Vol.2012-IOT-16 No.53
2012/3/16

Access Control List の等価性判定のためのテストケース生成

吉 浦 紀 晃[†1]　佐 山 弘 和[†2]

Access Control List は、Layer3 スイッチや Firewall などのネットワーク機器において、どのようなパケットを通過させ遮断させるかを記述したものである．Access Control List はネットワークトポロジーやセキュリティポリシーの変更、セキュリティインシデントの発生などで変更されることがある．ネットワーク管理者はその度に変更することになるが、この変更が積み重なると Access Control List には冗長な記述が増えてくることがある．ネットワーク管理者が Access Control List を書き換える場合、その前後で Access Control List の意味が変わっていないことを確認する必要がある．本論文では、2 つの Access Control List の等価性を判定するためのテストケースの作成方法を提案する．このテストケースはパケットであり、2 つの Access Control List をそれぞれ設定したときのこのパケットを透過性をチェックすることで、2 つの Access Control List の等価性を判定する．

Test Generation for Equivalence of Access Control List

Noriaki Yoshiura[†1] and Hirokazu Sayama[†2]

Computer network security is one of the important issues in the Internet age. Network Administrators of companies or universities filters IP packets at network equipment between their organizations and the Internet to keep the security of the computer networks. Access control lists are lists of rules, which describe permission or denial of packet transition by source IP address, destination IP address, port numbers and so on. Access control lists are not always fixed; network administrators change access control list according to the change of network topology or network security policy. After several changes, access control lists may include redundancies and network administrators have to modify the access control list to remove redundancies. This modification keeps the semantics of access control list. After modification, the network administrators have to confirm that the semantics of access control list does not change. For this purpose, this paper proposes the method of generating test packets to confirm the equivalence of two access control lists.

1. はじめに

近年，ネットワークは多様に変化・発展し，利便性が増す一方で，不正アクセスや情報漏洩の事件がメディアを賑わす事も少なくなくなった．　それに伴い，セキュリティの分野は，日々進歩するネットワークの分野の中で，重要な位置を占めている．

Access Control List(ACL) は、Layer3 スイッチや Firewall などのネットワーク機器において、どのようなパケットを通過させ遮断させるかを記述したものである．ネットワーク上での攻撃元となる IP アドレスからのパケットを遮断するためや、特定の IP アドレスへの通信を遮断することで、その IP アドレスを有する機器のセキュリティを確保するために ACL は利用される．ACL はネットワークトポロジーやセキュリティポリシーの変更、セキュリティインシデントの発生などで変更されることがある．ACL の記述の変更を行う場合、ネットワーク管理者は変更に伴い誤って変更すべきではない ACL の記述を変更してはいけないので、ミスが起きないように記述の変更を行う。このような変更の仕方をすると、ACL には冗長な記述が増えてくることがある．変更を行う度に、ACL の記述は冗長になり、設定可能な上限を越えてしまう場合もありうる。特に、ACL がハードウェア処理される場合には、設定可能な ACL の上限が決まっている。また、冗長な ACL は可読性が低くなり、管理することも難しくなる[2),3)]．故に、冗長になった ACL を整理し修正する必要が生じる．また、記述の多い ACL は、ACL がソフトウェア処理される場合には処理の低下を、ハードウェア処理される場合には無駄な電力を消費するという問題がある．

ネットワーク管理者が ACL を整理する場合、変更前後で ACL の意味が変化しないように注意する必要がある．ACL の意味とはパケットの通過の許可と不許可のことであり、変更前後で意味が変わると、許可すべきではないパケットが通過するなどセキュリティ面で重大な問題が生じる。そこで、ネットワーク管理者が ACL を整理した場合、変更前後で ACL の意味が変化していないことを確認する必要がある。

ACL の意味が変化していないこと、つまり、2 つの ACL を確認するためにはいくつかの方法が考えられる。例えば、ACL の記述をモデル化してそのモデルの等価性を調べるなどが考えられる．しかし、例えば ACL の最適化は NP-complete であることなどから[1)]，モ

†1 埼玉大学
Saitama University
†2 NTT データシステム技術
NTT Data System Technology

Vol.2012-IOT-16 No.53
2012/3/16

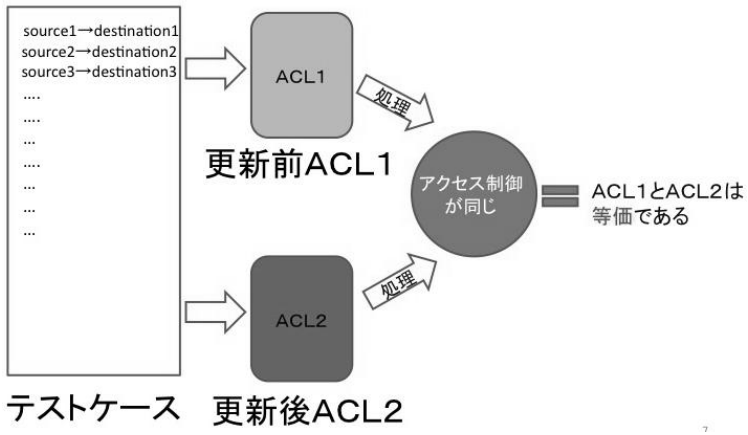

図 1 等価性の判定

デル化などによる方法は計算時間を考えると現実的ではない場合が多い．

そこで、本論文では、2 つの ACL の等価性を確認する方法として、等価性を確認するためのテスト用のパケットを生成し、実際に ACL が設定されている機器にそのパケットを送信し、そのパケットの通過や不通過を確認することで 2 つの ACL の等価性を判定する方法を提案する．本論文では，2 つの ACL からこれらの ACL の等価性を判定するためのテストケースを生成する方法を提案し，小さなテストケースで等価性が判定可能になるようにテストケースを生成する方法を提案する。

2. テストケースによる等価性判定

図 1 は本論文で提案する ACL の等価性判定の方法を示している．テストケースはテストパケットの集合からなる．テストパケットは，送信元 IP アドレスと送信先 IP アドレスの組である．本研究では，アドレス資源を 32 ビットで管理している IPv4 の環境下を想定しているので，送信元 IP アドレス，送信先 IP アドレスは各々最大 2^{32} 通りになる．つまり，互いを組み合わせた 2^{64} 通りのテストデータを持つテストケースが最大のテストケースとなる．

2.1 OmniSwitch における ACL

本研究では，埼玉大学で実際に用いられていた ACL を参考とする．その文法はアルカテル・ルーセント社の Layer3 スイッチである OmniSwitch に従う．次に OmniSwitch における ACL の例を表 1 に示す[4),5)]．

表 1 OmniSwitch における ACL の例

policy network group EK 133.38.212.0 mask 255.255.252.0
133.38.211.0 mask 255.255.255.192
policy network group EKT 133.38.209.0 mask 255.255.255.192
policy condition cEK source network group EKT destination network group EK
policy action OK
policy rule rEK precedence 10 condition cEK action OK

2.2 テストケース生成

前述したように等価性を判定出来る最大のテストケースは，2^{64} 個のテストデータを持つテストケースである．しかし，このように莫大なテストデータ数では，等価性判定処理時間がかかりすぎであり，目的である等価性判定処理の時間を，実用化可能な範囲内に収めることは出来ない．故に，2^{64} 個のテストデータから，利用しないで済む無駄なテストデータを出来る限り削除し，処理効率をあげることが目的達成には必要不可欠となる．そこで，本論文では，ACL における policy の condition で記述された通信路のみに着目し，その source と destination を抽出することで，テストケースを小さくすることを考える．

2.3 ACL の condition を利用したテストケース生成

ACL の condition で設定されている送信元 (source)IP アドレスと送信先 (destination)IP アドレスを利用したテストケース生成を考える．ACL の condition を利用したテストパターン生成の一部を表 2 と表 3 を例として説明する．

表 2 ACL の condition を利用したテストパターン生成 1/2

policy network group G1 IPaddressA/mask IPaddressB/mask
policy network group G2 IPaddressC/mask
policy network group G3 IPaddressD/mask IPaddressE
policy condition C1 source network group G1 destination network group G2
policy condition C2 source network group G1 destination network group G3
policy action OK
policy rule R1 precedence 100 condition C1 action OK reflexive
policy rule R2 precedence 10 condition C2 action OK reflexive

表 2 の condition の C1 という通信路では，source が G1 で destination が G2 となる．

表 3 ACL の condition を利用したテストパターン生成 2/2

IPaddressA/mask → IPaddressC/mask
IPaddressB/mask → IPaddressC/mask
IPaddressA/mask → IPaddressD/mask
IPaddressA/mask → IPaddressE
IPaddressB/mask → IPaddressD/mask
IPaddressB/mask → IPaddressE

また，どちらも network group により IP アドレスでグループ化されているので，policy network group G1 を参照すれば，G1 は IPaddressA/mask IPaddressB/mask であることが分かる．同様に G2 も IPaddressC/mask，つまり C1 は IPaddressA/mask IPaddressB/mask から IPaddressC/mask への通信を記述している．組み合わせを考慮すれば，テストパターンは IPaddressA/mask → IPaddressC/mask　IPaddressB/mask → IPaddressC/mask となる．source が G1 で destination が G2 となっている C2 という通信路に対しても同様な処理を行えば，テストデータとテストパターンは IPaddressA/mask → IPaddressD/maskIPaddressA/mask → IPaddressE，IPaddressB/mask → IPaddressD/mask，IPaddressB/mask → IPaddressE となり以下表 3 のようなテストパターンが生成される．

3. アルゴリズム

前述した ACL の condition で生成したテストケースのアルゴリズムの擬似コードを示す．

ACL1 ACL2:テストケース生成に用いる ACL
　PT:テストパターンの集合を保存するファイル
　i,j,k,line:integer
ACL1 を一行読み込む
while ACL1 の最後の行ではない **do**
　if 読み込み内容が policy condition にマッチングした **then**
　　if source が network group で記述されている **then**
　　C_i:=network group の名称
　　A_i:=C_i
　　else if source が IP アドレスで記述されている **then**
　　A_i:=source の IP アドレス
　　else if source が any で記述されている **then**
　　A_i:=any
　　end if
　　if destination が network group で記述されている **then**
　　D_i:=network group の名称
　　B_i:=D_i
　　else if destination が IP アドレスで記述されている **then**
　　B_i:=destination の IP アドレス
　　else if destination が any で記述されている **then**
　　B_i:=any
　　end if
　　A_i → B_i の表記でテストパターンを PT に追加
　　i を +1 する
　end if
　ACL1 を一行読み込む
end while
ACL1 を一行読み込む
while ACL1 の最後の行ではない **do**
　if 読み込み内容が policy network group にマッチングした **then**
　　if C_j に対応する network group がある **then**
　　E_j:=その network group でグループ化されている IP アドレス
　　else if D_j に対応する network group がある **then**
　　F_j:=その network group でグループ化されている IP アドレス
　　end if
　　j を +1 する
　end if
　ACL1 を一行読み込む
end while
PT を一行読み込む
while PT の最後の行ではない **do**

while C_k が存在する $\vee D_k$ が存在する **do**
if 読み込み内容が C_k にマッチングした **then**
A_k を E_k に変換する
else if 読み込み内容が D_k にマッチングした **then**
B_k を F_k に変換する
end if
k を +1 する
end while
PT を一行読み込む
end while

ACL2 も ACL1 と同様に処理する．

このアルゴリズムを更新前と更新後の ACL 両方に適応することで, 各々でテストパターンの集合を保存するファイルが出来るので，　それを合わせればテストケースを生成できる．

3.1 サブネットマスク整理によるテストデータ削減

ここまでで作成されたテストケースをさらにサブネットマスクを利用し，各ネットワークセグメントから 1 つのアドレスを利用すればテストケースとしては十分である．よって，各ネットワークセグメントから 1 つのアドレスを取り出しテストケースを作る．サブネット整理によるテストデータ削減のアルゴリズムの擬似コードは以下である．

PT:既に作成したテストパターンを含むテストケース
a,b:integer
PT を一行読み込む
while PT の最後の行ではない **do**
if マスクがかかった送信元 IP アドレスにマッチング **then**
マッチングしたマスク付き IP アドレスの範囲を求める
min_a:=求めた範囲の最小 IP アドレス,max_a:=最大 IP アドレス
min_a,max_a を十進数に変換して PT2 に追加．a を＋ 1 する
else if マスクがかかっていない送信元 IP アドレスにマッチング **then**
$nmas_b$:=マッチングした IP アドレス
$nmas_b$ を十進数に変換して PT2 に追加,b を＋ 1 する
end if
PT を一行読み込む
end while

i,l,m,n:integer
min:マスクから求めた, 範囲の最小 IP アドレスが格納された配列
max:マスクから求めた, 範囲の最大 IP アドレスが格納された配列
$nmas$:マスクがかかっていない IP アドレスが格納された配列
sp,dp:セレクトポイントが格納された配列
s:重複排除, ソート済み, 十進数が格納された配列
PT2 の数値群を昇順でソートし, 重複を排除する
while PT2 の最後の行ではない **do**
if 十進数の数値にマッチング **then**
s_i:=マッチングした数値
end if
i を＋ 1 する
PT2 を一行読み込む
end while
l=1
for l<i **do**
m=l+1
if s_l が min か max に含まれている **then**
隣り合う数値 2 点から範囲の中間点 (=セレクトポイント sp) を取る
$(s_l + s_m) \div 2 = sp_l$
else if s_l が nmas に含まれている **then**
sp_l:=s_l
end if
l を＋ 1 する
end for

セレクトポイントの値を IP アドレス形式の 2 進数に変換する

同様に送信先 IP アドレス (destination) に対しても処理を行い, セレクトポイント dp_n を得る．

l=1,n=1
while sp_l が存在する **do**
while dp_n が存在する **do**
$sp_l \rightarrow dp_n$
n を＋ 1 する
end while
l を＋ 1 する
end while

3.2 テストケースの正当性証明

ここまでで示したテストケースの生成アルゴリズムでは次の性質が成り立つ.

定理 1. 次の *2* つの命題は同値である.

- *2* つの *ACL* が等価である
- *2* つの *ACL* から生成されるテストケースに含まれる任意のテストデータについて、通信の許可不許可が *2* つの *ACL* で一致する

この定理により，2 つの ACL がテストケースについて同じ振る舞いをすれば 2 つの ACL は等価であることが言える.

4. 実　　験

この章では，実際に埼玉大学で用いられている ACL の一部を，本研究で実装したプログラムにより処理し，出力するテストケースの大きさ (テストデータ数) を調べる．これにより仮想環境下において等価判定に要する処理時間を予想する.

なお，本実験で得られたテストケースを，ACL が適用された firewall 上で実際に動作させ，等価性を判定することは行わない.

4.1 実 験 環 境

本研究で実装したプログラムは，2 つの ACL(更新前 ACL と更新後 ACL) を入力とし，それらに対応して　等価性判定を可能とするテストケースを出力するプログラムである．また，開発において使用したプログラム言語は，正規表現によるテキスト処理に強い Perl を用いた．理由として，ACL 中の文字列の中から特定のパターンを見つけ出したり，置換したりすることが頻出する事が開発前から想定されていたので，そのような処理が容易に出来るプログラム言語を選択する必要があったからである．そして，Perl の開発環境として ActivePerl を採用した．実際に使用する ACL を表 4 に示す．生成されたテストケースに含まれるテストデータ数を以下の表 6 に示す.

表 4　実験で用いる ACL

	ACL1	ACL2
Maximun number of policy rules	103	176
Maximun number of policy conditions	103	176
Maximun number of policy actions	2	2
Maximun number of policy service	13	13
Maximun number of policy groups(network, MAC, service, port)	181	271

次に，本実験をした環境の PC 性能を以下の表 5 に示す.

表 5　実験環境

OS	Windows Vista 32bit 版
CPU	Intel(R)Core2 Duo 2.40Ghz
メモリ	2.00GB

表 6　テストケースに含まれるテストデータ数

テストケース生成法	テストデータ数 [個]
全てのテストデータを網羅したテストケース (I)	2^{64}
ACL の condition を利用したテストケース (Ⅱ)	27399981123850($\fallingdotseq 2^{45}$)
Ⅱ のサブネットマスクを整理したテストケース　(Ⅲ)	160446($\fallingdotseq 2^{17}$)

4.2 等価性判定処理の予想時間

次に，表 6 の結果を受け，テストケースを ACL に処理させた際に生じる等価性判定処理の時間を予想する．想定環境は 1Gbps ギガビットイーサを用い，通信機器が理論上の最高転送速度となるワイヤースピードで動作すると仮定する．ネットワーク機器のパケット処理速度は，スイッチが 1 秒間に処理することができるフレームの「数」を表す pps(packet per second) により表される．1Gbps のワイヤスピードの性能を有するネットワーク機器ではパケット処理速度は 1488100pps となる．1 パケットがひとつのテストデータに対応することを考慮し，等価性判定処理の予想時間を以下の表 7 に示す.

Ⅱ は第 4 章のアルゴリズムにより生成されたテストケースであり，Ⅲ は第 5 章のアルゴ

表 7 等価性判定処理の予想時間

	テストデータ数 [個]	予想時間 [s]
I	2^{64}	約 12297829382473(≒393080 年)
II	27399981123850(≒ 2^{45})	約 18412728(≒ 213 日)
III	160446(≒ 2^{17})	約 0.10781

リズムにより生成されたテストケースである．

5. 考　　察

本研究の目的は，ネットワークにおける ACL の更新前後で，それらが同様のアクセス制御を行っているかを確かめる等価性判定処理の方法を考えること．そして，その等価性判定処理の時間を，従来法よりも短縮させ，実用化可能な範囲内に収めることであった．

従来の方法では，仮想的環境下において等価性判定処理に約 97 分かかっていたのに対して，今回提案した方法では約 0.1 秒となり，等価性判定処理の時間を，既存の方法よりも短縮することが出来た．また約 0.1 秒という短時間で処理を行えるため，常駐システムを稼動させた状態で更新することが可能となり，処理時間は実用化可能な範囲内に収められたと考えられる．

目的達成の一因として，第 5 章のサブネットマスク整理によるテストデータ削減があげられる．マスク単位で許可属性が決まっているので，ある IP アドレスにおけるサブネットマスクによって割り当て可能な IP アドレスは，全て同じ許可属性を示す性質を利用し，その割り当て可能な IP アドレスを代表する IP アドレスを一つだけ抽出することで，それ以外の無駄　なテストデータを大幅に削減出来た．

しかしながら，OmniSwitch におけるどのような ACL に対しても，上記で述べたような処理時間になるわけではなく，本研究では特定の ACL を用いてひとつの実験結果を示したに過ぎない．故に，更なるサブネットマスク整理の効率的なアルゴリズムの開発・改良を行うことは必要であると考える．

6. おわりに

今回，ACL に記述された condition という policy を利用したテストケース生成と，IP アドレスのサブネットマスク整理によるテストデータ削減を提案したことで，ネットワークにおける ACL の更新前後で，それらが同様のアクセス制御を行っているかを確かめる等価性判定処理方法の確立と，その処理時間を従来法よりも短縮し，等価性判定処理時間を実用化可能な範囲内に収められることを本論文で示すことが出来た．これにより，許可属性を変更したくない ACL 更新の際に，記述の誤りや設定ミスなどに気付き，それが原因のセキュリティ被害を事前に防げるようになった．

第 6 章で述べた，サブネットマスク整理のアルゴリズムは，その正当性こそ証明したものの，最適なアルゴリズムであるとは言えない．故に，これから更に最適アルゴリズムへ向けた開発・改良を行うことが必要である．また，その他にもプログラムの拡張 (IPv6 対応，多様なハードウェアに対する互換性の獲得) 問題もあり，解決しなくてはいけない．そして，最終的には実際に動作している firewall における ACL を使用して，等価性判定処理の実験を行っていきたい．

参 考 文 献

1) Vic Gront，John McGinn，John Davies.”Real-time optimisatoin of access control lists for efficient Internet packet filtering”，J Heuristics，2007
2) Dominique Alessandri: ”Access Control List Processing in Hardware”,Diploma Thesis,Zurich,Switzerland:ETH,Electrical Engineering Department,1997.
3) Shingo Ata,Haesung Hwang,Koji Yamamoto,Kazunari Inoue,Masayuki Murata: ”Managment of Routing Table in TCAM for Reducing Cost and Power Consumption”, IEICE technical report.information networks,2007.
4) Alcatel-Lucent:http://www.alcatel-lucent.com/wps/potal/
5) Alcatel: ”OmniSwitch 7700/7800 OmniSwitch 8800 Network Configuration Guide”,2005.