Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22

協調型ネットワーク侵入検知システム Brownie の提案と評価

花岡美幸[†1]　河野健二[†1,†2]　廣津登志夫[†3]

近年，ネットワークの高速化や攻撃の巧妙化に伴い，単体のネットワーク侵入検知・防御システム (NIDS/NIPS) でのネットワーク監視は性能面において難しくなってきている．本論文では，組織内に複数設置された NIDS を相互に協調させることで性能向上をする機構 Brownie を提案する．1 カ所に高価なマシンや並列に動作する複数のマシンを置くのではなく，他の NIDS の負荷やルール設定を元に，過負荷のマシンや無駄なルール設定がなくなるように再設定する．実験では，実際のパケットトレースを用い，冗長なルール設定が除去され負荷が分散されたことを確認した．

Brownie: Collaboration of Network Intrusion Detection Systems

Miyuki Hanaoka,[†1] Kenji Kono[†1,†2]
and Toshio Hirotsu[†3]

Because of today's increased traffic volume and sophisticated attacks, implementing a network intrusion detection/prevention system (NIDS/NIPS) with a single workstation has been challenging. In this paper, we propose Brownie, a system for performance improvement via collaboration between already-existing NIDSs in an organization. Instead of installing one expensive hardware or parallel NIDS at a vantage point, our Brownie achieves performance improvement by offloading overloaded NIDS and eliminating redundant rules by examining other NIDSs' load status and configurations. The experimental results with a university full-packet trace suggest that Brownies successfully offloads overloaded NIDS and eliminates redundant rules.

1. はじめに

ネットワーク経由の攻撃の検知及び防御にネットワーク侵入検知・防御システム (NIDS/NIPS: Network Intrusion Detection/Prevention System) が広く用いられている．現在，オープンソースや商用で数多くの NIDS が提供されているが，その多くがそれほど高価でない汎用 PC を元に作られている．そのため，一台の NIDS で全てのトラフィックを監視するのは難しくなっている．その理由として，ネットワークトラフィックの量や速度が CPU の高速化に勝っていること，攻撃が複雑化しているために以前よりもトラフィックを細かく検査する必要が出てきていることなどがあげられる．また，攻撃を防ぐために，パケットの検査が終わるまでそのパケットを止める必要がある NIPS は，NIDS に比べて性能に与える影響が大きい[1)]．

そこで，いくつかの研究では，一台の NIDS の代わりに，並列に動作する複数の NIDS を用いて処理を分散させることで，高速化を目指す手法を提案している．例えば，NIDS Cluster[2)] や Active Splitter[3)] などがある．これらの手法では，フロントエンドにあたるマシンが，複数置かれたセンサと呼ばれるトラフィックを検査するマシンに，トラフィックの一部を割り振ることで処理を分散し，システム全体としての性能を向上させている．この手法は非常に有効であるが，導入はそれほど容易ではないという問題がある．多くのマシンを導入するため，コストがかかるためである．マシンの購入費だけでなく，それらを適切に設定する必要もある．また，多数のマシンを置くための場所の確保や，十分な電源を供給しなければならないといった問題も出てくる．

そこで，本論文では，別の観点から NIDS の処理を分散させる手法を提案する．既にネットワーク内にある複数の NIDS を連携させることで，全体としての性能向上を目指す．我々の着目点は，大学や企業など多くの組織では，組織への入り口だけでなく，内側のネットワークの様々な位置や階層に複数の NIDS を設置している場合が多い，ということである．例えば大学では，大学の入り口と外のネットワークとの境界以外にも，各学部や各研究室で個別に NIDS を置いていることがある．これらの NIDS はそれぞれの管理者が別々に設定

†1 慶應義塾大学
Keio University
†2 科学技術振興機構 CREST
CREST, Japan Science and Technology Agency
†3 豊橋技術科学大学
Toyohashi University of Technology

Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22

を行うため，同じルールが設定されていることもある．そのため，トラフィックはそれらのNIDS を通るごとに，同じルールについて何回も検査されることになる．また，他の NIDS の負荷が高くなくても，過負荷になったある 1 つの NIDS が原因でネットワーク全体の性能が低下することがある．

本論文では，これらの同じネットワーク内にある複数の NIDS が協調することで，ネットワーク全体の性能を向上させることができることを示す．提案システムでは，ネットワークの経路上にある他の NIDS と相互にルール設定や負荷の状況をやりとりし，NIDS の負荷を減らしたり冗長なルール設定を排除する．1 つの NIDS に集中している処理を分散させることで，各 NIDS の負荷を減らし，過負荷の NIDS が全体のボトルネックになることを回避することができる．また，複数の NIDS で重複しているルールをなくすことで，1 つのパケットに対して同じルールが複数回検査されることがなくなり，ネットワーク遅延が少なくなることが期待できる．従来の手法は，ある 1 点に置かれた NIDS の性能向上を目指しているが，我々は 1 つではなく複数の NIDS による全体の性能を向上することを目指す．

ここで，我々の手法により，セキュリティが低下することはないことに注意してほしい．全てのトラフィックは，提案手法がないときと同じルールセットに対してチェックされる．ルールの移譲や削除は，NIDS を通る全てのトラフィックが，必ずどこか別の NIDS においてそのルールに対してチェックすることが保証されている時のみ行う．

提案手法の効果を確認するために，マシンの負荷に応じてルール設定を変えていくシステム Brownie を実装した．Web ベンチマークを用いた実験では，提案手法によりスループットが 13%増加した．また，豊橋技術科学大学で取得したフルパケットトレースを用いた実験でも，Brownie によって適切に負荷分散ができた．

2. 関 連 研 究

NIDS の性能向上を目指した研究は数多く提案されている．これらは，複数のマシンを用いることによるものと，1 つの NIDS の性能向上を目指したものの，2 つに大別できる．

並列に動作する複数のマシンを用いて NIDS の構成し，性能を向上する研究として，NIDS Cluster[2], Active Splitter[3]，Kruegel らによるもの [4] などがある．これらは，基本的にセンサと呼ばれるトラフィックの検査を行う複数のマシンと，センサにトラフィックを割り当てるフロントエンドから構成されている．トラフィック検査を行うセンサを複数置き，処理を分散させることによって，一台のマシンよりも性能を向上させることができる．Kruegel ら[4] は，高速かつ高精度な検査を可能にするために，三層構造のフロントエンドを提案している．Active Splitter[3] は，性能向上を目的としたフロントエンドの最適化手法を提案し，NIDS Cluster[2] は，センサ同士のやりとりを強化したシステムを提案している．しかしながら，これらのシステムを導入するためのコストは少ないとは言えない．管理者は，複数のマシンを購入し，設置や設定をし，管理しなければならない．また，多くのマシンを設置するのに必要な場所や電源の確保も容易ではない場合がある．本研究は，既に組織内のさまざまな場所に置かれている NIDS を利用し，それらを協調させることで性能向上を目指すという，新しいアプローチを提案している．

一台の NIDS の性能を向上させる手法も数多く提案されている．古くからあるものでは，NIDS の処理の中で最も重い処理である文字列マッチングを高速化する手法の提案であり，Aho-Corasick[5]，Wu-Manber[6]，Commentz-Walter[7] 等が提案されている．また，ハードウェアと組み合わせたものとして，ネットワークプロセッサを用いたもの[8)–10)] や，FPGA を用いたもの[11)–14)] が数多く提案されている．また，近年では，マルチコアプロセッサを用いたもの[15)] や，グラフィックプロセッサを用いたもの[16)] が提案されている．これらと提案手法は共存できる関係にあり，各 NIDS がこれらの手法を取り入れた上で他の NIDS と協調することで，さらに全体の性能向上が期待できる．

3. 提　　案

本論文では，同じネットワーク内にある複数の NIDS が協調することで，性能を向上する手法を提案する．本手法では，同じネットワーク内の NIDS 同士がそれぞれの負荷状況やルール設定をやりとりし合い，適切に再設定していく．従来のように，1 カ所に性能のよい高価な NIDS を置く，または 1 カ所に並列に動作する複数の NIDS を置くのではなく，既にネットワーク内にある NIDS を活用することでネットワーク内全体の性能向上を目指す．

我々の着目点は，大学や会社など多くの組織では，インターネットと組織のネットワークの間に置かれる NIDS 以外にも，組織内ネットワークの様々な場所に NIDS が置かれることが多くある，ということである．図 1 のように，組織内ネットワークの入り口に置かれた NIDS A 以外にも，その下流に，各部門や課によって独自の NIDS が置かれることがある．例えば大学では，学科や研究室でそれぞれ NIDS を設置するといったことが考えられる．図 1 の例では，NIDS B1 と B2 は学科によって，NIDS C1 と C2 は研究室によって設置された，という具合である．本手法では，各 NIDS はそれぞれの一つ上流の NIDS（親 NIDS）とすぐ下流の NIDS（子 NIDS）と通信をする．例えば，NIDS A は NIDS B1 および B2 と，NIDS B2 は NIDS A, C1, C2 とやりとりをする．

Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22

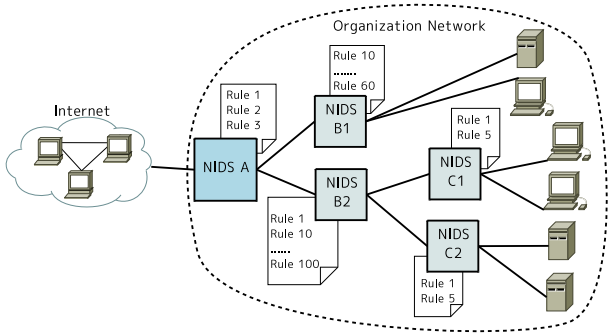


図 1 NIDS の設置例

提案手法では，他の NIDS と協調し，1) 過負荷になった NIDS の負荷を減少させ，2) NIDS 間の冗長なルール設定を削除する，の 2 つのアプローチをとることにより性能向上を目指す．

3.1 過負荷 NIDS の負荷を減少

NIDS は，大量のトラフィックを受け取った，多くのルールに対して検査しなければならない，またマシンが低性能であるなど，様々な理由により過負荷になる．しかし，組織内に置かれた複数の NIDS のうち，全てが過負荷になるという状況はほとんどない．そこで，過負荷になった NIDS の負荷の一部を，ネットワーク経路上にある他の低負荷である NIDS に分散することで，過負荷になった NIDS の負荷を減らす．これにより，過負荷になった NIDS がネットワーク全体のボトルネックになることを防ぐことができ，結果としてスループットの向上や通信遅延の減少が期待できる．負荷を分散させるために，提案システムでは，過負荷な NIDS から低負荷な NIDS にルールを移譲する．すなわち，過負荷な NIDS でいくつかのルールを無効にし，代わりに低負荷な NIDS で同じルールを有効にする．トラフィックの量やマシンの性能をすぐに変更することは難しいため，ルールの数を減らすことで負荷を減らす．

例えば図 1 では，NIDS B2 に多くのルール設定がされており，過負荷であるとする．一方，その下流にある NIDS C1, C2 は設定されているルール数が少なく，低負荷である状況を考える．このとき，NIDS B2 は NIDS C1, C2 に，例えばルール 70〜100 を移譲する．つまり，ルール 70〜100 を NIDS B2 で無効にし，代わりに NIDS C1, C2 で有効にする．こうすることで，NIDS B2 の負荷の一部を NIDS C1, C2 に移すことができる．

ここで，ルールを移譲した後も，元のセキュリティレベルは維持していることに注意してほしい．提案手法では，元々ルールを有効にしていた過負荷 NIDS を通る全てのトラフィックが，ルールを移譲した先の NIDS のどれかを必ず通るためである．そのため，トラフィックは，宛先マシンに届く前に，移譲されたルールに対して必ずチェックされることが保証されている．ルールが上流から下流の NIDS に移譲される場合を考える．これは例えば，図 1 において NIDS B2 から NIDS C1 と C2 にルールを移譲したときである．このとき，上流 NIDS(NIDS B2) を通る全てのパケットは，下流 NIDS のどれか 1 つ (NIDS C1 か C2) を通ることになる．すなわち，全てのパケットは下流 NIDS のどれか 1 つ (NIDS C1 か C2) で，移譲されたルールに対してチェックされることになる．

一方，下流 NIDS から上流 NIDS にルールを移譲する場合，下流 NIDS を通るトラフィックが必ず上流 NIDS も通ることは明らかである．例えば，NIDS B2 から A に移譲した場合，NIDS B2 を通るトラフィックは，NIDS A も通る．しかし，ネットワーク内のトラフィックの一部は，上流 NIDS ではチェックできないものもあるため，別に扱う必要がある．例えば，NIDS C1 の下にあるマシンと，NIDS C2 の下にあるマシンとの通信は，NIDS B2 では監視できるが，NIDS A では監視できない．このような通信を監視するため，下流 NIDS では送信元・宛先 IP アドレスを元に，ネットワーク内のトラフィックに対してのみ移譲したルールを有効にしておく．通常，インターネットからのトラフィックの方がネットワーク内トラフィックに比べて何倍も多いため，一部のトラフィックに対してルールを有効にしたままでも，負荷を軽減することができると考えられる．

また，NIDS の下流に通常のホストと NIDS が同じ階層に設置されていた場合も，必要に応じて一部のトラフィックに対してはルールを有効にしたままにしておく．すなわち，上流 NIDS から下流 NIDS にルールを移譲するとき，上流 NIDS は直接接続されているホスト宛のパケットについては，移譲したルールに対してもチェックを行う．例えば，NIDS C1 が設置されていない構成で，NIDS B2 から NIDS C2 にルールが移譲された時を考える．このとき，NIDS B2 は（今は存在しない）NIDS C1 の下にあるマシン宛のパケットのみはチェックするようにしておくことで，全てのトラフィックが移譲されたルールに対してチェックされるようにする．

3.2 冗長なルール設定を削除

あるネットワーク経路上に置かれた複数の NIDS が，同じルールを重複して有効にしていることは，珍しいことではない．これは，全ての NIDS が同じ管理者によって管理されているわけではなく，各 NIDS がそれぞれ別々の管理者によって管理されているためである．

Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22

例えば上記の大学の例では，大学の入り口に置かれた NIDS は大学のネットワークの管理者によって管理されるが，学科の NIDS は学科のネットワーク管理者によって管理される．ネットワーク経路上に置かれた複数の NIDS に同じルールが設定されていると，その経路を通るパケットは同じルールに対して何度もチェックされることになる．重複しているルールをなくし，ネットワーク経路上のどこか一カ所だけで検査すれば，通信遅延を少なくすることができると考えられる．ここで，前節と同様に，重複したルールをなくすことで，セキュリティレベルが低下することはない．ルールの無効化は，ネットワーク経路上の他の NIDS でも同じルールが有効にしており，その NIDS でチェックされる時のみ行うからである．

例えば図 1 では，NIDS A, B2, C2 の 3 つの NIDS でルール 1 を有効にしている．そのため，NIDS C2 以下にあるマシン宛のパケットは，ルール 1 に対して 3 回検査されることになる．そこで，例えば NIDS B2 と C2 ではルール 1 の設定を無効にして，NIDS A のみでルール 1 について検査するようにすれば，その分通信遅延やマシンの負荷を減らすことができる．また，NIDS B2 と C2 でルール 1 を無効にすることで，セキュリティが低下することはない．なぜなら，NIDS B2 と C2 に届くインターネットからのトラフィックは，NIDS A ですでにルール 1 に対してチェックされているからである．ネットワーク内トラフィックや，直下に通常のマシンがある場合は，3.1 節で述べたのと同様に，該当するトラフィックだけそのルールに対して検査する．

4. 実　　装

NIDS 協調による性能向上ができることを示すため，Brownie を実装した．各 Brownie は各 NIDS を管理し，他の Brownie と通信をすることによって，負荷分散を図る．NIDS を Brownie と共に動作させるには，管理者は親 NIDS の IP アドレスなどの情報を Brownie に与える．NIDS の起動時，対応する Brownie は親 NIDS と共に動作している Brownie とコネクションを確立し，以降それぞれの負荷状況やルール設定を交換する．また，コネクション確立時に，それぞれの NIDS で有効または無効にしているルールリストを交換する．

今回の実装では，NIDS として広く普及しているオープンソース NIDS である Snort[17] を用いた．現在の実装では全ての NIDS が Snort であるが，将来的には様々な NIDS が動作している環境にも適用できるようにしたいと考えている．もう 1 つのオープンソース NIDS である Bro[18] は，動的ルール書き換えやセンサ間の通信機構を備えているという点で，用いる NIDS の候補として考えられる．

4.1 過負荷 NIDS の負荷の減少手順

NIDS の負荷を減少させる手順の基本的な考え方は，自分と子 NIDS の負荷を比較し，負荷が高い方から低い方にルールを移譲する，ということである．つまり，自分の NIDS の負荷が，どの子 NIDS の負荷よりも高い場合，いくつかのルールを子 NIDS に移譲する．自分の NIDS と子 NIDS の負荷が同じくらいになるまで，これを繰り返す．ここで，3 つのことについて考える必要がある．

まず，NIDS の負荷の指標として何を用いるかである．NIDS の負荷は，ルール中に書かれたバイトパターンとのマッチングによるものが多いため，ルール数が指標として考えられる．しかし，1 つのルールが与える負荷は，NIDS の性能やルールの複雑さ，合致するトラフィックの量によって変化する．そのため，同じルール数でも，同じ負荷がかかっているとはいえない．そのため，負荷を測る指標として CPU 使用率を用いた．NIDS の処理のほとんどは CPU インテンシブであるため，マシンの性能が異なる場合でも適切に負荷を分散することができると考えられる．

第二に，ルールの移譲をいつ開始していつ終了するかである．過負荷の NIDS の負荷を減少させるという観点から，NIDS が過負荷になったら（CPU 使用率が 100% 近くなったら）ルール移譲を開始し，CPU 使用率がある設定値を下回ったら終了する，という方法が考えられる．しかしこの方法では，終了させるための CPU 使用率を決定するのが困難である．負荷を適切に減少させるためには小さい値がよいが，小さすぎると今度は逆のルールを移譲された NIDS が過負荷に陥ってしまう．そこで，過負荷の NIDS の負荷を減少させるというよりむしろ，負荷分散を目指す手法を用いた．すなわち，Brownie は自分が管理している NIDS の CPU 使用率と子 NIDS の CPU 使用率との間にある程度の差があれば，ルールを移譲し，CPU 使用率にそれほど差がなくなったらルール移譲を終了する．NIDS 間の許容 CPU 使用率差をパラメータ *Diff* として定め，以下の実験では 5 と設定した．

最後に，どのルールをどれだけ移譲するかである．効率よくルールを移譲できれば，それだけ早く過負荷 NIDS の負荷が減る．現在は，移譲するルールの数だけを考慮し，CPU 使用率の差に基づいて，移譲するルールの数を決めている．負荷の差が大きいときはより多くのルールを移譲することで，早く負荷を分散させる狙いである．もちろん上記で述べたように 1 つのルールが与える負荷はさまざまな要因によって決まるが，有効にされているルールが増えれば，負荷が増える可能性は高いと考えられるためである．移譲されるルールの数は，*Factor* と CPU 使用率の差との積とした．ここで，*Factor* は設定パラメータであり，実験では 10 とした．移譲するルールは，ランダムに選ぶようにした．

Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22

以上をまとめると，過負荷 NIDS の負荷を下げる手順は以下の通りである．まず，Brownie は，自分が管理する NIDS と子 NIDS の CPU 使用率を収集する．これをそれぞれ，c_{my}, c_i $(0 \leq i < n, n$ は子 NIDS の数) とする．もし c_{my} が $max(c_i)$ より大きく，その差が *Diff* より大きい場合，Brownie は子 NIDS にルールを移譲する．もし c_{my} が $min(c_i)$ より小さく，その差が *Diff* より大きい場合，Brownie は子 NIDS からルールを移譲する．これを定期的に繰り返す．実験では，30 秒ごとに行った．

4.2 冗長なルールの削除手順

冗長なルールの削除はシンプルである．Brownie は，自分が管理している NIDS と下流の NIDS の両方で有効にされているルールがあると，下流の NIDS でそのルールを無効にする．これによって，冗長なルールは全て自分が管理している NIDS でのみ有効となる．冗長なルールを下流の NIDS で無効にすることにより，セキュリティが低下することがないのは，3.2 節で述べたとおりである．インターネットからのトラフィックは，ルールを無効にした下流 NIDS に届く前に，ルールを有効にしてある上流 NIDS によってチェックされている．

この方法では，全ての冗長なルールが上流 NIDS でのみ有効となるが，上流 NIDS が突然過負荷になるということは考えにくい．なぜなら Brownie が動作していない場合でも，これらのルールは上流 NIDS で有効にされているからである．それでももし上流 NIDS の負荷が高くなった場合には，4.1 節で述べた手順にしたがって，上流 NIDS の負荷を下流 NIDS に移譲すればよい．下流 NIDS では，冗長なルールが無効とされたために負荷が軽減されている可能性が高く，上流 NIDS の負荷を引き受けやすくなっていると考えられる．

5. 実　　験

5.1 ベンチマークによる実験

5.1.1 実験環境および実験方法

提案手法を用いて NIDS が協調することにより，性能が向上することを示すために，まず最初に web サーバのベンチマークを用いた実験を行った．実験は，NIDS 計算機 3 台，クライアント計算機 2 台，サーバ計算機 2 台の計 7 台を使用して，図 2 に示すようなネットワークを構築して行った．計算機同士は，1Gbps イーサネットで接続されている．Dual-Core Intel Xeon 2.33GHz CPU 2 個，メモリ 2GB，250GB 7200rpm のディスクで構成された計算機を上流 NIDS に用い，他は全て Pentium 4 2.8GHz，メモリ 512MB，36GB 7200rpm のディスクで構成された計算機を用いた．これは，下流 NIDS より上流 NIDS の方が性能がよい構成となっている．ただし，上流 NIDS 計算機の CPU は，1 つのコアのみを有効にした．オペレーティングシステムには Fedora 8 (Linux 2.6.24)，web サーバには Apache2.2.8 を使用した．NIDS としては，Snort 2.8.0.1[17] を inline モードで用い，ルールセットは 2008 年 1 月 28 日現在のものをデフォルトのまま用いた．ワークロードには WebStone2.5 を各クライアント上で実行し，それぞれ同時接続数は 10 とした．Apache，WebStone，Snort の設定はデフォルトのままとした．

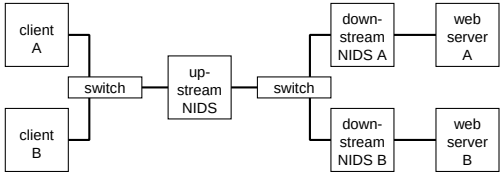


図 2 実験環境

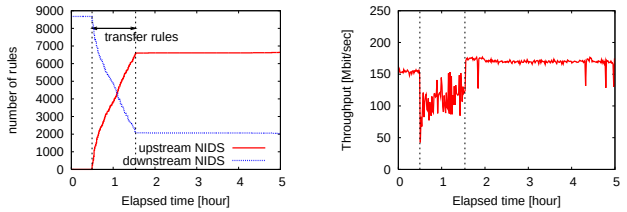
3 つの NIDS の初期ルール設定として，2 種類用意した．1 つめは，下流 NIDS のルール設定をともにデフォルト (ルール数 8676)，上流 NIDS のルール設定を全て無効 (ルール数 0) にしたものである (以降，DOWN と表す)．この初期設定では，下流 NIDS が過負荷になるため，過負荷 NIDS の負荷を減少させる手法の効果を知ることができる．2 つめは，全て (上流・下流とも) の NIDS のルール設定をデフォルトにしたものある (以降，BOTH と表す)．この初期設定では，全てのルールが上流と下流で重複して有効になっているため，冗長なルールの削除の効果を知ることができる．それぞれの実験において，各 NIDS での有効ルール数，CPU 使用率，ベンチマークのスループットを 10 秒ごとに計測した．初期設定による性能を計測するため，実験開始から 30 分後に Brownie が動作を開始するようにした．

5.1.2 実 験 結 果

図 3 と 4 に，それぞれ初期設定が DOWN と BOTH の時の実験結果を示す．それぞれ，Brownie が動作を開始した時刻と，調整がほぼ終了した時刻を示す垂線を引いた．

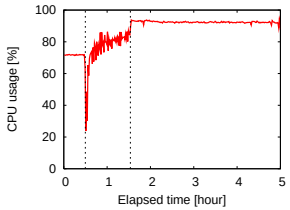
初期設定 DOWN：図 3(a) は各 NIDS における有効ルール数の変化を示す．実験開始 30 分後に Brownie が動作を開始し，ルールの移譲が始まる．上流 NIDS の有効ルール数が徐々に増加し，逆に下流 NIDS の有効ルール数は徐々に減少する．約 1 時間後に各 NIDS の負荷が同じ程度になり，ルールの移譲が停止する．このときの有効ルール数は，上流 NIDS で 6600，下流 NIDS で 2076 であった．下流 NIDS のマシン性能が上流 NIDS より低いため，同程度の負荷での有効ルール数は上流 NIDS の方が多いという結果になった．

Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22

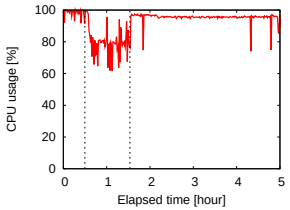


(a) ルール数

(b) スループット

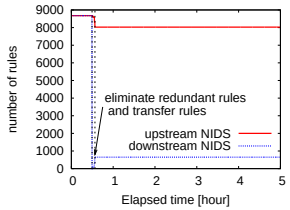


(c) 上流 NIDS の CPU 使用率

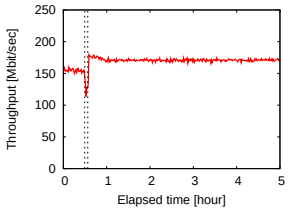


(d) 下流 NIDS の CPU 使用率

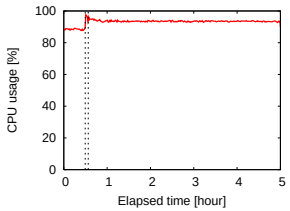
図 3 ベンチマークによる実験結果: DOWN



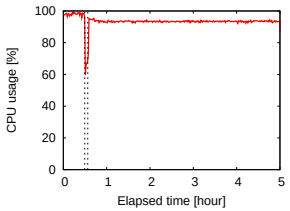
(a) ルール数



(b) スループット



(c) 上流 NIDS の CPU 使用率



(d) 下流 NIDS の CPU 使用率

図 4 ベンチマークによる実験結果: BOTH

図 3(b) はベンチマークのスループットの変化である．現在の実装では，ルール設定変更のために NIDS は再起動を行う必要があるため，ルール移譲中は一時的にスループットが落ちるところがある．しかし，ほぼ終了して落ち着くと，スループットは 174 Mbit/sec と，調整前の 154 Mbit/sec と比べて 13%程度増加した．

これは，Brownie によって過負荷になっている NIDS がなくなり，3 台の NIDS に均等に負荷がかかるようになったためである．図 3(c) と (d) にそれぞれ上流と下流 NIDS の CPU 使用率を示す．初期設定では，下流 NIDS がほぼずっと 100%と過負荷になっているのに対し，上流 NIDS は 80%に満たない程度であった．しかし，移譲をしていくにつれて上流 NIDS の CPU 使用率は徐々に増加していき，調整終了後は上流下流 NIDS ともに 100%を少し下回る程度で均等に負荷がかかっている状態となった．

初期設定 BOTH:図 4(a) は初期設定が BOTH の時の有効ルール数の変化である．Brownie が動作を開始すると，まずすぐに冗長なルール設定の削除を行う．初期設定では全ての NIDS で同じルールを有効にしているため，下流 NIDS では全てのルールが無効になり，上流 NIDS でのみ有効になる．その後，負荷を均等にするためにいくつかのルールが下流 NIDS に移譲され，4 分ほどで完了する．ベンチマークのスループットは，初期設定の 155 Mbit/sec から 173 Mbit/sec に，12%増加した．これは，初期設定では全てのルールに対して 2 度チェックされる必要があったからである．また，初期設定では下流 NIDS が過負荷になっていたことも要因の 1 つであると考えられる．図 4(d) で示すように，最初の 30 分間の下流 NIDS の CPU 使用率は，ほぼ 100%に達している．

5.2 パケットトレースによる実験

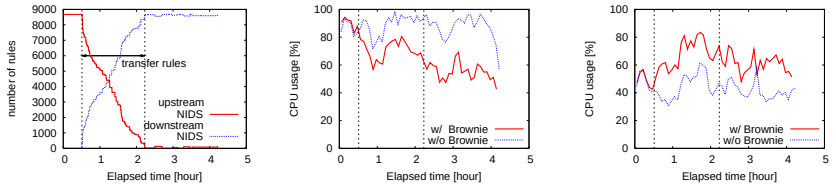
5.2.1 実験環境および実験方法

現実のネットワークトラフィックでの Brownie の効果を示すため，ワークロードとしてパケットトレースを用いた実験を行った．用いたパケットトレースは，豊橋技術科学大学で 2008 年 3 月 23 日から 4 日間かけて取得したフルパケットトレースで，全部で 673GB ある．

実験は，NIDS 計算機 3 台と，パケットを送信および受信する計算機 1 台の計 4 台を用いて行った。NIDS 同士は図 2 と同様に接続されており，全ての NIDS がパケットを送受信する計算機に接続している．ネットワークは，1Gbps イーサネットである．上流 NIDS の構成は Intel Quad-Core Xeon 2.33GHz CPU，メモリ 4GB，下流 NIDS の構成は Intel Pentium Dual-Core 2GHz CPU，メモリ 1GB，パケットを送受信する計算機の構成は Intel Core2 Duo 2.4GHz CPU，メモリ 2GB である．ソフトウェア設定は 5.1.1 節と同様である．

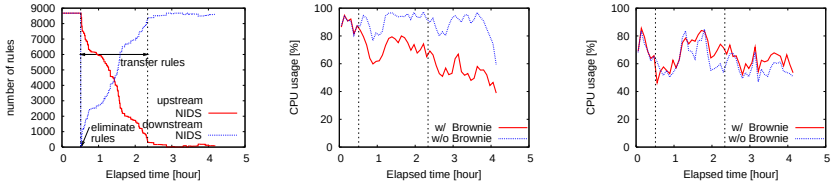
初期ルール設定は，2 種類用意した．1 つめは，下流 NIDS のルール設定をともに全て無

Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22



(a) ルール数　(c) 上流 NIDS の CPU 使用率　(c) 下流 NIDS の CPU 使用率

図 5 パケットトレースによる実験結果: UP



(a) ルール数　(b) 上流 NIDS の CPU 使用率　(c) 下流 NIDS の CPU 使用率

図 6 パケットトレースによる実験結果: BOTH

効 (ルール数 0)，上流 NIDS のルール設定をデフォルト (ルール数 8676) としたものである (以降，UP と表す)．2 つめは，全て (上流・下流とも) の NIDS のルール設定をデフォルトにしたものある (以降，BOTH と表す)．各 NIDS に十分な負荷を与えるため，パケットは最速で再送した．また，ワークロードがパケットトレースであり，スループットや通信遅延は計測できないため，各 NIDS における有効ルール数と CPU 使用率のみ計測した．

5.2.2 実 験 結 果

図 5 と 6 に，それぞれ初期設定が UP と BOTH の時の実験結果を示す．

初期設定 UP：図 5(a) は各 NIDS における有効ルール数の変化を示す．Brownie の動作開始後，下流 NIDS のルールは増加し，逆に上流 NIDS のルールは減少した．図 5(b) と (c) に上流 NIDS と下流 NIDS それぞれにおける，5 分ごとの平均 CPU 使用率の変化を示す．比較のため，Brownie を動作させず初期設定のままの時の CPU 使用率を点線で示す．Brownie がない場合，全てのルールが上流 NIDS にあるため，上流 NIDS の CPU 使用率は 100%近くになっている．Brownie を動作させることで，ほとんどのルールは下流 NIDS に移譲され，全ての NIDS がほぼ同じ負荷になる．

初期設定 BOTH：図 6(a) に示すように，まず最初に下流の全てのルールが無効にされる．この状態は，初期設定 UP と同じ設定である．そのため，その後初期設定 UP の場合と同じように，上流 NIDS から下流 NIDS にルールが移譲される．Brownie が動作しない場合，上流 NIDS の CPU 使用率は 100%近くになることがある．Brownie を動作させることで，ルールを移譲するに従って，上流 NIDS の CPU 使用率が減少する．

6. 応　用　例

本論文では，Brownie で負荷分散を行うことによる性能向上を提案したが，ネットワーク内にある NIDS が相互に協調することは，他にも様々に応用例があると考えている．

6.1 NIDS の障害検知と回避

Brownie は，通信をしている他の Brownie が管理している NIDS の障害検知や障害隠蔽ができると考えられる．Brownie では，上流や下流の Brownie と定期的にやりとりをしているため，通信をしている NIDS に障害が起こったことを検知できる．また，障害が起こった NIDS でどのルールが有効になっていたかを知っているため，代わりにそのルールを有効にすることで，トラフィックをチェックし攻撃を検知することができる．

6.2 シグネチャの同期

シグネチャデータベースの更新やシグネチャの自動生成[⋆1]によって新たなシグネチャを導入した NIDS は，Brownie を通して他の NIDS に対して新たなシグネチャがあることを通知することができる．この機能が Brownie に導入されれば，ネットワーク内のある NIDS が更新されれば，全ての NIDS が更新されることになり，NIDS のシグネチャデータベースの管理コストの大幅な削減が期待できる．

7. ま　と　め

ネットワークの高速化や攻撃の巧妙化に伴い，単体の NIDS での監視は難しくなっている．本論文では，組織内に複数配置された NIDS を相互に協調させることで性能向上することを提案した．ネットワークの経路上にある NIDS 同士がマシンの負荷やルール設定をやりとりし，過負荷になっている NIDS から負荷が低い NIDS に処理を移したり，冗長なルール設定をなくしたりすることによって，その組織内全体の性能の向上が見込める．実験では，過負荷の NIDS から負荷が低い NIDS にルールを移すことで，スループットが向上することを示した．

⋆1 シグネチャの自動生成は数多く提案されている．[19)–22)]

Vol.2009-ARC-183 No.4
Vol.2009-OS-111 No.4
2009/4/22

今後は，負荷分散のための手法を改良したいと考えている．ネットワークトラフィックや各ルールを分析することで，負荷の高いルールを予想するといったことが考えられる．Dregerらによる負荷予想モデル[23]を用いることができれば，ルールの振り分けを即座に決定できる可能性もある．

参 考 文 献

1) Paxson, V., Asanović, K., Dharmapurikar, S., Lockwood, J., Pang, R., Sommer, R. and Weaver, N.: Rethinking Hardware Support for Network Analysis and Intrusion Prevention, *Proc. of the 1st USENIX Workshop on Hot Topics in Security (HotSec '06)*, pp.63–68 (2006).
2) Vallentin, M., Sommer, R., Lee, J., Leres, G., Paxson, V. and Tierney, B.: The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware, *Proc. of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID '07)*, pp.107–126 (2007).
3) Xinidis, K., Charitakis, I., Antonatos, S., Anagnostakis, K.G. and Markatos, E.P.: An Active Splitter Architecture for Intrusion Detection and Prevention, *IEEE Transactions on Dependable and Secure Computing*, Vol.3, No.1, pp.31–44 (2006).
4) Kruegel, C., Valeur, F., Vigna, G. and Kemmerer, R.: Stateful Intrusion Detection for High-Speed Networks, *Proc. of the 2002 IEEE Symposium on Security and Privacy (S&P '02)*, pp.285–293 (2002).
5) Aho, A.V. and Corasick, M.J.: Efficient String Matching: An Aid to Bibliographic Search, *Communications of the ACM*, Vol.18, No.6, pp.333–340 (1975).
6) Wu, S. and Manber, U.: A Fast Algorithm for Multi-pattern Searching, Technical report, TR-94-17 (1994).
7) Commentz-Walter, B.: A String Matching Algorithm Fast on the Average, *Proc. of the 6th Colloquium on Automata, Languages and Programming*, pp.118–132 (1979).
8) Clark, C., Lee, W., Schimmel, D., Contis, D., Koné, M. and Thomas, A.: A Hardware Platform for Network Intrusion Detection and Prevention, *Proc. of the 3rd Workshop on Network Processors & Applications (NP3)* (2004).
9) Bos, H. and Huang, K.: Towards Software-Based Signature Detection for Intrusion Prevention on the Network Card, *Proc. of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID '05)*, pp.102–123 (2005).
10) de Bruijn, W., Slowinska, A., van Reeuwijk, K., Hruby, T., Xu, L. and Bos, H.: SafeCard: A Gigabit IPS on the Network Card, *Proc. of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID '06)*, pp.311–330 (2006).
11) Baker, Z.K. and Prasanna, V.K.: Time and Area Efficient Pattern Matching on FPGAs, *Proc. of the 12th International Symposium on Field Programmable Gate Arrays (FPGA '04)*, pp.223–232 (2004).
12) Dharmapurikar, S., Krishnamurthy, P., Sproull, T.S. and Lockwood, J.W.: Deep Packet Inspection using Parallel Bloom Filters, *IEEE Micro*, Vol.24, No.1, pp.52–61 (2004).
13) Song, H., Sproull, T., Attig, M. and Lockwood, J.: Snort Offloader: A Reconfigurable Hardware NIDS Filter, *Proc. of the 15th International Conference on Field Programmable Logic and Applications (FPL '05)*, pp.493–498 (2005).
14) Gonzalez, J.M., Paxson, V. and Weaver, N.: Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention, *Proc. of the 14th Conference on Computer and Communications Security (CCS '07)*, pp. 139–149 (2007).
15) Paxson, V., Sommer, R. and Weaver, N.: An Architecture for Exploiting Multi-Core Processors to Parallelize Network Intrusion Prevention, *Proc. of the 2007 IEEE Sarnoff Symposium* (2007).
16) Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E.P. and Ioannidis, S.: Gnort: High Performance Network Intrusion Detection Using Graphics Processors, *Proc. of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID '08)*, pp.116–134 (2008).
17) Roesch, M.: Snort - Lightweight Intrusion Detection for Networks, *Proc. of the 13th USENIX Systems Administration Conference (LISA '99)*, pp.229–238 (1999).
18) Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time, *Computer Networks*, Vol.31, No.23–24, pp.2435–2463 (1999).
19) Kreibich, C. and Crowcroft, J.: Honeycomb – Creating Intrusion Detection Signatures Using Honeypots, *Proc. of the 2nd Workshop on Hot Topics in Networks (HotNets-II)* (2003).
20) Kim, H.-A. and Karp, B.: Autograph: Toward Automated, Distributed Worm Signature Detection, *Proc. of the 13th USENIX Security Symposium*, pp.271–286 (2004).
21) Newsome, J., Karp, B. and Song, D.: Polygraph: Automatically Generating Signatures for Polymorphic Worms, *Proc. of the 2005 Symposium on Security and Privacy (S&P '05)*, pp.226–241 (2005).
22) Wang, X., Li, Z., Xu, J., Reiter, M.K., Kil, C. and Choi, J.Y.: Packet Vaccine: Black-box Exploit Detection and Signature Generation, *Proc. of the 13th Conference on Computer and Communications Security (CCS '06)*, pp.37–46 (2006).
23) Dreger, H., Feldmann, A., Paxson, V. and Sommer, R.: Predicting the Resource Consumption of Network Intrusion Detection Systems, *Proc. of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID '08)*, pp. 135–154 (2008).