通信端点におけるTCPレベル侵入検知モジュールの実装

渡辺 祐介† 前田 敦司† 山口 喜教†

筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻†

1.はじめに

ネットワーク経由での攻撃を防ぐため、ネットワーク侵入検知システム(NIDS)を設置することが一般的になってきている。NIDSは、ネットワーク上のトラフィックを検査し、不正アクセスと思われるデータを発見したときに、システムの管理者に警告を発するなどといったアクションを行うシステムである。一般にNIDSは外部ネットワークとの接続点付近に設置し、保護対象のネットワーク内に流れてくる全ての通信を集中的に検査する。しかし、ネットワークの高速化・大容量化に伴い、スケーラビリティの欠如が問題となっている。また、ネットワーク内部からの攻撃を検知することが困難であるという問題もある。

これらの問題点の解決策として、我々はバックボーンで集中的に検査を行う代わりに、通信の端点となる個々の計算機上において、その上で実行されるサーバプログラムに対する攻撃を検知するためのモジュール(idsmon)を提案してきた[1]。idsmonは端点となる計算機上でセッションごとに起動されるため、可能な限り軽量であることが必須である。そのためidsmonは、侵入検知ルール集合のうちで特定の通信の検査に必要なルールのみを抜き出し、それをメモリ上にコンパクトな形で保持して検査を行なう。

本稿では、まずNIDSとidsmonの基本仕様について述べる。次に、このidsmonの起動時間とメモリ使用量を改善するための手法について述べ、最後にその改善に対する評価を行う。

2. NIDS

NIDSの実装として主流の方式は、あらかじめ登録した文字列のパターン(シグネチャ)にマッチした際に指定したアクションを実行するルールベースのものである。NIDSの実装の代表的なものとしてSnort[2]が挙げられる。Snortの動作概要を以下で説明する。Snortにパケットが到着すると、第1段階としてパケットのペイロードのパターンマッチを行う。パターンマッチは検知パターンを登録したDFAを表の形式にコンパイルしたものを使用して高速に実行される。パターンにマッチしたパケットが見つかると、検知ルールに基づいてそれが攻撃パケットかどうかを判断するための詳細なチェックが行われる。

本研究におけるidsmonも、基本的にこの手順で通信の検査を行う。ただし、今回は検知の第1段階にあたるペイロードのパターンマッチのみを対象とする。

3. idsmon

3.1 idsmonの概要

idsmonは以下のような仕様を持つ。

・idsmonはxinetdから起動される。xinetdは監視対象のサーバプログラムのポートを監視しており、クライアントからサーバへリクエストが来たときに、クライアントとのTCPコネクションを確立した状態でidsmonを起動させる。

・起動されたidsmonは検査のためのセットアップを行った後、子プロセスとしてサーバプログラムを起動し、クライアントとサーバ間の通信を中継する。この際に通信内容をルールに従って検査し、ルール中のパターンにマッチした場合には、指定されたアクションを実行する。

監視の対象となるプログラムがスタンドアロン型のサーバの場合は、xinetdから起動されたidsmonがTCPソケットを使用してサーバプログラムが待ち受けるポートへ直接接続し、TCPソケット経由で通信を中継してやることで監視をおこなう。

3.2 ルールの選択的ロードとその問題点

idsmonは先に述べたセットアップ時にSnortのルールファイルを読み込み、文字列のパターンマッチングを行うmatcherを作成する。このmatcherは我々が軽量侵入検知モジュール向けに研究を行っているもので、Snortで用いられているmatcherと比較して、速度の低下を抑えつつ、メモリ使用量を1/7程度にするものである[1][3]。一般的なNIDSはNIDS以下のネットワーク内への全ての通信の監視に対応するために全てのルール及びmatcherをメモリ上に保持しておく必要がある。しかし、idsmonプロセスは特定のサーバプログラムの1つのセッションに対してのみ検査を行うため、必要のないルールは破棄することができる。これにより、従来のNIDSよりも格段にメモリ使用量を抑えることが可能となる。ルールの選別処理は以下に示す。

・検査対象の通信のアドレス情報を得る。

・ルールは同一のアドレス情報ごとに線形リストで保持されているので、上記の情報からIPアドレスとポートがマッチしないルールリストを削除していく。

この手法をとることで、idsmonのメモリ消費量を削減することができるが、クライアントからリクエストが要求されてから通信が始まるまでに時間がかかるという問題点がある。また同一のサーバに多数のリクエストが同時に来た場合は、それぞれのidsmonプロセスに同じmatcherが作成されてしまい、無駄が出てしまう。そこで、一度作成したmatcherをローカルに保存しておき、同じmatcherを必要とする通信がきた場合にそれを再利用する機能を提案する。また、matcherをロードする際に共有メモリ領域に置くことで、matcherを利用するidsmonプロセス間で共有させる機能を提案する。

4. idsmonの改良

4.1 動作の概要

改良版idsmonの動作概要を以下に示す。

・対象の通信のアドレス情報を取得する。その後そのアドレス情報をキーとして後述する4次元インデックステーブルを引き、対象の通信の監視に必要なmatcherファイルを特定する。

・matcherファイルが存在する場合は、そのファイルをmmapする。これでmatcherが共有メモリ上に置かれることになり、以後このmatcherを使用するidsmonプロセスは共有メモリ上のmatcherを参照することになる。
・matcherファイルが存在しない場合は、RuleBuilderを呼び出して、matcherファイルを作成させる。その後作成されたmatcherファイルをmmapする。

4.2 RuleBuilder

RuleBuilderはmatcherファイルとそれの検索に用いる4次元インデックステーブルを作成するためのプログラムである。プログラム引数にアドレス情報を渡して起動すると、そのアドレス情報に対する通信を検査するためのmatcherファイルが作成される。また、引数に"-i"を指定するとインデックステーブルを作成する。検知ルールを追加・削除・更新した際は、これを実行して、インデックステーブルを更新しなければならない。

4.3 4次元インデックステーブル

4次元インデックスデーブルはmatcherの作成とmatcherファイルの検索のために用いられる。テーブルはソースIPアドレス・ポート、ディスティネーションIPアドレス・ポートの4つからなる。テーブルの各要素にはIPアドレスもしくはポートの範囲と、その範囲を対象としたルールの番号が格納されている。各々のテーブルに対して検査対象の通信のIPアドレス、ポートをキーとしてインデックステーブルを引き、得られたルール集合の論理和をとると、目的とするルール集合を過不足なく得られる。matcherファイルは、得られたルール集合中に記述されている検査パターン文字列をmatcherに登録し、matcher保存ルーチンを実行することで作成することができる。

matcherファイルのファイル名は、上記の方法で得られたルール集合の文字列表現を使用しているため、matcherファイルの検索も上と同様の操作で行える。

5. 評価

はじめに、idsmonのセットアップに要する時間を測定する。セットアップ時間は、idsmonが起動され、クライアント・サーバ間の通信が開始できる状態になるまでの時間とする。測定対象は(1)従来のidsmon、(2)改良版idsmon(matcherファイルが存在しない場合)、(3)改良版idsmon(matcherが存在する場合)とする。想定する通信はsrc ip:192.168.0.1、src port 10000、dst ip 192.168.0.2、dst port 21(ftp)とした。評価環境はIntel Core 2 Duo 2.2GHz, 2GB RAM, Mac OS X 10.5.1, gcc 4.0.1である。結果を表1に示す。

表1.idsmonのセットアップ時間の比較

	セットアップ時間(sec)
(1)idsmon	0.233
(2)改良版idsmon (ファイルなし)	0.268
(3)改良版idsmon (ファイルあり)	0.00305

表1から明らかなように、改良版idsmonはmatcherファイルが存在する場合において従来のidsmonよりも素早く検査をはじめることができる。matcherファイルが存在しない場合は従来のidsmonと同程度の時間がかかるが、これは初めてそのmatcherが必要な通信が来たときの最初の1回のみであり、以後同じmatcherを利用する通信がきた場合は(3)の速度でセットアップを終了させることが可能なため、同時間に多数の通信が来た場合にも十分耐えられるものであると考えられる。

次に、matcherを複数のidsmonプロセスで共有させたときのメモリ使用量削減効果を見る。この実験では同じmatcherを使用するidsmonプロセスを複数起動させ、matcherの共有がある場合とない場合での、全てのidsmonプロセスの実メモリ使用量合計の比較を行った。想定する通信は先の実験と同様である。また、この通信に対するmatcherのメモリ使用量は324KBである。

結果を図1に示す。プロセス数が多くなるほどmatcherの共有によるメモリ使用量の削減の効果が大きくなることが分かる。アクセス数が多いサーバでは、同じmatcherを使用するidsmonプロセスが同時間に数多く起動されることが予想されるので、matcher共有によるメモリ使用量削減の効果は大きいと思われる。

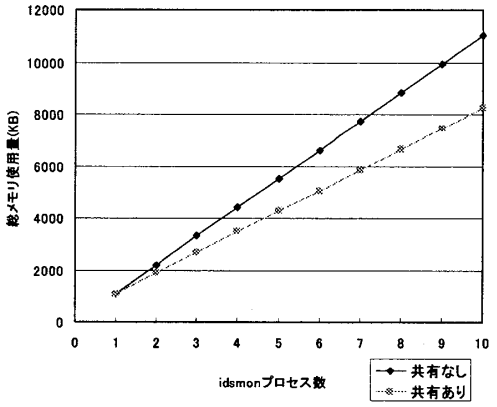

図1.matcher共有によるメモリ使用量の比較

6. まとめ

本稿ではサーバマシン上でセッション単位で動作する軽量侵入検知モジュールの概要とその改善点、及び評価について述べた。改善の結果、検知モジュールのセットアップに要する時間が短縮され、また複数の検知モジュールプロセスを実行させたときのメモリ使用量を削減することができた。今後は、パターンマッチング後の詳細な検査を行うデーモンの実装を行う予定である。

7. 謝辞

本研究の一部は，科学研究費補助金特定領域研究「情報爆発IT基盤」(領域番号456)「通信端点における分散検知モジュールによる侵入防止機構」(課題番号19024008)をうけて行われた

参考文献

[1]前田敦司 渡辺祐介 西孝王 山口喜教, "通信端点における軽量侵入検知モジュールの試作", 信学技報, Vol.106, No.436 (CPSY2006-54), pp.13-18 (2006)
[2]Snort, http://www.snort.org
[3]西孝王 前田敦司 山口喜教, "軽量ネットワークIDS向け検知ルール圧縮法の提案", 情報処理学会 第70回全国大会(2008)