踏み台攻撃防止のための通信状態ベースアクセス制御

安藤 玲未† 佐々木 貴之† 島 成佳† 岡村 利彦†

†日本電気株式会社 クラウドシステム研究所

211-8666 神奈川県川崎市中原区下沼部 1753

r-ando@ap.jp.nec.com

あらまし 多くの組織は，インターネットから組織網内にあるサーバや端末等のノードへのリモートアクセスを許可しており，そのリモートアクセスはファイアウォール等によって境界で制御されることが一般的である．組織網内のノードへのアクセスが許可されると，そのノードを踏み台にして組織のシステムを攻撃される恐れがあるが，境界でのアクセス制御ではそのような踏み台攻撃の防止が困難である．本稿では，リモートアクセスされたノードから組織のシステムへのアクセスを制限することで踏み台攻撃によるリスク軽減を図る，通信状態ベースアクセス制御を提案する．

Communication State-Based Access Control for Preventing Stepping-stone Attacks

Remi Ando† Takayuki Sasaki† Shigeyoshi Shima† Toshihiko Okamura†

†Cloud System Laboratories, NEC Corporation
1753, Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa, 211-8666, JAPAN
r-ando@ap.jp.nec.com

Abstract Many organizations allow remote accesses to internal systems. These accesses are controlled by a firewall or a gateway at the border between internal network and the Internet. Once an access to an internal node is allowed, the employee can access all internal systems via the node. Thus, it is hard to prevent a situation where a compromised node attacks other internal nodes (called stepping-stone attack).
In this paper, we propose a communication state-based access control mechanism which safely allows remote accesses and reduces risks of the stepping-stone attacks.

1 はじめに

近年，より高速な無線ネットワークや，高機能化したモバイル機器が普及し，組織外での業務が効率的に行えるようになった．加えて，在宅勤務や組織外での業務効率化等，働き方の多様化が進んでいる．そのため，組織外にいる際は組織網へリモートアクセスをして仕事をするワークスタイルが主流となっている．このようなワークスタイルでも作業を効率的に行うためには，組織内にいる時と同様の環境で作業できることが必要である．そ

のような中で，組織内の端末にログインをして業務を行いたいというニーズがある．その理由は，組織網内でユーザが普段使用しているノードにアクセスをすると，組織外にいても組織内と同様の環境で作業することができるためである．

しかし，リモートアクセスには踏み台攻撃というリスクが伴う．具体的には，組織網内の端末へのアクセスがファイアウォールによって許可されると，すべてのシステムが踏み台攻撃の危険にさらされる．その理由は，インターネットから組織網内にある個人ノードへアクセスする際，ファイアウォールに代表される境界でのアクセス制御が一般的であるためである．例えば，リモートデスクトップ接続により組織網内のノードへアクセスし，組織内にいる時と同様に作業をする場合を考える．この場合，攻撃者が組織網内にある個人ノードへのアクセスに成功すると，攻撃者はそのノードを踏み台として操作し，組織内の他のノードを攻撃できる．

そのため，踏み台攻撃を防止する技術が必要である．しかし，ファイアウォールなどの既存技術では個人ノードにアクセスを許可するか否かの制御しかできない．よって，踏み台攻撃のリスクを負って個人端末へのリモートアクセスを許可するか，個人端末のアクセスを禁止するかを選択することになり，リモートアクセスによる利便性と踏み台攻撃防御による安全性の両立ができない．

そこで本稿では，ノードがアクセスしているネットワークの種類や状態に基づいて制御することにより，リモートアクセスと踏み台攻撃の防御が両立する仕組みを提案する．

2　課題と解決策

2.1　課題

既存のアクセス制御には，リモートアクセスによる利便性と踏み台攻撃防止による安全性のトレードオフがある．現状を通信状態の観点から考察すると，通信状態は以下の 2 つの状況に分類できる．

図 1：踏み台となる通信状態

A)　ノード 1 からノード 2 へ通信①をした後，ノード 2 からノード 3 へ通信②を行うとき(ノード 2 が「踏み台」となる)

B)　ノード 2 からノード 3 へ通信②をした後，ノード 1 からノード 2 へ通信①を行うとき(ノード 2 が「踏み台」となる)

ファイアウォールやゲートウェイ等は，ネットワークの境界と，組織内部とに設置される．具体的には，A)の状況においてノード 1 が組織外にある時，図 1 では①の通信部分にファイアウォールが設置される．このファイアウォールでは，①の部分しか監視しないので，組織外からの踏み台攻撃を防御するためには，①の通信を禁止する．すなわち組織内の個人ノードへのリモートアクセスを禁止することになり，利便性と安全性が両立できない．

②には，内部ファイアウォールが設置される．内部ファイアウォールで踏み台攻撃を防ぐ場合，リモートアクセスされているか否かに関わらず常にノード 2 のアクセス先を制限する必要がある．そのため，組織内で直接ノード 2 を使うときでもアクセスが制限され，不便である．よって，内部のファイアウォールも，利便性と安全性を両立できない．

ノード 1 が組織内にある場合，①や②には内部ファイアウォールが設置される．前述した内部ファイアウォールの現状と同様に，ノード 1 及びノード 2 のアクセス先を常に制限する必要があり，利便性と安全性を両立できない．

上記では，A)の状況について分析したが，B)の状況においても，A)の状況と同様に利便性と安全性を両立できない．そこで本稿では，利便性と安全性が両立するシステムを提案する．

2.2　解決策のアプローチ

本稿では，以下の 2 点の特徴を持つアーキテクチャにより，2.1 節で述べた課題を解決する．

① ノード状態ベースのアクセス制御
② ネットワークモニタリングによるノード状態の動的判定

まず，①のノード状態ベースのアクセス制御について述べる．組織外からリモートで組織内情報にアクセスして仕事を行う際と，リモートアクセスされた後，踏み台攻撃となる際のノードの使われ方を分析し，以下のことが分かった．

- **仕事をする際にノードをローカルで操作しているとき**：組織内にあるノードは，社内の様々なシステムにアクセスするクライアント状態として動作している．「クライアント状態」とは，ノードからコネクションを張る状態である．
- **リモートアクセスにて操作をしているとき**：組織内にあるノードは，組織外からリモートアクセスを受け付けるサーバ状態として動作している．「サーバ状態」とは，コネクションを受け付けた状態である．
- **踏み台攻撃のとき**：あるノードが同時にコネクションを受けて，更に他のノードへアクセスすることから，クライアント状態及びサーバ状態として動作している．

よって，踏み台攻撃を防止するためには，クライアント状態もしくはサーバ状態となることを許可し，同時に両方の状態になることを禁止すればよい．これを，①ノード状態ベースのアクセス制御で実現する．

次に，②ネットワークモニタリングによるノード状態の動的判定について述べる．利便性低下を防ぐため，ネットワーク全体のトラフィックをモニタリングすることでノードの状態を動的に判断し，アクセス制御を行う．動的に状態を判断するため，組織内でノードを直接使用している時は，アクセスは制限されず，リモートアクセスしている時のみアクセスが制限される．必要な時のみアクセス制御を行うので，利便性の低下を防ぐことができる．

加えて，ユーザごとのアクセス制御や例外処理にも対応するため，ノードの状態だけではなく，例外ルールによるアクセス制御も必要である．本手法では，業務に必要な正当な踏み台も禁止してしまうが，例外ルールを記載することにより，一部の踏み台を許可することができる．

以上，①による攻撃の防御と，②による利便性低下の防止により，課題を解決する．

3　通信状態ベースアクセス制御

前章で述べた 2 つの機能を持つ Policy Decision Point(PDP)とノード間の通信のフックを行う Policy Enforcement Point(PEP)を備えたアーキテクチャを提案する(図 2)．PDP は，PEP から，ノード間の通信情報を受け取り，その通信可否をノード状態に基づいて判断する．そして，その判断結果を PEP へ返すことで通信を制御する．

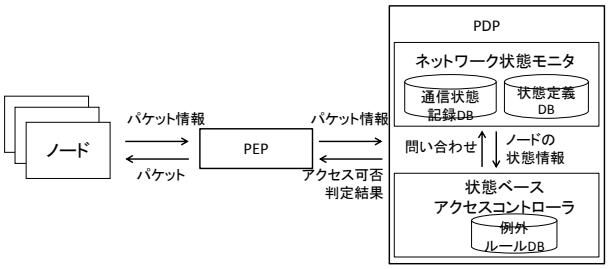

図 2：アーキテクチャ

3.1　PDP

2.2 節で述べた 2 つの特徴を実現するために，PDP は，ノードの状態を動的に判定するためのネットワーク状態モニタと，ノードの状態に基づいてアクセスの可否を決定する状態ベースアクセスコントローラを持つ．状態ベースアクセスコントローラは，ネットワーク状態モニタへノードの状態を問い合わせ，ノードの状態に基づいてアクセス制御を行う

ことで，あるノードが，同時にクライアント状態かつサーバ状態になることを防止する．

3.1.1 ネットワーク状態モニタ

ネットワーク状態モニタは，組織内の通信全てを監視して各ノードの通信状態を記録し，ノードの状態を判定する．そのために，通信状態を記録する通信状態記録データベースを備える．加えて，「サーバ状態」と「クライアント状態」を定義するために，状態定義データベースを備える．

通信状態記録データベース

通信状態記録データベースは，図 3 に示すように，送信元 IP アドレス(src_ip)・送信元ポート番号と，送信先 IP アドレス(dst_ip)・送信先ポート番号を記録する．ネットワーク状態モニタは，状態ベースアクセスコントローラから問い合わせがあった場合に，問い合わせに含まれる上記の 4 つの情報を検索し，データベースに登録されていなければ，新規エントリを追加する．

状態定義データベース

状態定義データベースは，図 4 に示す構造であり，サーバ状態とクライアント状態を通信の状態のパターンとして予め定義しておく．通信状態のパターンは，1.どのポート番号から送信/受信したか，2.状態をサーバ/クライアントと定義するか，という 2 つの情報で定義される．例えば図 4 の 1 番目のエントリは，ポート番号 3389(RDP)を用いてコネクションを受け付けたならサーバ状態とすることを意味し，2 番目のエントリは送信元ポート番号が 1234 を用いてコネクションを張ったならクライアント状態とすることを表している．

ノードの状態判定

ネットワーク状態モニタは，状態ベースアクセスコントローラから問い合わせがあった場合，通信状態記録データベースと，状態定義データベースを用いてノードの状態を判定する．具体的には，状態ベースアクセスコントローラから受信したノードの IP アドレスと通信状態記録データベース，状態定義データベースに登録されているノードの IP アドレスをマッチングさせることで状態判定を行う．

例えば，状態定義データベースが図 4 の 1 番目のエントリが定義されている場合を考える．この場合，あるノードがポート番号 3389 番によってコネクションを受け付けていた場合はサーバ状態である．具体的には，図 3 の 2 番目のエントリの 192.168.0.3 のように，ポート番号 3389 で接続を受けており，ポート番号 3389 が図 4 の 1 番目のエントリのように，dst と定義されている場合はサーバ状態となる．また，図 4 の 2 番目のエントリのような定義がある場合，図 3 の src_ip に，ノードの IP アドレスが記録されており，かつ送信元ポート番号が 3389 の場合にクライアント状態となる．また，「サーバ状態」「クライアント状態」のいずれでもない場合，「初期状態」とする．

src_ip	送信元 ポート番号	dst_ip	送信先 ポート番号
192.168.0.1	5001	192.168.0.2	5001
192.168.0.2	3389	192.168.0.3	3389

図 3：通信状態記録データベース

ポート番号	送信元・送信先	ノードの状態
3389	dst	サーバ
1234	src	クライアント
5001	dst	サーバ
5678	src	クライアント

図 4：状態定義データベース

3.1.2 状態ベースアクセスコントローラ

状態ベースアクセスコントローラは，端末の状態に基づいてアクセス可否の判定を行う．また，状態ベースアクセスコントローラは，ユーザごとのアクセス制御や例外処理を行うために例外ルールデータベースを持つ．アクセス可否判定は，例外ルールが優先され，状態ベースアクセス制御は，通信が例外ルールに一致しないときに行われる．全体の動作を以下に示す．

1. PEP から問い合わせを受ける.
2. 例外ルールをチェックする.
3. ノードの状態を，ネットワーク状態モニタに問い合わせる.
4. 状態に基づいてアクセス可否の判定を行う.
5. PEP にアクセス可否の判定結果を送信する.

以下では，状態ベースアクセスコントローラの特徴である，2.例外ルールと 4.状態ベースアクセス制御について説明する.

例外ルールデータベース

例外ルールデータベースは図 5 のような構造であり，ネットワークから届いたパケットは，まずこの例外ルールデータベースにエントリがあるか確認される．PEP は，パケットに合致する例外ルールがあればそのアクションに書かれている情報を判断結果として PEP へ返す. 例えば受信したパケットの情報が「src_ip=192.168.0.2, dst_ip=192.168.0.3」であった場合，図 5 に示す例外ルール DB を検索すると，1 番目のエントリと合致する．このパケットに関するアクションは deny であるため, PDP はパケットを破棄するように, PEP に指示する.

パケットの情報に合致する例外ルールがない場合，PEP は，状態ベースのアクセス制御を行う.

src_ip	dst_ip	アクション
192.168.0.2	192.168.0.3	deny
*	192.168.0.1	allow
192.168.0.2	*	allow

図 5：例外ルールデータベース

状態ベースアクセス制御

2.2 の解決策のアプローチで示したように，あるノードが同時にサーバ状態とクライアント状態にならないように通信を制御すればよい．よって，あるノードがコネクションを張ろうとしたときに，そのノードと，通信先のノードの両方に対して，以下のようなパターンに基づいて判定を行い，どちらも許可であれば通信が許可され，いずれかが禁止であれば通信は禁止される.

表 1：通信元ノードのアクセス可否判定

コネクションを張るノードの条件	アクション
サーバ状態の時	禁止
クライアント状態の時	許可
初期状態の時	許可

表 2：通信先ノードのアクセス可否判定

コネクションを受けるノードの条件	アクション
クライアント状態の時	禁止
サーバ状態の時	許可
初期状態の時	許可

表 1, 2 のアクセス可否判定ルールにより，同時に 2 つの状態になることが防げる．具体的には，既にサーバ状態として動作している時，コネクションを張ると同時にクライアント状態にもなるため，これを禁止する(表 1). 反対に，既にクライアント状態として動作している時，コネクションを受け付けてしまうとサーバ状態となるため，これを禁止する(表 2).

3.2 PEP

PEP は，ノード間の通信をフックし，PDP の状態ベースコントローラへ通信の可否を問い合わせ, PDP の指示に基づいて通信の転送と遮断を行う．具体的には，すべての通信について，以下の 3 つのステップを実施する.

1. ノード間の通信をフック
2. 新しい通信であれば，PDP に，通信の可否を問い合わせる. 問い合わせ済みの通信であれば，過去の PDP の指示を参照する.
3. PDP の指示が許可であれば，ノード間の通信を転送し, 禁止であれば通信を遮断する.

4 実装

本稿の提案手法の機能性を評価するために,

OpenFlow [1]を用いて実装し，評価実験を行った.

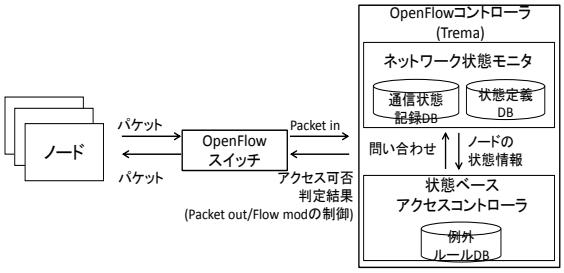

図 6：OpenFlow を用いた実装

4.1 OpenFlow

OpenFlow は，スイッチやルータなどのネットワークノードのデータプレーンとコントロールプレーンを分離し，OpenFlow スイッチはパケットの転送とパケットの改変をするネットワークプレーンとして動作し，分離されたコントロールプレーンは，コントローラと呼ばれるサーバにより集中して OpenFlow スイッチを制御する．OpenFlow コントローラはフローを単位として OpenFlow スイッチへフローの制御内容を指示する．このため，従来のスイッチの挙動と異なり，コントローラによって任意の粒度・任意のレイヤで扱うことができ，柔軟かつ自由にプログラムできる特徴を持つ.

OpenFlow スイッチは，未知のパケットを受信した際，コントローラにそのパケットを通知する．これは Packet in メッセージと呼ばれており，パケットの送受信に必要な情報が含まれている．Packet in メッセージでコントローラに送られてきたパケットを本来の宛先に送るために，コントローラからスイッチ側へ送り返す際は Packet out メッセージを用いる．また，コントローラが作成したフローをスイッチへと送るために Flow mod メッセージを用いる．フローの情報は，スイッチのフローテーブルに記録され，2 番目以降のパケットはフロー情報を基に転送される.

本提案手法を，従来のファイアウォールを用いて実装しようとすると，全てのホスト間の通信を監視するために，多数のファイアウォールやそれらを管理するコントローラが必要である．一方，OpenFlow では，コントローラによるスイッチの集中管理が既に実装されており，コントローラのみに提案手法を導入すれば，提案するアーキテクチャを実装可能であるため，OpenFlow を採用した.

4.2 OpenFlow を用いた実装

前章で述べた PDP を OpenFlow コントローラ，PEP を OpenFlow スイッチとして実装した．OpenFlow コントローラにはオープンソースの Trema[2]を用い，Trema のアプリケーションとして，ネットワーク状態モニタと，状態ベースアクセスコントローラを実装した．PEP は，OpenFlow スイッチを，改造せずに使用した.

図 2 中の「パケット情報」は，OpenFlow での Packet in メッセージに相当する．取得した情報は，OpenFlow コントローラ内に実装された各データベースに登録される．図 2 中の「ノード状態によるアクセス可否もしくは例外ルール」は Packet out メッセージによるパケットの送信，Flow mod メッセージによるスイッチのフローテーブルへの書き込みとして実装した．以下では，3 章で示した各コンポーネントに関し，Trema 上でどのように実装したかを説明する.

ネットワーク状態モニタ

ネットワーク状態モニタの通信状態データベースと状態定義データベースは，Ruby の hash を用いて実装した. ネットワーク状態モニタは，状態ベースアクセスコントローラから，パケットの状態情報に関する問い合わせがあった際，状態ベースアクセスコントローラから渡されたパケットの送信元・送信先 IP アドレスやポート番号と 2 つのデータベースを確認し，ノードの状態情報を状態ベースアクセスコントローラへ返す.

状態ベースアクセスコントローラ

状態ベースアクセスコントローラは，OpenFlow スイッチから渡された Packet in メッセージに含まれるパケットの送信元・送

信先 IP アドレスやポート番号情報を取得する．取得結果をもとに例外ルールデータベースを検索し，該当するルールがあれば，ルールに記載されているパケットのアクセス制御結果(allow,deny)をスイッチへ返す．なければネットワーク状態モニタからホストの状態を受け取り，状態によるアクセス制御を行う．

パケットのアクセス制御結果が allow の際は，PDP である OpenFlow コントローラから PEP である OpenFlow スイッチへ Packet out メッセージを送ることにより，OpenFlow スイッチはパケットのアクセス許可が下りたと判断し，パケットを目的のノードへ送信する．また，Packet out メッセージと同時に OpenFlow コントローラから OpenFlow スイッチへ Flow mod メッセージを送ることで，OpenFlow スイッチのフローテーブルにこのパケット(first packet)についてのルールが記載されるため，次回同じパケットが OpenFlow スイッチに届いた場合，OpenFlow コントローラへ問い合わせをすることなく処理することができる．

パケットのアクセス制御結果が deny だった場合は，OpenFlow コントローラから Packet out メッセージも Flow mod メッセージも送らない．このように，パケットの制御結果により，packet_out メッセージと flow_mod メッセージを制御することで状態ベースのアクセス制御を実現している．

スイッチングハブ機能

図 2 のアーキテクチャには示されていないが，OpenFlow を用いる際，スイッチの基本機能として，パケットを転送する機能が必要なので，スイッチングハブ機能を実装した．スイッチングハブ機能は，FDB(Forwarding DB)に，ノードの MAC アドレスとポートの組を学習することで，パケットの転送を行う．

5 評価

提案手法を導入することにより利便性と安全性が両立できていることを確認するため，機能評価を行った．また，提案手法の処理にかかるオーバヘッドを確認するため，提案手法を導入する前後でかかる処理時間の比較を行った．

5.1 機能評価

機能評価では，以下のことを確認した．

- ネットワーク状態モニタリングによるノード状態の動的判定が行えること
- ノード状態ベースのアクセス制御が行えること

Trema のネットワークネームスペース機能である Netns を利用し，OpenFlow スイッチ 1 台とノード 3 台のネットワークをシミュレーションした．想定するシナリオは，3 台のノードがそれぞれ，端末，踏み台のデスクトップ，サーバとして動作している場合である．このとき，デスクトップを踏み台として利用しているときは，デスクトップはサーバにアクセスできなければよい．

まず，端末からデスクトップへ netcat コマンドにより TCP コネクションを張り，端末がデスクトップへリモートアクセスした状況を作り出した．この時，状態定義データベースによりデスクトップは「サーバ状態」として動作していると判定された．この状態で，netcat コマンドにより TCP コネクションを張り，リモートアクセスしているデスクトップからサーバにアクセスしようとすると，アクセスが拒否されることが確認できた．これは，デスクトップは「サーバ状態」であると判定されているため，コントローラはデスクトップがクライアントとして動作する今回の通信を禁止する判定をしているためである．これらのことから，1.動的にノードの状態判定が行われ，2.その状態に基づいてアクセス制御が実施されていることが確認された．

5.2 性能評価

提案手法は，新規通信が開始する時，First packet に対してアクセス可否の判定を行う．

そのため，First packet 以降のパケットに関して遅延が発生することはない．そこで，First packet のアクセス制御の判断にかかる処理時間が通信処理全体の時間に比べて十分小さいことを確認する．

表 3 に示した評価環境にて Trema の通信処理全体の時間とアクセス制御の処理時間を表示させた．具体的には，以下の 2 点を行った．

1. 提案手法を導入する前後の通信処理全体の遅延を ping により検証
2. Trema のアクセス可否判定部分にかかる時間を time 関数により取得

1.による通信処理全体の遅延時間を計測したが，提案手法の導入前後でどちらも 33msec であり，差はなかった．また，2.を測定したところ 5μsec であり，無視できるほど小さく，通信全体に影響はないことが明らかになった．

表 3：評価環境

CPU	Intel Xeon 2.4GHz
メモリ	64Gbyte
OS	Ubuntu12.04
Trema バージョン	0.3.3

6 関連研究

踏み台攻撃の防止を課題として挙げている研究は，外部からの通知による不正侵入被害拡散防止システム[3]や xfilter[4]などがある．

不正侵入被害拡散防止システムでは通信ポリシによる踏み台攻撃の拡散防止を行おうとしている．この通信ポリシは，通信を行う全マシンに導入されているため，あるホストのセキュリティ機能が無効化されても接続先のホストで接続をブロックできるので，踏み台攻撃による被害を最小限にすることができる．

xfilter は，仮想マシン(VMM)内において，VM のメモリ解析を行って取得したゲスト OS 内の情報を用い，踏み台攻撃を行っているプロセスからのパケットのみを破棄するフィルタを VMM 内に設置する手法である．本稿の提案手法はネットワークをモニタリングして通信をアクセス制御する手法であるため，これらのホストに制御機構を導入する仕組みと共存可能であり，同時に用いることでさらなる性能向上が見込める．

7 おわりに

踏み台によるリスクを軽減させるために，本稿ではネットワークをモニタリングすることでノードの状態を判定し，ノードの状態に基づいてアクセスを制御するアーキテクチャを提案した．具体的には，通信を確立させようとしている 2 台の端末が，コネクションを受け付けた状態である「サーバ状態」，または，コネクションを張る状態にある「クライアント状態」にあるかどうかの判定を行い，その判定結果，及び，予めシステム管理者等により用意されたルールに基づき，アクセス制御を行う．さらに，OpenFlow を用いて実装・評価を行い，想定したシナリオの踏み台を防止できることを明らかにした．また，パフォーマンス評価を行い，オーバヘッドは無視できるほど小さいことを示した．

今回の提案手法では，業務に必要な正当な踏み台と踏み台攻撃の違いを検知できない．これらを検知することが今後の課題である．

参考文献

[1] OpenFlow：http://www.openflow.org/

[2] Trema：http://trema.github.io/trema/

[3]高橋健一，藤井雅和，桜井幸一，”他者からの知らせによる不正侵入被害拡散防止モデルの提案と評価”, 電子情報通信学会論文誌 D, pp.1114-1124，2010 年 7 月．

[4]安積武志，光来健一，千葉滋，”踏み台攻撃だけを抑制できる VMM レベル・パケットフィルタ”，情報処理学会論文誌 Vol.50 No.2，pp.1234-1241，2010 年 2 月．