

# FOCUS on Linux: Clear Text Communication - Slaying the Beast Part Two

Hal Flynn 2000-10-25

## . Introduction

In our last article, we discussed moving to secure communication by eliminating telnet, in favor of SSH. We discussed the technical skills needed in such a transition, as well as the soft skills.

In this article, we go a step further in eliminating clear text passwords, exploring mail retrieval protocols. We discuss freely available mail software, features of the software, and how to implement these packages to provide secure authentication and communication. By implementing a cryptographic communication posture, we take one more step towards *Slaying the Clear Text Beast*.

## . Going Further

Much like the Titanic couldn't be saved with just a cork after hitting the iceberg, implementing *OpenSSH* isn't a single fix to all clear text communication problems. Consider the other services running on the network that may contain sensitive information, and communicate via clear text. In some places, moving to policy which eliminates clear text traffic altogether isn't entirely possible. The ability to communicate with outside entities makes using local encryption difficult at best, and impossible in most cases. While a *Public Key Infrastructure* can solve part of this, creating a secure tunnel between two different places divided by political boundaries, there are still risks that have to be considered locally. In situations like this, making every effort to make an eavesdroppers listening more difficult is a success.

Another service typically using clear text is email. While in transit, mail is susceptible to snooping when going from one mail server to another via SMTP. It is for this reason that mail shouldn't contain any sensitive information. There are efforts underway, notably that of the [Sendmail Consortium](#), to move *Sendmail* to secure server communication. This however, is outside of the scope of our discussion. Instead, we'll focus more on the user's communication with the mail server.

## • The Future of Email

While we'll not touch on the privacy issues involved with email, the means to gain access to one's email usually involves a username and a password. But with that consideration, it's also possible to read one's email without even having a username and a password. A system sitting compromised on a network is likely to be listening to the network, and listening much more closely than those tasked with the maintenance and security of the network. Considering the system resources required to perform the sniffing and archival of email, the likelihood of such an incident is very low without at least notice by administrators. However, it is trivial to hide a process and text file that sniff and store usernames and passwords.

In the future, mail will be encrypted in multiple layers. It will be encrypted and signed by the sender of the mail, sent to the *Mail Transport Agent* over a cryptographically secure connection, then securely transmitted to another mail server. From the moment it is typed in the Mail User Agent, to its arrival in the mailbox of the intended recipients, it will traverse the network encrypted by multiple means.

This future has already begun a manifestation as reality. Currently, *Request For Comments 2595*, authored by [Chris Newman](#) of *Innosoft International, Incorporated*, sits as a Proposed Standard. This Request For Comments addresses using *Transport Layer Security* for the authentication of both usernames and passwords. The draft outlines using TLS with the protocols *Post Office Protocol Version 3*, *Interactive Mail Access Protocol*, and *Application Configuration Access Protocol*.

This excerpt is taken from [RFC 2595](#):

Copyright (C) The Internet Society (1999). All Rights Reserved.

Many sites have a high investment in authentication infrastructure (e.g., a large database of a one-way-function applied to user passwords), so a privacy layer which is not tightly bound to user authentication can protect against network eavesdropping attacks without requiring a new authentication infrastructure and/or forcing all users to change their password. Recognizing that such sites will desire simple password authentication in combination with TLS

encryption, this specification defines the PLAIN SASL mechanism for use with protocols which lack a simple password authentication command such as ACAP and SMTP. (Note there is a separate RFC for the STARTTLS command in SMTP [SMTPTLS].)

SASL as mentioned in this document is in reference to the *Simple Authentication and Security Layer*, defined by [RFC 2222](#) and updated by [RFC 2444](#). Note also the mention of TLS implemented with SMTP. This is defined in [RFC 2487](#), titled *SMTP Service Extension for Secure SMTP over TLS*. We will not address these as they are outside of our focus. We will however look at securing common protocols using freely available tools.

## • Client Availability

One of the biggest obstacles in moving common services to secure authentication or even full cryptographic transit is support. This stumbling block can be seen in many of today's end user applications. The availability of services such as mail through POP3 and IMAP are standard. The means of authenticating to these services through clear text usernames and passwords is usually also standard. It is possible to cloak authentication information in cryptography and only allow secure communication with these services on the server side. At the user's desk however, the applications that communicate with these services may prevent the user from ever being able to use the service. Applications such as Netscape Messenger are affected by this shortcoming.

*Netscape Messenger* is a Mail User Agent bundled with the *Netscape Communicator* suite. This MUA usually supports a few different mail retrieval protocols, varying by platform. The two commonly supported protocols across all platforms are Post Office Protocol Version 3, and Interactive Message Access Protocol.

Netscape Messenger can retrieve mail from IMAP servers over port 143, the *IANA Well Known Port* for IMAPv2, and port 993, which is the *IANA Well Known Port* for IMAPv4 protocol over TLS/SSL. A security conscious site could use IMAP servers to provide their users with security by using IMAP. Messenger can additionally send outbound email via TLS to a mail server that supports the protocol.

But what of POP3? The implementation of Netscape Messenger in *Netscape Communicator 4.76* supports POP3 via port 110, the *IANA Well Known Port* for POP3.

There is, however, no support for POP3 over TLS.

In contrast, there's *Microsoft Outlook Express*. Outlook Express, like Netscape Messenger, supports IMAP over port 143, and IMAP with TLS over port 993. Unlike Messenger, Outlook Express supports both POP3 over port 110, and POP3 with TLS over IANA Well Known Port 995.

The relevance of this problem will become evident later, when we discuss using POP3 over TLS.

## • **Server Availability**

There are many popular server packages available to service POP3 and IMAP. Some of these are available for a small fee, while others are available for under a free license such as the *Berkeley Software Distribution License* or the *General Public License*. Our focus is on those which are free.

## • **Post Office Protocol Version 3**

*Post Office Protocol Version 3* is one of the one of the most commonly used mail retrieval protocols available on the Internet. Large Internet Service Providers such as *Earthlink* and *Mediaone* use it exclusively to service their customers. The simplicity of the POP3 protocol as well as the ease of installation and maintenance make it appealing to Administrators and users alike. While POP3 is the easiest of protocols to use, its use is not without sacrifice.

There are a number of POP3 server packages available. The most common and popular is the [Qpopper](#) package available from [Qualcomm](#). It supports a wide variety of platforms, and is easily built. Configuring Qpopper is also easy. Qpopper does not support TLS natively, however. Additionally, Qpopper's [license agreement](#) and history of security issues makes its use less desirable.

Instead, we look at the *GNU POP3* daemon, led by developer [Jakob Kaivo](#), and found [here](#). The GNU POP3 daemon is a [GPL](#) piece of software that has been developed with both the best performance in mind as well as the strictest adherence to RFC guidelines. It can be run either as a daemon, or from inetd.

The GNU POP3 daemon, like Qpopper, also does not support TLS natively, a sacrifice in functionality. There is another piece of software that can act as a TLS intermediary between the client and the server, and does so in a protocol friendly manner. This piece of software is *Stunnel*. We will discuss Stunnel later.

## • Interactive Mail Access Protocol

A software package servicing IMAP under the GPL is *Courier-IMAP*. The Courier-IMAP project is lead by developer [Sam Varshavchik](#), and available [here](#). Courier-IMAP comes with a variety of features. One feature is support of the *Maildir* mail storage format, making it a drop-in service for [Qmail](#). Other features include support for virtual users, support for [OpenLDAP](#), and a lower consumption of resources than that of its competitors, namely the [UW-IMAP](#) Server.

Courier-IMAP also supports TLS natively. To use TLS with Courier-IMAP, the [OpenSSL](#) package is required. Courier-IMAP makes use of X.509 certificates to communicate with a client capable of negotiating IMAP over TLS. Thus, the entire session between the client and the server can be encrypted using OpenSSL with Courier-IMAP.

The OpenSSL implementation was covered in our first part in this series, and provides reference to the building and installation of OpenSSL. We will not cover an implementation of Courier-IMAP either. The best reference for an implementation is the Courier-IMAP software documentation. Courier-IMAP can be tedious to implement. Therefore, read the software [documentation](#) and [FAQ](#) meticulously.

## • The Use of Stunnel

*Stunnel* is to TLS what *netcat* is to TCP/IP, a versatile, flexible tool that can be used in many different ways and practices. Stunnel is described at its [homepage](#) as the *Universal SSL Wrapper*, and serves its purpose well. It is scalable, and runs on multiple platforms. It also supports several Operating Systems.

Stunnel is dependent upon the OpenSSL library. Stunnel is capable of wrapping virtually any TCP based service. It can be used to provide a secure means of communication to POP3 clients, as well as IMAP.

As mentioned previously, the GNU POP3 daemon does not support SSL. Using Stunnel, it

is possible to provide POP3 over TLS to Mail User Agents that support SSL. It is also possible provide TLS to clients that do not support SSL. We will not cover an implementation for clients that do not support SSL. In the following example, we cover a client that supports TLS, and a server that does not.

A Linux Server in our hypothetical server farm runs GNU POP3 to service POP3 clients. We'll call this server *Athens*. A Windows 98 Client running Outlook Express needs connectivity to this server. We'll call this client *Troy*. Our Site *Security Policy* dictates that there are to be no clear text passwords used on the network.

In this example, gnu-pop3d is a inetd started service. To run the pop3d securely, we remove the entry for the pop3d from `/etc/inetd.conf`. We then use Stunnel to launch the daemon after authentication. To do so, we would start Stunnel in the following way:

```
# stunnel -d pop3s -l /usr/local/sbin/gnu-pop3d
```

The `-d` argument tells stunnel to listen on port 995 for pop3s connections. The `-l` argument tells stunnel to spawn gnu-pop3d after authentication. This will allow Troy to connect to Athens on port 995, authenticate securely, and retrieve email. This is application of stunnel for this purpose.

In another example, perhaps we have the POP3 service is already running on Athens in daemon mode. Stunnel can instead be used as a pseudo-proxy to connect to the already existing POP3 service. To do this, we would instead launch Stunnel in the following fashion:

```
# stunnel -d pop3s -r localhost:imap
```

The `-d` argument tells stunnel to listen on port 995. The `-r` flag tells stunnel to redirect all traffic to address 127.0.0.1 port 143.

Stunnel can be applied to many situations of this type. It can be applied to an IMAP server that doesn't support SSL, or even a news server. We'll not dig any further into the features. Instead, you're encouraged to look at the product documentation and the [FAQ](#).

## • Final Thoughts

Implementing TLS within a network is never easy. From the technical skills and assets to the soft skills, the hurdles are many. Organization, tact, and a good plan for implementation are the keys to a successful transition.

Of the mail retrieval protocols, picking the one best suited for your needs requires planning and evaluation of your surroundings. In a structured environment where all the clients are company employees, it's easy to plan for a group of users that will likely be using the same Mail User Agents. In a less structured environment where the client is a user that could be using any MUA, this problem is not as easily solved. The best decision, of course, is one that meets your business needs, and protects all of your users.

## • Conclusion

In this article, we discussed clear text communications and mail retrieval protocols. We discussed the use of the GNU POP3 daemon and Courier-IMAP server for the retrieval of mail. We also discussed the use of OpenSSL and Stunnel to secure communication between the client and the server, concluding our series *Clear Text Communication: Slaying the Beast*.

### Relevant Links

[Clear Txt Comm. Pt. I](#)

*Hal Flynn*

[RFC 1939](#)

*J. Meyers & M. Rose*

[RFC 2595](#)

*Chris Newman*

[Stunnel](#)

*Michal Trojnara*

[Apache with SSL](#)

*Dale Coddington*

[OpenSSH Homepage](#)

*OpenSSH Project*

[OpenSSL Homepage](#)

*OpenSSL Project*

[Privacy Statement](#)

Copyright 2006, SecurityFocus