Journal of Information Assurance and Security 4 (2009) 161-169

Developing a Security Typed Java Servlet

Doaa Hassan[1], Sherif El- Kassas[2] and Ibrahim Ziedan[3]

[1]National Telecomm Institute, Computers and Systems Department
5 Mahmoud El Miligui Street, 6th district - Nasr City.Cairo - Egypt.
doaa@nti.sci.eg

[2]American University in Cairo, Computer Science Department
AUC Avenue, New Cairo 11835, Egypt
sherif@aucegypt.edu

[3]Zagazig University, Faculty of Engineering, Computers and Systems Department
Zagazig, Ash Sharqiyah governorate
iziedan@yahoo.com

Abstract: The Lack of security policy enforcement in web development languages is one of the most important challenges in web application systems development, as there is no formal check for security policy violation that may occur during web application system development. To check for policy compliance, the programmer must walk through all the code and check every line to make sure that there are no security violations. For example, a developer may develop a web application system connected to data base that seems to work properly, but it can make a certain security policy violation by permitting unauthorized users to access the data base system. This paper proposes a solution for the above problem by developing and application of a security typed Java servlet that can run on the web server side safely. This servlet is developed by embedding the Java code produced by compiling the Java information flow language (Jif) (a security-typed programming language that extends Java with support for information flow control and access control, both at compile time and at run time) into a servlet code format . The code produced by compiling Jif language is security typed and support servlet with means of flow control and access control. Hence we can guarantee that when we run this servlet into a web application system it will check input data trough the web application system for information flow security policy violation.

Keywords: Information flow control, Jif language, decentralized label model, threat model, web application system, Java servlet.

1. Introduction

The current Java-web based development languages such as Java Sever Pages (JSP) do not check for security policy violation, especially when connecting to database. For example an unsafe JSP script could use unchecked input strings to compose SQL queries and then have the DBMS execute query, which potentially allows the attacker to insert arbitrary commands in the SQL query (the so-called SQL-injection attack)[17]. To prevent this, we need to make sure that the queries performed by the system comply with the desired security policies on the data stored in the database. In current development environments, if we want to achieve the security policy goals of maintaining the confidentiality and integrity of data, we must check the web scripts for security policy violations at compile and runtime.

In this paper we propose extensions to the web scripting environments that enable both the static (compile-time) and dynamic (runtime) checking for information flow security policy violations. We try to extend the Java-servlet with new features that enable it to make static and runtime checking for security policy violations during compile and runtime phases respectively.

We achieve our goals by developing a security typed Java servlet that can run on the web server side and permit accessing to the system resources safely. This servlet is developed by embedding the Java code produced by compiling the Java information flow language (Jif) [2], [3] language code into a Java servlet code format. Since the Jif language is an extension to Java that keeps Java programming language features in addition to permitting static checking of information flow policies. Jif is an implementation of JFlow language described in [4]. In other words, our new Java servlet enforces the security features of Jif in a web application system framework when run inside the web servers.

In this way we are able to ensure the static checking for information flow security policy violations that can harmfully affect the confidentiality and integrity of data in a web application developed using the servlet technology, hence in Jif, the type of a variable may be annotated with a label specifying a set of principals who own the data and a set of principals that are permitted to read the data in addition to a set of principals that are permitted to modify the data.

Labels are checked by the compiler to ensure that the information flow policies are not violated. Since, Jif is an extension to Java and needs Java environment to run, in our new secure servlet, we can keep many features of normal Java servlet operation in addition to the new security

Received June 10, 2009

features added to our new secure servlet that are inherited from Jif.

The remainder of this paper is organized as follows; section 2 provides some background about the Information flow control in web application System, Decentralized label Model, Jif Language and Java servlet. Section 3 presents our methodology to develop our new secure Servlet and implementation. Section 4 presented the related work, and then we conclude in section 5 with directions to future work.

2. Background

2.1 Information flow control in web application System

Information flow control of program is very important for large scale distributed system such as web application system to provide more precise control of information propagation to the public, as it is not practical to manually checks all the code for complexity reasons to ensure that code does not leak confidential information.

It aims to track all the dependency between objects which may cause confidential information leakage. For example when the value of an object Y is affected by an object X we say that there is an information flow from X to Y. The problem of information flow is relevant if X stores sensitive information and Y is a public system output. In this case the control of how sensitive information propagates in the program is crucial for protecting confidentiality. Generally, program data can be associated with security levels which constitute a security lattice. The higher security level in the lattice is associated with the more sensitive information. To get a secure information flow, the data must flow up the lattice. If, for some reason, data must flow down the lattice, declassification must take place.

We identify the information flow in web applications as the information that flows over multiple requests, such as a request sent to a server by a user may contain information about user's previous request and response. The input data coming from the client web browser can be treated as having the security labels public and tainted, while the information on the server side have the security labels secret and untainted.

The information flow model in web application system [15] is presented by a simple lattice with two security level. The higher security in the lattice is associated with the sensitive information such as web resources, including databases or system files, while the low security level presents the public information that presents the input data coming from the client web browser. Figure (1) shows a simple lattice form for information flow in web application system.

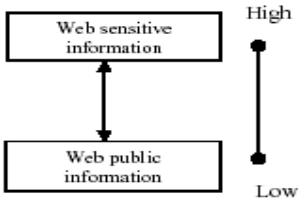

Figure 1. Example of web security lattice

2.2 Decentralized label Model

The Decentralized label model (DLM) [14] is a language based approach of information flow control. DLM is implemented in Jif Language to allow users to declassify their information in controlled way without traditional impractical security constraints. Its main elements are: principals- labels- constraints and declassification.

Principals in DLM are users or groups or roles which may own, update and release or declassify information. Labels are used to guarantee confidentiality. Every label consists of a set of policies that express privacy requirements. A privacy policy has two parts: an owner and set of readers, which is written in the form owner: readers. By definition, an owner is implicitly contained in reader set. A principal is allowed to read data if and only if it is contained in the reader set of all policies of the label attached to the data. Constraints require the authority of a principal to perform actions that may compromise the security of a principal such as declassification of sensitive information or permitting a certain principal to act for another one. This is can be used to build more complex access control mechanisms. Declassification is called downgrading [16] and it is needed to weaken the security policies of application when it is necessary.

For example, a login program must declassify some information about the confidential password when it accepts or rejects the user's login attempt.

2.3 Jif Language

Jif Language was developed by Andrew Myers and hid research team as an extension to the Java language that that keeps many features of it in addition to control the propagation of confidential data by permitting static checking of information flow policies and implementing the decentralized label model (DLM) [14] by adding *labels* that express restrictions on how information may be used. Jif is based on the JFlow language described in [4].Its primary goal is to prevent confidential and/or untrusted information from being used improperly.

In Jif, the security policy for confidentiality or integrity is incorporated into the code by adding labels to variables and objects. Labels specify the ownership, read and write permissions. The following example shows the declaration of a variable x of type integer that owned by Alice, and both Alice and Bob can read the data.

```
Int {Alice:Bob} x;
```

Labels are used to guarantee confidentiality and integrity. A

label is a pair of a confidentiality policy and an integrity policy. We write a label $\{c;d\}$, where c is a confidentiality policy, and d is an integrity policy. A confidentiality policy has two parts: an owner and set of readers, which is written in the form owner: reader or $o \rightarrow r$ allows the owner to specify which principals can read information. By definition, an owner is implicitly contained in reader set. A confidentiality policy $o \rightarrow r$ says that o permits a principal q to read information only if q can act for either the owner of the policy or the specified reader. An integrity policy has two parts: an owner and a set of writers, which is written in the form owner!: writer *or* $o \leftarrow w$ allows the owner to specify which principals may have influenced ("written") the value of a given piece of information. The policy $o \leftarrow w$ means that according to the owner o, a principal q could have influenced the value of the information only if q can act for the owner o or the specified writer w.

In Jif, the implicit information flow problem is controlled by associating a static program counter label (pc) with every statement and expression, representing the information that might be learned from the knowledge that the statement or expression was evaluated.

Principals in the Jif language, the entities that have security requirements, are explicitly represented, both as type annotations and as values that can be manipulated at run time. Jif provides the ability to create user-defined principals, so that programs can define their own authentication and authorization procedures. The Jif interface (jif.lang.Principal) is used to represent principals, and Jif programs may implement this interface to define their own principals.

Jif allows classes and interfaces to be parameterized by explicitly declared label and principal parameters. The Jif compiler automatically infers label and principal parameters for local variables. This makes programming in Jif more convenient, because it is not necessary to explicitly instantiate parameterized classes in most places where these types can be mentioned.

Jif provides mechanisms to downgrade the confidentiality and the integrity of information, via explicit program annotations to weaken the security policies of information as part of their intended functionality. For example, a login program must declassify some information about the (confidential) password when it accepts or rejects the user's login attempt. The expression declassify(e, L_1 to L_2) relabels the result of an expression e from the initial label L_1 to the label L_2, where the integrity specified by L_2 must be at least as restrictive as the integrity specified by L_1. That is, a declassify expression can weaken only the confidentiality, not the integrity, of information. Similarly, the expression endorse(e, L_1 to L_2) relabels the result of an expression e from L_1 to L_2, where the confidentiality specified by L_2 must be at least as restrictive as the confidentiality specified by L_1.

During compilation, the Jif compiler translates Jif language to Java language that can be executed with standard Java virtual machine after the Jif type checker guarantee that labels specified by the programmer satisfy the required security policy. If there are violations in the security policy declared in the labels, the Jif compiler reports them and reject the program as unsecure. Although information flow security policy enforcement is mostly done by Jif at compile time, Jif does also allow some enforcement to take place at run time. Therefore, Jif programs in general require the Jif run-time library. For more details about Jif language, refer to [2],[3],[4].

2.4 Java Servlet overview

A servlet is a server side platform independent Java program that can be used to handle data between client and web server based on interactively viewing or modifying that data using dynamic web page generation techniques. Since servlets run inside server side, they do not need graphical user interface.

A client program which could be a browser or some other program that can make connections across the internet, access the web server and makes a request. Servlet can respond to the client requests by dynamically constructing a response that is sent back to the client. Each client request is represented by a servlet request object (of type ServletRequest). The response sends to the client is represented by a servlet response object (of type servletResponse). Figure (2) shows the basic servlet operation.

A single servlet can be invoked multiple times to serve a request from multiple clients. A servlet can handle multiple requests concurrently and can synchronize requests. Also it can forward requests to other servers and servlets. To invoke servlet, a URL command pointing to the location of the servlet must be issued as the servlets are located in any directory or machine where a web server is running.

Because servlets written in Java, they are ideal for implementing complex business application logic, that allows clients to access relational databases through dynamic web page. In addition they are portable and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major web server. Also adding servlet support to a web server, no matter the cost of it is generally free or cheap. For more details about Java servlet refer to [1], [12].

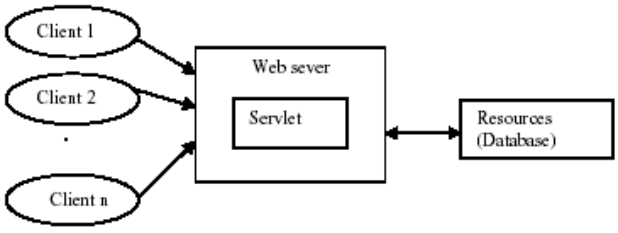

Figure 2. Basic Servlet operation.

3. Proposed solution and Implementation

In the presented work we develop our servlet by joining the work between the Jif environment and Java environment in two implementation phases. In the Jif environment we need the security type checking of Jif to produce a security typed Java code, as the Jif program is compiled and produces a Java code that have the same behavior of Jif code according to type safety . The Java code produced by Jif compiler in Jif environment is embedded into a Java servlet format in a Java environment. The result will be a normal Java servlet that can be run on any Java servelt container or in other words Java application server, but with some security constrains inherited from Jif language. This security features can keep the developer time from walking through all the code and check every line to make sure that there are no security violations. Our new secure Java servlet development life cycle is illustrated into figure (3).

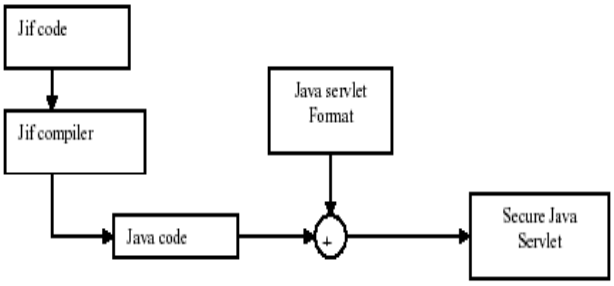

Figure 3. Secure Java Servlet Development life cycle.

3.1 Jif Implementation

For implementation we, consider a case of study that requires some security constrains. This is the case when a normal user needs to open and read a system file. The user will access the file through a dynamic web application system that is built using our new secure Java servlet.

Since Jif language interacts with file system [18], so it will take into consideration the permission given by the operating system to read, write and execute the file by any user that want to access the file if the application policy is less restrictive than the operating system, but if the application policy is more restrictive than the operating system, then the access control policy specified in the application (.i.e. Jif program) will be applied to the file. In our case we write a Jif program that enforce a file access permission policy that prevent the normal user from reading the file and give the permission to file owner only (.i.e. the user who creates the file) . So the result of execution will prevent any user except the owner from reading the file as he has more restrictive access control permissions than the user has. This access permission is applied to the file if it is more restrictive than the access permission given to the file from the operating system. Figure (4):a, b respectively show a Jif program (cat.Jif) that permit the file owner only to open and read the file and a part of the Java code produced by compiling it (cat.java) that have the same security policy of cat.Jif to access the file.

3.2 Threat model

The threat model [13] addressed by our application environment is the attempt of current user of the system specified in (cat.Jif) by Runtime class, which instantiated by getRuntime() and parameterized only by the user who executed this program (.i.e. the current user of the operating system) to access the file owned by certain principal specified by the access permission of the file from the operating system, although there is mismatching between the current user of the operating system who executes the program and the file owner specified by the operating system. Also the current user of the operating system does not act for file owner. The model here covers the operation of reading the file, not editing or writing in the file (.i.e. the confidential information release, not the integrity of this information).

3.3 Trust model

In the trust model [13], presented in our work, the current user of the system can access the file owned by certain principal, as this principal trusts the current user of the system to perform this operation (.i.e. reading the file) , if and only if the access permission of the file from the operating system states that owner of the fileis matching the current user of the system who executes the Jif program specified by getruntime() in (cat.Jif).

3.4 Java Implementation

The Java code introduced above is embedded into a Java servlet format and produces our new secure servlet (catsrvlet.Java) that have enforce the same access control policy to the file as (cat.Jif) as shown in figure (5).

This servlet can be used by many clients to access the files stored on web server depends on the access permission specified in our new secure servlet that is the same like the access permission to the file that is specified in cat.jif.

We used the Apache Tomcat server [11] as a Java Servlet container on which our new secure Java servlet runs.

4. Related work

The idea of using programming-language techniques for enforcing information flow policies was discussed for some time in [5]. The focus of this research was on work that uses static program analysis to enforce information flow policies. It assumes that computation using confidential information is possible, and that it is important to prevent the results of computation from leaking even partial information about confidential inputs. This approach addressed the practical security mechanisms in language-based techniques for soundly enforcing end-to end confidentiality policies continues.

The type-checking approach has been implemented in security typed language such as Jif [2], which extends Java, and FlowCaml [10], which extends Caml. Both of these

languages provide type systems that enforce information flow policies. In the type-checking approach, every program expression has a security type with two parts: an ordinary type such as int, and a label that describes how the value may be used. Security is enforced by type checking, the compiler reads a program containing labeled types and in type checking, the program ensures that it cannot contain improper information flows at run time.

However, to date, a security-typed language such as Jif has only been used to build simple "toy" programs. There

```
1  import jif.runtime.Runtime;
2  import jif.lang.*;
3  import java.io.*;
4
5  public class cat {
6      public static void main{p:}(principal{} p, String{}[]{} args):{p:}
7      throws (SecurityException) {
8          final label{} lb = new label{p:};
9
10         Runtime[p] runtime;
11         try {
12             runtime = Runtime[p].getRuntime();
13         }
14         catch (SecurityException e) {
15             runtime = null;
16         }
17
18         if (runtime == null) return;
19
20         PrintStream[{p:;_!:_}]{p:} out = runtime.out();
21         PrintStream[{p:;_!:_}]{p:} err = runtime.err();
22         if (err == null || out == null) { return; }
23
24         try {
25             String{} filename = args[0];
26             FileInputStream[{*lb}]{*lb} fis = runtime.openFileRead(filename, lb);
27
28             InputStreamReader[{*lb}]{*lb} reader = new InputStreamReader[{*lb}](fis);
29             BufferedReader[{*lb}]{*lb} br = new BufferedReader[{*lb}](reader);
30             String{*lb} line = br.readLine();
31             while (line != null) {
32                 out.println(line);
33                 line = br.readLine();
34             }
35         }
36         catch (ArrayIndexOutOfBoundsException e) {
37             err.println("usage: cat file");
38         }
39         catch (IOException e) {
40             err.println(e.getMessage());
41         }
42         catch (NullPointerException e) {
43             err.println(e.getMessage());
44         }
45     }
46 }
```

Figure 4 :(a) a Jif program (cat.Jif) that permit the file owner only to open and read the file.

```
4
5  public class cat {
6
7      public static void main(final String[] args) throws SecurityException {
8          final Principal p = Runtime.user(null);
9          {
10             final Label lb =
11               LabelUtil.toLabel(
12                 LabelUtil.readerPolicy(p, PrincipalUtil.topPrincipal()),
13                 LabelUtil.writerPolicy(PrincipalUtil.bottomPrincipal(),
14                                        PrincipalUtil.bottomPrincipal()));
15             Runtime runtime;
16             try {
17                 runtime = Runtime.getRuntime(p);
18             }
19             catch (final SecurityException e) { runtime = null; }
20             if (runtime == null) return;
21             PrintStream out = runtime.out();
22             PrintStream err = runtime.err();
23             if (err == null || out == null) { return; }
24             try {
25                 String filename = args[0];
26                 FileInputStream fis = runtime.openFileRead(filename, lb);
27                 InputStreamReader reader = new InputStreamReader(fis);
28                 BufferedReader br = new BufferedReader(reader);
29                 String line = br.readLine();
30                 while (line != null) {
31                     out.println(line);
32                     line = br.readLine();
33                 }
34             }
35             catch (final ArrayIndexOutOfBoundsException e) {
36                 err.println("usage: cat file");
37             }
38             catch (final IOException e) { err.println(e.getMessage()); }
39             catch (final NullPointerException e) {
40                 err.println(e.getMessage());
41             }
42         }
43     }
44
45     public cat cat$() {
46         this.jif$init();
47         {  }
48         return this;
49     }
```

Figure 4 :(b) A part of Java code (cat.java) produced by compiling cat.Jif.

```
1  import jif.runtime.Runtime;
2  import jif.lang.*;
3  import java.io.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6
7  public class catservlet extends HttpServlet {
8      public void doGet(HttpServletRequest req, HttpServletResponse res)
9                                      throws ServletException, IOException {
10    res.setContentType("text/plain");
11    PrintWriter out = res.getWriter();
12
13
14
15        final Principal p = Runtime.user(null);
16        {
17          final Label lb =
18            LabelUtil.toLabel(
19              LabelUtil.readerPolicy(p, PrincipalUtil.topPrincipal()),
20              LabelUtil.writerPolicy(PrincipalUtil.bottomPrincipal(),
21                                     PrincipalUtil.bottomPrincipal()));
22            Runtime runtime;
23
24             try {
25                runtime = Runtime.getRuntime(p);
26             }
27             catch (final SecurityException e) { runtime = null; }
28             if (runtime == null) return;
29             PrintStream out1 = runtime.out();
30             PrintStream err = runtime.err();
31             if (err == null || out1 == null) { return; }
32
33            out.println("from jif");
34
35            try {
36
37
38                    String name = req.getParameter("name");
```

Figure 5. A part of code of (catservlet.java), a security-typed Java servlet developed as a front-end to cat.Jif.

are currently only few applications written in Jif [6- 9]. However the most closely related work is SIF [19]. It was a software framework for building high-assurance web applications; it used language-based information-flow control to enforce security. It was written in Jif 3.0 [2], an extended version of the Jif programming language, to track the flow of information within a web application, and information sent to and returned from the client. SIF is an extension to the Java Servlet framework, and requires the Jif 3.1 compiler to compile a web application in the SIF framework. It achieves the same goals like ours, but it uses different methodology for implementation.

Stephen Chong, Jed Liu Andrew C. Myers, Xin Qi ,K. Vikram, Lantian Zheng and Xin Zheng introduced Swift[20] principled approach to write web applications that are secure by construction. Application code is written as Java-like code annotated with information flow policies that specify explicitly the confidentiality and integrity of web application information. The compiler uses these security annotations to decide where code and data in the system can be placed securely. It automatically partitions the program into JavaScript code running in the browser, and Java code running on the server. This technique permits a web application to be split according to a policy into JavaScript code that runs on the client and Java code on the server.

The presented research propose a solution for the problem of how to enforce the information flow security policy during the web application development phase and introduces the development of a new security typed Java servlet based on the Java language variant Jif. This servlet is developed by embedding the Java code produced by compiling the Jif [2], [3] language code into a Java servlet code format. The presented servlet can run on the web server side and prevents the security policy violation caused by unauthorized access to system resources through dynamic web application system that is built based on servlet technology.

5. Conclusion and future work

In this paper we introduce the developing and application of a security typed Java servlet that can run on the web server side and enforce the access control policy for file system based on access permission specified in the policy contained

in the Jif label. This servlet is developed by embedding the Java code produced by compiling the Java information flow language (Jif) into a servlet code format. The code produced by compiling Jif language is security typed and support servlet with means of flow control and access control. Hence we can guarantee that when we develop a dynamic web application system using this servlet, it will check the input data through it for security policy violation and prevent unauthorized access to system resources.

There are several interesting directions for our future work. First we need to study the application of the introduced secure Java servlet to access data base system through dynamic web application system.

Second, the process of picking up the Java code produced from compiling the Jif code and embedding it into servlet is done manually and we are looking forward to do it automatically. Finally, this introduced secure Java servlet can be considered as an important stage in the development of a secure dynamic web scripting language that acts as a front-end to the information flow language (Jif).

Acknowledgments

We should thank Steve Chong at Cornell University for his endless patience and prompt responses to our questions about settings of Jif runtime environment and features of Jif language. Also we should thank Mr. Ayman Abdo, a senior web developer in Royal Scientific Society of Jordan for helping us in deploying the java code produced by compiling the Java information flow language (Jif) in java servlet content and also Mr. Jack Fayez, a web developer in National Telecommunication Institute for helping us in running the proposed web application.

References

[1] Jeanne Murray."Building Java HTTP servlets". *IBM developerWorks,12 September 2000. Available via:* http://www.digilife.be/quickreferences/PT/Building%20Java%20HTTP%20servlets.pdf

[2] Stephen Chong, Andrew C. Myers, K. Vikram, and Lantian Zheng. "Jif Reference Manual". *June 2006. Available via:* http://www.cs.cornell.edu/jif/doc/jif-3.1.1/manual.html

[3] Andrew C. Myers. *"Mostly-Static Decentralized Information Flow Control"*. Ph.D. Thesis, January 1999.

[4] Andrew C.Myers. "JFlow: Practical Mostly-Static Information Flow Control". *Proceedings of the 26th ACM Symposium on Principales of Programming Languages (POPL'99)*,San Antonio, Texas, USA, January 1999.

[5] A. Sabelfeld and A. C. Myers." Language-based information- flow security". *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, Jan. 2003.

[6] R. Heldal and F. Hultin. "UMLS Bridging Model-based and Language-based Security". In E. Snekkenes and D. Gollmann, editors, *Computer Security-ESORICS2003, volume 2808 of LNCS*, pages 235-252. Springer, 2003.

[7] Boniface Hicks,Kiyan Ahmadizadeh and Patrick McDaniel. "From Languages to Systems: Understanding Practical Application Development in Security-Typed Languages". *Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference,* p.153-164, December 11-15, 2006.

[8] Aslan Askarov and Andrei Sabelfeld. "Security-typed languages for implementation of cryptographic protocols: A case study". In *Proc.of ESORICS 2005*, Milan, Italy,Sept.12- 14,2005.LNCS.©Springer-Verlag 2005.

[9] Boniface Hicks, Sandra Rueda, Trent Jaeger and Patrick McDaniel. "Integrating SELinux with Security-typed Languages". NSRC Technical Report NAS-TR-0052-2006.

[10]Vincent Simonet. "Flow Caml in a nutshell". In Graham Hutton, editor, *Proceedings of the first APPSEM-II workshop*, pages 152-165, Nottingham, United Kingdom, March 2003.

[11] Martin Bond and Debbie Law. *"Tomcat Kick Start"*. June 2003.

[12]James Duncan Davidson with Suzanne Ahmed. ***"Java™Servlet API Specification: Version2.1a".*** Sun Microsystems, Inc. © 1998.

[13] Peng Li and Yun Mao. "Integrity Extension in Jif". *CIS-670 Course Project Report on Advanced Topics in Programming Languages: Safety and security,* Spring 2003.

[14] Andrew C.Myers and Barbara Liskov. "Protecting Privacy using the Decentralized Label Model". *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442, October 2000.

[15] Peng Li and Steve Zdancewic. " Practical Information flow Control in Web-based Information Systems". In *Proceedings of the 18th IEEE Computer Security Foundation Workshop (CSFW)*, June 2005.

[16] P. Li and S. Zdancewic. "Downgrading policies and relaxed noninterference". *In Proc. 32th ACM Symp. On Principles of Programming Languages (POPL)*, 2005.

[17] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, D. T. Lee, Sy-Yen Kuo. "Securing Web applications code by static analysis and runtime protection". *In Proceedings of WWW 2004,* Manhattan, New York, USA., May 17–22, 2004.

[18] Boniface Hicks, Sandra Rueda, Trent Jaeger, and Patrick McDaniel. "From trusted to secure: Building and executing applications that enforce system security". In *Proceedings of the USENIX Annual Technical Conference*, Santa Clara, CA, USA, June 2007.

[19] Stephen Chong, K. Vikram, Andrew C. Myers. "SIF: Enforcing Confidentiality and Integrity in Web Applications". *Proceedings of USENIX Security Symposium 2007*, pages 1–16, August 2007.

[20] Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, Xin Zheng. "Secure Web pplications via Automatic Partitioning". *Proceedings of the 1st ACM Symposium on Operating Systems Principles* SOSP'07), pages 31–44, October 2007.

Author Biographies

Doaa Hassan She was born in Sharqiyah governorate in Egypt. She earned her B.S in telecommunication and electronics from the faculty of engineering in Zagazig University in Egypt. After that she earned her master degree in the same field from the same university in September 2002. Currently she works as assistant lecturer at the computers and systems department in National and Telecommunication Institute in Cairo. Her research interest includes Security policy, Information-flow policies, language-based security, network security and Cryptography.

Sherif El Kassas He received his Ph.D. in computer science from the Eindhoven University of technology in the Netherlands in 1994. He is currently a faculty member at the department of computer Science, AUC and associate Director of Academic Computing Services.

El-Kassas' research interests are focused on Security Management, the application of formal methods in Software engineering and Computer Security, and open source technologies.
El-Kassas is also a consultant for various organizations; Member of the board of trustees of the Information Technology Institute; Member of the board of the Egyptian Open Source Business Consortium NGO; and Member of various professional computing societies.

Ibrahim Ziedan He occupied the position of dean of the Faculty of Engineering in Zagazig University in 1997. In addition he was serving as the head of the deapartement of computers and systems departmentin in the same faculty for several years since 1997. He is currently a professor at the department of computer and systems, Faculty of Engineering Zagazig University and he is the director of Cisco Academy in Zagazig university .
Ziedan' research interests are focused on Computer organizations,Computer network and programming languages.