Use of Role Based Access Control for Security-purpose Hypervisors

Manabu Hirano
Department of Information and Computer Engineering
Toyota National College of Technology
Toyota, Japan
hirano@toyota-ct.ac.jp

David W Chadwick
School of Computing
University of Kent
Canterbury, United Kingdom
D.W.Chadwick@kent.ac.uk

Suguru Yamaguchi
Graduate School of Information Science
Nara Institute of Science and Technology
Ikoma, Japan
suguru@is.naist.jp

Abstract— This paper shows the design and implementation of a Role Based Access Control (RBAC) mechanism for securing a hypervisor called BitVisor. BitVisor is a small hypervisor that provides security functions like encryption services for I/O devices in its hypervisor-layer. BitVisor enforces security functions without the help of guest OSs, but it only supports a static configuration file for machine set up. Consequently, we employ the RBAC system called PERMIS, a proven implementation of an RBAC policy decision engine and credential validation service, in order to provide dynamic configuration control. By using PERMIS, we can write finer grained authorization policies and can dynamically update them for the security-purpose hypervisor.

Security-purpose hypervisor; Role Based Access Control (RBAC); Authorization policies; Virtual Machine Monitors

I. INTRODUCTION

BitVisor is a security-purpose hypervisor that has been developed since 2008 [1][2][3][4][5]. The source code of BitVisor is available from http://www.bitvisor.org. BitVisor was initially developed by a research and development project initiated by National Information Security Center (NISC) of Japan and executed by several universities and companies. The purpose of the project is to prevent information leakages from desktop computers and laptop computers in governmental and corporate organizations. Many information leakage cases are caused by unauthorized use of unencrypted USB thumb drives, theft of laptop computers with unencrypted internal storage, etc. Fig.1 shows the architecture of BitVisor. BitVisor is a small hypervisor that is designed to enhance the security of computing systems by providing data encryption and decryption services for both storage media and network connections. BitVisor can enhance the security of a computer that uses any OS (e.g. Windows, Linux, FreeBSD, etc.) because its security functions work on a hypervisor-layer. BitVisor is aimed at preventing information leakage with minimal overhead. The basic features of BitVisor are the following [5]:

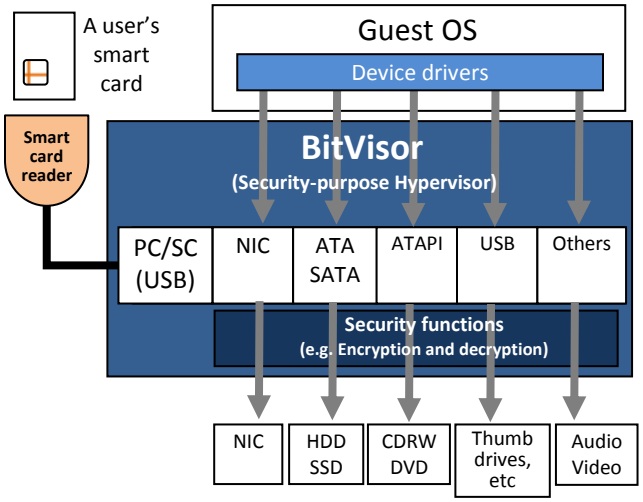

Figure 1. Security architecture of BitVisor.

- Boot up login authentication by passwords or PKI-based smart cards.
- Data encryption for HDD, SSD, USB storages, CD-RW, and other storage media. An encryption key is stored in each user's smart card by the organization's administrator. The smart card is protected by the user's PIN number.
- BitVisor provides a Virtual Private Network (VPN) client function by using its built-in IPsec and IKEv1 module. Once BitVisor is installed, BitVisor is transparent to end users (i.e. VPN connections can be achieved without the help of any client application on the guest OS.).
- Providing surveillance points of I/O devices (e.g. storage, network, etc.) in the hypervisor layer.

The BitVisor's security model assumes that a computer can be used by many different users, but only by one user at once and that each user is identified to the computer by inserting his or her smartcard (or his or her password). The OS is then booted up for this user. The current implementation of

BitVisor has no "switch user" facility available, only shutdown and re-boot.

II. Problems of Current BitVisor

BitVisor has a static configuration file. The configuration file can specify which VPN gateway the computer has to connect to, which HDD or SSD devices have to be encrypted, and whether the use of USB thumb drives is permitted or not, etc. The problems of the current BitVisor are the follows: (1) the configuration file of each computer is not written based on the user's identity or roles, but is the same for all users of the machine. Therefore, we cannot use any RBAC policy for its configuration. Also, (2) the current configuration file is installed once at the set up time of each machine by an administrator. Therefore it is difficult to change the configuration of each machine after distributing the pre-setup machines in the organization. In addition, (3) we can not specify detailed conditions in the current syntax of BitVisor's configurations. For example, it cannot specify when the user can use the machine or network.

To solve the above problems, this paper shows the design and implementation of an integrated combination of an RBAC policy decision engine (PERMIS) and a security-purpose hypervisor (BitVisor). BitVisor already has its X.509 PKI-based authentication function. By introducing PERMIS's X.509 PMI-based RBAC authorization function onto BitVisor, administrators will be able to control its security functions in a more flexible manner. PERMIS policies are fine grained, and can be dynamically updated. We show some useful examples of authorization policies that can be used for security-purpose hypervisors. This paper's contribution is to show a novel implementation of role-based access control mechanism for security-purpose hypervisors.

III. Policy Decision Mechanisms of PERMIS

This section presents a brief overview of PERMIS's authorization mechanisms. There are two types of policy in PERMIS [6]. The first policy is for the Credential Validation Service (CVS), which controls the user-role assignments. The second policy is for the Policy Decision Point (PDP) which controls the role-permission assignments. Fig. 2 shows the relationship between the NIST RBAC model [7] and PERMIS. PERMIS also can handle SOD (Separation of duty) constraints [8] and role hierarchies. All of these policies can be handled by the PERMIS policy engine.

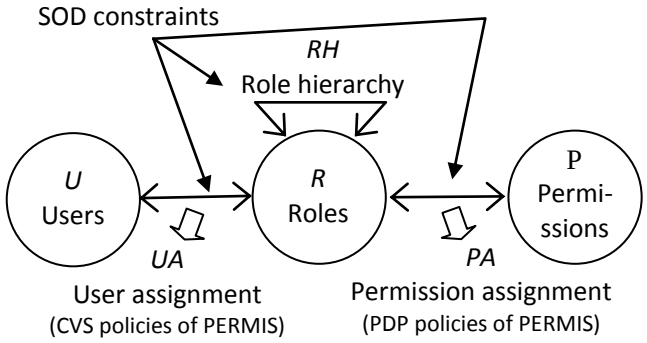

Figure 2. Relationship between NIST RBAC model and PERMIS.

A. Credential Validation Service (CVS) Policies

Table 1 shows an example policy of the Credential Validation Service (CVS) in the new system. This policy specifies who can assign which roles to whom, under which constraints. BitVisor uses two roles, permisRole and securityLevel. The permisRole is used for assigning each user's role. The securityLevel is used for distinguishing the security level of each computer. In Table 1, we define the same issuer (Source of Authority – SOA) for these two types of role. This policy permits operations of user assignments for both the users and the computers belonging to the specified subject domain, O=TNCT, L=Toyota, ST=Aichi, C=JP by the SOA who has the subject distinguished name (DN) of CN=SOA, O=TNCT, L=Toyota, ST=Aichi, C=JP. If we were to have different SOAs for users and computers, then we would use different issuers for each role type. Fig. 3 shows an actual CVS policy in the system. We defined three roles for users: permisRoles of Manager, Employee, and Trainee. We also defined five (integer) roles for computers: securityLevel of 1 through to 5, where 1 is the lowest security level and 5 is the highest. Alternatively we could have defined role hierarchy policies in the PERMIS's policy syntax for string values e.g. securityLevel 2 is superior to securityLevel 1 etc.

Table 1. CVS policies for user-role assignment.

Issuer	Role	Subject domain	Constraints
CN=SOA, O=TNCT, L=Toyota, ST=Aichi, C=JP	permisRole (Users' roles)	O=TNCT, L=Toyota, ST=Aichi, C=JP	Time constraints and delegation constraints
CN=SOA, O=TNCT, L=Toyota, ST=Aichi, C=JP	securityLevel (Computers' roles)	O=TNCT, L=Toyota, ST=Aichi, C=JP	Time constraints and delegation constraints

```
<SubjectPolicy>
  <SubjectDomainSpec ID="SubjectDomain">
    <Include LDAPDN="o=TNCT,l=Toyota,st=Aichi,c=JP" />
  </SubjectDomainSpec>
</SubjectPolicy>
<SOAPolicy>
  <SOASpec ID="TheSOA"
      LDAPDN="cn=SOA,o=TNCT,l=Toyota,st=Aichi,c=JP" />
</SOAPolicy>
<RoleAssignmentPolicy>
  <RoleAssignment>
    <SOA ID="TheSOA" />
    <SubjectDomain ID="SubjectDomain" />
    <RoleList>
      <Role Type="permisRole" Value="Manager" />
      <Role Type="permisRole" Value="Employee" />
      <Role Type="permisRole" Value="Trainee" />
      <Role Type="securityLevel" Value="1" />
      <Role Type="securityLevel" Value="2" />
      <Role Type="securityLevel" Value="3" />
      <Role Type="securityLevel" Value="4" />
      <Role Type="securityLevel" Value="5" />
    </RoleList>
    <Delegate Depth="0" />
  </RoleAssignment>
</RoleAssignmentPolicy>
```

Figure 3. A part of a CVS policy for the system.

```
<TargetPolicy>
 <TargetDomainSpec ID="USBdevicesTargetDomain">
     <Include LDAPDN="cn=USBdevices,o=TNCT,l=Toyota,st=Aichi,c=JP" />
 </TargetDomainSpec>
</TargetPolicy>
<ActionPolicy>
    <Action ID="diskIO" Name="DiskIO">
        <Argument Name="encryption" Type="Boolean" />
        <TargetDomain ID="USBdevicesTargetDomain" />
    </Action>
</ActionPolicy>
<TargetAccessPolicy>
 <TargetAccess>
  <TargetList>
    <Target><TargetDomain ID="USBdevicesTargetDomain" />
    <AllowedAction ID="diskIO" /></Target>
  </TargetList>
  <RoleList><Role Type="permisRole" Value="Employee" /></RoleList>
  <IF>
   <AND>
    <AND>
     <GE><Environment Parameter="securityLevel" Type="Integer" />
          <Constant Value="1" Type="Integer" /></GE>
     <LE><Environment Parameter="securityLevel" Type="Integer"/>
          <Constant Value="4" Type="Integer" /></LE>
    </AND>
    <EQ><Arg Name="encryption" Type="Boolean" />
          <Constant Value="true" Type="Boolean" /></EQ>
   </AND>
  </IF>
 </TargetAccess>
 <TargetAccess>
  <TargetList>
   <Target><TargetDomain ID="USBdevicesTargetDomain" />
            <AllowedAction ID="diskIO" /></Target>
  </TargetList>
  <RoleList><Role Type="permisRole" Value="Manager" /></RoleList>
  <IF>
   <AND>
    <EQ><Environment Parameter="securityLevel" Type="Integer" />
         <Constant Value="5" Type="Integer" /></EQ>
    <EQ><Arg Name="encryption" Type="Boolean" />
          <Constant Value="true" Type="Boolean" /></EQ>
   </AND>
  </IF>
 </TargetAccess>
</TargetAccessPolicy>
```

Figure 4. A part of a permission assignment policy for the system.

B. PDP Policies

Fig. 4 shows a part of a role permission assignment policy in the new system. TargetPolicy defines a name and DN of a target resource for access control decisions. This example says the target resource is USB devices. Next, ActionPolicy defines the actions that have to be controlled. This example defines the USB devices' action named "DiskIO" with a Boolean parameter that specifies encryption or not. Finally, TargetAccessPolicy defines the permitted actions on targets for roles, along with any conditions. The first TargetAccess permits an Employee to perform IO operations on a USB devices' disk with the following condition: (1) The computer's securityLevel is between 1 and 4; and (2) the disk IO operations are executed with encryption. The next TargetAccess permits a Manager to perform IO operations on a USB devices' disk with the following conditions: (1) the computer's securityLevel is 5 and (2) the disk IO operations are executed with encryption. Any action that is not defined in this policy is automatically denied. For example, another role Trainee cannot use any USB devices even if he or she uses the encryption functions.

C. Role Management and Policy Elements for Security-purpose Hypervisors

This subsection describes the role management method for security-purpose hypervisors. We also describe the building blocks of PERMIS's policies. PERMIS's policies are basically constructed by elements of Issuers, Subjects, Actions, Targets, and Environments. We describe the relationship between these elements and the security-purpose hypervisor, BitVisor, as follows:

- Issuers are Sources of Authority (SOAs) that issue attributes (roles) to subjects, as digitally signed X.509 attribute certificates (ACs) [9].
- Subjects are the users using particular computers. A user's name (i.e. Subject DN) is retrieved from the user's X.509 public key certificate (PKC) which is stored on his or her smartcard when the administrator sets up the smart card before distributing it. In the new system, a user's roles are dynamically retrieved as X.509 attribute certificates (ACs) from LDAP servers after executing BitVisor's authentication function. By introducing this method, administrators can change a user's roles dynamically even if the smart card is already setup and distributed to the user.
- The computers are used by the users, but they are also Subjects in terms of role assignments. The computer's identifier (i.e. Subject DN) is stored in the computer's X.509 PKC by an administrator at setup time. Like users' roles, the computer's role is retrieved as an X.509 AC from an LDAP server. As a result, administrators can change a computer's role dynamically even if the computer is already setup and distributed to users. A computer's role expresses the computer's security level.
- Actions state what the user wants to do on the Targets. In BitVisor, Actions basically state input and output operations and optionally attachment operations of external devices. The input and output operations for storage devices and for NIC devices can take arguments for options such as encryption.
- Targets are the types of resource. In BitVisor, they are I/O devices including storage devices, NIC devices, etc. BitVisor supports the following I/O devices: storage devices (ATA, AHCI and AHCI_ATAPI), external optical discs (ATAPI), USB devices (USB), and network devices (NIC).
- In PERMIS's policy, Environment states (conditions on) environment variables like the current time and date, or the GPS location. In the prototype system, the Environment is used to state the required security level of the computer and whether encryption is required or not.

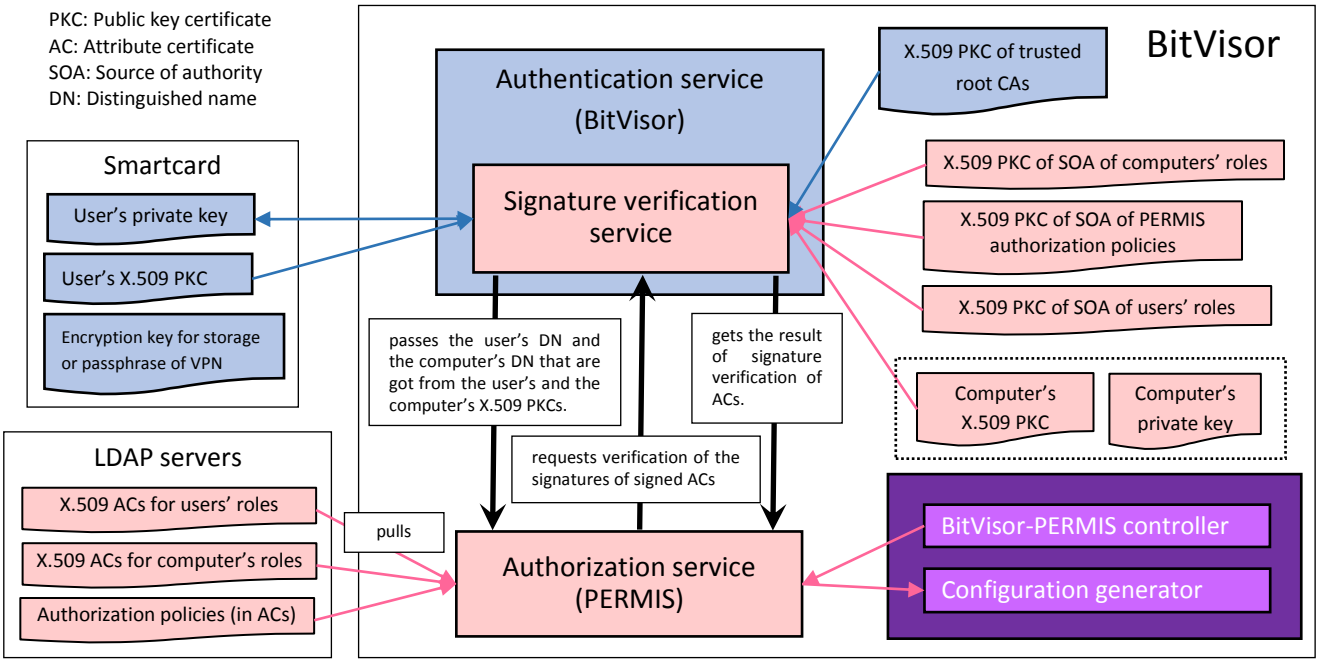

Figure 5. The relationship between user's subject DN, user's roles, computer's subject DN, computer's roles (i.e. security level), and policies in BitVisor.

Fig. 5 shows the relationship between user's subject DN, user's roles, computer's subject DN, computer's roles, and authorization policies in the integrated system of PERMIS and BitVisor. The red parts show the data and functions for PERMIS. The purple box is newly developed for the system. The blue parts show the data and functions that were already developed in the current BitVisor.

PERMIS's authorization polies are encapsulated in the form of ACs, and are retrieved from an LDAP server. Therefore, administrators can change the authorization policy after the setup of users' smart cards and computers. In Fig. 5, the system has three X.509 PKCs of the different SOAs, for verifying their issued X.509 ACs. However, we can use a single common X.509 PKC of a single SOA if it is issues all of the different ACs. Also, the system has the computer's X.509 PKC and private key, and the root CAs' self-signed PKC. It is important to protect the latter two in BitVisor. We can protect the root X.509 PKCs, computer's private key, BitVisor, and the system's software by using trusted boot technology [3].

IV. Static Configuration File for BitVisor

As described in section II, this paper's purpose is to enhance the BitVisor management method with a secure RBAC policy mechanism. Before describing the design of the new system, we have to describe the current management method of BitVisor with its static configuration file. This section shows an example of the static configuration of BitVisor. Fig. 6 shows a definition part for encryption operations of USB devices in the current BitVisor's configuration file.

```
1  vmm.driver.concealEHCI=1
2  vmm.driver.usb.uhci=1
3  vmm.driver.usb.ehci=1
4  storage.encryptionKey0.place=IC
5  storage.conf1.type=USB
6  storage.conf1.host_id=-1
7  storage.conf1.device_id=-1
8  storage.conf1.lba_low=0
9  storage.conf1.lba_high=0x9FFFFFFF
10 storage.conf1.keyindex=0
11 storage.conf1.crypto_name=aes-xts
12 storage.conf1.keybits=256
```

Figure 6. A definition part for encryption operations in BitVisor's configuration file

Lines 1 to 3 state that the encryption function is enabled. Line 4 states that the encryption key is stored in a user's smartcard. Lines 5 to 12 state how to encrypt USB devices. In the current BitVisor architecture, an administrator installs this configuration file onto a computer at the first setup time. Therefore, an administrator cannot change this configuration after the initial setup of the computer. BitVisor's configuration file also has definition lines for the VPN function of BitVisor, which contains the VPN gateway's IP address, VPN gateway's authentication method, etc. This paper only describes an example policy for storage encryption. However, the new system can apply equally well for VPN services and the other services of BitVisor such as HDD, SSD, CD-RW etc. A detailed specification of BitVisor's configuration is described in [5].

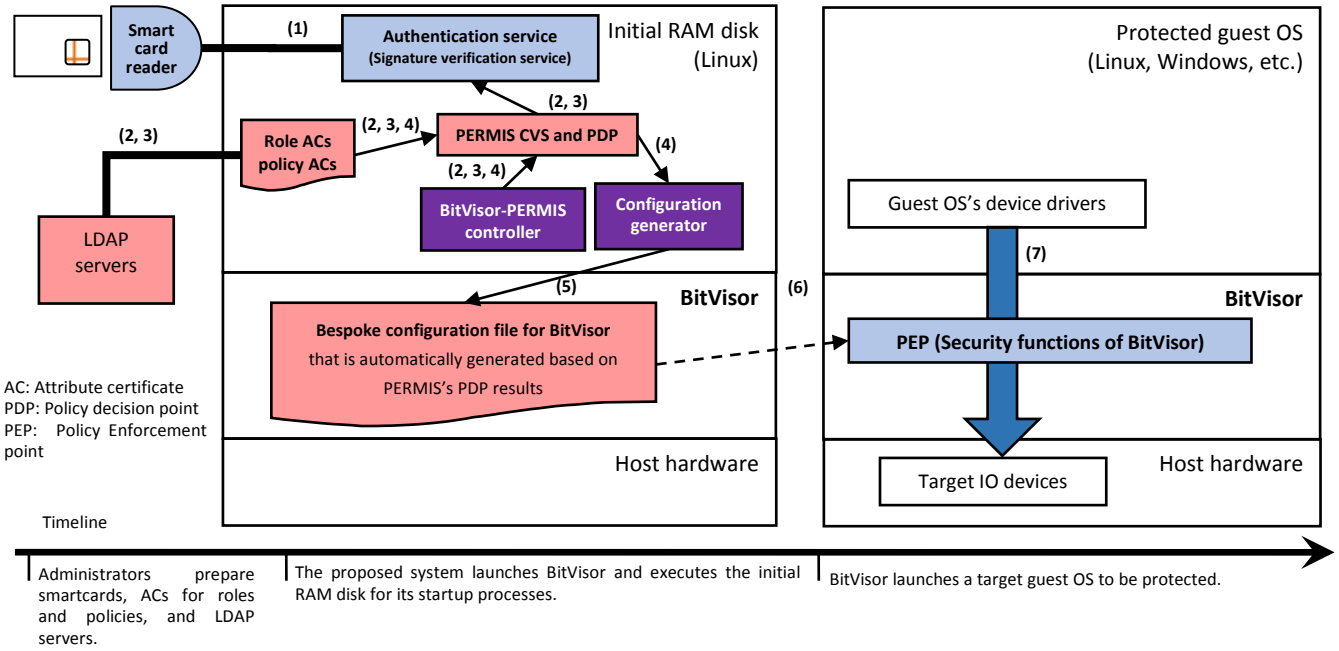

Figure 7. The design of the secure RBAC system for BitVisor.

V. DESIGN

BitVisor has a preparation step before launching the target OS that is to be protected. At the boot time, BitVisor launches a special guest OS (a custom initial RAM disk of Linux) to process the user authentication function based on the PKI-based smart card. If the authentication is successful, then BitVisor shuts downs the first operating system automatically and launches the target OS to be protected, based on its static configuration file. Fig. 7 shows the design of the new system. The red boxes show the RBAC extension parts. It works in the following way (the steps marked (New) indicate the new developments of the system):

(1) The ID management function authenticates a user using his or her smartcard. The smartcard has the user's private key and his or her X.509 PKC. The user's X.509 PKC is verified by using the X.509 PKC of the trusted root CA(s) that was (were) securely stored in the computer at the setup time.

(2) (New) BitVisor-PERMIS controller constructs a PERMIS authorization system passing it the DN of the policy SOA and details of the LDAP server to use. PERMIS retrieves the authorization policy AC from the LDAP server, and asks the authentication service to validate that it is signed by the trusted policy SOA.

(3) (New) BitVisor-PERMIS controller calls the PERMIS CVS twice, once to verify the user's roles, and once to verify the computer's roles. PERMIS retrieves the user's X.509 role ACs from the configured LDAP servers based on his or her DN which BitVisor retrieved from the user's X.509 PKC in step (1). PERMIS also retrieves the computer's X.509 role ACs from the LDAP servers by using the computer's DN that BitVisor retrieved from the computer's X.509 PKC. (The computer's X.509 PKC and private key are stored securely within the initial RAM disk image that can be verified by TPM and by using the X.509 PKC of the trusted root CA(s).) PERMIS checks that each role AC is signed by the correct SOA, as detailed in the CVS policy (see Fig.3), by calling the signature verification service. This uses the SOAs' X.509 PKCs which are included within the initial RAM disk image.

(4) (New) BitVisor-PERMIS controller places the validated user's roles into the subject field, and the computer's roles into the environment field and calls the PERMIS's PDP multiple times. This is to execute a series of tests that are used to generate the appropriate configuration file for BitVisor with the guest OS, based on the current authorization policy.

(5) (New) The configuration generator dynamically generates a bespoke configuration file for BitVisor as described below.

(6) After shutdown of the first guest OS (the initial RAM disk of Linux), BitVisor launches the target guest OS to be protected. This two-step execution mechanism is already used by the current release of BitVisor [3].

(7) BitVisor enforces the security functions based on the configuration file that was generated by the previous steps.

As described above, this paper shows an enhanced method of using a secure RBAC function for BitVisor with minimal changes to the original system. The main contribution of this paper is the translation mechanism from the results of the PERMIS's PDP tests to generate a bespoke

configuration file for BitVisor based on the user's role, the computer's role, and the authorization policy. The BitVisor-PERMIS controller program is executed on the initial RAM disk of Linux, not on BitVisor, because of the following reasons: (1) PERMIS needs a Java VM but BitVisor cannot execute a Java VM (this is because BitVisor is written in C and assembler), (2) BitVisor has only a limited number of targets and actions, therefore it is reasonable to generate a static configuration file in advance (the characteristics of the BitVisor environments only require a simple solution), and (3) it is difficult to provide high performance from XML policy parsing (insufficient for BitVisor's requirements).

Fig. 8 shows the architecture of the configuration generator. In the system, an administrator has to prepare test patterns including targets (e.g. USB devices) and actions (e.g. Input and output operations) with encryption options (e.g. true or false). For each test pattern, the administrator prepares a piece of configuration file that corresponds to the result of the test. The system checks all test patterns with the retrieved roles by executing the PERMIS PDP engine. Then the system selects the appropriate piece of configuration file based on the result of the PERMIS PDP engine. Finally, the constructed configuration file is used by BitVisor. Fig. 9 gives an example test and piece of configuration file. In this example the test is checking if the user has a role of manager and if so, then he can write encrypted files to USB devices.

VI. IMPLEMENTATION

We implemented the system by using BitVisor 1.3 [5] and the latest version of PERMIS decision engine（5.0.2） [10]. We used the PERMIS's policy editor 5.2.6 for generating the X.509 authorization policy ACs. We also used the PERMIS's attribute certificate manager 5.0.1 for generating X.509 ACs for users' roles and computers' roles. We employed OpenLDAP 2.2.29 to store the ACs. We used two different computers, one for the LDAP server and one that is protected by BitVisor. We used Linux kernel 2.6.31.6 for making the initial RAM disk image described in section V. We tested Ubuntu 12.04LTS as the target guest OS. Table 2 shows the hardware specification used in the prototype implementation. We prepared three smartcards with roles of Manager, Employee, and Trainee by using the ID management programs of BitVisor. We evaluated the prototype system by using the authorization policies shown in Fig. 3 and Fig. 4. For example, a user with the role Employee could not use unencrypted USB devices on computers that have securityLevel between 1 and 4. A user with the role Manager could use encrypted USB devices on a computer that has the highest securityLevel 5.

VII. DISCUSSION

The new system does not implement RBAC policies directly onto hypervisors. Instead, we implement RBAC in the custom initial RAM disk image for BitVisor. The initial

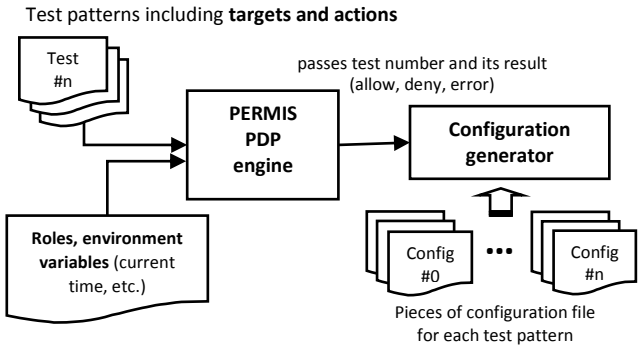

Figure 8. Architecture of the configuration generator program for BitVisor.

Table 2. Hardware specification.

Smart card reader	SCM SCR3310
Smart card	NTT Communications eLWISE (JICSAP 2.0)
Host hardware	ThinkPad X121e
USB thumb drive	BUFFALO RUF3-C 32GB

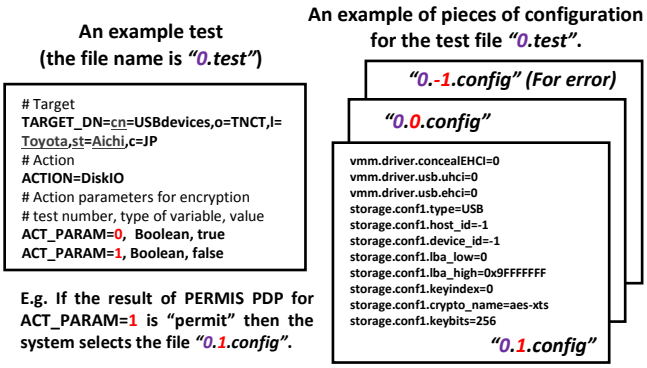

Figure 9. An example test and piece of configuration file.

RAM disk image includes the X.509 PKC of the trusted root CA and the SOA's PKCs. The initial RAM disk image of Linux is launched first and is protected by BitVisor. In addition, we can check the integrity of the initial RAM disk image before launching it by using a TPM [3]. We can use trusted boot mechanisms for this purpose.

If we implemented access control mechanisms onto hypervisors, we have to implement more faster and efficient programs than current PDP engines. For example, Linux's Flask architecture [19] (i.e. SELinux) employs an efficient access control method for Linux kernel by translating security policies into a special bitmap called access vector and uses this bitmap with cache mechanisms.

Finally, even if we deploy security systems based on hypervisors shown in this paper, we have to prevent many types of attacks [16][17] against hypervisors in operation.

VIII. RELATED WORK

We developed the ID management framework for BitVisor in [4]. Our previous work was to enhance BitVisor by using user's identities, but not roles. This paper shows the novel design and implementation of RBAC functions for BitVisor. There are many security applications constructed based on

BitVisor technology. HyperSafe is one of BitVisor's applications [11]. HyperSafe provides self-protection and an integrity guarantee mechanisms. TreVisor is a hypervisor-based full disk encryption system that provides resistant characteristics to main memory attacks [12]. Oyama et al. show a malware detection mechanism by using signatures in BitVisor [13].

XACML [18] is a standard XML policy for access control, but XACML does not support credentials and does not have a credential validation capability. PERMIS on the other hand was originally developed for constructing highly secure RBAC systems with digitally signed credentials and policies. The current version of BitVisor has the X.509 PKI-based authentication function; therefore we employ PERMIS to provide PMI facilities from the same standard using X.509 ACs. Now PERMIS has many applications like a privacy preserving authorization system for the Cloud [14], self-adaptive authorization framework for federated access environments [15], etc.

OpenStack [20] is open source IaaS Cloud computing software. OpenStack uses a hypervisor for building an IaaS Cloud, but not for securing client computers. OpenStack has an integrated authentication and RBAC authorization service called Keystone, which manages user identities and assigns them roles for accessing compute, storage and networking resources.

A recent development for securing client computers is the use of Cloud Terminal [21]. Users only use a thin cloud terminal program for securing network connections between client computers and Cloud servers, for transferring input and output data, etc. Since BitVisor protects a running OS in a local computer, then we might be able to deploy our system for protecting such a thin cloud terminal application that is executed on an untrusted local computer.

IX. CONCLUSION

The system described in this paper can support flexible changes of configurations for the security-purpose hypervisor, BitVisor. By introducing PERMIS's RBAC facilities, we can write authorization policies, users' roles and computers' roles and strongly protect them as digitally signed X.509 ACs, thus preventing any unauthorized person from modifying the configuration. This can be used to strongly control the behavior of the distributed computing systems in organizations. This paper shows the design and implementation of an integrated system with PERMIS's RBAC engine and BitVisor.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 23700095.

REFERENCES

[1] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, et al., BitVisor: a Thin Hypervisor for Enforcing I/O Device Security, In Proceedings of the 2009 ACM SIGPLAN/SIGOPS VEE09, pp.121-130, 2009.

[2] Manabu Hirano, Takahiro Shinagawa, Hideki Eiraku, Shoichi Hasegawa, Kazumasa Omote, Kouichi Tanimoto, et al., Introducing Role-Based Access Control to a Secure Virtual Machine Monitor: Security Policy Enforcement Mechanism for Distributed Computers, Asia-Pacific Services Computing Conference, pp.1225-1230, 2008.

[3] Manabu Hirano, Takahiro Shinagawa, Hideki Eiraku, Shoichi Hasegawa, Kazumasa Omote, Kouichi Tanimoto, et al., A Two-Step Execution Mechanism for Thin Secure Hypervisors, Emerging Security Information, Systems and Technologies (SECURWARE '09), pp.129-135, 2009.

[4] Manabu Hirano, Takeshi Okuda, Eiji Kawai, Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, et al., Portable ID Management Framework for Security Enhancement of Virtual Machine Monitors, Engineering the Computer Science and IT, Chapter 24, pp. 477-488, ISBN: 978-953-307-012-4, InTech, 2009.

[5] IGEL, BitVisor 1.1 Reference Manual, http://sourceforge.jp/projects/sfnet_bitvisor/downloads/bitvisor/manual/1.1/bitvisor-1.1-manual.pdf.

[6] D.W. Chadwick, O. Otenko, The PERMIS X.509 role based privilege management infrastructure, Future Generation Computer Systems 19(2), Elsevier Science Publishers B.V., pp. 277– 289, 2003.

[7] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security (TISSEC), 4(3):224–274, 2001.

[8] Botha, Separation of duties for access control enforcement in workflow environments. IBM SYSTEMS JOURNAL 40(3), 2001.

[9] S. Farrel, R. Housley, and S. Turner, RFC5755: An Internet Attribute Certificate Profile for Authorization, 2010.

[10] Modular PERMIS Project, http://sec.cs.kent.ac.uk/permis/.

[11] Zhi Wang; Xuxian Jiang, HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity, Security and Privacy (SP), 2010 IEEE Symposium on , pp.380,395, 2010.

[12] Tilo Müller, Benjamin Taubmann, and Felix C. Freiling, TreVisor: OS-independent software-based full disk encryption secure against main memory attacks, In Proceedings of the 10th international conference on Applied Cryptography and Network Security (ACNS'12), ACM, pp.66-83, 2012.

[13] Yoshihiro Oyama, Tran Truong Duc Giang, Yosuke Chubachi, Takahiro Shinagawa, and Kazuhiko Kato, Detecting malware signatures in a thin hypervisor. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12), ACM, pp. 1807-1814, 2012.

[14] Chadwick, D.W. and Fatema, K., A privacy preserving authorisation system for the cloud, Journal of Computer and System Sciences, 78 (5). pp. 1359-1373. ISSN 0022-0000, 2012.

[15] Bailey, C., Chadwick, D.W., de Lemos, R., Self-Adaptive Authorization Framework for Policy Based RBAC/ABAC Models, Dependable, Autonomic and Secure Computing (DASC), IEEE Ninth International Conference on , pp.37-44, 2011.

[16] Baozeng Ding; Yanjun Wu; Yeping He; Shuo Tian; Bei Guan; Guowei Wu, Return-Oriented Programming Attack on the Xen Hypervisor, Availability, Reliability and Security (ARES), 2012 Seventh International Conference on , pp.479,484, 2012.

[17] J. Rutkowska and R. Wojtczuk. Preventing and detecting Xen hypervisor subversions. In Black Hat USA, 2008.

[18] OASIS "eXtensible Access Control Markup Language (XACML) Version 2.0" OASIS Standard, 1 Feb 2005.

[19] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In Proc.USENIX Tech. Conf., FREENIX Track, pages 29–42, 2001.

[20] OpenStack project, OpenStack: The Open Source Cloud Computing Software, Available: http://www.openstack.org/.

[21] L. Martignoni, P. Poosankam, M. Zaharia, J. Han, S. McCa- mant, D. Song, V. Paxson, A. Perrig, S. Shenker, and I. Stoica, Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems, in Proc. USENIX ATC, 2012