsoft introduces a few new TCP/IP settings, starting with Windows XP SP2, in order to "reduce the threat" of worms spreading fast without control. The number of possible TCP connection attempts per second is limited to a certain amount. This behavior may deny the connection to the server for some scanning clients, causing instability in both client and server (because of the so-called "half connection" - **connected**, but **not accepted** by the service).
- On UNIX systems, the opened files limit (`ulimit -n`) is very important for SAVAPIbased applications. The open files limit is shared between SAVAPI instances. If the limit is too low, it may happen that scan errors or failed connection errors are reported. The higher the archives' recursion level, the more open files will be used during the archive processing. The recommended value for the open files limit is: `archive_max_rec * poolscanners`.
- On Windows systems, the maximum length allowed for a path is 260 characters. ASCII software cannot access longer paths. UNICODE software accepts paths that are 32767 characters long, where the path must have the "\\?" prefix. On Windows 10, the prefix is added internally by the OS libraries, if needed. On older Windows versions the prefix must be added by the developer. This is one limitation of the Windows operating system. Further information can be found on [MSDN](#).



Note SAVAPI real-time scanning is available on Windows x86 platforms. SAVAPI does not provide any anti-adware or anti-spyware features. SAVAPI does not provide any registry protection features. SAVAPI is not a dedicated anti-rootkit product; it offers only basic anti-rootkit detection via VDF signatures.



1.2.2 Binary integrity

SAVAPI binaries undergo an integrity check during initialization. Any modification of the binaries will result in failures during load and initialization will exit with an error. For Windows, modifications to the files caused by Authenticode signing (adding, removing, replacing, appending signatures) will not affect the integrity check.

To further secure the files, it is recommended to place them in a trusted location with restricted file permissions, as the system loader works with file paths and the files can potentially be replaced between validation and usage.



Note On macOS systems, SAVAPI binaries undergo Code Signing to guarantee compatibility with the operating system. The integrator is free to resign the files with its own certificate.

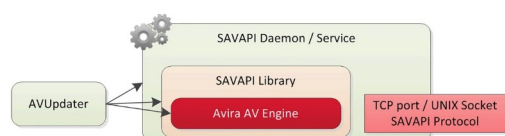
1.3 Integrating Anti-malware SDK (SAVAPI)

SAVAPI provides more interfaces and modes of use to the customers. These modes allow the customers to integrate SAVAPI according to their needs. The figure below illustrates each of the three modes, highlighting possible ways to interact with SAVAPI.

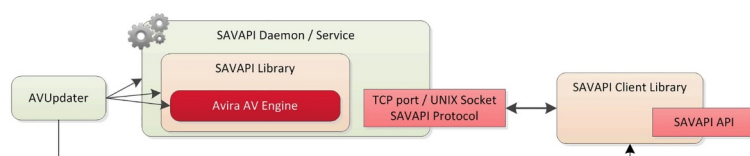
SAVAPI Library Mode:



SAVAPI Daemon / Service Mode:



SAVAPI Client Library + Daemon / Service Mode:



The following chapters present these modes, some recommended usage scenarios and advantages for every SAVAPI mode:

- [Chapter 2. Anti-malware SDK \(SAVAPI\) Library](#)
- [Chapter 3. Anti-malware SDK \(SAVAPI\) Service](#)
- [Chapter 4. Anti-malware SDK \(SAVAPI\) Client Library](#)

1.4 Third-party libraries

The third-party libraries and their license files are listed in the `legal` folder of the package.

1.5 Third-party license duties

Most third-party software ("TPS") licenses are granted under certain terms and conditions that have to be accepted by anyone using their software and/or passing the TPS on to others. Sometimes it is sufficient to provide the user or the receiving party of the TPS the license text, the access to the source code, copyright notices and legal disclaimer, and mention the usage of the third-party libraries and the connected obligations in the documentation.

However, since there are very diverse usage and integration scenarios of the SDK possible and the third-party licenses are licensed under very different terms and conditions, we strongly recommend to



check each third party and the corresponding license (text) with a competent and skilled counsel to ensure you comply with the requirements and to not infringe any intellectual property rights.

The materials of the third-party licenses can be found in the *.Vegal* folder of the SDK package.

1.6 Handling of containers/archives

To understand how containers are scanned, one must know that they are divided into two main categories:

- `Native containers` are "normal" archives that most of end-users are aware of. Examples: zip, tar, rar.
- `Non-native containers` are special archives that look like one single file to the common users. Examples: pdf, installers, packers.

While the scanning of `Native containers` can be controlled by the user (e.g. via `ARCHIVE_SCAN` options), `Non-native containers` are always opened and scanned to ensure complete detection. If speed is more important than the detection accuracy (e.g. OnAccess context), one can overwrite this behavior through the `SAVAPI_CALLBACK_ARCHIVE_OPEN` callback.

2 Anti-malware SDK (SAVAPI) Library

2.1 General description

SAVAPI Library is a cross-platform library that allows the customer to initialize the scan engine, configure its options and scan a file. The functionality is exposed through an API written in C language.

The application based on SAVAPI Library must either load the library on runtime or link with the library at compile time; then call the library API to perform initialization, configuration, scanning, etc.

The application is also responsible for maintaining the scan jobs, creating threads to scan, allocating data, terminating the instances and uninitializing the library. If the customer wants to have complete control over these resources, then he should opt for SAVAPI Library mode.

2.2 Library integration

The client-designed applications should link to the SAVAPI Library (either implicit or explicit) and use the SAVAPI Library API to access the SAVAPI technology. The SAVAPI Library API allows the user to configure the scan engine and other SAVAPI options, to process files and retrieve information about the processing status. For more details, please refer to the API documentation.

A very simple example of how one can use the SAVAPI Library is offered in the SAVAPI SDK kit. (You can refer to the SDK documentation under doc/README.)

2.2.1 OnAccess file scanning



Note This functionality is available only on Windows systems.



Note Terms introduced: OnAccess file scanning, SAVAPI OnAccess

Starting with version 4.4, SAVAPI offers OnAccess file scanning. This feature brings a new SAVAPI module, SAVAPI OnAccess.

In order to use SAVAPI's OnAccess file scanning capability, additional virtual driver files and runtime libraries need to be installed, as well as a valid SAVAPI OnAccess product license. The needed files (virtual drivers, runtime libraries), as well as an install/uninstall command line script are available in the SAVAPI package. Without those, SAVAPI will only provide on-demand file scanning.



2.3 Library configuration

The SAVAPI Library can be configured through the API. In order to set an option, the customer will just have to call the `SAVAPI_set ()` function with the appropriate parameters.

2.3.1 File Operation Structure (FOPS)

Avira Engine's FOPS is a mechanism which allows any SAVAPI client to implement a special method of scanning objects, other than files, on disk. Obviously, the name of the feature shows that its origin is file-related (File OPS), but it is possible to implement almost any access method, as long as the interface is correctly implemented.

If the user wants to process a different kind of data, other than files (an I/O stream for example), he can implement his own File Operation Structure (FOPS) and provide it to SAVAPI Library. FOPS is a collection of functions that give the SAVAPI Library access to the desired user data. The user would not be able to scan this data only with SAVAPI Service, nor with the SAVAPI Service + SAVAPI Client Library combo.

Real implementations of the FOPS include, but are not limited to:

- Implementing scanning on an encrypted file system – The application which uses the SAVAPI Library implements the logic for reading and writing in the encrypted file system, allowing the engine to read the files as if they were normally stored (without encryption).
- Implementing scanning on memory streams – The application is closely integrated with a server (email, file, http) which processes its data in memory, receiving the files as streams.

2.3.2 What is FOPS?

The FOPS structure is a collection of I/O routines which are usually used to access files on a normal disk, having a standard File Allocation Table. These routines are organized in an array of pointers to functions.



Note The order in this array must be kept exactly as in the examples in the SDK.

Each function receives certain parameters, depending on the function type.

```
AVE_FOPS engine_fops =  
{ my_open,my_close,my_read,my_write,my_tell,my_seek,my_getfattr,my_s  
etfattr,my_getfsize,my_unlink,my_rename,my_access,  
my_malloc,my_free,my_gets,my_puts,my_getc,my_putc, my_ungetc, my_flush,  
my_get_last_error };
```



Note It is mandatory that all functions are defined. Do not simply add a NULL instead of a pointer. Doing so will make the engine abort its initialization.

The engine uses such functions to process any object, right after the `SAVAPI_scan` function is called. Internally, there is such a structure already implemented, having an implementation similar to the functions in the two examples provided with the SDK (but not the same). The two examples are fully functional, but simplified in order to allow an easy understanding of the process.

Having such a structure in place and allowing the user to change it at any time, SAVAPI has to make sure to differentiate between “normal” engine operations (like opening the VDF files and creating or deleting temporary files) and the object scanning operations. The engine needs the normal FOPS for the first category because these files must always reside on disk (a RAM Disk is also considered a disk). The second category, representing the objects which must be scanned, have to be accessed using the user's FOPS.

2.3.3 How to use FOPS: The SAVAPI_scan call

Once the `SAVAPI_set_fops` is called with a pointer to the structure defined above, the engine will dynamically use one of the two FOPS structures, depending on the context.

This function requests the engine to scan an object.



If you scan a file on disk, the second parameter is the filename, so no further information is required.

If you scan an object or a stream, there are some things which have to be specially configured.

The entire idea is to provide to the `FOPS::open` function enough information to access the object referenced by the second parameter of the `SAVAPI_scan` function (`file_name` parameter). Since this is a character buffer, you have to write somehow the address and the size of the memory area where the object to be scanned resides.

For example, you could use a string like this:

```
SAVAPI_scan(&inst, "0xAddress,size,Filename.ext")
```

Where "0xAddress" represents the address of the pointer to the memory area where the object resides and the "size" is the size of the memory area. If you provide the "Filename.ext" parameter, the engine is going to be able to make some assumptions based on the file type and from case to case perform deeper analysis on the object.

This means that the open function which has the prototype

```
int fops_open (FOPS_HANDLE *fh, void *filename, int mode, void
*file_context, void *fops_context)
```

receives the following information:

```
FOPS_HANDLE *fh : undefined
```

You have to allocate this according to your needs. For example, having the value given above, you must save the address and the size. Later on, you have to convert this address to a valid pointer.

```
Struct MyFH
{
void* addr;
long size;
}
```

Do not forget to allocate your FOPS Handle structure:

```
MyFH* myfh = (MyFH)malloc(sizeof(MyFH));
... do processing here ...
*fh = (void*)myfh;
```

1. First extract the "address" and the "size" from the "filename".
2. Convert the hexadecimal string into an int

```
int32 address = convertHexStringToInt32( "0xaddress");
// you must write your own conversion function to a safe type
// (32 and 64 bits, depending on architecture)
```

Assign to the file handle pointer the value

```
fh->addr = (char*)address;
fh->size = size;
```



Note This is just a simplified example which is not safe and probably not cross-platform.

```
void *filename: "0xAddress,size,filename.ext"
```

This is the file name provided in the `SAVAPI_scan` function. Extract the address and the size and use them to allocate the "fh"

```
int mode : 0 (OPEN_RO)
void *file_context: undefined or NULL - ignore this value
void *fops_context: the same value you defined when you called
SAVAPI_set_fops
```



Note The handle that was allocated in the `open` function must be released in the `close` function. It is possible that the `open` function will be called multiple times for the same file. The `close` function



will also be called multiple times, corresponding to the number of `open` calls, for the same handle. In the function `getfsize` where you have to return the size, just return `fops->size`.

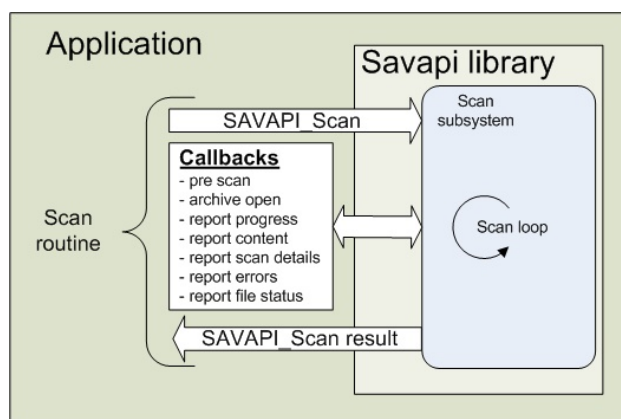
2.3.4 Library callbacks

The scanning of malware files is a very complex and time consuming task, especially when malware is found in multiple archives layers scenarios. When scanning archives, a recursive scan is used: archives in archives are also unpacked and scanned for viruses and unwanted programs. The files are scanned, decompressed and scanned again. During the scan the SAVAPI Library communicates with the partner code by using a range of callbacks. Callbacks inform or trigger application specific actions. They represent user defined, registered C functions that must obey the SAVAPI_CALLBACK structure as defined in the header file *savapi.h*.

To use these callbacks, they have to be registered before performing a scan operation by using the function `SAVAPI_register_callback`. The callbacks are instance specific and chaining is not supported. If such a behavior is nonetheless required, an appropriate mechanism should be implemented and managed by the application. However, when they are of the same user defined code, a callback can be registered multiple times, for different callback IDs.

At run-time, the function determines exactly in what context it was called by checking the callback type defined in the SAVAPI_CALLBACK_DATA structure from the same *savapi.h*. If no callback is registered for a certain ID it will be silently ignored.

SAVAPI Library callbacks can be triggered multiple times in an unpredictable order depending on the scanned content. The overall scan process including the callback triggers is depicted by the following graphic:



The callback types are the following:

- **SAVAPI_CALLBACK_PRE_SCAN**: triggered right before scanning. It is useful to create file filters based on the file information provided by data parameter
- **SAVAPI_CALLBACK_ARCHIVE_OPEN**: asks the application if it should open and scan an archive
- **SAVAPI_CALLBACK_PROGRESS_REPORT**: informs the application periodically about the scan progress
- **SAVAPI_CALLBACK_CONTENT_REPORT**: reports various types of information, e.g. detection of I-frames
- **SAVAPI_CALLBACK_SCAN_DETAILS_REPORT**: informs the application about some scan related details, like detected malware or, if infected content is repairable
- **SAVAPI_CALLBACK_REPORT_ERROR**: informs about encountered errors and warnings
- **SAVAPI_CALLBACK_REPORT_FILE_STATUS**: reports the evaluation of the scanned file, e.g. if it is clean or infected, if the malware name is applicable etc.



From the above described callbacks only SAVAPI_CALLBACK_PRE_SCAN and SAVAPI_CALLBACK_ARCHIVE_OPEN can alter the SAVAPI Library's internal execution path through internal return codes as follows:

- SAVAPI_CALLBACK_PRE_SCAN If the returned code is success (SAVAPI_S_OK), the file will be scanned, otherwise it will be skipped.
- SAVAPI_CALLBACK_ARCHIVE_OPEN If the returned code is success (SAVAPI_S_OK), the archive will be opened, otherwise it will be skipped from scanning.

The other callbacks are purely informative and any return code from them is silently discarded.

Examples

Although the order of callbacks is not guaranteed and depends on the scanned content, here are some callback examples for the scanning of various file types, using the [EICAR test file](#). The *lib_basic_complete_example* was used to generate the following outputs:

- **eicar.com**

```
PRE_SCAN callback called for file 'eicar.com'
  file type: regular file
  archive recursion level: 0
FILE_STATUS callback called for file 'eicar.com'
  scan answer: infected
  File 'eicar.com' is infected!
  malware name: Eicar-Test-Signature
  malware info: Contains code of the Eicar-Test-Signature virus
FILE_STATUS callback called for file 'eicar.com'
  scan answer: finished`
```

1. PRE_SCAN is triggered before scanning begins. This function must return 0 in order to continue, otherwise the scan will be aborted.
2. FILE_STATUS is triggered with answer "infected", reporting that malware is found.
3. FILE_STATUS is triggered with answer "finished", reporting that the scan is finished.

- **eicar.zip ("eicar.com" archived)**

```
PRE_SCAN callback called for file 'eicar.zip'
  file type: regular file
  archive recursion level: 0
ARCHIVE_OPEN callback called for file 'eicar.zip'
  file type: archive file
  archive recursion level: 0
PRE_SCAN callback called for file 'eicar.com'
  file type: file inside archive
  archive recursion level: 1
FILE_STATUS callback called for file 'eicar.com'
  scan answer: infected
  File 'eicar.com' is infected!
  malware name: Eicar-Test-Signature
  malware info: Contains code of the Eicar-Test-Signature virus
FILE_STATUS callback called for file 'eicar.com'
  scan answer: finished
FILE_STATUS callback called for file 'eicar.zip'
  scan answer: finished
```

1. PRE_SCAN is triggered before scanning of "eicar.zip" begins. This function must return 0 in order to continue, otherwise the scan will be aborted.
2. ARCHIVE_OPEN is triggered before "eicar.zip" file is decompressed. This function must return 0 in order to continue, otherwise the scan will be aborted.
3. PRE_SCAN is triggered before scanning "eicar.com" (the file inside "eicar.zip"). This function must return 0 in order to continue, otherwise the scan will be aborted.
4. FILE_STATUS is triggered with answer "infected", reporting that a malware is found in "eicar.com".
5. FILE_STATUS is triggered with answer "finished", reporting that the scan for "eicar.com" is finished.
6. FILE_STATUS is triggered with answer "finished", reporting that the scan for "eicar.zip" is finished.



- **eicar_encrypted.zip ("eicar.com" archived with password)**

```
PRE_SCAN callback called for file 'eicar_encrypted.zip'
  file type: regular file
  archive recursion level: 0
ARCHIVE_OPEN callback called for file 'eicar_encrypted.zip'
  file type: archive file
  archive recursion level: 0
PRE_SCAN callback called for file 'eicar.com'
  file type: file inside archive
  archive recursion level: 1
ERROR callback called for file 'eicar_encrypted.zip'
  error occurred during scan, code: 25
FILE_STATUS callback called for file 'eicar.com'
  scan answer: finished
ERROR callback called for file 'eicar_encrypted.zip'
  error occurred during scan, code: 28
SAVAPI_scan failed with error code: 25
```

1. PRE_SCAN is triggered before scanning of "eicar_encrypted.zip" begins. This function must return 0 in order to continue, otherwise the scan will be aborted.
2. ARCHIVE_OPEN is triggered before "eicar.zip" file is decompressed. This function must return 0 in order to continue, otherwise the scan will be aborted.
3. PRE_SCAN is triggered before scanning "eicar.com" (the file inside "eicar.zip"). This function must return 0 in order to continue, otherwise the scan will be aborted.
4. ERROR callback is triggered because the archive is encrypted (error 25, SAVAPI_E_ENCRYPTED).
5. FILE_STATUS is triggered with answer "finished", reporting that the scan for "eicar.com" is finished.
6. ERROR callback is triggered reporting that the scan was not finished (error 28, SAVAPI_E_INCOMPLETE).

For more details, see API documentation and examples in the SAVAPI package.

2.4 Logging

2.4.1 Initialization

In order to receive the messages logged by the SAVAPI Library, you need to have a special callback function. The function must have the following format:

```
void(*SAVAPI_LOG_CALLBACK) (SAVAPI_LOG_LEVEL log_level, const SAVAPI_TCHAR
*message, void *user_data);
```

The function's parameters are described in the SAVAPI Library API.

Besides the function's header, there are no restrictions about the implementation.

The function must be registered in the SAVAPI Library with the following function:

```
int SAVAPI_EXP SAVAPI_set_log_callback(SAVAPI_LOG_CALLBACK log_fct,
SAVAPI_LOG_LEVEL min_level, void *user_data);
```



Note This function can be called without initializing the library. The function's parameters are described in the SAVAPI Library interface section.

2.4.2 Configuration

The logging mechanism can be configured using the SAVAPI_set_log_callback function. The min_level parameter will set the minimum log level of the messages received through the registered callback. All logs from higher levels will be logged. The DEBUG level has the highest verbosity and the ERROR level has the lowest verbosity.

Log-levels

In decreasing order:



`SAVAPI_LOG_DEBUG` – this is the most verbose log level; messages on this level will contain low level information, such as:

- details about the initialization and un-initialization of SAVAPI Library
- details about the creation and release of instances
- information about interface function behavior

`SAVAPI_LOG_INFO` – messages on this level will contain information about the general workflow of the library such as:

- key stages of the initialization and un-initialization
- key stages in the creation and release of library's instances

`SAVAPI_LOG_WARNING` – messages on this level will contain information about unexpected but recoverable errors that will not stop the execution of the library or of its functions. For example:

- invalid product ID given in the initialization structure
- VDF files are older than two weeks

`SAVAPI_LOG_ALERT` – messages on this level will contain information about alerts that have been raised during runtime. For example:

- malware was found while scanning a file

`SAVAPI_LOG_ERROR` – this is the least verbose log level; messages on this level will contain information about unrecoverable errors that occur during runtime, in one of the library's functions. All error logs will contain a short description of the action that failed, followed by the error code, and the error's description.

If you want to stop the logging of the SAVAPI Library, the `SAVAPI_set_log_callback` function must be called with `NULL` as the `log_fct` parameter. Also, if the log function or the minimum log-level needs to be modified, the same function can be called again with the new configuration.

In addition, the `user_data` parameter grants the user freedom to send any necessary information to the callback function. Usually this parameter contains a user-defined structure with multiple fields. For example:

- The log file name or the log file descriptor which can be used by the log callback to write the logs inside the respective file
- Information about the running instance number: in a configuration that uses multiple scanning instance, the log messages can be divided between multiple files or have different formats for a better tracking of their thread source
- Counters for the number of errors or alerts found in a running session
- Counters for the number of infected files found

2.4.3 Logging guidelines

The logging mechanism allows you to receive feedback about the library's activity and performance. It should be started before the initialization of the SAVAPI Library, in order to log any possible errors.

The setting of the minimum log level determines the amount of log messages that will be received by the log callback. If the log level is set to the minimum level (`SAVAPI_LOG_DEBUG`), the application's performance may be affected, because of the increased number of disk write-accesses that are performed during the writing of the log messages into the log files. This level should only be used to track and debug issues. For example: After receiving an unexpected error, the behavior can be reproduced on this level and you may be able to track the source of the problem or give a more detailed feedback to the Support team.

In order not to affect the application's performance, the following log level configurations are recommended:



`SAVAPI_LOG_INFO` – Should be the default setting. It offers all available information about the unexpected events (alerts, errors, warning) and keeps track of the key stages in the initialization workflow.

`SAVAPI_LOG_ERROR` – If the application's workflow is not important, or if the application is stable enough not to worry about unexpected events such as warnings and alerts, this level will guarantee minimum feedback and maximum performance.

2.4.4 Malware name generation rules

In case of malware detection in the scanned items, SAVAPI returns some information about the identified malware. This information is based on the return code of the scan engine, following the rules Avira virus names are structured.

The following example shows the basic structure of such a name:

`PREFIX/SUFFIX.MALWARENAME.VARIANT`

The Avira virus names are built of 4 possible elements, that are:

*prefix

*suffix

*malware name

*variant

The combination of prefix, suffix, malware name and variation is also called **signature identifier**.

Prefixes are used to categorize malware. In case of malware detection, the prefix is used to match a specific and accurately formulated malware information notice. This notice offers the user details about the category / class the virus / code belongs to.

This malware notice will show the prefix in its expanded form together with the malware name (if existing).

The "." (dot) separates the malware name from possible variations of this malware.

To get a valid virus detection name, the prefix and the suffix have to be delimited with "/" (slash).

The suffix is only added to malware categories that have a quite static behavior. The following are examples for names:

`TR/Spy.Zbot.XXX`

`TR/Dldr.Agent.XXX`

Prefixes are always written in capital letters or should be converted to capital letters by the application. The signature identifier must not be longer than 21 characters and must not contain any spaces. The variant is an incremented combination of letters and/or numbers.

All newly detected malware is built on the above mentioned structure. As a basic rule, a valid Avira virus name has to contain at least the "/" and one "." as delimiters.

Since some definition list entries belong to older virus names, most likely old DOS viruses, not all names have the "/" as delimiter or not a valid prefix either.

2.5 Selective file repair

You can check if a file is repairable and start a repair attempt:



Note Please note that files contained in archives or other containers cannot be repaired, unless the application itself unpacks the contents and provides the files to SAVAPI one by one. In this case it is the application's responsibility to pack the contents again.

2.5.1 Steps

1. Set `SAVAPI_OPTION_REPAIR` to 1. With this option enabled, and if SAVAPI detects during the scan that a file contains a removable malware, it will automatically attempt to repair it.



2. Scan the file. In case of an infection, the callback for `SAVAPI_CALLBACK_REPORT_FILE_STATUS` is triggered.
3. When `SAVAPI_FILE_STATUS_DATA` callback is triggered, and if the field `SAVAPI_FILE_STATUS_DATA.malware_info.removable` has the value 1, then the file is repairable. `SAVAPI_CALLBACK_REPORT_FILE_STATUS` is triggered again to signal that the scan has finished (with this call, the repairable field will be reset to 0). If the repair fails, the callback for `SAVAPI_CALLBACK_REPORT_ERROR` is triggered with the error `SAVAPI_E_REPAIR_FAILED`.

2.6 Extracting malware names

The VDF files, which are loaded in memory to improve scan performance, contain various information needed to detect malware. However, the malware names are only needed in the rare event of detections/ alerts. In order to reduce active memory load, SAVAPI can extract the malware names from memory and dump them to disk.

To extract malware names, call the following function, only after the engine is successfully loaded (i.e. with a call to `SAVAPI_initialize`, `SAVAPI_reload_engine` or `SAVAPI_reload_engine_ex`):

```
int SAVAPI_EXP SAVAPI_extract_malware_names(const SAVAPI_TCHAR *dir_path);
```

In the provided directory path, the function creates a file in which the malware names are extracted. By default, if no directory path is provided, the function uses the system temporary directory.

Ensure that SAVAPI has the appropriate permissions to create the file and that there is enough free disk space (approximately 100MB).

The filename uses the following naming scheme:

AV-malware-names-<process-PID>-<6 random chars>

To retrieve the file path, use the `SAVAPI_get()` function, with the option `SAVAPI_OPTION_MALWARE_NAMES_FILE`.

The function enables malware names extraction only for the current engine. For every new loaded engine (e.g. after an Update) the function has to be called again. Every function call creates a different file, which is deleted when the engine is unloaded (i.e. with a successful call to `SAVAPI_uninitialize`, `SAVAPI_reload_engine` or `SAVAPI_reload_engine_ex`).

SAVAPI can only extract the malware names after SAVAPI has been fully initialized and completely loaded into memory (including the malware names). This means, if SAVAPI is using the malware names extraction feature, it needs the same amount of memory for startup and initialization, no matter whether the extraction function is used or not.

After extracting the malware names, the SAVAPI memory load is reduced with approximately 60 Mb. The reduced memory load may be visible or not, depending on the platform policies regarding freed memory caching. Usually on UNIX, the freed memory is kept in the process address space, until it is explicitly requested by kernel to be used for other processes. However, this freed memory is available for SAVAPI to reuse for other operations, like scanning.



Note Using the malware names extraction function increases the I/O load, which might have a slight impact on the performance, in case of a higher amount of detections/ alerts. Once the malware names were extracted, they cannot be relocated back in memory, without re-initializing/ reloading SAVAPI. Malware names extraction is an optional SAVAPI feature. By default SAVAPI will run with malware names loaded in memory.

For more information about this feature, see the documentation of SAVAPI Library API.

2.6.1 Troubleshooting: "350 Failed to read VDF file"

Since the malware names file is needed by SAVAPI, it is recommended not to alter it. If the file is deleted, moved or its content is modified, the engine may be unable to find the names of the viruses.

If the file has been altered, when the `SAVAPI_scan()` function processes an infected file for the first time, it does not report the virus(es); it triggers two error-callbacks (with the `SAVAPI_E_VDF_READ` and `SAVAPI_E_INCOMPLETE` codes) and it returns the `SAVAPI_E_VDF_READ` error code.



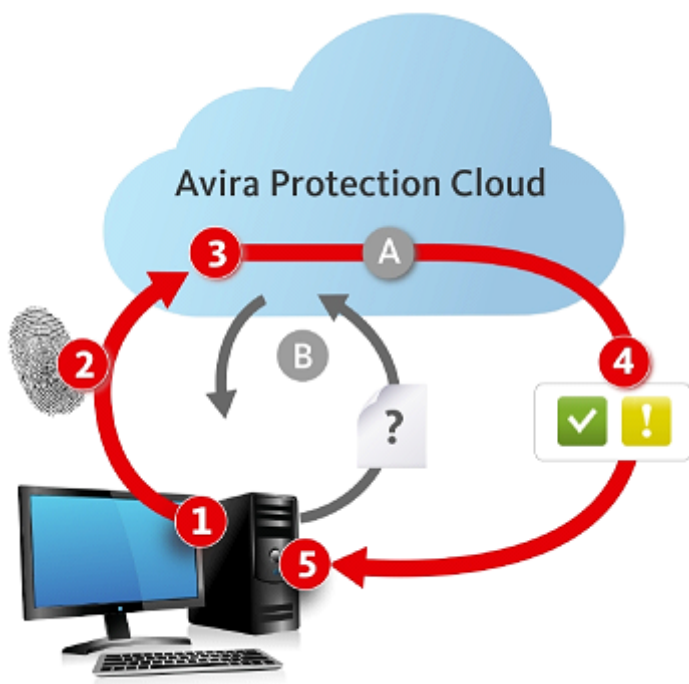
Subsequent `SAVAPI_scan()` calls would trigger an error-callback (with the `SAVAPI_E_VDF_READ` code) and would return the `SAVAPI_E_VDF_READ` error code, no matter if the file was infected or not.

Furthermore, apart from the error-callbacks, the `SAVAPI_scan()` also triggers log callbacks, containing explicit error messages.

In order to fix the problem, please release the instance and reload the engine, or perform a new library initialization.

2.7 File reputation API support

File reputation API has also the codename "APC", where APC stands for Avira Protection Cloud. Starting with version 4.0, the SAVAPI suite offers support for scanning files with APC. Users can improve their malware detection rate by performing additional scan processes on the Avira servers. This enables users to doublecheck suspicious files, if their local Antivirus engine considers the files to be safe. An overview of how the system operates, is shown in the graphic below: Avira Protection Cloud SDK.



- (1) The application scans a suspicious file ('clean' executable) on a computer.
- (2) The file's fingerprint is extracted and sent to Avira Protection Cloud for review.
- (3) The fingerprint is compared with those that have previously been analyzed by the Protection Cloud.

This progress can have two outcomes:

- (A) The fingerprint belongs to a file that has been previously analyzed by the Avira Protection Cloud. It is immediately labeled as either clean, or malware.
- (B) The fingerprint is new to the Avira Protection Cloud. The complete file is uploaded to the Protection Cloud, thoroughly examined and judged as clean or malware.
- (4) Avira Protection Cloud sends the status of the fingerprint, clean or malware, to the application (SAVAPI) on the user's machine
- (5) If the file is classified as malware, SAVAPI will handle the threat.



Note For the fingerprint check and upload, the Protection Cloud supports PE and non-PE files and a comprehensive dynamic list of other file types, which will be communicated by other means.



Note For systems that use a proxy to connect to the Internet, the proxy address is read from one of the following sources, presented in the order of their priority: the configuration file of SAVAPI, Internet Explorer settings (for Windows systems only), Winhttp settings (for Windows systems only), environment variables: `https_proxy`, `HTTPS_PROXY`, `http_proxy`, `HTTP_PROXY`, `all_proxy`, `ALL_PROXY`.

For more information about this feature, see the documentation of SAVAPI Library API.

2.7.1 File Reputation extension caching

In order to increase the scanning speed and to save bandwidth, the Avira Protection Cloud includes cache support, meaning that it transparently stores data in memory, so that future requests for the same data can be served faster. The size of the cache greatly affects the time needed by the APC to finish processing the request. The more size available, the more data can be stored and used later, thus for high-intensive applications, a bigger value is recommended.

The default memory size for the cache is 5 MB.

When exiting SAVAPI, the cache will be dumped into the `savapi_apc_cache_XXXXXXXXX.dat` file. This file will be created in the temporary folder, unless otherwise stated. Also, when starting SAVAPI, this cache file is reloaded. That way, no cache information will be lost during SAVAPI restarts.

2.7.2 File Reputation extension blackout mechanism

The APC component of SAVAPI, when activated, requires a permanent Internet connection. When the Internet connection is interrupted or becomes very slow, this could lead to performance issues while SAVAPI performs multiple scans. In order to avoid those issues, an APC "blackout" mechanism is implemented that will temporarily disable the APC. It is configured using two options, *Blackout retries number* and *Blackout timeout*, and it works as follows:

- If *Blackout retries number* consecutive scans using APC fail because of connection problems or timeouts, SAVAPI will declare APC as unavailable and will stop using it. The rest of the scans will be performed using the local engine only.
- After *Blackout timeout* seconds, SAVAPI allows one worker to use APC for scanning. The rest of the workers will still use the local engine only.

a) If this APC scan succeeds, SAVAPI will declare APC as available and will start using it.

b) If the scan fails because of the previously mentioned problems, APC will remain unavailable and another attempt to use APC will be made after another *Blackout timeout* seconds.

Because SAVAPI can detect cloud availability problems, for example limited or no Internet connection, SAVAPI switches to scanning files locally until the cloud is available again. This enables high scanning rate to be maintained.

2.7.3 File Reputation extension hash scanning

SAVAPI can not only scan files with APC, but it can also directly scan the files' fingerprints (also referred to as hashes). Users can compute the hashes themselves by using the `apchash` library and then scan the resulted hashes with SAVAPI. Multiple hashes can be verified in a single scan, thus reducing the total scan time.



Note All engine-related scanning options have no effect in this scanning mode.



Note Some APC-related options have no effect in this scanning mode:
`APCCheckRiskRatingLevel`, `APCUploadRiskRatingLevel`.

2.7.4 Computing the File Reputation extension hash using the `apchash` library

Two files were added in the SAVAPI package for computing the APC hash of a specified file. These files are located in:

- bin folder: `libapchash.so` for Unix or `apchash.dll` for Windows;



- include folder: `apc_hash.h`.

The library computes the APC hash of a given file by using the following two functions:

```
// compute the hash
int APC_hash_file_compute(TCHAR* file_path, char **apc_hash);
// free the memory allocated internally for the hash
void APC_hash_free(void **ptr);
```

Simple example for using these functions:

```
char *hash = NULL;
// compute the hash
int ret = APC_hash_file_compute("/home/test/eicar.com", &hash);
// use this hash in SAVAPI lib, clientlib or daemon
.....
// free the memory allocated by the hash
APC_hash_free((void**) &hash);
```

2.7.5 Scanning the hash with Anti-malware SDK (SAVAPI)

Some simple examples of scanning an APC hash:

- SAVAPI Library and SAVAPI Client Library

```
SAVAPI_TCHAR *scan_buffer = NULL;

// convert the hash string to a SAVAPI_TCHAR
ret = CharToSTCHAR(&scan_buffer,
                  "apchash://
275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabbf651fd0f");

// scan the converted hash string
ret = SAVAPI_scan(instance, scan_buffer);

// free the memory allocated by the conversion
SAVAPI_free((void **) &scan_buffer);
```

- SAVAPI daemon

```
SCAN apchash://
275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabbf651fd0f,5b4
67461218087f00c1aef83c10819d17a27bc69b7307e2dc63e4d56c49857e7

310
275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabbf651fd0f <<<
Eicar-Test-Signature ; APC/DOS ; Detected by Avira APC

310
5b467461218087f00c1aef83c10819d17a27bc69b7307e2dc63e4d56c49857e7 <<<
TR/APC.SE.Gen ; APC/TR ; Detected by Avira APC
319 OK
```



Note If the scanned hash is not known by APC, the error callback will be triggered with `SAVAPI_E_APC_UNKNOWN_CATEGORY`.

2.7.6 Updating the File Reputation extension hash library

In order to maintain its functionality and to fix possible issues, APC hash library needs updates. This library can be updated with an integrated updater module (the Avira Updater) which is available on all the APC hash supported platforms.

Through command line parameters or configuration file *avupdate-apchash-product.conf* for the Updater, the following operations can be carried out:

- Check if new updates are available;
- Update APC hash library from Avira's update servers or from user-defined servers, with the proper update structure;



- Mirror the already configured update servers, see [7.1 Mirroring the Updater's server structure](#).

The binary name of the Updater is `avupdate.bin` (UNIX) and `avupdate.exe` (Windows).

The APC hash library update command:

```
<Updater_binary_name> -C avupdate-apchash-product.conf
```

At the end of each update cycle, the status of the update will be displayed in the console. The messages (error, warning, etc.) displayed by the Updater when executed are set in the binary file `avupdate_msg.avr`. This file has to exist in the same folder as the `avupdate` binary file, being mandatory for starting the Updater binary.

The APC hash library module update name is `APC_HASH`. This can be used to specify modules by adding the `"update-modules-list"` option in command line. For more information about configuration parameters, see [7.3 Avira Updater's configuration parameters](#).

2.7.7 File Reputation extension scan callback

The APC scan callback (of type `SAVAPI_CALLBACK_APC_SCAN`) is triggered at key stages of the APC scan of a file and has the purpose of:

- Giving you additional information regarding the currently scanned file
- Allowing you to have more control over the APC scan process of the file, by giving you the option at each stage of the scan to either continue, stop the scan or report the detection of a file

Among the information that can be received on the callback, the most relevant are the following:

- The APC scan stage in which the current callback was triggered
- The file's hash (or fingerprint)
- The risk rating level of the file (the probability that it contains malware)
- A FOPS handle to the file through which it can be accessed
- A pointer to a function through which a detection can be reported by the user

Controlling the File Reputation extension scan workflow

If registered, the APC scan callback can be triggered at one or more of the following stages of an APC scan:

- Before the file is filtered for scanning with APC
- After the file passed the local APC filter, but before the file's hash is checked with APC
- After the file's hash was checked with APC, but before the upload (in case the file's detection is not known)
- After the file was fully scanned with APC



Note Filtering files for scanning with APC means checking that certain preconditions are met before files can be scanned with APC (file type, file size, etc.).

You can read the stage the APC scan is in by accessing the `stage` member of the data structure provided by the callback.

At each of the above stages you can control the workflow of the scan by returning one of the following codes:

- `SAVAPI_APC_SCAN_CONTINUE` - Continue with the next stages of the scan
- `SAVAPI_APC_SCAN_STOP` - Stop the scan for the current file
- `SAVAPI_APC_SCAN_REPORT` - Report a detection and stop the scan for the current file



For example, if you wish to allow APC hash checks but not uploads, you might implement the callback in the following way:

```
int APC_scan_callback(SAVAPI_CALLBACK_DATA *data)
{
    SAVAPI_APC_SCAN_DATA *apc_scan_data = data->callback_data.apc_scan_data;

    switch (apc_scan_data->stage)
    {
        case SAVAPI_APC_STAGE_PRE_FILTER:
        case SAVAPI_APC_STAGE_PRE_HASH_CHECK:
        case SAVAPI_APC_STAGE_POST_SCAN:
            return SAVAPI_APC_SCAN_CONTINUE;

        case SAVAPI_APC_STAGE_PRE_UPLOAD:
            return SAVAPI_APC_SCAN_STOP;
    }
}
```

Reporting a file detection

With the APC scan callback, you have the possibility to access the currently scanned object and thus can scan it with his own technology. The result of the scan can be further integrated into SAVAPI by reporting the found detection. To do so, you must:

- Call the report function with the detection information
- Return `SAVAPI_APC_SCAN_REPORT` in the APC scan callback

When reporting a detection, you have the possibility to cache it, either for clean or malware files. To do so, you must fill two more fields from `SAVAPI_APC_REPORT_DATA`:

- `store_cache` – Must be set to 1 in order to enable caching
- `ttd` – How many seconds to keep the detection in cache



Note For a `ttd` of 0, a default value of 600 (10 minutes) will be applied.



Note When a detection is retrieved from the cache, only the malware name is maintained as initially reported, while the other fields can contain static or no information.



- Adding to the previous example, if you wish to also scan files with your own cloud and cache the detection for the next 5 minutes, you might implement the callback in the following way:

```
int APC_scan_callback(SAVAPI_CALLBACK_DATA *data)
{
    SAVAPI_APC_SCAN_DATA *apc_scan_data = data->callback_data.apc_scan_data;
    SAVAPI_APC_REPORT_DATA report_data;

    switch (apc_scan_data->stage)
    {
        case SAVAPI_APC_STAGE_PRE_FILTER:
            return SAVAPI_APC_SCAN_CONTINUE;

        case SAVAPI_APC_STAGE_PRE_HASH_CHECK:
            // scan the file with own cloud

            if (file_is_infected)
            {
                report_data.scan_answer = SAVAPI_APC_ANSWER_INFECTED;
                report_data.malware_info.name = malware_name;
            }
            else if (file_is_clean)
            {
                report_data.scan_answer = SAVAPI_APC_ANSWER_CLEAN;
            }
            else
            {
                // if no detection was found, continue scanning with APC
                return SAVAPI_APC_SCAN_CONTINUE;
            }

            // cache the detection
            report_data.store_cache = 1;
            report_data.ttl = 5 * 60;

            apc_scan_data->set_report_info(apc_scan_data->savapi_fd,
                &report_data);
            return SAVAPI_APC_SCAN_REPORT;

        case SAVAPI_APC_STAGE_PRE_UPLOAD:
            return SAVAPI_APC_SCAN_STOP;

        case SAVAPI_APC_STAGE_POST_SCAN:
            return SAVAPI_APC_SCAN_CONTINUE;
    }
}
```



Note Detections reported in the APC scan callback function will be reflected in the SAVAPI_CALLBACK_REPORT_FILE_STATUS callback.

2.7.8 File Reputation extension quota

Based on your contract type with Avira, there is a limitation to the number of APC requests in a specific time interval.

A quota is the number of requests you can make to the APC within an agreed time interval. The time interval is defined in the contract, and can be any period of time between one minute and one year.

When the quota limit is reached, APC may stop responding to requests until the end of the current time interval, or it may continue to respond to requests, depending on the contract agreements.

If the quota interval is a minute, SAVAPI will try to connect again to APC in one minute, otherwise SAVAPI will try in an hour.

You may have one or several quotas depending on your needs.



Note Within a quota, Avira may also limit the number of inbound requests over a period shorter than the time interval, in order to protect both you and Avira from extraordinary bursts of requests.



Such bursts may overwhelm the service or use up your entire quota in a period much shorter than the contracted time interval.

2.8 Anti-malware SDK (SAVAPI) OnAccess



Note Currently, the OnAccess functionality is implemented only in the library version of SAVAPI.



Note This functionality is available only on Windows systems.

In addition to on-demand file scanning, SAVAPI supports OnAccess scanning via the SAVAPI OnAccess module.

With OnAccess scanning the user is given the possibility to automatically scan files stored on the drives which are being accessed by the operating system or user processes. This adds an additional degree of security since in order for the malware to propagate, the file containing the malware code must first be read or executed. The low-level operation of reading from (including reading for execution) and writing to the file is intercepted by SAVAPI OnAccess, and the file is scanned for viruses before allowing access to it to the requesting process. Therefore, reading from, writing to, or trying to execute the file will trigger SAVAPI to scan the file for malware first. Depending on the SAVAPI OnAccess cache, chosen SAVAPI scan options and file types, the file will either be scanned for malware or directly considered safe. In case the file is considered safe, or has been scanned and found clean, the access to the file is granted to the requesting process.

Alternatively, if the file is found to be infected, access to any other process is restricted and the user is notified.

In case no user interaction with SAVAPI OnAccess is acceptable, then configuring automatic action for infected files is also possible. Please see the configuration section of the manual and the API documentation.

The SAVAPI OnAccess module can be customized by setting options in the protocol and configuration file for the SAVAPI Service, or before SAVAPI Library OnAccess initialization. These options allow control over the time spent in file scan, whether or not files should be rescanned when a process modifies them, scan network drives, extension, file and process exclusions.

For a complete list of configuration options, see [Configuration](#) and, if implementing SAVAPI into a new product, the API documentation.



Note The accessed files are cached, but the caching mechanism is not controllable by the user or by the person implementing SAVAPI.



Note Only clean files are cached so as to not be scanned several times. Altering the file will also update its cache information.



Note If both APC and OnAccess are enabled, only the portable executable (PE) files will be sent to APC upon their execution. Non portable executable (Non-PE) files will not be sent to APC, but will continue to be scanned with the local engine.

2.8.1 Dependencies



Note Terms introduced: OnAccess virtual driver, SAVAPI OnAccess runtime libraries



Note The OnAccess virtual driver is made up of several Windows driver files (kernel modules, catalog files, etc.)

Unlike other SAVAPI modules represented in this manual, SAVAPI OnAccess has a few dependencies. Aside from the valid SAVAPI OnAccess product license, OnAccess virtual driver and additional runtime libraries are also needed for proper operation.



The OnAccess virtual driver, additional SAVAPI OnAccess runtime libraries and install/uninstall driver setup binary are supplied within the SAVAPI package. The setup binary with argument install has to be executed before installing/starting SAVAPI in order for SAVAPI OnAccess to be enabled. If the OnAccess virtual driver needed by SAVAPI OnAccess is not installed or, if the OnAccess license is not valid or has expired, SAVAPI will not provide OnAccess file scanning.



Note The additional dynamic link library file required for SAVAPI OnAccess is *avgio.dll*.

As in the future more than one library might be needed, this library will be referred to as "SAVAPI OnAccess runtime libraries". SAVAPI will try to load the runtime libraries either from the current working directory (location of SAVAPI Service executable) or the library load path specified as command line parameter [Command line parameters --ldpath](#).

As rule of thumb, the OnAccess libraries must be in the same location as the SAVAPI dynamic link library.

2.8.2 Object exclusions

The SAVAPI OnAccess module allows specified objects to be excluded from scanning. This could be helpful when optimizing the application for speed and user responsiveness. The objects to be excluded must be specified by the development team.



Warning Please define exclusions with caution! The excluded files may end up being malware or important attack vectors.

Scanning only certain extensions

OnAccess can be configured to scan only a certain list of extensions. These extensions are only considered for files not being mapped for execution. Any file mapped for execution, no matter its extension, will be scanned. It is therefore possible that files having extensions others than those defined will be scanned.

Defining exclusions for file objects

Files to be omitted from real-time scanning can also be defined. Any file or folder specified here will be ignored, even if they will be mapped for execution. If a directory is excluded, all its sub-directories are automatically also excluded. Keeping this list short is HIGHLY RECOMMENDED, since every file accessed by the operating system will be cross-checked against it.

Wildcards are accepted only if present after the last path separator (backslash), if any.



Note Exclusions for file objects are case sensitive.



Note All options starting with SAVAPI_OPTION_G_OA_ must be separated by a semicolon.

Example: C:\Folder\file.exe;C:\Folder\Subfolder*.doc?;C:\Exclude\All
\From\ Here\;\Device\HarddiskDmVolumes\PhysicalDmVolumes\BlockVolume1\;
.mdb;.md?;F:

Statements like:

C:\Folder*\file.exe

C:\Folde?\fi*.exe

\\.\c:\file.exe

\??\c:\file.exe

are considered invalid.

When excluding an entire drive, not using the backslash after the drive quotation mark is faster. F: performs faster than F:\.

Statements exclusively comprising the following characters are invalid:



* (asterisk), ? (question mark), / (forward slash), \ (backslash), . (dot), : (colon).

Defining excluded processes

All file actions performed by processes defined in this list will be excluded from the onaccess scan operation. Wildcards are accepted only if present after the last path separator(backslash). If no path separator is present, then the expression containing wildcards is invalid.

Excluding a process without full path details only applies to processes where the executable files are located on hard disk drives. Full network path is required for processes whose executable is located on remote drives. Do not specify any exceptions for processes where the executable files are located on dynamic drives (e.g.: USB keys).

The Windows Explorer and the operating system itself cannot be excluded. The specified path and file name of each process must contain a maximum of 255 characters.



Note Excluded processes are case sensitive.

Example:

C:\Folder1\Subfolder\application.exe

C:\Folder2\Subfolder\applicatio?.exe

C:\Folder3\Subfolder\app*.exe

C:\Folder4\Subfolder*.exe

Application.exe

Application1.exe;Application2.exe

App*.exe

Statements like

C:\Folder*\file.exe

C:\Folde?\fi*.exe

\\.c:\file.exe

\??\c:\file.exe

are considered as invalid.

Statements exclusively comprising the following characters are considered invalid:

* (asterisk), ? (question mark), / (forward slash), \ (backslash), . (dot), : (colon).

2.8.3 Updating OnAccess

In order to update the OnAccess files, simply call the *avupdate.exe* specifying the OnAccess configuration file (*avupdate-on-access-savapilib-product.conf*) and the OnAccess info file (*\idx\savapi4oalib-win32-en.info.gz* or */idx/savapi4oalib-win64-en.info.gz*).



Note When updating the OnAccess component, the SAVAPI Library must also be updated in order to keep them synchronized.



Note Updating the OnAccess files (binaries and drivers) does not involve uninstalling the old drivers and installing the newer ones. Updated are just the components from the install directory specified when calling the updater.

More details about the update process in general can be found at [7. Updating Anti-malware SDK \(SAVAPI\)](#).



Updater related files

The following table contains a minimal description of the OnAccess update related files from the SAVAPI package:

Name	Description
<i>avupdate-on-access-savapilib-product.conf</i>	The SAVAPI Library update configuration file for OnAccess modules.
<i>ams_setup.exe</i>	Executable file used for installing/uninstalling OnAccess drivers.

Anti-malware SDK (SAVAPI) OnAccess *ams_setup* binary details

1. Generic considerations

The *ams_setup.exe* has been designed as a multi-platform configurable SAVAPI OnAccess driver maintenance utility. This tool should be used to install and uninstall the SAVAPI OnAccess drivers. Currently, it does not repair a malfunctioning SAVAPI OnAccess installation.

During the update process of SAVAPI OnAccess, if any of the drivers are updated, no automated installation or uninstallation of the drivers will be performed.

2. The log file

This application will write all its messages to a log file. The messages are always appended to the log file and the log file can be opened as read-only when the application is running.



Note The log file is never trimmed/rotated/cleaned; this must be done by the integrator.

3. Usage information

This is a WINAPI application, and therefore does not use any standard I/O streams (*stdout*, *stdin*, *stderr*). The application accepts configuration via arguments presented at start-up, will write all messages to a log file and will have an exit code of 0 (zero) for success.

This executable accepts arguments that control its behavior. It can, for instance, be started via command-line as '*ams_setup.exe arg1 arg2*'.



Note The tool will not abort execution on errors, but errors will be detected and logged.



Note This application must be granted administrator credentials. Otherwise it will not start.

3.1 The install argument

If the 'install' argument is specified, the tool will perform SAVAPI OnAccess driver install operations. These include checking the current state of the environment (already installed components), choosing the correct driver location relative to the respective operating system version and installing the driver components (**.inf**, *.cat* and *.sys files*). All progress report is written to the log file.

A return code of 0 (zero) means that the process finished successfully. All other return codes indicate errors.

Usage: *ams_setup.exe install*

3.2 The "uninstall" arguments

If the 'uninstall' argument is specified, the tool will start the driver uninstallation process. All messages will be appended to the log file.

An exit status of 0 (zero) means that the uninstall process finished successfully. All other return codes indicate errors.

Uninstalling the drivers will require a system restart. A popup message will ask for the user permission. To avoid the popup message, uninstallation must be called with the additional argument *silent*.

Usage: *ams_setup.exe uninstall [silent]*

4. Restarting the system

In order to remove certain components, a system restart is needed: A delete operation is scheduled for the next system start. It is not critical to reboot the system at uninstall time, but a new `ams_setup install` operation will encounter errors if the respective components were not fully removed (in other words, if a system reboot was not performed).

5. Proper driver folder structure

The `ams_setup.exe` will always detect the operating system version and the architecture, but the OnAccess drivers that have to be installed must be contained within a folder having a certain folder structure. The structure is as follows (as present in the SAVAPI SDK package):

```

├── win32
│   ├── win7
│   └── win8
└── win64
    ├── win7
    └── win8
    
```

It is not necessary to have at all times all flavors of drivers present in the containing driver folder; only drivers for the machine the tool is being executed on is needed.

Note The **win8** folder contains drivers compatible with both Windows 8.x and Windows 10 systems.

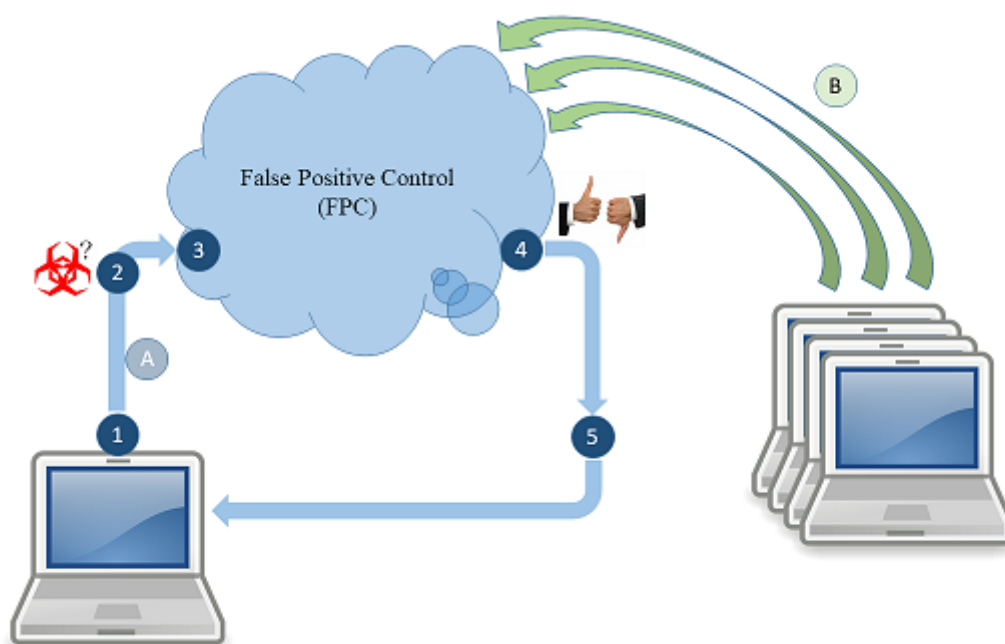
2.9 FPC support

Note This chapter introduces the following terms:

False Positive – A file that is clean/safe, but its contents resemble known malware, so the engine heuristic considers it a member of a known family of malware and reports it as such.

FPC – False Positive Control. An On/Off functionality that controls malware reporting to the developer when malware is found. When turned on, the FPC cloud (based on swarm intelligence) is queried whether this is a true malware (most of the cases) or a false positive (rare). In case of a confirmed false positive, the file is considered clean.

The graphic below illustrates the high-level data flow of the Avira FPC mechanism.



- A – the process between any local instance that is FPC enabled and the False Positive Control cloud:



1. The application has detected an infected file (potential false positive) on a computer.
 2. The file's fingerprint and possible malware information are both extracted and sent to the False Positive Control cloud for review.
 3. The information is analyzed by the False Positive Control cloud.
 4. The yes/no decision on false positive is taken.
 5. The response is sent back to the requester.
- B – the process by which Avira sensor network instances contributes to the cloud knowledge:

In the past, once a file was scanned and malware was detected, SAVAPI offered no functionality to check if the detection is indeed malware or actually a clean file. Certain files can randomly resemble parts of known malware considered to be still in circulation or, in the case of programs, contain unusual or malware-like pieces of code, but are otherwise safe to be opened or executed. Furthermore, the engine heuristic is a feature that is crucial in detecting new members of a malware family, but, also depending on its configured heuristics level, can report clean files as being malware.

It has been an age-old problem of tweaking the engine heuristics to offer the best in-the-field detection of new malware types while having low false positive detection rates, but the false positive detections' issue cannot be solved by tweaking the engine and cloud heuristics alone.



Note The Avira FPC cloud **does not store** information that can be used to identify the users. The Avira FPC cloud, however, does store relevant information about the file being queried, such as, but not limited to: file name, name of possible malware detection, file size, unique file hash, file creation and modification time, etc.

The FPC functionality comes to complement the Avira engine and cloud detection capabilities by introducing an additional verification step before reporting the detection as malware. This verification step means querying information about the file (such as file name, location, size, unique hash, date of creation, issuer, etc) to the Avira FPC servers, where the definitive infected-or-clean status of the file could be known. If the respective file's status is known to be a false positive (clean file reported as infected), then the local engine and APC detection is ignored, and the developer is informed that the file is clean. In case no information about the file's status exists in the Avira FPC cloud, the file is considered to be truly infected for the time being, and the developer is notified about malware status and specific type. If at some point in the future, after the file has been investigated by a virus analyst, it is found to be clean, subsequent queries to the Avira FPC cloud will return the file status as clean.



Note The FPC functionality is turned off by default and must be explicitly activated. When implementing the SAVAPI Library, set `SAVAPI_OPTION_FPC` for each SAVAPI instance. When using the SAVAPI daemon, set/get option `FPC`.

Virus analysts and researchers routinely check the files reported to the Avira FPC cloud, gather relevant information from the Avira sensor network and update information stored in the Avira FPC cloud. However, the Avira FPC cloud should be regarded as an automated query-based system, where an answer is given depending on what information is available at that *specific time*.



Note In order for **the Avira FPC to work, a network connection is required**. If FPC is enabled and no proxy is defined by the developer at SAVAPI initialization, then one will be searched in the environment and, in the case of Windows systems, in the system settings. A proxy for the SAVAPI daemon can be set in the configuration file. If no proxy is set and FPC is enabled, then one will be searched in the environment and, in the case of Windows systems, also in the system settings.



Note When implementing the SAVAPI Library, the proxy can be set by calling `SAVAPI_global_set` with the option `SAVAPI_OPTION_G_PROXY` before creating any SAVAPI instances. When using the SAVAPI daemon, set the option `Proxy` in the configuration file.

2.9.1 FPC blackout mechanism

The FPC blackout mechanism improves the scan performance in scenarios where a connection to the Avira FPC cloud is generally available, but with interruptions. When such a temporary availability issue is detected, SAVAPI's FPC module is temporarily disabled (for a configurable amount of time), so that



no time will be lost trying to reach the Avira FPC cloud using a temporarily faulty network connection. The scan process will still be reliable, even though it will rely on the local engine and, depending on the respective network issue, the APC scan service.

The FPC blackout mechanism behaves identical to the APC blackout mechanism (please see [Blackout mechanism](#)).

If during normal operation a consecutive number of time-outs occur while querying the Avira FPC cloud, the SAVAPI FPC module will be disabled for a configurable amount of time. Once that amount of time has passed, a single query will be issued to the Avira FPC cloud. If the Avira FPC cloud returns an answer to the respective query, then SAVAPI fully re-enables the FPC module. If the query fails, then SAVAPI's FPC module re-enters the blackout.

The time before a time-out is declared, the number of consecutive time-outs and the time SAVAPI's FPC module will be disabled are configurable for both the SAVAPI Library and the SAVAPI daemon.



Note When implementing the SAVAPI Library, the options that have to be set in order to configure the blackout timings are `SAVAPI_OPTION_FPC_TIMEOUT`, `SAVAPI_OPTION_G_FPC_BLACKOUT_TIMEOUT` and `SAVAPI_OPTION_G_FPC_BLACKOUT_RETRIES`. When using the SAVAPI daemon, options `FPCTimeout`, `FPCBlackoutTimeout` and `FPCBlackoutRetries` have to be set in SAVAPI daemon's configuration file.

3 Anti-malware SDK (SAVAPI) Service

3.1 General description

SAVAPI Service is an application based on SAVAPI Library. It offers all the features of the SAVAPI product, **except**:

Scanning files mapped in memory (possible only if the combo SAVAPI Service + SAVAPI Client Library is used)

SAVAPI Service is a multi-thread application. When started, it creates a pool with worker threads and listens to client requests on a network socket (TCP/IP or UNIX local socket). When a client request arrives, it is associated to a worker thread that will process and answer all the client commands.

SAVAPI Service communicates with the client via a text-based protocol. Through this protocol, the client applications can

- configure the service,
- obtain information about the service status, configuration option values or licensing,
- send scan requests,
- stop the service.

The **maximum command size** allowed by the SAVAPI Service is set to:

```
2 * PATH_MAX + 64
```

This setting allows you to scan any possible file-path of the current file-system.

- `PATH_MAX` is the maximum length of a file-system path. It is a platform-dependent system variable on UNIX. On Windows, it is limited to 8192.
- The maximum value allowed by SAVAPI includes the command's arguments, line terminating characters (EOL) and also the string terminating NULL character: `<command><arguments><EOL><\0>`



Note `ENCODE_FILENAMES` should be used for sending paths and file names that contain control characters, or which begin or end with white-spaces through the socket. An alternative to `ENCODE_FILENAMES` is the `SCAN hex_enc://` command (see [The SCAN command](#)).



3.2 Integration

3.2.1 On-demand file scanning

In this mode, the SAVAPI Service will be started and will listen on a specified interface (or the default interface, if none is specified).

The client application(s) will:

- connect to the network socket where SAVAPI Service is listening,
- perform the handshake with the service,
- set/ read options value,
- send scan requests,
- read scan answers.

The client application is responsible to implement all the communication-related workflow (create a socket, connect to the SAVAPI Service listening socket, read and write from the socket, react to socket I/O errors) and its own workflow (create threads for parallel requests, prioritize the scan requests, etc).

3.2.2 OnAccess file scanning



Note Currently, the OnAccess functionality is implemented only in the library version of SAVAPI.



Note This functionality is available only on Windows systems.

To use the OnAccess file scanning functionality provided by SAVAPI, a valid SAVAPI OnAccess product license, the OnAccess virtual driver and additional SAVAPI OnAccess runtime libraries have to be installed.

SAVAPI will try to access the virtual device associated with the installed OnAccess virtual driver and the SAVAPI OnAccess runtime libraries. Failure in accessing either of these, or failing to validate the SAVAPI OnAccess product license will make SAVAPI disable the OnAccess module. The SAVAPI Service implements functionality exported by the SAVAPI Library.

Further information about the SAVAPI OnAccess module can be found in chapter [Anti-malware SDK \(SAVAPI\) OnAccess](#) and its sub-chapters.

3.3 Configuration

There are more ways to configure the SAVAPI Service features and options.

- **Using command line parameters**

The service accepts command line parameters at startup. A complete list with the accepted parameters, their functions and accepted values is detailed below. See [3.3.1 Command line parameters](#).

- **Using configuration file options**

Through a command line parameter, the service can receive a configuration file containing the options and their values. A complete list with the accepted options, their default values and their valid values is detailed below. See [3.3.2 Configuration file options](#).

- **Using the SAVAPI protocol SET commands**

SAVAPI Service uses a text-based protocol to communicate with the outside world. Through the protocol commands, the SAVAPI Service can be configured to answer to commands and requests. See [3.3.3 Protocol](#).



3.3.1 Command line parameters



Note Some of the command line parameters are not available for all the supported platforms. For example, the parameters regarding the user and group are available only for UNIX-based operating systems, because in Windows the service runs in the background as a Service under the Local System Account user.

SAVAPI will consider all numeric values as decimal (base 10).

The command line parameter types include:

- **Options** – allow you to set some specific options at program startup
- **Commands** – allow you to send some specific commands to the service/daemon
- **Information** – allow you to retrieve various data regarding the service/daemon (help message, version information, etc)

Options

- **-N**

Starts the SAVAPI Service in the foreground. SAVAPI will run as a process. This option is not activated by default. The service will start in the background.

- **--pool-scanners=<scanners number>**

Starts the SAVAPI Service with the specified number of workers. Default: 24; Minimum: 1; Maximum: 300.

- **-C, --config=<configuration file location>**

Specifies the location of the SAVAPI Service configuration file.

Default: <SAVAPI_service_binary_directory>/savapi.conf.

- **--temp=<scanning temporary files location>**

Specifies the location of the scanning temporary files. The temporary files created while scanning and unpacking will be stored in this location.

Default: /var/tmp on UNIX, %temp% on Windows.



Note It is recommended for the location to not be a directory that contains sensitive files, such as SAVAPI binaries or configuration files.

- **--key-file=<key file location>**

Specifies the location of the SAVAPI Service key file.

Default: <SAVAPI_service_binary_directory>/HBEDV.KEY or
<SAVAPI_service_binary_directory>/hbedv.key

- **--vdf-dir=<VDF files location>**

Specifies the location of the VDF files.

Default is the location of the SAVAPI Service binary.

- **--pid-dir=<PID file location>**

Specifies the location of the PID files.

The default location is: '\\var/tmp'.



Note This option is not available on Windows systems.


- **--modules-dir=<modules_directory_location>**




Specifies the directory where SAVAPI will store the files needed for [Non-disruptive service update](#).

Default is the value of the option `--temp=<scanning temporary files location>`.

The specified location must exist and must have the "rwx" permissions for the user who owns the SAVAPI process.

 **Note** The directory must exist on an executable file system, since SAVAPI will try to load and execute the files from that path. In order to optimize some internal processes, it is recommended to have both installation and modules directories on the same partition.

 **Note** This option will be ignored if the option `--duplicate-modules` is disabled.

- **--duplicate-modules**

Activates the usage of modules files duplication in order to perform updates without disrupting the service. The option does not require a value. If this option is not set, the function is disabled.

For detailed information, see [Non-disruptive service update](#).


- **--socket-file=<socket file location>**

Specifies the location and the name of the UNIX local socket. The UNIX local socket file is removed upon SAVAPI Service exit.


- If the option [CreateSocketDir](#) is disabled, the location provided for the local socket must exist; the user and group used to start the SAVAPI Service must have proper access permissions to the location. Otherwise, SAVAPI Service does not start and it returns an error message, describing the reason for which the location cannot be used. For example, for a socket file specified as `--socket-file=/ non_existent_dir/socket_file`, SAVAPI Service does not start and it returns an error message, stating that *the directory /non_existent_dir/ does not exist*.

- If the option [CreateSocketDir](#) is enabled and the provided socket file path does not exist, SAVAPI Service creates the parent directory of the socket file at startup. For example, for a socket file specified as `--socket-file=/ non_existent_dir/socket_file`, SAVAPI Service creates the path `/ non_existent_dir/`.


The directory is created with the same user and group used to start SAVAPI Service. The directory permissions are calculated as the addition of the socket permissions and the executable bit. By default, socket permissions are 0600, so the default permissions for the created directory are 0700. The created directory is not deleted when SAVAPI Service is stopped.

 **Note** The socket file and directory owners (user and group) may be different than the user and group used to start SAVAPI Service, depending on the `setuid` and `setgid` bits.

Only the socket file's parent directory is created; if more than one directory from the provided socket file path do not exist, SAVAPI Service returns an error.

 **Note** If the socket directory exists, it will not be changed in any way regarding user, group or permission settings.

Default: `/var/tmp/.SAVAPI`

 **Note** This option is not available on Windows systems.

- **--socket-permissions=<mode>**

Specifies the permissions for the UNIX local socket. By default, the socket file is created with the same `user/group` as the running SAVAPI Service and with `mode=0600`. The socket permissions will also be used when creating directories from the socket file location.

 **Note** This option is not available on Windows systems.



- **--log-file=<log file location>**

Specifies the log file name and location. There is no default value.

- **--ldpath=<dependencies location>**

Specifies the path to SAVAPI libraries. Default: location of the SAVAPI Service binary.

- **--install**

On Windows, it registers the service in Service Control Manager (description, display name, binary image path, auto start).



Note This option is available only on Windows systems.

- **--uninstall**

On Windows, it removes the registration of the service from the Service Control Manager.



Note This option is available only on Windows systems.

- **--tcp=[host:]port**

Specifies the port (and optionally, the host) to which the SAVAPI TCP/IP listening socket bounds. If no host is provided, SAVAPI bounds to localhost.

`host` can be either a host name or an IP address in any accepted format (dotted or not): decimal, hexadecimal, octal.



Note SAVAPI supports only IPv4 addresses.

On UNIX systems this forces SAVAPI to use TCP/IP sockets rather than UNIX local sockets. On Windows systems this option is **mandatory**.

The service will not start, if the interface is not specified, either through this option or through the configuration file.

- **--pool-connections=<pool connections number>**

Defines the length of the pending connections queue. Default: 48; Minimum: number of `--pool-scanners`; Maximum: 900.

- **--ave-dir=<AVE files location>**

Specifies the location of the engine files. The default is the location of SAVAPI Service binary.

- **--fops-lib=<path/to/fops/library>**

Specifies the location of SAVAPI's `fops` plug-in library. Default value: internal SAVAPI `fops` implementation will be used.

- **--fops-lib-params="<blank separated list of parameters>"**

Specifies the SAVAPI's external `fops` library blank separated parameters. Default value: No parameters will be sent to the SAVAPI external `fops` library.

- **--no-spv**

SAVAPI starts without the supervisor process.



Note This option is not available on Windows systems.

- **--apc-mode="<auto|disabled|check-only|full>"**

Specifies the mode in which the Avira Protection Cloud (APC) component will be used for scanning files.



Available options:

`auto` - will try to activate `full` mode if possible, otherwise the default is used `disabled`

`disabled` - APC scanning disabled

`check-only` - only send file fingerprints to the APC

`full` - complete APC scanning functionality (send file fingerprints, upload files)

Default value: `auto`

- **--apc-cert-dir="`<APC certificate directory path>`"**

Specifies the certificate directory path. Default: the main SAVAPI Service binary location.

Commands

- **--stop**

Stops SAVAPI. This will work only if you specify the running SAVAPI interface, either by command line options (`--tcp`) or by configuration file (`--config=<configuration_file>`).



Note The `stop` command will not be executed if the user who sends the command does not have enough privileges (on UNIX it means that the user does not have enough privileges to connect to the SAVAPI Service). This command is not available on Windows systems.

- **--status**

Retrieves the current status of SAVAPI. This will work only if you specify the running SAVAPI interface, either by command line options (`--tcp`) or by configuration file (`--config=<configuration_file>`).



Note The `status` command will not be executed if the user who sends the command does not have enough privileges (on Windows this means the user cannot send commands to services, on UNIX it means that the user does not have enough privileges to connect to the SAVAPI Service). On Windows systems, this command will work only if SAVAPI is registered as a service, using the option `--install`.

- **--reload-engine**

Reloads the engine. This enables SAVAPI to use the newly updated engine without restarting or disrupting the service. In case the reload fails, SAVAPI will try to continue with the old engine, otherwise the service will stop.



Note To use this command, make sure you started the SAVAPI Service with the option `--duplicate-modules`.

For detailed information, see [Non-disruptive service update](#).

- **--report**

Prints relevant startup information about all running SAVAPI processes started from the same binary directory. Command's output format:

```
<process_info_1>\n<process_info_2>\n..  
<process_info_n>\n  
<process_info_1> = <info_1>\n<info_2>\n..  
<info_m>\n  
<info_1> = [<pid>] <field_name>: <field_value>\n  
<pid>= the process ID of the SAVAPI process  
<field_name> = CMDLINE, INTERFACE, DUPLICATE_MODULES
```

The values of each field name specifies the following:

- `CMDLINE`



Specifies the full startup command line of the SAVAPI process.

CMDLINE: <"full_process_startup_cmdline">

- INTERFACE

Specifies the listen interface of SAVAPI process. It can be an Unix socket or a TCP socket.

INTERFACE: <host/socket_file:port_number/0>

- DUPLICATE_MODULES

Specifies the value of the option `--duplicate-modules`. This shows if the SAVAPI process can successfully execute a reload-engine command. A value of "1" means that the option is activated.

DUPLICATE_MODULES: <0/1>

Example for three SAVAPI processes started from `/var/tmp/savapi/savapi`:

```
# cd /var/tmp/savapi
# ./savapi --no-spv -N &
# ./savapi --no-spv -N --tcp=9999 &
# ./savapi --tcp=8888 --duplicate-modules
# ./savapi --report
[18329] CMDLINE: "/var/tmp/savapi/savapi" "--no-spv" "-N"
[18329] INTERFACE: /var/tmp/.SAVAPI:0
[18329] DUPLICATE_MODULES: 0
[7876] CMDLINE: "/var/tmp/savapi/savapi" "--no-spv" "-N"
"--tcp=9999"
[7876] INTERFACE: localhost:9999
[7876] DUPLICATE_MODULES: 0
[8123] CMDLINE: "/var/tmp/savapi/savapi" "--tcp=8888"
"--duplicate-modules"
[8123] INTERFACE: localhost:8888
[8123] DUPLICATE_MODULES: 1
```

If no SAVAPI processes are running, a relevant message is displayed when running the command:

```
# ./savapi --report
No processes are running.
```

This command is intended to ease the usage of supervisors or update scripts, as it provides startup information in a format easy to parse with command line tools.



Note This command is not available on Windows systems.



Note The report functionality uses shared memory segments and semaphores. If the system limits because of maximum open shared memory segments, or if the semaphores are too low, the SAVAPI will not provide any report information and the `--report` command might fail.

For example: On a UNIX system set the maximum semaphores to 0: `$ sudo sysctl -e kernel.sem="0 32000 32 0"`

Start SAVAPI: `$ <some_dir>/savapi --report`

Try to check the report: `$ <some_dir>/savapi --report`

Report failed. Please check log for details

Information

- **-V, --version**

Displays the version information.

```
# ./savapi --version
```

Product build: Linux (x86_64, glibc 2.19)

SAVAPI Service version: 4.0.0.148

Component versions:



SAVAPI Library version: 4.0.0.148

Engine version: 8.3.32.0

Packlib version: 8.4.0.80

VDF version: 8.11.242.104

APC Library version: 2.8.0.3

- **-h, --help, -?**

Displays the help message.

3.3.2 Configuration file options

All the lines starting with '#' or ' ' (space) in the configuration file are ignored, when options are parsed. The format of the configuration is as much as possible a 1:1 match with the SAVAPI TCP/IP protocol.

Accepted forms of writing the parameters in the configuration file are:

```
'Attribute=value'
```

```
'Attribute value'
```

Attributes are case insensitive. Values are case sensitive. Attributes are separated by CR.

The default values are provided for all the options, where such values exist.

The SAVAPI Service will not start, if invalid values are provided for any of the configuration file options. An error will be printed regarding the invalid value.

SAVAPI will consider all numeric values as decimal (base 10).

System hard-limit:

On UNIX, SAVAPI uses the system hard-limit (see `ulimit -Hn`) for the maximum number of opened files allowed by the process. This limit is shared between SAVAPI connections (the higher the archives' recursion level, the more files will be created). When using large values for the options `PoolScanners`, `PoolConnections` and `ArchiveMaxRec`, unexpected errors (scanning proc-errors, failed connections) may occur, if the system hard-limit is too small.

Anti-malware SDK (SAVAPI) Service running options

- **CurrentWorkingDirectory**

```
CurrentWorkingDirectory </path/to/working/folder>
```

Sets the current working directory for SAVAPI. This eliminates the need to specify full paths in file names, when commands that accept file names as parameters are used.

Available values: only absolute paths are accepted; relative paths are not accepted and the service exits with an error.

Default value: the SAVAPI Service binary location.

- **TextMode**

```
TextMode <ASCII-PRINT | LOCALE | UTF-8>
```

Specifies the method used for character encoding. The SAVAPI protocol provides three modes for handling text. The modes are applied to both incoming and outgoing data.

Available values:

- ASCII-PRINT

Only printable characters of the ASCII set are considered valid characters. These include the characters 9 and 32-126. All other character values are converted to '?' (63). This is a conservative setting for clients that want to keep communication restricted to 7-bit printable text.



- LOCALE

All characters will be handled "as-is". It is the responsibility of the client to ensure that the SAVAPI Service will be able to correctly interpret the text (for example, with matching locale settings between client, service and file system).



Note 0-value characters will not be handled correctly in this mode. Encoding such as UTF-16LE or UTF-32LE are not supported by the SAVAPI TCP/IP protocol.

- UTF-8

All texts must be correctly encoded as UTF-8. Invalid encodings will result in error responses.

Default value: LOCALE



Note If invalid text is received, the SAVAPI Service will output an error (whenever this is possible) or drop the connection.

- **PoolScanners**

`PoolScanners <no_of_workers>`

Specifies the number of SAVAPI workers. SAVAPI will start with the specified number of workers.

SAVAPI will not accept an infinite number of scanners. The maximum accepted number is 300.

Available values: 1 - 300

Default values: 24

- **PoolConnections**

`PoolConnections <no_of_pool_connections>`

Specifies the length of pending connections queue.

SAVAPI will not accept an infinite number of pending connections. The maximum accepted number is 900.

Available values: 1 - 900

Default values: 48



Note Since it makes no sense having more available workers than accepted connections, if the number of `PoolConnections` is lower than `PoolScanners`, the value received from `PoolScanners` will be used for both options.

- **User**

`User <user>`

Specifies the SAVAPI Service credentials.

SAVAPI Service will start and will drop its credentials, as soon as possible, to the specified user.

Available values: user name.

Default value: There is no default value for this option. The service will run with the user and group it was started with (if nothing is specified in the configuration file).



Note This option is available only on UNIX systems.

- **Group**

`Group <group>`


Specifies the SAVAPI Service credentials.



SAVAPI Service will start and will drop its credentials, as soon as possible, to the specified group.

Available values: group name.

Default value: There is no default value for this option. The service will run with the user and group it was started with (if nothing is specified in the configuration file).

 **Note** This option is available only on UNIX systems.


- **PidDir**

`PidDir </path/to/the/pid/dir>`

Specifies the SAVAPI Service PID file location.

Available values: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: `/var/tmp`.

 **Note** This option is available only on UNIX systems.

- **KeyFile**

`KeyFile </path/to/the/key/file>`

Specifies the location of the SAVAPI license key file.

Available values: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: `<SAVAPI_service_binary_directory>/HBEDV.KEY` or `<SAVAPI_service_binary_directory>/hbedv.key`

- **AveDir**

`AveDir </path/to/the/ave/dir>`

Specifies the location of the engine files.

Available values: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: The location of the SAVAPI Service binary.

- **VdfDir**

`VdfDir </path/to/the/vdf/dir>`

Specifies the location of the VDF files.

Available values: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: The location of the SAVAPI Service binary.

- **ModulesDir**

`ModulesDir </path/to/the/modules/dir>`


Specifies the directory where SAVAPI will store the files needed for [Non-disruptive service update](#).


Available values: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: The value of the option `--temp=<scanning temporary files location>`

The specified location must exist and must have "rwx" permissions for the user who owns the SAVAPI process.



 **Note** The directory must exist on an executable file system, since SAVAPI will try to load and execute the files from that path. In order to optimize some internal processes, it is recommended to have both installation and modules directories on the same partition.

 **Note** This option will be ignored if the option `--duplicate-modules` is disabled.

- **AttachToGuard**

`AttachToGuard <0|1>`

The SAVAPI Service will register with the Avira Realtime Protection (Guard) service, to prevent the Guard service from scanning the files that are scanned by the SAVAPI Service.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1 (enabled).

 **Note** This option is available only on Windows systems.

- **DuplicateModules**

`DuplicateModules <0|1>`

Activates the usage of modules files duplication in order to perform updates without disrupting the service.

Available values: 0 (disabled) or 1 (enabled).

Default value: 0 (disabled).

For detailed information, see [Non-disruptive service update](#).

- **FopsLib**

`FopsLib <path/to/fops/library>`

Specifies the location of SAVAPI's `fops` plug-in library.

Available values: Only absolute paths are accepted. Relative paths will not be accepted and the service will exit with an error.

Default value: Internal SAVAPI `fops` implementation will be used.

- **FopsLibParams**

`FopsLibParams "<blank separated list of parameters>"`

Specifies the SAVAPI's external `fops` library blank parameters.

Available values: A list of parameters separated by blank spaces.

Default value: No parameters will be sent to SAVAPI external `fops` library.


- **SocketPermissions**

`SocketPermissions <mode>`

Specifies the permissions for the UNIX local socket.

Available values: mode permissions

Default value: The socket file is created with the same `user/group` as the running SAVAPI Service and with `mode=0600`. The socket permissions will also be used when creating directories from the socket file location.

 **Note** This option is not available on Windows systems.



- **CreateSocketDir**

CreateSocketDir <value>

If this option is enabled and the provided socket file path does not exist, SAVAPI Service creates the parent directory of the socket file at startup.

For more details, see the description of the command line option `--socket-file=<socket file location>` and the configuration file option [ListenAddress](#).

Available options: 0 (disabled) or 1 (enabled).

Default value: 0.



Note This option is not available on Windows systems.

Connection-related options

- **Listen Address**

Specifies the SAVAPI Service listening interface.

Syntax for TCP connections:

ListenAddress <inet:port[@host]>

The host can be either an IPv4 address or a hostname, in any accepted format (dotted or not): decimal, hexadecimal, octal.

Syntax for UNIX local sockets connections:

ListenAddress <unix:/path/to/AF_UNIX/socket>

For more details about the usage of UNIX local sockets, see the description of the command line option `--socket-file=<socket file location>`.

Available values: listening interface location.

Examples:

Bind to TCP port 3333 on local host: ListenAddress=inet:3333

Bind to TCP port 4444 on loop back interface: ListenAddress=inet:4444@127.0.0.1

Bind to TCP port 5555 on public interface: ListenAddress=inet:5555@testcomputer

Bind to UNIX local socket "/var/tmp/.SAVAPI": ListenAddress=unix:/var/tmp/.SAVAPI

Default value: '/var/tmp/.SAVAPI' on UNIX. There is no default value on Windows.

- **ConnectionTimeout**

ConnectionTimeout <value-in-sec>

Specifies the timeout in seconds for the worker connection. A value of "0" means no timeout.

Available values: 0 – INT32_MAX.

Default value: 60.

Available only in daemon/service mode.

- **ProtocolStrict**

ProtocolStrict <0|1>

If set, SAVAPI closes the connection for unknown protocol commands, otherwise the connection is kept open whilst the unknown commands are discarded.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1 (enabled)



Note: It is strongly recommended to keep this option enabled in order to mitigate against Cross-site request forgery attacks.

- **Proxy**

Proxy <proxy-server>

Specifies the connection proxy server.

Available values: Valid http/socks proxy server.

Examples:

Proxy=10.0.0.1:3128

Proxy=http://proxy-server:3128

Proxy=socks4://socks-proxy-server

Proxy=http://username:password@proxy-server:3128

Default value: none



Note

- To specify a port number in this string, append [port] to the end of the host name. If not specified, SAVAPI will use the port 1080 as default.
- The proxy string may be prefixed with [scheme]:// to specify the kind of proxy to be used. Supported schemes are: http://, socks4://, socks4a:// and socks5://.
- If no protocol is specified, the proxy will be treated as a HTTP proxy server.
- If the proxy requires authentication, the credentials can be specified by adding username:password before the host name, as shown in one of the examples above.
- If no proxy is provided, SAVAPI will try to read it from other sources, in the following order: Internet Explorer settings (for Windows systems only), Winhttp settings (for Windows systems only), environment variables in the following order: https_proxy, HTTPS_PROXY, http_proxy, HTTP_PROXY, all_proxy, ALL_PROXY. Please note that the no_proxy variable is not considered as environment variable when using the --system-proxy command. See also --system-proxy
- Winhttp proxies are set using proxycfg / netsh tools. For x64 Windows operating systems it is important to set Winhttp info in the proper registry. For SAVAPI 32 bits version, it will try to read Winhttp settings from Wow6432Node registry root. In order to set proxy information there, the user must run proxycfg / netsh in the folder C:\Windows\SysWOW64. This is not needed for 32 bits Windows operating systems or, when SAVAPI 64 bits is used (proxycfg / netsh can be called from any folder).
- For Windows systems, please note that Internet Explorer settings will be available only when running SAVAPI as a process, and not as a service.
- If no proxy is found in any of the sources mentioned above, SAVAPI will use a direct connection.
- Please note that although the HTTPS proxy scheme is accepted, HTTP will be used instead for proxy tunneling. There is no support for HTTPS proxies, but the scheme is still accepted for backward compatibility reasons. HTTPS is supported for APC and avupdater communications with Avira servers, but the already encrypted traffic will not be re-encrypted for the proxy tunneling.

Scan-related options

- **FPC**

FPC <0|1>



Enables or disables "False Positive Control", that cross-checks detections against a known list of false positives (stored in the Avira FPC Cloud).

Available values: 0 (disabled, default) or 1 (enabled)

Default value: 0 (disabled)

- **FPCTimeout**

`FPCTimeout <seconds>`

Specifies the number of seconds SAVAPI will wait for an FPC response before timing out. This value is applied individually for each scanned object.

Available values: 0 - 86400

Default value: 20

A value of 0 (zero) sets the wait time to infinity.

The value must be smaller than [ScanTimeout](#)

- **FPCBlackoutTimeout**

`FPCBlackoutTimeout <seconds>`

Specifies the number of seconds SAVAPI will wait in blackout, before checking that the connection to the Avira FPC Cloud has been re-established.

Available values: 1 - 86400

Default value: 300

- **FPCBlackoutRetries**

`FPCBlackoutRetries <count>`

Specifies the number of consecutive timeouts allowed before declaring FPC unreachable.

Available values: 0 - 32767 (0 - INT16_MAX)

Default value: 5

A value of 0 (zero) makes SAVAPI always try to contact the Avira FPC Cloud, effectively disabling the blackout mechanism.

The count applies to all items being scanned (for instance objects inside an archive).

- **APCMode**

`APCMode <auto|disabled|check-only|full>`

Specifies the mode in which APC will be used for scanning files.

Available values:

`auto` – will try to activate `full` mode if possible, otherwise APC will be set to `disabled`.

`disabled` – APC scanning disabled

`check-only` – only send file hashes to the APC

`full` – complete APC scanning functionality (send file hashes, upload files)

Default value: `auto`

- **APCCacheSize**

`APCCacheSize <size>`

Sets the maximum allowed size for APC cache size.

Available values: 0 - 104857600 (0 - 100 MB)

Default value: 5M



A value of "0" means that the APC cache is disabled. <size> can include K or M as a label. Otherwise the number is assumed to be in bytes. A value below 1024 will be automatically set to 1024.

Examples: 100M or 32K. (1K=1024 bytes. 1M=1024^2 bytes.)

- **APCCacheDumpFile**

APCCacheDumpFile <0|1>

Enable APC cache dump. When SAVAPI starts, an existing cache from the previous dump will be imported, if there is any.

Available values: 0 (disabled), 1 (enabled)

Default value: 1 (enabled)

- **APCCacheDumpFilePath**

APCCacheDumpFilePath </path/to/dump/file>

Specifies the cache dump file path.

Available values: Only absolute paths are accepted. Relative paths are not accepted and the service will exit with an error.

Default value: Default Operating System temporary folder/ savapi_apc_cache_XXXXXXXXX.dat

- **APCConnectionTimeout**

APCConnectionTimeout <time-in-seconds>

Specifies the number of seconds SAVAPI will wait for establishing a connection to APC, before timing out. This value applies to each object scanned with APC (for example, a PE file in an archive). This value must be smaller than APCScanTimeout.

Available values: 0 - 86400 (1 second - 24 hours)

Default value: 20

A value of "0" means that SAVAPI will wait indefinitely to establish a connection to APC.

- **APCScanTimeout**

APCScanTimeout <time-in-seconds>

Specifies the number of seconds SAVAPI will wait for data transfer to/from APC before timing out. This value applies to each scanned object (for example, a file in an archive). This value must be greater than APCConnectionTimeout and smaller than ScanTimeout.

Available values: 0 - 86400 (1 second - 24 hours)

Default value: 30

A value of "0" means that SAVAPI will wait indefinitely for data transfer to/from APC.

- **APCBlackoutRetries**

APCBlackoutRetries <value>

Specifies the maximum number of consecutive timeouts allowed before declaring APC as unreachable.

Available values: 0 - 32767 (0 - INT16_MAX)

Default value: 5

A value of "0" tells SAVAPI to always attempt to use APC for scanning files.

- **APCBlackoutTimeout**

APCBlackoutTimeout <time-in-seconds>

Specifies the number of seconds after which SAVAPI will try to establish another connection to APC.



Available values: 1 - 86400 (1 second - 24 hours)

Default value: 300

- **APCCertDir**

APCCertDir <path/to/the/cert/dir>

Specifies the location of the certificate file.

Available values: Only absolute paths are accepted. Relative paths will not be accepted and the service will exit with an error.

Default value: The SAVAPI Service folder (the location where the SAVAPI Service binary resides).



Note The name of the certificate must be `cacert.crt`, otherwise the APC functionality will not work.

- **APCCheckRiskRatingLevel**

APCCheckRiskRatingLevel <value>

This option sets a minimum threshold for hash requests sent to Avira Protection Cloud regarding the risk rating of the file. The risk rating is based on a frequently updated mathematical model to evaluate the risks posed by a certain file.

A threshold of 0 allows hash requests to be sent even if the files have a very low risk rating, while a value of 7 allows hash requests only for files with the highest assumed risk.

Available values: 0 - 7 (0=very low risk, 2=low risk, 3=moderate risk, 4=high risk, 7=very high risk)

Default value: 4

- **APCUploadRiskRatingLevel**

APCUploadRiskRatingLevel <value>

This option sets a minimum threshold for uploads into the Avira Protection Cloud regarding the risk rating of the file. The risk rating is based on a frequently updated mathematical model to evaluate the risks posed by a certain file.

A threshold of 0 uploads files even if they have a very low risk rating, while a value of 7 uploads only the files with the highest assumed risk.

Specifies the pre-upload filter threshold for unknown files.

An unknown file will be uploaded to APC only if the malware probability is greater than or equal to this threshold.

Available values: 0 - 7 (0=very low risk, 2=low risk, 3=moderate risk, 4=high risk, 7=very high risk)

Default value: 4



Note If `APCCheckRiskRatingLevel` is greater than `APCUploadRiskRatingLevel`, the file will be uploaded only if the malware probability is greater than or equal to `APCCheckRiskRatingLevel`.

- **APCPEMode**

APCPEMode <disabled|check-only|full>

This option specifies the mode in which APC will be used for scanning portable executable (PE) files.

Available values:

`disabled` – PE files will not be scanned with APC

`check-only` – only hashes of PE files will be checked with APC

`full` – full APC scanning functionality for PE files



Default value: full



Note When APCMode is "check-only", PE files will not be uploaded even if APCPEMode is "full".

- **APCELFMode**

APCELFMode <disabled|check-only|full>

This option specifies the mode in which APC will be used for scanning ELF (UNIX executable) files.

Available values:

disabled – ELF files will not be scanned with APC

check-only – only hashes of ELF files will be checked with APC

full – full APC scanning functionality for ELF files

Default value: disabled



Note When APCMode is "check-only", ELF files will not be uploaded even if APCELFMode is "full". This option depends on the value of APCMode. Full APC ELF functionality also depends on APC backend settings.

- **APCMachOMode**

APCMachOMode <disabled|check-only|full>

This option specifies the mode in which APC will be used for scanning Mach-O and Apple Universal Binary files.

Available values:

disabled – Mach-O files will not be scanned with APC

check-only – only hashes of Mach-O files will be checked with APC

full – full APC scanning functionality for Mach-O files

Default value: disabled



Note When APCMode is "check-only", Mach-O files will not be uploaded even if APCELFMode is "full". This option depends on the value of APCMode. Full APC Mach-O functionality also depends on APC backend settings.

- **APCFileExtensionsPolicy**

APCFileExtensionsPolicy <auto|custom>

This option specifies the policy for the files that will be scanned with APC.

Available values:

auto – all file extensions supported by SAVAPI internal list will be scanned with APC

custom – user-defined list of extensions to be scanned with APC

Default value: custom



Note Setting this option will reset APCFileExtensionsDisabled, APCFileExtensionsCheckOnly and APCFileExtensionsFull lists. This option depends on the global APCMode.

- **APCFileExtensionsDisabled**

APCFileExtensionsDisabled <extensions>

This option specifies a list of extensions for the files which will not be scanned with APC.

Available values: A string containing extensions, separated by semicolons



Example: .xlsx;.pdf;.doc

Default value: None



Note This option has a higher priority and will refine the APCFileExtensionsPolicy option. This option depends on the global APCMode.

- **APCFileExtensionsCheckOnly**

APCFileExtensionsCheckOnly <extensions>

This option specifies a list of extensions for the files that will be hash-checked with APC.

Available values: A string containing extensions, separated by semicolons

Example: .xlsx;.pdf;.doc

Default value: None



Note This option has a higher priority and will refine the APCFileExtensionsPolicy option. This option depends on the global APCMode.

- **APCFileExtensionsFull**

APCFileExtensionsFull <extensions>

This option specifies a list of extensions for the files that will be hash-checked or uploaded to APC.

Available values: A string containing extensions, separated by semicolons

Example: .xlsx;.pdf;.doc

Default value: None



Note This option has a higher priority and will refine the APCFileExtensionsPolicy option. This option depends on the global APCMode.

- **ScanTemp**

ScanTemp </path/to/temporary/folder>

Sets the temporary folder used for scan related operations. SAVAPI may also use other temporary folders for non-scan operations.

Available options: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: The default temporary folder of the Operating System.



Note It is recommended for the location to not be a folder that contains sensitive files, such as SAVAPI binaries or configuration files.

- **ArchiveScan**

ArchiveScan <value>

Activates archive detection and scanning.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0.

- **ArchiveMaxSize**

ArchiveMaxSize <size>

Sets the maximum allowed size for any file within an archive, mailbox or email.

Available values: File sizes up to INT64_MAX bytes.



Default value: 1073741824

The value "0" means the maximum allowed value (INT64_MAX bytes).

<size> can include K, M or G as a label. Otherwise the number is assumed to be in bytes.
Examples: "100M" or "32K". (1K = 1024 bytes. 1M = 1024^2 bytes. 1G = 1024^3 bytes.)

This setting is ignored if ArchiveScan, MailboxScan and MimeScan are all disabled.

- **ArchiveMaxRec**

ArchiveMaxRec <recursion-level>

Sets the maximum allowed recursion within an archive, mailbox or email.

Available values: 0 – 1000.

Default value: 200.

This option is limited to 1000 recursion levels.

A value of "0" means the maximum allowed value (1000 recursion levels).

This setting is ignored if ArchiveScan, MailboxScan and MimeScan are all disabled.

- **ArchiveMaxRatio**

ArchiveMaxRatio <expansion-factor>

Sets the maximum allowed decompressing-ratio within an archive, mailbox or email. The decompression will be aborted as soon as the limit is exceeded. E.g. For a 1MB compressed file and ArchiveMaxRatio 150, the decompression will be aborted when the uncompressed file exceeds 150MB.

Available values: 0 – INT32_MAX

Default value: 150.

A value of "0" means the maximum allowed value (INT32_MAX).

This setting is ignored if ArchiveScan, MailboxScan and MimeScan are all disabled.

- **ArchiveMaxCount**

ArchiveMaxCount <count>

Sets the maximum allowed number of files within an archive, mailbox or email.

Available values: 0 - INT64_MAX

Default value: 0

A value of "0" means the maximum allowed value (INT64_MAX).

This setting is ignored if ArchiveScan, MailboxScan and MimeScan are all disabled.

- **MailboxScan**

MailboxScan <0|1>

Activates detection and scanning of mailboxes.

Available values: 0 (disabled) or 1 (enabled)

Default value: 0

- **HeurMacro**

HeurMacro <0|1>

Activates heuristic macro detection.

Available options: 0 (disabled) or 1 (enabled)

Default value: 1



- **HeurLevel**

HeurLevel <0-x>

Sets the heuristic level for the engine.

Available values:

0 - disable heuristic detection.

1 - lazy heuristic detection. This is the lowest possible mode. Detection is not very good, but the false positives number will be low.

2 - normal heuristic detection. This is the recommended heuristic detection.

3 - high heuristic detection. This is the highest possible mode, but it will also increase the number of false positives.

Default value: 2

- **DetectAdspy**

DetectAdspy <0|1>

Activates detection for software that displays advertising pop-ups or sends userspecific data to third parties without the user's consent and might therefore be unwanted.

ADSPY denotes adware or spyware. This kind of malware is able to change browser settings, e.g. by manipulating registry settings or by using NTFS-streams. Very often IE-exploits are used to manipulate the browserhelp.dll.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1

- **DetectAdware**

DetectAdware <0|1>

Activates the detection of software that displays banner ads or pop-up windows through a bar that appears on a computer screen.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1

- **DetectAppl**

DetectAppl <0|1>

Activates detection for applications of uncertain origin or which might be hazardous to use.

Available values: 0 (disabled) or 1 (enabled).

Default value: 0

- **DetectBdc**

DetectBdc <0|1>

Activates detection for Backdoor-Client programs. Such programs are used to spy out or change data on a computer.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1

- **DetectDial**

DetectDial <0|1>

Activates detection for Dial-Up programs, that charge for a connection fee.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1



- **DetectGame**

DetectGame <0|1>

Activates game detection.

Available values: 0 (disabled) or 1 (enabled).

Default value: 0

- **DetectHiddenExt**

DetectHiddenExt <0|1>

Activates the detection of hidden file extensions. The concerning file contains an executable, which is disguised by a harmless file extension.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1

- **DetectJoke**

DetectJoke <0|1>

Activates the detection of joke programs that usually does not contain malicious code.

Available values: 0 (disabled) or 1 (enabled).

Default value: 0

- **DetectPfs**

DetectPfs <0|1>

Activates the detection of fraudulent software, also known as "scareware" or "rogueware" that pretends that your computer is infected by viruses or malware. The term "PFS" (Possible Fake Software) describes a software that usually requires a fee but has no functionality or installs other suspicious components.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1

- **DetectPhish**

DetectPhish <0|1>

Activates Phishing detection. Phishing is a technique that often tries to persuade the user to reveal personal information, such as usernames, passwords, and credit card information, by redirecting the user to fake websites where this personal information is requested.

Available values: 0 (disabled) or 1 (enabled).

Default value: 1

- **DetectSpr**

DetectSpr <0|1>

Activates the detection of programs that violate the private domain (Security Privacy Risk). This concerns software that may be able to compromise the security of your system, initiate unwanted program activities, damage your privacy or spy on your user behavior and could therefore be unwanted.

Available values: 0 (disabled) or 1 (enabled).

Default value: 0

- **DetectPua**

DetectPua <0|1>



Activates the detection of Potentially Unwanted Applications. These are hidden applications, unknowingly downloaded alongside legitimate apps which clutter your PC with ads, hijack your browser, slow down your PC – and frequently collect data on what you click on.

Available values: 0 (disabled) or 1 (enabled)

Default value: 1

- **DetectAllTypes**

`DetectAllTypes <0|1>`

Activates/ deactivates all detection types: DetectAdware, DetectAdspy, DetectAppl, DetectBdc, DetectDial, DetectGame, DetectHiddenExt, DetectJoke, DetectPfs, DetectPhish, DetectPua, DetectSpr.

Available values: 0 (all disabled) or 1 (all enabled).

Default value: There is no default value for this option.



Note If DetectAllTypes is active, it will override the settings for all the other Detect-options.

- **ScanTimeout**

`ScanTimeout <time-in-seconds>`

Sets the maximum number of seconds allowed to scan a file before aborting.

Available values: 0 - 86400 (1 second - 24 hours)

Default value: 0 (disabled).

- **Repair**

`Repair <0|1>`

Activates the repairing of infected files. The actual repairing occurs during the "SCAN" request.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0

- **SavapiNotifyRepair**

`SavapiNotifyRepair <0|1>`

Activates notification for a repairable infected file.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0

- **SavapiNotifyOffice**

`SavapiNotifyOffice <0|1>`

Activates the detection of Microsoft Office OLE documents.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0

- **SavapiNotifyOfficeMacro**

`SavapiNotifyOfficeMacro <0|1>`

Activates the detection of macros in Microsoft Office OLE documents.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0

- **SavapiNotifyOfficeMacroAutostart**



`SavapiNotifyOfficeMacroAutostart <0|1>`

Activates the detection of macros with auto-start enabled in Microsoft Office OLE documents.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0

- **SavapiNotifyAlertURL**

`SavapiNotifyAlertURL <0|1>`

Activates the notification with the Avira virus description URL.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0



Note This option does not display the alert URLs when the file is detected as malware by APC.

- **AlertURL**

`AlertURL <alert_url>`

Specifies the Avira virus description URL.

Available values: Virus description URL. The URL must contain the string "`?q=%1`" for validation purposes. The keyword '`%1`' passes the virus name.

Default value: `http://www.avira.com/en/threats?q=%1`

- **ScanMode**

`ScanMode <SMART | ALL | EXTLIST>`

Sets the scanning method.

Available options:

- SMART

The files scanned for malware are chosen by SAVAPI. The choice is made based on the file's content. This is the recommended setting.

- ALL

All files are scanned for malware, no matter their content or extension.

- EXTLIST

All files that match a predefined internal list of extensions are scanned for malware content. Files inside archives are not filtered by extension.



Important This predefined list of extensions EXTLIST cannot be altered or changed.

Default value: SMART

- **ReportEncryptedMime**

`ReportEncryptedMime <0|1>`

Activates reporting of encrypted mime emails.

Available options: 0 (disabled) or 1 (enabled).

Default value: 0



Note This setting is ignored if ArchiveScan, MailboxScan and MimeScan are all disabled.

- **MimeScan**



```
MimeScan <0|1>
```

Activates detection and scanning of emails.

Available values: 0 (disabled) or 1 (enabled)

Default value: 1

- **MalwareNamesDir**

```
MalwareNamesDir </path/where/to/create/the/file>
```

Sets the directory where SAVAPI should create a file containing the extracted malware names.

This option is ignored, if the option `MalwareNamesExtract` is disabled.

For more information about extracting malware names, see [2.6 Extracting malware names](#).

Available options: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: The SAVAPI scanning temporary path.

- **MalwareNamesExtract**

```
MalwareNamesExtract <0|1>
```

Enables/ disables the extraction of malware names from memory to disk. You can specify the directory path where the file will be created, by using the option `MalwareNamesDir`.

For more information about extracting malware names, see [2.6 Extracting malware names](#).

Available values: 0 (disabled) or 1 (enabled).

Default value: 0 (disabled).

Since the malware names file is needed by SAVAPI, it is recommended not to alter it. If the file is deleted, moved or its content is modified, the engine may be unable to find the names of the viruses.

If the file has been altered, when first scanning an infected file, the terminal answer is set to "319 INCOMPLETE VDF_READ_ERROR". Then, subsequent SCAN commands would directly return the error "350 Failed to read VDF file".

Example:

```
SCAN infected_object
[310 heur_virus_1] (heuristic is detected w/o the VDFs)
319 INCOMPLETE VDF_READ_ERROR
SCAN any_type_of_file
350 Failed to read VDF file
```

In order to fix the problem, you must restart the SAVAPI Service.

Logging-related options

- **LogFileName**

```
LogFileName </path/to/log/file>
```

Specifies the log file location and name. If this parameter is set, then file logging is activated.

Available options: Only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: None.

- **ReportLevel**

```
ReportLevel <0-3>
```



Specifies the log verbosity level.

Available values:

- 0 Log errors
- 1 Log errors and alerts
- 2 Log errors, alerts, warnings and info
- 3 Log errors, alerts, warnings, info and debug messages

Default value: 0



Note "alerts" record information about potential malicious code.

- **LogRotate**

`LogRotate <0|1>`

Available options: 0 (disabled) or 1 (enabled).

Default value: 0 (disabled) on UNIX; 1 (enabled) on Windows.

Only 10 log files will be created during log rotation. Older log files will be deleted after the limit is reached and new log files will be created. See `LogFileSize` for more details.

- **LogFileSize**

`LogFileSize <size>`

Sets the maximum allowed size for the log file.

Available values: File size up to `INT64_MAX` bytes.

Default value: 2M.

If log rotation is enabled (`LogRotate=1`), the log file is closed when the maximum size is reached; a new file is created and used for logging (i.e. the file "log" is renamed to *log.001*, and the logging will continue to the new *log* file). If rotation is disabled (`LogRotate=0`), no action is performed when maximum size is reached, meaning that the logging will stop.

Size can include K, M, or G as a label. Otherwise the number is assumed to be in bytes. Examples: "100M" or "32K". (1K = 1024 bytes. 1M = 1024² bytes. 1G = 1024³ bytes.)

- **LogTemplate**

`LogTemplate = ${DAY}/${MONTH}/${YEAR}`

`${HOUR}:${MINUTE}:${SECOND} ${HOST} ${PROGRAM} [${PID}] :`

`${SOURCE}: ${LEVEL}: ${MSG}`

Enables the templates for log messages.

Available templates are:

- SOURCE,
- LEVEL,
- YEAR,
- MONTH,
- DAY,
- HOUR,
- MINUTE,
- SECOND,
- WEEKDAY,
- TZOFFSET,



- TZ,
- HOST,
- FULLHOST,
- PROGRAM,
- PID,
- MSG,
- TID,
- WINDATE,
- WINTIME

Default value: "\${DAY}/\${MONTH}/\${YEAR} \${HOUR}:\${MINUTE}:\${SECOND} \${HOST} \${PROGRAM}[\${PID}]: \${SOURCE}: \${LEVEL}: \${MSG}".

- **SyslogFacility**

SyslogFacility <value>

Specified the facility for the UNIX default system logger.

Available values: Desired system logger facility.

Default value: "user".



Note This option is available only on UNIX systems.

- **DisableSystemLogger**

DisableSystemLogger <0|1>

Activates/ deactivates the default operating system logger.

Available options: 0 (system logger enabled) or 1 (system logger disabled).

Default value: 0 (system logger enabled).

3.3.3 Protocol

Once a TCP/IP connection is established, the SAVAPI TCP/IP protocol can be used. The protocol specifies exchange of commands and information using synchronous communication in the form of messages. The messages consist in text lines (ASCII PRINT, LOCALE, or UTF-8) followed by a new line.

Synchronous communication proceeds by sending a request and receiving a response. Responses include status codes to help classify the response. Some status codes imply that other responses are guaranteed to follow (thus informing the application that it must not yet send another request).

The general form of messages is as follows:

- **request:**

<command> <data>\n (for UNIX)

<command> <data>\r\n (for Windows)

- **response:**

<status-code> <data>\n (for UNIX)

<status-code> <data>\r\n (for Windows)

The SAVAPI Service initiates the communication by sending the following line:

```
100 SAVAPI:4.0
```

where '0' (in '4.0') is the minor version number of the SAVAPI protocol. The minor version number of the protocol is to be changed for extensions in the protocol.



Any 4.x version of the SAVAPI protocol must be 100% backwards compatible to a lesser 4.x version. If any incompatibilities should be introduced, the major version number ('4') of the protocol must be changed.



Note '4.0' is the protocol version number, not a version number for the SAVAPI Service binaries themselves.

The initial SAVAPI "greeting" uses <10> as its new line.

Response Status Codes

The following table contains a list of status codes that will be returned after various requests.

Not all status codes apply to all requests. (Non-terminal responses imply that more responses are guaranteed to follow.)

Code	Definition	Type
100	Information	Typically as a response to a request
199	"Pong" response with optional ping-text	Terminal
200	File was not an archive, no alert found	Terminal
210	File was an archive, no alert found	Terminal
220	A connection timeout occurred	Terminal
310	Alert found	Non-terminal
319	Scan finished, alert found	Terminal
350	Error occurred	Terminal
401	Low-level alert information	Non-terminal
404	Too many clients connected	Terminal
420	Repairable alert found (the alert itself will follow)	Non-terminal
421	Microsoft Office OLE document found	Non-terminal
422	Microsoft Office OLE document with macros found	Non-terminal
423	Microsoft Office OLE document with macros having auto-start enabled found	Non-terminal
430	Alert URL	Non-terminal
450	Plugin response	Non-terminal
499	Information	Non-terminal



Note Code " 300: *File was not an archive, alert found* " is obsolete.

The new engine can detect multiple malware in a regular file – non-archive file, without embedded objects (PDF, chm, etc.).

The only way to report multiple infections in a file is to list all non-terminal alerts and then to report the end of the scan. This is happening only if there is at least one malware detection (this means "alert"). If the file doesn't contain malware, the normal code 200 is returned.

Requests

The following sections describe the various requests that are allowed with the SAVAPI protocol. Any non-specified requests will result in an error response "350 <error-message>".

All request descriptions are followed by a list of possible status codes that can be used in response messages.



Note All requests apply to the current session, with the exception of RELOAD. For more details, see the RELOAD command description).

SAVAPI will consider all numeric values as decimal (base 10).

**SET (write only)**

"SET" requests are available to configure SAVAPI. Usually a "SET" request also has a "GET" request counterpart to retrieve current settings. However, the following commands do not have a "GET" counterpart and are therefore labelled as "write only".

- **SET PRODUCT <id>**

Set the key-id that is required by the application. SAVAPI will check if the key-id is within the license and that it is not expired. If it is available and is valid, the application is free to use SAVAPI. If not, this command and all *SCAN* requests will result in an error response, for example: "350 operation not allowed (license restriction)".

(100, 350)

GET (read only)

"GET" requests are available to retrieve current SAVAPI settings. The response to a "GET" request is in the form of:

```
100 <key>:<value>
```

This makes it possible to combine multiple "GET" requests into a single request:

```
GET <key1> <key2> <key3>
```

```
<key1>:<value1> <key2>:<value2> <key3>:<value3>
```



Note If values include ':' (colon) or ' ' (space) characters, the response could be ambiguous.

Usually a "GET" request also has a "SET" request counterpart to configure SAVAPI. However, the following commands do not have a "SET" counterpart and are therefore labelled as "read only".

- **GET SAVAPI**

Retrieves SAVAPI protocol version number.

(100, 350)

- **GET AVE**

Retrieves engine version number.

(100, 350)

- **GET VDF**

Retrieves VDF set version number.

(100, 350)

- **GET PID**

Retrieves the process-id for the SAVAPI process that is currently handling the connection.

(100, 350)

- **GET EXPIRE**

Retrieves the expiration date of the SAVAPI license. The date is in the form of YYYYMMDD.

(100, 350)

- **GET VDF_SIGCOUNT**

Retrieves the number of signatures in the VDF set.

(100, 350)

- **GET SELECTABLE_DETECT**

Retrieves the various types that can be detected (and dynamically turned on/off). The types are returned as a comma separated list.



(100, 350)

- **GET DESCR_DETECT_<type>**

Retrieves the English description for the given type. <type> is one of the types returned by "GET SELECTABLE_DETECT".

(100, 350)

- **GET MALWARE_NAMES_FILE**

Retrieves the file containing the extracted malware names.

For more information about extracting malware names, see [2.6 Extracting malware names](#).

(100, 350)

GET/SET (read/write)

"SET" requests are available to configure SAVAPI. For the following requests, a "GET" counterpart is also available and these are therefore labelled as "read/write". The "GET" response will return the same data that is provided with the "SET" request (the representation of the data may be different. For example, a "SET" request with "10K" will lead to a "GET" response with "10240".)



Note Almost all SET commands exist in the same format in the configuration file. Exception: SET CONF.

For more details, see [3.3.2 Configuration file options](#).

- **CWD <directory>**

Sets the current working directory for SAVAPI. Thus you don't need to specify full paths in file names anymore.

Default value: The SAVAPI Service binary location.

(100, 350)

- **CONF <configuration-file>**

Specifies the configuration file used. The configuration file will be (re-)read as part of this request. Global SAVAPI configuration parameters will be ignored and only workerspecific parameters will be read from the original configuration file. There is no default value.

(100, 350)

The following parameters will be ignored when a SET CONF command is executed:

User, Group, CreateSocketDir, ListenAddress, AttachToGuard, LogRotate, LoadPlugins, SocketPermissions, SyslogFacility, ModulesDir, AveDir, KeyFile, VdfDir, APCCertDir, PidDir, FopsLib, MalwareNamesExtract, MalwareNamesDir, LogTemplate, LogFileSize, ReportLevel, DisableSystemLogger, PoolScanners, PoolConnections, APCCacheSize, APCCacheDumpFile, APCCacheDumpFilePath, APCBlackoutRetries, APCBlackoutTimeout, DuplicateModules, Proxy, LogFileName.

- **ARCHIVE_SCAN <0|1>**

Activates archive detection and scanning.

Default value: 0

(100, 350)

- **ARCHIVE_MAX_SIZE <size>**

Sets the maximum allowed size for any file within an archive, mailbox and email. A value of "0" means the maximum allowed value. A size can include K, M or G as a label. Otherwise the number is assumed to be in bytes. Examples: "100M" or "32K". (1K = 1024 bytes. 1M = 1024^2 bytes. 1G = 1024^3 bytes.)



This setting is ignored, if ARCHIVE_SCAN, MAILBOX_SCAN and MIME_SCAN are all disabled.

Default value: 1073741824

(100, 350)

- **ARCHIVE_MAX_REC <recursion-level>**

Sets the maximum allowed recursion within an archive, mailbox and email. This option is limited to 1000 recursion levels. A value of "0" means the maximum allowed value (1000 recursion levels).

This setting is ignored, if ARCHIVE_SCAN, MAILBOX_SCAN and MIME_SCAN are all disabled.

Default value: 200

(100, 350)

- **ARCHIVE_MAX_RATIO <expansion-factor>**

A value of "0" means the maximum allowed value. This setting is ignored, if ARCHIVE_SCAN, MAILBOX_SCAN and MIME_SCAN are all disabled.

Default value: 150

(100, 350)

- **ARCHIVE_MAX_COUNT <count>**

Sets the maximum allowed number of files within an archive, mailbox and email. A value of "0" means the maximum allowed value. This setting is ignored, if ARCHIVE_SCAN, MAILBOX_SCAN and MIME_SCAN are all disabled.

Default value: 9223372036854775807

(100, 350)

- **MAILBOX_SCAN <0|1>**

Activates detection and scanning of mailboxes.

Default value: 0

(100, 350)

- **HEUR_MACRO <0|1>**

Activates heuristic macro detection.

Default value: 1

(100, 350)

- **HEUR_LEVEL <0-3>**

Sets the heuristic level for the engine.

Available values:

0 - disable heuristic detection.

1 - lazy heuristic detection. This is the lowest possible mode. Detection is not very good, but the false positives number will be low.

2 - normal heuristic detection. This is the recommended heuristic detection.

3 - high heuristic detection. This is the highest possible mode, but it will also increase the number of false positives.

Default value: 2

(100, 350)

- **DETECT_<type> <0|1>**



Activates detection of various other types. <type> is one of the types returned by "GET SELECTABLE_DETECT". An additional pseudo-type (ALLTYPES) is also available to generally represent all types. (ALLTYPES is only available for "SET" requests, and not available for "GET" requests.)

Default values:

- DETECT_ADSPY 1
- DETECT_ADWARE 1
- DETECT_APPL 0
- DETECT_BDC 1
- DETECT_DIAL 1
- DETECT_GAME 0
- DETECT_HIDDENEXT 1
- DETECT_JOKE 0
- DETECT_PHISH 1
- DETECT_PFS 1
- DETECT_PUA 1
- DETECT_SPR 0

(100, 350)

- **SCAN_TEMP <directory>**

Sets the temporary directory used for scanning files. SAVAPI may use other temporary directories for files that are not being scanned. These other directories can be specified with command-line arguments or in a configuration file.

Default value: The default temporary folder of the Operating System.

(100, 350)



Note It is recommended for the location to not be a directory that contains sensitive files, such as SAVAPI binaries or configuration files.

- **SCAN_TIMEOUT <time-in-seconds>**

Sets the maximum number of seconds allowed to scan a file before aborting.

Default value: 0 (disabled)

(100, 350)

- **TEXT_MODE <ASCII-PRINT|LOCALE|UTF-8>**

Specifies the character encoding used to read requests and post results. This is covered in details later in the section "TEXT SUPPORT".

Default value: LOCALE

(100, 350)

- **REPAIR <0|1>**

Activates repairing of infected files. The actual repairing occurs during the "SCAN" request.

Default value: 0 (disabled)

(100, 350)

- **SAVAPI_NOTIFY_REPAIR <0|1>**



Activates notification if an infected file can be repaired. This will activate the usage of 420 status codes.

Default value: 0 (disabled)

(100, 350)

- **SAVAPI_NOTIFY_OFFICE <0|1>**

Activates detection of Microsoft Office OLE documents. This will activate the usage of 421 status codes.

Default value: 0 (disabled)

(100, 350)

- **SAVAPI_NOTIFY_OFFICE_MACRO <0|1>**

Activates detection of macros within Microsoft Office OLE documents. This will activate the usage of 422 status codes.

Default value: 0 (disabled)

(100, 350)

- **SAVAPI_NOTIFY_OFFICE_MACRO_AUTOSTART <0|1>**

Activates detection of macros with auto-start enabled within Microsoft Office OLE documents. This will activate the usage of 423 status code.

Default value: 0 (disabled)

(100, 350)

- **SAVAPI_NOTIFY_ALERTURL <0|1>**

Displays alert URLs for detected alerts. This will activate the usage of 430 status codes.

Default value: 0 (disabled)

(100, 350)



Note This option does not display the alert URLs when the file is detected as malware by APC.

- **REPORT_ENCRYPTED_MIME <0|1>**

Activates reporting of encrypted mime emails. This setting is ignored if `ARCHIVE_SCAN`, `MAILBOX_SCAN` and `MIME_SCAN` are all disabled.

Default value: 0 (disabled)

(100, 350)

- **SCAN_MODE <SMART|ALL|EXTLIST>**

Sets the scanning method. Possible values are:

- `ALL` – All files are scanned for malware, no matter their content or extension.
- `EXTLIST` – Only files provided by the user, that match a list of specific extensions, are scanned for malware content. Files inside archives are not filtered by extension.
- `SMART` – The files scanned for malware are chosen by SAVAPI. The choice is made based on the file's content. This is the recommended (default) setting.

(100, 350)

- **MIME_SCAN <0|1>**

Activates or deactivates the detection and scanning of emails. By default `MIME_SCAN` is enabled. It is recommended to always keep this option enabled.



If `MAILBOX_SCAN` is enabled, emails contained in mailboxes are always scanned (no matter if `MIME_SCAN` is enabled or not).

Default value: 1 (enabled)

(100, 350)

- **ENCODE_FILENAMES <0|1>**

Activates the filename encoding feature in the SAVAPI protocol. When activated, the path and filename arguments of scanning commands must be encoded in their hexadecimal representation (Every byte of the path and filename must be represented by its hexadecimal value, with two hexadecimal digits. For example, *savapi.exe* in ASCII encoding: *7361766170692e657865*).

The paths and filenames in the answers received from SAVAPI will also be encoded in the same way.

Default value: 0 (disabled)

(100, 350)



Note This option should be used for sending paths and file names that contain control characters, or which begin or end with white-spaces through the socket. An alternative to `ENCODE_FILENAMES` is the `SCAN hex_enc: //` command. (see [The SCAN command](#) below).

- **APC_CONNECTION_TIMEOUT <time-in-seconds>**

Specifies the number of seconds SAVAPI will wait for establishing a connection to APC, before timing out. This value applies to each object scanned by APC (for example, a PE file in an archive). This value must be smaller than `APCScanTimeout`.

Default value: 20

(100, 350)



Note This option is available only if APC was initialized.

- **APC_SCAN_TIMEOUT <time-in-seconds>**

Specifies the number of seconds SAVAPI will wait for data transfer to/from APC before timing out. This value applies to each scanned object (for example, a file in an archive). This value must be greater than `APCConnectionTimeout` and smaller than `ScanTimeout`.

- **APC_CHECK_RISK_RATING_LEVEL <level>**

This option sets a minimum threshold for hash requests sent to Avira Protection Cloud regarding the risk rating of the file. The risk rating is based on a frequently updated mathematical model to evaluate the risks posed by a certain file.

A threshold of 0 allows hash requests to be sent even if the files have a very low risk rating, while a value of 7 allows hash requests only for files with the highest assumed risk.

Available values: 0 - 7 (0=very low risk, 2=low risk, 3=moderate risk, 4=high risk, 7=very high risk)

Default value: 4

(100, 350)



Note This option is available only if APC was initialized.

- **APC_UPLOAD_RISK_RATING_LEVEL <level>**

This option sets a minimum threshold for uploads into the Avira Protection Cloud regarding the risk rating of the file. The risk rating is based on a frequently updated mathematical model to evaluate the risks posed by a certain file.

A threshold of 0 uploads files even if they have a very low risk rating, while a value of 7 uploads only the files with the highest assumed risk.



Available values: 0 - 7 (0=very low risk, 2=low risk, 3=moderate risk, 4=high risk, 7=very high risk)

Default value: 4

(100, 350)



Note If APC_CHECK_RISK_RATING_LEVEL is greater than APC_UPLOAD_RISK_RATING_LEVEL, the file will be uploaded only if the malware probability is greater than or equal to APC_CHECK_RISK_RATING_LEVEL. This option is available only if APC was initialized.

- **APC_PE_MODE <DISABLED|CHECK-ONLY|FULL>**

This option specifies the mode in which APC will be used for scanning portable executable (PE) files.

Available values:

DISABLED – PE files will not be scanned with APC

CHECK-ONLY – only hashes of PE files will be checked with APC

FULL – full APC scanning functionality for PE files

Default value: full

(100, 350)



Note When APCMode is "check-only", PE files will not be uploaded even if APC_PE_MODE is "full". This option depends on the global APCMode.



Note The actual behavior of hash-checks and upload of files to APC also depends on your contractual relationship for APC services.

- **APC_ELF_MODE <DISABLED|CHECK-ONLY|FULL>**

This option specifies the mode in which APC will be used for scanning ELF files.

Available values:

DISABLED – ELF files will not be scanned with APC

CHECK-ONLY – only hashes of ELF files will be checked with APC

FULL – full APC scanning functionality for ELF files

Default value: full

(100, 350)



Note When APCMode is "check-only", ELF files will not be uploaded even if APC_PE_MODE is "full". This option depends on the global APCMode.



Note The actual behavior of hash-checks and upload of files to APC also depends on your contractual relationship for APC services.

- **APC_MACH_O_MODE <DISABLED|CHECK-ONLY|FULL>**

This option specifies the mode in which APC will be used for scanning Mach-O and Apple Universal Binary files.

Available values:

DISABLED – Mach-O files will not be scanned with APC


CHECK-ONLY – only hashes of Mach-O files will be checked with APC


FULL – full APC scanning functionality for Mach-O files



Default value: full

(100, 350)

 **Note** When APCMode is "check-only", Mach-O files will not be uploaded even if APC_PE_MODE is "full". This option depends on the global APCMode.

 **Note** The actual behavior of hash-checks and upload of files to APC also depends on your contractual relationship for APC services.

- **APC_FILE_EXTENSIONS_POLICY <AUTO|CUSTOM>**

This option specifies the policy for the files that will be scanned with APC.

AUTO – all file extensions supported by SAVAPI internal list will be scanned with APC

CUSTOM – user-defined list of extensions to be scanned with APC

Default value: custom

(100, 350)

- **APC_FILE_EXTENSIONS_DISABLED <extensions>**


This option specifies a list of extensions for the files which will not be scanned with APC.

Available values: A string containing extensions, separated by semicolons.

Example: .xls;.bin;.doc

Default value: None

(100, 350)

 **Note** This option has a higher priority and will refine the APC_FILE_EXTENSIONS_POLICY option.

- **APC_FILE_EXTENSIONS_CHECK_ONLY <extensions>**


This option specifies a list of extensions for the files that will be hash-scanned with APC.


Available values: A string containing extensions, separated by semicolons.

Example: .xls;.bin;.doc

Default value: None

(100, 350)

 **Note** This option has a higher priority and will refine the APC_FILE_EXTENSIONS_POLICY option. This option depends on the global APCMode.

 **Note** The actual behavior of hash-checks and upload of files to APC also depends on your contractual relationship for APC services.

- **APC_FILE_EXTENSIONS_FULL <extensions>**

Specifies a list of extensions for the files that will be hash-checked or uploaded to APC.


Available values: A string containing extensions (including the dot), separated by semicolons.


Example: .xls;.bin;.doc

Default value: None

(100, 350)



 **Note** This option has a higher priority and will refine the APC_FILE_EXTENSIONS_POLICY option. This option depends on the global APCMode.

 **Note** The actual behavior of hash-checks and upload of files to APC also depends on your contractual relationship for APC services.

- **FPC <0|1>**

Enables or disables the FPC module for the respective session.

Available values: 0 - 1 (0=disabled, 1=enabled)

Default value: 0

(100, 350)

- **FPC_TIMEOUT <time-in-seconds>**

Sets the FPC timeout to the requested value.

Available values: 0 - 86400 seconds (0=infinity)

Default value: 30

(100, 350)

The SCAN command

The most important request (and most complex) is the SCAN request. This request invokes the engine for a specified file.

SCAN command syntax:

- **regular scan**


SCAN <file path>


SAVAPI scans the file with the specified file path.

- **hex-encoded scan**

SAVAPI decodes the <hex_encoded_file_path> from hex to binary, and scans the resulted file without performing any conversion. The <hex_encoded_file_name> must contain only hex characters, upper case or lower case.

- The SCAN hex_enc command expects the decoded file path's binary sequence to be exactly as stored on disk for the specified file. As a result, SCAN hex_enc does not perform any conversions from TEXT_MODE and may be used when locales are not available or just to avoid conversions.

 **Note** The SCAN hex_enc command is not supported on Windows; If the command is used, SAVAPI returns the error message: " 350 Unsupported feature ". ENCODE_FILENAMES, whether enabled or disabled, does not influence scanning with SCAN hex_enc.

 **Note** The SCAN hex_enc command accepts only absolute file paths; If the specified file path is relative, SAVAPI returns an error message: " 350 Not an absolute path ".

- **apc hash scan**

SCAN apchash://apc_hash_of_file>[, <apc_hash_of_file>, ..., <apc_hash_of_file>]

SAVAPI can only scan APC hashes when APC is enabled. To compute the APC hash of a given file, please see [Computing the File Reputation extension hash using the apchash library](#).

For more information regarding APC hash scanning, please see [Scanning the File Reputation extension hash with SAVAPI](#).



SCAN command responses

Depending on the SAVAPI settings and the contents of the file, various responses can be returned.

- **200 <scan-keyword> [<scan-keyword> ...]**

This response means that the file does not contain any alerts. Various keywords can be returned to communicate additional information about the scan. Although no alert was found in the file, depending on the keywords, the file may still contain an alert (i.e.: if the `INCOMPLETE` keyword appears it means that only a part of the file was scanned and only that part of the file is clean. The rest of the file might contain malware). This is a terminal response, meaning that the application is free to make another request.

- **210 <scan-keyword> [<scan-keyword> ...]**

This response is identical in format to "200". It means that the file does not contain any alerts. However, it has the additional meaning that the file was detected as an archive and its contents were also scanned. Since an archive has many special attributes (multiple files, compression, recursion) there are various keywords that can be returned. Although no alert was found in the archive, depending on the keywords, the file may still contain an alert (i.e.: if the `HIT_MAX_REC` keyword appears, it means that only a part of the archive was scanned and only that part of the archive is clean. The rest of the archive might contain malware). This is a terminal response, meaning that the application is free to make another request.

- **220 timeout reached while waiting commands**

This message means that the connection was idle for too long and it was closed by the server. For more details regarding the connection timeout, see [Connection-related options](#).

- **310 [fileA-in-archive[--> fileB-in-fileA] <<<]alert-name ; type ; english-text-message**

This response means that the file contains an alert. However, it has the additional meaning that the file was detected as an archive and its contents were also scanned. Since archives contain files (and these files may also be archives), a type of archive recursion is present. This recursion is represented with an additional separator " --> " (space minus minus greater-than space). This separator is followed by the name of the file in the next level of recursion. (Note that the file name provided with the "SCAN" request is not included.) After all recursion has been represented, a second separator " <<< " is used (space less-than less-than less-than space). After this separator, the alert name, alert type and an English alert description are provided. These fields are separated by " ; " (space semi-colon space).

- The application is responsible for parsing this output. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request. (Note that the " <<< " separator may not appear, if the archive itself, rather than its contents, triggers the alert.)

The "310" response should not appear multiple times for a single "end file" within an archive, i.e. it must not be possible to generate more "310" responses than the number of files to scan.

- **319 <scan-keyword> [<scan-keyword> ...]**

This response means that the scanning of an alert-containing object is complete. This is a terminal response, meaning that the application is free to make another request. This response will come after one or more "310" responses. As with response "210", multiple keywords are returned to describe information about the scan.

- **350 <scan-keyword> [<scan-keyword> ...]**

This response is returned whenever the `SCAN` command cannot be performed, due to the I/O errors. For example:

- The product ID was not previously set, i.e. " 350 Operation not allowed (license restriction) "
- `SCAN` was called without an argument, i.e. " 350 no file given "
- The file does not exist, i.e. " 350 File open error "
- The `hex_enc` argument is a relative path, i.e. " 350 Not an absolute path "



- The argument cannot be converted in the given TEXT_MODE, i.e. " 350 Conversion error "
- The SCAN command is refused, because the engine is unable to find the name of the viruses, i.e. " 350 Failed to read VDF file "

This is a terminal response, meaning that the application is free to make another request.

- **420 [fileA-in-archive[--> fileB-in-fileA] <<<]alert-name ; type ; english-textmessage**

The infected file is repairable. The format of this response is identical to "310". This response will be followed (although not necessarily directly) by a "310" response with the same format. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request. Note: To repair the file, the repair mode must be activated and a scan performed again. Files that have been successfully repaired will no longer cause "310" or "420" responses.

- **421 [fileA-in-archive[--> fileB-in-fileA] <<<]office-type**

An Microsoft Office OLE document has been found. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

- **422 [fileA-in-archive[--> fileB-in-fileA] <<<]macro-name**

An Microsoft Office OLE document containing macros has been found. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

- **423 [fileA-in-archive[--> fileB-in-fileA] <<<]autostart-macro-name**

An Microsoft Office OLE document containing macros having auto-start enabled has been found. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

- **430 [fileA-in-archive[--> fileB-in-fileA] <<<]URL**

A URL that will lead to detailed information about an alert found. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.



Note Code " 300: File was not an archive, alert found " is obsolete.

Scan keywords

Here is a list of scan keywords that could appear with "200", "210" or "319" responses.



Note Please note that scan keywords may change between releases and the application must be prepared to handle new scan keywords.

Scan keywords	Description
ABORTED	Scan was aborted by signal (200, 210, 319)
APC_CONNECTION	Error connecting to cloud (200, 210, 319)
APC_TEMPORARILY_DISABLED	APC disabled due to many APC scan failures (200, 210, 319)
APC_DISABLED	APC permanently disabled (200, 210, 319)
APC_INCOMPLETE	Not completely scanned with APC (200, 210, 319)
APC_TIMEOUT	APC scan timed out (200, 210, 319)
APC_AUTHENTICATION	Error authenticating to cloud (200, 210, 319)
APC_QUOTA	APC quota limit reached (200, 210, 319)
ENCRYPTED	Contains encrypted contents (210, 319)
ENCRYPTED_MIME	Contains encrypted mime emails (210, 319)
HIT_MAX_COUNT	Maximum number of files reached (210, 319)



Scan keywords	Description
HIT_MAX_RATIO	Maximum extraction ratio reached (210, 319)
HIT_MAX_REC	Maximum recursion limit reached (210, 319)
HIT_MAX_SIZE	Maximum extraction size reached (210, 319)
INCOMPLETE	Not completely scanned (210, 319)
MEMORY_LIMIT	Internal memory limit reached (210)
OK	No problems detected (200, 210, 319)
PARTIAL	Is part of a multi-volume archive (210, 319)
PROC_ERROR	Processing archive (210, 319)
TIMEOUT	Scan timed out (200, 210, 319)
UNSUPPORTED	Archive format detected (210, 319)
VDF_READ_ERROR	The engine is unable to find the viruses name (319)

If the `ENCODE_FILENAMES` option is activated, the path and file name arguments of the `SCAN` command must be encoded in their hexadecimal representation. (Every byte of the path and file name must be represented by its hexadecimal value, with two hexadecimal digits. For example, *savapi.exe* in ASCII encoding: `7361766170692e657865`.)

The paths and file names in the answers received from SAVAPI will also be encoded in the same way.



Note `ENCODE_FILENAMES` should be used for sending paths and file names that contain control characters, or which begin or end with white-spaces through the socket. An alternative to `ENCODE_FILENAMES` is the `SCAN hex_enc://` command. (see [The SCAN command](#)).

Update Support

Internet updates will occur in a background process. Updates may require the SAVAPI Service to restart without informing the connected applications. Applications using SAVAPI must be able to handle unexpected closing of the TCP/ IP socket and try to reestablish a connection before interpreting the event as an error.

The *avupdate* component is a separate binary and it is no longer managed by SAVAPI. See [7. Updating Anti-malware SDK \(SAVAPI\)](#).

Other commands

The following commands are also available:

- **QUIT**

Gracefully disconnects from SAVAPI. This signals SAVAPI that the application is finished. Rather than providing a response, SAVAPI will close the connection. The SAVAPI Service will continue to run. (closed socket)

- **RESET**

Commands SAVAPI to return to the same configuration state as upon the initial connection, except for the case when it follows a `RELOAD` command. In this case, SAVAPI will return to the configuration state after `RELOAD`. (100, 350)



Note The configuration file will not be re-read.

- **RELOAD**

Commands SAVAPI to reload its original configuration file: the configuration file specified when the service is started (on non-Windows systems) or registered (on Windows systems). Global SAVAPI configuration parameters will be ignored and only worker-specific parameters will be read from the original configuration file. This will affect only the current session and the newly opened sessions. The existing sessions will not be affected by this command.

(100, 350)



Note The following parameters will be ignored when a RELOAD command is executed: User, Group, CreateSocketDir, ListenAddress, AttachToGuard, LogRotate, LoadPlugins, SocketPermissions, SyslogFacility, ModulesDir, AveDir, KeyFile, VdfDir, APCCertDir, PidDir, FopsLib, MalwareNamesExtract, MalwareNamesDir, LogTemplate, LogFileSize, ReportLevel, DisableSystemLogger, PoolScanners, PoolConnections, APCCacheSize, APCCacheDumpFile, APCCacheDumpFilePath, APCBlackoutRetries, APCBlackoutTimeout, vDuplicateModules, vProxy, LogFileName.

- **SHUTDOWN**

Commands SAVAPI Service to stop. SAVAPI will close the connection, without providing a response. This could cause unexpected socket-close events for other connected applications. (closed socket)

On Windows systems, because the SAVAPI Service is configured by default to restart as a failure action, the service will be restarted by the Service Control Manager after the SHUTDOWN command is received. The service can be stopped either with "SAVAPI -stop" command or "net stop SAVAPI" command.



Note This command is marked as obsolete in SAVAPI. It drops the connection and aborts any processing, but does not stop the service. Starting with the next SAVAPI release, the command will be removed and the service will reply with an error.

- **PING [<ping-text>]**

Test if the SAVAPI daemon is alive. The optional ping-text can be a maximum of 128 bytes in length. This same text will be returned in a 199 "pong" response. This command is also useful for synchronizing communication with SAVAPI. (199)

- **HELP [<command> [<command-arg>]]**

Provides available commands, arguments, and descriptions. The help output is generated using multiple "499" responses. (100, 350)

Other responses

- **499 informational-text**

A line of informational text. This response appears, for example, after the "HELP" request. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

Signals

SAVAPI also supports signals. These are commands that are sent asynchronously to SAVAPI as a trigger for certain actions. Signals look similar to requests but do not receive a response.

- **SCAN_ABORT**

SAVAPI aborts a scan and adds the "ABORTED" scan keyword to the scan response. This signal is ignored, if SAVAPI is not currently scanning a file.

Text support

The SAVAPI protocol provides three modes for handling text. The modes can be set using the "SET TEXT_MODE" request. Modes are applied to both incoming and outgoing data. The modes are described below.

If invalid text is received, the SAVAPI Service will immediately drop the connection.

Since some special characters are used within the SAVAPI protocol itself, some bytes of a file name must be converted to '?' (63) in responses. The bytes of a file name that must be converted are 10, 13, 60, 62. (Note that this does not affect SAVAPI's ability to scan and detect alerts. It is simply a cosmetic change in a response to allow a client to reliably parse the response.)

- **ASCII-PRINT**



Only printable characters of the ASCII set are considered valid characters. These include the characters 9 and 32-126. All other character values are converted to '?' (63). This is a conservative setting for clients that want to keep communication restricted to 7-bit printable text.

- **LOCALE**

All characters will be handled "as-is". It is the responsibility of the client to ensure that the SAVAPI Service will be able to correctly interpret the text (for example, with matching locale settings between client, service, and file system).

0-value characters will not be handled correctly in this mode. Encoding such as UTF-16LE or UTF-32LE are not supported by the SAVAPI TCP/IP protocol.

- **UTF-8**

The entire text must be correctly encoded as UTF-8. Invalid encoding will result in error responses.

3.4 Exit codes

The following table contains a list of exit codes, returned by SAVAPI Service when it completes its execution.

Exit code	Description
200	Program aborted, not enough memory available
201	Program aborted, invalid parameter
202	Program aborted, daemon not/already initialized
203	Program aborted, conversion error
204	Error parsing the command line arguments
205	Error parsing the configuration file
206	Invalid port specified (Obsolete, not used anymore)
207	Invalid IP specified (Obsolete, not used anymore)
208	Cannot start on specified interface (invalid IP/port)
209	Error on loading vdf files
210	Problem when trying to start the engine
211	Program aborted because the self check failed
212	No valid license found (Obsolete, not used anymore)
213	Error when trying to start/stop the SAVAPI Service
214	Program aborted because GET command failed
215	Program aborted because SET command failed
216	Error when trying to open/read/write a file
217	Error when trying to set uid/gid
218	SAVAPI Service timeout
219	SAVAPI Service not running
220	Failed to stop SAVAPI Service
223	Error when trying to execute report command
250	General daemon failure (specific information not available)



Note Obsolete exit codes:

- *206: Invalid port specified*
- *207: Invalid IP specified*
- *212: No valid license found*



3.5 Logging

3.5.1 Initialization

The logging in the SAVAPI Service is initialized when the process starts and, depending on the facility used, it can be divided in 3 types:

- **Console logging** – Cannot be configured or deactivated. It uses the STDOUT to report the results of a given command (e.g. `--version`, `--help`) and the STDERR to print the error reason (if any). For example: *“Validation failed for option 'config' with value 'inexistent_path'. Path does not exist.”*
- **File logging** – Fully configurable and disabled by default. It can be enabled through the command line parameter:
`--log-file=<log/file/path>`
or by using the configuration file parameter:
`LogFileName=<log/file/path>.`
- **System logging** (syslog daemon on UNIX, Event Logger on Windows) – Partially configurable and enabled by default. It can be enabled/disabled through the configuration file parameter `DisableSystemLogger=0/1`.

3.5.2 Configuration

File logging

The file logging is fully configurable, using the following options:

- **The log file location and name**

It can be set with the command line option:

```
--log-file=<file/path>
```

or the configuration file option:

```
LogFileName=<file/path>
```

Because the file logging is disabled by default, the user also enables the logging by setting the location of the file.

SAVAPI Service will try to open/create the file right after the privileges are dropped (i.e. the user&group are changed) and will return an error if the opening fails. If the given file is already present on the disk, SAVAPI Service will open it in appending mode, in order to write all messages at the end of the file. Else, if the file does not exist, it will be created by SAVAPI Service with read&write and read permissions for the configured user and configured group, respectively.

- **The logging level**

The default level of logging is 0 and it can be changed with the configuration file option `ReportLevel=<level>`. Messages on different log levels can be distinguished by the level-specific tag name (e.g. the level 0 has the `ERROR` tag assigned). The available log levels are the following:

- 0 – Only errors will be logged

This level includes messages sent when the execution of SAVAPI Service is aborted (i.e. fatal errors during startup or running time). Additionally, SAVAPI Service uses this level in order to log the file-scanning errors (e.g encrypted file, non-writable scan-temp folder). The associated tag is: `ERROR`.

- 1 – Only errors and alerts will be logged

Setting this level will determine SAVAPI Service to also log the malware found – when a file is detected as infected, a message containing the file path and the malware details will be logged. The associated tag is: `ALERT`.

- 2 – Errors, alerts and warnings



This level adds the warning messages that do not affect the process condition (state). They are mainly used to point the user to a possible issue or inconvenience: the used VDFs are old, the SAVAPI Library is signed with developer key, etc. The associated tag is named: `WARNING`.

- 3 – Errors, alerts, warnings, info and debug

This is the most verbose level and adds the informational messages, used to broadly describe SAVAPI Service's actions (command line and configuration file parsing, socket creation, socket message interpretation, etc) and the debug messages – more detailed information about the performed actions. Here are 2 tags: one for the info logs, named `INFO`, and the other for the debug messages, named `DEBUG`.



Note There are situations when SAVAPI Service is logging messages, regardless of the value of the `ReportLevel`.

Currently, 2 messages are logged directly on `WARNING` level, in order to warn the user about an accidental stop, performed by another instance:

- A running SAVAPI Service is stopped by another SAVAPI Service which starts: *"WARNING: Another running SAVAPI Service instance detected on the specified interface. The existing instance will be replaced by the new one. SAVAPI Service will stop."*

- The SAVAPI Service startup detects another running instance: *"WARNING: Another running SAVAPI Service instance detected on the specified interface. The existing instance will be replaced by the new one."*

- **The log template**

Besides the message itself, SAVAPI Service can also add the time when the log is written in the file, the host-machine where the process is running, the PID of the process, etc. This extra information is possible due to the configuration file option `LogTemplate=<user/log/template>`, which contains some predefined macros, automatically expanded by the SAVAPI Service's logger before writing the message in the file. The default log template is:

```
"${DAY}/${MONTH}/${YEAR} ${HOUR}:${MINUTE}:${SECOND} ${HOST} ${PROGRAM}
[${PID}]: ${SOURCE}: ${LEVEL}: ${MSG}"
```

The available log macros are the following (macro X must be written as `${X}` in order to be accepted):

- `SOURCE` – specifies the source of the message. Currently, there are 2 possible sources: "Supervisor" (the message is sent by the SAVAPI service's supervisor) and "Service" (the message is sent by the SAVAPI Service itself).

- `LEVEL` – will display the level of the message logged. The available levels are: "ERROR", "ALERT", "WARNING", "INFO" and "DEBUG".

- `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `WEEKDAY`, `TZ` (the time zone abbreviation: EEST, CET, etc) and `TZOFFSET` (the time zone offset related to UTC/GMT: +0300 for Romania, +0200 for Germany), – add the time-related information.

- `HOST` and `FULLHOST` – adds information about system's host name.

- `PROGRAM` – adds the name of the binary: "savapi" on UNIX and "savapi.exe" on Windows.

- `PID` – adds the process id.

- `MSG` – adds the message itself to the file.

- `WINDATE` and `WINTIME` – available only on Windows platforms; adds information about the local date and local time.

- **The log file size**

By default, the size of the log file is unlimited and this may be an inconvenience when using a verbose level (i.e. 3), because the file size can grow to a large size very quickly. Therefore, SAVAPI Service offers the possibility to set the maximum size of the log file through the configuration parameters: `LogFileSize=<max-size>` and `LogRotate=<0|1>`. By default, log rotation is disabled on UNIX and enabled on Windows.



If log rotation is activated (i.e. `LogRotate=1`), the log files will be rotated (i.e. file `"log"` is renamed to `"log.001"`, file `"log.001"` is renamed to `"log.002"` and so on) when the maximum size is reached. SAVAPI Service keeps a maximum of 10 rotated files – from `"log.001"` to `"log.010"`.

If rotation is not active (`LogRotate=0`) SAVAPI Service will continue to write in the given file, regardless of the configured size.



Note Control characters (range [1..31] in the ASCII table) will be replaced by the # character in the log files.

System logging

SAVAPI Service sends the same messages to the system logger, as for the file logging.

The system logging uses the **syslog** daemon on UNIX and the **Event Logger** on Windows.

The following configuration options are available:

- **Enabling/ Disabling**

By default, the system logging is enabled on both platforms (UNIX and Windows). The user can enable/ disable it, using the configuration file parameter:

```
DisableSystemLogger=0 | 1
```

- **The level of logging**

The logging levels (and default) for system logging are the same as for file logging and they can be configured in the same way. Before sending the message to the system logger, SAVAPI Service maps the log level with the corresponding level from the system logger:

- On UNIX (syslog daemon):

ERROR ? LOG_ERR, ALERT ? LOG_ALERT, WARNING ? LOG_WARNING, INFO ? LOG_INFO, DEBUG ? LOG_DEBUG (check syslog's documentation for more information about its logging levels: <https://en.wikipedia.org/wiki/Syslog>)

- On Windows (Event logger):

ERROR ? EVENTLOG_ERROR_TYPE, ALERT and WARNING ? EVENTLOG_WARNING_TYPE, INFO and DEBUG ? EVENTLOG_INFORMATION_TYPE (see Event Logger's documentation for more information about its logging levels).

- **The facility used** (only for UNIX)

By default, the syslog facility used is "user". This can be changed with the configuration file parameter `SyslogFacility=<facility>`. The available syslog facilities are the following:

- "mail",
- "auth" (security/authorization messages),
- "authpriv" (security/ authorization messages – private),
- "cron" (clock daemon),
- "daemon" (system daemons),
- "ftp" (ftp daemon),
- "kern" (kernel messages),
- "lpr" (line printer subsystem),
- "news" (network news subsystem),
- "syslog" (messages generated internally by syslog daemon),
- "user" (random user-level messages),
- "uucp" (UUCP subsystem),
- "local0" to "local7" (reserved for local use).



Recommendations

SAVAPI Service uses a synchronous logging mechanism, meaning that the thread (scanning instance) execution is interrupted until the message is written in the configured facilities. The interruption time depends on many factors: the file writing policy of the operating system (caching/ non-caching), the disk I/O performance, the number of threads/ processes writing to the same file, etc.

Therefore, SAVAPI Service may have serious performance penalties due to the heavy logging; Verbose log levels should be set only for debugging purposes.

The following logging configuration is recommended:

- Set a minimum log level: 0 or 1 (errors and alerts)
- Since SAVAPI Service also logs to the System Logger (which is a separate process meaning an asynchronous logging, i.e. less dead-time), you should enable the file logging only if it is really needed. The user can easily find the system logging messages.

Issues with the Anti-malware SDK (SAVAPI) Protocol when using a telnet client

If the telnet client is used to communicate with the SAVAPI Service and UNICODE characters are sent, it may happen that the connection fails. This is a telnet issue: On certain operating systems with telnet clients, the UNICODE characters returned by SAVAPI cannot be displayed and the connection appears to be hanging. When the socket traffic is verified with a debugging utility like `strace`, it can be clearly seen that SAVAPI writes its answers correctly on the socket, but the telnet client does not write them back to the user interface.

3.6 Non-disruptive service update

For critical applications, security has to be on the highest level without disturbing the operating processes. Since malware threats became very dynamic and are constantly changing, the malware signatures need to be updated frequently. The update process is a delicate operation that should by no means disrupt or alter the application's running services. For meeting these constraints, SAVAPI duplicates some of its internal modules, like the engine or the malware signature files, from its installation directory during its life span.

The process is described by the following picture.



In order to be backward compatible, SAVAPI needs to be started with the option [DuplicateModules](#) to enable this feature. The location of the modules management directory can be specified via command line or configuration file by using the `--modules-dir` switch or the `ModulesDir` configuration file option. In order to let optimization measures take effect, it is recommended to have the modules and the install directory on the same partition. After a non-disruptive service update, SAVAPI is signaled to reload its newly updated modules on the fly without perturbing the application's operations in any way. The transition is a lengthy process. Busy workers' are performing the switch as soon as their jobs are finished, implying that multiple versions of the same module may be active at a time. The older modules get purged from the specified modules' location when no longer used.



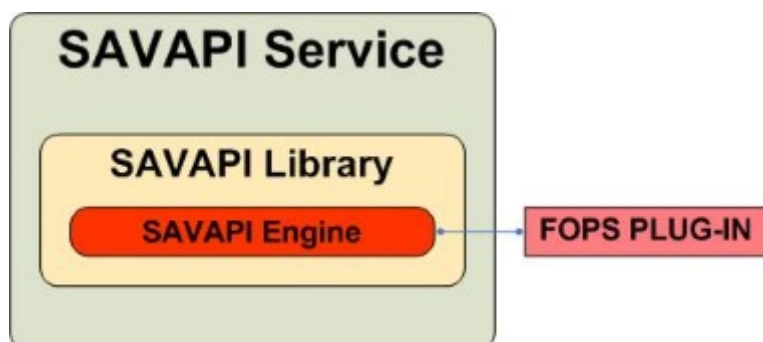
Note If you choose to supervise SAVAPI externally, you should start SAVAPI with both `"-N"` and `"--duplicate-modules"`, to avoid service interrupts after an update is performed.

3.7 Fops plug-in

SAVAPI Service users sometimes have to scan other objects than files, on disk, memory or other input streams, in a similar way with SAVAPI Library. This is possible by using the Fops plug-in which is in fact a library, loaded dynamically at runtime, containing the necessary functions for accessing those objects as specified in chapter [File Operation Structure \(FOPS\)](#).



The architecture of the system with the plug-in loaded is illustrated by the following graphic.



The parameters for loading and configuring the Fops plug-in (`--fops-lib` and `-- fops-lib-params`) are described in details in [Configuration](#). Once the service is started, all objects will be scanned by the functions of the plug-in.

3.8 Cloud component

As opposed to other SAVAPI Service components, the Avira Protection Cloud (APC) technology described in chapter [2.7 File reputation API support](#) is offered as an online service. Most of the computational effort takes place on Avira servers, decreasing the load on clients' side. In order to use APC scanning, a permanent network connection is required.

For more information about how to configure APC in SAVAPI Service, please consult [3.3.1 Command line parameters](#) and [3.3.2 Configuration file options](#).

3.9 OnAccess file scanning

Note Currently, the OnAccess functionality is implemented only in the Library version of SAVAPI. Please see chapter [2.8 Anti-malware SDK \(SAVAPI\) OnAccess](#).

4 Anti-malware SDK (SAVAPI) Client Library

4.1 General description of Anti-malware SDK (SAVAPI) Client Library

SAVAPI Client Library is a cross-platform library that allows the user to create network connections to the SAVAPI Service, configure the service options and scan files (found on disk or mapped in memory) or APC hashes.

Note The structure used for scanning objects from memory uses the type: "unsigned int" for the size of the object in memory. This limits the maximum size for a scanned object to approximately 4GB (2^{32}) on most of the 64-bit systems.

Note Even if it is enabled, APC will not be used when scanning objects from memory.

SAVAPI Client Library offers a cross-platform API, written in C language (the same as for the SAVAPI Library). The API allows the SAVAPI integrator to initialize the library, create an instance, register callback functions to monitor the processing progress, configure the instance settings, scan files, destroy the instance and uninitialize the library.

Basically, it does all the tasks related to network connection management:

- Initialize the network socket
- Configure and start the network socket
- Verify that the network socket I/O operations were performed with success



- Close the network socket and destroy all the related resources

You can implement the other parts of your application, without worrying about the network connections management.

4.2 Integration of Anti-malware SDK (SAVAPI) Client Library

The client-designed applications should (implicitly or explicitly) link to the SAVAPI Client Library and use the SAVAPI Client Library API to access the SAVAPI technology. The SAVAPI Client Library API allows you to configure the scan engine and other SAVAPI options, to process files and to retrieve information about the processing status.


In order to use the SAVAPI Client Library, the SAVAPI Service has to be installed and started.

A very simple example of how one can use the SAVAPI Library is offered in the SAVAPI SDK kit. (You can refer to the SDK documentation under *doc/README*.)

4.3 Configuration of Anti-malware SDK (SAVAPI) Client Library

The SAVAPI Client Library can be configured through the exposed API. In order to set an option, you call the `SAVAPI_Set()` function, with the appropriate parameters.

The SAVAPI Client Library API is the same as the SAVAPI Library API. For more details, see the documentation of SAVAPI Library API.

 **Note** The timeouts should be set according to the running system's load and performance. If the system's resources are low, exceeding timeouts can break the communication between SAVAPI Client Library and the SAVAPI Service. If the timeouts are infinite or too long, the client connection may block while waiting for a server response.

4.4 Logging in Anti-malware SDK (SAVAPI) Client Library

The SAVAPI Client Library's logging mechanism is the same as the one in SAVAPI Library. See [2.4 Logging](#).

4.5 Extracting malware names


From the client library, you can only retrieve the path of the file containing the extracted malware names, using the `SAVAPI_Get()` function, with the option `SAVAPI_OPTION_MALWARE_NAMES_FILE`.

5 Installation


5.1 Installation on Windows

You must have a valid Avira license in order to install and use the product.

5.1.1 Installing the OnAccess driver

 **Note** Only needed for the SAVAPI Library. Currently, the SAVAPI Service does not offer the OnAccess functionality.

 **Note** Installing the OnAccess virtual driver is not mandatory for on-demand scanning.

 **Note** Dependencies for SAVAPI's OnAccess scanning module can be found in chapter [2.8.1 Dependencies](#).

Installing the OnAccess virtual driver means configuring the Windows operating system to load the OnAccess virtual driver during system startup.



Note The OnAccess driver will not be loaded by the operating system when starting in safe mode.

In order to use SAVAPI's OnAccess file scanning module, the OnAccess virtual driver must be installed. To install it, from an elevated command prompt, execute `ams_setup.exe install`. The installation tool will copy the OnAccess virtual driver files to the Windows system driver directory and will configure the operating system to load the drivers from there. Installing the OnAccess driver will make the kernel immediately load it. The driver, as well as install/uninstall command-line scripts are available in the SAVAPI package.

Without the drivers, OnAccess runtime libraries or a valid OnAccess-enabled license, SAVAPI will disable the OnAccess module. SAVAPI on-demand file scanning will not be affected by the missing SAVAPI OnAccess module components or by disabling the OnAccess module altogether.

The SAVAPI OnAccess runtime libraries, as defined in section [2.8.1 Dependencies](#), must be present in the same location as the binary loading the SAVAPI library.

To uninstall, from an elevated command prompt, execute `ams_setup.exe uninstall`.



Note Please note that the system must be restarted for the virtual driver to actually be unloaded from the running kernel.

5.1.2 Installing the Anti-malware SDK (SAVAPI) Service

The product does not require further installation. The SAVAPI Service can be directly used from the folder the SAVAPI package has been extracted to.

The SAVAPI Service must first be registered in the Windows Service Control Manager. This is done by adding the command line option `--install` to the command line used with the service.

In order to start on a Windows operating system, the SAVAPI Service must retrieve at least the interface it is supposed to start with, from the user. This can be done in two ways:

- Using the command line option `--tcp`.

For example: `savapi --tcp=9999 --install`

- Using the configuration file option `ListenAddress` and passing the configuration file to the service, using the command line option `--config`.

For example: `savapi --config=c:/path/to/my_config_file --install` where the file `my_config_file` should contain at least the following line: `ListenAddress=inet:9999`

For more details about the command line options and the configuration file options, please refer to the following chapters:

- [3.3.1 Command line parameters](#)
- [3.3.2 Configuration file options](#)

5.2 Installation on UNIX

The product does not require installation. Just unpack the corresponding archive in a directory and you can start using SAVAPI.

Please copy the license key into the SAVAPI installation directory. If the use of APC is intended, it is mandatory to execute the `apc_random_id_generator` binary as an user with write rights on the binary folder. A new file name `apc_random_id` will be created, needed by the APC licensing.

6 Licensing

Your company will receive one dedicated `.key` file, which can be delivered to all your customers together with your own application's key. Please note that you always have to deliver the `.key` file, otherwise SAVAPI will not scan.



This key is unique for you as a company. It relates (among other things) to the expiration date and the Product ID that you will receive from Avira (in the same time with the new key).

Warning You are responsible for integrating, distributing or exchanging the key in your application, which integrates SAVAPI (Library or Service). Your application must send the product ID to SAVAPI, which uses it to initialize the engine. If the product ID and the key do not match, then the product does not scan.

The key you will receive from Avira represents a license which is valid for an agreed amount of time. A license can be renewed or blacklisted whenever necessary. Practically, it is up to you if your customers will even see that they have two licenses. For them, the SAVAPI license key can be just another file included in the package.

Of course, your product must make sure that the key does not expire, otherwise it will not scan anymore.

When the key expires, the product will scan using its existing signatures. If you update the engine and the expiration date in the key is older than the date in the engine, SAVAPI will not scan.

APC needs a unique random 40-characters string read from the `apc_random_id` file from the install folder. The file must be generated once, during installation, using the `apc_random_id_generator` utility and must be preserved for the entire installation lifespan. If, for various reasons, the `apc_random_id` file cannot be accessed or its content is invalid, as fall-back a new random ID is calculated based on the MAC address. If this operation fails also, SAVAPI will exit with an error.

If you use Avira Updater (`avupdate.exe/.bin`), you can enable the license verification with the `--check-product` parameter. See [7. Updating Anti-malware SDK \(SAVAPI\)](#).

7 Updating Anti-malware SDK (SAVAPI)

As a protection technology, SAVAPI needs regular updates, to keep its detection patterns up to date and to fix possible issues. The SAVAPI product comes with an integrated updater module (the Avira Updater), which can be used to update SAVAPI on all supported platforms, having the same functionality on all supported operating systems.

Through command line parameters or configuration files for the Updater, the following operations can be carried out:

- Check if new updates are available, for any of SAVAPI's components (binary, library files, engine, signatures).
- Update any of the SAVAPI components from Avira's update servers or from user defined servers, with the proper update structure.
- Mirror the already configured update servers.

The Updater mirrors the modular structure of SAVAPI, meaning that SAVAPI can be updated entirely or by modules:

- Product update - Updating the binaries, libraries, engine and signatures depending on the SAVAPI mode used in the implementation: Service or Library.
- Engine update - Updating the detection files (`ae*.*` and `xbv*.vdf`).

For each SAVAPI update mode (product, engine) a set of modules is selected and sent to the Updater as parameters, either in command line or in a configuration file. The platform package already includes configuration files for the main update scenarios. (see table on next page, please). The table below presents the correspondence between SAVAPI runmodes and updating configuration files.

Configuration file	SAVAPI runmode	Updated modules
<code>avupdate-savapi-engine.conf</code>	daemon/ service	engine, signatures
<code>avupdate-savapi-product.conf</code>	daemon/ service	all modules
<code>avupdate-savapilib-engine.conf</code>	library	engine, signatures



Configuration file	SAVAPI runmode	Updated modules
<i>avupdate-savapilib-product.conf</i>	library	all modules

The binary name of the Updater is *avupdate.bin* (UNIX) and *avupdate.exe* (Windows).

The Updater's help message can be obtained with the command:

```
<binary_name> --help
```

The Updater can be called with a certain configuration, using the following command line:

```
<binary_name> -C <avupdate-savapi....conf>
```

At the end of each update cycle, the status of the update will be displayed in the console. If you need to see the progress status, use the `--show-progress` option when calling the updater binary.

The messages (error, warning, etc.) displayed by the Updater when executed are set in the binary file *avupdate_msg.avr*. This file has to exist in the same folder as the *avupdate* binary file, being mandatory for starting the updater binary.

7.1 Mirroring the Updater's server structure

The updater utility offers the capability to create local mirrors of the existing updater server structure. Mirroring a server will create a fully functional structure that can be used to update the product from. During this operation, the updater copies all needed files from the server, based on the provided configuration parameters of the file.

Example for Windows:

```
avupdate.exe --mirror --config=<config file> --install-dir=<path>
```

Example for UNIX:

```
./avupdate --mirror --config=<config file> --install-dir=<path>
```

When using the `--mirror` option, specifying the `--config` and `--install-dir` parameters is mandatory.

The `<path>` provided as `--install-dir` value is the destination of the mirroring process. The directory has to exist and it has to grant writing privileges to the user running the mirroring process.

7.2 Anti-malware SDK (SAVAPI) update structure and modules

7.2.1 Updater related files

The following table contains a minimal description about the Updater related files from the SAVAPI package.

Name	Description
<i>avupdate.bin</i> (UNIX) <i>avupdate.exe</i> (Windows)	The Updater binary itself, performs all actions needed for an update.
<i>avupdate_msg.avr</i>	The Updater resource file in binary format, that contains the compiled message used for logging. It is used and needed by the Updater binary.
<i>avupdate-savapi-product.conf</i> <i>avupdate-savapi-engine.conf</i>	The SAVAPI Server update configuration files for product (all modules) and engine (AVE2 and VDF modules).
<i>avupdate-savapilib-product.conf</i> <i>avupdate-savapilib-engine.conf</i>	The SAVAPI Library update configuration files for product (all modules) and engine (AVE2 and VDF modules).
<i>update.sh</i> (UNIX) <i>update.bat</i> (Windows)	The script file containing the basic Updater commands needed to perform updates for SAVAPI Server and Library.
<i>savapi_stub</i> (UNIX) <i>savapi_stub.exe</i> (Windows)	The executable file used during the update process in order to execute the PRETEST, POST actions. It will be referenced as the STUB executable.
<i>savapi_pretest.sh</i> (UNIX) <i>savapi_pretest.bat</i> (Windows)	The script file used during the update process in order to execute the PRETEST action. It will be referenced as PRETEST script.



Name	Description
<i>savapi_pre.sh</i> (UNIX) <i>savapi_pre.bat</i> (Windows)	The script file used during the update process in order to execute the PRE action. It will be referenced as the PRE script.
<i>savapi_post.sh</i> (UNIX) <i>savapi_post.bat</i> (Windows)	The script file used during the update process in order to execute the POST action. It will be referenced as the POST script.

7.2.2 Introduction to Anti-malware SDK (SAVAPI) update

The Updater binary can be used to check, download and/or mirror the updates of SAVAPI's components (Server, Library, Client Library, engine, VDF).

In order to check if there are updates available for a specific SAVAPI update configuration the option "check-if-update-available", along with the "skip-master-file" option, should be specified in the command line: `</path/to/Updater/binary> --config=<path/to/SAVAPI/update/config/ file> --check-if-update-available --skip-master-file`

In order to perform the actual update for a specific SAVAPI update configuration the following command line may be used ("show-progress" option can be appended in order to retrieve real-time download progress): `</path/to/Updater/binary> --config=<path/to/SAVAPI/update/config/ file>`

To mirror the entire update structure of a specific SAVAPI update configuration the "mirror" option should be used in the command line: `</path/to/Updater/binary> --config=<path/to/SAVAPI/update/config/ file> --mirror --install-dir=<path>`

7.2.3 Modules of the Anti-malware SDK (SAVAPI) update

The update of SAVAPI can be done for all its components (i.e. the whole product update, which can be done using the *avupdate-savapi-product.conf* configuration) or only for specific modules by using the "update-modules-list" option (e.g. the *avupdatesavapi-engine.conf* configuration updates only the engine binaries and the virus definition files).

The most important modules of SAVAPI are the following:

- **SELFUPDATE:** Contains the Updater binary's files (the binary itself and the resource file). This module is updated first in order to be able to re-trigger the whole update process if there are new Updater files. The SELFUPDATE module can be skipped by using the option "skip-selfupdate".
- **SAVAPI:** Contains the SAVAPI binary and SAVAPI Library files. If a file from this module is updated, the running SAVAPI processes need to be restarted.
- **VDF:** Contains the virus definition files. If a file from this module is updated, the running SAVAPI processes need to be reloaded. If SAVAPI is started with duplicate-modules option enabled, the update can take place without stopping. For more information on this scenario, see SAVAPI option "--reload-engine".
- **AVE2:** Contains the engine binary files. If a file from this module is updated, the running SAVAPI processes need to be reloaded. If SAVAPI is started with duplicate-modules option enabled, the update can take place without stopping. For more information on this scenario, see SAVAPI option "--reload-engine".

7.2.4 Anti-malware SDK (SAVAPI) update script details

During the update process of SAVAPI components, various scripts are used in order to perform specific actions, in the following sequence: test if SAVAPI can start with the new files (PRETEST), detect the running SAVAPI processes (PRE), reload/restart the running SAVAPI processes (POST).

The scripts are triggered by the Updater binary through a proxy executable, *savapi_stub.exe*, which is called for every update action: PRETEST, PRE and POST.

Then, depending on the action, the STUB will choose a specific script that will actually run the given action:



- `<path/to/pretest/install/dir>/savapi_pretest.[sh|bat]` `<path/to/pretest/install/dir>` called for the PRETEST action in order to test if the new files can be used by SAVAPI.

The script is copied from the installation directory (see the "install-dir" option of Updater binary) to the PRETEST directory and it is run from the new location with one single argument - path to the PRETEST directory. The script is triggered for the files belonging to the modules: SAVAPI, VDF and AVE2.

- `<path/to/install/dir>/savapi_pre.[sh|bat]` [`<parent/process/ identifier>`] `<path/to/install/dir>` `<post_action>` called for the PRE action and it is used to inform the POST script about the action (reload or restart) to be applied to the running SAVAPI processes.

The script is run from the installation directory and contains as arguments the process identifier of the Updater binary (only on UNIX), the path to the installation directory and the post-action (i.e. one of "reload" or "restart" string). The script is triggered for the files belonging to the modules: SAVAPI - with "restart" action, VDF, AVE2 - with "reload" action.

- `<path/to/install/dir>/savapi_post.[sh|bat]` [`<parent/process/ identifier>`] `<path/to/install/dir>` called for the POST action in order to restart or reload the running SAVAPI processes.

The script will automatically detect all SAVAPI processes which are started from the provided installation directory and it will reload/restart as a SAVAPI daemon using the command line parameters that they were started with. The script is run from the installation directory and contains as arguments the process identifier of the Updater binary (only under UNIX) and the path to the installation directory. The script is triggered for the files belonging to the modules: SAVAPI, VDF or AVE2.

The STUB executable is overwritten each time an update operation is performed, residing in the update-server structure.

7.2.5 How to use Anti-malware SDK (SAVAPI) update scripts

By default, the SAVAPI update scripts requires that all modules (SAVAPI binary, SAVAPI Library, engine binaries and virus definition files) are placed in the installation directory. In some exotic cases configuration requires (e.g. when the engine binaries and virus definition files are placed in a distinct directory, other than the SAVAPI binary and SAVAPI Library) - changes need to be performed to the PRETEST, PRE and POST scripts accordingly.



Note All the described scripts can be changed by the user with exception of the STUB executable which is located in SAVAPI's server side update structure, meaning that it will be overwritten every time when the online version does not correspond with the local one.

To demonstrate, how the SAVAPI update scripts can be used to meet various setup scenarios, one can consider the case when the Updater binary, SAVAPI binary and SAVAPI Library files (i.e. the files from the SELFUPDATE and SAVAPI modules) are placed in a directory called DIR_SAVAPI and the engine binary and the virus definition files (i.e. the files from the AVE2 and VDF modules) are located in a distinct directory called DIR_ENGINE. In order to update the engine files (AVE2 and VDF modules), the following must be done:

- **copy** (direct copy or hard link) the PRETEST, PRE and POST scripts in the DIR_ENGINE directory.
- **modify** the copied PRETEST and the POST scripts in order to set the right path (i.e. the path to DIR_SAVAPI) to SAVAPI files. For simplicity, modify the value of the SAVAPI_DIR_PATH variable with DIR_SAVAPI.
- **run** the Updater binary by setting the installation directory to the DIR_ENGINE and by adding the `update-modules-list=AVE2, VDF` parameter.

In the described scenario, to update the SAVAPI files (SAVAPI module), the following must be done:

- **copy** (direct copy or hard link) the PRETEST, PRE and POST scripts in the DIR_SAVAPI directory.
- **modify** the copied PRETEST script in order to set the right path (i.e. the path to DIR_ENGINE) to engine files. For simplicity, modify the value of the ENGINE_DIR_PATH variable with DIR_ENGINE.



- run the Updater binary by setting the installation directory to the DIR_SAVAPI and by adding the `update-modules-list=SAVAPI` parameter.

In general, in order to be able to update SAVAPI components that are spread over multiple directories when SAVAPI is started with DIR_ENGINE directory, the SAVAPI update scripts (without the STUB executable) should be placed in the directory used for installation of those specific module files (i.e. the `install-dir` option of Updater binary) and they should be modified in order to point at valid paths used in the performed actions (PRETEST, PRE and POST).

7.2.6 Limitations

The files from the AVE2 and VDF modules must be located in the same directory.

7.3 Avira Updater's configuration parameters

Before downloading and updating product files, Avira Updater reads the configuration from *avupdate.conf* from its default location, or from any other configuration file specified when calling the binary.



Note The commands in command line have a higher priority and override the options in the configuration file.

The Updater supports transfer protocols like: HTTP, HTTPS (with or without proxies) and SMB.

With a few exceptions, the configuration parameters listed in this section can be used both in command line and in configuration files. (In configuration files, the parameters are called without the preceding dashes "--".)

Example of parameter in *avupdate.conf*: `temp-dir=/tmp`

Example of parameter in command line: `avupdate --temp-dir=/tmp`

The general syntax for configuring the Updater in command line is:

`./avupdate.bin [options] – for UNIX`

`avupdate.exe [options] – for Windows`

The parameter names are cross-platform and **not** case-sensitive. The parameter values have to be compliant with the running platform specifications.

7.3.1 General parameters

- **--help**

Displays the help message, about the Updater options.



Note This option is available only in command line.

- **--version**

Displays Updater's version.



Note This option is available only in command line.

- **--config**

Contains the path to the configuration file. Example:

`avupdate.bin --config=avupdate.conf`



Note This option is available only in command line.

The default value: `avupdate.conf`

- **--no-config**



The Updater will not read any configuration file. All parameters are given in command line.



Note This option is available only in command line.

- **--quiet**

The Updater will not log messages on screen.



Note This option is available only in command line.

The default value: `false`

- **--show-progress**

Shows the download progress.

The default value: `false`

Directories and files

- **--temp-dir**

Temporary directory for downloading update files. Example:

```
avupdate.bin --temp-dir=./tmp
```

The default value: `<install directory>/avupdate_tmp_XXXXXX`

- **--backup-dir**

Backup directory for the existing files, before updating.

The default value: `<install directory>/avupdate_backup`

- **--install-dir**

Specifies the installation directory for updated product files.

```
avupdate.bin -- install-dir=.
```

- **--update-dir**

Specifies the location of engine, vdf and *build.dat* in order to get the needed information for the user-agent string. If this parameter is not given, the Updater will use the path from the `--install-dir` option.

- **--cache-dir**

Specifies the cache directory for internal usage.

The default value: `<install directory>/idx`

- **--key-dir**

Specifies the directory in which the Updater should search for a valid key.

- **--master-file**

Specifies the *master.idx* file. Example:

```
avupdate.bin --master-file=/idx/master.idx
```

- **--local-master-file**

Specifies the full path to a local master file to be used instead of the one from the installation directory.

- **--product-file**

To specify the product file. Example:



```
avupdate.bin --product-file=/idx/savapi4-win64-en.info.gz
```

The product-file parameter can be used both in command line and in config files. It defines the platform where SAVAPI is running. Using this information, the Updater determines which files to check for updates and to download, if necessary.

The currently possible values for this parameter are:

- [savapi4|savapi4lib]-linux32-en.info.gz
- [savapi4|savapi4lib]-linux_arm64-en.info.gz
- [savapi4|savapi4lib]-win32-en.info.gz
- [savapi4|savapi4lib]-win64-en.info.gz

This list of values may change, when new supported platforms are added.

- **--add-var-pair**

Used to define pairs of variables and values in the INFO file. Example:

```
DESTINATION=%MY_INSTALL_DIR%
```

In command line:

```
avupdate.bin --add-var-pair=MY_INSTALL_DIR=/usr/lib/<yourproduct>
```

- **--module-install-path**

Indicates a specific installation path for a module.

The default value: The path from the install-dir option. Example:

```
./avupdate.bin --update-modules-list=MODULE1,MODULE2
--module-install-path="MODULE1=<install_directory>/MODULE1_DIR"
--module-install-path="MODULE2=<install_directory>/MODULE2_DIR"
```

- **--cert-path**

The path to one or multiple certificate files to be used for authenticating the https update servers which are specified by either the internet-srvs or the peak-handling-srvs options. The certificates must be in PEM format.

This option depends on the value of --cert-verify-policy:

- If --cert-verify-policy is auto or ca-pinning, --cert-path will contain a list of CA bundle paths, separated by ';'

```
./avupdate.bin --cert-path="valid_ca_bundle_A.crt" --cert-verify-policy="auto"
```

```
./avupdate.bin --cert-path="valid_ca_bundle_A.crt;valid_ca_bundle_B.crt" --cert-verify-policy="ca-pinning"
```

- If --cert-verify-policy is pubkey-pinning, --cert-path will contain a list of certificate paths or SHA256 hashes in base64 of the certificates' public keys separated by ';'

```
./avupdate.bin --cert-path="valid_certificate_A.crt" --cert-verify-policy="pubkey-pinning"
./avupdate.bin --cert-path="valid_certificate_A.crt;valid_certificate_B.crt" --cert-verify-policy="pubkey-pinning"
./avupdate.bin --cert-path="sha256//63quvhJLXsIUY0+rfroz7vw2RrFYE6Sr+zchZGxbzU4=;valid_certificate_B.crt"--cert-verify-policy="pubkey-pinning"
```

The default value: 'none' (which means that https update servers are not authenticated by default)

- **--crl-path**



The path to a Certificate Revocation List file to be used for certificate validation when authenticating the https update servers which are specified by either the `internet-srvs` or the `peak-handling-srvs` options.

The default value: `none`



Note This option is only considered if the `cert-path` option is also specified.

--cert-verify-policy

The policy by which the server certificate is verified. It can have the following values:

- `auto`: system-wide certificate store + certificate bundles specified in `--cert-path`
- `ca-pinning`: certificate bundles only, specified in `--cert-path`
- `pubkey-pinning`: certificate paths or the SHA256 hashes in base64 of the certificates' public keys, specified in `--cert-path`

The default value: `auto`



Note This option is ignored when `--cert-path` is not specified.

Steps to obtain the SHA256 of Avira public key:

1. Download the certificate from the Avira servers.

```
echo -n | openssl s_client -connect oem.avira-update.com:443 | sed -ne
'/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > oem.avira-update.com.pem
```

2. Extract the public key from the certificate.

```
openssl x509 -in oem.avira-update.com.pem -pubkey -noout > oem.avira-
update.com.pem.pubkey
```

3. Convert the public key to DER format.

```
openssl asn1parse -noout -inform pem -in oem.aviraupdate.com.pem.pubkey
-out oem.avira-update.com.pem.pubkey.der
```

4. Hash the key with SHA256 algorithm and convert the result to base64.

```
openssl dgst -sha256 -binary oem.avira-update.com.pem.pubkey.der |
openssl base64 63quvhJLXsIUy0+rfroZ7vw2RrFYE6Sr+zchZGxbzU4=
```

- **--no-host-check**

Specifies that the host names of https update servers should not be verified against the servers' certificate names.

The default value: `false` (the host names are checked)

- **--no-cert-verify**

Disables the https certificate validation.

This option disables the verification for the https update server certificate expiration date and also the checks for the server's authenticity (in case `--cert-path` and `--cert-verify-policy` options are used).

The default value: `false` (the update server certificate is validated)

- **--internet-srvs**

The list of Internet update servers.

```
./avupdate.bin --internet-srvs=https://oem.avira-update.com/update
```

An ftp or an http server can also be used for updates. For example:

```
./avupdate.bin --internet-srvs=ftp://server
```

or

```
./avupdate.bin --internet-srvs=http://server
```



- **--peak-handling-srvs**

Contains the servers used (ftp, http or https) in case no Internet server is available or if both `--ipv4-peak-server-limit` and `--ipv6-peak-server-limit` are reached (see below).

- **--ipv4-peak-server-limit**

If both `--ipv4-peak-server-limit` and `--ipv6-peak-server-limit` are reached, the list of servers from `--peak-handling-srvs` will be used for updates.

If this limit is set to 0, the Updater will try to update from all IPv4 servers (`--internetsrvs`) before trying to update from the `--peak-handling-srvs` list.

The default value: 0

- **--ipv6-peak-server-limit**

If both `--ipv4-peak-server-limit` and `--ipv6-peak-server-limit` are reached, the list of servers from `--peak-handling-srvs` will be used for updates.

If this limit is set to 0, the Updater will try to update from all IPv6 servers (`--internetsrvs`) before trying to update from the `--peak-handling-srvs` list.

The default value: 0

- **--internet-protocol**

The Internet protocol version. It can have three values: `auto`, `ipv4`, `ipv6`. Example:

```
avupdate.bin --internet-protocol=ipv6
```

The default value: `auto`

7.3.2 Update mode

- **--mirror**

Performs a mirror update (meaning no pre / post / unpost applications are executed).

See [7.1 Mirroring the Updater's server structure](#). This parameter is not shown in the `--help` output.

The default value: `false`

- **--check-if-update-available**

If this option is set, the Updater will not install any files. It will only check if an update is available.

There are two situations:

- If `--skip-master-file` is set, the Updater will download the *product.info* files and it will check all local files against the online ones. It will also log the files that are dirty and must be updated.

- If `--skip-master-file` is **not** set, the Updater will download only the *master.idx*. It will check only the local *master.idx* against the online one. If identical, it will return *Nothing to update*. If not identical, it will return *Update is available*.

The default value: `false`

- **--update-modules-list**

Specifies the modules that must be updated (comma-separated list).

For both SAVAPI runmodes, the following values can be assigned to this parameter, depending on what needs to be updated:

SAVAPI daemon/service runmode:

- `SAVAPI` - includes SAVAPI's binary files in the update process (Savapi and libsavapi.so on UNIX systems; or savapi.exe, savapi.dll on Windows systems);

- `SAVAPI_COMMON` - special module that enables pre and post update actions;

- `AVE2` - includes the detection engine files in the update process;



- VDF - includes the virus definition files in the update process;
- SELFUPDATE - includes the files related to the Updater in the update process (*avupdate_msg.avr* and *avupdate.bin* on UNIX systems; or *avupdate_msg.avr* and *avupdate.exe* on Windows systems)

SAVAPI Lib runmode:

- SAVAPI - includes SAVAPI's libraries in the update process (*libsavapi.so* on UNIX systems; or *savapi.dll* on Windows systems);
- AVE2 - includes the detection engine files in the update process;
- VDF - includes the virus definition files in the update process;
- SELFUPDATE - includes the files related to the Updater in the update process (*avupdate_msg.avr* and *avupdate.bin* on UNIX systems; or *avupdate_msg.avr* and *avupdate.exe* on Windows systems)

- **--skip-master-file**

Skip master file (Use this option if you want to mirror the entire structure, not only the changed files).

The default value: *false*

- **--skip-selfupdate**

Skip installing Updater files.

The default value: *false*

- **--no-deltaupdate**

Specifies that the Updater will not use the delta update feature.

The default value: *false*

- **--no-version-check**

Specifies that the Updater will not check the files version in mirror mode. In this case it will only check the md5.

The default value: *false*

- **--no-signature-check**

Specifies that the Updater will not check if the files are signed.



Note Only uncompressed binary files are checked.

The default value: *true*

- **--signature-info-ssl-check**

Check if the info files are signed with the OpenSSL signature (option valid only on Linux and macOS platforms).

Default value: *false*

- **--ext-program-timeout**

Timeout for waiting for an executed pre / post / unpost application (in seconds).

The default value: 1800 s

- **--depend**

If in the product.info file, module 1 depends on module 2, then "update-modules-list=module 1" will update both modules 1 and 2.

The default value: *false* (meaning the Updater will use the new "depend" functionality)

- **--ignore-srvs-list**



For updates from a share or internal http server, the Updater must ignore the servers list present in the IDX file and download everything from the local network and not from the Internet.

The default value: `false`

7.3.3 Connection settings

- **--user-agent**

Specifies the user agent string, which is reported to the http server.

The default value:

```
@AUVI@1.0;<product_name>-UpdateCP/<updater version>(<license
types>;<products>;<language>;AVE <engine version>;VDF <VDF version>;
<operating system name>;<operating system details>;<country>;<serial>;
<license serials>;<operating system language>;)
```

By default, the `<product_name>` is AntiVir. If the `--product-name-file` option is specified or if the default `productname.dat` file exists, `<product name>` is replaced with the content of the respective file.

Examples:

- If the `--product-name-file` option is not specified:

```
@AUVI@1.0;AntiVir-UpdateCP/2.0.3.6 (SAVXE; SAVAPILINUX_ GLIBC24_X86_64-
EN; EN; AVE 8.2.10.52; VDF 7.11.28.140; LINUX X86_64 2.6.38-13-
GENERIC; DISTRO RELEASE SQUEEZE/SID GLIBC 2.13; EN_US.UTF-8;
A182365FA39EE0327E3A4918B0358475; 2100133080-ASRTE- 0000001; EN_US.UTF-8; )
```

- If the default `productname.dat` file applies:

```
@AUVI@1.0;SAVAPI4.0.0.1-UpdateCP/2.0.3.6 (SAVXE; SAVAPILINUX_
GLIBC24_X86_64-EN; EN; AVE 8.2.10.52; VDF 7.11.28.140; LINUX X86_64
2.6.38-13-GENERIC; DISTRO RELEASE SQUEEZE/SID GLIBC 2.13; EN_US.UTF-8;
A182365FA39EE0327E3A4918B0358475; 2100133080-ASRTE- 0000001; EN_US.UTF-8; )
```

- **--product-name-file**

Specifies the file in which the product name is stored (for example `SAVAPI4.0`). The file path is relative to the update binary location. The product name is added to the `<product_name>` field in the `--user-agent` string.

The file must be readable and it must contain the product name, as ASCII printable string, without whitespaces and with a maximum length of 64 characters. Otherwise, an error message is displayed and the update process stops.

If no `product-name-file` is specified and if the default `productname.dat` file does not exist, no changes are made to the user-agent string.

The default value: `productname.dat` containing the following text format: `SAVAPI<VERSION>;` where `Version` represents the major and minor versions (e.g. `4.0`).

Example:

```
avupdate.bin --product-name-file=my_product_name.dat
```

- **--system-proxy**

Tells the Updater to use the system proxy settings.

The default value: `false`

- **--proxy-username**

User name for the proxy authentication.

- **--proxy-password**

Password for the proxy authentication.



- **--proxy-host**
The name of the proxy server.
- **--proxy-port**
Proxy port.
- **--username**
User name for accessing a shared folder, an ftp server or an http server.
- **--password**
The password for accessing a shared folder, an ftp server or an http server.
- **--update-auth-type**
Authentication type that will be used for updates. It can be one of the following:
 - basic
 - digest
 - ntlm
 - anyIf 'any' is chosen, the Updater will first query the site to see which authentication methods it supports and then it will pick one of them.
The default value: `any`
- **--connect-timeout**
The maximum time in seconds that the connection to the server is allowed to take.
This is only used to limit the connection phase.
The default value: `30 seconds`
- **--receive-timeout**
The timeout (in seconds) for receiving a response after a request to an update server.
The default value: `30 seconds`
- **--retries**
Number of retries.
The default value: `0`



Note The retry mechanism is only applied to the download errors. In case of connect errors, the 'retries' number is 0 and cannot be changed.

- **--retry-timeout**
Timeout between retries (in seconds).
The default value: `0 seconds`
- **--force-update**
Bring all modules to the same state as they are on the update server.
The default value: `false`
- **--xvdf-always-merge**
Merge xvdf files even if no files were updated.
The default value: `false`



- **--no-dns-resolve**

Do not perform DNS resolving for the update servers.

The default value: `false`



Note This option is implicitly enabled for https connections, except in the case when **--no-host-check** is also used.

7.3.4 Notification emails

- **--mailer**

Specifies the method for sending emails. Available values:

- smtp - for using your own smtp engine
- sendmail - for using the sendmail binary

The default value: `smtp`

- **--sendmail-path**

When **--mailer** is set to `sendmail`, this parameter specifies the local path of the `sendmail` binary.



Note The path must be a valid and secured `sendmail` executable to avoid potential security risks of executing arbitrary commands.

On UNIX, the default value is: `/usr/sbin/sendmail`

It also searches in `/usr/lib/sendmail`

- **--sendmail-arguments**

If **--mailer** is set to `sendmail`, this parameter specifies the arguments for running the `sendmail` binary.

On UNIX, the default value is: `-oem -oi`

- **--email-to**

The recipient of the notification emails, if **--notify-when** is not 0 (see below).

The default value: `root@localhost`

- **--email-from**

The sender of the notification emails, if **--notify-when** is not 0 (see below).

The default value: `root@localhost`

- **--notify-when**

Sends email notifications to the address set with **--email-to**. Available values:

- 0 - no email notifications are sent (default value)
- 1 - email notifications are sent in case of "successful update", "unsuccessful update" or "up to date"
- 2 - email notification only in case of "unsuccessful update"
- 3 - email notification only in case of "successful update"

The default value: `0`

- **--email-footer**

Changes the footer of the notification email.

The default footer of the notification emails is:

-- © 2016 Avira Operations GmbH & Co. KG. All rights reserved.



Syntax:

```
avupdate.bin --email-to=<yourmail@domain.com> --email-footer=<custom footer>
```

- **--auth-method**

When set in *avupdate.conf*, the Updater requires the smtp login data, `smtp-user` and `smtp-password` (see below), in order to send email notifications to the address set with `--email-to`.

The default value: `false`

- **--smtp-user**

If `notify-when` is not 0 and `auth-method` is set in *avupdate.conf*, the Updater requires the smtp login data: `smtp-user` and `smtp-password`.

- **--smtp-password**

If `notify-when` is not 0 and `auth-method` is set in *avupdate.conf*, the Updater requires the smtp login data: `smtp-user` and `smtp-password`.

- **--smtp-server**

The smtp server for sending email notifications, if `--notify-when` is not 0.

- **--smtp-port**

The smtp port for sending email notifications, if `--notify-when` is not 0.

The default value: 25

- **--smtp-timeout**

Timeout for receiving data when connecting to an smtp server (in seconds).

The default value: 30 seconds

7.4 Avira Updater's logging

The default behavior of the Updater is to create a log file for each attempted update. The log file is created in the directory from which the Updater is called.

The Updater offers logging functionalities that can be configured by using the following parameters:

- **--log**

Specifies a different name of the log file.

The default name: `avupdate.log`

If this option is not present, a default log file with the name *avupdate.log* will be created in the same directory as the Updater binary.

- **--log-rotate**

Overwrites the log files by rotation, meaning that, for each updater execution a new log file will be created. For maintaining a log history, up to 10 recent log files will be kept. For example:

avupdate.log.001, *avupdate.log.002*, *avupdate.log.003*, ..., *avupdate.log.009*

The default value: `false`



Note This parameter is automatically disabled if `--skip-selfupdate` is enabled.




Note This parameter is mutually exclusive with `--log-append` (see below). If both parameters are present, the `--log-append` behavior will be ignored.

- **--log-append**



Appends to log file instead of creating new files. When there is no more space in the log file available (when `log-file-size` is set to a value different than 0, which means infinite), the rotation is automatically enabled, so new files are created like in `log-rotate`.

The default value: `false` (the log is overwritten).

 **Note** This parameter is mutually exclusive with `--log-rotate`. If both parameters are present, the `--log-append` behavior will be ignored.

- **--log-file-size**

Sets the log file maximum allowed size in bytes (0 = unlimited). If the size is exceeded, the log rotation is automatically enabled and a new log file is created.

The default value: 0 (unlimited)

- **--log-template**

Option to specify the format of the log file.

The default template used for logging is:

```
${DAY}/${MONTH}/${YEAR} ${HOUR}:${MINUTE}:${SECOND} ${FULLHOST}[ ${PID} ] :  
${SOURCE} : ${LEVEL} : ${MSG}
```

where:

- `${DAY}/${MONTH}/${YEAR}` - date format
- `${HOUR}`, `${MINUTE}`, `${SECOND}` - time format
- `${FULLHOST}` - host name
- `${PID}` - pid of the program generating the log
- `${LEVEL}` - the message level, as set by the program (*DEBUG*, *INFO*, *MESSAGE*, *WARNING*, *ERROR*, *FATAL*, *MAX_LEVEL*, *UNDEFINED*)
- `${MSG}` - the message sent to the log

Example of log entry:

```
16/07/2016 12:34:04 abc-desktop : UPD: ERROR: Smtplib engine returned error: Connection refused
```

The user can specify the desired information to be logged.


For example:

- in *avupdate.conf*:

```
log-template=${MSG}
```

- in the command line:

```
avupdate.bin --log-template=${MSG}
```

 **Note** All the described scripts can be changed by the user with exception of the STUB executable which is located in SAVAPI's server side update structure, meaning that it will be overwritten every time when the on-line version does not correspond with the local one.

7.5 Avira Updater's return codes

The Updater's main return codes are:

- 0 - successful update
- 1 - nothing to update
- -1 - unsuccessful update



7.6 xVDF files merging

After a successful update, in order to improve the performance, the xVDF files need to be merged into one or more larger files named localXXX.vdf. (ex: local000.vdf). This is done automatically by the Updater binary, but if one chooses to update SAVAPI by other methods, the xVDF merge library must be used after the custom updater finishes.

7.6.1 Merging the xVDF files using the xvdfmerge library

Two files were added to the SAVAPI package for merging the xVDF files. These files are located in:

- bin folder: libxvdfmerge.so for Unix or xvdfmerge.dll for Windows;
- include folder: xvdfmerge.h.

Also an example was added, located in the advanced/xvdfmerge_example folder.

The library merges the xVDF files by using the following function:

```
XVDF_files_merge(const TCHAR* engine_dir, const TCHAR* xvdfs_dir);
```

Simple example for using this function:

```
XVDF_MERGE_STATUS xvdf_ret = xvdf_files_merge("/home/savapi/bin", "/home/savapi/bin");
```

If the merge was successful, the function will return the value XVDF_MERGE_S_OK.

7.6.2 Updating the xVDF merge library

In order to maintain the functionality and to fix possible issues, xVDF merge library needs updates. The library can be updated with an integrated updater module (the Avira Updater) which is available on all xVDF merge supported platforms.

Through command line parameters or configuration file *avupdate-xvdfmerge-product.conf* for the Updater, the following operations can be carried out:

- Check if new updates are available;
- Update xVDF merge library from Avira's update servers or from user-defined servers, with the proper update structure;
- mirror the already configured update servers see

The binary name of the Updater is avupdate.bin (UNIX) and avupdate.exe (Windows).

The xVDF merge library update command:

```
<Updater_binary_name> -C avupdate-xvdfmerge-product.conf
```

At the end of each update cycle, the status of the update will be displayed in the console. The messages (error, warning, etc.) displayed by the Updater when executed are set in the binary file *avupdate_msg.avr*. This file has to exist in the same folder as the *avupdate* binary file, being mandatory for starting the Updater binary.

The xVDF merge library module update name is XVDF_MERGE. This can be used to specify modules by adding the "update-modules-list" option in command line. For more information about configuration parameters, see .

8 Contact information

8.1 Support services

During evaluation, integration, and live use

If you are evaluating or starting to integrate Avira's technology into your solution, or if your integration is finalized and you are going to release your solution to your customers, the Integration Support engineers will answer your technical questions — from planning the architecture of the integration, to detailed code-related routines and live use.



To contact the OEM support team for technical issues, <mailto:oemssupport@avira.com>

Partner Portal

For our OEM customers we also provide a login to our Partner Portal which includes all the latest news and information about Avira's technology, SDK downloads, and documentation: [OEM Partner Portal](#)

8.2 Contact

Avira Operations GmbH Kaplaneiweg 1 D-88069 Tett nang Germany

You can find further information about us and our products on the [Avira OEM website](#).

9 Appendix

9.1 Anti-malware SDK (SAVAPI) binaries

Windows

Assuming SAVAPI is installed in a folder, it will contain the following files:

Description	File name on disk	Installation	Redistributable
SAVAPI binary	<i>savapi.exe</i>	Mandatory	YES
SAVAPI library and its dependencies	<i>savapi.dll savapiclient.dll</i>	Mandatory	YES
Engine binaries	<i>ae*.*</i>	Mandatory	YES
SAVAPI key file	<i>*.key</i>	Mandatory	YES
SAVAPI configuration file	<i>savapi.conf</i>	Optional	YES
Engine VDFs	<i>xbv00000.vdf, ..., xbv00255.vdf</i>	Mandatory	YES
APC library and its dependencies	<i>apcfile.dll</i>	Mandatory for APC support	YES
APC certificate	<i>cacert.crt</i>	Mandatory for APC support	YES
APC random id generator	<i>apc_random_id_generator.exe</i>	Mandatory for APC support	YES
APC hash library	<i>apchash.dll</i>	Mandatory for APC support	YES
SAVAPI OnAccess	<i>avgio.dll</i>	Mandatory for OnAccess	YES
xVDF merge library	<i>xvdfmerge.dll</i>	Optional	YES

9.1.1 The files for the Avira Updater

Description	File name on disk	Installation	Redistributable
Libraries (shared with SAVAPI)	<i>avupdate_msg.avr</i>	Mandatory	YES
Main binary	<i>avupdate.exe</i>	Mandatory	YES
Configuration file	<i>avupdate-savapi-product.conf</i>	Optional	YES
Configuration file	<i>avupdate-savapi-engine.conf</i>	Optional	YES
Configuration file	<i>avupdate-savapilib-product.conf</i>	Optional	YES
Configuration file	<i>avupdate-savapilib-engine.conf</i>	Optional	YES
Product name file	<i>productname.dat</i>	Optional	YES

9.1.2 UNIX

If you have any key files, you can install them in the binaries' directory. Make sure that the permissions for the key files are correct.



Description	File name on disk	Installation	Redistributable
SAVAPI binary	<i>savapi</i>	Mandatory	YES
SAVAPI library and its dependencies	<i>libsavapi.so libsavapiclient.so</i>	Mandatory	YES
Engine binaries	<i>ae*.*</i>	Mandatory	YES
SAVAPI key file	<i>*.key</i>	Mandatory	YES
Engine VDFs	<i>xbv00000.vdf, ..., xbv00255.vdf</i>	Mandatory	YES
SAVAPI configuration file	<i>savapi.conf</i>	Optional	YES
APC library and its dependencies	<i>libapcfile.so</i>	Mandatory for APC support	YES
APC certificate	<i>cacert.crt</i>	Mandatory for APC support	YES
APC random id generator	<i>apc_random_id_generator</i>	Mandatory for APC support	YES
APC hash library	<i>libapchash.so</i>	Optional for APC support	YES
xVDF merge library	<i>libxvdfmerge.so</i>	Optional	YES

The files for the Avira Updater

Description	File name on disk	Installation	Redistributable
Libraries (shared with SAVAPI)	<i>avupdate_msg.avr</i>	Mandatory	YES
Main binary	<i>avupdate.bin</i>	Manatory	YES
Configuration file	<i>avupdate-savapi-product.conf</i>	Optional	YES
Configuration file	<i>avupdate-savapi-engine.conf</i>	Optional	YES
Configuration file	<i>avupdate-savapilib-product.conf</i>	Optional	YES
Configuration file	<i>avupdate-savapilib-engine.conf</i>	Optional	YES
Product name file	<i>productname.dat</i>	Optional	YES

Europe Middle East, Africa

Avira
Kaplaneiweg 1
88069 Tetttnang, Germany
Tel: +49 7542 5000

Americas

Avira, inc
c/o WeWork, 75 E Santa Clara Street
Suite 600, 6th floor San José
CA 95113 United States

Asia/Pacific and China

Avira Pte Ltd
50 Raffles Place
32-01 Singapore Land Tower
Singapore 048623

Japan

Avira GK
8F Shin-Kokusai Bldg
3-4-1, Marunouchi Chiyoda-ku
Tokyo 100-0005, Japan