SELinux の利用とポリシー作成

レッドハット株式会社
ソリューションアーキテクト 森若和雄
2017.09.04

概要

- SELinux とは
- ターゲットポリシー
  - 未知のプログラムはどう扱われるか
- 典型的なポリシー作成手順

概要

- SELinux とは
- ターゲットポリシー
  - 未知のプログラムはどう扱われるか
- 典型的なポリシー作成手順

SELinux とは？

- 権限管理を拡張して、通常のパーミッションの仕組みで許可された操作をさらに追加でチェックする仕組み
  - パーミッションで拒否される場合は特にチェックはしない
- 通常のパーミッションよりはるかに細かくポリシーが定義できる
- 権限管理のモデルとしては以下の仕組みを提供。 Red Hat が RHEL の一部として提供するポリシーは Type Enforcement を主に利用
  - RBAC (role base access control)+Type Enforcement
  - MLS (multi level security)
  - MCS (multi category security)

通常のパーミッションと SELinux の違い

- パーミッションは
  - 権限管理の対象： ファイルとプロセス、 SysV IPC
  - 識別方法 : uid, gid
  - 操作の分類 : r (read), w (write), x(execute) の 3 種 + sticky bit
- SELinux は
  - 権限管理の対象： ファイル、プロセス、ソケット、 TCP/UDP ポート、共有メモリ、パイプ等、事実上 OS が提供するリソース全て + SELinux 対応アプリケーションが管理するリソース (dbus, systemd のサービス等)
  - 識別方法 : uid, gid とは独立した「タイプ」
  - 操作は対象 class 毎に細かく分類

例 : file クラスに対して定義されている操作
append, execmod, ioctl, open, relabelto, unlink, audit_access, execute, link, quotaon, rename, write, create, execute_no_trans, lock, read, setattr, entrypoint, getattr, mounton, relabelfrom, swapon

SELinux の価値

- ある操作を明確に許可するためのポリシーが存在しない場合、何も許可されない
- SELinux ポリシーは管理者だけが集中的に管理し、ユーザーの設定や操作に影響されない
- 通常のパーミッションでは粗くなってしまう権限の付与を細かく実施できる

  → 脆弱性を攻撃された時に影響範囲を限定

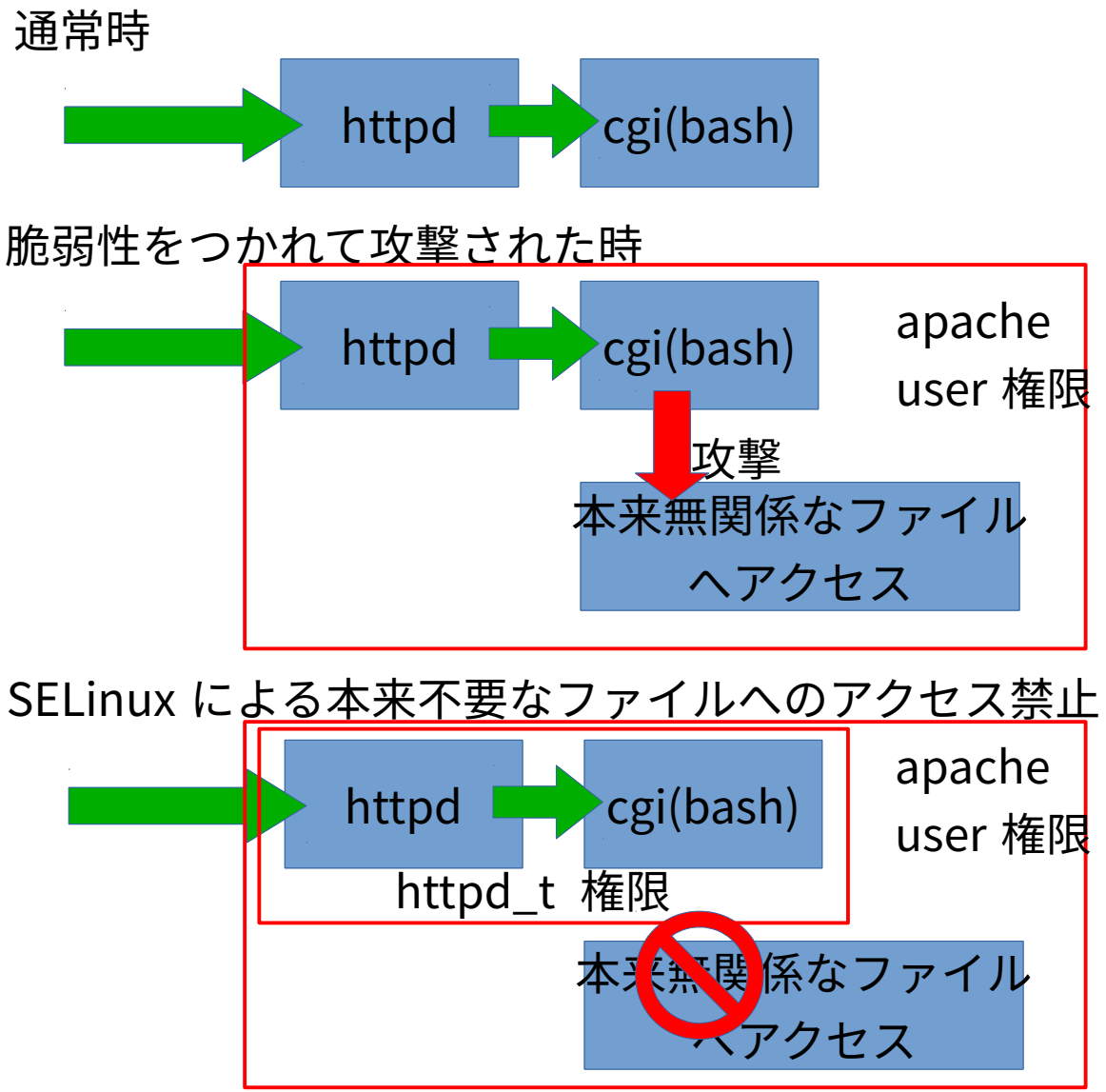
  → バグや間違った設定による影響範囲を限定

SELinux の効果イメージ

通常のパーミッションでは権限管理の粒度が粗いため、本来不要なファイルへのアクセスなども可能になっている。

SELinux では type づけと細かなポリシー設定により必要最低限の権限のみを付与できる。

SQL インジェクション攻撃のような攻撃には対応できない。本来アクセスできる DB に通常とは違うクエリを発行しているだけなので SELinux でチェックされる操作がない。

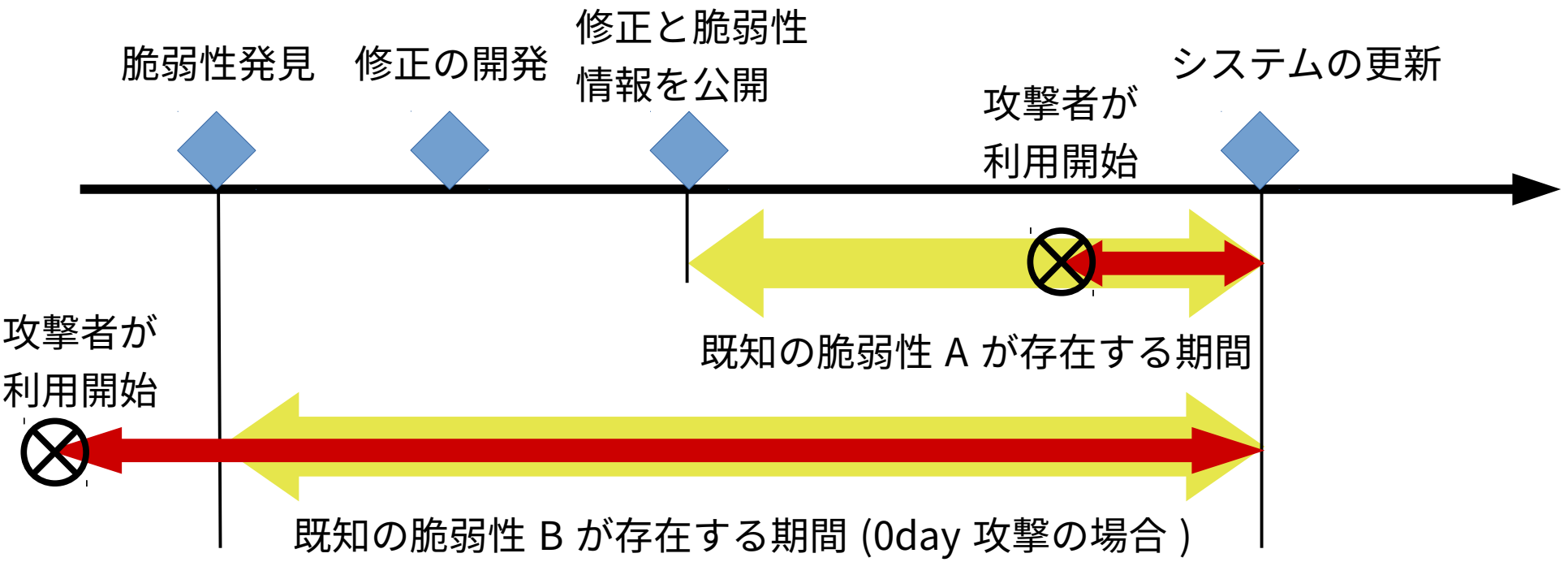


SELinux 導入検討の優先順位

- SELinux はアンチウイルスソフトウェアなどと同じく「攻撃された時の被害を緩和する」もの
- 多くの場合、攻撃を予防するための「インベントリ管理の徹底」、「脆弱性情報の把握」、「早期のソフトウェアの更新」の方が SELinux の適用よりも優先度が高い
  - これらを実施できていない環境で SELinux に手を出すのは少し不思議
- 以下のような場合は SELinux は有益な選択肢
  - 利用ソフトウェアが SELinux に対応しており手間をかけず適用できる
  - 諸事情でソフトウェアを更新できない場合があり、緩和策が必要
  - SELinux を導入することがポリシーで必要とされている
  - 未知の脆弱性を利用する 0day 攻撃に備えたい

脆弱な期間と SELinux

- 更新を定期的に実施していても、既知の脆弱性を持っている期間が存在する
- SELinux はこの期間の緩和策として特に有益



SELinux の type

- SELinux ではあらゆるものに type を付与する
  - /usr/bin/httpd の type を確認する例 :

```
$ ls -Z httpd
system_u:object_r:httpd_exec_t:s0 httpd
```

- プロセス起動時のタイミングで、ファイルとユーザに関連づけられた type に切り替える
  - 例 : httpd_exec_t (実行ファイルのタイプ) を systemd からサービスとして起動 → httpd_t (プロセスのタイプ)

type によるポリシー

- 「操作元 (source type) が
  操作対象 (target type) に対して
  どういうジャンル (class) で
  何をする (permission) 」という形で特定の操作を許可する
- 操作の許可以外にも以下のようなものを指定する
  - 他の type に変更 (type_transition)
    - execve のタイミングで type を変更する
    - 元の type 、実行ファイルの type 、変更先の type の組み合わせで指定
  - 監査ログに記録しない (dontaudit)
    - 成功失敗にかかわらずログに出力しない。問題ないディレクトリのスキャンなどで過剰なログがでないようにする
  - 許可するが監査ログに記録もする (auditallow)

ポリシーの例

- user_t(普通のユーザが実行しているプロセスにつく type) のプロセスが passwd_exec_t のファイルを実行できる

```
allow user_t passwd_exec_t:file { execute getattr
open read };
```

- ntpd_t のプロセスが dhcpc_t に SIGCHLD を送信できる

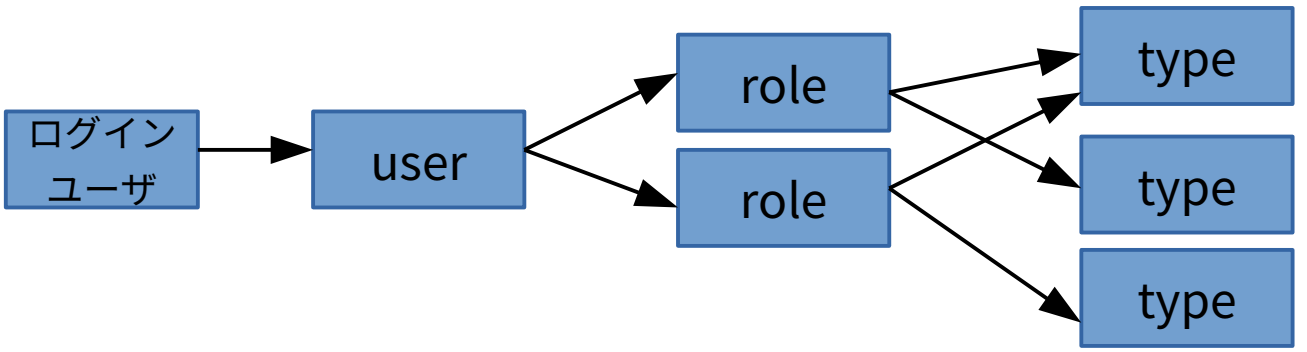
```
allow ntpd_t dhcpc_t:process sigchld;
```

- ping_t が rawip_socket を作って書き込みできる

```
allow ping_t ping_t:rawip_socket { append bind
connect create getattr getopt ioctl lock read
setattr setopt shutdown write };
```

SELinux の user, role, type

- SELinux の user は通常のユーザからは独立
- user はログイン時にマッピングに従い割り当てされ、変更不可
  - semanage login -l で対応が確認できる
- あらかじめ定義されている以下 2 組の対応関係でどの type としてふるまえるか決まる
  - user と role の対応関係 (seinfo -u -x または semanage user -l で確認)
  - role と type の対応関係 (seinfo -r -x で確認)
  - ただし sudo で role と type の変更が可能



リソースのタイプを確認する

- ユーザーのタイプを id コマンドで確認

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

- プロセスのタイプを ps コマンドで確認

```
$ ps auxcZ|grep sshd
system_u:system_r:sshd_t:s0-s0:c0.c1023 root 1156 0.0  0.0 97852
7188 ?        Ss   Aug18   0:00 sshd
```

- ファイルのタイプを ls コマンドで確認

```
# ls -Z /etc/passwd
system_u:object_r:passwd_file_t:s0 /etc/passwd
```

概要

- SELinux とは
- ターゲットポリシー
  - 未知のプログラムはどう扱われるか
- 典型的なポリシー作成手順

ターゲットポリシー
(リファレンスポリシー)

- Red Hat Enterprise Linux で提供されるデフォルトの SELinux ポリシー
- ターゲットポリシーの特徴 :
  - SELinux ポリシーの事実上の標準
  - ログインユーザに対する制限はほとんどない
  - サービスは基本的に必要最小限の権限を付与
    - 各サービスに対応するプログラム、ログ、データ用ファイル、ポート、ソケットなどのタイプを定義
    - 利用可能な機能全てを基本的には許可
    - 一部のセキュリティ的に制限したいが必要な場合もある権限は許可 / 拒否を切り替え可能にしている (boolean)
  - モジュール構造になっており部分的な追加・削除が可能

ポリシーがカバーしている プログラムは何か？

- RHEL 同梱プログラムは基本的に SELinux enforcing 環境でのみテストされている
  - Red Hat Satellite や OpenShift などは enforcing でのみサポート
- SELinux targeted policy は RHEL に含まれないソフトウェア向けモジュールも同梱
  - サポートは提供されない
  - 参考にはなるので RHEL 同梱でないソフトウェアについて作業する前に semanage module -l で確認

ターゲットポリシー下で未知のプログラムはどう扱われる？

- ログインしたユーザーは unconfined_t という強力な権限が付与される
  - ほとんど何も制限されない
  - ユーザがログインしてコマンドラインや GUI から起動する場合、プログラムはほぼ何も制限をうけない
- サービスとして未知のプログラムを実行すると、unconfined_service_t という権限で実行される。
  - 制限は非常に緩い
  - 他サービスへ接続する unix domain socket とのやりとりが許可されていないなど無制限ではない。
    - 例 : systemd と通信してサービスを管理しよう→ NG
  - 動作確認と必要に応じてポリシーの追加が必要

ターゲットポリシーがカバーしないプログラムへの対応

- ISV が提供しているモジュールがあればそれを適用する
- 独自にモジュールを作成する
  - 一度作成したモジュールは rpm パッケージ化し、他システムへ転用可能
- 最低でもデフォルトの unconfined_t, unconfined_service_t で問題がないか確認する

※「ISV のポリシーにより SELinux enforcing 環境ではサポート不可」の場合もある。そのプログラムについては SELinux の適用をあきらめて他の緩和策を検討する。専用の仮想マシンを用意して分離するなど

概要

- SELinux とは
- ターゲットポリシー
  - 未知のプログラムはどう扱われるか
- 典型的なポリシー作成手順

典型的なポリシー作成手順 (1)

- プログラム用にポリシーのひながたを作成

```
$ sepolicy generate --application mynewapp -n mynewapp
```

- 関連するリソースをあらいだして利用するタイプを決定、ファイルとのマッピングを定義
  - 実行ファイル、設定ファイル、 systemd 用 service ファイル、ログファイル、利用ポート、呼び出す外部コマンドなど
  - 詳細が不明な場合は strace や lsof, ss などで調査

マッピング定義の例 :

```
/usr/bin/mynewappd    --  gen_context(system_u:object_r:mynewapp_exec_t,s0)
/usr/lib/systemd/system/mynewapp.service --
gen_context(system_u:object_r:mynewapp_unit_file_t,s0)
/var/log/mynewapp.log  --  gen_context(system_u:object_r:mynewapp_log_t,s0)
```

典型的なポリシー作成手順 (2)

- 作成した仮のポリシーを導入
- SELinux を permissive mode(チェックしてログに残すが実行拒否をしない特殊なモード) にする
- テスト実行

  → SELinux で許可されていない操作が監査ログに記録される
  (ソフトウェアのテストスイートを実行するのがベスト)

ログの例 :

```
type=AVC msg=audit(1499941607.383:669): avc:  denied  { execute_no_trans }
for  pid=9626 comm="pmie_check" path="/usr/bin/hostname" dev="dm-1"
ino=101536461 scontext=system_u:system_r:pcp_pmie_t:s0
tcontext=system_u:object_r:hostname_exec_t:s0 tclass=file permissive=1
type=AVC msg=audit(1499941609.723:671): avc:  denied  { lock } for
pid=9773 comm="runlevel" path="/run/utmp" dev="tmpfs" ino=1947
scontext=system_u:system_r:pcp_pmie_t:s0
tcontext=system_u:object_r:initrc_var_run_t:s0 tclass=file permissive=1
```

典型的なポリシー作成手順 (3)

- 「このプログラムがどう動作するべきか」について知見があれば操作のうち失敗してもよい (または失敗するべき) 操作は無視する
- 知見がない場合は操作を禁止した場合の影響範囲が予想できないため、基本的には記録された操作を全て許可する
  - 操作対象が特定少数のファイルなど明らかなパターンがある場合はタイプを追加してできるだけ制限できるよう調整する
- 全て許可する場合は audit2allow という専用のコマンドがある。
  - 監査ログの出力に含まれている操作を許可するポリシーを生成
  - audit2allow の出力を見て非常に一般的な場合は絞れないか調査する

典型的なポリシー作成手順 (4)

- 生成したポリシーを読み込ませて再度テストを行い、抜け漏れを減らしていく。
- 「不要な操作」を禁止した場合、ポリシー更新の上 SELinux を enforcing mode で実行して動作させて失敗させ、プログラムの動作に影響がないことを確認する。
  - 過剰なログ出力を抑えるため dontaudit(ログに記録しないポリシー) の記述を考える

SELinux の利用時対応

- 導入後にポリシーで許可されていない権限を必要とした場合、操作が失敗する
  - 副作用が見えない場合からプログラムの異常終了まで失敗の影響は様々
- 問題への対応
  - 監査ログに「いつ、どのタイプのプログラムが、どのタイプのリソースに、何をしようとして失敗した」と記録されるので確認の上ポリシーを更新する
  - ausearch -c コマンド名 --raw でプログラムに対応するログを検索
- ポリシーには、抜け漏れや間違いがある前提で作業する
  - ある時点でポリシーが完全なものであっても、対象ソフトウェアの動作がかわることもある

setroubleshoot

- SELinux による拒否への対応を半自動化
  - 監査ログを監視して拒否を検出
  - 関連する boolean などを探す
  - 関連する boolean を変更する、許可するためのポリシーを作成する、バグレポートを提出するなどの対策を提案
- 利用方法は http://red.ht/2gxfIAY

SELinux モジュールの導入・配布

- SELinux モジュールの導入・配布には rpm パッケージを使うのが一般的
  - 作成手順の中で紹介した sepolicy generate コマンドはパッケージ用 spec ファイルのひながたも生成する

関連資料

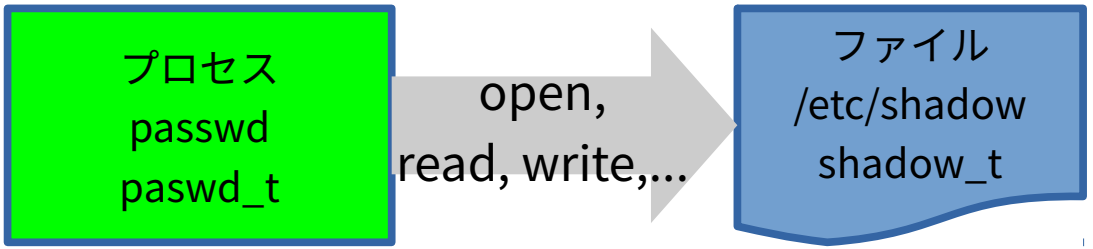
- SELinux ユーザーおよび管理者のガイド
  - 主に SELinux 一般とターゲットポリシーの説明
  - http://red.ht/2v9Gohj
- SELinux for Red Hat developers
  - 新規に作成したプログラム用に SELinux モジュールを作成するホワイトペーパー
  - http://red.ht/2vTXDWx
- SELinux Notebook
  - SELinux project のリファレンスドキュメント
  - https://selinuxproject.org/page/Category:Notebook
- SELinux System Administration Second Edition
  - SELinux を使いはじめる前に概要を掴むためにちょうど良い本
  - http://bit.ly/2jaK5St

backup slides

ポリシーの内容確認

- どのようなモジュールがあるか (どのようなプログラム向けのポリシーがあるか)
  - semanage module -l
- ターゲットポリシーを調整する boolean を探す
  - semanage boolean -l
  - man 8 passwd_selinux のような
    モジュール名 +”_selinux” の man page を読む
  - ドキュメント「 SELinux ユーザーおよび管理者のガイド」に
    主要なものの説明

現在のポリシーで
何が許可されているか確認する



passwd コマンドが /etc/shadow へのオープン、読み書きを許可される例

passwd_t が shadow_t に対して可能な操作を検索

```
$ sesearch -A -s passwd_t -t shadow_t
allow passwd_t shadow_t:file { append create getattr ioctl link lock open
read relabelfrom relabelto rename setattr unlink write };
```

→ file クラスとして append, create, getattr,.... の操作が許可されている

shadow_t に write できるタイプを検索する例

```
$ sesearch -A -t shadow_t -c file -p write
allow cockpit_session_t shadow_t:file { append create getattr ioctl link
lock open read rename setattr unlink write };
allow files_unconfined_type file_type:file { append audit_access create
execute execute_no_trans getattr ioctl link lock mounton open quotaon read
relabelfrom relabelto rename setattr swapon unlink write };
```

(以下略)

SELinux 関連 log

- SELinux は他の kernel による権限管理と異なり、監査ログの仕組みを持っている
  - SELinux による拒否が発生しているかの確認が容易
  - 過剰なログ出力を回避するため、 dontaudit ルールでログ出力をしないようポリシーが書かれる場合がある
    - 必要な場合は semodule -DB コマンドにより dontaudit ルールを無効化できる
- 監査ログは syslog ではなく auditd がファイルに保存する
  - デフォルトでは /var/log/audit/audit.log
- 監査ログの一部が SELinux に関連する
  - SELinux による拒否は以下コマンドで検索できる：
    ausearch -m avc,user_avc,selinux_err,user_selinux_err --raw
  - 各 type については http://red.ht/2vEJ8HJ

AVC denial log を読む

```
type=AVC msg=audit(1499941607.383:669): avc:  denied
{ execute_no_trans } for  pid=9626 comm="pmie_check"
path="/usr/bin/hostname" dev="dm-1" ino=101536461
scontext=system_u:system_r:pcp_pmie_t:s0
tcontext=system_u:object_r:hostname_exec_t:s0 tclass=file permissive=1
```

- type=AVC: カーネル内のポリシーチェック関連
- msg=audit(14999...:669): 監査された時刻のタイムスタンプ (epoch からの経過秒数) と、同じイベントに関連した複数行のログをまとめるシリアル番号
- denied { execute_no_trans }: “execute_no_trans” に分類される操作をしようとして拒否 (denied) された
- pid=, comm= : 操作するプロセスの情報
- path=, dev=, ino=: 操作されるファイルの情報 (class により変わる)
- scontext: 操作するリソースのコンテキスト
- tcontext: 操作されるリソースのコンテキスト
- tclass: 操作されるリソースを何として評価したか (file として)
- permissive: selinux の動作モード