Security Audit of the Secure List Server Part I

Guus Sliepen <guus@sliepen.org>

May 24, 2013

Abstract

The subject of this audit is Mailman 2.1.11-2 with the pgp-smime patch of 2008-07-03. The list of known, open issues was reviewed. The code provided in the patch was also reviewed. A test setup was created and the administration and usage of secure lists were reviewed. A number of issues were found with this version of the Secure List Server. Although the bare functionality of a secure list server is implemented, there is still some way to go before it can be called robust and fool-proof. The developers should work first on providing 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.

Before looking at the cryptographic functionality offered by SLS, I will describe the basic functionality of a mailing list manager (hereafter called a list manager). A mailing list manager manages zero or more mailing lists (hereafter called a list). Each list can have zero or more subscribers. Both lists and subscribers are associated with email addresses. If a subscriber of a certain list sends an email to the list's address (herafter called posting), the list manager will forward a copy of that email to all the other subscribers. In some cases, a person not subscribed to a particular list (a non-subscriber), is allowed to post to that list, in other cases it is not allowed. The list manager generally allows people to subscribe themselves to the lists of their choice. A list manager can hold messages for moderation by a list moderator. A list manager can maintain archives of the emails sent to the list.

The original Mailman is not aware of OpenPGP or S/MIME signatures or encryption, and will forward messages that are signed and/or encrypted verbatim, without checking signatures or performing decryption. The pgp-smime patch brings awareness of signatures and encryption to Mailman. If SLS knows the public keys of the subscribers for a list, it can verify the OpenPGP or S/MIME signatures on incoming emails, and reject emails without proper signatures, if so configured. If a list is associated with a given public/private keypair, SLS can optionally sign outgoing emails with the list key, allowing subscribers to verify that the emails they receive really were sent via the list manager. If a subscriber posts a message encrypted with the list's public key, SLS can decrypt the message and optionally re-encrypt it with the receivers' public keys before forwarding the message, allowing messages to be exchanged in confidentiality within the group of subscribers.

It is important to realise that subscribers have certain expectations from secure lists, and that not all of those expectations can simply be met by throwing some cryptography at the list manager. The scope of this security audit therefore does not only include the use of the GNU Privacy Guard (GnuPG) and OpenSSL in the pgp-smime patch, but the whole functionality SLS offers to the list administrators and subscribers.

1.2 Test setup

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

- Asus EeePC 701 with 2 GB RAM and 20 GB storage
- Debian unstable (last update on 2008-07-21)
- Mailman 2.1.11-2 patched with `mailman-2.1.11-pgp-smime_2008-07-03.patch`
- Postfix 2.5.2-2 mail server
- Mutt 1.5.18-3 mail client
- Lighttpd 1.4.19-4 light-weight web server
- GnuPG 1.4.9-2
- OpenSSL 0.9.8g-11
- Python-GnuPGinterface 0.3.2-9

Prior to this setup, an almost identical one was used, except that `mailman-2.1.11-pgp-smime_2008-06-25.patch` was used. That particular version of the patch lacked the ability to encrypt or sign outgoing messages, and would always send messages in plaintext, even on lists where encryption was mandatory. This gross oversight was fixed in the next patch, but could have been prevented by having a test suite that the developers can use to determine the correct functioning of SLS before new patches are released.

2 Review of TODO list

SLS contains a list of open items in the file `TODO.PGP-SMIME`. I have reviewed this list, and have the following remarks for some of the items:

0006 *Using the web roster, any subscriber can view any subscribers' preferences, including public key. And maybe even change.*
The web roster should normally only be accessible by the list administrator, not by individual subscribers.

0009 *Harden this thing: re-encrypt immediately after decrypting.*
The best way to ensure confidential information is not leaked is indeed to keep the time the decrypted information is present to a minimum. Although normally memory that is freed by an application is discarded or reused for other purposes, it is possible that temporarily used memory is stored more permanently, outside the control of the application. If the list manager runs on a system with swap enabled, it is possible that parts of the memory are copied to disk, where it can remain indefinetely. This risk can be avoided by using the mlock() call to prevent memory from being swapped out to disk. The list manager should also immediately overwrite sensitive information with zeroes or random bits. The list

manager should also avoid writing decrypted information to temporary files, as it is even harder to manage sensitive information on disk than in memory.

0012 *When creating a list, make sure the listadmin password is not sent via plain email.*
A chain is only as strong as its weakest link. For lists that require confidentiality, and hence only accepts and forwards encrypted emails, the management of that list should also go via an encrypted channel, either via encrypted emails from the list admin to the list manager, or via an SSL encrypted connection to the web interface of the list manager. It would also be better to use signed emails or client certificates than to use passwords.

0013 *Perhaps we should suggest an empty passphrase for list keys in our interface.*
Passphrases only improve the security of key material for human interaction. Perhaps it is better to *require* empty passphrases for list keys, so as not to give the false impression that a passphrase would enhance security.

0015 *We should refuse to create an html list archive for secure lists.*
It should be possible to create archives of secure lists while still preserving the intended integrity and/or confidentiality of the list. Lists that just require integrity can maintain an archive provided that the signature of archived emails is kept. Lists that require confidentiality can either store the incoming emails in the archive without decrypting them, and then re-encrypt them to another subscriber when they request old emails from the archive. Another option is to decrypt incoming emails, and store a copy in the archive that is encrypted to *all* current subscribers. Both options have their own advantages and disadvantages, perhaps this could be a configurable setting.

0024 *If a post is properly signed, accept it, no matter wether the From-adress is subscribed and no matter the sender moderation policy.*
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.

0030 *Deal with subscribers without public keys.*
The best way to deal with this issue is not to allow someone to subscribe without providing a public key for lists that require one. For subscription via email, require that the subscription request is properly signed, and automatically store the public key along with other subscriber's details. For subscription via the web, require that the subscriber uploads his public key in the same form as the subscription request. In both cases, require that the subscriber's response to the verification email is also signed.

0031 *When bouncing e-mail because list policy was violated [. . .] only bounce the headers, not the complete e-mail message.*
Indeed, by allowing the body of an email to be included in the bounce, an attacker can send emails containing viruses or other unwanted payloads to a list, and can forge the From header so the bounce will be sent to a victim of choosing. This item was marked as difficult to implement, wishlist severity. I cannot believe it would be difficult to strip the body from an email, and preventing SLS from being used as a spam or virus redistributer should be high priority.

0033 *All defaults should be strict.*
Indeed, for SLS the defaults *must* be as strict as possible to prevent a list being accidentally less secure that intended.

0036 *When re-encrypting a signed message, the original signature gets lost: this makes it possible for one list member to pose as another list member. In theory, it should be possible to keep the original signature after decryption.*

Indeed, since GnuPG has no option to decrypt a message without removing its signature, SLS currently cannot preserve the original signature, but rather adds its own when forwarding an email. One should ask (and perhaps sponsor) the GnuPG developers to implement this missing feature. In the mean time, one should not allow messages to be posted where the From header does not match one of the email addresses associated with the public key used to sign the email (see also item 0024 above).

0050 *Make sure posts get encrypted and signed if needed.*
Although this item was marked as being S/MIME specific, this statement also applies to GnuPG of course. 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.

0060 *emailf00f by Guus Sliepen deals with PGP. Study its source.*
The most relevant feature in emailf00f for SLS is that by sending it a single signed email, it can setup an association between the public key and an identity, and automatically retrieves and stores the public key for further communication. See also the suggestions in item 0030.

3 Code audit

I have reviewed the PGP and S/MIME related source code files of the patched Mailman to some extent (a line-by-line analysis of the whole Mailman code base is not possible in the scope of this security audit). The patch makes an effort to put most PGP and S/MIME related functionality into their own Python classes, most notably in GPGUtils.py and SMIMEUtils.py. These classes are well structured, and contain clearly defined functions to process the various cryptography-related aspects of messages. However, the source code files that handle reception and sending of emails, most notably Handlers/Moderate.py and Handlers/SMTPDirect.py, contain all the glue logic for PGP and S/MIME handling interspersed with the original mail handling functions. In these files, the code is harder to read, and it is probable that (future) errors are more easily made. For the code quality it would be better if the GnuPG and S/MIME related functionality in these files were moved to their own files, and that this functionality would be called from the incoming and outgoing email handlers by callbacks.

The following issues were found in the code:

- The GnuPGInterface library, by default, does not use a full path when calling the gpg binary. This can allow someone with access to environment or to a local bin directory to divert calls to gpg to a subverted binary. SLS should override the call variable of the GnuPG object with the full path to the gpg binary.
- SMIMEUtils.py calls openssl without a full path. See the above item.
- SMIMEUtils.py makes heavy use of temporary files. In particular, plaintext copies of messages are written to temporary files. Although the temporary files are removed directly after use, this allows some with read access to the temporary files to obtain copies of the plaintext. It also allows the plaintext to end up on a storage device, where it may stay indefinitely (removing a file normally does not actually remove the contents from disk).
- The logic of handling signatures and encryption is such that if both `gpg_post_sign` and `smime_post_sign` are set to Force, that only emails are accepted which have both an OpenPGP and an S/MIME signature at the same time. The same logic applies to `gpg_posting_allowed`

and `smime_post_encrypt` for encryption, and for the other combinations of signing and encryption of outgoing messages. It is clearly not desirable behaviour. If a list administrator enables both OpenPGP and S/MIME, he means that people can use *either* OpenPGP *or* S/MIME. It also makes perfect sense to allow some subscribers of a list to use OpenPGP, and the others S/MIME, as it's not the technology that is used that is important, but rather just the integrity and confidentiality guarantees they provide.

- Poor error handling in GPGUtils.py. When calling gpg, only the statement `p.wait()` is surrounded by a `try..catch` block. However, when gpg is called with wrong arguments, it will terminate before accepting input from stdin, resulting in an unhandled Broken Pipe exception.

4 Usage audit

SLS was set up from scratch together with a mail server (Postfix) and a web server (Lighttpd). A test list was created and subscribtions were made from various email addresses. Various settings for the test list have been tried and in each case plaintext, signed, encrypted and signed+encrypted emails have been sent to the test list, from both subscriber and non-subscriber addresses. The following issues were found:

- The list administrator interface is not very clear. For example, one of the options is: *"Should messages be GPG signed? Yes means: hold for approval. (No, Yes, Force)"*. It is unclear from just this question what the difference between Yes and Force is. Will Force add a signature if there is none present, or twist the subscriber's arm until he does? If I answer Yes, does that mean that signed messaged are held for approval? It takes a while before one guesses the true meaning of the choices. It is better to rephrase the question to *"Allow unsigned messages? (Yes, Hold, No)"*, or to elaborate the choices in the original question: *"No, Yes (hold unsigned), Force (drop unsigned)"*. The sames goes for all the other three-choice option.

- OpenPGP and S/MIME are treated as two completely different things. One can configure a list to require OpenPGP signatures, but S/MIME encryption. This does not make much sense. It would be better if the options were made technology agnostic: *"Require emails to be signed?"*, *"Require emails to be encrypted?"*, etcetera, and to enable OpenPGP and/or S/MIME based on whether the relevant keys were provided.

- Although one can easily upload OpenPGP keys via the website, there is no way to upload an S/MIME list key. With the current patch, the list administrator somehow has to put the S/MIME list key in `/var/lib/mailman` on server running SLS.

- SLS requires the list administrator to generate a public/private keypair and to upload both to the list manager. Uploading via unencrypted HTTP to the web interface severily compromises the secrecy of the private key. Although this problem can be solved by enforcing SSL for the web administration interface, it might be even better to allow the list manager to generate public/private keypairs for lists. This simplifies setting up new lists and obviates the need for the private key to be moved around.

- SLS does not allow minimum requirements set for key sizes, cipher and digest algorithms. Instead, the defaults of GnuPG and OpenSSL are used, which may be quite liberal. A list administrator might want to enforce stricter security to limit the chances of compromising communication on a secure list.

- Once someone is subscribed to the list (possibly only after getting permission from the list administrator), he can log in to the list manager web interface with just a password and change his settings. This also allows him to change his public key. Since a public key is a much stronger credential of someones identity than just an email address, and since it is much easier to guess a password than to crack a key, it should not be allowed to change the public key without explicit permission from the list administrator.

- Sending unencrypted emails to a list for which encryption is mandatory always results in an "Encryption required" email being sent to the email address mentioned in the From header of the original email. The complete original message is attached to the response. This allows an attacker to use SLS as an anonimising remailer for spam or viruses.
- If an encrypted message sent to a list is for some reason automatically discarded, the auto-discard notification is sent to the list administrator, but it contains an decrypted copy of the original message. This breaches the confidentiality of the original message. An encrypted message should either be forwarded encrypted or not at all.
- SLS does not check the serial number or timestamps of a signature. Therefore it is possible to resend old messages to the list, and an attacker who can get hold of one message sent to the list does not need to be able to decrypt it or alter its signature to cause a denial of service (DoS) attack by sending this same message to the list over and over again, causing SLS to forward the message to the other subscribers over and over again.
- Emails with a valid signature of a known subscriber are accepted regardless of whether the address in the From header matches one of the email addresses associated with the key. Since the original signature is removed before the mail is sent to the other subscribers, this allows one subscriber to impersonate another subscriber or even an outsider.
- SLS can be misused as a so-called oracle. It does not care about the contents of a message, but will happily decrypt and sign messages sent to it and forward them. A subscriber can get a signature with the list key on any message it wants, just by sending that message to the list. A subscriber can also resend to the list any old message that was sent encrypted to that list, and receive a copy that has been decrypted and re-encrypted for the subscriber. Care must be taken that a list key is unique and used only for that list, and that timestamps on signatures are verified to prevent new subscribers to be able to decrypt old emails sent to the list, if that is undesirable.

5 Priority of security-related issues

Most important is to implement a test suite so the developers can check for regressions before publishing new versions of SLS. The administrators who install SLS will probably not check every aspect of SLS before upgrading their installation, so it would be best for the developers to prevent the situation from the patch of 2008-06-25 from happening again. The test suite does not necessarily have to be a fully automated set of scripts that perform various checks, it can be a checklist that a developer manually works through to check the functionality of SLS, as long as it works correctly.

Second most important is to make sure that when a list is set up for integrity and/or confidentiality, these aspects are *always* enforced. This means that an encrypted message sent to a list *never* leaves SLS unencrypted. I also strongly recommend to try to keep the original signature of messages sent to a list, but in case of OpenPGP this might not be possible without support from the GnuPG developers.

Third most important is the simplicity and consistency of the user interface. The harder it is for administrators or subscribers it is to understand certain settings, the more mistakes will be made. The more hoops one has to jump through to upload a key, the more mistakes will be made. Mistakes can cause SLS to be configured with lesser security than intended, and might inadvertently compromise communications.

Then there are a number of issues dealing with aspects of SLS that are not as secure as OpenPGP and S/MIME are, like the rather simple password protected access to the list settings. Again, a chain is only as strong as its weakest link, so try to remove or strengthen any link weaker than the OpenPGP and S/MIME part.

The source code revealed some possibly insecure uses of pathnames and temporary files, but these issues are less important if SLS is run on a reasonably secured server.

6 Conclusion

Although the bare functionality of a secure list server is implemented, there is still some way to go before SLS can be called robust and fool-proof. The developers should work first on providing strict integrity and confidentiality enforcement.