システムソフトウェアと
オペレーティング・システム 84－28
（２０００．５．２６）

悪意ある外部プログラムによるリソース濫用の防止
－ファイルキャッシュのケーススタディ－

金子 済[1]　河野 健二[2,3]　益田 隆司[2]

[1] 東京大学 大学院理学系研究科 情報科学専攻
[2] 電気通信大学 情報工学科
[3] 科学技術振興事業団 さきがけ研究 21

E-mail: wataruk@is.s.u-tokyo.ac.jp, {kono, masuda}@cs.uec.ac.jp

要旨

本論文は，サーバなどの利用する信頼できない外部プログラムが，正当なアクセス権を持つリソースを濫用し計算機の機能を損なう危険性を実証する．ファイルキャッシュを濫用するCGIの想定実験では，ウェブサーバのスループットは5～81%低下した．そこで濫用を防ぐ方法として，濫用プロセスのファイルキャッシュ積極破棄とスケジューリング優先度低下を提案する．提案方式の下の実験では，ウェブサーバのファイルキャッシュが残っている場合スループット低下を90%以上防いだが，すべて奪われると15%しか抑制できなかった．これはファイルキャッシュ積極破棄とスケジューリング優先度低下が十分に機能しなかったためで，改良の必要がある．

Defending against Resource Abuse by Malicious Codes: A Case Study of File Cache

Wataru Kaneko[1]　Kenji Kono[2,3]　Takashi Masuda[2]

[1]Department of Information Science, Faculty of Science, University of Tokyo
[2]Department of Computer Science, University of Electro-Communications
[3]PRESTO, Japan Science and Technology Corporation

E-mail: wataruk@is.s.u-tokyo.ac.jp, {kono, masuda}@cs.uec.ac.jp

Abstract

This paper demonstrates that untrusty external programs executed by servers can damage local computers by abusing authorized access to resources. Experiments show the web server throughput lowers 5% to 81% under the file cache abuse by malicious CGI. Therefore we propose the defence system against file cache abuse. This system vitimizes monopolized file caches actively, and marks down the scheduling priority of the abuser process. In experiments, our system keeps the throughput degradation more than 90% while the server still holds file caches. But when the server loses all its caches, our system keeps the degradation only 15%, because both of victimizing monopolized caches and low scheduling does not work well. We need to improve them.

1 はじめに

アプリケーションやサーバの外部プログラムをネットワークから入手して利用することが一般的になっている．例えば，ウェブブラウザでは，新たなマルチメディア機能を追加したり，対話性のあるウェブページを実現するため，プラグインやアプレットなどの外部プログラムが利用できる．また，ウェブサーバでは，一般のユーザでも動的にコンテンツを生成できるように，CGIと呼ばれる外部プログラムが利用できる．CGIは一般のユーザが作成したり，インターネット経由で入手できる．そのため，悪意あるユーザが作成したプログラムが存在する可能性があり，外部プログラムは必ずしも信頼できない．

信頼できない外部プログラムを安全に実行する枠組みとして，SFI[1, 2], PCCode[3], 多重保護ページテーブル [4, 5], Palladium[6], SPIN[7], VINO[8], などが提案されている．これらの研究は，外部プログラムのアクセス権を細かく制御できるようにし，不正アクセスを防止することを狙っている．そのため，信頼できない外部プログラムに対しても，実行に必要なリソースにはアクセスを許可する必要がある．この点を悪用すると，正当なアクセス権を持つリソースを濫用し，計算機の機能を損なわせることができる．これは，サービス不能攻撃 (denial-of-service attack) の一形態と言える．

本論文では，悪意あるプログラムによる濫用リソースの例として，ファイルキャッシュを対象にリソースの濫用を防止する手法を提案し，その結果についての考察を報告する．ファイルキャッシュを濫用するCGIが存在する場合，ウェブサーバがアクセスするファイルであっても，ファイルキャッシュから追い出され，サーバのファイルI/O性能が低下する．実際，ウェブサーバのひとつであるApacheを用いた実験では，ファイルキャッシュを濫用するCGIによってウェブサーバのスループットが5%から81% 低下することが確認された．

本論文で提案する手法では，ファイルキャッシュを濫用するプロセスに対し，(1) ファイルキャッシュを積極的に破棄し，(2) スケジューリングの優先度を下げる，というペナルティを与える．(1) のファイルキャッシュ積極的破棄の狙いは，他のプロセスがファイルキャッシュを確保できるようにすることにある．(2) のスケジューリング優先度低下の狙いは，悪意あるプロセスが継続的にファイルI/Oを発行し，破棄されたキャッシュを再び確保できないようにすることにある．

この機構をLinux 2.2.13をベースに実装し，再度実験を行ったところ，最善の場合，スループットの低下を90%以上抑制できた．しかしながら，スループットの低下が最大となる場合では，スループットの低下を15%程度しか抑制できなかった．これは，前に述べた二つのペナルティを受けても，ファイルキャッシュを濫用しているプロセスはファイルキャッシュを大量に占有し続けることができたからである．本論文で用いた実装では，キャッシュの積極的破棄を実現するため，濫用プロセスのファイルキャッシュにはセカンドチャンスアルゴリズムにおけるセカンドチャンスを与えないようにしている．また，スケジューリングの優先度を下げるためにnice値を変更している．この実験結果から，この実装では十分に対応できない場合があることがわかった．

2 ファイルキャッシュの特性

ファイルキャッシュには，(1) キャッシュに対するアクセスの検出が困難であり，(2) キャッシュを所有するプロセスの特定が困難である，という特徴がある．それゆえに，ファイルキャッシュの濫用を検出し，濫用しているプロセスを特定することは容易でない．ファイルキャッシュがこのような特徴を有するのは，次の二つの特性から来ている．

- **共有リソース**
  ファイルキャッシュは，すべてのプロセスによって共有されたリソースであり，各ファイルキャッシュとそのキャッシュを使用しているプロセスとを対応付ける情報はない．そのため，あるプロセスがファイルキャッシュを独占し他のプロセスによる使用が妨げられていたとしても，カーネルはそのような濫用を検出できない．また，何らかの手段でファイルキャッシュの濫用を検出できたとしても，ファイルキャッシュを濫用しているプロセスを特定できない．

- **ファイルキャッシュの透明性**
  ユーザプログラムがファイルにアクセスする方法は，`read()`, `write()` などのシステムコールを介する方法だけでなく，`mmap()` などを用いてファイルをメモリにマップしてアクセスする方

法がある．前者の場合は，ユーザプログラムによるファイルキャッシュへのアクセスはシステムコール内で発生するため，カーネルが容易にそのアクセスを検出できる．しかし後者の場合，ファイルキャッシュはユーザプログラムのアドレス空間にマップされ，ユーザプログラムは通常のメモリアクセス命令 (load,store など) によってファイルキャッシュにアクセスするので，カーネルがそのアクセスを検出するのは容易でない．したがって，ファイルキャッシュのアクセスを検出し，各ファイルキャッシュとプロセスとを関係付けることはできない．

本研究においてファイルキャッシュに焦点をあてた理由は，キャッシュの濫用によって実際にウェブサーバの性能を低下させられるというだけでなく，上記の特性ゆえに，ファイルキャッシュの濫用を検出・防止する機構の実現は容易ではないと考えたからである．

3 ファイルキャッシュ濫用防止機構の実装

前節で指摘したように，ファイルキャッシュはその濫用の検出および濫用しているプロセスの特定が容易ではない．3.1 節では，キャッシュの濫用の検出方法および濫用プロセスを特定する方法について述べる．3.2 節では，本論文の実装で用いたキャッシュ濫用検出のポリシーについて論じる．3.3 節では，キャッシュ濫用の防止策について述べる．

3.1 キャッシュの濫用・所有プロセスの検出

3.1.1 キャッシュ所有プロセスの特定

2 節で指摘したように，ファイルキャッシュは複数のプロセスから共有されるため，その所有者を特定することが難しい．本論文で用いた方式では，ファイルキャッシュ用のメモリを確保したプロセスを，そのキャッシュの所有者とみなすようにした．ファイルキャッシュを確保したプロセスは，そのキャッシュを利用することが期待できる．そのため，キャッシュを確保したプロセスは，キャッシュを所有するプロセスのよい近似になっていると期待できる．

3.1.2 キャッシュ濫用の検出

ファイルキャッシュの濫用を検出するために，各プロセスが確保したファイルキャッシュの容量と解放した容量とを記録するようにした．各プロセスごとにキャッシュの確保量から解放量を引けば，プロセスごとのファイルキャッシュ使用量を知ることができる．ファイルキャッシュ保持量が最大であるプロセスをもって，キャッシュを濫用していると判断するポリシーが最も直感的である．

ファイルキャッシュの確保量と解放量とを別々に記録することによって，キャッシュの振る舞いをより詳細に知ることができ，キャッシュ濫用を検出する様々なポリシーを実現することが可能になる．例えば，キャッシュの保持量 (確保量 − 解放量) が十分に大きくとも確保量と解放量が著しく多いプロセスは，キャッシュの確保と解放ばかりが繰り返されており，キャッシュのヒット率が低く，頻繁にディスク I/O を発行していると推測できる．ウェブサーバのファイルキャッシュがそのような状態になっていれば，他のプロセスがファイルキャッシュを圧迫していると推測できる．したがって，他のプロセスに割り当てるキャッシュを減らすなどのポリシーが実現できる．

ユーザレベルのプロセスを用いて，キャッシュ濫用検出のポリシーが実現できるよう，各プロセスのキャッシュの確保量・解放量は，proc ファイルシステムを通じてユーザプロセスに公開するようにした．

3.2 キャッシュ濫用検出のポリシー

3.1 節で述べたキャッシュ濫用検出のメカニズムや，3.3 節で述べる濫用の防止策は，特定のキャッシュ濫用検出のポリシーに依存したものではなく，様々なポリシーの実現に利用できる．本節では，ポリシーの実現例として，4 節の実験で用いたポリシーを述べる．

ファイルキャッシュの確保量・解放量は，proc ファイルシステムを通じてユーザレベルで参照でき，様々なポリシーがユーザレベルで実現できる．4 節の実験で用いたポリシーでは，proc ファイルシステムを参照し，ファイルキャッシュの保持量 (確保量 − 解放量) が最も大きいプロセスをもって，キャッシュを濫用するプロセスとした．

ただし，プロセスの所有者がroot またはnobody である場合には，ファイルキャッシュの保持量が最大

であっても，ファイルキャッシュを濫用しているとはみなさないことにした．これは，UNIX では，ウェブサーバなどのサーバはroot またはnobody 権限で動作しているからである．

ポリシーを実現するユーザプロセスは，ファイルキャッシュの確保・解放状況を検査し終えたら約1秒間スリープするようにし，サーバの実行に与える影響を少なくするように配慮した．また，リソースを濫用するプロセスの影響によって，ポリシーを実現したプロセスの動作に支障が発生しないように，スケジューリングの優先度を高く設定した．

3.3 キャッシュ濫用の防止策

3.3.1 キャッシュの積極的破棄

ファイルキャッシュの濫用を検出した場合，検出ポリシーを実現したプロセスは，キャッシュを濫用しているプロセスのキャッシュを積極的に破棄するようにカーネルに指示する．この指示を受けたカーネルは，ファイルキャッシュの入れ替えアルゴリズムを切り替え，キャッシュを濫用しているプロセスのファイルキャッシュを犠牲者 (victim) として積極的に破棄するようにする．

Linux 2.2.13 をベースにした我々の実装では，次のように実装を行った．Linux 2.2.13 は，ファイルキャッシュのメモリを，LRU の近似的実装であるセカンドチャンスアルゴリズムで管理している．このアルゴリズムを変更し，キャッシュを濫用しているプロセスが確保したキャッシュは，used bit がオンであっても，セカンドチャンスを与えずただちに破棄するようにした．これによって，キャッシュを濫用するプロセスのキャッシュが積極的に破棄されるようになる．

3.3.2 スケジューリング優先度の低下

ファイルキャッシュを濫用しているプロセスは，ファイルに対するアクセスを頻繁に実行している．そのため，このプロセスが確保したファイルキャッシュを破棄しても，ただちに再確保してしまう可能性が高い．この問題に対処するため，キャッシュを濫用しているプロセスのnice 値を最大値 (20) に設定し，スケジューリング優先度を低下させて，ファイルキャッシュを継続的に確保しにくいようにした．

4 実験

ファイルキャッシュの濫用がウェブサーバに及ぼす影響と，本論文で提案するファイルキャッシュ濫用防止機構の有効性を検証する実験を行った．

4.1 ファイルキャッシュの濫用によるウェブサーバのスループットの低下

ファイルキャッシュを濫用するプロセスによって，実際にウェブサーバのスループットが低下することを確認するため，キャッシュを濫用するプロセスを実行した場合のスループットと，実行しなかった場合のスループットとの比較を行った．

実験では，Linux 2.2.13 をOS として使用し，ウェブサーバとして Apache 1.3.9 を用いた．スループットの測定にはSPECweb99[9] ベンチマークを用いた．本実験のSPECweb99 の測定では，ウェブサーバに同時接続するクライアント数を10 に設定した．SPECweb99 は，表1 に示すパターンでサーバにリクエストを送る．

転送要求のサイズ	リクエストに占める割合
1KB 以下	35%
10KB 以下	50%
100KB 以下	14%
1000KB 以下	1%

表 1: SPECweb99 のリクエストパターン

実験に用いたハードウェアの構成は，ウェブサーバ機が Pentium II 400MHz，メモリ 128MB，Ultra SCSI 4GB HDD，Ethernet 100Mbps であり，クライアント機が Pentium II 300MHz，メモリ 128MB，SCSI2 2GB HDD，Ethernet 100Mbps である．

ファイルキャッシュを濫用するプロセスは，あらかじめ用意したファイルをメモリマップし，繰り返しアクセスする．このファイルサイズを 0MB から 128MB まで 4MB ずつ変化させ，その都度スループットを測定した．なお，上述の通り 128MB はウェブサーバの動作する計算機の物理メモリ量に等しい．

図1 に，ファイルキャッシュを濫用するプロセスがある場合とない場合での，ウェブサーバのスループットを示す．

ファイルキャッシュを濫用するプロセスによるウェブサーバのスループット低下は，濫用プロセスがマ

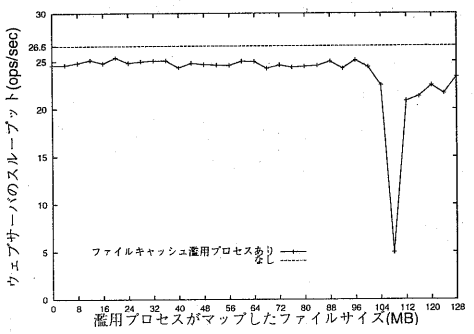

図 1: ファイルキャッシュの濫用によるスループットの低下

ップしたファイルサイズが 20MB のとき最も小さく 5%, 108MB のとき最も大きく 81%で, 平均は 11%であった.

4.1.1 比較した方式

提案方式の有効性を検証するため, 次の 5 通りの方式それぞれについてウェブサーバのスループットを測定した.

1. ORG: 提案方式を組み込んでいない Linux 2.2.13 である.
2. NON: ペナルティを与えない. 提案方式によるオーバーヘッドの調査が目的である.
3. FC: キャッシュを濫用しているプロセスのファイルキャッシュを積極的に破棄する. FC はファイルキャッシュの略である.
4. PRI: ファイルキャッシュを濫用しているプロセスのスケジューリングの優先度を下げる. PRI はプライオリティの略である.
5. FC+PRI: キャッシュを濫用しているプロセスのファイルキャッシュを積極的に破棄し, さらにそのスケジューリングの優先度を下げる. すなわち, FC と PRI の両者を行う.

なお, 提案方式を組み込んでいない Linux カーネルを用いた場合を, 実験に用いた環境は, 前節の実験における環境と同じである.

4.1.2 スループットの変化

図 2 に, ファイルキャッシュを濫用するプロセスがある場合のウェブサーバのスループットを示す.

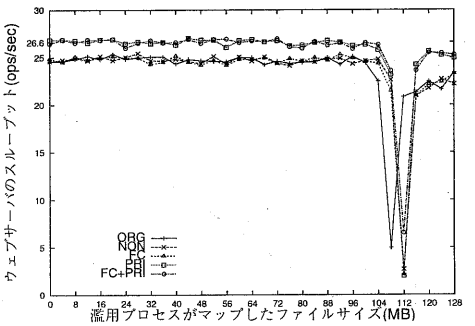

図 2: ファイルキャッシュの濫用によるスループットの変化

ファイルキャッシュ濫用プロセスがマップしたファイルサイズが 0MB から 104MB の間は, PRI と FC+PRI の場合, 濫用プロセスがない場合と比較して, スループットの低下を防ぐことができたことがわかる. しかし FC の場合, 濫用プロセスがない場合と比較してスループットは約 10%低下しており, ORG との差がなく, スループットの低下を抑制できていない. また, NON と ORG を比較するとスループットに大きな差はない. よって, 提案方式のオーバーヘッドは, ウェブサーバのスループットに影響を与えない程度に小さいと言える. この結果から, スケジューリング優先度を低下させる方式が有効であったのに対し, ファイルキャッシュを積極破棄する方式は効果が少ないことが知れる.

ファイルサイズが, ORG では 108MB のとき, NON, FC, PRI, FC+PRI では 112MB のとき, スループットが急激に低下している. FC+PRI の場合に ORG と比較して 5%, NON と比較して 15%, スループット低下を抑制しただけで, FC, PRI では全く効果がなかった.

ファイルサイズが 112MB を超えると, ウェブサーバのスループットが大きく回復する. スケジューリング優先度を下げる方式が効果的であるのは, ファイルサイズが 112MB より小さい場合と同様である.

4.1.3 ファイルキャッシュ占有量の変化

提案方式によってファイルキャッシュの濫用が防止できているかどうかを調べるため，図3に濫用プロセスの占有したファイルキャッシュ保持量を示し，図4に，ウェブサーバのファイルキャッシュ保持量を示す．

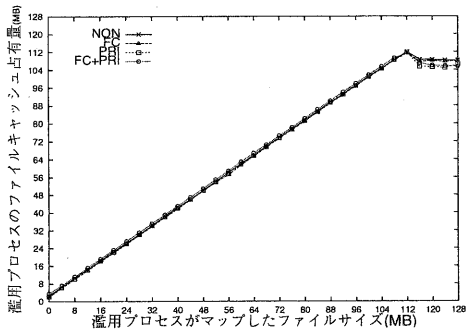

図 3: 濫用プロセスによるファイルキャッシュ占有量

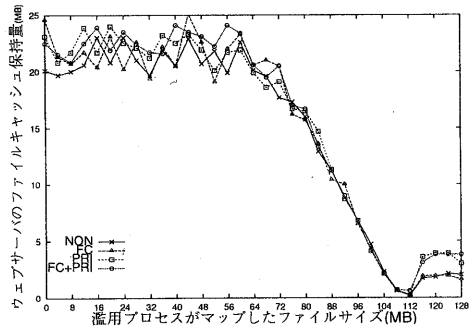

図 4: ウェブサーバのファイルキャッシュ保持量

図3によると，濫用プロセスによるファイルキャッシュの占有量は，ファイルキャッシュを優先的に破棄するFC, FC+PRI あっても，112MB までマップしたファイルサイズと同じ量を達成している．一方，ウェブサーバのファイルキャッシュ保持量は単調に減少し，濫用プロセスのマップしたファイルサイズが112MBの時ほぼ0MBになっている．したがって，提案方式ではファイルキャッシュの濫用を防止できていないことがわかる．

4.1.4 ファイルキャッシュの確保と解放

キャッシュを濫用するプロセスのファイルキャッシュが優先的に破棄されているかどうかを調べるため，濫用プロセスのファイルキャッシュの確保量と解放量とを測定した．その結果を図5に示す．

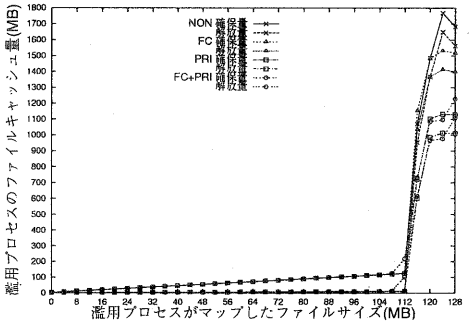

図 5: 濫用プロセスによるファイルキャッシュの確保と解放

ファイルキャッシュの積極破棄を行うFC とFC+PRI であっても，解放されたファイルキャッシュの量は増加していない．したがって，3.3.1 節で述べた実装では，ファイルキャッシュの積極破棄は実現できていないことがわかる．

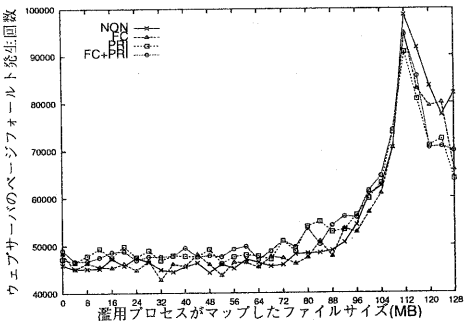

図 6: ウェブサーバのページフォールト発生回数

このようにファイルキャッシュの積極破棄の実装がうまく機能しなかったため，濫用プロセスのマップするファイルサイズが112MBのとき，ウェブサーバが頻繁にアクセスするファイルすらキャッシュから追い出されてしまっている．実験で利用したApache ウェブサーバでは，HTML ファイルをメモリマップ

しているため，ファイルキャッシュにないファイルをアクセスするとページフォールトが発生する．実際，図6に示したように，濫用プロセスのマップするファイルサイズが112MBのとき，スループットが急激に低下する前と比べてウェブサーバのページフォールトは約2倍に増加している．

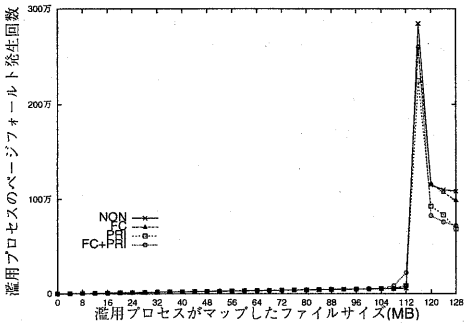

図 7: 濫用プロセスのページフォールト発生回数

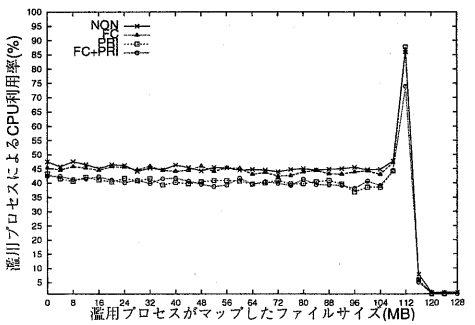

図 8: 濫用プロセスによる CPU 利用率の変化

濫用プロセスがマップしたファイルサイズが112MB を越えると，濫用プロセスによるファイルキャッシュの占有量は若干減少し，ウェブサーバのキャッシュ保持量はその分だけ回復している．これは，濫用プロセスがアクセスするファイルが，物理メモリに収まりきらなくなり，ファイルにアクセスしようとするたびに，ページフォールトが発生し，I/O待ちになるからである．図7に示したように，マップしたファイルサイズが112MB を超えると，濫用プロセスのページフォールト発生回数は200倍以上に増加している．また，図8に示したように，濫用プロセスの CPU 利用率が著しく低下している．

5 関連研究

本論文で想定している CGI を悪用した攻撃は，悪意ある外部プログラムがサーバのリソースを直接濫用する．このようなサービス不能攻撃の形態とは別に，ネットワーククライアントがサーバの弱点を攻撃し，サーバにリソースを濫用させるサービス不能攻撃の形態がある．

Banga ら [10] は，リソース割り当ての対象として，プロセスより粒度の小さいリソースコンテナという概念を用いることを提案している．リソースコンテナを用いると，I/O 割り込みを処理するスレッドの課金対象が明確になり，クライアントからの接続要求ごとにサーバのリソースを割り当てられる．この点を利用して，ひとつの接続要求が利用できるリソースの総量を規制すれば，サービス不能攻撃の防御に応用できる．リソースコンテナは，外部プログラムによるサービス不能攻撃は想定しておらず，本論文で想定する攻撃からの防御には利用できない．

Escort [11] は，Scout OS [12] の提供するパス機構を利用し，ネットワーククライアントからのサービス不能攻撃に対処することを目指している．パス機構とは，カーネル内の I/O 処理の流れを抽象化したものであり，パスごとにカーネルリソースの使用量の制限が可能である．信頼できないクライアントからの接続要求には，使用できるリソースが制限されたパスを割り当てサービス拒否攻撃を防ぐ．パス機構を用いた防御策は，パスごとに割り当てられる資源の濫用を防止するには有効であるが，ファイルキャッシュのようにすべてのプロセスで共有する資源の濫用には適用しにくい．

6 まとめ

本論文は，インターネットで一般的に利用されるようになった信頼できない外部プログラムが，正当なアクセス権を持つリソースを濫用して計算機の機能を損なわせる危険性を実験によって示した．ファイルキャッシュを濫用する CGI を想定した実験を行ったところ，ウェブサーバのスループットは 5%から81%低下した．

また，本論文はファイルキャッシュを対象にリソースの濫用を防止する手法を提案し，その有効性を検証した．提案した手法では，ファイルキャッシュを

濫用するプロセスに対し，ファイルキャッシュの積極破棄とスケジューリング優先度の低下というペナルティを与える．目的は，ウェブサーバがファイルキャッシュを確保できるようにし，濫用プロセスが破棄されたキャッシュを再確保するのを阻止することである．これを実装した下での再実験では，スループット低下の抑制に成功し効果がある場合もあったが，濫用プロセスによるファイルキャッシュの占有阻止には失敗した．ウェブサーバのスループット低下が最大となるとき，実装した機構では15%しか低下を抑制できなかった．

結論として，ファイルキャッシュの濫用を防止するために，以下のことが必要であるとわかった．(1) ファイルキャッシュを積極破棄するアルゴリズムを改良し，濫用プロセスのファイルキャッシュ占有量を減少させることができるようにする．(2) スケジューリングアルゴリズムを変更し，濫用プロセスによるファイルキャッシュの確保を阻止可能にする．

ファイルキャッシュ以外のリソース濫用防止について，本研究を応用できるかは今後の課題である．

参考文献

[1] Wahbe, R., Lucco, S., Anderson, T. E. and Graham, S. L.: Efficient Software-Based Fault Isolation, *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pp. 203–216 (1996).

[2] Ali-Reza, Adl-Tabatabai, Langdale, G., Lucco, S. and Wahbe, R.: Efficient and Language-Independent Mobile Programs, *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation*, pp. 127–136 (1996).

[3] Necula, G. C. and Lee, P.: Safe Kernel Extensions Without Run-Time Checking, *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation*, pp. 229–243 (1996).

[4] Takahashi, M., Kono, K. and Masuda, T.: Efficient Kernel Support of Fine-Grained Protection Domains for Mobile Code, *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pp. 66–73 (1999).

[5] 品川高廣, 河野健二, 高橋雅彦, 益田隆司: 拡張コンポーネントのためのカーネルによる細粒度軽量保護ドメインの実現, 情報処理学会論文誌 Vol.40 No.6, pp. 2596–2606 (1999 年).

[6] cker Chiueh, T., Venkitachalam, G. and Pradhan, P.: Integrating segmentation and paging protection for safe, efficient and transparent software extensions, *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pp. 140–153 (1999).

[7] Bershad, B. N., Savage, S., Pardyak, P., Sirer, E. G., Fiuczynski, M. E., Becker, D., Chambers, C. and Eggers, S.: Extensibility safety and performance in the SPIN operating system, *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (1995).

[8] Seltzer, M. I., Endo, Y., Small, C. and Smith, K. A.: Dealing with disaster: surviving misbehaved kernel extensions, *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation* (1996).

[9] Standard Performance Evaluation Corporation: *SPECweb99 Benchmark*. http://www.spec.org/osg/web99/.

[10] Banga, G., Druschel, P. and Mogul, J. C.: Resource Containers: A New Facility for Resource Management in Server Systems, *Proceedings of the Third USENIX Symposimum on Operating Systems Design and Implementation*, pp. 45–58 (1999).

[11] Spatscheck, O. and Peterson, L. L.: Defending Against Denial of Service Attacks in Scout, *Proceedings of the Third USENIX Symposimum on Operating Systems Design and Implementation*, pp. 59–72 (1999).

[12] Mosberger, D. and Peterson, L. L.: Making Paths Explicit in the Scout Operating System, *Proceedings of the Second USENIX Symposimum on Operating Systems Design and Implementation*, pp. 153–167 (1996).