Methodology for Deriving and Integrating Countermeasures Design Models for Electronic Commerce Systems

Tanvir Tahir[1], Kamar Abbas[2]

[1]Department of Computer Science & Engineering, Institute of Engineering & Technology, Sitapur Road, Lucknow, UP, India
[2]Azad Institute of Engineering & Technology, Lucknow, UP, India

Abstract: ***In this paper, we provide a detailed description of methodology for deriving and applying Electronic Commerce (EC) security countermeasures design models from the existing IT standards. Our goal is to describe a model-based approach of how to extend such a model or "specialize" it in order to apply it to e-commerce systems.***

Keywords: e-commerce, malicious user, Sniffing, Spoofing, Brute-force, Cryptography & Decryption.

1. Introduction

This paper describes a new methodology for deriving countermeasures design models for e-commerce systems. The methodology is based on the E-Commerce security services model. Our approach focuses on satisfying legitimate user requirements while blocking malicious user requirements at system design time. This assessment shows that our methodology is systematic through a case study that derived four countermeasures design models for authentication, authorization, access control enforcement, and transaction privacy. The derived countermeasures design models were assessed through a realistic case study on a SET-integrated in e-commerce systems. These models were also proven to be effective against all security attacks related to the e-commerce domain.

2. A Unique Aspect of our Approach

A unique aspect of our approach is that it focuses on introducing the appropriate security countermeasures early in the design process [1] while taking into consideration the different types of *malicious users* and the attack power each type might have. In general, malicious users can be grouped into three categories: crackers, intruders, and insiders. A *cracker* is someone who breaks systems for nefarious ends. An *intruder* is someone who gains access into systems by force. An *insider* is a person who is in a position of power or has access to system confidential information. The power such users might wield and the threats they may impose vary from one user category to another [2]. Therefore, security designers, and our methodology in this case, have to take into account these different categories and design countermeasures accordingly.

3. Understanding Malicious Users and their Motivations

Taking into consideration the above-mentioned "malicious user" categories supports our methodology for the systematic introduction of the proper security countermeasures early in the system design process as follows. Every EC system is normally built upon a set of user requirements. In the context of security, user requirements are: availability, integrity, confidentiality, accountability, and assurance.

Malicious user requirements (MUR), on the other hand, are requirements allowing malicious users to attack the EC system. Our methodology captures these requirements through a new notion: "attack enablers". By capturing these requirements, we will be able to understand malicious users and their motivations. This enables an effective security design that provides effective countermeasures against all known malicious behaviors.

4. A New Design Goal: Block Malicious User Requirements

In a typical EC system life cycle, user requirements are captured during the system design phase and the EC system is designed to satisfy those requirements. Our proposed methodology deals with malicious user requirements (MURs) on a similar basis with the exception that the system design must block these requirements rather than satisfy them.

It is worth noting that MUR can exist in different system components: EC system itself, operating system platform, third party software components, etc. In this paper, we will deal only with MURs for the EC system at the design phase.

Phases of the Methodology

Our methodology, depicted below, can be divided into two functional phases.

Phase 1: Select features and derive design models
Phase 2: Instantiate and integrate the derived models into an e-commerce system design

In **Phase 1**, security features are selected and security-oriented design models are derived and verified. In **Phase 2**, the derived security-oriented design models from Phase 1 are instantiated and integrated into an existing e-commerce system design.

Phase 1: Select Features and Derive Design Models

Step 1 - Select security features.
In this phase, we apply steps 1 through 2.4 of our methodology. In this paper, we select four security features of the security services model; namely

- Authentication,
- Authorization,
- Access control enforcement, and
- Transaction privacy.

For each selected security feature, we will derive a security-oriented design model.

Step 2 - For each security feature, derive a countermeasures design model:

Step 2.1 – Identify and abstract all attacks related to the security feature.

This includes all attacks referenced in OSSTMM as well as other attacks that are applicable to the selected security feature from other referenced literature. In this paper, we perform an exhaustive investigation into the set of all known attacks related to authentication, authorization, access control enforcement, and transaction privacy. This list of attacks will be used in this step of the methodology. The goal of having an abstract description of the attacks is to mask platform-specific execution details, such as operating system releases, and concentrate more on the functionality and requirements for understanding how the attack mechanism works?

5. Applying the Design for Security Methodology to the Security Services Model

Every section will begin with a definition of the security feature to which we intend to apply our methodology.

1. We select security features. For our case study we select *authentication*, *authorization*, *access control enforcement*, and *transaction privacy*.

2. For each feature,
2.1. We identify and abstract all related security attacks
2.2. For each attack,
2.2.1. We derive all attack enablers
2.2.2. We prescribe appropriate security countermeasures
2.2.3. For each countermeasure,
2.2.3.1. We analyze the countermeasure for residual vulnerabilities
2.2.3.2. We add corrective measures to overcome vulnerabilities
2.3. We derive the complete security-oriented countermeasures design model for the feature

Security-Oriented Authentication Design Model

Step 1 is to select a security feature. Here we select *authentication*.

Authentication is the process of verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in a system. [3] The identity of a certain user or process is challenged by the system and proper steps must be taken to prove the claimed identity.

Authentication models may depend on specific technologies. An example of such a model is the open *authentication* model supporting electronic commerce in distributed computing [4] that is based on CORBA technology [5] and provides an extension to the Kerberos *authentication* framework [6] using a public key cryptosystem. Another example of an *authentication* model is a general framework for constructing and analyzing *authentication* protocols in realistic models of communication networks. [7].

The above referenced literature emphasizes satisfying standard security requirements while providing extensions to different technologies such as CORBA and Kerberos. As discussed earlier, the main emphasis in current standards for security requirements is on satisfying legitimate user requirements from a security point of view; however, in this paper, we emphasize blocking malicious user requirements at system design time (this is also known as preventive security design) [8]. The goal of this section is to show how our methodology is useful for deriving a preventive design model for *authentication* that can either be incorporated into any integrated authentication model (such as above) or can be implemented as a standalone authentication module in e-commerce systems.

Step 2.1: Identify all attacks related to authentication
Security attacks related to authentication can be identified from the literature and from personal experience. In this paper, we have projected all known security attacks onto various types of e-commerce authentication models described [9, 10, 11,7,12 & 4]. This comprises most known attacks applicable to authentication in the domain of EC systems.

Note that the main focus of this paper will be on attacks directly related to e-commerce systems. Attacks related to network components, to third-party software components, and attacks against the operating system that is supporting the e-commerce application will not be discussed here. In practice, of course, these types of attacks also have to be handled.

The specific security attacks related to *authentication* in e-commerce systems are ([13, 2,14,15,16 & 17] :

Sniffing attacks (also known as man-in-the-middle attacks)

- Dictionary attacks
- Replay attacks
- Brute-force attacks
- ID spoofing attacks (also known as spoofing attacks)
- Credential decryption attacks (supplementary to other types of attacks)
- Side-channel attacks

We will now consider each type of attack, derive attack enablers, and prescribe effective countermeasures.

Step 2.2: For each authentication attack, derive its enablers and countermeasures

This section provides a succinct abstract description of all known *authentication*-related security attacks. Attack enablers are then identified, and effective countermeasures are prescribed. The attacks are presented and discussed below in order of dependence, since some of them are related (e.g. dictionary attacks depends on brute-force attacks).

5.1 Sniffing Attacks

Sniffing attacks [13, 2, 14 & 15] (also known as the *man-in-the-middle attacks*) are the digital analogues to phone tapping or eavesdropping. This attack captures information as it flows between a client and a server. Usually, a malicious user attempts to capture TCP/IP transmissions, because they may contain information such as usernames, passwords, or the actual contents of an email message.

A ***sniffing attack*** is often classified as a ***man-in-the-middle attack*** because in order to capture packets from a user, the machine capturing packets must lie in between the two systems that are communicating (a *man-in-the-middle attack* can also be waged on either one of the two systems).

The *attack enabler* in this case is the process of sending data across communication channels in ***clear text*** format. In this paper preventing access to the communication channel is not a valid countermeasure, because this is due to the open nature of the internet. However, by encrypting the communication channel between the user/process and the system, sniffing attacks are disabled, i.e., sniffing retrieves only useless encrypted information. However, the information can be duplicated and substituted for subsequent transmissions. This type of attack is known as “replay attacks” and will be discussed later on in this section.

5.2 ID Spoofing Attacks

ID spoofing attacks [13, 2, 14, 17 & 15] occur when a malicious user or process claims to be a different user or process. This attack allows an intruder on the internet to effectively impersonate a local system's IP address. If other local systems perform session authentication based on the IP address of a connection (e.g. rlogin with .rhosts or /etc/hosts.equiv files under UNIX), they will believe incoming connections from the intruder actually originate from a local "trusted host" and will not require a password. The ***attack enabler*** for this attack is for authentication to rely on static information such as IP addresses, host names, etc. This is equivalent to trusting certain hosts or processes according to some pre-defined static information. The system authenticates the user or process only by checking the given static information. In such a case, the attacker will attempt, through complex attack tools, to “spoof” the system by claiming that he/she is the trusted host or process. Since no challenge is attempted in this case, the attack has a great chance of succeeding.

The ***countermeasure*** for such an attack is to use ***challenge based authentication***. Challenge-based authentication includes the use of certificates, user/password combinations, etc. If challenge-based authentication is inapplicable for a certain specific case, then *least privilege static authentication* must be applied. *Least privilege static authentication* means giving the least possible access privilege to the fewest possible number of users, processes or hosts after successful authentication. By doing so, the risk associated with relying on static authentication, when challenge-based authentication cannot be applied, is kept to a minimum.

5.3 Brute-Force Attacks

A Brute-force attack [13, 2 & 15] is any form of attack against a password file that attempts to find a valid username and password by successive guessing. This type of attack is *enabled* by gaining access to the credential (user names and passwords) storage medium. The attacker first retrieves a copy of the database system or system file holding credential information. If the credential information is encrypted, a brute-force attack tool will try all possible combinations of user’s name and passwords. For each combination, the user name and password are encrypted using the same encryption algorithm that was used to encrypt the original credential information. Then, the encrypted data is compared to the retrieved copy of credential data.

Different types of encryption algorithms are used and the attack proceeds until both credentials (user name and password) match. The *countermeasure* for this type of attack is to enforce access permissions through a strong access control policy at the operating system level. By doing so, malicious users will fail to retrieve a copy of credential information and, thus, the brute-force attack is disabled.

5.4 Dictionary Attacks

A *dictionary attack* [2 & 15] is the “smart” version of brute-force attacks and is directed towards finding passwords in a specific list, such as an English dictionary. Dictionary attacks are also executed using automated tools. Moreover, these tools are capable of working on web interfaces without access to the encrypted format of credential information. These tools require the prior knowledge of the user name only. Once given a user name, the attack tool will try all possible combinations of that user name with a huge database (such as a dictionary) of possible passwords. This attack has a high probability of succeeding since we, as humans, tend to use passwords that are easy to remember. The ***attack enabler*** is a “high” number of allowed consecutive unsuccessful authentication attempts.

The ***countermeasure,*** in this case, is to prevent the automation of the attack by setting an upper limit on the allowed number of successive unsuccessful authentication attempts. This can be done through an account auto-lock or a timeout procedure. In other words, when a certain number of consecutive, unsuccessful authentication attempts is reached, the system will automatically lock or disable the account and will alarm the system administrator. This will prevent the dictionary attack from proceeding and, thus, the attack is disabled. Enabling or unlocking the account can be done

either by the user or automatically by the system after a certain period of time. A ***residual vulnerability*** of the countermeasures in this case, account auto-locks or timeouts, occurs when malicious users target them as means for *denial of service attacks* [13, 2, 14 & 1].

5.5 Replay Attacks

A ***replay attack*** occurs when a malicious user captures an authentication sequence that was transmitted through the network by an authorized user, and then replays the same sequence to the server to get himself/herself authenticated [15]. The ***attack enabler*** in this case is, again, access to the communication channel and data sent in clear text format. The proper ***countermeasure*** is to encrypt and time-stamp all sensitive data sent across the communication channel. By doing this, "replayed" messages can be recognized and discarded and this type of attack is disabled.

5.6 Credential Decryption Attacks

Credential decryption is a basic supplementary attack for sniffing attacks, brute-force attacks, and dictionary attacks. A tool whose aim is to break the encryption algorithm that was used to encrypt credential information usually performs these attacks [16].

- ***Attack enablers*** for this attack might be a weak cryptographic algorithm, a weak credential policy, or an incorrect implementation of the cryptographic algorithm.
- ***Cryptography*** increases the probability of success for a brute-force attack or a sniffing attack by allowing the use of cryptographic systems that are easy to crack. Its countermeasure is to use a strong cryptographic algorithm that is hard to crack.
- Please note that a weak cryptography is not the same as a weak credential policy.
- A ***weak credential policy***, on the other hand, increases the probability of a dictionary attack's success by allowing the existence of easy-to-guess passwords.
- Its ***countermeasure*** is to have a strong credential policy that forces legitimate system users to create and maintain a safe password that is easy to remember for a legitimate user and difficult to guess for a malicious user.
- The ***countermeasure*** to an *incorrect implementation of cryptography* is to thoroughly verify the cryptographic algorithm after system implementation. This cannot be done at design time, and is therefore not discussed further in this paper.

5.7 Sides-Channel Attacks

***Side-Channel Attacks*:** In cryptographic devices such as smart cards, data useful to an attacker other than input data and output data may 'leak out' during cryptographic procedures. The computation time of cryptographic procedures is one kind of such data, as is power consumption. Because the smart card uses an external power source, power consumption can be monitored [35] developed a side channel attack in which an attacker infers stored secret information in a cryptographic device by using such leaked data. This type of attack, which includes a timing attack, a Simple Power Analysis (SPA) attack, and a Differential Power Analysis (DPA) attack, renders smart cards particularly vulnerable.

- A ***timing attack*** is a side channel attack in which an attacker infers the secret information by using computation time as leaked data. Some methods of timing attack use statistical analysis to reveal the secret information, others infer it from a one-time computation.
- A ***Simple Power Analysis (SPA) attack*** is a side channel attack in which an attacker infers the secret information by using power consumption as leaked data. An SPA attack captures the secret information by direct observation of a device's power consumption without the need for statistical analysis.
- A ***Differential Power Analysis (DPA) attack*** is a side channel attack in which an attacker infers the secret information by using statistical analysis of power consumption. This attack is the most powerful side channel attack.

For the purpose of our research, smart cards might be used for authentication purposes only at the client side. In this case, the card and the external power source are both assumed to be secure since they are used by the client and not by the EC system itself. The main emphasis of our research is on securing the EC system. In other words, our goal is to secure data transmitted and received by the EC system. Accordingly, this paper will not deal with these types of attacks. Yet, it is important that security architects be aware of the existence of these types of attacks.

6. Applying and Integrating Authentication Design Model

We have applied our methodology to the authentication security feature of the security services model. The result was a countermeasures design model that is effective against the set of all known security attacks related to authentication. In this section, we will apply the derived countermeasures design mode to the e-commerce system design described, in order to check the effectiveness and applicability of the derived authentication countermeasures design model to a realistic e-commerce system design with SET. The first step in applying the derived countermeasures design model is to instantiate its features. This is a straightforward process that takes every feature of the model and converts it into an implementable design. A description of how the authentication model features described are instantiated is provided below. This process is not dependent on any order of instantiation, and therefore, can be done in any desired order.

6.1 Encrypted Channel

The "encrypted channel" feature is instantiated to SSL (Secure Sockets Layer) [18]. SSL is a protocol developed by Netscape Communications Corporation to provide security and privacy over the internet. This protocol supports server and client authentication, is application independent, and allows HTTP (Hypertext Transfer Protocol) to be layered on top of it transparently. Furthermore, SSL is optimized for

HTTP and is able to negotiate encryption keys as well as authenticate the server before the web browser exchanges data. The SSL protocol maintains the security and integrity of the transmission channel by using encryption, authentication and message authentication codes. In our case, SSL is selected because it provides an encrypted channel of communication with no requirements at the client side. The only client requirement is to have an SSL enabled web browser. The majority of existing web browsers, and all major web browsers such as Internet Explorer™, Netscape Communicated™, and Mozilla™, have built-in SSL support.

6.2 Strong Cryptography

The "strong cryptography" feature is instantiated to use the RSA cryptosystem with 1024 bits keys. The reason for selecting RSA is because it is a standard for secure cryptography. RSA Laboratories (http://www.rsasecurity.com/) currently recommends key sizes of 1024 bits for corporate use. Several recent standards specify a 1024-bit minimum for corporate use. Less valuable information may be encrypted using a 768-bit key; as such a key is still beyond the reach of all known key-breaking algorithms.

6.3 Strong Password Policy

The "strong password policy" feature is instantiated to obey the SANS standard [36] for strong password policies as follows:

- All system-level passwords (e.g., root, enable, NT admin, application administration accounts, etc.) must be changed on at least a quarterly basis.
- All user-level passwords (e.g., email, web, desktop computer, etc.) must be changed at least every six months. The recommended change interval is every four months.
- User accounts that have system-level privileges granted through group memberships or programs such as "sudo" must have a unique password from all other accounts held by that user.
- Passwords must not be inserted into email messages or other forms of electronic communication.
- All user-level and system-level passwords must:
- Contain both upper and lower case characters (e.g., a-z, A-Z)
- Have digits and punctuation characters as well as letters e.g., 0-9,!@#$ %^&*()_+|~-=\`{}[]:";'<>?,./)
- Are at least eight alphanumeric characters long.
- Is not a word in any language, slang, dialect, jargon, etc.
- Are not based on personal information, names of family, etc.
- Passwords should never be written down or stored on-line. Try to create passwords that can be easily remembered. One way to do this is create a password based on a song title, affirmation, or other phrase.

For example, the phrase might be: "This May Be One Way to Remember" and the password could be: "TmB1w2R!" or "Tmb1W>r~" or some other variation.

Security-Oriented 'Authorization' Design Model

Step 1 is to select a security feature. Here we select ***authorization*.**

Authorization is the process of giving someone the permission to do or have something. In multi-user computer systems such as EC systems, a system administrator defines which users are allowed access to the system and what privileges of use (such as access to which components, hours of access, and so forth). Assuming that someone has logged in to an EC system, the system may want to identify what resources the user can be given during this session. Thus, *authorization* is sometimes seen as both the preliminary setting up of permissions by a system administrator and the actual checking of the permission values that have been set up when a user requests access.

Authorization models might rely on specific frameworks or models for implementation guidance. An example of such guidance is the *AAA Authorization Framework* which is not intended to be a standard but serves as an asset for modeling authorization into EC systems [19, 20 & 21]. The purpose of this framework is to provide the base requirements for a*uthorization*. It presents an architectural framework for understanding the *authorization* of internet resources and services and deriving requirements for *authorization* protocols.

The above referenced literature emphasizes satisfying standard security requirements. As discussed earlier, the main emphasis in current standards for security requirements is to satisfy legitimate user requirements from a security point of view. Testing against the existence of malicious user requirements is done after the system is implemented. In this paper, we emphasize blocking malicious user requirements at system design time (this is also known as preventive security design [8]. The goal of this section is to have a preventive design model for *authorization* that can be incorporated into any integrated *authorization* model (such as above) or can be implemented as a standalone authorization module in e-commerce systems.

Step 2.1: Identify all attacks related to authorization
Security attacks related to authorization can be identified from the literature and from personal experience. In this paper, we have projected the identified security attacks onto various types of e-commerce *authorization* models described in [22 & 23]. This comprises most known attacks applicable to authentication in the domain of EC systems. It is important to remind that the main focus of this paper will be on attacks directly related to e-commerce systems. Attacks related to network components, to third-party software components, and attacks against the operating system that is supporting the application will not be discussed here. In practice, of course, these types of attacks also have to be handled.

The specific security attacks related to *authorization* in e-commerce systems are [13, 2, 14, 17 & 15]:
Session hijacking attacks

- Authorization bypassing attacks

- Privilege brute-force attacks
- Replay attacks
- ID Spoofing attacks

We will now consider each type of attack, derive attack enablers, and prescribe effective countermeasures.

Step 2.2: For each authorization attack, derive its enablers and countermeasures

This section provides a succinct abstract description of *authorization*-related security attacks. Attack enablers are then identified and effective countermeasures are prescribed. The attacks are presented and discussed below in order of dependence; since some of them are related (e.g. Session hijacking attacks depend on ID spoofing attacks).

6.4 ID Spoofing Attacks

ID spoofing attacks [13, 2, 14, 17 & 15] occur when a malicious user or process claims to be a different user or process. The ***attack enabler*** for this type of attacks on EC systems is the stateless nature of the Internet part of EC systems. Stateless means that there is no record of previous system interactions and that each interaction request has to be handled based entirely on information provided with it.

The proper ***countermeasure*** is to provide a stateful EC system. Stateful means the system keeps track of the state of interaction, usually by setting values in a storage field designated for that purpose. In other words, stateful means the ability to identify the user across multiple EC system requests. In this case, state information can be kept through the usage of session management techniques [22, 19 & 20].

Session Management is a technique that enables web applications, and EC systems in our case, to transform the Internet from a stateless medium to a stateful one. From a security point of view, it is also responsible for protecting the EC system from malicious behavior of application clients whether intentional or non-intentional.

Most popular session management implementations make use of cookies and/or URLs in web browsers to save session information. A *residual vulnerability* for introducing session management is enabling session hijacking attacks. This type of attacks will be detailed later on in this section.

6.5 Replay Attacks

Replay attacks in the case of authorization are similar to a certain extent to the case of authentication. The ***attack enabler*** for this type of attacks on EC systems is the stateless nature of the Internet part of EC systems. The proper ***countermeasure*** in the case of authorization is to provide a stateful EC system through the usage of session management techniques [22, 19 & 20]. This will prevent malicious users from claiming false identities when attempting to seek authorization. Session management implementations make use of cookies and/or URLs in web browsers for saving session information.

6.6 Session Hijacking Attacks

A ***residual vulnerability*** for introducing session management as a countermeasure is that malicious users will be able to execute s*ession hijacking attacks* [13 & 15]. This type of attack involves an attacker using captured, brute forced, or reverse-engineered authorization information (such as session information) to seize control of a legitimate user's session while that user is logged into the EC system. This usually results in the legitimate user losing access or functionality to the current EC system session, while the attacker is able to perform all normal application functions with the same privileges of the legitimate user.

This type of attacks usually relies on a combination of other simpler session management attacks (such as brute-force attacks and replay attacks). The act of taking control of the session after successfully obtaining or generating an authentication token is called session hijacking. The user may or may not still have all or partial control of his EC system session, and may be forced out in the process. An attacker might be able to take control of an active session simply by pasting a URL into his web browser or by loading stolen cookie data and accessing a particular web site or URL (similar to a replay attack).

Session management information is usually encrypted and sent to the client side where web browsers save a copy for later usage. There are three options for saving session information on the client side: using cookies, URLs, or hidden HTML forms. Whether saved in the URL or in a hidden HTML form, session information is clearly seen by anyone. Thus, for the purpose of our discussion, using URLs and hidden HTML forms are similar.

The session hijacking ***attack enabler*** has three properties: weak session encryption, access to cookie information, and weak URL session information. The first ***attack enabler*** property is a weak encryption algorithm that allows malicious users to capture session information, decrypt it, and perform session hijacking attacks. The ***countermeasure*** for this attack enabler property is to have a strong session encryption algorithm. This will prevent malicious users from retrieving useful session information for the purpose of session hijacking attacks in a timely manner.

The second ***attack enabler*** property, access to cookie information, occurs when a session lifetime is longer than the web browser session. Web browsers, in this case, are forced to save cookies on local user hard drives. This allows malicious users, through complex tools, to retrieve saved session information and perform session hijacking attacks. The ***countermeasure*** for this attack enabler property is to use volatile cookies. Volatile cookies are not saved on local user hard drives. They are saved in the system memory, and once the web browser session ends, i.e. the web browser is closed, the cookie is erased from memory and cannot be retrieved.

The third ***attack enabler*** property is using weak URL session information. This usually occurs when cookies are not available and is, practically, similar to saving session information in hidden HTML forms. In this case, session

information cannot be volatile since it can be seen by the human eye either in the URL or in the HTML file. Malicious users can, through special tools, retrieve a copy of the session information and start a session hijacking attack. The proper ***countermeasure*** is to have URL enforcement. This includes but is not limited to re-authenticating the user before critical actions are performed (such as finalizing purchase orders, requesting money transfers, etc.), and mapping session information to web browser instances. By doing so, if a malicious user successfully steals session information and starts a session hijacking attack, our system will either block him by capturing the fact that a different browser instance is used or will prevent him from causing extensive damage by requesting re-authentication before critical actions.

6.7 Buffer Overflow Attacks

A ***buffer overflow*** is possibly the most prevalent software vulnerability used to infiltrate a computer or a system [2]. A buffer is a memory location in which a program stores variable data. This data is often supplied directly by the user, as in a name or other text entry, or by a client program such as a web browser or email client. These buffers are usually of a fixed, predetermined size. If the program restricts the amount of data that can be input to the amount of data that the buffer can store, a buffer overflow is not possible. Some programs do not do this. Buffer overflow occurs when more data is entered than there is space for in the buffer. The extra data then gets written to the memory locations after the buffer. Often, the memory after the buffer location originally contains code, and that memory location is referenced for future execution. A buffer overflow attack is possible when an attacker is able to input data that will cause either a program crash (creating a Denial of Service attack) or cause the system to run code granting the attacker further access.

The ***attack enabler*** for buffer flow attacks is, as discussed above, not restricting the amount of data that can be input to the amount of data that the buffer can store. Restricting the size of data on the client side in EC systems is infeasible because clients are usually web browsers. EC systems cannot depend on web browsers from a security point of view because malicious users have full control of the web browser.

The proper ***countermeasure*** is to use static or dynamic source code analyzers to check the written code for buffer overflow problems. Another possible countermeasure is to change the programming language compiler so that it performs bounds checking for protecting certain addresses from overwriting.

6.8 Denial of Service Attacks

A denial of service attack [24] is characterized by an explicit attempt by attackers to prevent legitimate EC system users from using the system. Examples include attempts to "flood" a network, thereby preventing legitimate network traffic; or attempts to disrupt connections between two machines, thereby preventing access to a certain specific service supporting the EC system. Not all service outages, even those that result from malicious activity, are necessarily denial-of-service attacks. Other types of attack may include a denial of service as a component, but the denial of service may be part of a larger attack. Denial-of-service attacks can essentially disable the EC system or the network that the system resides on. Some denial-of-service attacks can be executed with limited resources against a large, sophisticated site. This type of attack is sometimes called an "asymmetric attack." Denial of service attacks are therefore, related to computer networks. Thus, our research will not deal with this type of attacks. Yet, we are interested in one subsidiary of this type of attacks: component failure attacks.

6.9 Component Failure Attacks

Component failure attacks occur when an EC system component fails to respond to other components while providing a service. Malicious users might execute a denial of service attack on a certain component of the EC system with an attempt to disable it. If successful, malicious users will then attempt to attack the EC system through simpler attacks such as spoofing attacks, replay attacks, and session hijacking attacks. An example of such attacks might occur in the case of centralized authorization.

Malicious users might execute a denial of service attack against the EC system component that provides authorization mechanisms. If the attack fails, malicious users will attempt to perform a session hijacking attack on the EC system using erroneous authorization information.

The *attack enabler* is when system components provide tolerance for erratic trusted components. This might lead to unsafe trust relationships, thus, rendering the EC system behavior unpredictable. The proper *countermeasure* for this type of attack is to have a zero-tolerant trust model. [25] By doing so, components that fail while providing the service will be considered un-trusted and, thus, the attack fails.

6.8 Backdoors

A ***backdoor***, also called a ***trapdoor***, is an undocumented way of gaining access to an EC system. [2] Backdoors (a2), whether intentional or non-intentional, are usually written by the developer who writes the code for the EC system and are often only known by the developer who wrote those [37].

Backdoors might exist in any component of the EC system. Backdoors related to the operating system platform (such as backdoors caused by operating system services like telnet and FTP) are beyond the scope of this paper. For the purpose of our discussion, we are concerned with backdoors that are directly related to our EC system. For this purpose, two types of backdoor are identified: coding backdoors and underlying backdoors. Coding backdoors are usually generated while coding the EC system. Developers might, intentionally or non-intentionally, write code that would provide illegal access to the EC system.

The ***attack enabler*** in this case is having extra code that does not support the proper functionality of the EC system. The proper *countermeasure* is to have good code coverage testing (also known as white box testing). As a result of applying a good code coverage testing process, uncovered

code can be identified and observed. Underlying backdoors are backdoors existing in critical components supporting the EC system such as web servers. An example of such is having weak file access permissions or weak web server aliases.

Weak file access permissions might give unauthorized access to insiders to the EC system. An insider, in this case, will log into the system as a legitimate user, and then exploit these file access permissions to perform his attack. Web server aliases, on the other hand, might give unauthorized access to any type of user through the internet. In this case, a web server alias, created with the intention of simplifying a user's task, might give uncontrolled access to other system components. The ***attack enabler*** in both cases is a weak implementation of the access control policy. The proper ***countermeasure*** is to enforce the access control policy through an explicit check after system installation. By doing so, any existing backdoor will be uncovered and fixed.

7. Applying and Integrating Access Control Enforcement Design Model

We applied our methodology to the access control enforcement security feature of the security services model. The result was a countermeasures design model that is effective against the set of all known security attacks related to access control enforcement. In this section, we will apply the derived access control enforcement countermeasures design model derived to the e-commerce system.

This application is intended to prove the applicability of the derived access control enforcement countermeasures design model. The first step in applying the derived countermeasures design model is to instantiate its features. This is a straightforward process that takes every feature of the model and converts it into an implementable feature. A description of how the access control enforcement model features, are instantiated is provided below.

7.1 Zero-Tolerance Trust Model

Our "zero-tolerance trust model" feature is instantiated to become a role-based access control model [26, 27]. In our discussion of access control enforcement, we showed that one of the most challenging problems in managing large networked systems is the complexity of security administration. Today, security administration is costly and error-prone because system administrators usually specify access control lists for each user on the system individually. Role based access control (RBAC) is a technology that is attracting increasing attention because of its potential for reducing the complexity and cost of security administration. The SET-integrated system can make use of this technology for enforcing its access control policy. Our main interest in RBAC is to be able to apply a "don't trust, verify" access control policy. [25] In our case, this means that some application agent A might make an assertion to agent B. Agent B must verify the assertion before acting on it. If A lies, the false assertion cannot cause much damage. At worst, it causes B to waste resources checking on a false assertion. Furthermore, Security System provides an implementation of an RBAC web server that is helpful for our case study during the implementation phase.

7.2 Access Control Policy Enforcement

The "access control policy enforcement" feature is instantiated to using an automation tool called COPS for checking access permissions to our system implementation [28]. COPS is a collection of programs that each attempt to tackle a different problem area security. For example, some of the checks that COPS performs are:

- File, directory, and device permissions/modes
- Poor passwords.
- Content, format, and security of password and group files
- Scheduled system program files
- A CRC check against important binaries or key files to report any changes therein
- Anonymous ftp setup
- Unrestricted tftp, decode alias in sendmail, SUID uudecode problems, hidden shells inside inetd.conf, rexd running in inetd.conf.
- Dates of CERT advisories vs. key files. These checks the dates that various bugs and security holes were reported by CERT against the actual date on the file in question.

7.3 Code-Coverage Testing

Code coverage testing is a countermeasure that can be performed once the system implementation is done. Yet, planning for this type of testing must be done while the system is still being implemented. Furthermore, test cases required to perform this type of testing are directly related to the system design. Each project must choose a minimum percent coverage for release criteria based on available testing resources and the importance of preventing post-release failures. Clearly, safety-critical software, such as our case study, should have a high goal. We might set a higher coverage goal for unit testing than for system testing since a failure in lower-level code may affect multiple high-level callers [29]. In our case, the "code coverage testing" feature is instantiated to attain 80%-90% code coverage with technical reviews discussing uncovered code before system release.

One might argue that setting any goal less than 100% coverage does not assure quality. Yet, our main interest is to avoid backdoors in our system. Code coverage results of 80%-90%, accompanied with formal technical reviews discussing uncovered code, are enough, in our opinion, to ensure that no backdoors exist in our system.

7.4 Source-Code Analysis

The "source code analysis" feature can be instantiated depending on the programming language used to develop our case study. The purpose of this countermeasure is to avoid buffer overflows in our system implementation. If the Java™ programming language is to be used to develop the system, buffer overflow errors are impossible. The Java™ language simply does not provide any way to store data into memory that has not been properly allocated. If the C/C++ language is to be used to develop the system, then our

"source code analysis" feature can be instantiated to use BOON (Buffer Overrun detection) [30]. BOON is a tool for automatically finding buffer overrun vulnerabilities in C source code and can be downloaded from http://www.cs.berkeley.edu/~daw/ boon/ releases. html.

For the purpose of our case study, Java™ was used to develop the e-commerce system. Thus, no source-code analysis tool is required. Yet, BOON might be required to test any third-party source code involved with SET-integrated modules.

7.5 Access Control Enforcement Summary

Based on the instantiation discussion the derived access control enforcement countermeasures design model:

- Can be directly incorporated into the high-level design document of our case study e-commerce system.
- Contains effective countermeasures against the set of all known security attacks related to access control enforcement
- Provides implementation guidelines to avoid security pitfalls for the different types of security attacks related to access control enforcement.

Security-Oriented Transaction Privacy Design Model

Step 1 is to select a model security feature. Here we select *transaction privacy*.

Government and private systems are increasingly required to maintain the privacy of individuals using EC systems. The ***transaction privacy*** service protects against loss of privacy with respect to transactions being performed by an individual on the EC system. In the concept of *transaction privacy* is defined as a service for preventing unauthorized disclosure of transaction contents, parties involved, location of parties involved, and the exact time of occurrence of the transaction [31].

Transaction privacy, thus, includes the following:

- **Data privacy**: The contents of the transaction must be protected from disclosure to unauthorized parties.
- **Source and destination privacy:** The parties involved in the transaction should not be revealed to unauthorized parties.
- **Location privacy:** The location of the parties performing the transaction should not be disclosed to unauthorized parties.
- **Time privacy:** The exact time when a transaction occurs should not be disclosed to unauthorized parties. Security architects may depend on research approaches as an asset to derive *transaction privacy* models. An example of such a research is the SET specification document [32] that serves as an open standard for protecting the privacy, and ensuring the authenticity, of electronic transactions.

The above referenced literature emphasizes satisfying standard security requirements. As discussed earlier, the main emphasis in current standards for security requirements is to satisfy legitimate user requirements from a security point of view. Testing against the existence of malicious user requirements is done after the system is implemented. In this paper, we emphasize blocking malicious user requirements at system design time (this is also known as preventive security design [8]. The goal of this section is to have a preventive design model for *transaction privacy* that can be incorporated into any integrated *transaction* model (such as above) or can be implemented as a standalone *transaction privacy* module in e-commerce systems.

Step 2.1: Identify all Attacks Related to Transaction Privacy

Security attacks related to *transaction privacy* can be identified from the literature and from personal experience. In this paper, we have projected all known security attacks onto various types of e-commerce access control models and frameworks described [33, 34 & 32]. This comprises most known attacks applicable to *transaction privacy*. It is important to remind that the main focus of this paper will be on attacks directly related to e-commerce systems. Attacks related to network components, to third-party software components, and attacks against the operating system that is supporting the application will not be discussed here. In practice, of course, these types of attacks also have to be handled.

The specific security attacks related to *transaction privacy* in e-commerce systems are [13, 2, 14 & 17]:

- **DBMS exploits, or attacks targeted towards exploiting security of Data Base Management Systems**
- **Log data mining attacks, also known as log data analysis attacks**
- **Sniffing attacks, also known as man-in-the-middle attacks**

We will now consider each type of attack, derive attack enablers, and prescribe effective countermeasures.

Step 2.2: For each Transaction Privacy Attack, Derive its Attack Enablers and Countermeasures

This section provides a succinct description of all known transaction privacy related security. Attack enablers are then identified, and proper countermeasures are prescribed. The attacks are presented and discussed below in order of dependence, since some of them are related (e.g. DBMS exploits attacks depend on log data mining attacks).

7.6 Sniffing Attacks

As described earlier, s*niffing attack* [13, 2, 14 & 17] (also known as the man-in-the-middle attacks) are the digital analogues to phone tapping or eavesdropping. This attack captures information as it flows between a client and a server. In the case of transaction privacy, such attacks might be successful at retrieving transaction information while the transaction is being performed. The ***attack enabler*** in this case is the process of sending data across communication channels in clear text format. Preventing access to the communication channel is not a valid countermeasure in this

case due to the open nature of the internet. By encrypting the communication channel between the user/process and the system, sniffing attacks are defeated, i.e., sniffing cannot retrieve any useful information. Yet, encryption might have an impact on the EC system from a performance perspective.

Thus, the ***proper countermeasure*** for this attack is to encrypt the transaction information itself, at least, instead of encrypting all data being communicated between the client and the EC system.

7.7 Log Data Mining Attacks

Data mining is the search for significant patterns and trends in large databases by using sophisticated statistical techniques and software. This provides information crucial to helping businesses and industries improve products, marketing, sales, and customer service. Similarly, ***log data mining attacks*** can be targeted towards retrieving private transaction information from EC systems' log data files. The *attack enabler* for this type of attack has two properties: having access to the log data file, and logging sensitive transaction information in the log data file.

The first ***attack enabler*** property, ***physical access to the log data file***, allows malicious users to retrieve a copy of the logged data and perform log data mining attacks.

The proper ***countermeasure*** for this attack enabler property is to enforce access permissions on the log data file at the underlying operating system level. By doing so, malicious users will not be able to have access to the log data file and, thus, the attack is disabled.

The second ***attack enabler*** property, logging sensitive information, allows ***log data mining attacks*** to retrieve useful transaction information in case malicious users gain access to the log data file. This is highly probable in the case of malicious users categorized as insiders.

The proper ***countermeasure*** for this attack enabler property is to prevent the EC system from logging sensitive data. This will prevent malicious users from succeeding in retrieving useful transaction information. In case sensitive data logging is required, database management systems (DBMS) must be used to save this type of data.

A ***residual vulnerability*** for using DBMS to log sensitive data is the fact that the EC system security will be dependent on the security of the DBMS. This residual vulnerability is discussed next.

7.8 DBMS Exploits

As discussed above, relying on Data Base Management System (DBMS) security is considered a residual vulnerability. This is because malicious users might be able to exploit the EC system by exploiting the DBMS itself.

The ***attack enabler*** **i**n this case might be any exploit in DBMS security.

The proper ***countermeasure*** is to enforce security at the DBMS level by keeping it up-to-date with security fixes and patches. This will help prevent malicious users from exploiting our EC system by exploiting the DBMS system that we rely on for transaction privacy in general, and sensitive data logging in particular.

8. Applying and Integrating Transaction Privacy Design Model

We applied our methodology to the transaction privacy security feature of the security services model. The result was a countermeasures design model that is effective against the set of all known security attacks related to transaction privacy.

In this section, we will apply the derived transaction privacy countermeasures design model derived in to the e-commerce system.

This application is intended to prove the applicability of the derived transaction privacy countermeasures design model.
The first step in applying the derived countermeasures design model is to instantiate its features. This is a straightforward process that takes every feature of the model and converts it into an implementable feature. A description of how the transaction privacy model features are instantiated is provided below.

8.1 Encrypted Channel / Transaction Information

The "encrypted channel / transaction information" feature is instantiated to use SET encryption [32]. SET uses symmetric encryption, Data Encryption Standard (DES), as well as asymmetric, or public-key, encryption to transmit session keys for DES transactions. Although this has disturbing connotations for a "secure" electronic transaction protocol because public key cryptography is only used to encrypt DES keys and for authentication, and not for the main body of the transaction, the computational cost of asymmetric encryption is cited as reason for using weak 56 bit DES. Other reasons such as export/import restrictions and the perceived need by law enforcement and government agencies to access the plain-text of encrypted SET messages may also play a role.

For the purpose of instantiating "encryption", our case study SET-integrated system makes use of SET modules. These modules are responsible for encrypting transaction information while the transaction is in progress. Furthermore, any modification to these modules at the encryption level might lead, in most cases, to incompatibility problems with other SET-enabled parties. Thus, the only instantiation possible in this case is to use SET encryption as specified [32].

8.2 Saving Sensitive Data in DBMS

While discussing SET, we described specific cases where SET reveals credit card information to merchants. Our typical system scenarios also showed that order capture tokens and receipts are involved in a SET transaction. This

type of information is considered sensitive as it contradicts with two security objectives: privacy and confidentiality. On the other hand, a merchant system requires saving this type of sensitive information for future use. Thus, protecting this information is of high importance to ensure system security. After identifying the sensitive information involved in a transaction, our security feature of using the DBMS for saving sensitive transaction information is instantiated to become "using the DBMS for saving credit card information, order capture tokens, and receipts".

8.3 Not Logging Sensitive Information

The "don't log sensitive information" feature is kept as is. This feature will serve as a guideline for the system implementation and will be performed during the system testing phase. The fact that this feature is built into the system security design model better is by itself an enhancement to system security. This will help avoid discovering defects during the system testing phase that are related to the system design model. The only part to be instantiated is the term "sensitive information." Based on the discussion this feature is instantiated to become "don't log credit card information, order capture tokens, and receipts."

8.4 Enforce DBMS Security

The "enforce DBMS security" feature is instantiated to become the process of keeping the DBMS up-to-date with security fixes and patches. This, of course, requires the prior selection of a stable DBMS having a clean history with security issues. While it is not our direct concern to secure the selected DBMS, we must make sure a secure one is selected in the system specification. This was the main purpose of having this security feature in the countermeasures design model in the first place.

8.5 Log File Access Enforcement

The "log file access enforcement" feature is instantiated to become using COPS to ensure proper log file access permissions [28]. This step must be done after the system is implemented and installed. Yet, the reason for having it as a feature in the transaction privacy countermeasures design model is to avoid having defects related to the system specification during the system testing phase. By including this feature in the model, the system specification will be aware of including it as a guideline for system implementation and testing. The reason for selecting COPS as the tool to automate this step is because we already use COPS to check file access permissions in the instantiated access control enforcement model.

Transaction Privacy Summary
Based on the instantiation discussion the derived transaction privacy countermeasures design model is:

- Can be directly incorporated into the high-level design document of our case study SET-integrated e-commerce system.
- Contains effective countermeasures against the set of all known security attacks related to transaction privacy
- Provides implementation guidelines to avoid security pitfalls for the different types of security attacks related to transaction privacy.

9. Conclusions

This paper describes a new methodology for deriving countermeasures design models for e-commerce systems. The methodology is based on the e-commerce security services model. Our approach focuses on satisfying legitimate user requirements while blocking malicious user requirements at system design time. This assessment shows that our methodology is systematic through a case study that derived four countermeasures design models for authentication, authorization, access control enforcement, and transaction privacy. The derived countermeasures design models were assessed through a realistic case study on a SET-integrated in e-commerce systems. These models were also proven to be effective against all security attacks related to the e-commerce domain. The primary benefits of our research are as follows:

i. A comprehensive matrix listing and mapping all known security attacks to four security features in e-commerce systems; namely authentication, authorization, access control enforcement, and transaction privacy.
ii. Four new security models that extend the e-commerce security services model for ecommerce systems. These models are proven to be effective against all known security attacks related to e-commerce systems.
iii. A faithful implementation of a countermeasures design model was proven to be guaranteed to block all known security attacks related to that feature.
iv. Security architects can avoid expensive system development life cycles fixes. This is achieved by having an effective countermeasures design model that is directly applicable to EC systems and that specifies detailed requirements for the security feature.
v. A cost-effective, systematic methodology for deriving countermeasures design models for the other security features of e-commerce systems.
vi. An overview of all known security attacks related to the four security features discussed in this thesis; namely ***authentication, authorization, access control enforcement,*** *and* ***transaction privacy.***

10. Future Scope of Work

Little research has been done on methodologies and approaches for designing secure e-commerce systems. Therefore, many research opportunities are still available.

Further research is needed to optimize our methodology and countermeasures design models.

In particular, further studies should:
(i) Apply our methodology and approach to the remaining features of the e-e-commerce security services model.
(ii) Further enhance the methodology to map other security-related features, such as impact on performance, into the design process.

(iii) Formally describe the methodology and provide automation.
(iv) Apply the methodology to other standard security models once available.
(v) Update the derived countermeasures design models with new security attacks once available.

Most of these questions were raised during our research. We were not able to answer these questions because of necessary constraints on the scope of the work. However, we believe we have established a foundation for such future research.

11. Acknowledgement

The authors are thankful to Institute of Engineering & Technology, Sitapur Road, Lucknow, UP and Integral University, Lucknow, UP for providing laboratory and other facilities.

References

[1] G. Treese and L. Stewart, “Designing Systems for Internet Commerce”, pp. 209- 294, Addison-Wesly, 1998. ISBN: 0-201-57167-6.
[2] J. Viega and G. McGraw, “Building Secure Software”, Addison-Wesley, 2002. ISBN: 0-201-72152-X.
[3] National Institute of Standards and Technology (NIST), Special Publication 800-33, “Underlying Technical Models for Information Technology Security”, 2001. website:http://csrc.nist.gov/publications/nistpubs/800-33/sp800-33.pdf.
[4] K. Chang, B. Lee, and T. Kim, “Open Authentication Model Supporting Electronic Commerce in Distributed Computing”, Electronic Commerce Research, Vol 2, pp. 135-149, 2002, Kluwer Academic Publishers.
[5] OMG, “CORBA Services: Common Object Security Specification.” Available at ftp://ftp.omg.org/pub/docs/security/99-12-02.pdf.
[6] J. Kohl and C. Neuman, “The Kerberos Network Authentication Service (V5)”, Network Working Group, RFC 1510, September 1993, ftp://ftp.isi.edu/in-notes/rfc1510.txt.
[7] M. Bellare, R. Canetti, and H. Krawczyk, “A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols”, 1998, ACM 0-89791-962-9 98.
[8] E. Jonsson, “An Integrated Framework for Security and Dependability”, Proceedings of the 1998 workshop on New security paradigms, pp. 22-29, ACM Press, 1998, ISBN: 1-58113-168-2.
[9] S. Wilson, “Certificates and trust in electronic commerce”, Information Management & Computer Security, Vol. 5, No. 5, 1997, pp. 175-181, MCB University Press. ISSN: 0968-5227.
[10] W. Ford and M. Baum, “Secure Electronic Commerce”, Prentice Hall, 1997. ISBN 0- 13-476342-4.
[11] G. Agnew, “Cryptography, Data Security, and Applications to E-commerce”, Electronic Commerce Technology Trends Challenges and Opportunities, Feb. 2000, IBM Press, pp. 69-85. ISBN: 1-58347-009-3.
[12] S. Hawkins, D. Yen, and D. Chou, “Awareness and Challenges of Internet Security”, Information Management & Computer Security, Vol. 8, No. 3, 2000, pp. 131-143, MCB University Press. ISSN: 0968-5227.
[13] P. Herzog, “The Open Source Security Testing Methodology Manual”, version 1.5, May 2001, website: http://ideahamster.org/.
[14] H. Nguyen, “Testing Applications on the Web”, pp. 285-310, John Wiley & Sons, 2001. ISBN: 0-471-39470-X.
[15] R. Anderson, “Security Engineering: A Guide to Building Dependable Distributed Systems”, John Wiley & Sons, 2001. ISBN: 0-471-38922-6.
[16] SecuriTeam Security Tools Archive. Website: http://www.securiteam.com/tools/archive.html.
[17] B. Schneier, “Secrets and Lies: Digital Security in a Networked World”, John Wiley & Sons, 2000. ISBN: 0-471-25311-1.
[18] Freier, Karlton, P., and Kocher, P., “The SSL Protocol Version 3.0”, Nov. 1996. Website: http://wp.netscape.com/eng/ssl3/draft302.txt.
[19] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence “AAA Authorization Framework”, Network Working Group,RFC2904, August 2000, website: http://www.faqs.org/rfcs/rfc2904.html.
[20] J. Vollbrecht, Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M. and D. Spence, "AAA Authorization Application Examples", Network Working Group, RFC 2905, August 2000, website: http://www.faqs.org/rfcs/rfc2905.html.
[21] S. Farrell, et al., "AAA Authorization Requirements", Network Working Group, RFC 2906, August 2000, website: http://www.faqs.org/rfcs/rfc2906.html.
[22] V. Atluri and A. Gal, “An Authorization Model for Temporal and Derived Data: Securing Information Portals”, ACM Transactions on Information and Systems Security, Vol. 5, No. 1, Feb. 2002, pp. 62-94.
[23] H. Wedde and M. Lischka, “Modular Authorization”, 2001, ACM 1-58113-350- 2/01/0005.
[24] CERT Coordination Center, website: http://www.cert.org/.
[25] C. Kahn, “Tolerating Penetrations and Insider Attacks by Requiring Independent Corroboration”, ACM Press, 1998, ISBN:1-58113-168-2.
[26] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli, “Proposed NIST Strandard for Role-Based Access Control”, ACM Transactions on Information and Systems Security, Vol. 4, No. 3, August 2001, pp. 224-274.
[27] P. Bonatti, S. Vimercati, and P. Samarati, “A Modular Approach to Composing Access Control Policies”, 2000, ACM 1-58113-203-4/00/0011.
[28] Farmer, D., “Computer Oracle and Password System”, May 1993. Website: http://www.fish.com/cops/.
[29] Cornette, S., “Code Coverage Analysis”, Oct. 2002. Website: http://www.bullseye.com/webCoverage.html.
[30] Wagner, D., Foster, J., Brewer, E., and Aiken A., “A First Step towards Automated Detection of Buffer Overrun Vulnerabilities”, Network and Distributed System Security Symposium 2000. Website: http://www.cs.berkeley.edu/~daw/papers/overruns-ndss00.ps.

[31] C. Candolin and H. Kari. “Time Privacy of Electronic Transactions”. Proceedings of the Helsinki University of Technology Seminar on Network Security, 2000, website: http://www.tcm.hut.fi/Opinnot/Tik-110.501/2000/papers/candolin.pdf.
[32] SET Secure Electronic Transaction Specification, Book 1: Business Description, Version 1.0, May 31, 1997. Website: http://www.setco.org/download/set_bk1.pdf.
[33] L. Bouchard, “Securing Transactions and Ensuring Non-Repudiation: The Experience of the Department of Justice in Quebec”, Electronic Commerce Technology Trends Challenges and Opportunities, Feb. 2000, IBM Press, pp. 339-354. ISBN: 1-58347-009-3.
[34] C. Liddy and A. Sturgeon, “Seamless Secured Transactions”, Information Management & Computer Security, Vol. 6, No. 1, 1998, pp. 21-27, MCB University Press. ISSN: 0968-5227.
[35] C. Kocher, “Cryptanalysis of Di?e-Hellman, RSA, DSS, and Other Systems Using Timing Attacks” Website: http://www.cryptography.com/
[36] “Password Policy”, website: http://www.sans.org/ newlook/resources/policies/Password_Policy.pdf.
[37] George Mason University Center for Secure Information Systems, “Security Glossary”, website: http://www.ise.gmu.edu/~csis/glossary/