情報処理学会論文誌　コンピューティングシステム　Vol.5　No.2　51–62 (Mar. 2012)

情報セキュリティ向上に向けたOS研究の動向

橋本 正樹[1,a)]　安藤 類央[1,2]　前田 俊行[3]　田中 英彦[1]

受付日 2011年7月19日, 採録日 2011年12月9日

概要：近年，情報システムが社会基盤化しているため，そのセキュリティ耐性向上が急務であるが，一方で，セキュリティ・インシデントの発生数は年々増加している現実がある．本稿では，情報セキュリティを担保する最も基礎的なソフトウェアとしてOSを位置付け，参照モニタの設計要件と対応付けながら，特に，仮想化技術，OSプログラムの検証技術，アクセス制御技術に焦点をあてて近年の研究動向を紹介する．また，各技術について，今後の展望と課題を整理する．

キーワード：オペレーティングシステム，仕様記述・仕様検証，ディペンダブルコンピューティング

A Survey of Security Research for Operating Systems

Masaki Hashimoto[1,a)]　Ruo Ando[1,2]　Toshiyuki Maeda[3]　Hidehiko Tanaka[1]

Received: July 19, 2011, Accepted: December 9, 2011

Abstract: In recent years, information systems have become the social infrastructure, so that their security must be improved urgently. In this paper, we introduce the results of the survey of virtualization, operating system verification and access control technologies in association with the design requirements of the reference monitor. Additionally, we show the prospects and challenges for each technology.

Keywords: operating system, specification/verification of specification, dependable computing

1. はじめに

近年，情報システムが社会基盤化しているため，そのセキュリティ強化は重要な課題となっている．今や我々の社会生活は，情報システムによる様々な支援により成立しているため，セキュリティ・インシデント発生時の被害は以前にも増して深刻化している．同時に，情報システムへの要請が，その利用用途の拡大に合わせて多様化しているため，その構成は複雑化しており，インターネットに接続するノード数の爆発的な増加と合わせて，セキュリティ・インシデントの発生数が年々増加する一因となっている．

情報セキュリティを強化するためには，技術・管理・法制・倫理の抜本的な裏づけが不可欠であり，そのための様々な研究・開発や検討が行われている．このうち，特に技術に着目すると，暗号技術・侵入検知技術・認証技術・証拠保全技術等，幅広い研究が行われているが，それらはいずれもそれらの機能が働く基盤の健全性を仮定したものがほとんどである．基盤はいわゆるOSであるが，OSが脆弱であると，その上で動作する様々なセキュリティ強化機能は砂上の楼閣であるし，OSが健全であった場合にも，OSからどのようなセキュリティ支援機能を上位層に提供するべきかは検討が必要な課題である．

そのため，OSそのもののセキュリティ強化と，健全なOSから提供すべきセキュリティ支援機能については，古くから検討が行われ，米国国防総省のTrusted OSをはじめ，近年では，SELinuxやTrusted Solaris等，多くの研究・実装例が存在する．しかしながら，これらは様々な脅威を抜本的に減らす重要な要素ではあるが，いまだ十分な

[1] 情報セキュリティ大学院大学情報セキュリティ研究科
Graduate School of Information Security, Institute of Information Security, Yokohama, Kanagawa 221–0835, Japan
[2] 情報通信研究機構
National Institute of Information and Communications Technology, Koganei, Tokyo 184–8795, Japan
[3] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[a)] hashimoto@iisec.ac.jp

解決策とはいえないため，現在でも各所で活発な研究が行われている．

本稿では，情報セキュリティを担保する最も基礎的なソフトウェアとしてOSを位置付け，特に，OSの仮想化技術，OSプログラムの検証技術，アクセス制御技術に焦点をあてて近年の研究動向を紹介する．はじめに，2章で，OSにおけるセキュリティ研究について，本稿での分類方針を参照モニタの設計要件との関連から説明し，この中で，上記3技術の関係を述べる．その後，3章と4章，5章では，各々について研究動向を紹介し，今後の課題を述べる．最後に，6章で本稿をまとめる．

2. OSにおけるセキュリティ研究

Trusted Computing Base（TCB）は，米国国防総省がTCSEC（Trusted Computer System Evaluation Criteria）[1]の中で定義した概念で，高信頼システムを実現するための基本コンポーネントである．TCBは，参照モニタ*1や形式的に検証可能なセキュリティポリシを含み，特に参照モニタについては，(i) 耐タンパ性があること，(ii) 制御対象が迂回不能であること，(iii) 完全性保証のために十分小さいこと，をその設計要件にあげている．

OSは，情報システムを構成するソフトウェア・スタックの最下層にあることから，上位層の動作を捕捉可能である点で参照モニタの設計要件 (ii) を自然に満たしている．そのため，従来より，OSはTCBを構成する主たるソフトウェアとしての役割を担い，また，OSを参照モニタとしてより強固に構成することで，情報セキュリティを担保する研究が現在でも行われている．同時に，OSは，応用プログラムに実行環境を提供する基本ソフトウェアとしての役割もあるため，ハードウェアの進化と情報システムに対する新しい要請に応えるべく，参照モニタとしての側面以外からの研究も進められてきた．特に，近年では，組み込みシステムや携帯端末，クラウドに代表される分散コンピューティングの普及により，これらからの要請に応える新しいOS技術の研究が各所で行われている．

したがって，OS研究は様々な側面からとらえることができるが，本稿では，特に情報セキュリティとの関連から，近年のOS研究を以下の3つに分類して整理する．すなわち，本稿では，情報システムに対する脅威の中からOSに関係の深いものとして，OSそのものに対する攻撃とOS上で動作するプログラムに対する攻撃に焦点をあてる．本分類では，主に前者への対抗策としてOS検証技術を，主に後者への対抗策としてアクセス制御技術を，両者への対抗策として仮想化技術を整理する．データや入出力に対する暗号化，認証技術，デジタルフォレンジック等もOSと情報セキュリティに関係する技術であるが，本稿では対象外とする．

(1) 仮想化技術
仮想化技術は，ハードウェア資源を有効利用するために古くから研究されてきた技術であり，近年では，クラウド・コンピューティングの基盤技術として広く普及している．OSの仮想化技術を特にTCBとの関連から見たとき，仮想化技術は参照モニタの要件 (i)〜(iii) のすべてに関連する．

(2) OS検証技術
OS検証技術は，従来からあるプログラム検証技術をOSに適用することで，OSの完全性を確認するための技術である．前述したように，TCBの完全性を保証できることは高信頼システムを実現するための要件とされており，仮想化技術とOS検証技術が参照モニタの要件 (iii) に関連する．

(3) アクセス制御技術
アクセス制御技術は，参照モニタを用いて情報システム全体の安全性を担保するための技術で，参照モニタの要件 (ii) に関連する．アクセス制御技術は，参照モニタが完全であることを前提としており，他プログラムをどう制御するかが主な課題となる．

3. OSの仮想化技術

近年，ハードウェア性能の著しい向上により，OSの仮想化技術に対する研究・開発が活発になっている．OSの仮想化技術は，クラウド・コンピューティングの基盤技術としても広く利活用されており，そのセキュリティ強化が重要な検討課題となっている．本章では，仮想化技術に関する近年の研究を，「ハイパーバイザによる仮想マシンの観測」「主記憶装置の仮想化」「入出力機構の仮想化」「仮想マシンの完全性検証」の4つの側面から整理して紹介する．また，今後の課題についても述べる．

なお，4つの分類の論拠であるが，はじめに，情報セキュリティ対策を適切に施すためには，情報システム内で起こるある事象について，防御側からは観測可能で，攻撃側から観測されないことが重要で，そのための研究が活発に行われていることに鑑み，ハイパーバイザによる仮想マシンの観測を第1項目とした．次に，ハイパーバイザによる観測対象は，現状では主に主記憶装置と入出力であるため，これらについても各々整理した．最終項目は，攻撃側が観測を行うために混入するソフトウェアを締め出すためには，ブートプロセスも含めた実行シーケンスの完全性検証が必要で，このために，ハードウェアによるサポートを前提とした研究が各所で行われているため，これらを別途整理したものである．

*1 参照モニタは，アクセス主体とアクセス対象間に正当なアクセス関係を強制する仕組みで，1972年に発表されたアンダーソンレポート [2] により，セキュアなコンピューティングに必須の要素として提示された．

3.1　ハイパーバイザによる仮想マシンの観測技術

ハイパーバイザは，仮想マシンとハードウェアの中間に位置し，リソース管理やスケジューリングを仮想マシンに対して行う．ハイパーバイザにより享受できる利点は，第1に，運用面で，複数の仮想マシンを一元管理できること，すなわち，稼動状況にあわせて割り当てるリソースを動的に伸縮できることがあげられる．第2は，仮想マシンのセキュリティ強化で，ハイパーバイザを用いることで，複雑化し脆弱性を内包するようになった汎用OS上のアクセス制御を強化したり，マルウェアから検出不可能な観測・解析器を構築したりできることである．ハイパーバイザの代表的な実装例としては，Xen [3]，KVM [4]，VMware [5] がある．

ハイパーバイザを用いて仮想マシンを監視し，観測，防御，隔離等の処理を行う概念は，VMI（Virtual Machine Introspection）[6] として提示された．この中で，VMIは，仮想マシンからハイパーバイザ側のコードを修正できないこと，ハイパーバイザ側から仮想マシンのすべての状態を観測できること，仮想マシンから発行されるコードを捕捉できることの3点により，不正アクセス検知と防御に有効であると指摘されており，本研究を参照した仮想マシンの観測についての論文が多数発表されている．

VMIによる観測方法には，能動的な方法と受動的な方法がある．能動的な方法は，仮想マシンの状態を外部から一定時間ごとに取得し，情報を抽出するもので，代表的な研究例として，メモリのスナップショットを取得解析するVolatality [7] がある．受動的な方法は，仮想マシン内部でリソースアクセス等のイベントが起きた場合に関連する情報を引き出す方法で，代表的な研究例として，Lares [8]，Xenprobes [9]，VMScope [10] がある．

受動的な方法と能動的な方法の具体的な実装方法としては，内部観測（in-the-box）方式と外部観測（out-of-the-box）方式の2通りあり，これらは観測機構の攻撃者側からの検出可能性とセマンティックギャップ [11] の解消可能性[*2]，実装に係るコストのトレードオフ [12] との関連性から論じられることが多く，現状では，観測対象の仮想マシン内部から観測機構が検出困難であるという理由から，外部観測方式が採用されることが多い．一方で，外部観測方式には，内部観測方式と比べて，観測可能な情報量が低下するという課題がある．

3.2　主記憶装置の仮想化

仮想化技術を実装する際には，主記憶装置，特にページング機構の扱いが重要な課題となる．仮想マシンが利用している物理アドレスは，ハイパーバイザによって仮想化された擬似的な物理アドレスであり，真の物理アドレスへのアクセスを調停するためには，通常の場合と比べ，二重のアドレス変換が必要となる．仮想化技術特有のこの2段階以上のページングは，ソフトウェアで行われる場合とハードウェアで行われる場合がある．

ソフトウェアによる主記憶装置の仮想化には，代表的なものにXenのShadow Pagingがある．Shadow Pagingは，主記憶装置のページフォールトを捕捉し，仮想マシンによる主記憶装置へのアクセスを仮想化するもので，仮想マシンが認識している物理アドレスを，実マシンの物理アドレスに変換するための機構をShadow Page Tableという．これに関連し，Shadow Page Tableを修正することで，rootkit等の悪意のあるカーネル拡張を検出する研究がさかんに行われている．

たとえば，Panorama [13] では，Google Desktop [14] をケーススタディとした汚染検出の手法が提案されているし，Ether [15] では，ハードウェアによる仮想化支援機能を用いてマルウェアを検出する手法が提案されている．同様に，nEther [16] では，RDTSCからのデータ採取により，ゲストOSから外部で動いているマルウェアを検出する方法が提案されている．

その一方で，攻撃者の視点から，2006年には，ハイパーバイザが提供する強力な隔離方式を利用することで，ゲストOSからは不可視のrootkitを構築可能なSubVirt [17] やBlue Pill [18] が提案されている．同様に，Ristenpartらは，Amazon EC2を対象として，同じ物理マシン上で動作する仮想マシン間のサイドチャネル攻撃が可能であることを示した [19]．そのほか，Chenらは，防御側のハイパーバイザによる解析をマルウェアが検出する方法を包括的に調査し，その分類を行っている [20]．

ハードウェアによる主記憶装置の仮想化には，代表的なものにIntel VT-dやAMD-Vがある．Intel VT-dやAMD-Vでは，仮想化技術にハードウェアによるメモリ管理機構が利用可能で，これにより，ハイパーバイザによるアドレス変換の負荷や実装の負荷を低減することができる．これに関連し，Intel VT-dのEPT（Extended Page Table）を利用することで，不正なカーネル拡張を検出する研究として，HUKO [21] がある．HUKOは，Shadow Pagingにおいて共有されていたテーブルを物理的に隔離し，ゲストOSごとに所有することで，ゲストOSのページングによるオーバヘッドを削減すると同時に，アクセス制御を強化したものである．仮想化技術を用いたアドレス空間の分離による保護については，SIM [12] もあるが，SIMがShadow Pagingを用いているのに対し，HUKOはハードウェア支援によるページングを用いてVMM遷移やTLBにかかる負荷を低減させている．

[*2] セマンティックギャップとは，仮想マシン内部のリソースアクセスに関連するセマンティック情報を，ハイパーバイザ側で得られる情報から再構築できないことをいう．すなわち，セマンティックギャップの解消とは，ハイパーバイザが仮想マシン上で起こるリソースアクセスのセマンティックを再構築可能することである．

3.3　入出力機構の仮想化

ハイパーバイザから仮想マシンに対してアクセス制御や完全性検証等，セキュリティ強化を行う際には，前節で紹介した主記憶装置の仮想化に加えて，入出力を仮想化する必要がある．入出力の仮想化は，主記憶装置の仮想化同様に，ソフトウェアによるものとハードウェアによるものがある．

ソフトウェアによる入出力機構の仮想化では，デバイスドライバによる入出力要求の捕捉と中継が行われる．Xen の Split Kernel Driver は，その代表例で，仮想マシンとハイパーバイザの間で共有メモリを設置し，イベントチャンネルを用いて入出力を仮想化する．また，この応用として，XenAccess [22] では，Split Kernel Driver である blktap ドライバを用いて仮想マシンのイベントの修正し，システム全体の負荷を軽減することができる．また，sHype [23] は，仮想マシンのスナップショット生成・破棄時や仮想割込み発生時に，セキュリティポリシに従った動作を強制することができる．

別の実装例としては，BitVisor [24] がある．BitVisor は，準パススルーと呼ばれるアーキテクチャを採用したハイパーバイザである．BitVisor は，他のハイパーバイザと比較して，仮想マシンが生成する I/O 要求のうち，アクセス制御や暗号化に関連するもののみを捕捉し，中間処理を行うことでセキュリティを強化する．同アーキテクチャを採用することにより，ハイパーバイザ側は，制御 I/O およびデータ I/O のみを処理すればよく，仮想マシン間の保護やスケジューリングが不要になる．BitVisor のコード数は，コア部分で約 20 KLOC，準パススルードライバは通常のドライバの約 10 分の 1 程度とすることに成功している．また，情報セキュリティの CIA の観点から見ると，他のハイパーバイザが機密性（Confidentiality）の確保に傾倒しているのに対し，BitVisor は機密性に加えて，完全性（Integrity）の確保にも効果的な機能を持っていることが特徴である．

ハードウェアによる入出力機構の仮想化では，仮想マシンによるハードウェアへの直接的なアクセスを支援する IOMMU が提案されている．IOMMU は，DMA（Direct Memory Access）時のアドレス・リマッピング機能をハードウェアで実現することで，物理デバイスのアドレスを仮想マシンが直接扱うことを可能にする．これと関連し，Intel VT-d では，さらに TXT（Trusted Execution Technology）[25] によって，DMA を利用した不正なコード転送を防止する機能も実現されている．TXT は，DMA の際に発生するアドレス・リマッピング機能に介入し，保護された領域への DMA を禁止する．また，Intel Vt-i と Intel Vt-d によりゲスト OS のメモリアクセスを含む I/O 空間の完全な分離が可能になることで，機密性を確保することができる．

3.4　仮想マシンの完全性検証

仮想マシンへの攻撃が増加するにつれ，仮想化マシンの構成要素に対して完全性検証を行う研究がさかんに行われるようになっている．物理マシンに対する完全性検証を可能とするデバイスに TPM（Trusted Platform Module）があるが，この TPM を仮想化し，物理マシン上の複数の仮想マシンの構成証明を行う vTPM [26] が提案されている．

vTPM では，仮想マシンから TPM への入出力要求を捕捉するために，先に紹介した Split Device Driver を仮想マシンと仮想マシンモニタの双方で実装している．TPM を用いた Trusted Boot には，SRTM（Static Root of Trust Measurement）と DRTM（DynamicRoot of Trust Management）の 2 種類がある．STRM は，Sailer らの研究 [27] や Bitlocker [28] 等で利用されている．2004 年の Sailer らの研究では，ブートローダからカーネル，実行されるアプリケーションにわたって信用の連鎖（Chain of Trust）を確保する手法が提案されている．これにより，OS にロードされるコードやデータは，動的に TPM によって完全性が検証されるようになる．Kauer は，2007 年時点での SRTM への攻撃手法と AMD の SKINIT を用いた OpenSecure Loader（OSLO）の構築方法を提案している [29]．

また，より進んだ完全性検証を行うものとして，TOCTTOU（time of check to time of use）の一貫性を検証することが可能な HIMA [30] が提案されている．HIMA では，設計目標として強力な隔離と TOCTTOU 攻撃に対する防御にプライオリティを置き，そのための手法として，ゲスト OS に対する能動的な観測とゲスト OS のメモリ保護機能を提案している．

HyperSentry [31] は，別の研究例で，ハイパーバイザ自体の完全性検証を目的としている．HyperSentry では，out-of-channel と呼ばれる検証対象からは検出不能な専用の観測機構と通信路を用いて，ハイパーバイザそのものの完全性を検証する．具体的には，ハイパーバイザ内に測定機構を設置し，SMI（System Management Interrupt）handler と IPMI（Intelligent Platform Management Interface）を経由して，リモートにある検証器と通信する．これにより，ハイパーバイザからも十分な隔離性を確保しつつ，ハイパーバイザを含むシステム全体の完全性検証を実現している．

3.5　今後の課題

OS の仮想化技術に関連する今後の課題は，前節までに紹介した各研究成果を応用することで，仮想化技術に関連する各種セキュリティ機能をよりいっそう強化することである．これは，汎用 OS がますます大規模化・複雑化し，脆弱性を内包する可能性が高まる一方で，今後は組み込みをはじめとした各分野で汎用 OS を採用する傾向が高まることが予想されるので，セキュリティ機能を実現する領域の

表 1　OS 検証手法の比較
Table 1　Comparison of OS verification approaches.

検証手法	検証可能な性質	検証のコスト
定理証明支援系（4.1 節）	優	可
ソースコードモデル検査（4.2 節）	良	良
安全なプログラミング言語（4.3 節）	可	優

構築と隔離を，よりハードウェアに近い層であるハイパーバイザで行う仕組みが必要となることが想定されるためである．

たとえば，アクセス制御に関連するセキュリティ機能の強化としては，不正なカーネル拡張を検出防止する HUKO のようなシステムが，現在各所で活発に研究が進められているし，マルウェア検出に関連するものとしては，マルウェア検出・解析用のハイパーバイザに関する検討が進められており，現在は，その実装方法が課題となっている．また，セマンティックギャップを解消することで，仮想マシン内の悪意のある操作をハイパーバイザで検出する研究 [32] も進められている．

4. OS の検証技術

OS は計算機システムの基盤となるソフトウェアであり，その信頼性・安全性は，計算機システム全体のセキュリティにも大きく影響する．このため，OS プログラムを直接的に検証しようとする研究が長年行われてきた．本章では，このうち 3 つの手法「定理証明支援系を用いた検証手法」「ソースコードモデル検査を用いた検証手法」「安全なプログラミング言語を用いた検証手法」についてそれぞれ紹介する．なお**表 1** は 3 つの検証手法を簡単に比較したものである．本章ではまた，OS 検証の今後の課題についてもふれる．

4.1　定理証明支援系を用いた検証手法

定理証明支援系とは，何らかの定理とその証明を入力として受け取り，その証明の正しさを判定するプログラムである [33], [34]．まず OS プログラムの安全性・信頼性を定理として表現し，次にその証明を作成して，定理証明支援系を用いてその証明の正しさを検証することで，OS の安全性・信頼性を保証・検証する．具体的には，たとえば，OS プログラムの振舞いを抽象状態機械上の状態遷移として表現し，その抽象状態機械が何らかの性質を満たしていること（たとえば状態機械が不正なメモリ操作を行わないことや状態機械が予期せず停止したり，無限ループに陥ったりしないこと）を証明することで，OS の性質を保証・検証することができる．

定理証明支援系を用いた検証手法の利点は，その支援系で扱うことができる任意の性質の検証が可能な点である．一方，定理の証明を人手で作成する必要があるため，検証のコストが非常に大きくなるのが欠点である．

Kit [35] は，そのプログラムが直接検証された最初の OS カーネルである．Kit は，約 300 行の機械語（von Neumann 型の仮想的な計算機の機械語）からなる非常に小さなカーネルであり，カーネル内のタスクの独立性等が Boyer-Moore 定理証明器 [36] を用いて検証されている．具体的には，カーネルが満たすべき抽象仕様および機械語の実装を Boyer-Moore 論理を用いて定義し，抽象仕様がカーネル内の各タスクの独立性を保証していることと，実装が抽象仕様を正しく実現していることを，Boyer-Moore 定理証明器を用いて証明している．

より大規模な OS の例としては，seL4 [37] があげられる．seL4 は約 8700 行の C 言語プログラムと約 600 行のアセンブリコードによって実装されたカーネルであり，フォーマルに与えられた仕様をこの実装が正しく実現していることが，Isabelle/HOL [33] を用いて保証・検証されている（ただし，seL4 はマイクロカーネルであるため，メモリ管理等の複雑な部分はカーネル外にあり，検証の対象とはなっていない）．実際の検証作業は以下のように行われた．まず，カーネルが満たすべき性質・振舞いを抽象仕様として定義する（たとえばシステムコールのインタフェースや振舞い等）．次に，抽象仕様の抽象的実装として，実行可能仕様を定義する．具体的には，プログラミング言語 Haskell [38] のサブセットを用いて抽象仕様を実装するプログラムを記述し，そのソースコードから実行可能仕様を自動生成する．そして，こうして得られた実行可能仕様が，抽象仕様と正しく対応していることを証明する．次に，実行可能仕様を C 言語のサブセットを用いて実装する．この C 言語のサブセットの意味論は，Isabelle/HOL 上でフォーマルに定義されているため，C 言語による実装を Isabelle/HOL でほぼ直接扱うことができる．最後に，この C 言語による実装が，実行可能仕様と正しく対応していることを証明する．なお，これらの検証作業には，全体として約 22 人年を要したとのことである．

4.2　ソースコードモデル検査を用いた検証手法

ソースコードモデル検査 [39], [40] とは，モデル検査技術 [41] を直接プログラムのソースコードに適用する手法である．より具体的には，入力として与えられたソースコードからモデルを抽出し，プログラムがとりうる状態を網羅的に検査することで，プログラムの性質を保証・検証する．

ソースコードモデル検査を用いてOSを検証することの利点は，定理証明支援系を用いた手法とは異なり，人手を基本的に必要としない点である．一方，欠点としては，一般にモデル検査には多くの計算資源（CPU時間・メモリ等）が必要となることが多い点があげられる．

SDV（Static Driver Verifier）[42]は，Windowsのデバイスドライバを検証するためのフレームワークである．より具体的には，デバイスドライバが，SLICという仕様記述言語で記述された仕様（主にカーネルAPIの利用方法）を守っているかどうかを，ソースコードモデル検査器SLAM [43]を用いて検証する．OSカーネル本体の検証は対象ではない．なお，SDVはMicrosoftのデバイスドライバ開発キット（Windows Driver Kit）に含まれており，すでに実用に供せるレベルにある．

また，直接ソースコードモデル検査が適用されているわけではないが，類似した方法でOSカーネルを検証する試みとして，Verveにおける Nucleusと呼ばれる部分の検証 [44]があげられる．Verveは型安全性が保証・検証されたOSであり，その一部であるNucleusは，主にメモリ管理（ガーベジコレクション）やスレッド管理（スタック管理），ハードウェア管理（割込み処理・デバイスドライバ等）を担っている．このNucleusの検証は次のようにして行われた．まずNucleusのアセンブリコード実装，およびNucleusが満たすべき仕様を，Boogie [45]というプログラム検証器上で表現し，Boogieへ入力として渡す．するとBoogieは，Nucleusが仕様どおりに振る舞うかどうかを表す検証条件（verification condition）を生成し，この条件が満たされるかどうかを，Z3 [46]と呼ばれるSMT solverを用いて検証する．このように，Nucleusの検証は基本的に自動である．なおNucleusは，約4,500行のBoogieコードからなる（これはアセンブリコードの行数では約1,400行に対応する）が，このうち約7%は，完全に自動で検証を行うためのヒントとして人手で与えられた注釈とのことである．また，アセンブリコード実装および仕様のBoogie上での記述には，約9人月を要し，Boogieによる Nucleus自動検証には，CPUがIntel Core2 2.4 GHz・メモリが4 GBのマシン上で，272秒を要したとのことである．

4.3　安全なプログラミング言語を用いた検証手法

本稿では，「安全なプログラミング言語」とは，厳密な型検査等により，実行時にプログラムが不正なメモリ操作・コード実行等を行わないことを保証・検証できるようなプログラミング言語のことを指す．このような安全なプログラミング言語を用いてOSプログラムを記述することにより，OSが不正なメモリ操作を行わないこと等を，型検査等により保証・検証することができる．厳密な型検査は，プログラムを解析することで，プログラム中の変数や式を型（整数やメモリ参照等，データやコードの種類を表す分類）で分類し，プログラムが実行時にその分類（型）に従って振る舞うかどうかを検査する．このため，厳密な型検査に通ったプログラムは，型エラーを生じるような不正なメモリ操作やコード実行を行わないことが保証される．

安全なプログラミング言語を用いてOSプログラムを記述することの利点は，OSプログラムの検証が，プログラミング言語の型検査等により自動的に行われる点である．また，ソースコードモデル検査と比べて，検証に要する計算資源（CPU時間・メモリ等）が少ないという利点もある．一方，検証できる性質が基礎的なものに限られるという欠点がある．また一般のOS開発者にとって，安全なプログラミング言語を用いてOSを記述することは敷居が高いという問題もある．

SPIN [47]は，安全に機能を拡張することができる，マイクロカーネル方式のOSカーネルである．伝統的なマイクロカーネルでは，カーネル拡張をカーネル本体と異なる特権レベルで実行することでカーネルの安全性を保証しようとするが，カーネル拡張とカーネル間の通信，またカーネル拡張間の通信にオーバヘッドが生じるという問題があった．これに対し，SPINのカーネル拡張は，型安全なプログラミング言語 Modula-3 [48]を用いて記述できるため，カーネル本体と同じ特権レベルで実行しても安全性が保証され，また通信によるオーバヘッドを低減できる．具体的には，仮想メモリ領域管理やネットワーク通信処理等，低レイテンシが求められる処理が，Modula-3を用いたカーネル拡張として実装されている．なおSPINでは，カーネル本体は検証の対象ではない．またModula-3のコンパイラにバグがあると，カーネルの安全性が破れる可能性がある．つまり，Modula-3のコンパイラをシステムの Trusted Computing Base（TCB）に含める必要がある．

Singularity [49]は，型安全なプログラミング言語C# [50]，C#を拡張したプログラミング言語Sing#，およびアセンブリ言語で記述されたOSである．厳密にいえば，Sing#は，C#を拡張したプログラミング言語Spec#をさらに拡張したものである．Spec#は，オブジェクトのメソッドの事前条件・事後条件等を明示的に記述できるようにC#を拡張したものである．Sing#は，Spec#をさらに拡張し，プロセス間の通信手段としてチャネルを導入したものである．Singularityでは，OSの各コンポーネントはそれぞれプロセスとして実装されており，コンポーネント間の通信は上述のチャネルを介して行われる．具体的には，Singularityは約280,000行のC#プログラムと約90,000行のSing#（およびアーキテクチャ依存のアセンブリプログラム）からなり，C#およびSing#で記述された部分の型安全性・コンポーネント間の独立性等が保証される．ただし，メモリ管理やスレッド管理等の部分は直接の検証の対象外である．なお，C#・Sing#のプログラムは型安全なアセンブリプログラム（型付きアセンブリ言語 [51]）

の一種に変換され，アセンブリプログラムのレベルで型安全性を検査することができるため，（SPIN とは異なり）C#・Sing# のコンパイラをシステムの TCB に含める必要はない．

TOS [52] は，OS カーネルの記述向けに拡張された型付きアセンブリ言語 TALK [53], [54] を用いて記述された OS カーネルである．TALK は，メモリを表すための可変長配列やアドレス計算を扱うための整数制約，メモリの安全な strong update（メモリ領域の型を変更するようなメモリ操作）等を型システムで扱えるようにすることで，従来の型付きアセンブリ言語では難しかった OS カーネルの記述とその型検査を可能としている．TOS は約 3,000 行の TALK プログラムからなる，機能が限定された OS カーネルであるが，他の安全なプログラミング言語を用いた OS 検証の研究では直接の対象にされてこなかった，メモリ管理やスレッド管理等の部分も含めて TALK で記述されており，そのメモリ安全性が TALK の型検査によって保証・検証可能である点が特徴である．

4.4　今後の課題

前節までに紹介したとおり，OS プログラムの直接検証は研究レベルではすでに実証されつつあり，今後の課題は，より大規模かつ実際的な OS へ検証技術を適用することにある．これを実現するためには，前節で紹介したいくつかの検証技術を組み合わせて利用することが現実的であると考えられる．実際，前述の Verve [44] では，Nucleus 以外の部分については，別途 C#（すなわち，安全なプログラミング言語）で記述されており，全体として OS カーネルの型安全性が保証されるようになっている．

また，より学術的・技術的な課題としては，近年広く普及しているマルチコア CPU 等，複数のプログラムが同時に実行される環境における OS プログラムの安全性をどのように検証するか [54] 等があげられる．従来の OS 検証の研究は，基本的に単一 CPU の計算機環境を仮定しており，特に，複数のプログラムが共有のメモリを同時に操作する場合に生じる競合状態やメモリ一貫性モデルの問題 [55], [56], [57] への対処が必要となる．

5.　アクセス制御技術

アクセス制御技術は，情報セキュリティを支える最も基礎的な要素技術の 1 つであり，機密性・完全性・可用性のいずれの保護にも関係する．このため，2 章で述べたとおり，アクセス制御機能を TCB としての OS から上位層に提供し，情報セキュリティを担保するための研究が半世紀近く前から継続的に行われており，初期の研究成果は，TCSEC（Trusted Computer System Evaluation Criteria）[1] に見ることができる．また，近年では，ISO/IEC10181-3 [58] として，高度に抽象化されたアクセス制御システムのモデルが標準化されている．以降の節では，OS によるアクセス制御機能の主要な構成要素として，セキュリティポリシモデル，セキュリティポリシ記述言語，セキュリティポリシ検証技術，アクセス制御メカニズムについて，それぞれ近年の研究動向を紹介し，今後の課題について述べる．

5.1　セキュリティポリシモデル

情報システムは，その用途に応じて様々な水準の機密性・完全性・可用性を必要とするため，この特性を反映した様々なセキュリティポリシモデルが従来から研究されてきた．主要な類型は，機密性保護型，完全性保護型に加え，これらの混成型で，各類型の代表的なセキュリティポリシモデルは，機密性保護型として Bell-LaPadula モデル [59]，完全性保護型として Biba Integrity モデル [60] と Clark-Wilson モデル [61]，混成型として Chinese Wall モデル [62] と RBAC モデル [63], [64] 等がある．以下に，近年の代表的なセキュリティポリシモデルの研究例を示す．

RBAC モデル [63], [64] は，Role（役割）を中核としたセキュリティポリシモデルで，情報システム内のアクセス制御を実世界の組織構成と各人の責務に合わせて実現しやすいため，直感的に理解しやすく，正しく運用すれば最小特権の原則に沿ったアクセス制御を実現可能であるため，SELinux や Solaris をはじめとした様々な OS で採用が進んでいる．RBAC モデルの近年の研究成果としては，RBAC モデルの形式的な定義と基礎モデルの拡張 [65] や，RBAC モデルの標準化・ANSI 標準への採用 [66] がある．そのほかにも，RBAC モデルには，誰がどのように RBAC システム内の様々な関係を設定するべきか，という管理上の課題があるが，これに対しては，ARBAC の諸研究 [67], [68] や，管理上の課題に焦点をあてた RBAC モデルの分析がある [69]．また，実際の OS に実装された RBAC モデルについて，そのセキュリティ特性を形式的に検証する方法も提案されている [70]．

TBAC（Task-Based Authorization Controls）モデル [71] は，Authorization Step（AS）として複数の認可を認可手順としてグループ化し，アクセス主体・アクセス対象・操作内容に加えて，AS 名と AS 内の工程名を指定して認可判定を決定するセキュリティポリシモデルである．TBAC では，AS ごとにサブジェクトに付与するアクセス権限を，その前後関係を考慮しながら逐次有効化・無効化するため，たとえば，トランザクションの処理状況に応じて細かなアクセス権限を指定できる．これにより，TBAC は，特定の認可手順を抽象的に構造化し，サブルーチンのように繰り返し利用可能とすることで，ポリシ全体の見通しを向上させることができる．

5.2　セキュリティポリシ記述言語

セキュリティポリシ記述言語は，セキュリティポリシを

表現するためのもので，アクセス制御の仕様を記述する言語として見ることができる．近年，セキュリティポリシ記述言語の研究では，述語論理をはじめとする数理論理学の知見を応用した研究が行われており，主要な課題は，アクセス権限の委譲・制限・否定等に対する表現力，人間にとっての文法的な明快さ・読みやすさ，簡潔で明快な意味論，効率的な認可判定手順，拡張性等がある．以下に，近年の代表的なセキュリティポリシ記述言語の研究例を示す．

SecPAL [72] は，比較的大規模なシステムで，セキュリティポリシを管理領域ごとにモジュール化して構成することを前提とし，分散管理された領域間でアクセス権限の委譲を柔軟に記述するためのセキュリティポリシ記述言語である．SecPAL は，制約論理言語による高レベルなセキュリティポリシ記述言語で，認可判定要求は節の集合に対する問合せが成功したときに承認される．SecPAL の文法は自然言語に近く，意味づけは 3 つの推論ルールから構成されており，否定的なクエリや再帰的な述語，回数を指定可能な権限委譲やその他様々な制限をサポートすることで，多くのセキュリティポリシモデルを汎用的に表現可能である．SecPAL は，クラウド OS である Windows Azure のセキュリティポリシ記述言語として実装するために，現在も研究が進められている．

Lithium [73] は，論理的な否定を形式的に正しく推論できるセキュリティポリシ記述言語である．Lithium は，一階述語論理を基礎にした高レベル言語で，再帰的な表現を制限することで否定を含んだ推論を実現している．Lithium は，たとえば，複数のポリシをマージしてそのアクセス制御仕様を分析する場合に，あるアクセスが未認可であることを形式的に確認できる点で有用であるが，一方で，多くのセキュリティポリシモデルで必要となる，アクセス権限の委譲を簡潔に表現できないという課題がある．Lithium には，その文法や一階述語論理の知識がなくても利用可能とするために，通常の英文法によってポリシ記述ができるようなフロントエンドも開発されている [74]．

5.3　セキュリティポリシ検証

セキュリティポリシ検証は，セキュリティポリシが特定のアクセス制御に関する仕様を満たすことを検証するための技術である．近年の情報システムは大規模・複雑化しているため，前節で紹介したような言語によるセキュリティポリシの記述量が特に増大する傾向にあり，実現されるアクセス制御仕様を人間が明快に把握できないことが多い．この課題に対処するために，記述されたセキュリティポリシを容易に解析するための検証手法が求められており，活発な研究が進められている．以下に近年の代表的なセキュリティポリシ検証の研究例を示す．

RBAC-PAT [75] は，ARBAC モデルを対象として，ある Role を始点にしたときの情報流の到達性や可用性，情報システムの区画化，最弱点等を検証することができる．RBAC-PAT は，複数の管理者によってアクセス権限の変更が随時行われるような ARBAC モデルでは，ポリシの分析が困難となるという課題に対処している．具体的には，あるユーザに対する特定 Role の到達性・可用性や，Role 間の内包性・最小セット等の関係，不要 Role の発見，オブジェクト間の情報流を検証することができる．

PALMS [76] は，MLS ポリシの形式的な仕様を定義することで，分析対象の MLS ポリシに存在する全情報流を検証することができる．また，2 つの MLS ポリシ間の整合性を自動的に検証することができる．PALMS は，Prolog ベースのツールとして実装されており，与えられた MLS ポリシが，$*$ プロパティやシンプルセキュリティコンディション等を満たすこと，アプリケーションの MLS ポリシが，そのホスト OS の MLS ポリシと整合がとれていること等を検証することができる．PALMS は，MLS ポリシのみを対象としているが，現実の情報システムでは，複数のセキュリティポリシモデルが連携する場合が多いため，MLS ポリシと他のセキュリティポリシモデルが連携する際にも検証可能とするように研究が進められている．

5.4　アクセス制御メカニズム

アクセス制御メカニズムは，セキュリティポリシ記述言語によって記述された個々のアクセス制御規則を情報システムに強制するための仕組みである．具体的な実装手法としては，ACL 方式と Capability 方式が従来から存在し [77]，長年 ACL 方式が広く採用されてきたが，最小特権の原則によるアクセス制御を実現しやすいという利点があるため，Capability 方式の研究も継続されている．また，セキュリティポリシとの連携に重点を置いた強制アクセス制御機構を既存の OS に追加するための研究も行われている．以下に，近年の代表的なアクセス制御メカニズムの研究例を示す．

seL4 [37] は，L4 マイクロカーネルをベースにセキュリティ拡張を行った研究用 OS で，take-grant モデル [78] を Capability 方式と組み合わせて実装し，各種カーネルオブジェクトに対するアクセス制御を実現している．ここで，L4 のカーネルオブジェクトとは，スレッド，アドレス空間，プロセス間通信，未使用の物理メモリを示す untyped-memory で，これらに対するアクセス権限は Capability を通して授受する仕組みになっている．別の Capability 方式による研究例としては，Capsicum [79] がある．Capsicum は，UNIX の標準 API を拡張する研究用 OS で，Capability を用いることでプロセスやアプリケーションのサンドボックス化を容易かつ細粒度に実現している．Capsicum は，OS からアプリケーション層の区画化を一元的に制御するのではなく，個々のアプリケーションが自身の区画化を容易に実現できるよう OS として支援するという設計思想で

研究が進められている．

Flask セキュリティアーキテクチャ（FLSA）[80] は，セキュリティポリシによる決定を強制的に執行するオブジェクトマネージャと，与えられたセキュリティポリシに従ってアクセス可否の決定を下すセキュリティサーバから構成される．各々は，あらかじめ定められたプロトコルに従ってデータ通信を行い，FLSA 全体として参照モニタの役割を果たす．FLSA は，様々なセキュリティポリシモデルを柔軟に実現できることが大きな特徴である．SELinux は FLSA の Linux に対する実装で，そのほか，SEBSD も FLSA の実装例である．同様の強制アクセス制御機構としては，TrustedBSD MAC Framework [81] が提案されており，FreeBSD や Darwin に実装されている．

5.5　今後の課題

アクセス制御技術は，比較的古くから研究が進められてきたもので，最も基礎的な理論はすでに確立されている．同時に，情報セキュリティへの要請の変遷に合わせて，継続的にその拡張や応用が求められており，近年では，情報システムの社会基盤化に合わせて，前節までに紹介したような諸研究が進行中である．

今後の課題は，仮想化技術を基礎としたクラウド環境や，スマートフォンをはじめとした組み込み環境等，要請されるアクセス制御仕様や，これを実現するための前提が異なる環境にこれらの成果を応用することである．具体的には，分散システム向けのセキュリティポリシモデルやその記述言語，それを強制する分散アクセス制御メカニズムが求められており，これに向けた研究は，すでに各所で検討が進められている．たとえば，FLSA の適用範囲を，スタンドアローン・システムの OS 層で捕捉可能な対象から，アプリケーション層や他システムの対象に拡張するためのアクセス制御メカニズム等が研究されている [82], [83]．

6.　まとめ

本稿では，OS に関する近年の様々な研究を情報セキュリティとの関連から整理し，各技術分野の課題を示した．諸研究の整理にあたっては，参照モニタの設計要件との対応から，特に仮想化技術・OS 検証技術・アクセス制御技術に着目して分類し，各技術分野をさらに細分化したうえで，項目ごとに概要と研究例，課題を述べた．具体的には，仮想化技術については，ハイパーバイザによる仮想マシンの観測・主記憶装置の仮想化・入出力機構の仮想化・仮想マシンの完全性検証に，OS 検証技術については，定理証明支援系を用いた検証手法・ソースコードモデル検査を用いた検証手法・安全なプログラミング言語を用いた検証手法に，アクセス制御技術については，セキュリティポリシモデル・セキュリティポリシ記述言語・セキュリティポリシ検証・アクセス制御メカニズムに，各々細分化して説明した．

今後の情報社会の健全な発達には，強固な OS によるセキュリティ支援が必要不可欠であるが，本稿で紹介したように，これを実現するための各要素技術はいまだ課題を多くかかえている．これはひとえに，OS に対する要請が長年にわたって大きく変化しなかったこともあり，特に実利用される OS では，その基本原理が半世紀近くほとんど変わってこなかったことに由来すると考えられる．しかしながら，今後を見ると，新しいコンピューティング環境の急速な普及に合わせて，OS に対する要請もこれまでになく大きく変化することが想定され，特に情報セキュリティの重要性と，OS が自然に備える TCB や参照モニタへの適合性に鑑みたときに，本稿で取り上げた諸技術の残存課題が解決され，実社会であたりまえのように利用されるよう普及することが期待される．本稿が近い将来この仕事にあたる方々の足がかりとなり，わが国の情報セキュリティ研究の発展を支える一助となれば幸いである．

謝辞　貴重なご助言をいただいた編集委員と匿名査読者の方々に感謝いたします．

参考文献

[1] US Department of Defense: Trusted Computer Systems Evaluation Criteria, Technical Report CSC-STD-001-83, DoD Computer Security Center, Fort Meade, MD (1983).

[2] Anderson, J.P.: Computer Security Technology Planning Study, Technical Report Vols. I and II, USAF Electronic Systems Div., Bedford, Mass (1972).

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *Proc. 19th ACM Symposium on Operating Systems Principles*, *SOSP'03*, pp.164–177, New York, NY, USA, ACM (2003).

[4] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: The Linux virtual machine monitor, *Proc. Linux Symposium*, Vol.1, pp.225–230 (2007).

[5] Waldspurger, C.A.: Memory resource management in VMware ESX server, *SIGOPS Oper. Syst. Rev.*, Vol.36, pp.181–194 (2002).

[6] Garfinkel, T. and Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection, *Proc. 10th Annual Network and Distributed System Security Symposium* (*NDSS*), Vol.1, ISOC, pp.253–285 (2003).

[7] Volatile Systems: The Volatility Framework: Volatile memory artifact extraction utility framework, (2006), available from ⟨https://www.volatilesystems.com/default/volatility⟩.

[8] Payne, B.D., Carbone, M., Sharif, M. and Lee, W.: Lares: An Architecture for Secure Active Monitoring Using Virtualization, *IEEE Symposium on Security and Privacy*, Vol.0, pp.233–247 (2008).

[9] Quynh, N.A. and Suzaki, K.: Xenprobes, a lightweight user-space probing framework for Xen virtual machine, *2007 USENIX Annual Technical Conference on Proc. USENIX Annual Technical Conference*, pp.2:1–2:14, Berkeley, CA, USA, USENIX Association (2007).

[10] Breslau, L., Chase, C., Duffield, N., Fenner, B., Mao, Y. and Sen, S.: VMScope: A virtual multicast VPN performance monitor, *Proc. 2006 SIGCOMM Workshop on Internet Network Management*, *INM'06*, pp.59–64, New York, NY, USA, ACM (2006).

[11] Chen, P.M. and Noble, B.D.: When Virtual Is Better Than Real, *Proc. 8th Workshop on Hot Topics in Operating Systems*, pp.133–138, Washington, DC, USA, IEEE Computer Society (2001).

[12] Sharif, M.I., Lee, W., Cui, W. and Lanzi, A.: Secure in-VM monitoring using hardware virtualization, *Proc. 16th ACM Conference on Computer and Communications Security*, *CCS'09*, pp.477–487, New York, NY, USA, ACM (2009).

[13] Yin, H., Song, D., Egele, M., Kruegel, C. and Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis, *Proc. 14th ACM Conference on Computer and Communications Security*, *CCS'07*, pp.116–127, New York, NY, USA, ACM (2007).

[14] inc., G.: Inside Google Desktop, available from ⟨http://googledesktop.blogspot.com⟩.

[15] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: Malware analysis via hardware virtualization extensions, *Proc. 15th ACM Conference on Computer and Communications Security*, *CCS'08*, pp.51–62, New York, NY, USA, ACM (2008).

[16] Pék, G., Bencsáth, B. and Buttyán, L.: nEther: In-guest detection of out-of-the-guest malware analyzers, *Proc. 4th European Workshop on System Security*, *EUROSEC'11*, pp.3:1–3:6, New York, NY, USA, ACM (2011).

[17] King, S.T., Chen, P.M., Wang, Y.-M., Verbowski, C., Wang, H.J. and Lorch, J.R.: SubVirt: Implementing malware with virtual machines, *IEEE Symposium on Security and Privacy*, Vol.0, pp.314–327 (2006).

[18] Rutkowska, J.: Subverting VistaTM Kernel For Fun And Profit, *Black Hat Briefings* (2006).

[19] Ristenpart, T., Tromer, E., Shacham, H. and Savage, S.: Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds, *Proc. 16th ACM Conference on Computer and Communications Security*, *CCS'09*, pp.199–212, New York, NY, USA, ACM (2009).

[20] Chen, X., Andersen, J., Mao, Z., Bailey, M. and Nazario, J.: Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware, *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, 2008, DSN 2008*, pp.177–186, IEEE (2008).

[21] Xiong, X., Tian, D. and Liu, P.: Practical protection of kernel integrity for commodity OS from untrusted extensions, *Proc. 18th Annual Network and Distributed System Security Symposium* (*NDSS*), ISOC (2011).

[22] Payne, B., Carbone, M. and Lee, W.: Secure and flexible monitoring of virtual machines, *Annual Computer Security Applications Conference 2007*, *ACSAC'07*, pp.385–397, IEEE Computer Society (2007).

[23] Sailer, R., Valdez, E., Jaeger, T., Perez, R., Van Doorn, L., Griffin, J., Berger, S., Doorn, L., Linwood, J. and Berger, G.: sHype: Secure Hypervisor Approach to Trusted Virtualized Systems, *IBM Research Report RC23511*, IBM Research Division (2005).

[24] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: A thin hypervisor for enforcing i/o device security, *Proc. 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, *VEE'09*, pp.121–130, New York, NY, USA, ACM (2009).

[25] Corporation, I.: Trusted Execution Technology (Intel TXT), available from ⟨http://download.intel.com/technology/security/downloads/315168.pdf⟩.

[26] Berger, S., Cáceres, R., Goldman, K.A., Perez, R., Sailer, R. and van Doorn, L.: vTPM: Virtualizing the trusted platform module, *Proc. 15th Conference on USENIX Security Symposium - Volume 15*, *SSYM'06*, pp.305–320, Berkeley, CA, USA, USENIX Association (2006).

[27] Sailer, R., Zhang, X., Jaeger, T. and van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture, *Proc. 13th Conference on USENIX Security Symposium - Volume 13*, *SSYM'04*, p.16, Berkeley, CA, USA, USENIX Association (2004).

[28] Corporation, M.: BitLocker Drive Encryption, available from ⟨http://windows.microsoft.com/en-US/windows7/products/features/bitlocker⟩.

[29] Kauer, B.: OSLO: Improving the security of trusted computing, *Proc. 16th USENIX Security Symposium - Volume 16*, *SSYM'07*, pp.16:1–16:9, Berkeley, CA, USA, USENIX Association (2007).

[30] Azab, A., Ning, P., Sezer, E. and Zhang, X.: HIMA: A hypervisor-based integrity measurement agent, *Annual Computer Security Applications Conference 2009*, *ACSAC'09*, pp.461–470, IEEE (2009).

[31] Azab, A.M., Ning, P., Wang, Z., Jiang, X., Zhang, X. and Skalsky, N.C.: HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity, *Proc. 17th ACM Conference on Computer and Communications Security*, *CCS'10*, pp.38–49, New York, NY, USA, ACM (2010).

[32] Litty, L., Lagar-Cavilla, H.A. and Lie, D.: Hypervisor support for identifying covertly executing binaries, *Proc. 17th Conference on USENIX Security Symposium - Volume 17*, *SSYM'08*, pp.243–258, Berkeley, CA, USA, USENIX Association (2008).

[33] Nipkow, T., Paulson, L.C. and Wenzel, M.: *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, LNCS, Vol.2283, Springer (2002).

[34] Bertot, Y. and Castéran, P.: *Interactive Theorem Proving and Program Development, Coq'Art: the Calculus of Inductive Constructions*, Springer-Verlag (2004).

[35] Bevier, W.R.: Kit: A Study in Operating System Verification, *IEEE Trans. Softw. Eng.*, Vol.15, No.11, pp.1382–1396 (1989).

[36] Boyer, R.S. and Moore, J.S.: *A computational logic handbook*, Academic Press Professional, Inc., San Diego, CA, USA (1988).

[37] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H. and Winwood, S.: seL4: Formal verification of an OS kernel, *Proc. SOSP'09*, pp.207–220 (2009).

[38] Jones, S.P. (Ed.): *Haskell 98 Language and Libraries – The Revised Report*, Cambridge University Press, Cambridge, England (2003).

[39] Ball, T., Majumdar, R., Millstein, T. and Rajamani, S.K.: Automatic predicate abstraction of C programs, *Proc. PLDI'01*, pp.203–213 (2001).

[40] Henzinger, T.A., Jhala, R., Majumdar, R. and Sutre, G.: Lazy abstraction, *Proc. POPL'02*, pp.58–70 (2002).

[41] Clarke, Jr., E.M., Grumberg, O. and Peled, D.A.: *Model checking*, MIT Press, Cambridge, MA, USA (1999).
[42] Ball, T., Bounimova, E., Kumar, R. and Levin, V.: SLAM2: Static driver verification with under 4% false alarms, *Proc. FMCAD'10*, pp.35–42 (2010).
[43] Ball, T., Cook, B., Levin, V. and Rajamani, S.K.: SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft, *Proc. IFM'04*, pp.1–20 (2004).
[44] Yang, J. and Hawblitzel, C.: Safe to the Last Instruction: Automated Verification of a Type-Safe Operating System, *Proc. PLDI'10* (2010).
[45] Barnett, M., Chang, B.-Y.E., DeLine, R., 0002, B.J. and Leino, K.R.M.: Boogie: A Modular Reusable Verifier for Object-Oriented Programs, *Proc. FMCO'05*, pp.364–387 (2005).
[46] De Moura, L. and Bjørner, N.: Z3: An efficient SMT solver, *Proc. TACAS'08*, pp.337–340 (2008).
[47] Bershad, B.N., Savage, S., Pardyak, P., Sirer, E.G., Fiuczynski, M., Becker, D., Eggers, S. and Chambers, C.: Extensibility, Safety and Performance in the SPIN Operating System, *Proc. SOSP'95*, pp.267–284 (1995).
[48] Nelson, G. (Ed.): *System Programming in Modula-3*, Prentice Hall (1991).
[49] Hunt, G.C., Larus, J.R., Abadi, M., Aiken, M., Barham, P., Fähndrich, M., Hawblitzel, C., Hodson, O., Levi, S., Murphy, N., Steensgaard, B., Tarditi, D. and Zill, T.W.B.: An Overview of the Singularity Project, Technical Report MSR-TR-2005-135, Microsoft Corporation (2005).
[50] Hejlsberg, A., Torgersen, M., Wiltamuth, S. and Golde, P.: *The C# Programming Language*, 3rd edition, Addison-Wesley Professional (2008).
[51] Morrisett, G., Walker, D., Crary, K. and Glew, N.: From System F to Typed Assembly Language, *ACM Trans. Prog. Lang. Syst.*, Vol.21, No.3, pp.528–569 (1999).
[52] Maeda, T. and Yonezawa, A.: Writing an OS Kernel in a Strictly and Statically Typed Language, *Formal to Practical Security*, LNCS 5458, pp.181–197 (2009).
[53] Maeda, T. and Yonezawa, A.: Writing practical memory management code with a strictly typed assembly language, *Proc. SPACE'06* (2006).
[54] Maeda, T. and Yonezawa, A.: Typed assembly language for implementing OS kernels in SMP/multi-core environments with interrupts, *Proc. SSV'10* (2010).
[55] Adve, S.V. and Gharachorloo, K.: Shared Memory Consistency Models: A Tutorial, *IEEE Comp.*, Vol.29, No.12, pp.66–76 (1996).
[56] Boudol, G. and Petri, G.: Relaxed memory models: an operational approach, *Proc. POPL'09*, pp.392–403 (2009).
[57] Atig, M.F., Bouajjani, A., Burckhardt, S. and Musuvathi, M.: On the verification problem for weak memory models, *Proc. POPL'10*, pp.7–18, New York, NY, USA, ACM (2010).
[58] International Organization for Standardization (ISO): Information Technology – Open Systems Interconnection – Security Frameworks in Open Systems – Part 3: Access Control, Technical report, ISO/IEC 10181-3 (1996).
[59] Bell, D. and LaPadula, L.: Secure computer systems: Mathematical foundations and model, Technical Report M74-244, The MITRE Corporation, Bedford, MA (1973).
[60] Biba, K.J.: Integrity Considerations for Secure Computer Systems, Technical Report MTR-3153, The MITRE Corporation, Bedford, MA (1977).
[61] Clark, D. and Wilson, D.: A comparison of commercial and military computer security models, *Proc. IEEE Computer Society Symposium on Security and Privacy*, pp.184–194, IEEE Computer Society (1987).
[62] Brewer, D. and Nash, M.: THE CHINESE WALL SECURITY POLICY, *Proc. IEEE Computer Society Symposium on Security and Privacy*, p.206, IEEE Computer Society (1989).
[63] Ferraiolo, D., Cugini, J. and Kuhn, D.: Role-based access control (RBAC): Features and motivations, *Proc. 11th Annual Computer Security Application Conference*, pp.241–248 (1995).
[64] Ferraiolo, D. and Kuhn, R.: Role-Based Access Control, *15th NIST-NCSC National Computer Security Conference* (1992).
[65] Sandhu, R., Coyne, E., Feinstein, H. and Youman, C.: Role-based access control models, *Computer*, Vol.29, No.2, pp.38–47 (1996).
[66] Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D. and Chandramouli, R.: Proposed NIST standard for role-based access control, *ACM Trans. Information and System Security* (*TISSEC*), Vol.4, No.3, pp.224–274 (2001).
[67] Sandhu, R. and Bhamidipati, V.: Role-based administration of user-role assignment: The URA97 model and its Oracle implementation, *Journal of Computer Security*, Vol.7, No.4, pp.317–342 (1999).
[68] Sandhu, R., Bhamidipati, V. and Munawer, Q.: The ARBAC97 model for role-based administration of roles, *ACM Trans. Information and System Security* (*TISSEC*), Vol.2, No.1, pp.105–135 (1999).
[69] Crampton, J. and Loizou, G.: Administrative scope: A foundation for role-based administrative models, *ACM Trans. Information and System Security* (*TISSEC*), Vol.6, No.2, pp.201–231 (2003).
[70] Jha, S., Li, N., Tripunitara, M., Wang, Q. and Winsborough, W.: Towards formal verification of role-based access control policies, *IEEE Trans. Dependable and Secure Computing*, pp.242–255 (2007).
[71] Thomas, R.K. and Sandhu, R.S.: Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Autorization Management, *Proc. IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI: Status and Prospects*, pp.166–181, London, UK, Chapman & Hall, Ltd. (1998).
[72] Becker, M., Fournet, C. and Gordon, A.: Design and Semantics of a Decentralized Authorization Language, *CSF '07: Proc. 20th IEEE Computer Security Foundations Symposium*, pp.3–15, Washington, DC, USA, IEEE Computer Society (2007).
[73] Halpern, J. and Weissman, V.: Using first-order logic to reason about policies, *ACM Trans. Information and System Security* (*TISSEC*), Vol.11, No.4, pp.1–41 (2008).
[74] Weissman, V. and Lagoze, C.: Towards a Policy Language for Humans and Computers, *Research and Advanced Technology for Digital Libraries*, Heery, R. and Lyon, L. (Eds.), Lecture Notes in Computer Science, Vol.3232, pp.513–525, Springer Berlin, Heidelberg (2004).
[75] Gofman, M., Luo, R., Solomon, A., Zhang, Y., Yang, P. and Stoller, S.: RBAC-PAT: A Policy Analysis Tool for Role Based Access Control, *Tools and Algorithms for the Construction and Analysis of Systems*, Kowalewski, S. and Philippou, A. (Eds.), Lecture Notes in Computer

Science, Vol.5505, pp.46–49, Springer Berlin, Heidelberg (2009).
[76] Guttman, J., Herzog, A., Ramsdell, J. and Skorupka, C.: Verifying information flow goals in security-enhanced Linux, *Journal of Computer Security*, Vol.13, No.1, pp.115–134 (2005).
[77] Saltzer, J. and Schroeder, M.: The protection of information in computer systems, *Proc. IEEE*, Issue 9, Vol.63, pp.1278–1308 (1975).
[78] Lipton, R.J. and Snyder, L.: A Linear Time Algorithm for Deciding Subject Security, *J. ACM*, Vol.24, pp.455–464 (1977).
[79] Watson, R.N.M., Anderson, J., Laurie, B. and Kennaway, K.: Capsicum: Practical capabilities for UNIX, *Proc. 19th USENIX Conference on Security*, *USENIX Security'10*, p.3, Berkeley, CA, USA, USENIX Association (2010).
[80] Spencer, R., Corporation, S.C., Smalley, S., Loscocco, P., Agency, N.S. and Andersen, M.H.D.: The Flask Security Architecture: System Support for Diverse Security Policies, *Proc. 8th Conference on USENIX Security Symposium - Volume 8*, *SSYM'99*, pp.123–139, USENIX Association (1999).
[81] Watson, R.N.M.: TrustedBSD: Adding Trusted Operating System Features to FreeBSD, *Proc. FREENIX Track: 2001 USENIX Annual Technical Conference*, pp.15–28, Berkeley, CA, USA, USENIX Association (2001).
[82] KaiGai, K.: Security Enhanced PostgreSQL (2006), available from ⟨http://code.google.com/p/sepgsql/⟩.
[83] Macmillan, K., Brindle, J., Mayer, F., Caplan, D., Tang, J. and Technology, T.: Design and implementation of the SELinux policy management server, *Proc. Security Enhanced Linux Symposium*, pp.1–6 (2006).

橋本 正樹　（正会員）

2010 年 3 月情報セキュリティ大学院大学情報セキュリティ研究科修了，博士（情報学）．同年 4 月より情報セキュリティ大学院大学情報セキュリティ研究科助教．電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．電子情報通信学会 ISS・情報通信システムセキュリティ研究会専門委員．

安藤 類央　（正会員）

2006 年慶應義塾大学大学院政策・メディア研究科後期博士課程修了（政策・メディア）．同年情報通信研究機構所属．ネットワークセキュリティ，仮想化システムセキュリティの研究に従事．

前田 俊行　（正会員）

1977 年 12 月生．2006 年東京大学大学院情報理工学系研究科博士課程修了．博士（情報理工学）．現在，東京大学大学院情報理工学研究科コンピュータ科学専攻助教．

田中 英彦　（名誉会員，フェロー）

1970 年東京大学大学院工学系研究科電気工学専門課程修了，工学博士．東京大学にて計算機アーキテクチャ，並列処理，人工知能，自然言語処理，分散処理，メディア処理等の教育・研究に従事．東京大学工学部教授，同大学院情報理工学系研究科長を経て，2004 年情報セキュリティ大学院大学情報セキュリティ研究科長・教授．情報処理学会名誉会員，人工知能学会論文賞，ACM SIGGRAPH'99 Impact Paper Award，人工知能学会功績賞，東京都科学技術功労者表彰，経済産業大臣表彰等受賞．情報・システム研究機構教育研究評議会評議員，IEEE Fellow，東京大学名誉教授．