Linux サーバ実習 2

セキュアな Web サーバ構築

日本工学院八王子専門学校 IT カレッジ

2009 年後期

この実習では各種サーバのセキュリティを向上させる方法を学ぶ。

目次

第 1 章

セキュリティを考慮した Apache のビルドと設定

CentOS など現在の Linux ディストリビューションでは Web サーバとして Apache が自動的にインストールされ、何も設定しなくても動作するが初期状態の設定はセキュリティなどは考慮されていないため安全な運用が出来ない。初期状態のままで使用することはとても危険である。また、Web サーバに限らず様々なアプリケーションはパッケージとして管理されており、何らかの脆弱性が発見されてもそれを適用したパッケージがすぐに登場するわけではない。そこで、最新のソースファイルを取得し、セキュリティを考慮した設定を施して運用する方法を確認する。

1.1 最新のソースコードから Apache をビルドする

The Apache HTTP Server Project[*1]のサイトから最新のソースコードの圧縮された TAR ボールを取得し、ビルド、インストールする。今回はインストール先として/home/httpd を使用する。

```
$ tar jxf httpd-2.2.17.tar.bz2
$ cd httpd-2.2.17
$ ./configure --prefix=/home/httpd
$ make
$ sudo make install
```

図 1.1　ソースからのビルド

1.2 ユーザとグループを追加する

Apache を動作させるための下記のユーザまたはグループを Linux に登録する。ユーザはログインの出来ないユーザにすること。

1. Apache の実行ユーザとグループ：httpd

[*1] http://httpd.apache.org/

2. Web サイトコンテンツ管理グループ：webcont
3. CGI 開発グループ：webdev

1.3 httpd.conf の編集

ビルド設定時に prefix を指定した場合、Apache の設定ファイル httpd.conf は **prefix で指定したディレクトリ /conf/httpd.conf** となる。

1.3.1 httpd の実行ユーザとグループの設定

Apache の実行時ユーザとグループはディレクティブ User と Group で指定する。先ほど追加したユーザ、httpd に変更する。

1.3.2 安全なディレクトリ構造を使う

設定ファイル (httpd.conf 等) で次のディレクティブでより安全なディレクトリを指定し、httpd などの実行関連ファイル、Web コンテンツ、ログはそれぞれ異なるディレクトリに配置する。下記のように設定する。

- ServerRoot： /home/httpd
- DocumentRoot： /www/htdocs
- ScriptAlias： /cgi-bin/ ”/www/cgi-bin/”
- CustomLog： /www/logs/access_log common
- ErrorLog： /www/logs/error_log
- <Directory： /home/httpd/htdocs> を <Directory /www/htdocs>
- <Directory： /home/httpd/cgi-bin> を <Directory /www/cgi-bin>

上記のディレクトリが存在していなければ作成し、必要なファイルは Apache のインストール先からコピーする (/home/cgi-bin -> /www)。

1.3.3 ディレクトリインデックスの無効化

インデックスファイル (index.html など) がないディレクトリへのアクスセスがあった時、そのディレクトリのファイルリストを表示させるのはセキュリティ上好ましくない。httpd.conf を編集し、ディレクトリ/と DocumentRoot のディレクトリインデックスを無効にする。(Options ディレクティブで-Indexes を指定する)。

1.3.4 バージョン番号の非表示

URL に存在しない場所を指定した時の表示にサーバの名前、バージョンなどを情報を表示する必要はない。これらの情報がわかると脆弱性を突いた攻撃を受ける恐れがある。これを抑止するには ServerSignature ディレクティブを Off に設定する。

1.3.5 ユーザの上書きの無効化

Apache にはユーザのディレクトリなどにファイル.htaccess を置き、サーバの動作をユーザ毎、ディレクトリ毎に変更する機能がある。これもセキュリティを低下させる原因となるのでそれを禁止する。httpd.conf を次のように編集する。

リスト 1: ユーザの上書きの無効化設定

```
<Directory />
  AllowOverride None
</Directory>
```

1.4 ファイルとパーミッションの設定

Apache の運用に必要なファイルやディレクトリのパーミッションを Web サーバの管理、コンテンツの開発や管理に関わるユーザやグループのみに限定するように設定する。

1. DocumentRoot 以下のファイルやディレクトリのオーナとグループを httpd と webcont にする。
2. DocumentRoot 以下のディレクトリとファイルのパーミッションを 2570 にする。
3. ScriptAlias 以下のディレクトリのオーナとグループを httpd と webdev にする。
4. ScriptAlias 以下のディレクトリのパーミッションを 2570 にする。
5. ログファイルのディレクトリ以下のオーナとグループをともに root にする。
6. ログファイルのディレクトリ以下のパーミッションを 700 にする。
7. Web サイトコンテンツ管理グループに既存のユーザ doraemon を追加する。

1.5 動作チェック

1. Apache が正しく起動するか
2. Apache のプロセスがユーザ httpd の権限で実行されているか。
3. http://localhost/を開いてどう見えるか
4. DocumentRoot のディレクトリにディレクトリ test を作成し、そこをブラウザからアクセスするとどう見えるか
5. DocumentRoot に index.html(内容は何でもいい) を置いてどう見えるか
6. ユーザ doraemon でログインし、DocumentRoot にアクセスでき、ディレクトリ deoraemon とファイル index.html を作れるか
7. ユーザ httpd でログインし、DocumentRoot にアクセスでき、ファイルやディレクトリを作れるか
8. ブラウザで http://localhost/cgi-bin/にアクセスし、サーバのバージョン情報が表示されるかを確認する。

第 2 章

CGI スクリプトのリスク軽減

Web サーバのセキュリティホールの一番の原因が CGI スクリプトで、プログラマのセキュリティに対する知識の少なさがシステムに不法侵入経路やセキュリティホールを作ってしまいます。

2.1 情報の漏洩

CGI スクリプトの作りによってシステムの情報が漏洩することがある。

例えば、引数で指定されたファイルを表示するスクリプト showpage.cgi がある。このスクリプトを使い、ファイル /doc/article1.html を表示させる時は次の URL を指定する。

http://unsafe-site.com/cgi-bin/showpage.cgi?pg=/doc/article1.html

この URL の表記、ファイル名、実際の動作からこのスクリプト showpage.cgi はシステムの動作や設定関連のファイルも表示できると想像できる。そこでこの機能を悪用してパスワードファイルを盗み見るにはつぎの URL を使えばよいことが容易に想像できる。

http://unsafe-site.com/cgi-bin/showpage.cgi?pg=/etc/passwd

このスクリプトは全くセキュリティを考慮していないのでパスワードファイルの内容は全て見られてしまう。

2.2 OS コマンドインジェクション

CGI スクリプトを介して OS コマンドを実行できることを **OS コマンドインジェクション**[*1]といい、不用意に OS コマンドを実行できるようなスクリプトはシステムの安全性を脅かす。

perl では system() 関数を使うと OS コマンドを実行できる。次のリストは whois コマンドを実行する perl のスクリプトです。

リスト 2: 不用意に書かれた CGI スクリプトによるリスクの例 (whois.pl)

```
#!/usr/bin/perl
# 目的をよく考えずに書かれた CGI スクリプトのセキュリティリスクを示す。
use CGI qw(:standard);
# クエリー文字列からドメイン名を取得する
my $domain = param("domain");
# 適切なコンテンツタイプを出力する。
# whois の出力はプレーンテキストなので、
# ここではコンテンツタイプとして text/plain を使う。
```

[*1] インジェクション (injection) は注入するという意味。

```
print "Content-type: text/plain\n\n";
# 不適切なシステムコール
system("/usr/bin/whois $domain");
# 逆引用符を使ったもう 1 つの不適切なシステムコール
# my $output = `/usr/bin/whois $domain`;
# print $output;
exit 0;
```

このスクリプトはブラウザのアドレスに次のように URL を入力し、動作させる。

http://unsafe-site.com/cgi-bin/whois.pl?domain=anydomain.com

次のリストはドメインに neec.ac.jp を指定したときの出力である。

```
[Querying whois.jprs.jp]
[whois.jprs.jp]
[ JPRS database provides information on network administration. Its use is    ]
[ restricted to network administration purposes. For further information,     ]
[ use 'whois -h whois.jprs.jp help'. To suppress Japanese output, add'/e'     ]
[ at the end of command, e.g. 'whois -h whois.jprs.jp xxx/e'.                 ]

Domain Information:
a. [Domain Name]                NEEC.AC.JP
g. [Organization]               Nihon Kogakuin College
l. [Organization Type]          Professional School
m. [Administrative Contact]     AH035JP
n. [Technical Contact]          AH035JP
p. [Name Server]                ns1.neec.ac.jp
p. [Name Server]                ns2.neec.ac.jp
p. [Name Server]                ns2.iprevolution.co.jp
[State]                         Connected (2010/02/28)
[Registered Date]               1996/02/08
[Connected Date]                1996/02/29
[Last Update]                   2009/04/07 10:08:20 (JST)
```

これを次のように書き換えると任意のコマンドを実行できる。

http://unsafe-site.com/cgi-bin/whois.pl?domain=anydomain.com|ps -ax

実際に実行した結果の一部を示す。ドメインの情報ではなく、サーバのプロセス一覧が表示されてしまう。

```
10409 ?        Ss     0:00 /sbin/dhclient -1 -q -lf /var/lib/dhclient/dhclien
10522 ?        Ss     0:00 /home/httpd/bin/httpd -k start
10523 ?        S      0:00 /home/httpd/bin/httpd -k start
10524 ?        S      0:00 /home/httpd/bin/httpd -k start
10525 ?        S      0:00 /home/httpd/bin/httpd -k start
10526 ?        S      0:00 /home/httpd/bin/httpd -k start
10527 ?        S      0:00 /home/httpd/bin/httpd -k start
```

```
10529 ?        S      0:00 /bin/sh /usr/lib/firefox-3.0.14/run-mozilla.sh /us
10557 ?        Rl     0:09 /usr/lib/firefox-3.0.14/firefox -UILocale ja
10581 ?        S      0:00 /home/httpd/bin/httpd -k start
10652 ?        SN     0:00 /bin/bash /usr/bin/run-parts /etc/cron.daily
11831 ?        SN     0:00 /bin/sh /etc/cron.daily/mlocate.cron
11832 ?        SN     0:00 awk -v progname=/etc/cron.daily/mlocate.cron progn
11836 ?        DN     0:04 /usr/bin/updatedb -f sysfs?rootfs?bdev?proc?cpuset
11866 ?        S      0:00 /usr/bin/perl /www/cgi-bin/whois.pl
11867 ?        S      0:00 sh -c /usr/bin/whois neec.ac.jp|ps ax
11869 ?        R      0:00 ps ax
```

この危険性を排除するには必要な文字以外を削除するようなプログラムに変更する。

リスト 3: 不要な文字を削除し、リスクを軽減した CGI スクリプト (whois2.pl)

```
#!/usr/bin/perl -w
# 目的:前の whois.pl スクリプトの改良版
# ドメイン名に許される文字セットを変数に代入する。
use CGI qw(:standard);
# 許される文字セットの定義
my $DOMAIN_CHAR_SET = '-a-zA-Z0-9_.';
# クエリー文字列からドメイン名を取得する
my $domain = param("domain");
# 許される文字セットに含まれない文字を削除する。
$domain =~ s/[^$DOMAIN_CHAR_SET]//g;
# 適切なコンテンツタイプを出力する。
# whois の出力はプレーンテキストなので、
# ここではコンテンツタイプとして text/plain を使う。
print "Content-type: text/plain\n\n";
# 不適切なシステムコール
system("/usr/bin/whois $domain");
# 逆引用符を使ったもう 1 つの不適切なシステムコール
# my $output = `/usr/bin/whois $domain`;
# print $output;
exit 0;
```

このプログラムであれば、先ほどの URL を指定してもプロセス一覧は表示されない。

```
[Querying whois.internic.net]
[whois.internic.net]

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

No match for domain "NEEC.AC.JPPSAX".
```

```
>>> Last update of whois database: Sun, 18 Oct 2009 16:53:38 UTC <<<
```

2.3 演習

上記のファイルを自分の Linux 上で実行し、そのリスクを確認しなさい。ファイルは指定された場所からダウンロードし、前回作成した CGI スクリプト用のディレクトリに置き、適切なパーミッションを設定すること。ただし、whois は学内のプロキシサーバに阻まれ、実行できないので完全な動作は自宅などプロキシサーバのない環境で試しておくこと。

第 3 章

CGI スクリプトのラップ

3.1 ラッピング

CGI スクリプトのリスクを減らすには CGI スクリプトを使用しないことであるが、現実的ではない。CGI スクリプトを集中管理し、開発状況を監視すれば適切なスクリプトを作れるがこれもなかなか難しい。そこで CGI スクリプトには何らかのリスクを含まれていることを前提にもし、そのリスクによる被害が発生してもそれを最小限に食い止めるようにする。

CGI スクリプトはデフォルトで Apache のユーザ権限で実行されるが、これをスクリプトの所有者の権限で実行できればもし何らかの被害があってもその被害は実行ユーザの UID の範囲に限定でき、被害を他のユーザに及ぼすことがない。Apache の実行 UID 以外の UID で CGI スクリプトを実行するには**ラッパー**という特殊なプログラムを使う。

3.2 suEXEC

suEXEC は Apache に付属するサポートアプリケーションで、ユーザの UID で CGI スクリプトを実行させる仕組みである。suEXEC ラッパーは CGI、SSI プログラムを実行する前にスクリプトなどに対して次のテストをし、要求が有効であるかを確認出来た時にそれを実行させる。

- ラッパーを実行できるユーザがその CGI スクリプトを所有していること
- CGI ディレクトリにまたは CGI スクリプトが所有者以外のユーザに可書き込みを禁止していること

3.2.1 suEXEC のインストールと設定

前回導入した Apache は suEXEC をサポートしていないので再 configure し、ビルド、make、インストールする。手順は次の通り。

```
$ ./configure --prefix=/home/httpd \
              --enable-suexec \
              --with-suexec-caller=httpd \
              --with-suexec-userdir=public_html \
              --with-suexec-uidmin=100 --with-suexec-gidmin=100 \
              --with-suexec-safepath="/usr/local/bin:/usr/bin:/bin"
$ make
$ sudo make install
```

図 3.1 Apache を suEXEC 対応にする

3.2.2 起動チェック

suEXEC を有効にした Apache を起動すると error_log に次のようなメッセ-が記録されるのでこれを確認し、suEXEC が有効かを確認する。

リスト 4: suEXEC 有効時のメッセージ例

```
[Wed Oct 15 22:15:58 2008] [notice] suEXEC mechanism enabled (wrapper: /home/httpd/bin/suexec)
```

3.2.3 suEXEC の動作確認

suEXEC の動作を確認するにはユーザのホームページが必要なので UserDir ディレクティブを有効にする。今回の Apache では httpd.conf にはユーザディレクトリの設定はなく、別ファイルの conf/extra/httpd-userdir.conf に記されている。

1. httpd.conf でこのファイルをインクルードするように設定を変更する。
2. httpd.conf の AddHandler cgi-script の行のコメント文字を削除し、行末に.pl を追加する。
3. httpd-userdir.conf の Options に ExecCGI を追加する。
4. apachectl のオプション t で設定ファイルをチェックする。
5. Apache を再起動する。

3.3 CGIWarp

CGIWrap は Apache に手を加えずに CGI スクリプトを安全に実行する仕組みで、アクセスコントロールも可能です。

3.3.1 ファイルの入手とインストール

GIWrap[*1]の TAR ボールをダウンロードし、次の手順でビルドする。

[*1] http://cgiwrap.sourceforge.net/

1. ファイルの展開
2. 展開したディレクトリに移動
3. configure でビルドに必要なファイルを作る。オプションは次のものを設定する。
   (a) インストール先　/www/cgi-bin
   (b) Apache 実行ユーザ　httpd
   (c) ログファイル/www/logs/cgiwrap.log
   (d) アクセスコントロールファイル cgiwrap.allow と cgiwrap.deny を/www/cgi-bin に
4. ビルドし、インストールする

実行するコマンドは次の通り。

```
$ tar zxfv cgiwrap-4.1.tar.gz
$ cd cgiwrap-4.1
$  ./configure --with-install-dir=/www/cgi-bin \
               --with-httpd-user=httpd \
               --with-logging-file=/www/logs/cgiwrap.log \
               --with-allow-file=/www/cgi-bin/cgiwrap.allow \
               --with-deny-file=/www/cgi-bin/cgiwrap.deny
$ make
$ sudo make install
```

図 3.2　CGIWrap の導入方法

インストールに成功すると必要なファイルが指定したディレクトリ (今回は/www/cgi-bin) にコピーされ、オーナー、パーミッション、リンクが適切に設定される。

```
-rwsr-xr-x 2 root    root   64272 10月 22 18:15 cgiwrap
-rwsr-xr-x 2 root    root   64272 10月 22 18:15 nph-cgiwrap
```

図 3.3　CGIWrap のインストール結果

3.3.2 アクセス制御ファイルの作成

CGIWrap では configure で指定したアクセス制御ファイルによって CGI スクリプトへのアクセスを制御できる。今回はアクセス許可用のファイルは/www/cgi-bin/cgiwrap.allow、アクセス拒否用ファイルは/www/cgi-bin/cgiwrap.deny である。

このファイルにはユーザ名あるいはユーザ名とホスト IP アドレスとサブネットマスクを記述する。たとえば、ユーザ doraemon と IP アドレス 172.16.114.1、サブネットマスク 255.255.255.0 のホストからのユーザ sakabe のアクセスを許可、あるいは拒否する時は次のように記す。

リスト 5: cgiwrap.allow および cgiwrap.deny ファイルの内容

```
doraemon
sakabe@172.16.114.1/255.255.255.0
```

3.3.3 動作確認

CGIWrap のインストール、アクセス許可および拒否用ファイルの準備が完了したら上記ファイルに追加したユーザの public_html に cgi-bin を作り、そこに CGI スクリプトを置く。スクリプトファイルは Apache に付属の printenv などを使い、そのオーナー、グループ、パーミッションを適切に設定する。

ブラウザからのアクセスは次の URL を使う。

http://サーバ名/cgi-bin/cgiwrap/ユーザ名/スクリプト名

次に示す状態の時の動作をブラウザでのメッセージ、ログファイルの内容を確認する。

1. CGI スクリプトのオーナーを他のユーザに変更した時
2. CGI スクリプトのグループを他のグループに変更した時
3. CGI スクリプトがオーナー以外で書き込み可能な時

3.4 演習

各自の Linux で suEXEC を使えるように設定、インストールし、テスト用スクリプトにアクセスし、suEXEC が正しく動作しているかをブラウザの表示と各ログファイル (error_log と/home/httpd/logs/suexec_log) から確認する。

1. ユーザがホームページを公開できるように設定ファイルを修正する。
2. テストプログラム (リスト 6) を~/public_html に保存し、実行権を付加する。
3. ブラウザで http://localhost/~ユーザ名/test.pl にアクセスし、環境変数一覧が表示されるか確認認する。
4. test.pl をその他のユーザでも書き込めるように変更してアクセスする。
5. test.pl の所有者を root など別ユーザに変更しアクセスする。
6. test.pl の所有者を元に戻し、public_html のパーミッションを所有者以外も書き込み可能にし、アクセスする。
7. test.pl のグループを他のグループに変更し、アクセスする。

リスト 6: テスト用プログラム

```
#!/usr/bin/perl
#
#
my ($key,$value);
print "Content-type: text/html\n\n";
print "<h1>Test of suEXEC</h1>";
foreach $key (sort keys %ENV){
  $value = $ENV{$key};
  print "$key = $value <br>";
}
exit 0;
```

第 4 章

コンテンツへのアクセスを制限する

Web サーバは全てのホスト、ユーザにコンテンツを提供するのが基本だが、よりセキュリティを高めるには特定のホスト、あるいはユーザのみがアクセスできるようにすればよい。コンテンツへのアクセス制限は次の 2 つの方法がある。

1. IP アドレスまたはホスト名による制限
2. HTTP 認証スキーマの利用での制限

4.1 IP アドレスまたはホスト名による制限

Apache のモジュール mod_access を使い、IP アドレスまたはホスト名による制限を実施できる。このモジュールでは次の 3 つのディレクティブを使い、アクセス制御を行う。

Allow　あるディレクトリにアクセスできるホストのリストを定義するディレクティブ
Deny　あるディレクトリにアクセスできないホストのリストを定義するディレクティブ
Order　Allow および Deny ディレクティブを評価する順番を定義するディレクティブ

例えば、ホスト host.exmaple.com のアクセスからのディレクトリ/www/htdocs/conts へのアクセスを拒否し、それ以外のホストからのアクセスを許可する場合の設定は次のリストのように記す。

リスト 7: アクセス制御の例

```
<Directory "/www/htdocs/conts">
  Order Allow,Deny
  Deny from host.example.com
  Allow from all
</Directory>
```

この例のように Allow from ホスト名など、Deny from ホスト名などと記されているように許可あるいは制限するホストの名前を記します。

さらに GET、POST、PUT などの HTTP 要求メソッドへの制限もできる。つぎのリストは/cgi-bin に対して POST 要求を実行できるホストを限定する例です。

リスト 8: HTTP 要求メソッドによる制限

```
<Location /cgi-bin>
  <Limit POST>
```

```
    Order Deny,Allow
    Deny from ALl
    Allow from example.com
  </Limit>
</Location>
```

4.2 認証を使う

ユーザ名とパスワードを使った認証を実現するには mod_auth や mod_auth_digest を使う。mod_auth モジュールを使うとテキストファイルに保存されたユーザ名、グループ、パスワードを使った認証ができる。少人数での使用に適している。

認証を使うために、httpd.conf で次の二つのディレクティブを設定しておく。

1. AccessFileName .htaccess
2. AllowOverride All

4.2.1 ユーザ名とパスワードを要求させる

ディレクトリ/www/htdocs/readonly へのアクセスをユーザ名 doraemon だけに制限する。このユーザのパスワード nobita とする。次の手順でアクセス制限を実現する。

1. ディレクトリ/www/htdocs/readonly を作成し、そこに適当な内容のファイル index.html を置く。
2. htpasswd コマンドでユーザファイルを作成する

```
# htpasswd -c /www/secrets/.htpasswd doraemon
```

図 4.1 パスワードの作成

3. .htaccess ファイルを作成する

リスト 9: .htaccess

```
AuthName "Doraemon Only"
AuthType Basic
AuthUserFile /www/secrets/.htpasswd
require user doraemon
```

4. ファイルのパーミッションを設定する

上記設定が完了したら Web ブラウザでディレクトリ (http://localhost/readonly/) へアクセスする。
ユーザ名とパスワードの入力を求められ、設定した情報を入力し、正しく動作するかを確認する。

4.2.2 ユーザグループにアクセスを許可する

ユーザだけでなく、複数ユーザをグループにし、そこに属しているユーザに対してアクセスを許可する事もできる。ここではユーザ nobita、shizuka をグループ化する。グループ名は doraemons とする。

手順は次の通り。

1. htpasswd コマンドでユーザを追加する。-c オプションは不要。
2. グループファイル (/www/secrets/.htgroup) を作成し、ユーザを追加する。

リスト 10: グループファイルの内容

```
doraemons: nobita shizuka
```

3. /www/htdocs/readonly/.htaccess ファイルを編集する

リスト 11: グループを追加した.htaccess

```
AuthName "Doraemon Only"
AuthType Basic
AuthUserFile /www/secrets/.htpasswd
AuthGroupFile /www/secrets/.htgroup
require user doraemon
require group doraemons
```

設定が完了したら Web ブラウザでアクセスし、正しく動作するかを確認する。

4.3 演習

1. 検証用に用意した PC の IP アドレスからのアクセスを拒否するように設定する
2. 認証テスト用のディレクトリ/www/htdocs/authtest を作り、このディレクトリでの認証用のユーザとして doraemon を登録し、認証の効果を確認しなさい。パスワードはユーザと同じにする。
3. http://localhost/manual/mod/mod_auth_digest.html[*1]を読み、ダイジェスト認証を使った認証を可能にしなさい。

[*1] httpd.conf、391 行をコメントアウトし、サービスを再起動すれば http://localhost/manual/mod/mod_auth_digest.html でも同様の文書を読める。

第 5 章

Apache に SSL を適用する

HTTP に限らず、多くのプロトコルではネットワーク上を流れるデータは平文のままで、その内容を盗聴することも可能です。今回は HTTP の通信内容を暗号化する SSL を適用した Web サーバを構築します。

5.1 SSL とは

Secure Socket Layer の略。インターネットでプライバシー情報やクレジットカード番号などを安全に送受信できる暗号化通信を実現する OSI 基本参照モデルのトランスポート層のプロトコルである。

アプリケーションプロトコルでは暗号化を意識することなく透過的に利用できる。

SSL は Netscape Communications 社が開発したプロトコルであるが、現在は IETF が引き継ぎ TLS という名称に変更し、修正したバージョンを規定している。

SSL では公開かぎ暗号、秘密かぎ暗号、デジタル署名、ハッシュ関数などの複数のセキュリティ技術を利用し安全性を確保している。

- 利用する鍵の長さは 40、56、128 ビットである。
- URL スキーマは http ではなく https である。
- TCP のポート 443 番を使うのでファイアウォールでこのポートを閉じている場合には利用できない。
- データの暗号化・復号化にはサーバで膨大な処理を必要とするためサーバのパフォーマンスを低下させる。
- これを改善するために SSL アクセラレータという専用ハードウェアがある。

5.2 SSL 暗号化通信の流れ

クライアントとサーバ間では次の手順で暗号化通信を行っている。

1. クライアントとサーバ間で暗号化に関する情報を交換する。
2. サーバからクライアントへサーバの証明書が送付される。認証局の証明書も同時に送付される。
3. クライアントは受信した証明書が正しいかの認証を行う (サーバ認証)。
4. かぎ生成の元データを交換し、共通かぎを生成する
5. クライアントからサーバにクライアントの証明書を送付する。
6. サーバは受信した証明書が正しいかの認証を行う (クライアント認証)。
7. 通信開始

5.3 OpenSSL の導入

SSL の実装例で無料で使用できる OpenSSL[*1]のソースコードをウェブサイトからダウンロードし、ビルド、インストールします。

```
$ tar zxfv openssl-1.0.0c.tar.gz
$ cd openssl-1.0.0.c
$ ./config
$ make
$ make test
$ sudo make install
```

図 5.1 OpenSSL のビルド手順

5.4 Apache を SSL 対応でビルド

以前導入した Apache で SSL を利用できるように再設定を実行し、ビルドする。configure スクリプトに SSL 関連の指示を追加する。

```
$ ./configure --prefix=/home/httpd \
              --enable-ssl \
              --with-ssl=/usr/local/ssl \
              --enable-so
$ make
$ sudo make install
$ sudo /home/httpd/bin/apachectl -k start
$ sudo /home/httpd/bin/apachectl -l | grep ssl
  mod_ssl.c
```

図 5.2 Apache の再ビルド、インストール、モジュールの確認

5.5 設定ファイルの編集

ビルド、インストールが完了したら SSL を利用できるように複数の設定ファイル httpd.conf、httpd-ssl.conf、openssl.cnf を編集する。Apache の設定ファイルは再インストールしても以前の設定が残っているのでこれまでの設

[*1] http://www.openssl.org/

定はそのまま適用される。

5.5.1 httpd.conf を編集する

初期状態では SSL に関する設定ファイルは読み込まれないので httpd.conf で SSL の設定を読み込ませ、有効にする。リスト 12 のように変更する。

リスト 12: httpd.conf の変更

```
#include conf/extra/httpd-ssl.conf
  ↓
include conf/extra/httpd-ssl.conf
```

SSL 経由でアクセスするディレクトリを/home/httpd/htdocs にし、このディレクトリへのアクセスを可能にする以下の設定を httpd.conf の末尾に追加する。

リスト 13: SSL 向け DocumentRoot の設定

```
<Directory "/home/httpd/htdocs">
  Options -Indexes -FollowSymLinks
  AllowOverride All
  Order allow,deny
  Allow from all
</Directory>
```

5.5.2 extra/httpd-ssl.conf の編集

Apache で SSL を利用するための設定ファイルの 78 行目のサーバ名を localhost に変更する。

リスト 14: httpd-ssl.conf

```
ServerName www.example.com:443
    ↓
ServerName localhost:443
```

5.5.3 openssl.cnf の編集

OpenSSL の設定ファイル (/etc/pki/tls/openssl.cnf) の 37 行目を編集し、署名、暗号鍵ファイルに関する設定を変える。

リスト 15: openssl.cnf

```
dir             = ../../CA
  ↓
dir             = ./demoCA
```

5.6 認証局、署名、暗号鍵の作成

SSL 通信に必要な認証局、秘密鍵などを作る。作業は/usr/local/ssl で行い、CA.sh というスクリプトを使う。

5.6.1 認証局証明書と秘密鍵の作成

認証局 (CA : Certification Authority) はデジタル公開鍵証明書を発行する信頼された第三者であるが、今回の実習では自分自身が認証局になるため、正式なものではなく、ブラウザにとっては信頼できるか分からない認証局となる。

認証局証明書、秘密鍵作成のためにパスフレーズ、その住所などの情報を入力する。(図 5.3 を参照)

```
$ su -
パスワード入力
# cd /usr/local/ssl
# ./misc/CA.sh -newca
CA certificate filename (or enter to create) Enter キーを押す
(略)
Enter PEM pass phrase:自分で決めたパスフレーズを入力する
Verifying - Enter PEM pass phrase:上で入力したパスフレーズを再入力
(略)
Country Name (2 letter code) [GB]:JP と入力
State or Province Name (full name) [Berkshire]:Tokyo と入力
Locality Name (eg, city) [Newbury]:Hachiouji と入力
Organization Name (eg, company) [My Company Ltd]:NEEC と入力
Organizational Unit Name (eg, section) []:ITSP と入力
Common Name (eg, your name or your server's hostname) []:localhost と入力
Email Address []:自分のメールアドレスを入力

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:Enter キーを押す
An optional company name []:Enter キーを押す
Using configuration from /etc/pki/tls/openssl.cnf
Enter pass phrase for ./demoCA/private/./cakey.pem:上で入力したパスフレーズを入力
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            9f:ea:30:fd:81:e7:22:1a
        Validity
(中略)
Write out database with 1 new entries
Data Base Updated
```

図 5.3 認証局証明書と秘密鍵の作成

5.6.2 認証要求書の作成

次に認証要求書を作成する。図 5.4 を参照。

```
# ./misc/CA.sh -newreq
Generating a 1024 bit RSA private key
.......++++++
(./misc/CA.sh -newca と同じ問に対しては同じことを入力)

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:  Enter キーを押す
An optional company name []:  Enter キーを押す
Request is in newreq.pem, private key is in newkey.pem
```

図 5.4　認証要求書の作成

5.6.3 証明書の署名の作成

次は証明書の署名作成。図 5.5 を参照。

```
# ./misc/CA.sh -sign
Using configuration from /etc/pki/tls/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:上で入力したパスフレーズを入力
(略)
Certificate is to be certified until Nov 12 01:58:23 2009 GMT (365 days)
Sign the certificate? [y/n]:y と入力


1 out of 1 certificate requests certified, commit? [y/n]y と入力
(結果を表示)
Signed certificate is in newcert.pem
```

図 5.5　証明書の署名作成

5.6.4 作成したファイルのコピー

以上の操作で/usr/local/ssl に newcert.pem、newkey.pem、newreq.pem が作成される。このうち、認証局の署名ファイル newcert.pem と秘密鍵のファイル newkey.pem をそれぞれ所定のディレクトリに指定した名前でコピーする。

```
# cp newcert.pem /home/httpd/conf/server.crt
# cp newkey.pem /home/httpd/conf/server.key
```

図 5.6 ファイルのコピー

5.7 動作の確認

以上の作業が終わったら端末で Apache を起動すると次のようにパスフレーズの入力を求められるので、自分の入力したパスフレーズを入力する。

```
# /home/httpd/bin/apachectl start
Apache/2.2.14 mod_ssl/2.2.14 (Pass Phrase Dialog)
Some of your private key files are encrypted for security reasons.
In order to read them you have to provide the pass phrases.

Server localhost:443 (RSA)
Enter pass phrase:  ここでパスフレーズを入力

OK: Pass Phrase Dialog successful.
```

図 5.7 SSL 対応 Apache の起動

Apache が起動したら、自分の Linux から Web ブラウザから https://localhost/にアクセスし、SSL が有効かを確認する。localhost への接続が確認できたら検証用に用意している Windows Vista からもアクセスできるかを確認する。

SSL が有効になっているとブラウザには図 5.8 のような不正なセキュリティ証明書である表示が現れる。最後の行の**例外として扱うこともできます**をクリックする。

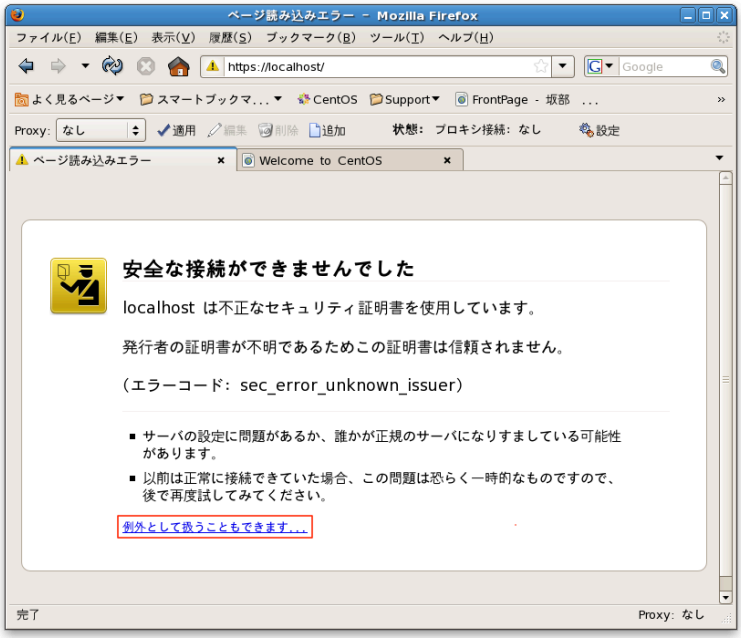

図 5.8 不正なセキュリティ証明書

セキュリティ例外の追加 (図 5.9) が表示されたら**証明書を取得**をクリックし、証明書 (図 5.10) を表示する。

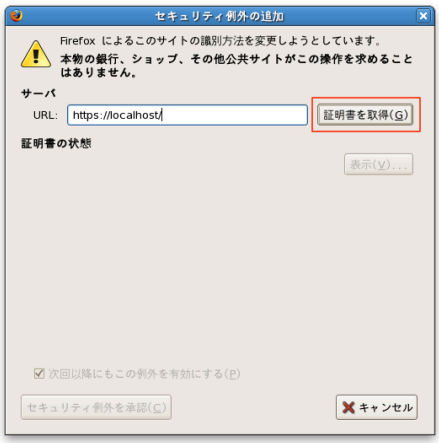

図 5.9 セキュリティ例外の追加

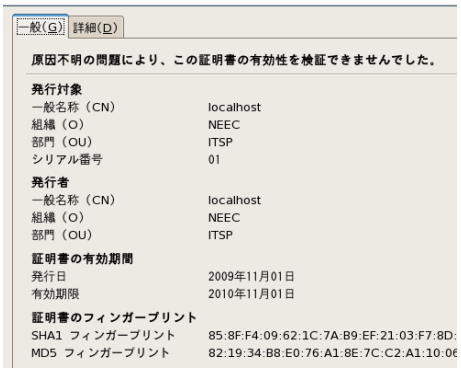

図 5.10 証明書の内容

証明書を確認したらそれを閉じ、セキュリティ例外の追加ダイアログ下部の**セキュリティ例外を承認**をクリックすると該当するページが表示される。

第 6 章

Apache の DoS 攻撃対策

Apache に DoS 攻撃対策モジュール mod_evasive[*1]を導入し、DoS 攻撃、DDoS 攻撃、brute force[*2]攻撃に対処する方法を確認する。

6.1 モジュールの導入

Apache の DSO[*3]機能を利用し、モジュールを動的に組み込みます。まず、Apache が DSO に対応しているかを調べます。もし、対応していなければ、ビルドし直します。

```
# /home/httpd/bin/httpd -l | grep mod_so
mod_so.c ← これが表示されれば OK
```

図 6.1　Apache が DSO 対応かを調べる

モジュール mod_evasive は http://www.zdziarski.com/projects/mod_evasive/あるいは学内実習用サーバからダウンロードし、Apache の apxs[*4]を使って導入します。

```
$ tar zxfv mod_evasive_1.10.1.tar.gz
$ cd mod_evasive
$ sudo /home/httpd/bin/apxs -i -a -c mod_evasive20.c
```

図 6.2　モジュールインストール

コンパイルなどが実行され、モジュールが所定のディレクトリにコピーされ、httpd.conf にモジュールのロード設定が書き込まれます。

[*1] evasive は責任逃れの、回避的なという意味の単語
[*2] パスワード解読のための文字列組み合わせの総当たり攻撃。brute force はむき出しの (荒々しい、激しい) 力、物理的な力という意味
[*3] Dynamic Shared Object:動的共有オブジェクト
[*4] Apache eXtenSion tool

6.2 動作テスト

mod_evasive のディレクトリにある動作試験用スクリプト test.pl を使いチェックする。

```
$ perl test.pl
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
...
...
HTTP/1.1 200 OK
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
...
...
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
```

図 6.3 mod_evasive のテスト

HTTP のレスポンス 200 が続き、その後 DoS 攻撃対策モジュールがアクセスを拒否し、レスポンスが 403 になります。mod_evasive モジュールが入っていないときは HTTP のレスポンスはすべて 200 になります。

test.pl 以外に ApacheBench も使えます。オプション n は試行回数、オプション c は同時接続数の指定。

```
$ /home/httpd/bin/ab -n 1000 -c 100 http://localhost/
```

図 6.4 ApacheBench でのテスト

```
$ /home/httpd/bin/ab -n 1000 -c 100 http://localhost/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Apache/2.2.14
Server Hostname:        localhost
Server Port:            80

Document Path:          /
Document Length:        81 bytes

Concurrency Level:      100
Time taken for tests:   0.244 seconds
Complete requests:      1000
Failed requests:        982
   (Connect: 0, Receive: 0, Length: 982, Exceptions: 0)
Write errors:           0
Non-2xx responses:      982
Total transferred:      409136 bytes
HTML transferred:       199822 bytes
Requests per second:    4096.60 [#/sec] (mean)
Time per request:       24.410 [ms] (mean)
Time per request:       0.244 [ms] (mean, across all concurrent requests)
Transfer rate:          1636.78 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.3      0       8
Processing:    13   23  10.3     21      58
Waiting:       13   23  10.2     20      57
Total:         13   23  11.5     21      64

Percentage of the requests served within a certain time (ms)
  50%     21
  66%     23
  75%     25
  80%     25
  90%     43
  95%     55
  98%     61
  99%     62
```

mod_evasive の動作結果は httpd の error_log と/var/log/messages にそれぞれ記録される。

リスト 16: error_log への記録

```
[Wed Nov 19 23:26:47 2008] [error] [client 172.16.114.1] client denied by server configuration: /www/htdocs/
```

リスト 17: messages への記録

```
Nov 19 23:26:47 fc6 mod_evasive[10817]: Blacklisting address 172.16.114.1: possible DoS attack.
```

6.3 httpd.conf の編集

Apache の設定ファイル httpd.conf を編集しなくても mod_evasive は次のデフォルト設定で動作します。

- 1 秒間に同一ページに対し 2 回以上のアクセスがあった場合、不正アクセス・リストにそのサイトを加える。
- 1 秒間に同一サイトに対し、50 回以上のアクセスがあった場合、不正アクセス・リストにそのサイトを加える。
- 最後のアクセスから 10 秒後に不正アクセス対象リストから除外する。

デフォルト以外の設定は httpd.conf に mod_evasive の設定をリスト 18 のように追加します。

リスト 18: httpd.conf への追加項目

```
<IfModule mod_evasive20.c>
  DOSHashTableSize      3097

  DOSPageCount          2
  DOSPageInterval       1

  DOSSiteCount          50
  DOSSiteInterval       1

  DOSBlockingPeriod     10

  DOSLogDir             "/www/logs"
</IfModule>
```

これらの設定項目は次の意味を持つ。

DOSHashTableSize　不正アクセス・リスト用のメモリ容量

DOSPageCount　指定時間 (DOSPageInterval) 内で DoS 攻撃と見なすページのアクセス数

DOSPageInterval　ページ単位で DoS 攻撃を検出する間隔。秒単位で指定。

DOSSiteCount　DOSSiteInterval 内で DoS 攻撃と見なすサイトのアクセス数

DOSSiteInterval　サイト単位で DoS 攻撃を検出する間隔。

DOSBlockingPeriod　不正アクセス・リストに登録されたホストからのアクセスを阻止する時間。

DOSLogDIr　IP アドレスごとのログファイルの保存ディレクトリ。httpd のプロセスから書き込みできるディレクトリである事。

DOSEmailNotify　新たなリストが追加されたらメールで通知する。

DOSSystemCommand　新たなリストが追加された指定したコマンドを実行する。

DOSWhiteList　不正アクセスリストから除外するホストの IP アドレス指定。ネットワーク単位の指定はクラス C のみが可能でホスト部を*にする。(192.168.1.*)

6.4 実習

以下の実習を行い、どんなコマンドを使ったか、どんなメッセージが表示されるかをよく観察・記録し DoS 攻撃対策の方法を理解する事。

1. mod_evasive 同梱の test.pl の実行結果を確認する。
2. mod_evasive を導入する。
3. mod_evasive を導入していないときといないときとでの test.pl の出力を比較し、mod_evasive の効果を確認しなさい。
4. 各自の PC と検証用 PC から自分の PC にテストプログラムや Web ブラウザからアクセスし、DoS 攻撃をブロックしているかを検証しなさい。
5. ホワイトリストに localhost の IP アドレスを登録し、テストプログラムの動作を確認しなさい。
6. 新たなリストが追加されたら学籍番号のユーザにメールが送信されるように設定しなさい。メールを送信するには sendmail などの MTA が動作している必要がある。

第 7 章

Apache のトラフィック量制限

今回はクライアント単位でのトラフィック量、コネクション数を制限するモジュール mod_bw を Apache に導入し、特定のクライアントによる DoS 攻撃対策の方法を確認する。

7.1 mod_bw の特徴

mod_bw はクライアントの IP アドレス以外に、ブラウザ、バーチャルホストに対する設定も可能で次の特徴を持つ。

- クライアント単位でトラフィック量を制限できる
- クライアント単位でコネクション数を制限できる
- ディレクトリごとに設定を使い分けれる
- 制限対象に URL 以外にもファイルの種別やサイズを使える
- バーチャル・ホストでも設定できる
- アクセス制限されたクライアントへの通知画面を自由に設定できる
- プロキシサーバ経由の攻撃があった場合、そのプロキシサーバを利用するクライアントがすべて制限対象となる

7.2 モジュールの導入

mod_bw も DSO 機能を利用して Apache に組み込む。mod_bw の配布元[*1]から TAR ボールアーカイブを取得、展開し、apxs コマンドで組み込みます。

```
$ tar zxfv mod_bw-0.8.tgz
$ cd mod_bw
$ sudo /home/httpd/bin/apxs -i -a -c mod_bw.c
```

図 7.1　モジュールインストール

上記操作でコンパイルなどが実行され、モジュールが所定のディレクトリにコピーされ、httpd.conf にモジュールのロード設定が書き込まれます。

[*1] http://apache.ivn.cl/

7.3 設定

mod_evasive と同様に mod_bw に対する設定も httpd.conf に記述します。

リスト 19: mod_bw に対する設定

```
<IfModule mod_bw.c>
# mod_bw を利用する場合は必要
  BandWidthModule On
# すべてのリクエストに対し制限のチェックを強制的に実施
  ForceBandWidthModule On
# ローカル・ホストからの接続は無制限（0 を指定）にする
  BandWidth localhost 0
# 192.168.2.5 からの接続は 10240 バイト/秒（10K バイト/秒）に制限
  BandWidth 192.168.2.5 10240
# UserAgent が Mozilla/5 で始まるプラウザ（Firefox など）では 10K バイト/秒に制限
  BandWidth "u:^Mozilla/5(.*)" 10240
# 拡張子が.avi のファイルはファイル・サイズが 500K バイト以上なら、10K バイト/秒に制限
  LargeFileLimit .avi 500 10240
# ディレクトリ単位で設定
  <Location /foo>
# ネットワークアドレス単位で最大同時接続数制限を指定
    Bandwidth 192.168.0.0/24 10000
# 192.168.0.0/24 からの同時接続数を最大 10 に制限
    MaxConnection 192.168.0.0/24 10
  </Location>
# 1 クライアントだけなら 100K バイト/秒に
# 複数クライアントでは 50K バイト/秒に制限
  BandWidth    all 102400
  MinBandWidth all 50000
# アクセス制限時のメッセージ指定
  ErrorDocument 510 /..pathto../error.html
# ステータスコードを 510 に変更
  BanaWidthError 510
</IfModule>
# バーチャルホストに対する指定
<VirtualHost ...>
  <IfModule mod_bw.c>
    BandwidthModule On
    ForceBandWidthModule On
    Bandwidth all 1024000
  </IfModule>
    Servername www.example.jp
    ... 省略...
</Virtualhost>
```

上記設定の内容を見て分かるとおり、BandWidth、LimitFileLimit、MaxConnection ディレクティブが何度も利用されている。これらのディレクティブは次のような意味と書式で利用する。

BandWidth　クライアントごとの最大トラフィック量 (バイト/秒) を指定するディレクティブ。最初の引数でクライア

ントのホスト名、ドメイン名、IP アドレス、ネットワークアドレスを指定し、二番目の引数に最大トラフィック量をバイト/秒単位で指定する。0 は無制限を意味する。
ネットワークアドレスの指定は 192.168.1.0/24 または 192.168.1.0/255.255.255.0 のいずれかの形式で指定する。
all はすべてのクライアントを意味する。

リスト 20: BandWidth の書式

```
BandWidth 制限対象 最大トラフィック量
```

MinBandWidth　クライアントごとの最小トラフィック量を指定する。BandWidth ディレクティブであらかじめ対象の最大トラフィック量を指定しておく。

リスト 21: MinBandWidth の書式

```
MinBandWidth 制限対象 トラフィック量
```

LargeFileLimit　ファイル種別やサイズによってトラフィック量を制御するディレクティブ。最初の引数で対象ファイル種別を拡張子、二番目の引数でトラフィック量の制限サイズの最小値を、三番目の引数でトラフィック量を指定する。
ファイル種別は*.avi といった書式で指定する。

リスト 22: LargeFileLimit の書式

```
LargeFileLimit 種別 対象ファイルのサイズ トラフィック量
```

MaxConnection　クライアントごとに最大接続数を指定するディレクティブ。BandWidth ディレクティブであらかじめ対象の最大トラフィック量を指定しておく。

7.4 演習

1. mod_bw を導入する。
2. リスト 19 を参考にし、次に指定する制限を実施できるように設定し、それぞれの設定が正しく反映されているかをそれぞれの設定を行うごとにローカルと検証用の Windows PC から確認し、どんな設定をしたかを記録する。
   - 指定されたサーバからテキストファイル oui.txt と JPEG ファイルをコピーし、DocumentRoot のディレクトリにコピーする。
   - DocumentRoot のディレクトリに txt と imgs というディレクトリを作り、oui.txt をディレクトリ txt に、JPEG ファイルをディレクトリ imgs にコピーする。
   - DocumentRoot のディレクトリには制限はない
   - ディレクトリ txt には全ホストからのアクセスに対し、200K バイト/秒の帯域制限を実施する
   - ディレクトリ imgs には全ホストからのアクセスに対し、10K バイト/秒の帯域制限を実施する
   - 200K バイトを越える JPEG ファイルにのみ帯域制限を 200KB/秒にする

第 8 章

WAF(Web Application Firewall)

WAF とは Web Application Firewall の略で、アプリケーションレベルでクライアントからのデータを検証するファイアウォールとして動作し、一般的なファイアウォールや iptables では対処できない XSS[*1]やインジェクション攻撃に対応できる。

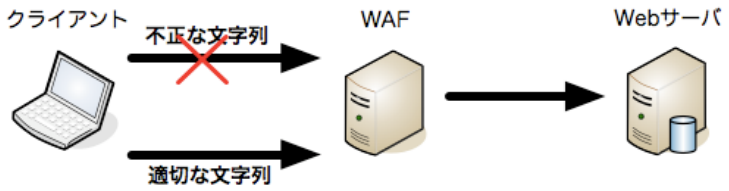

図 8.1　WAF のイメージ

WAF には様々な製品があるが、今回は無償で利用できる ModSecuriy[*2]を Apache に組込み、WAF の機能を体験する。mod_evasive や mod_bw と同様に Apache の DSO として動作する。

8.1　modSecurity の特徴

今回利用する modSecurity の主な機能は次の通り。

- HTTP リクエストがサーバで処理される前に、監査を実施できる
- フォームデータ、リクエストヘッダの中身など、監査対象を細かく設定できる
- 監査ルールに引っ掛かった際の動作を細かく設定できる
- 監査内容に正規表現を使用できる
- HTTP レスポンスに対して、監査を実施できる
- 監査ログを記録できる

[*1] Cross Site Scripting:クロスサイトスクリプティング
[*2] http://www.modsecurity.org/

8.2 インストール

ModSecurity を動作させるには modsecurity 以外に pcre ライブラリが必要で Apache を pcre 対応でビルドし直す必要がある。

8.2.1 pcre の導入

学内サーバ[*3]あるいは pcre のウェブサイト[*4]から pcre-8.00.tar.bz2 をダウンロードし、次の手順で導入する。

```
$ tar jxfv pcre-8.00.tar.bz2
$ cd pcre-8.00
$ ./configure
$ make
$ sudo make install
```

図 8.2 pcre の導入手順

8.2.2 Apache を pcre 対応でビルドし直す

Apache を pcre 対応でビルドし直す。

```
$ cd httpd-2.2.14
$ ./configure --prefix=/home/httpd \
    --with-pcre=/usr/local/bin/pcre-config
$ make
$ su
# make install
# ln -s /home/httpd/bin/apr-1-config /usr/local/bin/apr-config
# ln -s /home/httpd/bin/apu-1-config /usr/local/bin/apu-config
# exit
$ cd ..
```

図 8.3 Apache の再ビルド

[*3] http://172.16.32.10/~sakabe/
[*4] http://www.pcre.org/

8.2.3 modsecurity の導入

modsecurity[*5]をダウンロードし、次の手順 (図 8.4) で導入する。make test の実行結果で必ず All tests passed (540). と表示される事を確認する事。

```
$ tar zxfv modsecurity-apache_2.5.11.tar.gz
$ cd modsecurity-apache_2.5.11/apache2
$ ./configure --with-apxs=/home/httpd/bin/apxs
$ make
$ make test
...
Passed: 17; Failed:  0

All tests passed (540).
$ sudo make install
```

図 8.4 modsecurity のビルド

8.2.4 mod_unique のインストール

modsecurity を動作させるには mod_unique が必要で次の手順 (図 8.5) で導入する。

```
$ cd ../../httpd-2.2.14/modules/metadata
$ su
# /home/httpd/bin/apxs -i -a -c mod_unique_id.c
# exit
```

図 8.5 mod_unique の導入

8.2.5 httpd.conf の編集

modSecurity を使うにはモジュールのロード指定と動作設定を httpd.conf に追加する。設定例をリスト 23 に示す。なお、405 行目近辺の SSL の設定ファイルの読み込みはコメントにする。

リスト 23: modSecurity 対応の httpd.conf

[*5] http://www.modsecurity.org/download/direct.html

```
# 56 行目付近に次の行を追加
LoadModule security2_module   modules/mod_security2.so

# 以下の各行をファイルの末尾に追加
<IfModule mod_security2.c>
# mod_security2 の基本設定
  SecRuleEngine On
  SecRequestBodyAccess On
  SecResponseBodyAccess Off
# デフォルトアクションの設定
  SecDefaultAction phase:2,log,auditlog,deny
# デバッグログのファイルの指定
  SecDebugLog logs/modsec_debug.log
  SecDebugLogLevel 3
# 監査ログの指定
  SecAuditEngine RelevantOnly
  SecAuditLog logs/modsec_audit.log
# 監査ルールの設定
  SecRule REMOTE_ADDR "192\.168\.221\.[0-9]{1,3}$" "log,deny"
  SecRule ARGS_GET "attack" "log,deny"
  SecRule ARGS_POST "evil" "log,deny"
  SecRule ARGS "(\"|>|<|'|script|onerror)" "log,deny"
  SecRule ARGS "foo" "log,pass"
</IfModule>
```

SecRequestBodyAccess　サーバへの要求を監査するかの指定

SecResponseBodyAccess　サーバからの応答を監査するかの指定

SecDeafultAction　デフォルトの動作指定。phase は次の 5 つがある。要求ヘッダ処理時、要求ボディ処理時、応答ヘッダ処理時、応答ボディ処理時、ログ処理時

SecDebugLog　デバッグログファイルの指定

SecDebugLogLevel　デバッグログの記録レベル

SecAuditEngine　監査ログの記録方法の指定。RelevantOnly はフィルタにマッチしたときのみ記録する。

SecAuditLog　監査ログファイルの指定。

SecRule　監査ルールの指定

SecRule の書式は次の通り。

リスト 24: SecRule の書式

```
SecRule 変数 オペレータ [アクション]
```

REMOTE_ADDR　クライアントのアドレス

ARGS_GET　GET メソッドに含まれる文字列

ARGS_POST　POST メソッドに含まれる文字列

ARGS　GET または POST メソッドに含まれる文字列

この例では log,deny アクションは記録を行い、そのアクセスを拒否し、log,pass アクションは記録し、そのアクセスを通過させます。

入力を終えたら誤りがないかをチェックし、誤りがないことを確認できたら今回のモジュールがロードされるかを確認する。

```
# /home/httpd/bin/httpd -t
Syntax OK
# /home/httpd/bin/httpd -M
Loaded Modules:
....
....
unique_id_module (shared)
security2_module (shared)
```

図 8.6 httpd.conf の文法チェック

8.3 動作検証

httpd.conf を正しく設定できたら、次の HTML ファイル、Perl スクリプト、.htaccess ファイルを htdocs のディレクトリに置き、動作を確認する。test.pl のパーミッションは 755 に変更し、httpd.conf で DocumentRoot の Options ディレクティブにに ExecCGI を追加する。

リスト 25: テスト用フォーム (testform.html)

```
<html>
<head>
<title>modSecurity test</title>
</head>
<body>
<h1>modSecurity</h1>

<h2>GET test</h2>
<form action="test.pl" method="GET">
<input type="text" name="txt">
<input type="submit" name="submit">
<input type="reset">
</form>

<h2>POST test</h2>
<form action="test.pl" method="POST">
<input type="text" name="txt">
<input type="submit" name="submit">
<input type="reset">
</form>
</body>
</html>
```

二つあるフォームのうち上が GET、下が POST メソッドで送信します。

リスト 26: テスト用フォーム処理スクリプト (test.pl)

```
#!/usr/bin/perl -w
print "content-type: text/html\n\n";
print "<h1>Result</h1>";

my @param = "";
my $data = "";

if($ENV{'REQUEST_METHOD'} eq 'GET'){
  $data = $ENV{'QUERY_STRING'};
}

if($ENV{'REQUEST_METHOD'} eq 'POST'){
  read (STDIN, $data, $ENV{'CONTENT_LENGTH'});
}

@param = split(/&/,$data);
@head = split(/=/,$param[0]);
print $head[1];
```

リスト 27: .htaccess

```
Options +ExecCGI
```

監査ルールに引っ掛からない適切な文字列の場合はその文字列が、不適切な文字列のときは Forbidden が表示されるます。

図 8.7　フォーム

図 8.8　適正な文字列の場合

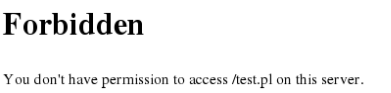

図 8.9　不適切な文字列の場合の表示

8.4　modSecurity の提供するルールの適用

modSecurity は細かく文字列を指定できる分、複雑です。modSecurity ではルールを記述したファイルが配布されているのでそれを適用する方法を示します。modSecurity のホームページからダウンロードしたルールファイル modsecurity-core-rules_2.5-1.6.1.tar.gz を展開し、/home/httpd/conf に置きます。設定ファイル httpd.conf に次のような行を追加することでルールを適用できます。

リスト 28: ルールファイルの適用

```
<IfModule mod_security2.c>
  Include conf/mod_sec_conf/modsecurity_crs_20_protocol_violations.conf
</IfModule>
```

8.5 ログファイルの確認

ModSecurity が正しく動作しているかログファイル (/www/logs/error_log) を確認する。その出力例を示す。

リスト 29: ModSecurity のログの例

```
[Tue Dec 02 16:57:49 2008] [notice] ModSecurity for Apache/2.5.7 \
    (http://www.modsecurity.org/) configured.
[Tue Dec 02 17:47:14 2008] [error] [client 192.168.221.1] ModSecurity: \
    Access denied with code 403 (phase 2). Pattern match \
    "192\\.168\\.221\\.[0-9]{1,3}$" at REMOTE_ADDR. [file \
    "/home/httpd/conf/httpd.conf"] [line "515"] [hostname \
    "192.168.221.128"] [uri "/"] [unique_id "STT2EsCo3YAAAC9EDrUAAAAA"]
[Wed Dec 03 10:44:51 2008] [error] [client 127.0.0.1] ModSecurity: Access \
    denied with code 403 (phase 2). Pattern match "atack" at ARGS_GET:txt. \
    [file "/home/httpd/conf/httpd.conf"] [line "516"] [hostname \
    "localhost"] [uri "/test.pl"] [unique_id "STXkk8Co3YAAAC
9IFzYAAAAE"]
```

8.6 演習

1. modSecurity を導入し、指定してルール通り動作することを確認する。http://localhost/testform.html でアクセスする。
2. 検証用 PC からのアクセスを拒否する設定を追加する
3. Internet Explorer からのアクセスを拒否する設定を追加する (User-agenet 文字列に MSIE が含まれている)

第 9 章

Apache の動作状況とログ解析

Apache の動作状況を確認するモジュール mod_info と mod_staus を使って Apache の動作状況を確認する。

9.1 mod_info

サーバの設定の外観を提供するモジュールで configure 時に明示的に指定しないと利用できない。

9.1.1 ビルド

```
$ cd Apache のディレクトリ
$ ./configure --prefix=/home/httpd \
              --with-pcre=/usr/local/bin/pcre-config \
              --enable-info
$ make
$ sudo make install
```

図 9.1 Apache で mod_info を有効にする

9.1.2 設定

mod_info を利用するには httpd.conf で mod_info と mod_status 用の設定ファイルを読み込むように設定しておく。

リスト 30: httpd.conf の変更箇所

```
#Include conf/extra/httpd-info.conf
   ↓
Include conf/extra/httpd-info.conf
```

さらに、httpd-info.conf 内のアクセスを許可するホスト指定を.example.com から localhost.localdomain に変更する。

リスト 31: httpd-info.conf の変更箇所

```
Allow from .example.com
   ↓
Allow from localhost.localdomain
```

9.1.3 動作確認

ブラウザで http://localhost/server-info にアクセスすると情報が表示される。

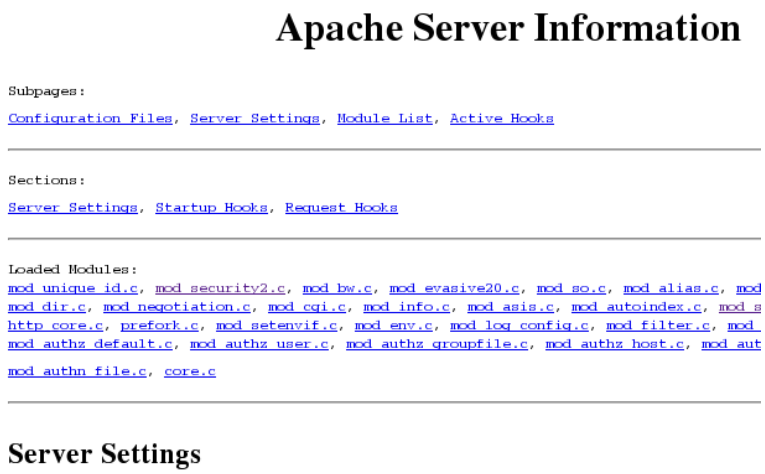

図 9.2 server-info

9.2 mod_status

mod_status はサーバの活動状況と性能に関する情報を提供するモジュールで Apache をビルドする時点で最初から組み込まれているが設定をロードしないと利用できない。ブラウザで http://localhost/server-status にアクセスすると情報が表示される。

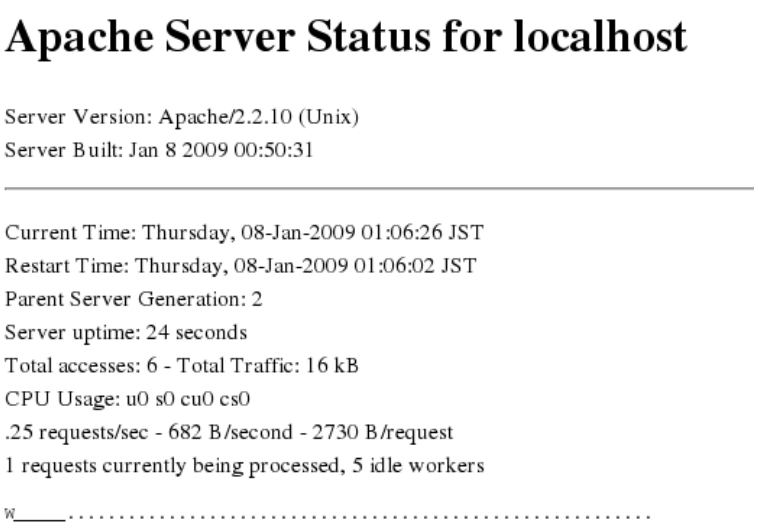

図 9.3 server-status

URL に更新間隔などを指定できる。

- URL を http://localhost/server-status?refresh=N にすると N 秒間隔で情報がリフレッシュされる。
- URL を http://localhost/server-status?auto にすると機械読み取り可能な形式になる。

```
Total Accesses: 20
Total kBytes: 24
Uptime: 167
ReqPerSec: .11976
BytesPerSec: 147.162
BytesPerReq: 1228.8
BusyWorkers: 1
IdleWorkers: 5
Scoreboard: ___W__..................
```

図 9.4 機械読み取り可能な server-status

9.3 演習

http://httpd.apache.org/docs/2.2/ja/mod/mod_info.html や http://httpd.apache.org/docs/2.2/ja/mod/mod_status.html を参考にし、各自の Apache の設定情報などの表示方法を調べ、実際に試してみる。

第 10 章

Apache のログ解析

Webalizer[*1]は Apache のログファイルを集計・分析するプログラムで、Apache のログファイルを定期的に集計・解析し、その結果は Web ブラウザで表示します。Webalizer の生成したファイルの表示例を図 10.1 に示す。

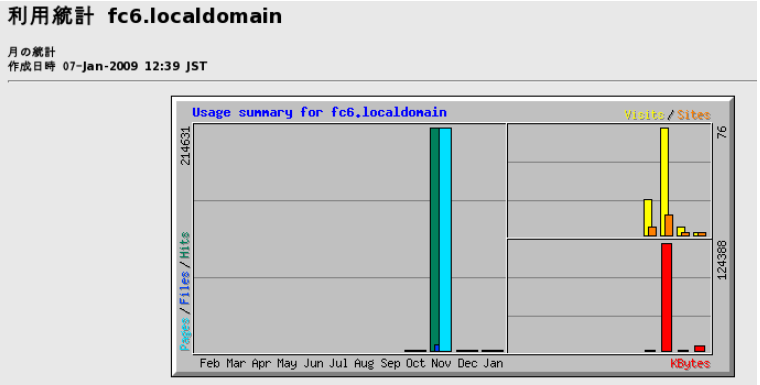

図 10.1　Webalizer

10.1　ビルド・インストール

Webalizer の Web サイトからソースファイルを取得し、ビルドします。学内サーバ (http://172.16.32.10/~sakabe/) にもファイルを置いてあります。Webalizer で日本語表示するように指定しておきます。

```
$ tar jxfv webalizer-2.21-02-src.tar.bz2
$ cd webalizer-2.21-02
$ ./configure --with-language=japanese
$ make
$ sudo make install
```

図 10.2　Webalizer のビルド手順

[*1] http://www.webalizer.org/

インストールが完了すると実行ファイルは/usr/local/bin に、設定ファイルは/usr/local/etc にそれぞれインストールされます。設定ファイルは/etc/webalizer.conf で、Web サーバと一緒に導入されたものがすでに存在します。

10.2 設定ファイルの編集

設定ファイル/etc/webalizer.conf は今回の実習で導入した Apache 向けの設定ではないのでそれを変更します。

```
# cp /usr/local/etc/webalizer.conf.sample /usr/local/etc/webalizer.conf
# vim /usr/local/etc/webalizer.conf
```

図 10.3 設定ファイルの編集

表 10.1 /usr/local/etc/webalizer の設定項目

設定項目	設定値
LogFile	/www/logs/access_log
OutputDir	/www/usage

10.3 httpd.conf の変更

Webalizer の出力を Apache からアクセスできるように設定を追加し、サーバを再起動しておく。

リスト 32: httpd.conf への追加設定

```
Alias /usage /www/usage

<Location /usage>
  Order deny,allow
  Deny from all
  Allow from localhost.localdomain
</Location>
```

10.4 cron で実行されるファイルを変更する

webalizer は cron によって毎日定期的に実行されるよう設定されている。実行されるシェルスクリプトは/etc/cron.daily/00webalizer で、その内容は次のように修正しておく。

リスト 33: /etc/cron.daily/00webalizer

```
#! /bin/bash
# update access statistics for the web site

if [ -s /www/logs/access_log ]; then
   exec /usr/local/bin/webalizer -Q
fi
```

10.5 手動で更新

cron を使って定期的にアクセスログを集計するが、その時間は毎日午前 4 時 2 分のため、現在は集計結果がないので手動でアクセスログを集計しておく。

```
# mkdir /www/usage
# webalizer
Webalizer V2.21-02 (Linux 2.6.18-164.6.1.el5 i686) Japanese
Using logfile /www/logs/access_log (clf)
Creating output in /www/usage
Hostname for reports is 'beta.localdomain'
Reading history file... webalizer.hist
Generating report for October 2009
Generating report for November 2009
Generating report for December 2009
Saving history information...
Generating summary report
60110 records in 1 seconds, 60110/sec
```

図 10.4 手動でアクセスログを集計

アクセスログの集計が完了したらブラウザを開き、http://localhost/usage/にアクセスすれば、図 10.1 のような集計結果が表示されるはずである。

10.6 演習

Webalizer を動作させ、そこから何が分かるかを確認する。

第 11 章

SNMP

今回は Apache ではなく、ネットワーク全体のトラフィックを監視し、視覚化する方法を学ぶ。ネットワークの監視には SNMP[*1]を使い、視覚化には MRTG[*2]を使う。

11.1 SNMP

SNMP は UDP で動作し、管理用の機器のマネージャと管理対象の機器のエージェントに分かれている。マネージャとエージェント間のやりとりが SNMP で、エージェントは MIB[*3]とよぶ管理情報データベースの参照および値の設定により、機器の状態を得られる。

MIB はツリー状のデータで数字列を使い、表現される。標準 MIB とメーカ独自の拡張 MIB がある。

11.2 SNMP のインストール

SNMP の実装例として net-snmp がある。yum でインストールする。

```
$ sudo yum -y install net-snmp net-snmp-utils
```

11.3 net-snmp の設定

net-snmp の設定ファイルは/etc/snmp/snmp.conf で、まず次の 4 項目を設定する。

表 11.1 net-snmp の設定

項目	内容
com2sec	セキュリティ名の定義
group	グループ名
view	取得を許可する情報の範囲指定
access	グループに対するアクセス権設定

[*1] Simple Network Managemento Protocol
[*2] Multi Router Traffic Grapher
[*3] Management Information Base、ミブ

11.3.1 com2sec

com2sec は SNMP 要求の送信元ネットワークとコミュニティ名の組み合わせでセキュリティ名を定義する。コミュニティ名は情報にアクセスするためのパスワードに当たる。SNMP 機器ではコミュニティ名のデフォルト値を持ち、多くは読み取り専用の public、読み書き可能な private が設定される。ここではリスト 34 のように 2 つのセキュリティ名 local と securenet を設定する。

リスト 34: com2sec 設定

```
#         sec.name         source           community
#com2sec notConfigUser    default          public
com2sec   local            localhost        localcom
com2sec   securenet        172.16.12.0/24   securcom
```

11.3.2 group

セキュリティ名と SNMP のセキュリティモデルの二つの値の組み合わせでグループを定義する。ここでの定義は access 行でのアクセス許可の設定で使用する。セキュリティモデルの v1、v2、usm は SNMP のバージョン 1、2、3 に対応する。最初の 2 行をコメントにし、local と securenet のセキュリティ名を local_group と secure_gruop というグループ名を定義します。

リスト 35: group 設定

```
#        groupName        securityModel securityName
#group   notConfigGroup   v1            notConfigUser
#group   notConfigGroup   v2c           notConfigUser
group    local_group      v1            local
group    local_group      v2c           local
group    local_group      usm           local
group    secure_group     v1            securenet
group    secure_group     v2c           securenet
group    secure_group     usm           securenet
```

11.3.3 view 設定

view では MIB ツリーの特定範囲を view 名で定義する。

リスト 36: view 設定

```
#        name          incl/excl  subtree          mask(optional)
#view    systemview    included   .1.3.6.1.2.1.1
#view    systemview    included   .1.3.6.1.2.1.25.1.1
view     view_all      included   .1
view     view_mib2     included   .1.3.6.1.2.1
view     view_ucdavis  included   .1.3.6.1.4.1.2021
```

11.3.4 access 設定

access ではアクセス許可設定で、ここまでで定義した group 名、view 名を使い、グループに対してどんなアクセスを認めるかを設定する。

リスト 37: access 設定

```
# group context sec.model sec.level prefix read write notif
#access notConfigGroup "" any noauth exact systemview none none
access local_group "" any noauth exact view_all none none
access secure_group "" any noauth exact view_mib2 none none
```

11.3.5 その他設定

httpd のプロセス監視のために次の行を追加する。

リスト 38: その他の設定

```
proc httpd 10 1
```

11.4 MIB 情報の検索

snmpwalk コマンドで mib ツリーの情報を取得できる。

```
# snmpwalk -v1 -c localcom localhost .1.3.6.1.2.1|head
SNMPv2-MIB::sysDescr.0 = STRING: Linux beta.localdomain \
    2.6.18-164.6.1.el5 #1 SMP Tue Nov 3 16:18:27 EST 2009 i686
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (29436) 0:04:54.36
SNMPv2-MIB::sysContact.0 = STRING: Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
SNMPv2-MIB::sysName.0 = STRING: beta.localdomain
SNMPv2-MIB::sysLocation.0 = STRING: SNMP Test Server(CentOS 5.4)
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (9) 0:00:00.09
SNMPv2-MIB::sysORID.1 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.2 = OID: TCP-MIB::tcpMIB
SNMPv2-MIB::sysORID.3 = OID: IP-MIB::ip
```

図 11.1 snmpwalk での情報取得

第 12 章

MRTG

12.1 MRTG とは

MRTG とは SNMP エージェントから取得したデータを加工し視覚化するツールです。監視対象の機器に対して SNMP リクエスト (iso(1).org(3).dod(6).internet(1).mgmt(2).mib2(1).intreface(2).ifTable(2).ifEntry(1) ifInOctets(10).I/F ポート (1) および iso(1).org(3).dod(6).internet(1).mgmt(2).mib2(1).intreface(2).ifTable(2).ifEntry(1) ifOutOctets(10).I/F ポート (1)) を送信し、取得したデータを PNG 画像にします。監視対象のシステムから収集したすべてのデータは過去 2 年間分保持され、過去 1 日間、7 日間、4 週間、12 カ月間のトラフィックのグラフを生成するのに利用します。

12.2 MRTG のインストール

mrtg も yum を使ってインストールします。

```
$ sudo yum -y install mrtg
```

図 12.1 MRTG のインストール

12.3 設定ファイルの作成

MRTG には設定ファイル生成ツール cfgmaker コマンドを使う。書式は次の通り

 # cfgmaker <オプション> コミュニティ名@ホスト名

MRTG 用ディレクトリ/www/htdocs/mrtg を作成し、MRTG の設定ファイルを/etc/mrtg/mrtg_test.cfg に作成します。

```
# mkdir /www/htdocs/mrtg
# cfgmaker --output=/etc/mrtg/mrtg_test.cfg localcom@127.0.0.1 securecom@172.16.114.131
```

図 12.2 MRTG 設定ファイル生成

設定ファイルが生成されたら冒頭部分を次のように修正します。この設定のディレクトリがないときは作成しておく。

リスト 39: MRTG 設定ファイルの修正

```
### Global Config Options

#  for UNIX
# WorkDir: /home/http/mrtg
Language: eucjp
HtmlDir: /www/htdocs/mrtg
ImageDir: /www/htdocs/images
IconDir: /icons
LogDIr: /www/logs
```

12.4 MRTG 動作確認

以下のコマンドを三回実行する。最初の二回はメッセージが表示されるが問題ない。

```
env LANG=C mrtg /etc/mrtg/mrtg_test.cfg
```

コマンドの実行を終えたら http://localhost/mrtg/ターゲット名.html をブラウザで開く。

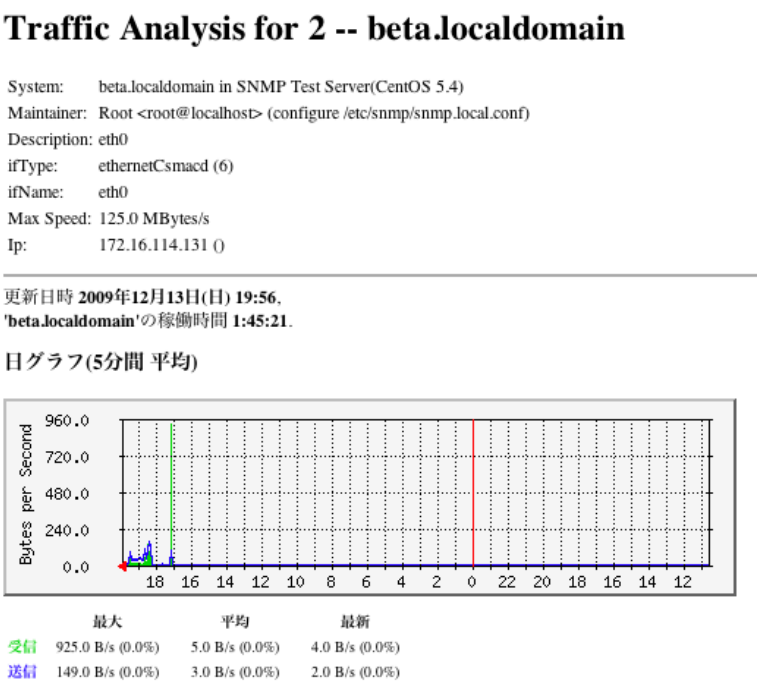

図 12.3 MRTG の表示例

12.5 定期的に MRTG を実行

MRTG を定期的に実行するには cron を使用する。root で crontab -e コマンドを実行し、次の一行を最後の行に追加する。

リスト 40: MRTG の定期的実行

```
*/5 * * * * /usr/bin/mrtg /etc/mrtg/mrtg_test.cfg
```

12.6 ディスク監視

snmp.conf の disk / 10000 の行のコメントを外し、下記の設定を mrtg_test.cfg に追加。SNMP を再起動しておく。確認は http://localhost/mrtg/disk.html。

リスト 41: ディスクの監視

```
### Disk Used ###
#
Target[disk]: .1.3.6.1.4.1.2021.9.1.9.1&.1.3.6.1.4.1.2021.9.1.9.1:localcom@127.0.0.1
MaxBytes[disk]: 100
Unscaled[disk]: dwmy
Options[disk]: gauge, absolute, growright, nopercent, noinfo
YLegend[disk]: Disk Used(%)
ShortLegend[disk]: (%)
LegendI[disk]: / Disk used
```

```
LegendO[disk]: / Disk Used
Legend1[disk]: / Disk used
Legend2[disk]: / Disk used
Title[disk]: Disk Used
PageTop[disk]:
```

12.7 cpu 使用率

mrtg_test.cfg に次の設定を追加。確認は http://localhost/mrtg/cpu.html

リスト 42: CPU の監視

```
### CPU Usage ###
Target[cpu]: .1.3.6.1.4.1.2021.11.50.0&.1.3.6.1.4.1.2021.11.52.0:localcom@127.0.0.1
MaxBytes[cpu]: 100
#Unscaled[cpu]: dwmy # 縦軸を 100% にしないで実績値に合わせる（100% にするとグラフがみえないため）
Options[cpu]: growright, noinfo, nopercent
YLegend[cpu]: CPU usage(%)
ShortLegend[cpu]: (%)
LegendI[cpu]: ユーザ
LegendO[cpu]: システム
Legend1[cpu]: CPU 使用率（ユーザ）(%)
Legend2[cpu]: CPU 使用率（システム）(%)
Title[cpu]: ＣＰＵ使用率
PageTop[cpu]:
```

12.8 メモリ使用の監視

まず、free コマンドでメモリとスワップの最大値を取得。

```
# free
total used free shared buffers cached
Mem: 515296 491140 24156 0 125604 188592
-/+ buffers/cache:     176944     338352
Swap:      1048568      56428     992140
```

図 12.4 free コマンド

次の設定を mrtf_test.cfg に追加する。確認は http://localhost/mrtg/mem.html。

リスト 43: メモリの監視

```
### Memory Free ####
Target[mem]: .1.3.6.1.4.1.2021.4.6.0&.1.3.6.1.4.1.2021.4.4.0:localcom@127.0.0.1
MaxBytes1[mem]: 515296  ← 物理メモリ MAX 値を指定する
MaxBytes2[mem]: 1048568  ← スワップメモリ MAX 値を指定する
```

```
Unscaled[mem]: dwmy
Options[mem]: gauge, absolute, growright, noinfo
YLegend[mem]: Mem Free(Bytes)
ShortLegend[mem]: Bytes
kilo[mem]:1024
kMG[mem]: k,M,G,T,P
LegendI[mem]: Real
LegendO[mem]: Swap
Legend1[mem]: 空き物理メモリ [MBytes]
Legend2[mem]: 空きスワップメモリ [MBytes]
Title[mem]: 空きメモリ量
PageTop[mem]:
```

12.9 演習

SNMP と MRTG を動作させ、ネットワークの状態監視を実現する。