ENFORCING SECURE DATA SHARING IN WEB APP FRAMEWORKS THROUGH INFORMATION FLOW CONTROL

Susheel S, Naren N and
Prof. R.K Shyamasundar

Todays Talk

- Background and Motivation.
- Possible attack in a system without IFC.
- Our solution using RWFM and it’s overview.
- Video demo of the attack and it’s mitigation.
- Our solution (explained).
- Conclusion and Future Work.

How are Todays Web Applications Developed?

- Developers want to realize applications from concepts to launch as quickly as possible. Developers focus is on the **don’t repeat yourself** (**DRY**) principle.
- They use frameworks like Django, Ruby on Rails etc.
- Alleviate the overhead associated with common activities performed in web development.
- On a security point of view they provide necessary hooks that enable the developer to avoid the common security mistakes, like **Cross site scripting, Cross site request forgery, SQL injection, Clickjacking etc.**

Problems in Frameworks with Regard to Secure Information Flow

- Most of the frameworks have an authentication and authorization system built in (but very generic).
- Developers use **access control** as the main means for preventing information from being disseminated.
- Developers have to use **ad-hoc code** for managing permissions.
- Specified policy is **intertwined throughout code** which is error prone and is not scalable. (better options exist like XACML)
- **Difficult to "prove"** for audit or quality assurance purpose.
- Access control logic **prone to Trojan Horse attacks**.

Attack

- 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.

- Access control logic can control the release of information not its **subsequent propagation in the system**.

- Most of todays web applications are **collaborative and social** in nature such as Google Docs, Office Online etc. Controlling the flow of information among the various stakeholders is very much required in todays web scenario.

Solution

- 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.
- Information Flow Control (IFC) which is a form of mandatory access control prevents these types of indirect data misuse.
- Question - Which IFC model to use in the web setting?

RWFM Overview

RWFM Label Format

(admin, readers, writers)

- First component is a single subject denoting the *owner* of the information.
- Second component is a set of subjects denoting the permissible readers of the information.
- Third component is a set of subjects denoting the influencers of the information; also, denotes the permissible writers.

Permissible Flows in RWFM

- Given any two RW classes $RW_1=(s_1,R_1,W_1)$ and $RW_2=(s_2,R_2,W_2)$, information is allowed to flow from RW_1 to RW_2, denoted $RW_1 \leq RW_2$ only if $R_1 \supseteq R_2$ and $W_1 \subseteq W_2$. Formally,

$$\frac{R_1 \supseteq R_2 \qquad W_1 \subseteq W_2}{(s_1,R_1,W_1) \leq (s_2,R_2,W_2)}$$

Join and Meet of RW Classes

- Let $RW_1=(s_1,R_1,W_1)$ and $RW_2=(s_2,R_2,W_2)$, be any two RW classes.
- Their join ($\oplus$) and meet ($\otimes$) are defined as follows:

$$(s_1,R_1,W_1) \oplus (s_2,R_2,W_2) = (-,R_1 \cap R_2,W_1 \cup W_2)$$

$$(s_1,R_1,W_1) \otimes (s_2,R_2,W_2) = (-,R_1 \cup R_2,W_1 \cap W_2)$$

- For permissible flows, and join and meet, the first component – administrative component – does not play a role.

Experimental Demonstration of the Attack

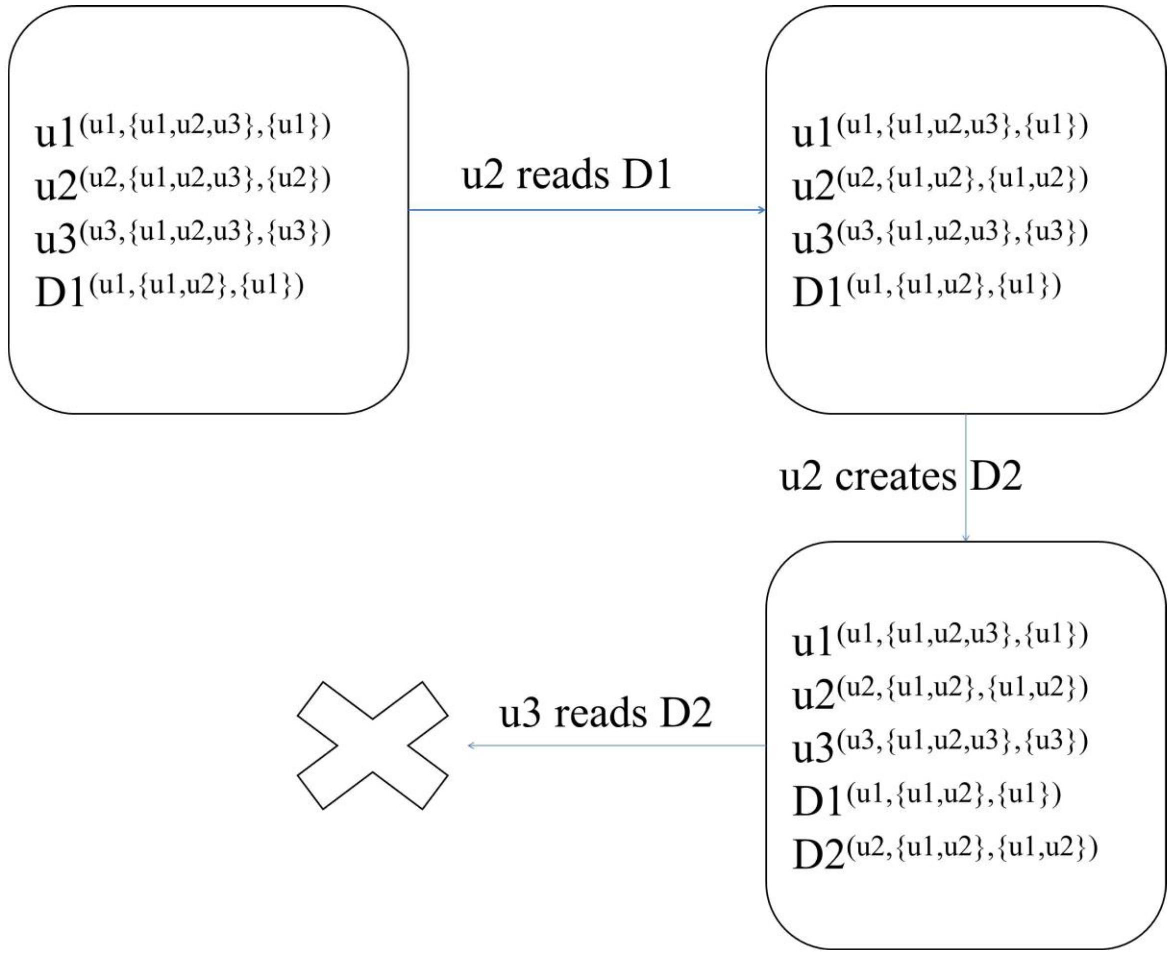
- The problems discussed the previous slides exist in most of the applications and not merely theoretical.
- We have chosen an application written using Django which helps users in creating, sharing and exploring database schemas or models (A perfect example of todays web applications social, collaborative and RESTful).
- – **isPublic** 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 **canView or canEdit**.
- This kind of implementation common in many web applications like Google Docs, Dropbox etc.

Attack Scenario in DBpatterns

- Jack shares a document, say D1, with Alice. Alice forks D1 and makes a local copy, call it D2. Alice now owns D2 and can set access restrictions on it.
- Alice gives read permission on D2 to Bob. This way Jack will be unaware of an information leak and Bob who did not have authorization to view D1 can now read D2 which essentially has the same information as D1.

- These kind of attacks that bypass access control logic are called Trojan horses.
- **Possible on any discretionary access control mechanism**, and are not specific to this web application.

Mitigating the Attack using RWFM


$u1^{(u1,\{u1,u2,u3\},\{u1\})}$
$u2^{(u2,\{u1,u2,u3\},\{u2\})}$
$u3^{(u3,\{u1,u2,u3\},\{u3\})}$
$D1^{(u1,\{u1,u2\},\{u1\})}$
u2 reads D1
$u1^{(u1,\{u1,u2,u3\},\{u1\})}$
$u2^{(u2,\{u1,u2\},\{u1,u2\})}$
$u3^{(u3,\{u1,u2,u3\},\{u3\})}$
$D1^{(u1,\{u1,u2\},\{u1\})}$
u2 creates D2
$u1^{(u1,\{u1,u2,u3\},\{u1\})}$
$u2^{(u2,\{u1,u2\},\{u1,u2\})}$
$u3^{(u3,\{u1,u2,u3\},\{u3\})}$
$D1^{(u1,\{u1,u2\},\{u1\})}$
$D2^{(u2,\{u1,u2\},\{u1,u2\})}$
u3 reads D2

Notation:

- u1, u2 and u3 denote Jack, Alice, and Bob respectively.
- S = {u1, u2, u3} denotes the set of all the users of the system.
- Entity (subject or object) e with label l is denoted e^l.

Salient Features of RWFM Implementation

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

Conclusions

- Clearly demonstrated the possible attacks that might occur in web applications if we don’t follow proper information flow control.
- Shown how RWFM can help.
- We have developed a prototype package in the framework of Django and the developer can use this package to overcome the security flaws automatically by complementing our package with the existing access control.

Future Work

- Build general purpose re-usable IFC by implementing RWFM within Web Application Frameworks.

- Implement various kinds of applications using Web Frameworks with RWFM and see how well it performs.

Thank You