2005－OS －98（8）
2005／2／23

Network IDS の攻撃検知情報を利用したサーバの安全性向上

嶋村　誠　　河野 健二

電気通信大学情報工学科
電子メール：tima@zeus.cs.uec.ac.jp，kono@cs.uec.ac.jp

要旨

インターネットにおいて不正攻撃を検知するために，ネットワーク侵入検知システム (NIDS) が広く用いられている．しかし，NIDS は攻撃の誤検知が多いため，攻撃を検知してもそのメッセージを破棄せずに警告を出すのみであることが多い．これは誤検知に起因したサービスの拒否を回避するためである．そこで，本論文では，不正攻撃からの防御を行いつつ，NIDS が誤検知を起こしたとしてもサービス拒否を起こさないようにするため，NIDS が検出した攻撃に応じ，サンドボックスを用いて攻撃対象となるサーバのアクセス権を限定する手法を提案する．本論文では本手法のプロトタイプを作成し，評価を行った．その結果，誤検知によるサービス拒否を回避しつつサーバの防御を行うことができた．また，サンドボックスのポリシー作成者の負担を軽減できた．NIDS とサンドボックスの連携処理に要する時間は 135 マイクロ秒程度であった．

Using attack information in Network-IDS to build secure servers

Makoto Shimamura　　Kenji Kono

Department of Computer Science, University of Electro-Communications,
E-mail: tima@zeus.cs.uec.ac.ac.jp, kono@cs.uec.ac.jp

Abstract

Network intrusion detection systems (NIDS) have been widely used for detecting malicious attacks from the Internet. Current NIDS only alerts even if it detects a malicious message because most of the alerts are false-positive. If NIDS drops the false-positive messages, the legal access to the service is rejected; i.e., denial-of-service. In this paper, we propose a technique that enables us to defend an attack without causing denial-of-service. In the proposed method, NIDS cooperates with a sandbox system running on the server side. When NIDS detects an attack message, it sends the sandbox system how to prevent the attack from succeeding. The experimental results demonstrate that the proposed system can defend attacks and the overhead incurred by the cooperation is reasonable.

1 はじめに

インターネット上のサーバは常に不正攻撃の脅威にさらされている．サーバの脆弱性を突き，悪意のあるコードを実行し，サーバを乗っ取る攻撃がしばしば報告されており，サーバのセキュリティを向上することが重要な課題になっている．

サーバに対する不正攻撃を検知する機構としてネットワーク侵入検知システム (以下 NIDS) が広く用いられている．しかし，NIDS は一般に誤検知が多い．SecurityFocus[1] によれば，NIDS の導入初期に発生する警告の 90%が誤検知である．このため，NIDS が検知した攻撃メッセージを廃棄し，攻撃を直接的に防御することは難しい．これは誤検知が発生したときに正しいメッセージを廃棄し，サービスを拒否してしまうからである．そのため，通常の NIDS では攻撃を検知しても警告を発するのみである．

本論文では NIDS とサンドボックスとを連携させ，NIDS が検知した攻撃を防御できる仕組みを提案する．サンドボックスとは，その監視下で動作するプロセスのアクセス権限を限定する機構であり，あらかじめ定められたセキュリティー・ポリシー（以下ポリシー）に従ってアクセスできる計算機資源を限定する．たとえば，ルート権限で動作するメール・サーバに対し，`/etc/passwd` の参照を禁止することができる．

本論文で提案する手法では，NIDS が検知した攻撃が成功しないよう，検知した攻撃に応じてアクセス制限を行う．具体的には，1) 保護対象となるサーバをサンドボックスの監視下で動作させる，2) 攻撃を検知した NIDS は，その詳細をサンドボックスに通知する，3) サンドボックスはその通知に従ってサーバのアクセス権を限定する．たとえば，`/bin/sh` を実行してしまうという脆弱性を持つ FTP サーバ [2] を動作させているとしよう．`/bin/sh` の実行を狙った攻撃メッセージを検知すると，NIDS はサンドボックスに「`/bin/sh` の実行を禁止」という通知を行う．この通知を受けたサンドボックスは`/bin/sh` の実行を禁止する．

本手法を用いることにより得られる利点は以下の通りである．

攻撃の防御ができる NIDS がバッファ溢れ攻撃などを検知すると，攻撃による被害を受けないようにサンドボックスが適切な防御を行う．そのため，攻撃による被害を防止することができる．

誤検知によるサービス拒否がない NIDS が攻撃と判断したメッセージであっても，そのメッセージを廃棄せず必ずサーバで受信する．そのため，誤検知が起こったとしてもサービス拒否は発生しない．

ポリシー作成の負担が軽減される 通常のサンドボックスでは，監視下にあるプロセスに不要な権限は全て禁ずるという運用方針をとることが多い．そのため，ポリシーの定義が困難になる傾向にあった．本論文で示す手法は，NIDS の持つ攻撃データベースを利用すれば，そのようなポリシーを作成しなくとも十分な防御が行えることを示唆している．

提案手法の有効性を検証するため，Linux 上でのプロトタイプシステムの作成を行った．NIDS には代表的な NIDS である Snort[3] を用い，サンドボックスには古典的なサンドボックスである Janus[4] と同等の機能を持ったものを用いた．Linux 上でこのプロトタイプを動作させ，評価を行った．その結果，誤検知によるサービス拒否を発生させずに攻撃の防御に成功したことを確認した．NIDS とサンドボックスの連携処理のオーバーヘッドは連携処理 1 回あたり 135 マイクロ秒程度であった．

以下，2 章では NIDS とサンドボックスの動作と問題点について示す．3 章では NIDS の攻撃検知情報の利用について示し，4 章で本論文で提案する手法の設計と実装について示す．5 章では実際に作成したプロトタイプの評価を示し，6 章で関連研究についてまとめ，7 章で本論文をまとめ，将来の課題を示す．

2 Network IDS とサンドボックス

2.1 NIDS

NIDS とは通信メッセージを監視し，ある一定のパターンと一致するメッセージを検出すると警告を出す機構である．NIDS の概念図を図 1 に示す．図 1 の内容について説明する．.htaccess を含むメッセージを攻撃メッセージとするという情報が NIDS に設定されている．このとき，正常なメッセージである “GET index.html” は NIDS を通過し，サーバに到達する．攻撃メッセージである “GET .htaccess” は

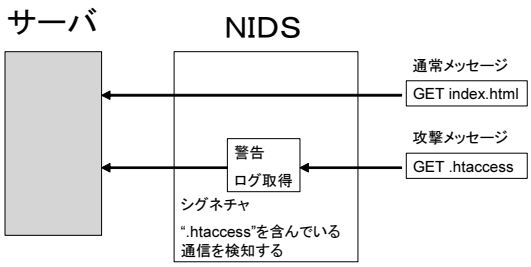

図 1: NIDS の動作

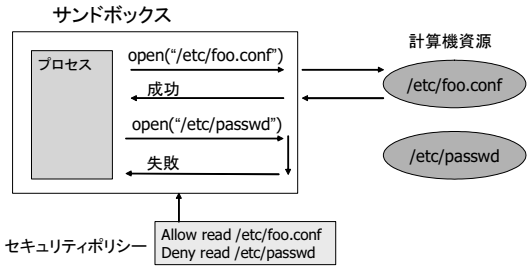

図 2: サンドボックスの動作

NIDS によって検知され，管理者に警告が出されたり，攻撃のログが取得される．この情報のことをシグネチャと呼び，この手法をシグネチャ・マッチング方式という．

シグネチャ・マッチング方式は誤検知が多い．例えば，シグネチャ・マッチング方式を用いている NIDS の代表例として Snort がある．Snort の誤検知の例として，Snort にはバージョン管理システムである CVS からの情報収集を試みるメッセージを検出するためのシグネチャが存在する．このシグネチャは，ポート 2401 で行われる通信に対して，文字列 “E Fatal error, aborting.” と “no such user” が通信内容に含まれていたときに検出を行っている．つまり，ユーザ名を間違ったときに発生する CVS のエラーメッセージを検知している．しかし，このシグネチャはユーザー名の単純な入力間違いによる誤検知を発生する．

従って，NIDS は攻撃を検知しても警告を出すだけにとどめる事が多く，攻撃そのものを防止することは出来ない．

NIDS にはシグネチャに一致したメッセージを廃棄し攻撃そのものを防御するものもある．しかしメッセージ廃棄を行う NIDS では，誤検知によってメッセージを廃棄するとサービス拒否が発生してしまう．そこで誤検知がない，確実に攻撃と認識できるメッセージのみを廃棄している．例えば，Snort においては，メッセージ廃棄を行うよう設定されたシグネチャは，誤検知を避けるため，攻撃コードそのものを検出するように設定されている．つまり，この手法はごく限られた場合においてのみ有効である．

本論文で提案する手法では，メッセージ廃棄を行わず，サンドボックスを用いてサーバを防御するため，誤検知によるサービス拒否を起こすことなく，サーバを防御する．

2.2 サンドボックス

サンドボックスはポリシーに応じて計算機資源へのアクセスの制御を行う．サンドボックスの概念図を図 2 に示す．図 2 に示したポリシーでは，`/etc/foo.conf` へのアクセスを許可する一方で，`/etc/passwd` へのアクセスを拒否するよう設定している．そのため`/etc/passwd` に対して open() をすると，その呼び出しはサンドボックスによって拒否される．サンドボックスでは，ポリシーにはどの計算機資源へのアクセスを許可するか，拒否するかが記述されている．

通常のサンドボックスでは，監視下にあるプロセスに不要な権限はすべて禁ずるという運用ポリシーをとることが多い．そこで，ポリシーを作成するに当たって，ポリシーの作成者は監視下にあるプロセスに必要な全ての計算機資源を適切に把握する必要がある．しかし，現在のプログラムはさまざまなライブラリをリンクして実行されるようになっている．例えば，ライブラリ関数 gethostbyname() を呼んだ際には，様々な設定ファイルを読み込み，必要に応じて DNS との通信を行う．このようにライブラリ関数一つだけでも多くの計算機資源へのアクセスが行われているので，あるプログラムがどのような計算機資源が必要かを把握することは，プログラムの作者でさえ難しい．

本論文で示す手法は，NIDS の持つ攻撃データベースを利用することにより，そのようなポリシーを作成しなくとも十分な防御が行える．

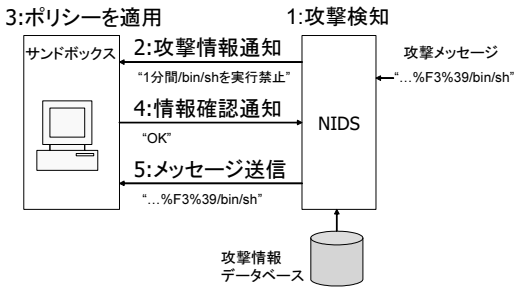

図 3: 提案機構の動作

3 NIDS の攻撃検知情報の利用

3.1 システムの動作概要

本論文で提案するシステムの動作の概要を図 3 に示す．まず，外から来た全てのメッセージは NIDS で一旦攻撃か否かの検査が行われる．攻撃を検知しなければメッセージはそのままサーバに渡され，サーバで処理が行われる．

NIDS とサンドボックスを通信するためのプロトコルを作成した．その概要を以下に示す．

1. NIDS は攻撃検知時に “攻撃データベースから取得した情報” を送信する．
2. サンドボックスは NIDS から受信し，防御ポリシー適用処理後に確認応答 “OK” を NIDS に送信する．
3. NIDS は確認応答を受信後，検知したメッセージを通過させる．

この通信プロトコルにより，サンドボックスに防御ポリシーが適用された後にサーバが検知メッセージを処理することを保証する．

NIDS が確認応答を受信した後で検知メッセージをサーバに渡し，サーバで処理が行われる．また，サンドボックスは設定された攻撃の持続時間が経過した後に攻撃検知時に適用したポリシーを廃棄する．

これによって，誤検知時は正しく処理が行われ，攻撃の時は攻撃が成功しないよう的確な防御が行われる．

3.2 NIDS の持つ情報

NIDS がサンドボックスに送信すべき攻撃検知情報とは，攻撃に対する具体的な防御方法と攻撃の持続時間である．攻撃に対する具体的な防御方法とは，例えば「`/bin/sh` の実行を禁止する」等の記述である．さらに，攻撃の防御方法は複数あることも想定される．例えば`/bin/sh` の実行を意図する攻撃の亜種である`/bin/bash`，`/bin/tcsh` 等を実行する攻撃に対応するよう，`/bin/sh`, `/bin/bash`, `/bin/tcsh` 等の実行を禁止するように記述することが考えられる．本論文で作成したプロトタイプでは，複数の防御方法の記述が可能である．

攻撃の持続時間とは，防御の対象となる攻撃がどの程度長く続くかという記述である．これは攻撃を受けている間だけ防御方法をサンドボックスのポリシーとして適用しておくことにより，攻撃が終わった後にも不必要なポリシー検索を行うような，無駄な負荷を発生させないようにするためである．本論文で作成したプロトタイプでは，この持続時間が過ぎたポリシーは廃棄される．

例えば，wu-ftpd の SITE-EXEC 脆弱性を狙った攻撃を検知した際には，Snort のシグネチャ情報を記述したドキュメントによると，防御方法として `/bin/sh` の実行を禁止すればよいということが分かる．また，このドキュメントには 1 回あたりの攻撃の持続時間については記述が存在しなかったが，バッファ溢れ攻撃はそれほど時間がかからないと考えることにより，暫定的にポリシーの有効期間は 1 分間であるという情報を設定できる．

本論文では NIDS が上の情報の組から成るデータベースを持っていることを想定している．こうしたデータベースを用意することは，本質的には困難ではない．なぜなら，NIDS のシグネチャ作成者は攻撃手法についての詳細な知識を持っており，その攻撃が成功した場合，どのような不正アクセスが起こりうるかを把握しているからである．

3.3 提案方式の利点

本手法の利点を 3 つ示す．第一に，バッファ溢れ攻撃や printf フォーマット攻撃などの NIDS で検知可能な攻撃が防御できることである．これは，NIDS で検知した攻撃の情報をサンドボックスに送信し，サンドボックスがその情報を用いてポリシー適用を行うことで可能である．

第二に，NIDS の誤検知によるサービス拒否が発生しないことである．これは，NIDS が攻撃と判断したメッセージであってもサーバに送信するため，

NIDS が誤検知を行ってもサービスが実行されないことはない．そして，そのメッセージが攻撃目的の不正なメッセージであった場合には防御が行われる．

第三に，サンドボックスのポリシー作成者の負担が減ることである．本手法では，不正攻撃を防御する目的のポリシーを記述しないようにできる．これでサンドボックスのポリシー作成者の負担を減らすことができる．

4 システムの実装

4.1 NIDS

本論文で提案したシステムのプロトタイプの実装を行った．NIDS 側は検知を行ったときにサンドボックスと通信を行うよう Snort の機能拡張を行った．使用する攻撃情報データベースは “Snort のシグネチャID” と “攻撃有効時間” と “サンドボックスに適用するポリシーを記述したもの” を対応付けたものである．例えば,「345 60 x:/bin/sh;x:/bin/tcsh」は「Snort シグネチャID345 の攻撃を，60 秒間`/bin/sh`と`/bin/tcsh` の実行を禁止することによって防御する」という意味になっている．このようなエントリを並べたデータベースを作成した．この例では x:/bin/sh が防御方法であり，x が実行禁止を意味する．そして，複数の防御方法を; で繋げることによって扱うことが出来る．

制御方法の指定として，本プロトタイプでは x 以外に，r(読み込み禁止)，w(書き込み禁止)，d(指定されたネットワークへのアクセスの禁止)，s(指定されたシステムコールの実行禁止) を用いることが出来る．同一のファイルに対するポリシーは一度に書くことも可能であり，rw:/etc/passwd とすると`/etc/passwd` の読み込みと書き込みを禁止する．

本手法は Snort の出力プラグインとして実現している．Snort のプラグインは初期化時に実行する手続き，検知時に出力を実行する手続き，終了時に実行する手続きを API でコールバックとして登録して実現した．そこで，初期化時には攻撃情報データベースを読み込み，サンドボックスが動いているサーバの IP アドレスを登録する手続きを作成した．そして，検知時にサンドボックスに対して攻撃情報を送る手続きを作成し，プラグインを作成した．

4.2 サンドボックス

今回の実装では，Linux 上で動作するプロセスを ptrace() システムコールを用いて制御するサンドボックスを作成した．このサンドボックスは Janus と同等の能力を持ち，ファイルの読み込み，書き込み，実行，システムコールの実行，ネットワークアクセスを制御できる．初期に設定されたポリシーに従って，サンドボックスの対象となるプロセスのシステムコールを制御している．

このサンドボックスを 1 つのスレッドとして動かして，NIDS からの通信を受け付けるスレッドを動かすように機能拡張を施した．NIDS からの通信を受け付けるスレッドは，NIDS から通信を受けたときにサンドボックスのポリシーを書き換えるようにした．また，常にサンドボックスが適用しているポリシーの監視を行い，設定された時間が経過すると，NIDS から追加されたポリシーは自動的に廃棄されるようになっている．

5 評価

5.1 定性的評価

Snort のシグネチャのドキュメントから得た，本手法で正しく防御できると考えられる攻撃の例を表 1 に示す．主に，バッファ溢れ攻撃や情報漏洩を狙った攻撃を例として挙げた．こういった攻撃は報告数も非常に多く，本手法は十分に有用であると考えられる．

実装したプロトタイプの実際の動作の例を示す．まず予備実験として，Redhat7 上において wu-ftpd 2.6.0 の脆弱性に対し攻撃を行った．攻撃方法については，この脆弱性に対応するバッファ溢れ攻撃を行うスクリプトを入手することができたので，それを動作させた．

まず，本手法なしでは攻撃が成功し，`/bin/sh` が動作してしまうことを確認した．次に本手法を適用し実験を行ったところ，システムの動作の結果，攻撃スクリプトから「攻撃失敗」を意味するメッセージが出力された．

この動作例から，検知時にメッセージはサーバまで届いて処理は行われることが確認された．これで，NIDS が誤検知を起こしていたとしても，メッセージは正当に処理され，クライアントに正しくサービ

表 1: 防御できる攻撃とその防御法の例

対象	攻撃の概要	被害	防御方法
Samba	バッファオーバーフロー	Samba 権限でシェルが起動	/bin/sh の実行禁止
ntpd	バッファオーバーフロー	NTP 権限でシェルが起動	/bin/sh の実行禁止
ISC BIND	TSIG buffer overflow	BIND 権限でシェルが起動	/bin/sh の実行禁止
wu-ftpd	SITE EXEC format string overflow	root 権限でシェルが起動	/bin/sh の実行禁止
Sendmail	パースエラーを利用した侵入	root 権限でシェルが起動	/bin/sh の実行禁止
Telnetd	外部からの root ログインが成功	root 権限でシェルが起動	/bin/sh の実行禁止
Telnetd	バッファオーバーフロー	root 権限でシェルが起動	/bin/sh の実行禁止
OpenSSH	CRC32 compensation attack	SSHd 権限でシェルが起動	/bin/sh の実行禁止
http サーバ	/usr/bin/cpp へのアクセス	サーバ上でバイナリ生成	/usr/bin/cpp の実行禁止
http サーバ	/cgi-bin/ls へのアクセス	コマンドの不正な実行	/cgi-bin/ls の実行禁止
Apache	バッファオーバーフロー	root 権限でシェルが起動	/bin/sh の実行禁止
Sendmail	バッファオーバーフロー	root シェルが起動	/bin/sh の実行禁止
ISC INN	バッファオーバーフロー	INN 権限でシェルが起動	/bin/sh の実行禁止

表 2: オーバーヘッドの測定

	本手法あり	本手法なし	低下率
実験 1	18.42MB/s	22.16MB/s	16.9%
実験 2	86.2KB/s	90.4KB/s	4.6%

スができる．よって，サービス拒否が起こり得ないことがわかる．

今回は，攻撃の結果バッファ溢れが起こったが，サンドボックスが/bin/sh の実行を拒否したため攻撃の被害はなかった．

本システムはサンドボックスの対象とするプログラムの挙動の詳細を知らなくても使うことができ，サーバの防御という役割を果たした．そのため，ポリシー作成者はセキュリティ関連のポリシーを考慮する必要が減少する．このことにより，ポリシー作成者の負担の軽減ができたと考えられる．

5.2 定量的評価

システムのオーバーヘッドを測定するために，ベンチマークとして，FTP で 200MB のファイルを転送する時間を計測した (実験 1)．次に，4KB のファイル 1000 個を転送する時間を計測した (実験 2)．FTP 通信は NIDS の監視下にあり，FTP サーバはサンドボックス上で動作している．また，比較のために本手法を全く用いないで，NIDS，サンドボックスを使わない状態での実験を行った．

サーバと NIDS は同一マシン上で動作している．サーバ機の OS は Linux 2.6.10-rc2，プロセッサは AMD AthlonMP 1800+ 2CPU，メモリ 512MB，NIDS として Snort-2.3.0RC1，FTP サーバとして proftpd-1.2.10 を用いた．クライアント機の OS は Linux 2.6.9，プロセッサ Intel Pentium4-3GHz，メモリ 1GB，FTP クライアントとして lftp-3.0.10 を用いた．サーバ機とクライアント機は 1000Base-TX のスイッチを介して接続した．

各実験を 3 回行って得られた結果の平均値を表 2 に示す．本システムを適用するオーバーヘッドにより転送率が低下している．本システムを利用しない場合の転送率を 100%として，低下の割合は実験 1 では 16.9%，実験 2 では 4.6%となった．実際の転送速度がネットワークデバイスの速度に比べて低いのは ftp サーバソフトの性能や，HDD の速度がネックになっているからと考えられる．実験 2 のオーバーヘッドは実験 1 に比べてかなり少ないが，これは FTP サーバの転送用子プロセスの生成時に fork() システムコールを使っていることによると考えられる．実験 2 では fork() の負荷が高いので，サンドボックスを使うことによる影響が少なくなっている．

次に，NIDS が攻撃情報を送信してから確認応答を受信するまでの時間を計測した今回の評価では同一マシン上で NIDS とサーバを動作させた．これは連携処理を計測するに当たって，物理的な通信時間による影響を小さくするためである処理時間は 12000 回の警告で 1.835 秒であった．よって処理 1 回は約 153 マイクロ秒であることがわかった．この連携処理は，オーバーヘッドとしてはそれほど大きくないと考えられる．

ベンチマークの結果から，NIDS とサンドボックスを連携する処理の負荷はそれほど大きくないと

考えられる．本論文の手法全体を見たオーバーヘッドはやや大きくなっているが，これは Linux 上において ptrace() システムコールを用いたサンドボックスを使っているためである．カーネルに組み込むなどの最適化を施したサンドボックスであれば，オーバーヘッドは減少すると考えられる．

5.3 本手法の制限

本手法には制限となる点が 2 つあげられる．第一に，http サーバの設定ファイルである.htaccess のようにソフトウェアが正当に使用するファイルに対する情報漏えいを狙った攻撃への対処は困難である．この攻撃は，ファイルへのアクセス禁止という形で防ぐことができないわけではない．しかし，アクセス禁止ポリシーを適用している間に正当な使用が発生した場合にはそれが失敗してしまう．これによって意図的にそのポリシーを適用させるようにするサービス拒否攻撃が可能になってしまう．また，誤検知によってサービスの拒否が発生してしまったりする．これは，サービス拒否を発生させないようにして防御を行うことを目的とする本論文の意図には沿わない．

第二に，サーバのバグをついた情報漏洩を狙う攻撃への対処は困難である．例えば，Apache のバグを利用してアクセスを行い，ディレクトリリストを流出させようとする攻撃 [5] がある．ディレクトリリストは他の用途で正当に使用する可能性があるため，単純に計算機資源へのアクセスを禁止して防御できるような攻撃ではない．これに対して本手法を応用して，NIDS から攻撃者の IP アドレスをサンドボックスに通知して当該 IP アドレスに対する処理をサーバで拒否することができるが，これは行うべきではない．それを行うと，IP を詐称することによるサービス拒否攻撃が可能になってしまったり，誤検知によるサービス拒否が発生してしまうので，本論文の意図には沿わない．

6 関連研究

6.1 NIDS

Snort，Bro[6]，Shield[7] など様々な NIDS がある．Snort はシンプルでオーバーヘッドの少ない NIDS を作成し，NIDS におけるコスト削減をすることを目的とした研究である．本論文は Snort を実装手段として用いている．本論文は NIDS そのものを改良することが目的ではなく，NIDS とサンドボックスの連携を行うことが目的である．

Bro はオーバーヘッドの少ない高速な NIDS を実現する研究である．PC 上で 1Gbps を超えるような高速ネットワークにも使える NIDS を実現している．本論文は NIDS の性能向上には全く関連していないが，NIDS における検知のオーバーヘッドが減ることは本システムの全体的な性能向上には有益であると考えられる．

Shield は，ネットワークスタックの中にシグネチャ・マッチングを実装し，パケット拒否による防御を行う研究である．Shield は，脆弱性を解消するパッチの当たっていない OS が多く稼動していることを問題視し，ネットワークスタック中に NIDS を実装することによって，最低限の防御が行えるようになっている．本論文の手法は，NIDS で防御を行うのではなく，サンドボックスと連携することによって攻撃からの防御を行う．

6.2 サンドボックス

サンドボックスの関連研究としては，Janus が代表的なものとして挙げられる．さらに，その他のサンドボックスとして，SoftwarePot[8] など様々な研究がなされている．

Janus は，Web から得た信頼できないデータを，信頼できないヘルパーアプリケーションで実行しなければならないということを問題としている．そこで，それらを安全に実行するという観点で，計算機資源へのアクセスを制御する研究である．

SoftwarePot は Janus と同様に，信頼できないソフトウェアを安全に実行するという観点から研究されている．Janus と異なる点は，独自のファイルシステムを含むサンドボックス（ポット）を作り，その中でプロセスの制御が可能であることである．これによって，既存のファイルシステムに依存しないでサンドボックスを構築することができる．

本論文では NIDS と連携して不正攻撃を防御するということを主眼においている．また，従来のサンドボックスの研究では，プロセスの制御をすることが主目的であり，前述のサンドボックスの研究のどれも，ポリシーの作成の困難さを解決することは目的になっていない．そこで，本論文はサンドボック

スのポリシーを作成するに当たって，NIDS の攻撃情報のデータベースを用いることを提案している．

7　まとめと将来の課題

従来，通常の NIDS では誤検知が多いために検知しか行えず，攻撃を直接的に防御が出来ないことが問題であった．本論文では，NIDS とサンドボックスを連携させ，NIDS が検知した攻撃を防御できる仕組みを提案した．具体的には，NIDS が検知した攻撃が成功しないよう，サンドボックスを利用してプログラムの計算機資源へのアクセス権の制御を行うことにより防御を行った．

この防御手法によって，一部の攻撃を防御可能な NIDS に存在するような，誤検知によるサービス拒否は発生しなかった．さらに NIDS の攻撃情報データベースをサンドボックスのセキュリティ・ポリシーのデータベースとして用いることにより，ポリシー作成者の負担を軽減することができた．

本論文では，この手法のプロトタイプを作成して，評価を行った．NIDS とサンドボックスを連携すること自体のオーバーヘッドは 135 マイクロ秒程度であり，実際にシステムを動作させる上でのオーバーヘッドはサンドボックスのオーバーヘッドが大半を占めていて，このオーバーヘッドは十分改善可能であると考えられる．

今後は攻撃の実データを使った検証などが必要である．

参考文献

[1] Timm, K.: Strategies to Reduce False Positives and False Negatives in NIDS, http://securityfocus.com/ infocus/1463 (2001).

[2] CVE: CVE-2000-0573 (2000).

[3] Roesch, M.: Snort: Lightweight Intrusion Detection for Networks, *Proceedings of the 13th USENIX Conference on Systems Administration*, pp. 229–238 (1999).

[4] Goldberg, I., Wagner, D., Thomas, R. and Brewer, E. A.: A Secure Environment for Untrusted Helper Applications, *Proceedings of the 6th Usenix Security Symposium*, pp. 1–13 (1996).

[5] CVD: CVE-2001-0731 (2001).

[6] Paxson, V.: Bro: a system for detecting network intruders in real-time, *Computer Networks*, Vol. 31, No. 23–24, pp. 2435–2463 (1999).

[7] Wang, H. J., Guo, C., Simon, D. R. and Zugenmaier, A.: Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits, *ACM SIGCOMM '04*, pp. 193–204 (2004).

[8] 大山恵弘, 神田勝規, 加藤和彦: 安全なソフトウェア実行システム SoftwarePot の設計と実装, コンピュータソフトウェア, Vol. 19, No. 6, pp. 2–12 (2002).