Towards Building Secure Software Systems

A. S. Sodiya, S. A. Onashoga, and O. B. Ajayi
University of Agriculture, Abeokuta, Nigeria

sinaronke@yahoo.co.uk bookyy2k@yahoo.com
tayoajayi@yahoo.co.uk

Abstract

Software security breaches are now very extremely common and a larger percentage is caused by software design defects. Since individuals and organizations now completely depend on software systems for their day-to-day operations, it is then important to produce secure software products. This paper discusses the problems of producing secure software products and provides a model for improving software security. The model – Secure Software Development Model (SSDM), is unified model that integrates security engineering with software engineering so as to ensure effective production of secure software products. Supporting structure in form of laws is also presented to guide developers throughout the development process. We then present our experience that validates the model.

Keywords: Security breaches, Software system, Software security, Software design, Design defects

Introduction

Computer software systems are increasingly faced with both internal and external penetrations. One major reason for this is the fact that software systems are still with development defects which still make them to be vulnerable. This has brought issue of security into sharp focus because organisations, including governments, depend largely on software systems for their day-to-day operations. The case is even more sensitive in environments where software systems are used for critical missions. This is why building secure software is gaining attention of today's business world and researchers in field of security. In addition, because customers (organizations) have experienced unfortunate security incidence, there is increase awareness and agitation for secure software products.

However, in building secure software systems, a lot has to be done. Security techniques have to be implemented in all the stages of the software engineering. Devanbu and Stubblebine (2000) stated that security concerns must inform every phase of software development, from requirements to design, implementation, testing and deployment. This is necessary because software developer might unknowingly inject defects in all stages of the development process. Microsoft found out that 50% of software security problems were caused by design flaws (McGraw, 2003). Wilander and Gustavsson (2005) reported that, in 2004, more than new security vulnerabilities were found in commercial and open source software everyday. Jones (2000) reported the software benchmark studies conducted

on hundreds of software projects and stated that the average specification, design, and implementation defects content of released software varies from 1 to 7 defects per thousand lines of new and changed code produced. Commonly computer systems are hacked by exploiting software bugs. Redwine and Davis, (2004) stated that no existing processes or practices have currently shown to consistently produce secure software. If there is no adequate security, the availability, reliability and safety of the software are not guaranteed.

Consequently, software development process must be carefully engineered and integrated with security requirements. Common development practices must change so as to produce software with few or no security weaknesses. The ultimate challenge for software engineers is then to develop software systems with desired quality, within the reasonable time and budget, and the software must be secure. Wilander and Gustavsson (2005) stated that to build more secure software, accurate and consistent security requirements must be specified. It is therefore important to continue to seek for ways of improving security of software systems. In this paper, we discuss the technical issues of software security and provided the model and support for improving software security.

The rest of this paper is organized as follows. The next section presents review of existing literature in software security. Software security issues are presented after that, followed by a discussion of the architecture for improving security of software systems. Supporting structures for secure software systems are discussed in the fifth section. A case study follows, and future work and conclusions are presented in the last section.

Overview of Existing Literature

Recently, there has been a lot of interest in building software securely. Pauli and Xu (2005) provided a threat-driven architectural design of secure information systems. The work presented threat modeling using misuse cases and stated that the findings can be used in detailed design and validation of implementation of software system. Wilander and Gustavsson (2005) mentioned in their own work that security requirements are normally poorly specified due to these three things: inconsistency in the selection of requirements, inconsistency in the level of detail and almost no standard requirements on some security solutions. The result of the study shows that security mainly treated as a functional aspect composed of security features such as login, backup, and access control. They stated that requirement on systems through assurance measures are left out. Devanbu and Stubblebine (2000) in their work emphasized that software engineering and security engineering must be integrated in order to have secure software systems. The two software articles of Ghosh, Howell, and Whittacker (2002) and Mead and McGraw (2003) also emphasized the need to build software securely from the ground.

Apart from stating and emphasizing the security needs of software system, many researchers have actually proposed methods for improving software security. One method is the correctness-by-construction proposed by Praxis Critical Systems Limited (Hall & Rod, 2004). The method operates on the principles that errors should not be introduced in first place and that errors should be removed as close as possible to the point they are introduced. The method also incorporates formal notations to specify system and design components with review and analyses for consistency and correctness. Hall and Rod (2004) reported that correctness-by-correction method produced defect densities ranging from 0.04 to 0.75 defects per thousand lines of code. Another method for achieving secure software system is the Cleanroom (Linger & Stacy, 2004). Cleanroom incorporates incremental development, functional-based specification and design, correctness verification, and statistical testing. Redwine and Davis (2004) reported the overall performance of application of cleanroom as ranging from 0.1 errors/KLOC (KLOC means Kilos of Lines of Codes) with full application to 0.4 defects/KLOC with partial application. Capability Maturity Models (CMMs) have also been used as process models to guide organizations in improving the capabil-

ity to perform a particular process. An example of security related CMM is Systems Security Capability Maturity Model (SSE-CMM), which was presented in Hefner (1997). Golderson and Gibson (2003) reported that CMMs have also helped in the overall reduction in design and implementation defects of software products.

Similar to CMM is threat modeling. Threat modeling is used to analyse potential threat to computer systems in order to guide against attacks or penetrations. The usefulness of threat modeling has demonstrated in some previous works (Mogilevsky, Lee, & Yurcik, 2005; Myagmar, Lee, & Yurcik 2005). Attack trees or graphs have also been used in determining what security measures to deploy in a system (Schneir, Lippman, & Wing, 2002; Sheyner & Wing, 2004). However, attack trees model a selected set of attacks via a finite state machine and feasible only in small scenarios. They also require compiling a list of potential threats before generating attack trees. The empirical performance of both threat modeling and attack trees are not readily available.

Another method for improving security of software is the Software Engineering Institute's Team Software Process (TSP) (Davis & Mullaney, 2003). The process incorporates the idea of managing and removing specification, design and implementation defects throughout development lifecycle, controlling and monitoring of process, and using predictive measures of removing defects. Davis and Mullaney (2003) reported an average of 0.06 delivered design and implementation defects per thousand lines of codes produced.

It has shown from the literature that software systems are still with defects (though minimal in some situations). The fact is that an attacker only needs to find one security flaw to compromise the whole system. It is therefore important to design a bug-free software because the implication of a bug might be catastrophic and results in loss of large amounts of money.

Software Security Issues

Software security is concerned with protection of software and its resources. A security risk is the probability of sustaining a loss of a specific magnitude during a specific time period due to a failure of security system (Rodgers, 2002).

Software Security Goals

The three security goals of computer system are Confidentiality, Integrity and Availability. These goals are commonly referred as the CIA of computer security (Sodiya, Longe, & Akinwale, 2004) and these goals also apply to computer systems.

a. ***Confidentiality***:- This has to do with the prevention of unauthorized disclosure of software resources such as codes, data, documents, files, GUI, and so on.

b. ***Integrity***:- This is prevention of unauthorized modification of software resources.

c. ***Availability***:- This is concerned with the unauthorized denial of the services of software resources.

Over the years, there have been several security mechanisms to achieve these goals. Some of these mechanisms are authentication and authorization, access control, encryption, and so on, but the rate and magnitude of attacks on computer system is increasingly alarming. Some of the interests of attacker are financial and knowledge gains, malicious intention, competitive edge and so on. The situation is even worse now that we have mobile codes in distributed network and exposure to Internet.

Problems of Producing Secure Software

Producing secure software is complex and requires high integration of security and software engineering. Based on our experience, we present the problems of producing secure software as follows:-

a. ***Attackers' Knowledge***:- As software security continue to gain the attention of security experts and organizations, the tools and techniques used by attackers are becoming increasingly sophisticated and invasive.

b. ***Complexity of software***:- Software development is a complex process because it involves many activities and specialties. It involves different units, stages and personalities, and all these have to be integrated to achieve successful software production. The situation is even worse with the connection to Internet and the use of mobile codes.

c. ***Security Education***:- Producing secure software requires a lot of security training, education and experience. Many software developers are not well grounded in computer security. To what extent of the techniques and tools of attackers do they understand? Also, many programming books do not teach how to write secure programs. Many curricular of schools offering computer science do not properly address computer security.

d. ***Attitude of Software Engineers***:- In the past, software developers were only interested in producing software product with the desired quality. Little attention was paid on making secure products. But, the situation is changing now that everybody is aware of the implications of vulnerabilities in software.

e. ***Inadequacy of Computer Security Models***:- Many security models are still not adequate for producing secure software products. Implementing some of them is complex and difficult to realize.

Secure Software Development

It has been said that producing software requires integrating Software Engineering (SE) process with Security Engineering. In doing this, a careful understanding of software development process is important. We now present an overview of software development process.

Overview of Software Development Stages

The summary of stages of software development is:

a. Requirement definition

- ❖ Fact findings
- ❖ Investigating the current system
- ❖ Modeling the current system
- ❖ Logical models of the required system

Result: Requirement specification

b. Design

- ❖ Structuring and partitioning of the design into software sub-components
- ❖ Detailed design of the components
- ❖ Formation of the system structure and relationships

Result: Design document containing system abstraction and their relationships

c. Coding

- ❖ Programming components of the system
- ❖ Unit testing

Result: Tested software subcomponents

d. System testing

- ❖ Integrating subcomponents together
- ❖ Integration testing
- ❖ System testing to ensure that the system meets its requirements

Result: Software product

e. Implementation and Maintenance

- ❖ Installing the system in `live` environment
- ❖ Training the users
- ❖ Maintenance
- ❖ Implementing enhancements as demanded by changes in the environment and users

Result: Working software product

Secure Software Development Model (SSDM)

SSDM is a model that integrates security engineering with software development process. As shown in Figure 1, it is a unified model that combines some existing software security techniques. It is structured towards creating secure software products. The dotted lines in Figure 1, shows continuous links along an engineering path. The model shows clearly how software development should be linked to security engineering in order to produce secure software products.

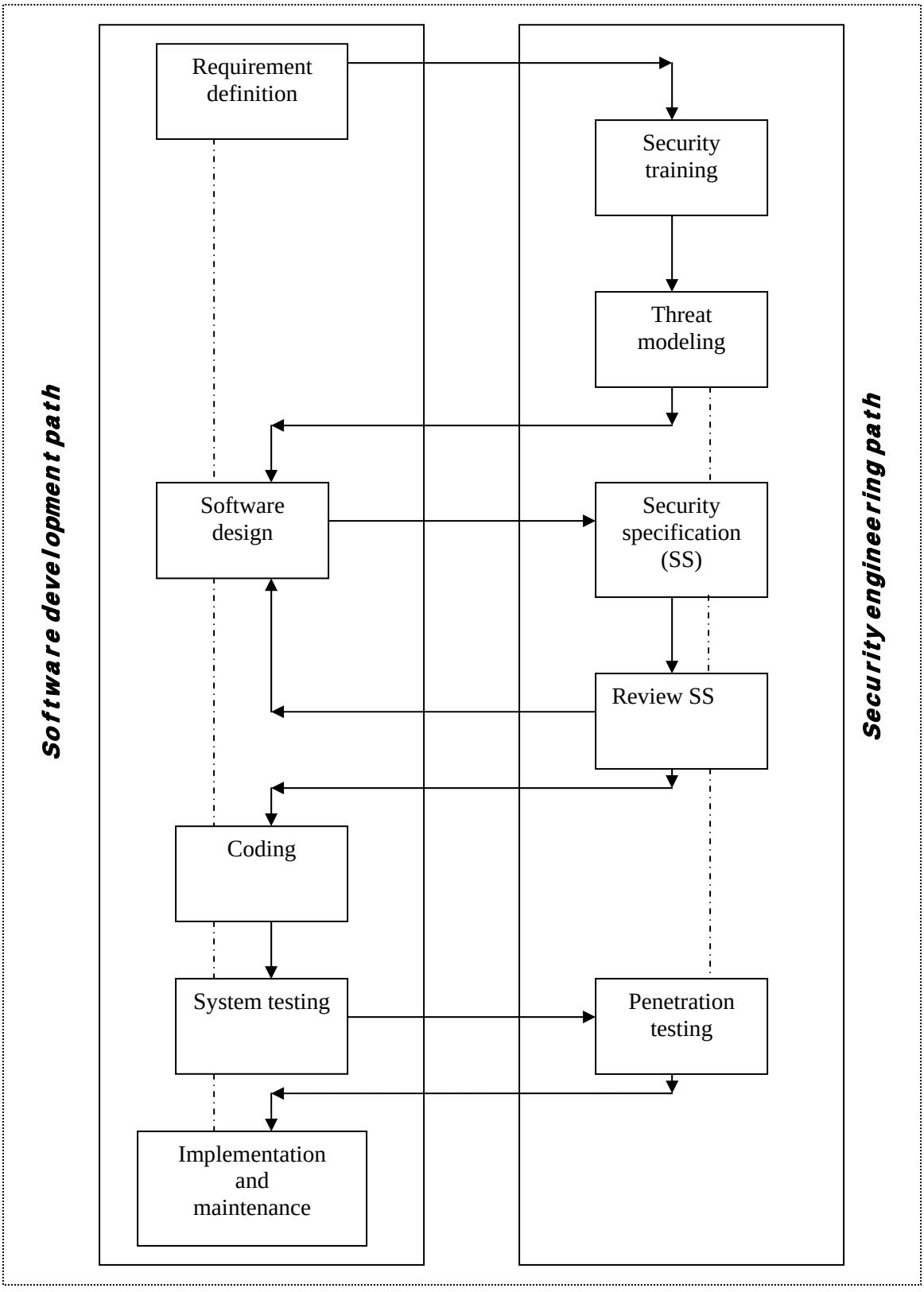


Figure 1: Secure Software Development Model (SSDM)

SSDM security engineering process

As shown in Figure 1, the security engineering path is divided into five stages.

a. ***Security Training***

Not all software engineers have adequate knowledge of computer security. The essence of security training is to provide adequate security education to stakeholders in software development. The key training requirements are:

- *Security awareness*:- This involves educating software engineers on general security concepts. Cases on previous security breaches and their consequences should be presented to them in order to appreciate the need for adequate protection of software products.
- *Knowledge of attackers on previous related applications*:- Software engineers should be educated on nature of previous attacks. There is need to understand attackers' techniques and tools and implications so as to properly understand attackers' behaviours. There is also need to understand penetration techniques of actions like logic bombs, trojans, virus, worms, and so on.
- *Understanding attackers' interests on software being developed*:- Software developers must be able to study a software system and identify software resources that might be of interest to potential attackers. This is important so as to develop techniques for protecting these resources.
- *Knowledge about secure development practices*:- Software developers should also be educated on previous practices on secure software development and their strengths and weaknesses so as to move secure software development forward.

b. ***Threat Modeling***

A threat model is used to effectively and comprehensively identify attackers and their capabilities. Every software development that must be secure must have it own threat model because the common security criteria might not be suitable for all software products.

SSDM threat model is divided into three parts. The classification is similar to what is presented in Myagmar et al., 2005, but our own seems to be more detailed.

i. Understand the nature of the software

This requires understanding:

- *The type of the software*:- The type of a software system determines the magnitude and nature of resources on the software system.
- *Complexity of the software*:- Provision of adequate security to big and complex system is a huge challenge. Complex software systems need more comprehensive threat analysis and modeling in order to capture potential attacks effectively.
- *Other associating software*:- The type and complexity of other software systems that are handshaking with the software under consideration must be investigated. The level of security of the associating software must be ascertained.

ii. Identify attackers/threats

The major issues here are:

- *Assessing the operating environment*:- There is need to study the environment in which the software must operate in order to identify the possible potential attackers and their categories.

- *Identifying the goals/targets of different category*:- Based on previous experience, we must be able to identify the possible interests and goals of attackers in the software.
- *Identifying techniques and tools used by attackers*:- It is also important to know the possible techniques and tools to be used by these attackers in order to develop techniques to block them.
- *Identifying possible future patterns/behaviours of attackers*:- Ensuring that software product are secured is not to think about the present situation alone, but to continue to plan ahead because of the changing nature of attackers' techniques and tools. There is need to identify future behaviours of attackers and undertake some measures to prevent them.
- *Constructing attack profile*:- An attack profile consisting of potential attacks, goals or target of attackers, techniques and tools of attackers and future behaviours of attackers, should be constructed.

iii. Identify possible vulnerabilities

Based on the information on the previous attacks, all likely vulnerabilities that may be introduced into the system must be identified. This will enable the software engineers not to carryout the design in a way that the vulnerabilities would be avoided.

c. ***Security Specification (SS)***

This entails stating the guidelines and procedures that guarantee security of the system The SS should contain the following:-

i. ***Security needs***:- This involves stating list of attacks to guide against and construction of vulnerability profile

ii. ***State security policies***:- This involves a clear definition of how to guide against attacks. Some specific issues to be included are:

- Avoiding errors through out the development
- Correct all errors at the time of first notice
- Employing programming tactics that prevent vulnerabilities
- Employing module integration method that guarantee security

iii. How to coordinate security implementation

iv. How to make the system to adapt to the changing landscape of security

v. How to monitor security postures of the software system

d. ***Review SS***

This is to check whether the design content of the software is in accordance with the SS. This is important because if the design is defective, then the software design must incorporate the security specifications. If the design has not guaranteed the items in the SS, then the design will be reviewed.

e. ***Penetration Testing***

Capabilities of the software in preventing attacks are tested here. It is important so as to:

- Test the security of the software and its resources

- Determine if the current security posture of the software is actually detecting and preventing attacks.

Penetration testing is carried out in a way that all identified attacks and future attack patterns are initiated on-line into the software, and the capability of the software in preventing these attacks is monitored.

Laws of Software Security

Based on our experience from the literature, and as security researchers and consultants, we present some basic laws as supporting structure towards producing secure software products. All stakeholders in software development must obey these laws in order not to introduce vulnerabilities into the system and ensure the production of secured software system. It is believed that if the software engineers have these laws at the back of their minds throughout the stages of the software production, it will ensure efficient production of secure software product. These 10 proposed laws are:-

a. ***Awareness law***:- It states that software developers must constantly update their security knowledge with new information about security.

b. ***Do- it- correct law***:- It states that any activity or process or task to be carried out by any developer should be done correctly first time and every time.

c. ***Patching law***:- It states that no error must be patched. Any error must be removed immediately it is recognized and it must be ascertained that the error will not resurface again.

d. ***Self Evaluation law***:- It states that every software developer should evaluate his/her work at the end of each process or task.

e. ***Security Inclusion law***:- It states that software engineers should bear in mind that security is a subset of quality.

f. ***None Rigidity law***:- It states that security requirements must not be rigid but must be realized. It must be rigid in the sense that suggestions about security issues should be welcomed. The details of the security specifications must be realized.

g. ***Protection law***:- It states that security engineering process itself must be secured. SS must be secret and known only to concerned individuals alone. SS must be guarded with integrity throughout the software engineering process.

h. ***Explicit law***:- It states that the details of SS must be clear and concise for easy implementation. Software developers must be able to understand clearly all issues in the SS.

i. ***Total Security law***:- It states that if software product P is secure, then all other software that must interact with P must also be secure.

Case Study

An accounting system called “Standard Accounting” or “SA” implements SSDM security engineering principles. Standard Accounting was an in-house design, which was implemented on intranet of an organization in Nigeria. The components of Standard are general ledger, sales and purchase ledger, and payroll. The development of Standard was conceived and started when the organization realized there were frequent security breaches on the old accounting package. The Accounting system of the organization operates in a way that different personnel from different units of the organization have access to it. The users of SA have privileges to use some resources

of the package and some of them misuse their privileges. Consequently, there were several security breaches reported with the old accounting package. Some of these security breaches are:

- Change of information in the salary table
- Stoppage of loans
- Cancellation of purchase records
- Total collapse of the system (Denial-of-service)

A nine-member team developed and implemented SA. This comprises of 6 internal staff of the organization and 3 external consultants for supervision of the project and as security experts. The software was designed using Oracle.

During the use of the old package, the security breaches recorded were collected and categorized according to computer security goals - CIA. There were total of 129 security breaches recorded at this time. The design of SA was completed about a year ago and was implemented at the same time. We have been monitoring the implementation of SA and have not recorded any security breaches since that time. Table 1 shows the summary of the security improvement by the implementation of SSDM.

Table 1: Performance validation of SSDM

S/No	Security threats classification	Old package	New (Standard)
1	Confidentiality	69	-
2	Integrity	42	-
3	Availability	18	-
	Total	129	

It can be seen that there were 129 records of security breaches for almost three years of usage. With the new accounting system that implemented SSDM principles, no incidence of security breach has been recorded in about one years9 of usage.

Future Work and Conclusions

In this paper, we examined the problem of producing secure software. The research in software security is gaining much attention because of the implication of security breaches. Problems towards producing secure software products were identified and stated in this work. This is important for researchers so as to know the real problems and the direction to follow in providing the methodologies for providing secure software product. We also proposed an integrated model for providing secure software products and presented a case study to ascertain that the model works. The model has some interesting features that guarantee the successful production of secure software products. SSDM has shown a way of separating security specification from functional specification. It is believed that if this model is carefully implemented; it will result in the production of secure software products. It was shown in this work that implementing SSDM improves the security of software systems.

The model presented in this work still needs to be more tested so as to really ascertain its performance. Some of the components of SSDM are still major areas of research that should be studied.

References

Davis, N. & Mullaney, J. (2003). The team software process in practice: A summary of recent results. *Technical Report CMU/SEI-TR-014*, September.

Devanbu, P. T. & Stubblebine (2000). Software engineering for security: A roadmap. *Proceedings of International Conference on Software Engineering (ICSE' 2000), Special volume on "Future of Software Engineering", 2000.*

Ghosh, A. K., Howell, C. & Whittacker, J. A. (2002). Building software securely from ground up. *IEEE Software magazine*, January/February.

Goldenson, D. R. & Gibson, D. L. (2003). Demonstrating the impact and benefits of CMMI. *Special Report CMU/SEI-2003-SR-009.* The Software Engineering Institute, Carnegie Mellon University.

Hall, A & Rod, C. (2004). Correctness-by-construction. Paper written for *Cyber Security Taskforce Subgroup on Software Process*, January.

Hefner, R. (1997). Lesson learned with the systems security engineering capability model. *Proceedings of the Conference on Software Engineering*, Boston, Massachusetts, USA.

Jones, C. (2000). *Software assessments, benchmarks, and best practices*. Reading, MA: Addison-Wesley.

Linger, R. & Stacy, P. (2004). Developing secure software with Cleanroom software engineering. Paper prepared for *Cyber Security Summit Task Force Subgroup on Software Process*, February.

McGraw, G. E. (2003). On the horizon: The DIMACS workshop on software security. *IEEE Security and Privacy*, March/April.

Mead, N. R. & McGraw, G. (2003). From the ground up: The DIMACS software security workshop. *IEEE Security and Privacy*, March/April.

Mogilevsky, D., Lee, A. J. & Yurcik, W. (2005). Defining a comprehensive threat model for high performance computational clusters. *ACM Computing Research Repository (CoRR)*, Technical Report cs.CR/0510046, October 16, 2005. Retrieved from http://www.ncassr.org/projects/threatmodeling/mogilevskyCorr05-threatmodel.pdf

Myagmar, S., Lee, A. J. & Yurcik, W. (2005). Threat modeling as a basis for security requirements. *Symposium on Requirements Engineering for Information Security (SREIS)* in conjunction with *13th IEEE International Requirements Engineering Conference (RE), Paris, France,* August 29.

Pauli, J. & Xu, D. (2004). Threat-driven architectural design on secure information systems. Presented at the *International Conference on Enterprise Information System, Miami, Florida, USA.*

Redwine, S. T. & Davis, N. (2004). Processes to produce secure software. *National Cyber Security Summit*, March.

Rodgers, D. H. (2002). Implementing a project security review process within the project management methodology. *GIAC Security Essentials Certification (GSEC), SANS Institute.*

Schneir, J. H., Lippman, J. R. & Wing, J. (2002). Automated generation and analysis of attack graphs. *Proceeding of IEEE Symposium on Security and Privacy, April.*

Sheyner, O. & Wing, J. (2004). Tools for generating and analysing attack graphs. *Proceeding of Formal Methods for Components and Objects.*

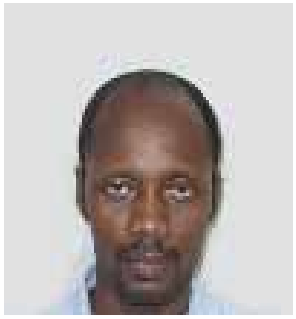
Sodiya, A. S., Longe, H. O. D. & Akinwale A. T. (2004). A new two-tiered strategy to intrusion detection. *Journal of Information Management and Computer Security*, *12*(1).

Wilander, J. & Gustavsson, J. (2005). Security requirements – A field study of current practice. Presented at the *Symposium on Requirement Engineering for Information Security (SREIS' 2005)*, Paris, France.

Bibliographies

Dr. **A. S. Sodiya** is presently a lecturer in the department of Computer Science, University of Agriculture, Abeokuta, Nigeria. He completed his PhD in the same university in 2004. His areas of research interest are Computer Security, Computer Network, Artificial Intelligence and Software Engineering. He has published both in local and international journals. He has also attended and presented papers in several conferences.

Mrs. **S. A. Onashoga** (formerly Mrs. Ibrahim, nee OKUNLAYA) is a Lecturer in the Department of Computer Science, University of Agriculture, Abeokuta, Nigeria. She graduated with B.Sc. degree in Mathematical Sciences (Computer Science option). She had a M.Sc. in Computer Science. She has publications in local and international journals. Her areas of research interest are Data Mining and Network Security. She is currently a Ph.D. student in Computer Science.



Mr. **O. B. Ajayi** is the e-Learning/Network Administrator of the Computing Centre of the University of Agriculture, Abeokuta. His primary responsibility is in the area of System Programming and Analysis. He graduated with B.Sc (Honours) degree in Mathematical Sciences (Computer Science Option). He later obtained his M.Sc in Computer Science from the same University and postgraduate diploma in Petroleum Engineering from the University of Ibadan, Ibadan. He is blessed with two kids and a lovely wife.