Enforcing Secure Data Sharing in Web Application Development Frameworks like Django Through Information Flow Control

S.Susheel[1], N.V.Narendra Kumar[2], and R.K.Shyamasundar[3]

[1] P.E.S Institute of Technology, Bengaluru, India,
susheel.suresh@gmail.com
[2] Tata Institute of Fundamental Research, Mumbai, India,
naren.nelabhotla@gmail.com
[3] Indian Institute of Technology, Mumbai, India
rkss@cse.iitb.ac.in

Abstract. The primary aim of web application development frameworks like Django is to provide a platform for developers to realize applications from concepts to launch as quickly as possible. While Django framework provides hooks that enable the developer to avoid the common security mistakes, there is no systematic way to assure compliance of a security policy while developing an application from various components. In this paper, we show the security flaws that arise by considering different versions of an application package and then show how, these mistakes that arise due to incorrect flow of information can be overcome using the Readers-Writers Flow Model that has the ability to manage the release and subsequent propagation of information.

1 Introduction

Web applications are increasingly becoming the primary curators of personal and corporate data. Social media applications like Facebook [2], LinkedIn [3] and Twitter [4] have transformed how users communicate with each other, while online document suites like Google Docs [5] or Office Online [6] have made online collaboration the norm. Much of the success of such Web applications is due to the flexibility in allowing third-party vendors to extend user experiences.

Most of todays web applications do not give users end-to-end security on their data. Lampson [7] describes most of the real word problems regarding the gold standard: authentication, authorization and auditing. This means that if a user A decides to share a piece of important/private information with another user B in the system, there is no way of tracking an information leak if, user B decides to make it public or share it with C, who could benefit from it and in turn harm or invade A's privacy.

In the age of a social web and semantic web [8], data belonging to one user might be accessible to an application installed by another user, introducing even more complexity. Due to the complex interactions that are possible in collaborative systems such as social networks, it is important to not only control the release of information (achieved through access controls), but also to control the flow of information among the various

stakeholders involved. This is achieved through information flow control (IFC). It is a well established fact that the traditional discretionary access controls are prone to attacks by Trojan horses that can cause indirect usage of data beyond the observation of access controls. IFC which is a form of mandatory access control prevents these types of indirect data misuse.

In this paper, we first experimentally demonstrate that current access controls in web applications are prone to Trojan horses. Then, we demonstrate how addition of simple checks using the RWFM model on the application permits us to enforce information flow control that resist Trojan horses. Fewer than 120 lines of code was needed to achieve this, and more importantly this is all the code needed to protect any application, thus, leading to an extremely scalable solution.

A summary of the contributions of this paper follows:

- Established that Trojan horse type attacks are very much possible in current security architectures
- Provided an effective way in which practical concepts of authorization are mapped onto the label model
- Illustrated the simplicity and efficiency of RWFM for securing data sharing in web applications; even the interference between two sessions of the same user are prevented
- Simplified policy compliance and audit trials

The rest of the paper is organized as follows: Section 2 provides an overview of current approaches to secure data sharing in web applications. Section 3 presents a quick introduction to our RWFM model. Experimental demonstration of the possibility of Trojan horse like attacks in an open source Django based web application, and its mitigation using RWFM are discussed in Section 4. Section 6 provides concluding remarks.

2 Overview of Currently used Access Controls in Web Applications

Historically, access control [9–11] has been the main means of preventing information from being disseminated. One of ways in which web development teams handle security policies is by decoupling their business logic from their authorization logic by externalizing authorization [12]. Externalizing authorization is possible using many access control paradigms like, XACML (eXtensible Access Control Markup Language) [13], ACL (Access Control Lists), and RBAC [10].

However, for simplicity web developers often use ad-hoc security to enforce and implement complex policies, that leads to a lot of problems like:

- the specified policy is intertwined throughout code which is error prone and is not scalable,
- makes the policy of an application difficult to “prove” for audit or quality assurance purpose, and
- causes new code to be pushed each time an access control policy needs to be changed.

The rapid progress of Web 2.0 [14] technologies have brought in a whole new category of web services which are content rich and allow users to interact and share information with each other easily. In addition, web applications publish APIs that enable software developers to easily integrate existing data and functions to create mashups which provide users enriched experience instead of building them by themselves. A lot of research is being conducted for securing web applications from third party extensions and mashups [15–18].

This calls for a model which is able to track and regulate information flow in a web application. An IFC model with a consistent labelling framework unifying both mandatory and discretionary access control is required. In the web setting, it must also enable formal assessment of conformance to security and privacy policies. Further, as advocated by Woo and Lam [30], and Abadi [29], the label must capture the entire essence of the transaction history in a succinct manner to cater to the practical performance.

3 Readers-Writers Flow Model

The first mathematical model for tracking information flow was given by Denning [26]. The lattice model was the first step in the right direction to understand the dynamic nature of information flows. A lot of work has been done in this field and a large body of literature has sprung up [19–22], yet information flow based enforcement/implementation mechanisms have not been used widely in the industry. Steve Zdancewic [23] presents the challenges for practical implementation of IFC.

In this section, we introduce the Readers-Writers Flow Model (`RWFM`) [27], [28] which is a novel model for information flow control. `RWFM` is obtained by recasting the Denning's label model [26], and has a label structure that: (i) explicitly captures the readers and writers of information, (ii) makes the semantics of labels explicit, and (iii) immediately provides an intuition for its position in the lattice flow policy.

Definition 1 (Readers-Writers Flow Model (`RWFM`)) `RWFM` *is defined as a five-tuple* $(S, O, S \times 2^S \times 2^S, (*, \supseteq, \subseteq), (*, \cap, \cup))$*, where* S *and* O *denote the set of subjects and objects in the information system respectively,* $S \times 2^S \times 2^S$ *is the set of security labels,* $(*, \supseteq, \subseteq)$ *denotes the can-flow-to ordering amongst labels, and* $(*, \cap, \cup)$ *denotes the label combining operator.*

The first component of a `RWFM` label denotes the ownership of information, second component denotes the permissible readers of information, and third component denotes the set of principals that influenced the information. For example, the `RWFM` label of *Alice*'s private key would be $(S, \{A\}, \{A, S\})$, indicating that S created it, A is the only permissible reader of it, and both A and S have influenced it, where A denotes *Alice* and S denotes key server/certificate authority.

Theorem 1 (Completeness) `RWFM` *is a complete model, w.r.to Denning's lattice model, for studying information flows in an information system.*

`RWFM` follows a floating-label approach for subjects, with $(s, S, \{s\})$ and $(s, \{s\}, S)$ as the "**default label**" and "**clearance**" for a subject s respectively, where S is the set of all

the subjects in the system. RWFM follows a static labelling approach for objects, with the exception of downgrading, which is allowed to change the label of an object by adding readers.

RWFM provides a semantics of secure information flow through *Information Flow Diagram* (IFD), which presents significant advantages and preserves useful invariants. IFD is a state transition system defined as follows:

Definition 2 (State of information system) *State of an information system is defined by the triple* (S, O, λ) *denoting the set of current subjects and objects in the system together with their current labels.*

State transitions of an information system are defined by RWFM considering the need for supporting the following operations: (i) subject reads an object, (ii) subject writes an object, (iii) subject downgrades an object, and (iv) subject creates a new object.

RWFM describes the conditions under which an operation is safe as follows:

READ Rule *Subject s with label* (s_1, R_1, W_1) *requests read access to an object o with label* (s_2, R_2, W_2).
If $(s \in R_2)$ then
    change the label of s to $(s_1, R_1 \cap R_2, W_1 \cup W_2)$
    ALLOW
Else
    DENY

WRITE Rule *Subject s with label* (s_1, R_1, W_1) *requests write access to an object o with label* (s_2, R_2, W_2).
If $(s \in W_2 \wedge R_1 \supseteq R_2 \wedge W_1 \subseteq W_2)$ then
    ALLOW
Else
    DENY

DOWNGRADE Rule *Subject s with label* (s_1, R_1, W_1) *requests to downgrade an object o from its current label* (s_2, R_2, W_2) *to* (s_3, R_3, W_3).
If $(s \in R_2 \wedge s_1 = s_2 = s_3 \wedge R_1 = R_2 \wedge W_1 = W_2 = W_3 \wedge R_2 \subseteq R_3 \wedge (W_1 = \{s_1\} \vee (R_3 - R_2 \subseteq W_2)))$ then
    ALLOW
Else
    DENY

CREATE Rule *Subject s labelled* (s_1, R_1, W_1) *requests to create an object o.*
Create a new object o, label it as (s_1, R_1, W_1) and add it to the set of objects O.

Given an initial set of objects on a lattice, IFD accurately computes the labels for the newly created information at various stages of the transaction / workflow.

4 Securing Information Flow among Users in Web Applications with RWFM

In this section, we illustrate simultaneously the simplicity and the power of RWFM for controlling and tracking information flow among users in web applications. Simplicity of RWFM enables an intuitive mapping of actions in the application to the operations of read and write in the abstract model.

For demonstrating the information flow control imposed by RWFM we chose an open source web application called DBpatterns [24][4] written in python using Django, a high-level Python Web Framework. We choose Django because it encourages rapid development, has a clean pragmatic design and is very popular among web developers. DBpatterns is an application that allows users to create, share and explore database schemas on the web. Further, this application supports features like collaboration with remote users, sharing information with fellow users, and ability to make information public or private, and also embodies a RESTful design by giving API endpoints for accessing data and certain functions. This fits well with the problem discussed before. By analysing the flow of information in this system and applying the RWFM, we show that a series of attacks that exists in the current architectures are resolved and propose that this model is easy to use and can be integrated with any web application.

4.1 Overview of DBpatterns

DBpatterns is web application which helps users in creating, sharing and exploring database schemas or models. The main features of the application are:

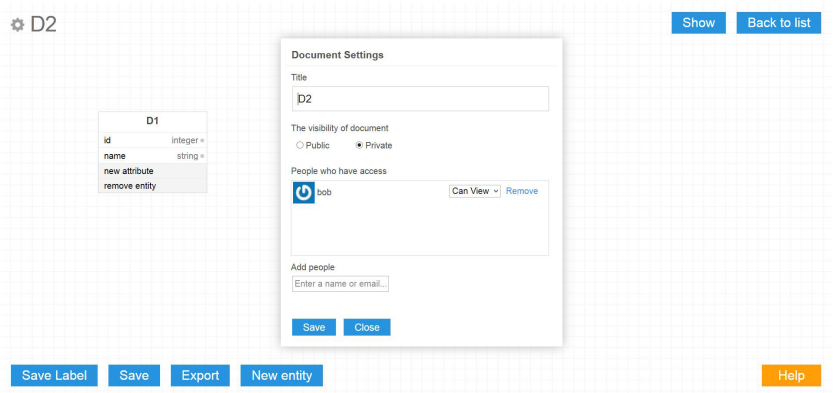
- User account creation and user authentication.
- User can explore database schemas which are public using public feed.
- User has the ability to fork public viewable schemas created by other members.
- User can create schemas and can set visibility to either public or private.
- User can add collaborators for a schema, and can grant view, edit permission or both.
- Users receive notification if they have been assigned any access in private or public schema documents.

Access Control Logic in DBpatterns The access control policy is written using ad-hoc security policies. The main access control related definitions are stored in the database with the schema document itself. It consists of,

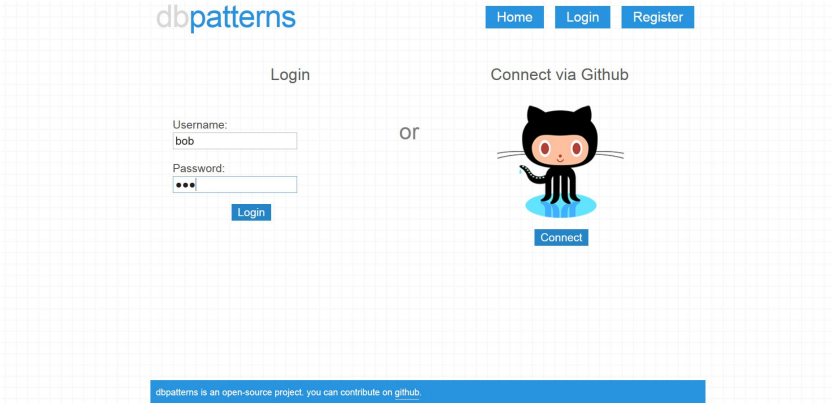
- *i*sPublic is a boolean value which specifies the visibility of the document.
- Assignees is an array which holds the assigned users for the document with their access permission like *c*anView or *c*anEdit.

Using these two attributes, access control checks are implemented throughout the application. This kind of implementation common in many web applications.

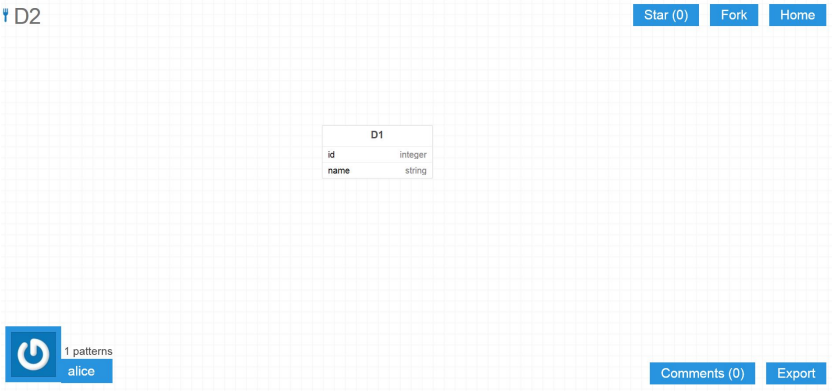
[4] DBpatterns is a service that allows you to create, share, and explore database models on the web. Uses Django, Tastypie, Backbone and MongoDB.



(A)



(B)



(C)

Fig. 1. (A) Alice grants Bob read access to document D_2. (B) Bob logs into the application, and (C) views schema document D_2.

We have studied the access control logic implementation and experimented on possible attacks that might occur targeting the application's lack of controlling information flow. Section 4.2 gives a detailed account for the possible attack scenarios.

4.2 Attack Scenarios in DBpatterns

Here we give a brief summary of the kind of attacks that were carried out on the DBpatterns web application.

Let the users of the system be Jack, Alice and Bob.

Attack 1 (Read - Attack) Jack shares a document, say D_1, with Alice. Alice forks D_1 and makes a local copy, call it D_2. Alice now owns D_2 and can set access restrictions on it. Alice gives read permission on D_2 to Bob. This way Jack will be unaware of an information leak and Bob who did not have authorization to view D_1 can now read D_2 which essentially has the same information as D_1. These kind of attacks that bypass access control logic are called Trojan horses. The screenshots of attack 1 are depicted in Figure 1.

Note that the attacks described in this section are possible on any discretionary access control mechanism, and are not specific to this web application.

4.3 Mitigating the Attacks using RWFM

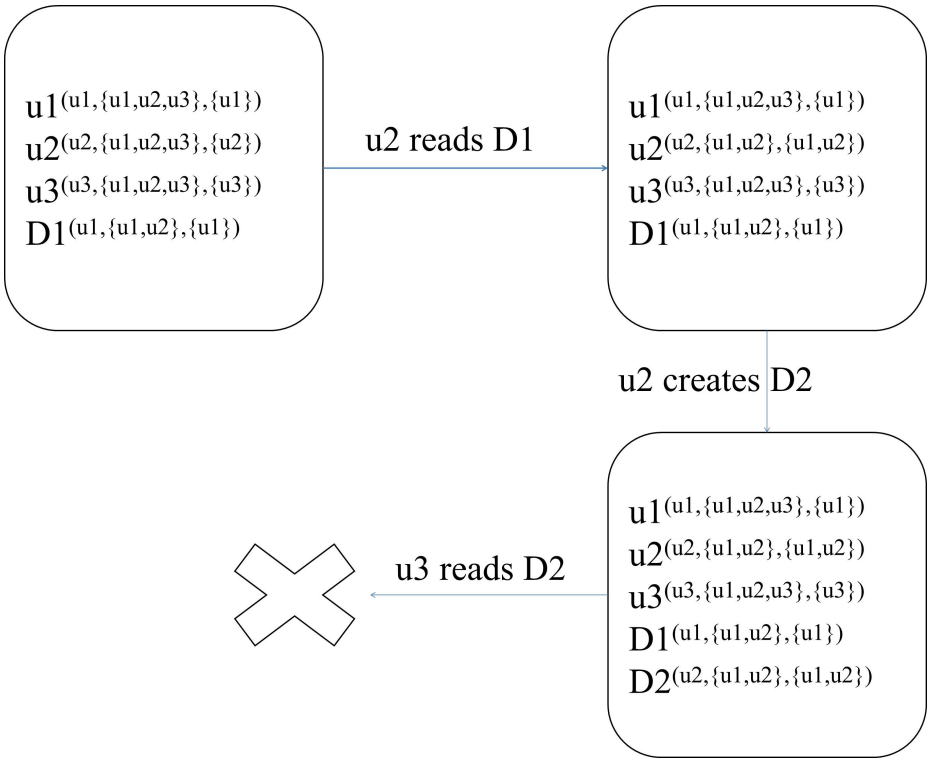
The flaws discussed in 4.2 exist in most web applications. Using an IFC based approach, we have successfully implemented RWFM in conjunction with the existing access control logic of DBpatterns and our implementation of RWFM will help web developers mitigate such attacks and also give them a clear picture of both subject and object labels at any given point of time.

RWFM automatically labels subjects and objects as the information flows in the system, given the initial state of DBpatterns application. Using the read, write, and create rules of RWFM, our implementation efficiently tracks information flow for all possible user interactions with the application and successfully resists the attacks presented in Section 4.2.

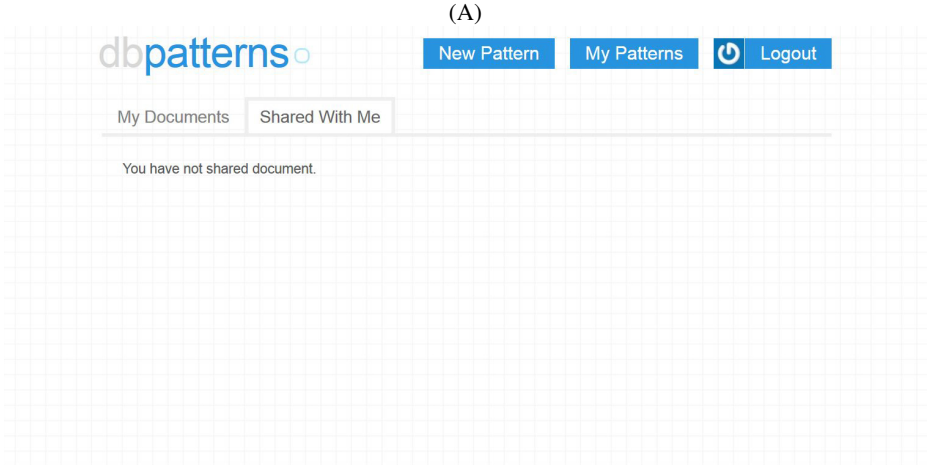
According to OWASP Top 10 Security Vulnerabilities of 2013 [25], Broken Authentication and Session Management was ranked second, our implementation takes into account, user sessions and protects the application from IFC related attacks through session fixation and manages multiple sessions from the same user. In our implementation on DBpatterns, user sessions are treated as subjects, and documents are treated as objects.

Notation: u_1, u_2 and u_3 denote Jack, Alice, and Bob respectively. $S = \{u_1, u_2, u_3\}$ denotes the set of all the users of the system. Entity (subject or object) e with label l is denoted e^l.

Information flow diagram for the scenario of attack 1 and its mitigation by RWFM is depicted in Figure 2. Note that in the last transition in Figure 3(A), u_3 is not allowed to read D_2, because u_3 is not a member of readers of D_2 (second component of its label). The screenshot in Figure 3(B) shows that our implementation denies user u_3 view of D_2. Compare this to the original application screenshot in Figure 1(C), where u_3 is permitted to view D_2.



(A)



(B)

Fig. 2. (A) Information flow diagram for attack 1. (B) RWFM mitigates attack 1, by not showing D_2 in Bob's view.

4.4 Discussion

In [14], authors show how cross-domain access to sensitive information can be enabled by using IFC on the client side. However, their approach forbids propagation of sensitive information which limits its usability. Their technique facilitates DOM (Document Object Model) operations and function calls within scripts at the add-on level, thus, enabling effective control over existing channels for cross-domain communication. A further limitation of their approach is that users will have to label and identify information important to them, thus resulting in a huge burden.

Client side mashup security is discussed in [16]. A security lattice of syntactic labels is used to label the classified objects, where an object's label corresponds to the origin from which it is loaded, and are used to track information flow within the browser. Their approach supports usage of a range of techniques, such as static and/or dynamic analysis to ensure that information from one origin does not flow to another one unless the information has been declassified. Using syntactic labels and purely discretionary downgrading rules leads to corrosion of flow lattice.

Salient features of RWFM implementation are:

1. Our implementation complements the existing access control logic.
2. No major code revisions have to be done for implementing this model.
3. Our implementation does works with any access control mechanism.
4. Subject and Object labels can be accessed at any point of time, this can play a vital role during a security audit.
5. RWFM accurately handles dynamic label changes automatically.
6. This model can scale on par with any web application project with minimal development overhead.

5 Conclusions

In this paper, through a case study, we have clearly demonstrated the possible attacks that might occur in web applications if we don't follow proper information flow control, and demonstrated how these can be overcome easily by using the RWFM model. Based on the experiences of this work, we plan to build a general-purpose re-usable IFC implementation to protect information flow in web applications developed using Django, such that any Django web application developer with the knowledge of information entering and leaving the objects in the system can easily and automatically enforce proper information flow restrictions by invoking this package at appropriate points in the application.

References

1. Django, https://www.djangoproject.com
2. Facebook, http://www.facebook.com
3. LinkedIn, http://linkedin.com
4. Twitter.com, http://twitter.com

5. Google Docs, https://www.google.co.in/docs/about/
6. Microsoft Office Online, https://office.live.com
7. Lampson, Butler W.: “Computer security in the real world.” Computer 37, no. 6 (2004): 37-46.
8. Gruber, Tom.: “Collective knowledge systems: Where the social web meets the semantic web.” Web semantics: science, services and agents on the World Wide Web 6,4-13 (2008)
9. Harrison, Michael A., Walter L. Ruzzo, and Jeffrey D. Ullman.: “Protection in operating systems.” CACM 19.8 461-471, (1976)
10. David Ferraiolo and Richard Kuhn. “Role-Based Access Control.” 15th NIST-NCSC (1992):554-563.
11. Barkley, J., Cincotta, A., Ferraiolo, D., Gavrila, S., and Kuhn, D. R. (1997, April). “Role based access control for the world wide web.” In 20th NCSC (pp. 331-340).
12. Kreizman, G.: “Technology Overview for Externalized Authorization Management,” https://www.gartner.com/doc/2358815/technology-overview-externalized-authorization-management
13. eXtensible Access Control Markup Language (XACML) Version 3.0, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
14. Murugesan, San. “Understanding Web 2.0.” IT professional 9.4 (2007): 34-41.
15. Li, Zhou, Kehuan Zhang, and XiaoFeng Wang. “Mash-if: Practical information-flow control within client-side mashups.” IEEE/IFIP DSN, 2010.
16. Ter Louw, Mike, Jin Soon Lim, and V. N. Venkatakrishnan. “Enhancing web browser security against malware extensions.” Journal in Computer Virology 4.3 (2008): 179-195.
17. Magazinius, Jonas, Aslan Askarov, and Andrei Sabelfeld. “A lattice-based approach to mashup security.” ACM 5th ASIACCS, 2010.
18. Philippe De Ryck, Maarten Decat, Lieven Desmet, Frank Piessens, and Wouter Joosen. “Security of web mashups: a survey.” Information Security Technology for Applications, LNCS 7127. 223-238 (2012).
19. Andrew C. Myers, and Barbara Liskov. “A decentralized model for information flow control.” ACM 16th SOSP. 129-142(1997).
20. Denning, D.E.: “Cryptography and Data Security”. Addison-Wesley, Reading (1982)
21. Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. “Information flow control for standard OS abstractions.” ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007, 321-334.
22. Andrei Sabelfeld, and Andrew C. Myers. “Language-based information-flow security.” Selected Areas in Communications, IEEE Journal on 21.1 (2003): 5-19.
23. Steve Zdancewic. “Challenges for information-flow security.” Proceedings of the 1st International Workshop on the Programming Language Interference and Dependence (PLID04). 2004.
24. DBpatterns, http://www.dbpatterns.com
25. OWASP, https://www.owasp.org
26. Denning, D.E.: “A lattice model of secure information flow.” Comm. ACM 19(5), 236–243 (1976)
27. N.V.Narendra Kumar, R.K.Shyamasundar. “Realizing purpose-based privacy policies succinctly via information-flow labels.” In: IEEE 4th BdCloud. 753–760 (2014)
28. N.V.Narendra Kumar, R.K.Shyamasundar. “POSTER: Dynamic Labelling for Analyzing Security Protocols.” In: ACM 22nd CCS, 2015.
29. M. Abadi. “Security protocols and their properties.” In Foundations of Secure Computation, NATO Science Series, pages 3960. IOS Press, 2000.
30. T. Y. C. Woo and S. S. Lam. “A lesson on authentication protocol design.” SIGOPS Oper. Syst. Rev., 28(3):2437, July 1994.