

AN OPERATING SYSTEM APPROACH TO SECURING E-SERVICES

安全な電子サービスを実現するための OS からのアプローチ

Chris Dalton and Tse Huang Choo

電子サービスのアプリケーション実行に
最適なプラットフォーム Trusted Linux の実装

多くのサービスが電子的になり、インターネットで開かれたものになるにつれ、それらサービスの多くは攻撃者達にとって格好の標的となる。非常に多くのインターネットセキュリティの侵害が、電子サービスを提供しているアプリケーションに侵入するという形で発生している。

電子サービスのアプリケーションは一般に複雑で、コードは大きなものになる。従ってこれらのコードにバグが含まれていたとしても、それは驚くべきことではない。実際、このような大きなアプリケーションでは、バグがないことを保証することは難しい。インターネットを介してサービスを提供するということは、システムの弱点を探し出そうとする攻撃者達にシステムをさらすことを意味する。これらのバグが利用されセキュリティの侵害につながるということとはありえないことではないし、実際過去にもそのようなケースはあった。

単一の機構を用いて多様なサービスを同時に配信することがますます増えてきている (ISP, ASP, xSP のサービス提供など)。ホストプラットフォームのセキュリティがアプリケーションへの侵入攻撃から保護されているということだけでなく、アプリケーションが互いに適切に保護されているということが非常に重要になってきているのである。

本稿ではアプリケーション侵入の問題点について詳説し、さらにそれらの問題点を解決するための方法について説明する。我々はアプリケーションサービスのバグを無くそうとしているわけではない (それは難しいことである)。その代わり、我々はアプリケーション実行に用いられる OS に特定の性質を加えることにより、この種の攻撃の影響をかなり軽減させた。

特に、HP 研究所で実装された Linux のセキュア・バージョンである Trusted Linux について考察する。我々は、これは電子サービスアプリケーション実行のプラットフォームとして理想的だと考えている。

アプリケーション侵入に対抗する手段として OS を用いる

カーネルによる制御を用いて、OS レベルでアプリケーション侵入の問題に取り組むことは有効であると思われる。カーネルに実装された制御は、どんなアプリケーションによっても、どんなユーザによってもユーザ空間から無効にされたり破壊されたりすることはない。カーネルによる制御は、個々のアプリケーションコードの質にかかわらず、すべてのアプリケーションに対して適用される。この方法の欠点は、カーネルによる制御は対象範囲が広すぎて使い勝手がよいとはいえない

という点である。しかしこの欠点は今回の場合、特にインターネットサービスを行う大規模アプリケーションに当てはまらない。このことに関しては Trusted Linux に関する詳細な議論において証拠を示し、カーネル制御の実現方法について述べる。

アプリケーション侵入とその影響を防ぐために、システムレベルでの必要事項が二つある。第一には、アプリケーションは攻撃から可能な限り保護され、外から見えるアプリケーションのインタフェースはできる限り狭くし、そのインタフェースに対するアクセスはできる限り適切に制御されなければならない。第二に、アプリケーションがシステムやシステム上の他のアプリケーションに対して与える損害に対して、何らかの制限を設けるべきである。

これら二つの必要事項は、コンテインメント¹ (containment) の抽象的性質によってある程度実現できる。あるアプリケーションにおいて、自分のアクセスできる資源（ファイル、プロセス、ネットワーク、ipc など）を厳密に制御しているなら、侵入を受けた場合でも情報流出は阻止される。コンテインメントは、外部からの攻撃やインタフェースからアプリケーションを保護する。

一旦アプリケーションが侵入されると（例えばバッファオーバーフロー攻撃によって）、攻撃者がシステムのセキュリティを破るために様々な方法で利用される可能性がある。コンテインメントの特徴はこれらの悪用を緩和することが可能であり、完全に防げる場合もある。コンテインメントの利点の中には、アプリケーション設計レベルでのみ得られるものもある。しかし正しく実装されているならば、現存する多くのアプリケーションを変更せずにセキュリティを高めるために有効になるだろう。

アプリケーションの侵入の結果起こる最も一般的な悪用は、以下のように四種類に分けられる（ただし、実際の攻撃はこれらの組み合わせになることもある）。

1. 保護されたシステム資源への直接アクセスを得るための権限の誤用。もしアプリケー

ションが特別な権限で実行されているなら（例えば標準的な Unix で root として実行されているようなアプリケーションなど）、攻撃者は意図せずともその権限を利用できる可能性がある。例えば、攻撃者は保護された OS 資源や同じマシンで走っている他のアプリケーションのインタフェースへアクセスするためにその権限を悪用できる。

2. アプリケーションに施されたアクセス制御の破壊。この種の攻撃は正当な資源（例えばアプリケーションが公開する資源など）へのアクセスを、正当でない方法で得ることである。例えば、Web サーバがそのコンテンツを配信する前にコンテンツのアクセス制御を実施する場合などは、この種の攻撃を受ける恐れがある。Web サーバはコンテンツに対する無制限の直接アクセス権限をもつので、もしその Web サーバを制御することが可能になれば攻撃者も同じ権限を持つことになる。

3. セキュリティ意志決定情報の偽造。この種の攻撃は、通常間接的な攻撃である。侵入されたアプリケーションがメインのサービスではなく、アクセス権限サービスのような補助サービスであったとする。この場合、侵入されたサービスは偽造した情報を供給するために使われ、攻撃者はメインのサービスへのアクセスを得るのである。これは、攻撃者がアプリケーションによって正当に公開された資源への不正なアクセスを得る別の方法である。

4. 保護されていないシステム資源の不正使用。これは、マシンのローカル資源へのアクセスを攻撃者が得る場合である。ここでの資源は保護されていないが、通常はアプリケーションによって公開されてもいない。これらの資源は、さらなる攻撃を行うために利用されることが多い。例えば、攻撃者があるマシンへのシェルアクセスを得ることに成功すれば、ここからそのマシンのアプリケーションやネットワークを介した他のマシンのアプリケーションに攻撃を行うのである。

コンテインメントを用いると、タイプ 1 の悪用はそれほど深刻ではない。たとえ攻撃者

¹ 訳注：「囲い込み」、もしくは「情報の流出制限」と訳すこともある。

がアプリケーションの特権を用いたとしても、アクセス可能な資源は、アプリケーションの内部で利用可能であったものに限られる。これはタイプ4の保護されていない資源へのアクセスについても成り立つ。コンテナメントを用いることにより、アプリケーションからネットワークへのアクセスを妨げることができる（あるいは、少なくとも正しい制御を設定することができる）。攻撃対象のアプリケーションの公開を制限するためにコンテナメントを用いると、タイプ3の悪用にも対処できる。サービスを提供するアクセスは正当なクライアント（アプリケーションサービス）からであると保証できるからである。

タイプ2の悪用は通常、アプリケーション設計、あるいは少なくとも構成レベルで解決されねばならない。ここでの設計方針は、巨大な高度アプリケーション（例えば Web サーバ）に対し、アクセス制御チェックの実施は期待すべきではないというものである。これはまた、保護されている資源に対して、これらのアプリケーションが直接アクセスすべきではないということも意味する。アクセス制御の決定をするコードへの侵入は極力なくしたいので、コードの目的を一つに限定し、できる限り小さくバグが無いようにすべきである。このコードと、それが保護するように設計された資源は、それら自身の区画でホスティングされるべきである。コンテナメントを用いると、巨大で信頼できないアプリケーションから保護された資源にアクセスする場合、より小さく、より信頼できるアプリケーションを通らなければならないようにすることが可能になる[2]。

理論的には、コンテナメントは、OS に備えられると非常に有効な性質であるように思える。

図1では、コンテナメントを用いることによって、アプリケーションが非常に重要なシステム資源から分離されるだけでなく、他のアプリケーションからも互いに分離した状態に保たれている。アプリケーションは別のアプリケーションの処理を妨害することはできないし、その重要なデータにアクセスすることもできない。コンテナメントは、OS が公開するのはアプリケーションの動作に必要なインタフェース（入力と出力）だけであ

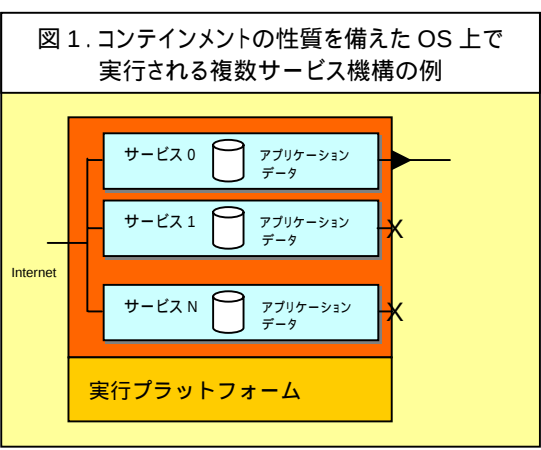


表1. 設定例 (Apache + Jakarta/VM)

設定	HTTP GET
Linux-2.4.0-test5	3298
Trusted Linux (ベースは Linux-2.4.0-test5)	3265
	1%のペナルティ

ることを保証している。これにより、ある特定のアプリケーションへの攻撃の範囲も制限されるし、アプリケーションが侵入された時の損害の程度も制限される。コンテナメントは、個々のアプリケーションだけでなく、それを実行するプラットフォーム全体としての一貫性を保つのに役立つのである。

コンテナメントの実装

カーネルによって実現される OS のコンテナメント機構が利用されるようになったのはここ数年のことである。その典型は機密情報を扱うために設計された[3,4]もので、この種の OS はしばしば、"Trusted" OS といわれる。

コンテナメントの性質は通常、強制アクセス制御 (mandatory access control : MAC) と権限の組み合わせによって実現される。MAC の保護方式は、ファイル、プロセス、ネットワークコネクションといったシステム資源に特定のアクセス制御ポリシーを用いる。このポリシーはカーネルによって行われ、ユーザや侵入されたアプリケーションによってくつがえされることはない。現在の多くの trusted OS は、BellLaPadula ポリシーモデルを用いている[1]。これは形式的モデルであり、システム周辺の情報の流れが見える

ような軍隊の組織のために開発されたものである。スーパーユーザの権限は分割・限定され、アプリケーションは自分が必要とする権限だけを与えられることになる。

コンテインメントという魅力的な性質を備えているにもかかわらず、trusted OS は特定の情報処理の分野以外では広く使われてきたわけではない。その理由は二つ考えられる。

第一に、従来の OS に trusted OS の特徴を加えようとする、既存の OS の特性を損なうことになるのが普通だからである。その OS にはもはや、標準的なアプリケーションや管理ツールが備わっていない。また、利用や管理に関しても非常に複雑になり、標準的な方法が適用できなくなる。

第二に、trusted OS の実装においてよくあることだが、情報流出への制限が余りに強力すぎて、アプリケーションが普通に機能できないのである。この種の問題に関しては通常、統合する努力がかなり必要になる（そしてしばしば、これには費用がかかる）。

Trusted Linux の実装における問題点は主に二つあった。

Trusted Linux

Trusted Linux は、標準的 Linux OS を階層的に拡張したものある。アプリケーション侵入に対する保護に有効であると思われるコンテインメントの性質を備えている。

伝統的な trusted OS のアプローチと同様、コンテインメントの性質をカーネルレベルでのプロセス、ファイル、ネットワーク資源の保護により実現した。しかしながら、我々の強制アクセス制御は伝統的 trusted OS におけるものとはやや異なっている。我々は、アプリケーション統合や、伝統的 trusted OS で存在した管理における問題を縮小することを試みた。

我々の trusted OS の鍵となる概念は、区画分け（compartment）である。マシン上のサービスやアプリケーションは、分離された区画の中で実行される。

区画分けの概念を実現するために、我々は単純な強制アクセス制御とプロセスのラベル付けを用いた。各々のプロセスにはラベルがふられ、同じラベルを持つプロセスは同一の区画に属する。ある区画のプロセスが他の区

画のプロセスと干渉しないことを保証するために、カーネルレベルでの強制チェックを用いた。アクセス制御は非常に単純であり、ラベルは一致するかしないかのどちらかである。BellLaPadula モデルに存在するようなラベルの階層的順序付けはシステム中にはない。

ファイルシステム保護も強制的である。伝統的な trusted OS とは異なり、ファイルシステムへのアクセスを直接制御するためにはラベルを用いない。各々の区画は、それぞれファイルシステムのセクションに関連付けられている。ファイルシステムのセクションは、メインのファイルシステムの chroot である。ある区画で走っているプロセスだけが、ファイルシステムの対応付けられたセクションへのアクセスを持つ。さらに重要なことに、カーネル制御を用いることにより、プロセスが区画中からルートへの遷移を行えないようにした。その結果、chroot をエスケープできなくなるのである。我々はまた、chroot 中のファイルを指定して、変更不可にすることもできる。

区画とネットワーク資源の間の通信経路は、カーネルレベルで制御されたインタフェースを通じて、TCP/UDP と多くの IPC 機構から柔軟に提供される。これらの通信インタフェースへのアクセスは、セキュリティ管理者が各々の区画について規定した規則によって統治される。従来の trusted OS と異なり、特権を用いたり、区画間やネットワーク資源との間の通信を許すユーザレベルでプロキシを用いたりして、強制アクセス制御を無効化する必要はない。

これらの性質により、コンテインメントを提供し、さらにアプリケーション統合をより簡明にする柔軟性を持ったシステムが得られる。これはさらに、管理のオーバーヘッドや trusted なシステムを稼働させるコストの減少につながる。

カーネルの実装

カーネル内部では、保護すべき各システム資源は拡張され、それぞれの資源が属する区画を示すタグが付いている。例えば次のようなデータ構造を含んでいる資源を考える。

- ・ 個々のプロセス
- ・ 共有メモリセグメント
- ・ セマフォ，メッセージキュー
- ・ ソケット，ネットワークパケット，ネットワークインタフェース，ルーティングテーブル要素

タグの割り当ては，多くは継承によって行われ，最初は init プロセスが用いられる．Init には区画 0 が割り当てられている．カーネルオブジェクトは全て，そのオブジェクトを生成したプロセスのラベルを継承する．カーネルソケットのデータ構造に付けられるタグの例を示す．

```
struct socket {
    ...
#ifdef TLINUX
    unsigned long compartment;
#endif
};
```

カーネルの適切な部分で，動的ロード可能なセキュリティモジュールを用いて区画のアクセス制御チェックが行われる．これは別の区画の資源に対するアクセス許可規則の表を参照することで実現しており，実行中のアプリケーションへの透明性が高まる．

アクセス制御規則

各々のセキュリティチェックは規則のテーブルを参照する．規則は次のような形を持っている．

```
source ->
destination method m [attr]
                        [netdev n]

source/destination:
    COMPARTMENT    (名前の付いた区画)
    HOST           (固定 Ipv4 アドレス)
    NETWORK        (Ipv4 サブネット)
m:                サポートされるカーネル機構
                  例 tcp, udp, msg (メッセージ
                  キュー), shm (共有メモリ)
attr:             機構 m を正当化するための属性
n:                利用可能なネットワーク
```

インタフェース 例 eth0

例えば，“WEB” という名前の区画にあるプロセスが，“CGI” という名前の区画が提供する共有メモリセグメントへ (shmat/shmdt()を用いて)アクセス可能にするためには，次のような規則を記述する．

```
COMPARTMENT:WEB -> COMPARTMENT:CGI
METHOD shm
```

区画内の通信を許可する暗黙の規則もいくつかある．例えば，あるプロセスは，自分と同じ区画にあるプロセスのプロセス識別子を見ることを許されている．これにより，規則を定めていない区画の機能を最小限度に留めているのである．区画 0 は例外であり，与えられる権限が少なくなっており，より多くの制約が適用されている．区画 0 は，カーネルレベルのスレッド (例えば swapper など) を実行するのに用いられることが多い．

区画をまたがるアクセスを明示的に許す規則は無いので，そのような試みはすべて失敗に終わる．これらの規則は，他区画の資源へのアクセスを明示的に許可した場合を除き，各区画の間を強制的に分断できるという効果がある．

規則は基本的に方向性を持つ．方向性を持つ規則の効果の一つとして，TCP のソケットコネクションの connect/accept のふるまいに適合する，というものがある．以下では，許可できる HTTP コネクションを特定するのに使われる規則を考えてみる．

```
HOST:* -> COMPARTMENT: X
METHOD TCP PORT 80
```

この規則では 80 番ポートに入ってくる TCP コネクションのみが許されるが，外へ出ていくコネクションは許さないように規定している (図 2) ．

規則の方向性では，外に出ていくコネクションを許すことなく内に入ってくるコネクションを正しく確立するために必要なパケットの逆流は許可している．

図 2. 設定例 外側からの TCP 接続のみを許可

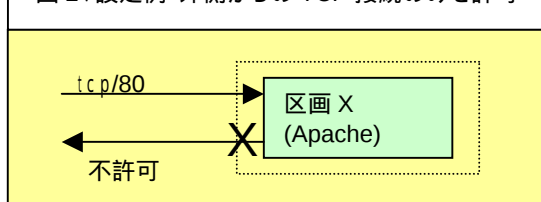


図 3. IP プロトコルスタックの半仮想化

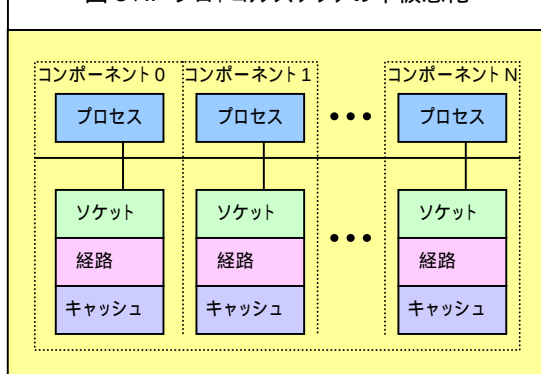
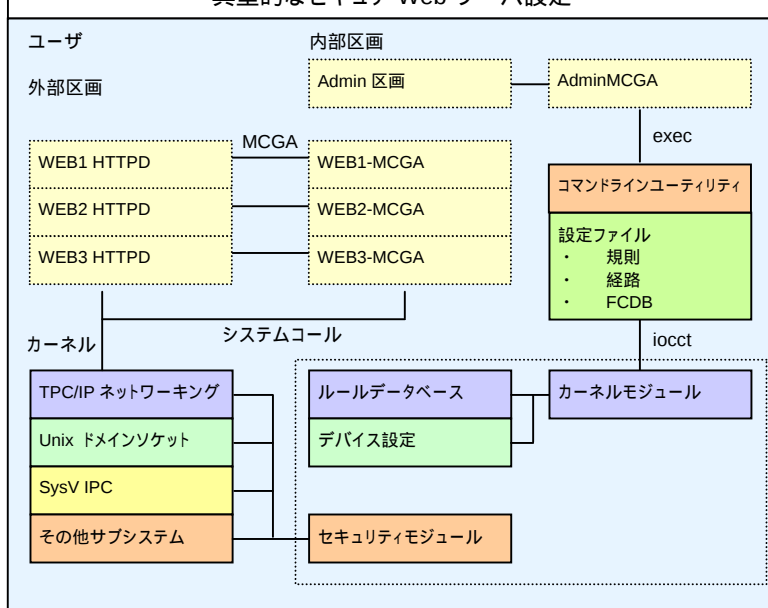


図 4. Trusted Linux におけるサンドボックス機構を実現する典型的なセキュア Web サーバ設定



プロトコルスタックの仮想化

IP プロトコルスタックで用いられるデータ構造の多くにはタグがつけられているので、これらのデータ構造に対する操作を区画ごとに一部を仮想化することが可能である。特に、各々の区画の個別ルーティングテーブルを記述することが可能である。IP スタックの中で用いられるデータの分割は、ソケット

のような高レベルの構成物から個々の IP フラグメントや ARP キャッシュの内容まで対応している (図 3)。

他に仮想化されたシステムとしては、System V の IPC 機構やプロセステーブルなどがある。ipcs コマンドが実行された時には、現在の区画の IPC オブジェクトだけが見えるようになる。同様の制限は ps を用いるプロセスにも適用される。仮想化は呼び出しプロセスの文脈に基づいて起こるので、ユーザレベルのコマンドに変更は必要ない。

インストールと構成

Trusted Linux は、通常の Red Hat に階層化した形でインストールするように設計されている。ファイルシステムに変更は必要ない。インストールにより、通常の Linux カーネルが置き換えられ、残りのネットワークサービスは全て、次のリポートで区画 0 で開始する。

MCGA

(multicompartment-mented gateway agent) とともに Apache を使用するには、特定のソフトウェアパッケージを利用する。MCGA は、離れた区画で個別に名前づけられた CGI バイナリを実行することにより、CGI サンドボックス機構を実現する。

図 4 は、典型的な構成で、三つの HTTP サーバが各自の区画に配置され、別の三区画では MCGA エージェントが実行されている。

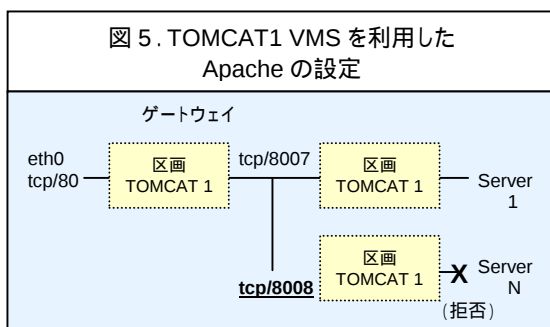
各々のパッケージはディスク容量により制限された区画の数まで、それ自身の区

画内にインストールすることができる (タグはマシンのワードのサイズである)。これにより、Web サーバの多様なインスタンスを互いに独立に実行することが可能となる。また、ある所のセキュリティ違反が他の所に影響しないようにすることができる。

ソースの変更なしに統合したアプリケーションには Apache, Jakarta/Tomcat, HP OpenMail, BEA WebLogic Server がある。

実行効率

最初のテストでは通常の Linux カーネルと比較し、多くの面で数パーセント下がった。テストシステムは、733MHz の CPU のツイン Pentium III、256MB の RAM、Intel Ether-Express Pro/100Mb NIC から成り、表 1 に示されるような結果となった。Apache 1.3.12 が走っており、HTTP GET 命令を用



い、静的な 1024 バイトの HTML ファイルにアクセスする。テストプログラムは Apache の配布ファイルに含まれるユーティリティ “ab” であり、次のように起動した。

```
ab -n 100000 -c 50 [url]
```

構成例：Apache+Jakarta/VMs

ここでは、Web サーバを含む区画化の設定について述べる。Apache は外部とのやりとりがあり、Java サブレットの操作や JSP ファイルの提供を Jakarta/Tomcat のインスタンスに委任している。このインスタンスは二つあり、それぞれ自身の区画で実行される。デフォルトでは、各々の区分は他の区画を妨害しないよう、chroot したファイルシステムを用いている。

図 5 は、ある区画 (WEB) にある Apache のプロセスを表している。この区画は次のような規則を用いることにより、外部からアクセス可能である。

```
HOST: * -> COMPARTMENT: WEB  
METHOD TCP PORT 80 NETDEV eth0
```

規則の中に NETDEV というキーワードを記述することで、Apache が使用可能なネット

ワークインタフェースが規定される。このことは、Apache が二重または多重接続されたゲートウェイシステムの外部インタフェースだけを使うように制限するのに有効である。これにより、Apache のインスタンスが侵入攻撃を受けても、内部へのネットワークインタフェースを用いたバックエンドのネットワークへの攻撃を防止できる。

WEB 区画は Jakarta/Tomcat の二つのインスタンス (TOMCAT1 と TOMCAT2) と通信することが許されている。これは次のような二つの規則による。

1.
COMPARTMENT: WEB -> COMPARTMENT: TOMCAT1
METHOD TCP PORT 8007
2.
COMPARTMENT: WEB -> COMPARTMENT: TOMCAT2
METHOD TCP PORT 8008

TOMCAT1 のサブレットは、次のような規則を用いて Server1 というバックエンドホストへのアクセスを許可されている。

```
COMPARTMENT: TOMCAT1 -> HOST: SERVER1  
METHOD TCP ...
```

しかし、TOMCAT2 はどのバックエンドホストへのアクセスも許可されていない。カーネルは TOMCAT2 からのアクセスを全て拒否する。このことによって、バックエンドネットワークのビューを、実行されているサービス毎に変更できるようになり、またバックエンドのホストを見せるかどうかの設定が区画毎に可能になる。

ここで強調したいのは、この構成例で必要なのはこれら四つの規則だけあるということである。他に規則がなくても、Java VM で実行されるサブレットは外部へのコネクションを張ることはできない。特に、Java VM を用いて、インタフェース eth1 を通じた内部のバックエンドネットワークへの攻撃は不可能である。さらに、Java VM は他の区画 (共有メモリセグメント、Unix ドメインソケット) から資源にアクセスすることはできないし、

リモートホストから直接到達することもできない。この場合、ソースを再コンパイルしたり変更したりすることなく Apache と Jakarta/Tomcat の振る舞いに強制的制限 (mandatory restrictions) をかけることができる。

まとめ

本稿では、個々に動的に実行されるサービスに対してコンテインメントを実現する、Linux ベースのプラットフォームについて述べた。このプラットフォームの主要な利用法は、区画化の手法で (多数の) サービスを実行するというものである。これにより、サービスへの侵入によって起こりうる損害の程度を限定することができる。特に、侵入されたサービスを悪用して同じプラットフォーム内の他のサービスに損害を与えることはできない。同様に、全体としてのプラットフォームの基本的な一貫性は、サービス悪用による攻撃から保護されている。

Trusted Linux は、従来の trusted OS における重厚なアプローチと一般的な OS の中間的な位置付けにある。厳密なアクセス制御をカーネルレベルで実現しているので、新たなセキュリティ機能を利用するために既存のアプリケーションを書き換える必要はない。

Trusted Linux はより安全なカーネルの層を作り、設定を行っていないサービスは特権の無い区画に残すことにより、通常の Linux のセキュリティを高めるように設計されている。ここで採用した方法は他の Unix (HP-UX, Solaris) にも適用可能で、さらに多くの通信機構に対応するような拡張も可能である。

現在のプロトタイプの可能性として興味深いのは、アクセス制御規則のフォーマットを拡張して、遠隔のホストの区分に関しても安全に記述する、というものである。これは IPSec の暗号化機能によって、区画間の通信を安全に行えば実現可能だろう。

文献

1. Bell, D. and Lapadula, L. Secure computer systems. Unified exposition and multics interpretation. Mitre Tech. Rep. MTR-1997, Mitre Corporation, 1975.
2. Dalton, C.I. and Griffin, J.F. Applying military grade security to the Internet. *Computer Net. And ISDN Syst.* 29, 15, (Nov. 1997).
3. Hewlett-Packard Co. HP-UX 10.16 CMW security features guide, 1996.
4. Sun Microsystems Corporation. Trusted Solaris Operating System; www.sun.com/trustedsolaris.

Chris Dalton (cid@hplb.hpl.hp.com) は、英国 Bristol のヒューレットパッカード研究所で技術研究者を勤めている。

Tse Huong Choo (tchoo@hplb.hpl.hp.com) は、英国 Bristol のヒューレットパッカード研究所で技術研究者を勤めている。

訳：対馬英樹 伊藤ちひろ (京都大学大学院・情報学研究科)