スレッド単位で権限分離を行うWebサーバのアクセス制御アーキテクチャ

松本　亮介[†a)]　　岡部　寿男[††]

Access Control Architecture Separating Privilege by a Thread on a Web Server

Ryosuke MATSUMOTO[†a)] and Yasuo OKABE[††]

あらまし　大規模高集積型のWebホスティングサービスにおいて，仮想ホスト単位で権限を分離するためには，Webサーバ上のアクセス制御手法であるsuEXEC等を利用する．しかし，既存のWebサーバにおけるアクセス制御アーキテクチャは，プロセスの生成，破棄が必要となり，パフォーマンスが低く，Web APIのような動的コンテンツに適していない．また，インタプリタやプログラム実行方式別に複数用意されており，システム開発者が扱いにくい．そこで，本論文では，コンテンツ処理時にサーバプロセス上で新規スレッドを生成し，スレッドで権限分離を行った上で，スレッド経由でコンテンツの処理を行うアクセス制御手法“mod_process_security”を提案する．この手法は，煩雑になっているWebサーバ上のアクセス制御手法を統一することで，システム開発者が扱いやすく，性能劣化も少ない．実装は，広く使われているLinuxとApache HTTP Serverに対してApacheモジュールとして組み込む形式をとった．

キーワード　Apache，VirtualHost，セキュリティ，アクセス制御，大規模

1. まえがき

HTTPプロトコルを利用したサービスの増加に伴って，HTTP上でプログラムやデータの連携を行うWeb APIが普及してきている．そのため，システム連携をWebサーバ経由で実現する機会が多くなり，Web API利用のための工夫[1]や，Webサーバの重要度が増してきている．

Webホスティングサービス[3]において，ドメイン名(FQDN)によって識別され，対応するコンテンツを配信する機能をホストと呼ぶ．単一のサーバプロセスで，複数のホストを仮想的に処理するマルチテナント型の仮想ホスト方式[4]を採用した場合に，ホスト間でファイルの閲覧や書き換えを防ぐためにアクセス制御が必要となる．これまで，Webサーバにおいて動的コンテンツを生成するプログラムのセキュリティを保護するためのアクセス制御手法としてsuEXE [2]等が利用されてきた．更に，Web APIの普及に伴ってWebサーバはプログラマブルな基盤になってきており，ホスティングサーバだけでなく，各システム連携にWeb APIを利用する状況で，サーバを実行する権限とプログラムを実行する権限等，プログラムの役割によって明確に権限を区別し適切にアクセス制御を行う必要がある．それによって，セキュリティ上のリスクを低減し，また，同一権限による他システムへの干渉やバグによる事故を減らすことができると考えられている．

Webサーバ上でのプログラム実行方式の一つであるCGI実行方式[5]に対するアクセス制御手法は十分に用意されている．しかし，CGI実行方式そのものが，プログラム実行ごとにプロセスを生成し，実行後にプロセスの破棄が必要なるため，サーバプロセスが直接プログラムを実行する場合と比較して，パフォーマンスが低くなる．また，CGIプロセスを仮想ホストの権限ごとに事前に起動させておく手法もあるが，大規模高集積型のホスティングを想定した場合，ホストの数に依存したプロセスを事前に起動させておかなければ

† 京都大学大学院情報学研究科，京都市
Graduate School of Informatics, Kyoto University, Yoshida-Honmachi, Sakyo-ku, Kyoto-shi, 606–8501 Japan
†† 京都大学学術情報メディアセンター，京都市
Academic Center for Computing and Media Studies, Kyoto University, Yoshida-Honmachi, Sakyo-ku, Kyoto-shi, 606–8501 Japan
a) E-mail: matsumoto_r@net.ist.i.kyoto-u.ac.jp

ならないためリソース効率が悪い．一方，サーバプロセスがプログラムを直接実行できるようにするために，サーバプロセスにインタプリタを直接組み込み，高速にプログラムを実行可能とする DSO 実行方式 [6] がある．しかし，現状 DSO 実行方式に対して，汎用性が高く，安全で，パフォーマンス劣化の少ないアクセス制御手法は存在しない．例えば，既存のアクセス制御手法である mod_ruid2 [7] は，安全にアクセス制御を適用するためには，サーバプロセスの生成，破棄が必要となり，CGI 実行方式よりもパフォーマンスが低下する．パフォーマンス劣化を少なくするためにシステムコールをフックする手法 [8] は，サーバプロセスを root 権限で起動する必要があり，脆弱性が生じた場合にセキュリティ上のリスクが高い点で，システム構築上，汎用性が低い．現状はこのように実行方式やインタプリタの種類によって複数のアクセス制御手法が存在し煩雑な状態になっているため，サーバ管理者（サービス提供者），システム開発者（サービス利用者）のいずれにとっても扱いにくいものとなっている．更に，この煩雑さゆえにしばしばシステム開発者がアクセス制御の必要性を認識できていない場合も多い．

今後，Web サーバを大規模高集積型に耐えうるシステムとして利用するためには，ハードウェアリソースを複数のシステムで共有する必要がある．そのためには，単一のサーバプロセスで複数のホストを処理する手法が必要となる．更に，ホスティングシステムや複数のシステムが共存している環境において，DSO 実行方式のようにプログラムを高速に動作させながらも，CGI 実行方式で用意されているのと同等のアクセス制御が適用できることが求められる．また，システム開発者に Web サーバ上でアクセス制御を適用するように促すためには，より使いやすく簡潔で，CGI や DSO 等のプログラム実行方式に依存せず，パフォーマンス劣化の少ないアクセス制御アーキテクチャでなければならない．そこで，本論文では，Web サーバにおいて，Web コンテンツの処理にスレッドを用いて権限分離を行うアクセス制御アーキテクチャを提案する．Web コンテンツを処理する際にサーバプロセスにスレッドを生成させ，スレッド単位で権限を分離した上で，スレッド上でコンテンツの処理を行う．そのため，プログラム実行方式によらない統一的なアクセス制御アーキテクチャが実現可能となり，開発者が扱いやすく汎用性も高い．また，スレッドの生成，破棄は処理が軽量であるため，DSO 実行方式を採用した場合でもパフォーマンス劣化を少なくできる．

提案するアクセス制御アーキテクチャは，Linux かつ Apache HTTP Server [9]（以降 Apache とする）上で動作する事を前提としており，Apache モジュールとして実装した．実装した Apache モジュール mod_process_security は，オープンソースとして公開しており，既に数社で利用されている．

以降，本論文の構成は以下のとおりである．**2.** ではプログラム実行方式と既存のアクセス制御アーキテクチャについて述べる．**3.** では mod_process_security のアーキテクチャについて説明する．**4.** でパフォーマンス評価を行い，**5.** でむすびとする．

2. Web サーバにおける既存のアクセス制御

Apache で構築された Web サーバ上で動的にコンテンツを生成するために動作するプログラムの実行方式として，典型的には以下の 2 種類の方式がある．一つは，Apache にモジュールとしてインタプリタを組み込み，Apache のサーバプロセス上で実行する方式（DSO 実行方式），もう一つは，モジュールとして組み込まず新たにプログラム実行用のプロセスを生成して，そのプロセス上で実行する方式（CGI 実行方式）である．Apache における仮想ホスト方式はサーバプロセス権限で全てのリクエストを処理する必要があり，全ての仮想ホストにおいてファイルやディレクトリの権限をサーバプロセス権限で操作可能にしなければならない．その仕様は，ある仮想ホストのプログラムから，他ホスト領域のファイル等を閲覧できる事を意味する．

図 1 に，他ホスト領域のファイルが閲覧可能となる一般的な仕組みと権限設定を示す．図 1 では，exmaple.com 仮想ホストの index.cgi を Apache の権限である uid500，gid101 で実行する．その場合，example.jp 仮想ホストのドキュメントルートである /var/www/hosts/example.jp/ ディレクトリに対して，gid101 からの読み取り権限が設定されている．更に，ディレクトリ配下のホスト領域内部のファイル群には Apache 権限でアクセスできるように全ユーザーに読み取り権限がある．そのため，example.com 仮想ホストの index.cgi 内でシェル等の外部コマンドを実行することで他ホストである example.jp の index.cgi のソースコードを閲覧しデータベースパスワード等を入手できる．また，Web API 等によってプログラム

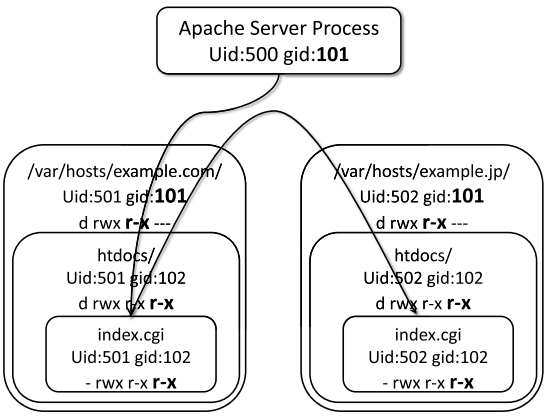

図 1　他ホスト領域の覗き見
Fig. 1　Peeping at other host.

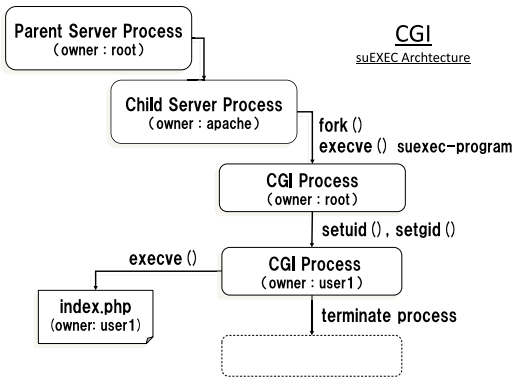

図 2　CGI 実行方式のアクセス制御アーキテクチャ
Fig. 2　The access control architecture for CGI.

を設置していた場合も，バグによって関係のないファイル等を操作してしまう恐れや，脆弱性を突かれた場合，Web サーバ上で管理している全てのファイルが危険な状態となり，被害が大きくなる．

2.1　CGI 実行方式のアクセス制御

そこで，CGI 実行方式に利用できるアクセス制御モジュールである suEXEC 機能を用いると，上記のようなリスクを低減できる．suEXEC を採用すると，クライアントから CGI にアクセスがあった場合，Apache によって CGI の実行処理を suEXEC に依頼する．suEXEC は index.cgi を実行する際に，index.cgi の権限である uid501，gid102 をサーバ設定から取得する．そして，プロセスの権限を変更するシステムコール (以降 setuid，setgid とする) を実行して，プロセスの権限を変更し，CGI を実行する．そのため，uid501，gid102 の権限では，/var/www/hosts/example.jp/配下での読み取り権限である uid502，gid101 がない．このように権限変更を行うことで，他ホスト領域へのアクセスやプログラムの閲覧を防止できる．また，Web API のプログラム群も用途別に適切に権限を分けておく事で，Apache 権限に統一して権限許可をする必要が無く，権限で区別されたプログラム群の範囲内で処理を行うことができる．脆弱性を突かれたプログラム経由の被害も，同一の権限内に収めることができる．

図 2 にサーバプロセスと suEXEC の詳細なアーキテクチャを示す．図 2 のように，suEXEC はプログラムを実行するたびに setuid-root の wrapper program を起動させて，いったん root 権限になり，そこから実行対象のプログラムの権限に setuid，setgid してからプログラムを実行する．このように，suEXEC 機能を使うためには CGI 実行方式が必須となり，プログラム実行ごとでのプロセス生成，破棄が必要となるため，パフォーマンスが低いという問題がある．

2.2　DSO 実行方式のアクセス制御

次に DSO 実行方式におけるアクセス制御について述べる．DSO 実行方式においては，mod_suid2 [10] や mod_ruid2 というモジュールを利用すると，アクセス制御が実現できる．mod_suid2 や関連研究 [11] では，Apache のサーバプロセスを root 権限で起動しておき，リクエストを処理するたびにユーザー権限に setuid，setgid する．これによって，Apache の権限とは別の権限でプロセスを実行できるため，suEXEC と同様，他ホスト領域を閲覧できなくなる．しかし，処理後はサーバプロセスが一般ユーザー権限であるため，権限を元の root 権限に戻すことができない．そのため，setuid，setgid されたプロセスをコンテンツ処理後に破棄する必要がある．その結果，サーバプロセスを再利用できず，DSO 実行方式を利用していたとしても，suEXEC よりもパフォーマンスが大きく低下する．しかし，Apache のプロセスがユーザー権限で起動している場合は，setuid や setgid を実行することができない．そこで，mod_ruid2 を利用すると，一時的にユーザー権限で起動しているサーバプロセスに，root の特権を細分化した Capability [12] と呼ばれる機構の内，CAP_SETUID，CAP_SETGID の特権を与えられる．

図 3 に mod_ruid2 のアーキテクチャを示す．特権を与えられたサーバプロセスは，root でなくても setuid，setgid を実行可能となる．その後，mod_suid2 同様

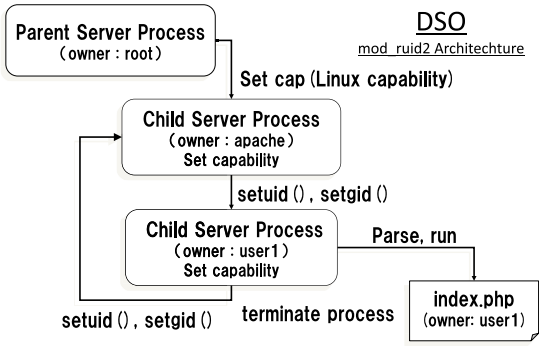

図 3　DSO 実行方式のアクセス制御アーキテクチャ
Fig. 3　The access control architecture for DSO.

に Apache のサーバプロセス自体を任意の uid，gid に権限変更してから処理を実行し，再度，元の uid，gid に戻す．この仕組みによって，DSO 実行方式であっても，プログラムは任意の権限で動作する．また，実行後でも，元のサーバプロセスの権限に戻すことで，サーバプロセスの再利用も可能にしているため，DSO 実行方式の高いパフォーマンスを維持できる．しかし，このようなプロセスは，root のように全ての権限をもたないものの，setuid，setgid を実行できる Capability を保持している．つまり，プログラムの脆弱性を突かれ，プログラム経由の権限変更によって root 昇格が可能となる．このように，サーバプロセスに権限を変更できる特権の保持を許すことは，同時に数多くの脆弱性を許すことになり，危険である．そこで，図 4 のように，setuid，setgid した後に CAP_SETUID，CAP_SETGID の Capability を放棄し，処理後にプロセスを復帰できないように改修すれば安全であるが，やはりサーバプロセスが再利用できなくなり，mod_suid2 同様パフォーマンスは著しく低下する．

以上から，パフォーマンス向上のための，サーバプロセスのアクセス制御を設定後，再度解除するというアプローチは，セキュリティを考える上では非常に危険であり，脆弱性をつかれた場合の利用者や閲覧者への被害は甚大であると考える．また，パフォーマンス劣化を少なくするためにシステムコールをフックする手法 [8] も，サーバプロセスを root 権限で起動しておく必要があり，同様に脆弱性が生じた場合にセキュリティ上のリスクが高い．

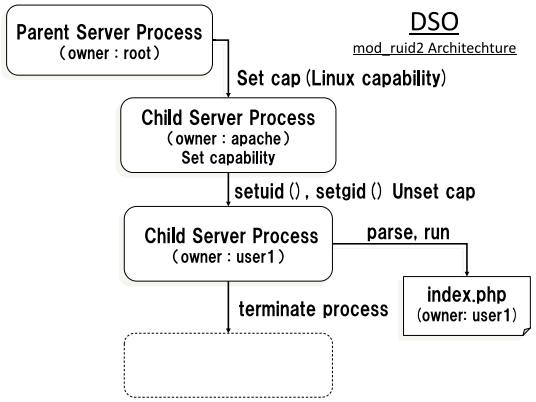

図 4　脆弱性を修正した DSO 実行方式のアクセス制御アーキテクチャ
Fig. 4　The access control architecture vulnerability is fixed for DSO.

2.3　大規模 Web ホスティングに対応したアクセス制御手法

我々は従来研究 [13] において，DSO 実行方式におけるパフォーマンス劣化と安全性及び汎用性の低さを考慮して，CGI 実行方式における大規模 Web ホスティングに対応した新たなアクセス制御手法を提案した．この手法では，Apache の仮想ホストを新たに追加する場合でも，Apache の再読み込み無く，追加された仮想ホストにアクセス制御を適用することができる．また，仮想ホスト単位で chroot するため，システム領域の覗き見も防止できる．この仕組みを利用して，コンテンツを共有ストレージに置き，複数台の Web サーバ群でそれらを共有した上で，ロードバランサ等によって負荷分散を行うような構成をとることで，大規模対応かつ，スケールアウト型の冗長構成による運用性の高いシステム構築を可能にしている．しかし，CGI 実行方式を採用したため，DSO 実行方式よりもパフォーマンスが低いという課題があった．

既存の実行方式とアクセス制御手法における問題として，まず，CGI 実行方式に対する既存のアクセス制御手法はセキュリティ面，及び，性能劣化を低く抑える要件を満たしているが，アクセス制御手法とは関係なく，CGI 実行方式そのもののアーキテクチャの性能が低い．また，現在主流となっている DSO 実行方式のアクセス制御手法である mod_ruid2 では，安全に動作させようとすると，CGI 実行方式にアクセス制御を適用した場合よりも動作が遅くなり，DSO 実行方式である意味がない．そこで，これらの問題を全て解

決するアクセス制御アーキテクチャを以降で説明する．

3. 提案するアクセス制御アーキテクチャ

2. の考察から，現在必要となるアクセス制御アーキテクチャの要件をまとめる．単一のサーバプロセスで，大規模なホスト数を高速に処理するためには，CGI 実行方式における fork() を行わずに，DSO 実行方式でセキュリティを担保するためのアクセス制御アーキテクチャが必要である．また，システム開発者が扱いやすいアーキテクチャにするために，DSO の高速処理という特徴を生かしつつ，プログラム実行方式によらない統一的なアクセス制御アーキテクチャである必要がある．更に，システムへのアクセス制御適用方法がシステム開発者にとって容易であることである．それらを満たすために，アクセス制御モジュール mod_process_security を開発した．以降で，mod_process_security のアーキテクチャを述べる．

3.1 DSO 対応と実行方式によらないアーキテクチャ

DSO 実行方式の利点は，プログラムを高速に実行できることである．そのため，DSO 実行方式のアクセス制御アーキテクチャを設計する上では，パフォーマンス劣化を十分考慮しなければならない．CGI 実行方式におけるアクセス制御手法である suEXEC のように，プログラム実行時に新たに子プロセスを生成し，コンテンツ処理後にプロセスを破棄すれば安全に実行できるが，パフォーマンスが低下する．また，mod_ruid2 のように，プロセスを生成せずにサーバプロセスに権限変更の特権を与えてプロセスを再利用すれば高速に実行できるが，**2.2** で述べたとおり，脆弱性が生じる．そこで，mod_process_security においては，Linux 上で動作する事を前提とし，Linux におけるスレッドを一時的に生成し，そのスレッド上で権限分離を行った後，スレッド配下でプログラムの処理を行い，最後にスレッドを破棄するアーキテクチャをとった．Linux におけるスレッドはプロセス内の同一メモリ空間上で実行でき，メモリ消費量等が軽減できる．また，スレッドの生成・破棄は，一般的にプロセスの生成・破棄よりも処理が非常に軽い．スレッドの生成・破棄を利用することにより，サーバプロセスを破棄する必要もない．

図 5 に DSO 実行方式に mod_process_security を適用した場合の，処理の流れを示す．Linux 上で動作する Apache は，親サーバプロセス（Parent Server Process）から事前に fork された複数の子サーバプロセス（Child Server Process）がリクエストを受け付けるために待機している．そして，リクエストを受け付けると，子サーバプロセス上で一時的にスレッド（Control Thread）を生成する．そして，一時的に生成したスレッドに対し権限変更の特権である Linux Capability の CAP_SETUID，CAP_SETGID を付与する．この特権によって，スレッドは任意の uid，gid に権限変更可能となる．その後，実行対象のプログラムの uig，gid 等の権限情報を動的に取得して，その権限にスレッドの権限変更を行う．スレッドの権限変更を行った後は，プログラムを実行する前にスレッドに付与された特権を破棄しておく．これによって，mod_ruid2 で生じたような，プログラム経由での権限変更を防止する．そして，スレッド上で直接プログラムを実行した後は，スレッドを破棄して，スレッドが属した子サーバプロセスは再度リクエスト受け付けに再利用される．

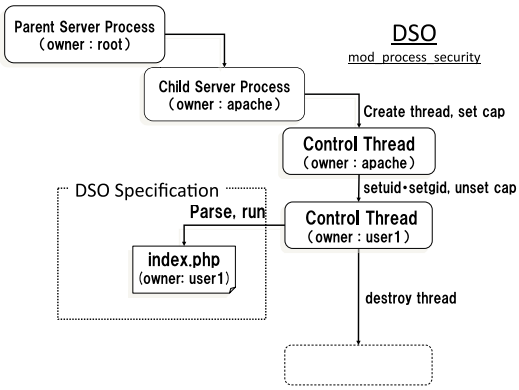

図 5 mod_process_security のアクセス制御 (DSO)
Fig. 5 FThe access control on mod_process_security for DSO.

これによって，既存の DSO 実行方式のアクセス制御アーキテクチャのように，サーバプロセスの生成破棄をすることなく，安全にアクセス制御を行える．また，スレッドの生成，破棄の処理時間の短さから性能劣化を低減し，DSO 実行方式の特徴である高いパフォーマンスを維持できる．パフォーマンス評価に関しては **4.** で行う．

図 6 に CGI 実行方式の場合の mod_process_security のアーキテクチャを示す．DSO 実行方式と CGI 実行方式が混在した環境であっても，一時的にスレッドを生成し，スレッドそのものの権限をプログラムの権限に変更する処理までのアーキテクチャは同じである．

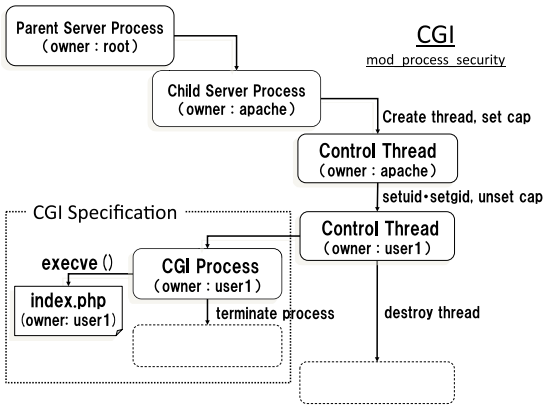

図 6　mod_process_security のアクセス制御 (CGI)
Fig. 6　The access control on mod_process_security for CGI.

そのため，図 6 のように CGI 実行方式の場合は，スレッドから CGI 用の一時プロセス（CGI Process）が生成され，プログラムがプログラムそのものの権限で execve() される事になる．スレッド以降の処理は実行方式特有の仕様となるが，上位のスレッドで権限分離を行っているため，mod_process_security では実行方式を意識する事なくアクセス制御を行う事ができる．CGI 実行方式における既存のアクセス制御手法とのパフォーマンス比較も **4.** で述べる．このように，mod_process_security はスレッドを利用したアーキテクチャをとることで，プログラム実行方式によらないアクセス制御が実現可能となる．

3.2　システム開発者にとって扱いやすい仕様

3.1 で述べたとおり，リクエストのあったプログラムから動的に権限情報を取得する方式を採用したのは，例えば Web ホスティングシステムにおいて，顧客領域を追加する際に，suEXEC では顧客ごとに設定ファイルに静的な権限設定が必要になる．その場合，単一のサーバプロセスで大規模な仮想ホスト数を収容した場合，ホストを追加するたびにサーバプロセスの再読み込みが必要となり，一時的なサービス停止によって多くのホストに影響を与えてしまう問題があった．しかし，権限情報を動的に取得することによって，新規顧客領域追加時や別権限の API 群設置時において，Apache プロセスの再読み込みや面倒な設定が必要無くなる．そのため，システム開発者にとって扱いやすくサービスの質が向上する．

mod_process_security は Apache モジュールとして実装しているため，モジュールファイルを一つ組み込むだけで，アクセス制御が簡単に実現できる．図 6 に，全てのファイルに対してアクセス制御行う場合の組み込み設定例を示す．システム開発者は，OS レベルで特別な設定等をすることなく，役割に沿ってプログラム群に適切に権限を設定し，図 7 の設定を書くだけで，プログラムの権限でプログラムが動作する．mod_process_security はプログラムだけでなく静的コンテンツを含めた全てのファイルに対して設定したり，任意の拡張子に対して設定したりすることができる．このように，DSO・CGI 等の実行方式やインタプリタの種類によってアクセス制御手法を区別する必要が無くなり，非常に簡単にアクセス制御を適用する事が可能となる．

```
# mod_process_securityモジュールの読み込み
LoadModule process_security_module modules/mod_process_security.so

# 全てのファイルに対してアクセス制御を適応
PSExAll On
```

図 7　mod_process_security 設定例
Fig. 7　Example configuration for mod_process_security.

以上のアーキテクチャによって，mod_process_security を組み込めば，DSO 実行方式に対応したインタプリタ（PHP，Perl，Python，Ruby 等）は DSO 実行方式を採用して，高速にプログラムを動作させる．その他シェルスクリプト等の場合は CGI 実行方式で扱う，というように，プログラム実行方式を気にすることなく，より柔軟に安全なシステムを設計可能になる．

4.　パフォーマンスの実験と評価

本章では，mod_process_security をアクセス制御に適用した場合のパフォーマンス評価を行う．その際に，既存のアクセス制御手法を適用した場合との比較を行う．表 1 にテスト環境の詳細を示す．実験においては，一台のサーバ計算機を用意し，別の一台のクライアント計算機から通信を行うことで評価を行った．ベンチマークソフトは httperf0.9.0 [14] を利用し，クライアントにおいて 1 秒間に生成するリクエスト数を変動させ，サーバ側で 1 秒間に処理が完了したリクエスト数を計測した．使用する言語は，CGI 実行方式及び DSO 実行方式両方で実行可能な PHP とし，実行方式の性能差を顕著にするために，PHP の設定情報を phpinfo() 関数で表示するだけの簡易なコードで比較を行った．

表 1　テスト環境
Table 1　Hardware configuration of test environments.

Client Machine	
CPU	Intel Core2Duo E8400 3.00GHz
Memory	4GB
NIC	Realtek RTL8111/8168B 1Gbps
OS	CentOS 5.6
Server Machine	
CPU	Intel Xeon X5355 2.66GHz
Memory	8GB
NIC	Broadcom BCM5708 1Gbps
OS	CentOS 5.6
Middleware	Apache/2.2.3

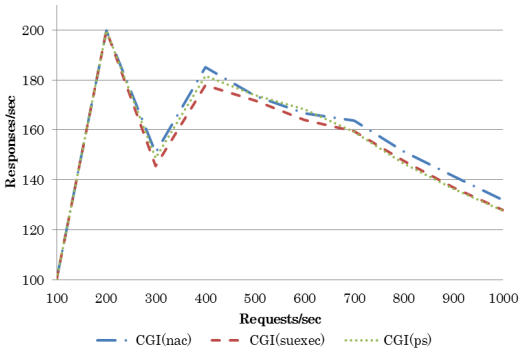

図 8　CGI のアクセス制御における性能比較
Fig. 8　Performance comparison of access control methods for CGI.

また，実験対象の仮想ホストの数は 1 ホストとした．実験で採用した Apache の仮想ホスト方式は，アクセスのあったホスト名から，それぞれに対応したファイルパス名を導出する．この導出方法において，単一の仮想ホストに同時接続する場合と，複数の仮想ホストに同時に接続する場合で，アクセス先のファイルパスが違うだけで，ミドルウェアの性能面での本質的な差はないと考えられる．また，本論文で提案した mod_process_security においては，仮想ホスト方式そのもののアーキテクチャは変更していないため，アクセス制御手法の性能差を比較する場合には，1 ホストに対してのみを実験対象に行っても十分であると考えられる．

4.1　アクセス制御のパフォーマンス評価

mod_process_security を DSO 実行方式と CGI 実行方式それぞれに適用した場合のパフォーマンスを評価する．評価には，アクセス制御手法を適用していない場合，既存のアクセス制御手法を適用した場合，mod_process_security を適用した場合の 3 パターンについて比較を行う．グラフにおいて “nac” はアクセス制御手法を適用していない場合，“ps” は mod_process_security を組み込んだ場合，“suEXEC” は suEXEC を組み込んだ場合，“ruid2_custom” は **2.2** で述べた mod_ruid2 の脆弱性をもった実装箇所を，サーバプロセスを破棄する方式に修正したモジュールを組み込んだ場合を示している．縦軸はサーバが 1 秒間に処理できたレスポンスの数，横軸はクライアントが 1 秒間に要求したリクエストの数を示している．

4.1.1　CGI 実行方式に適用した場合

CGI 実行方式の既存のアクセス制御手法は suEXEC を採用した．図 8 に CGI 実行方式での評価結果を示す．CGI 実行方式においては，プログラム実行ごとに必ずプロセスの生成，破棄を行うため，そのルーチンで処理がボトルネックとなる．そのため，アクセス制御適用可否において，クライアントのリクエスト数を変動させていっても，大きな性能劣化は見られなかった．また，同様の理由で，suEXEC と mod_process_security の性能差もほとんど見られなかった．また，アクセス制御を適用していないサーバが処理できたレスポンス数から，各種アクセス制御を適用したときのレスポンス数の低下数を，100 リクエスト増加するごとに算出し，その平均値を性能劣化率として算出した．それぞれのアクセス制御を適用しない場合の性能に対して，アクセス制御適用後の性能劣化率は，suEXEC の場合平均 2.15%であったのに対し，mod_process_security は平均 1.48%であった．

4.1.2　DSO 実行方式に適用した場合

DSO 実行方式の既存のアクセス制御は mod_ruid2 を採用した．ただし，mod_ruid2 は通常の使い方において root 昇格の脆弱性をもっているため，脆弱性を修正した方式でパフォーマンスを計測した．図 9 に DSO 実行方式での評価結果を示す．

脆弱性対応版 mod_ruid2 のパフォーマンスはすべてのリクエストに対して，1 秒間に約 4.5 レスポンス程度と非常に低く，DSO 実行方式にも関わらず，図 8 における CGI 実行方式のパフォーマンスの 160 前後よりも大きく低下した．これは，**2.** で述べた通り，既存の DSO 実行方式のアクセス制御アーキテクチャは，安全にアクセス制御を行うために，プログラム実行ごとにサーバプロセスの生成，破棄を必要と

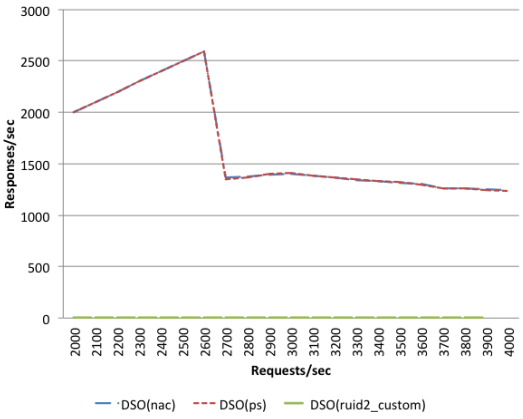

図 9　DSO のアクセス制御における性能比較
Fig. 9　Performance comparison of access control methods for DSO.

する．サーバプロセスの生成，破棄は CGI 実行方式におけるプログラム実行用のプロセス生成，破棄よりも処理に時間がかかるためだと考えられる．また，mod_process_security を組み込んだ場合は，アクセス制御を適用しない場合と比較してもほとんど性能劣化は見られず，平均 0.19%の性能劣化であった．既存のアクセス制御手法では，権限分離のために CGI プロセスやサーバプロセスの生成，破棄が必要となる．その処理は，Apache がリクエストのあったプログラムを実行してレスポンスを返す処理と比較し，無視できない程度に処理量が多かったため，性能劣化に大きく表れていた．一方で，mod_process_security では，権限分離にスレッドの生成，破棄を利用しており，その処理は，プロセスの生成，破棄と比較しても非常に軽量である．そのため，性能劣化が見られなかった理由は，リクエストを受けた Apache がプログラムを実行してレスポンスを返す処理と比較し，ほとんど無視できる程度にまでアクセス制御に必要な処理を低減できていたためだと考えられる．

以上のパフォーマンス評価より，DSO 実行方式において，既存のアクセス制御手法と比較して大きな優位性があり，性能劣化も非常に少ないことから実用に耐えうると考えられる．また，CGI 実行方式においては，ほとんど性能に差は見られなかったものの，mod_process_security は導入が容易で，CGI 実行方式と DSO 実行方式を意識せず，統一的に扱う事ができるため，利便性が大幅に向上したと考えられる．

5. むすび

本論文では，大規模高集積を想定した仮想ホストを利用した Web サーバ上のアクセス制御において，スレッド単位で権限を分離する事により，性能劣化を少なくし，また，煩雑になっている Web サーバ上のアクセス制御手法を統一することで，システム開発者が扱いやすいアクセス制御アーキテクチャを提案した．これまでは，プログラム実行方式ごとにアクセス制御を組み込む必要があり，DSO 実行方式においては性能劣化の少ない標準的なアクセス制御手法は存在しなかった．しかし，mod_process_security を組み込むことで，DSO 実行方式の実行速度で高速にプログラムを処理することができ，また，設計によって CGI 実行方式を導入する場合でも mod_process_security で全てのアクセス制御が可能となった．導入も容易になっており，システム開発者の扱いやすい仕様になっている．

今後の課題として，今まで疎かにされていた Web サーバへのアクセス制御適用を促していく．それによって，Web サーバ上で生じるセキュリティインシデントの低減に貢献できると考えている．可能であれば，Apache における標準的なアクセス制御モジュールとして組み込むように，Apache の ML において議論を行いたい．また，大規模で高集積なシステムに耐えうるべく，Apache の内部制御を容易にする仕組みの設計や，Apache の仮想ホスト単位でリソースをより緻密に管理するモジュールを設計して組み合わせることで，単一のプロセスであっても，仮想マシンに近いリソースの分離が可能となるような，安全でリソース管理のしやすい仮想ホストアーキテクチャを設計していく予定である．

謝辞　本論文を執筆するにあたり有益なコメントをいただいたファーストサーバ（株）川原将司氏に感謝する．

文　献

[1] 高井正成，阪口哲男，“Web API 利用のためのプログラムライブラリ自動生成，” 情処学研報，2912-IFAT-108, pp.1–8, Sept. 2012.
[2] The Apache Software Foundation, “suEXEC Support,” http://httpd.apache.org/docs/2.2/en/suexec.html.
[3] R. Prodan and S. Pstemann, “A survey and taxonomy of infrastructure as a service and web hosting cloud providers,” Grid Computing, 2009 10th IEEE/ACM International Conference, pp.17–25, Oct. 2009.
[4] The Apache Software Foundation, “Apache Vir-

tual Host documentation," http://httpd.apache.org/docs/2.2/en/vhosts/
[5] The Apache Software Foundation, "Apache Tutorial: Dynamic Content with CGI," http://httpd.apache.org/docs/2.2/en/howto/cgi.html
[6] The Apache Software Foundation, "Dynamic Shared Object (DSO) Support," http://httpd.apache.org/docs/2.2/en/dso.html
[7] H. Nakamitsu and P. Stano, "mod_ruid," http://websupport.sk/~stanojr/projects/mod_ruid/
[8] 原 大輔，中山泰一，"Hussa: スケーラブルかつセキュアなサーバアーキテクチャ～低コストなサーバプロセス実行権限変更機構，" 第 8 回情報科学技術フォーラム (FIT 2009) 講演論文集，RB-002, Sept. 2009.
[9] The Apache Software Foundation, "Apache HTTP SERVER PROJECT," http://httpd.apache.org/
[10] H. Nakamitsu, "mod-suid2," http://code.google.com/p/mod-suid2/
[11] 原 大輔，尾崎亮太，兵頭和樹，中山泰一，"Harache: ファイル所有者の権限で動作する WWW サーバ，" 情処学論，vol.46, no.12, pp.3127–3137, 2005.
[12] 日本 Linux 協会，"JM Project CAPABILITIES," http://archive.linux.or.jp/JM/html/LDP_man-pages/man7/capabilities.7.html
[13] 松本亮介，川原将司，松岡輝夫，"大規模共有型 Web バーチャルホスティング基盤のセキュリティと運用技術の改善，" 情処学論，vol.54, no.3, pp.1077–1086, March 2013.
[14] D. Mosberger and T. Jin, "httperf: A tool for measuring Web server performance," Proc. 1st Workshop on Internet Server Performance, pp.59–67, 1998.

（平成 25 年 1 月 31 日受付，5 月 29 日再受付）

松本 亮介 （学生員）

2008 阪府大･工･情報工学卒．同年ファーストサーバ（株）入社．レンタルサーバ (ホスティング) の基盤技術に関わる研究・開発・運用に従事．2012 年 4 月より，京都大学情報学研究科博士課程．情報処理学会学生会員．

岡部 寿男 （正員：フェロー）

1988 京都大学大学院工学研究科修士課程了．同年京都大学工学部助手．同大型計算機センター助教授などを経て 2002 より同学術情報メディアセンター教授．博士（工学）．2005 より国立情報学研究所客員教授．インターネットアーキテクチャ，ネットワーク・セキュリティ等に興味をもつ．情報処理学会，システム制御情報学会，日本ソフトウェア科学会，IEEE，ACM 各会員．