三重大学教育学部研究紀要　第55巻　自然科学（2004）　1－7頁

ポートフォワーディングにおける アクセス制御について

丁　　亜　希

Access Controls for Port Forwarding

Yaxi DING

要　　旨

組織内部ネットワークの入り口に設置されるファイアーウォールでは、1ケ所で集中的にアクセス制御を行うため、ユーザの多様なニーズに対処しにくい面がある。補助手段として、ポートフォワーディング技術を用いてファイアーウォールを越えるトンネルを構築し、それらのトンネルごとに対して個別的に十分なアクセス制御を行えば、内部ネットワークのセキュリティを弱体化することがなく、ユーザのニーズにも対応しやすくなる。

個別的にアクセス制御を行うには局所システムの管理者やユーザの参加が不可欠である。そのため、より安全かつ管理し易いトンネル構築アプリケーションが必要になる。そのため、本稿は、複雑な設定が不要で、従来のユーザ登録情報も活用できるアプリケーションを考案し、有効性を検証するための実験を行う。

1. はじめに

組織内部のネットワークを外部からの不正アクセスやウィルス感染から守るために、セキュリティを強化する必要がある。その強化措置のひとつとして、多くの組織では内部ネットワークの入り口でファイヤーウォールを設置し、特定の通信ポートに対して外部からのアクセスを制限する。たとえば、外部ネットワークから電子メールサーバへの利用を不能にしたり、組織内向けのホームページへのアクセスを遮断したりしている。本学でも、ファイヤーウォールが設けられ、学内の局所システム管理者が要求しない限り、学外からの接続は原則として遮断することになっている。

しかし、このようなネットワーク環境では、組織の構成員が出張などの場合、インターネットから内部ネットワークを利用しなければならないとき、不便を感じることがよくある。また、近年自宅からインターネットを利用することができる学生が多くなっている。しかしながら、これらの学生は構内システムの一部を自宅から利用することができない状態にいる。

ユーザからの要求があれば、ファイヤーウォールでの設定の変更により外部からの接続は可能になるが、希望の通り変更できないこともあり、さらに頻繁に変更を要求することになれば、ファイヤーウォール管理者に申し訳ないと思う。

三重大学教育学部

授業の展開に応じて一定の期間の内に構内のシステムを利用して課題などを学生に完成させることがある。木曜日の17：00から金曜日の9：00まで、そして土曜日と日曜日、あるマシンのある通信ポートを開放してもらう。設定してもらった後に、土曜日午後に別の活動があるので、学生がそのマシンを使うことがないことを気付いて、この時間帯に通信ポートを遮断してもらいたい。

大きな組織では、ユーザから細かい要求が頻繁にあった場合、ファイヤーウォールの管理者がユーザの指定した通りに通信パケットの通過・遮断の設定を即時に変更するのは難しいことである。

その結果として、ファイヤーウォールを越えるためのトンネル構築アプリケーションが数多く開発され、そのなかには組織内部ネットワーク安全性に危害を加えるものも多く含まれている。トンネルを構築するアプリケーションの使用を禁止すべきではないか。

本稿は、トンネルの構築を禁止することではなく、より安全・より効率的・より使いやすいトンネル構築アプリケーションをユーザに提供し、それによって組織内部ネットワーク・セキュリティの向上および利便性の向上を図るべきであると考えている。

次の第2節では、ファイヤーウォールでのアクセス制御に加えて、トンネルを単純に禁止せず局所にアクセス制御責任を分散させることについて述べる。第3節では、使い易いトンネル構築アプリケーションを考案する。第4節でこういうアプリケーションに関する実験を報告する。

2.　アクセス制御の分散

ファイヤーウォールを設置することによって生じる諸不便の原因の1つは、アクセス制御が主に入り口1ケ所で行うことである。

ファイヤーウォール技術の向上によって、要求に応じてより細かいアクセス制御を行うことができるようになっている。

ファイヤーウォールは内部ネットワークに接続されている数多くのシステムに対して、どの通信ポートを外部からのアクセスを許可するかをそれぞれ設定し、制御を行う。外からのアクセスを許可する場合は、どこからのアクセスならば許可し、どこからのアクセスならば遮断するかというような細かい設定もできる。さらに、特定のユーザからのアクセスのみを認めるように、ファイヤーウォールでユーザ認証を行うこともできる。

しかし、入り口1ケ所だけでのファイヤーウォールの上でこのような細かいアクセス制御を実施するのは、明らか非効率である。より高性能なシステムを導入しなければならないだけでなく、頻繁に変わる利用者のニーズに合わせ、柔軟かつ迅速なファイヤーウォール設定変更に対応できるようなネットワーク管理体系を作り上げなければならない。特に、後者は、大きな組織にとって、非常に実現し難しい。

入り口だけでなく、必要に応じて内部ネットワークの一部を1つにまとめ、すべての通信ポートを開放してもらい、独自にファイヤーウォールを設置するができる。規模が小さくなったので、より細かい設定ができ、ユーザのニーズに対応し易くなる。

しかし、せいぜい1つか2つ通信ポートだけを外部からアクセスできればよいシステムでは、せっかくの入り口ファイヤーウォールの恩恵を受けず独自にファイヤーウォールを構築することがシステム資源と精力の浪費である。

VPN（Virtual Private Network）やポートフォワーディング（Port Forwarding）などの技術も発達してきた。インターネットと内部ネットワークの間にトンネルを作り上げ、それらを経由してファイヤーウォールを越えて、内部ネットワークへのアクセスを実現することができる。

VPN やポートフォワーディングのアクセス制御方式および暗号化通信技術に関しても多くの研究がなされて、いくつかの実現方法が確立され実用化されている。

ネットワーク・アーキテクチャの OSI 参照モデルの階層から見れば、データリンク層の PPTP・L2TP など、ネットワーク層の IPsec など、トランスポート層の SSL・TLS や SOCKS5 など、そしてアプリケーション層の SSH・Zebedee・stunnel などある。

以下、よく使われている SSL[1] をベースにしたアプリケーション SSH について考察してみる。

SSL はインターネット上の通信データを暗号化するためのプロトコルで、TLS[2] はそれの改良版である。それらをベースにして作られたアプリケーション SSH を利用すれば簡単なポートフォワーディングを実現できる。

例えば、内部ネットワークにあるホスト mailserver に電子メールサーバおよびホスト inner に SSH サーバが設置されていて、SSH のポートフォワーディング機能が利用可能とされた場合、インターネット上のホスト outer で

```
ssh  -L  25:mailserver:25  inner
```

のように inner にログインすれば、outer と inner の間にトンネルが作られる。outer の25番ポートへの通信はトンネルを経由して inner から mailserver の25番ポートへのように見える。ここでは、mailserver と inner は同一のホストであっても構わない。

SSH では、ユーザ認証によるアクセス制御および通信データの暗号化が行われているが、絶対に安全であると言えない所が存在する。

SSH コマンドには -g オプションがあり、これを使わない限りに他のマシンのユーザがこのトンネルを無断に利用することができない。しかし、上記の outer では複数ユーザが同時に利用可能の場合、ユーザ A が inner にログインして作ったトンネルは、利用許可を受けてない同ホストを利用しているユーザ B によって利用される恐れがある。SSH では、トンネルを作るときにはユーザ認証を行い、トンネルを利用するときにはユーザ認証を行わないのである。

それに対して、SOCKS5[3] の場合は、接続ごとにユーザ認証を行うことができる。それによって、作られたトンネルは、他のユーザによる無断利用を避けることができる。

SOCKS5 はインターネットと内部ネットワークの間に置かれる通信代理（プロキシ）ためのプロトコルであり、ファイヤーウォールとしても使える。また、SOCKS5 では通信の暗号化処理が行わないので、暗号化通信機能を持つ SSL などのアプリケーションと一緒に使うべきである。

ファイヤーウォールでの集中的なアクセス制御と比べて、トンネルごとに対するアクセス制御はエンド・ユーザのニーズにより迅速に対応しやすくなる。フォワーディング対象とするポートによるが、トンネルに関するアクセス制御責任を局所のシステム管理者またはある程度の技術力を持つユーザに分散させればよい。しかし、充分な注意を払い確実なアクセス制御を行わなければ、作られたトンネルが外部から悪用されることがあり、組織内部ネットワーク全体の安全性に危害をもたらすことも考えられる。

そのため、利用するトンネル構築アプリケーションに対して安全面からの選別が必要である。少なくとも、接続ごとでのユーザ認証および通信データの暗号化を行う必要である。

アクセス制御責任を局所管理者やユーザに分散化とすれば、システムの設置及び管理のし易さも非常に重要な課題である。

特別な機材や高性能のシステムが一般的に必要としない。

トンネル構築アプリケーションにおいて、SOCKS5 のような強力のプロキシ機能を持たなくても良い。多機能のアプリケーションでは一般的に管理設定も複雑になり、人為的なミスなどが発生しやすくなる。管理設定の難しさにより、ユーザへの迅速な対応もできなくなる。むしろ単一機能で、どの通信ポートに対してどこへフォワーディングするか明確に分からければならない。

設定・管理の簡単さのもうひとつ側面としては、ユーザ認証が行うとすれば、既存のユーザ登録情報を有効に活用すべきである。

3. 設　　計

この節では、トンネルを構築するアプリケーションの設計を行う。単一ポートのフォワーディングで、接続ごとのユーザ認証および通信暗号化機能を持ち、既存のユーザ登録情報を利用できるものにしたい。

3.1 サ　ー　バ

サーバの起動方式を以下とする：

```
server  Lport  Rhost:Rport
```

ここで、Lport はクライアントからの接続を待つためのポート番号であり、Rhost と Rport はそれぞれフォワーディング先のホストとポート番号の指定である。特に設定ファイルなどを設けない。

サーバが起動されたら、クライアントからの接続を待つ。接続要求があったら、子プロセスを生成し後述のユーザ認証を行う。認証が合格になった場合のみ、フォワーディング先への接続を行う。この接続が確立された場合、クライアントとフォワーディング先との間の中継を行う。

3.2 クライアント

クライアントの起動方式を以下とする：

```
client  [ -u name  -p password ]
        [ -f flist ]  [ -a alist ]
        Cport  Shost:Sport
```

ここで、name と password はユーザ認証を行うためのユーザ名とパスワードである。省略された場合はクライアントが起動されてから入力することになる。flist と alist は、それぞれクライアントで使える一方向ハッシュ関数および暗号化アルゴリズムのリストである（後述）。省略された場合は、一方向ハッシュ関数では md5[5]、暗号化アルゴリズムでは blowfish-cbc[6] とする。Cport はトンネルを利用するユーザプログラムからのデータを受け取るポートである。Shost と Sport は、それぞれサーバ・ホストとポート番号の指定である。

3.3 ユーザ認証と暗号化

APOP ユーザ認証方式は電子メールを受信するためのプロトコル POP3[4] に使われる認証方式である。ユーザパスワードの平文がサーバ上に保存しているが、認証の過程では、それをネットワーク上に流さない。そのため、外部ネットワークから電子メールを受信する場合、APOP 認証方式のみを許可する組織が多いようである。既存のユーザ登録情報の有効活用という観点から APOP 認証方式を採用する。

一方、通信データの暗号化においては、共通鍵暗号化方式を利用する。共通鍵暗号化方式は、一般的に公開鍵暗号化方式より処理時間が少ない。SSL などでは通信開始時に、まず公開鍵暗号化方式で通信相手へこれから使われる共通鍵を知らせ、その後共通鍵暗号化方式に切り替えて通信データを暗号化する。

APOP ユーザ認証方式を利用する場合、ユーザパスワードの平文はサーバ上に保存しているので、サーバとクライアントが共通の情報を持っている。ユーザパスワードから共通鍵を生成すれば、サーバとクライアントとの間に互いにこれから使われる共通鍵を知らせる必要もなく、SSL のように通信開始時に公開鍵暗号化方式による通信も省くことができる。

以下、ユーザ認証の過程を示す。ここでは、C はクライアント、S はサーバを示す。

C： <Flist>
S： C1 F
C： U R <Alist>
S： C2 A

1) 接続の確立ができたら、クライアントからサーバへ利用可能な一方向ハッシュ関数リスト <Flist> を渡す。サーバがそのリストから 1 つを選んで、それを F とし、ランダム生成される文字列 C1（チャレンジ）と一緒にクライアントに返す。
2) クライアントがユーザパスワード P と C1 を用いて以下のようにレスポンス R を計算し、
$$R = F(C1 + P)$$
ユーザ名 U、およびこれから使いたい共通鍵アルゴリズムリスト <Alist> と一緒にサーバへ送る。
3) サーバもユーザパスワード P と C1 を知っているので、2) と同じように F を用いて計算を行い、その結果をレスポンド R と一致するかどうかを検証する。一致すれば認証が合格になる。合格になった場合は、サーバがリスト <Alist> から 1 つのアルゴリズムを選んで、それを A とし、ランダム生成される文字列 C2 と一緒にクライアントに渡す。
4) サーバとクライアントが以下のように暗号化アルゴリズム用の共通鍵 K を生成する。
$$K = F(C2 + P)$$
以降、K を用いて暗号化通信を行う。

4. 実験と考察

前節の設計に従って、4 台の Solaris マシン S1、S2、S3 と S4 を用いて、トンネル構築の検証実験を行った。なお、この実験では本学の運用中のファイヤーウォールを使わない。

S1 <----> S2 <====> S3 <----> S4

上記のように、S2 と S3 の間にトンネルを作り、このトンネルを経由して S1 から S4 へ telnet を用いて遠隔操作を行う。つまり、

a) S3 の上で、server　8001　S4:23 を

b) S2 の上で、client　8002　S3:8001 を

c) S1 の上で、telnet　S2　8002 を

それぞれ実行する。S4 に正しくログインでき、トンネルが正常に機能することを確認できた。S2 と S3 の間で snoop コマンドを用いて通信データの暗号化も確認できた。

この実験では、第3者によるトンネルの無断利用ができない。S2 から S3 の接続ではユーザ認証を行うが、その認証用のユーザ名とパスワードが S1 からアプリケーションを実行してから入力されることになるからである。しかし、上記の b) ではクライアントを、

```
client　-u name -p password　8002　S3:8001
```

のように実行すれば、c) の実行までの間に第3者による無断利用の恐れがある。クライアントがユーザ認証後8002番ポートを開けてトンネルを利用するアプリケーションからの接続を待つのは原因である。

これは、クライアントから直接アプリケーションを起動することによって解消することができるが、コマンド行でアクセス先を指定できないアプリケーションに対しては、SOCKS のある実装のように dlopen() 関数群などを用いてダイナミック・リンク・ライブラリを操作し、動的にアクセス先をトンネルに向かうようにする必要である。これは今後の課題として残す。

最後に、c) の telnet コマンドを実行してから、login のプロンプトが表示されるまでに、S2 と S3 との間では計2340バイト／22パケットが流れていたことを確認した。それに対して、同じ実験環境で SSH では、

```
ssh　-g -c blowfish-cbc -L 8003:S4:23 S3
```

のようにトンネルを構築する場合、login のプロンプトが表示されるまでに S2 と S3 との間では計6704バイト／43パケットが流れていたことが分かった。

5. おわりに

ファイヤーウォールに守られる組織内部ネットワーク環境で、ファイヤーウォールの性能または管理体制により、ユーザの多様なニーズに対応できないことを解消するために、本稿では、安全な方法でファイヤーウォールを越えるトンネルを構築することができれば、一部のアクセス制御責任を局所管理者やユーザに分散させると主張してきた。

また、局所管理者やユーザにトンネルのアクセス制御責任を任せるとすれば、設定・管理作業の少ないトンネル構築アプリケーションでなければ、ユーザの多様なニーズに対応できないだけでなく、内部ネットワークに危害をもたらすこともありうる。そのために、本稿では、安全かつ単純なトンネル構築アプリケーションを考案した。

しかし、ファイヤーウォールを越えるトンネルの構築は、どのような条件の下で許可させ、どのような監督・報告体制が必要であるかなどの問題について、さらに組織の中でよく検討する必要がある。

参 考 文 献

1) Hickman, K: "The SSL Protocol", Netscape Communications Corp., Feb 1995.
2) Dierks, T. and etc: "The TLS Protocol Version 1.0", RFC 2246, Jan 1999.
3) Leech, M. and etc: "SOCKS Protocol V5", RFC 1928, Apr 1996.
4) Myers J. and etc: "Post Office Protocol — Version 3", RFC 1939, May 1996.
5) Rivest, R.: "The MD5 Message Digest Algorithm", RFC 1321, Apr 1992.
6) Schneier, B.: "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)" Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp 191–204.