安全な CVS サーバの構築パッケージの開発

Secure CVS Server Setup

田中 哲 [1)]　上野 乃毅 [2)]　林 芳樹 [3)]
Akira TANAKA　Daiki UENO　Yoshiki HAYASHI

1) 産業技術総合研究所 情報処理部門
2) 早稲田大学
3) 東京大学

ABSTRACT. The administration of CVS server securely is hard task. Because CVS is not designed for distributed environment. The administration includes the setup of chroot/jail environment, CVS only user administration, etc. It is too difficult to usual administrators. So the distributed development environment using CVS is hard to provide, We developed a package to ease the installation and maintenance.

1 背景

オープンソースによる開発などでは、地理的に分散した開発者が共同して開発を進める状況が一般的であり、そのような開発を支援するツールが求められている。

このようなシステムとして、現在、CVS[1] というソフトウェアが de facto standard として用いられている。しかし、CVS はもともと 1980 年代に設計され、一つのサーバに開発者がログインして開発を行なうという前提で設計されている。

このため、分散環境で安全に利用するためには、安全な (暗号化を行なう) 通信路の設定、開発者にサーバ上で必要以上の権限を与えないようにする設定など、さまざまな設定が必要になる。管理者はこれらを正しく行なう設定を考案し、実現しなければならない。前述の通り、CVS は分散環境のためのツールではないので、それを無理矢理分散環境で安全に使うためにはかなり詳しい知識が必要になる。これは通常の管理者には難しく、可能であるにしてもかなり手間がかかる。また、複雑な設定が必要なため、間違いが入りやすく、安全性を確信することも困難である。

2 目的

CVS サーバの管理負担は CVS による分散開発環境の普及を防げており、分散開発環境を前提とするオープンソースプロジェクトの発展を防げている。我々はこの問題意識により、安全な CVS サーバを容易に設定できるシステムの構築を行なった。

3 安全な CVS サーバの設計

安全な CVS サーバを容易に設定できるシステムを既存の経験をもとに設計を行なった。具体的には `cvs.m17n.org` の運用経験において管理が困難な点を解決できるよう次の点を目標とした。

- 試行錯誤の繰り返しによってインストールが行なわれたが、これを単純作業で済ませるようにする。
- 設定を変更するのにさまざまなファイルを編集しなければならないが、これを一つのファイルだけで済ませるようにする。
- 長期間の運用により、さまざまなファイルに変更を加えていると、システムの全体像が把握しにくくなり、システムを修正しづらくなる。そこで、全体像を容易に把握できるようにする。

なお、経験の元になった `cvs.m17n.org` は次のソフトウェアなどの開発に使われている。

- APEL: Emacs Lisp のポータビリティを実現するライブラリ (2Mbytes)
- FLIM: Emacs 上の Internet message ライブラリ (6Mbytes)
- SEMI: Emacs 上の MIME ユーザインターフェースライブラリ (4Mbyts)
- Semi-gnus: SEMI を使うように Gnus から派生したニュースリーダ (50Mbytes)
- Liece: irchat から派生した IRC クライアント (1Mbytes)
- Wanderlust: IMAP4rev1 対応のメール/ニュース管理システム (12Mbytes)
- gcc の SH サポート (674Mbytes)

4 CVS サーバの安全性

立場により安全性にはいくつかの種類がある。

管理者にとっての安全性　CVS サーバの管理者にとって重要なことは、サーバが破壊される可能性をなくし、CVS というサービスを持続的に提供可能とすることである。また、サーバを他のマシンへの攻撃の踏台として使われることも防がなければならない。

開発者にとっての安全性　開発者 (CVS サーバを使用してソフトウェアの開発を行なう人) にとって重要なことは、リポジトリ (CVS サーバで保持しているデータ) が失われないことである。

ここで、開発者にとって重要なデータを失わないためにはバックアップにより対策する。また、管理者にとって重要なサーバの破壊や踏台になることの防止は開発者に必要以上の権限を与えないことによって対策する。具体的には、CVS サーバを特殊な環境下で動作させ、不要な可能性を排除することを行なう。

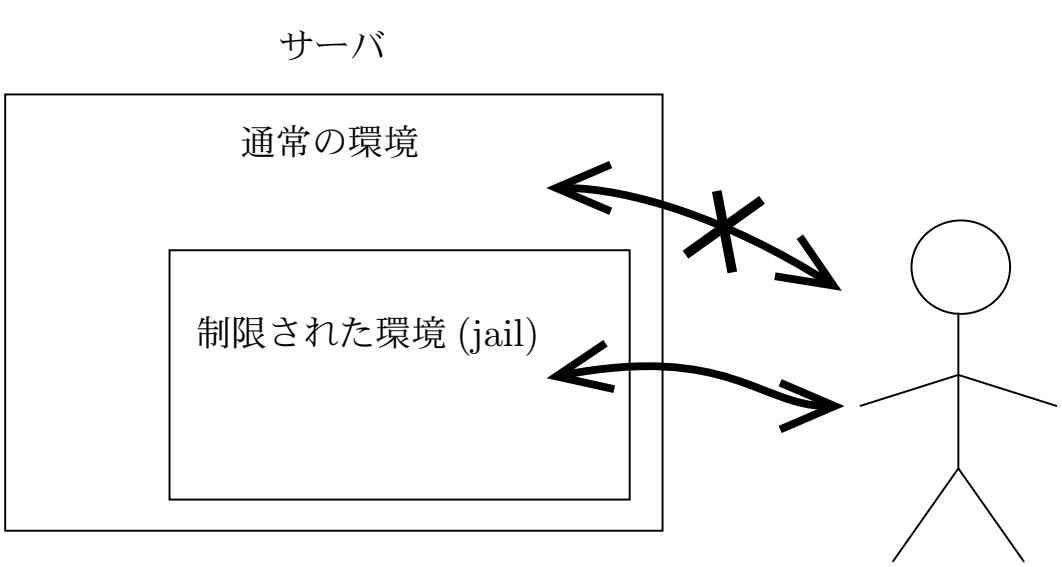

図 1: サーバ内に jail を作る

5 セキュリティモデル

ここでは CVS と 分散共同開発環境のセキュリティモデルについて述べる。

- CVS が前提とするセキュリティモデルは、CVS を起動したユーザができることは何をやってもセキュリティ問題にはならない、というものである。言い替えると、CVS が動作する OS のセキュリティモデルをそのまま利用しているということである。
- これに対し、近年の分散共同開発環境は異なるセキュリティモデルを要求する。一般に、Web などの分散アプリケーションがユーザに提供するものは Web ページだけであるのと同様に、分散共同開発環境でも、開発している対象のデータだけにアクセスを許すというのが自然なセキュリティモデルである。

しかし、通常のマシンに素朴に CVS をインストールし、共同開発を行なう場合、CVS を使う以上、CVS のセキュリティモデルを採用しなければならない。つまり、分散共同開発環境として望ましいセキュリティモデルを採用することはできない。

ここで、CVS のセキュリティモデルは OS のセキュリティモデルそのものであるから、OS のセキュリティモデルを分散共同開発環境に適したセキュリティモデルに設定すれば、適切なセキュリティモデルで CVS による分散共同開発環境を提供できる。われわれのパッケージはこのような設定を簡単に行なえるようにする。

6 最小権限で CVS サーバを動作させる

我々のパッケージは分散共同開発環境が要求するセキュリティモデルを OS のセキュリティモデルで実現するため、jail (ないしは chroot) という機構を利用する。

jail はプロセスを通常とは異なる環境で動作させるための機構であり、jail によって図 1 のようにサーバ内に特殊な環境を用意し、外部の人間がその環境内としか通信できないように設定することができる。

ここで、この特殊な環境を分散共同開発環境が要求するセキュリティモデルになるよう設定すれば安全な CVS サーバとなるわけである。我々は jail を次のように設定することでこれを実現した。

- jail 内には最小限のファイルのみを置く。とくに、setuid された実行ファイルは置かない。
- 開発対象以外の jail 内のファイルは整合性の検査を行なう。
- jail 内のファイル配置は自動的に行なう。

jail 内に置くファイルを最小限に抑えることにより、開発者に許す不要な可能性を最小限に抑え、ファイルの整合性を検査することにより、jail の破壊を可能な限り早期に検出し、ファイルの配置を自動的に行なうことにより、初期状態における間違いを減らすわけである。

7 `cvs-setup` の使いかた

7.1 インストール

我々のパッケージ (`cvs-setup`) による CVS サーバのインストールは図 2 ように単純である。

ここで、wget までの前半部分はパッケージの入手段階であり、それ以降の作業は configure, make, make install という一般的なやりかたを踏襲している。このことにより、管理者は混乱せずにインストールすることができる。

なお、事前に Objective Caml[5], Ruby[6], GNU make, autoconf をインストールしておく必要がある。ここで、Objective Caml が必要なのはこれは jail 内で動作するツールのうち、我々が新たに作成したもの (/bin/sh や ユーザのシェルなど) の記述に使用したためである。Objective Caml は、C などとは異なり配列の範囲検査が行なわれ、ポインタを直接扱えないなどの性質により buffer overrun が起きず[*1]、また、静的に型付けされていてコンパイル時に多くのバグが見つかる安全な言語である。このため、新たなセキュリティ問題を導入する可能性を最低限に抑えている。

7.2 リポジトリの生成

また、実際にリポジトリを作るには、図 3 のように、設定のテンプレートをコピーし、具体的な設定を行ない、設定を反映させることを行なう。

`cvs-setup` ではこの `site.rb` というファイルだけによって設定を行なう。従来の CVS がさまざまなファイル・ディレクトリの設定を個々に管理者が行なわなければならなかったのに対し、`cvs-setup` ではひとつのファイルのみ

[*1] むろん言語処理系に問題がある場合はこの限りではないが、その可能性は C で記述したアプリケーションに問題が作りこまれる可能性よりかなり低い。

```
% cvs -d :pserver:anonymous@cvs.cvs-security.sf.net:/cvsroot/cvs-security \
 login
% cvs -d :pserver:anonymous@cvs.cvs-security.sf.net:/cvsroot/cvs-security \
 co cvs-setup
% cd cvs-setup
% wget \
    http://prdownloads.sourceforge.net/cvs-nserver/cvs-nserver-1.11.1.3.tar.gz \
    ftp://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-3.0.2p1.tar.gz \
    http://www.openssl.org/source/openssl-0.9.6c.tar.gz \
    ftp://ftp.cs.mun.ca/pub/pdksh/pdksh-5.2.14.tar.gz \
    http://www.xinetd.org/xinetd-2.3.3.tar.gz \
    ftp://ftp.info-zip.org/pub/infozip/zlib/zlib-1.1.3.tar.gz
% autoconf
% ./configure \
    --with-cvs-nserver-src=cvs-nserver-1.11.1.3.tar.gz \
    --with-openssh-src=openssh-3.0.2p1.tar.gz \
    --with-openssl-src=openssl-0.9.6c.tar.gz \
    --with-pdksh-src=pdksh-5.2.14.tar.gz \
    --with-xinetd-src=xinetd-2.3.3.tar.gz \
    --with-zlib-src=zlib-1.1.3.tar.gz \
    --prefix=PREFIX
% gmake
% su
# gmake install
```

図 2: cvs-setup のインストール法

```
# cd PREFIX
# cp cvs-admin/share/cvs-setup/site.rb.dist cvs-admin/share/cvs-setup/site.rb
# vi cvs-admin/share/cvs-setup/site.rb
# cvs-admin/bin/cvs-setup install
```

図 3: cvs-setup でのリポジトリの生成

で全体の設定ができる。このことは全体の設定の見通しを良くし、設定の失敗を減らすことになる。

また、cvs-setup では管理者は本質的な設定以外は行なう必要がない。例えば、開発者を一人追加する場合、passwd ファイル, group ファイル、ホームディレクトリ、SSH の設定など、さまざまなファイルを設定しなければならない。cvs-setup では、ユーザ名などのいくつかの本質的な情報からそれらのファイルを適切に生成する。このように cvs-setup により、管理者は管理の手間を減らすことができる。

また、cvs-setup は Unix 上に分散環境に適したセキュリティモデルを実現する。これにより、単純に動くというだけではなく、安全な運用が可能なシステムを構築する。

さらに、この作業中には必要最小限のデバイスファイルや共有ライブラリを選択するなど、専門的な知識が要求される作業を行なう。通常の管理者にとってこれらは往々にして困難な作業であるが、cvs-setup により、専門的な知識無しに容易に達成できるようになる。

7.3 サーバの起動

サーバの起動は図 4 のようにして行なう。これにより、図 5 のように sshd および xinetd が起動し、ssh 経由の ext および pserver によるサービスが開始される。

この start-stop-script は System V 系の Unix から広まった daemon の起動・終了に使われる一般的なインターフェースを持っており、管理者は新しいインターフェースを学ぶ必要もなく容易に起動することができる。

7.4 設定の変更

設定の変更は次のようにして行なう。

```
# vi cvs-admin/share/cvs-setup/site.rb
# cvs-admin/bin/cvs-setup install
```

7.2 節で述べた通り、設定は site.rb だけを編集し、その設定を反映させるために cvs-setup install を実行するだけである。ここで、その実行の前にどのような変更が行なわれるかを確認するためには次のようにする。

```
# cvs-admin/bin/cvs-setup
```

このように引数をつけずに cvs-setup を実行することにより、どのファイルをどのように書き換えるかを事前に表示し、確認することができる。

7.5 整合性検査

予期しない jail 内のファイルの破壊を調べるためには、次のようにする。

```
# cvs-admin/bin/cvs-setup
```

これは設定の変更時の事前確認と同じであるが、設定を変更していなければ設定とは異なるファイルを見つけ出すことになる。

8 変更のメールによる通知

CVS による開発において、開発状況を把握するために、開発対象のファイルに変更が行なわれるたびにメールを送るということが一般的に行なわれている。例えば、CVS の配布に含まれている log, log_accum/commit_prep を始めとして、aftercommit[2], cvsmailer[7], cvsmail[4] など、さまざまなものがある。しかし、これらはいずれも Perl[9], Python[8], Ruby[6] などのスクリプト言語で書かれている。このため、これらを動作させるためには、Perl, Python, Ruby などのインタプリタを jail 内に配置しなけ

```
# cvs/host/PROJECTHOST/script/start-stop-script start
```

図 4: `cvs-setup` でのサーバ起動

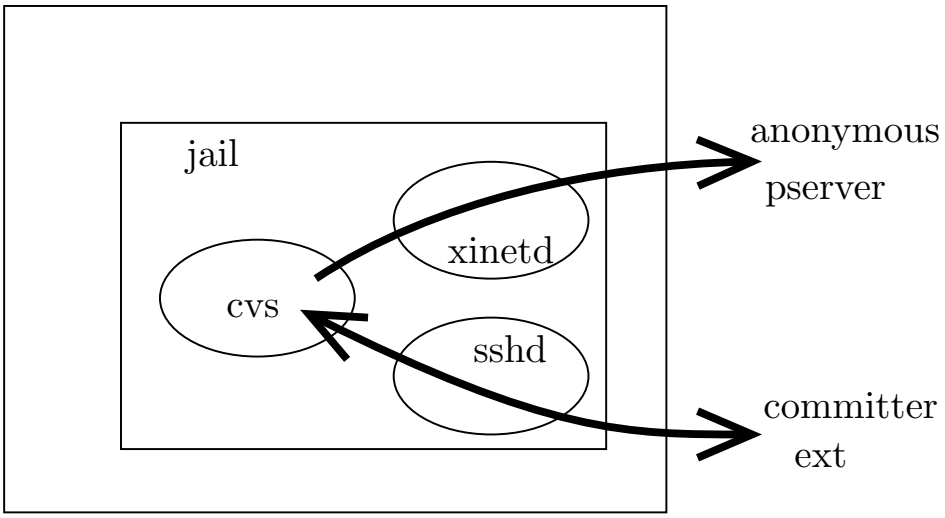

図 5: CVS との通信プロトコル

ればならない。これは jail 内には最小限のファイルしか配置しないという方針に反する。

そこで、われわれはこれらに代わる `cvs-info` というプログラムを開発した。`cvs-info` は次のような特徴を持つ。

- Objective Caml[5] で記述されており、Objective Caml native code compiler により、インタプリタを必要としないバイナリが生成できる。
  また、Objective Caml で記述することにより、Perl, Python, Ruby などと同様に buffer overrun などのセキュリティホールの問題が生じない。
- メールを出すために SMTP[3] サーバと直接通信する。これにより、`sendmail` などのメールを送るためのソフトウェアを jail 内に配置する必要がない。
- 一回の変更に対して一通のメールを送る。素朴に実装されたプログラムでは各ディレクトリ毎にメールを送ってしまうが、`cvs-info` は一回の変更が複数のディレクトリに跨っていた場合でも一通しかメール送らない。これにより、開発状況の把握を行ないやすくしている。

`cvs-info` の目的は、jail 内で動作するために、使用言語を選び、外部のコマンドに依存しないことにあるが、一回の変更に対して一通のメールを送る機構も特徴的である。

CVS において、変更時にメールを送るなどの作業をするためには loginfo という機構を使う。しかし、loginfo は一回の変更 (commit) が複数のディレクトリに渡る場合には各ディレクトリ毎にプログラムを起動する。つまり、ディレクトリ A とディレクトリ B に対する変更を行なった場合、次のようにプログラムが起動する。

1) loginfo で指定したプログラムがディレクトリ A に対して起動する。
2) loginfo で指定したプログラムがディレクトリ B に対して起動する。

ここで、起動プログラムが毎回メールを送った場合、各ディレクトリ毎にメールが送られてしまう。`cvs-info` はこれを防ぐため、次のような動作を行なう。

1) cvs が起動する。
2) loginfo で指定したプログラム (`cvs-info`) がディレクトリ A に対して起動する。`cvs-info` は daemon 化して、Unix domain socket を作り、接続を待つと同時に親プロセス (cvs) の終了を待つ。
3) loginfo で指定したプログラムがディレクトリ B に対して起動する。Unix domain socket を通じてディレクトリ B の変更情報を上記の daemon に伝える[*2]。
4) cvs が終了する。daemon 化した `cvs-info` がこれを検出すると、受け取った情報をまとめてメールを送る。

つまり、最初のディレクトリに対して起動したプロセスが daemon 化し、Unix domain socket により他のディレクトリの変更情報を受け取って情報をまとめ、CVS のプロセスが終了するのを待ってメールを送るのである。

なお、一回の変更に対して一通のメールを送るプログラムは log_accum/commit_prep を始めとして `cvs-info` 以外にも存在するが、`cvs-info` とは異なる機構を用いる。それらは loginfo の他に commitinfo という cvs の機構を用いる。commitinfo を用いた場合、次のような順序でプログラムが起動する。

1) commitinfo で指定したプログラムがディレクトリ A に対して起動する。
2) commitinfo で指定したプログラムがディレクトリ B に対して起動する。
3) loginfo で指定したプログラムがディレクトリ A に対して起動する。
4) loginfo で指定したプログラムがディレクトリ B に対して起動する。

ここで、カウンタを用意し、commitinfo から起動したプログラムでカウンタをインクリメントし、loginfo から起動したプログラムでカウンタをデクリメントすれば、loginfo からの最後の起動を検出することができ、そのタイミングでメールを送れば一回の変更に対して一通だけメールを送ることができる。

[*2] Unix domain socket に接続できなければ自身が daemon 化する。

しかし、commitinfo は commit の正否を判定するために用意されているものであり、このような用途のために用意されているものではない。そのため、このような用途に利用するのは適切ではない。具体的には、commitinfo で指定したプログラムにバグがあれば、commit 自体が失敗してしまうことがあり、また、commitinfo にプログラムを指定した場合、非常に多くのファイルが存在するディレクトリがあった場合、プログラムの起動に失敗し、commit に失敗してしまうことがある[*3]。

9 他のソフトウェアへのフィードバック

今回の開発にあたり利用した他のソフトウェアにいくつかの問題点が発見された。それらの問題点は開発元にフィードバックを行ない、修正が行なわれた。

- cvs init が匿名ユーザで使用可能。(DoS attack 可能)
- Tcl/Tk のライブラリの探索に失敗して Ruby の build が失敗する。

また、次の点についても報告を行なったが、CVS の開発者と我々のセキュリティモデルに対する立場の違いにより、修正は受け入れられなかった。我々のインストーラでサーバを構築した場合、CVS は分散環境に適切なセキュリティモデルに従うように変更されて build される[*4]。

- committer は cvs commit/update 時にサーバ上で任意のコマンドを実行できる。

10 おわりに

`cvs-setup` の開発により、CVS サーバの提供による管理者の手間を次のように削減した。

インストール
: `cvs-setup` は CVS, OpenSSH などの必要なパッケージを jail 環境に適切な設定を行ないつつまとめてインストールする。この点で、管理者の手間を削減している。

設定
: jail 環境は単一の設定ファイルで設定できる。また、OS の違いによる構成の変化 (共有ライブラリなど) は自動的に検出され、管理者は気にする必要がない。

保守
: 設定ファイルと jail 内の実際のファイルの整合性を検査することができる。 また、設定を修正した場合、その修正が jail 内にどのように波及するかを事前に調べることができる。このように管理者は jail 内の状況を簡単に把握することができる。

つまり、`cvs-setup` は jail という通常の管理者には馴染みのない技術や、jail 内部の環境の設定などのノウハウをソフトウェアとして提供することにより、管理者の手間を削減する。このように管理者の手間を削減することにより、CVS サーバの管理を容易にし、CVS サーバをより安全に運用することを可能とした。

11 参加企業及び機関

なし。

[*3] cvs は変更されたファイルをすべてプログラムの引数とするため、引数リストが非常に長くなり、`ARG_MAX` を越えると `execve` システムコールが失敗することになる。

[*4] 具体的には `Checkin-prog/Update-prog` リクエストが disable される。

12 参考文献

参考文献

[1] cvs. `http://www.cvshome.org/`.

[2] Ueno Katsuhiro. aftercommit. `http://www.blue.sky.or.jp/atelier/\#aftercommit`.

[3] John C. Klensin. Simple mail transfer protocol. RFC 2821.

[4] Frederic Lepied. cvsmail. `http://sf.net/projects/cvsmail/`.

[5] Xavier Leroy. Objective Caml home page. `http://pauillac.inria.fr/ocaml/`.

[6] Yukihiro Matsumoto. Ruby home page. `http://www.ruby-lang.org/`.

[7] Kengo Nakajima, Daichi Kanematu, and Akinori MUSHA. cvsmailer. `http://www.ruby-lang.org/en/raa-list.rhtml?name=cvsmailer`.

[8] Guido van Rossum. Python home page. `http://www.python.org/`.

[9] Larry Wall. Perl home page. `http://www.perl.com/`.