C/C++ セキュアコーディング

File I/O part1 :
UNIXの権限とパーミッション

2010年3月23日

JPCERTコーディネーションセンター

File I/O part1: UNIXの権限とパーミッション

JPCERT CC®

Part 1

パーミッションと権限

- **ユーザ識別所有者権限パーミッション**
- **プロセスの権限**
- **権限の変更と管理**
- **パーミッションの管理**
- **脅威の緩和方法**

ここ

Part 2

ディレクトリトラバーサル

- ファイルシステムのおさらい
- ファイルパス、シンボリックリンクとハードリンク
- ディレクトリトラバーサ
- chroot(), jail()

UNIX

Part 3

Race Condition

- 並列処理
- 競合状態・排他制御・デッドロック
- ロックファイルとファイルロック
- ファイルシステムへの攻
  ・脅威の緩和方法

POSIX

本モジュールのゴール

以下を理解・習得すること:

- UNIXにおける権限管理
- ファイルI/Oにまつわる危険
- 安全なファイルI/Oのためのノウハウ

目次

- ユーザ、ファイル、プロセス
- 権限の変更
- 権限の管理
- ファイルパーミッションの管理
- 脅威の緩和方法
- まとめ

C/C++ とファイル入出力

- ファイル入出力はOS環境の権限管理のもとに行われる
- C/C++言語仕様は、特定のファイルシステムに依存しないよう設計されている

OSやファイルシステムによって権限管理モデルは異なる．
例えば……

- Unix UFSやNFS(v3)
- OSのユーザ権限およびファイルパーミッション
- AFS(Andrew File System)やDCE/DFS
- ユーザ権限およびACL(アクセスコントロールリスト)

- セキュアなプログラムをつくるために知っておくべきこと
  - 言語仕様上の問題点とその対策について
  - 動作環境における問題点とその対策について

このセミナーでは、POSIX環境を前提として話をします．

UNIX環境におけるユーザ管理

- 複数のユーザがOS環境を使う
  - ユーザやプログラムが、他ユーザやシステムのデータを故意に(あるいは無意識に)変更できないように権限管理
  - 必要に応じてユーザプロセスに特権を与える仕組みがある(set*id関数)

ユーザの識別: ユーザ ID(UID)とユーザ名
- /etc/passwd
- UID = 0 (root) はすべてのファイルにアクセスできる特別な存在

ユーザ認証: パスワード
- /etc/passwd、/etc/shadow、/etc/master.passwd など

グループ: グループID(GID)とグループ名
- /etc/group
- 全てのユーザはなんらかのグループに属する(initial group, supplementary groups)

UNIXファイルシステムのファイルと権限

- 各ファイルには以下のような属性情報がついている
  - 所有者(UID)
  - グループ(GID) リスト
  - permission (所有者が設定する。 所有者あるいはrootだけが変更可能。)
    - 対象: 所有者(User)、グループ(Group)、その他のユーザ(Others)
    - アクセス: 読み取り(Read)、書き込み(Write)、実行(eXecute)
    - setuidビット、setgidビット
    - スティッキービット

UNIX のファイルパーミッション(1)

JPCERT CC®

-rwsr-xr-x 1 root root 28480 Feb 27 2007 /usr/bin/passwd

-rwxr-xr-x 1 root root 258896 Sep 16 2008 /usr/bin/ssh

-rwxr-sr-x 1 root tty 10456 Dec 22 2007 /usr/bin/wall

-rwxr-x--- 1 yozo yozo 1288172 Dec 19 2008 /home/yozo/bin/ssh

lrwxrwxrwx 1 root root 13 Feb 29 2008 /etc/motd -> /var/run/motd

drwxrwxrwt 73 root root 16384 Oct 14 19:02 /tmp

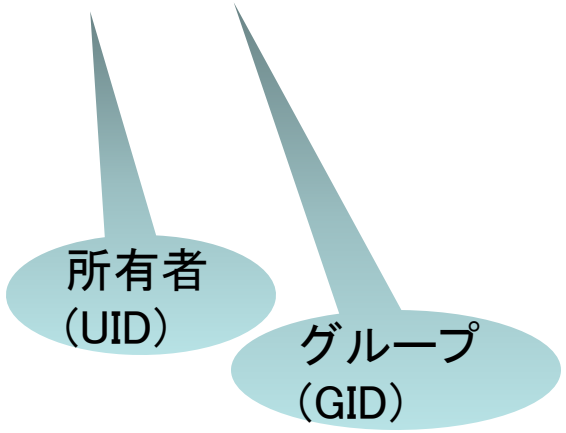
ファイルの種類: 通常ファイル -、ディレクトリ d、 シンボリックリンク l、デバイス b/c、 ソケット s、fifo f/p

permission: r、w、x、s (set*id)、t(スティッキー)

UNIX のファイルパーミッション(2)

ファイルのアクセス制御は、プロセスの実効ユーザID (EUID)や実効グループID (EGID)を、対象ファイルのUID/GID と比較して行われる

-rwxr-x--- 1 yozo staff 1288172 Dec 19 2008 /home/yozo/bin/ssh



- ユーザyozoのプロセスは読み/書き/実行が可能
- グループstaffのプロセスは読み/実行が可能
- それ以外のプロセスは読み/書き/実行すべて不可

UNIX のプロセスと権限

各プロセスは以下のような属性情報を持っている

- プロセスID、親プロセスID
- 所有者(UID)、グループ(GID)リスト
- cwd(current working directory)、環境変数リスト
- など……

カーネルはUID,GIDに基づきプロセスのアクセス制御を行う

プロセスが持つ三つのユーザID(とグループID)

JPCERT CC®

※各ユーザIDは親のユーザIDからコピーされるのが基本

- Real UID(RUID)
- Effective UID(EUID)
  - カーネルがプロセスに対するパーミッションを決定する際に参照する
  - setuidプログラムの実行時には実行ファイルの所有者ID
- Saved-set-UID(SUID)
  - setuidプログラムの実行時には実行ファイルの所有者ID

※グループIDについても同様。ただし、supplementary group list があり、EGIDとともにアクセス制御に使われる。

プロセスの生成

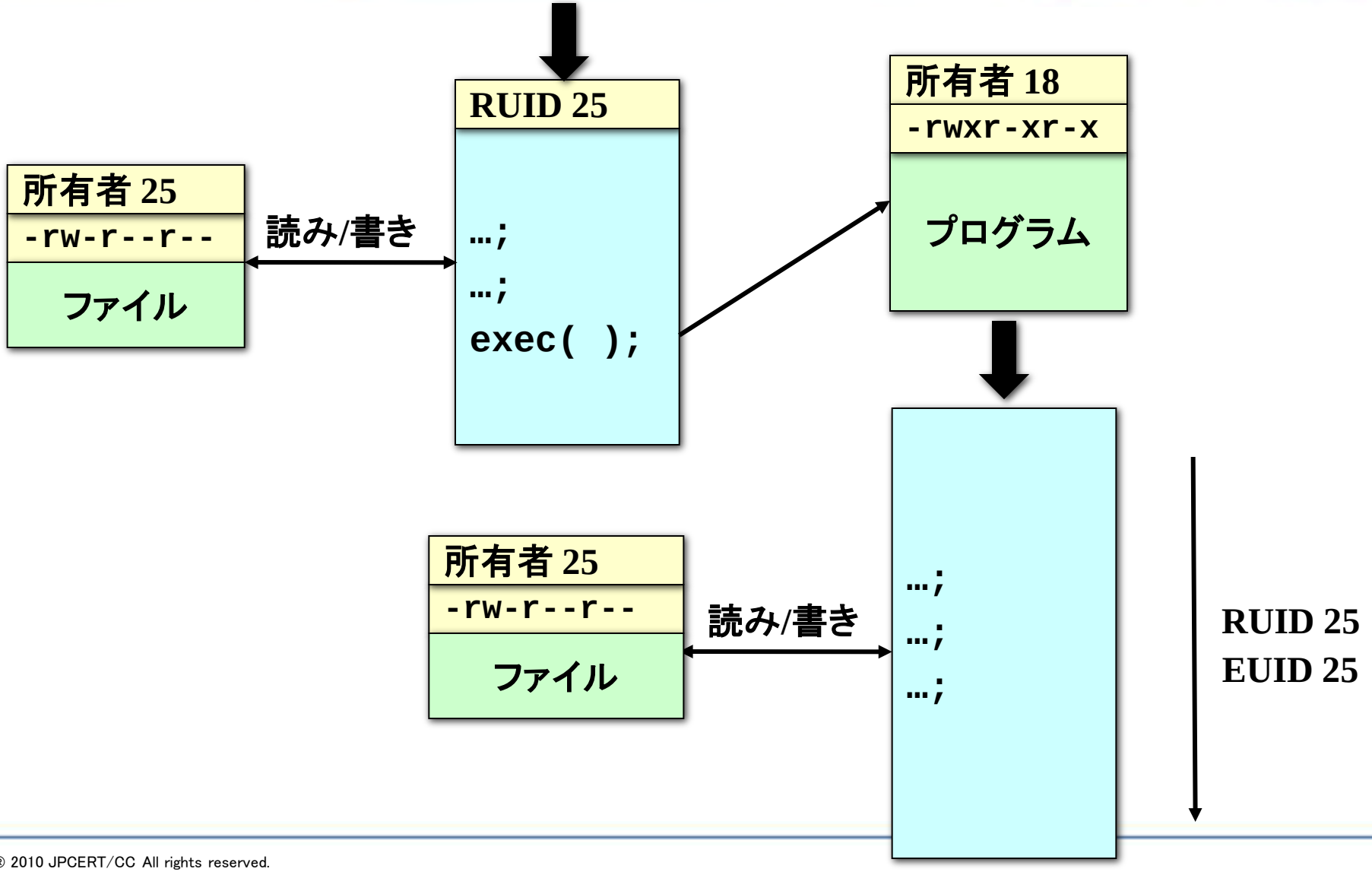
UNIX環境においてコマンドが実行されるときの動作：

- fork()によってプロセスをコピー(子プロセスの生成)
- exec()系システムコールによってコマンドを実行

生成された子プロセスは、親プロセスの属性を継承する

- RUID、RGID、EUID、EGID など
- 補助グループ ID

通常プログラムの実行



setuid ファイルモードビット

☆一般ユーザが別の権限で作業をするための仕組み

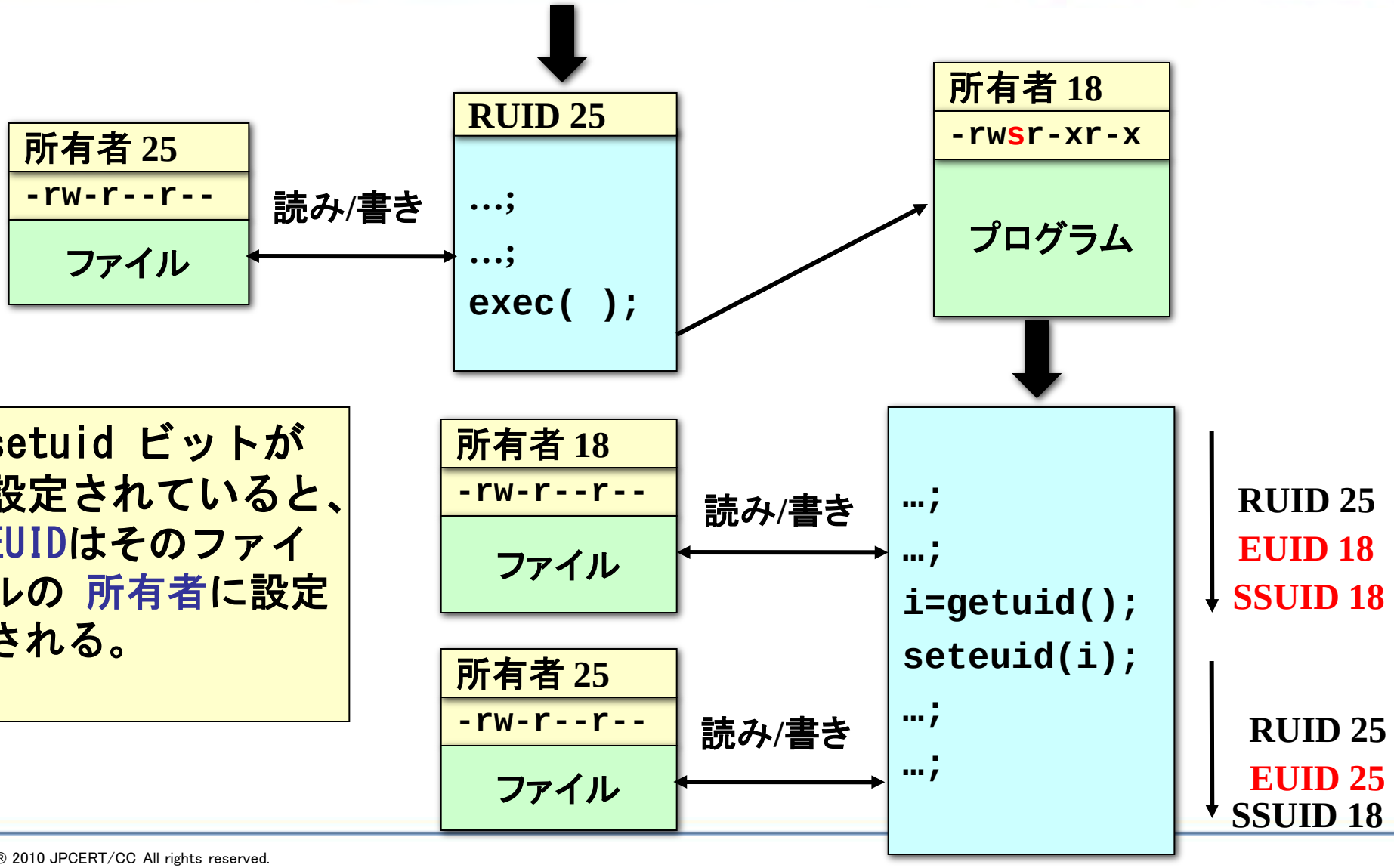
set-user-ID-on-execution ビット (**S_ISUID**)

- setuidビットが設定された実行可能ファイルを実行すると、起動されたプロセスはファイル所有者の権限を持つ
- (つまり、新しいプロセスの EUID と saved set-user-ID はファイルの所有者 ID の値となる)

set-group-ID-on-execution ビット (**S_ISGID**)

- setgidビットの設定された実行可能ファイルを実行すると、起動されたプロセスはファイルのグループの権限を持つ
- (つまり、新しいプロセスの EGID と saved set-group-ID はファイルのグループ ID の値となる)

setuid プログラムの実行



setuid ビットが設定されていると、EUIDはそのファイルの 所有者に設定される。

saved set-user-ID

saved set-user-ID 機能により、プログラムは最後の exec()の呼び出し時に設定された EUID を取り戻せる。

- さもなければ、同じ関数を実行するためにプログラムはスーパーユーザとして実行しなければならない。

同様に、saved set-group-ID 機能により、プログラムは最後の exec()の呼び出し時に設定された EGID を取り戻すことができる。

永久に権限を破棄するには、EUID を RUID に設定する。こうすることで昇格した権限は新しいプログラムに渡されない。

目次

- ユーザ、ファイル、プロセス
- 権限の変更
- 権限の管理
- ファイルパーミッションの管理
- 脅威の緩和方法
- まとめ

権限の変更

スーパーユーザ権限を持つプロセスは、通常ユーザの権限を永久に、あるいは、一時的に適用することが必要な場合がある。

スーパーユーザの権限を持っていないプロセスは、RUID と saved set-user-ID の間の切り替えが必要な場合がある。

一時的な権限の破棄は、

- ファイルにアクセスする際に役立つ
- 権限は取り戻すことができるため、任意のコードの実行を許可する脆弱性（バッファオーバーフローなど）の影響の制限にはならない

最小権限の原則

プロセスが昇格した権限で実行しており、共有ディレクトリやユーザディレクトリのファイルにアクセスする場合は、EUID を RUID (プロセスを実行したユーザのID)に設定すべし。

- オペレーティングシステムの権限モデルを活用
- ユーザによる許可されていない操作を防止

それでもなお、AFS など異なる権限モデルを使うファイルシステムでは、プログラムの安全性は十分でない可能性がある。

権限に関する脆弱性の例

FreeBSD 4.4(およびそれ以前)に含まれている OpenSSH は、スーパーユーザ権限で動作するが、ファイルを開く前に常に権限を破棄するとは限らない:

```
  fname = login_getcapstr(lc,“copyright”,NULL,NULL);
  if (fname != NULL && (f=fopen(fname,"r")) != NULL) {
   while (fgets(buf, sizeof(buf), f) != NULL)
     fputs(buf, stdout);
   fclose(f);
}
```

攻撃者は、~/.login_conf ファイルに細工しておくことで、ファイルシステムの任意のファイルを読み取ることができる:

```
  copyright=/etc/passwd
```

参考: FreeBSD-SA-01:63

許可される変更

プロセスは EUID を次の ID に変更できる。

- Real UID(プログラムを起動したユーザ)
- Saved Set-User-ID

これによりプロセスは、権限の切り替えが可能になる。

スーパーユーザ権限を持つプロセスは、あらゆる操作を実行できる。

権限変更のための関数

JPCERT CC®

プロセスが持つ3つのユーザIDの操作に使用する関数

- **seteuid()**
- **setuid()**
- **setreuid()**
- **setresuid()**

これらの関数は、異なるバージョンの UNIX 上では異なるセマンティクスを持つため、アプリケーションを移植する際にセキュリティ上の問題につながる可能性がある。

(参考: *Setuid Demystified.*
http://www.cs.berkeley.edu/~daw/papers/setuid-usenix02.pdf)

seteuid(),setuid(),setreuid(),setresuid()

プロセスの EUID を変更する

```
int seteuid(uid_t euid);
```

- スーパーユーザ権限で実行中のプロセスは、EUID を任意の値に設定できる。
- 一般ユーザプロセスは、EUID を RUID または SSUID にのみ設定できる。
- **setegid()**関数は、グループIDに対し、同様の動作をする。

seteuid()の例

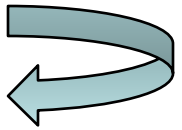
Admin(UID=1000)というユーザが、binユーザ (UID = 1) 所有の、setuid ビット が設定されたファイルを実行した場合

プログラムは、起動時には次のユーザ ID を持つ。

RUID 1000(admin)
EUID 1(bin)
SSUID 1(bin)

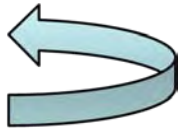
権限を一時的に手放すには、seteuid(1000)を呼ぶ

RUID 1000(admin)
EUID 1000(admin)
SSUID 1(bin)



権限を取り戻すには、seteuid(1) を呼ぶ

RUID 1000(admin)
EUID 1(bin)
SSUID 1(bin)



seteuid(),setuid(),setreuid(),setresuid()

JPCERT CC®

プロセスの EUID を変更する

```
int setuid(uid_t uid);
```

- **setuid()**関数は主に、権限を永久に破棄する目的で使われる
- 動作にはバリエーションがあり、過去のさまざまな実装の動作が反映されている。
- **setuid()関数よりも seteuid()関数を使うように心がけるべき.**

POSIX.1-1990 におけるsetuid()の動作

呼び出し元に「適切な権限」がある場合、**setuid()**は呼び出し元プロセスの以下を設定する。

Real UID
EUID
saved set-user ID

「適切な権限」がない場合には EUIDのみ設定する。

setuid ビットを設定してインストールされ、RUID を使って操作を実行する必要のあるアプリケーションに必要。

たとえば、**lpr** は権限付きの操作を実行するために昇格した EUID を必要とするが、ジョブはユーザの実際の RUID で印刷される。

setuid()の動作の相違

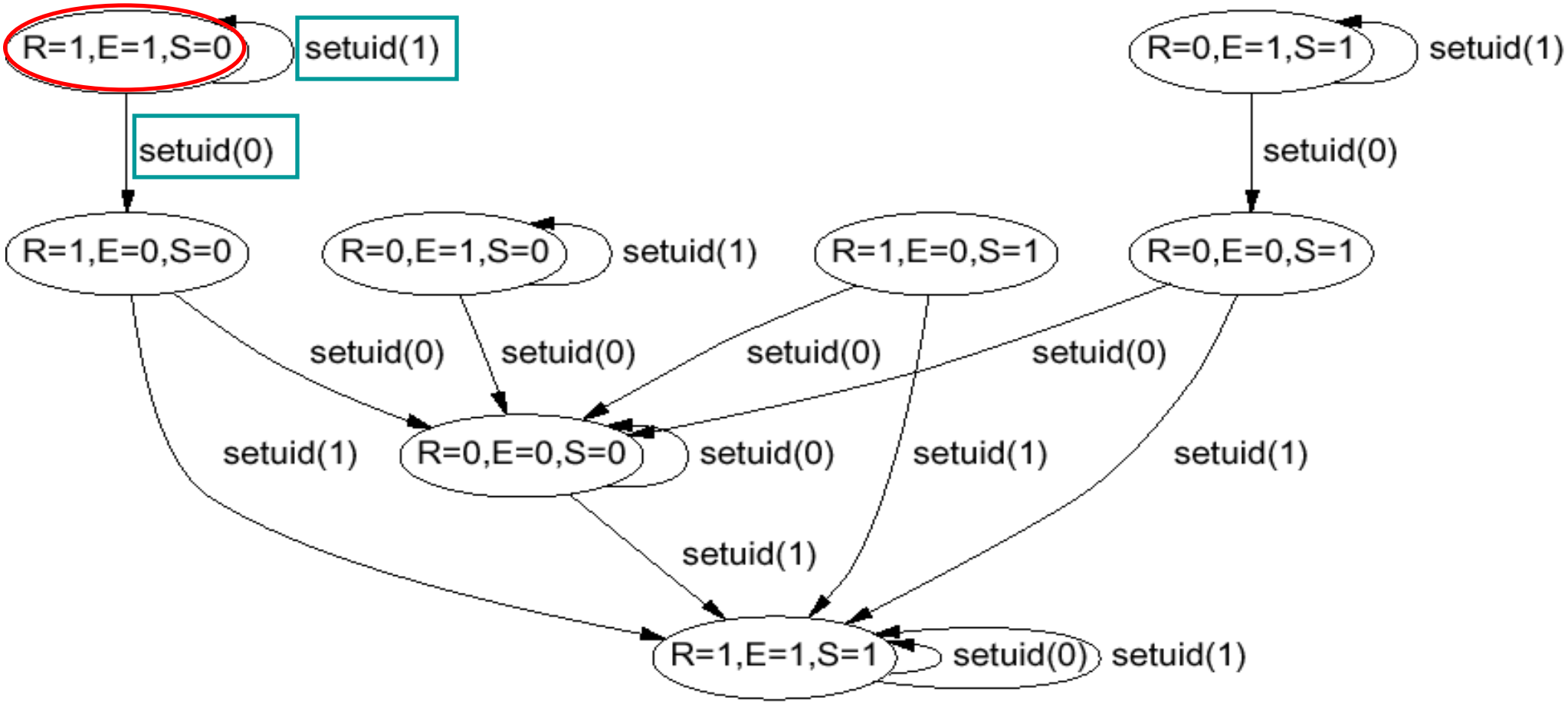
JPCERT CC®

「適切な権限」とは？

- Solaris: EUID=0（すなわち、プロセスが root として実行している）
- Linux: プロセスに **CAP_SETUID** 機能がある (参考:capabilities(7))

  EUIDが **{0,RUID,SSUID}** と等しくない場合、 **setuid(geteuid())**は失敗する。
- BSD: 全てのユーザが常に「適切な権限」を持っている

setuid()の複雑性



Linux の例

seteuid(),setuid(),setreuid(),setresuid()

RUID と EUID を、引数 ruidとeuidで指定された値に設定。

```
int setreuid(uid_t ruid, uid_t euid);
```

- **ruid**または**euid**が -1 の場合、当該ユーザIDは変わらない。
- 適切な権限を持つプロセスは両方のIDを任意の値に設定できる。
- 権限のないプロセスは、euid引数がプロセスの RUID、EUID または SSUID と等しい場合に EUID のみ設定できる。
- 適切な権限のないプロセスが、RUID を、プロセスの現在のRUID、EUID、SSUID のいずれかに変更できるかどうかは不定。

seteuid(),setuid(),setreuid(),setresuid()

RUID、EUID、SSUID を明示的に設定する。

```
int setresuid(uid_t ruid, uid_t euid, uid_t suid);
```

- **ruid, euid** または **suid**が -1 のとき、当該ユーザID は不変。
- スーパーユーザはこれらのIDを任意の値に設定できる。
- 権限のないユーザは、IDのいずれかを現在のIDのいずれかの値に設定できる。
- **setresuid()**には明確なセマンティクスがあり、さまざまな系統のUNIX(Linux、FreeBSD、HP-UX、OpenBSD3.3 以降)において同じ方法で実装されている。

目次

- ユーザ、ファイル、プロセス
- 権限の変更
- 権限の管理
- ファイルパーミッションの管理
- 脅威を緩和する方法
- まとめ

setuid および setgid プログラム

JPCERT CC®

setuid プログラム: set-user-ID-on-execution ビットが設定されているもの。

setgid プログラム: set-group-ID-on-execution ビットが設定されているもの。

setuid プログラムは、root (set-user-ID-root) として、または root 以外のより制限された権限での実行に使われる。

setuid rootプログラム

JPCERT CC®

setuid-root が必要な例：passwd、ping

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x  1 root  bin  [...] /usr/bin/passwd
$ ls -l /sbin/ping
-r-sr-xr-x  1 root  bin  [...] /sbin/ping
```

root 以外のsetuidおよびsetgidプログラム

通常は、制限されたあるいは特定のタスクの実行に使われる。

これらのプログラムは、EUID を RUID や SSUID にある値にのみ変更できる。

システム設計上、root に set-user-ID することは可能な限り避けるべき。

例: wallプログラム

各ユーザの端末デバイスにメッセージを書き込むことにより、システムの全ユーザにメッセージを一斉配信する。

相互の覗き見や他人の端末セッションの妨害を防ぐため、root 以外の通常ユーザは、他人の端末デバイスに直接的には書き込めない。

wall プログラムはsetgidを活用して一斉配信を実現している。

- **wall** は setgid-**tty** としてインストールされ、**tty** グループのプロセスとして実行される。

```
-r-xr-sr-x  1 root  tty  [...] /usr/bin/wall
```

- 端末デバイスは、グループによる書き込みが可能

```
crw--w----  1 usr1  tty  5, 5[...] /dev/ttyp5
```

setuid プログラムに関する注意1

setuid プログラムは、ファイルの所有者に許可されているすべての操作を実行できる。(setuid-rootではrootが実行できるすべて)

setuid プログラムでは以下を行わないように注意する。

- 信頼できないユーザに対する操作
- 信頼できないユーザへの機密データの受け渡し

最小権限の原則

- root 権限が不要になったら EUID を変更すること

setuid シェルスクリプト(避けるべき)

- 競合状態: setuid プログラムの実行を開始し、プログラムがロードされ実行される前にその内容を変更される危険性。
- いずれにしても一部のシステムは、シェルスクリプトの setuid ビットを無視する。

setuid プログラムに関する注意2

setuid プログラムの動作環境では、以下のような情報はユーザによって操作されている可能性があることを考慮すべし:

- ファイル記述子、引数、環境変数
- **cwd**、**tty**
- リソース制限
- タイマーとシグナル

UNIX のバージョンによっては、制御される項目のリストが異なるため、すべてのOS環境に対応する移植性のあるコードを記述するのは難しい。

Locally exploitable な脆弱性の攻撃で悪用される。

メール転送エージェントに要求される権限管理とは?

以下を行うメール転送エージェント（MTA）の権限管理について考察せよ。

- （信頼できない、ローカルおよびリモートの）ユーザからメッセージを受信する
- 受け取ったメッセージを宛先ユーザの受信箱（宛先ユーザが所有するファイル）へ配信する
- ローカルユーザからメッセージを受け取り送信先に配送する
- 各メッセージを送信した UID（および GID）にもとづきアクセス制御し、ログファイルに動作を記録する

メール転送エージェントの実装

- ユーザが所有する受信箱へ書き込むために、root 権限で実行する必要がある。
- 受信箱への書き込み時は一時的に通常ユーザのIDで動作するのが安全。

Sendmail, postfix, exim など著名なMTAがどのような実装をしているか、調べてみよう!

権限の放棄と復元

権限を一時的に破棄するには、特権をもつUID を EUID から削除し、SSUID に格納する。

- 後で SSUID から復元できる。

権限を復元するには、EUID を SSUID にセットする （SSUIDが権限つきUIDの場合）

権限を永久に破棄するには、特権をもつ UID を EUID と SSUID の両方から削除する。

- 昇格した権限はもう復元できない！

権限の放棄と復元

権限を一時的に破棄するには、特権UID を SSUID にコピーし、EUID には一般権限の ID をセットする。

- 特権IDは後で SSUID から復元できる。

権限を復元するには、 SSUID にコピーしておいた ID を EUID にセットする。

権限を永久に破棄するには、特権UID を EUID と SSUID の両方から削除する。

- 昇格した権限はもう復元できない！

一時的に権限を降格する

JPCERT CC®

```
/* 制限された操作を実行する */
setup_secret();

uid_t uid = /* 特権のないユーザID*/
/* 権限を一時的に uid に降格する */
if (setresuid( -1, uid, geteuid()) < 0) {
  /* エラー処理 */
}

/* 普通の処理を続ける */
some_other_loop();
```

権限を復元する

```
/* 特権のない操作を実行する */
some_other_loop();

/* 降格した権限を復元する。SSUIDに特権idがはいっている */
uid_t ruid, euid, suid;
if (getresuid(&ruid, &euid, &suid) < 0) {
  /* エラー処理 */
}
if (setresuid(-1, suid, -1) < 0) {
  /* エラー処理 */
}
/* 特権の必要な処理を続ける */
setup_secret();
```

権限を永久に降格する

```
/* 制限された動作を行う */
setup_secret();

/* 権限を永久に破棄する。RUID には特権がないと仮定。 */
if (setresuid(getuid(), getuid(), getuid()) < 0)
{
  /* エラー処理 */
}

/* 通常の処理を続ける */
some_other_loop();
```

グループ権限

setgid(),setegid(),setresgid()は**setuid(),seteuid(),setresuid()**に似たセマンティクスだがグループIDに対して動作する。

プログラムによっては、set-user-ID-on-execution ビットと set-group-ID-on-execution ビットの両方が設定されている。

set-group-ID-on-execution ビットのみ設定されていることの方が多い。

グループ権限を破棄する1

プログラムにset-user-ID-on-executionビットとset-group-ID-on-execution ビットの両方が設定されている場合、グループ権限の破棄を忘れずに。

```
/* 制限された操作を実行*/
setup_secret();

uid_t uid = /* unprivileged user */
gid_t gid = /* unprivileged group */
/* 権限を一時的に uid と gid 降格 */
if (setresgid( -1, gid, getegid()) < 0) {
  /* エラー処理*/
}
if (setresuid( -1, uid, geteuid()) < 0) {
  /* エラー処理 */
}
/* 通常の処理を続ける */
some_other_loop();
```

グループ権限を破棄する2

つぎの例は、間違った順序で権限を破棄している

```
if (setresuid( -1, uid, geteuid()) < 0) {
  /* エラー処理 */
}
if (setresgid( -1, gid, getegid()) < 0) {
  /* EUID はもう 0 ではないので失敗するかも */
}
```

EGIDが0でも、EUIDが root 権限とは限らない。

setresgid() の実行結果は OS に依存する。

また、補助グループの権限も必ず破棄すること。

```
/* int setgroups(int ngroups, const gid_t *gidset); */
setgroups(0, NULL);
```

権限管理を誤ったときの被害

特権を持つプログラムにおいて権限管理を誤ると、本来アクセス権のなかったファイルを攻撃者に操作される可能性がある。

考えられるシナリオ：

- 特権ファイルを読む （情報漏洩）
- ファイルの切り捨て
- ファイルにデータを追加
- ファイルの permission を変更する
- マシンを完全に制御する

目次

- ユーザ、ファイル、プロセス
- 権限の変更
- 権限の管理
- ファイルパーミッションの管理
- 脅威を緩和する方法
- まとめ

- プロセス権限の管理は方程式の半分。
- 残りの半分はファイルパーミッションの管理。
- ファイルパーミッションの管理は、一部はシステム管理者の責任であり、一部はプログラマの責任。

FIO01-C. ファイル名でファイル識別する関数の使用に注意

C99 の以下の標準関数はファイル名だけに頼ってファイルを識別するので危険

- **remove(), rename(), fopen(), freopen()**

ファイルに(繰り返し)アクセスするとき、ファイル名ではなく、ファイルディスクリプタや FILE ポインタでアクセスする方が確実

セキュアコーディング例

```
char *file_name;
FILE *f_ptr;
/* file_nameを初期化 */
f_ptr = fopen(file_name, "w");
if (f_ptr == NULL)  {
/* Handle error */
}
/* ... */
if(chmod(file_name, S_IRUSR)==-1)
{
  /* エラー処理 */
}
```

ファイル名で改めてファイルにアクセスしている

```
char *file_name;
intfd;
/* file_nameを初期化 */
fd = open(  file_name,  O_WRONLY |
O_CREAT | O_EXCL,  S_IRWXU);
if (fd == -1) {
  /* エラー処理 */
}
/* ... */
if (fchmod(fd, S_IRUSR) == -1) {
  /* エラー処理 */
}
```

POSIXのopen(),fchmod()を使うことで同一のオブジェクトに対して操作

書き込み可能なディレクトリの落とし穴

特定のユーザがディレクトリに書き込み可能な場合、そのユーザはディレクトリ内にあるディレクトリとファイルの名前を変更できる。

極秘データを収めたファイルをディレクトリ **`/home/myhome/stuff/securestuff`**に保存したとして、

どこかのユーザAが **`/home/myhome/stuff`** に書き込み可能だったら…

- ディレクトリ **`securestuff`** を別の名前に変更するかも
- 極秘データが入ったファイルにアクセスできなくなる！

例： sendmail

sendmailは読み書きするファイル(設定ファイルなど)のモードをチェックし、例えば次のようなファイルの読み取りは拒否する。

- グループによる書き込みが可能なファイル。所有者以外のユーザによって改ざんされている可能性があるため。
- グループによる書き込みが可能なディレクトリのファイル
  - /etc/mail/aliases
  - /etc/mail/sendmail.cf

  など

セキュアなディレクトリ1

セキュアなディレクトリ：

ユーザ自身と管理者以外はファイルの作成・リネーム・削除などの操作ができないディレクトリ。

- 他のユーザでもファイルを読んだりディレクトリを検索したりはできるが、ディレクトリのコンテンツは変更できない。
- セキュアディレクトリの親ディレクトリなど上位のディレクトリにあるファイルを、他のユーザが削除したりリネームしたりできてはならない。

セキュアなディレクトリでファイルを操作すれば、ファイルやファイルシステム改ざんによる攻撃を排除できる。

セキュアなディレクトリ2

ファイル名とファイルの実体との間には弱い結びつきしかない。そのため脆弱性がつくりこまれやすい。

(参考:FIO01-C.ファイル名でファイル識別する関数の使用に注意)

- セキュアなファイル操作が可能ならば、必ずそうすべき
- セキュアなファイル操作が困難な場合、セキュアなディレクトリ上でファイルを操作するようにデザインする。

セキュアなディレクトリ3

セキュアなディレクトリを作成するには、対象ディレクトリの上位のパスがすべて以下の条件を満たす必要がある:

- 当該ユーザかスーパーユーザによって所有されている
- それ以外のユーザには書き込み権限がない

上位のディレクトリが他のユーザによって削除されたりリネームされないようにすること。

ルートから末端までのディレクトリをくまなく精査し、攻撃者がそのいずれかのディレクトリをリネームしたり再作成できるような危険な競合状態(race condition)を避ける。

セキュアなファイルを作成するには

- パーミッションを所有者のみに限定

- `fopen()`や `mkstemp()`などの関数でファイルを作成する前に、`umask()`を呼び出して適切なパーミッションを設定する

umask1

UNIX では、プロセスごとに umask 値を持っており、ファイルやディレクトリを新規作成する場合にその umask が使用される。

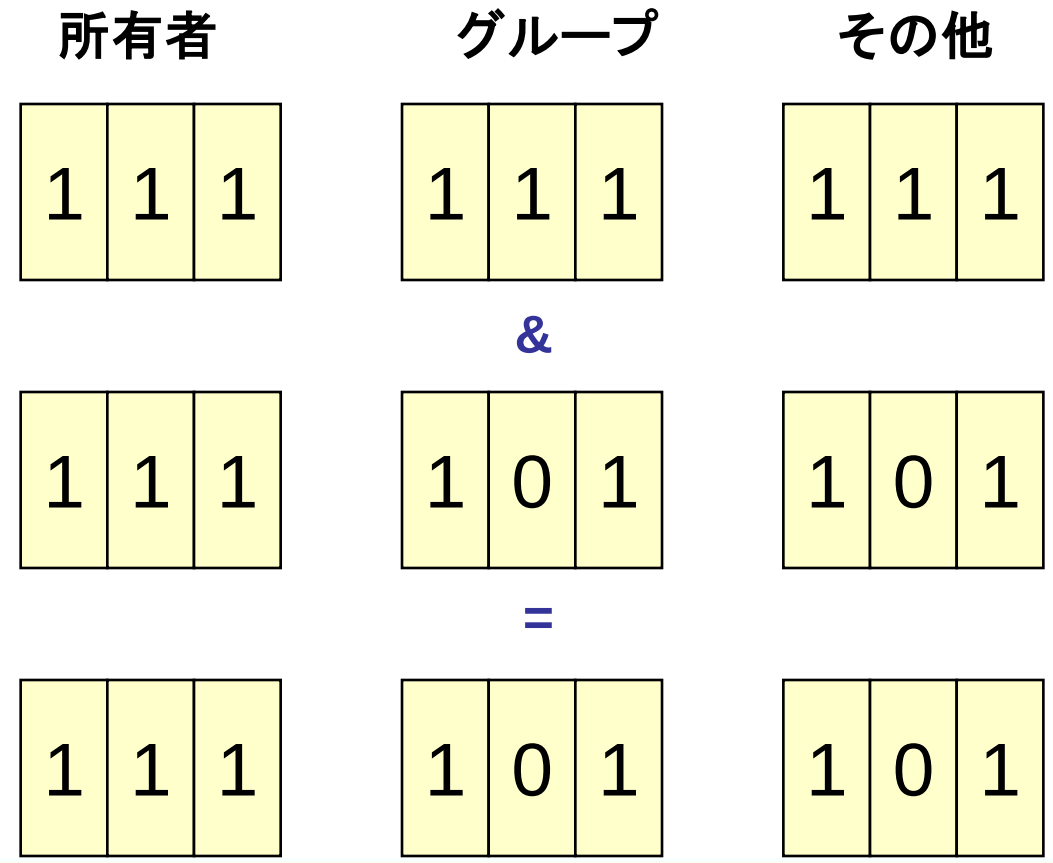
umask は、ファイルを作成するときにシステムコールに引数として指定されたビットをマスクする。

umask のビット == 「許可したくないビット」

- 次の関数の呼び出し時に渡された mode 引数の許可ビットを無効にする:
  open()、**creat()**、**mkdir()**、**mkfifo()**
- **chmod()**と **fchmod()**は umask の設定の影響は受けない

umask2

オペレーティングシステムは、umask の反転値とプロセスから要求されたパーミッションのビット毎の&を計算することによってアクセスパーミッションを決定する。



プロセスは、親プロセスから umask の値を継承する。

典型的には、シェルはユーザのログイン時に以下のumask 値をデフォルトとして設定する。

- 022(group, world のwritable bit を無効にする)
- 02(world の writable bit を無効にする)

ユーザは umask 値を変更できる。

ユーザが設定したumask 値は決して信用するな!!

fopen()を使用したファイルの作成 1

C99 の **fopen()**関数を使って新しいファイルを作成する場合、パーミッションは指定できない。デフォルトで0666 が設定される。

この動作を変更する唯一の方法は、次のどちらか：

- **fopen()**の呼び出し前に umask を設定
- ファイル作成後に **fchmod()**でパーミッション変更

ファイル作成後のパーミッション変更は、競合状態が生じるため避けるべき。

ファイルが作成されてからパーミッションが変更されるまでの間に、攻撃者がファイルにアクセスする可能性がある。

fopen()を使用したファイルの作成 2

正しい解決策は、ファイルの作成前に umask を変更することである。

```
mode_t old_umask = umask(~S_IRUSR);
FILE *fout = fopen("fred", "w");
. . .
fclose(fout);
```

0400になる

fopen() はC99の関数。 **umask()**はPOSIXの関数。

この 2 つの関数間のやり取りを定めた規格はない。

必ず使用するプラットフォームでテストすること!

mkstemp()のいくつかのバージョンは同じ方法で umask とやり取りする。

POSIX のopen()関数

POSIX の **open()**関数には、ファイルの作成時に使うパーミッションを指定する、オプションの第3引数がある。

```
fd = open("fred", O_WRONLY|O_CREAT, S_IRUSR);
```

fredという名前のファイルを、書き込み専用で新規作成し、パーミッションをユーザの読み取り許可に指定している。

目次

- ユーザ、ファイル、プロセス
- 権限の変更
- 権限の管理
- パーミッションの管理
- 脅威を緩和する方法
- まとめ

set*uid関数は使わないで済ませましょう！

出典：『**UNIX&インターネットセキュリティ**』、

Simson Garfinkel、Gene Spafford、オライリー・ジャパン、1998

推奨事項のリスト

JPCERT CC®

`setresuid()`を使う

`seteuid()`を使う

戻り値の状態をチェックする

昇格した権限はなるはやで破棄する

権限を分離する

setresuid()を使う

setresuid()関数は 3 つの UID すべてを設定し、成功したら 0 を、エラーが発生したら -1 を返す。

- すべての UID 関数の中で最も移植性が高い。
  - Linux と BSD で使用可能
  - Solaris では使用できない
- すべての UID 関数の中ではセマンティクスが最も簡潔
- 3つの UID すべてを指定された値に明示的に設定する
- 1つか 2つだけ設定することはなく、必ず設定しないか 3つすべてを設定する。
- EUID=0 の場合、または各引数が3つの UID 値のいずれかと一致する場合に常に成功する。

seteuid()を使う

seteuid()関数は、単に EUID を設定する。

- **setresuid()**よりも移植性が高い。
  - Linux、BSD、Solaris で使用可能。
- セマンティクスは **setresuid()** とほぼ同じ程度に簡潔
- EUID を設定する。RUID や SUID には影響しない。
- EUID=0 の場合常に動作する。
- EUID が 0 でなければ、
  - Linux とSolaris では、新しい EUID が実行前の3つのUIDのいずれかと一致すれば動作する。
  - BSDでは、新しく設定するEUID が実行前のRUIDまたはSSUID と一致すれば動作する。

戻り値をチェックする

各 set*uid 関数は、成功した場合 0 を返す。
失敗した場合は -1 を返し、適切な**errno**を設定。
☆ set*uid 関数の戻り値が 0 であることを確認せよ。

さらにsetuid プログラムを移植する場合には…

getuid()や **geteuid()**などを使って UID 値が正しく設定されていることを確認するとよい。

昇格した権限は可能な限り破棄する

昇格した権限を一時的/永久に破棄することで、プログラムは

- 権限を持たないユーザと同じ制限に従ってファイルにアクセスする
- ユーザがパーミッションのないはずの操作を行うことを防止する

しかし、以下のような欠点もある

- 任意のコード実行を可能にするような脆弱性 (例えばバッファオーバーフロー)の影響は防げない。一時的に権限を破棄しているだけなら、特権を復元できる。

権限を分離する1

JPCERT CC®

永遠の課題

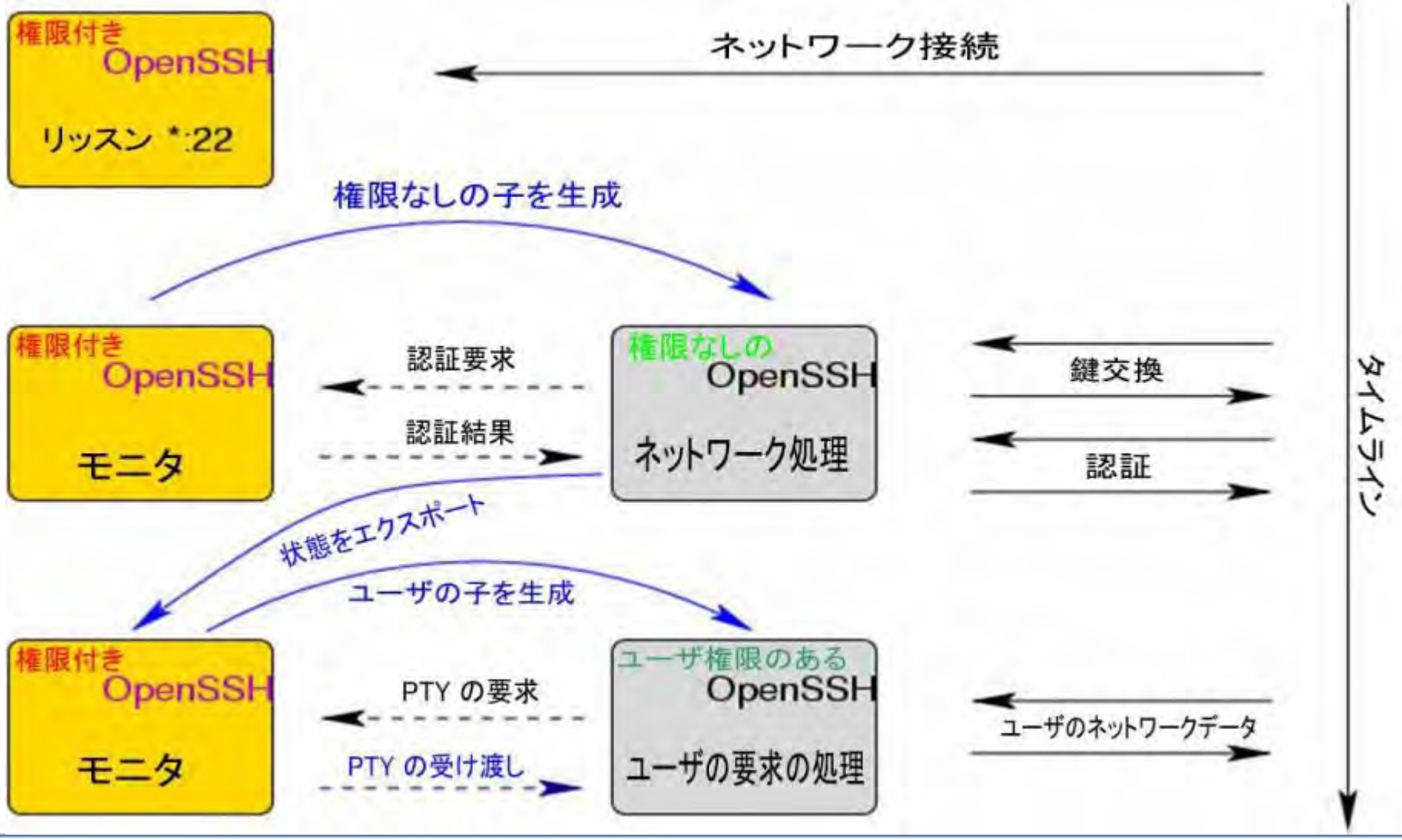
- setuid プログラムは可能なかぎり単純にすべき。
- 注意して使っていても失敗する可能性がある。
  時として予期しない方法で

権限を分離する$_2$

権限を一時的/永久に破棄するかわりに、特権を持つ親プロセスと権限を制限した子プロセス間のインターフェイスを定義し、プロセス毎に権限を分離するという方法もある。

- 子は権限を必要とする処理を親に委任する
- プロセス間のやり取りは、パイプを通じて行う
- 他の手段でエクスポートできない場合は共有メモリに保存
- 認証が成功したかどうかを判断するには、子プロセスは権限のある親プロセスに問い合わせる

権限分離の例： OpenSSH



まとめ

不適切な権限管理は、さまざまな脆弱性につながる。

既存の API は

- 複雑
- 直感的でない
- 実装間で異なる

権限管理は細心の注意で！！

キーワードは「**最小権限の原則**」と「**権限分離**」

参考文献

[Chen 02] Hai Chen, David Wagner & Drew Dean, *Setuid Demystified*. 11th USENIX Security Symposium, 2002.

[Dowd 06] M. Dowd, J. McDonald & J. Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities.* Boston, MA: Addison-Wesley, 2006. See http://taossa.com for updates and errata.

[Meunier 04] Pascal Meunier. CS390S: Canonicalization and Directory Traversal, November 2004.

[MITRE 07] MITRE. Common Weakness Enumeration, Draft 7. October 2007. http://cwe.mitre.org/

[Howard 02] Michael Howard & David C. LeBlanc, *Writing Secure Code,* 2nd ed., Redmond, WA: Microsoft Press, 2002 (ISBN 0-7356-1722-8).

[Viega 03] John Viega & Matt Messier, *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Networking, Input Validation & More*. Sebastopol, CA: O'Reilly, 2003 (ISBN 0-596-00394-3).

Simson Garfinkel, Gene Spafford『UNIX&インターネットセキュリティ』オライリー・ジャパン、1998

オンラインの参考資料

- Matt Bishop, *“Writing Safe Setuid Programs”*
  http://nob.cs.ucdavis.edu/%7Ebishop/secprog/
- **Apple Secure Coding Guide, *“Avoiding Race Conditions and Insecure File Operations”***
  http://developer.apple.com/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html
- **CWE-367 Time-of-check Time-of-use Race Condition**
  http://cwe.mitre.org/data/definitions/367.html
- POSIX open() のマニュアル
  http://www.opengroup.org/onlinepubs/009695399/functions/open.html