2008－CSEC－41 (7)
2008／5／22

デバイスドライバのセキュリティ強化

藤澤 一樹，宮本 久仁男， 田中 英彦
情報セキュリティ大学院大学

論文要旨 一般的にDevice Driverは，特権モードで動作するためセキュリティ上の致命的な問題となることが知られており，近年，攻撃者はDevice Driverを狙う傾向がある．そこで本論文では，Device Driverに起因するセキュリティ上の問題点と，その原因を明らかにし，既存研究の限界を指摘する．その後，Device Driverに対するリファレンスモニタを導入し，これと連携してシステムをDevice Driverの脆弱性から保護する機構を提案する．同時に，この保護機構そのものを仮想化技術を用いて隔離するアーキテクチャを示す．提案手法の実装対象としてはx86アーキテクチャのLinuxとXenを選択し，その解析及び，提案アーキテクチャの具体的な設計，保護機構の部分的な実装を行う．

Security Enhancement of Device Driver

Kazuki Fujisawa, Kunio Miyamoto, Hidehiko Tanaka
INSTITUTE of INFORMATION SECURITY

Abstract Generally, device drivers are invoked on privileged mode, so it is known that device driver becomes the critical problem in the security, and attackers tend to aim the device driver's vulnerabilities in recent year. In this paper, vulnerabilities of the device driver's are surveyed, and limits of the related works are pointed out. Based on that, a protection system is proposed that cooperated with reference monitor to prevent these types of problems. Furthermore, new architecture that isolates this protection system using virtualization technology is introduced. Linux and Xen on the Intel x86 architecture is analyzed for the implementation of proposed system and some functions of this architecture is partly implemented.

1. はじめに

デバイスドライバはデバイスを制御し，抽象化したインタフェースをKernelに提供するためのソフトウェアである．このデバイスドライバは一般に特権モードで動作しているためバグが脆弱性であった場合，攻撃者が特権を取得することが可能となり，セキュリティ上の致命的な問題になることが知られている．

また，普及を続ける携帯電話や情報家電などの様々な機器はWindows EmbeddedやSymbian OS，組み込みLinuxなどの汎用的な組み込みOSを利用する傾向が強まっており，これらは個々の専用OSに比べて多くのデバイスドライバを含んでいる．特に組み込みOSは，ROMに書き込むため脆弱性の修正が困難という問題がある．情報漏えい対策により普及が始まったシンクライアントやTCO（Total Cost of Ownership）削減のサーバ仮想化に用いられるVMM（Virtual Machine Monitor またはHypervisor）においてもデバイスドライバを含んでいる．

さらに，近年ではOSのセキュリティパッチの適用が自動化されたことや，OSを安全にするSecure OSなどのSecurity Moduleによって攻撃者が不正に特権を取得することが困難になってきており，デバイスドライバを狙った攻撃が増える傾向にある．以上の点から今後，デバイスドライバに関するセキュリティ上の問題が増えていくと考えられる．しかしながら，デバイスドライバに関する研究はこれら問題への対策よりも可用性を高める研究が主流である．

そこで本稿では，デバイスドライバの脆弱性の調査／分析から，特定した脆弱性をもとに,デバイスドライバを狙った攻撃からシステムを保護する．その方法として，デバイスドライバに対するリファレンスモニタを導入し，強制アクセス制御と，リファレンスモニタと連携してデバイスドライバに対する攻撃を防ぐ保護機構を導入する．リファレンスモニタは，Kernelの脆弱性によって改ざんやバイパスをされてしまい機能しなくなる恐れがある．そのため，仮想化技術を用いたリファレンスモニタを保護するアーキテクチャ示す．

本研究は，特殊なハードウェアやソースコードレベルの大幅な変更をすることなくデバイスドライバに脆弱性が

存在してもシステムを安全に動作させることを目的とする．そのため，デバイスドライバそのものに大幅に手を加えることを避けるために，OS の構造に着目する．ゆえにOSS（Open Source Software）である Linux と Xen を対象として，x86 アーキテクチャを限定として，提案アーキテクチャの設計，さらにアーキテクチャの部分的な実装を行った．

本稿は，第 2 節で，デバイスドライバの問題点と対応方法を，第 3 節で関連研究について述べ，第 4 説にて提案アーキテクチャと Policy について説明する．最後に，まとめと今後の課題について述べる．

2. デバイスドライバ脆弱性

本節では，デバイスドライバの問題点を指摘し，デバイスドライバの脆弱性の原因と対応方法を示す．

2.1 デバイスドライバの問題点

一般的にデバイスドライバは特権モードで動作し，安全なメモリコピーなどができない C 言語で書かれる．またデバイスが複雑になり，デバイスドライバのコード量が増えたことがある．

根本的な原因としてデバイスドライバは，Kernel を拡張するプログラムでありながら，その Security に関してデバイスドライバの開発者に依存し，Kernel が何も干渉できないことが問題解決を難しくしている．

2.2 デバイスドライバの脆弱性の原因と保護方法

デバイスドライバの脆弱性に起因する問題と，脆弱性の原因を明らかにするために調査／分析を行った[16]．ここでは本研究で対応する分析結果のみ述べる．デバイスドライバの脆弱性の根本的な原因は上位 4 件で 82%に相当し，本研究は以下に示す 4 件の原因に対応することを目指す．また，これら 4 件の根本的な原因から，脆弱性への対応方法は簡単に導き出すことができる．脆弱性と対応方法を表 1 に示す．

入力値の取り扱い：
デバイスドライバは，systemcall を通じてユーザからデータを受け取る．このデータのサニタイズやサイズのチェックの失敗や行わないことによって問題を起こす．

権限の取り扱い：
デバイスドライバに記述しなければならない権限項目を忘れたことや，権限の設定ミスによって問題を起こす．

メモリの取り扱い：
デバイスドライバはカーネル空間で動作するため，確保したメモリを開放しないこと，自分以外の別のアドレスに書き込んでしまうこと，初期化せずにメモリを扱ったため，カーネルが使用したデータをデバイスに送ってしまうことによって問題を起こす．

バッファの取り扱い：
デバイスドライバに記述されたバッファのサイズを越えてデータを渡すことによって問題を起こす．

表 1　脆弱性と対応方法

脆弱性	対応方法
入力値の取り扱い	サニタイズ
権限の取り扱い	MAC（強制アクセス制御）
メモリの取り扱い　メモリリーク	アドレスの解放
メモリの取り扱い　データリーク	アドレスの初期化
メモリの取り扱い　範囲外へのアクセス	アクセス拒否
バッファの取り扱い	バッファサイズの強制

ただし，デバイスドライバの脆弱性の中には，原因不明なものがある．しかし，その原因を推測することができる．例えば Information Disclosure の場合，ユーザから情報が見える状態にあったという点から，権限の取り扱いかメモリ範囲の取り扱いが原因であると考えられる．そのため表 1 の方法によって，ほぼ全てのデバイスドライバの脆弱性に対応可能と言える．

3. 関連研究

デバイスドライバのバグからシステムを保護する研究は様々あり，これらは，バグの極小化，特権モード動作の局所化，特権モードからの排除，デバイスドライバの隔離，MAC（強制アクセス制御）と 5 つに分類することができる．

特権モードの局所化は，バグを発見し減らす方法である．SCAT （Source Code Analysis Tool） [1] や，DSL (Domain-Specific Language) [2]，VAULT[3]，ソフトウェアテストなどがある．SCAT は，静的にソースコードからバグを発見する方法であるが，バグを全て発見することは困難である．また，DSL は，専門家によって DSL で書かれたソースコードが正しいか検証することができるが，コストが高くなるため普及が難しい．安全な C 言語として開発された VAULT は，安全なメモリコピーなどを限定的にしか提供せず，デバイスドライバに適さない．ソフトウェアテストは静的に発見できないバグを発見することが可能であるが，全てのバグを発見することは難しい．特に極小化のアプローチは必ず人の目が必要という問題点がある．

次の特権モード動作の局所化は，特権モードだけでなく一部を非特権モードで動作させる方法である．Windows Vista の Kernel Mode Driver (以降：KMD) と User Mode

Driver（以降：UMD）[4]が挙げられる．この方法は，一部を非特権モードで動作させるが，特権モードにデバイスドライバを残す．

特権モードからの排除は，非特権モードでデバイスドライバを動作させる方法でmicrokernel[5]と，その発展系のExokernel[6]は完全に特権モードから排除しているが一般的なアプローチでない．Xen[7]，L4VM[8][9]はともに特権VMの問題があり，Terra[10]では，NGSCB[11]とUMDを用いる方法を提案しているが特殊なハードウェアが必要である．

デバイスドライバの隔離は，microkernel，Exokernel，Xenなどであればデバイスドライバを隔離しておりデバイスドライバが故障してもOSをクラッシュさせることはない．しかし，monolithic kernelではクラッシュする．このmonolithic kernelであってもOSをクラッシュさせないNooks[12]がある．しかしNooksの隔離は弱く，攻撃対策でない．

最後はMACによってI/Oアクセスを制限する方法である．SELinux（Security Enhanced Linux）[13]はリファレンスモニタの保護が完全でなく，sHype（Secure Hypervisor）[14]は特権VMからデバイスドライバを分離できない．Xenの特権ドメインにPLPを実現するXSM（Xen Security Module）[15]がある．このXSMを用いてXenの特権VMから，デバイスドライバを分離できるが，脆弱性からシステムを保護できない．

そこで本研究は，デバイスドライバに脆弱性が存在しても，システムを安全に運用すべく，デバイスドライバに対してリファレンスモニタと，リファレンスモニタと連携して動作しデバイスドライバに対する攻撃から防御する保護機構を導入する．また，リファレンスモニタを保護するためにXenを用いる．これらの既存研究とその問題及び，本研究の特徴を表2に示す．

表2 既存研究の問題

分類	名前	隔離	防御	問題
極小化	SCAT	ー	ー	完全な検出は困難
	DSL	ー	ー	普及が困難
	VAULT	ー	ー	限定的なType-safety
	ソフトウェアテスト	ー	ー	完全な検出は困難
局所化	KMDとUMD	△	×	特権モードに残す
排除	microkernel	○	×	一般的でない
	Exokernel	○	×	一般的でない
	Xen	○	×	特権VMの問題
	L4VM	○	×	特権VMの問題
	Terra	○	×	特殊なハードウェアが必要
隔離	Nooks	△	△	攻撃対策でない
MAC	SELinux	×	×	リファレンスモニタの保護
	sHype	○	×	特権VMの問題
	XSM	○	×	ー
	本研究	○	○	ー

4. 提案アーキテクチャと評価

本節では，提案アーキテクチャの概要と攻撃からの保護方法，構成とポリシー及び，評価について述べる．

4.1 提案アーキテクチャの概要

本研究では，デバイスドライバに対するリファレンスモニタと，リファレンスモニタと連携しデバイスドライバに対する攻撃からシステムを保護する機構（以降：保護機）を提案する．提案アーキテクチャの概要を図1に示す．

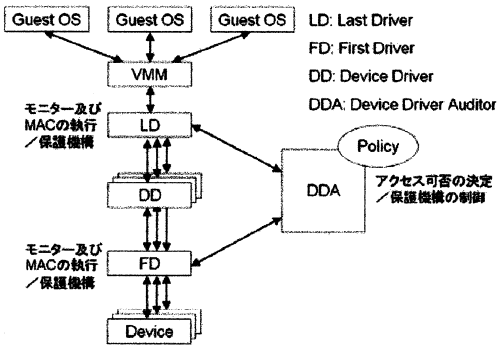

図1 提案アーキテクチャの概要

提案アーキテクチャはPolicyによるMACの執行を行うサブシステムであるFirst Driver（以降：FD），Last Driver（以降：LD）とセキュリティポリシーに従ってアクセス可否を決定するサブシステムであるデバイスドライバAuditor（以降：DDA）に分けて，より安全で柔軟なポリシーを扱えるリファレンスモニタを構築する．さらにMACの執行部とアクセス可否を決定するサブシステムに対して，デバイスドライバに対する攻撃を無効化する保護機構と，その制御部を備える．提案アーキテクチャは以下に示す方法で攻撃からシステムを保護し，アーキテクチャ自体を保護する．我々は，このFD／LD／DDAから構成される提案アーキテクチャをAegisと命名した．現状のAegisは，x86アーキテクチャのみに対応しXenを準仮想化で動作させ，Privileged Domain／Unprivileged DomainをLinuxとしてSELinuxを動作させた構成をとる．次に攻撃からの保護方法を示す．

4.2 攻撃からの保護

Aegisは，MACによって権限の取り扱いに対応し，保護機構によって，入力値，メモリ，バッファの取り扱いに対応する．また，FD／LD，デバイスドライバの改ざんやバイパスを防ぐためにDDAによる定期的に完全性を証明する．完全性の証明によってデバイスドライバを改変するようなLKM rootkitを検出することが可能である．さらに

リファレンスモニタの破壊，改ざん，バイパスから保護するため，Xen と CPU の Ring Protection を用いて防ぎ，起動時にメモリスペースを書き換えられることも想定し，DDA のリブートし，DDA が必ず最後に起動する特殊な起動方法をとる．ただし，Aegis は PLP（最小特権の原則）を実現しないため，Policy を保護できない．そこで PLP を実現するために SELinux を用いる．

Linux における systemcall によるアクセスは systemcall から始まり，systemcall 中に LSM（Linux Security Module）フック関数が存在し，その後，オペレーション関数によってデバイスドライバが実行される．また，systemcall からプロセスの権限を知ることができる．そこで systemcall による動作は図 2 のようになる．

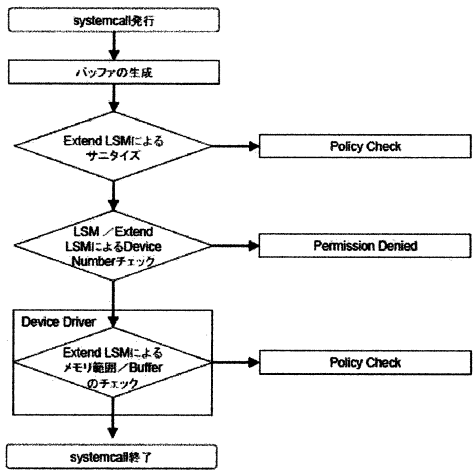

図 2 systemcall によるアクセスによる動作

Systemcall 発行後にカーネルバッファが作成される．このバッファに対して拡張 LSM フック関数（Extend LSM）によって受け取り，ポリシーに従ってサニタイズを実行する．その後，LSM フック関数によって SELinux 同様に権限のチェック及び，デバイスを識別するデバイス Number を基に MAC を執行する．そしてデバイスドライバロード時に取得しておいたデバイスドライバのメモリ範囲をもとにデバイスドライバのメモリ範囲のチェック，バッファのチェックを行う．この 2 つ問題に対して，デバイスを定義する構造体にバッファ専用の変数を用意し，その変数を操作する関数を Extend LSM として準備する．その関数の仕様をデバイスドライバの開発者に強制させることで変数操作用 Extend LSM によって範囲のチェックとバッファサイズを強制することで，メモリの取り扱いとバッファの取り扱いに対して対応する．

また，割り込みによるアクセスは，NIC にパケットが到達し，NIC が APIC に割り込み信号を送り，APIC が CPU に割り込み信号を通知する．そして CPU が割り込みハンドラを実行し，割り込みハンドラガ NIC のデバイスドライバの割り込み関数を実行する．割り込み関数実行直後にデバイスドライバに書かれた NIC の受信処理が実行され，バッファが生成されプロトコル依存処理が行われる．そこで，動作は，図 3 ようになる．ハードウェア割り込み発行後，Linux の割り込みハンドラが実行される．この割り込みハンドラのデバイスドライバの割り込み処理が呼び出される前に Extend LSM によって権限のチェックを行う．SELinux 等で使われている LSM フック関数は割り込みハンドラに存在しない．そして，inside 同様にデバイスドライバに対してメモリの取り扱い，バッファの取り扱いに対して Extend LSM 関数にて対応し，Web Server 等の Server Application によってリッスン状態であった場合，バッファが生成される．そのバッファに対してサニタイズを行う． Linux には，Firewall として iptables があるが，それはデバイスドライバの受信処理後のプロトコル依存処理を行う ip_rcv()関数中でフックしており，本手法は，それよりも早い段階でアクセス可否の決定ができる．

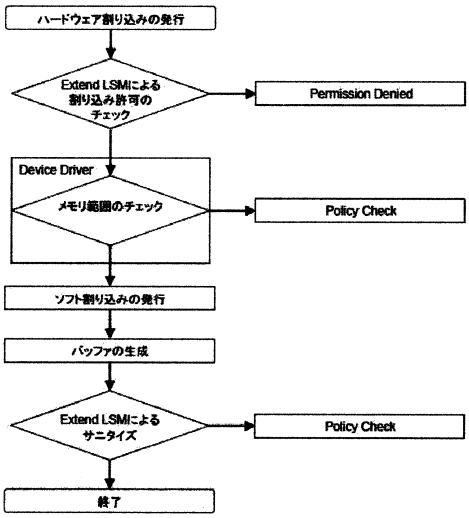

図 3 割り込みによるアクセスによる動作

4.3 Aegis の構成

提案アーキテクチャでは，最終的にデバイスを操作する 48 種類のオペレーション関数（file_operations，disk_operations，socket 関連のシステムコール）の中で，LSM が定義されていないものと，read／write systemcall，割り込み（do_IRQ）関数，デバイスドライバに用いられるバッファを操作する新しい関数を Extend LSM として実装を行う．また，Xen の hypercall を拡張し，提案アー

キテクチャの操作，及びevent channelの拡張を行う．

以上のことにより提案アーキテクチャは，図4のようになる．既に述べたようにPrivileged Domain, Unprivileged DomainをSELinuxを用いたLinuxとし，準仮想化で動作させたXenである．

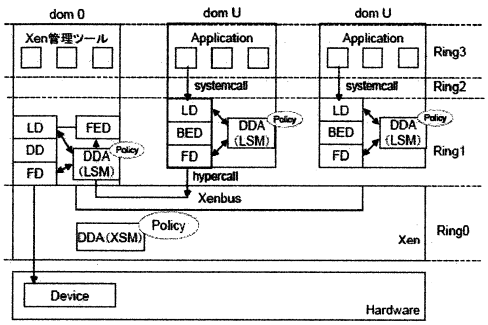

図4 提案アーキテクチャ

DDAをLSM／Extend LSM／XSMとして実装を行う．そのためFD／LDは，LSM／Extend LSMフック関数として実装することになる．LSM／Extend LSM／XSMを用いることで拡張が容易になり，Linux及びXenを大幅に修正する必要がない．

XSMのDDAはDefault Policyを持ち，LSMのDDAの完全性の証明を行う．LSMのDDAは，FD／LD，デバイスドライバの完全性の証明と，アクセス可否の決定及び，保護機構の動作を決める． 次にポリシーについて述べる．

4.4 Policy

Aegisにおける Policyは，MACと保護機構の動作，完全性の証明を行う頻度の設定ファイルで，Device Name Based Policy（以降DNBP）という独自のポリシーを採用している．DNBPは，Device名，ユーザ名，アクセス権，保護機構の動作を書くだけであり，理解し易いと考え多くのポリシー記述で採用されているXMLを採用していない．また，特徴としてはポリシー展開について考慮している．

想定しているポリシーを図5に示す．ポリシーの読み方は以下のようになっている．

1〜7行目： defaultのポリシーの定義を行う．完全性の証明を4時間ごとに行う．次にMACの定義を行い1／0でアクセス可否を決定する．ここでは全てアクセス拒否している．次に入力値の取り扱いの動作を1／0で保護機構の動作のON／OFFを示しており，ここでは，保護機構を動作させる．同様に，メモリの取り扱い，バッファの取り扱いを定義している．

8〜15行目： ネットワークデバイスであるeth0に対する設定を行う．defaultのポリシーをextendすることで異なる部分のみ記述する．完全性の証明を1日ごと行い，root に対して全てのアクセスを許可し，fujisawaに対しては，readのみ許可している．

```
class default {
    check = 4h;
    mac = {read, write, interrupt} = {0, 0, 0};
    input = 1;
    mem  = {init, clear, out} = {1, 1, 1};
    buff = 1;
}
# eth0 に対するポリシー
class eth0 extend default {
    check = 1d;
    mac{
        root        = {1, 1, 1};
        fujisawa    = {1, 0, 0};
    }
}
```

図5 想定しているポリシー

この想定しているポリシーをSELinux同様にPolicy CompilerによってAegis／SELinux／XSM用に相互変換を行い，必要であればXMLでの出力も行う．これによって，一つのPolicyによって，LSM，Extend LSM，XSMを操作でき，PLPの実現も容易となる．

4.5 実装と評価

実装：

提案アーキテクチャにおける実装は，Linux Kernel 2.6.18.8 を基に，デバイスを表す構造体の拡張及び，Extend LSMの実装を行った．

評価：

提案アーキテクチャは，I/Oに対するアクセスに対して，様々な動作を行うため，非常に大きくパフォーマンスが低下すると考えられる．そこでwrite systemcallに対するExtend LSMの計測を行った．通常のLinuxとExtend LSMを付加したLinuxに対して，2KBのデータを読み込み，書き込み時間の計測を行った．通常のLinuxに比べ2.56%程度増加するが，大幅な増加はなかった．この理由として，既にメモリ上に展開されているデータを扱うためではないかと考えられる．ただし，ファイルサイズの大きい場合や他のsystemcallによる計測を行っていないため一概には言えないが，この結果により提案アーキテクチャ

は，大きなパフォーマンスの低下を起こさず，ほとんど全ての脆弱性に対応可能である．

5. まとめと今後の課題

本稿では，デバイスドライバの危険性を指摘し，広範囲な既存研究の調査から，Aegis を提案した．この提案アーキテクチャの設計と Policy について述べた．今後，完全仮想化に対応することができれば，Xen の目指す，全てのサーバ，デスクトップ，ラップトップ，携帯電話に VMM が使われる Ubiquitous Virtualization 構想 によって，ほぼ全ての環境におけるデバイスドライバの脆弱性からシステムの保護が可能となる．しかしながら，以下の課題を残す．

- 残りの実装と評価／パフォーマンス低下への対策
- サニタイズのシグネチャ方式の導入
- メモリ範囲の限界
- Aegis／SELinux／XSM ポリシーの変換機構の導入
- 完全仮想化／x86 以外のアーキテクチャ／TPM への対応

本稿では， Kernel にチェック機能を持たせることで，殆どの脆弱性からシステムを保護できることを示した．

参考文献

[1] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, Benjamin Chelf, ``Bugs as deviant behavior: a general approach to inferring errors in systems code", In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, 2001.

[2] Christopher L. Conway, Stephen A. Edwards, ``NDL: A Domain-Specific Language for Device Drivers", In Proceedings of Languages, Compilers, and Tools for Embedded Systems (LCTES), June 2004

[3] The Microsoft Research, ``VAULT"
http://research.microsoft.com/vault/

[4] Microsoft ``Windows Driver Foundation"
http://www.microsoft.com/japan/whdc/driver/wdf/default.mspx

[5] The L4 microkernel family,
http://os.inf.tu-dresden.de/L4/

[6] Dawson R. Engler, M. Frans Kaashoek and James O'Toole Jr., ``Exokernel: an operating system architecture for application-level resource management", The 15th ACM Symposium on Operating Systems Principles, Copper Mountain Resort, Colorado, December 1995, pages 251-266.

[7] Keir Fraser, Steven Hand, Rolf Neugebauer, Ian Pratt, Andrew Warfield, Mark Williamson, ``Safe Hardware Access with the Xen Virtual Machine Monitor", OASIS ASPLOS 2004 workshop.

[8] Hidenari Koshimae, Yuki Kinebuchi, Shuichi Oikawa, Tatsuo Nakajima, ``Using a Processor Emulator on a microKernel-based Operating System ". The 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2006)
Sydney, Australia.

[9] Hidenari Koshimae, Yuki Kinebuchi, Shuichi Oikawa, Tatsuo Nakajima, ``Using a Processor Emulator on a microKernel-based Operating System ". The 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2006)
Sydney, Australia.

[10] Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, Stefan G"otz, ``Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines", Proceedings of the 6th Symposium on Operating Systems Design and Implementation, Dec. 2004.

[11] Microsoft, ``Next-Generation Secure Computing Base"
http://www.microsoft.com/resources/ngscb/default.mspx

[12] Michael Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy, ``Recovering Device Drivers" ACM Trans. on Computer Systems 24(4), November 2006.

[13] Stephen D. Smalley, ``NSA SecurityEnhanced Linux", Ottawa Linux Symposium BOF 2006.

[14] Ray Spencer, Peter Loscocco, Stephen Smalley, Mike Hibler, David Anderson, and Jay Lepreau, ``The Flask Security Architecture: System Support for Diverse Security Policies", In Proceedings of the 8th conference on USENIX Security Symposium, pp.123-139, August 1999.

[14] IBM Research, ``sHype: Secure Hypervisor",
http://www.research.ibm.com/secure_systems_department/projects/hypervisor/, 2005.

[15] George Coker, ``Xen Security Module (XSM) ", Xen Summit Fall 2006.

[16] 藤澤 一樹, 大久保 隆夫, 田中 英彦, ``デバイスドライバの脆弱性調査と分析", 情報処理学会, コンピュータセキュリティシンポジウム(CSS)2007, 11 月 2007 年.