LinuxCon North America 2012 (San Diego)

CaitSith
新しいルールベースのカーネル内アクセス制御

熊猫さくら

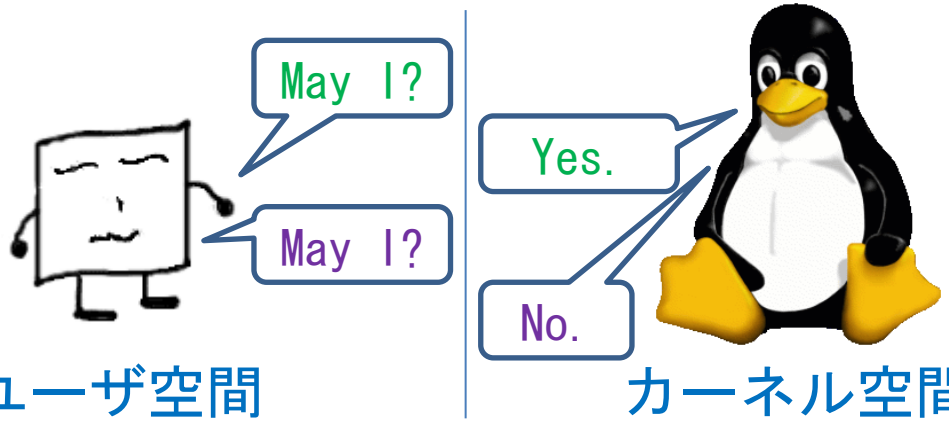
2012/8/29-2012/8/31

私は誰？

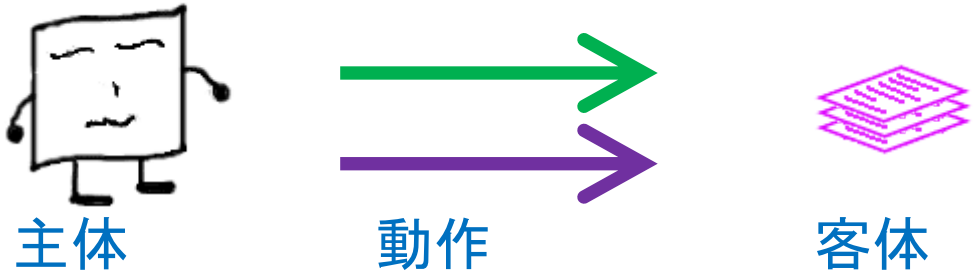
- 元職業プログラマ
- Linuxとの関わり
  - 2001.10-2003.3　Linuxシステム上で動作するユーザ空間アプリケーションの開発
  - 2003.4-2012.3　Linuxシステムのセキュリティ向上のためのカーネル機構の開発
  - 2012.4-　Linuxシステムのトラブル対応のためのユーザサポート窓口

今日はどんな話をするの？

- セキュリティ、特にカーネル内のアクセス制御の話

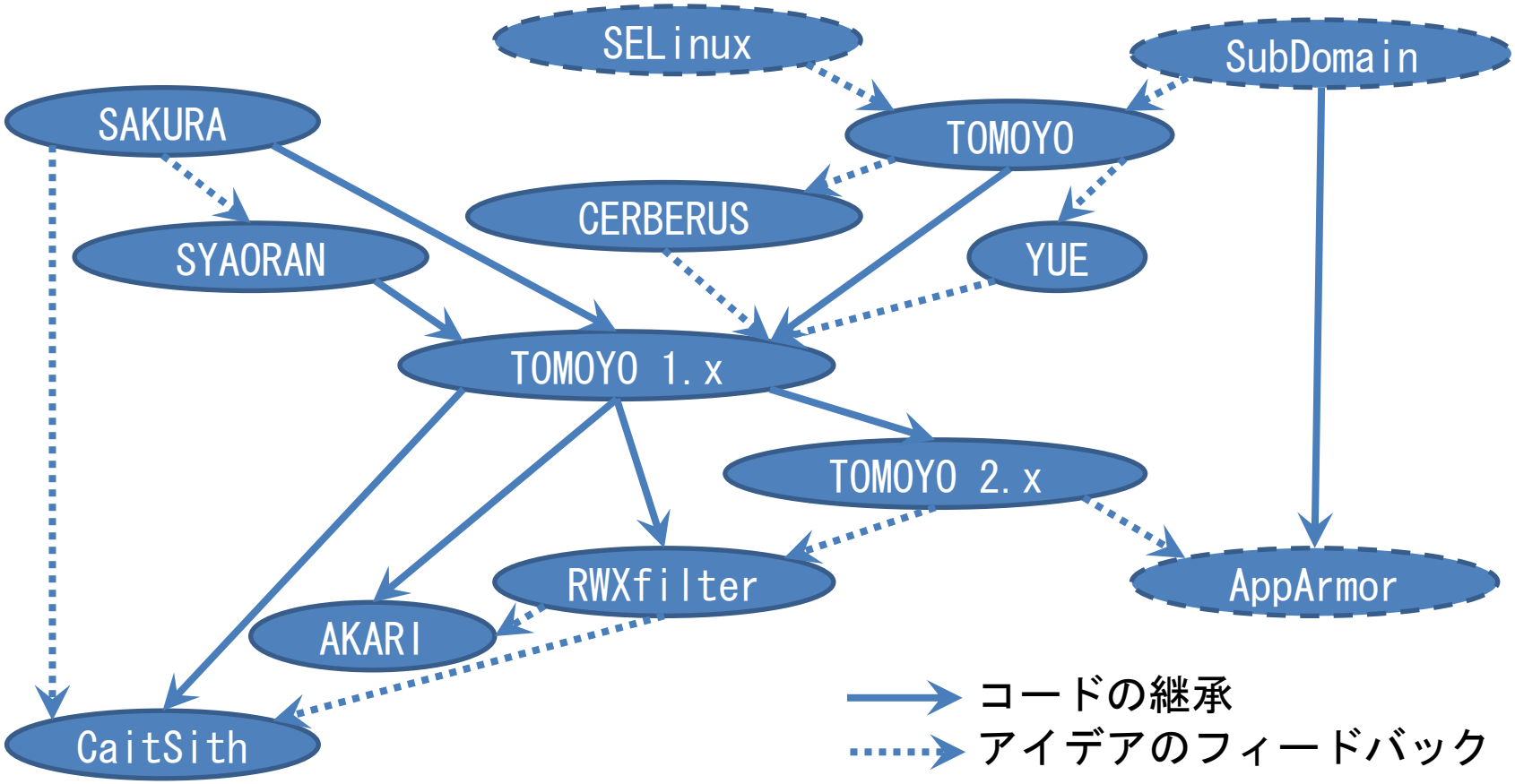


- 主体(プロセス)，動作，客体(資源)
  - たったこれだけ。でも、利用者のニーズに合ったルールを作るのはとても大変。



登場人物は？

- 私の話に登場するセキュリティ機構



この資料の構成は？

- 第１章・・・はじめに：CaitSithまでの出来事の要約
  - カーネル内アクセス制御に興味のある利用者向け
- 第２章・・・アクセス制御機能強化路線で経験したこと
  - アクセス制御機構を開発している開発者向け
- 第３章・・・使い勝手優先路線で経験したこと
  - 単純なカーネル内アクセス制御を探している利用者向け
- 第４章・・・CaitSith
  - 私の提案に興味のある利用者向け

第１章

はじめに：CaitSithまでの出来事の要約

セキュリティは十人十色？

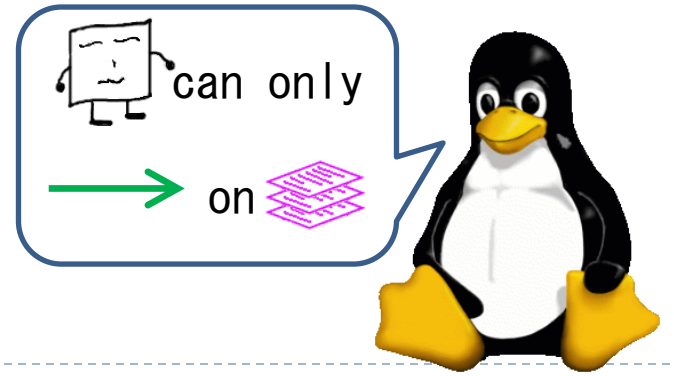
- 「セキュリティ」という言葉に対して人々が思い浮かべる内容は、彼らのスキルや信念により異なる。
  - １つのやり方を押しつけるのではなく、選択肢を与えることが大切と考える。
- 私の信念では、見える化が重要だと考える。
  - 今日のセキュリティ問題の多くは見えないことに起因する。
  - トレーサビリティ（追跡可能性）は納得感にも繋がる。

カーネル内でアクセス制御を行う試み

- 脅威はユーザ空間での振る舞いにある。
- しかし、ユーザ空間でのアクセス制御には問題がある。
  - プロセスの制御を奪われると回避されてしまう。
  - アクセス制御のレベルや粒度がバラバラである。
- ユーザ空間でのアクセス制御に加えて、カーネル空間でもアクセス制御をしてみよう。
  - カーネル内のアクセス制御で実現できることには限りがあるけれど、ベースラインとしての制限機能を提供することはきっと役に立つだろう。

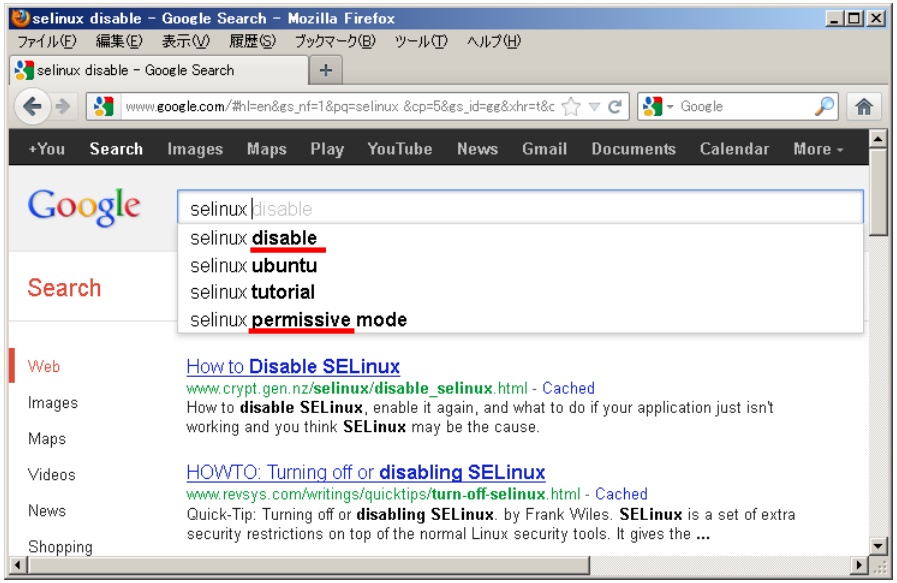
強制アクセス制御(MAC)

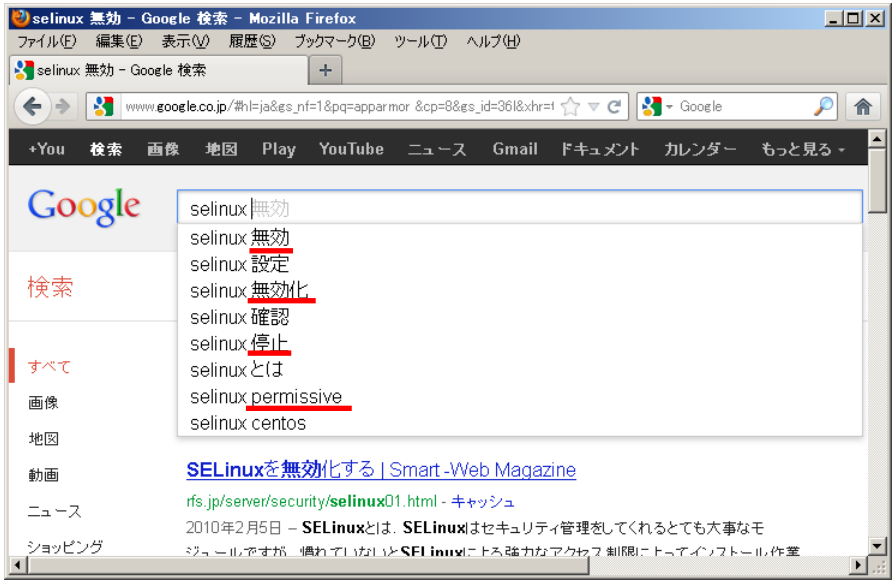
- カーネル空間でアクセス制御を行う実装
  - カーネル空間で行われるので、回避できない。
- 現時点では、SELinux, SMACK, TOMOYO, AppArmorが標準のLinuxカーネルに含まれている。
  - これらはみなLinuxセキュリティモジュール(LSM)インタフェースを用いた**ホワイトリスト**方式である。
  - これらは**主体**の視点からアクセス制御を行っている。



ディストリビュータのカーネルではいくつか有効にされているけれども．．．

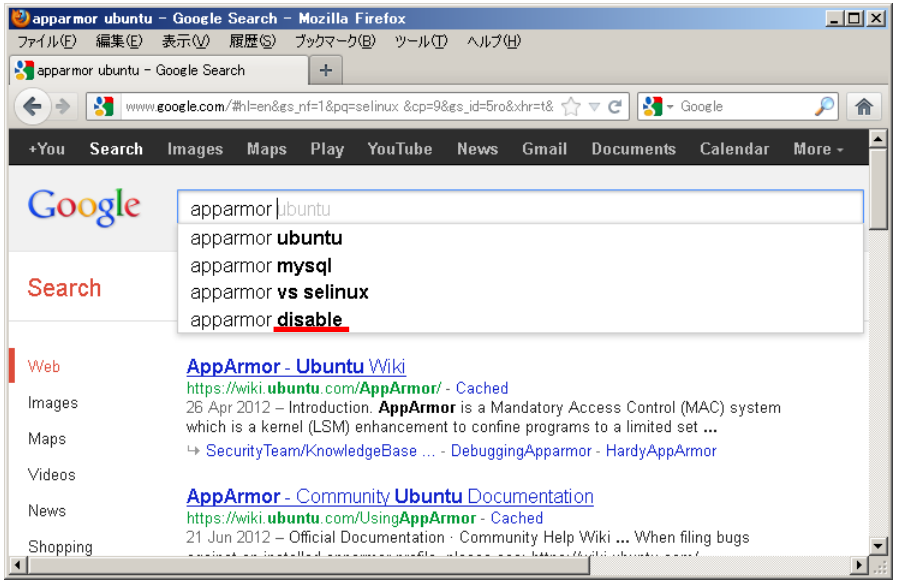
- 多くの人は未だにSELinuxを無効にしている。





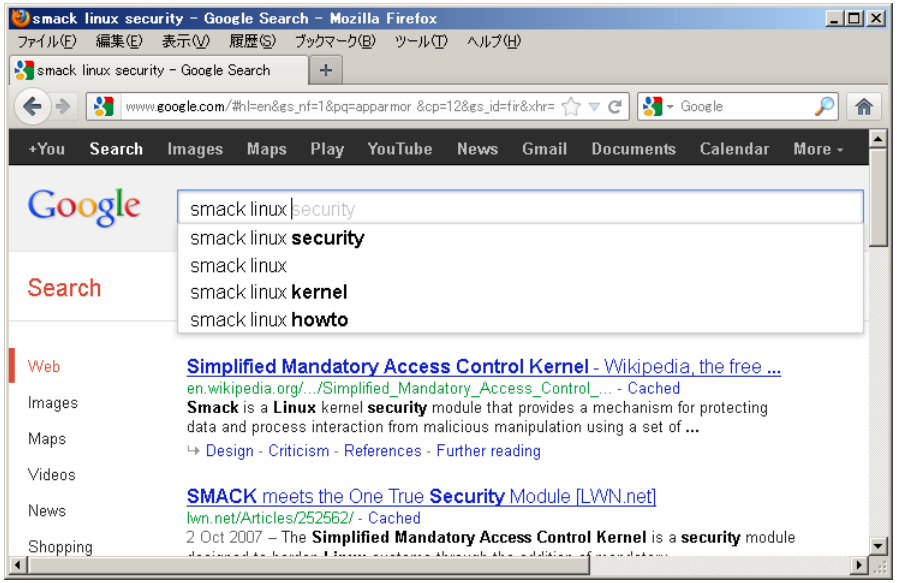
ディストリビュータのカーネルではいくつか有効にされているけれども．．．

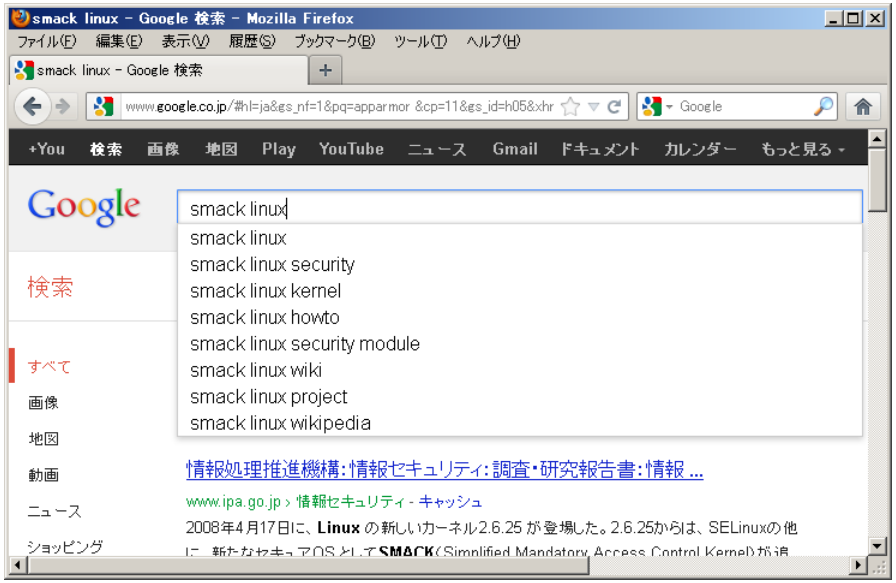
- SELinuxより簡単だと言われていたAppArmorでさえも無効にされている。



ディストリビュータのカーネルではいくつか有効にされているけれども．．．

- SMACKの場合は？

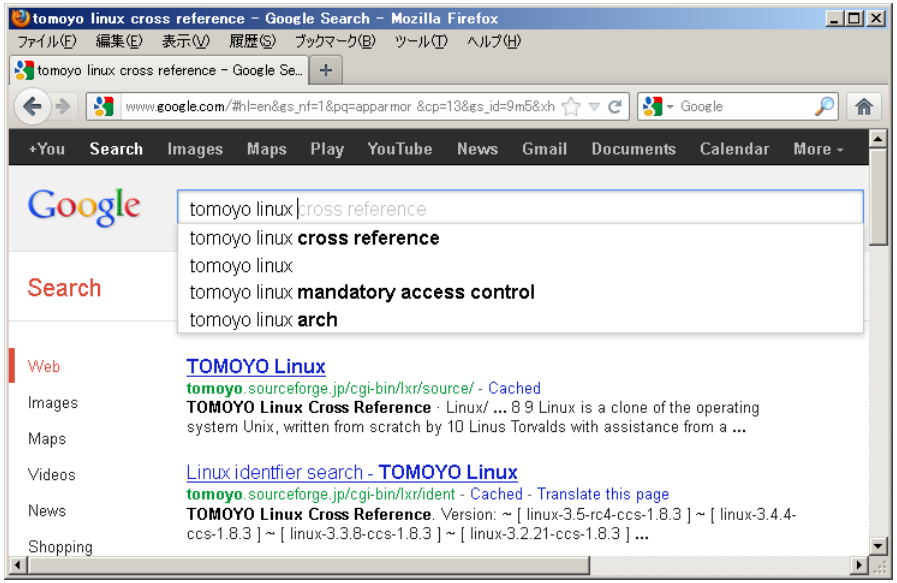


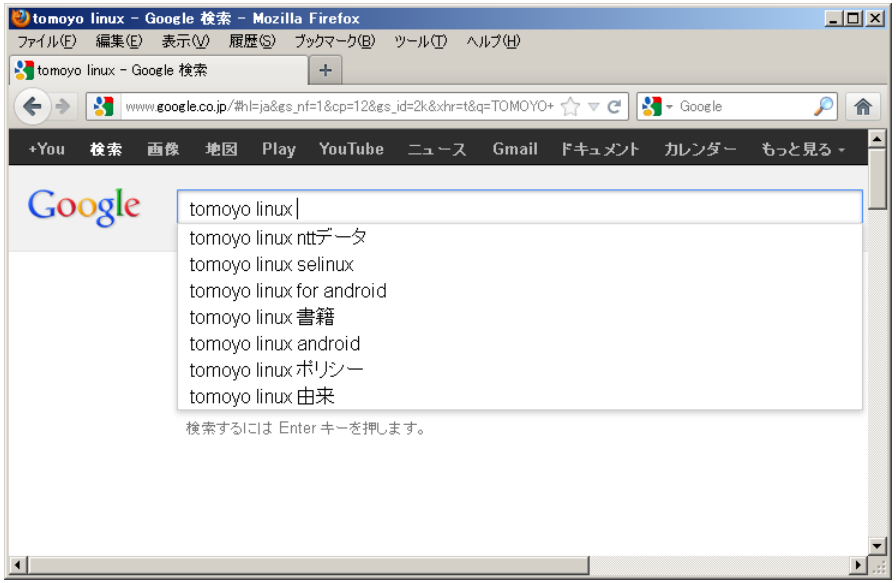


- SMACKは利用者の明示的な設定なしには有効化されないので、まだ無効化されていない。

ディストリビュータのカーネルではいくつか有効にされているけれども．．．

- TOMOYOの場合は？





- TOMOYOは利用者の明示的な設定なしには有効化されないので、まだ無効化されていない。

ディストリビュータのカーネルではいくつか有効にされているけれども．．．

- 無効にする理由？
  - 設定内容を理解せずに使うのが怖い
  - ホワイトリスト方式で必要な許可を与え忘れるのが怖い
- ドキュメントはたくさんあるけれど
  - 長すぎて読んでいられない
  - 利用者向けではなく開発者向け

「イエスかノーか」問題

- 理想的には、全ての主体と全ての客体にアクセス制御が適用されていること。
- 現実には、一部の主体に対してアクセス制御が適用できれば「上出来」と考えられている。
- 問題が発生した場合、ポリシーの設定方法と設定内容を理解していない限り**完全に無効化**せざるを得なくなってしまう。

TOMOYOが管理可能性にこだわり続けた理由？

- 出来合いのポリシーを配布するような余裕が無い。
  - SELinux(RedHat)やAppArmor(SUSE/Ubuntu)のようなディストリビュータをTOMOYOは持っていない。
- 出来合いのポリシーを使うとトラブル対応ができなくなる。
  - TOMOYOもSELinuxやAppArmorが迷い込んでいる道を辿ることになってしまう。
- 許可／禁止したいことは人それぞれである。
  - 全ての利用者のニーズを満たすような出来合いのポリシーなんて作れっこない。
- スイッチが１つしかないとトラブル発生時に無効化せざるを得ない。
  - 「イエスかノーか」問題

ポリシーを定義する上での問題

- 「アクセス制御」とは、事前に定義された規則に従ってアクセス要求を制限すること。
  - 利用者が規則を事前に定義できることが暗黙の前提条件となっている。
  - しかし、規則を事前に定義するためには、利用者はLinuxシステムの内部構造に詳しくなければいけない。
  - 多くの利用者にとっては普段意識しない世界なので、Linuxシステムの内部構造に詳しいことを期待できない。
- しかし、規則を定義するための負担は重要視されてこなかった。
  - TOMOYOではこの負担をどうにかしようと注力してきた。

TOMOYOが無効化されないためにしてきた工夫

- プロファイルを用いて動作単位で有効／無効を指定できるようにした。
  - 利用者のスキルに応じて制限の対象とする動作を選択できる。
- プロファイルを用いてドメイン単位で有効／無効を指定できるようにした。
  - 利用者のスキルに応じて制限の対象とする主体を選択できる。

TOMOYOが無効化されないためにしてきた工夫

- ポリシー違反を対話的に処理できるようにした。
  - 想定外のアクセス要求でもケースバイケースで判断できる。
- ドメイン遷移をツリー構造にすることで、全ての状態を理解可能なものにした。
  - 何が行われているのかが理解できる。

そんな中、ＲＨＥＬ利用者から思わぬ声が．．．

- 「特定のプロセスに対して制限をするのではなく、
  特定の資源(ファイル)に対して制限を行いたい」
  - TOMOYOでは、動作単位およびドメイン単位で有効／無効を指定することができるようになっていたが、ファイル単位で有効／無効を指定することには対応していなかった。

結局、既存のMAC実装は利用者のニーズに応えることができていたのだろうか？

- 利用者のニーズはホワイトリストであるとは限らないし、プロセス視点で設定したいとも限らない
  - 資源の視点から設定したい利用者もいる
  - ドメインの管理に興味のない利用者もいる
  - これらはTOMOYOにとって回避できない障壁

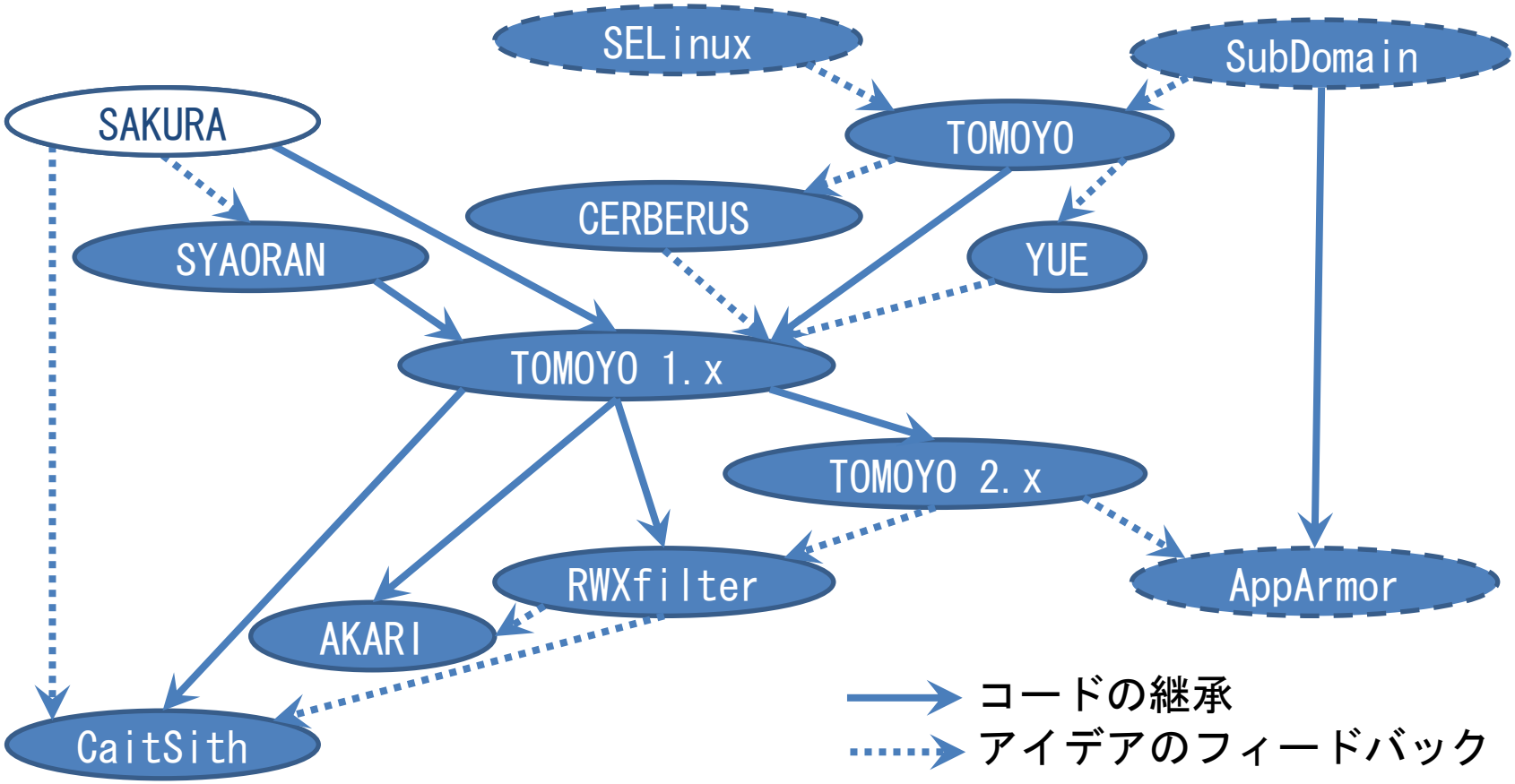
- 抜本的な方針転換が必要なのかもしれない
  - これがCaitSithを開発することになったきっかけ

第２章

アクセス制御機能強化路線で経験したこと

SAKURA (2003.4-)

- ポリシー管理を伴わずにLinuxシステムを保護する試み



SAKURA

- SELinuxを試したものの、難しすぎてすぐに諦めた。
  - 改ざん防止に限定すれば、ポリシー管理を
    省略できるのでは？
- コードネーム：Security Advancement Know-how Upon Read-only Approach for Linux
- SAKURAのトピック
  - 読み込み専用マウントによる改ざん対策
  - システム全体でのアクセス制限
  - ユーザ空間プログラムの修正による自発的権限放棄

読み込み専用マウントによる改ざん対策

- 改ざんのリスクを減らすために、可能な限り読み込み専用でファイルシステムをマウントする。
  - 読み込み専用でよいディレクトリと読み書きが必要なディレクトリとを分離するのを助けるために、-EROFSエラーとなったパス名を報告するようカーネルを修正した。
  - (/varや/tmpなど)書き込み可能でなければならないパーティションを除く全てのパーティションを読み込み専用モードでマウントできるようにし、ファイルシステムを介さずに改ざんする攻撃(読み込み専用モードでよいパーティションに対応するブロックデバイスファイルに対する書き込み)から保護するために読み込み専用メディアに格納するようにした。

システム全体でのアクセス制限

- 読み込み専用パーティションの上に、読み書き可能なパーティションをマウントされてしまっては元も子もない。
  - 名前空間の操作(mount, chroot, pivot_rootなど)をシステム全体として制限した。
  - 設定例は以下のようになる。

```
allow_mount devpts /dev/pts/ devpts 0x0
allow_mount any / --remount 0x0
allow_mount securityfs /sys/kernel/security/
securityfs 0x0
allow_mount none /proc/sys/fs/binfmt_misc/
binfmt_misc 0x0
allow_chroot /etc/avahi/
allow_chroot /var/empty/sshd/
```

プロセスの名前を指定していないことに注目

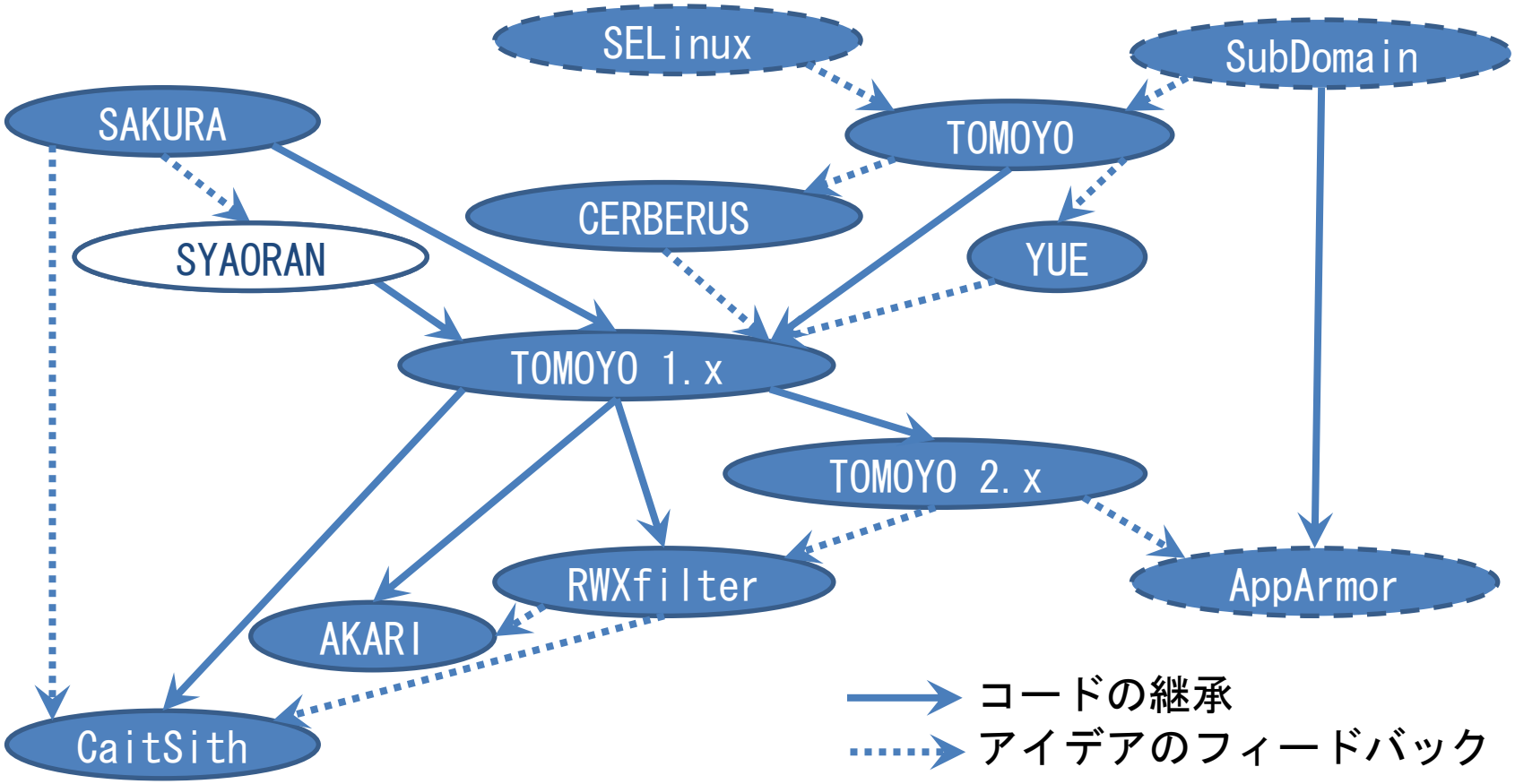
ユーザ空間プログラムの修正による自発的権限放棄

- システム全体でのアクセス制限をより効果的にするために、Linux 2.4カーネルのタスク構造体に独自のフィールドを追加することにより、自発的に権限を放棄できるようにした。
  - ユーザ空間のプログラムに対して、タスク構造体単位でexecve(), chroot(), pivot_root(), mount()の呼び出しおよび再び実効UIDが0になる権限を放棄できるようにした。
    - execve()成功後に権限が復活する一時的な放棄
    - execve()成功後も権限が復活しない永続的な放棄
- ユーザ空間のプログラムを修正する余力が無かったため、この機能はTOMOYO 1.4で削除された。

  => しかし、より包括的な権限放棄機構がseccomp mode 2としてLinux 3.5カーネルに採用された。

SYAORAN (2004.10-)

- /devパーティション向けの改ざん防止ファイルシステム



SYAORAN

- SAKURAを使うことで/パーティションを読み込み専用マウントできるが、/devパーティションを読み込み専用マウントすることはできない。
  - 既存の/dev用ファイルシステム（devfsやdevtmpfsなど）はユーザ空間からのリクエストによってディレクトリエントリの修正が可能。
  - /devパーティションのファイルが改ざんされたら大問題
    - もし/dev/nullが/dev/zeroの属性を持っていたら？
- コードネーム：Simple Yet All-important Object Realizing Abiding Nexus
- SYAORANのトピック
  - /devパーティション向けの改ざん防止ファイルシステム

/devパーティション向けの改ざん防止ファイルシステム

- tmpfsをベースに、属性チェックロジックを追加した/dev用ファイルシステムを作成した。
  - このファイルシステムを使うと、ファイル名とその属性の対応付けを強制できる。
  - 例えば、/dev/nullは常にキャラクタ-1-3の属性を持ち、/dev/zeroは常にキャラクタ-1-5の属性を持つ。

/devパーティション向けの改ざん防止ファイルシステム

- 設定ファイルの例は以下のようになる。

#filename	perm	owner	group	flags	type	major	minor
pts	755	0	0	0	d		
shm	755	0	0	0	d		
null	666	0	0	0	c	1	3
zero	666	0	0	0	c	1	5
random	644	0	0	0	c	1	8
urandom	644	0	0	0	c	1	9
tty	666	0	0	0	c	5	0
tty0	600	0	0	12	c	4	0
tty1	600	0	0	12	c	4	1

許可される操作(create/delete/chmod/chown/chgrp) が制限されていることに注目

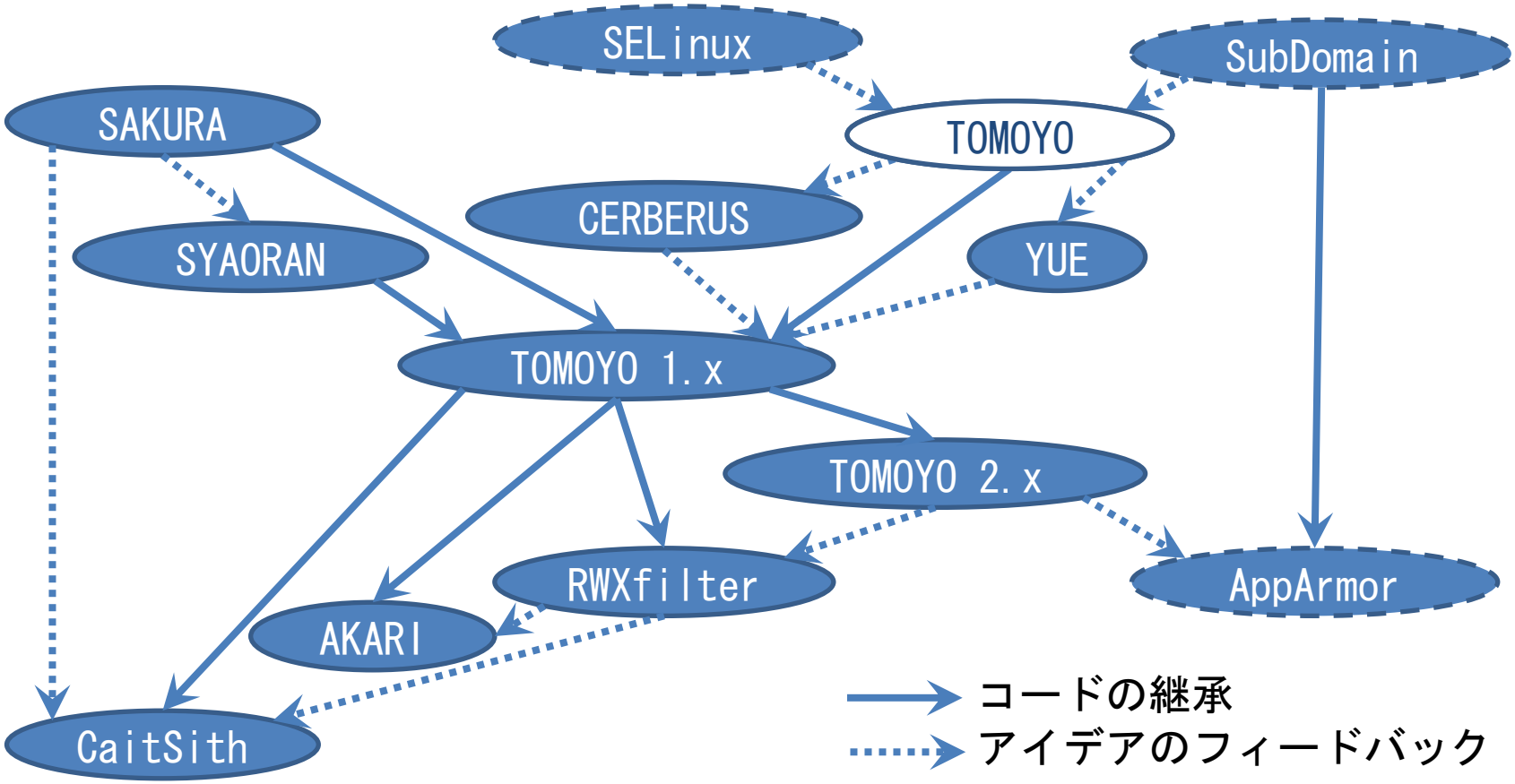
ポリシー管理は不可避？

- /devにSYAORANファイルシステムを使い、それ以外を
  可能な限りSAKURAで読み込み専用化することで、
  改ざんされるリスクを減らすことができる。
  - SAKURAとSYAORANという組み合わせにより、読み込み専用
    メディア内のファイルの改ざんを防止できるが、
    読み込み専用メディアに格納するとソフトウェア
    アップデートもできなくなってしまう。
  - 改ざんから保護しながらもソフトウェアアップデートも
    できるようにするために、ポリシーに基づく制限を
    使うことも検討するべきである。

  => TOMOYOへと続く

TOMOYO (2003.7-)

- 管理できるポリシーを実現する試み

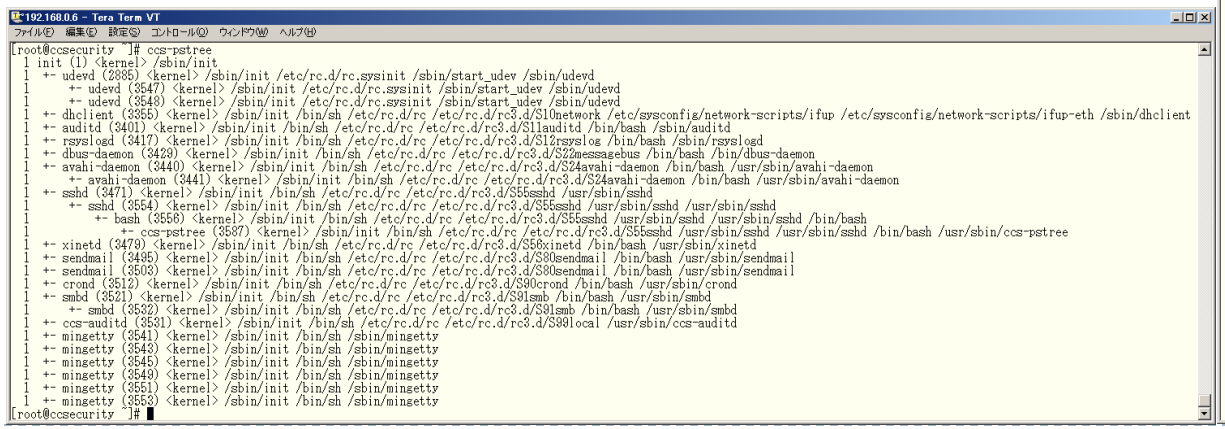


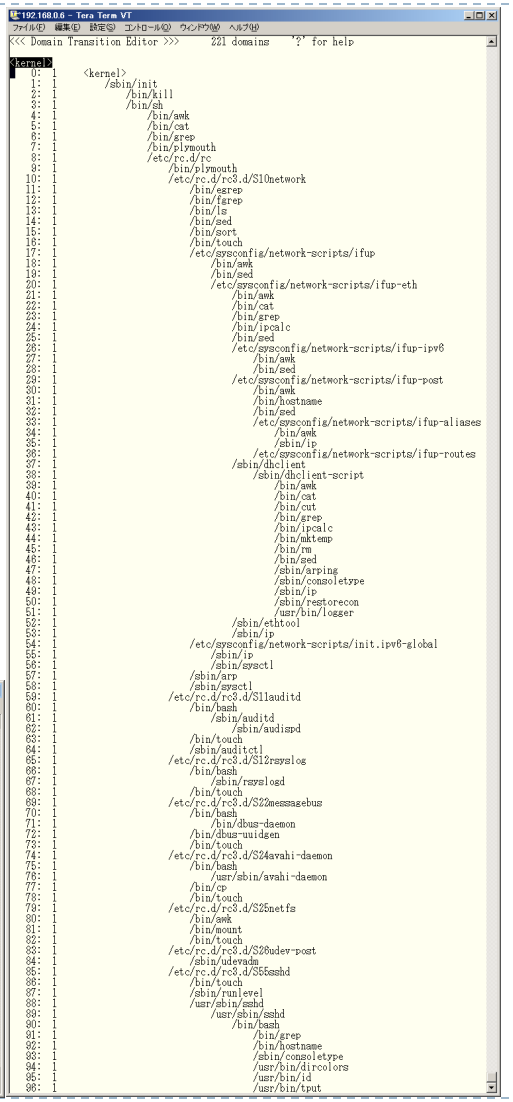
TOMOYO

- SELinuxのポリシーは難しすぎて使えない。必要な部分だけカバーしてくれるオリジナルのポリシーを作れないだろうか？
  - 観測されたプロセスの振る舞いのみ許可するポリシーを作ってみよう。
- コードネーム：Task Oriented Management Obviates Your Onus on Linux
- TOMOYOのトピック
  - システム全体にわたるドメイン遷移追跡機能
  - ドメイン単位のアクセス要求追跡機能
  - ドメイン単位のアクセス要求制限機能

システム全体にわたるドメイン遷移追跡機能

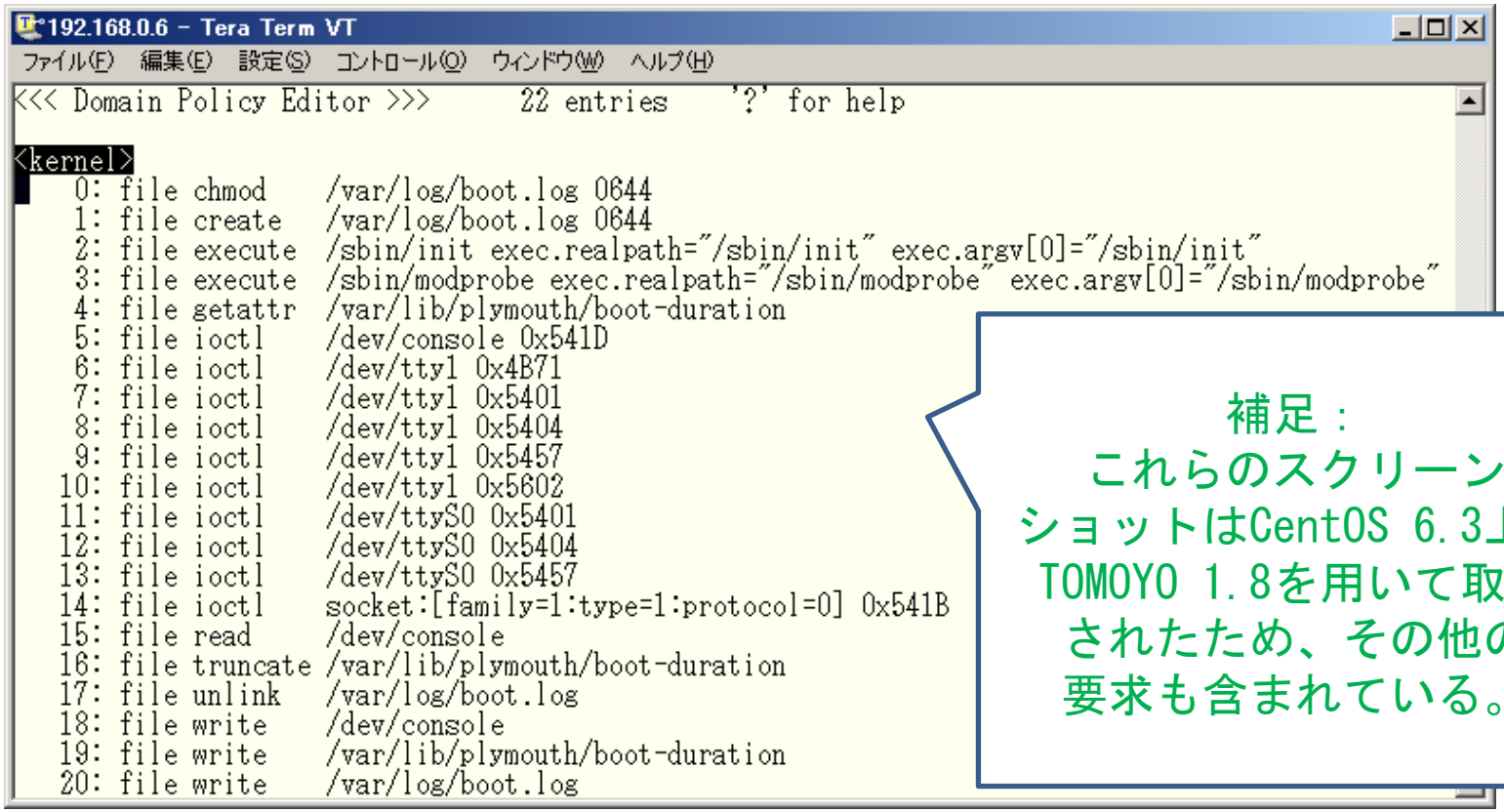
- Linux 2.4カーネルのタスク構造体に独自のフィールドを追加した。
  - fork()/execve()という仕組みを用いてツリー状の状態遷移を形成した。
    - それぞれの状態をドメインとして使用した。





ドメイン単位のアクセス要求追跡機能

- アクセス解析ツールとして始まった。
  - open()とexecve()要求をパス名を用いて追跡し、ドメインをキーとしてソートするようにした。



補足：
これらのスクリーンショットはCentOS 6.3上でTOMOYO 1.8を用いて取得されたため、その他の要求も含まれている。

ドメイン単位のアクセス要求追跡機能

192.168.0.6 - Tera Term VT

ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

```
<<< Domain Policy Editor >>>      70 entries    '?' for help

<kernel> /sbin/init
    0: file execute /bin/kill exec.realpath="/bin/kill" exec.argv[0]="/bin/kill"
    1: file execute /bin/sh exec.realpath="/bin/bash" exec.argv[0]="/bin/sh"
    2: file execute /etc/rc.d/rc.sysinit exec.realpath="/etc/rc.d/rc.sysinit" exec.argv[0]="/etc/rc.d/rc.sysinit"
    3: file getattr /etc/group
    4: file getattr /etc/init/
    5: file getattr /etc/init/control-alt-delete.conf
    6: file getattr /etc/init/init-system-dbus.conf
    7: file getattr /etc/init/kexec-disable.conf
    8: file getattr /etc/init/plymouth-shutdown.conf
    9: file getattr /etc/init/prefdm.conf
   10: file getattr /etc/init/quit-plymouth.conf
   11: file getattr /etc/init/rc.conf
   12: file getattr /etc/init/rcS-emergency.conf
   13: file getattr /etc/init/rcS-sulogin.conf
   14: file getattr /etc/init/rcS.conf
   15: file getattr /etc/init/serial.conf
   16: file getattr /etc/init/splash-manager.conf
   17: file getattr /etc/init/start-ttys.conf
   18: file getattr /etc/init/tty.conf
   19: file getattr /etc/ld.so.cache
   20: file getattr /etc/nsswitch.conf
   21: file getattr /etc/passwd
   22: file getattr /lib/libc-2.12.so
   23: file getattr /lib/libdbus-1.so.3.4.0
   24: file getattr /lib/libgcc_s-4.4.6-20120305.so.1
   25: file getattr /lib/libnih-dbus.so.1.0.0
   26: file getattr /lib/libnih.so.1.0.0
   27: file getattr /lib/libnss_files-2.12.so
   28: file getattr /lib/libpthread-2.12.so
   29: file getattr /lib/librt-2.12.so
   30: file getattr proc:/self/fd/
   31: file ioctl   /dev/console 0x5401
   32: file ioctl   /dev/console 0x5402
   33: file ioctl   /dev/console 0x540B
   34: file ioctl   /dev/null 0x4B4E
   35: file read    /dev/console
   36: file read    /dev/null
   37: file read    /dev/urandom
   38: file read    /etc/group
   39: file read    /etc/init/
   40: file read    /etc/init/control-alt-delete.conf
   41: file read    /etc/init/init-system-dbus.conf
   42: file read    /etc/init/kexec-disable.conf
   43: file read    /etc/init/plymouth-shutdown.conf
   44: file read    /etc/init/prefdm.conf
   45: file read    /etc/init/quit-plymouth.conf
   46: file read    /etc/init/rc.conf
   47: file read    /etc/init/rcS-emergency.conf
   48: file read    /etc/init/rcS-sulogin.conf
   49: file read    /etc/init/rcS.conf
   50: file read    /etc/init/serial.conf
   51: file read    /etc/init/splash-manager.conf
   52: file read    /etc/init/start-ttys.conf
   53: file read    /etc/init/tty.conf
   54: file read    /etc/ld.so.cache
   55: file read    /etc/nsswitch.conf
   56: file read    /etc/passwd
   57: file read    /lib/ld-2.12.so
   58: file read    /lib/libc-2.12.so
   59: file read    /lib/libdbus-1.so.3.4.0
   60: file read    /lib/libgcc_s-4.4.6-20120305.so.1
   61: file read    /lib/libnih-dbus.so.1.0.0
   62: file read    /lib/libnih.so.1.0.0
   63: file read    /lib/libnss_files-2.12.so
   64: file read    /lib/libpthread-2.12.so
   65: file read    /lib/librt-2.12.so
   66: file read    /var/run/utmp
   67: file write   /dev/console
   68: file write   /dev/null
```

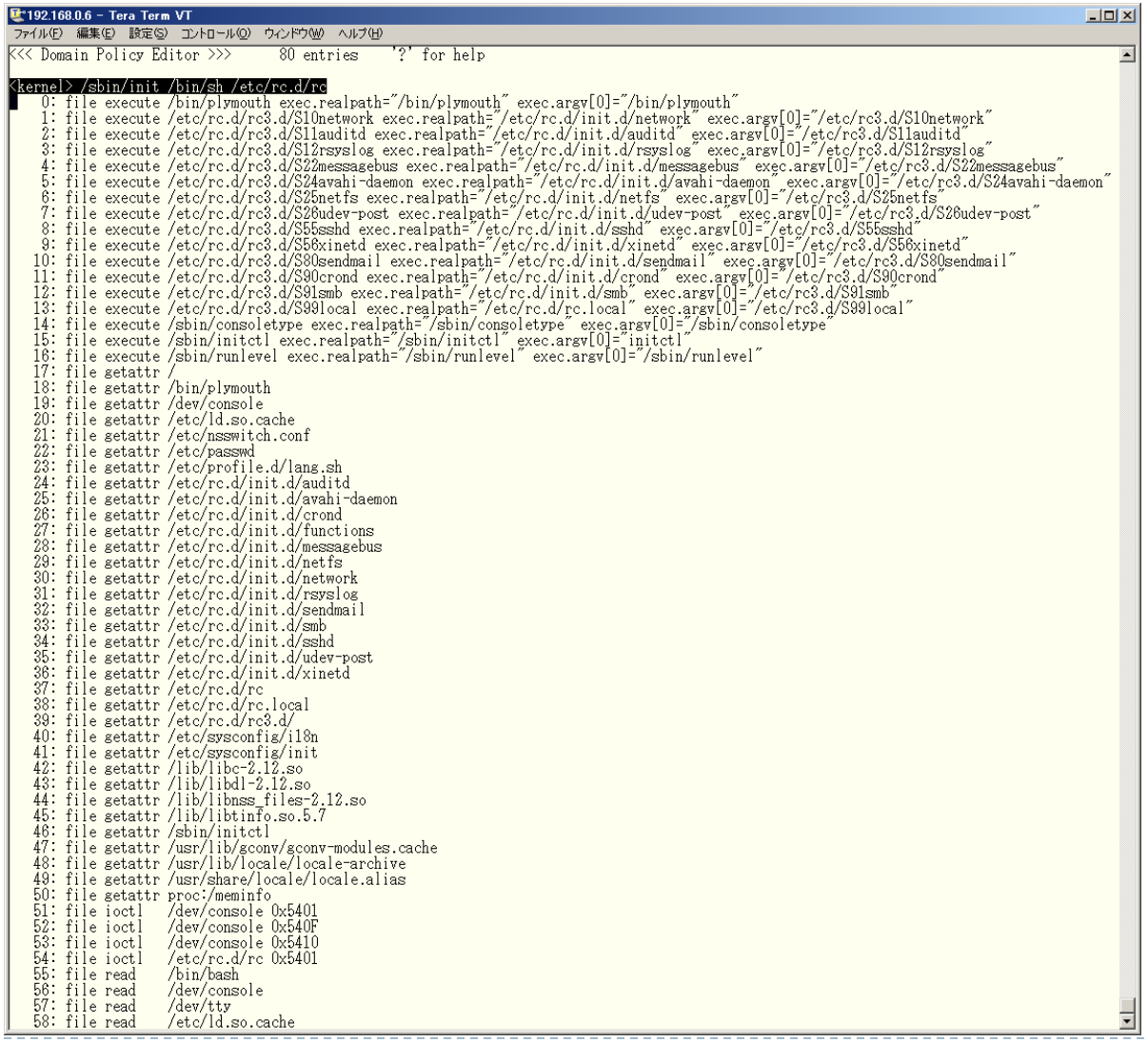
192.168.0.6 - Tera Term VT

ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

```
<<< Domain Policy Editor >>>      58 entries    '?' for help

<kernel> /sbin/init /bin/sh
    0: file execute  /bin/awk exec.realpath="/bin/gawk" exec.argv[0]="/bin/awk"
    1: file execute  /bin/cat exec.realpath="/bin/cat" exec.argv[0]="cat"
    2: file execute  /bin/grep exec.realpath="/bin/grep" exec.argv[0]="grep"
    3: file execute  /bin/plymouth exec.realpath="/bin/plymouth" exec.argv[0]="/bin/plymouth"
    4: file execute  /etc/rc.d/rc exec.realpath="/etc/rc.d/rc" exec.argv[0]="/etc/rc.d/rc"
    5: file execute  /sbin/initctl exec.realpath="/sbin/initctl" exec.argv[0]="initctl"
    6: file execute  /sbin/mingetty exec.realpath="/sbin/mingetty" exec.argv[0]="/sbin/mingetty"
    7: file execute  /sbin/telinit exec.realpath="/sbin/telinit" exec.argv[0]="telinit"
    8: file getattr  /
    9: file getattr  /bin/cat
   10: file getattr  /bin/grep
   11: file getattr  /etc/inittab
   12: file getattr  /etc/ld.so.cache
   13: file getattr  /etc/nsswitch.conf
   14: file getattr  /etc/passwd
   15: file getattr  /etc/sysconfig/init
   16: file getattr  /lib/libc-2.12.so
   17: file getattr  /lib/libdl-2.12.so
   18: file getattr  /lib/libnss_files-2.12.so
   19: file getattr  /lib/libtinfo.so.5.7
   20: file getattr  /sbin/initctl
   21: file getattr  /sbin/telinit
   22: file getattr  pipe:[2707]
   23: file getattr  pipe:[4815]
   24: file getattr  pipe:[4822]
   25: file getattr  pipe:[5304]
   26: file getattr  pipe:[5312]
   27: file getattr  proc:/meminfo
   28: file getattr  sysfs:/kernel/kexec_crash_size
   29: file ioctl    /dev/console 0x5401
   30: file ioctl    /dev/null 0x5401
   31: file ioctl    pipe:[2707] 0x5401
   32: file ioctl    pipe:[4815] 0x5401
   33: file ioctl    pipe:[5304] 0x5401
   34: file ioctl    pipe:[5312] 0x5401
   35: file read     /dev/
   36: file read     /dev/console
   37: file read     /dev/tty
   38: file read     /etc/ld.so.cache
   39: file read     /etc/nsswitch.conf
   40: file read     /etc/passwd
   41: file read     /etc/sysconfig/init
   42: file read     /lib/ld-2.12.so
   43: file read     /lib/libc-2.12.so
   44: file read     /lib/libdl-2.12.so
   45: file read     /lib/libnss_files-2.12.so
   46: file read     /lib/libtinfo.so.5.7
   47: file read     pipe:[2707]
   48: file read     pipe:[4815]
   49: file read     pipe:[5304]
   50: file read     pipe:[5312]
   51: file read     proc:/meminfo
   52: file truncate sysfs:/kernel/kexec_crash_size
   53: file write    /dev/console
   54: file write    /dev/null
   55: file write    /dev/tty
   56: file write    sysfs:/kernel/kexec_crash_size
```

ドメイン単位のアクセス要求追跡機能

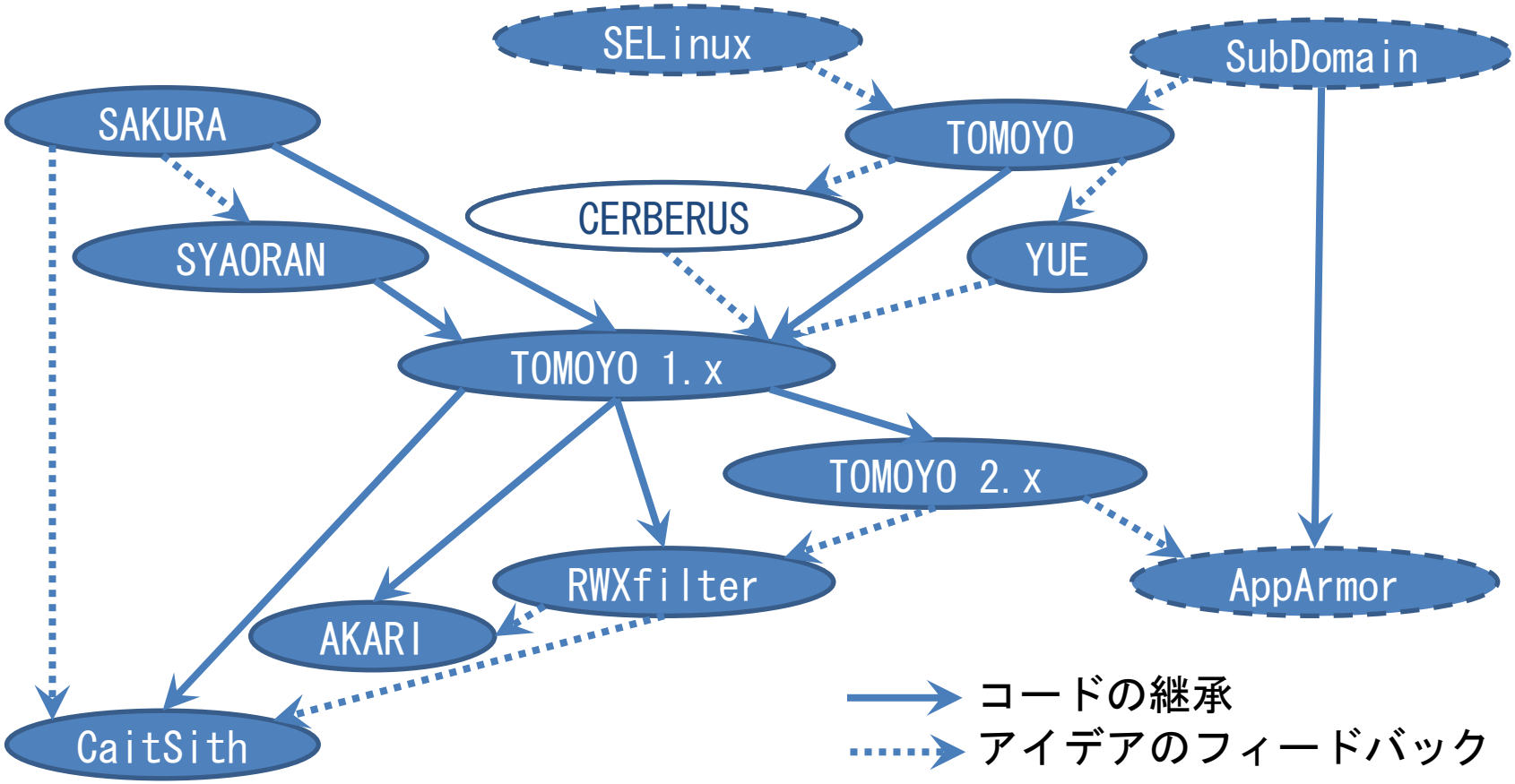


ドメイン単位のアクセス要求制限機能

- 出力からSELinuxのポリシーを生成することを試みた。
  - TOMOYOでのパス名をSELinuxのラベルにマッピング することができず、諦めた。
- 代わりに、観測された出力に基づいてアクセス要求を制限する機能を追加した。
  - この時点では、ディレクトリエントリを変更する操作を区別していない。
    - 言い換えると、粒度はDACのrwxに近い。

CERBERUS (2004.1-)

- ログインブルートフォース攻撃から保護するための、TOMOYOの応用例の一つ。

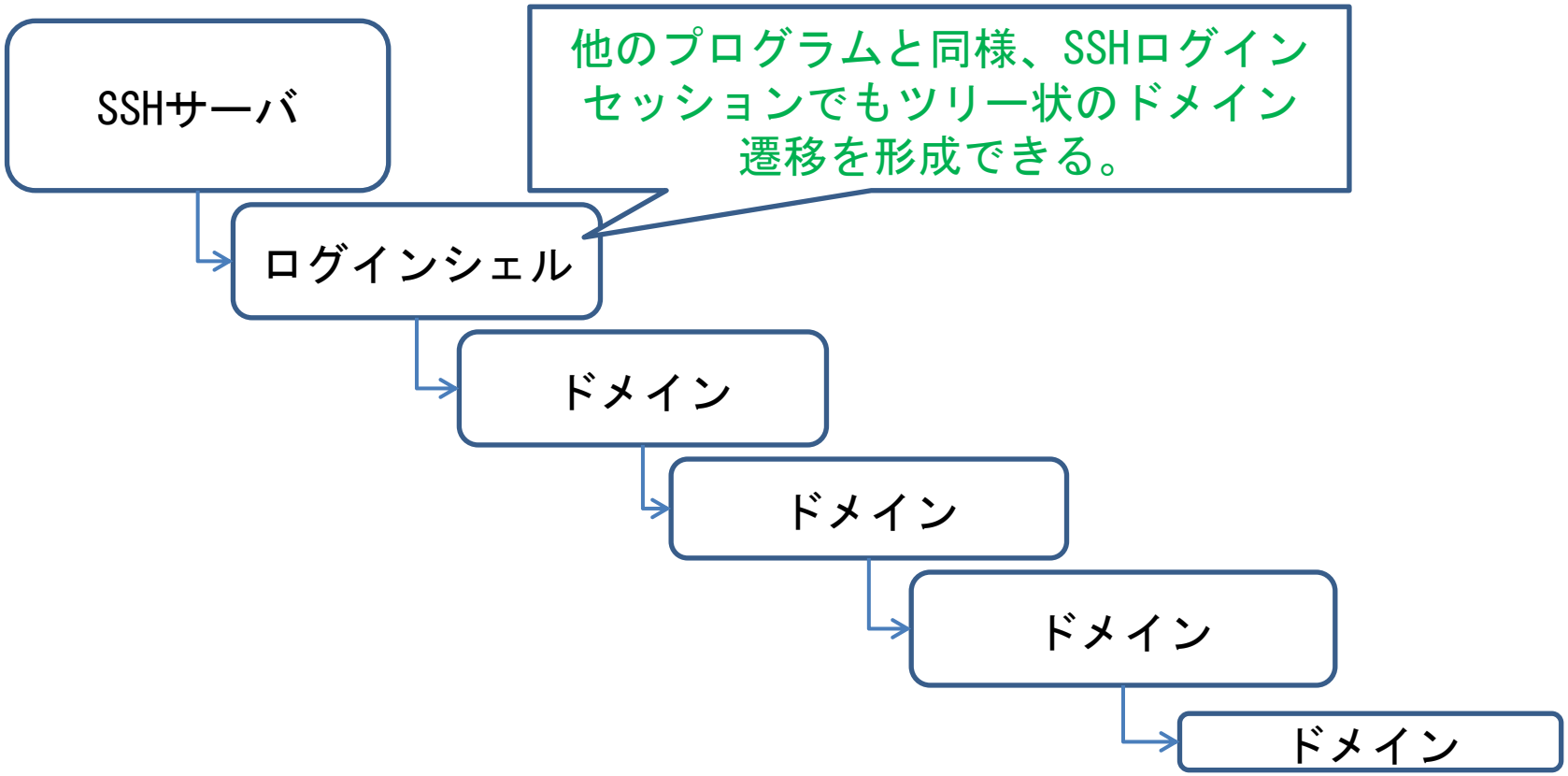


CERBERUS

- SSHブルートフォース攻撃はMACによる保護を突破してしまう。
  - 追加の認証を行うようにしてみたらどうだろうか？
- コードネーム：Chained Enforceable Re-authentication Barrier Ensures Really Unbreakable Security
- CERBERUSのトピック
  - 多重化されたユーザ認証を用いたブルートフォース対抗技術

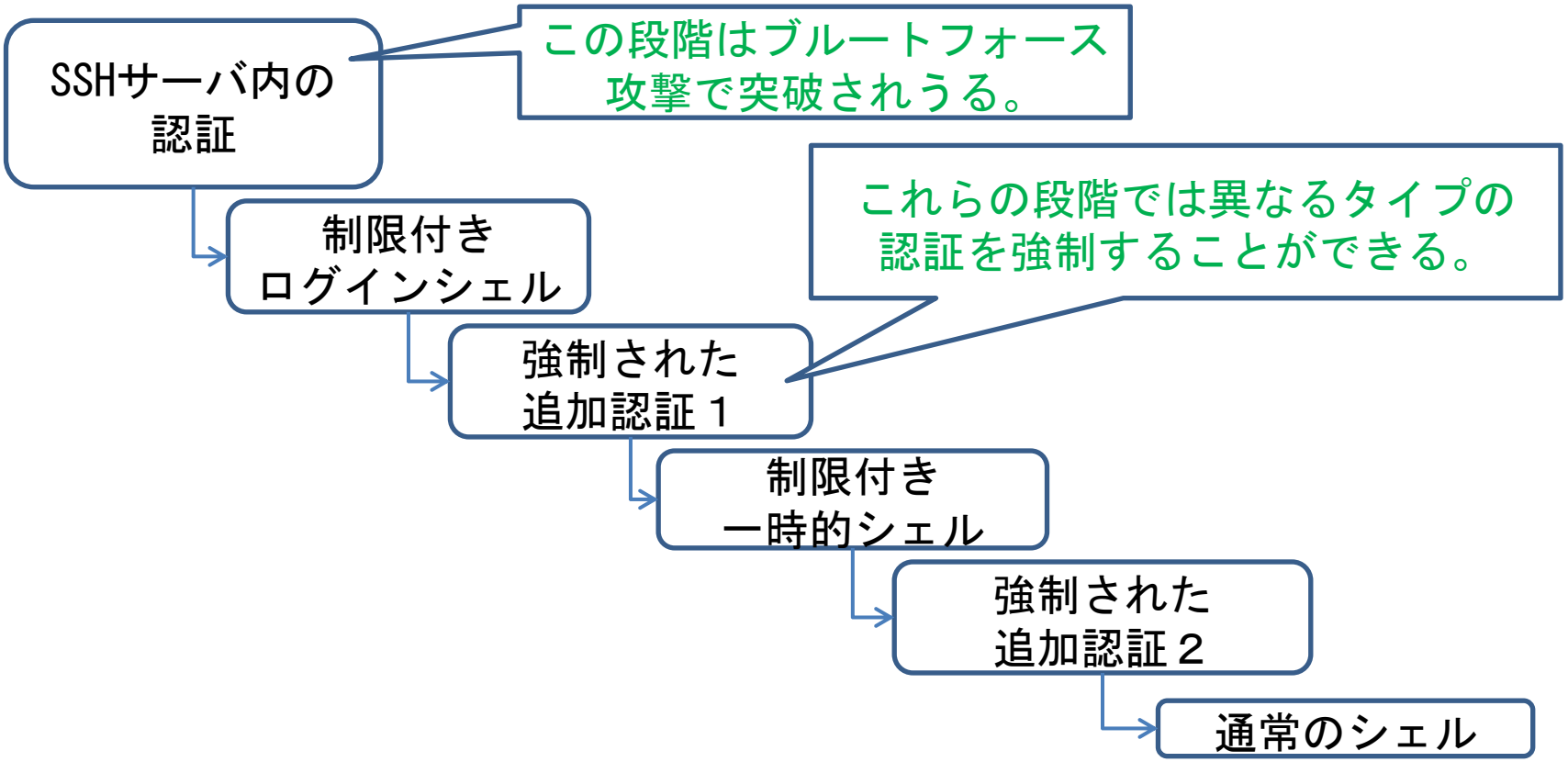
多重化されたユーザ認証を用いたブルートフォース対抗技術

- TOMOYOのツリー状の状態遷移を用いることで、ユーザ認証を複数回行うことができることに気が付いた。



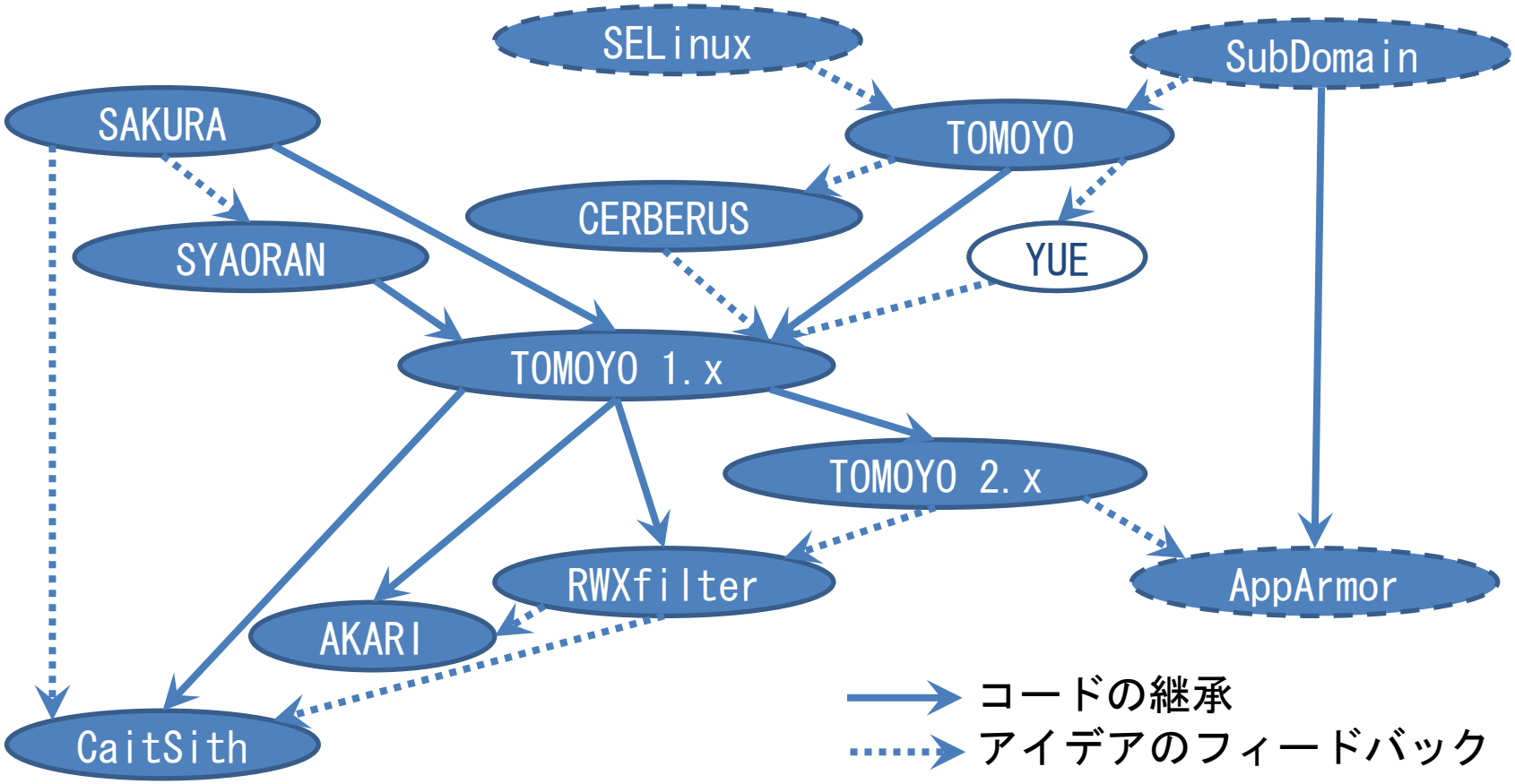
多重化されたユーザ認証を用いたブルートフォース対抗技術

- TOMOYOのツリー状の状態遷移を用いることで、ユーザ認証を複数回行うことができることに気が付いた。



YUE (2004.1-)

- 管理者業務のための権限を分割するための、TOMOYOの応用例の一つ。

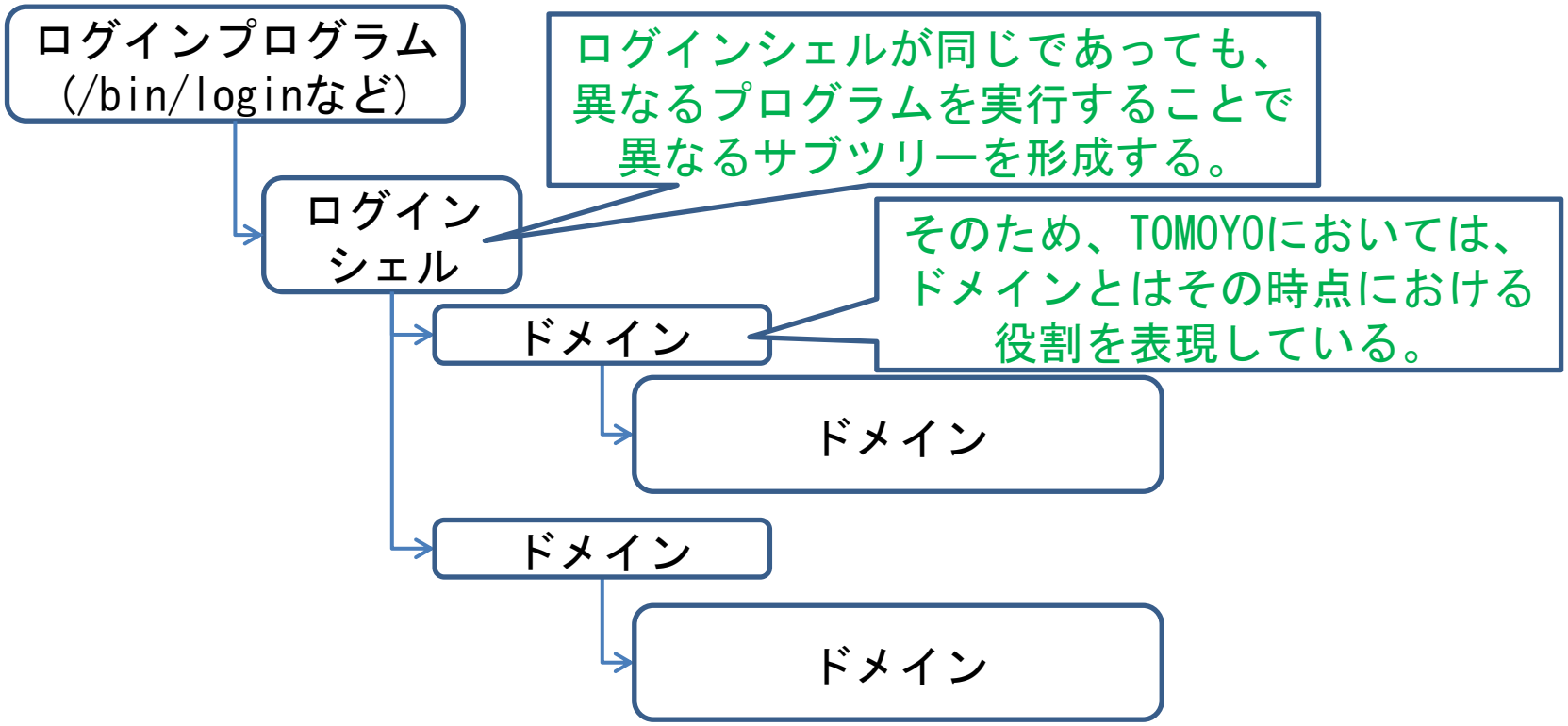


YUE

- 管理者業務をするにはroot権限が必要になる。
  - しかし、rootユーザは１つしか存在しない。
    - TOMOYOのツリー状の状態遷移を用いて権限分割を行ってみてはどうだろうか？
- コードネーム：Your User-role Enforcer
- YUEのトピック
  - Role Based Access Control(RBAC)風の権限分割技術

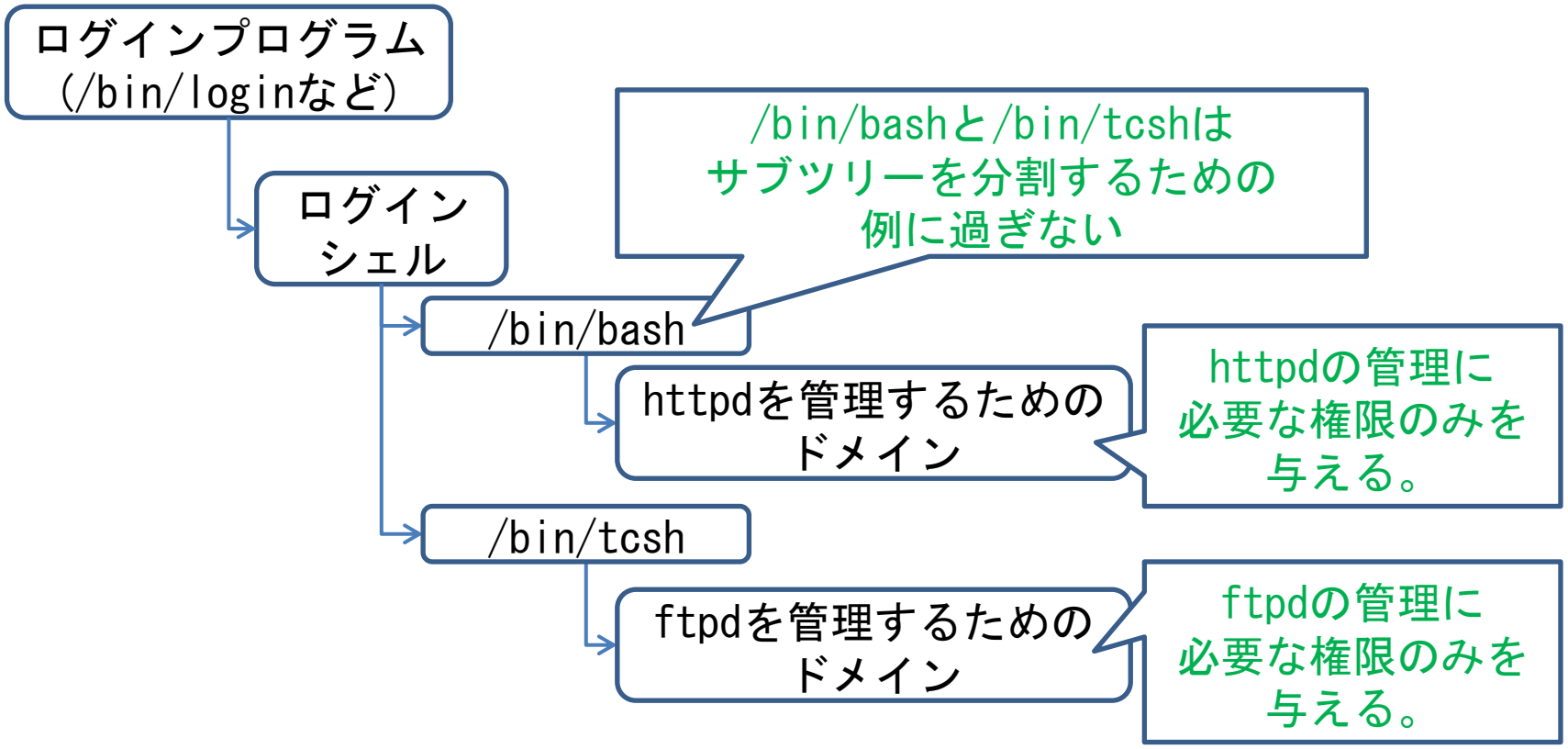
Role Based Access Control(RBAC)風の権限分割技術

- TOMOYOのツリー状の状態遷移を用いることで、管理者業務のための権限を任意のグループに分割できることに気が付いた。



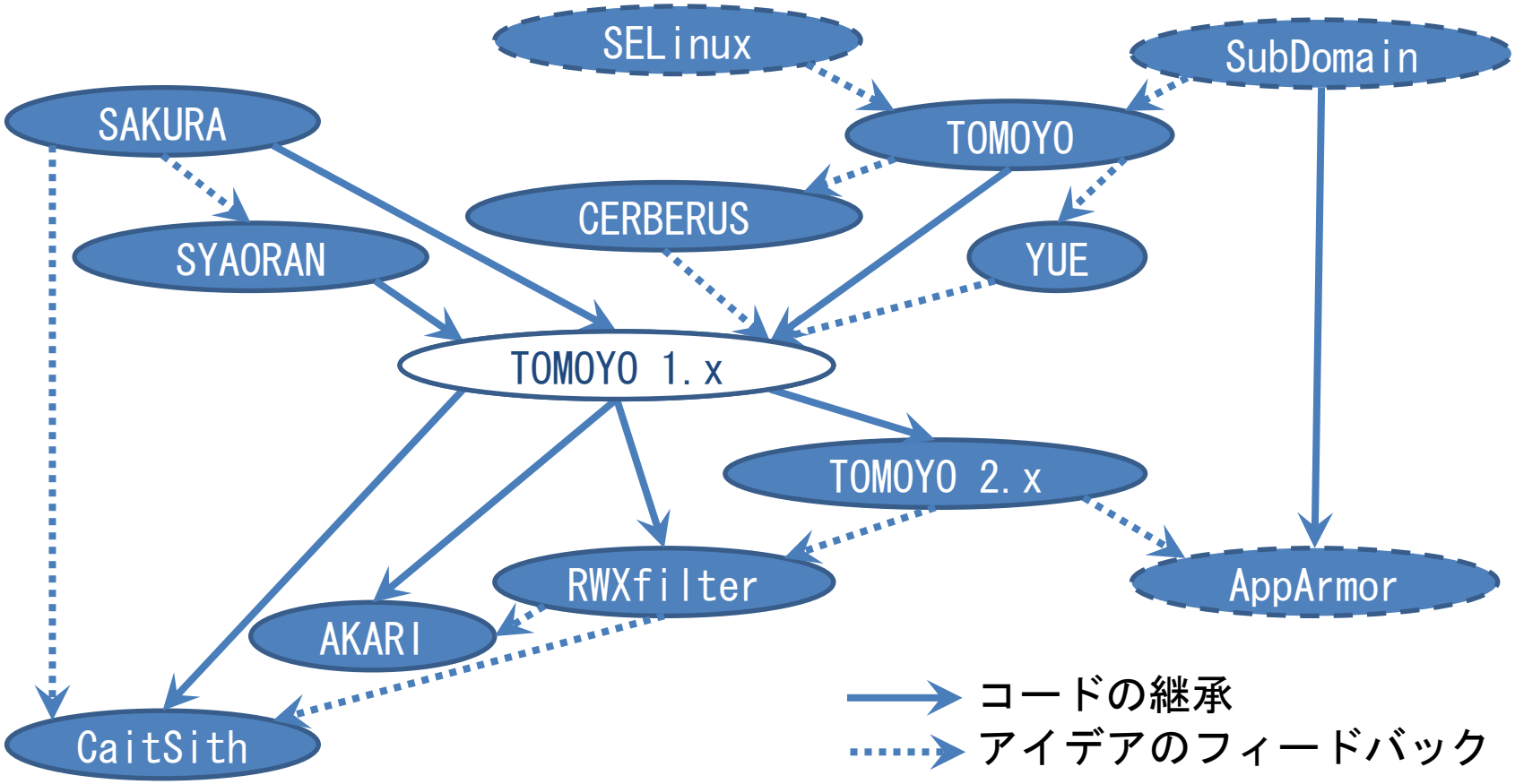
Role Based Access Control(RBAC)風の権限分割技術

- TOMOYOのツリー状の状態遷移を用いることで、管理者業務のための権限を任意のグループに分割できることに気が付いた。



TOMOYO 1.x (2005.11-)

- 私の様々な派生実装の源



TOMOYO 1.0 (2005.11-2006.3)

- 2003年4月からの成果をまとめてGPLのオープンソースとして公開されたバージョン
  - システム全体で名前空間の操作制限を担当するSAKURA
  - ドメイン単位でアクセス要求の制限を担当するTOMOYO
  - /devパーティションを保護するSYAORAN
  - ログインブルートフォース攻撃から保護するCERBERUS
  - 管理者業務の権限を分割するYUE

TOMOYO 1.1 (2006.4-2006.9)

- ユーザ空間のプログラムは名前に基づいて動作が変化する。
  - パス名が読み書き実行可能かどうかを制限するのに加えて、プログラムの振る舞いに影響を与える名前をより厳密に制限していく方向性が出された。
    - ディレクトリエントリを変更する操作(つまり mkdir, rmdir, create, unlink, mksock, mkfifo, mkchar, mkblock, link, symlink, rename, truncate)を書き込み操作(write)と区別する。
- ポリシー違反を対話的に処理できるようにした。
  - ソフトウェアアップデート(例えばyum/apt)の実行時などにしばしば起こる、予期せぬイベントに対応する機会をユーザに与える。

TOMOYO 1.2 (2006.9-2006.11)

- プログラム起動時の名前（別名argv[0]）もプログラム実行パーミッションのチェック時に一緒にチェックするようにした。
  - (例えばbusyboxのような)マルチコールバイナリは、プログラム起動時の名前によって振る舞いが変化するから。
- カレントプロセスのユーザIDやファイル所有者IDなども一緒にチェックするようにした。
  - パス名をチェックするだけでは不十分。

TOMOYO 1.3 (2006.11-2006.3)

- 0～255の整数値をとるプロファイル番号を導入することにより、ドメイン単位でアクセス制御の有効／無効を指定できるようにした。
  - プロファイル番号により、TOMOYOをSELinuxのtargeted policyのように使えるようになった。
  - プロファイル番号により、プログラムごとに制限する操作の範囲を指定できるようになった。
- 必要に応じてドメイン遷移を抑制／初期化することができるようにした。
  - さまざまなパターンのドメイン遷移を使えるようにした。

TOMOYO 1.4 (2007.4-2007.9)

- パス名の除外演算子をサポートした。
  - 一般に、ドットで始まるファイル名は特別扱いされるべき。
  - /var/www/html/¥*¥-.¥*
    - 例えば、/var/www/html/.htaccessと/var/www/html/index.htmlとを区別する。
- x86_64アーキテクチャに対応した。
- TOMOYO 2.xというLSM対応版の開発を開始し、標準のLinuxカーネルに採用されるための挑戦を開始した。
  - Ottawa Linux Symposium 2007でTOMOYO Linuxプロジェクトは BoFセッションを開催した。

TOMOYO 1.5 (2007.9-2008.3)

- TOMOYOとAppArmorとの差別化を図るために、使い勝手を向上させた。
  - 当時、TOMOYOは標準のLinuxカーネルに採用されるための挑戦をしていた。しかし、TOMOYOもAppArmorもポリシーファイルの中でパス名を使用していたため、「両方をLinux標準のカーネルに採用する必要はない」と考えられていた。
  - TOMOYOがどれだけ丁寧に設計されているかを示そうとした。
- TOMOYO 1.xをSELinuxと同時に有効にできるようにした。
  - ラベルに基づくMACと名前に基づくMACは相補的な役割を果たすものである。
    - それゆえ、両方を同時に有効にできるべきである。

TOMOYO 1.6 (2008.4-)

- 様々な機能強化／使い勝手向上
  - プログラム実行時にコマンドライン引数や環境変数などもチェックできるようにした
  - プログラム実行要求を横取りしてコマンドライン引数や環境変数などの検査や無害化を行えるexecute handler機能をサポートした。
    - この機能を使うと、不審なプログラムの実行要求(例えば適切なコマンドライン引数や環境変数が指定されていない/bin/shの実行要求)をしたプロセスを静かに終了させたりすることも可能。
  - 状態変数を含むaclをサポートするようにした。
  - その他いろいろ
- これはRWXfilterやTOMOYO 2.2のベースとなっている。

TOMOYO 1.7 (2009.9-)

- 現在の安定版
- パス名と一緒に渡された様々な属性も一緒にチェックする。
- TOMOYO 2.2が標準のLinuxカーネルに採用されたことを受け、TOMOYO 1.xのモジュール名をCCSecurityに変更した。
  - これを機に、役割分担を見直した。
    - システム全体のアクセス制御(SAKURA)をドメイン単位のアクセス制御(TOMOYO)に統合した。
    - /devファイルシステムによるアクセス制限(SYAORAN)をドメイン単位のアクセス制御(TOMOYO)に統合した。

システム全体でのアクセス制御vs.
ドメイン単位でのアクセス制御

- なぜシステム全体でのアクセス制御とドメイン単位でのアクセス制御とを別々に扱ってきたのだろう？
  - 主にコードネームに対する愛着が理由。
- TOMOYO 1.xは全てのプロセスに対してアクセス制限を適用できるのだから、より正確に制限するために、システム全体でのアクセス制御よりもドメイン単位でのアクセス制御にする方が良いのでは？
  - (当時は)その通りであると考えた。
- この判断の背後には、LXC仮想化(pivot_root)ユーザに対応するために、より細粒度での制限を行うことを目指していた。
  - TOMOYO 2.2ではTOMOYO(ドメイン単位でのアクセス制限)だけが含まれていたため、TOMOYO 1.7ではSAKURA(システム全体でのアクセス制限)は削除することにした。

ファイルシステムでのアクセス制御vs.
ドメイン単位でのアクセス制御

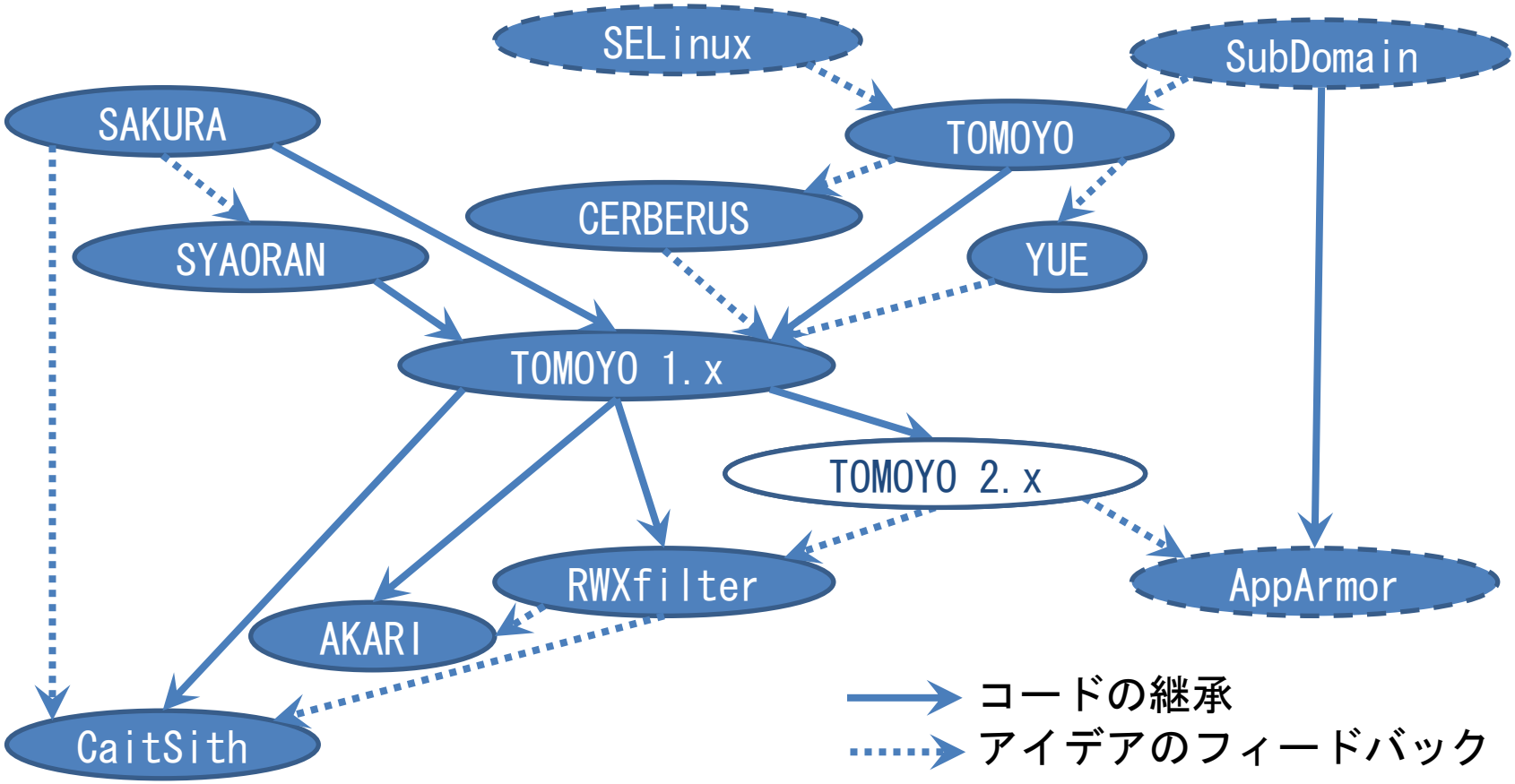
- なぜファイルシステム層で制限する必要があるのだろう？
今となっては、TOMOYOはファイル名だけでなくファイルの属性もチェックできる。今後も/dev用のファイルシステムをメンテナンスし続ける必要はあるのだろうか？
  - (当時は)その必要はないと考えた。
    - SYAORANファイルシステムはudev方式と競合するため、TOMOYO 1.7ではSYAORANファイルシステムを削除した。

TOMOYO 1.8 (2010.11-)

- Linux 2.6.27～3.5カーネルに対応している現在の最新版。
- 内部構造を見直し、冗長／もはや使われなくなった機能を削除し、ポリシー構文のキーワードを変更した。
- (タスク構造体に独自フィールドを追加することで生じるカーネルABIの変化を避けるために、)fork()とexit()の処理にフックを挿入することでカーネルABIを維持できるようにした。
  - これにより、ディストリビュータカーネルと同様に使えるようになった。
- これはAKARIやCaitSithのベースとなっている。

TOMOYO 2.x (2007.6-)

- Linux標準カーネルに採用されたバージョンのTOMOYO 1.x



TOMOYO 2.2 (2009.6-2010.10)

- 標準のLinux 2.6.30カーネルで採用されたバージョン。
- TOMOYO 1.6のコア機能だけが実装されている。
- ファイルに関する操作を制限する上で不足していた
  LSMフックの追加はLinux 2.6.33カーネルで完了。

TOMOYO 2.3 (2010.10-2011.10)

- Linux 2.6.36～3.0カーネルに取り込まれたバージョン。
- TOMOYO 1.7のファイルに関する主要な機能が実装されている。

TOMOYO 2.4 (2011.10-2012.1)

- Linux 3.1カーネルに取り込まれたバージョン。
- 実用に耐えうる機能を備えたバージョン。
- TOMOYO 1.8のファイルに関する主要な機能が実装されている。

TOMOYO 2.5 (2012.1-)

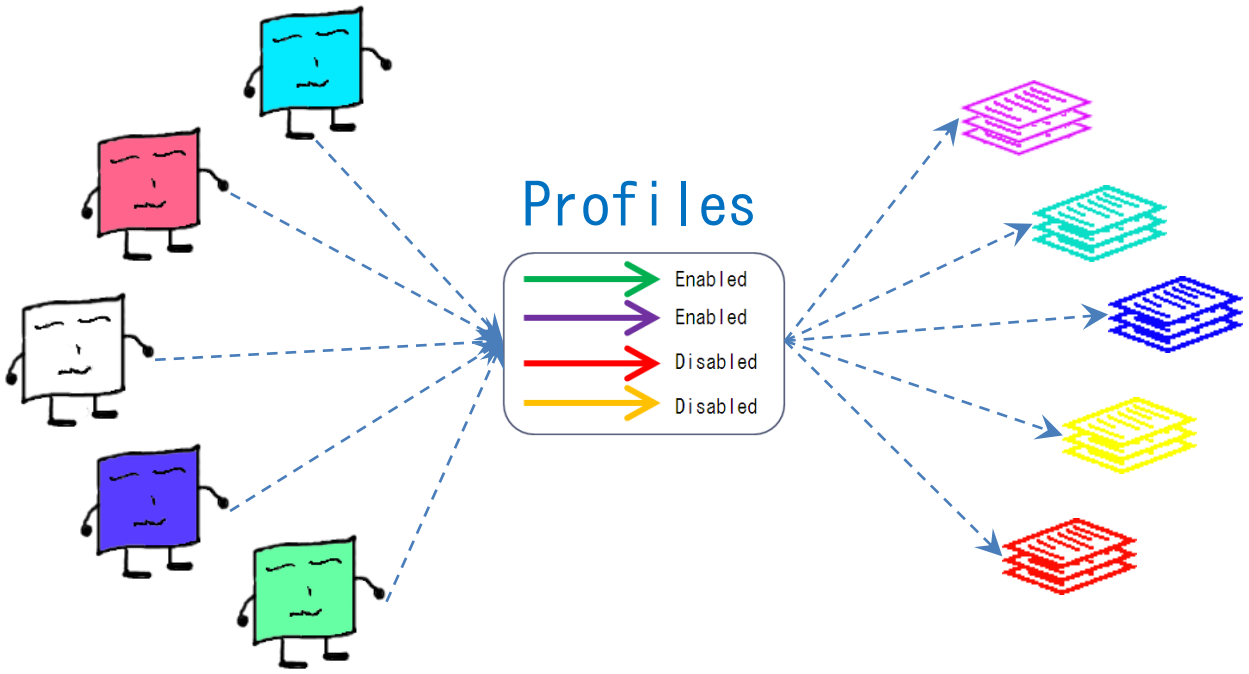
- Linux 3.2以降のカーネルに取り込まれているバージョン。
  - Linux 2.6.33カーネル以降に対してはsecurity/tomoyo/ディレクトリ内の修正のみでバックポート可能。
- TOMOYO 1.8の主要機能を実装。
- まだ実現できていない機能
  - execute handler
  - ネットワークの受信方向のパーミッションチェック
  - ケイパビリティ(seccomp mode 2で代用可能？)
  - バイナリローダのパーミッションチェック
  - 他のLSMモジュールと同時に有効にする

実現できたこと

- ユーザ空間への副作用を考慮したカーネル内アクセス制御
  - ファイルの読み書き実行の可否が変化しないだけでは不十分
  - 文字列や数値で表現可能な様々なパラメータをチェックするファイアウォール
- 起動からシャットダウンまでに起こる全ての振る舞いを知る
  - 全てのプロセスに対して適用できる
    - タスク構造体をドメイン定義に使用する
  - 全てのプロセスに対して適用できることで得られる安心感
    - TOMOYO 1.8はAndroid端末で使われている
  - 望まない振る舞いを防ぐことに重点が置かれている。

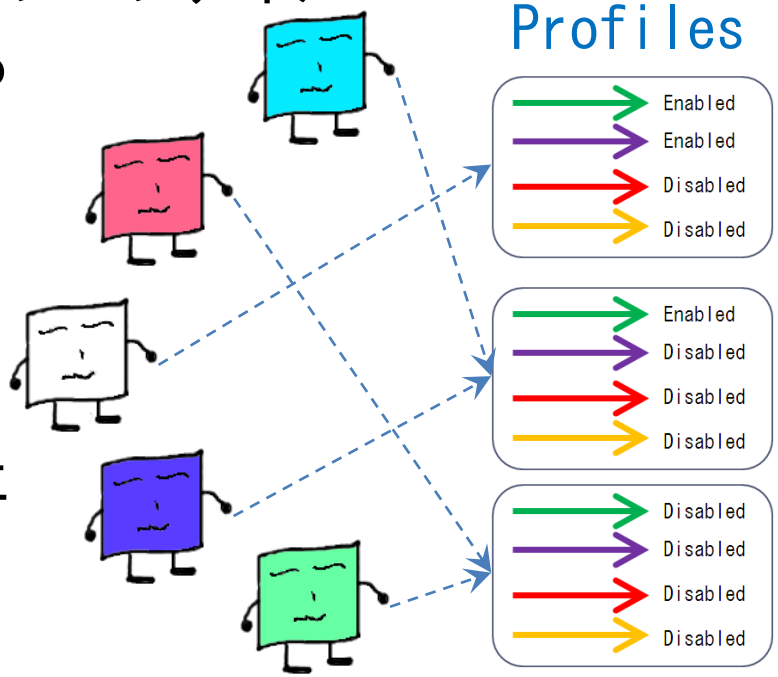
苦労したこと

- 有用だと考えられる機能を追加しつつ、心理的な障壁は最小限に維持すること。
- 第１段階（2005.11）
  - 操作単位で有効／無効を指定できるようにした。



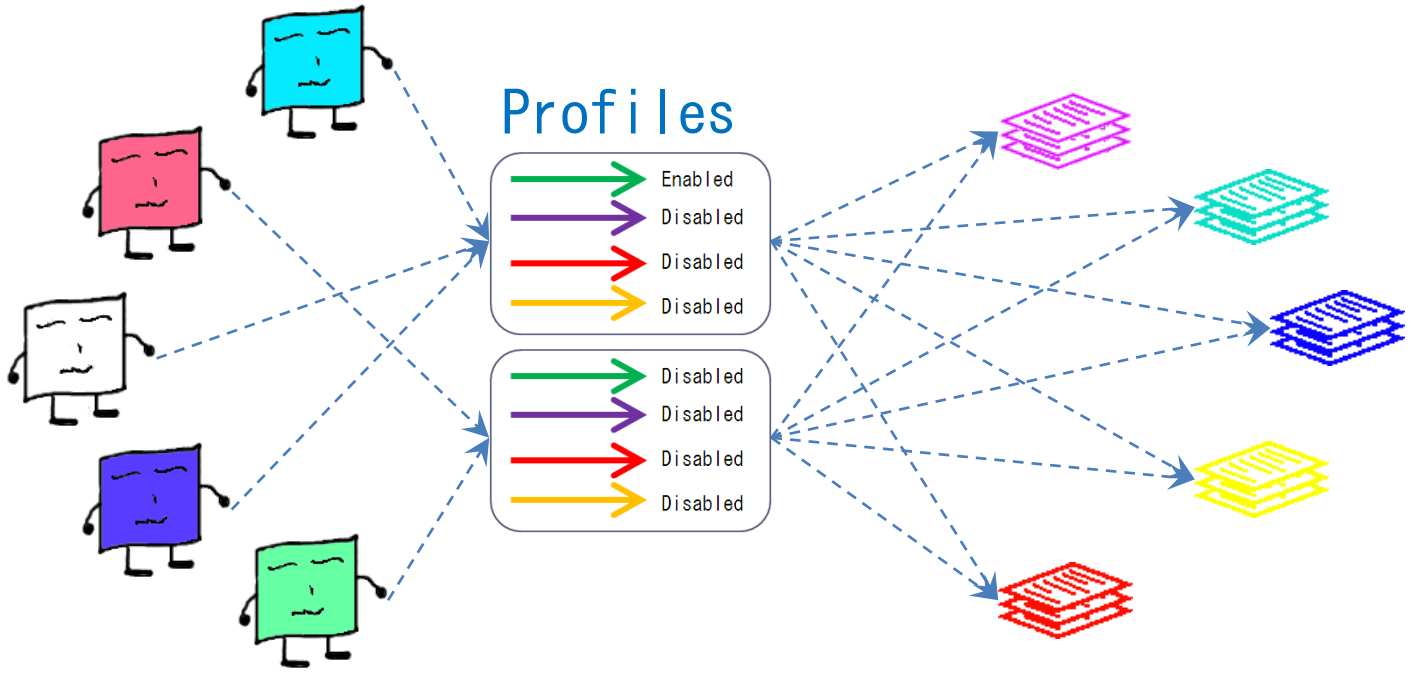
苦労したこと

- 第２段階(2006.11)
  - TOMOYOは全てのプロセスに対して適用できるけれども、利用者のスキルが追い付かない。
  - ドメイン単位で有効／無効を指定できるようにした。
    - ポリシー作成後にプロファイル番号を切り替えることにより、積み上げ型のポリシー作成アプローチが利用できるようになった。



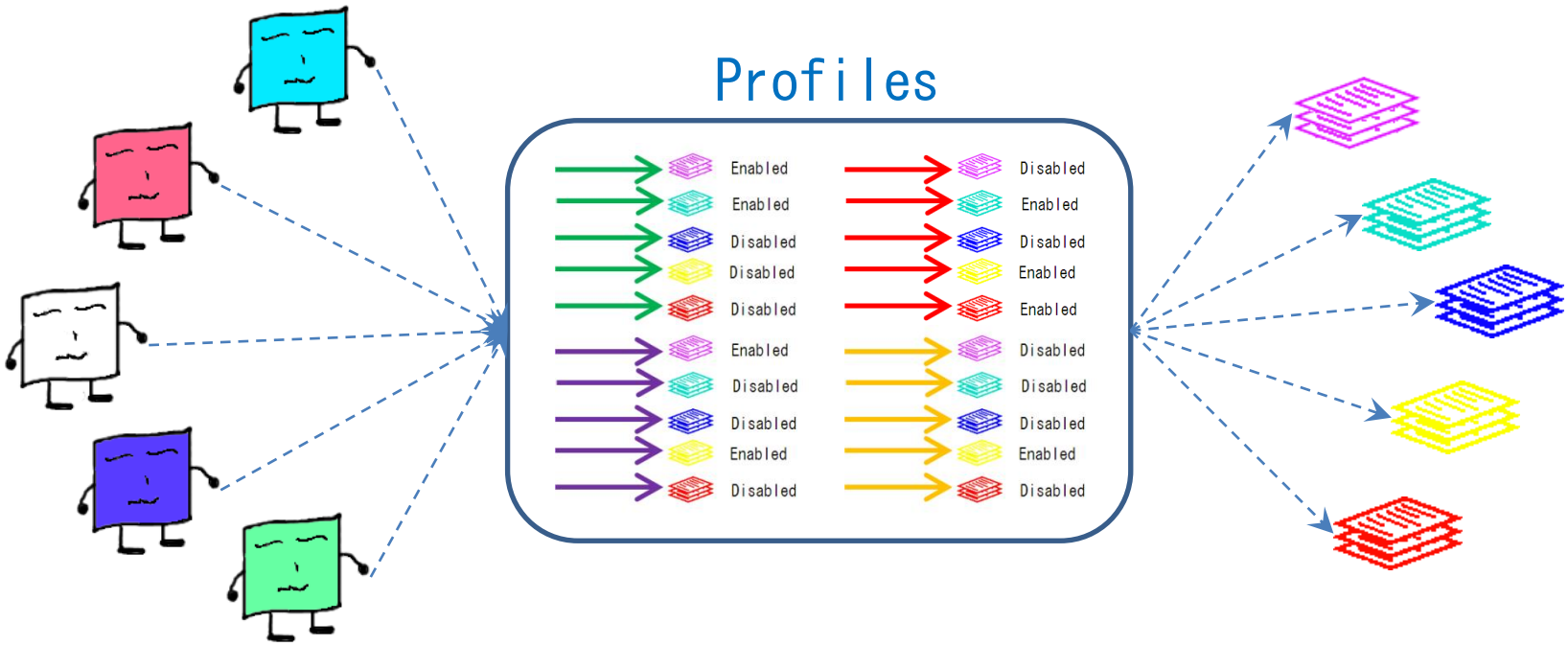
苦労したこと

- 第３段階(2011)
  - ファイルに対する操作のみを制限する場合であっても、一度に全てのファイルを強制モードに切り替えることは難しい。



苦労したこと

- 第３段階(2011)
  - ファイル名単位のプロファイルを検討した。
    - しかし、複雑になりすぎると考えて実装しなかった。



苦労したこと

- 第３段階(2011)
  - ブラックリスト方式を検討した。
    - しかし、プロファイルのアクセス制御モードと競合するため実装しなかった。
      - 確認モードとはアクセス要求をチェックするが拒否しないモードであるため、ブラックリストが追加されると、もはや確認モードではなくなってしまう。
    - ドメインを定義する前にブラックリストを定義できないという問題もある。
      - TOMOYOがドメインを自動的に定義してしまったときに、ブラックリストをどのように扱えば良いのだろう？

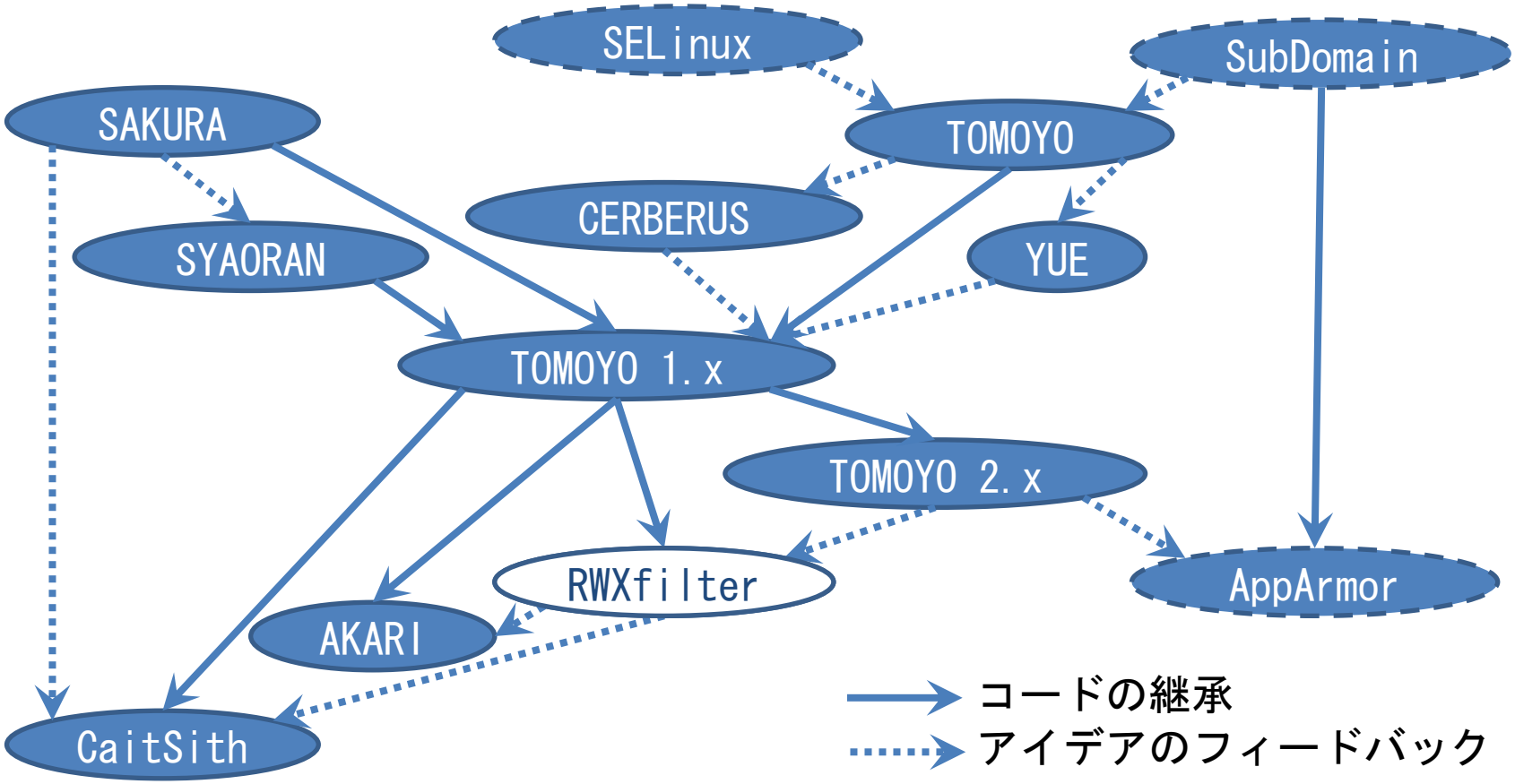
=> CaitSithへと続く

第３章

使い勝手優先路線で経験したこと

RWXfilter (2010.2-2010.4)

- Linux利用者の本音に耳を傾けることになったきっかけ



RHEL利用者からの要望

- SELinuxは難しすぎて使えない。RHELカーネルに
  ローダブルカーネルモジュールとして追加できる単機能な
  アクセス制御機構を開発してほしい。
  - LSMインタフェースを使うローダブルカーネルモジュール
    として実装することに。
- 特定のファイルに対してだけアクセス制御を適用できる
  ようにしてほしい。
  - ユーザに対する障壁を最小化するために、
    read/write/executeだけをチェック対象として
    実装することに。
    - コードネーム：Read/Write/Execute filter略して
      RWXfilter

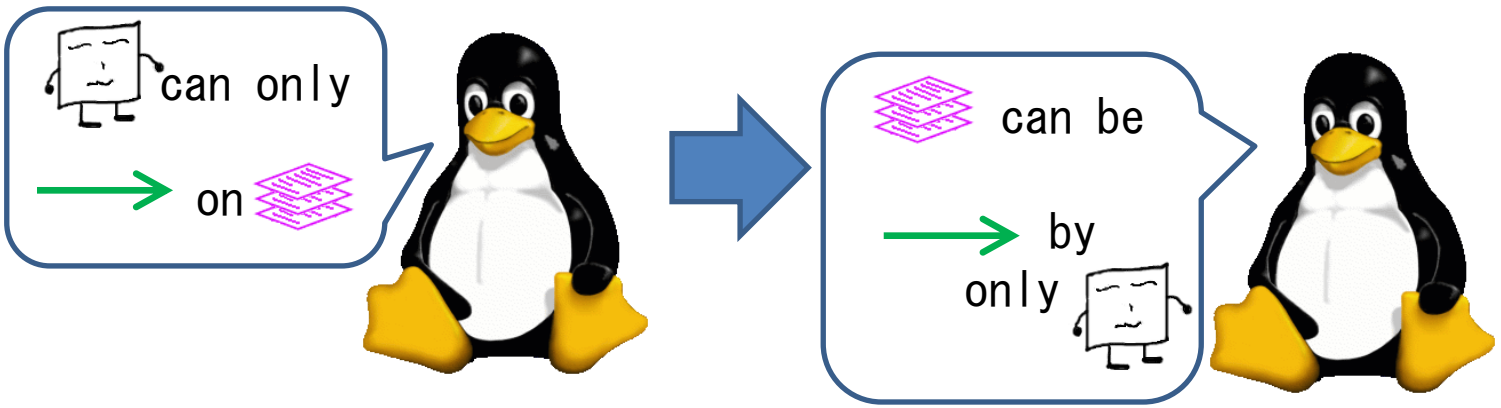
ジレンマ

- 既存のMACはまずドメインを定義し、それからドメインに対してパーミッションや資源を関連付けることが前提になっている。
  - しかし、全てのプロセスに対して行うことは困難である。
- しかし、アクセス制御が適用されないプロセスが存在してしまってはMACの価値を損ねてしまう。
  - しかし、特定の資源に対してアクセスを制限するためだけに、利用者に対して全てのプロセスを管理するという負担を押し付けるわけにもいかない。

視点を逆転した

- 「まずドメインを定義し、ドメインに対してパーミッションや資源を関連付ける」から「まず資源を定義し、資源に対してパーミッションやドメインを関連付ける」に切り替えた。

Capabilityモデル　　Access control listモデル



RWXfilterのポリシー構文

- "資源" "アクセス制御モード"
  - "動作1" by "動作1が許可されるドメイン1"
  - "動作2" by "動作2が許可されるドメイン2"
  - "動作3" by "動作3が許可されるドメイン3"
  - "資源"はTOMOYOのパス名表記
  - "アクセス制御モード"はpermissiveまたはenforcing
  - "動作X"はreadまたはwriteまたはexecute
  - "動作Xが許可されるドメインX"はTOMOYOのドメイン名表記

RWXfilterのポリシー例

- /etc/shadow enforcing
  read by <kernel> /sbin/init /sbin/mingetty /bin/login
  read by <kernel> /usr/sbin/sshd
  - この例では、/etc/shadowを読み込みモードでオープンするという動作を<kernel> /sbin/init /sbin/mingetty /bin/loginドメインまたは<kernel> /usr/sbin/sshdドメインに属しているプロセスのみに許可する。
  - この例では、アクセス制御モードがenforcingであるため、/etc/shadowを書き込みモードでオープンするという動作は拒否される。

結末

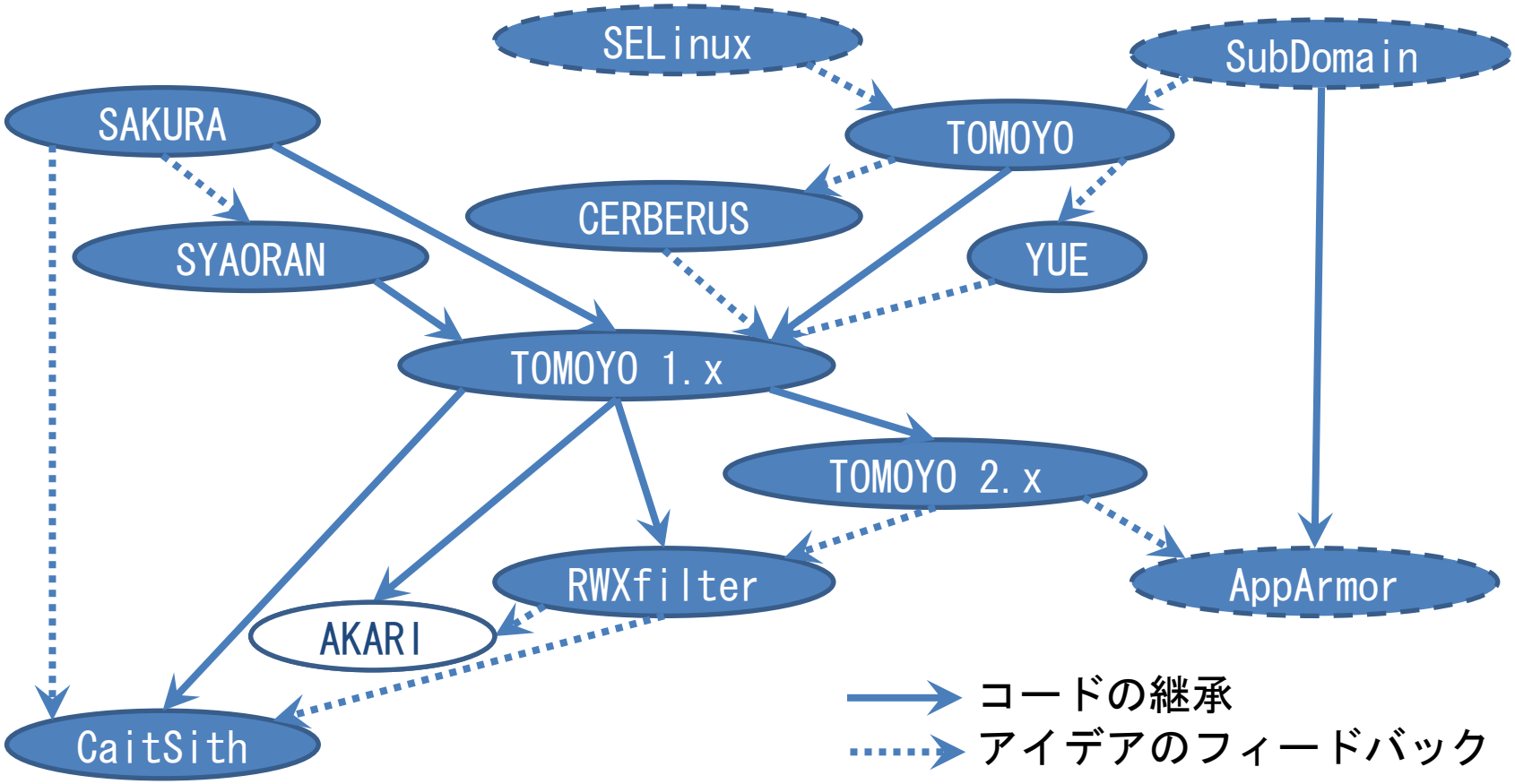
- 商用サポート体制を確立できなかったため、RWXfilterはお蔵入り。
  - しかし、RWXfilterは異なる視点からアクセス制御を行うことの可能性について探求するきっかけとなった。

  => CaitSithへと続く
- SELinuxを無効化せずに他のLSMモジュールをローダブルカーネルモジュールとして追加する手法を確立した。
  - TOMOYO 2.xの上でYamaも追加するというデモをLinuxCon North America 2010(Boston)と同時開催されたLinux Security Summitで行った。

  => AKARIへと続く

AKARI (2010.10-)

- TOMOYO 1.8のローダブルカーネルモジュール版



始まり

- TOMOYO 2.xがFedoraやRHELのカーネルで使えない状態が続く。
  - https://bugzilla.redhat.com/show_bug.cgi?id=542986
- カーネルパッケージの差し替えは大きな心理的障壁となっている。
  - どうにかしてもっと簡単にTOMOYOを試せないだろうか？

どう考えたか？

- LSMフックが幾つかの処理に対して提供されていないために、ローダブルカーネルモジュールでは実装できない機能がある。
  - TOMOYOはアクセス解析ツールとして始まった。
    - 解析目的でなら、幾つか実装できない機能があっても容認できるのではないか？
      - 割り切ることにしよう。

=> RWXfilterで確立した手法をTOMOYO 1.8に適用した。

結末

- FedoraやRHELのカーネルでも、TOMOYO 1.8の主要な機能が使えるようになった。
- AKARIはRWXfilterと同様のローダブルカーネルモジュールなので、解析用途では特に便利。
- http://akari.sourceforge.jp/

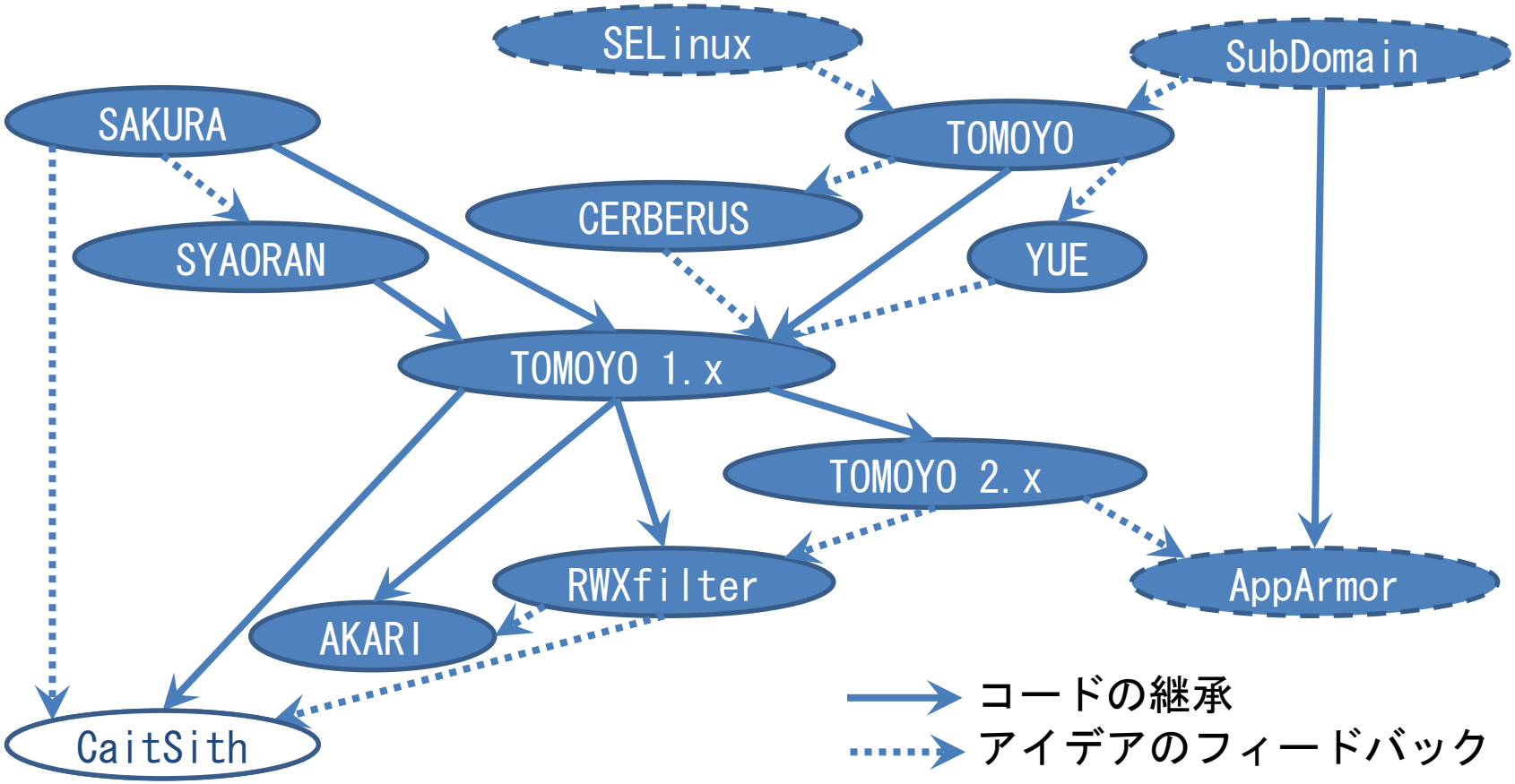
Access
Keeping
And
Regulating
Instrument.

第４章

CaitSith

CaitSith (2012.4-)

- 私の９年間の経験から導き出された、最も強力で柔軟な設定が可能なポリシー構文。



CaitSithの始まり

- 脅威はユーザ空間での振る舞いにある。
  - しかし、脅威はカーネル内アクセス制御が扱えない領域へとシフトしてきている。
    - 例えば、ユーザ空間のプログラムのバグにより、メールやツィートが誤って開示されたり予期せぬ相手と共有されたりする。
    - これは、カーネル内アクセス制御の限界でもある。
- seccomp mode 2がLinux 3.5カーネルで利用可能になった。

=> もう、カーネル内で無茶をする必要は無いのかもしれない。

MACのあるべき姿とは？

- アクセス制御機能強化路線で経験したこと：
  - ドメイン単位のアクセス制御は、全てのドメインに対してアクセス制御が適用されている限り、システム全体でのアクセス制御よりも正確に制限できる。
    => もし、アクセス制御が適用されないドメインが存在した場合、システム全体でのアクセス制御であれば発生しなかった抜け道となってしまう。
  - 現実には、アクセス制御が適用されないドメインが存在していることが普通である。
    => しかし、システム全体でのアクセス制御だけでも不十分である。

MACのあるべき姿とは？

- アクセス制御機能強化路線で経験したこと：
  - ホワイトリストではなくブラックリストでアクセスを制限したいユーザもいる。
    => ドメインを自動的に生成してしまうTOMOYOにおいては、ブラックリスト方式を実現することは困難。

MACのあるべき姿とは？

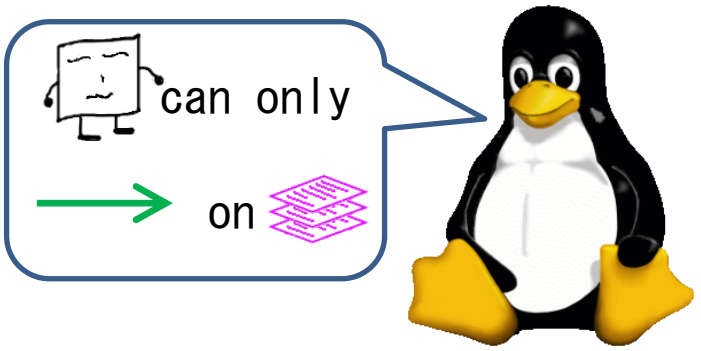
- 使い勝手優先路線で経験したこと：
  - プロセスの視点からではなく資源の視点からアクセスを制限したいユーザもいる。
    => ドメインありきのアクセス制御ではこの要望に対応できない。
  - 特定の資源に対してだけアクセスを制限したいユーザもいる。
    => ホワイトリスト方式のアクセス制御ではこの要望に対応できない。

MACのあるべき姿とは？

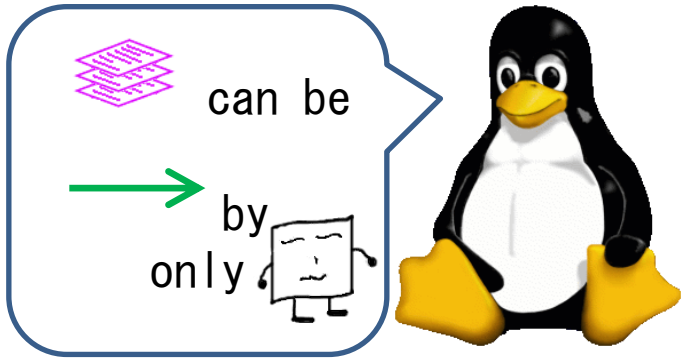
- 私の結論：
  - ドメイン(Domain Type Enforcement)依存やホワイトリスト依存から脱却する時が来た。
  - 最初から考え直してみよう。

どうすれば両方のアプローチの いいとこどりができる？

- TOMOYOは**主体**の視点から作られ、機能を強化することに重点が置かれている。

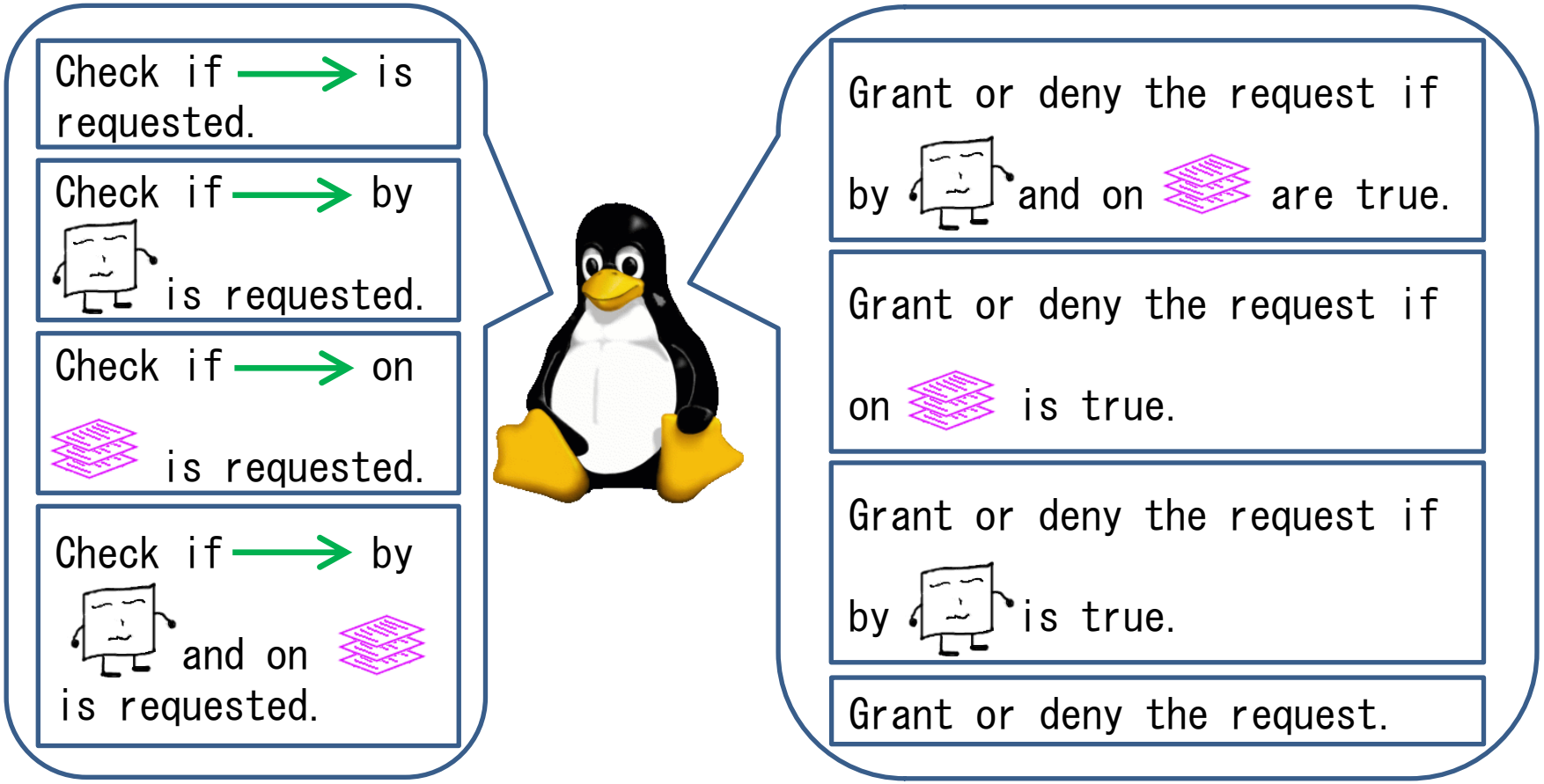


- RWXfilterは**客体**の視点から作られ、使い勝手を向上することに重点が置かれている。



動作をキーにしてみたら？

- Capabilityモデル + Access control listモデル
  => Action check listモデル



提案する構文

- acl "動作" "動作をチェックするかどうか"
  audit "アクセスログ取得方法"
  "判断1" "判断1を使う条件"
  "判断2" "判断2を使う条件"
  "判断3" "判断3を使う条件"
- 動作をキーとして指定し、必要に応じて条件を列挙する。
  - 必須パラメータ(位置パラメータ)を使わない。
  - (ドメイン名も含めて)全てのパラメータは任意指定。
- 「チェックするかどうか」と「許可／拒否するかどうか」という２段階に分けて扱う。
  - 有効／無効を指定する必要が無いので、プロファイルも使わない。

提案する構文の特徴

- ホワイトリストとブラックリストの両方をサポートしている。
- 動作をキーとして、主体の視点からの設定と客体の視点からの設定の両方をサポートしている。
- TOMOYOのパラメータチェック機能をフル活用できる。
- RWXfilterのように単機能な制限を行うこともできる。
- 動作をキーとしているため、システム全体に対する制限が容易に行える。

=> 上記の項目は後ほど例を示しながら解説する。

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
quota audit[1] allowed=0 denied=1024 unmatched=1024

0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
quota audit[1] allowed=0 denied=1024 unmatched=1024
```

クォータとグループを
定義するヘッダ部分

```
0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

ルールを定義する
ボディ部分

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
quota audit[1] allowed=0 denied=     unmatched=1024

0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

ポリシー構文のバージョン

最大で16MBのカーネルメモリを
アクセスログを保持するために
割り当てる。

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
quota audit[1] allowed=0 denied=1024 unmatched=1024

0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/
    100 allow task.exe="/
    10000 deny
```

audit[1]行の指定に従ってアクセスログを生成する。

allow行に一致した場合には最大0個、deny行に一致した場合は最大1024個、allow行にもdeny行にも一致しなかった場合は最大1024個のアクセスログを生成する。

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
                                                    
quot                 ｴﾘ」 allowed=0 denied=1024 unmatched=1024

0 acl modify_policy
      audit 1
      1 den     ask.uid!=0
      1 d       ask.euid!=
      1         w task.exe
                                                    n/caitsith-queryd"
```

チェック優先順位は、同じ動作のaclブロックが複数ある場合のチェック順序を制御する。

チェックを行うかどうかを判断する条件。省略すると無条件にチェックが行われる。

チェックする動作の名前。この例ではポリシーを変更するという動作。

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
c                                                    nmatched=1024
0 acl r          _policy
     au    e 1
     1 deny task.uid!=0
     1 deny task.euid!=0
     100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
     100 allow task.exe="/usr/sbin/caitsith-queryd"
     10000 deny
```

判断の優先順位は、当該aclブロック内の
複数の判断行がある場合の適用順序を制御する。

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
quota audit[1] allowed=0 denied=1024 unmatched=1024

0 acl modify
    audit
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

判断はallowまたはdenyのどちらか。

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
quota audit[1] allowed=0 denied=1024 unmatched=1024

0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

判断を適用するかどうかを決定する条件。
省略すると無条件に判断が適用される。

ポリシーファイルはどんな風に見える？

```
POLICY_VERSION=20120401
quota memory audit 16777
quota memory query 10485
quota audit[1] allowed=0

0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

このaclブロックは以下のようにルールを定義している。

(1) カレントスレッドのユーザIDまたは実効ユーザIDが0ではない場合にはポリシー設定の変更を拒否する。

(2) /proc/self/exeが/usr/sbin/caitsith-loadpolicyまたは/usr/sbin/caitsith-querydの場合は、ポリシー設定の変更を許可する。

(3) それ以外の場合は、ポリシー設定の変更を拒否する。

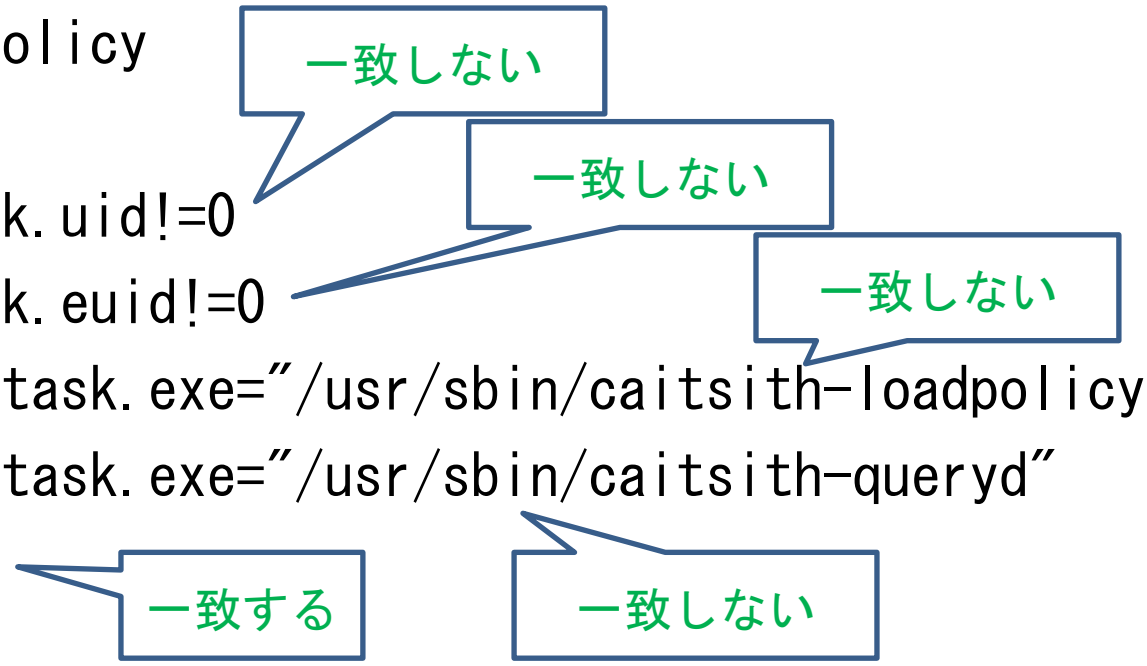
アクセスログはどんな風に見える？

- bashからポリシー設定を変更しようとしても拒否される。

```
# echo '1000 acl modify_policy' > /proc/caitsith/policy
-bash: echo: write error: Operation not permitted
```

- そして、アクセス拒否ログが生成される。

```
0 acl modify_policy
      audit 1
      1 deny task.uid!=0
      1 deny task.euid!=0
      100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
      100 allow task.exe="/usr/sbin/caitsith-queryd"
      10000 deny
```



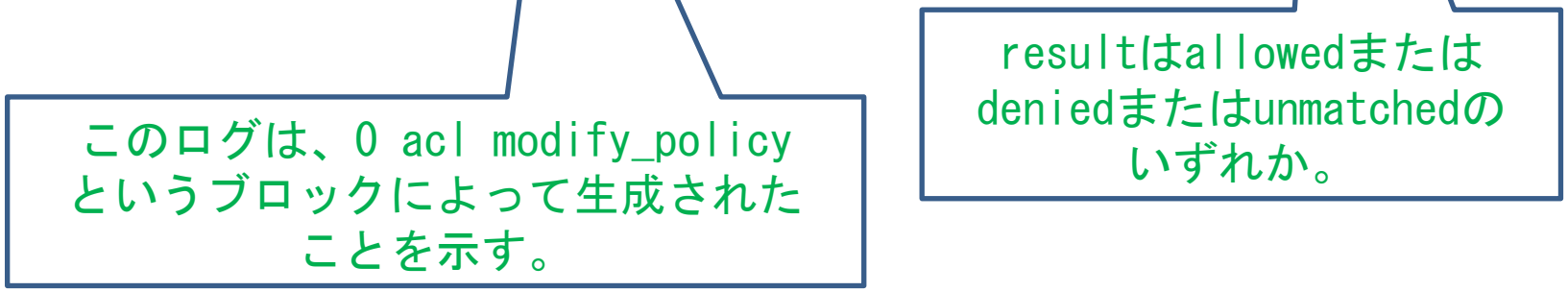
アクセスログはどんな風に見える？

- 以下は、bashからポリシーを変更しようとして
  拒否されたときのログである。
  - #2012/07/11 14:06:21# global-pid=3584 result=denied priority=0 / modify_policy task.pid=3584 task.ppid=3582 task.uid=0 task.gid=0 task.euid=0 task.egid=0 task.suid=0 task.sgid=0 task.fsuid=0 task.fsgid=0 task.type!=execute_handler task.exe="/bin/bash" task.domain="/usr/sbin/sshd"

このログは/proc/caitsith/auditから読み出すことができ、
caitsith-auditdプログラムによって保存される。

アクセスログはどんな風に見える？

- 以下は、bashからポリシーを変更しようとして拒否されたときのログである。
  - #2012/07/11 14:06:21# global-pid=3584 result=denied priority=0 / modify_policy task.pid=3584 task.ppid=3582 task.uid=0 task.gid=0 task.eu d=0 task.egid=0 task. uid=0 task.sgid=0 task.fsu =0 task.fsgid=0 tas ype!=execute_handler task.exe="/bin/b task.domain="/usr/sbin/ "



アクセスログはどんな風に見える？

- 以下は、bashからポリシーを変更しようとして拒否されたときのログである。
  - #2012/07/11 14:06:21# global-pid=3584 result=denied priority=0 / modify_policy task.pid=3584 task.ppid=3582 task.uid=0 task.gid=0 task.euid=0 task.egid=0 task.suid=0 task.sgid=0 task.fsuid=0 task.fsgid=0 task.type!=execute_handler task.exe="/bin/bash" task.domain="/usr/sbin/sshd"

これらの変数はアクセス要求内に含まれている。
必要に応じてこれらの変数を条件として指定することができる。

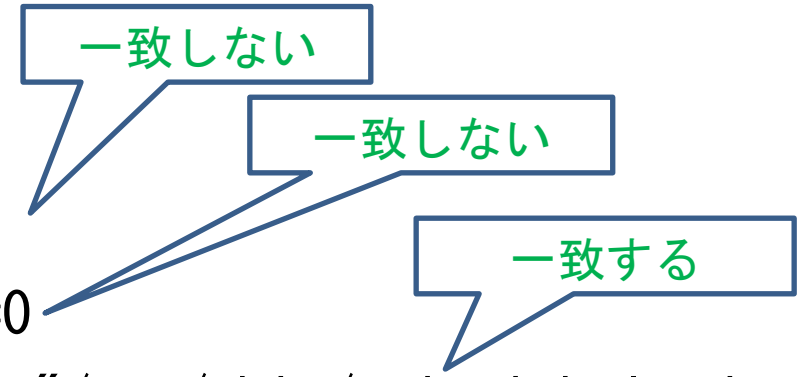
ポリシーのアップデート方法は？

- caitsith-loadpolicy を用いたポリシー設定の変更は許可される。

```
# (echo '0 acl modify_policy'; echo '1 deny
  task.gid!=0') | /usr/sbin/caitsith-loadpolicy
```

- ただし、アクセス許可ログは生成されない。

```
0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```



ユーザ空間のデーモンプロセスは？

```
POLICY_VERSION=20120401
quota memory audit 16777216
quota memory query 1048576
quota audit[1] allowed=0 denied=1024 unmatched=1024

0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=0
    100 allow task.exe="/usr/sbin/caitsith-loadpolicy"
    100 allow task.exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

ccs-querydプログラムが動作している場合に、対話的な判断を行うためにアクセス要求を保持するためのカーネルメモリとして最大1MBを割り当てる。

ユーザ空間のデーモンプロセスは？

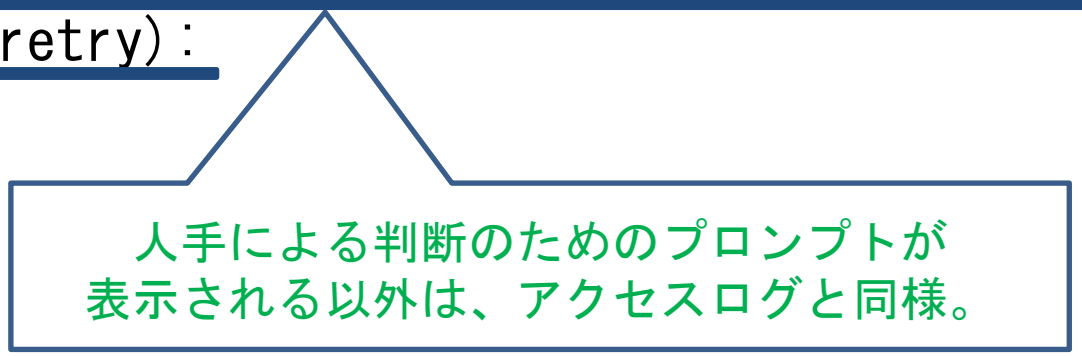
```
POLICY_VERSION=20120401
quota memory audit 1
quota memory query 1
quota audit[1] allow

0 acl modify_policy
    audit 1
    1 deny task.uid!=0
    1 deny task.euid!=
    100 allow task.e      /usr/sbin/caitsith-loadpolicy"
    100 allow task.      exe="/usr/sbin/caitsith-queryd"
    10000 deny
```

アクセス拒否ログが生成され、その後
対話的な判断のためにスプールされる。もし、
caitsith-querydが許可した場合、あたかも
deny行に一致しなかったかのように
パーミッションチェックが続行される。
それ以外は、アクセス要求は拒否される。

ユーザ空間のデーモンプロセスは？

- 以下はcaitsith-querydプログラムが表示しているクエリーの例である。
  - #2012/07/11 14:06:21# global-pid=3584 result=denied priority=0 / modify_policy task.pid=3584 task.ppid=3582 task.uid=0 task.gid=0 task.euid=0 task.egid=0 task.suid=0 task.sgid=0 task.fsuid=0 task.fsgid=0 task.type!=execute_handler task.exe="/bin/bash" task.domain="/usr/sbin/sshd" Allow? ('Y'es/'N'o/'R'etry/'S'how policy/'A'dd to policy and retry):



提案する構文の特徴

- ホワイトリストとブラックリストの両方をサポートしている。

```
1000 acl execute task.exe="/usr/sbin/httpd"
     audit 1
     100 allow path="/var/www/cgi-bin/counter.cgi"
     200 deny
2000 acl execute task.exe="/usr/sbin/httpd"
     audit 1
     100 deny path="/bin/sh"
     200 allow
```

ホワイトリストの場合、無条件deny行で完結する。

ブラックリストの場合、無条件allow行で完結する。

提案する構文の特徴

- 動作をキーとして、主体の視点からの設定と客体の視点からの設定の両方をサポートしている。

```
1000 acl execute task.exe="/usr/sbin/httpd"
     audit 1
     100 allow path="/usr/sbin/suexec"
     200 deny
```

/usr/sbin/httpdは
/usr/sbin/suexecだけを実行できる。

```
2000 acl execute path="/usr/sbin/suexec"
     audit 1
     100 allow task.exe="/usr/sbin/httpd"
     200 deny
```

/usr/sbin/suexecは
/usr/sbin/httpdだけが実行できる。

提案する構文の特徴

- 動作をキーとして、主体の視点からの設定と客体の視点からの設定の両方をサポートしている。

```
1000 acl inet_stream_listen task.exe="/usr/sbin/sshd"
     audit 1
     100 allow port=22
     200 deny
```

/usr/sbin/sshdはTCPソケットのポート22だけをlistenできる。

```
2000 acl inet_stream_listen port=22
     audit 1
     100 allow task.exe="/usr/sbin/sshd"
     200 deny
```

TCPソケットのポート22は/usr/sbin/sshdだけがlistenできる。

提案する構文の特徴

- TOMOYOのパラメータチェック機能をフル活用できる。

/dev/kvmに対するioctl要求をチェックする。

```
0 acl ioctl path.type=char path.dev_major=10
  path.dev_minor=232
    audit 1
    100 deny task.exe!="/usr/libexec/qemu-kvm"
    200 allow cmd=@PERMITTED_DEV_KVM_IOCTL_CMD_NUMBERS
    300 deny
```

/usr/libexec/qemu-kvmだけが
/dev/kvmに対してioctl要求を行える。

ポリシーファイルのヘッダ部分のnumber_group
PERMITTED_DEV_KVM_IOCTL_CMD_NUMBERS行で
定義されているコマンド番号のみが許可される。

提案する構文の特徴

- RWXfilterのように単機能な制限を行うこともできる。

TCPソケットのconnect要求をチェックする。

number_group PERMITTED_INET_CONNECT_PORTS行で定義されているポート番号だけが許可される。

```
1000 acl inet_stream_connect
    audit 1
    100 deny port!=@PERMITTED_INET_CONNECT_PORTS
    100 allow ip=@PERMITTED_INET_CONNECT_ADDRESSES
    200 deny
```

ip_group PERMITTED_INET_CONNECT_ADDRESSES行で定義されているIPv4/IPv6アドレスだけが許可される。

提案する構文の特徴

- 動作をキーとしているため、システム全体に対する制限が容易に行える。

/bin/mount だけがマウント要求を行える。

```
100 acl mount
    audit 1
    0 deny task.exe!="/bin/mount"
    1 allow target="/proc/" fstype="proc" flags=0x0
    1 allow target="/sys/" fstype="sysfs" flags=0x0
    1 allow target="/dev/pts/" fstype="devpts" flags=0x0
    1 allow target="/dev/shm/" fstype="tmpfs" flags=0x0
    1 allow target="/" fstype="--remount" flags=0x1
    1 allow target="/" fstype="--remount" flags=0x400
    2 deny
```

CaitSithの使い方は？

- 導入手順はTOMOYO 1.8とほぼ同じである。
  - なぜなら、CaitSithはTOMOYO 1.8で使われている
    カーネルパッチを使用しているから。
  - 殆どのフックは既にLSMの中に埋め込まれているため、
    カーネルパッチの適用は簡単。

- 使い方は
  (1) チェックしたいaclブロックを定義する。
  (2) 判断行をログから生成する。
  (3) 無条件のdeny(またはallow)行でそのブロックを
      完結させる。

どうぞCaitSithをお試しください。

- http://caitsith.sourceforge.jp/

  Characteristic

  action

  inspection

  tool.

  See

  if

  this

  helps.