Vol.2012-IOT-16 No.13
2012/3/15

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[†]　　岡部寿男[†]

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方式は，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，本稿では，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，高速に動作し，かつ，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なる．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．

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

Ryosuke Matsumoto[†]　and Yasuo Okabe[†]

In Web hosting services, hosting systems use access controls like suEXEC on apache Web servers to separate privilege by each virtual host. However, existing access control architectures on Web servers have a problem in their low performance and are not appropriate for dynamic contents like Web API since these architectures require termination of the process after each HTTP session. The system developers are not easy to install existing access controls since these are provided by each interpreter and program execution methods conventionally. In this paper, we propose the access control architecture “mod_process_security”. In this architecture a server process creates a new thread on the server process when accepting a request. Then, the web server separates privilege by the thread and processes the contents on the thread. The server process installed “mod_process_security” executes programs faster. System developers can easily install it on web servers since we replace it with the complicated existing access controls. “mod_process_security” can be installed for Apache HTTP Server on Linux as Apache Module which is widely used.

1. はじめに

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[1]等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，より Web サーバの重要度が増してきている．これまで，Web サーバにおけるプログラムのアクセス制御として suEXEC[2]等が利用されてきた．アクセス制御は，例えば，ホスティングサーバ（レンタルサーバ）[3]において，単一のサーバプロセスで複数のホストを処理する仮想ホスト方式[4]を採用した場合に, 他ホスト領域のファイルの閲覧や書き換えを防ぐために必要となる. さらに，Web API の普及に伴って Web サーバはより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，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，プログラム実行毎にプロセスを生成し，実行後にプロセス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．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．DSO 実行方式に対するアクセス制御は，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は存在しない．例えば，既存のアクセス制御である mod_ruid2[7]は，安全にアクセス制御を適応するためには，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．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]は，サーバプロセスのみに権限変更のシステムコールの実行を許すため，システム構築上，汎用性が低い．また，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が存在しており煩雑な状態になっているため，システム開発者にとっても扱いにくく，しばしばアクセス制御の必要性を認識できていない場合が多い．

今後，Web サーバがより重要なシステム連携の役割を担っていくことを考えると，プログラムは DSO 実行方式を採用し，高速に動作させ，かつ，アクセス制御を適用する事が求められる．また，システム開発者に Web サーバ上でアクセス制御を採用するように促すためには，より使いやすくシンプルで，CGI や DSO 等のプログラム実行方

[†] 京都大学 学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

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方式でなければならない．そこで，本稿では，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機能は，Linux かつ Apache HTTP Server[9]（以降 Apache とする）上で動作する Apache モジュールとして実装した．以降では，提案するアクセス制御を mod_process_security と呼ぶ．

本稿では，Web サーバがよりプログラマブルな基盤になっている事を考慮し，スレッドを利用することで，システム開発者が扱いやすく，プログラム実行方式に依存しない Apache のアクセス制御モジュール mod_process_security を提案する．2 章ではプログラム実行方式と既存のアクセス制御について述べる．3 章では mod_process_security のアーキテクチャについて説明する．4 章でパフォーマンス評価を行い，5 章でむすびとする．

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 サーバ上で動作するプログラム実行方式には，一般的に 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 のサーバプロセス内部で実行する方式（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，そこで実行する方式（CGI 実行方式）がある．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，すべ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 で，他ホスト領域のファイルを閲覧するための一般的な仕組みと権限設定を示す．図 1 では，index.cgi を Apache の権限である uid500，gid101 で実行する．/var/www/hosts/fuga2.com/ディレクトリは gid101 からの読み取り権限がある．さらに，ディレクトリ配下のホスト領域内部のファイル群は 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， 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の db.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

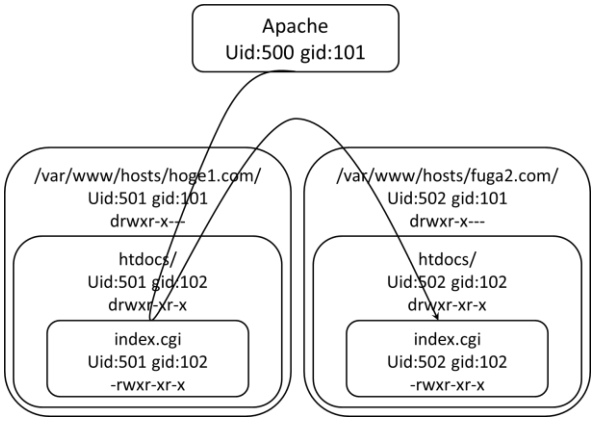

図 1　他ホスト領域の覗き見

きくなる．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/fuga2.com/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 db.cgi 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，一旦 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，プログラム実行毎でのプロセス生成，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．

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度にユーザー権限に 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持たないものの，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，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，システムコールをフックさせる等の手法は，システム設計の柔軟性を奪い，アクセス制御適応も困難になる．そこで，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 サーバ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応することができる．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．

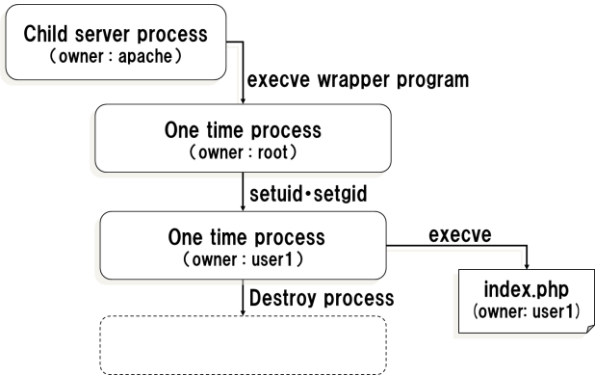

図 2　CGI 実行方式のアクセス制御アーキテクチャ

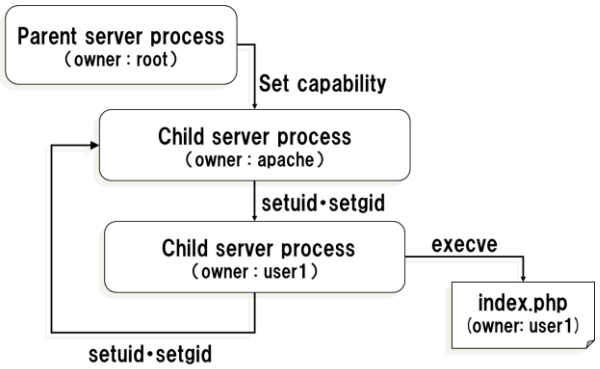

図 3　DSO 実行方式のアクセス制御アーキテクチャ

既存のアクセス制御における問題として，まず，CGI 実行方式における既存のアクセス制御は十分であるが，CGI 実行方式自体のプログラム実行速度が遅い．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 実行方式のアクセス制御よりも動作が遅くなり，DSO 実行方式である意味が無い．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 章の考察から，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．まず，DSO 実行方式に対応したアクセス制御であり，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アクセス制御アーキテクチャであることである．さらに，システムへのアクセス制御適応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．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，破棄を行え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，脆弱性が生じる．そこで，mod_process_security においては，スレッドを利用するアーキテクチャをとった．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，一般的にプロセスの生成よりも処理が軽い．

図 4 に mod_process_security のアーキテクチャを示す．まず，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 CAP_SETUID，CAP_SETGID を付与する．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，スレッド上でプログラムを実行後は，スレッドを破棄して，スレッドが属したプロセスは再度リクエスト受け付けに再利用される．これによって，既存の DSO 実行方式のアクセス制御のように，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 章で行う．

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も実現可能となる．DSO 実行方式の場合は上記で述べた動作をし，CGI 実行方式の場合は，mod_process_security によって生成されたスレッドからプログラム実行用の新規プロセスが生成され，そのプロセス上でプログラムが実行される．そのため，CGI プログラムはスレッドの権限，つまり，リクエストのあったファイルの権限で動作する．これによって，suEXEC と併用する必要もない．CGI 実行方式における既存のアクセス制御とのパフォーマンス比較も 4 章で述べる．

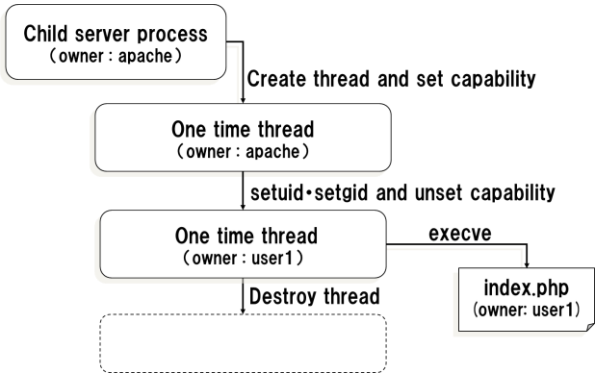

図 4　mod_process_security のアクセス制御

```
$ cat /etc/httpd/conf.d/process_security.conf
LoadModule process_security_module modules/mod_process_security.so
PSExAll On
```

図 5　mod_process_security 設定例

3.2 システム開発者にとって扱いやすい仕様

3.1 で述べた通り，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， 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 等の顧客毎の権限設定が必要である場合，Apache プロセスの再読み込みが頻繁に必要となっ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．また，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 1 つ組み込むだけで，アクセス制御機能が簡単に実現できる．図 5 に，すべ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 5 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．図 5 のように，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すべ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．以上から，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った．

以上のアーキテクチャによって，一旦 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 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を採用した場合の比較を行う．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．また，検証プログラムには PHP を利用し，PHP の設定情報を phpinfo()関数で表示するシンプルなコードを 5 記述した．

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 実行方式に適応した場合のパフォーマンスを評価する．評価には，アクセス制御を適応していない場合，既存のアクセス制御を適応した場合，mod_process_security を適応した場合について比較を行う．グラフにおいて”nac”はアクセス制御を組み込んでいない場合，”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 mod_ruid2 の脆弱性修正バージョンを組み込んだ場合を示している．

4.1.1 CGI 実行方式に適応した場合

CGI 実行方式の既存のアクセス制御には suEXEC を利用した．図 6 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，プログラム実行毎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，アクセス制御適応可否において大きな性能劣化は見られなかった．しかし，suEXEC よりも mod_process_security がより少ない性能劣化であることがわかる．これは，suEXEC は wrapper プログラムに処理を依頼し複雑な処理を行う一方で，mod_process_security ではスレッドを生成するにとどまるためだと考えられる．それぞれの性能劣化率は，suEXEC は平均 2.15%であったのに対し，mod_process_security は平均 1.48%であった．

4.1.1 DSO 実行方式に適応した場合

DSO 実行方式の既存のアクセス制御には mod_ruid2 を利用した．ただし，mod_ruid2 は通常の使い方において root 昇格の脆弱性を持っているため，脆弱性を修正した方式

表 1　テスト環境

クライアント	
CPU	Intel Core2Duo E8400 3.00GHz
Memory	4GB
NIC	Realtek RTL8111/8168B 1Gbps
OS	CentOS 5.6
サーバ	
CPU	Intel Xeon X5355 2.66GHz
Memory	8GB
NIC	Broadcom BCM5708 1Gbps
OS	CentOS 5.6
Middleware	Apache/2.2.3

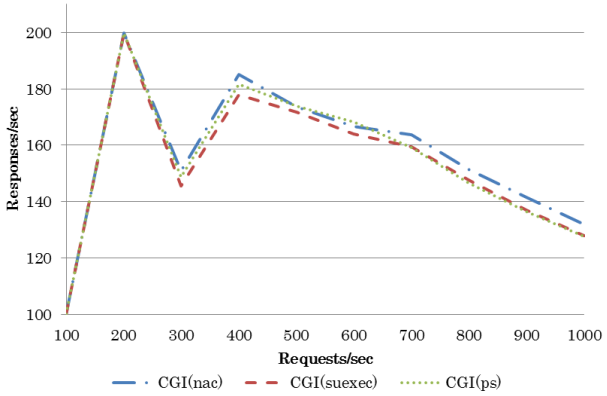

図 6　CGI のアクセス制御における性能比較

でパフォーマンスを計測した．図 7 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，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 章で述べた通り，既存の DSO 実行方式のアクセス制御を安全に利用するためには，プログラム実行毎にサーバプロセスの生成，破棄が行われるためである．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，

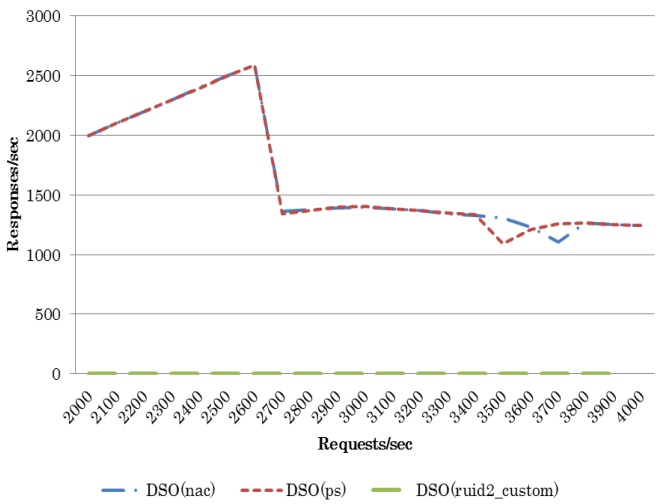

図 7　DSO のアクセス制御における性能比較

mod_process_security を組み込んだ場合は，アクセス制御を組み込ま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．

以上のパフォーマンス評価より，CGI，DSO 実行方式共に，既存のアクセス制御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．

5. むすび

本稿では，煩雑になっていた既存のアクセス制御を統一し，システム開発者が扱いやすく，性能劣化の少ない Web サーバ上で動作するアクセス制御アーキテクチャ mod_process_security を提案した．これまでは，プログラム実行方式毎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 ですべ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導入を促していく．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，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，組み合わせることで，安全でリソース管理のし易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．

参考文献

1) Wikipedia, “Web API”, http://en.wikipedia.org/wiki/Web_API.
2) The Apache Software Foundation, "suEXEC Support”, http://httpd.apache.org/docs/2.2/en/suexec.html.
3) Wikipedia, “ホスティングサーバ”, http://ja.wikipedia.org/wiki/%E3%83%9B%E3%82%B9%E3%83%86%E3%82%A3%E3%83%B3%E3%82%B0%E3%82%B5%E3%83%BC%E3%83%90
4) The Apache Software Foundation, “Apache Virtual 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) Hideo NAKAMITSU and Pavel 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 (Sep. 2009).
9) The Apache Software Foundation, "Apache HTTP SERVER PROJECT", http://httpd.apache.org/.
10) Hideo 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", 情報処理学会論文誌, 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, 汎用性の高い大規模共有型 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, インターネットと運用技術シンポジウム 2011 論文集, 2011,31-38 (2011-11-24).
14) Mosberger, D. and Jin, T.: httperf: A Tool for Measuring Web Server Performance, Performance Evaluation Review, Vol.26, No.3, pp.31{37 (1998). pp.59{67 (1998).