SOPHOS

Security made simple.

Anti-Spam Engine
Software Development Kit

Product Version 6.3.3
Sophos Limited 2018

Contents

1 Getting Started

The Sophos Anti-Spam Software Development Kit gives software developers access to the spam detection functionality that is usually provided as part of the Sophos PureMessage mail-filtering product.

With the Anti-Spam SDK you can:

- scan email to find the message's numerical spam probability.
- integrate anti-spam functionality into third party applications; for example, a mail transfer agent (MTA) or a mail user agent (MUA).
- create custom applications that require spam detection functionality.

The Sophos Anti-Spam Software Development Kit provides a C language API that developers can use to write applications in either C or C++ that access the engine. This guide introduces the Anti-Spam Engine API and explains how to use it to create applications that can access or integrate spam detection functionality.

The engine has the ability to enable live, DNS-based queries to Sophos, improving catch rates by way of real-time updates from SophosLabs and by allowing a larger set of anti-spam data to be maintained. If a spam decision cannot be made based on local data, the engine will perform queries to Sophos to compare sending IP addresses, URIs within messages, and other message attributes with the latest data from SophosLabs. In order to help Sophos maintain good anti-spam engine performance, the engine also has the ability to send certain performance and latency statistics back to Sophos using a feedback tool. For optimum performance, it is recommended that you configure your engine to share these statistics.

1.1 Installation

This section discusses what you need to do to prepare for, and install, the Sophos Anti-Spam Software Development Kit.

1.1.1 Supported Platforms

The Sophos Anti-Spam Software Development Kit can be installed on:

- Windows 2000, XP, 2003
- Linux on x86 (Red Hat 6.2 through 8.x, and RHEL 4.x)

1.1.2 Distribution

The anti-spam engine is distributed as an archive (.zip or .tar.gz) file, located at http://pmx.sophos.com/as-sdk. You will also need to download the antispam data file (.zip or .tgz). A user account and password are required to retrieve the SDK and updates to the data files. This information will be sent by email.

Note: If you want to use the `sample.c` program included in the distribution, the location specified in the `core.datadir` attribute must match the location of the installation directory. For more information, see the “Core Attributes” section of Anti-Spam Engine Attributes.

Archive Contents

The archive contains the following directories and files. The first two files appear directly beneath the directory into which you extracted the distribution (indicated below as *InstallDir*).

***InstallDir*/bin/**	**Windows: pmx_engine.dll**	This file (or `pmx_engine.lib`) must be installed on Windows systems, in a directory that is identified in the system’s `PATH` variable.
	Windows: feedback.exe	An executable used to upload engine statistics to Sophos.
	Linux: feedback	An executable used to upload engine statistics to Sophos.
***InstallDir*/datadir/**	The location where antispam data files must be installed.	
***InstallDir*/eg/**	**Makefile**	A file that can be used to build the samples.
	README	The file describes how to use the Makefile to build `sample.c`.
	sample.msg	A sample message used by `sample.c`
	sample.c	A sample program that uses the .dll or .so.

***InstallDir*/include/**	**pmx_engine.h**	The C header file required to use the Sophos Anti-Spam Engine.
	pmx_message.h	An optional C header file that can be included to provide more control over the message object used in the Sophos Anti-Spam Engine.
***InstallDir*/lib/**	**Linux: libpmx_engine.so**	The shared object for building and compiling the application in UNIX. This is a required file on UNIX systems, and it must be installed in a directory identified in the `LD_LIBRARY_PATH` environment variable.
	Windows: pmx_engine.lib	The static library for building and compiling the application in Windows. Either this file or `pmx_engine.dll` is required on Windows systems.

***InstallDir*/MANIFEST**	Contains filenames and MD5 hash space separated in one line, for example:
	`8e02a050d34bacb81c75d0d68191d603 pmx_engine.dll`
	`b9d73c166f31623f5fab45c5384ef008 eg/sample.c`	

***InstallDir*/ReadMe.html**	This file links to online documentation.	

Antispam Data File

Extract the following two files from the antispam data file (.zip or .tgz) and install them in the **datadir** directory, beneath the directory into which you extracted the distribution (indicated above as ***InstallDir*/datadir/**).

asdb.antispam	This file, which is required on all systems, must be installed in the directory referenced by the `core.datadir` attribute.
db.summary	This file, which is required on all systems, must be installed in the directory referenced by the `core.datadir` attribute.

Related concepts

Core Attributes on page 44

1.1.3 Installing the Anti-Spam SDK

The Anti-Spam update package contains the most recent version of the engine runtime library, the required data files, and the `MANIFEST` file.

1. Navigate to http://pmx.sophos.com/as-sdk. A user account and password are required to retrieve the Anti-Spam SDK package. From the root directory, find the desired update version and operating system. Download the Anti-Spam package and its MD5 checksum.

   Sophos maintains separate directories for each of the various engine versions. To get the latest version, download your install package from the `2.x-latest` directory, which automatically points to the latest engine version. Note that if there are API changes, Sophos may not update the `2.x-latest` link.

   For example:

   ```
   http://pmx.sophos.com/as-sdk/2.6/win32/antispam.zip
   http://pmx.sophos.com/as-sdk/2.6/win32/antispam.zip.md5
   ```

   Or:

   ```
   http://pmx.sophos.com/as-sdk/2.6/linux/antispam.tgz
   http://pmx.sophos.com/as-sdk/2.6/linux/antispam.tgz.md5
   ```

2. Compare the MD5 checksum with your most recently downloaded package. If the checksums are different, download the `antispam.zip` package again.

   a) If required, pass the file through your update verification system.

   b) If required, repackage to a different distribution format.

3. Publish the update package on your site for your customers to download.

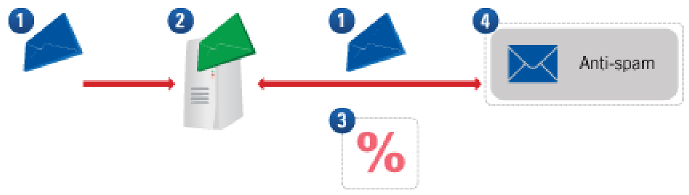
The majority of updates will not contain engine library changes. To determine if the update package contains engine changes, use the MD5 checksum to review the library file in the `MANIFEST`.

Alternatively, review the library version number inside the binary. The number is incremented whenever the library changes.

Data files contained within the anti-spam update must only be used with the corresponding version of the engine software, found in the same package. Using incompatible versions will result in engine failure.

1.2 API Overview

The following diagram demonstrates how a host application could integrate with the Sophos Anti-Spam Engine API.



1. Message
2. Host Application (for example, an MTA)
3. Spam Probability
4. Sophos Anti-Spam Engine

Description:

- The host application (for example, a Mail Transfer Agent [MTA] or a policy framework) passes an email message to the anti-spam engine.
- Using defined rules, the engine scans the email message for specific spam features.
- A spam probability for the email message is returned to the host application.

Note: For information on what a client can do with an email message and an associated spam probability, see “Scanning Message Details”.

Related concepts
Scanning Message Details on page 22

1.2.1 The Object Model

The Sophos Anti-Spam API is comprised of four types of objects. Each object type is prefixed with 'PMX'. These object types are:

- **PMX_ENGINE**: The anti-spam engine itself. Used to scan email messages for spam.
- **PMX_MESSAGE**: The complete email message. This includes all headers (for example: `Subject, To, From`), body text, and attachments.
- **PMX_HASH**: A hash table of key/value pairs that can be loaded from, or saved to, a file on disk. Keys and values are of type string. This interface object is required when specifying trusted relays. See the `eg/sample.c` file for details of its implementation.

- **PMX_WEIGHTS**: This object is provided for backwards compatibility, do not use it. A hash table of key/value pairs. Keys are of type string and values are floating point numbers. This interface object is not required for basic SDK usage.

These objects are implemented in the Perl programming language and are exposed via a C wrapper layer. The Perl interpreter is part of the anti-spam engine distribution and is located in the `lib` directory.

The C API uses a struct with a virtual function table (`vtbl`) for each of the object's methods. For example, a method must be accessed though the `vtbl` as follows:

```
my_engine->vtbl->method(...);
```

The `PMX_ENGINE` is passed individual `PMX_MESSAGE` objects and computes a number indicating the email message's spam probability. This is done by aggregating values for spam features found in the compiled data.

The engine passes information back to the program that called it using several client-defined callback functions. These callbacks are passed to the engine as function pointers. For example, the engine's `scan_message` method returns information on found spam features and the aggregated spam probability via callback. For the required structure of a specific callback function, see "Callbacks" in the Anti-Spam Engine Reference section.

A `PMX_MESSAGE` object that has been created by one instance of `PMX_ENGINE` can also be used by other `PMX_ENGINE` instances.

Related concepts
Callbacks on page 29

1.2.2 The Threading Model

The host application is expected to manage multiple `PMX_ENGINE` objects in order to perform concurrent scanning. The behavior when multiple threads share the same object is not defined.

There are two exceptions: The `create_message()` method may be called by any thread, and the `PMX_MESSAGE` object returned from it may be passed to a different `PMX_ENGINE` object. The `create_hash()` method may be called by any thread, and the `PMX_HASH` object returned from it may be passed to a different `PMX_ENGINE` object.

1.2.3 API Versions

All virtual function tables (`vtbl`) include a version number as the first element. This allows later versions of the engine to work successfully with hosting environments that implement earlier versions of the `PMX_ENGINE`, or `PMX_MESSAGE` API.

1.3 Release Notes

This document describes the changes included in each Sophos Anti-Spam Software Development Kit release. The most recent releases are described first.

Version 2.7.2 (June 2009)

This release introduces version 2.7.2 of the Sophos Anti-Spam Engine, which contains a variety of enhancements that ensure continued best protection against spam threats.

Version 2.7.1 (April 2009)

This version of the Sophos Anti-Spam Engine contains a number of fixes that ensure continued best protection against spam threats.

Version 2.7.0 (March 2009)

Version 2.7.0 of the Sophos Anti-Spam Engine contains a variety of enhancements that ensure continued best protection against spam threats. In addition, an event code and an associated core attribute have been added that provide SophosLabs with important data about the configuration of trusted relays. When the `PMX_EV_FUR` event code is specified, the engine determines the IP address of the first untrusted relay. This event code only takes effect if the `core.event.fur` attribute has also been set.

In order for this data to be useful to SophosLabs, it is recommended that you append it to an informational header, so that the IP address of the first untrusted relay is included in all false positives and false negatives submitted to Sophos. The format for this header is described in the “Event Codes” section of the Anti-Spam Engine Reference.

See the “Anti-Spam Engine Reference” and “Anti-Spam Engine Attributes” sections of the documentation for more about `PMX_EV_FUR` and `core.event.fur`.

Version 2.6.1 (September 2008)

Version 2.6.1 of the Sophos Anti-Spam Engine contains a variety of enhancements that ensure continued best protection against spam threats. In addition, specific improvements to the way in which the anti-spam engine alerts Sophos about SXL issues will result in fewer SXL-related timeouts.

For optimal performance, it is recommended that you enable the Feedback Tool. Turning on this feature provides statistical information to SophosLabs about spam that has been processed by the anti-spam engine. See the "Using the Anti-Spam Engine" section of the documentation for more information.

The following improvements have also been made:

- Previously, when performing SXL look-ups, the engine queried DNS servers using the round robin technique. This sometimes caused delays if servers were unavailable. The engine now queries the primary DNS server first, querying secondary servers only when it is necessary. This has increased engine reliability and performance.
- To comply with RFC 1918, the network 172.16.0.0/12 has been added to the group of private networks that are exempt from network-based tests. See the “net attributes” section of the documentation for more information.

Version 2.6.0 (November 2007)

This release contains the following improvements:

- **SXL Plug-In**: Performs real-time, DNS-based queries to Sophos regarding IP addresses, URIs within messages, and image fingerprints. Queries are triggered when the anti-spam engine has been unable to determine if a message is spam. These real-time lookups provide zero latency between the time that Sophos makes new anti-spam data available and when it is available for use by the anti-spam engine. This functionality is enabled by default. For more information, see “SXL Attributes” in the Anti-Spam Attributes section.

- **Feedback Tool**: This tool makes it possible to report summary statistics to Sophos about spam that has been processed by the anti-spam engine. SophosLabs uses this information to improve the engine’s ability to detect spam. You are strongly encouraged to enable this feature. For more information, see “Feedback Tool” in the Using the Anti-Spam Engine section.

Version 2.5.2 (July 2007)

This release provides SophosLabs with new tools to combat spam in PDF and other attachment formats.

Version 2.5.1 (March 2007)

This release extends detection capabilities to further improve SophosLabs ability to respond to image spam campaigns.

In addition, a new core attribute has been added, `core.data-version`, which returns the version of the anti-spam data that the engine currently uses.

Version 2.5.0 (October 2006)

Version 2.5.0 improves the sender information available to SophosLabs for use in spam rules.

To take advantage of these enhancements, ensure that you are using the `plugin.net.trusted-relays` attribute.

Version 2.4.0 (May 2006)

Version 2.4.0 contains significant improvements to "spam identities" (checksum/signature-based detection) that will enable SophosLabs to respond to certain HTML and image spam campaigns more quickly.

This release no longer includes the `pmx-compile` utility included in previous releases of the Anti-Spam Engine.

Version 2.3.0.0 (February 2006)

This release includes the following new features:

Improvements	The character set of a message is now identified early on in message processing. Anti-spam rules can now specify which character set they apply to. This allows SophosLabs to publish language-specific rules which are used only on messages of a matching character set. A new `stop_scan` option allows the engine to stop scanning a message once it is definitively determined to be either spam or not spam. This increases the efficiency of the engine. As further

	rules are not triggered after the scanning is stopped, the cumulative spam score may be different and the excluded rules will not appear in anti-spam reports and headers. This option is disabled by default. The `sig` plugin can now be used effectively against messages larger than 10K. SophosLabs now publishes spam identities for individual paragraphs within these larger messages. URI extraction is now much faster which has improved the throughput performance of the engine.

Version 2.2.0.0 (January 2006)

This release includes the following new features:

New Plugin	The `redb` plug-in works like the `re` plug-in by extracting certain patterns from a message and checking the extracted portions against various databases generated by SophosLabs (for example, databases of phone numbers owned by known spammers).

Version 2.1.0.0 (July 2005)

This release includes the following new features:

New Plugin	The sig plugin checks messages against "spam identities" generated by SophosLabs. Spam identities are content-based checksums generated from captured spam used to detect messages from specific spam campaigns.
Improvements	URI extraction is much more aggressive in what is considered a URI. More URI forms are recognized (for example, raw, canonicalized, path-less) including phishing targets. The message size limit (`core.max-bytes-scanned` attribute) has been increased from 8K to 10K. Improved obfuscated word and invisible text detection.

Version 2.0.3.2 (April 2005)

Bug Fixes	This release fixes a bug which caused the `word_obfu` plugin to produce inconsistent results.

Version 2.0.3.1 (March 2005)

Bug Fixes	This release fixes a bug which caused excessive scan times for messages containing large blocks of repeated characters.

Version 2.0.3.0 (January 2005)

This release includes the following new features:

Bug Fixes	Fixed a bug where messages containing long repeats of the same character would take a long time to scan. Fixed a bug where comments in `/etc/resolve.conf` settings was being ignored.
Documentation Fixes	The documentation has been updated to reflect the changes from 1.6 to 2.0.

Version 2.0.2.0 (October 2004)

This release includes the following new features:

Heuristic Updates New Rules	Added new rules to catch 419 Nigerian spam and "unobfuscated" medical spam.
Rules Removal	Removed several rules due to false positives.
Improvements	Improved both the obfuscated word detection system and the invisible text detection system.
Bug Fixes	Major fix to the obfuscated word detection system where SDK versions 1.6.x.x would erroneously fire on "not-spam" emails. Major fix to the SpamEngine to prevent it from crashing on complex MIME messages.

DNSBLs	Added several third party DNSBLs. These DNSBLs are disabled and in a zero weighted state. Be advised that enabling these DNSBLs may require permission from the DNSBL provider. Be aware that enabling and weighting blackhole DNSBLs causes all mail from a specified region to be caught as spam, regardless of actual content.

Version 2.0.1.0 (September 2004)

This release includes the following new features:

Rules Removal	Many rules have been removed in this release. These rules were removed due to any of the following reasons: causing false positives, having no appreciable weight, or no longer firing in spam.
Bug Fixes	Minor fixes to the obfuscated word detection system and the uri extraction system.
Heuristic Updates	Various heuristic updates and rule modifications.

Version 2.0.0.0 (August 2004)

This release includes the following new features:

Compiled Data	The anti-spam engine now reads data from a single binary file. All engines share the same binary file, reducing the CPU time and memory required to initialize the engine.
Lower Memory Usage	Each thread memory maps the same compiled data. Additionally, the per-thread memory usage is lower than previous versions due to various internal optimizations.
Faster Scanning	This engine version provides a small performance increase over previous versions.
Message Sharing Support	A `PMX_MESSAGE` object created in one thread can be scanned by a `PMX_ENGINE` object in another thread. Applications can employ a pool of threads that create `PMX_MESSAGE` objects, and use a different pool of scanner threads.

High-Memory Watermarks	The `PMX_MESSAGE` object has several layers of high water marks that prevent large or pathologically complex messages from consuming resources.
Message Size	The default maximum message is 64 KB. Once the message reaches its watermark, further calls to `append()` are silently ignored. Previously, large messages could cause large amounts of RAM to be allocated.
Number of MIME Parts	The maximum number of MIME parts is 128. Previously, there was no limit to the number of MIME parts.
Extended PMX_MESSAGE API	The `PMX_MESSAGE` object returned from the `create_message()` method can be cast into a `PMX_MESSAGE_WRITABLE` object. This exposes a new virtual function table that provides several new features.
Provide Connection Information	Application can use the `PMX_MESSAGE_WRITABLE` interface to set the envelope sender and envelope recipients, set the relay IP address and relay hostname, and set the `SMTP HELO` line.
Append From File	A `PMX_MESSAGE` object can be configured to read on-demand from a file: `wmsg->wvtbl->read_from_file(wmsg, "filename");`
Append From Callback Function	A `PMX_MESSAGE` object can call an application provided callback function whenever more data needs to be read: `wmsg->wvtbl->read_from_func(wmsg, &reader);`

Known Issues:

Linux threading (26724)	The Linux version of the SDK is thread safe, but recycling the engine results in a memory

	leak. For this reason, using the engine in a threaded application is not recommended.

1.4 Copyrights and Trademarks

PCRE License

somewhere reasonably visible in your documentation and in any relevant files or online help data or similar. A reference to the ftp site for the source, that is, to

```
ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/
```

should also be given in the documentation. However, this condition is not intended to apply to whole chains of software. If package A includes PCRE, it must acknowledge it, but if package B is software that includes package A, the condition is not imposed on package B (unless it uses PCRE independently).

3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software.
4. If PCRE is embedded in any software that is released under the GNU General Purpose Licence (GPL), or Lesser General Purpose Licence (LGPL), then the terms of that licence shall supersede any condition above with which it is incompatible.

Note: The PCRE library distributed with the Sophos Anti-Spam SDK has been modified.

1.5 Contact Sophos

Sophos Support

If you encounter a problem with your Sophos product, or if it does not function as described in the documentation, contact technical support: http://www.sophos.com/support/.

Corporate Contact Information

To contact your local Sophos office, see: http://www.sophos.com/companyinfo/contacting/

2 Using the Anti-Spam Engine

The code examples in the following pages demonstrate how to use the Anti-Spam Engine API.

These examples are extracted from the `sample.c` program located in the `eg/` directory.

Suggested Best Practices

Sophos recommends the following for ensuring high catch rates when using and integrating the anti-spam engine.

1. **Keep up to date with anti-spam engines**: This can be accomplished by downloading from the "2.x-latest" area of the Sophos update page. Alternatively, if you point to specific version directories for testing purposes, try to test and publish the new engine as soon as possible once Sophos has announced it (ideally within a day or two).
2. **Keep up to date with anti-spam data**: Create a mechanism to regularly check the Sophos site for updates, preferably once a minute. Each data update has an associated checksum that can be compared against the checksum of the last downloaded package. Using this method is improves efficiency because the data is only downloaded if the checksums do not match. Even though updates are not actually published every minute, minimizing latency as much as possible will improve catch rates. Apply any available updates quickly once they are available. Implement monitoring of data versions to ensure they are updating on a regular schedule. Data older than 1-2 hours can indicate an updating issue.
3. **Leave all default rules/checks enabled including network checks**: Sophos recommends leaving all default rules and checks enabled in the anti-spam engine, including network checks such as DNSBL lookups and reverse DNS checks, (that is, the `core.local-tests-only` attribute should not be set to true). Network checks contribute significantly to the anti-spam catch rate. *If* you are doing specific DNSBL checks outside of the anti-spam engine and wish to avoid duplicating these queries in the engine, Sophos recommends disabling only those specific DNSBL rules in the engine, still allowing other network checks to occur.
4. **Ensure accurate spam source information is available to the engine**: A number of checks rely on accurate information on the spammer's sending/connecting IP, including DNSBL-type rules and reverse-DNS rules. Specifying any IPs/relays that should be regarded by the engine as trusted/internal should be done using the `plugin.net.trusted-relays` attribute, allowing the engine to use the first untrusted relay as the spamming IP. You may also want to exclude internal hosts from network checks by specifying them in the `plugin.net.internal-hosts` attribute (`10.0.0.0/8`, `127.0.0.0/8`, `172.16.0.0/12`, and `192.168.0.0/16` are automatically excluded). If applicable, DNS servers should be specified using the `plugin.net.dns-severs` attribute. Also, if the Anti-Spam SDK does not get email directly from a spamming host, there must be a received header added by a front-line MTA. The format of the `Received` header should resemble the standard ones (such as Sendmail or Postfix), so that the Anti-Spam SDK can parse it to determine the first untrusted relay.
5. **Use a 50% threshold**: The Sophos-recommended threshold is 50% (or 0.5) for categorizing a message as spam. Some customers choose a higher threshold (for example, 90%) for more aggressive disposition, such as discarding, but this results in a higher potential for false positives.

6. **Add X-header(s) to messages for troubleshooting**: Sophos recommends that customers add the following information to messages processed by the anti-spam engine to assist with any troubleshooting or investigation of miscategorized messages:
   - The version of the anti-spam engine in effect when the message was processed, which is reported by the `core.version` attribute.
   - The version of the anti-spam data in effect when the message was processed, which is reported by the `core.data-version` attribute.
   - The rules that fired when the message was processed, which is reported by the `PMX_EV_FOUND_FEATURE` event code. Ideally, include both parameters.

   Note: Adding these headers is believed to have a negligible impact on performance in most situations relative to the spam engine processing. If this is not the case, you can add these headers only to messages that were not determined to be spam. This will allow analysis of missed spam, although analyzing false positives will not contain this information.
7. **Provide full header information with samples submitted to Sophos**: Message samples should contain the full source including the spam X-header(s) added during processing and the full received chain.
8. **Report any missed spam or false positives to SophosLabs**: Samples should include the full, intact headers including the anti-spam X-header(s) and the full received chain.
9. **Set up a spam trap**: Sophos encourages customers to set up spam traps where mail from unused and old addresses or domains can be routed directly to Sophos. Traps should not receive any legitimate mail, only spam. Setting up a spam trap will help give Sophos visibility into any campaigns that are targeting your particular domains.

2.1 Initializing the Engine

The following example shows how to create and initialize a `PMX_ENGINE` object.

```
/* Create an engine, have it use "/usr/local/lib/datadir"
* as its datadir */

PMX_ENGINE e;
pmx_load(NULL);

if (pmx_create_engine(&e, error, NULL) != PMX_ERR_OK) {
    printf("Could not create PMX_ENGINE instance\n");
    return 1;
}

e->vtbl->add_attribute(e, "core.datadir",
    PMX_TYPE_STRING, "/usr/local/lib/datadir", 10, 1);

e->vtbl->load_antispam_data();
```

Description:

- Initialize the library using `pmx_load` function

- Create an instance of the engine using the `pmx_create_engine` function.
- Set the `core.datadir` attribute to indicate the locations of all data files.
- Load the data file.

2.2 Setting Trusted Relays and Internal Hosts

The following example shows how to set the `plugin.net.trusted-relays` and `plugin.net.internal-hosts` attributes.

```
   /* Set up two trusted relays */

   PMX_HASH trusted_relays, internal_hosts;

   e->vtbl->create_hash(e, &trusted_relays, NULL, 0);
   trusted_relays->vtbl->set(trusted_relays, "213.31.172.15", "1",
1);
   trusted_relays->vtbl->set(trusted_relays, "213.31.172.14", "1",
1);
   trusted_relays->vtbl->save(trusted_relays);

   e->vtbl->add_attribute(e, "plugin.net.trusted-relays",
PMX_TYPE_HASH,
   trusted_relays, 0, 1);

   /* Set up an internal host */
   e->vtbl->create_hash(e, &internal_hosts, NULL, 0);
   internal_hosts->vtbl->set(internal_hosts, "74.202.89.140", "1",
1);
   internal_hosts->vtbl->save(internal_hosts);

   e->vtbl->add_attribute(e, "plugin.net.internal-hosts",
PMX_TYPE_HASH,
   internal_hosts, 0, 1);
```

Description:

- Create a hash and add trusted relays to it.
- Set the `plugin.net.trusted-relays` attribute to skip over the relays specified in the PMX_TYPE_HASH when working through the Received header chain to find the first external relay.

  Note: By default, all external IP addresses found in a message's Received headers are checked against DNSBL lists. This behavior is backwards-compatible with previous releases, but it has a significant risk of false positives. This can be solved by maintaining a `Trusted Relays` list and setting the `plugin.net.trusted-relays` attribute. The `plugin.net.trusted-relays` attribute should always be used unless your application will always be run on an edge server (never behind any relays). Also, if applicable, you should specify `plugin.net.internal-hosts` to exempt internal hosts from network-based tests. You may also want to specify `plugin.net.dns-servers`.

- Create a hash specifying an internal host.
- Set the `plugin.net.internal-hosts` attribute to exempt internal hosts from network-based tests.

Related concepts

net Attributes on page 46

2.3 Scanning a Message

The `PMX_ENGINE` scans a `PMX_MESSAGE` object. The following code illustrates reading an email message from disk into a `PMX_MESSAGE` object, scanning it with a `PMX_ENGINE`, and then destroying the `PMX_MESSAGE` object.

```
for (i = 1; i < argc; i++) {
    PMX_MESSAGE msg;
    FILE *f = fopen(argv[i], "r");
    char buf[4096];
    size_t nread;

    if (!f)
        continue;

    e->vtbl->create_message(e, &msg);
    while (!feof(f) && !ferror(f)) {
        nread = fread(buf, 1, sizeof(buf), f);
        pmx_append_to_message(msg, buf, nread);
    }
    fclose(f);

    /* Perform actual scanning with scan_cb callback */

    e->vtbl->scan_message(e, msg, NULL, &scan_cb, NULL);

    msg->vtbl->destroy(msg);
}
```

Description:

- For each filename specified on the command line, this program reads the file from disk into a `PMX_MESSAGE` object created by the engine's `create_message` method.
- The `PMX_MESSAGE` object is then passed to the engine's `scan_message` method.
- After the email message is scanned, it is then destroyed with its `destroy` method.

2.3.1 Scanning Message Details

The PMX_EV_SPAMPROB event contains the email message's spam probability. The probability is calculated according to the score assigned to the rules matched by the email message.

```
static void
scan_cb(void *host, PMX_EVENT e, PMX_TYPE t1,
    const void *v1, size_t l1, PMX_TYPE t2,
    const void *v2, size_t l2)
{
    switch (e) {
    case PMX_EV_FOUND_FEATURE:
        if (t1 == PMX_TYPE_STRING)
            printf("Found feature: %s\n", v1);
        break;
    case PMX_EV_SPAMPROB:
        if (t1 == PMX_TYPE_DOUBLE)
            printf("Spam probability for this message:
%5.3f%%\n",*(double*)v1);
        break;
    }
}
```

Description:

- The scan_cb callback function is passed several parameters. These include the event that triggered the callback and the values with their associated types.
- If the callback function is triggered by a feature found in an email message, the event type is PMX_EV_FOUND_FEATURE and the first value is the name of the found feature.
- If the callback function is invoked because scanning is complete and the engine has a spam probability to report, the event type is PMX_EV_SPAMPROB.
- The spam probability for the scanned email message is the value passed.

2.4 Terminating the Engine

The following example shows how to terminate the PMX_ENGINE object and release associated resources.

```
/* Unload PMX_ENGINE e */

e->vtbl->destroy(e);
pmx_unload();
```

Description:

- Free engine resources by invoking the engine's destroy method.
- Use pmx_unload before terminating the program.

Note: It is highly recommended that you stabilize memory usage and overall performance by retiring a `PMX_ENGINE` after 2048 email messages have been scanned. The embedded Perl interpreter dynamically allocates memory during the scanning process, and can be slow to release it. This is especially important if very large email messages are regularly scanned.

2.5 Feedback Tool

The feedback tool is used to upload engine statistics to Sophos. It is recommended that you make use of this tool because it provides SophosLabs with valuable information that it uses to increase spam catch rates. It should be set to run once every five minutes.

This stand-alone application can be run from a shell (such as by cron on Linux/Unix platforms) or as a sub-task. The feedback tool has a number of options (defined below). For example, if you need to specify a proxy server, use **`--proxy`**. Or, to preview the data that will be uploaded to Sophos, use the **`--dry-run`** option.

Usage (Windows):

```
feedback.exe [OPTIONS] FEEDBACK_FILE
```

Usage (Linux):

```
feedback [OPTIONS] FEEDBACK_FILE
```

Options:

```
--statsdir   -s    location of statistics data
(core.statistics-directory)
--proxy      -p    proxy location (for example,
webproxy.sophos.com:8080)
--username   -n    username for proxy authentication
--password   -a    password for proxy authentication
--url --cdfs -u    HTTP URL for feedback data (CDFS)
--cert       -c    Optional Certificate Authority certificate to use
                   for authentication with server
--verbose    -v    displays the uploaded report
--dry-run    -d   do not actually upload the report; implies --verbose
--help       -h    display this help and exits
```

The `--statsdir` or `-s` is the only required option, and it specifies the feedback directory. The feedback file is required.

The feedback file must contain at a minimum:

```
Report-Version: 3
```

Other information may be included, using the format `KEY: VALUE`.

3 Spam Probabilities

Anti-spam rule files contain sets of test definitions that are used to identify potential spam features in an email message. Each feature, regardless of the rules file it is defined in, has an associated weight that contributes to the message's total spam score.

Weights are specified as numerical values and can be either positive or negative numbers. Positive weights increase the likelihood that a message is spam, while negative values decrease the likelihood.

An email message with several positive spam features will thus have a higher aggregated score than a message with negative or few spam features. Generally, a message must contain multiple spam features in order to result in a high aggregated spam score.

After an email message is scanned and all features are tested for, the anti-spam engine totals all associated weights into a spam score. The engine then converts the spam score into a percentage indicating the probability that the message is spam.

Example: A Spam Percentage

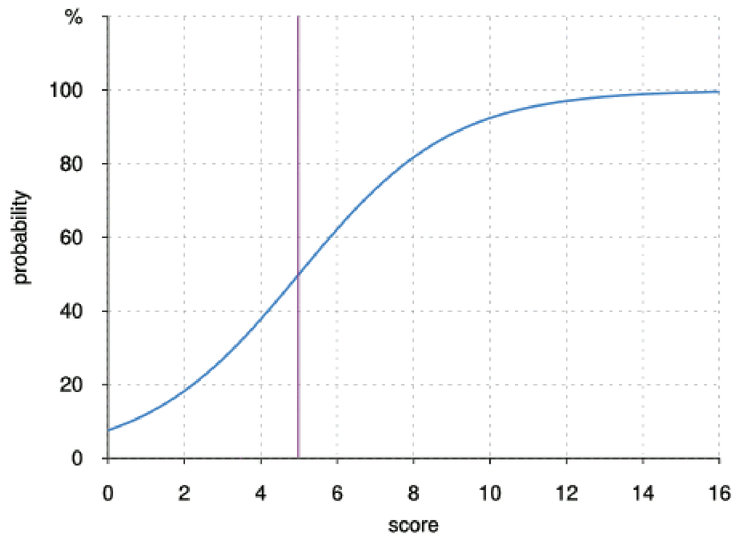
```
Spam probability for this message: 99.412%
```

The following formula shows how the spam score is converted into a spam probability:

```
PROB = 1/(1 + exp(-(BIAS+SCORE)/2))
```

Where `BIAS` = -5 and `SCORE` is the sum of weights of triggered features.

The chart below shows the relationship between scores and percentages. A score of '5' results in a 50% spam probability.



3.1 Filtering On Probability Ranges

Any application that you develop using the Anti-Spam SDK will need to handle messages based on the returned spam probability. The anti-spam engine is set, by default, to recognize spam at or above a 50% threshold (a spam probability range of 50% to 100%). A message with a spam probability under 50% is considered legitimate. Sophos recommends that your applications should treat messages with a greater than 90% spam probability as being "definitely spam," which can then be discarded or rejected as appropriate. Messages that return a 50% to 90% spam probability should be treated as "likely spam," which can then be tagged, stored in a folder, quarantined, etc.

The following example defines a set of sample probability ranges based on the suggested spam probability ranges, with filtering actions becoming stricter as the spam probability range increases.

Example Filtering Actions Based on Spam Probability Ranges

- **0-49%**: Add an `X-Sophos-AntiSpam:` header to the email message with the probability appended. Deliver the message to envelope recipients.

  Note: Adding a header to a message is useful for testing and debugging purposes. For example, if a message is determined to be a false negative (spam delivered as legitimate email), reviewing the header easily reveals the spam probability of the message.

- **50-90%**: Add an `X-Sophos-AntiSpam:` header to the email message. Append a '#' mark for every 10% calculated over a probability of 50%. For example, a message with a probability of 60% will have an `X-Sophos-AntiSpam: #` header added. A message with a 70% probability will have an `X-Sophos-AntiSpam: ##` header added. Add an `X-Sophos-AntiSpam-Hits:` header to the email message to identify the spam features that contribute to the message's spam probability. Modify the message's `Subject:` header, and then deliver the message to envelope recipients. Envelope recipients can then filter messages into different folders on their mail client by using substring matches on the '#' mark.

  A Resulting Message Header with an 88% Spam Probability:

```
Received: ...
From: fooey@spammer.ick
X-Sophos-AntiSpam: ### (88%)
X-Sophos-AntiSpam-Hits: AMAZING_STUFF, APPLY_ON_LINE, RCVD_IN_CBL,
...
Subject: [SPAM: 88%] original subject line
```

- **Over 91%**: Send the email messages to a spam directory or quarantine. Notify envelope recipients of the blocked spam messages. Allow recipients to access and release blocked messages.

Note: Thresholds, email modifications, and filtering actions are only suggested uses for the returned email message spam probability. Your host application may have different message-filtering attributes.

3.2 Handling False Positives and Negatives

The number of false positives (messages that are identified as spam, but are not) and false negatives (messages that are not identified as spam, but are) can be reduced by forwarding these messages to Sophos. SophosLabs analyzes characteristics of submitted messages and adjusts the anti-spam data accordingly. False positive messages should be sent to not-spam@labs.sophos.com, and false negatives should be sent to is-spam@labs.sophos.com.

You should consider including the capability that allows users or administrators to report such falsely identified messages. If you do, note the following requirements:

- The message forwarded to Sophos must include the whole message source, including headers, sent as an RFC822 attachment.
- The headers must include the version number of the engine and the version and date of the data package used to scan the message.
- The headers must include the hits fired by the engine and the spam probability assigned to the message.

Note: Spam is defined as scoring greater than or equal to 50% and less than 50% for not spam, even if your application defines these differently. Sophos requires that all host applications that wish to report spam/not-spam to Sophos use the 50% mark as the spam/not-spam dividing line. Any reports of is-spam/not-spam made using a different dividing line will be ignored, and may lead to all reports from the application being ignored.

4 Anti-Spam Engine Reference

This section provides a detailed reference to the Anti-Spam Engine API.

4.1 Event Codes

The `PMX_EVENT` codes are used in the `PMX_ENGINE::scan_message()` on page 33 callback. They indicate how the variable parameters are to be interpreted.

`PMX_EV_FOUND_FEATURE`	The engine reports that a rule or condition has been triggered. Parameter 1 is of type `PMX_TYPE_STRING` on page 28 and contains the feature name. Parameter 2 is normally not used, but some plugins pass a `PMX_TYPE_STRING` parameter containing extra information about why the feature fired.
`PMX_EV_FUR`	The engine determines the IP of the first untrusted relay by scanning the message headers and identifying the first connecting host that is not in its list of trusted relays. This event code only takes effect if the `core.event.fur` attribute has been specified and set to "1". For more information, see “Core Attributes” in the Anti-Spam Engine Attributes section. In order for this data to be useful to SophosLabs, it is recommended that you append it to an informational header, so that the IP address of the first untrusted relay is included in all false positives and false negatives submitted to Sophos. This header should follow the format shown below: `X-PMX-Version: product-name, Antispam-Engine: 2.6.1.350677, Antispam-Data: 2009.2.13.62815, FUR: 1.1.1.1` See also: `plugin.net.trusted-relays` under "net Attributes" in the Anti-Spam Engine Attributes section.

PMX_EV_SPAMPROB	The engine generates this event when it has determined the probability of the message being spam. Parameter 1 will be of type `PMX_TYPE_DOUBLE` on page 28 containing a probability between 0.0 and 1.0. Parameter 2 is not used.

4.2 Type Codes

Some `PMX_ENGINE` APIs use dynamically typed arguments, these are:

- `PMX_ENGINE::add_attribute()` on page 31 method
- callback of the `PMX_ENGINE::enumerate_attributes()` on page 32 method
- callback of the `PMX_ENGINE::scan_message()` on page 33 method

At the C level, dynamically typed values are always represented by 3 variables:

```
PMX_TYPE type, void *value, size_t size
```

The 'value' variable contains:

PMX_TYPE_NULL	undefined, value is not used
PMX_TYPE_CALLBACK	pointer to callback function
PMX_TYPE_DOUBLE	pointer to double
PMX_TYPE_INTEGER	int (not a pointer to an int)
PMX_TYPE_STRING	pointer to string, “size” contains length
PMX_TYPE_MESSAGE	`PMX_MESSAGE` object (is already a pointer)
PMX_TYPE_HASH	`PMX_HASH` object (is already a pointer). The `PMX_TYPE_HASH` interface object is required to specify trusted relays.
PMX_TYPE_WEIGHTS (deprecated)	`PMX_WEIGHTS` object (is already a pointer) **Note:** The `PMX_TYPE_WEIGHTS` interface object is not required for basic SDK usage. This object is provided for backwards compatibility only.

4.3 PMX_* Objects

All PMX_* objects start with a pointer to their vtbl. This vtbl may be shared by all instances of the same implementation. Access to the object is only through the vtbl methods.

4.4 Callbacks

The anti-spam engine makes use of callback functions in various places. All callbacks take a void *host as the first parameter. This parameter is not used by the engine at all and is only passed through to allow the hosting application to pass data to the callback. If the hosting application uses multiple instances of the engine simultaneously, the host parameter can be used to identify the corresponding engine object to the callback.

It is always valid to specify NULL for the callback if you do not want to be notified about the corresponding events.

The callbacks of the enumerate_*() methods return a boolean value: if it is 0, then enumeration is aborted immediately. Otherwise enumeration continues until it runs out of values. The enumerate_*() methods return the number of elements enumerated. If the enumeration is aborted this way, then the return value only reflects the number of elements actually visited. If an exception occurs during execution of the enumeration, the enumerator returns a count of -1, even if the callback function has already been called.

Any data passed to a callback function is only valid for the duration of the callback. If you want to preserve the data for later access, then the data must be copied from inside the callback.

4.5 Exported Functions

The anti-spam engine library contains the following externally visible entry points:

pmx_load()	pmx_load() must be called to initialize the library. The reserved parameter is only used on UNIX and must be NULL on Windows.
pmx_unload()	pmx_unload() should be called after all PMX_ENGINE on page 31 objects have been destroyed to free up any remaining resources the library may be using.
pmx_create_engine()	The pmx_create_engine() function creates an instance of the PMX_ENGINE on page 31 object. It takes a PMX_ERROR_CB callback that will be invoked whenever the engine runs into problems. The level parameter should be one of the following strings: DEBUG, INFO, NOTICE, WARNING, ERR, CRIT, ALERT or EMERG. The

	`error` parameter is a freeform text description of the error condition. It is not terminated by a newline character. Whenever the engine throws an exception, the error callback is invoked with an `ALERT` entry containing the exception message. The engine function then returns with an error (normally `PMX_ERR_SYS`).
`pmx_append_to_message()`	The `pmx_append_to_message()` function adds data to the default implementation of the `PMX_MESSAGE` on page 34 object. It will not work with any other implementation of the `PMX_MESSAGE` object. This function is not part of the `pmx_message_vtbl` because the engine does not modify the message object, and there is no specific paradigm that governs how the message objects should be created or updated. `pmx_append_to_message()` should not be called after any member functions have been invoked on the `PMX_MESSAGE` object.

4.6 PMX_ENGINE Methods

`destroy()`	`void (*destroy)(PMX_ENGINE e);` The `destroy()` method should be called to release all resources held by the `PMX_ENGINE` object. This destroys the embedded Perl interpreter. If the engine has been used to instantiate any `PMX_MESSAGE` on page 34, `PMX_HASH` or `PMX_WEIGHT` objects, then those must be destroyed prior to the destruction of the `PMX_ENGINE` object.
`create_message()`	`PMX_ERROR (*create_message)(PMX_ENGINE e, PMX_MESSAGE *m);` Creates an instance of the default implementation of the `PMX_MESSAGE` on page 34 object. The `pmx_append_to_message()` on page 30 function can be used to add the actual message content to the object.
`add_attribute()`	`PMX_ERROR (*add_attribute)(PMX_ENGINE e, const char *name, PMX_TYPE t, const void *value, size_t sz, int replace);` Adds an attribute that is used to configure the engine and its plug-ins. The `replace` parameter indicates that all previous values for this parameter should be discarded. If the `replace` flag is `0` and the attribute is already set, the value will be added to a list of values for that attribute. (Certain attributes can have multiple values).
`del_attribute()`	`PMX_ERROR (*del_attribute)(PMX_ENGINE e, const char *name, int all);`

	Deletes the specified attribute. If the `all` flag is set to `0`, then only the first entry in the list of attribute values is deleted. If the `all` flag is set to a non-zero value, then all the values for that attribute will be deleted.

`enumerate_attributes()`	`int`
	`(*enumerate_attributes)(PMX_ENGINE`
	` e, const char *name,`
	
	`PMX_ATTRIBUTE_CB cb, void *host);`
	
	The engine (and its plug-ins) can be configured using attributes. Some attributes can have multiple values (all of the same `PMX_TYPE`).
	
	The `replace` parameter to `add_attribute()` indicates that all previous values for this parameter should be discarded.
	
	If the `all` flag to `del_attribute()` is 0, then only the first entry in the list of attribute values is deleted.
	
	If an attribute of type `PMX_TYPE_MESSAGE` on page 28, `PMX_TYPE_HASH` on page 28 or `PMX_TYPE_WEIGHTS` on page 28 is added to the engine, then that object becomes owned by the engine and should no longer be accessed by the hosting application. The `del_attribute()` method invokes the destructors as necessary.

`load_plugin()`	`PMX_ERROR`
	`(*load_plugin)(PMX_ENGINE e,`
	`const char *plugin);`
	
	Loads the specified spam detection plug-in, enabling it for use.

`enumerate_loaded_plugins()`	`int`
	`(*enumerate_loaded_plugins)(PMX_ENGINE`
	` e, PMX_PLUGIN_CB cb, void`
	`*host);`
	
	Invokes the callback `cb` for each plug-in that

	has been loaded.
`enumerate_plugins()`	`int (*enumerate_plugins)(PMX_ENGINE e, PMX_PLUGIN_CB cb, void *host);` The engine library includes a number of spam detection plug-ins. Only the plug-ins loaded into the engine are used to determine spam features and probabilities. A feature group must be loaded before its attributes are set or modified.
`load_antispam_data()`	`PMX_ERROR (*load_antispam_data)(PMX_ENGINE e);` Load antispam data from `core.datadir` on page 44. This auto-loads all plug-ins required to run the data, declaring their attributes immediately. The application must set the `core.datadir` attribute before calling this method, so that the engine can find the `asdb.antispam` file.
`scan_message()`	`PMX_ERROR (*scan_message)(PMX_ENGINE e, PMX_MESSAGE m, PMX_WEIGHTS w, PMX_SCAN_CB cb, void *host);` The `scan_message()` method parses the message in the `PMX_MESSAGE` on page 34 object, detecting spam features using the loaded plug-ins. Found features are reported to the `PMX_SCAN_CB` callback. They are accumulated with weights from `PMX_WEIGHTS` to determine a final spam probability rating, which is also reported to the callback. The callback can record these events for future use after `scan_message()` has returned.
`compile()`	`PMX_ERROR (*compile)(PMX_ENGINE e);`

	The `compile()` method reloads all plug-ins and clears or refreshes any internally cached values. If any plug-in fails to initialize correctly, compile generates log messages describing the failure. If the `core.compile` on page 44 attribute is set to `true`, calling `scan_message()` causes the engine to automatically call this method; failures are ignored internally.
`create_hash()`	`PMX_ERROR (*create_hash)(PMX_ENGINE e, PMX_HASH *h, const char *file, int readonly);` The `create_hash()` method creates an instance of the default implementation of the `PMX_HASH` object. The current implementation uses a file to store the table. **Note:** It is not safe to update the table while other processes or threads may be reading the same table.

4.7 PMX_MESSAGE Methods

Many of the `PMX_MESSAGE` accessor methods return parts of the message. The returned data may be a copy of the original message data, or just a pointer and a length into the internal message buffer. For this reason, the returned data cannot be guaranteed to be `'\0'` terminated. The engine always treats all returned data as read only. It also calls the message's `free()` method on the returned data, just in case the implementation returns a copy that must be deallocated, which is the case for the default Perl implementation.

free()	`void (*free)(PMX_MESSAGE m, char *chunk);` The `chunk` returned by the various message methods must be freed by the caller before the `PMX_MESSAGE` object is destroyed.
get_message_size()	`PMX_ERROR (*get_message_size)(PMX_MESSAGE m, size_t *sz);` Returns the size of the raw message, in bytes.
get_message()	`PMX_ERROR (*get_message)(PMX_MESSAGE m, size_t offset, size_t len, const char **chunk, size_t *sz);` Returns a chunk of the raw message, starting at `offset`, of length `len` bytes. If `len` is zero, returns the rest of the message from `offset`. The chunk is returned in `chunk`, and the length of the string is returned in `sz`.
get_envelope_sender()	`PMX_ERROR (*get_envelope_sender)(PMX_MESSAGE m, const char **sender, size_t *sz);` Returns the envelope sender. The default implementation of the `PMX_MESSAGE` object does not maintain any envelope information.
enumerate_envelope_recipients()	`int (*enumerate_envelope_recipients)(PMX_MESSAGE m, PMX_RCPT_CB cb, void *host);` Invokes the callback `cb` for each recipient

	named.
`get_header_size()`	`PMX_ERROR (*get_header_size)(PMX_MESSAGE m, size_t *sz);` Returns the size of the RFC 822 header block, in bytes. Returns 0 if the raw header block is not available.
`get_header()`	`PMX_ERROR (*get_header)(PMX_MESSAGE m, size_t offset, size_t len, const char **chunk, size_t *sz);` Fetch a chunk of the raw RFC 822 header block, if available. If `len` is zero, returns the rest of the header block from `offset`.
`enumerate_headers()`	`int (*enumerate_headers)(PMX_MESSAGE m, const char *header, PMX_HEADER_CB cb, void *host);` Invokes the callback `cb` for each header named `header`. If `header` is `NULL`, invokes `cb` on every header.
`get_body_size()`	`PMX_ERROR (*get_body_size)(PMX_MESSAGE m, size_t *sz);` Returns the size of the body in bytes.
`get_body()`	`PMX_ERROR (*get_body)(PMX_MESSAGE m, size_t offset, size_t len, const char **chunk, size_t *sz);` Returns a chunk of the raw body starting at `offset` with length `len` bytes. If `len` is zero,

	returns the rest of the body.

`get_decoded_body()`	`PMX_ERROR (*get_decoded_body)(PMX_MESSAGE m, size_t offset, size_t len,`
	
	`const char **chunk, size_t *sz);`
	
	Returns a chunk of the decoded body starting at `offset` with length `len` bytes. If `len` is zero, returns the rest of the decoded body.
	
	The decoded body undoes base-64 and quoted-printable encodings.

`enumerate_parts()`	`int (*enumerate_parts)(PMX_MESSAGE m, PMX_PART_CB cb, void *host);` Invokes the callback `cb` for each subpart of the message object. Each subpart is represented by its own `PMX_MESSAGE` object passed to the callback. The `part` object is only valid during the lifetime of the callback. It cannot be saved.
`get_relay()`	`PMX_ERROR (*get_relay)(PMX_MESSAGE m, const char **ip, size_t *sz);` Returns the IP address of the connecting relay as a string. The default message implementation always returns `PMX_ERR_NOATTR`, which the engine interprets as a signal not to perform relay checks.

`get_relay_hostname()`	`PMX_ERROR (*get_relay_hostname)(PMX_MESSAGE m, const char **hostname,`
	
	`size_t *sz);`
	
	Returns the hostname of the connecting relay as a string, or the empty string (a size of zero) if the relay has no hostname. If implemented,

	the result of performing a reverse DNS lookup on the connecting relay's IP address (the one returned from the `get_relay()` method). The SDK treats this hostname as unforgable, thus suitable for whitelisting and blacklisting. The default message implementation always returns `PMX_ERR_NOATTR`, which the engine interprets as a signal not to perform hostname checks or its own DNS lookup.
`get_helo()`	`PMX_ERROR (*get_helo)(PMX_MESSAGE m, const char **helo, size_t *sz);` Returns the SMTP `HELO` line from the SMTP transaction as a string. The default message implementation always returns `PMX_ERR_NOATTR`, which the engine interprets as a signal not to perform `helo` checks.

4.8 PMX_MESSAGE_WRITABLE Methods

The `PMX_MESSAGE_WRITABLE` interface can be used with messages returned from the engine's `create_message()` on page 31 method. This interface is not required for basic SDK usage.

`set_envelope_sender()`	`PMX_ERROR (*set_envelope_sender)(PMX_MESSAGE_WRITABLE m,`
	`            const char *sender,`
	`            size_t len);`
	
	Sets the message's envelope sender. The string is copied into the message object. The `get_envelope_sender()` on page 35 method subsequently returns a pointer to the copied string.

`add_envelope_recipient()`	`PMX_ERROR (*add_envelope_recipient)(PMX_MESSAGE_WRITABLE m,`
	`            const char *recip,`
	`            size_t len);`
	
	Adds an envelope recipient. The string is copied into the message object. The `enumerate_envelope_recipients()` on page 35 method subsequently reports the copied string.

`set_relay()`	`PMX_ERROR (*set_relay)(PMX_MESSAGE_WRITABLE m,`
	`            const char *relay,`
	`            size_t len);`
	
	Sets the relay IP address as a string. The string is copied into the message object. The `get_relay()` on page 37 method subsequently returns a pointer to the copied string.

`set_relay_hostname()`	`PMX_ERROR (*set_relay_hostname)(PMX_MESSAGE_WRITABLE m,`
	
	`const char *hostname,`
	
	`size_t len);`
	
	Sets the relay hostname as a string. The string

	is copied into the message object. The `get_relay_hostname()` on page 37 method subsequently returns a pointer to the copied string.
`set_helo()`	`PMX_ERROR (*set_helo)(PMX_MESSAGE_WRITABLE m, const char *helo, size_t len);` Sets the `SMTP HELO` string. The string is copied into the message object. The `get_helo()` on page 38 method subsequently returns a pointer to the copied string.
`append()`	`PMX_ERROR (*append)(PMX_MESSAGE_WRITABLE m, const char *data, size_t len);` Appends data to the message. This method is called internally by `pmx_append_to_message()` on page 30.
`set_limit()`	`PMX_ERROR (*set_limit)(PMX_MESSAGE_WRITABLE m, PMX_MESSAGE_LIMIT what, size_t limit);` Sets internal limits in the message object. There are several limits to tweak:

`PMX_MLIMIT_BYTES`	The maximum number of bytes to allocate for raw MIME data. If more data than this is allocated, further calls to `append()` are silently ignored. Default: 64KB

PMX_MLIMIT_PARTS	The maximum number of MIME parts to parse in the message object. If more MIME parts than this are encountered, MIME parsing is aborted. The value returned from `enumerate_parts()` on page 37 never exceeds this limit. Default: 128
PMX_MLIMIT_SIBLINGS	The maximum number of MIME sibling parts allowed anywhere in the MIME tree. If any part with more subparts than this is encountered, MIME parsing is aborted. Default: 50

These parameters should not need modification as they are designed to limit the amount of work done by a high performance mail scanning application. The limits can be lowered further to increase scanning performance on large messages.

`preload()`

```
PMX_ERROR
(*preload)(PMX_MESSAGE_WRITABLE,
 size_t maxbytes);
```

The `preload()` method forces the message object to parse at least `maxbytes` bytes from its input stream. If EOF or the watermark is found before `maxbytes`, the object stops reading. This can be used to guarantee that a certain amount of data is ready for the engine before calling `scan_message()` on page 34. In a multithreaded application, this may be more efficient, as the scanning thread will not block

	for IO. The `preload()` method is incompatible with the `append()` method. Use it with the streaming input methods `read_from_file()` or `read_from_func()`. This method should not be called after any `vtbl` member functions have been invoked on the `PMX_MESSAGE` object.
`read_from_file()`	`PMX_ERROR (*read_from_file)(PMX_MESSAGE_WRITABLE m, const char *file);` The `read_from_file()` method set the message to read MIME on demand from the specified file. The file is opened and remains open (using up a file descriptor) until the message is destroyed, the watermark is hit, or EOF is encountered. The `read_from_file()` method is incompatible with the `append()` method. Use one or the other, but not both. This method should not be called after any `vtbl` member functions have been invoked on the `PMX_MESSAGE` object.
`read_from_func()`	`PMX_ERROR (*read_from_func)(PMX_MESSAGE_WRITABLE m, void *host, PMX_MESSAGE_READER, PMX_MESSAGE_READER_FREE);` The `read_from_func()` method sets the message to read MIME on demand from a callback function. The application provides a reader function that the message object will call whenever it needs more data. The reader callback is passed the opaque pointer provided to `read_from_func()` by the calling application. It is also passed a buffer; the size of the buffer, and, as an out parameter, the number of bytes stored in the buffer. The reader is expected to return `PMX_MESSAGE_EOF` when the stream is exhausted.

	The free function is provided so that resources used by the host can be freed when the stream is exhausted. This is called when the message object is destroyed, the watermark is hit, or the reader returns EOF. The `read_from_func()` method is incompatible with the `append()` method. Use one or the other, but not both. This method should not be called after any `vtbl` member functions have been invoked on the `PMX_MESSAGE` object.

5 Anti-Spam Engine Attributes

This document describes the Sophos Anti-Spam SDK configuration attributes.

5.1 Core Attributes

`core.compile`	If true, calling `scan_message()` on page 33 causes the engine to automatically compile the anti-spam data in the directory specified by `core.datadir` on page 44. Failures are ignored internally. Type: `PMX_TYPE_INTEGER` on page 28 Default: 1
`core.datadir`	This attribute specifies the location (for example, `../datadir`) of data files that the anti-spam engine uses. **Important:** This is a required attribute. Type: `PMX_TYPE_STRING` on page 28 Default: None
`core.data-version`	This attribute returns the version of the anti-spam data that the engine currently uses. If the data version cannot be determined, <unknown> is returned. If the engine has been loaded but `load_antispam_data()` on page 33 has not been called, the `core.data-version` attribute is not defined. If the anti-spam data is loaded, it returns a string containing the data version. Type: `PMX_TYPE_STRING` on page 28 Default: none
`core.event.fur`	This attribute must be specified and set to "1" in order for the PMX_EV_FUR event code to be reported to the PMX_SCAN_CB callback. For information about PMX_EV_FUR, see "Event Codes" in the Anti-Spam Engine Reference section.

	The discovery of first untrusted relays will vary depending on whether and how you have configured the `plugin.net.trusted.relays` attribute. For more information, see "net Attributes" in the Anti-Spam Engine Attributes section. See also: `plugin.net.trusted-relays` under "net Attributes" in the Anti-Spam Engine Attributes section. Type: `PMX_TYPE_INTEGER` on page 28 Default: 0
`core.local-tests-only`	If true, plug-ins that would normally connect to the internet to perform tests will skip such tests. Type: `PMX_TYPE_INTEGER` on page 28 Default: 0
`core.max-bytes-scanned`	The maximum number of bytes the engine processes per message part. Regardless of the overall message size and the number of bytes sent to the engine, only the number of bytes specified in this attribute are scanned, which provides a predictable processing time. This attribute only limits CPU usage during scanning; entire messages (or the parts of messages sent to the engine) are stored in memory. Type: `PMX_TYPE_INTEGER` on page 28 Default: 10240
`core.schema-version`	This attribute returns the data schema version for the current anti-spam engine. Type: `PMX_TYPE_STRING` on page 28 Default: None
`core.statistics-directory`	Specifies a directory into which the engine statistics will be written. This same directory must be specified to the feedback tool when it is used to upload statistics to Sophos. Note that if this directory is not specified, then the engine statistics will be written to the '`stats`' subdirectory of the location specified by the `core.datadir` on page 44 attribute.

	Type: PMX_TYPE_STRING on page 28 Default: none
core.stop-scan	If true, the engine stops scanning once a message is definitively determined to be either spam or not spam (that is, has a very high positive or negative spam score). The resulting cumulative spam score may differ from the results of a full scan. Excluded rules will not appear in anti-spam reports and headers. Type: PMX_TYPE_INTEGER on page 28 Default: 0
core.version	This attribute returns the version of the current anti-spam engine. Type: PMX_TYPE_STRING on page 28 Default: None

5.2 “net” Attributes

plugin.net.internal-hosts	A PMX_TYPE_HASH on page 28 object used by the "net" plugin to exempt internal hosts from network-based tests. Hosts in the IANA unroutables are always exempted:
	
	`10.0.0.0/8`
	`127.0.0.0/8`
	`172.16.0.0/12`
	`192.168.0.0/16`
	
	If hosts in other domains should be considered "internal", they should be added to this hash.
	
	Type: PMX_TYPE_HASH on page 28
	
	Default: none

plugin.net.trusted-relays	A PMX_TYPE_HASH on page 28 object used by the "net" plug-in to skip over certain relays in the 'Received' header chain as it tries to find the first external relay. Certain network tests must only be run on the first external relay.

	If this attribute is not provided, the "net" plug-in assumes that the first external IP address is the relay. See also: PMX_EV_FUR under “Event Codes” in the Anti-Spam Engine Reference section, and `core.event.fur` under “Core Attributes” in the Anti-Spam Engine Attributes section. Type: `PMX_TYPE_HASH` on page 28 Default: none
`plugin.net.dns-servers`	Specifies the default DNS servers. If no default DNS servers are specified in the `plugin.net.dns-servers` attribute, the net plug-in obtains the system DNS servers from the Windows registry. On UNIX systems, the DNS servers are referenced from `/etc/resolv.conf` if they're not provided with this attribute. Type: `PMX_TYPE_STRING` on page 28 Default: takes the DNS server settings from the system default
`plugin.net.dns-timeout`	The maximum number of seconds to wait for a DNS reply. **Note:** For backwards compatibility reasons, if `plugin.net.dns-timeout` is not set, the attribute `plugin.re.dns-timeout` is used. Type: `PMX_TYPE_INTEGER` on page 28 Default: 2
`plugin.net.dns-retry`	The number of times to retry a failed DNS lookup. **Note:** For backwards compatibility reasons, if `plugin.net.dns-retry` is not set, the attribute `plugin.re.dns-retry` is used. Type: `PMX_TYPE_INTEGER` on page 28 Default: 2

`plugin.net.skip-dns-checks`	If set to 1, this disables all tests using DNS. **Note:** It is not recommended to skip DNS checks Type: `PMX_TYPE_INTEGER` on page 28 Default: 0

5.3 “dnsbl” Attribute

`plugin.dnsbl.relay-tests-only`	If set to `1`, this disables DNSBL tests for all IP addresses except the first external relay. Type: `PMX_TYPE_INTEGER` on page 28 Default: 0 **Note:** This option is disabled by default, because it requires that either (a) there are no trusted relays; or (b) that the `plugin.net.trusted-relays` on page 46 attribute is set.
`plugin.dnsbl.skip-rbl-tests`	If set to 1, this disables all tests using DNS. Type: `PMX_TYPE_INTEGER` on page 28 Default: 0 **Note:** It is not recommended to skip RBL checks.

5.4 SXL Attributes

`plugin.sxl.enable`	Controls whether the SXL functionality is enabled or disabled. This attribute may be used to test SXL before it is generally enabled. Type: `PMX_TYPE_INTEGER` on page 28 Default: none
`plugin.sxl.hashed-uris`	Prevents the anti-spam engine from making DNS queries on actual URIs; instead they will first be hashed. The disadvantage is that

	SophosLabs will be unable to analyze these URIs. For this reason, it is not recommended that you enable this feature. However, organizations with privacy policies that protect URIs and domains included in email messages may require this feature. Type: `PMX_TYPE_INTEGER` on page 28 Default: 1