Security Audit of the Secure List Server
Part II

Guus Sliepen <guus@sliepen.org>

May 24, 2013

Abstract

The subject of this audit is Mailman 2.1.12 patched with the pgp-smime patch of 2009-04-02, and follows the progress since the previous audit. The changes in the list of known, open issues are reviewed. A test setup was created and the code and its behaviour were reviewed. A number of new issues were found in the Secure List Server (SLS). Most importantly, although the default settings are reasonably safe, if the administrator enables more list options there are many possibilities for decrypted messages to be bounced, forward or stored without being re-encrypted. The developers should reduce the number of options available to list administrators, and continue to work on strict integrity and confidentiality enforcement.

1 Introduction

1.1 Subject of the security audit

The subject of this security audit is the Secure List Server (SLS), which consists of the Mailman mailing list server augmented with the pgp+smime patch which allows Mailman to handle emails which are signed and/or encrypted with OpenPGP or S/MIME in a meaningful way.

Part I of the audit covered a coarse code audit, and was focused more on usage aspects of SLS. After reviewing the TODO list and NEWS in section **??**, the flow of mail through the SLS codebase and the issues found therein will be looked at in detail in section **??**. In section **??** I will remark on the impliciations of the recent vulnerabilities found in the SHA-1 algorithm for SLS. In section **??** I will give recommendations for future work on SLS.

1.2 Test setup

To test the functionality and behaviour of SLS, the following test setup was created:

- Asus EeePC 901 with 2 GB RAM and 20 GB storage
- Debian unstable (last update on 2009-03-25)
- Mailman 2.1.12 patched with `mailman-2.1.12-pgp-smime_2009-04-02.patch`
- Postfix 2.5.5-1.1 mail server
- Mutt 1.5.18-6 mail client
- Lighttpd 1.4.19-4 light-weight web server
- GnuPG 1.4.19-5

- OpenSSL 0.9.8g-15
- Python-GnuPGinterface 0.3.2-9

2 Review of TODO list and NEWS since 2008-07-03

According to the `NEWS.PGP-SMIME` file, the following items have changed since the patch from 2008-07-03, which was reviewed in part I of the audit:

- A test suite has been added.

  At this moment, the test suite consists of a shell script that creates two sets of three mailing lists: one set for PGP and one for S/MIME, each set containing three lists with different security levels. It subscribes one member to each list, and can send a number of prepared messages to each list.

  There are no prepared messages included in SLS however, so one has to create them oneself. The test suite also does not check the result of sending the messages to the list. The test suite is still far from an automated tool for checking regressions.

- Names and description of security settings have been changed.

  Some previously confusing options for allowing or requiring encryption and/or signatures have been renamed to be more consistent, and the descriptions of those options in the web GUI have been clarified. This will reduce the chance of an administrator misconfiguring secure lists.

  The problem that there are two sets of options, one for PGP and one for S/MIME, remains. Again, it would be better to have one set of options for encryption and signing that is independent of the protocols used for them.

- Members cannot change their public key once set.

  This prevents an attacker who can get the password, which is assumed to be much easier than getting the private key, to replace a subscriber's public key with its own.

- Bounces, moderator notifications and logs of encrypted emails no longer contain the decrypted bodies.

  Previously, a decrypted copy of an encrypted email was sent without re-encryption to moderators, or sent to the syslog, which could result in unintended information disclosure. SLS now only sends the headers of the original encrypted email in those cases.

- From address must now match one of the signature's uids.

  Since SLS strips the original signature from emails and adds its own, the other recipients can only determine the original sender by looking at the `From` header. SLS now rejects incoming emails whose From-address does not match an uid of the PGP key which signed the post.

The file `TODO.PGP-SMIME` has been updated, the following items have changed:

0024 "*If a post is properly signed, accept it, no matter whether the From-address is subscribed and no matter the sender moderation policy. However: Although that is possible and perhaps desirable, one should remember that only the body of an email is signed and/or encrypted, but not the headers. If a subscriber is allowed to change the From header at will, he can try to impersonate another person when sending an email to the list. It is best to restrict the contents of the From header to the email address(es) listed in the subscriber's public key.*"

SLS now indeed restricts the contents of the `From` header, so this item can be removed from the TODO list.

0050 "*Make sure posts get encrypted and signed if needed. Perhaps it is best determine whether an incoming email as signed or encrypted, and mark this somewhere in its headers, such that the marking does not get removed while the email is being processed by SLS. When sending outgoing emails, preferably right before the email is sent to the SMTP server, it should be verified that if the message is marked signed, the outgoing email is indeed signed. The same goes for encryption (and PGP).*"

SLS now adds a `X-Mailman-SLS-decrypted` header to emails that have been decrypted. It is however only used to strip the body from emails that are being bounced in plaintext to the list administrator.

0059 "*Don't use openssl and GnuPGInterface, but pyme. KMail is said to use GPGME for S/MIME. Study its source. SMScrypto.py from http://smallsister.org/git/SmallMail.git implements crypto in python using pyme. Study its source.*"

Indeed, if there is an external library that can encrypt, decrypt, sign and verify emails, and it provides all the functionality that SLS needs, it is better to use that instead of keeping a (different) private implementation.

Using pyme would remove the need for SLS to use temporary files and to manually start external programs such as `gpg` and `openssl`. This removes potential security problems from the SLS code.

3 Code audit, part II

In this second part of the code audit, I have followed the lifetime of an email, as it is processed by SLS. I have done a line-by-line review of the points in the code where emails are decrypted and re-encrypted. First I will give an overview of the various subsystems of mailman and how email flows through them. Then I will look at the possible issues with the handling of decrypted, but not yet re-encrypted, emails by SLS. Finally I will describe my findings of the code dealing with decryption and encryption.

3.1 Mail flow

When a mailing list is created with SLS, a list of email aliases is updated. This list of aliases can be read by the mail transfer agent (MTA), and tells the MTA what to do with emails for all the relevant mailing list addresses. The email aliases generated by SLS for the Postfix MTA look like this:

```
listname:             "|/var/lib/mailman/mail/mailman post listname"
listname-admin:       "|/var/lib/mailman/mail/mailman admin listname"
listname-bounces:     "|/var/lib/mailman/mail/mailman bounces listname"
listname-confirm:     "|/var/lib/mailman/mail/mailman confirm listname"
listname-join:        "|/var/lib/mailman/mail/mailman join listname"
listname-leave:       "|/var/lib/mailman/mail/mailman leave listname"
listname-owner:       "|/var/lib/mailman/mail/mailman owner listname"
listname-request:     "|/var/lib/mailman/mail/mailman request listname"
listname-subscribe:   "|/var/lib/mailman/mail/mailman subscribe listname"
listname-unsubscribe: "|/var/lib/mailman/mail/mailman unsubscribe listname"
```

Where `listname` is replaced with the name of the mailing list. These aliases instruct the MTA to pipe the emails it through the `/var/lib/mailman/mail/mailman` command. That command in turn is a simple wrapper that runs a Python script from `/var/lib/mailman/scripts/` with the

Figure 1: Representation of the Incoming and Outgoing queues. Encryption and decryption happens in the orange boxes, red boxes process decrypted messages, red arrows signify decrypted messages being sent in plaintext. Several handlers in the Incoming queue leak plaintext to local storage or to the Internet.

same name as the first argument. So in effect, posts to the mailing list address without any suffix will be piped through `/var/lib/mailman/scripts/post listname`.

In turn, these Python scripts themselves only put the messages in so-called switchboard queues. Messages posted to the mailing list address without a suffix will be put in the Incoming queue, which normally resides in `/var/lib/mailman/qfiles/in`.

For each switchboard a `/var/lib/mailman/bin/qrunner` process is running. These processes are normally started at boot time by the `/etc/init.d/mailman` script. Each `qrunner` has a loop in which it dequeues messages from its switchboard. Each dequeued message is handled by the `_dispose()` function. For the Incoming queue, it is `IncomingRunner._dispose()`. This function in turn puts the message in a message processing pipeline.

The pipeline consists of a number of handler objects, each of which have a `process()` function which deals with the email. Each handler can change the message before it is passed on to the next handler. A handler can also raise an exception, in which case the rest of the pipeline will be skipped and the message will be either discarded, rejected, or held. By default, the pipeline consists of the following handlers, in order of execution:

`SpamDetect`, `Approve`, `Replybot`, `Moderate`, `Hold`, `MimeDel`, `Scrubber`, `Emergency`, `Tagger`, `CalcRecips`, `AvoidDuplicates`, `Cleanse`, `CleanseDKIM`, `CookHeaders`, `ToDigest`, `ToArchive`, `ToUsenet`, `AfterDelivery`, `Acknowledge`, `ToOutgoing`.

The pipeline will be described in more detail later. If no exception is raised by any of the handlers, then the last handler, `ToOutgoing`, will enqueue the message in the Outgoing switchboard. The Outgoing queue is handled by `OutgoingRunner._dispose()`. By default, it processes the message with just the `SMTPDirect` handler.

Decryption and encryption is taking place in the `Moderate` and `SMTPDirect` handlers respectively. This means that decrypted content is handled by all the handlers in-between, and SLS must take special care that decrypted content is not inadvertently leaked in these handlers. Even so, due to the flexibility of the pipeline architecture, an administrator or an add-on package can easily change existing handlers or insert new handlers.

3.1.1 Incoming message pipeline

`SpamDetect` This handler uses regular expressions on the message header to classify a message as spam or not. It can either discard, reject or hold spam. It is configured using the `privacy/spam` page on the list admin site.

`Approve` This handler checks if the message is pre-approved or if the list admin has added an approval header. If so, the message tag 'approved' is set. The message is otherwise unaltered and will always pass to the next handler.

`Replybot` This handler will send an automated reply in response to an incoming message. It is configured using the `autoreply` admin page. The incoming message itself is unaltered and will always pass to the next handler.

`Moderate` This handler originally only discarded, rejected, held, or allowed a message based on whether the sender was a list member or not, or if Mailman was explicitly configured to discard, reject, hold or allow messages from that sender. It was configured using the

`privacy/sender` admin page. In SLS, message decryption and signature verification has been spliced into this handler. Decryption and verification is done first, then messages are rejected, held or approved depending on the state of the encryption and signature, as configured using the `privacy/gpg` and `privacy/smime` admin pages. After that the original moderation rules are applied. Note that if messages are held in this and later handlers, they have already been decrypted.

A detailed description of the decryption and signature verification code is given in section **??**.

`Hold` Unless the message has already been tagged as 'approved', this handler holds messages if they contain administrativia that were erroneously sent to the mailing list address, if it has too many recipients, if it contains suspicious headers or if it is too large, as configured using some of the options on the `general` admin page.

`MimeDel` This handler can remove MIME parts as configured on the `contentfilter` admin page, or can discard, reject or forward messages to the list owner if it contains forbidden parts.

`Scrubber` If configured so on the `nondigest` admin page, this handler "scrubs" attachments from messages and stores them in a web-accessible archive. The attachment is then replaced with a URL to the archived copy.

`Emergency` If the "Emergency moderation of all list traffic" option on the `general` admin page is set, this handler will hold all messages unless they have been approved.

`Tagger` This handler uses regular expressions to categorise messages into "topic buckets", as configured on the `topics` admin page. Messages are always passed unaltered, except for the possible addition of an `X-Topics` header if it matches any topics.

`CalcRecips` This handler determines which list members to forward this message to, based on their individual settings for delivery.

`AvoidDuplicates` This handler culls duplicates from the list determined in the `CalcRecips` handler, if these recipients have configured duplicate message removal.

`Cleanse` This handler removes certain headers that should not occur in outgoing messages, such as approval passwords. If the list has been configured as anonymous on the `general` admin page, it will also remove the original `From`, `Reply-To`, `Sender` and `X-Originating-Mail` headers, and replace the first two with the mailing list address.

`CleanseDKIM` This handler removes DomainKeys headers.

`CookHeaders` This handler adds `X-BeenThere`, `X-Mailman-Version`, `Precedence`, `Reply-To`, `List-Id` and some other mailing list related headers to the message, and possibly changes the `Subject` header to add a list prefix.

`ToDigest` If the mailing list is configured to be digestible on the `digest` admin page, this appends the message to the list's `digest.mbox` file. If the digest file exceeds the size threshold, it creates a digest email and puts it in the Virgin queue, which in turn pipes it through the `CookHeaders` and `ToOutgoing` handlers.

`ToArchive` If the message is not a digest, does not contain a `X-No-Archive` or `X-Archive: no` header, and the mailing list is configured on the `archive` admin page to archive messages, this handler will put a copy of the message in the Archive queue, which in turn passes it to the archiver.

`ToUsenet` If the list had been configured on the `gateway` admin page to send copies to a Usenet group, this handler will put a copy of the message in the News queue, which in turn will forward the message via NNTP.

`AfterDelivery` This handler updates the statistics about the last post time and number of posts received on the list.

`Acknowledge` This handler sends an acknowledgement to the sender if he has configured so.

`ToOutgoing` This handler puts the message in the Outgoing queue.

There are many handlers which can, if the mailing list is configured to do so, hold, reject, forward or otherwise store decrypted messages without re-encrypting them first. Unless measures are taken to keep the confidentiality of previously encrypted messages, SLS must either drop the message in those situations, send back a rejection notification with just the original headers, or just permanently disable features that interfere with the confidentiality of messages. This is a list of issues in the Incoming pipeline that should be addressed:

- Decrypted messages being held in the `Hold` handler.
- Rejected, held and forwarded messages in the `MimeDel` handler.
- Decrypted, scrubbed attachments being made web-accessible in the `Scrubber` handler.
- Messages being stored unencrypted for extensive periods and re-sent time-delayed in the `ToDigest` handler.
- Messages being archived unencrypted in the `ToArchive` handler.
- Messages being forwarded unencrypted to Usenet groups in the `ToUsenet` handler.

As a safeguard, the functions `BounceMessage()` and `HoldMessage()` should check for the presence of the `X-Mailman-SLS-decrypted` header and prevent decrypted message bodies from being stored or sent.

3.1.2 Outgoing message pipeline

SMTPDirect This handler sends a copy of the message to each of the recipients, possibly signed and/or encrypted as configured using the `privacy/gpg` and `privacy/smime` admin pages.

A detailed description of the signature generation and encryption code is given in section **??**.

3.2 Decryption and encryption

I will now describe the code dealing with the decryption, encryption and signature verification and generation in detail, pointing out issues where I found them.

3.2.1 `Moderate.process()`

Incoming posts to lists are handled by `process()` in `Moderate.py`. The variable `msg` contains the incoming message, and is modified in a number of places. First decryption of messages is tried:

- If the `gpg_post_encrypt` option is set, `decryptGpg()` is called.
  - If `gpg_post_encrypt` is set to `2`, and the message was not encrypted or if decryption failed, the message is rejected.

Afterwards `msg` contains the decrypted message.

- If the `smime_post_encrypt` option is set, `decryptSmime()` is called.
  - If `smime_post_encrypt` is set to `2`, and the message was not encrypted or if decryption failed, the message is rejected.

  Afterwards `msg` contains the decrypted message.

The fact that there are different settings for PGP and S/MIME can lead to strange, inconsistent behaviour:

- `gpg_post_encrypt` = `smime_post_encrypt` = `1`: Accepts mail that is S/MIME encrypted first and then PGP encrypted, or mail that is either S/MIME or PGP encrypted, but not mail that is first PGP encrypted and then S/MIME encrypted.
- `gpg_post_encrypt` = `smime_post_encrypt` = `2`: Accepts mail that is S/MIME encrypted first and then PGP encrypted, but rejects the reverse order and rejects mail that is either S/MIME or PGP encrypted.

There are only a few combinations of `gpg_post_encrypt` and `smime_post_encrypt` that make sense. As mentioned in part I of the audit, it would be better to have one option, `post_encrypt`, and allow PGP and/or S/MIME encrypted emails based on whether PGP and/or S/MIME keys have been set for the list.

The decryption functions also check signatures if present, but only for encrypted+signed messages. The next part in the code handles messages that are only signed:

- If `gpg_post_sign` is set, but the variable `signed` is not set:
  - If the message is of type `multipart/signed`, the last MIME part with type `application/pgp-signature` is used as the signature, and the last MIME part with any other type is used as the payload.
  - Else, if the message is of type `text/plain` and not multipart, it is assumed that the message is inline PGP, and the entire message is used as the payload.
  - The `gh.verifyMessage()` function is called.
- If `smime_post_sign` is set, but the variable `signedByMember` is not set:
  - The `sm.verifyMessage()` function is called.
- If `gpg_post_sign` is set:
  - If the variable `key_ids` is empty, hold or discard the message.
  - Else, check if any sender matches any uid of the key.
    - If not, hold or discard the message.
    - Else, if the signature belongs to any of the keyids of any list member, set `signedByMember`.
- If `smime_post_sign` is set, but the variable `signedByMember` is still not set:
  - Hold or discard the message.

Here also there are some issues:

- Just as with the `gpg_post_encrypt` and `smime_post_encrypt` options, there are many nonsensical combinations of `gpg_post_sign` and `smime_post_sign` possible.

- Although SLS now checks the sender against PGP key's uids, it only checks if *any* of the senders match any uid. This allows a list member to send a signed email with himself plus other addresses in the `From` header. Since the original signature is stripped, other mailing list members cannot know which of the senders signed and sent the email.
- For S/MIME, there are still no checks if the sender is the same as the signer. For X.509 certificates, the `emailAddress` field in the subject DN could be used in the same way as the uid of a PGP key. Another option would be to check only the key(s) of the sender instead of checking the signature against all members' certificates in `sm.verifyMessage()`.
- For `multipart/signed` messages, it is assumed by SLS that the body consists of exactly two parts, but this assumption is not *asserted*.

The last issue allows an attacker to reuse another member's post to send any message he wants. The variable `payload` is set to the last part that is not of type `application/pgp-signature`, and `signature` is set to the last part that is of that type. The function `gh.verifyMessage()` is then called with these variables. This means that an attacker can construct a `multipart/signed` message with four parts (see figure **??**): the first two with his own message and signature, which can be made with any key. The last two parts are copied from a valid message previously sent to the list. SLS accepts such a message if the `From` header matches any uid of the key of the copied message's signature.

Mutt, a widely used Mail User Agent (MUA), handlers `multipart/signed` messages with more than two parts in a different way. It expects the first part to be the message body, and the other parts to be signatures. It will honour the `Content-Length` header though, and will only read that amount of bytes from the message, so it is possible to make it read only the first two parts.

When given the attacker's four-part message, it will display the first part in-line, and will print a "Good signature from" message above it. If the other parts are not ignored due to the `Content-Length` header, it will display an additional "Bad signature" message for the second `application/pgp-signature` part.

SLS must either discard those parts it does not check, or reject the whole message.

3.2.2 `Moderate.decryptGpg()`

The `decryptGpg()` function tries to find the ciphertext to pass to the `gh.decryptMessage()` function. If the message is of type `multipart/encrypted`, it assumes there is one part of type `application/octet-stream` that contains the ciphertext. If there are more parts, it only uses the last of that specific type as the ciphertext, and will discard the other parts.

Although there is no information leak here, it would be better to assert that the message only contains one part, and otherwise treat it as unencrypted at all, to prevent silent removal of MIME parts.

3.2.3 `SMTPDirect.verpdeliver()`

Outgoing messages are ultimately handled by `verpdeliver()` in `SMTPDirect.py`. The following happens:

- If the `gpg_distrib_encrypt` option is set, try to encrypt the message for each recipient.
  - Also sign the message if `gpg_distrib_sign` is set.
  - If no key is known for the recipient or encryption fails, and `gpg_distrib_encrypt` = `1`, send plaintext.

```
From: Alice <alice@example.org>
To: list@example.org
Mime-version: 1.0
Content-Type: multipart/signed; micalg=pgp-sha1;
        protocol="application/pgp-signature";
boundary="boundary"
Content-Disposition: inline


--boundary
Content-Type: text/plain; charset=us-ascii
Content-Disposition: inline
Content-Transfer-Encoding: quoted-printable

Mallory's message goes here.

--boundary
Content-Type: application/pgp-signature; name="signature.asc"
Content-Description: Digital signature
Content-Disposition: inline

-----BEGIN PGP SIGNATURE-----
Mallory's signature goes here.
-----END PGP SIGNATURE-----

--boundary
Content-Type: text/plain; charset=us-ascii
Content-Disposition: inline
Content-Transfer-Encoding: quoted-printable

Email copied from Alice, a list member, goes here.

--boundary
Content-Type: application/pgp-signature; name="signature.asc"
Content-Description: Digital signature
Content-Disposition: inline

-----BEGIN PGP SIGNATURE-----
Alice's signature goes here.
-----END PGP SIGNATURE-----

--boundary--
```

Figure 2: Layout of `multipart/signed` message with four parts. In SLS, only the last two parts are verified. In Mutt, the first part and a "Good signature" message is shown.

  - Else, discard the message.

- If the `smime_distrib_encrypt` option is set, try to encrypt the message for each recipient.
  - Also sign the message if `smime_distrib_sign` is set.
  - If no key is known for the recipient or encryption fails, and `smime_distrib_encrypt = 1`, send plaintext.
  - Else, discard the message.

The issues found here are:

- Again, there are possible but nonsensical combinations of `gpg_distrib_*` and `smime_distrib_*` options.
- The extraction of plaintext before it is passed to the encryption functions is different between the PGP and S/MIME cases. This should be unified.
- If `*_distrib_encrypt = 1`, SLS's behaviour allows one to send an unencrypted message to the list and have it distributed encrypted, or allows an encrypted message send to the list be distributed unencrypted, depending one whether subscribers have uploaded their public keys or not. It would be better to always distribute messages encrypted if they were posted encrypted, and unencrypted if they were posted unencrypted. To handle the case of a dissident[1] who wants to ensure emails are always sent to him encrypted, a per-subscriber option, like `always_encrypt`, could be added.

3.2.4 `GPGUtils.py`

The following issues were found in `GPGUtils.py`:

- The function `checkPerms()` is called three times, but the return value is never checked.
- In `getMailaddrs()`, the regular expression used to extract the email address part from the uids is flawed. It matches the first string between angular brackets, however this means an uid like "`Mallory (<alice@example.org>) <mallory@example.com>`" would yield the wrong address, allowing Mallory to use "`Alice <alice@example.org>`" in the `From` header. Of course, the list administrator should check all uids carefully anyway, since one can add any uid he wants to his key.
- In `decryptMessage()`, the short key id from the `GOODSIG` response is used. However, it would be better to use the long key id from the `VALIDSIG` response, to reduce the chance of a hash collision which would allow an attacker to fake the signature of a list member.
- In `decryptMessage()`, the `SIG_ID` response should be checked: the date should not be too far in the past or future, and it should not match any previously found `SIG_ID`. This would prevent replay attacks and time delay attacks.
- In `decryptMessage()`, properly encrypted messages but which contain bad signatures are still accepted, even though `key_ids` is left empty. It would be better to outright reject such questionable messages.
- There is a lot of code duplication between `encryptMessage()` and `encryptSignMessage()`.
- In `verifyMessage()`, `VALIDSIG` and `SIG_ID` should be used as mentioned above.

[1] See http://en.wikipedia.org/wiki/Debian_Free_Software_Guidelines#debian-legal_tests_for_DFSG_compliance for a description applied to software, but which equally applies to any kind of information a person would want to spread.

3.2.5 SMIMEUtils.py

The following issues were found in SMIMEUtils.py:

- In all cases, openssl is called without an absolute pathname. This allows an attacker who can create an arbitrary file in the $PATH of the SLS process to insert a fake OpenSSL binary.
- In decryptMessage(), a encrypted+signed message that is decrypted properly but whose signature verification failed is accepted anyway, just as in the GPGUtils case. It would be better to reject such messages.
- In decryptMessage(), if decryption failed an empty message is returned instead of None.
- Message signatures are checked against the list of all known certificates, but there is no information about which certificate signed the message. This makes it difficult to check signatures against From headers.
- In encryptMessage(), both plaintext and ciphertext are stored in temporary files created with mkstemp(). This allows an attacker who can read files from /tmp owned by the SLS process to acquire the plaintext of encrypted messages. It also allows plaintext to stay indefinitely on disk. Furthermore, according to the Python documentation, "*there is thus no guarantee that the generated filename will have any nice properties, such as not requiring quoting when passed to external commands via* ***os.popen()***". It is better to use the c_in handle to pipe the plaintext into openssl, having the ciphertext on disk is not a risk.
- In encryptSignMessage(), temporary files are used as in encryptMessage(). It would be better to use the c_in and c_out handles to pipe plaintext into and out of openssl, but if this does not work due to blocking of the pipes, then the AsyncRead() and AsyncWrite() functions from GPGUtils.py should be used.
- In decryptMessage(), the cmd variable is a tuple of strings, in encryptMessage() it is just a string. If there are spaces in any paths or in recipfile, these will be interpreted as argument separators in encryptMessage(). The tuple form should be used here as well.
- Nowhere is the status variable checked to see if the openssl command succeeded.
- There is code duplication between decryptMessage() and verifyMessage().

4 Weaknesses in SHA-1 and its impact on SLS

Recent advances in cryptography have led to the discovery of attacks against the SHA-1 algorithm[2]. The SHA-1 algorithm is used to derive fingerprints (and hence, keyids) from PGP keys, and is used to generate signatures made with those keys. Many X.509 certificates also specify the SHA-1 algorithm for creating signatures. The recently discovered attack reduces the complexity of breaking SHA-1 from 2^{79} to 2^{52} operations, and many believe that further research will result in even better attacks which will further reduce the complexity. Therefore, many cryptographers recommend moving away from SHA-1 to the SHA-2 suite of hash algorithms, and to SHA-3 in the future. At the moment, the free software community is still evaluating how to deal with the situation[3]. SLS is calling external programs to deal with keys and signatures, so there is little that will need to change. However, once migration to new keys and certificates which use stronger hash algorithms is common, SLS should disallow members to upload keys and certificates which specify the use deprecated algorithms, and should disallow emails with signatures that were created using deprecated algorithms.

[2] http://eprint.iacr.org/2009/259.pdf
[3] http://lwn.net/Articles/337091/

5 Recommendations

The lax checking of `multipart/signed` messages should be fixed first, since it allows an attacker to pass his own message to the list if he can obtain a signed message from a list member.

After reviewing the code, I believe that afterwards, the most important task for the developers is to seriously reduce the number of configuration options available to the list administrator. Options that inherently compromise security, such as attachment scrubbing, archiving and support for Usenet, should be permanently disabled and removed from the list administrator pages. The duplicate sets of options for PGP and S/MIME, and the code duplication behind it, should be unified into two options:

sign_policy None, voluntary, mandatory.

> When set to none, the list should not check or add signatures. When set to voluntary, signatures should be checked, and if an incoming message is signed, the outgoing message must be signed as well, otherwise it should not be signed. When set to mandatory, both incoming and outgoing messages must be signed.

encrypt_policy None, voluntary, mandatory.

> When set to none, the list should not try to decrypt or encrypt messages. When set to voluntary, encrypted messages should be decrypted, and if an incoming message was encrypted, the outgoing message must be encrypted as well, otherwise it should not be encrypted. When set to mandatory, both incoming and outgoing messages must be encrypted.

The reduction in options will make it easier for list administrators to make the right choice, and will simplify code and remove many possibly dangerous code paths. For list members, the semantics of these options follow the principle of least surprise[4]; signed messages in are signed messages out, encrypted messages in are encrypted messages out.

The next most important task is to continue developing the test suite as mentioned in part I of the audit.

Then I would recommend fixing the processing of status and return codes from external `gpg` and `openssl` processes. An attacker might craft malformed PGP and S/MIME messages which could trigger unexpected behaviour if left unchecked.

Also important, but for now I believe of lesser importance than the above, are the issues found with plaintext being stored on the host running SLS and possible attacks by local users on that host.

6 Conclusion

The security of SLS has improved since the previous audit. However, a large number of options available to list administrators allow unsafe behaviour to be enabled. The developers should remove those options, and continue work on providing strict integrity and confidentiality enforcement.

[4] http://en.wikipedia.org/wiki/Principle_of_least_surprise