iLayer: Toward an Application Access Control Framework for Content Management Systems

Gorrell Cheek, Mohamed Shehab, Truong Ung, Ebonie Williams
Department of Software and Information Systems
College of Computing and Informatics
University of North Carolina at Charlotte
Charlotte, NC
{gcheek, mshehab, tung4, enwillia}@uncc.edu

***Abstract*—Content Management Systems (CMS) simplify the process of web content creation, publication, and management. Many CMS platforms are extensible via third party developed applications providing additional functionality such as search, site navigation, and location services. However, most CMS platforms don't have manageable access control mechanisms that regulate third party applications. Unfettered and unchecked access of third party applications is a security vulnerability that puts web sites at risk. We introduce iLayer – an Application Access Control Framework for Content Management Systems. iLayer is a least privilege based model that protects content management systems from third party developed applications. iLayer makes policy recommendations to CMS administrators for third party applications. These policies are reviewed and set by the CMS administrator and enforced by the iLayer Framework. To verify the feasibility of our approach, we implemented a prototype of our framework on a popular open source content management system.**

I. Introduction

Content Management Systems (CMS) are used to simplify the process of web content creation, publication and management. Traditionally, they don't require in-depth technical knowledge of operating systems, programming languages, etc. There are both commercial and open source varieties on the market today including Joomla, WordPress, MediaWiki, Plone, and Drupal. Content management systems are deployed in various forms supporting a variety of different market segments including online publishing, online social networking, enterprise content management, and document management. Content management systems support many companies, government entities, and academic institutions. For instance, Whitehouse.gov, Popular Science Magazine, and the New York Observer web sites are all powered by Drupal [6]. Joomla supports several notable web sites including Harvard University and the United Nations Regional Information Center [12].

Third party applications expand the capabilities and functionalities of content management systems. There are thousands of applications available for most CMS platforms providing functionality like news publishing, location mapping, photo galleries, etc. Can all these applications be trusted? Few formal application development security practices are in place, e.g., code reviews of third party applications, etc. There are some secure coding guidelines available [19], [1], [13], [11]. But, there is little enforcement of their use.

We believe that web site administrators need additional tools and mechanisms to protect their information assets from attacks via third party applications. We propose an application access control framework for content management systems that is based on a least privilege security model [18]. This framework gives visibility into the accesses that third party applications request, in addition to aiding and guiding administrators in making policy decisions. The security policies are set at installation time and are enforced at run time. The framework adds another layer of protection to the web site and is an improvement of how CMS platforms implement access control for third party applications. For example, some content management systems regulate access to third party applications via file permissions, which is cumbersome and difficult to administer. Administrators would need to analyze the application, know the intricacies of the CMS platform, and be familiar with the underlying operating system to effectively set the appropriate file permissions. This is a difficult and tedious undertaking. With our proposed approach (iLayer), the CMS administrator need only set the policy at installation time and the framework enforces that policy at run time.

Our contribution is an Application Access Control Framework for Content Management Systems. Our framework: 1) is based on a least privilege model, 2) protects content management systems from third party applications, 3) protects CMS third party applications from other CMS third party applications, and 4) provides CMS administrators with third party application policy setting functionality, including recommendations for policy settings. Finally, we implemented a prototype of our framework on an open source content management system.

The rest of the paper is organized as follows. Section II provides an overview of content management systems. Section III outlines the motivation behind our work. We describe our iLayer Framework in Section IV and Section V provides an overview of our prototype. Finally, we discuss related work in Section VI and conclude the paper in Section VII.

II. Preliminaries

A. Content Management Systems Overview

A Content management system is an online application that provides users the ability to create, design, publish and manage

the content of a web site. In addition, content management systems manage work flow allowing for collaboration. Users of content management systems are not required to have an in depth technical knowledge of web design or programming languages. Users traditionally just leverage a browser based interface to easily build and manage the content of a web site. Content management systems support multiple users with varying roles, e.g., content contributors, content consumers, site administrators, etc.

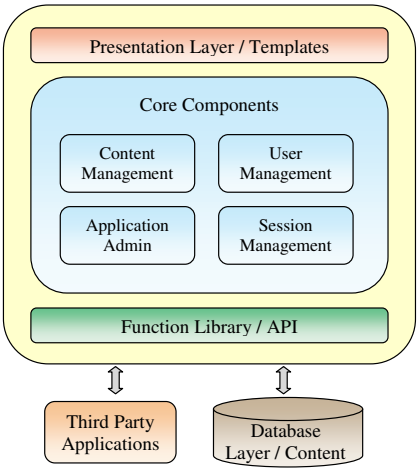


Fig. 1. Content Management System

Typically, a CMS consists of four components: Presentation Layer, set of Core Components, Function Library, and Database Layer. See Figure 1. What follows is a brief description of each:

1) **Presentation Layer:** Displays to the visitor of the web site the output (or content) of the CMS. Typically, templates populated with content are used to facilitate the creation of the web site.
2) **Core Components:** Provides foundational CMS functionality including basic content management, user management, application administration, session management, etc.
3) **Function Library / API:** Library of functions that perform various tasks (e.g., database calls, etc.). In addition, an Application Programming Interface (API) is provided for interfacing with third party applications.
4) **Database Layer:** Stores all content such as users' profiles, blogs, files, etc. It also contains configuration and policy management information.

One of the primary benefits of most content management systems is that they can easily be expanded to include additional functionality. Content management systems publish their APIs which allow third party developers to create applications that interface with the CMS. These third party applications provide additional capabilities. For example, a third party developed calendar application can provide user scheduling management. Or, a third party developed mapping application can provide location information for a restaurant web site. More often than not, these applications are free to the community to download, install and use.

B. Application Access Control Approaches in Content Management Systems

Content management systems provide user access control and management capabilities. However, access control functionality for third party applications is not as well developed. Third party applications can, and in some cases do, have full administrator level access to the web site's content and data, e.g., configuration information, user information, passwords, etc. CMS platforms, such as WordPress, Plone, and Joomla, allow web site administrators to easily install third party applications. But, most CMS platforms advise the administrator to secure their database and server configurations and change all the default passwords after installing new versions of software [21], [14], [12]. CMS platforms have traditionally addressed application access control as an after thought.

Third party applications developed for WordPress have full access to the content management system and the database, unless restricted through file permissions. There is no mechansim where an administrator can set permissions during the installation of a third party application. File permissions are not handled through a designated WordPress screen. Instead, the administrator must set the file permissions via the command line of an operating system shell. The administrator needs to find the right balance between restrictive and lax permissions [21]. Application and WordPress software functionality is affected if permissions are too strict. Permissions that are too lax present a security risk. The skills to set these permission levels is not universal among CMS administrators. Managing third party application access in WordPress is not trivial and can be easily misconfigured.

The Plone CMS platform has strong user-based access controls. Plone uses the Zope content management interface to set access control lists, groups, and roles [14]. Roles have permissions; roles are assigned to groups; and, users are added to groups. This approach is extended to third party applications as well. However, this access control framework is centered on the user and not the application. File permissions are the primary way to restrict the access of third party applications in Plone. Similar to other CMS platforms, Plone third party applications have full access to the content management system including the database.

Joomla is another CMS platform that allows third party developed applications to interface with its content management system. The administrator must also use file permissions to control the access given to these applications [12]. Other than file permissions prohibiting access, third party application access is not explicitly controlled like user access. There is not a permissions granting process during installation either.

Thus, even though there are methods that allow web site administrators to control the access of third party developed applications, they are limited in capability and are not very easy to use. The administrator would need to understand and

TABLE I
SAMPLE OF DRUPAL CORE DATABASE TABLES ACCESSED BY 412 THIRD PARTY APPLICATIONS

Table Name	Table Description	Potential Impact	% of Modules That Require Access
sessions	Contains user session information, e.g., userID, sessionID, user IP address, etc.	Session hijacking	2%
users_roles	Lists the assignments between users and roles	Privilege escalation	5%
node_revisions	Contains edits / revisions of node content	Content compromise	7%
permissions	Lists each user role's permissions	Privilege escalation	7%
users	Contains usernames, passwords, profile information, etc.	Account compromise	23%

translate the access requirements of the third party application into file permissions (e.g., in the case of Joomla) and, possibly, database and server configurations. This is not an easy undertaking and, more often than not, administrators take minimal or no action to secure their web sites from third party developed applications. The average web site administrator may not have sufficient skills or experience to know all of the risks associated with third party applications and how they may impact their web site.

III. MOTIVATION

Most CMS platforms have user access control mechanisms based on a least privilege security model that enforces the limits of what a user can and can't do. A user is assigned a role which has a set of permissions that relate to the actions that the user is allowed to perform. For example, an administrator level user has full access to the web site. They can add, modify, and delete content, and manage settings and applications. A content consumer user has more limited access. They can edit their profile and read (or consume) the web site's content.

A similar least privilege based approach to access control for CMS third party applications is not available. As previously mentioned, third party developed application access control is either non-existent or hard to configure for the average web site administrator, in addition to having limited capability. A least privilege based access control framework for third party developed applications within content management systems is needed. Applications need only be given access to the objects and resources they require, as opposed to the entire content management system.

Unfettered and unchecked access of third party applications is a security vulnerability that puts web sites at risk. For example, a hypothetical social networking web site called *Social123* is powered by a CMS platform. The administrator has installed several third party applications to customize and enhance the social networking site. One installed application, *BdayCal*, has full access to the database, yet only requires a subset of access, e.g., users' birthdays to be displayed on a calendar. An attacker can target *Social123* via *BdayCal* to gain access to the login credentials of all of the users on the social networking site. These credentials are stored in the *users* table within the database. A poorly written third party application may find itself vulnerable to a SQL injection attack which would expose all the tables, including the *users* table. Even though *BdayCal* only requires access to the table containing users' birthdays, all tables are available to the application and thus increase the risk of compromise and misuse.

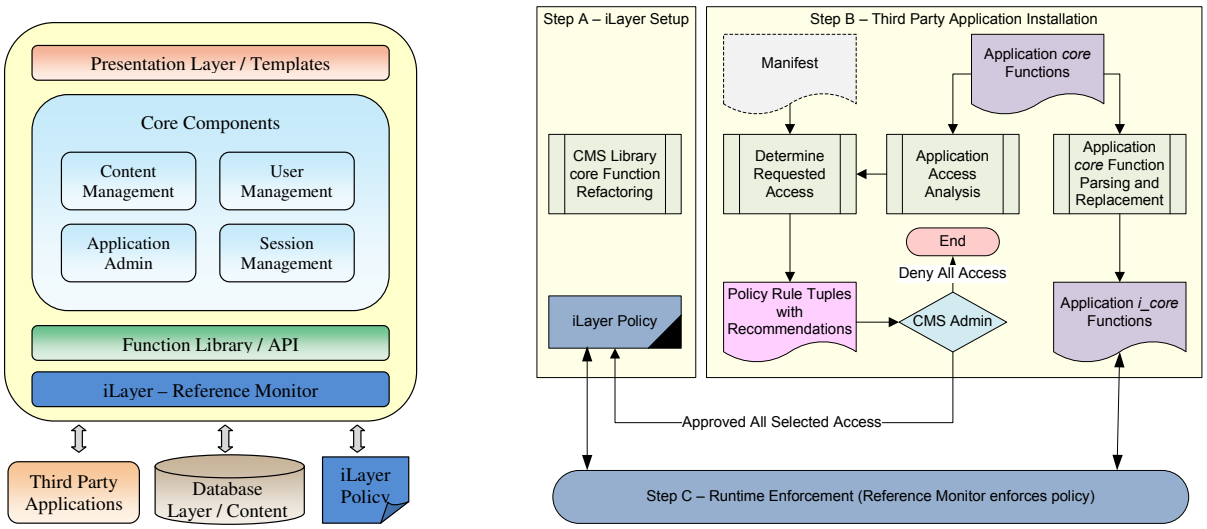
On a popular CMS platform called Drupal [6], we conducted a study of 412 popular third party applications in order to analyze the database calls made by them. Through static analysis, we extracted the database table accesses made by these applications. In Drupal, database calls are made by using the *db_query()* function. For every database call, we parsed through the SQL statement to find the database tables and recorded the results. We found that applications accessed tables they created and the core tables provided by Drupal.

We also found that third party applications have significantly more access to the Drupal core tables than what is required. Only 2% of the applications required access to the *sessions* table. That leaves 98% who didn't require access and yet had access thus leaving the web site vulnerable to session hijacking. See Table I. Only 7% of the modules required access to the *node_revisions* and *permissions* tables leaving roughly 382 applications (or modules) access to these tables and thus vulnerable to potential privilege escalation and content compromise attacks. The *users* table is a default table that holds basic user information such as user names and passwords. Approximately 77% of the applications didn't need access to this table but did have access. These are clearly not examples of least privilege and therefore the risk of compromise increases.

IV. ILAYER

In order to improve the security of content management systems, we introduce iLayer – an application access control framework. iLayer's primary goal is to set and enforce least privilege access for third party developed applications that integrate with content management systems. A subsequent design goal of iLayer is ease of use with minimal administration while enhancing the security of the content management system.

iLayer's Architecture has two primary components: Reference Monitor and a corresponding Policy. See Figure 2(a). The reference monitor controls all access from third party applications to the database. Basically, the reference monitor verifies that third party application database requests are allowed by the policy. There are three primary steps in establishing the architecture: iLayer Setup, Third Party Application Installation, and Runtime Enforcement. See Figure 2(b).



(a) iLayer Architecture

(b) Establishing the iLayer Architecture

Fig. 2. The iLayer Framework

A. iLayer Setup

The first step in installing the iLayer Architecture within a CMS platform is creating a table in the database to store and manage the iLayer Policy. A policy is made up of three components:

- ***subject:*** third party application that will be granted access
- ***object:*** database table being given access to
- ***permission:*** access privilege that is granted which could be either:
  1) *read* (select)
  2) *write* (delete, insert, update)
  3) *read & write*

After the policy table is set up, all core CMS platform functions that perform database calls are identified in the CMS Function Library, e.g., *core(arg)*, where *arg* contains the database table name and the requested action, i.e., select, delete, insert, or udpate. After which, the core functions are refactored [8], i.e., a corresponding *i_core(3PA_Params, arg)* function is added to the CMS Function Library for each core function that makes a database call, e.g., *core(arg)*. The *i_core(3PA_Params, arg)* function will have the same arguments as its corresponding core function with the addition of the calling third party application parameters, where:

```
3PA_Params = array(
      "name" => "Application Name is...",
      "id" => "Application ID is...",
      ...
)
```

The *i_core* function performs a policy check. If the policy is violated, access to the requested database table is denied and an error message is returned. If the policy is not violated, the *i_core* function calls its corresponding *core* function and operation proceeds normally. Figure 3 shows a code sample of a *core* function and its corresponding *i_core* function.

Original core()	**Refactored core() to i_core()**

function core(arg) {
…
//extract table name and action from the arg
…
}
	function i_core(3PA_Params, arg) {
	…
	//extract table name and action from the arg
	//loop for all table names
	if(matchPolicy(3PA_Params, table, action) = null)
	errorHandler();
	else
	core(arg);
	}

Fig. 3. Refactoring Core function

B. Third Party Application Installation

After the iLayer is setup within a CMS platform, third party applications can be installed. The first step in the installation process is the determination of the requested access by the third party application. Policy recommendations are then presented to the CMS administrator for review and the policy is selected and approved by the CMS administrator. After which, the third party application code is parsed and all instances of *core* functions are replaced with their corresponding *i_core* functions. Finally, the remaining installation steps for the third party application proceed normally.

1) Determination of Requested Access: Two approaches are leveraged in determining the requested access by the third party application: 1) Manifest provided by the third party developer, and 2) Application access analysis. A manifest is a file provided by the third party application developer that outlines the required and optional application privileges; the application developer declares all of the application's database accesses. The manifest is stored in XML format and contains a set of *(subject, object, permissions, required_flag, comments)* policy rule tuples. Figure 4 displays a sample file.

The *required_flag* indicates whether the access is required for the proper execution of the application. If the flag is

```
<manifest>
        <policy_rule id="pr1">
                <subject>appName</subject>
                <object>birthday_table</object>
                <permission>select</permission>
                <required_flag>0</required_flag>
                <comments>Access is not required; but...</comments>
        </policy_rule>
        <policy_rule id="pr2">
        …
        </policy_rule>
</manifest>
```

Fig. 4. Sample manifest file

not set, the *comments* field can be used by the developer to elaborate on the optional nature of the access, e.g., "Access to *birthday_table* is not mandatory; but if access is not provided, ages will not be displayed."

Regardless of whether a manifest is available, the application code is statically analyzed at installation time for all database calls. Similarly as for the manifest, the output of the application access analysis is a set of *(subject, object, permissions)* policy rule tuples that describe the third party application database accesses, e.g., *(application_name, database_table_name, read)*. In addition to identifying all database calls, all called third party applications functions are identified and their respective iLayer Policies are retrieved from the policy repository.

2) Setting the Policy: The manifest and the output of the application access analysis (to include all called application iLayer Policies) are presented to the CMS administrator as a series of policy rule tuples. In addition, CMS administrators are presented with additional information to assist them in making their policy decisions. For each policy rule tuple, a thumbs up or thumbs down policy rule recommendation is presented, where a thumbs up recommends adoption of the policy rule tuple and a thumbs down does not. This recommendation is an indicator of the community's usage of the policy rule tuple.

Policy Rule Recommendation: The thumbs up/down policy rule recommendation is based on the maximum likelihood of the set of possible permission combinations for all requested objects based on historically granted accesses. Let A equal the set of all possible accesses that can be requested by third party applications, $A = \{a_1, a_2, \ldots, a_n\}$, where a_i is an object to permission pairing, e.g., *sessions-read*. Let λ equal the set of all previously granted third party application accesses, where $\lambda \subseteq A$. Figure 5 displays a sample of previously or historically granted accesses (Application IDs 001 to 412), where the decision variable:

$$x_i = \begin{cases} 1 & \text{if } a_i \; is \; granted \\ 0 & \text{if } otherwise \end{cases}$$

When installing a third party application, the set of requested accesses (object-permission pairings) is denoted by R, where $R \subseteq A$. In our framework, the set R is obtained from the application access analysis and / or the manifest. For

Application ID	Granted Accesses (object - permission)								
	sessions - read	*Sessions - write*	*user_roles - read*	*user_roles - write*	*node_revs - read*	*node_revs - write*	…	*files - read*	*files - write*
001	0	1	0	0	0	1		0	0
002	0	0	0	0	0	0		1	0
003	1	1	1	0	0	0		0	1
…									
412	0	1	0	0	0	1		0	1
413	0	0	0	0	0	0		?	?

Fig. 5. Historically granted accesses of third party applications

example, in Figure 5, Application ID 413 is being installed and $R = \{$*files-read, files-write*$\}$. $\bar{R}$ denotes the set of accesses that were not requested. The accesses that were not requested ($\bar{R}$) are automatically not granted to the third party application.

The policy rule recommendation for the requested accesses is computed based on statistically examining the set λ. Without the loss of generality, let $R = \{a_1, a_2, \ldots, a_{r-1}\}$ and let $\bar{R} = \{a_r, a_{r+1}, \ldots, a_n\}$. The decision is not to grant $\bar{R}$ – therefore, using the decision variable, $\bar{R} = \{x_r = 0, x_{r+1} = 0, \ldots, x_n = 0\}$. For R, the values (or recommendations) for $x_1, x_2, \ldots, x_{r-1}$ are computed by:

$$X = \arg \max_{x_1,\ldots,x_{r-1}} P(x_r = 0, \ldots, x_n = 0 \mid x_1, \ldots, x_{r-1})$$

$X = \{x_1, \ldots, x_{r-1}\}$ is the set of recommended accesses that maximize the conditional probability of the set of accesses that were not requested (or granted) given the possible permission combinations for all requested objects taking into consideration the historically granted accesses λ. In other words, this mechanism chooses the recommendations X that are most probable based on historical accesses. For example, Application ID 413 requests two accesses {*files-read, files-write*}, see Figure 5. There are four possible recommendations, see Figure 6. The conditional probability $P(\bar{R}|X)$ is computed for each possible recommendation combination. In our example, allowing both requested accesses {*files-read, files-write*} are recommended because this combination has the maximum probability.

files - read	*files - write*	X	$P(\bar{R} \mid X)$
deny	deny	$\{x_1 = 0, x_2 = 0\}$	0
deny	allow	$\{x_1 = 0, x_2 = 1\}$	0
allow	deny	$\{x_1 = 1, x_2 = 0\}$	.2
allow	allow	$\{x_1 = 1, x_2 = 1\}$	.5

Fig. 6. Recommendation computation example

For each access x_i, the policy rule recommendation is presented to the CMS administrator in the form of a thumbs up/down where a thumbs up recommends the adoption of the requested object-permission pairing x_i and conversely for a thumbs down. For example, if $x_i = 1$, a thumbs up is presented

to the CMS administrator signifying a recommendation for this access request and a thumbs down would be presented when $x_i = 0$.

The number of conditional probability computations equal 2^n, where n equal the number of requested accesses. Our research shows that the number of requested accesses n is relatively small and therefore manageable from a computational perspective. Figure 7 shows the distribution of the number of accesses (database table-permission) for 412 Drupal modules (third party applications). The average number of accesses is 2.45 and the median is 2.

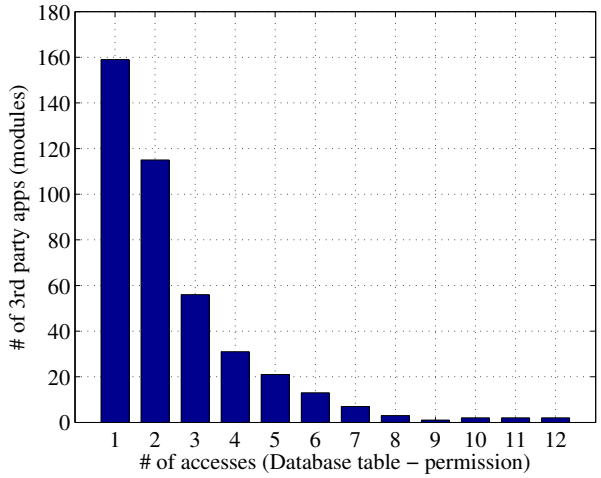


Fig. 7. Accesses of Drupal modules

Policy Decision: All the policy rule tuples are presented to the CMS administrator who can then can review each and grant the appropriate access as necessary, see Figure 9(b) for an example screen shot. After reviewing and granting access for each policy rule tuple, the CMS administrator has two options:

- **Approve all selected access:** All selected accesses (from both the manifest file and the application access analysis) are approved and the policy is written to the *iLayer_policy* table.
  - Note: Required accesses (as specified in the manifest file) cannot be disallowed. The only option, other than allowing the required accesses, is to "Deny all access".
  - Note: When only application access analysis is presented, some knowledge of the third party application is required; otherwise, selectively disallowing access could lead to unpredictable behavior at execution.
- **Deny all access:** All access presented is denied and installation of the application is terminated.

3) Function Parsing and Replacement: After the policy is composed and stored in the *iLayer_policy* table, the third party application code is parsed and all instances of *core* functions are replaced with their corresponding *i_core* functions that were added to the CMS Function Library during the iLayer Setup phase. The remaining native installation steps for the third party application are unchanged and the installation of the application proceeds normally.

C. Runtime Enforcement

Upon execution of the third party application, the iLayer Reference Monitor enforces the iLayer Policy that was set at installation time. The default action of the reference monitor is to deny all access, i.e., if there does not exist an explicit permit statement in the policy, the reference monitor denies all access attempts. During execution time, the *i_core* functions are called and thus invoke a policy check for every database call. The *i_core* function takes the third party application name and the query arguments as its parameter. It will then extract the table name, action (select, delete, insert, or update) from the query arguments and try to establish a match with one of the policy statements in the *iLayer_policy* table. If a match is found, the corresponding *core* function is called and normal operations are allowed to proceed. Otherwise, an *errorHandler* is called to display an access denied error message to the user.

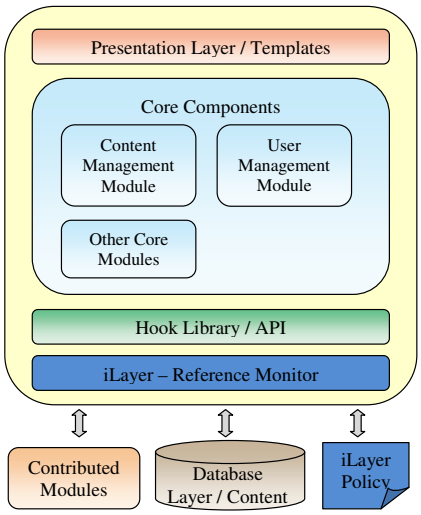


Fig. 8. Drupal iLayer Architecture

V. CMS Application Access Control Prototype

A. Drupal Overview

We prototyped our application access control framework on Drupal. Drupal is a popular open source CMS platform capable of providing a wide range of services, from personal web sites to the foundation of a social networking site. Like other CMS platforms, Drupal has a mature user-based access control framework. Also, like other CMS platforms, third party applications allow Drupal web site administrators to add custom capabilities and features. And, similarly to other CMS platforms, Drupal has very limited third party application access control capabilities – mainly only in the form of file permissions.

Using the concepts of modules, nodes and hooks, Drupal provides an abstracted approach to handling web content and functionality [20]. Drupal is based on a modular framework. Its functionality is provided in the form of these modules or

Original db_query()

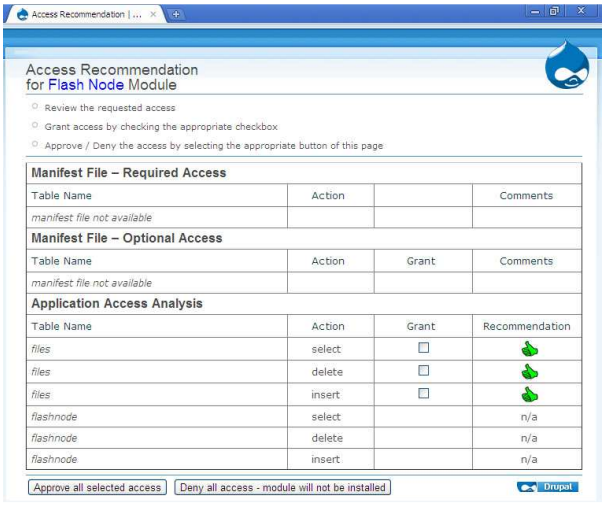
```
function db_query($query) {
  …
  //extract table name and
  action from the $query

  return _db_query($query);
}
```

Refactored db_query() to idb_query()

```
function idb_query($moduleID, $query) {
    …
    //extract table name and
    action from the $query
    //loop for all table names
    foreach($strArray as $key => $table_name) {
        $action = 0;    // 0 is read, 1 is write
        if($strArray[0] ==”select”) $action = 0; else $action = 1;
        …
        $result = db_query("SELECT count(*) as cnt
            FROM {iLayer_policy} WHERE
            moduleID = '%s' and
            tableName = '%s' and
            grantAccess = %d",
            $moduleID, $table_name, $action);
        $row = db_fetch_object($result);
        if($row->cnt  < 1)
            //no match found; display error; return null
    }
    //after all matches found, forward call to original _db_query
    return _db_query($query);
}
```

(a) Refactoring db_query function



(b) iLayer Policy review page for Drupal Flash Node Module

Fig. 9. iLayer DB refractoring and the policy review process

applications. Modules can be part of the base installation - built in core modules. Modules can also be developed by the Drupal community, called contributed modules and previously referred to as third party applications. See Figure 8. There are thousands of contributed modules available for download. These modules expand the functionality of Drupal. But, they also increase the attack surface and thus potentially increase the risk of compromise.

The primary way to access Drupal functionality is via hooks implemented in modules. In the most basic sense, hooks act as functions and are stored in the Hook Library. Drupal functionality is usually delivered in the form of content. A node in Drupal is a piece of content. Drupal supports many different node types, e.g., blog node type, page node type, story node type, etc. The characteristics of each node (or piece of content) is inherited from a node type. Therefore, managing content by node type impacts all nodes of that type. For instance, in order to add a user access rule for a blog entry node, the administrator only needs to modify the blog node type and the change will propagate to all of the other blogs on the site.

B. Drupal iLayer Setup

The first step in implementing the iLayer Architecture in Drupal is setting up the policy table. We created a table called *iLayer_policy*. Policy statements are stored in the policy table and are made up of three components: the module name (subject), the database table name (object) and the access request (permission). Next, all the core Drupal hooks (functions) that perform database calls are identified and refactored. We chose to refactor the *db_query()* hook for our prototype because *db_query()* is the main Drupal hook that every module uses in order to execute a database call / query. We added the refactored *db_query()* hook, *idb_query()*, to the Drupal Hook Library. It has the same arguments as *db_query()* with the addition of the calling module name. The *idb_query()* hook performs a policy check. If the policy is not violated, *idb_query()* forwards the call to the original *db_query()* hook and operation proceeds normally. If the policy is violated, access to the requested table is denied and an error message is returned. See Figure 9(a) for a code sample of *db_query()* and its corresponding *idb_query()* hook.

C. Contributed Module Installation

After the iLayer Architecture has been setup on the Drupal platform, we then can install third party applications or contributed modules. One Drupal module we installed in our prototype was Flash Node [17]. Flash Node allows CMS administrators to easily add flash content to their sites. As part of the module installation process, the access control policy is presented to the CMS administrator. Since Flash Node doesn't come with a developer generated manifest file that declares the required and optional accesses for the module, application access analysis is performed to determine the requisite database calls for Flash Node. The output of the static analysis is presented to the CMS administrator for review and action, see Figure 9(b). The CMS administrator is presented the module policy in the form of a subject (Flash Node), object (Table Name) and permissions (select, delete, insert, update) policy rule tuple. The CMS administrator reviews the policy, to include the policy rule recommendation in the form of a thumbs up/thumbs down, where thumbs up means the policy rule is recommended and conversely for a thumbs down. After which, the CMS administrator grants the appropriate access by checking the box for each selected policy rule tuple and then clicks on *Approve all selected access*. The policy is written to the *iLayer_policy* table. After which, all instances of *db_query()* in the Flash Node module are replaced by the refactored *idb_query()* hook. The remaining native Flash Node module installation steps proceed normally.

D. Runtime Enforcement

Upon execution of the Flash Node module, database accesses are made by calling *idb_query()*. The *idb_query()* hook

takes the module name and query arguments as its parameters. The called database table name and action (select, delete, insert, or update) are extracted from the query arguments. These two parameters coupled with the module name are used as criteria to find a policy statement match in the *iLayer_policy* table. If a matching policy statement is found, access to the table is granted and execution proceeds normally. If a policy statement is not found, an error message is displayed and access to the requested table is disallowed.

VI. Related Work

Barth et al. analyzed the security implications of browser extensions [3]. Browser extensions are third party developed applications that add functionality to the browser. Firefox is a popular web browser, in part because of its extensions. By default, extensions are given full privilege to the web browser. Barth et al. conducted a case study to assess the access privileges granted versus needed by Firefox extensions. They found that 88% of extensions did not need full privileged access. They proposed a browser extension system that imposes the principle of least privilege. The approach presented requires developers to rewrite their code, to include adding a manifest file declaring requisite accesses, in order to be able to leverage the enhanced browser extension system. The iLayer Framework also is based on least privilege, in addition to allowing for a manifest file - though not mandatory. Requested accesses can also be obtained within the iLayer Framework via application access analysis during the installation process allowing for backward compatibility.

Enck et al. researched how smartphones fail to provide visibility and control over third party developed applications [7]. They introduce TaintDroid, which monitors and tracks third party developed Android applications' use of sensitive data. Their research found that the potential for misuse of users' sensitive data is great amongst third party applications. TaintDroid dynamically monitors the behavior of these applications at run time. iLayer statically analyzes the third party application accesses at installation time. But, iLayer goes one step further by enforcing set policies at run time.

Significant research has been dedicated to collaborative filtering recommendation systems where the community's interests are a predictor for a specific user's interests [4], [9], [10], [2]. There also has been much work published on the privacy risks of recommendation systems [16], [15], [5]. But, little research has been dedicated to using recommendation systems for access control policy settings. We introduce a novel approach that leverages the community's access control settings to make recommendations for third party application policies.

VII. Conclusions / Future Work

We presented an Application Access Control Framework for Content Management Systems. Our framework is based on least privilege and provides content management systems protection from third party applications. In addition, our framework provides CMS administrators third party application policy setting functionality, including a policy setting recommendation capability that enhances knowledge in making policy decisions. Finally, we implemented a prototype of our access control framework on the Drupal Content Management System platform.

We plan to extend our work by providing CMS administrators the ability to review and update third party application policies post installation. Our framework will also be extended by expanding our view of *objects* beyond database tables to include function calls. We plan to benchmark the performance impact of policy enforcement. In addition, we plan to conduct a detailed survey and analysis of threat vectors. Finally, we plan to further validate our model via an expanded user study.

References

[1] CERT Secure Coding. "http://www.cert.org/secure-coding", 2010.

[2] X. Amatriain, N. Lathia, J. M. Pujol, H. Kwak, and N. Oliver. The wisdom of the few: a collaborative filtering approach based on expert opinions from the web. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 532–539, New York, NY, USA, 2009. ACM.

[3] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting browsers from extension vulnerabilities. In *NDSS*, 2010.

[4] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the14th Annual Conference on Uncertainty in Artificial Intelligence (UAI98)*, pages 43–52, 1998.

[5] S. Chen and M.-A. Williams. Towards a comprehensive requirements architecture for privacy-aware social recommender systems. In *APCCM '10: Proceedings of the Seventh Asia-Pacific Conference on Conceptual Modelling*, pages 33–42, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[6] Drupal.org. Drupal - Open Source CMS. "http://Drupal.org", 2010.

[7] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of OSDI 2010*, October 2010.

[8] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.

[9] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, New York, NY, USA, 1999. ACM.

[10] R. Jin, J. Y. Chai, and L. Si. An automatic weighting scheme for collaborative filtering. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 337–344, New York, NY, USA, 2004. ACM.

[11] Joomla! Secure Coding Guidelines. "http://docs.joomla.org/Secure_coding_guidelines", 2010.

[12] Joomla.org. Joomla! "http://www.joomla.org/", 2010.

[13] OWASP. Open Web Application Security Project. "http://www.owasp.org", 2010.

[14] Plone.org. Plone CMS: Open Source Content Management. "http://Plone.org", 2010.

[15] N. Ramakrishnan, B. Keller, B. Mirza, A. Grama, and G. Karypis. Privacy risks in recommender systems. *Internet Computing, IEEE*, 5(6):54 –63, Nov/Dec 2001.

[16] J. Riedl. Personalization and privacy. *Internet Computing, IEEE*, 5(6):29 –31, Nov/Dec 2001.

[17] S. Greenfield. Flash Node. "http://drupal.org/project/flashnode", 2010.

[18] J. H. Saltzer. Protection and the control of information sharing in multics. *Commun. ACM*, 17(7):388–402, 1974.

[19] R. C. Seacord. *The CERT C Secure Coding Standard*. Addison-Wesley Professional, 2008.

[20] J. K. VanDyk and D. Buytaert. *Pro Drupal Development*. Apress, Berkely, CA, USA, 2007.

[21] WordPress.org. WordPress. "http://WordPress.org", 2010.