Vol.2017-GN-102 No.7
Vol.2017-SPT-23 No.7
2017/5/12

# プロセス間通信と端末間通信の把握による踏み台追跡

今田　寛[,a)]　大坐畠　智[,b)]　加藤　聰彦[,c)]

**概要**：不正アクセスから情報を守るためにはアクセス制御を適切に行う必要がある．端末を遠隔操作し通信内容を中継させる踏み台が行われると接続元が変わるため接続元によるアクセス制御が回避される．操作される端末やシステムに通信を追跡する機能がないため，本来の接続元や経由する端末・プロセスに応じたアクセス制御ができない．本稿では，プロセス間・端末間通信を把握することにより踏み台を追跡し，踏み台の末端や経由する端末・プロセスに応じてアクセス制御を行う手法を提案する．

## 1. はじめに

あらゆる情報システムがインターネットに接続されるようになり，特定の企業や組織を狙う標的型攻撃が問題となっている [1]．標的型攻撃では，端末の遠隔操作に Remote Access Trojan (RAT) と呼ばれるマルウェアが使用される [1], [2]．RAT は C&C サーバから命令を受け取り，他のマシンと通信をしてウイルスを送り込むなどの活動を行う．つまり，RAT は端末を遠隔操作し，遠隔操作により他端末へ接続する．このように，ある端末を遠隔から操作し，通信を中継させ，さらに別の端末へ接続を行うこと，またはその状態のことを「踏み台」と呼ぶ．踏み台は悪用される場合には「踏み台攻撃」とも呼ばれ，直接接続できないネットワークへのアクセスの他に，踏み台を行った場合に踏み台から新たにパケットが送信されることを利用し，発信元を隠ぺいする目的でも用いられる．現在ファイアウォールなどによって行われているアクセス制御では，制御の判定に使用している IP や TCP など，単体では追跡できる範囲が限られる情報やプロトコルを使用している．MAC アドレスや IP アドレス，ポート番号などは中継する端末やプロセスなどにより違うものへ置き換わり，その通信の発信元が不明確になるため，必要なアクセス制御をすることが難しい．

この問題を解決するために，通信の相関があることを確認する方法が研究されている [3], [4]．これらの手法では観測地点における 2 つの通信の間で踏み台が行われていることを検出する場合には有効だが，複数の踏み台を経由した場合の正確な経路の把握や発信元の検出はできない．

本稿では，端末内において中継を行おうとするプロセスの発信をその都度検知し，接続されているすべての端末のプロセス間通信を追跡することにより通信経路や発信元を特定する手法を提案する．また，それによって得られた情報を用いてアクセス制御を行う方式を提案し，端末間・プロセス間の通信追跡，追跡結果を用いたパケットフィルタを実装して実験を行い，提案方式の有効性を明らかにする．

## 2. 踏み台と関連技術

### 2.1 踏み台

踏み台とは，ある端末を遠隔から操作し，新たに発信あるいは通信を中継させ，さらに別の端末への接続を行うこと，またはその状態のことである．踏み台には，SSH(SecureSHell)[5] によるポート転送や HTran[6] などのパケット転送ツールを用いて通信内容をそのまま中継する，Telnet や SSH(SecureSHell)[5] などを用いて外部からコマンドを実行するなどの命令を送信しその命令が実行された結果別の端末への接続が開始される，DNS アンプ攻撃 [7] などのように外部からの細工された要求に対し応答した結果中継のような動作をする，RAT(Remote Access Trojan) などのウイルス等により能動的に命令を受け取り能動的に接続動作をするなどの形態が考えられる．

ほとんどの場合でセッション層以上の内容のみがやり取りされ，他の端末への接続は踏み台に使用されている端末から開始されるため，発信元となる IP アドレスや MAC アドレスが変わる．そのため，発信元の端末の所属するネットワークから直接アクセスが不可能なネットワークや端末へのアクセスの他，発信元を隠ぺいする目的でも用いられる．

[†1] 現在，電気通信大学
Presently with The University of Electro-Communications
[a)] imadahirosi@uec.ac.jp
[b)] ohzahata@uec.ac.jp
[c)] kato@is.uec.ac.jp

### 2.2 通信の追跡に関する先行研究

踏み台を介した通信の間の，相関を調べて踏み台を検知する研究がされている [3], [4], [9]．通信に相関があるかという観点から，パケットの容量の相関を確認する手法，パケットの到着間隔の相関を確認する手法 [3]，パケットに電子透かしを挿入する手法 [4] などが考案されている．これらの手法では観測地点における 2 つの通信の間で踏み台が行われていることを検出する場合には有効だが，途中の正確な経路を把握することや発信元の検出はできない．

行われている踏み台が踏み台攻撃か攻撃でない通信かを識別する方法として Round-Trip-Time(RTT) を用いて連続する中継コネクション (コネクションチェーン) の長さを測り，一定以上の長さのコネクションチェーンを攻撃とみなす手法が提案されている [9]．この手法では，コネクションの連続の長さを問題としているが，長さが短い場合でも攻撃の場合があるため，そのような場合に検出できない．

通信を行っているプロセスの特定方法について，フォレンジックの分野で山本ら [10] の `netstat` コマンドを使用した手法や，三村ら [11] の API フックによる手法などが提案されている．これらの方法により，消失する可能性の高い情報である通信を行ったプロセスに関する情報をログに記録しておくことで，不審な通信の原因特定に利用する．しかし，いずれの方法も複数のプロセスや端末にまたがった通信とそれに関するプロセスを把握することは行われていない．

既存研究では，観測地点で得られる限られた情報のみを使用して踏み台の検出や攻撃の識別を行っていた．そのため，誤検知等により機密性や可用性を低下させる可能性がある．正確なアクセス制御を行うためには，より正確で詳細な情報を収集し，それを基に判断する必要がある．既存研究でもプロセスの通信状況の把握を検討した研究はあるが，現在ソケット通信を行っているプロセスの把握にとどまり，その他のプロセス間通信や，他端末の通信状況の収集は検討されていない．

## 3. 提案方式

### 3.1 概要

本方式では，各端末のシステムソフトウェアまで管理が可能な個人や組織のネットワークでの使用を前提とする．ネットワーク管理者がアクセス許可しない端末が踏み台によりアクセス制御を回避できる問題に対して，踏み台が行なわれた場合でも，踏み台による接続の本来の発信元となる端末や中継を行っているプロセスを把握し，それぞれアクセス可否判定を行うことにより，問題のある踏み台を識別し，適切なアクセス制御を実現する．

プロセスどうしの通信を他端末内部の通信状況も含めてすべて収集することにより，踏み台に関連する接続の末端となる端末やプロセス，中継地点となる端末やプロセスを把握する．これにより，従来不可能だった踏み台により接続されている端末やプロセスと新たに接続しようとする接続先とのアクセス権の確認が行えるようになる．

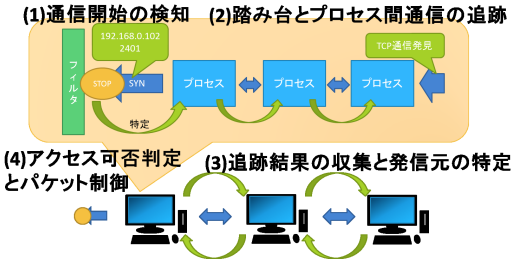


**図 1** システム概要

本方式は図 1 のように 1 通信開始の検知，2 踏み台とプロセス間通信の追跡，3 追跡結果の収集と発信元の特定，4 アクセス可否判定とパケット制御の 4 つの手順で構成される．

踏み台の接続状況は変化するため，プロセスの通信開始をその都度検知し，最新の状況から判断を行う．踏み台に関連する接続の末端となる端末やプロセスの把握は，プロセス間通信を追跡して，通信を開始しようとしているプロセスと通信を行っている他のプロセスが通信を行っているか確認することにより通信の中継を検出する．検出した通信の中継をたどり，発信元を特定する．

端末上で実行されているプロセスの状況監視と踏み台の検出を確実に行うため，端末のシステムにプロセスの通信状況を監視する機能を導入する．また，追跡して得られたプロセスの一覧と末端の IP アドレス・ポート番号を判断に用いたアクセス制御を行う．アクセス権は，アクセス権リストによって管理する．

また，本方式を Linux 環境上で動作するシステムとして実装する．ユーザランドで動作させるため，実装には Netfilter[8] の NFQUEUE 機能を用いた．`netstat` などの外部コマンドの実行や他端末との連携に必要な通信の実装を考慮し，nfqueue ライブラリを用いて Ruby で実装した．また，アクセス権リストを 1 か所で管理するため専用のサーバに分離した．他端末との連携やアクセス権リストサーバとの通信には HTTP を用いた．提案方式の詳細は次節以降で述べる．

### 3.2 通信開始の検知

通信開始の検知では，端末上で動作するプロセスがソケットを使用して TCP 接続を開始しようとする際に発信される SYN パケットをフックし，送信を保留する（図 1 (1)）．TCP の接続処理のうち，SYN フラグの立ったパケットを送信段階で介入した場合，接続の確立手順が完了しないため，TCP の接続処理を中断させることができる．

コールバック関数を NFQUEUE に登録し，`iptables` に本実装によるパケットを除いた SYN パケットを NFQUEUE

```
iptables -A OUTPUT -p tcp -m state --state NEW -j
    NFQUEUE --queue-num 0 --queue-bypass
```

**図 2** iptables コマンド

のフィルタに転送するように指定することで，SYN パケットをコールバック関数で扱うことができる．nfqueue ライブラリを用いてコールバック関数を Ruby で実装し，図 2 のように iptables コマンドを実行し，設定を行った．コールバック関数を NFQUEUE に登録する際に，iptables に指定した queue-num と同一の番号を指定して登録することにより，iptables でキューにパケットが入れられた際にコールバック関数が呼び出されるようになる．これによって，送信される SYN パケットをすべてフックし，コールバック関数で処理できるようになる．

### 3.3 踏み台とプロセス間通信の追跡

踏み台とプロセス間通信の追跡では，はじめにフックした SYN パケットから，SYN パケットを発信したプロセスを特定する（図 1 （2））．特定したプロセスを起点としてプロセス間通信を追跡する．また，他端末へ通信状況の問い合わせを行う機構を用意し，端末外と通信しているプロセスのプロセス間通信の相手プロセス・プロセス間通信の相手プロセスの端末外との通信状況を収集して，踏み台の状態を把握する．

実装ではコールバック関数内と他端末からの問い合わせに対応する部分は，実装を簡単にするため CGI として実装した．また，CGI が使用する通信も TCP であり，追跡対象とするとデッドロック状態になるため，使用するポートは特例として常時すべての通信を許可するよう設定した．プロセス間通信の追跡は，親子関係にあるプロセスがパイプを通して通信していることを利用し，プロセスツリーを探索する方法を用いた．手順を図 3 に示す．

### 3.4 追跡結果の収集と発信元の特定

追跡結果の収集と発信元の特定（図 1 （3））では，図 4 のように，踏み台とプロセス間通信の追跡による踏み台の検出結果をもとに，端末（3）から端末（2）へプロセスに接続している IP アドレスへ問い合わせを行い，問い合わせを受けた端末（2）はプロセス間通信の追跡を実行する．端末（2）も同様に繰り返すことにより，最終的に発信元の端末 (1) が発見される．その後，発信元の端末（1）は追跡結果を問合せ元の端末（2）へ返し，結果を受信した端末（2）は自端末での追跡結果を追加して前端末（2）へ返す．これを繰り返すことにより，追跡を開始した端末（3）へすべての応答が届き，端末（3）から発信元の端末（1）までのすべての踏み台とプロセス間通信の追跡結果が得られる．これにより，発信元の特定と追跡結果の収集が行われる．実装では，NFQUEUE のコールバック関数で問い合わせを行い，追跡問い合わせサービスで問い合わせを受信し動作する．それぞれの通信には HTTP，データ形式 JSON を用いた．追跡問合せサービスの手順を図 5 に示す．

1. フックした SYN パケットの送信元 IP アドレスとポート番号を調べる．
2. netstat コマンドを実行し，SYN SENT 状態になっているプロセスの中から IP アドレスとポート番号がフックした SYN パケットと一致するものを探す．
3. 2. により得られたプロセスについて，ps コマンドを用いて親プロセスをたどり，netstat コマンドの結果と照合して端末外と ESTABLISHED 状態で通信しているプロセスがないか確認する．
4. 端末外と通信しているプロセスが無かった場合は自端末を発信元として HTTP によりアクセス可否判定の問い合わせを行う．
5. 端末外と通信しているプロセスがあった場合は，端末外と通信している親プロセスの通信先 IP アドレスの追跡問い合わせサービスに対して，通信に使用されている IP アドレスとポート番号を用いて HTTP により通信追跡の問い合わせを行う．
6. 2. により得られた結果から発信元となる端末を調べ，プロセス追跡結果と発信元の端末の IP アドレス，発信元のポート番号，発信先の IP アドレス，発信先のポート番号を用いて HTTP によりアクセス可否判定の問い合わせを行う．

**図 3** コールバック関数の動作手順

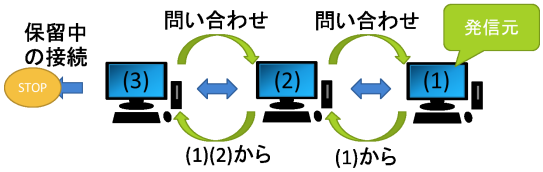


**図 4** 追跡結果の収集と発信元の特定

1. netstat コマンドを実行する
2. netstat の結果から，問合せで受け取った IP アドレスとポート番号と一致するプロセスを探す．
3. 2. により得られたプロセスについて，ps コマンドを用いて親プロセスをたどり，netstat コマンドの結果と照合して端末外と通信しているプロセスがないか確認する．
4. 端末外と通信しているプロセスが無かった場合は自端末を発信元として応答を行う．
5. 端末外と通信しているプロセスがあった場合は，端末外と通信している親プロセスの通信先 IP アドレスの追跡問い合わせサービスに対して，通信に使用されている IP アドレスとポート番号を用いて通信追跡の問い合わせを行う．
6. 2. により得られた結果に自端末のプロセス追跡結果を付加して応答を行う．

**図 5** 追跡問合せサービスの手順

### 3.5 アクセス可否判定とパケット制御

アクセス可否判定では，追跡結果の収集と発信元の特定（図 1 （3））により集められた結果と，事前に定義されたアクセス許可リストを照合し，アクセスの許可・不許可を判定する（図 1 （4））．アクセス許可や中継許可についてあらかじめ定義を作成しておき，把握したコネクション関

- 発信元 IP アドレス (接続時点での末端)
- 発信元ポート番号 (接続時点での末端)
- 発信先 IP アドレス (接続時点での末端)
- 発信先ポート番号 (接続時点での末端)
- プロセス許可リストの種類（ホワイトリスト・ブラックリスト）
- プロセス許可リスト（許可または拒否されるプロセス名の一覧）

**図 6** アクセス権リストへの設定項目

1. アクセス権リストで許可されている発信元・発信先の IP アドレス・ポート番号のペアが通信追跡結果に含まれる発信元・発信先の IP アドレス・ポート番号と一致するか確認する．
2. 1. で一致が確認できなかった場合はアクセス不可（DROP）とする．
3. 1. で一致が確認できた場合はアクセス権リストの 2. と一致する行のプロセスのブラックリストまたはホワイトリストで 1. 通信追跡結果に含まれるプロセスと一致するプロセスが無いか確認する．
4. ブラックリストの場合は一致なかった場合，ホワイトリストの場合はすべて一致した場合にアクセス可（ACCEPT）とする．

**図 7** アクセス可否判定手順

係と事前に定義されたアクセス許可，中継許可を比較することで中継が適切であるか判断する．アクセス許可は管理者によって事前定義されたホワイトリストまたはブラックリストで定義される．

また，パケット制御では，通信開始の検知 でフックした SYN パケットを送信または破棄する．アクセス可否判定とパケット制御の組合せによりパケットフィルタ機能を提供する．アクセス可否判定におけるアクセス権リストの設定項目とアクセス可否判定手順は次に示すとおりである．

### 3.5.1 アクセス権リスト

アクセス権リストへの設定項目は図 6 に示す 6 つである．発信元 IP アドレス，発信元ポート番号，発信先 IP アドレス，発信先ポート番号では，SYN パケット送信時点での踏み台とプロセス間通信の追跡結果全体の末端どうしのアクセス許可を確認する．これらの組合せにマッチしたパケットは，プロセス許可リストの種類に指定される方法により，その接続に関連するすべてのプロセスに対してプロセス許可リストとの照合が行われる．プロセス許可リストでは，SYN パケット送信時点でのプロセス間通信の追跡結果全体に含まれるプロセスの接続先へのアクセス許可を確認する．プロセス許可リストの種類がホワイトリストの場合は，すべてのプロセスがプロセス許可リストとマッチした場合に，プロセス許可リストの種類がブラックリストの場合は，すべてのプロセスがプロセス許可リストとマッチしなかった場合に接続開始が許可される．その他の場合はは接続拒否となり，設定されていない接続は開始することができない．

### 3.5.2 アクセス可否判定手順

アクセス可否判定は図 7 に示す手順で行う．アクセス可否判定には，図 8 に示すテキスト形式のアクセス権リストを使用する．アクセス権リストはカンマ区切りで図 8 のような書式で記述される．発信元 IP，発信先 IP は 8 ビット

```
Source IP,Source Port,Destination IP,Destination
    Port,Kind of Permitted Process List,Permitted
    Process List #Comment
```

**図 8** アクセス権データベースの形式

```
{"trace"=>[process trace list], "prev"=>[trace
    result of previous terminal], "hostname"=>"
    host name"}
```

**図 9** 追跡結果のハッシュの形式

ごとに “.” で区切られた 10 進数で記述される．プロセス許可リストの種類は white または black という値が入り，それぞれホワイトリスト・ブラックリストを表す．プロセス許可リストはプロセスのイメージ名が入り，“:” で区切られる．#以降から改行まではコメントとして扱われ，無視される．

NFQUEUE では，関数の戻り値として送信や破棄などの処理方法を返すことにより，フックしたパケットの処理が決定される．アクセス可否判定の結果から，NFQUEUE で使用する関数の戻り値を決定することにより，判定結果によるアクセス制御を行う．

### 3.6 追跡とアクセス制御の動作ログ出力

追跡やアクセス権の確認などの動作を確認するため，各端末で動作ログの記録を行う．動作ログにおいて，追跡結果を表す形式には，Ruby のハッシュ形式を用いる．Ruby のハッシュ形式は波カッコ “{ }” で囲まれており，“=>” の左側がラベル，右側がデータという値のペアを，カンマで区切った形式となっている．動作ログに出力する追跡結果は，図 9 のように trace， prev， hostname の 3 つのデータを格納する形式になっている．trace は，端末のプロセス間通信の追跡結果が，“[ ]” で囲まれ，値がカンマで区切られる配列形式で接続順に格納される． prev は踏み台されている前の端末の追跡結果が入れ子となる． hostname は，trace の内容が追跡された端末の端末名が文字列形式で格納される．

追跡結果の trace の配列内に格納される値は，図 10 のようなハッシュ形式となっている． type には記録内容の種類格納， ipaddr には IP アドレス， port にはポート番号， pid にはプロセス番号， name にはプロセス名， sha1sum にはプロセスの実行ファイルの sha1sum がそれぞれ文字列形式で格納される．TCP 接続の場合は，type に ip という文字列が格納され，ipaddr と port が同じハッシュ内に続いて格納される．この場合，接続元と接続先，または接続先と接続元の順に隣り合わせて trace 配列に格納することにより，TCP 接続の関係と方向を表す．プロセスの場合は，type に ps という文字列が格納され，pid と name，sha1sum が同じハッシュ内に続いて格納される．trace 配

```
TCP Connection:
{:type=>"ip", :ipaddr=>"IP address", :port=>"port
    number"}

Process:
{:type=>"ps", :pid=>"process number", :name=>"
    process name", :sha1sum=>"sha1sum of
    executable file"}
```

**図 10** プロセス追跡リストの形式

**表 1** 実験環境

| | |
|---|---|
| 計算機資源 | Basic A0 (1 コア，0.75 GB メモリ) |
| OS | Ubuntu Server 16.04 LTS |

**表 2** IP アドレス一覧

| | IP アドレス | ホスト名 |
|---|---|---|
| 端末 1 | 10.1.3.4 | jikken1 |
| 端末 2 | 10.1.3.8 | jikken2 |
| 端末 3 | 10.1.3.6 | jikken3 |
| アクセス権リスト用サーバ | 10.1.3.7 | jikkendb |

列内において，TCP 接続を表す type が ip となっているハッシュ形式のデータにプロセスを表す type が ps となっているハッシュ形式のデータが隣り合わせている場合は，プロセスがソケットを使用していることを表す．また，プロセスを表す type が ps となっているハッシュ形式のデータどうしが隣り合わせている場合は，そのプロセスどうしのプロセス間通信を表す．

## 4. 評価

### 4.1 実験の目的

実験では，以下の通り動作しているか確認を行う．

1. 端末上のプロセスの発信先や踏み台となっているプロセス間通信の接続関係を把握．
2. 複数の端末をまたいで通信の末端から末端までに経由したプロセスを把握．
3. 得られたプロセスの一覧と末端の IP アドレス・ポート番号を判断に用いてアクセス制御．

上記項目の確認のため，通信とプロセス追跡の確認，パケットフィルタ機能の確認を行った．また，本方式の通信への影響を確認するため，オーバヘッドの確認も行った．

### 4.2 実験環境

実験は Microsoft Azure 上の仮想マシンで実施した．仮想マシンは表 1 のような設定で作成した．仮想マシンは 4 台用意し，図 11 のように同一ローカルネットワーク内になるように構成した．また，うち 1 台をアクセス権リスト用サーバとした．IP アドレスは表 2 のように割り当てを行った．

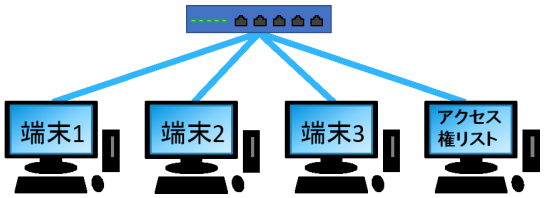


**図 11** ネットワーク構成

### 4.3 通信とプロセス追跡の確認

本方式の手順の一つである，端末間・プロセス間の通信追跡ができているかを確認するため，ssh，nc，tmux を用いて，次の手順で行う．ssh と nc はソケット通信を行うプログラムとして用いる．また，tmux は仮想ウィンドウを開くツールであり，ssh により接続を行う際にも使用される．

**（1）多段 SSH の追跡**

1. 端末 1 から端末 2 へ ssh を用いて接続する
2. ssh により表示された端末 2 のプロンプトから，さらに端末 3 へ ssh を用いて接続する

**（2）SSH とその他のプログラムの通信の追跡**

1. 端末 1 から端末 2 へ ssh を用いて接続．
2. ssh により表示された端末 2 のプロンプトから，さらに端末 3 へ nc を用いて接続．

**（3）SSH ポート転送の追跡**

1. 端末 1 から端末 2 へ ssh を用いて接続する．その際端末 3 から端末 1 へのポート転送を設定．
2. 端末 1 のプロンプトから，ポート転送が設定されているポートへ ssh を用いて接続．

**（4）tmux を用いた多段 SSH の追跡**

1. 端末 1 から端末 2 へ ssh を用いて接続．
2. ssh により表示された端末 2 のプロンプトから，tmux を起動．
3. tmux に表示された端末 2 のプロンプトから，さらに端末 3 へ ssh を用いて接続．

実験手順を実行した結果は以下の通りであった．各端末内に記録された動作ログファイルから，追跡結果が含まれる部分を抜粋し記載する．

**（1）多段 SSH の追跡**

図 12 の 8 行目に注目し，prev に入っている追跡結果をたどっていくと，jikken1 という端末の/usr/bin/ssh から，jikken2 の/usr/sbin/sshd，/bin/bash，/usr/bin/ssh を経由し，IP アドレス 10.1.3.6，ポート番号 22 へ接続しようとしていることがわかる．よって，この場合はプロセス間通信を含めた踏み台の追跡ができていることがわかる．

**（2）SSH とその他のプログラムの通信の追跡**

SSH と nc による通信時のログは図 13 の通りであっ

```
1  [2017-01-23 14:04:04 +0900]: "it is SYN packet"
2  [2017-01-23 14:04:04 +0900]: {:src_addr=>"10.1.3.8", :src_port=>45844, :
       dst_addr=>"10.1.3.6", :dst_port=>22, :iphdr_len=>20, :tcphdr_len=>40,
        :status=>{:urg=>0, :ack=>0, :psh=>0, :rst=>0, :syn=>1, :fin=>0}, :
       payload=>[]}
3  [2017-01-23 14:04:05 +0900]: "source PID is 57934"
4  [2017-01-23 14:04:06 +0900]: [[{:type=>"ip", :ipaddr=>"10.1.3.4", :port
       =>"46540"}, {:type=>"ip", :ipaddr=>"10.1.3.8", :port=>"22"}, {:type
       =>"ps", :pid=>"57798", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"57832", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"57833", :name=>"/bin/bash", :sha1sum=>"8
       af9680bba1983b73a250ee2e9f8521d75b4c4e1\n"}, {:type=>"ps", :pid
       =>"57934", :name=>"/usr/bin/ssh", :sha1sum=>"71
       df9eba2248b99e883f91aeedf8defc39b8fcb6\n"}]]
5  [2017-01-23 14:04:06 +0900]: "inquiry to: 10.1.3.4:46540"
6  [2017-01-23 14:04:09 +0900]: "response by 10.1.3.4: {\"trace\":[[{\"type
       \":\"ps\",\"pid\":\"44727\",\"name\":\"/usr/bin/ssh\",\"sha1sum
       \":\"71df9eba2248b99e883f91aeedf8defc39b8fcb6\\n\"},{\"type\":\"ip
       \",\"ipaddr\":\"10.1.3.4\",\"port\":\"46540\"},{\"type\":\"ip\",\"
       ipaddr\":\"10.1.3.8\",\"port\":\"22\"}]],\"prev\":[],\"hostname\":\"
       jikken1\"}"
7  [2017-01-23 14:04:09 +0900]: "trace end"
8  [2017-01-23 14:04:09 +0900]: {"trace"=>[[{:type=>"ip", :ipaddr=>"10.1.3.4",
        :port=>"46540"}, {:type=>"ip", :ipaddr=>"10.1.3.8", :port=>"22"}, {:
       type=>"ps", :pid=>"57798", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"57832", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"57833", :name=>"/bin/bash", :sha1sum=>"8
       af9680bba1983b73a250ee2e9f8521d75b4c4e1\n"}, {:type=>"ps", :pid
       =>"57934", :name=>"/usr/bin/ssh", :sha1sum=>"71
       df9eba2248b99e883f91aeedf8defc39b8fcb6\n"}]], "prev"=>[{"trace"=>[[{"
       type"=>"ps", "pid"=>"44727", "name"=>"/usr/bin/ssh", "sha1sum"=>"71
       df9eba2248b99e883f91aeedf8defc39b8fcb6\n"}, {"type"=>"ip", "ipaddr
       "=>"10.1.3.4", "port"=>"46540"}, {"type"=>"ip", "ipaddr"=>"10.1.3.8",
        "port"=>"22"}]], "prev"=>[], "hostname"=>"jikken1"}], "hostname"=>"
       jikken2"}
9  [2017-01-23 14:04:09 +0900]: "check_all_permission: processing res from
       jikken2"
10 [2017-01-23 14:04:09 +0900]: "check_all_permission: processing res from
       jikken1"
11 [2017-01-23 14:04:09 +0900]: {"trace"=>[[{"type"=>"ps", "pid"=>"44727", "
       name"=>"/usr/bin/ssh", "sha1sum"=>"71
       df9eba2248b99e883f91aeedf8defc39b8fcb6\n"}, {"type"=>"ip", "ipaddr
       "=>"10.1.3.4", "port"=>"46540"}, {"type"=>"ip", "ipaddr"=>"10.1.3.8",
        "port"=>"22"}]], "prev"=>[], "hostname"=>"jikken1"}
12 [2017-01-23 14:04:09 +0900]: "inq_permission(10.1.3.7, 8200, 10.1.3.4,
       46540, 10.1.3.6, 22)"
13 [2017-01-23 14:04:11 +0900]: "{\"permission\":\"ACCEPT\"}"
14 [2017-01-23 14:04:11 +0900]: "ACCEPT this SYN packet."
```

**図 12** 多段 SSH を行った場合の通信時のログ

```
1  [2017-01-23 14:12:26 +0900]: "it is SYN packet"
2  [2017-01-23 14:12:26 +0900]: {:src_addr=>"10.1.3.8", :src_port=>45972, :
       dst_addr=>"10.1.3.6", :dst_port=>22, :iphdr_len=>20, :tcphdr_len=>40,
        :status=>{:urg=>0, :ack=>0, :psh=>0, :rst=>0, :syn=>1, :fin=>0}, :
       payload=>[]}
3  [2017-01-23 14:12:26 +0900]: "source PID is 59612"
4  [2017-01-23 14:12:28 +0900]: [[{:type=>"ip", :ipaddr=>"10.1.3.4", :port
       =>"46670"}, {:type=>"ip", :ipaddr=>"10.1.3.8", :port=>"22"}, {:type
       =>"ps", :pid=>"59306", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"59366", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"59367", :name=>"/bin/bash", :sha1sum=>"8
       af9680bba1983b73a250ee2e9f8521d75b4c4e1\n"}, {:type=>"ps", :pid
       =>"59612", :name=>"/bin/nc.openbsd", :sha1sum=>"
       fdd56f8c6f606697bf310f3a626d47781a1bd5d8\n"}]]
5  [2017-01-23 14:12:28 +0900]: "inquiry to: 10.1.3.4:46670"
6  [2017-01-23 14:12:30 +0900]: "response by 10.1.3.4: {\"trace\":[[{\"type
       \":\"ps\",\"pid\":\"46440\",\"name\":\"/usr/bin/ssh\",\"sha1sum
       \":\"71df9eba2248b99e883f91aeedf8defc39b8fcb6\\n\"},{\"type\":\"ip
       \",\"ipaddr\":\"10.1.3.4\",\"port\":\"46670\"},{\"type\":\"ip\",\"
       ipaddr\":\"10.1.3.8\",\"port\":\"22\"}]],\"prev\":[],\"hostname\":\"
       jikken1\"}"
7  [2017-01-23 14:12:30 +0900]: "trace end"
8  [2017-01-23 14:12:30 +0900]: {"trace"=>[[{:type=>"ip", :ipaddr=>"10.1.3.4",
        :port=>"46670"}, {:type=>"ip", :ipaddr=>"10.1.3.8", :port=>"22"}, {:
       type=>"ps", :pid=>"59306", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"59366", :name=>"/usr/sbin/sshd", :sha1sum=>"
       e38b2d539f123263767d216798c727de3a92e08d\n"}, {:type=>"ps", :pid
       =>"59367", :name=>"/bin/bash", :sha1sum=>"8
       af9680bba1983b73a250ee2e9f8521d75b4c4e1\n"}, {:type=>"ps", :pid
       =>"59612", :name=>"/bin/nc.openbsd", :sha1sum=>"
       fdd56f8c6f606697bf310f3a626d47781a1bd5d8\n"}]], "prev"=>[{"trace
       "=>[[{"type"=>"ps", "pid"=>"46440", "name"=>"/usr/bin/ssh", "sha1sum
       "=>"71df9eba2248b99e883f91aeedf8defc39b8fcb6\n"}, {"type"=>"ip", "
       ipaddr"=>"10.1.3.4", "port"=>"46670"}, {"type"=>"ip", "ipaddr
       "=>"10.1.3.8", "port"=>"22"}]], "prev"=>[], "hostname"=>"jikken1"}],
       "hostname"=>"jikken2"}
9  [2017-01-23 14:12:30 +0900]: "check_all_permission: processing res from
       jikken2"
10 [2017-01-23 14:12:30 +0900]: "check_all_permission: processing res from
       jikken1"
11 [2017-01-23 14:12:30 +0900]: {"trace"=>[[{"type"=>"ps", "pid"=>"46440", "
       name"=>"/usr/bin/ssh", "sha1sum"=>"71
       df9eba2248b99e883f91aeedf8defc39b8fcb6\n"}, {"type"=>"ip", "ipaddr
       "=>"10.1.3.4", "port"=>"46670"}, {"type"=>"ip", "ipaddr"=>"10.1.3.8",
        "port"=>"22"}]], "prev"=>[], "hostname"=>"jikken1"}
12 [2017-01-23 14:12:30 +0900]: "inq_permission(10.1.3.7, 8200, 10.1.3.4,
       46670, 10.1.3.6, 22)"
13 [2017-01-23 14:12:31 +0900]: "{\"permission\":\"ACCEPT\"}"
14 [2017-01-23 14:12:31 +0900]: "ACCEPT this SYN packet."
```

**図 13** SSH と nc による通信時のログ

```
1  [2017-01-23 14:22:08 +0900]: "it is SYN packet"
2  [2017-01-23 14:22:08 +0900]: {:src_addr=>"10.1.3.8", :src_port=>46134, :
       dst_addr=>"10.1.3.6", :dst_port=>22, :iphdr_len=>20, :tcphdr_len=>40,
        :status=>{:urg=>0, :ack=>0, :psh=>0, :rst=>0, :syn=>1, :fin=>0}, :
       payload=>[]}
3  [2017-01-23 14:22:08 +0900]: "source PID is 63084"
4  [2017-01-23 14:22:10 +0900]: []
5  [2017-01-23 14:22:10 +0900]: "trace end"
6  [2017-01-23 14:22:10 +0900]: {"trace"=>[], "prev"=>[], "hostname"=>"jikken2
       "}
7  [2017-01-23 14:22:10 +0900]: "check_all_permission: processing res from
       jikken2"
8  [2017-01-23 14:22:10 +0900]: {"trace"=>[], "prev"=>[], "hostname"=>"jikken2
       "}
9  [2017-01-23 14:22:10 +0900]: "trace is blank. is it requested by me?"
10 [2017-01-23 14:22:10 +0900]: "inq_permission(10.1.3.7, 8200, 10.1.3.8,
       46134, 10.1.3.6, 22)"
11 [2017-01-23 14:22:12 +0900]: "{\"permission\":\"ACCEPT\"}"
12 [2017-01-23 14:22:12 +0900]: "ACCEPT this SYN packet."
```

**図 14** SSH ポート転送を使用した場合の通信時のログ

```
1  [2017-01-23 15:05:48 +0900]: "it is SYN packet"
2  [2017-01-23 15:05:48 +0900]: {:src_addr=>"10.1.3.8", :src_port=>47078, :
       dst_addr=>"10.1.3.6", :dst_port=>22, :iphdr_len=>20, :tcphdr_len=>40,
        :status=>{:urg=>0, :ack=>0, :psh=>0, :rst=>0, :syn=>1, :fin=>0}, :
       payload=>[]}
3  [2017-01-23 15:05:48 +0900]: "source PID is 55706"
4  [2017-01-23 15:05:49 +0900]: []
5  [2017-01-23 15:05:49 +0900]: "trace end"
6  [2017-01-23 15:05:49 +0900]: {"trace"=>[], "prev"=>[], "hostname"=>"jikken2
       "}
7  [2017-01-23 15:05:49 +0900]: "check_all_permission: processing res from
       jikken2"
8  [2017-01-23 15:05:49 +0900]: {"trace"=>[], "prev"=>[], "hostname"=>"jikken2
       "}
9  [2017-01-23 15:05:49 +0900]: "trace is blank. is it requested by me?"
10 [2017-01-23 15:05:49 +0900]: "inq_permission(10.1.3.7, 8200, 10.1.3.8,
       47078, 10.1.3.6, 22)"
11 [2017-01-23 15:05:50 +0900]: "{\"permission\":\"ACCEPT\"}"
12 [2017-01-23 15:05:50 +0900]: "ACCEPT this SYN packet."
```

**図 15** SSH により接続した端末上で tmux を使用した場合の通信時のログ

た．図 13 の 8 行目に注目すると，各通信でのポート番号の他，(1) において/usr/bin/ssh であった部分が/bin/nc.openbsd に置き換わっている．つまり，jikken2 において/usr/bin/ssh の代わりに/bin/nc.openbsd が使用されたことが追跡できていることがわかる．

よってこの場合も，プロセス間通信を含めた踏み台の追跡ができていることがわかった．この結果から，プロセス間通信を追跡するためにプロセスツリーの探索を行う方法でも，プロセス間通信の追跡ができることがわかる．ただし，後述の実験により，この方法では不十分であることがわかった．

### （3）SSH ポート転送の追跡

SSH ポート転送を使用した場合の通信時のログは図 14 の通りであった．図 14 の 3 行目に注目すると，自端末上で通信を発生させたプロセスは PID（プロセス番号）が取得できているが，6 行目においてプロセス間通信の追跡結果を表す trace 配列が空であることから，追跡ができておらず他端末との関係も把握できていないことがわかる．そのため，9 行目において自端末からの発信として扱われてしまい，適切な判断ができていないことがわかる．

### （4）tmux を用いた多段 SSH の追跡

SSH により接続した端末上で tmux を使用した場合の通信時のログは図 15 の通りであった．図 15 の 3 行目・6 行目に注目すると，SSH ポート転送を使用した場合と同様に自端末上で通信を発生させたプロセスは PID（プロセス番号）が取得できているが，プロセス通信のトレースがで

きておらず，他端末との関係も把握できていない．そのため，9 行目において自端末からの発信として扱われてしまい，適切な判断ができていないことがわかる．

これより，(3) SSH ポート転送の追跡，(4) tmux を用いた多段 SSH の追跡 については，追跡に失敗していることがわかる．この原因について調査したところ，同じプロンプトから起動したプログラムでも，起動したプロンプトの子プロセスにならない場合があり，この場合についてはプロセスツリーを探索する方法が使えないことがわかった．

### 4.4 パケットフィルタ機能の確認

本研究の目的の一つである，追跡結果を用いたパケットフィルタの機能について，アクセス権リストに指定したとおりにパケットの送信・破棄が行われていることを確認する．全ての端末で本研究によるシステムの実装と 8000/tcp で動作する Web サーバを起動しておき，基本的な動作が設計通りであるかを確認するため，図 16 に示す操作と図 17 に示すフィルタの設定の組合せすべてについて想定する動作と一致するか確認を行う．ここで，`nc`，`curl`，`ssh` は TCP 接続を行う 3 つの異なるプロセスとして扱う．また，`sshd` は ssh 接続を待機するサーバ，`bash` は `sshd` により起動され，さらに ssh や他のプログラムを起動するために使用されるため，プロセス間通信を行っている．そのため，フィルタに追加される．

1. 端末 1 から端末 2 の 8000/tcp へ `curl` を用いて接続.
2. 端末 1 から端末 2 の 22/tcp へ `ssh` を用いて接続.
3. 端末 1 から端末 2 の 22/tcp へ `nc` を用いて接続.
4. 端末 1 から端末 3 の 8000/tcp へ `curl` を用いて接続.
5. 端末 1 から端末 3 の 22/tcp へ `ssh` を用いて接続.
6. 端末 1 から端末 3 の 22/tcp へ `nc` を用いて接続.
7. 端末 2 から端末 3 の 8000/tcp へ `curl` を用いて接続.
8. 端末 2 から端末 3 の 22/tcp へ `ssh` を用いて接続.
9. 端末 2 から端末 3 の 22/tcp へ `nc` を用いて接続.
10. 端末 1 から端末 2 の 22/tcp へ `ssh` を用いて接続し，ssh に表示されている端末 2 のプロンプトから端末 3 の 8000/tcp へ `curl` を用いて接続.
11. 端末 1 から端末 2 の 22/tcp へ `ssh` を用いて接続し，ssh に表示されている端末 2 のプロンプトから端末 3 の 22/tcp へ `ssh` を用いて接続.
12. 端末 1 から端末 2 の 22/tcp へ `ssh` を用いて接続し，ssh に表示されている端末 2 のプロンプトから端末 3 の 22/tcp へ `nc` を用いて接続.

**図 16** 操作

結果は想定する動作と一致しており，判定やパケット制御の基本的な動作については正しく動作していることが確認できた．

パケットフィルタ機能の確認では，判定やパケット制御の基本的な動作については正しく動作することが確認できた．しかし，4.3 節の通信とプロセスの追跡の確認において，正しく追跡が行えない場合があったため，追跡の失敗や正確さについての判定を行う方法について検討する必要があると考えられる．追跡に失敗した場合，得られる情報は従来と同じフィルタが動作する地点での情報に限られるため，踏み台が行われたかどうかの確認を行うことができず，パケットの発信元が特定できない．確認できないままアクセスを許可した場合は機密性が失われ，アクセスを拒否した場合は可用性が失われることになるため，適切なアクセス制御ができない．

1. すべての端末を記述しない
2. すべての端末に対して，すべての通信を許可.
3. 端末 1 から端末 2 の 22/tcp への通信のみ許可し，他の設定は記述しない.
4. 端末 1 から端末 3 の 22/tcp への通信のみ許可し，他の設定は記述しない.
5. 端末 1 から端末 2，端末 2 から端末 3 の 22/tcp への通信のみ許可し，他の設定は記述しない.
6. 端末 1 から端末 2，端末 1 から端末 3，端末 2 から端末 3 の 22/tcp への通信のみ許可し，他の設定は記述しない.
7. 端末 1 から端末 2 の 22/tcp へ の `ssh`,`sshd`,`bash` を経由した通信のみ許可し，他の設定は記述しない.
8. 端末 1 から端末 2，端末 1 から端末 3，端末 2 から端末 3 の 22/tcp への `ssh`,`sshd`,`bash` を経由した通信のみ許可し，他の設定は記述しない.
9. 端末 1 から端末 2，端末 1 から端末 3，端末 2 から端末 3 の 22/tcp への通信のうち，`nc` を経由した通信のみ拒否.

**図 17** フィルタの設定

**表 3** パケットフィルタの各設定による接続可否結果

凡例 o 接続できた　o 成功
x 接続できない　o 失敗

| フィルタの設定 \ 操作 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | x | x | x | x | x | x | x | x | x | x | x | x |
| 2 | o | o | o | o | o | o | o | o | o | o | o | o |
| 3 | x | o | o | x | x | x | x | x | x | x | x | x |
| 4 | x | x | x | x | o | o | x | x | x | x | x | x |
| 5 | x | o | o | x | x | x | x | o | o | x | x | x |
| 6 | x | o | o | x | o | o | x | o | o | x | o | o |
| 7 | x | o | x | x | x | x | x | x | x | x | x | x |
| 8 | x | o | x | x | o | x | x | o | x | x | o | x |
| 9 | o | o | x | o | o | x | o | o | x | o | o | x |

```
time ssh -tt jikken2 ssh -tt jikken1 ls
```

**図 18** コマンド例

### 4.5 オーバヘッドの確認

実装した機能について，どれくらいのオーバヘッドがあるのかを評価する．`time` コマンドを図 18 のように用いて，`ssh` 接続，`ls` の実行，切断までの時間を計測する．

評価項目:

1. 端末 1 から端末 2 への接続.

表 4　オーバヘッド計測結果

| | システム有効 | システム無効 |
|---|---|---|
| 評価項目 1 | 3.034s | 0.471s |
| 評価項目 2 | 9.151s | 0.904s |

2. 端末 1 から端末 2 を経由し端末 3 への接続.

オーバヘッドを `time` コマンドで計測した結果を表 4 に示す．提案方式を使用した場合，システムを使用しなかった場合と比較して 10 倍程度の時間がかかっていることがわかった．これは，処理の遅いインタプリタ言語で実装したこと，実装の中で複数のループが入れ子になっていること，システムの状況を取得するために `netstat` などの外部コマンドを実行していること，大量のテキストのマッチングが行われていることなど，実装の方法に問題があったと考えられる．これについては，処理の早いコンパイラ言語を用いる，ループ処理を減らす，外部コマンドを実行する代わりにシステムの API を用いる，テキストによる処理を行わないなどの対策が考えられる．

## 5. 議論

本節では，通信を詳細に複数の端末にまたがって追跡する場合に考慮すべき問題を議論する．

### 5.1 追跡範囲の制限

提案手法では，通信をプロセス間通信のレベルまで追跡するため，プロセスツリーを探索する方法を用いた．しかし，プロセスツリーを探索する方法では，プロセスツリーと対応しないプロセス間通信を追跡できないという問題があった．これは，オペレーティングシステムの提供する API を使用してプロセス間通信の情報を取得したり，プロセス間通信に使用される API をフックするなど，より詳細で正確な情報を取得する方法を検討する必要がある．また，プロセス間通信のレベルまでの追跡では追跡を回避される可能性についても考慮する必要がある．ただし同時に，どこまで追跡すれば十分なのか，追跡のコストに対する効果についても検討する必要がある．

### 5.2 プライバシー

提案手法では，すべての通信について通信をプロセス間通信のレベルまで追跡する．この場合，使用者のプライバシーの問題が考えられる．プロセス通信よりさらに詳細な追跡を行うことも考えられるため，どこまでの追跡を行うべきかを検討する必要がある．

## 6. まとめと今後の課題

本稿では，通信の中継により発信元が不明確になり，適切なアクセス制御ができない問題について，端末間・プロセス間の通信を複数端末にまたがって追跡し，その結果を使用してアクセス制御を行う手法を提案した．実験では，プロセスツリーの探索では一部のプロセスについては追跡することができたが，正確なプロセス間通信を把握することができず，問題があることがわかった．また，追跡結果を用いたパケットフィルタリングは可能だが，追跡の方法や実装の方法により未使用時の 10 倍程度のオーバヘッドが発生することがわかった．

これらの問題の解決のため，オペレーティングシステムの API を用いるなど方式・実装の改善方法の調査や実際の RAT などのマルウェアを使用した実験，追跡機能の設置方法や追跡情報の信頼性，追跡機能を設置できない端末への対応も検討が必要である．

## 参考文献

[1] 独立行政法人情報処理推進機構: 情報セキュリティ白書 *2016*, ISBN978-4-905318-41-5, 2016
[2] Ping Chen, Lieven Desmet, Christophe Huygens: *A study on advanced persistent threats*, CMS 2014, LNCS 8735, pp. 6372, 2014
[3] Ting He, Lang Tong: *Detecting encrypted stepping-stone connections*, IEEE Transactions on Signal Processing, Vol. 55, No. 5, pp. 1612-1623, 2007
[4] Xinyuan Wang, Douglas S. Reeves: *Robust correlation of encrypted attack traffic through stepping stones by flow watermarking*, IEEE Transactions on Dependable and Secure Computing, Vol. 8, No. 3, pp. 434-449, 2011
[5] Tatu Ylonen, Chris Lonvick: *The Secure Shell (SSH) Connection Protocol*, RFC4254, 2006
[6] GitHub - zcnhonker/HTran: HTran is a connection bouncer, a kind of proxy server. A “listener” program is hacked stealthily onto an unsuspecting host anywhere on the Internet. When it receives signals from the actual target system, it redirects it to the hacker ’s server, 入手先 ⟨https://github.com/zcnhonker/HTran⟩ (2017.03.26)
[7] R. Vaughn, G. Evron, ”Dns amplification attacks (preliminary release).” March, 2006.
[8] netfilter/iptables project homepage - About the netfilter/iptables project, 入手先 ⟨https://www.netfilter.org/about.html⟩ (2017.3.14)
[9] Shou-Hsuan Stephen Huang, Hongyang Zhang, Michael Phay: *Detecting Stepping-Stone Intruders by Identifying Crossover Packets in SSH Connections*, 2016 IEEE 30th International Conference on Advanced Information Networking and Applications, IEEE, 2016
[10] 山本匠, 河内清人, 桜井鐘治: 不審プロセス特定手法の実装及び評価, 研究報告マルチメディア通信と分散処理 (DPS), Vol. 2014, No. 33, pp. 1-8, 2014
[11] 三村聡志, 佐々木良一: プロセス情報と関連づけた通信情報保全手法の提案, 情報処理学会論文誌, Vol. 57, No. 9, pp. 1944-1953, 2016