Manuale di FreePOPs

6th July 2004

Contents

1 Introduzione

FreePOPs e' un demone POP3 piu' un interprete LUA e alcune librerie extra per il parsing di HTTP e HTML. Il suo scopo principale e' tradurre richieste POP3 locali in azioni HTTP remote per le webmail supportate, ma in realta' e' piu' flessibile di cosi': per esempio esiste un plugin per leggere news da un sito web come se fossero messaggi in una mailbox. Si puo' facilmente estendere FreePOPs al volo, senza neanche farlo ripartire; si puo' aggiungere un plugin o modificarne uno esistente semplicemente cambiando uno script, dato che i plugin sono scritti in LUA e sono interpretati al volo.

1.1 Situazioni d'uso

FreePOPs puo' essere utile in molte situazioni, qui descriviamo le piu' ovvie:

- Siete dietro un firewall che chiude la porta 110 ma volete comunque leggere la posta elettronica e la webmail del vostro provider fa schifo.
- Il vostro mail provider non vi permette di accedere alla mailbox con il protocollo POP3 ma solo tramite il servizio di webmail.
- Preferite usare la vostra mailbox invece di sfogliare le news di un qualche sito.
- Dovete sviluppare un server POP3 in meno di una settimana e volete che sia ragionevolmente veloce e che non consumi molta memoria.
- Non siete hacker del C, ma volete trarre beneficio da un frontend ad un server POP3, veloce e scritto in C, ma non volete perdere un mese a scrivere il backend in C. LUA e' un linguaggio davvero semplice e leggero, una settimana e' abbastanza per impararlo e poterlo usare in modo produttivo.

1.2 Feature

FreePOPs e' l'unico software che conosciamo con queste feature:

- Server POP3 compliant con RFC (non con tutte le feature, ma compliant).
- Portabile (scritto in C e LUA il quale e' scritto in C, quindi tutto e' scritto nel linguaggio piu' portabile del mondo).
- Piccolo (in termini di risorse utilizzate) e ragionevolmente veloce.
- Estremamente estendibile al volo mediante un linguaggio semplice e potente.

- Piuttosto documentato.
- Rilasciato sotto la licenza GNU/GPL (questo significa che FreePOPs e' Software Libero).

1.3 Plugin

Questi sono i plugin correntemente inclusi in FreePOPs:

libero.lua Questo plugin supporta in pieno la webmail di `www.libero.it` per mailbox con domini come `@libero.it`, `@iol.it`, `@inwind.it`, `@blu.it`.

tin.lua Questo plugin supporta in pieno la webmail di `communicator.virgilio.it` per mailbox con domini come `@tin.it`, `@virgilio.it`.

lycos.lua Questo plugin supporta la webmail `mail.lycos.it` per mailbox col dominio `@lycos.it`

popforward.lua Questo e' un plugin usato principalmente per testare moduli di FreePOPs. Esso agisce come un forward POP3, semplicemente si comporta come mediatore tra voi ed un vero server POP3. Questo plugin ci ha permesso di testare FreePOPs senza avere alcun vero plugin gia' scritto. Potreste usarlo per mascherare un server POP3 con molti bug che puo' venire facilmente compromesso grazie a richieste malevole. Ovviamente suggeriamo di esaminare per bene questo plugin, e fare un po' di hacking per prevenire richieste malevole al vostro server.

aggregator.lua Molti siti forniscono un backend RSS per indicizzare le loro news. Questo plugin fa si' che RSS si comporti come una mailbox in cui potete trovare una mail per ogni news.

flatnuke.lua Questo e' un aggregator piu' potente per siti basati sul CMS FlatNuke, e permette il download dell'intero corpo delle news.[1]

kernel.lua Questo e' un plugin per tenersi aggiornati sulle ultime versioni del kernel Linux.

2 Storia

FreePOPs non nasce dal nulla. Un progetto simile (solo nella situazione d'uso principale) e' LiberoPOPs.

[1] Vedete HTTP://flatnuke.sourceforge.net per l'homepage del progetto

L'antenato di FreePOPs e' completamente scritto in C per ragioni poco interessanti. LiberoPOPs supporta "plugin" ma in maniera piu' statica e complessa. Il frontend al server POP3 potrebbe essere collegato ad un backend scritto in C, questo significa che dovreste ricompilare e far ripartire LiberoPOPs ogni volta che cambiate una riga in un plugin. Un altro punto interessante e' che LiberoPOPs era stato creato dal nulla in un tempo molto breve (dovete essere Italiani e usare un indirizzo di posta`@libero.it` per capire perche'), cio' vuol dire che era nato con molti bug e FIX-ME nel codice.

Il progetto LiberoPOPs ebbe un rapido successo, perche' tutti ne avevano bisogno, quindi avevamo molti utenti. Nella filosofia della comunita' open-source (e anche di Linux) devi rilasciare il software frequentemente, e questo e' cio' che facevamo: rilasciavamo nuove versioni ogni due giorni. Non avevamo a che fare con utenti Unix, ne' hacker, ma per la maggior parte utenti Win32. Ad un certo punto capimmo che questi erano pigri/stufi di aggiornare il software ogni due giorni. Il brutto mondo Win insegna che il software si auto-aggiorna, si auto-installa e probabilmente si auto-scrive.

Cercammo di risolvere il problema tirando fuori dal motore in C la maggior parte del codice che cambiava piu' spesso, ma questo era molto difficile visto che il C non e' pensato per questo genere di cose. Una volta che LiberoPOPs si fu stabilizzato iniziammo a pensare a come risolvere meglio la cosa.

Un linguaggio di scripting/interpretato sembro' una buona scelta e dopo una lunga ricerca in rete e nei newsgroup universitari trovai LUA.. Questo non e' il luogo per dire al mondo quanto sia bello questo linguaggio quindi non ne parlero' oltre qui. Integrare l'interprete LUA in LiberoPOPs non fu cosi' difficile e FreePOPs ne e' il risultato. Ora e' davvero piu' facile scrivere/testare un plugin e (anche se non e' ancora implementato) un sistema di auto-aggiornamento e' molto facile da scrivere dato che non c'e' bisogno di ricompilare il nucleo C in quasi nessun caso.

3 File di configurazione di FreePOPs

FreePOPs non ha bisogno di una vera configurazione. La maggior parte degli utenti non dovrebbe modificare il file di configurazione. Se siete sviluppatori o utenti curiosi il file di configurazione e' `config.lua`, che si trova nella directory del programma sotto win32 o in `/etc/freepops/` in ambiente posix.

Piu' avanti vedremo come i plugin sono associati al dominio di un indirizzo di posta, e alcuni di questi plugin hanno alias per altri domini per rendere piu' facile la raccolta di news da alcuni siti. Leggete la documentazione dei plugin per maggiori informazioni su di essi, e magari inviate una mail con il vostro nuovo alias se volete che venga integrato nella prossima versione di FreePOPs.

Dalla versione 0.0.11 il file `config.lua` ha una sezione policy. In questa sezione potete escludere o accettare classi di indirizzi mail. Questo puo' essere

utile ad amministratori di rete.

4 Parametri a riga di comando di FreePOPs

La vera configurazione di FreePOPs viene impostata tramite argomenti a riga di comando. Questi sono descritti in dettaglio nelle pagine del man in ambienti Unix, qui presentiamo solo i piu' utili:

-p <port>, –port <port> Per default FreePOPs fa bind sulla porta 2000. Per modificare questo comportamento basta usare questo switch.

-P <host>:<port>, –proxy <host>:<port> Per dire a FreePOPs quale e' il vostro proxy HTTP.

-v, –verbose, -w, –veryverbose Questo dice a FreePOPs di loggare alcune informazioni utili per riportare bug.

-t <num>, –threads <num> FreePOPs puo' gestire connessioni multiple, fino a num. Il default e' 5.

In ambienti posix come Debian GNU/Linux potete avviare FreePOPs al boot come servizio standard. In questo caso gli switch a riga di comando sono memorizzati in `/etc/default/freepops`, in alcuni sistemi basati su rpm dovreste trovare lo stesso file con nome `/etc/sysconfig/freepops`.

5 Configurazione del client email

Per configurare il client email dovete cambiare le impostazioni del server POP3. Solitamente dovrete usare localhost come nome del server POP3 e 2000 come porta. Nel caso in cui installiate FreePOPs in un altro computer della vostra LAN, dovreste usare il nome di quell'host invece di localhost, mentre nel caso in cui abbiate cambiato la porta di default con lo switch `-p` dovrete immettere la stessa porta anche nel client email.. Dovete sempre usare come nome utente l'indirizzo di posta completo, per esempio `qualcosa@libero.it` invece che solo `qualcosa`. Questo e' perche' FreePOPs sceglie il plugin da caricare guardando al nome utente che deve quindi contenere tutte le informazioni. Piu' sotto presentiamo tutti i plugin e i loro domini associati.

6 Plugin

Qui diamo una descrizione dettagliata di ogni plugin.

6.1 libero.lua

Questo plugin vi permette di leggere le mail che avete in una mailbox `@libero.it`, `@iol.it`, `@inwind.it` e `@blu.it`. Cio' significa che potete ancora usare il vostro mail reader preferito invece di usare la webmail. Questo plugin agisce come un browser che sfoglia il vostro account webmail e lo fa apparire come un server POP3. Per maggiori informazioni su questo plugin potete guardare il sito web di LiberoPOPs (antenato di FreePOPs) a`http://liberopops.sourceforge.net`

Per usare questo plugin dovete usare il vostro indirizzo email completo come username e la vostra password reale come password.

6.2 tin.lua

Questo plugin vi permette di leggere le mail che avete in una mailbox `@virgilio.it`, `@tin.it`. Per usare questo plugin dovete usare il vostro indirizzo email completo come username e la vostra password reale come password.

6.3 lycos.lua

Questo plugin vi permette di leggere le mail che avete in una mailbox `@lycos.it`. Per usare questo plugin dovete usare il vostro indirizzo email completo come username e la vostra password reale come password.

6.4 popforward.lua

Questo plugin e' stato sviluppato per testare FreePOPs prima che fossero scritti altri veri plugin. Esso semplicemente inoltra richieste locali verso un vero server POP3. Puo' essere usato per mascherare un server POP3 che ha dei bug, ma se pensate di averne bisogno dovreste esaminare attentamente il codice del plugin e aggiungere controlli per migliorare il rilevamento/la prevenzione di richieste malevole, dato che il plugin stesso non e' nato con in mente la sicurezza.

Per usare questo plugin dovete modificare il file config.lua. Questo perche' non possiamo aggiungere tutti i server POP3 esistenti al file :) Il plugin richiede due argomenti, l'host POP3 e la porta (di solito `110`) su cui il server e' in ascolto. Questo e' un esempio di una riga di configurazione per questo plugin, in cui ogni indirizzo email del dominio `@virgilio.it` sono inoltrati a `in.virgilio.it:110`:

```
-- popforward plugin
freepops.MODULES_MAP["virgilio.it"]     = {
       name="popforward.lua",
       args={
```

```
            port=110,
            host="in.virgilio.it"
        }
}
```

6.5 aggregator.lua

Solitamente potete trarre beneficio dal formato RSS del W3C quando leggete news da qualche sito web. Il file RSS indicizza le news, fornendo un link verso di esse. Questo plugin puo' far si' che il vostro client di posta veda il file RSS come una mailbox da cui potete scaricare ogni news come se fosse una mail. L'unica limitazione e' che questo plugin puo' prelevare solo un sunto delle news piu' il link alle news.

Per usare questo plugin dovete usare un nome utente casuale con il suffisso `@aggregator` (es.: `foo@aggregator`) e come password l'URL del file RSS (es.: `http://www.securityfocus.com/rss/vulnerabilities.xml`). Per comodita' abbiamo aggiunto per voi alcuni alias. Questo significa che non dovrete cercare a mano l'URL del file RSS. Abbiamo aggiunto alcuni domini, per esempio `@securityfocus.com`, che possono essere usati per sfruttare direttamente il plugin aggregator con questi siti web. Per usare questi alias dovrete usare un nome utente nella forma `qualcosa@aggregatordomain` e una password a caso. Questa e' la lista di alias per il plugin aggregator.

aggregatordomain	descrizione
freepops.rss.en	http://freepops.sourceforge.net/ news (Inglese)
freepops.rss.it	http://freepops.sourceforge.net/ news (Italiano)
flatnuke.sf.net	http://flatnuke.sourceforge.net/ news (Italiano)
ziobudda.net	http://ziobudda.net/ news (sia Italiano che Inglese)
punto-informatico.it	http://punto-informatico.it/ news (Italiano)
gaim.sf.net	http://gaim.sourceforge.net/ news (Inglese)
linuxdevices.com	http://linuxdevices.com/ news (Inglese)
securityfocus.com	http://www.securityfocus.com/ new vulnerabilities (Inglese)
games.gamespot.com	http://www.gamespot.com/ computer games news (Inglese)
news.gamespot.com	http://www.gamespot.com/ GameSpot news (Inglese)
kerneltrap.org	http://kerneltrap.org news (Inglese)
linux.kerneltrap.org	http://linux.kerneltrap.org news (Inglese)

6.6 flatnuke.lua

Questo plugin e' un plugin aggregator specializzato nei siti web fatti con il CMS FlatNuke[2] , o altri siti che usano lo stesso formato delle news come il sito di

[2]HTTP://flatnuke.sourceforge.net

FreePOPs. Dato che in un sito FlatNuke le news sono memorizzate in semplici file xml questo plugin e' in grado di prelevare tutte le news, non solo le intestazioni come il plugin aggregator. Cio' e' molto utile se non vuoi sfogliare l'intero sito web per leggere le news.

Per usare questo plugin dovete avere un nome utente con il dominio `@flatnuke` (es.: `qualcosa@flatnuke`) e l'URL di una homepage flatnuke come password (es.: `http://flatnuke.sourceforge.net/`, non c'e' bisogno di URL di file RSS visto che FlatNuke mette gli RSS in una posizione nota e fissata. Ci sono alcuni alias per siti FlatNuke, vedi la documentazione del plugin aggregator per sapere cosa significa):

aggregatordomain	descrizione
freepops.en	http://freepops.sourceforge.net/ full news (Inglese)
freepops.it	http://freepops.sourceforge.net/ full news (Italiano)
flatnuke.it	http://flatnuke.sourceforge.net/ full news (Italiano)

6.7 kernel.lua

Questo e' un plugin specializzato per tenersi aggiornati sulle ultime versioni del kernel Linux. La pagina ufficiale che pubblica la lista delle versioni correnti del kernel Linux e' `http://kernel.org`. Esiste un metodo comune, per i progetti sviluppati in sistemi GNU, per aggiornare l'utente sulle modifiche effettuate nelle nuove versioni di un programma. In ogni pacchetto e' infatti presente il file `ChangeLog` che descrive le novita' apportate dagli autori. Anche il kernel Linux ha un `ChangeLog` per ogni versione nuova. Se desideri essere aggiornato sulle novita' apportate nelle versioni del kernel e, quindi, visionare il `ChangeLog`, puoi utilizzare questo plugin. Sara' sufficiente inserire come nome utente `qualcosa@kernel.org` per essere aggiornato, tramite `ChangeLog`, su ogni nuova versione, oppure `qualcosa@kernel.org.24` o `qualcosa@kernel.org.26` per visualizzare nella propria mailbox, rispettivamente, i `ChangeLog` dell'ultima versione del ramo 2.4 e del 2.6. Come password e' possibile inserire una qualsiasi stringa casuale.

7 Creare un plugin

Seguono due sezioni, la prima e' una panoramica veloce su cosa un plugin deve fare, la seconda e' un tutorial piu' dettagliato. Prima di procedere oltre suggeriamo di leggere un po' di documentazione alla base della scrittura dei plugin:

1. Dato che i plugin sono scritti in LUA dovete leggere almeno il tutorial LUA (HTTP://lua-users.org/wiki/LuaTutorial); molte grazie a chi l'ha scritto.

LUA e' un linguaggio di scripting piuttosto semplice, facile da imparare, e facile da leggere. Se siete interessati a questo linguaggio dovreste leggere IL libro su LUA ("Programming in LUA" di Roberto Ierusalimschy HTTP://www.inf.puc-rio.br/~roberto/book/). E' davvero un buon libro, credetemi.

2. Visto che dobbiamo implementare un backend POP3 dovreste sapere cos'e' il POP3. La RFC 1939 e' inclusa nella directory doc/ del pacchetto dei sorgenti di FreePOPs, ma potete prelevarla anche dalla rete HTTP://www.ietf.org/rfc/rfc

3. Leggete attentamente questo tutorial, e' lontano dall'essere ben fatto ma e' meglio di niente.

4. Il sito web contiene, nella sezione doc, un bel po' di documentazione sui sorgenti. Dovreste tenere un web browser aperto alla pagina della documentazione sui moduli LUA mentre scrivete un plugin.

5. Dopo aver creato un prototipo, dovreste leggere un plugin completo. Il plugin libero.lua e' davvero ben commentato, iniziate pure da li'.

6. Ricordate che questo software ha un forum ufficiale (HTTP://freepops.diludovico.it) e degli autori a cui potete chiedere aiuto.

7.1 Panoramica sui plugin

Un plugin e' essenzialmente un backend per un server POP3. I plugin sono scritti in LUA[3] mentre il server POP3 e' scritto in C. Qui esamineremo l'interfaccia tra il nucleo C e i plugin LUA.

7.2 L'interfaccia tra il nucleo C ed un plugin

Come abbiamo spiegato prima il frontend POP3 in C deve essere collegato ad un backend in LUA. L'interfaccia e' molto semplice se conoscete il protocollo POP3. Qui riassumiamo brevemente il significato, ma la RFC 1939 (inclusa nella directory `doc/` della distribuzione dei sorgenti) e' molto breve e facile da leggere. Come il vostro intuito dovrebbe suggerirvi il client POP3 puo' richiedere che il server POP3 conosca qualcosa delle mail che sono nella mailbox e prima o poi prelevare/cancellare dei messaggi. E questo e' esattamente cio' che fa.

Il backend deve implementare tutti i comandi POP3 (come USER, PASS, RETR, DELE, QUIT, LIST, ...) e deve restituire al frontend il risultato. Diamo un semplice esempio di una sessione POP3 dalla RFC:

[3] Il sito web del linguaggio e' HTTP://www.lua.org

```
 1  S: <wait for connection on TCP port 110>
 2  C: <open connection>
 3  S:    +OK POP3 server
 4  C:    USER linux@kernel.org
 5  S:    +OK now insert the pasword
 6  C:    PASS gpl
 7  S:    +OK linux's maildrop has 2 messages (320 octets)
 8  C:    STAT
 9  S:    +OK 1 320
10  C:    LIST
11  S:    +OK 2 messages (320 octets)
12  S:    1 320
13  S:    .
14  C:    RETR 1
15  S:    +OK 120 octets
16  S:    <the POP3 server sends message 1>
17  S:    .
18  C:    DELE 1
19  S:    +OK message 1 deleted
20  C:    QUIT
21  S:    +OK dewey POP3 server signing off (maildrop empty)
22  C:  <close connection>
23  S:  <wait for next connection>
```

In questa sessione il backend verra' chiamato per le righe 4, 6, 8, 10, 14, 18, 20 (tutte le righe `C:`) e rispettivamente le funzioni che implementano i comandi POP3 verranno chiamate in questo modo

```
user(p,"linux@kernel.org")
pass(p,"gpl")
stat(p)
list_all(p)
retr(p,1)
dele(p,1)
quit_update(p)
```

Piu' tardi chiariremo cos'e' p. Speriamo di toglierlo e renderlo implicito per completa trasparenza. E' facile capire che c'e' un mapping 1-1 tra i comandi POP3 e le chiamate a funzione del plugin. Potete vedere un plugin come l'implementazione dell'interfaccia POP3.

7.3 L'interfaccia tra un plugin e il nucleo C

Prendiamo in esame la chiamata a `pass(p,"linux@kernel.org")`. Qui il plugin dovrebbe autenticare l'utente (se c'e' un qualche tipo di autenticazione) e

informare il nucleo C del risultato. Per ottenere questo ogni funzione dei plugin deve restituire un flag di errore, per essere piu' precisi uno di questi errori:

Code	Significato
`POPSERVER_ERR_OK`	Nessun errore
`POPSERVER_ERR_NETWORK`	Errore di rete
`POPSERVER_ERR_AUTH`	Autenticazione fallita
`POPSERVER_ERR_INTERNAL`	Errore interno, segnalate il bug
`POPSERVER_ERR_NOMSG`	Il numero del messaggio e' fuori range
`POPSERVER_ERR_LOCKED`	Mailbox bloccata da altre sessioni
`POPSERVER_ERR_EOF`	Fine trasmissione, usata nel popserver_callback
`POPSERVER_ERR_TOOFAST`	Non e' possibile riconnettersi al server ora, attendere e riprovare
`POPSERVER_ERR_UNKNOWN`	Non ho idea di che errore ho trovato

Nel nostro caso i codici d'errore piu' appropriati sono `POPSERVER_ERR_AUTH` e `POPSERVER_ERR_OK`. Questo e' un caso semplice, in cui un codice d'errore e' abbastanza. Ora analizziamo il caso piu' complesso della chiamata a `list_all(p)`. Qui dobbiamo restituire un codice d'errore come prima, ma dobbiamo anche informare il nucleo C della grandezza di tutti i messaggi nella mailbox. Qui abbiamo bisogno del parametro p passato ad ogni funzione del plugin (notate che tale parametro potra' divenire implicito in futuro). `p` indica la struttura dati che il C si aspetta venga riempita chiamando funzioni appropriate come `set_mailmessage_size(p,num,size)` dove num e' il numero del messaggio e size e' la grandezza in byte. Solitamente e' molto comune mettere insieme piu' funzioni. Per esempio quando guardate la pagina di una webmail con la lista di messaggi conoscete il numero dei messaggi, la loro grandezza e lo UIDL cosi' che potete riempire la struttura dati p con tutte le informazioni per LIST, STAT, UIDL.

L'ultimo caso che esaminiamo e' `retr(p,num,data)`. Poiche' un messaggio di posta puo' essere molto grande, non e' un modo elegante di scaricare l'intero messaggio senza far si' che il client di posta si lamenti per la morte del server. La soluzione e' usare un callback. Ogni volta che un plugin ha dei dati da mandare al client dovrebbe chiamare la `popserver_callback(buffer,data)`. `data` e' una struttura opaca che il popserver necessita per compiere il suo lavoro (notate che questo parametro potra' venire rimosso per semplicita'). In alcuni casi, per esempio se sapete che il messaggio e' piccolo o state lavorando su una rete veloce, potete prelevare l'intero messaggio e mandarlo, ma ricordate che questo consuma piu' memoria.

7.4 L'arte di scrivere plugin (tutorial sui plugin)

In questa sezione scriveremo un plugin passo passo, esaminando ogni dettaglio importante. Non scriveremo un vero e completo plugin poiche' puo' diventare un po' difficile da seguire, ma creeremo una webmail ad-hoc per i nostri scopi.

7.4.1 (step 1) Lo scheletro

La prima cosa che faremo sara' copiare il file `skeleton.lua` in `foo.lua` (perche' scriveremo il plugin per la webmail *foo.xx* , *xx* sta per un dominio vero, ma non vogliamo menzionare alcun sito qui...). Ora con il vostro editor migliore (suggeriamo vim su Unix e scintilla per win32, visto che supportano il syntax highlighting per LUA, ma qualsiasi altro editor di testo va bene) aprite `foo.lua` e cambiate le prime righe aggiungendo il nome del plugin, la versione, il vostro nome, il vostro indirizzo email e un breve commento, nei posti appropriati.

```
-- ************************************************************************** --
--  FreePOPs @--put domain here-- webmail interface
--
--  $Id: manual-it.lyx,v 1.11 2004/07/06 10:02:37 ncocchiaro Exp $
--
--  Released under the GNU/GPL license
--  Written by --put Name here-- <--put email here-->
-- ************************************************************************** --

PLUGIN_VERSION = "--put version here--"
PLUGIN_NAME = "--put name here--"
```

Ora abbiamo un plugin vuoto, ma non e' abbastanza per iniziare a farci hacking. Dobbiamo aprire il file `config.lua` (nella distribuzione win32 si trova nella directory principale, mentre nella distribuzione Unix e' in `/etc/freepops/`; altre copie di questo file possono essere incluse nelle distribuzioni, ma sono copie di backup) e aggiungete una riga come questa

```
-- foo plugin
freepops.MODULES_MAP["foo.xx"]      = {name="foo.lua"}
```

all'inizio del file. Prima di finire il primo passo dovreste provare se il plugin viene correttamente attivato da FreePOPs quando necessario. Per questo dovremo aggiungere alcune righe a `foo.lua`, in particolare dovremo aggiungere un valore di ritorno di errore a `user()`.

```
-- ------------------------------------------------------------------------ --
-- Must save the mailbox name
function user(pstate,username)
        return POPSERVER_ERR_AUTH
end
```

Ora la funzione user fallisce sempre, restituendo un errore di autenticazione. Dovrete ora lanciare FreePOPs (se e' gia' in esecuzione non e' necessario farlo ripartire) e lanciare telnet (sotto win32 dovreste aprire un prompt DOS, sotto Unix avrete una shell) e digitate `telnet localhost 2000` e poi digitate `user test@foo.xx`.

```
tassi@garfield:~$ telnet localhost 2000
Trying 127.0.0.1...
Connected to garfield.
Escape character is '^]'.
+OK FreePOPs/0.0.10 pop3 server ready
user test@foo.xx
-ERR AUTH FAILED
Connection closed by foreign host.
```

Il server risponde chiudendo la connessione e stampando un messaggio di autorizzazione fallita (va bene, dato che la funzione `user()` del nostro plugin restituisce questo errore). Nel file standard error (la console sotto Unix, il file `stderr.txt` sotto Windows) vengono stampati i messaggi d'errore, non vi prestate attenzione per ora.

7.4.2 (step 2) Il login

La procedura di login e' la prima cosa da fare. Il protocollo POP3 ha due comandi per il login, *user* e *pass*. Prima il client esegue uno user, poi dice al server la password. Come abbiamo gia' visto nella panoramica questo significa che prima verra' eseguito user() e poi `pass()`. Questo e' un esempio di login:

```
tassi@garfield:~$ telnet localhost 2000
Trying 127.0.0.1...
Connected to garfield.
Escape character is '^]'.
+OK FreePOPs/0.0.10 pop3 server ready
user test@foo.xx
+OK PLEASE ENTER PASSWORD
pass hello
-ERR AUTH FAILED
```

Se lanciate FreePOPs con il parametro `-w` dovreste leggere questo sullo standard error/standard output:

```
freepops started with loglevel 2 on a little endian machine.
Cannot create pid file "/var/run/freepopsd.pid"
DBG(popserver.c, 162): [5118] ?? Ip address 0.0.0.0 real port 2000
DBG(popserver.c, 162): [5118] ?? Ip address 127.0.0.1 real port 2000
```

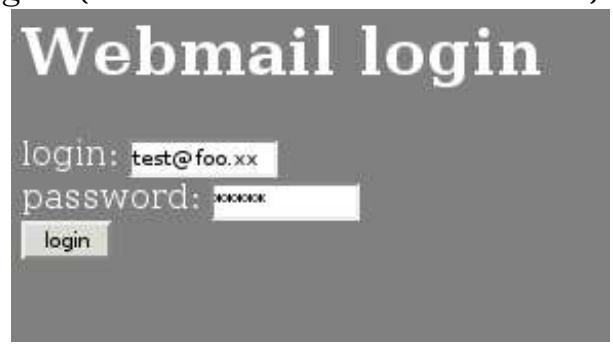
```
DBG(popserver.c, 162): [5118] -> +OK FreePOPs/0.0.10 pop3 server ready
DBG(popserver.c, 162): [5118] <- user test@foo.xx
DBG(log_lua.c,  83): (@src/lua/foo.lua, 37) : FreePOPs plugin 'Foo web mail' version '0.
*** the user wants to login as 'test@foo.xx'
DBG(popserver.c, 162): [5118] -> +OK PLEASE ENTER PASSWORD
DBG(popserver.c, 157): [5118] <- PASS *********
*** the user inserted 'hello' as the password for 'test@foo.xx'
DBG(popserver.c, 162): [5118] -> -ERR AUTH FAILED
AUTH FAILED
DBG(threads.c,  81): thread 0 will die
```

il plugin e' stato modificato un po' per memorizzare i dati dell'utente e stampare delle informazioni di debug. Questo e' il plugin che ha dato questo output:

```
foo_globals= {
username="nothing",
password="nothing"
}
-- ------------------------------------------------------------------------ --
-- Must save the mailbox name
function user(pstate,username)
foo_globals.username = username
print("*** the user wants to login as '"..username.."'")
return POPSERVER_ERR_OK
end
-- ------------------------------------------------------------------------ --
-- Must login
function pass(pstate,password)
foo_globals.password = password
print("*** the user inserted '"..password..
    "' as the password for '"..foo_globals.username.."'")
return POPSERVER_ERR_AUTH end
-- ------------------------------------------------------------------------ --
-- Must quit without updating
function quit(pstate)
return POPSERVER_ERR_OK
end
```

Qui vediamo delle importanti novita'. Per prima cosa, la tabella `foo_globals` che contiene tutti i valori globali (valori che devono essere a disposizione di chiamate a funzioni successive) di cui abbiamo bisogno. Per ora ci abbiamo messo il nome utente e la password. La funzione`user()` ora memorizza il nome utente passato nella tabella `foo_globals` e stampa qualcosa sullo standard output. La funzione `pass()` allo stesso modo memorizza la password nella tabella globale e stampa qualcosa. La funzione `quit()` restituisce semplicemente `POPSERVER_ERR_OK` per far felice FreePOPs.

Ora che sappiamo come FreePOPs si comportera' durante il login dobbiamo implementare il login nella webmail, ma prima decommentiamo alcune righe nella funzione `init()` (chiamata alla partenza del plugin), la quale carica il modulo `browser.lua` (il modulo usato per fare login nella webmail). Ecco la pagina di login della webmail vista con Mozilla e il codice sorgente della stessa pagina (con Mozilla lo si vede con Ctrl-U).




```
<html>
<head>
<title>foo.xx webmail login</title>
</head>
<body style="background-color : grey; color : white">
<h1>Webmail login</h1>
<form name="webmail" method="post" action="http://localhost:3000/">
login: <input type="text" size="10" name="username"> <br>
password: <input type="password" size="10" name="password"> <br>
<input type="submit" value="login">
</form>
</body>
</html>
```

Abbiamo due campi di input, uno chiamato username e uno chiamato password. Quando l'utente fa click su login il browser web eseguira' `POST` sul `HTTP://localhost:3000/` contenuto del form (ho usato un indirizzo locale per comodita', ma dovrebbe essere qualcosa come `HTTP://webmail.foo.xx/login.php`). Questo e' cio' che il browser invia:

```
POST / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.6) Gecko/20040614 Firefox/0.8 A
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

username=test%40foo.xx&password=hello
```

Non ci interessa la prima parte (l'header HTTP, visto che il modulo browser se ne occupera'), bensi' l'ultima, i dati inviati. Poiche' i campi del form erano username e password, i dati inviati sono `username=test%40.foo.xx&password=hello`. Ora vogliamo riprodurre la stessa richiesta HTTP con il nostro plugin. Questo e' il semplice codice che fara' proprio quello.

```
-- ------------------------------------------------------------------------ --
-- Must login
function pass(pstate,password)
foo_globals.password = password

print("*** the user inserted '"..password..
     "' as the password for '"..foo_globals.username.."'")

-- create a new browser
local b = browser.new()

-- store the browser object in globals
foo_globals.browser = b

       -- create the data to post
       local post_data = string.format("username=%s&password=%s",
              foo_globals.username,foo_globals.password)
       -- the uri to post to
       local post_uri = "http://localhost:3000/"

       -- post it
       local file,err = nil, nil

       file,err = b:post_uri(post_uri,post_data)

       print("we received this webpage: ".. file)
       return POPSERVER_ERR_AUTH
end
```

Prima creiamo un oggetto browser, poi mettiamo insieme `post_uri` e `post_data` usando un semplice `string.format` (una funzione simile a printf). E questa e' la richiesta risultante

```
POST / HTTP/1.1
```

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.6) Gecko/20040322 Firefox/0.8
Pragma: no-cache
Accept: */*
Host: localhost
Content-Length: 35
Content-Type: application/x-www-form-urlencoded

username=test@foo.xx&password=hello
```

questo e' essenzialmente come lo volevamo fare (dovremmo fare url-encode dei post data con `curl.escape()`). Abbiamo salvato l'oggetto browser sulla tabella globale, perche' vogliamo usare lo stesso browser tutte le volte.

Ora che abbiamo fatto login, vogliamo controllare la pagina risultante, e magari estrarre un ID di sessione che useremo poi. Questo e' il codice per estrarre l'ID di sessione e la pagina HTML che abbiamo ricevuto in risposta alla richiesta di login

```
        ... come sopra qui ...

        print("we received this webpage: ".. file)

        -- search the session ID
        local _,_,id = string.find(file,"session_id=(%w+)")

        if id == nil then
               return POPSERVER_ERR_AUTH
        end

        foo_globals.session_id = id
        return POPSERVER_ERR_OK
end
```

e questa e' la pagina web restituita.


```
<html>
<head>
<title>foo.xx webmail</title>
</head>
```

```
<body style="background-color : grey; color : white">
<h1>Webmail - test@foo.xx</h1>
Login done! click here to view the inbox folder.
<a href="http://localhost:3000/inbox.php?session_id=ABCD1234">inbox</a>
</body>
</html>
```

Notate che abbiamo estratto l'ID di sessione usando `string.find(file,"session_id=(%w+)")`. Questa e' una funzione molto importante nella libreria LUA e, anche se e' descritta nel tutorial LUA su HTTP://lua-users.org, parleremo un po' di capture anche qui. Guardiamo i sorgenti della pagina. Ci interessa la riga
`<a href="HTTP://localhost:3000/inbox.php?session_id=ABCD1234">inbox</a>`
che contiene il session_id che vogliamo catturare. La nostra espressione e' *`session_id=(%w+)`* che significa che vogliamo trovare tutte le stringhe che iniziano con `session_id=` e poi continuano con uno o piu' caratteri alfanumerici. Siccome abbiamo scritto `%w+` tra parentesi tonde, intendiamo catturare il contenuto delle parentesi (la parte alfanumerica). Cosi' string.find restituira' tre valori, i primi due sono ignorati (assegnati alla variabile dummy _) mentre il terzo e' la stringa catturata (nel nostro caso `ABCD1234`). Il tutorial LUA su lua-users e' molto ben fatto e su HTTP://sf.net/projects/lua-users potete trovare il LUA short reference che e' un riassunto di tutte le funzioni standard LUA ed e' anche un gran bel documento (mille grazie a Enrico Colombini). Se vi piace molto LUA dovreste comprare IL libro su LUA chiamato *"Programming in Lua"* di Roberto Ierusalimschy (consideratelo il K&R per LUA).

7.4.3 (step 3) Ottenere la lista dei messaggi

Ora dovremo implementare la funzione `stat()`. La stat e' probabilmente la funzione piu' importante. Essa deve prelevare la lista dei messaggi nella webmail, il loro UIDL e la loro grandezza. Nel nostro esempio useremo il modulo mlex per tirare fuori le informazioni importanti dalla pagina, ma potete usare il modulo per le stringhe di LUA per fare la stessa cosa con i capture. Questa e' la nostra pagina inbox

e questo e' il corpo HTML (solo i primi due messaggi sono riportati)

```
<h1>test@foo.xx - inbox (1/2)</h1>
<form name="inbox" method="post" action="/delete.php">
<input type="hidden" name="session_id" value="ABCD1234">
<table>
<tr><th>From</th><th>subject</th><th>size</th><th>date</th></tr>
<tr>
  <td><b>friend1@foo1.xx</b></td>
  <td><b><a href="/read.php?session_id=ABCD1234&uidl=123">ok!</a></b></td>
  <td><b>20KB</b></td>
  <td><b>today</b></td>
  <td><input type="checkbox" name="check_123"></td>
</tr>
<tr>
  <td>friend2@foo2.xx</td>
  <td><a href="/read.php?session_id=ABCD1234&uidl=124">Re: hi!</a></td>
  <td>12KB</td>
  <td>yesterday</td>
  <td><input type="checkbox" name="check_124"></td>
</tr>
</table>
<input type="submit" value="delete marked">
</form>
<a href="/inbox.php?session_id=ABCD1234&page=2">go to next page</a>
</body>
```

Abbiamo prelevato l'HTML usando il browser e il metodo`get_uri()` (ricordate che la URI per l'inbox era nella pagina di login). Come vedete i messaggi sono in una tabella, e tale tabella ha la stessa struttura per ogni messaggio.

Proprio questo e' il posto in cui usare mlex. Semplicemente, prendete tutto cio' che c'e' tra <tr> e </tr> di una riga di un messaggio e cancellate tutto tranne i nomi dei i tag. Poi sostituite tutti gli spazi vuoti (chiameremo spazio la stringa tra due tag) con un".*". Ecco cosa abbiamo ottenuto (dovrebbe essere tutto sulla stessa riga, qui andiamo a capo per mancanza di spazio) dal primo messaggio.

```
.*<tr>.*<td>.*<b>.*</b>.*</td>.*<td>.*<b>.*<a>.*</a>.*</b>.*</td>.*
<td>.*<b>.*</b>.*</td>.*<td>.*<b>.*</b>.*</td>.*
<td>.*<input>.*</td>.*</tr>
```

Questa espressione e' usata per fare match con la riga della tabella che contiene informazioni sul messaggio. Ora copiate e incollate a parte la riga e sostituite ogni spazio e ogni tag con O (la lettera, non la cifra 0) o X. Mettete una X nei campi interessanti (nel nostro esempio la grandezza e il tag input, che contiene lo UIDL del messaggio).

```
0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0
<0>0<0>X<0>0<0>0<0>0<0>0<0>0<0>0<0>0
<0>0<X>0<0>0<0>0<0>
```

Mentre la prima espressione verra' usata per fare match con la riga della tabella, questa verra' usata per estrarre i campi importanti. Questo codice lancia mlex sull'HTML e riempie la struttura dati popstate con i dati catturati.

```
-- ------------------------------------------------------------------------ --
-- Fill the number of messages and their size
function stat(pstate)
      local file,err = nil, nil
      local b = foo_globals.browser
      file,err = b:get_uri("http://localhost:3000/inbox.php?session_id=".. 
              foo_globals.session_id)
      local e = ".*<tr>.*<td>.*<b>.*</b>.*</td>.*<td>.*<b>.*<a>"..
              ".*</a>.*</b>.*</td>.*<td>.*<b>.*</b>.*</td>.*<td>.*"..
              "<b>.*</b>.*</td>.*<td>.*<input>.*</td>.*</tr>"
      local g = "O<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0"..
              "<0>0<0>X<0>0<0>0<0>0<0>0<0>0<0>0<0>0<0>0<X>0<0>0<0>"
      local x = mlex.match(file,e,g)
      --debug print
      x:print()

      set_popstate_nummesg(pstate,x:count())
      for i=1,x:count() do
              local _,_,size = string.find(x:get(0,i-1),"(%d+)")
              local _,_,size_mult_k = string.find(x:get(0,i-1),"([Kk][Bb])")
```

```
                local _,_,uidl = string.find(x:get(1,i-1),"check_(%d+)")

                if size_mult_k ~= nil then
                        size = size * 1024
                end
                if size_mult_m ~= nil then
                        size = size * 1024 * 1024
                end

                set_mailmessage_size(pstate,i,size)
                set_mailmessage_uidl(pstate,i,uidl)
        end

        return POPSERVER_ERR_OK
end
```

Il risultato di `x:print()` e' il seguente

```
{'20KB','input type="checkbox" name="check_123"'}
```

and the telnet session follows

```
+OK FreePOPs/0.0.11 pop3 server ready
user test@foo.xx
+OK PLEASE ENTER PASSWORD
pass secret
+OK ACCESS ALLOWED
stat
+OK 1 20480
quit
+OK BYE BYE, UPDATING
```

Non abbiamo indicato come abbiamo aggiunto la riga `return POPSERVER_ERR_OK` alla funzione `quit()`. Il codice sorgente riportato sopra usa mlex per estrarre le due stringhe interessanti, poi la scorre cercando la grandezza, il suo moltiplicatore e lo UIDL. Di seguito imposta gli attributi dei messaggi. Potete vedere che abbiamo processato solo il primo messaggio. Per processare gli altri dobbiamo informare il modulo mlex che il tag <b> e' opzionale (potete notare che solo il primo messaggio e' in grassetto). Quindi cambiamo le espressioni in

```
.*<tr>.*<td>[.*]{b}.*{/b}[.*]</td>.*<td>[.*]{b}.*<a>.*</a>.*{/b}[.*]</td>.*
<td>[.*]{b}.*{/b}[.*]</td>.*<td>[.*]{b}.*{/b}[.*]</td>.*
<td>.*<input>.*</td>.*</tr>
```

and

```
0<0>0<0>[0]{0}0{0}[0]<0>0<0>[0]{0}0<0>0<0>0{0}[0]<0>0
<0>[0]{0}X{0}[0]<0>0<0>[0]{0}0{0}[0]<0>0
<0>0<X>0<0>0<0>
```

Ora il comando stat risponde con `+OK 4 45056` e la stampa di debug e'

```
{'20KB','input type="checkbox" name="check_123"'}
{'12KB','input type="checkbox" name="check_124"'}
{'10KB','input type="checkbox" name="check_125"'}
{'2KB','input type="checkbox" name="check_126"'}
```

Ora abbiamo una vera e propria funzione stat che riempie la struttura dati popstate con le informazioni che il server POP necessita per rispondere ad una richiesta di stat. Poiche' le richieste list, uidl, list_all e uidl_all possono essere soddisfatte con gli stessi dati, useremo la funzione standard fornita dal modulo common.lua. Esso verra' spiegato nel prossimo passo, ma dobbiamo aggiungere due righe importanti alla funzione`stat()` per evitare una doppia chiamata.

```
function stat(pstate)
       if foo_globals.stat_done == true then return POPSERVER_ERR_OK end

       ... the same code here ...

       foo_globals.stat_done = true
       return POPSERVER_ERR_OK
end
```

La funzione piu' importante e' pronta, ma dobbiamo fare delle precisazioni. Primo, mlex e' molto comodo a volte, ma potreste trovare piu' utile la libreria per le stringhe di LUA o la libreria regualerexp (espressioni regolari estese posix) per raggiungere lo stesso scopo. Secondo, questa implementazione si ferma alla prima pagina di inbox. Dovreste visitare tutte le pagine di inbox, forse usando la funzione `do_until()` nella libreria `support.lua` (che descriveremo brevemente alla fine di questo tutorial). Terzo, non facciamo nessun controllo degli errori. Per esempio la variabile file puo' essere nil e dobbiamo controllare queste cose per fare un buon plugin.

7.4.4 (step 4) Le funzioni comuni

Il modulo comune ci da' alcune funzioni precotte che dipendono solo da una `stat()` ben implementata (una stat che puo' essere chiamata piu' di una volta). Ecco la nostra implementazione di queste funzioni

```
-- ------------------------------------------------------------------------ --
-- Fill msg uidl field
function uidl(pstate,msg) return common.uidl(pstate,msg) end

-- ------------------------------------------------------------------------ --
-- Fill all messages uidl field
function uidl_all(pstate) return common.uidl_all(pstate) end

-- ------------------------------------------------------------------------ --
-- Fill msg size
function list(pstate,msg) return common.list(pstate,msg) end

-- ------------------------------------------------------------------------ --
-- Fill all messages size
function list_all(pstate) return common.list_all(pstate) end

-- ------------------------------------------------------------------------ --
-- Unflag each message merked for deletion
function rset(pstate) return common.rset(pstate) end

-- ------------------------------------------------------------------------ --
-- Mark msg for deletion
function dele(pstate,msg) return common.dele(pstate,msg) end

-- ------------------------------------------------------------------------ --
-- Do nothing
function noop(pstate) return common.noop(pstate) end
```

ma prima aggiungete il codice per caricare il modulo comune alla vostra funzione `init()`.

```
        ... the same code ..

        -- the common module
        if freepops.dofile("common.lua") == nil then
                  return POPSERVER_ERR_UNKNOWN
        end

        -- checks on globals
        freepops.set_sanity_checks()

        return POPSERVER_ERR_OK
end
```

7.4.5 (step 5) Cancellazione dei messaggi

La cancellazione di un messaggio e' solitamente un normale POST e un esempio di `post_data` e' `session_id=ABCD1234&check_124=on&check_126=on`. Il codice segue

```
-- ----------------------------------------------------------------------- --
-- Update the mailbox status and quit
function quit_update(pstate)
      -- we need the stat
      local st = stat(pstate)
      if st ~= POPSERVER_ERR_OK then return st end

      -- shorten names, not really important
      local b = foo_globals.b
      local post_uri = b:whereamewe() .. "/delete.php"
      local session_id = foo_globals.session_id
      local post_data = "session_id=" .. session_id .. "&"

      -- here we need the stat, we build the uri and we check if we
      -- need to delete something

      local delete_something = false;
      for i=1,get_popstate_nummesg(pstate) do
            if get_mailmessage_flag(pstate,i,MAILMESSAGE_DELETE) then
                  get_mailmessage_uidl(pstate,i).. "=on&"
                delete_something = true
            end
      end

      if delete_something then
            b:post_uri(post_uri,post_data)
      end

      return POPSERVER_ERR_OK
end
```

Considerate che facciamo il POST solo se almeno un messaggio e' segnato per la cancellazione. Un'altra cosa importante da tenere a mente e' che fare un solo POST per tutti i messaggi e' meglio che farne uno per ognuno. Quando possibile dovreste ridurre il numero di richieste HTTP al massimo dato che e' qui che portiamo FreePOPs da lepre a tartaruga.

7.4.6 (step 6) Scaricare messaggi

Potrete chiedervi perche' parliamo di questo argomento solo al punto 6, d'altronde avere la posta e' probabilmente cio' che volete da un plugin. Implementare la

funzione `retr()` e' di solito facile. Dipende in realta' dalla webmail, ma qui parleremo del caso semplice, mentre alla fine del tutorial vedrete come gestire webmail complesse. Il caso base e' quello in cui la webmail ha un pulsante per salvare i messaggi, e il messaggio salvato e' un file di testo semplice che contiene sia l'header che il corpo del messaggio. Ci sono solo due questioni interessanti in questo caso, e cioe' quelle relativa ai messaggi grandi al punto.

I messaggi grandi causano timeout. Si', il modo piu' semplice di scaricare un messaggio e' chiamare `b:get_uri()` e memorizzare il messaggio in una variabile, poi mandarlo al client di posta con `popserver_callback()`. Ma pensate che una mail da 5MB, scaricata con una connessione DSL da 640Kbps, alla piena velocita' di 80KBps, impiega 64 secondi di download. Questo significa che il vostro plugin non mandera' dati al client di posta per oltre un minuto, facendo si' che il client si disconnetta da FreePOPS pensando che il server POP3 sia morto. Per cui, dobbiamo mandare dati al client di posta appena possibile. Per questo abbiamo la funzione `b:pipe_uri()` che chiama un callback ogni volta che ha dei dati freschi. Il codice seguente e' la funzione di callback factory, che crea un nuovo callback da passare al metodo `pipe_uri` del browser.

```
-----------------------------------------------------------------------------
-- The callback factory for retr
--
function retr_cb(data)
        local a = stringhack.new()
        return function(s,len)
                s = a:dothack(s).."\0"
                popserver_callback(s,data)
                return len,nil
        end
end
```

Qui potete vedere che il callback usa `popserver_callback()` per passare dati al client di posta, ma prima di fare cio' manipola i dati con lo stringhack. Ma questa e' la seconda questione interessante.

Il protocollo POP3 deve terminare la risposta al comando retr con una riga che contiene solo tre byte, "`.\r\n`". Ma che succede se una riga, dentro il corpo della mail, e' un semplice punto? Dobbiamo cambiarlo in "`..\r\n`". Non e' cosi' difficile, una `string.gsub(s,"\r\n.\r\n","\r\n..\r\n")` e' tutto cio' che ci serve... ma non nel caso dei callback. Il callback di invio verra' chiamato con dati freschi, e piu' di una volta se la mail e' grande. E se il pattern cercato e' troncato tra due chiamate il metodo `string.gsub()` fallira'. Per questo il modulo stringhack ci viene incontro. L'oggetto `a` vive fintantoche' la funzione di callback viene chiamata (vedi il tutorial LUA) e terra' a mente che il pattern cercato puo' essere troncato.

Infine, il codice della `retr()`.

```
-- ------------------------------------------------------------------------ --
-- Get message msg, must call
-- popserver_callback to send the data
function retr(pstate,msg,pdata)
        -- we need the stat
        local st = stat(pstate)
        if st ~= POPSERVER_ERR_OK then return st end

        -- the callback
        local cb = retr_cb(data)

        -- some local stuff
       local session_id = foo_globals.session_id
       local b = internal_state.b
       local uri = b:wherearewe() .. "/download.php?session_id="..session_id..
                "&message="..get_mailmessage_uidl(pstate,msg)

        -- tell the browser to pipe the uri using cb
        local f,rc = b:pipe_uri(uri,cb)
        if not f then
                log.error_print("Asking for "..uri.."\n")
                log.error_print(rc.."\n")
                return POPSERVER_ERR_NETWORK
        end
end
```

7.4.7 (step 7) Test

Per fare un buon plugin ci vuole un sacco di testing. Dovreste cercare beta tester presso il forum di FreePOPs (HTTP://freepops.diludovico.it) e chiedere agli autori del software di includerlo nella distribuzione principale. Dovreste anche leggere il contratto della webmail, controllare se c'e' qualcosa come "*Non usero' mai un server webmail->pop3 per leggere la mia posta*" e inviare una copia agli autori del software.

7.4.8 (step 8) La tanto anticipata parte finale del tutorial

Ci sono un sacco di cose che abbiamo tralasciato.

La multi-page stat e' la vera buona implementazione per `stat()`. Abbiamo detto sopra che la nostra implementazione elenca solo i messaggi nella prima pagina. Il codice per il parsing e l'estrazione di informazioni interessanti da una pagina e' gia' scritto, ci serve solo una funzione che controlli se siamo all'ultima pagina e se no cambi il valore di una variabile

uri. La variabile uri in questione sara' usata dalla funzione di prelevamento. In questo caso dovreste usare il modulo di supporto con il ciclo do_until. Questo e' un semplice esempio di do_until()

```
-- ------------------------------------------------------------------------ --
-- Fill the number of messages and their size
function stat(pstate)
        ... some code as before ...

        -- this string will contain the uri to get. it may be updated by
        -- the check_f function, see later
        local uri = string.format(libero_string.first,popserver,session_id)

        -- The action for do_until
        --
        -- uses mlex to extract all the messages uidl and size
        local function action_f (s)
                -- calls match on the page s, with the mlexpressions
                -- statE and statG
                local x = mlex.match(s,e,g)

                -- the number of results
                local n = x:count()

                if n == 0 then return true,nil end

                -- this is not really needed since the structure
                -- grows automatically... maybe... don't remember now
                local nmesg_old = get_popstate_nummesg(pstate)
                local nmesg = nmesg_old + n
                set_popstate_nummesg(pstate,nmesg)

                -- gets all the results and puts them in the popstate structure
                        ... some code as before ...

                        set_mailmessage_size(pstate,i+nmesg_old,size)
                        set_mailmessage_uidl(pstate,i+nmesg_old,uidl)
                end

                return true,nil
        end

        -- check must control if we are not in the last page and
        -- eventually change uri to tell retrive_f the next page to retrive
        local function check_f (s)
                local tmp1,tmp2 = string.find(s,next_check)
                if tmp1 ~= nil then
```

```
                        -- change retrive behaviour
                        uri = "--build the uri for the next page--"

                        -- continue the loop
                        return false
                else
                        return true
                end
        end

        -- this is simple and uri-dependent
        local function retrive_f ()
                local f,err = b:get_uri(uri)
                if f == nil then
                        return f,err
                end

                local _,_,c = string.find(f,"--timeout string--")
                if c ~= nil then
                        internal_state.login_done = nil
                        session.remove(key())
                        local rc = libero_login()
                        if rc ~= POPSERVER_ERR_OK then
                                return nil,"Session ended,unable to recover"

                        uri = "--uri for the first page--"
                        return b:get_uri(uri)
                 end

                 return f,err
        end

        -- initialize the data structure
        set_popstate_nummesg(pstate,0)

        -- do it
        if not support.do_until(retrive_f,check_f,action_f) then
                  log.error_print("Stat failed\n")
                  session.remove(key())
                  return POPSERVER_ERR_UNKNOWN
        end

        -- save the computed values
        internal_state["stat_done"] = true
        return POPSERVER_ERR_OK
end
```

Le uniche cose strane sono la funzione di prelevamento e quel che serve per salvare la sessione. Dato che le webmail a volte fanno timeout dovreste controllare se la pagina prelevata sia valida o no, ed eventualmente ritentare il login. Il salvataggio della sessione e' la prossima questione.

Salvare la sessione e' il modo per rendere FreePOPs davvero simile ad un browser. Cio' significa che la prossima volta che controllate la posta FreePOPs ricarichera' semplicemente la pagina inbox senza rifare il login. Per fare questo avete bisogno di una funzione `key()` che crea un ID unico per ogni sessione

```
-----------------------------------------------------------------------------
-- The key used to store session info
--
-- This key must be unique for all webmails, since the session pool is one
-- for all the webmails
--
function key()
        return foo_globals.username .. foo_globals.password
end
```

e una funzione di serializzazione `foo_globals`

```
-----------------------------------------------------------------------------
-- Serialize the internal state
--
-- serial.serialize is not enough powerful to correcly serialize the
-- internal state. The field b is the problem. b is an object. This means
-- that it is a table (and no problem for this) that has some field that are
-- pointers to functions. this is the problem. there is no easy way for the
-- serial module to know how to serialize this. so we call b:serialize
-- method by hand hacking a bit on names
--
function serialize_state()
        internal_state.stat_done = false;
        return serial.serialize("foo_globals",foo_globals) ..
                internal_state.b:serialize("foo_globals.b")
end
```

Ora dovete dire a FreePOPs di salvare lo stato nella funzione `quit_update()` e caricarlo nella `pass()`. Questa e' la nuova struttura `pass()`

```
function pass(pstate,password)
        -- save the password
```

```
        internal_state.password = password

        -- eventually load session
        local s = session.load_lock(key())

        -- check if loaded properly
        if s ~= nil then
                 -- "\a" means locked
                 if s == "\a" then
                          log.say("Session for "..internal_state.name..
                              " is already locked\n")
                          return POPSERVER_ERR_LOCKED
                 end

                 -- load the session
                 local c,err = loadstring(s)
                 if not c then
                          log.error_print("Unable to load saved session: "..err)
                          return foo_login()
                 end

                 -- exec the code loaded from the session string
                 c()

                log.say("Session loaded for " .. internal_state.name .. "@" ..
                         internal_state.domain ..
                         "(" .. internal_state.session_id .. ")\n")

                return POPSERVER_ERR_OK
        else
                -- call the login procedure
                return foo_login()
        end
end
```

dove `foo_login()` e' la vecchia funzione `pass()` con cambiamenti minori. Non dimenticate di chiamare `session.unlock(key())` nella funzione `quit()`, perche' dovrete rilasciare la sessione in caso di fallimento (e `quit()` viene chiamata qui) e salvare la sessione in `quit_update()`

```
-- save fails if it is already saved
session.save(key(),serialize_state(),session.OVERWRITE)
-- unlock is useless if it have just been saved, but if we save
-- without overwriting the session must be unlocked manually
-- since it would fail instead overwriting
```

```
session.unlock(key())
```

La funzione top() e' piuttosto complessa. Non la descriveremo in modo completo, ma suggeriamo di guardare il plugin `libero.lua` se il server web che vi manda i messaggi supporta il campo "`Range:`" nelle richieste HTTP, o il plugin HTTP request field, o il plugin `tin.lua` se il server deve essere interrotto in malo modo. Ricordate che la `top()` ha bisogno che qualcuno conti le righe e qui abbiamo di nuovo il modulo stringhack, che conta ed eventualmente elimina delle righe.

Il javascript e' l'inferno delle webmail. I Javascript possono fare qualsiasi cosa e dovrete leggerli per emulare cio' che fanno. Per esempio potrebbero aggiungere alcuni cookie (e dovrete fare lo stesso a mano con `b:add_cookie()` come in tin.lua) oppure possono cambiare alcuni campi di form (come nel codice di bilanciamento del carico in `libero.lua`).

I cookie sono abbastanza appetibili per noi, visto che il modulo browser se ne occupa al posto nostro.

I file standard sono decisamente dipendenti dal sistema. Sotto Windows dovrete costantemente guardare `stderr.txt` e `stdout.txt`, mentre sotto Unix dovrete solo lanciare FreePOPs con il parametro `-w` e guardare la console.

La forza bruta si chiama ethereal. A volte le cose non funzionano nel modo giusto e l'unico modo per fare debug e' attivare curl debugging per vedere cosa fa FreePOPs (`b.curl:setopt(curl.OPT_VERBOSE,1)`) e sniffare cio' che fa un vero browser.

Il modo open source e' il modo migliore di avere software di buona qualita' Questo significa che dovrete rilasciare molto spesso il vostro plugin nella fase di sviluppo e interagire molto con i vostri tester. Fidatevi, funziona, o leggete "*The cathedral and the bazaar*" di Eric Raymond.

Il modulo mimer e' molto beta mentre scriviamo queste righe, ma e' cio' di cui avete bisogno se siete nel caso sfortunato di una webmail che non ha un pulsante per salvare i messaggi. Il plugin `lycos.lua` e' un esempio di cosa puo' fare. La principale funzione interessante e' `mimer.pipe_msg()` che prende un header di messaggio, il testo del corpo (in html o testo semplice) e gli URI di alcuni attachment, scaricati al volo, composti in una vera e propria mail che viene inoltrata al client di posta.

8 Segnalare un bug

Quando avete problemi o pensate di avere trovato un bug, vi preghiamo di seguire alla lettera questo *iter*:

1. Aggiornate alla versione piu' recente di FreePOPs.
2. Cercate di riprodurre il bug, se questo non e' facilmente riproducibile siamo sfortunati. Si puo' ancora tentare qualcosa, se il software e' andato in crash potreste compilarlo dai sorgenti, installare valgrind, lanciare freepopsd con valgrind e sperare che i messaggi d'errore siano interessanti.
3. Pulite i file di log
4. Lanciate FreePOPs con lo switch -w
5. Riproducete il bug
6. Inviate agli sviluppatori il log, piu' ogni altra informazione utile come che tipo di sistema avete e come riprodurre il bug.

9 Autori

Questo manuale e' stato scritto da Enrico Tassi <gareuselesinge@users.sourceforge.n
e rivisto e tradotto da Nicola Cocchiaro <ncocchiaro@users.sourceforge.net>

9.1 Sviluppatori

FreePOPs e' sviluppato da:

- Enico Tassi <gareuselesinge@users.sourceforge.net>
- Alessio Caprari <alessiofender@users.sourceforge.net>
- Nicola Cocchiaro <ncocchiaro@users.sourceforge.net>
- Simone Vellei <simone_vellei@users.sourceforge.net>

LiberoPOPs e' sviluppato da:

- Enico Tassi <gareuselesinge@users.sourceforge.net>
- Alessio Caprari <alessiofender@users.sourceforge.net>
- Nicola Cocchiaro <ncocchiaro@users.sourceforge.net>
- Simone Vellei <simone_vellei@users.sourceforge.net>
- Giacomo Tenaglia <sonicsmith@users.sourceforge.net>

10 Ringraziamenti

Ringraziamenti speciali vanno agli utenti che hanno testato il software, agli hacker che hanno reso possibile avere un ambiente di sviluppo affidabile e libero come il sistema Debian GNU/Linux.