# 隠伏サーバを実現するTCP認証方式*

㈱東芝 研究開発センター コンピュータネットワークラボラトリ†
高橋 俊成‡　梅澤健太郎‡

今日、サーバ計算機で公開しているサービスが攻撃を受けることが問題となっているが、その多くはポートスキャンとソフトウェア脆弱性を利用した無差別かつ広範囲な単純攻撃である。我々はその鬱陶しいパケット群を避ける方法としてTCPサーバの存在そのものを隠すことが近道であると考え、TCPのSYNパケットに認証情報を載せ正当なユーザのみがTCP connectionを張ることのできる認証システムTAP(TCP layer Application Protector)[1-5]を開発した。

本報告では、TCP規格を変更せずに認証機能を盛り込むためのプロトコル策定の工夫と、それをOSカーネルの修正なしに実装する方式を中心に述べる。安全性を向上させながらも、スループットや耐DoS性能に優れ、簡便で実用的なツールに仕上げられたことを紹介したい。

## 1.はじめに

インタネット上のサーバ計算機でサービスを公開する場合、ソフトウエア脆弱性を利用した攻撃の影響を特に深刻に捉える必要がある。その理由は、システム全体の安全性の拠り所としている(TLSやSSH等の)セキュリティ根幹モジュールが破られる危険に直結するからである。特にサービスの遂行がその背後にある顧客の個人情報や取引履歴等の重要なデータベースへのアクセスを伴う場合、プロセス実行を安全に行うアプローチ(SELinuxの導入やchrootプログラミング等)では全く対抗できない。また、その攻撃手法の多くは極めて単純かつ高頻度であり、システムログが肥大化する等によりシステム管理を困難にするという問題もある。

開発したシステム(TAP)では、TCPコネクションの最初のステップであるSYNパケットの送信時に秘密鍵に基く認証を行い、認証に成功した場合のみ返答(SYN/ACKパケット)を返すという方法により、特定サービスへの通信そのものを制限し、権限者以外にはそのTCPサーバの存在が秘匿できるサーバ(隠伏サーバ)を実現した。これによりアプリケーション脆弱性への単純攻撃に対する現実的な対策としている。これに対し、TCPwrapperのような、IP address等を元にサービスを隠す一般的な方法では、モバイル環境からの利用やコンピュータウイルスによる内部からの攻撃に対応できない。

TCP接続時に認証する際の懸念事項は、SYNパケットに余分な認証情報を加えた副作用により、それがTCP規格に適合しなかったり、適合しても経路上の高機能ルータに想定外パケットとして弾かれネットワーク上通過性が低下したりすることである。TAPでは、1回のTCP接続 に対し4つの異なるSYNパケットを送信し、それぞれがTCP規格に違反しない形で認証情報を少しずつ持つことで、この問題を回避した。

一方、TAPが4つのSYNパケットのセッション管理や認証処理を行うことによりTCP本来の処理性能を損なうのではないかという懸念もあるが、これについても全く問題無いことを測定により証明した。

TAPサーバはLinux(R)、TAPクライアントはLinux、Windows(R)、NetBSDで実装されており、その実装方法についても紹介する。

## 2.技術位置付け

一般に、期待しないクライアントが(アプリケーション層にある)サーバ・リソースにアクセスすることを防ぐ方法として、予めセキュアチャネルを確立するアプローチ(IPsecやSSL-VPNなど)と、TCPコネクションを受け付けない「隠伏サーバ」とするアプローチ(TCPwrapperなど)がある。伝統的な暗号技術は、盗聴／改竄可能なパケットを保護するアルゴリズム実装から始まったが、TLSやSSHでアプリケーションを構

---

* A new TCP authentication mechanism realizing hidden internet servers

† Computer & Network Systems Laboratory, Corporate Research and Development Center, TOSHIBA Corporation
‡ TAKAHASHI Toshinari <takahasi@isl.rdc.toshiba.co.jp>, UMESAWA Kentaro

築するのが常識となった今日では、むしろポートスキャンから始まる単純で執拗な脆弱性攻撃が脅威であり、後者のアプローチの重要性が相対的に高まりつつあると考えられる。

住宅に喩えるならば、破壊不能な頑丈な玄関ドアの設置よりも、どこにドアがあるのか他人には簡単に判らない家を作る方が、安全と利便のバランスが良い、そのくらい常時多くのピッキング犯がうろつくようになったのである。

従来用いられていた隠伏サーバの技術は、source IP addressで制限するなどの単純な「気休め」的保護を実現するものであったが、これでは例えば出張先ホテルの部屋やDHCP環境の接続ポイントから自宅のマシンにSSH loginを許そうとしてもSSHの脆弱性に対する危惧から実現できない。開発したTAPでは暗号技術に基くユーザ認証のメカニズムをTCP接続(SYNパケット送信)に採り入れ、特定のサービスを簡便に「隠伏サーバ」化することを目的としている。

TAPはTCP接続時に認証を行う機能に特化しており、暗号化やTCPセッション保護は行わない。アプリケーション層での認証メカニズム(TLS等)と組み合わせて利用すべきものである。また、TCPポート単位に管理するため、VPN等と異なり特定のサービスのみをリモートアクセス許可するのに向いたメカニズムである。

## 3. TAP の仕組み

### 3.1. 隠伏の基本原理

現在、多くのサービス（サーバ）はTCPにより実装されている。TCPはクライアントからサーバへのSYNパケットを送信し、サーバはSYN/ACKパケットを返答し、クライアントがACKパケットを送信する、3-way handshake後に通信を開始する(図1)。そこにはユーザ認証の機構は含まれないから、通常はSSHなどのセキュリティ・アプリケーションがユーザ認証を引き続き行う。しかし、アプリケーションのセキュリティ・ホール（脆弱性）はしばしば見付かっており、バッファオーバランなどの単純な攻撃に破れるリスクは小さくない。すなわち、攻撃者から見れば脆弱性攻撃コードを送り付けるステップに容易に辿り着ける。その結果、例えば管理者のメインテナンス用に用意したSSHサーバの口を攻撃されると被害は甚大である。

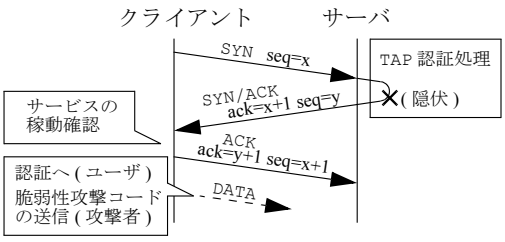


図 1. TCP 3-way handshake と TAP の隠伏点

TAPはSYNパケット受信時にクライアントを認証し、権限の無いクライアントにはSYN/ACKパケットを返信しない。これにより万一基幹アプリケーションに脆弱性があっても、攻撃にさらされる確率が極めて小さくなる。

### 3.2. 認証情報の分割送信

SYNパケットに認証情報(認証子)を載せる最も簡単な方法は、TCPのdata部(通常は空であり、任意のサイズが指定できる)に書き込むことである。しかし通常使用しないフィールドを使うと機器の実装により通過しない危険がある。実際、調査により多くの市販ルータでこのフィールドを破棄していることを確認している。

そこでTAPでは、冗長なフィールドの隙間に認証情報を埋め込む。しかし埋め込み可能な情報量が少ないため、1回のTCP connection要求につき4つのSYNパケットを送信することで不足を補う(図2)。すなわちカーネルが発行した1つのSYN-ORIGに対し、SYN0～SYN3の4つ組SYNパケットを作り、認証情報(およびTimeStamp)を分割して格納し、送信する。4つ組にはmarkを付けておく。TAPサーバは受信したパケットをmarkに基き結合し、認証情報を検証して認証に失敗すればパケットを廃棄する。認証に成功すればSYN-ORIG相当のパケット1個だけを通過させ、以降通常のTCPコネクション処理が行われる。SYN-ORIGパケットの通過後、TCP通信終了までのパケットにTAPサーバは関知しない。

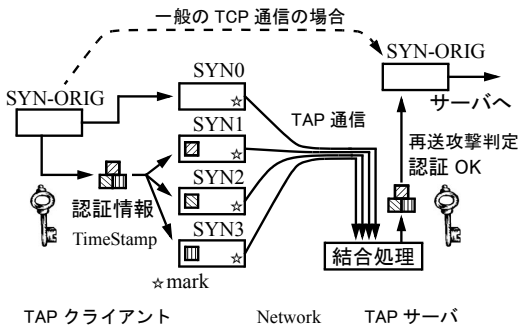


図 2. 認証情報の分割と結合

4つ組SYNパケットはTCP規格上はそれぞれ別個のTCP connection要求としてTCP規格に適合するため、送信したSYNパケットがネットワーク機器を混乱させることはない。

また、サーバが返すSYN/ACKはSYN-ORIGに対する応答のみであるが、これはSYN0と同一sequence numberを持つため、仮に経路上の機器が(不正アクセス防御機能等により)TCPセッション管理を行っていたとしてもSYN/ACKパケットおよび後続のTCPパケットがネットワーク機器を混乱させることはない。

### 3.3. TAP frame protocol

TAPにおいて、認証情報をSYNパケットのどのフィールドに乗せるかを規定したのが、TAP frame protocolである。

送信される4つ組SYNパケットの内容は、クライアントのOSカーネルが作成したオリジナルのSYNパケット(SYN-ORIG)に対し、図3に示す書き換えを行ったものである。

認証子の演算に関るデータは32bitのsequence numberフィールドに格納される。図中ハッチングを付けた部分は、SYN0～SYN3のいずれであるかを区別する目的のデータである。TimeStampとして、協定世界時刻UTC(Universal Time Coordinated)の紀元(1970/1/1 0:0:0)からの経過時間を用いる。その秒単位部分(tv_sec)の下位29bitをSYN2に格納し、秒未満部分の2進表記の上位8bitをSYN1に格納する。SYN1には他に(複数ユーザの)秘密鍵を指定するためのKeyID16bitと将来の拡張用フラグ5bit(現在全て0)が入る。SYN0にはSYN-ORIGと同じsequence numberが入る。同じ値にしているが故に、SYNパケットに後続するTCPパケットの情報がOSのTCPスタックの情報と矛盾しない。よってTAPは以後のTCPセッション管理に関知する必要が無い。SYN3には30bitの認証子を格納する。認証子の算出方法は3.5節で述べるが、典型的にはSYN0～SYN2の実データ部分(ハッチングの無い部分)とKeyIDで指定された秘密鍵をハッシュ関数に入力した結果を30bitに圧縮して用いる。

window sizeフィールドには共通の乱数(mark)を格納する。つまりTAPサーバから見て、(必ずしも連続して到着するわけではない)4つ組SYNのセッション管理をするために、(source IP address, destination port number, window size)の3つの値の組を用いる。同一のsource IP addressから同時に複数のTCPパケットが同一のTCPサービスに来ることは必ずしも偶然性が高くないので、それを補うためである。最悪のケースとしてwindow sizeが通過しない環境があったとしても(実際にはあまり無いが)残る2つの情報だけで概ねセッション管理可能で認証は失敗しない。

SYN0～SYN3は、それぞれが独立してTCP規格に則ったSYNパケットとして振舞うことを狙っているため、各source port numberフィールドには別々の値(かつOSが他で使用していない値)を使用する。その際、SYN0のsource port numberはSYN-ORIG と同じ値とする。理由は sequence numberの場合と同じである。

書き換えに伴いchecksumは再計算する。

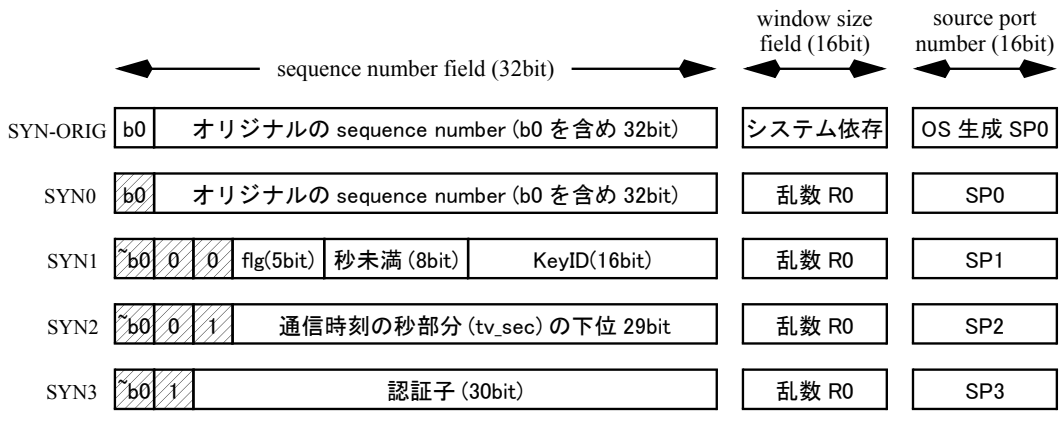


図 3. TAP frame protocol

### 3.4. TAP frame protocol 選定の根拠

認証情報を埋め込むために利用するフィールドの選定にあたっては、経路でのフィールド書き換え状況を調べる専用のサーバ／クライアントプログラムを作成し、各種プロバイダにサーバ／クライアントを配置してデータ収集を行った[3]。その際、各種ルータ機器（製品）の性質や、NAT(ネットワークアドレス変換)の実装による影響も調べた。

選定のポイントとして到達可能性(通信路で情報損失しないこと)、送信情報量(1つのSYNパケットにより多くの情報量を載せる)、TCP規格整合性、実装容易性の4つが考えられる。可能性のあるフィールドをいくつか抜粋し、その観点でまとめたのが表1である。

最初に候補から除外したフィールドは3つある。まず、source port number は実装が極めて困難である。なぜなら、OSの未使用番号が判明しない状態でその番号に情報を載せて伝える実際的な方法は思い当たらない。また、data 部は送信情報量が大きく好都合であるが、現実にはそれを廃棄するルータ機器が多く普及しており全く利用できない。最後に destination port number は、他の TCP サービスを妨害せずに使える送信情報量が小さい上、多くの port 番号をそのサービス専用に割り当てることはサーバ側 firewall の運用が不可能で、個人管理の単体サーバなどは別として、実用システムには使えない。なお、類似システムであるportknocking[6]の実装では、実装容易性を最優先し destination port number が用いられている(一般のネットワークプログラミング API で任意の値が設定できるのは destination port numberだけである)。

表 1：各フィールド利用価値比較

| TCP/SYN パケット内利用候補フィールド | 到達可能性 | 規格整合性 | 実装容易性 | 送信可能情報量(bit) |
|---|---|---|---|---|
| source port number | ○ | ◎ | × | 5〜15 |
| destination port number | △ | ◎ | ◎ | 1〜16 |
| ◎ sequence number | ○ | ◎ | △ | 〜32 |
| acknowledgement number | ○ | × | △ | 32 |
| ○ window size | ○ | △ | △ | 〜16 |
| data | △ | ◎ | △ | 大 |

Acknowledgement number は SYN パケットにおいて意味を持たない(通常0)ため完全に32bit利用できる。しかも測定の範囲では特にその情報が落とされる環境は無いので全く問題が無い。しかし、明らかに通常のパケットと異なる形状の通信を行うことは、将来性の面で不安があるためこのフィールドは使用しなかった。

以上の考察により、sequence number と window size の2つを利用することとしたが、規格整合性の低い window size は補助的役割を果たすようにした。3.3節で述べたとおり、sequence numberの情報のみでも通信が可能なようにTAPプロトコルは設計されている。なお、window size は本来その時点でクライアントが受信可能なデータ量を伝えるものであり、TAP が勝手に乱数を振ることはTCP規約上illegalである(システム本来の window size より大きく設定してしまいオファードウィンドウにシュリンクが発生する)が、実際にはACK の window size 値により再計算されたウィンドウが利用されるため、過剰なデータが送信される心配は無い。

また、別のアイデアとして、ネットワーク環境ごとに利用フィールドを動的に変更し、「そのとき使えるものを使う」実装も考えられるが、それは効果が無いことが測定から判った。静的に通信できない環境において、動的にすれば通信可能となるという状況がほとんど無い。

### 3.5. TAP-hmac protocol

SYN3にどのような認証子(30bit)を載せるかを規定したのが、TAP-hmac protocol である。

TAP は予め鍵交換を済ませたサーバ／クライアント間で認証を行うシステムであるから、TAP frame protocol に従ってさえいれば、認証子部分に何を埋めるかについては個別の協議で選択が可能である。例えばユーザ毎に認証システムが異なっていても良いし、KeyIDをアルゴリズム選択フィールドとして使用しても良い。あるいはサービス毎に(destination port number 毎に)異なる認証方法を用いても良い。

現在公開している TAP の実装では、予めサーバ／クライアントで秘密鍵を共有し、複数のユーザ鍵を区別するためにKeyID(ユーザID)を用いている。認証子の計算には、hmac-md5[7]を用い、SYN0, SYN1, SYN2のsequence number field値(斜線部は除く)を入力として鍵付きハッシュを行った結果(128bit)のうち30bitを使っている。こ

の1方式だけが実装されている。

前に述べた通り、異なるTAP-hmac protocolを採用したTAPの実装が複数存在して良い。しかしながら、混乱を避けるため、我々の公開する実装、およびそれと互換性のある方式、あるいはその将来の拡張がTAPの標準的プログラムとして使われるのが望ましいと考えている。

### 3.6. TCPの再送処理

TCP規格ではSYNパケットに対するSYN/ACK返答が無ければSYNの再送を繰り返し行う。TAP処理では4つ組パケットに対しサーバは1つのSYN/ACKしか返答しないが、クライアントOSのTCPスタックはSYN-ORIGの1つだけを送信しているので、それに対するSYN/ACK返答を受けることで再送は通常発生しない。

また、実際にネットワーク障害等でSYN/ACKが戻らなかった場合、OSの通常の手続きによりSYN-ORIGが再送され、TAPは新たなSYN-ORIGに対する4つ組SYNパケットを送信する。その際TAPはそれが再送パケットであるか否かにより処理を分けない。つまり、再送パケットに対しては全く新たにSYN0～SYN3を作成するため、組管理に混乱は起こらない。サーバ側から見れば全く同じSYN-ORIGを受信するので、通常のTCP再送を受けたのと同じ状況となり問題が無い。

TAPは1回目の4つ組SYNと再送(2回目)の4つ組SYNを区別することができ、混乱することはない。なぜならwindow sizeフィールドの値が新たに振られるからである(図4)。必ずしもwindow sizeが伝わらなくてもTAPは動作すると前に述べたが、そのような環境を考慮するには、TAPが4つ組を待つ時間を適切に設定する必要がある。通常この値はTCPの再送間隔と比べ十分小さな値とするので問題は生じない。

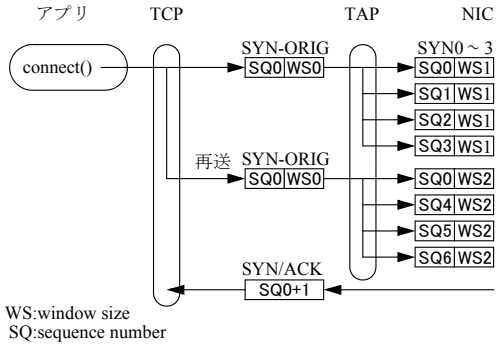


図4. TCPの再送とTAP

### 3.7. 各フィールド長の選定と安全性

TAPでは1回のTCP接続ごとに4つのSYNパケットが送信される。勿論これを5つ、6つと増やしてゆけば送信可能情報量は増え、(暗号理論上の)安全性はより高くなる。しかしその数を増やすことはサーバ負荷を高めスループットを下げるだけでなく、DoS攻撃にも弱くなり結果的にサーバの(可用性の意味での)安全性は低下する。本節では4つに絞ることができると考えた根拠として各フィールド長が実用上十分であることを示す。

まず、認証子の30bitであるが、一般に安全とされる鍵付きハッシュの長さよりかなり短い。これは総当り攻撃(brute force attack)の可能性と比較して決めるパラメータである。総当り攻撃は攻撃が有効な時間(クライアントの許容時間差)に比例して容易になる。10分の時計の狂いを許した場合、毎秒1,000回のトライとして

600×2×1000=約20bit

となるので、20bitでは少ないことが判る。一方、現実には不可能な仮定として毎秒10万回のトライをしたとしても

600×2×10万=約27bit

である。よって30bitは暗号理論的に十分な長さを確保していないものの大抵の攻撃は防げる。極端な高頻度アクセスの防御は専用のDoS防御装置等に任せるべきであり、単純攻撃の防御を目的としたTAPの役割ではないとした。

次に、通信時刻(timestamp)を記録するフィールド長であるが、秒単位部分については下位29bitのみ使用するため約17年で同じ値が戻って来る。これは再送攻撃耐性に影響するが問題無いことを次節で述べる。秒未満部分は8bitであり約4ミリ秒の精度しか無いが、これは前後のTAP接続と同一時刻かつ同一sequence numberにならなければ良いので、問題ない。

### 3.8. 再送攻撃対策

隠伏サーバの宿命として、チャレンジレスポンス等の無い一方向認証を行う必要があるが、その特性により再送攻撃(replay attack)対策が必要である。例えば、認証を通る4つ組パケットを盗聴した攻撃者がそれを保存しておき、SSHのアプリケーション脆弱性が見つかると同時にTCP接続を行う、という攻撃を防ぎたい。

TAPサーバは認証に成功したパケットのハッシュ値(例えば認証子そのもの)を記憶し、再度

(ハッシュ値が)同一のパケットが到着したらそれを再送攻撃とみなす。しかし、全ての履歴を永遠に記憶するわけにはゆかないし、再送攻撃でなくても同一のパケットは発生するから、以下の方法としている。

STEP0: TAP サーバと TAP クライアントの時刻はほぼ同期している(例えば誤差が数十秒以内)とする

STEP1: 4 つ組 SYN の結合処理終了後、到着時刻と、記述された時刻の差の大きいパケットは廃棄する。

STEP2: 時刻が許容範囲であれば認証子のチェックを行い、一致しなければ廃棄する。

STEP3: 既に同一のパケットが記録されており、かつその記録時刻が一定時間内であれば再送攻撃とみなし廃棄する。

STEP4: 再送攻撃ではない(正当なアクセスである)と判断し、それをパケットの到着時刻と共に記録する。

なお、TAPのSYNパケットに記録される時刻の秒部分は29bit であり、約17年で同じ値が戻る。よって、同一の鍵を17年使い続け、かつ17年越しの再送攻撃を受けると破れるが、いずれの仮定も通常成り立たないと考え、この値とした。

パラメータ指定に関しては、同一パケットを記録するためのテーブルサイズが小さいとハッシュ値の衝突が起こり誤って再送攻撃として判定される確率が上がる(その場合でもTCPの再送機構により通信は成功する)。また、クライアントの時刻誤差の許容値を大きくすればその傾向が強くなる。これらの条件は、現在普及している汎用 PC の性能であれば問題にならないが、TAP サーバ機能を持つ専用機器を作ろうとすると、時計管理やパラメータ設定を厳密に行う必要がある。

## 4. パケット変換の実装

TAP通信を実装するには、TCP/SYNパケット内部フィールドを書き換えたり読み出したりする必要があるが、一般的なネットワーク処理用関数に許されているのはdestination IP addressなど一部のフィールドに対しての指定と、IP またはTCPのraw packet送信のみである。

我々はOSカーネルの書き換えを行うことを避けるため、各OSに用意されたパケット変換メカニズムを利用することとした。それぞれの実装方法を紹介する。

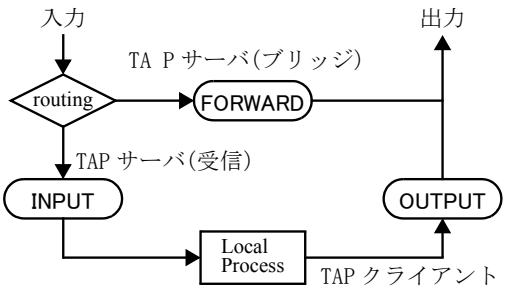


図 5. Linux netfilter の filter フックポイント

### 4.1. Linux への実装

Linux 版(サーバおよびクライアント)は、netfilterとiptablesにより実装した。

netfilter[8] はLinux2.4.xおよび2.6.xカーネルに組み込まれたパケットフィルタリングの機構であり、その抜粋したものを図5に示す。ネットワーク・スタックの5箇所(図では3箇所)にhookが配置されていて、それぞれにルールが記述できる。そのテーブルはiptablesコマンドにより設定できる。テーブルはfilter, nat, mangleの3つがあるが、このうちfilterテーブルがTAPの実装に使われており、LOCAL_IN(INPUT), FORWARD, LOCAL_OUT(OUTPUT) の3箇所で用途に応じてフックされる。

TAP クライアントでは、次の設定例のようにLOCAL_OUTでSYNパケット出力をフックする。

```
% iptables -I OUTPUT 1 -p tcp --syn
  --dport dd -m state --state NEW -j QUEUE
  -d dd.dd.dd.dd
```

これにより、指定したサーバおよび destination port への SYN パケットをフックし QUEUE ターゲットに落とす。

ここで、QUEUEターゲットとはiptablesに提供されるターゲットモジュールで、ユーザ空間か

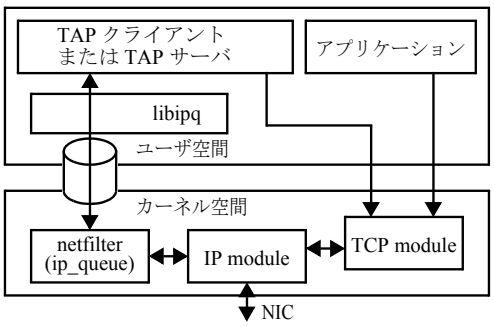


図 6. QUEUE ターゲット処理概略

らカーネル空間のパケット処理を可能とする(図6)。ユーザ側インタフェースはlibipqによって提供され、カーネル空間での処理はカーネル・モジュールのip_queueで実現される。ip_queueはルールにマッチしたパケットをユーザ空間のキューに上げ、アプリケーションはlibipqを利用してキューからパケットを取り出し、それを修正したり廃棄したりキューに戻したりする。

```
// キューの生データを読み出す
ipq_read(handle, buf, len, timeout);
// packet 構造体に変換
packet = ipq_get_packet(buf);
// 書き換え処理後、キューに戻して送信許可する
ipq_set_verdict(handle, packet->packet_id,
     NF_ACCEPT, 0, NULL);
```

TAPクライアントはこの方法で SYN-ORIG を SYN0 に変換し、送出する。SYN1～SYN3を追加して送出することはlibipqの機能ではできないので、通常の raw packet 送信を行う。

```
sock=socket(AF_INET,SOCK_RAW,IPPROTO_RAW);
setsockopt(sock,IPPROTO_IP, IP_HDRINCL,
     &const_1, sizeof const_1);
sendto(sock, buf, len, 0);
```

sendto()で送信したSYN1～SYN3のパケットは多くの場合 SYN-ORIG をフックするためのip_queueのルールにマッチするので、ipq_read()で再び自分に戻って来る。よって、自分(TAPクライアント)の送信したSYNパケットはTAPの変換処理を行わず、通過させる。

TAPサーバでは同様に LOCAL_IN で受信したSYN パケット入力を処理する。または別ホストで稼動しているサーバに転送する構成の場合には FORWARDで処理する。

### 4.2. BSD への実装

BSD 版クライアントは、bpf (Berkeley Packet Filter raw network interface)[9] により実装した。bpfはBSD系OSに実装されている仕組みで、bpf用の仮想デバイスを通じてパケットをユーザプロセスに渡したり、ユーザプロセスから生のIPパケットを投げたりすることができる。

bpf はアセンブラ風の命令列を次例のようなbpf_insn 構造体にセットすることでフィルタを定義する。任意のバイト位置にアクセスできるため、見た目ほどに設定は難しくない。また、tcpdump コマンドのオプションでこの構造体を出力することもできる。

```
struct bpf_insn bpf_insn[] = {
// 12byte 目は ether ヘッダの type 部
BPF_STMT(BPF_LD+BPF_H+BPF_ABS, 12),
BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K,
   ETHERTYPE_IP, 0, 12), // IP?
// 14byte 目は IP ヘッダの先頭
BPF_STMT(BPF_LD+BPF_B+BPF_ABS, 14),
BPF_STMT(BPF_ALU+BPF_RSH+BPF_K, 4),
// A>>=4 ( 下位 4bit) は IPv4 か ?
BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, 4, 0, 9),
  ... ..
};
struct bpf_program filter;
filter.bf_insns = bpf_insn;
filter.bf_len = sizeof(bpf_insn)/
    sizeof(bf_insn[0]);
bpf_fd = open("/dev/bpf0", O_RDONLY, 0);
ioctl(bpf_fd, BIOCSETF, filter);
```

bpfでは、仮想デバイスからの読み出し／書き込みができるが、送信するはずのパケットをブロックすることはできない。従ってBSD版クライアントでは、SYN-ORIGに加えてSYN0～SYN3の計5つのSYNパケットが送出されるため、SYN-ORIGは権限の無いアクセスとしてTAPサーバに廃棄される。実用上問題無いが余分なパケットが送信されることになる。

### 4.3. Windows への実装

Windows版クライアントは、NDIS(*)中間ドライバ(Intermediate NDIS Driver)[10]により実装した。

NDIS 中間ドライバは上位インタフェース(Miniport IF)と下位インタフェース(Protocol IF)の各ハンドラを登録しておくことで利用可能となる。上位プロトコルドライバからパケットの送信要求が発生したときに MPSendPackets() ハンドラ(MPはMiniportの意)が呼び出されるので、パケットを作成した上で NdisSend() 関数により送信を行う。

Windows の場合、他の OS とは異なり環境設定や TAP 機能の起動 / 終了等のコントロールがグラフィカルなユーザ・インタフェースであることが期待されるので、別に作成した「サービス」(常駐するバックグラウンドアプリケーション)が中間ドライバと交信して鍵情報の受け渡しなどを行う仕掛けとした。

TAP が「タスクバー」にアイコンとして表示され、マウス操作によって設定ファイルの新規作成、設定ファイルの編集、TAP 機能の起動／停止を中間ドライバに伝える機能を持ち、TAP の稼動状態をアイコンで表示する。また、中間ドライバと通信して現在までの SYN パケットの処理数、TAP処理を行った数などを表示する。

＊ NDISはMicrosoft社と3Com社が共同で定めたデータリンク層のネットワーク・ドライバ・インタフェース仕様

## 5. 性能評価

TAP を導入することによるオーバヘッドは、SYNパケット送信時(TCP connection時)のみなので通常問題にならない。しかしサーバはDoS攻撃を受ける等の特殊事情があるため、その性能を知っておく必要がある。TAPサーバにおいて一般のTCP通信よりも負荷のかかる要因を重大な順に挙げると以下のことが考えられる。

1. 1回のTCP接続につき4つのSYNパケットを受信する。その4つは連続して来るとは限らないので、コネクション管理をする必要があり、メモリとCPUを消費する。
2. 再送攻撃判定のために履歴管理テーブルを管理するコストがかかる
3. 認証処理 (hmac-md5 計算 ) の演算量がある。
4. TAPは4つのSYNパケットが1つでも届かないと処理できず、TCP再送により再び4つのSYNが送信されるため、パケットロスの大きい環境では再送の効率が悪くなる。

基本的にTAPで保護されたサーバは隠伏しているので、攻撃者に存在を知られることは無いと楽観視することも可能だが、ここでは、隠伏サーバの存在が知られてしまった際に攻撃を受けることを想定し、測定した[4,5]。

基本通信能力の測定、および3種類の典型的な攻撃を受けた際の性能測定をそれぞれ行った。

なお、いずれの測定においても、Linux版のTAPサーバ／クライアントを用い、100baseTX full duplesにてhubを介して各測定マシンを接続し、測定ツールnetperf[11]にて測定を行った。また測定マシンは全てPentium4 2.4GHz、メモリ512MBないしはそれに準ずる性能の個人向けデスクトップPCを使用した。

### 5.1. 高負荷時性能

測定条件: WWWサーバ(apache)へのTCPコネクション後に32byte送信、1024byte受信する操作を繰り返す

結果: TAPを適用することによりトランザクション数は1627回/secが1618回/secになり、スループットは94.1Mbpsで変わらなかった。いずれについてもTAPを用いない場合と全く性能差が無いと考えられる。

### 5.2. ACK 返答待ちを狙った DoS 攻撃の耐性

TAPはTCPのSYNパケットのみを処理対象とするため、SYNパケットの大量送信が最も強力なDoS攻撃であると考え、以下の測定を行った。

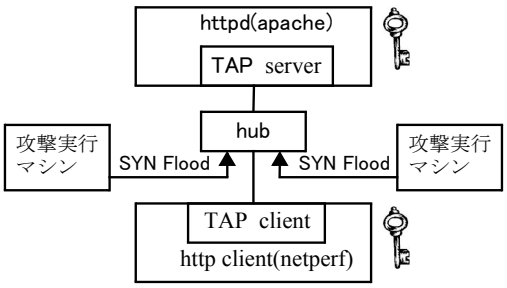


図 7. 攻撃下性能の測定方法

測定条件: SYN Floodを行うクライアントマシンを2台設置し、SYN Flood遂行中に、正当なTAPクライアントがWeb閲覧を並列に100回試み、その成功確率を計測した(図7)。その際、ユーザの感覚に近付けるため、SYNパケット送信のタイムアウトをデフォルトの180秒から45秒に変更した。

結果: 46,000packet/sec程度のSYN Flood攻撃下でも、TAPを利用したWeb閲覧がほぼ100%成功する(図8)。これはTAPを用いない一般のWebサーバが100packet/sec程度のSYN Flood攻撃で接続不能状態に陥るのに比べ極めて性能が高い。SYNパケットがTAP処理の時点で破棄されTCPセッション処理へ渡らない(ACK待ちをしない)からである。ただし、一般にはコネクションキューの制限値によって接続不能になるためTAPの理論性能そのものが460倍良いということではない。それでもTAPではlibipqなどのライブラリのオーバヘッドを含めた値であるから、メカニズムの優位性は示唆されている。

また、一般のTCP通信でACK待ち攻撃を防ぐ最も単純なメカニズムは、Linux等で採用されているSYN cookie(*)と考えられるが、それと比べても性能的に特に見劣りしない。

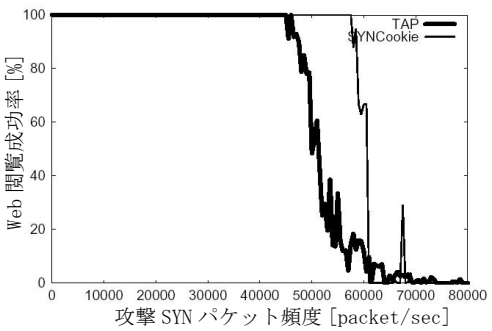


図 8. SYN flood 攻撃耐性

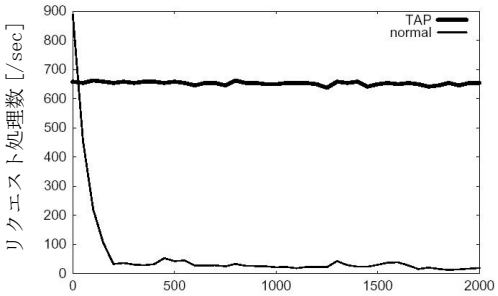


図 9. アプリケーション DoS 攻撃耐性

### 5.3. サーバ・アプリを起動するDoS攻撃の耐性

次に、TCP 接続まで行いサーバ・アプリケーションを起動させるタイプの DoS 攻撃に関して測定を行った。

測定条件: 図6と同様の構成で、http GET を連続して試みるクライアントマシンを 2 台動作させた状態で、正当なクライアントが Web 閲覧を100回試み、単位時間あたりのリクエスト処理数を、サーバが TAP を利用する場合と利用しない場合それぞれについて行った。なお、リクエスト処理数についてはベンチマークツール ab(Apache に標準で付属している Apache Bench) の出力をそのまま採用した。

結果 : TAP を利用しないサーバの場合、毎秒200コネクション程度の攻撃であっても、アクセス性能が急激に低下するのに対し、TAP で保護されたサーバの場合、攻撃の頻度にかかわらず安定した性能を提供することができる(図9)。この結果は、処理の重さについて TCP 接続およびアプリケーション起動処理が支配的であり、それらを攻撃者に許さないTAPのメリットが、TAP 自体の処理の負荷よりもはるかに大きいことを示している。SSHなど、起動自体が重い(公開鍵処理を持つ)アプリケーションであれば、TAPを適用する効果が一層大きいと考えられる。

### 5.4. TAPプロトコルを使ったDoS攻撃耐性

秘密鍵を知らない攻撃者が TAP プロトコルを用いて接続を試み、失敗することを繰り返すDoS攻撃に対する性能を測定した。

測定条件 : TAP アクセスを連続して試みる攻撃者のいる状態で、正当な TAP クライアントが Web閲覧を並列に100回試み、その成功確率を計測した。

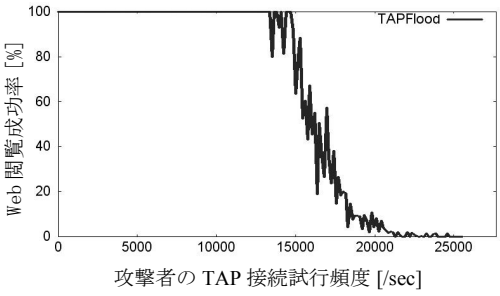


図 10. TAP flood 攻撃耐性

結果 : 14,000接続/sec程度のTAP接続攻撃下でも、正当な TAP クライアントからの TAP を利用した Web 閲覧がほぼ100% 成功する(図10)。この値は送信する SYN パケットの数に換算すると56,000packet/sec 程度であり、実験 5.2 の一般の SYN Flood攻撃の場合と比べても、TAPサーバが TAP flood攻撃に弱いという性質は全く持たないことが確認できる。

## 6. 運用上の留意点

TAP は、TCP 通信それ自体の規格整合は考慮しているが、結果的に運用上の不整合が生じる可能性をいくつか想定し、それぞれ問題が無いことを示す。

### 6.1. 初期シーケンス番号予測可能性

一般に初期シーケンス番号(SYNパケットに含まれる sequence number) が予測可能であると TCP のセッションハイジャック攻撃に弱くなるとされる。TAP の SYN パケットには予測可能性の高いもの (SYN2 など ) が含まれるが、実際の TCPデータ通信に使われるのはSYN0に含まれる(OSの生成した)番号であるため、問題無い。

### 6.2. 接続拒否による副作用

TCP wrapper など多くの接続認証メカニズムは、ホスト単位の認証を行っているが、TAP ではTCP 接続ごとに任意の鍵情報を用いた認証ができるため、クライアントアプリケーション単位の認証やユーザ単位の認証が可能である(*)。このため、実装上の都合で、TCP 接続に失敗したサーバには再度接続を試みないなどのメカニズムが加えられている場合、判断を誤る可能性がある。例えば、IPv4 と IPv6 の混在環境において不要な接続完了待ち時間を無くすメカニズムがあり、最初のユーザがTCP接続に失敗すると、

＊本来乱数であるTCP SYN/ACKのsequence numberに意味のある値(cookie)を埋め込むことにより、TCP ACKのacknowledgement numberのみでセッション確認し、ACK受信までsocketリソース確保を遅延させることができる

＊現在の実装では、BSDはホスト単位の認証のみ、Linuxはユーザ単位の認証が定義可能、Windowsは個人の鍵ファイルを(変更して)使うことができるが同時に複数ユーザが認証を切り替えて使うことはできない。

別のユーザのTCP接続は試みないなどの状況が考えられるが、今のところ具体的に問題となる症例は観測されていない。なお、TAPの現実装はIPv6に対応していない。

### 6.3. 負荷分散サーバ

高負荷のサーバでは負荷分散装置(ロードバランサ)を入れる場合がある。TAPでは4つ組SYNを同一TAPサーバで処理する必要があるため、同一のサービス(destination port 番号)を別々のサーバに負荷分散されると処理できない。よってTAPサーバは負荷分散装置の手前(クライアント側)に置くのが理想的である(図11上)。負荷分散後にTAPサーバ処理を入れると処理できない可能性がある(同一ホストからのSYNを完全にランダムに分散するという設定は一般に行われないので、これはあまり問題にならない)。

また、負荷分散装置ではなくDNS round robinで複数マシンを1つのサーバに見せ負荷を分散させる場合、TAPはひとつの宛先アドレスを4つ組SYN全てに振るので、4つ組は必ず1つのアドレス宛てに送られ、問題は生じない。ただし、そのような運用の場合、全てのサーバが秘密鍵情報を共有するという制約が生じる(図11下)。

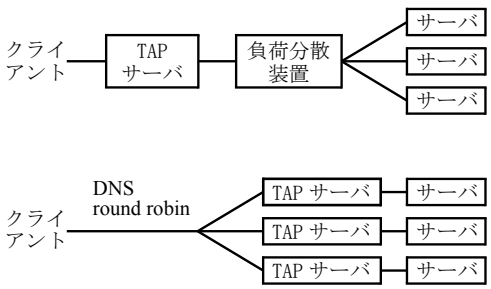


図 11. サーバ負荷分散と TAP

### 6.4. アプリケーション proxy

proxyサーバを経由したwebアクセスなどでは、sequence numberの書き換えが発生するのでTAPは接続できない。TAPプロトコルを転送する専用proxyを用いれば良いが、他人の運営するproxyを置き換えさせることはできないので、これはTAPの適用外とする。同様に、IP reachableな環境においてもまれにsequence numberの書き換えの発生が観測されている(VMware[12]に実装されているゲストOS用NATなど)

### 6.5. TAP パケットの盗聴

TAPはSSHなど別のセキュリティ・アプリケーションを組み合わせて使うことを想定しているため、TAP自体は暗号化機能を持たない。したがって、TAPの秘密鍵を持った正当なユーザがサーバにアクセスしようとするタイミングでパケットを盗聴/ブロックし、セッションを乗っ取る方法でTCP connectionを確立することができる。TAPでは、サーバアプリケーションの脆弱性を利用した任意の場所からの単純攻撃を防ぐという目的から、この点は考慮していない。また、同様に、TAPの秘密鍵を持った正当なユーザの計算機に侵入してTAP通信を奪うことについても、守備範囲外としている。

### 6.6. ルーティングに関する注意

TAPクライアントを使っている計算機を経路(ルータ)として TAP 通信を行うと、TCP connection要求を最初に発した計算機ではなく経路のroot権限でTAPサーバアクセスを行ってしまう。TAPアクセスの権限を奪われないようルーティングを適切に設定する必要がある。TAPの環境設定では、オプションでデバイス名やsource IP address(mask)が記述できる(後述)ので、抜け穴を作ってしまう心配は少ない。

### 6.7. RTT 測定に与える影響

SYNパケット送信からSYN/ACKパケット受信までのRTT(Round Trip Time)は、TAPサーバの認証処理の分だけ遅くなる。TCPが誤ったRTTを元に最適化を行うと計算を誤る可能性がある。しかし、RTTはTCPコネクションが続いている間更新されてゆくため特に問題は発生しないと考えられる。

なお参考データとして、サーバ/クライアント1台同士の単純な接続において、通常のTCPでのSYN/ACKまでのRTTは1〜2msecであるのに対し、TAPを用いた場合は2〜3msecであるという測定結果が出ている。

## 7. 使用方法

本章では、実装したプログラムの利用方法や設定について紹介する。

### 7.1. TAP サーバ(Linux 版)の使用方法

TAPサーバには、TAPservd.conf, server.key, ACL.confの3つの設定ファイルがある。

TAPservd.confにはサーバ・デーモン(TAPservd)の環境変数と、どのTCP通信をTAP化するかの条件(ルール)が記述されている。1ルールは1行で記述され、最初にマッチしたルールが適用される(どれもマッチしなければ通常のTCP通信に

なる)。各行には下記第1例のとおり入力デバイス(indev)、出力デバイス(outdev)、送信ホスト(src_host)、宛先ホスト(dst_host)、ポート番号(dst_port)、local/bridge/routerのルーティングの種別(type)が記述できる。多くの単体サーバでは第2例のようにdst_portだけ定義すれば済む。

```
RULE:indev=eth0, outdev=ANY, src_host=
  202.33.69.0:255.255.255.0, dst_host=
  211.7.133.3, dst_port=8080, type=LOCAL
RULE:dst_port=www
```

server.keyには各秘密鍵のリストを1つの鍵につき1行で定義する。典型的には、鍵番号(key_no)は一人のユーザ(人間)に対応する。

```
key_no=10,key=0xA1949A68393EA4B339FB8A597C
  38DBC9
key_no=11,key=0xC9A287541C12164770881DCBFC
  473C2C
```

ACL.confには各サービス(dst_port)に対し、アクセスを許可する鍵番号リストを定義する

```
port=80,key_no="10,11"
port=8080,key_no="10,20-23"
```

### 7.2. TAP クライアント (Linux) の使用方法

TAPクライアントは、TAPclientd.confを設定ファイルとし、デーモン(TAPclientd)の環境変数と、どのTCP通信をTAP化するかの条件(TAPサーバの場合と同様)およびその秘密鍵と鍵番号を記述する。最初にマッチしたルール行が適用される。user=で定義されているのは、そのTCPコネクションを張ったローカルユーザ名またはuidである。多くの場合、第2例のようにdst_host,dst_port,key_no,keyを定義すれば済む。

```
RULE:outdev=eth0,user=takahasi,src_host=
  192.168.0.0:255.255.255.0,dst_host=211.7
  .133.3,dst_port=8080,key_no=10,key=0xA19
  49A68393EA4B339FB8A597C38DBC9,type=LOCAL
RULE:dst_host=211.7.133.3,dst_port=80,key_
  no=11,key=0xC9A287541C12164770881DCBFC473C2C
```

### 7.3. TAP クライアント (Windows) の使用方法

Windows版は秘密鍵の管理ポリシーが他と根本的に異なる。他の(UNIX(R)系)OSではroot権限で全ての設定を管理し、TAPパケットを処理するdaemonが常駐するが、Windows版ではユーザが任意の設定ファイルを作成しPassPhraseにより暗号化し、TAP機能を有効化する際に鍵情報を中間ドライバに伝える。TAP機能を有効化/無効化したり設定ファイルの編集ツールを起動するための「サービス」が常駐する(図12)。Windows版ではUSBフラッシュメモリなどの取り外しデバイスにTAPアクセス用の鍵を入れて持ち歩く使い方も想定し、このような仕様とした。

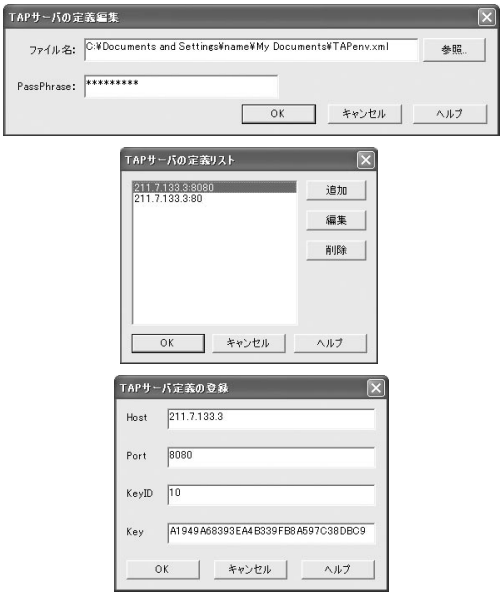

図12. Windowsでの鍵指定

## 8. 想定されるシステム適用例

### 8.1. 重要基幹サーバのメインテナンス

重要なサーバをリモート・メインテナンスする場合、単純にSSHサーバを起動したのではゼロデイアタック(セキュリティ・パッチ適用前を狙った攻撃)などによる脆弱性攻撃の懸念がある。特に緊急時に出先のネットワーク環境(DHCP環境)からであれば一層リスクが大きい。

SSHサーバにTAPを被せておけば、そのリスクがかなり低減できる。

また、サーバがDoS攻撃を受けている状況下においても、正当なTAPアクセスは比較的接続しやすいため、その意味でも非常用のメインテナンス用入り口として有益である。

### 8.2. 一般のサーバの運用容易化

顧客情報等の重要機密を持たないサーバや、一般には攻撃対象にならないと考えられるローカルなサーバの場合、TAPを用いることにより脆弱性対策の緊急度が下がるため、パッチを当てるのは深夜まで延期するなど、計画的な作業がやり易くなり、保守性が向上する。

### 8.3. イントラネット限定サービスの公開

例えば企業では従業員限定の社内サービス等をセキュリティ上の理由でイントラネット内に限定している場合が多い。しかし、在宅勤務やPCの持ち歩きが当たり前になった今日、企業外からも企業内サービスにアクセスしたい。例えばVPNなどのトンネリング技術で接続を許すと、企業内のあらゆるシステムの安全性確保が極めて困難になる。TAPを用いて、社外アクセスを許したいサーバへのTCP接続だけを許可すれば、安全性が高まり、また社員個別のTAP鍵を配布すれば、誰が接続したかTCPアクセスログも残すことができる。

### 8.4. 積極的なサービス公開

重要度がさほど高くないサーバ、例えば個人で管理している自宅のサーバを便利にするためにTAPを使うことができる。例えば、外出先からいつでも自宅にloginしたりビデオ録画／鑑賞などが自宅同様にできる。単純にSSHサーバを立ち上げるのは不安であるが、TAPと一緒であれば普通考えられる攻撃は気にしなくて良い。

また、さらに利便性を追求すれば暗号化されていないプロトコルを飛ばすこともできる。例えば、Windows XPのリモート・デスクトップのサービスを職場で立ち上げておけば、普段持ち歩くUnix系OSのノートPCでも、Microsoft RDPプロトコルを喋るrdesktop[13]コマンドを用いてWindows限定と化した職場の業務を行うのは容易である。危険と感じるかもしれないが機密の入ったPCを常時持ち歩くよりはマシである。

どこまで利便性追求が許されるかはシステム形態や利用目的に依存するが、TAPの適用でその安全上の制約は大きく取り払われるだろう。

## 9.まとめ

TCPサーバへの未知脆弱性攻撃やゼロデイアタックといった単純かつ広範囲で執拗な攻撃行為を防ぐ新しい方法として、TAPを提案しその有効性を実証した。

TAPは負荷の高い暗号理論的完備性を求めるアプローチではなく、一パス片方向認証を採用し隠伏サーバとすることで、執拗な単純攻撃を現実的に防ぎ、例えばパッチの更新遅れの脅威が軽減できるなどサーバのセキュリティ管理ストレスを軽減させる。

測定により、耐DoS性能も高いことが判った。サーバおよびクライアントのアプリケーションを全く改造することなく、daemonを立ち上げるだけで利用できる。サーバに皮を被せる方法としてTCPwrapper同様に手軽なツールとすることができた。

今後、適切な認証アルゴリズムの選定など、汎用ツールとしての完成度を上げてゆきたい。

## 参考文献


[1]梅澤健太郎，高橋俊成，鬼頭利之，サービスを秘匿するTCPコネクション確立方法の提案，電子情報通信学会 2004年暗号と情報セキュリティシンポジウム(SCIS2004)，2004/1

[2]梅澤健太郎，高橋俊成，鬼頭利之，サービスを秘匿するTCPコネクション確立方式の設計，第3回情報科学技術フォーラム(FIT2004)，2004/9

[3]高橋俊成，梅澤健太郎，TCPレイヤでの攻撃防御実験1 -- 概要と接続性，2005年電子情報通信学会総合大会，2005/3

[4]梅澤健太郎，高橋俊成，TCPレイヤでの攻撃防御実験2 -- 耐攻撃性の評価，2005年電子情報通信学会総合大会，2005/3

[5]梅澤健太郎，高橋俊成，サービスを秘匿するTCPコネクション確立方式の有効性検証，第29回コンピュータセキュリティ研究会(CSEC2005)，2005/5

[6]M. Krzywinski, Port Knocking - Network Authentication Across Closed Ports, Sys Admin Magazine Volume 12 Number 6, pp.12-17, 2003/6

[7]H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, RFC2104, 1997/2

[8]netfilter/iptables project homepage - http://www.netfilter.org/

[9]S. McCanne, V. Jacobson, The BSD Packet Filter:A New Architectur for User-level Packet Capture, USENIX winter 93, pp. 259-269, 1993/1

[10]MSDN Library - Windows DDK(Driver Development Kit) Documents - Network Devices and Protocols

[11]Netperf Homepage - http://www.netperf.org/

[12]VMware Inc. - http://www.vmware.com/

[13]rdesktop: A Remote Desktop Protocol Client for accessing Windows NT Terminal Server - http://www.rdesktop.org/




## 付録：公開予定

TAPライブラリのLinux版サーバ／クライアントとWindows版クライアントを本発表までに無償公開する予定である。シンポジウム開催から当分の間は、接続実験用のTAPサーバも公開したい。これらの公開アナウンス用URLは http://tapannounce.jplaza.com/ である