TECHNIQUES FOR VALIDATION AND CONTROLLED EXECUTION OF PROCESSES, CODES AND DATA
A Survey

Dipankar Dasgupta, Sudip Saha and Aregahegn Negatu
Department of Computer Science, The University of Memphis, Memphis, TN 38152, U.S.A.

Keywords: Trusted Execution Technology, Trusted Platform Module, Digital Signature, Code Signing, Watermarking, Integrity Checker, Magic Cookie.

Abstract: Various security mechanisms are available to validate, authenticate and permit codes, data and scripts for executing in a computing device. Accordingly, different techniques and tools have been developed to preserve integrity and confidentiality at the process, protocol, system and communication levels. For example, Trusted Platform Module, Intel Trusted Execution Technology and Windows Vista Kernel Mode security ensure system level integrity and security, whereas, Digital Signature, Code Signing, Watermarking, Integrity Checker and Magic Cookies address integrity of data and executables in transit. A brief survey of these techniques is described here with how these techniques help to secure computing environment.

1 INTRODUCTION

Many techniques have been proposed and implemented to address various security issues (Lin and Loui, 1998, Gong and Schemers, 1998; Gong and Dageford, 2003). For example, security features have been implemented at the hardware level as in Trusted Platform Module (TPM) (Pearson, 2003; TCG, n.d.), Trusted Execution Technology (TXT) (Intel, n.d.). Specifically, TPM is a microcontroller that can securely store artifacts for authentication of code to run in a device (PC, mobile phones, network equipment, or any embedded device). These artifacts can include passwords, certificates, or encryption keys. The main feature of TPM is that security credentials are stored in hardware as it can better protect from cyber attacks. Such storage can hold platform measurements that help ensuring trustworthiness of the platform. Authentication and attestation (a process to prove that the platform has not been breached) are necessary steps to ensure safer computing environments. Many PCs and servers are now shipped with TMP, and many applications are being developed to establish a secure execution environment, protect data at rest or in transit, and demonstrate compliance with numerous data security regulations. Software and hardware manufacturers are also finding new ways to put the TPM to work (TCG, n.d.)

Intel's TXT (Intel, n.d.) is a set of extensions, which integrates new security features and capabilities into the processor, chipset and other platform components. TXT supports many capabilities including integrity, confidentiality, measurement, protection, attestation, and protected execution, at the hardware level so that a chain of trust for an execution environment can be built upon.

Prior to the introduction hardware-level security support (McCune et al, 2008), many system and application level security techniques have been in use. Most system level applications use isolation mechanism also sometimes called Domain Separation Mechanism (Rushby, 1984). Virtual machines (VM), such as Xen (Dragovic et al, 2003), and VMWare (Waldspurger, 2002) provide coarse-grained isolation, provide address space separation and restricted external interfaces. Such logically isolated environments enable applications running as if on different hardware.

However, other system level isolation techniques are implemented inside the kernel or at the application level, via system call interposition. Kernel Module Security (Conover, 2006; Microsoft, 2006) and Linux Security Module (LSM) (Wright et al, 2002) are examples of kernel-level security techniques. System level security via system call interposition techniques (Goldberg et al 1996; Garfinkel, 2003; Liang et al 2003) are widely used and used to create sandboxes, perform intrusion detection, prevent damages by untrusted code, or variable tuning of privileges. While kernel techniques are typically fast, interposition techniques have the advantage of flexibility. Sandboxing provides protection at the granularity of processes. There are many sandboxing based operating systems. AppArmor (AppArmor, n.d.) is a sandboxing technique designed as kernel (LSM) enhancement to better protect Linux operating systems such as Ubuntu-9. Sandboxing is also becoming a preferred method as the security model for application development platforms such as java (Gong and Dageford, 2003). Sandboxing isolation mechanism limits the damage from untrusted programs by reducing a process's privileges to the minimum and thwarts threats, whether it comes from a malicious program or security vulnerability of a program. Integrity checker is one of the earliest security techniques to prevent host intrusions such as detection of Trojan programs and backdoors (e.g. Rootkits).

Security at communication is very hard to ensure; but sophisticated techniques like digital signature, code signing, magic cookies etc. go a long way to secure the process of data communication and access control. While malicious access and alteration of resource is important and is addressed by the methods mentioned so far, it is also important to secure the right on those resources. Digital Watermarking handles this issue. To have trusted computing environment security measures should be put in place that spans the entire cross section of information technology (from hardware to applications and their network transactions). Here follows a brief description of representative security modules and techniques.

2 CONTROLLED EXECUTION ENVIRONMENT

Trusted Platform Module (TPM) often called the "TPM chip" or "TPM Security Device" is a hardware device for key management (generation, creation, encryption and decryption, and storage). Furthermore, it can perform a hash of a summary of the current hardware and software configuration of the machine to ensure that it is sealed, and that no alteration has been brought to it. It can be used for full disk encryption as it is fast enough to perform this seamlessly for the user. Software-wise, it is unbreakable, as any alteration to software can be detected and blocked, and the encryption keys are in kernel-space and hard to reach. It can, however, be broken by hardware means via a cold-boot attack (before the information disappears from memory, the system is booted in a small OS off a *USB* stick and the memory dumped; the keys are in system memory and can be retrieved easily, it has been proved that this is achievable with no special equipment). The cold boot attack relies on the machine being powered on, in sleep mode, or just been powered off. TPM can be thought of a small microcontroller that performs system integrity operation. It has the following capabilities (Bajikar, 2002; Parno, 2008)

- Crypto Capabilities
- RSA Accelerator
- Engine for SHA-1 hash algorithm
- Random number Generator
- Limited NVRAM for TPM contents

The cryptographic computation occurs inside TPM hardware and outside environment cannot have access to that execution; only I/O communication is performed. For authentication, digital signing and key wrapping operations TPM provides the facility of RSA encryption/decryption, too. SHA-1 procedure is implemented to provide hashing facilities. All these features make TPM a powerful small hardware entity to maintain system security.

TPM maintains an internal hardware protected storage of sensitive secret information to provide security facilities. These include keys for PKI communication, Attestation Identity key and various certificates. Three types of certificates are stored in TPM – endorsement certificate, platform certificate and conformance certificate. The contents are illustrated in the following diagram of (Bajikar, 2002).

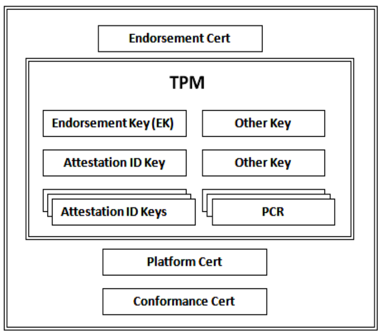


Figure 1: Contents of TPM storage (Bajikar, 2002).

TPM is a useful concept for PCs. But, it is more important in the context of notebooks because several threats concerning notebooks are addressed by TPM. Notebooks have various threats in higher order than PCs. Physical data theft is one of them. As notebooks often need to access internet from outside of the organization firewall, so there is a higher danger of data communication attack. Here is a threat matrix described in (Bajikar, 2002) that indicates how TPM addresses these issues.

Table 1: Threat matrix for notebooks and TPM (adopted from (Bajikar, 2002)).

Threats	Current Solutions	Weakness	TPM Solutions
Data Theft	Data Encryption (EFS, VPN, Encrypted email, etc.)	Encryption keys are stored on the hard disk and are susceptible to tampering	Protected storage of keys through hardware
Unauthorized access to platform	Username/ Password Biometrics and external tokens for user authenticat ion	Subject to dictionary attacks Biometrics can be spoofed Authenticatio n credentials not bound to platform	Protection of authenticati on credentials by binding them to platform
Unauthorized access to network	Windows network logon, IEEE 802.1x	Can be bypassed Certification can be spoofed Authenticatio n data is stored on the hard disk and is susceptible to tampering	PKI based method for platform authenticati on Hardware protection of authenticati on data

Trusted Execution Technology (TXT): is a hardware extension to some Intel microprocessors, meant to improve security. It is made up of two major components, TMP (as described above) and DMA (Direct Memory Access) page protection. Also, it improves upon the previous microprocessor and chipset technology of Intel such that each application runs in a concealed environment, which cannot be accessed by other applications; and data sent to and from an I/O device can only be read by the desired recipient. The capabilities of Intel Trusted Execution Technology include (Greene, n.d.):

- Protection of execution and memory spaces;
- Sealed storage of encryption keys;
- Attestation which ensures proper invocation of TXT environment and provides a verified measurement of the software running in the protected space.

The benefits provided by TXT are modeled by three strategies.

- Local verification which uses the measurement capability of TXT environment to make local user confident about the proper configuration of the system.
- Remote verification makes remote entities assured of the platform configuration.
- Multi-level operation takes advantage of TXT memory protection to run two or more applications or operating systems safely.

Kernel-Mode Security: The kernel is the most central part of an operating system. It is the first piece of code to boot up a computer; it enables the computer software to talk to the computer hardware; and it is responsible for low-level OS tasks such as memory management, multiprocessor synchronization and scheduling/launching of processes. Keeping the integrity of the kernel is critical for the performance, reliability and security of the entire computer. Windows Vista adds a set of security measures to prevent the kernel from being altered, or at least discontinue running once the kernel had been compromised. The improvements consist of checks being performed on the files to be run, based on a signed certificate containing the correct hashes for the files. As the certificates cannot be faked, an attacker must attempt to bypass these rather than trying to break. The kernel (NTOSKRNL.EXE) is loaded (and can only be loaded) by WINLOAD.EXE which performs checks on itself, the kernel, and all the essential and non-essential boot drivers. The checks on itself and the non-essential boot drivers can be ignored or disabled by being in kernel debug mode. NTOSKRNL.EXE depends on CI.DLL that handles Code Integrity. This dependency is the essential part of boot drivers and run-time drivers checked by WINLOAD.EXE. This file checks that every application that has a certificate embedded into it has not been altered. These checks can be disabled by system administrators if desired. NTOSKRNL.EXE has an internal part called PatchGuard that performs the self-check based on a 5-10 min timer that cannot be disabled. The kernel does not allow any program in user space to access physical memory directly. Windows Vista kernel mode security includes the following features (Conover, 2006):

- Driver Singing,
- PatchGuard,
- Kernel-mode code integrity checks,
- Optional Support for secure Bootup using a TPM hardware chip,
- Restricted User mode access to \Device\Physical Memory.

Driver signing mechanism requires any driver to be installed in kernel to have a certificate from a trusted third party. As long the certificate giving process is secured no malicious driver will be installed in the kernel. However, if any certificate is published or compromised, then the overall security becomes vulnerable.

The PatchGuard is designed to prevent kernel patching, which degrade security, reliability and performance. The kernel-mode code integrity check protects the operating system by verifying system binaries haven't been tampered with and by ensuring that there are no unsigned drivers running in kernel mode. Other security steps e.g. secure bootup and limited user access to physical memory adds extra protection to the kernel mode security.

Microsoft, by the integration of these security features in its OS, has raised the security bar to a higher level by preventing injection of unsigned, possibly malicious code. Enforcement of the security measure rests on the stated impossibility of disabling these component functions of the Kernel-Mode Security. But, as Symantec's research (Conover, 2006) shows, there is limit to the effectiveness of these security measures. These protections can be disabled by patching the kernel (NTOSKRNL.EXE). Once this is patched, WINLOAD.EXE will refuse to load it, so loader also needs to be patched. During execution, if PatchGuard manages to perform the self-check, it will halt Windows as it found the patched system files, so it needs to be disabled too. Disabling PatchGuard does not need patching. There is a subtle way to forcefully disable a timer, which PatchGuard relies upon to wake up and execute its checking, so that it never signals an event. This means, although the kernel-mode security has enhanced the security of Windows OS to prevent most of the malicious codes, it does not provide an iron-clad security.

Sandboxing. Sandboxing (Singh, 2004) is any technique used to separate running programs. The term is general, and can refer to virtual machines such as general purpose machines like VMware or KVM, or application-specific, such as JVM or CLR in Windows. It can also refer to protecting system calls by trapping them and limiting their use to specific applications; these can be outside the kernel, and hook into it or loaded as modules. Modern OS virtual memory also separates processes from each other by running each of them in a separate address space.

Java development environment uses sandboxing to incorporate security measures. JVM could run local as well as remote (usually applets) executable codes. The sandbox provides a restricted environment in which limitations are enforced on the system resources untrusted code can access or request. So, as used in other platforms, sandbox in Java is used for safe running of untrusted code, which comes from untrusted source. In Java 2 security model (Sun, 1997), untrusted code does not imply just applets; security check is extended to all java programs including applications. Java 2 allows fine-grained access control with easily configurable security policy as well as easily extensible access control structures.

As such Java 2 does not just a sandbox with a fixed boundary but provides multiple sandboxing environments each with different access control settings or permissions. Java's overall security is enforced via three-tier defense: a) bytecode verifier – along with the JVM, ensures legitimacy of bytecode and guarantees language safety and baseline security at run time; b) class loader – provide an important security feature of separating name spaces for various software classes and using separate class loaders, a degree of isolation is established between the instances of classes; c) security management – this is a mechanism (security manager, access controller) for applications to check the current effective policy and perform access control states.

Figure 2 depicts the java sandboxing mechanism. The important concept in the mechanism is the *protection domain*, which is a domain that encloses classes whose instance objects that are directly accessible by a principal (an entity – individual, corporation, login ID) to which a set of permissions are granted. *Security policy* is a mapping from classes (and their instances) to protection domains, which in turn is mapped to corresponding set of permissions. Protection domains can be bound to static set of permission that is granted despite the current dynamic policy setting. The protection domain can also be initialized to use the static permission as well as the dynamic security policy (editable by a policy tool). Protection domains fall into two distinct categories: system domain through which all protected external resources such as file and network I/Os must be accessed; and application domain that encloses rest of the resources. The protection domain and its associated set of permission define a specific sandbox. The CodeSource (the URL of a class and certificate used in code signing) binds an application (bytecode) to a sandbox. An executable that runs in the sandbox environment under defined policy, gets access to the granted application and system domain resources.

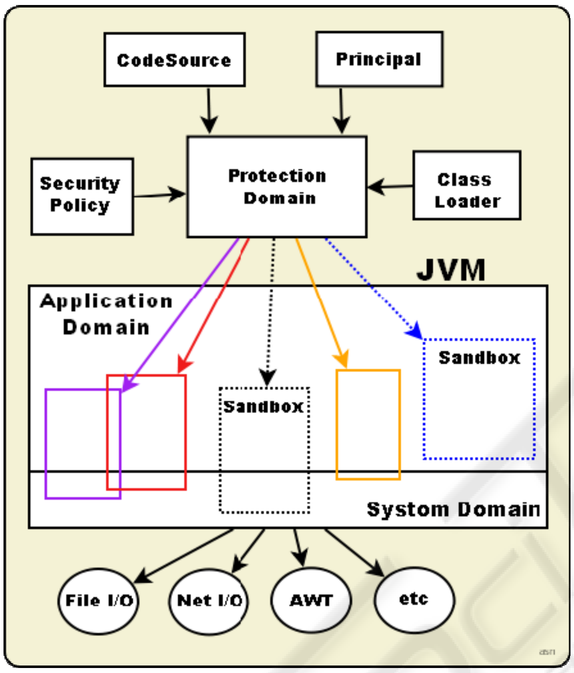


Figure 2: Sandboxing Mechanism of Java 2.

3 VERIFICATION AND VALIDATION METHODS

Digital Signature (Lysyanskaya, 2002)**.** It is a digital code that is attached to an electronic document which uniquely identifies the sender and ensures integrity. It is like a hand written signature, which is used to check the authenticity of the sender of a document. Digital signature can be used on any document once it is stored digitally. For example, emails and digital contracts use this scheme extensively. Particularly for ecommerce transactions digital signature is very important.

Digital signature is realized by cryptographic techniques. Three processes are involved: key generation, signing and signature verification. The key generation process produces a public-private key pair. The sender signs the document with the private key. The receiver verifies it by using the corresponding public key. Several cryptographic algorithms are used in this regard, e.g., RSA, DSA and ECDSA. A diagram illustrating digital signature by RSA is shown in Figure 3 (Lysyanskaya, 2002).

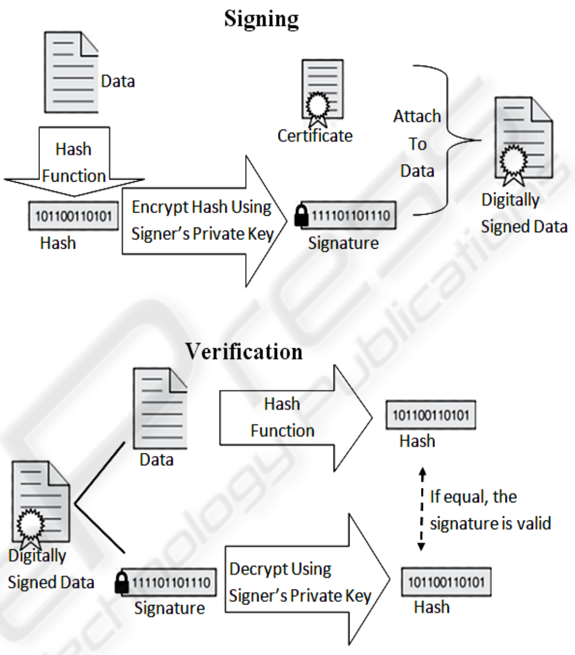


Figure 3: Signing and Verification in RSA digital signature (Lysyanskaya, 2002).

Digital signature is being widely used to ensure authenticity of documents. The first highly used software package offering digital signature was IBM's Lotus Notes 1.0 released in 1989.

Code Signing. Digital Signature when applied on codes is called code signing. It is a form of verifying authenticity of codes. The author digitally signs executables and scripts so that the end user can confirm the author of the code and make sure it has not been corrupted after deployment. In this way, the process ensures security of codes while deploying. Usually, it works by a public key infrastructure (PKI). The author signs the document with a secret private key. The end user uses the public key to verify the signature. Many publicly available executables are distributed after code signing. For example, Linux and Windows update services use code signing to ensure that malicious updates or patches are not installed at client system. The processes of code signing and code verification are illustrated in (Fleischman, n.d.) are shown in Figures 4(a) and Figure 4(b).

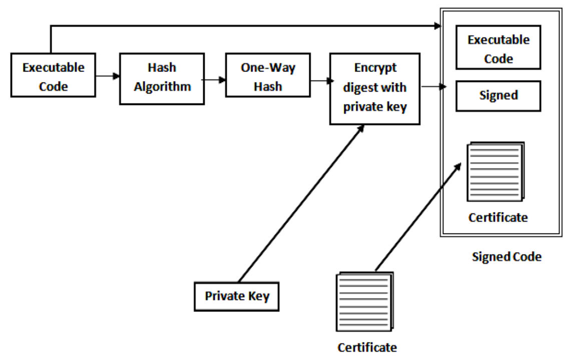


Figure 4(a): Code-Signing Process (Fleischman, n.d.).

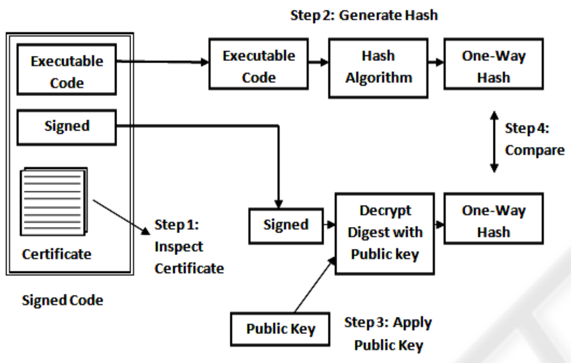


Figure 4(b): Code Verification Process (Fleischman, n.d.).

Digital Watermarking (Cox et al, 2008)**:** is the process of embedding additional information into a digital signal, e.g., image, audio or video. The information that is embedded is a bit pattern that infers its validity or copyright information. The term was derived from faintly visible watermarks of producer information seen on stationary products. Unlike this printed watermarking, invisible signal is embedded in case of digital watermarking. It is invisible in case of image or video clips, it is inaudible in case of audio clips.

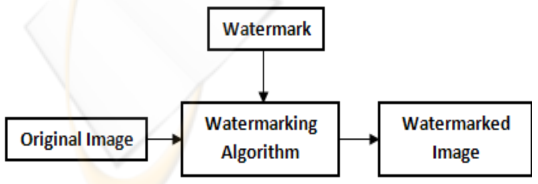


Figure 5: Diagram showing steps involved in watermarking (WolfGang and Podilchuk, 1999).

When a file being watermarked is copied, the watermark information is also copied and the watermark can be retrieved from that copied file. A robust watermark should sustain malicious modification made by a copier, so that it can be retrieved. Watermarking has important applications in copyright protection and stegonography (hidden message).

Integrity Checker: is simple application software that checks the integrity of files. At first the known size of a file is stored. A hash is computed from the contents of the file and it is also stored. Later, if a file violates these two data kept by the Integrity Checker, then a flag pops up that indicates probable file integrity violation. There exists many host-based integrity checking or change auditing tools, which include *Tripwire, AIDE, AFICK and Samhain*. Here, we will briefly discuss the former two.

Tripwire: is an integrity verification tool which automatically detects unauthorized or unintended changes of critical system files and allows system administrators to immediately be aware of the compromise so that remedial steps can be taken. In general, it is a process of detecting changes by comparing an image of a system with an optimal baseline security setting to the image of the same system running at any operational state. Tripwire's basic configuration and operations are shown in the figure 6 below. The first step is to setup a baseline secure system with tripwire installed on it and identify the files that need to be checked for integrity. Second, *edit* and save general Tripwire configuration (location of database, number of reports, etc) and Tripwire policy (what and how to monitor). Once the initializations of the reference database with the pristine or baseline information of the monitored files are configured, the Tripwire system is ready for integrity checking. The checking process involves the comparison of current copies of files with that in the reference database. After installation of new software or change of system configuration, the system administrator should *update* the reference database and this is done by updating the reference database with a new snap-shot of the monitored files. Enterprise Tripwire, the commercial version, provides quicker remediation and alert by detecting, reconciling and reporting changes over large quantity and type of data elements.

AIDE (Advanced Intrusion Detection Environment)**:** is another open-source competition to Tripwire but with additional features: a) its database stores various file attributes including permissions, inode number, user, group, file size, mtime and ctime, atime, growing size, number of links and link name as well as an easy way of mixing attributes for

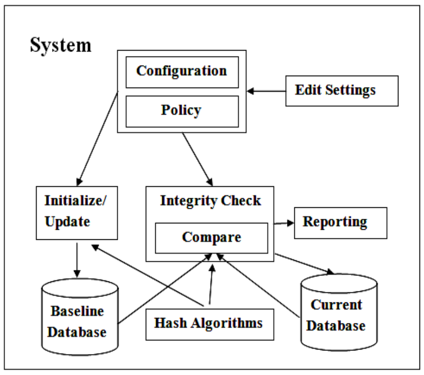


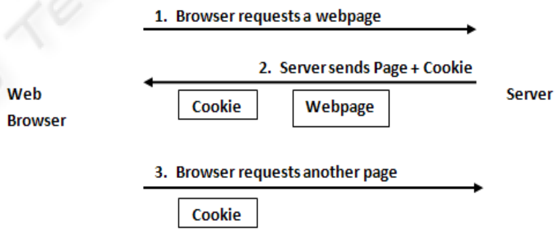
Figure 6: Integrity checking process.

setting up different set of monitoring policies; and b) creates hash of each file using one or a combination of its many message digest algorithms. Except these enhancements, AIDE is similar to Tripwire in editing configuration and monitoring policy, initializing and updating of databases, checking of integrity, and the qualitative nature of reporting.

Security Content Automation Protocol (SCAP): System administrators and security managers spend many man hours in the identification, remediation, and reporting of system security vulnerabilities. While many organizations implement well defined processes to enhance security, most processes still involve manual processing. To enhance efficiency and quality of information security, a set of standard approaches and their integration process has been devised. In the past, Department of Defense (DoD) has been using a vulnerability management process (with a number of manual tasks) called IANA - Information Assurance Vulnerability Alerts (Martin, 2005). This process encompasses the steps: i) discovering the new software security flaw (New IAVA requirement); ii) assessment of the flaw; iii) deporting the status of the flaw; iv) remediation of the flaw; and v) implementing the change and subsequent return to complaint state. Without standards and automatic processing, it is difficult for big organizations to maintain and streamlined vulnerability control and removal.

Security Content Automation Protocol (SCAP) is a method for using specific standards to enforce policy compliance evaluation and automated vulnerability management (CVE, n.d.; CPE, n.d.; CCE, n.d.; CVSS, nd.). It consists of a number of open standards that are used to determine the number of software flaws and configuration issues related to security. These standards measure systems' vulnerabilities and offer methods to evaluate the possible impact. Accordingly, SCAP can be used for maintaining the security of the enterprise systems. In particular, it can be used to automate the verification of patch installation, checking system security configuration settings, and examining the overall system. SCAP validates the specific versions of vendor products based on the platforms they support. It reconciles software flaws from US CERT and MITRE repositories. The Information Assurance Vulnerability Alerts (IAVA) process (Martin, 2005) that had been used by DoD earlier are streamlined and automated by transforming the IANA process using a set of SCAP standards, which enable secure information flow between machines.

Magic Cookies: Are tokens or tickets that are passed between communicating parties for performing some operation. Cookies are used for authentication in communication. A server leaves a cookie on the client side first time the client authenticates and accesses it. That cookie serves as a token of authentication. Later, the client shows this cookie to avoid authentication again. The content of the cookie is opaque to the client but the server uses it for its purpose. It is encrypted so that no unauthorized party can pretend with its own cookie. At present, magic cookies are important part of worldwide web. When cookies are stored on user's computer by the web browser it is called http cookie. An illustration of http cookie is shown in Figure 7.


1. Browser requests a webpage
2. Server sends Page + Cookie
Web Browser
Cookie
Webpage
Server
3. Browser requests another page
Cookie
Figure 7: HTTP Cookie: Interaction between web browser and server (Lin and Loui, 1998).

4 SUMMARY

Ensuring computer security depends on securing it in all aspects such as execution, storage, communication. This survey puts forward the most common techniques/tools that try to provide the secure environment in computing (Austin and Durbi, 2003) – application security, file security, process security, and system level security. In order to harden systems from sophisticated attacks, it is very important to know the available techniques to protect systems in a layered fashion and thus to

Table 2: Summary of various techniques and their applicability in securing computing systems.

At which Layer	Name of tool/Technique	Example Vendors	Validation mechanism
Application	TXT (Greene, n.d.)	Intel	Multi-level operation
	Code Signing (Dean, 1999)	VeriSign	Cryptographic hashing through PKI
System	TPM (Bajikar, 2002)	Trusted Computing group	Encryption mechanisms
	TXT (Greene, n.d.)	Intel	Local verification
	Kernel Mode Security (Conover, 2006)	Windows Vista	Driver signing, PatchGuard, Kernel mode integrity checks, Restricted access to physical memory
Process/ executables/ scripts	Sandboxing (Singh, 2004)	VMware, JVM, CLR	Mechanism of separation
	Code Signing (Dean, 1999)	VeriSign	Cryptographic hashing through PKI
Network/Traffic	TPM (Bajikar, 2002)	Trusted Computing Group	PKI based methods
	TXT (Greene, n.d.)	Intel	Remote Verification
	Magic Cookies (Magic Cookies, n.d.)	X Windows System	Token-based security operations
Data/File/ Directory	TPM (Bajikar, 2002)	Trusted Computed Group	Protected storage of keys and encryption mechanism
	Digital Signature (Lysyanskaya, 2002)	IBM	Cryptographic signing and verification process
	Digital Watermarking (Wolfgang and Podilchuk, 1999)	Digimarc, ISAN	Embedding copyright information cryptographically
	Integrity Checker (Dean, 2006)	Tripwire, AIDE	Storing hash and other relevant information for change detection

enhance security to both wider and deeper extents. Table 2 summarizes various validation and verification techniques described thus far and indicates the layer specific role of various security and validation mechanisms. Some of these mechanisms are generally used in combination, while others are special-purpose.

REFERENCES

AppArmor, Linux Application Security, http://www.novell.com/linux/security/apparmor//overview.html

Austin , R. D., Darby, C. A., 2003. The Myth of Secure Computing. Harv Bus Rev. ed 81(6).

Bajikar, S., 2002. Trusted platform module (TPM) based security on notebook PCs, Intel White Paper.

CCE, http://cce.mitre.org

Conover, M., 2006. Assessment of Windows Vista Kernel-Mode Security, Symantec Advanced threat research.

Conover, M., 2006. Analysis of the Windows Vista Security Model,

Cox, I. J., Miller, M. L., Bloom, J. A., Fridrich, J., Kalker, T., 2008. *Digital Watermarking and Steganography*, Morgan Kaufmann, 2nd Edition.

CPE, http://cpe.mitre.org

CVE, http://cve.mitre.org

CVSS, http://www.first.org/cvss

Dean, R. D., 1999. Formal Aspects of Mobile Code Security," PhD thesis, Princeton University.

Dean, S., 2006. Integrity Checker, http://www.sdean12.org/IntegrityChecker.htm

Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer, R., 2003. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*.

Fleischman, E., Code Signing, The Internet Protocol Journal, Vol 5, No 1.

Garfinkel, T., 2003. Traps and pitfalls: Practical problems in system call interposition based security tools. In *Proc. Network and Distributed Systems Security Symposium*.

Goldberg, I., Wagner, D., Thomas, R., Brewer, E. A., 1996. A secure environment for untrusted helper applications (confining the wily hacker). In *Proc. of the USENIX Security Symposium*, San Jose, California.

Gong, L., Schemers, R., 1998. *Signing, Sealing, and Guarding Java™ Objects*, Book chapter, Springer Verlag.

Gong, L., Ellison, G., Dageford, M., 2003. Inside Java 2 Platform Security: Architecture, API Design and Implementation, 2nd Edition.

Greene, J., Intel Trusted Execution Technology, Intel Technology Whitepaper.

Intel®, Trusted Execution Technology Architectural Overview, http://www.Intel.com/technology/security

Liang, Z., Venkatakrishnan, V. N., Sekar, R., 2003. Isolated program execution: An application transparent approach for executing untrusted programs. In *ACSAC*, pp 182–191.

Lin, D., Loui M. C., 1998. Taking the bite out of cookies: privacy, consent, and the Web, ACM SIGCAS Computers and Society, Volume 28 , Issue 2 pp. 39 – 51.

Lysyanskaya, A., 2002. Signature Schemes and Applications to Cryptographic Protocol Design, PhD thesis, MIT.

Magic Cookies, Wikipedia, http://en.wikipedia.org/wiki/Magic_cookie

Martin, R. A., 2005. Transformational Vulnerability Management Through Standards.

McCune, J. M., Parno, B., Perrig, A., Reiter, M., Seshadri, A., 2008. How Low Can You Go? Recommendations for Hardware-Supported Minimal TCB Code Execution, In *Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), ACM*.

Microsoft, 2006, First Look: New Security Features in Windows Vista, *TechNet,* http://www.microsoft.com/technet/technetmag/issues/2006/05/FirstLook/default.aspx

Parno, B., 2008. Bootstrapping trust in a "trusted" platform. *In Proceedings of the 3rd Conference on Hot Topics in Security (San Jose, CA)*, USENIX Association, Berkeley, CA, 1-6.

Pearson S., 2003. Trusted Computing Platforms: TCPA Technology in Context, Prentice Hall PTR.

Rushby, J., 1984. A Trusted Computing Base for Embedded Systems. In *Proceedings of the 7th DoD/NBS Computer Security Conference*, Gaithersburg, Maryland, pp. 294-311

Singh, A., 2004. A Taste of Computer Security.

Sun Microsystems, 1997. Java Security Model.

TCG, Enterprise Security: Putting the TPM to Work.

Waldspurger, C., 2002. Memory resource management in VMware ESX server. In *Fifth Symposium on Operating Systems Design and Implementation*.

Wolfgang, R. B., Podilchuk, C. I., 1999. Perceptual Watermarks for Digital Images and Video, In Proceedings of the IEEE, Vol 87, No 7.

Wright, M., Cowan, C., Morris, J., Smalley, S., Kroah-Hartman G., 2002. Linux Security Modules: General Security Support for the Linux Kernel, In *Proceedings of the 11th USENIX Security Symposium*.