IPv4-to-v6 トランスレータを介した FTP 通信時のペイロード変換処理の実装

2007MI088 壁谷 司
指導教員　後藤 邦夫

1　はじめに

過去の卒業研究 [3] で IPv4-to-IPv6 トランスレータの実現がなされた．このトランスレータの改善点の一つに，FTP 通信時におけるペイロード変換処理の実装がある．これは，FTP 通信を行う際，クライアントとサーバはまずコマンド用のコネクションを確立させる．この時，FTP コマンドと一緒にクライアント，もしくはサーバ側の IP アドレスをペイロード部分に付加してパケットを送る．トランスレータはこのペイロード部分の IP アドレスの変換を行えないため，IPv4 と IPv6 間の FTP 通信が不可能である．本研究で行うことは，過去の卒業研究と同様，ネットワークエミュレータ GINE を用いて，トランスレータによるペイロード変換を実現し，IPv4 と IPv6 間の FTP 通信を可能にすることである．

2　システム概要

この節では，実験に用いた GINE(GOTO's IP Network Emulator)，そして本研究での要点である FTP について説明する．

2.1　GINE(GOTO's IP Network Emulator)

本研究では，過去のトランスレータと同じように，本研究室で開発中の GINE を利用する．GINE 上で IPv4 側のネットワークに FTP クライアント, IPv6 側に FTP サーバを仮想化させる．

2.2　FTP

FTP は，サーバへの接続時のコマンド用のコネクション (制御コネクション) とは別に，データ転送用のコネクション (データコネクション) を確立する．この確立方法にアクティブモードとパッシブモードの二つのモードがある．アクティブモードは，クライアントがサーバに待ち受ける (listen)IP アドレスとポート番号を通知し，通信を開始する．それに対し，パッシブモードはサーバがクライアントへ待ち受けるポート番号を通知する．実際の FTP 通信では主に ls，get，put といった PORT/PASV コマンドを用いる．[1] しかしこれは IPv4 のみの FTP 通信で用いるもので，IPv6 では EPRT/EPSV コマンドを使う．[2] 本研究ではこれらのコマンドを用いて通信できるかを確かめる．この時のコマンドとともに送る IP アドレスは，パケットのペイロード部分にあり，Ethernet ヘッダ，IP ヘッダ，上位層ヘッダの変換を行っていた過去のトランスレータでは，ペイロードの変換を実装していないので，このトランスレータを介した IPv4 と IPv6 間の FTP 通信はデータコネクションを確立する段階で不可能である．

3　実現

本節では，過去のトランスレータを介した FTP 通信を実現するための、具体的なペイロード変換処理について述べる．過去のトランスレータの一部である変換ゲートウェイに以下のような処理を追加する．
メソッド IPv4tov6() では，IPv4 から IPv6 への変換を行うので，クライアント側からサーバ側への通信，つまり PORT コマンドと PASV コマンドの変換を行えるようにする．

```
struct tcphdr *thdr=f->packet + iph->ihl*4;
unsigned char *tcppayload =
(unsigned char *) thdr + thdr->th_off*4;
memcpy(pay,tcppayload,iph->tot_len - 20);
```

まずはじめに, TCP ペイロード部分の文字列を探すために，IPv4 ヘッダ長 + TCP ヘッダ長のポインタで TCP ペイロードの先頭を指し示し，IP ヘッダ長から TCP ヘッダ長を引くことで，TCP ペイロードの長さがわかる．そして TCP ペイロード全体の文字列を pay という配列に格納する．ペイロード変換処理の実験をするにあたって，FTP クライアントが PORT/EPRT コマンドを入力した場合に送受信されたパケットを, FTP 制御コネクションの 21 番ポートを tcpdump を用いてキャプチャした．16 進数でダンプされているのが実際のデータ，その右側がそれらを ASCII 表示にしたものである．4500〜0104 までが IP ヘッダ，f878〜0000 までが TCP ヘッダである．よってペイロードは 504f〜0d0a までとなる．なお，ASCII コード'0x20'，'0x0d'，'0x0a' はそれぞれ「スペース」,「CR(復帰)」,「LF(改行)」を意味する．これらは PORT/EPRT, PASV/EPSV コマンドの全てのペイロードに含まれている．送信先ポートが 0x0015 つまり 21 であった場合，ASCII16 進数で書かれているペイロード文字列を ASCII 文字に直す．

```
const char *delimit_a ="'PORT', ";
if(th_dport == htons(21)){
sprintf(payload_a,"%s",payload_a1);
if(strstr(payload_a ,"PORT")){
token1 = strtok(payload_a,delimit_a);
sprintf(pay,"%s.",token1);
```

TCP ヘッダの送信先ポート番号が 21 番であるかを判別する．21 番でなかったなら，PORT コマンドから EPRT コマンドの場合，ペイロードの中から"PORT" という文字列を先頭から検索する．見つかった場合，strtok() を用いて文字列分解を行う．分解対象文字列を

「'PORT'」,「(スペース)」,「,」として分解を行うことで，実験プログラム内では，PORT 192,168,0,4,156,65 というトークンが得られる．IPv4 アドレス部分を抜き出し，コンマで区切られているので，IPv4 アドレスの正しい表記にするため，ドットで区切るように別に用意したメソッド strrep() を用いてコンマをドットに置換する．また，ポート番号は元の 10 進数の形に戻しておく．

```
sprintf(pay,"EPRT |2|2002:0:0:2::2|
%d|\r\n",p);
for(i=0;pay[i]!='\0';i++){
strcpy(payload_a2+strlen(payload_a2),pay[i]);
sprintf(payload_a2,"%s",payload_a2);
```

この IPv4 アドレスに対応する仮 IPv6 アドレスを変換表に問い合わせる．そして IP4to6() 内で IP ヘッダ・上位層ヘッダ変換を行い，ポート番号と返ってきた仮 IPv6 アドレスを使い，ペイロードのコマンド文字列を sprintf() で書式を変換する．この時，文字列の末尾に復帰・改行のコードもいれておかなければならない．最後にそれぞれの文字を ASCII16 進数に直し，元のペイロード文字列に格納する．ペイロード変換処理の後，ペイロードの変換に伴い，IP ヘッダのペイロード長の変更とチェックサムの再計算の必要が生じる．ペイロード長の変換処理は，clone->packet の packetlen（パケットの長さ）及び framelen（フレームの長さ）を変えることで行われる．この場合，ペイロード内でも IPv4 アドレスから IPv6 アドレスへ変換したので、ペイロード長はその分増えている．チェックサムの再計算には，メソッド内の checksumsdjust() を用いる．変換前のデータと変換後のデータの変換された部分のみを計算し直すものだが，ペイロード変換処理によって 2 回目のチェックサムの再計算をしなければならない．TCP チェックサムはペイロードの終端まで全て再計算し，IPv4 ヘッダチェックサムはヘッダ全部を対象に再計算を行う．

```
if(strstr(payload_a,"PASV")){
while(strrep(payload_a,"PASV","EPSV"));
```

一方ペイロード内に ”PASV ” という文字列が見つけた場合，PASV コマンドから EPSV コマンドへの変換を行う．この時の変換はペイロードを PASV という文字列を EPSV に置換するだけである．また，”PORT ”や ”PASV ” に該当する文字列が見つからなかった場合は，変換処理を行わない．
メソッド IPv6to4() は IPv6 から IPv4 への変換を行うので，サーバからクライアントへのレスポンスデータの変換処理を行えるようにする．

```
if(th_sport == htons(21)){
if(strstr(payload_ar,"EPRT")){
while(strrep(payload_ar,"EPRT","PORT"));
while(strrep(payload_ar,"EPSV","PASV"));
```

サーバ側からのパケットの TCP 送信元ポートが 21 番であった時，ペイロード内に ” EPRT ” という文字列を検索．見つかった場合，つまり EPRT コマンドから PORT コマンドへの変換をする場合，ペイロード内の EPRT という文字列から PORT という文字列へ置換すると，ペイロードには 200 PORT command successful. と書かれる．(実験では EPSV から PASV への置換も行っている) また，レスポンスコードは PORT・EPRT ともに 200 なので変換の必要はない．

```
if(strstr(payload_pr1,"Entering")){
token2 = strtok(payload_pr1,delimit_p);
sprintf(pay_p,"%s.",token2);
token2 = strtok(NULL,delimit_p);
strcpy(pay_p+strlen(pay_p),token2);
printf("%s\n",token2);
p = atoi(token2);
p1 = p / 256;
p2 = p % 256;
sprintf(payload_pr1,"227 Entering Passive
Mode (10.0.0.0,%d,%d))",p1,p2);
while(strrep(payload_pr1,".",","));
printf("%s\n",payload_pr1);
```

一方 Entering という文字列が見つかった場合, EPSV コマンドから PASV コマンドへの変換を行う．分解対象文字を「|」として, strtok() で文字列を分解し, sprintf() で書式を変換する．レスポンスコードは EPSV の時が 229，PASV が 227 なので，レスポンスコードを含めての文字列変換を行う．ポート番号を 10 進数から 256 で割った商と余りの形に直し, 変換表からサーバの仮 IPv4 アドレスを問い合わせ，ドットをコンマに置換して変換する．その後の IP アドレスの ASCII 変換及び文字列操作や各ヘッダの変換，チェックサムの再計算は PORT から EPRT コマンドへの変換のときと同様である．

4 実験，評価

実現したトランスレータを介した FTP 通信の動作確認と性能評価実験についてだが，本研究ではここから先はできなかった．

5 おわりに

本要旨では過去の卒業研究のトランスレータの，FTP 通信時の具体的なペイロード変換処理の方法について述べた．今後の課題は以下の通りである．

1. ペイロード変換処理プログラムの完成.
2. 実験及び性能評価.

参考文献

[1] J. Postel, J. R.: RFC959 - FILE TRANSFER PROTOCOL (FTP).

[2] M. Allman, S. Ostermann, C. M.: RFC2428 - FTP Extensions for IPv6 and NATs.

[3] 森　知恵，畑佐宏輝：IPv4-to-v6 トランスレータの実現とネットワークエミュレータ上での評価，卒業論文，南山大学数理情報学部情報通信学科 (2010).