Unixドメインソケットを用いた強制アクセス制御の実装

喜多 陽花[1,2,a)]　石川 裕[2,4,b)]　竹房 あつ子[2,3,c)]　小口 正人[1,d)]

概要：不正アクセスを防ぐためのアクセス制御の必要性が高まっている．本研究では，ユーザ空間で強制アクセス制御を実現する認証・認可デーモンを実装する．本実装では，管理者がアクセス制御リストを作成し，アプリケーションの証明書を Linux 拡張ファイル属性として付与することで認証を行う．また，Linux の seccomp 機能を用いて，アプリケーションプロセスが直接 socket や bind，connect を呼び出せないように制限する．アプリケーションプロセスは，Unix ドメインソケットでデーモンと接続し，これらシステムコールの実行を依頼する．デーモンは，要求を受けた際にアプリケーションの証明書を検証，アクセス制御リストを確認し，認証・認可された場合に限り，これらのシステムコールを処理する．これにより，アプリケーションは受け取ったファイルディスクリプタを利用して通信を行うことが可能となる．本システムの実装を通じて，アプリケーションコードを変更することなく，ユーザ空間においてアプリケーション単位で細かくアクセスポリシを設定可能な，柔軟なアクセス制御が実現できることを示す．

キーワード：強制アクセス制御，Unix ドメインソケット，seccomp，Linux 拡張ファイル属性

Implementation of Mandatory Access Control using Unix Domain Sockets

Kita Haruka[1,2,a)]　Ishikawa Yutaka[2,4,b)]　Takefusa Atsuko[2,3,c)]　Oguchi Masato[1,d)]

Abstract: There is an increasing need for access control to prevent unauthorized access. In this research, we implement an authentication and authorization daemon that realizes mandatory access control in user space. In this implementation, authentication is performed by having the administrator create an access control list and assigning the application's certificate as a Linux extended file attribute. Furthermore, the Linux seccomp function is used to restrict application processes from directly calling socket(), bind(), and connect(). An application process connects to the daemon via a Unix domain socket and requests the execution of these system calls. Upon receiving a request, the daemon verifies the application's certificate, checks the access control list, and processes these system calls only if the application is authenticated and authorized. This allows the application to communicate using the received file descriptor. Through the implementation of this system, we demonstrate that flexible access control can be achieved in user space, allowing fine-grained access policies to be set on an application-by-application basis, without modifying the application code.

Keywords: Mandatory Access Control, Unix Domain Sockets, seccomp, Linux Extended File Attributes

1. はじめに

我々は前回の研究 [1], [2] において，Unix ドメインソケットを活用し，アプリケーションからの直接的なリソースア

[1] お茶の水女子大学
Ochanomizu University, Bunkyo, Tokyo 112–8610, Japan
[2] 情報・システム研究機構 データサイエンス共同利用基盤施設 セキュアコンピュータシステム研究開発センター
Research Organization of Information and Systems Joint Support Center for Data Science Research, Center for R&D on Secure Computer Systems, Chiyoda, 101–0051, Tokyo
[3] 情報・システム研究機構 国立情報学研究所
National Institute of Informatics, Chiyoda, 101–8430, Japan
[4] 大妻女子大学
Otsuma Women's University, Chiyoda, 102–8357, Tokyo
[a)] g2120513@is.ocha.ac.jp
[b)] ishikawa.yutaka@rois.ac.jp
[c)] takefusa@nii.ac.jp
[d)] oguchi@is.ocha.ac.jp

クセスを防止する強制アクセス制御システム MACUDS を設計し，実装した．本論文では，MACUDS をさらに実装し，性能評価を行ってその実用性を検証する．

MACUDS が想定する脅威は，主に二つある．第一に，正規のバイナリが改ざんされ，外部サーバと TCP 通信を行うケースである．例えば，開発者や攻撃者が正規のバイナリに外部通信処理を挿入し，利用者がそれを実行することで情報が外部へ送信される場合が挙げられる．第二に，不正なソフトウェアが持ち込まれ，外部と通信を試みるケースである．典型的には，フィッシングメールなどを通じて侵入したプログラムが実行され，認証情報を外部に送信する場合である．これらの脅威に対して，本研究では実行バイナリに対する認証・認可機構を導入し，TCP 通信を行う際には正規のバイナリのみが利用できるようにする．

従来研究としては，SELinux[3] に代表される強制アクセス制御機構が存在し，プロセスやファイルに対して詳細なアクセス制御を可能にするが，バイナリの改ざんには対応できない．一方で，Linux IMA（Integrity Measurement Architecture）[4] は，ファイルに対してリアルタイムにハッシュ値を計算・記録し，TPM（Trusted Platform Module）を用いて測定値の改ざん耐性を確保することができる．さらに，Appraisal 機能により，保存されたハッシュや署名と照合して，改ざんされたバイナリの実行を制御できる．しかし，これらの技術はカーネルレベルで動作している．本研究では，正規のバイナリだけが外部通信できる仕組みをユーザ空間で実装する．

本論文では，まず 2 章で既存のアクセス制御システムの概要と課題について説明する．次に，3 章で MACUDS を実現するための関連技術について紹介する．4 章では MACUDS の実装について説明する．その後の 5 章で MACUDS の性能評価について説明する．最後に，本論文の結論と今後の展望についてまとめる．

2. 関連研究

カーネル空間の実装として，SELinux，TOMOYO Linux[5]，AppArmor[6]，Smack[7] がある．SELinux は，TE（Type Enforcement），RBAC（Role-Based Access Control），MLS（Multi-Level Security）の 3 つのアクセス制御モデルを実装しており，用途に応じてアクセスを管理する [8]．しかし，SELinux はポリシ設定が複雑で，動的なポリシ変更に対応していないという課題がある．TOMOYO Linux は，アクセス要求の記録と自動学習により直感的なポリシ管理を提供する．しかし，プロセスの実行履歴に基づいてドメインを生成するため，ドメイン数が非常に多くなり，結果としてポリシ数も膨大になる可能性がある．AppArmor は，アプリケーションごとにプロファイルを定義し，アクセスを制御する．しかし，AppArmor はネットワークソケットに対する柔軟なアクセス制御を提供していない [9]．Smack は，シンプルさを重視したシステムで，ラベルに基づいてアクセスを管理する．しかし，シンプルなラベルベースの制御モデルであるため，複雑な条件に基づいたアクセス制御ができないという制約がある．

ユーザ空間における実装としては，WEB アプリケーションのアクセス制御に用いられる OAuth（Open Authorization）プロトコル [10] や，Publish/Subscribe 通信モデルを実現する DDS（Data Distribution Service）のセキュリティ拡張（DDS Security）[11] がある．OAuth では，クライアントが認可サーバと通信してアクセストークンを取得し，そのトークンを使用して Web アプリケーションにアクセスする仕組みを提供する．ユーザが関与することを前提としており，人が関与しないアプリケーション同士の認証や認可には対応していない．一方，DDS Security では，Identity 認証局が発行した証明書と Permissions 認証局が署名したアクセス制御リスト（ACL）を使用して認証・認可を行う．DDS では認証認可された Participant 同士で共通暗号鍵を保持して暗号通信を行っており，認証認可されていない Participant(通信者) は UDP パケットが読めたとしても復号できないようになっている．このように DDS では UDP 通信を使用しており，データは暗号化されるため，単一計算機上の通信でもオーバヘッドが発生する．

3. 関連技術

MACUDS を実現するための関連技術について説明する．実装においては，Unix ドメインソケット，seccomp 機能，拡張ファイル属性，`LD_PRELOAD` 環境変数を利用する．

3.1 Unix ドメインソケット

Unix ドメインソケットは，同一ホスト内でのプロセス間通信（IPC: Inter-Process Communication）を効率的に実現する手法である [12]．ネットワークソケットと似たインターフェースを持ちながら，TCP/IP プロトコルを使用せず，カーネル内で直接データを転送する点が特徴である．Unix ドメインソケットでは，ファイルシステム上のパス名または抽象的な名前空間を用いて通信先を指定する．

Unix ドメインソケットでは，`sendmsg()` と `recvmsg()` を通じてメッセージの本体とは別に補助データ（ancillary data）を使用することができる．補助データを使用してファイルディスクリプタやプロセス認証情報（プロセス ID，ユーザ ID，グループ ID など）を他のプロセスに転送する機能を提供する．特に，ファイルディスクリプタの転送は，プロセス間でリソースを共有する場合に有用である．例えば，あるプロセスが開かれたソケットのファイルディスクリプタを別のプロセスに転送することで，ファイルディスクリプタを受信したプロセスは通信を継続できる．さらに，補助データに含まれるプロセス認証情報は，プロセス間のセキュリティ情報として利用できる．本研究

では，アプリケーションと認証・認可デーモンとの間の通信に利用する．

3.2 seccomp 機能

seccomp 機能は，Linux カーネルにおいてプロセスのシステムコールを制限するための機能である．主に，プロセスが意図しないシステムコールを呼び出さないように制御するために使用される．seccomp の設定は設定したプロセスだけでなく子プロセスにも継承される．

seccomp には, strict モード（SECCOMP_SET_MODE_STRICT）と filter モード（SECCOMP_SET_MODE_FILTER）の 2 種類のモードが存在する [13]．strict モードでは，限定的なシステムコール（例：read，write，_exit など）のみが許可される．

filter モードでは，カスタムフィルタルールを定義して，どのシステムコールを許可または拒否するかを厳密に制御できる．例えば，SCMP_ACT_NOTIFY フィルタルールを使用すると，システムコールを傍受してユーザー空間プロセスに渡し，ユーザー空間プロセスがそれを許可するか拒否するかを決定できる．本研究では，アプリケーションが socket や bind，connect，close といったシステムコールを直接呼び出せないよう制限し，LD_PRELOAD でロードした seclib による処理を経由させるために seccomp 機能を使用する．

3.3 拡張ファイル属性

拡張ファイル属性（Extended Attributes）は，ファイルやディレクトリに恒久的に関連付けられた属性名と値のペアである [14]．これらは，システム内のすべての inode に関連付けられた通常の属性（例：stat() データ）を拡張するもので，ファイルシステムに追加の機能を提供するために使用される．例えば，アクセス制御リストなどの追加のセキュリティ機能は，拡張ファイル属性を使用して実装されることがある．

拡張属性の読み書きにはシステムコールが用意されている．setxattr を実行すると，値を設定することができる．getxattr を実行すると属性の値全体を取得し，それをバッファに格納する．listxattr を実行すると，そのファイルまたはディレクトリに定義されている属性名のリストを取得できる．本研究では，アプリケーションの認証を行う際に，署名データを保存するために拡張ファイル属性を利用する．

3.4 LD_PRELOAD 環境変数

LD_PRELOAD 環境変数は，他のすべての共有ライブラリより優先的にロードされる，追加のユーザ指定の共有ライブラリのリストである [15]．プログラムを実行すると，まずプログラムに必要な共有ライブラリがロードされ，実行準備が整った後にプログラムが実行される．LD_PRELOAD を使用すると，指定された共有ライブラリが他の共有ライブラリよりも優先的にロードされる．

この機能は, 他の共有ライブラリの関数を選択的に上書きするために使用できる．例えば，socket() の動作をカスタマイズしたい場合, 独自の socket() を定義したライブラリを LD_PRELOAD に設定してロードすれば，既存の socket() を上書きできる．本研究では socket() や bind() などのライブラリ関数の動作を独自の処理に置き換えるために利用する．

4. MACUDS の実装

本章では，第 3 章で説明した技術を用いて，MACUDS と呼ばれるユーザ空間認証・認可デーモンを実装する．MACUDS では，アプリケーションが認証・認可を必要とするシステムコールライブラリを呼び出す際に，カーネル呼び出しをトラップするのではなく，デーモンに要求を送信することで，認可されたアプリケーションのみが通信できる仕組みを提供する．このデーモンの特徴は，アクセスポリシを動的に変更できることであり，アプリケーションのソースコードを変更することなくアクセス制御を適用できる．

図 1 に MACUDS の処理の流れを示す．ここでは，同一計算機上でクライアント・サーバアプリケーションが動作している環境を例としている．

4.1 アプリケーションの署名

管理者はアクセス制御リストを作成し，アプリケーションへのアクセス権限を設定する．また，アプリケーションの証明書を作成し，それをアプリケーションファイルの拡張ファイル属性に設定する．具体的には，RSA 暗号を用いて秘密鍵および公開鍵の鍵ペアを生成し，秘密鍵を用いてアプリケーションファイルのハッシュ値に対して暗号化して証明書を作成する．この署名データは，Base64 エンコードされた形式で拡張ファイル属性に設定される．

4.2 サーバプロセス・クライアントプロセスの起動

4.2.1 サーバプロセスの起動

サーバアプリケーション起動プロセスは Linux の seccomp 機能を用いて，アプリケーションが socket，bind，close といったシステムコールを sysenter 機械語などで直接呼び出せないよう制限する．

サーバプロセスを起動する際に，LD_PRELOAD を使用して，別のライブラリを優先的にロードする．このライブラリを seclib と呼び，socket()，bind()，close() の関数呼び出しがフックされ，seclib で定義される処理を行うようになる．

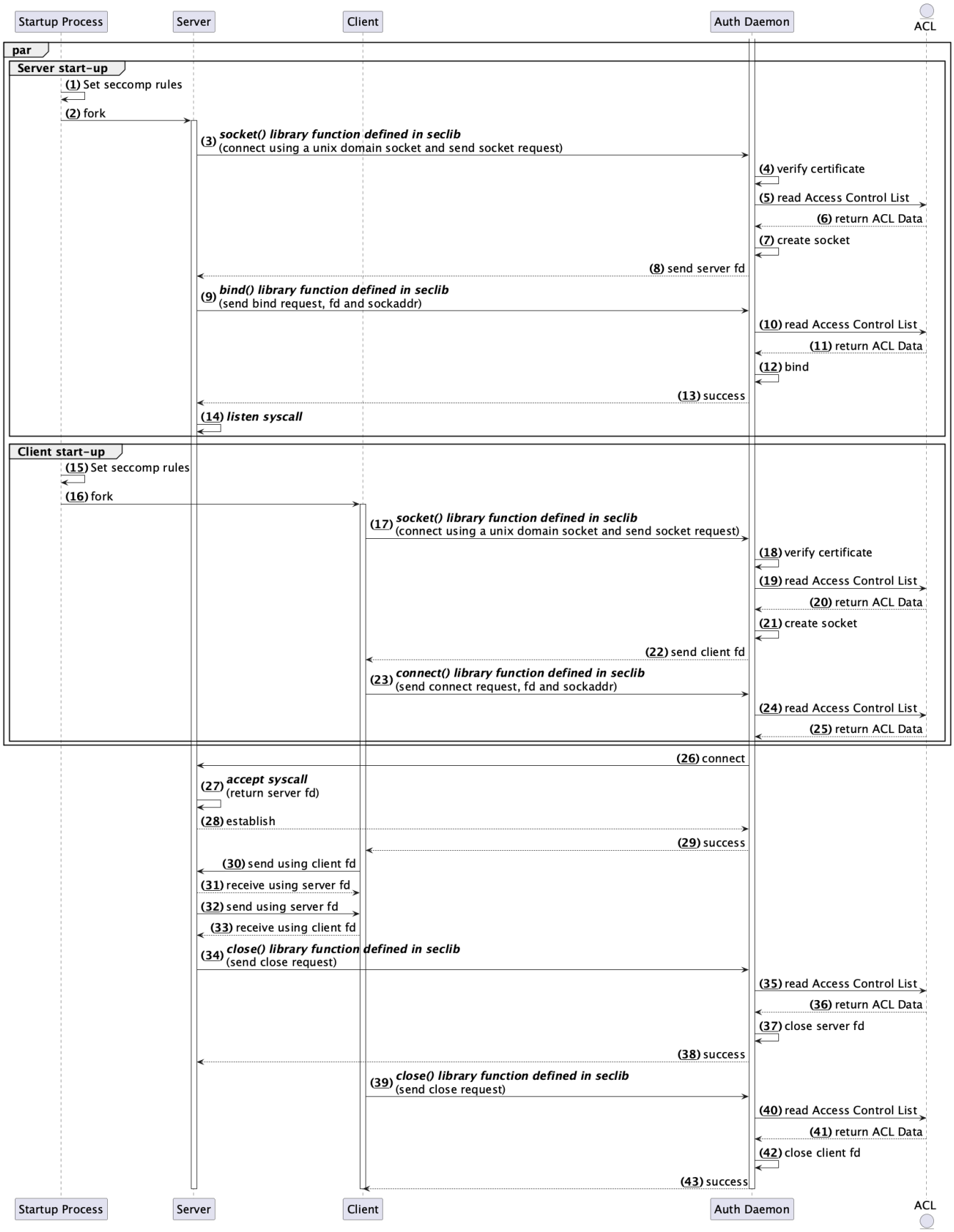

図 1 MACUDS の処理の流れ

4.2.2 クライアントプロセスの起動

クライアントプロセスも同様に，seccomp により socket，connect，close システムコールの直接呼び出しが制限される．起動時に `LD_PRELOAD` で seclib をロードすることで，ク

ライアントプロセスでも socket()，connect()，close() の関数呼び出しがフックされる．

4.3 デーモン接続

アプリケーションが socket() ライブラリ関数などを呼び出した際，デーモンとの Unix ドメインソケット接続が未確立であれば，その時点で接続を確立する．デーモンは accept システムコールでアプリケーションからの接続を受け付け，コネクション毎に専用スレッドを生成する．生成されたスレッドは，サーバ向けスレッドとクライアント向けスレッドに分かれ，それぞれのアプリケーションの要求を処理する．以降，サーバアプリケーションの処理をするスレッドをサーバ向けスレッド，クライアントアプリケーションの処理をするスレッドをクライアント向けスレッドと呼ぶ．

4.4 認証・認可要求

スレッド生成後，デーモンはまずアプリケーションの認証を行う．デーモンはアプリケーションから送信されたメッセージから recvmsg() を用いてサーバプロセス・クライアントプロセスのプロセス ID を取得する．これにより，プロセス ID の偽造は防止される．デーモンはプロセス ID からアプリケーションファイルの拡張属性に格納されている証明書を取り出し，公開鍵を用いて署名を検証することで，アプリケーションが正規のものであることを確認する．

認証が成功すると，デーモンはアプリケーションからの要求を受け取り処理する．受信可能なコマンドは以下の 4 種類である：

- CMD_SOCKET：デーモン側で新たなソケットを生成し，対応するファイルディスクリプタを返す．
- CMD_BIND：アプリケーションから受け取ったソケットアドレス構造体とファイルディスクリプタを用いて bind() を実行する．
- CMD_CONNECT：同様に，connect() 処理をデーモン側で代行する．
- CMD_CLOSE：デーモン側で，対応するファイルディスクリプタを全て閉じる．

アプリケーションからコマンドを受け取ると，デーモンは ACL を確認し，必要な権限があるか検証する．ACL の記述形式は現在設計中であり，今後の実装を通じて具体的方針を決定する予定である．

ファイルディスクリプタ（FD）はプロセスごとに異なる整数値を持つ．例えば，デーモンで生成されたソケットが FD=4 でその値を sendmsg() の補助データでアプリケーションに送信しても，アプリケーション側は異なる値，例えば FD=5，として受信する．このとき，デーモンで作成されたソケットに対してアプリケーション側が何かの操作をしたい場合，アプリケーション側で FD=5 を指定しても，デーモン側では FD=7 などが割り振られてしまう可能性がある．この問題に対処するため，デーモンとアプリケーション間で FD の対応を管理する表 (表 1，表 2) を導入した．対応表では以下のように区別する：

- **デーモン側 FD**：デーモンがソケット作成時に割り当てられる整数値
- **アプリケーション側 FD**：補助データ経由でアプリケーションが受け取る FD
- **Unix ドメインソケット FD**：デーモンがアプリケーションとの通信に使用する FD

なお，ここでの対応表は実際には構造体で管理している．

表 1 アプリケーションが持つ対応表

アプリケーション側 FD	デーモン側 FD
5	4
6	7
4	6

表 2 デーモンが持つ対応表

デーモン側 FD	Unix ドメインソケット FD
4	6
7	5
8	4

4.5 socket 要求の処理

アプリケーションが seclib に定義された socket() を呼ぶと，カーネルの socket システムコールは直接呼び出されず，Unix ドメインソケットを介してデーモンにソケット作成要求が送信される．デーモンは認可を行った後，ソケットを作成する．デーモンは作成したソケットの FD をデーモン側 FD として，現在アプリケーションとの通信に使用している FD を Unix ドメインソケット FD として対応表に記録する．また，作成したソケットの FD をメッセージ本体と補助データの両方に入れて送信する．アプリケーション側では，受け取ったデーモン側 FD（メッセージ本体）とアプリケーション側 FD（補助データ）を対応表に記録する．これにより，以降の bind() や connect() 呼び出し時に，デーモン側 FD を対応表から取得して送信できる．

4.6 bind，connect 要求

アプリケーションが seclib の bind() を呼び出すと，カーネルの bind システムコールは直接呼び出されず，以下の情報がデーモンに送信される：

- コマンド（bind 要求）
- デーモン側 FD（対応表上の値）
- ソケットアドレス構造体

デーモンはサーバ向けスレッドで認可を行った後，受け取ったデーモン側 FD とソケットアドレス構造体を用いて bind() を実行する．実行結果（成功・失敗）はサーバプロ

セスに通知される．その後，サーバプロセスは `listen()` を呼び出して，他プロセスからの接続を待機する．

`connect()` も同様の手順で処理される．クライアント向けスレッドが認可を行った後，対応表からデーモン側 FD を取得し，受信したソケットアドレス構造体とともに `connect()` を実行する．サーバプロセスは `accept()` を呼び出して接続用の FD を取得する．デーモンは `connect()` の実行結果（成功・失敗）をクライアントプロセスに通知する．

これにより，サーバプロセスとクライアントプロセス間で通信可能な接続が確立される．

4.7 close 要求

アプリケーションが seclib の `close()` を呼び出すと，カーネルの close システムコールは直接呼び出されず，close 要求がデーモンに送信される．

デーモンは close 要求を受け取ると，対象プロセスとの通信に使用している Unix ドメインソケット FD を参照し，対応表に記録されているデーモン側 FD をすべて閉じる．対応表からも削除する．

5. 評価

本研究で実装したライブラリと認証・認可デーモンを用いた場合の TCP ソケット通信におけるシステムコールのオーバヘッドを，通常の TCP ソケット通信と比較して評価する．

5.1 実験環境

表 3 に本実験で使用した実験機器およびソフトウェア構成を示す．測定は Docker 上で単一マシン内で行った．

表 3 実験機器の構成

CPU	Apple M4, 10 cores
メモリ	16 GiB
ホスト OS	macOS 15.6 (Build 24G84)
Docker	Docker 28.0.4, Docker Compose v2.34.0-desktop.1
コンテナ OS	Ubuntu 22.04.5 LTS (Jammy Jellyfish)

5.2 評価手法

本評価では，実行時に seclib で置き換えたライブラリ関数（`socket()`，`bind()`，`connect()`，`close()`）の処理時間を測定した．各操作は 10 回実行し，その平均値を算出した．また，同条件下で通常の TCP ソケット通信における処理時間も測定し，オーバヘッドを比較した．

図 1 に示す処理フローに対応させると，測定対象は以下の通りである．

- サーバ `socket()`：(3)-(8)
- サーバ `bind()`：(9)-(13)
- サーバ `close()`：(34)-(38)
- クライアント `socket()`：(17)-(22)
- クライアント `connect()`：(23)-(29)
- クライアント `close()`：(39)-(43)

5.3 結果

本研究で実装した seclib を用いたライブラリ関数と通常の TCP ソケット通信における処理時間を図 2 に示す．なお，縦軸は対数スケールである．

本手法（w/ MACUDS）のオーバヘッドは，特に `socket()` 呼び出しで顕著に現れ，w/o MACUDS と比較して約 50〜60 倍に達することが確認された．この要因としては，デーモン側で行われる認証処理が主なボトルネックであると考えられる．サーバアプリケーションでは，`bind()` や `close()` の操作においてもオーバヘッドが生じており，`bind()` で約 5.8 倍，`close()` で約 1.4 倍の増加が見られた．クライアントアプリケーションでも同様に，`connect()` で約 1.5 倍，`close()` で約 4.8 倍のオーバヘッドが確認された．

しかしながら，インターネットアクセスを行うアプリケーションにおいては，ネットワーク遅延が支配的であるため，システムコール単体でのオーバヘッドは実運用上ほとんど影響を与えないことが予想される．特に Web アクセスなどでは，ミリ秒単位の遅延は無視できる．このことから，本手法の適用による性能低下はアプリケーション依存であり，評価においてはアプリケーションレベルでの実測が必要になる．

6. まとめ

本研究では，Unix ドメインソケットおよび Linux の seccomp 機能を活用し，ユーザ空間で動作するアクセス制御システム MACUDS の実装と評価を行った．アプリケーションの署名や拡張ファイル属性を用いた認証機構，seccomp によるシステムコール制限，`LD_PRELOAD` によるライブラリ関数のフックを組み合わせることで，アプリケーションがデーモンに許可された場合にのみ通信を行える仕組みを実現した．これにより，従来のカーネル空間でのアクセス制御における複雑なポリシ設定や動的変更の難しさといった課題を解決し，柔軟なアクセス制御を提供できることを示した．また，本研究は open システムコールなどにも適用することで，ファイルアクセスに対しても制御を適用できる．今後は認可機能の実装と評価を行い，システムの実用性を検証する予定である．

7. 謝辞

本研究は，JST 経済安全保障重要技術育成プログラム【JPMJKP24U4】の支援を受けた．

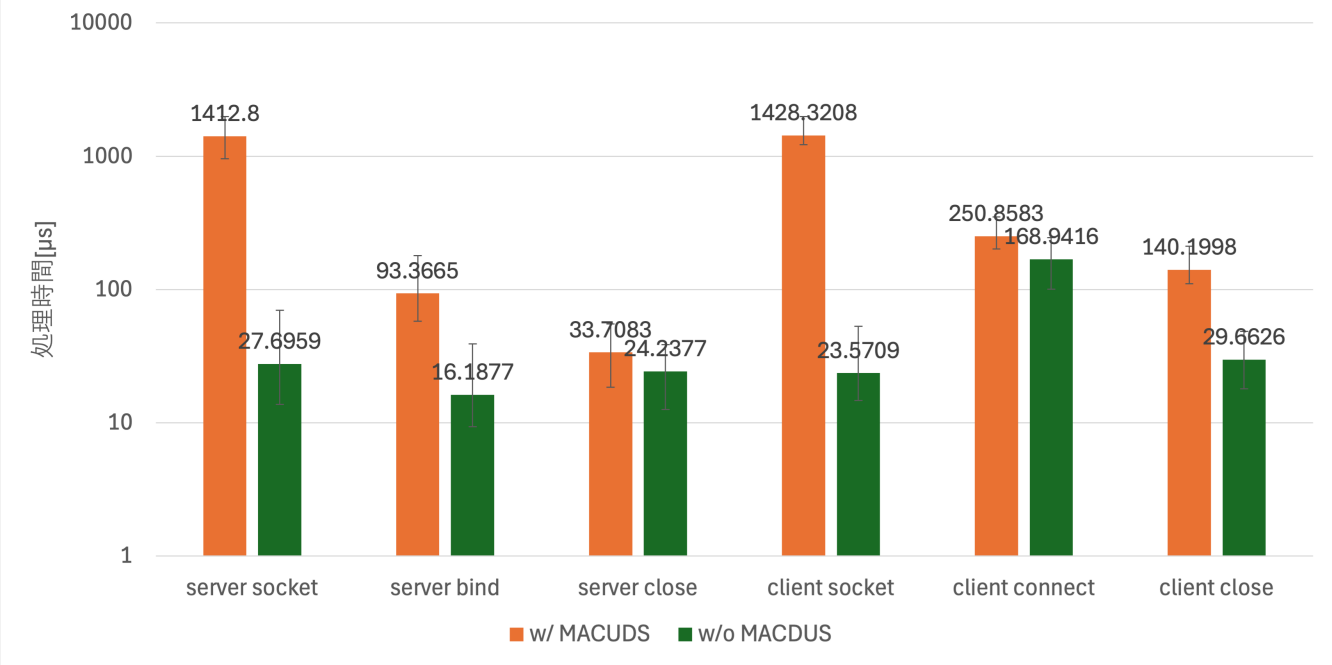

図 2 実験結果

参考文献

[1] 喜多陽花, 石川裕, 竹房あつ子, 小口正人: Unix ドメインソケットと seccomp を用いた強制アクセス制御実現の設計, 第 17 回データ工学と情報マネジメントに関するフォーラム (DEIM2025) (2025).

[2] 喜多陽花, 石川裕, 竹房あつ子, 小口正人: Unix ドメインソケットと seccomp を用いた強制アクセス制御実現の設計と実装, 第 167 回システムソフトウェアとオペレーティング・システム研究発表会 (2025 年 5 月研究発表会) (2025).

[3] SELinux Project: SELinux Notebook, `https://github.com/SELinuxProject/selinux-notebook` (2024).

[4] 宗藤, 須崎有康: Linux のセキュリティ機能: 5. 高信頼を実現する Linux の新しい機能, 情報処理, Vol. 51, No. 10, pp. 1284–1293 (2010).

[5] Harada, T., Horie, T. and Tanaka, K.: Task oriented management obviates your onus on Linux, *Linux Conference*, Vol. 3, p. 23 (2004).

[6] : About AppArmor, `https://gitlab.com/apparmor/apparmor/-/wikis/About`.

[7] : Description from the Linux source tree, `https://schaufler-ca.com/description_from_the_linux_source_tree`.

[8] 海外浩平ほか: Linux のセキュリティ機能: 2. SELinux のアーキテクチャとアクセス制御モデル, 情報処理, Vol. 51, No. 10, pp. 1257–1267 (2010).

[9] : Ubuntu manpage: apparmor.d, `https://manpages.ubuntu.com/manpages/focal/man5/apparmor.d.5.html`.

[10] Hardt, D.: The OAuth 2.0 Authorization Framework, RFC 6749 (2012).

[11] Object Management Group: DDS Security Specification 1.2, `https://www.omg.org/spec/DDS-SECURITY/1.2/PDF` (2024).

[12] : unix(7) —Linux manual page, `https://man7.org/linux/man-pages/man7/unix.7.html`.

[13] Linux Kernel Organization: Seccomp BPF (SECure COMPuting with filters), `https://www.kernel.org/doc/html/v5.1/userspace-api/seccomp_filter.html`.

[14] : xattr(7) — Linux manual page, `https://man7.org/linux/man-pages/man7/xattr.7.html`.

[15] : ld.so(8) — Linux manual page, `https://man7.org/linux/man-pages/man8/ld.so.8.html`.