Indonesian Journal of Electrical Engineering and Computer Science
Vol. 16, No. 2, November 2019, pp. 860~881
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v16.i2.pp860-881

Security assessment of four open source software systems

Faraz Idris Khan[1], Yasir Javed[2], Mamdouh Alenezi[3]
[1,2,3]Security Engineering Lab, Prince Sultan University, Riyadh, South Korea
[2]Network Security Research Group, Faculty of Computer Science and Information Technology,
Universiti Malaysia Sarawak, Malaysia

Article Info

Article history:

Received Jan 29, 2019
Revised Apr 30, 2019
Accepted May 21, 2019

Keywords:

Open source software security (OSS)
Secure programming
Secure software development
Software security

ABSTRACT

Incorporating Open Source Software (OSS) tools in software development is increasing day by day due to their accessibility on the internet. With the advantages of OSS comes disadvantages in terms of security vulnerabilities. Therefore, in this paper, we analyzed four famous open source software tools (i.e. Moodle, Joomla, Flask and VLC media player) which are used by software developers nowadays. For the analysis of each system, security vulnerabilities and weakness were identified, threat models were modeled and code inspection was performed. The findings are discussed in more details.

Corresponding Author:

Faraz Idris Khan,
Security Engineering Lab,
Prince Sultan University,
Riyadh, South Korea.
Email: fikhan00@gmail.com

1. INTRODUCTION

Technology advancement has enabled the creation of smaller and cheaper computing devices. Because of such developments, it has made possible for almost anyone to own multiple devices which can be connected to the internet. Such devices are used by people in their everyday life. It is possible to do communication, perform important tasks anywhere at any time because of such a wide availability of computing devices.

As our reliance on computing devices and their internetworking are increasing day by day, at the same time concerns of security is also proportionally increasing. Because of the penetration of these devices in our everyday life activities, the information that these devices hold is of extreme importance in terms of privacy and security. Revealing such information lead the attackers to sabotage the normal operation of the computing devices and the services provided by them. Especially, the growing popularity of e-commerce application has also raised the security threat for organizations relying on such online applications.

System security is affected by software security problems. As attackers can exploit the defects in the software to sabotage the normal operation of computer systems. Open source softwarehas further aggravated the situation.

Nowadays free of cost and widespread accessibility to open source software have motivated organizations to leverage a variety of open source products. These open source products include libraries available for code, operating systems, software stack,and applications. Using open source incurs a cost at the same time enables flexibility but put forth challenges in terms of security. Security of open source has become a key concern for enterprises, which are thinking of incorporating it as part of the software stack. Open source software is developed by communities of software programmers who write code publically available to everyone over the internet. Due to this fact, open source software is less secure as compared to proprietary applications. The other major concern of open source software is that the open source developer's community is slow in releasing software patches handling known vulnerabilities in the software.

In order to contribute towards the study of open source software security we selected four popular open source software used by developers. We deeply analyze the architecture of open source software in terms of security flaws. For each open source software we highlight security weaknesses by performing code inspection and discussed possible solutions. Most of the latest work [1-5] in analyzing security of these four open source softwares are descriptive and hard for any one who is not expert of security to comprehend open source software security issues.We summarized our analysis which can act as a guide for sofware engineers who are not expert of network security. The study can help the software development to quickly review the security issues of popular open source software which can assist in developing secure software. Contribution our work is as follows.

a) Summarize security vulnerabilities of each of the open source software
b) Perform code inspection of each open source software to highlight security weaknesses and possible solutions

We organize our paper as follows. Section 2 discusses the literature review. In Section 3, we discuss our research method, where we discuss security weaknesses and concerns of four famous open source software tools used by developers. Along with the weaknesses and issues, we discuss possible solutions. In Section 4, we present our results of code inspection of four open source software systems. Section 5, presents discussion on the highlighted security issues. In section 6, we conclude our paper.

2. LITERATURE REVIEW

Research in open source software security has received attention from the research community. In this section, we discuss some of the efforts in this direction. In [6], the authors, seek to probe into how the organizations have incorporated security in open source software. Furthermore, the authors put an effortinto classifying the ways in which security in open source software is incorporated. The authors in [7] identified the gaps in open source software security and perform studies on the issue which resulted in suggestions for anyone who is going to use open source software. The authors in [8] perform analysis on around a hundred fifty research papers and identified the open source project communities which received attention from the research community and research questions which are put forward by the researchers.

In [9] the authors presented a brief review of the research on software development related practices for ensuring security in open source software. The authors in [10] summarized the literature on open source software security and also presented, findings in the literature, in categorized form. Also, in [11] the authors reviewed some research on open source software security and organize them into categories.

Software vulnerability assessment research is categorized into two types of software vulnerability analysis and software vulnerability discovery. Analysis of software vulnerability is more focused on discovering characteristics of vulnerabilities, which helps in determining the cause. Along with cause, it also helps in determining position. On the other hand, software vulnerability discovery attempts to search for existing but undiscovered vulnerabilities. Previous work in the direction of software vulnerability analysis can be found in [12]. And, previous work in software vulnerability discovery can be found in [13].There are various techniques proposed in the literature for software testing such as that found in [14].

As web applications are dependent on internet connectivity which make them vulnerable to security attacks over the internet. Such applications are usually subjected to attacks which usually modifies sensitive data and affect the availability of the services to authorized users. Open web application security project (OWASP) have identified top 10 web application security risks which are injection, cross-site scripting (XSS), broken authentication and session management, insecure direct object references, cross-site request forgery, security misconfiguration, insecure cryptographic storage, insufficient logging and monitoring, broken access control and insecure deserialization. There are various techniques in literature to counter these web application security vulnerabilities. Such work can be found in [15-16]. Usually techniques like static analysis or automated black box testing. Techniques like fault injection are employed to counter vulnerabilities in web applications [17-18] authors have used a cross-site scripting technique for testing vulnerabilities in open source applications. [19-20] highlights in the selection of software metrics that can help in the evaluation of the system that covers the evaluation of the security of the system. [20] highlights how cyclomatic complexity and source code can help in detecting vulnerabilities. The more source code keeps on adding or sustain vulnerabilities in different versions as shown in [21].

In this paper, we present a security analysis of four well-known open source software systems, which are Moodle [22], Joomla [23], Flask [24], VLC media player [25]. We summarize the research efforts in identifying security challenges in these open source software systems. Websites which use content management systems (CMS) such as Joomla experience attacks like SQL Injection, remote file inclusion, directory traversal, cross-site scripting (XSS), Spam, remote command execution (RCE), file upload.Security-enhancing techniques for CMS (i.e. Joomla) against security flaws and vulnerabilities can be found in [26-27]. As e-learning software platforms are connected over the internet hence are vulnerable to web application security attacks. Manipulating e-learning platforms by exploiting security vulnerabilities leads to the situations where accessibility, availability,and reliability of the platform arebadly affected. Techniques of enhancing the security of the Moodle platform can be found in literature such as in [28]. Flask security challenges and analysis can be found in literature such as in [29]. In literature, there exists various security enhancing techniques for web applications frameworks like Flask one such work can be found in [30]. Security analysis of media players (i.e. VLC) can be found in [31].

3. RESEARCH METHOD: SECURITY ANALYSIS AND SOLUTIONS

In this section, we discuss security weaknesses, threat analysis and solutions to improve the security of four famous open source software tools used for software development. Findings related to each software is discussed section wise.

3.1. Moodle

3.1.1 Introduction

Moodle is a free and open-source learning management system written in PHP and distributed under the GNU General Public License.Developed on pedagogical principles, Moodle is used for blended learning, distance education, flipped classroom and other e-learning projects in schools, universities, workplaces and other sectors. As an E-learning tool, Moodle has a wide range of standard and innovative features such as calendar and Gradebook. Moodle is a leading virtual learning environment and can be used in many types of environments such as education, training,and development and in business settings. Users can download and install Moodle on a Webserver, such as Apache HTTP Server, and a number of database management systems, such as PostgreSQL, are supported. Pre-built combinations of Moodle with a Web server and database are available for Microsoft Windows and Macintosh. Other automated installation approaches exist, such as installing a Debian package, deploying a ready-to-use TurnKeyMoodle appliance, using the Bitnami installer, or using a "one-click install" service such as Installatron. Certified Moodle Partners provide other Moodle services, including hosting, training, customization and content development. This network of providers supportsthe development of the Moodle project through royalties. There are four types of users supported by Moodle administrators, teachers, students,and guests.

3.1.2 Architecture

Moodle is structured as an application core, surrounded by numerous plugins to provide specific functionality. Moodle is designed to be highly extensible and customizable without modifying the core libraries, as doing so would create problems when upgrading Moodle to a newer version. So when customizing or extending your own Moodle install, always do so through the plugin architecture. Plugins in Moodle are of specific types. That is, an authentication plugin and an activity module will communicate with the Moodle core using different APIs, tailored to the type of functionality the plugin provides. Functionality common to all plugins (installation, upgrade, permissions, configuration,) are, however, handled consistently across all plugin types. The standard Moodle distribution includes Moodle core and a number of plugins of each type so that a new Moodle installation can immediately be used to start teaching and learning. After installation, a Moodle site can be adapted for a particular purpose by changing the default configuration option and by installing add-ons or removing standard plugins. The architecture of Moodle is shown in Figure 1 extracted from the the Moodle website and [32].

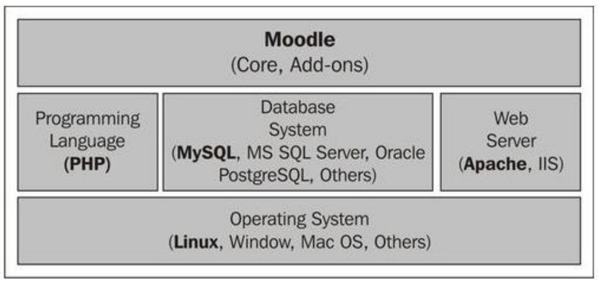

Moodle
(Core, Add-ons)
Programming Language (**PHP**)
Database System (**MySQL**, MS SQL Server, Oracle PostgreSQL, Others)
Web Server (**Apache**, IIS)
Operating System (**Linux**, Window, Mac OS, Others)

Figure 1. Moodle architecture

Topmost layer in the architecture is moodled core and plugins. The layer below the top layer contains the database systems and the Apache web server. The bottommost layer contains the operating system on which the Moodle run

3.1.3 Security Vulnerabilities

There is certainly a vested interest in acting maliciously in Moodle, especially from students since they have the biggest incentive with grades being so important to them, virtually all actors have a “good enough” incentive. It doesn’t have to be for personal gain or to cause harm to others, the reason can simply be “because it was funny”.
Consider the following examples:

1) A disgruntled administrator deciding to take it out on a particular user he/she doesn’t particularly like, kicking them of a course or removing material they uploaded
2) A user deducting grades from a student, for which a teacher will come under fire
3) A user adding grades go a student, for which the student will come under fire
4) A user replacing the material for a certain course with similar yet incorrect material harming both teachers and students
5) A user removes some material prior to a certain exam, as students tend to download material fairly close to examsUsing the STRIDE model, we can classify the threats facing Moodle as such:

a) Spoofing

Considering the nature of Moodle, revolving around education, and how roles operate, we can clearly see an interest in impersonating someone else (a student impersonating a teacher to add grades as a prime example).

b) Tampering

From the material alteration example, it can be clearly seen that altering material can cause serious damage,teachers being potentially fired or have their careersharmed for appearing to teach misleading or flat out wrong material.

c) Repudiation

As we established that there isavestedinterestinactingmaliciously,aswellastheharmthatsaid actions can do, it is crucially important that the actors do not have the ability to deny having committed the act.

d) Information Disclosure

There isn’t much private or potentially harmful whendisclosedinformationwhenitcomesto Moodle, but there are some that can be harmful. Consider a paid training course which uses Moodle, a user who is n’t enrolledin the course manages to getaccesstothematerialsofferedbythatcoursewhichisbyall metrics is considered stealing.

e) Denial of Service

As traffic to Moodle is normally tied to a special (an exam for example), DoS attack certainly is something to worry about.

f) Elevation of Privilege

As the role structured in such a way that each role has some degree of control over the one below it (admins create course, teachers control and enroll students to said courses, students get access to materials, guests can only see the course titles), users have a good incentive to gain privileges they wouldn’t normally have. A prime example would be a student gaining the privilege to edit their grades. In terms of information, we can use the CIA triad in order to get a betterunderstandingofwhattheyare and how they could be used to harm its users. Information Moodle has access to login information, course information (enrollment statistics, exam passwords, teaching material, etc), grades, user information (name, age, IP address to name a few), messages between users.

g) Confidentiality

Unauthorized access to someinformation,suchasviewingprivatemessagessentbetweenusers which is an invasion of privacy, or passwords which will harm the integrity of the data. However, some might not be as damaging as it is the case for viewing the grade of another student.

h) Integrity

As the business model of Moodle revolves around education, the majority of attacks will aim at altering grades or material, which has the potential to cause anything between ethical issues to criminal litigations.

Availability Since the traffic spikes around certain events, it is critical that Moodle be available during such times. With the information above in mind, it is clear that the integrity of data is the most important aspect, as the whole business model is built around sharing and storing information. Should tampering with data be a common occurrence, it will no longer be used.

3.1.4 Threat Model

In this section, we analyze the model and present a threat model that will help in identifying potential security exploits in the system.

The machine boundaries of the system depend on the platform you are using. For example, if you are using the ios app of Moodle on your phone then the machine boundary of the system is your phone (same applies to the Android app). But if you are accessing Moodle via a browser on your pc then the boundary will be the PHP server that is running the software. Threat model of moodle as shown in Figure 2.

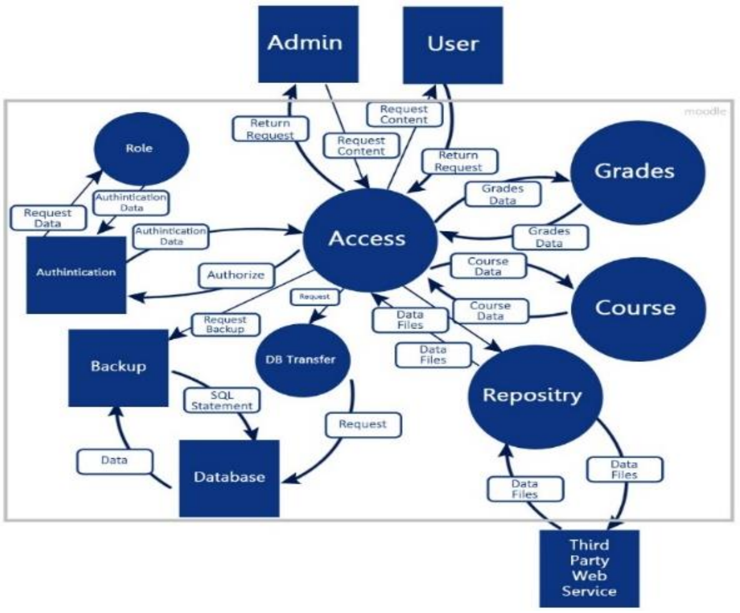


Figure 2. Threat model of Moodle

Moodle is already available on different platforms, so the system (with its subsystems) will not defer whether it is deployed on an ios app, android app, or web application. This also applies whether the software is developed using Java or any other programming language; the subsystems remain the same. The only thing that will defer is the machine boundary of the system.

As sees in the diagram, we only have one trust boundary in our system which contains the main subsystems (access, authentication, backup, DB transfer, database, course, grades, repository, and role) that makes up the trust boundary of our system.

3.2. Joomla

3.2.1 Introduction

Joomla is a content management system (CMS) used as a base template for any kind of web application that requires extensive management of data where users can customize their website by adding specific plugins that can facilitate their jobs in addition to making the template looks different.Joomla's user base consists of both large-to-medium entities, such as companies and governmental sectors and small entities, such as hobbyists and freelancers. These users could come from different domains, such as business domains and educational domains and others. Joomla's purpose is to provide a free, adaptable, collaborative, stable, secure, and easy-to-use web platform for all types of users from around the world, sustainable by volunteers. It is designed to be used by anyone even if they do not have a solid experience in dealing with the web development process.The main development of Joomla is done by Open Source Matters, Inc. However, many more features and extensions that make Joomla the popular CMS it is are being regularly developed by voluntary developers. In addition, independent developers can take the base application and fully customize it without any external modules.

In short, most vulnerabilities in Joomla compromise the confidentiality and integrity of the data on the system. In addition, most of these vulnerabilities and risks are caused not by Joomla itself but by the enormous amounts of unchecked open-source modules on the internet that can easily be installed on he system.

3.2.2 Architecture

The architecture of the Joomla system varies quite a bit based on the activity of the website using the CMS and the kind of extensions it's using. Nevertheless, regardless of what type of system it is, there are some universal components that compose almost all Joomla systems. From inspecting files and folders of Joomla. These components are as follows:

Database: All content and information that exists within Joomla systems including administrative information are stored and managed within a MySql database. In addition, any plugins or modules added to the system also use the database to store and retrieve any data that they may require.

The Joomla Framework: Which are all the basic functionalities and libraries that make the core Joomla system and works as a base for the development of modules, components, and plugins. Furthermore, it allows developers to use the functionalities of Joomla! to build an API service without the need for a whole website and its UI.

Components: They are considered a mini-application that runs on top of Joomla and usually has two parts. One part is the administrator part through which the component can be managed. The other part is the website part which renders whatever content the component manages and displays it for the user. An example of a component is the user management component, which is an essential and core (meaning it comes preinstalled with Joomla and requires no installation) component in Joomla which manages users of the website, in addition to their roles and privileges.

Modules: They are lightweight extensions that usually work as renderers rendering some content on the website. For example, there could be a module that displays a list of the most popular postings on the website.

Plugins/Extensions: Plugins and extensions are one of the most essential components in the architecture of Joomla since it allows the whole framework to be heavily extended. They allow adding additional functionalities and processes to the core framework in addition to any front-end functionalities. For example, user authentication can be a plugin added to the system.

Template System: The templating system does not only allow Joomla systems to have different colors or themes but also allows to completely revamp the website and control what is to be displayed or not. For example, templates can be used to change the default look of the website which is a simple article displaying a website into an e-commerce website selling all kinds of products.

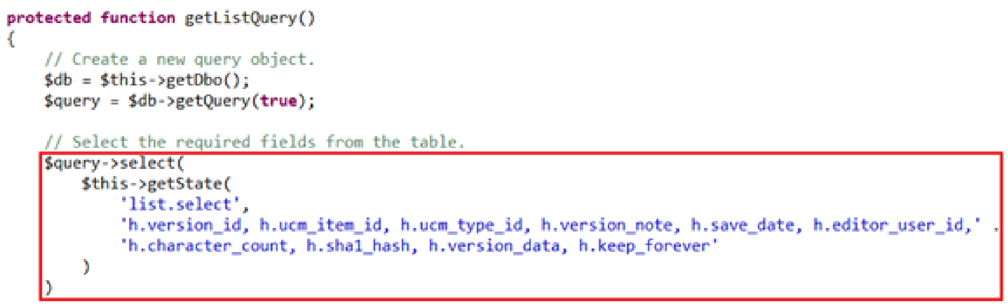
3.2.3 Security Vulnerabilities

Types of potential attackers heavily depend on the type of website like many other things. If the website is just a personal blog then most attackers might be script kiddies trying to exploit some vulnerabilities they found on the internet. On the other hand, websites that hold sensitive information that make a high profile target such as credit card information to deal with more sophisticated and experienced attackers.Most threats that Joomla faces are threats related to the confidentiality of the system. The reason being that most resources that are worth compromising the systems are in the databases of the system. Though to get access to the database an attacker might also need to alleviate their privileges.Attackers can try to find vulnerabilities through two main methods. The first method, which requires less experience and is most likely to be used by amateurs, is to look for it on vulnerability databases online, such as the Exploit-DB website. Sometimes these databases and resources also demonstrate how to exploit a vulnerability, which makes it convenient and easy for attackers to execute. The other method, which requires higher levels of experience, is to actually inspect the code of the system since Joomla's source code is open for everyone to see. This might not guarantee that the same vulnerabilities found in the source code exist in a production website, but will give an insightful look at how the system works and how it might be exploited.

The majority of attacks against Joomla historically are Cross-Site Scripting, Injection, Remote Code Execution, Privilege Escalation, and Directory traversal attacks.All these vulnerabilities harm the confidentiality of the different parts of the system. In addition, Injection attacks and privilege escalation can also harm the integrity of different parts of the system.The vulnerabilities found are not specific to this domain only, because they can be foundon any website. The reason is that these vulnerabilities exist by default and people without security knowledge will fall in it every time they develop a website. For example, queries on the DB is vulnerable unless you validate the input of the user. And it is not specific to a certain technology. For example, XSS can happen to any website whether it is written in PHP, NodeJS, Java, etc.Following is a discussion of three of the vulnerabilities mentioned above:

A. Injection Attacks

This type of attack mostly occurs due to the lack of proper data sanitization and input validation. While this vulnerability is present in some of the basic Joomla components, it mostly appears in the supported Joomla extensions (modules). The frequency of SQL Injection attacks had a noticeable increase in 2018, which could be attributed to the release of version 3.9. This could be due to unexpected behavior introduced by the integration of new components, or changes in existing ones.This vulnerability was present in versions 3.2 to 3.4.4 of Joomla as shown in Figure 3. Exploiting it allowed the attacker to possibly gain full control of the system.

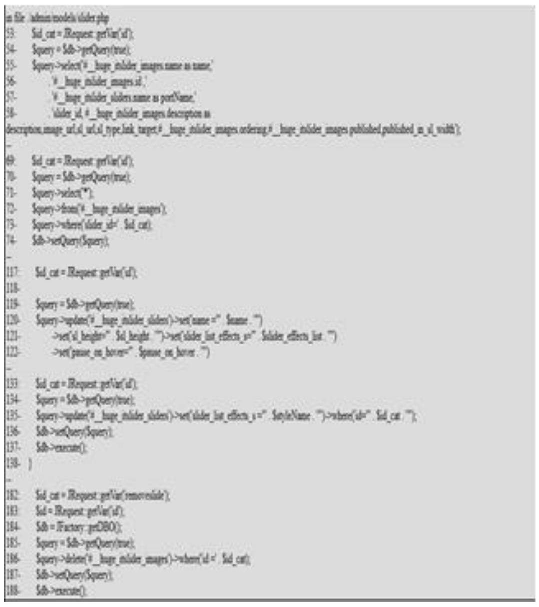


```
protected function getListQuery()
{
    // Create a new query object.
    $db = $this->getDbo();
    $query = $db->getQuery(true);

    // Select the required fields from the table.
    $query->select(
        $this->getState(
            'list.select',
            'h.version_id, h.ucm_item_id, h.ucm_type_id, h.version_note, h.save_date, h.editor_user_id,' .
            'h.character_count, h.sha1_hash, h.version_data, h.keep_forever'
        )
    )
```

Figure 3. A snippet of the area of code where vulnerability to this attack was found

Further information regarding the above exploit can be found here.This vulnerability was also found in the slider extension as shown in Figure 4.



```
in file: admin/models/slider.php
     $id_cat = JRequest::getVar('id');
     $query = $db->getQuery(true);
     $query->select('#__huge_itslider_images.name as name,'
          '#__huge_itslider_images.id ,'
          '#__huge_itslider_sliders.name as portName,'
          'slider_id, #__huge_itslider_images.description as
description,image_url,sl_url,sl_type,link_target,#__huge_itslider_images.ordering,#__huge_itslider_images.published,published_in_sl_width');
-
     $id_cat = JRequest::getVar('id');
     $query = $db->getQuery(true);
     $query->select('*');
     $query->from('#__huge_itslider_images');
     $query->where('slider_id=' . $id_cat);
     $db->setQuery($query);
-
     $id_cat = JRequest::getVar('id');

     $query = $db->getQuery(true);
     $query->update('#__huge_itslider_sliders')->set('name ="' . $name . '"')
          ->set('sl_height="' . $sl_height . '"')->set('slider_list_effects_s="' . $slider_effects_list . '"')
          ->set('pause_on_hover="' . $pause_on_hover . '"')
-
     $id_cat = JRequest::getVar('id');
     $query = $db->getQuery(true);
     $query->update('#__huge_itslider_sliders')->set('slider_list_effects_s ="' . $styleName . '"')->where('id="' . $id_cat . '"');
     $db->setQuery($query);
     $db->execute();
 }
-
     $id_cat = JRequest::getVar('removeslide');
     $id = JRequest::getVar('id');
     $db = JFactory::getDBO();
     $query = $db->getQuery(true);
     $query->delete('#__huge_itslider_images')->where('id =' . $id_cat);
     $db->setQuery($query);
     $db->execute();
```

Figure 4. A snippet of Slider's code where an SQL injection attack could occur

B. Directory Traversal Attacks

These types of attacks mostly occur due to the lack of proper data sanitization. Performing these attacks could grant the attacker access restricted areas of the server to step out of the root directory and access other parts of the file system. This type of attack mostly occurred pre-2010, and was at the time one of the most frequently executed, although it has since seen a decline in frequency. This could be attributed to the introduction of better data sanitization techniques with the newer versions of Joomla. This vulnerability was present in the jqueryform component, which lacked the proper data sanitization techniques needed to prevent this attack.

C. Privilege Escalation:

This vulnerability has been deemed as a critical one in Joomla 3.44 through 3.6.3. This snippet is a module where it creates an arbitrary account with administrative privileges, after that, an email is sent to activate the attacker's account. By exploiting this loophole, an attacker can have administrative privileges and upload a backdoor to control

The website completely and it may evolve to gain full access to the hosting server. All in all, a high number of vulnerabilities in Joomla are introduced through modules added to the CMS. This poses a huge security risk for two reasons. First, developers are not paying attention to how their own modules are reacting with other modules. Second, users who are not tech savvy can easily add vulnerable modules to their systems without knowing the risks. Threat model Joomla as shown in Figure 5.

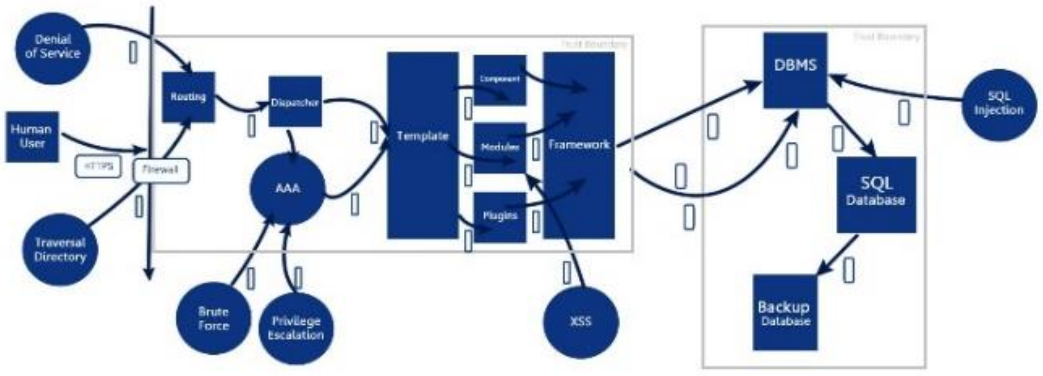


Figure 5. Threat model Joomla

3.2.4 Threat Model

Client-side can access the system using the HTTPS protocol: the firewall first will intercept the client-side request based on its rules.The threat model is shown in Figure 5. If it is valid, the request will make its way to the system. Otherwise, the request will be rejected. Using the right SSL version, this could prevent sniffing attacks. After the request gets through the firewall, it will be routed based on the user request. If the user wants to access parts that need authentication, the routing will take the user request into the authentication part of AAA. If the user is authenticated, the request will go to the specific destination if the user has the right authorization that will be determined based on the privileges given to them. Even after authentication, the user is not allowed to access parts of the system that they are not authorized to access. Furthermore, all user actions are to be logged to the database, and a backup snapshot of the system is saved in the event that reverts to a more stable version if needed.

The first defensive layer of the system is the firewall. If the attacker was able to bypass the firewall, another defensive layer will come into place, which is the routing and AAA. The router will not route the user to the specified area of the system unless they are authenticated and authorized to access that area of the system. At this point, there are two possible attacks to these defenses, DoS attacks,and directory traversal attacks. A DoS attack would overload the resources of the system, causing the routing mechanism to fail, and consequently destroying the dispatcher and the AAA mechanisms. This enables the attacker to access areas of the system without needing to be authenticated or authorized.

However, the firewall has a feature to circumvent that, called the Bad Bots Banned. This feature senses user requests and is triggered once a large number of malicious requests are received, banning these requests, subsequently mitigating most of the effects of the DoS attacks. Another possible attack on the routing and AAA mechanisms is the directory traversal attacks. This attack allows the attacker to access files and folders in the parent directories which they should not normally be allowed to access, revealing possibly sensitive data. This attack can be mitigated by configuring the AAA mechanism properly and by adding restrictive code using the .htaccess file inside the Joomla framework. One other way an attacker could gain unauthorized access to the system is by using brute force attacks to assume the identity of a superuser in the system. This is mitigated by three main features implemented in Joomla, which are Bad Bots Banned, Block Failed Login. Bad Bots Banned will stop bots from brute-forcing their way into the system by detecting automated, large-volume login attempts and stopping them, while Block Failed Login will reject all login attempts from the attacker’s IP address when a certain number of failed logins is reached. Furthermore, in Joomla, advanced CAPTCHA mechanisms are used to ensure real human users are making legitimate requests and prevent any type of brute force attacks.

Another way to have access to sensitive information of the system or perform malicious actions, the attacker might try to get higher privileges to do that. To mitigate that, we have three mechanisms which are: separate admins login from normal users, performing a health check, update feature. Separate admins login from normal users mean that we have different logins for users and admins. Performing health check is a feature from the firewall where it notifies you if you downloaded a plugin with a vulnerability that might make your website vulnerable in addition to that, it will alerts you if some hidden code is found such as a backdoor. Therefore, there is a backup mechanism where it tells you whether you have the latest version of Joomla or not, because the old versions of any application have many vulnerabilities that are taken care of in the newer ones.

One component that might be vulnerable to attack is the module component. The attacker might use an XSS attack in order to execute undesirable instructions on the server side, or even at the user's side.

MySQL database that Joomla uses by default is vulnerable to SQL injection attack. The attacker might use SQL injection in order to get data that he is not supposed to have access to, and in the worst case, he might inject or delete data. To mitigate this, the system is using the backup to rollback to the correct version if the attacker somehow was able to tamper with the data. In addition, we are able to activate the two-factor authentication method in Joomla. For example, a secret key will be sent to the user device additionally to his credentials to log in. Thus, even the attacker could reach to the user information, he will not be able to log in because the secret key is missing. Moreover, the firewall can detect and block SQL attempts when attackers try to inject them. Another attack might be from the insiders rather than outsiders. For example, an admin can use his privileges in order to perform actions that are harmful to the system. To mitigate this, the system has a logging mechanism that records every action performed on the system.

3.3. Flask

3.3.1 Introduction

Flask is a micro web framework written in python. It is classified as a micro-framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.However, Flask support extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program. Flask is commonly used with MongoDB which allows it more control over databases and history.

In 2004, the pocco team was formedasan international group of enthusiasts from the Python community working on open source Python software without commercial interest and for the benefit of others. Flask was created by Armin Ronacher of Pocoo. Armin Ronacher is an Austrian open source software programmer and the creator of the Flask web framework for Python.

3.3.2 Architecture

Flask is based on two subsystems, Werkzeug WSGI toolkit,and Jinja2 template engine. Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

Werkzeug It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. Figure 6 shows how the working of the layers of the flask.

On the other hand, Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages. Flask comes with no batteries included approach (very light and depends on extensions), so all subsystems exist in extensions that are not a part of Flask itself.

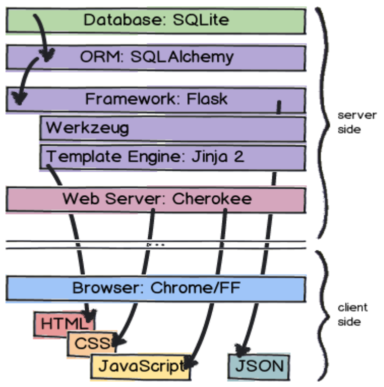
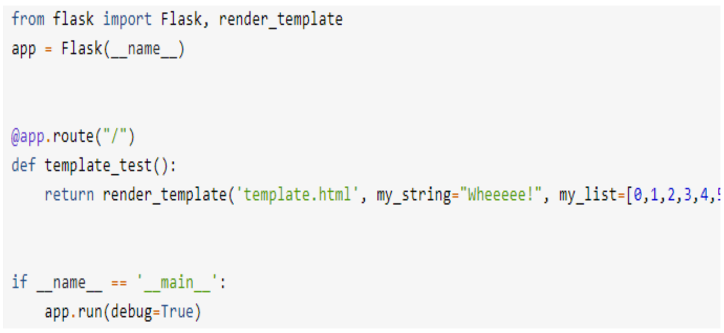


Figure 6. This figure shows how the layers of flask work together in any web application

One of the biggest advantage that WSGI gives us flexibility. You can actually change the web stack components without changing the codes at all, and without even changing the application that runs the WSGI servers. This means that your application on flask can be configured with any server (like apache) without changing the code each time you shift from server to another.

WSGI servers promote scaling. Serving thousands of requests for dynamic content at once is the domain of WSGI servers, not frameworks. WSGI servers handle processing requests from the web server and deciding how to communicate those requests to an application framework's process. The segregation of responsibilities is important for efficiently scaling web traffic and also reduces the chances of DDoS attack.

The developer can make a mistake in the configuration WSGI file that runs flask on the web server, and the result of any mistake in this file can prevent the application from running on the server. Jinja2 is a template engine that transforms your code in python to HTML files and handles the interaction between them, for example, by looking at the code shown in Figure 7. Misspelling ‘template.html’ or the names of the variables(my_string) or even forgetting to import render_template module will prevent your page from loading. Jinja2 is more sensitive and exposed to vulnerabilities since it’s a linking between the python code and the HTML files, and errors can happen because of misspelling or changing the name of HTML file. Any mistakes happen in Jinja2 or WSGI code can prevent users from accessing your website, which is an availability issue. Causing problems in Jinja2 code will open debug mode of the flask(which must be disabled in deployment) and make you easy execute python commands to the web application, which is a confidentiality and integrity issue.In Flask 0.10 and lower, jsonify() did not serialize top-level arrays to JSON as shown in Figure 7. This was because of a security vulnerability in ECMAScript 4. ECMAScript 5 closed this vulnerability, so only extremely old browsers are still vulnerable. All of these browsers have other more serious vulnerabilities, so this behavior was changed and jsonify() now supports serializing arrays.



```
from flask import Flask, render_template
app = Flask(__name__)


@app.route("/")
def template_test():
    return render_template('template.html', my_string="Wheeeee!", my_list=[0,1,2,3,4,


if __name__ == '__main__':
    app.run(debug=True)
```

Figure 7. Flask vulnerability

A. Dealing with Cookies

If your authentication information is stored in cookies, you have implicit state management. The state of “being logged in” is controlled by a cookie, and that cookie is sent with each request to a page. Unfortunately, that includes requests triggered by 3rd party sites. If you don’t keep that in mind, some people might be able to trick your application’s users with social engineering to do stupid things without them knowing. Say you have a specific URL that when you sent POST requests to will delete a user’s profile (say http://example.com/user/delete). If an attacker now creates a page that sends a post request to that page with some JavaScript they just have to trick some users to load that page and their profiles will end up being deleted. Imagine you were to run Facebook with millions of concurrent users and someone would send out links to images of little kittens. When users would go to that page, their profiles would get deleted while they are looking at images of fluffy cats. How can you prevent that? Basically, for each request that modifies content on the server, you would have to either use a one-time token and store that in the cookie and also transmit it with the form data. After receiving the data on the server again, you would then have to compare the two tokens and ensure they are equal. Why does Flask not do that for you? The ideal place for this to happen is the form validation framework, which does not exist in Flask. So consider how architecture might change over time.

Flask architecture can change be applying one of the following decisions:

1) Including one of the important extensions that many developers use to be one of the main subsystems in a flask rather than dealing with externally.
2) Changing its main components like dealing with another template engine rather than Jinja2

We need to consider that Flask the main goal from using flask is less configuration for beginners and more customization for experts, which means that the architecture of flask will not change a lot as long it remains a micro framework.

3.3.3 Security Vulnerabilities

Web applications normally face all types of security issues and it's extremely difficult to get everything proper. Flask attempts to resolve some of these issues for you, but there are issues that need to be taken care of. Cross-site scripting is the thought of inserting willful HTML (and with it JavaScript) inside the context of a website. To solve this, developers have to perfectly avoid text so that it cannot include willful HTML tags.Flask configures Jinja2 to automatically dismiss all values unless clearly told otherwise. This must rule out all XSS issues brought in templates, but there are yet other areas where you have to be watchful:

a) Creating HTML without the aid of Jinja2.
b) Calling Markup on data submitted by users.
c) Sending out HTML from uploaded files, never do that, use the Content-Disposition: attachment header to prevent that problem.
d) Sending out text files from uploaded files. Some browsers are using content-type guessing based on the first few bytes so users could trick a browser to execute HTML.

Another thing that is very crucial is unquoted attributes. While Jinja2 can defend you from XSS problems by escaping HTML, there is one thing it cannot protect you from XSS by attribute injection. To counter this potential attack vector, be sure to constantly quote the attributes with either double or single quotes when using Jinja expressions in them:

```
<input value="{{ value }}">
```

Why is this mandatory? Because if you would not be doing that, an attacker could simply inject custom JavaScript handlers. For example an attacker could inject this piece of HTML+JavaScript:

```
onmouseover=alert(document.cookie)
```

When the user would then flow with the mouse over the input, the cookie would be given to the user in an alert window. But rather of displaying the cookie to the user, a good attacker might also perform any other JavaScript code. In combination with CSS injections, the attacker might even make the element fill out the entire page so that the user would just have to have the mouse anywhere on the page to trigger the attack. There is one class of XSS issues that Jinja's escaping does not defend against. The tag's href attribute can contain a javascript: URI, which the browser will execute when clicked if not secured properly.

```
<a href="{{ value }}">click here</a>
<a href="javascript:alert('unsafe');">click here</a>
```

To prevent this, you'll need to set the Content Security Policy (CSP) response header. HTTP Strict Transport Security (HSTS) Tells the browser to change all HTTP requests to HTTPS, preventing man-in-the-middle (MITM) attacks.

```
response.headers['Strict-Transport-Security'] = 'max-age=31536000; includeSubDomains'
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

Flask leverages the Jinga2 engine. It's easy to use and is configured out-of-the-box to auto-escape content in .html, .htm, .xml, and .xhtml files. Flask allows for the creation of templates using strings of HTML in the Python source code or laid out in static files in a templates directory local to your project.

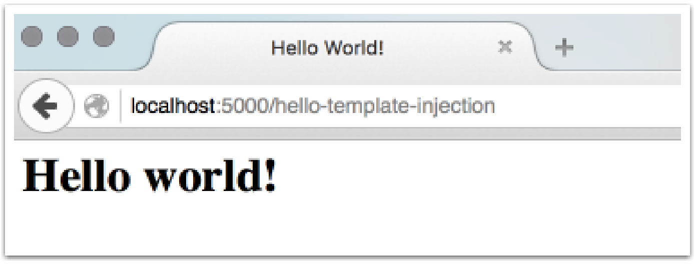
A. Server-Side Template Injection

The template engine provided within the Flask framework may allow developers to introduce Server-Side Template Injection vulnerabilities. This vulnerability allows an attacker to inject language/syntax into templates. Execution of this input occurs within the context of the server. Figures 8, 9 and 10 SQL injection attack scenario. Depending on the context of the application this could allow for arbitrary remote code execution (RCE) Let's take a look at using the template string functionality to explore some security concerns. Consider the following snippet of code:

```
from flask import Flask, request, render_template_string, render_template
app = Flask(__name__)
@app.route('/hello-template-injection')
defhello_ssti():
        person={'name':"world", 'secret':"UGhldmJoZj8gYWl2ZnZoei5wYnovcG5lcnJlZg=="}
        if request.args.get('name'):
        person['name'] = request.args.get('name')
template = '''<h2>Hello %s!</h2>''' % person['name']
returnrender_template_string(template, person=person)
####
# Private function if the user has local files.
###
defget_user_file(f_name):
        with open(f_name) as f:
                returnf.readlines()

app.jinja_env.globals['get_user_file'] = get_user_file # Allows for use in Jinja2 templates
if __name__ == "__main__":
        app.run(debug=True)
```

Trying out the application:

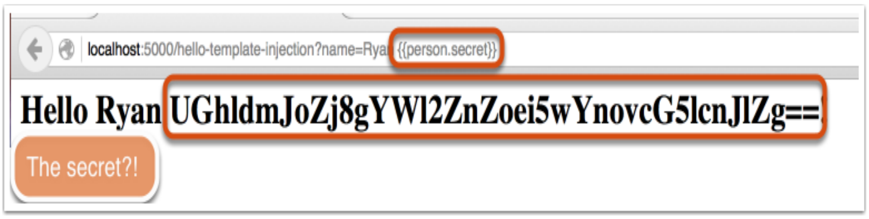

Hello World!

localhost:5000/hello-template-injection

Hello world!

Figure 8. SQL injection input

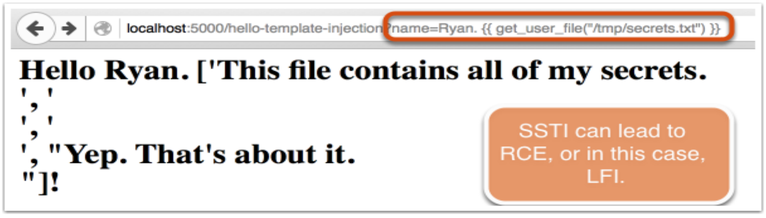
When the Input is given Ryan. The following output is achieved as shown in Figure 9


localhost:5000/hello-template-injection?name=Rya{{person.secret}}

Hello Ryan UGhldmJoZj8gYWl2ZnZoei5wYnovcG5lcnJlZg==

Figure 9. Output SQL injection

Let's try the Input Ryan again the following output as shown in Figure 10 is achieved


localhost:5000/hello-template-injection?name=Ryan. {{ get_user_file("/tmp/secrets.txt") }}

Hello Ryan. ['This file contains all of my secrets.
', '
', "Yep. That's about it.
"]!

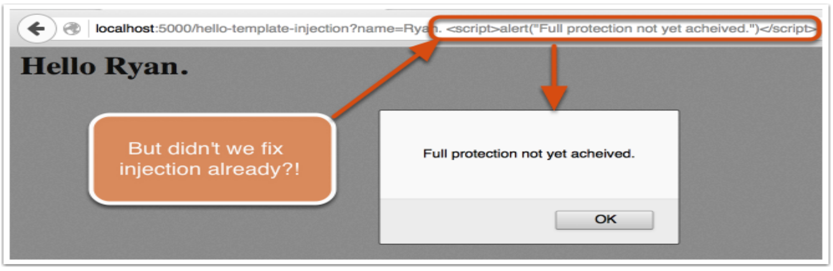
Figure 10. Output SQL injection second scenario

B. Cross-Site Scripting

As stated above, Flask provides an auto-escape feature on certain file types. While this is excellent there are some caveats:

1) Templates can disable this feature
2) Template strings and non-common file extensions do not enable auto-escape by default

Template Strings auto-escaped? To fix the issue, we can pipe our output through the manual escape filter to ensure proper output escaping before reflection to the user. Figure 11 shows the output of a XSS attack.


localhost:5000/hello-template-injection?name=Ryan. <script>alert("Full protection not yet acheived.")</script>

Hello Ryan.

Full protection not yet acheived.

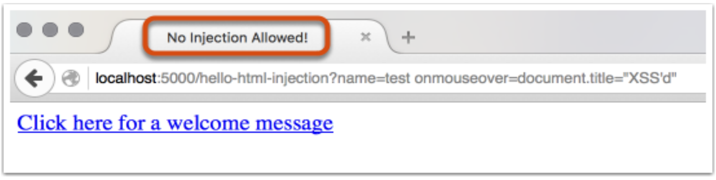
Figure 11. Output XSS attack

C. Remarks about Escaping Output in Flask

The escaping function doesn’t protect against HTML attribute injection. Using the following code as an example:

```
defhello_hi():
template = '''<title>No Injection Allowed!</title>
<a href=?name=>
Click here for a welcome message</a>'''
          name = "world"
          if request.args.get('name'):
          name = request.args.get('name')
returnrender_template_string(template, name=name)
```

Surround our variable with `` and using the filter to manually escape output. Figure. 12 shows the output with no injection allowed onmouseover.


No Injection Allowed!

localhost:5000/hello-html-injection?name=test onmouseover=document.title='XSS'd'

Click here for a welcome message

Figure 12. Onmouseover no injection allowed

3.3.4 Threat Model

After analyzing the situation, A spoofing of source data audit files can lead to incorrect data delivered to the database.

Figure.13 shows the threat model of the flask. So considering a standard authentication mechanism to identify the destination data store will be a good option. Also,an XSS attack would happen in the browser. Another one occurs where “Cookie Based Session” and “Custom-Cookie Based Session” can also be spoofed by an attacker which may lead to data being written by the attacker towards the attacker’s target also data being sent to the attacker’s target instead of the real user session.

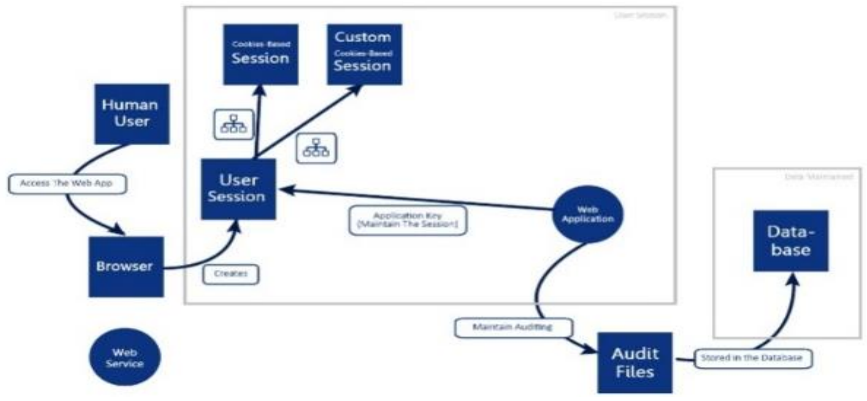


Figure 13. Threat model Flask

Also, an external agent may interrupt the application key held by the web application when it is coming out of the trustboundary toward the user session in order to verify its integrity. And it is also applied for the auditing data that comes out from the web application to the database (tampering is taking the place).
Human User: the user is always a vulnerability as humans are always prone to social engineering attacks and XSS attacks if they have low awareness ofcybersecurity
Browser: Browsers are outside the trust boundary and they are prone to XSS attacks and can be a target to malicious attacks, awareness must be provided to users about possible dangers that could arise from this, and a very good solution is to limit the user on using certain browsers that are up to the developer's standards on security.
Application key: the application key can be hijacked by an attacker when it comes out of the trust boundary and when it is hijacked the whole session could be vulnerable to attack making it a priority in terms of what should be secured
User Session: the user session must be handled correctly although it is inside the trust boundary its security depends on aspects that come from outside the trust boundary and if an attacker manages to trick the system into believing that they are legitimate user, this is as far as they need to go as by this point they can manage to access most of the application's assets.
Cookies: in a cookie-based session, cookies are an important asset, and they are stored outside the trust boundary if targeted by an attacker and if the application key is stored in the cookies, an attacker can access the application key, cookies are also vulnerable to injection attacks and spoofing.
Audits: data from audits usually is stored in files before it is sent to the database if accessed before written to the database an attacker can attack the integrity of the application by pushing wrong data to audit before it is stored in the database.
Database: the database can be secured in many ways, we can encrypt data going into it, and encrypt data that are stored in it, we can add a firewall for its access, but due to the fact that files stored in it may come from files that were at one point stored in the users device, it is vulnerable to false data being injected to it.

Are there locations in the architecture where no assets reside? Did you miss any assets there? Yes, Front End assets and Management Commands are missing. We haven't considered the end-user as an asset in the earlier phases of the project, but after considering the attacks that the end user is vulnerable to we recognized how much of an important asset they are. Outside the trust boundary in the side of the end-user. The attacker may attack & may target many assets, they can target the browser and perform cross-site scripts and they can attack the cookies and possibly the application key where they would be able to cause massive harm to the security of our applications.

3.4. VLC Media Player

3.4.1 Introduction

VLC is a free and open source media player and streaming media server developed by the video LAN project. It is available for the desktop operating system and mobile platforms. With VLC it is possible to watch all different types of videos using any platform with the smallest amount of resources. The project started as a student project at the French Ecole Centrale Paris in 1996. Currently, the team contains a hundred developers. In the beginning, VLC was designed to be a client that streams from satellite dishes across campus networks. Around 2001 it was redesigned from scratch to be a media player hence the name was changed to the VLC media player. Main priorities include running as many video formats on any type of platforms maintaining simple GUI and low memory requirements.

3.4.2 Architecture

VLC systems consist of a combination of two architectures layered and modular. Modular architecture is shown in Figure 14.

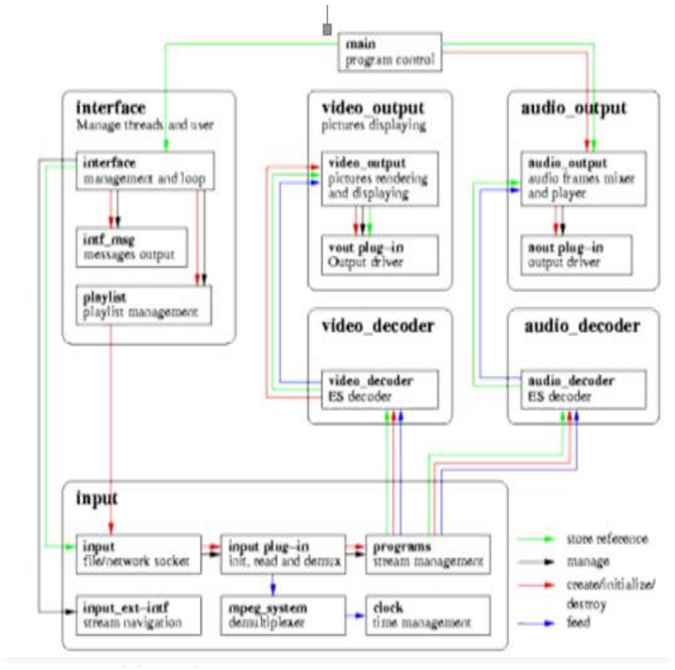


Figure 14. Modular architecture

VLC has a core subsystem which contains all the features and it is called libVLCore. This subsystem is a library and contains other subsystems which are actual features they are interface, playlist, video output, audio output,and other miscellaneous utilities. VLC also haslibVLC on top of LibVLCcore which allow external application builders to access all features of the core.

3.4.3 Security Vulnerabilities

Using the STRIDEmodel we analyze the threats to VLC media player as follows:

a) Spoofing: It is not a threat to the software
b) Tampering: It is a threat
c) Repudiation: It is not a threat to the software
d) Information disclosure: It is a threat
e) Denial of service: It is a threat
f) Elevation of privilege: It is not a threat to the software

Since integrity and confidentiality are compromised as discussed above tampering and information disclosure are possible threats to the system besides DOS attacks which compromises the availability feature. As there is no signing in feature which can be used in spoofing or elevation of privilege repudiation which can be done as an attacker can have total control in the device when performing the attacks as discussed before. Software vulnerabilities exist in the streaming server and online features of the VLC media player. DOS and code execution on VLC have boomed.

3.4.4 Threat Model

The VLC media player is not affected by other programs when installing in the system. By looking at the threat model it can be seen that libVLCore is a subsystem which is an integral part of VLC and it is the cause of many functionalities which system provides as a whole. Figure. 15 shows the threat model of the VLC media player.

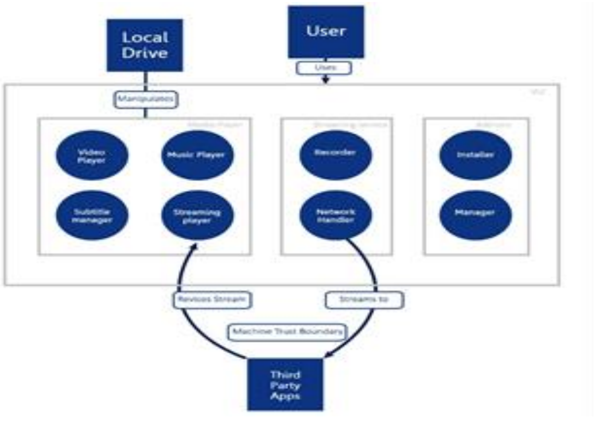


Figure 15. Threat model VLC media player

4. RESULT

4.1. Moodle Code Inspection and Solutions

In this section, we discuss our findings of the code inspection activity that we perform over moodle.

1) /course/delete:

No one should have the permission to delete the front page,that would be considered a "superuser" privilege. By giving anyone that ability you are opening yourself to risks that you don't have a dealwith.Deal with it the cavemanway, remove the ability to delete the front page from anyone who has it, if you want to delete the front page you will have to delete the file manually.

a) If the user doesn't have the permission to delete a course, it should be dealt with then and there, there is no point going forward. Printing an error message alone won't suffice. Consider handling the lack of permission and wrapping everything in an else clause.

Checking the hash for confirmation is good, adds another layer of security. As it is, you are checking if you have confirmation even when the user doesn't have permission to delete a page – exposing yourself to risks that you shouldn't have to deal with. This gives more credence to the point raised above.

a) Again - in the event that the hash won't match, you are not dealing with it. You are allowing the user to go along – consider including the navbar updates in this if clause, and adding an else clause where you deal with the event where the user doesn't get confirmation,either by redirecting the user to another page or updating the current one to reflect the fact that he/she got denied.

After inspecting this source code, we were expecting to see it have multiple layers of security which it did (authentication and hashing) but it wasn't handling unauthorized users well, it just showed them an error message that they are not allowed to perform this and then let them continue. Where they should have had some sort of exception handlings such as try-catch or any other method.

2) login/confirm.php:https://github.com/moodle/moodle/blob/master/login/confirm.php

a) If the authentication plugin is not enabled, then there is no point going forwards, it only serves to expose you to risks you don't have to deal with, consider redirecting the user to a generic page if you decide to handle the exception.

b) You don't need to check if $data is not empty twice since you have an OR operator in your if condition, consider checking if $data is not empty first, then checking $p AND $s are not empty after via an else if clause

c) If the user is confirmed successfully via an authentication plugin but cannot be found, it's a good indication that foul play is at work here. Printing an error message will not suffice, consider invoking re-authentication at this point.

After inspecting this source code, we have found that Moodle has not handled the exception when it occurs, and only printing an error message while in fact, they should try using a try-catch or any other exception handling method. We also found an error where once the user is confirmed successfully but cannot be found, the system just prints an error message while it should invoke re-authentication. We have also found an algorithm mistake where it checks if $data is empty twice, this reduces the system's performance.

4.2. Joomla Code Inspection and Solutions

By inspecting the file structure of Joomla We can observe that it is divided into general files containing utilities and user components, and administrator utilities and components. The modules which were chosen for inspection are as follows:

1) Login administrator component: This component is quite essential and crucial since it allows access to the entire administrator dashboard which in turn allows access to all other components and controls of the system
2) User's administrator component: This component allows the management of users, roles,and privileges, which makes it a crucial component in the system. If this component was to be compromised, an attacker can gain admin privileges and thus access to all the system.
3) Content component: This component is an essential component in the whole system of Joomla! Since it is responsible for displaying content to users visiting the website. A security threat arises from its compromise because an attacker can use it to launch all kinds of injection and cross-site attacks.

We believe our selection is good since all modules selected have direct security functionalities and have quite a big impact if compromised. Selecting files to inspect in Joomla systems is not difficult since the system is somewhat documented, in addition to its clear file structure which makes the process of finding security-related modules and files a bit easier.

The general design of Joomla is heavily using the factory pattern. In addition to the use of MVC architecture. Furthermore, the design is decomposed to:

a) General Application Class (CMSApplication): which is a class that has multiple general functionalities regarding the whole Joomla! application, among which is the login function and logout function. In addition, input to the application is accessed through an Input object which is contained within this class.
b) Model classes (JModelLegacy): accessing models is done through Model objects. Model objects have predefined strings to manage which model is being accessed. For example, if the login model is needed for access it can be accessed by calling the appropriate function as shown shortly and specifying the name of the model.
c) View class (JViewLegacy): which manages how all views are displayed to users and how they appear in the specific template chosen by the admin.

4.2.1 Subpackage Login Administrator Component

This subpackage handles login and logout requests, the creation,and termination of user sessions, and the creation of and sending tokens to users. Mistakes in this part of the code could make it easy for attackers to assume the identities of legitimate users, allowing them to gain access to and retrieve or tamper with user data and information which they are otherwise not authorized to access. For example, they can create articles, modify or delete them, modify their profiles and many other things.

Main files inspected within the component:

1) models/login.php: this file is responsible to set the credentials and load login modules regardless of the access level. Also, the Login page is accessible to the public.
   a) populateState(): This function is used to set username, password and secret key. A secret key is used if two-factor authentication was implemented. If the secret key is exposed, this would reduce the defense in depth mechanism because the hackers now only need to know the password.
2) views/login/view.html.php: it is a view for login component
   a) display(): this function tries to prevent clickjacking vulnerability where it implements x-frame-options. It can frame any other website to deceive normal users. If this x-frame-options: SameOrigin does not exist, this would allow an attacker to use the Joomla! the site as his frame to deceive other users.
3) login/controller.php: This file is responsible for implementing logout and login requests. For instance, when the user clicks logout, it will be executed.
   a) logout(): it is responsible for resetting the session with the user. If it doesn't reset the session after the user closes the window or after a certain period of time, and if the computer was shared with other people, the attacker can go back and get to the same session and get unauthorized access to his account.
   b) login(): allows the user to login to the system. If the input validation is not implemented correctly, this could allow an attacker to use SQL injection to get into the system.

4.2.2 Subpackage Users Administrator Component

This subpackage handles the authorization of users with regards to which system views they are allowed access to, as well as handling user emails, notes, and other user-related information. It also allows admins to create new groups of users with different privileges and levels of authority. Developers' mistakes in this code would allow attackers to elevate their own privileges, possibly elevating themselves into admins. It also allows them to have access to views and information that they are not supposed to access.

Main files inspected within the component:

1) Controllers/user.php: this file is responsible for defining a set of activities that a user can do in his account.

Allowedit(): this function will check if a person is a super admin or not. However, the superuser should not exist from in the first place but here it exists. Moreover, superuser existence makes the system vulnerable to privilege escalation attack which gives him the full authority to do anything in the system and thus attack the system.

2) Controllers/users.php: it is a dashboard for admins to control the operations that can be done on users records.

Changeblock(): this function is used to change the block status on a user record. It checks the token first and based on that it determines what operation it should apply. If an attacker could get the token or the token is not securely randomly generated, it may allow them to add or remove users

3) Controllers/groups.php: This file is responsible for controlling the user groups lists to do some operations on it.

Checkin(): This function locks out users from editing articles that other users are already in the process of editing. It checks if the user has admin permission; which, if the user acquires in some way, could allow them to do unauthorized actions, such as editing articles which they are locked out of editing.

4) Helpers/users.php: this file is responsible to handle actions users can do such as making two-factorauthentication, viewing access levels and others.

Gettwofactormethods(): it enables two-factor authentication mechanism to apply defense in depth mechanism. If an uneducated user disables this function, it will facilitate the process to intruders to get into the system.

5) Models/users.php: this file is responsible for performing operations on the database for user-related activities.

Getitems(): Allows the user to get a list of users based on permission. However, the critical part here is that Joomla! clears the cache first before requesting to view the users. So, here if an attacker were to perform cache poisoning, it will display all the users with their personal information.

6) Views/user/tmpl/edit.php: a form where it receives input from the user to redirect it to the appropriate handler for it.

It includes a form where users can enter information, so if they use get method instead of post, this will be less secure as get shows sensitive information in the URL. In addition, a token hidden input is established. If the token is not randomly secure, so the attacker may guess the pattern and this would support his intrusion process.

4.2.3 Subpackage Content Component

This component handles the retrieval of Joomla web content which is mainly articles and renders content in a view appropriate for the user privilege level. For example, a content writer can view content with the option to edit or create new content, while a normal user would only be able to view the content. This component is part of the components subsystem and affects the posting asset in the system.

Main files inspected within the component:

a) Content.php:
   a) Responsible for authorizing users and determining if they are allowed to edit content or not (view edit options or not).

b) Controller.php:
   b) Responsible for receiving specific article information through a request, make sure of user privileges of this article and configure all view options for the content to be displayed.

All of the previously mentioned models represent a type of content and the code functionality is similar in terms of executing queries to fetch the request content type from the database.Mistakes in this module are quite impactful since there are many mechanisms and authentication operations performed at different files to ensure only users with appropriate access level can view content with appropriate options.

4.3. Flask Code Inspection and Solution

The process begins by adding a tests directory under the application root. Then create a Python file to store our tests (test_flaskr.py). When formatting the filename like test_*.py, it will be auto-discoverable by pytest. Next, we create a pytest fixture called client() that configures the application for testing and initializes a new database.:

```
import os
import tempfile
import pytest
```

```
from flaskr import flaskr
@pytest.fixture
def client():
db_fd, flaskr.app.config['DATABASE'] = tempfile.mkstemp()
flaskr.app.config['TESTING'] = True
        client = flaskr.app.test_client()
        with flaskr.app.app_context():
flaskr.init_db()
        yield client
os.close(db_fd)
os.unlink(flaskr.app.config['DATABASE'])
```

This client fixture will be called by each individual test. It gives us a simple interface to the application, where we can trigger test requests to the application. The client will also keep track of cookies for us.During setup, the TESTINGconfig flag is activated. What this does is disable error catching during request handling, so that you will get better error reports when performing test requests against the application.

Because SQLite3 is filesystem-based, we can easily use the tempfile module to create a temporary database and initialize it. The mkstemp() function does two things for us: it returns a low-level file handle and a random file name, the latter we use asthe database name. We just have to keep the db_fd around so that we can use the os.close() function to close the file.

4.4. Basic Test

Now it’s time to start testing the functionality of the application. Let’s check that the application shows “No entries here so far” if we access the root of the application (/). To do this, we add a new test function to test_flaskr.py, like this:

```
deftest_empty_db(client):
        """Start with a blank database."""

rv = client.get('/')
        assert b'No entries here so far' in rv.data
```

Notice that the test functions begin with the word test; this allows pytestto automatically identify the function as a test to run.

By using the client.getwe can send an HTTP GET request to the application with the given path. The return value will be a response_class object. We can now use the data attribute to inspect the return value (as string) from the application. In this case, we ensure that 'No entries here so far' is part of the output.

Run it again and you should see one passing test:

```
$ pytest -v

================ test session starts ================
rootdir: ./flask/examples/flaskr, inifile: setup.cfg
collected 1 item

tests/test_flaskr.py::test_empty_db PASSED

============= 1 passed in 0.10 seconds ==============
```

4.4.1 Other Testing Tricks

Besides using the test client,as shown above, there is also the test_request_context() method that can be used in combination with the witha statement to activate a request context temporarily. With this, you can access the request, g and session objects like in view functions. Here is a full example that demonstrates this approach:

```
import flask

app = flask.Flask(__name__)

with app.test_request_context('/?name=Peter'):
        assert flask.request.path == '/'
assert flask.request.args['name'] == 'Peter'
```

All the other objects that are context-bound can be used in the same way.

If you want to test the application with different configurations and there does not seem to be a good way to do that, consider switching to application factories (see Application Factories). Note however that if you are using a test request context, the before_request() and after_request() functions are not called automatically. However teardown_request() functions are indeed executed when the test request context leaves thewiththe block. If you do want the before_request() functions to be called as well, you need to call preprocess_request() yourself:

```
app = flask.Flask(__name__)

with app.test_request_context('/?name=Peter'):
app.preprocess_request()
```

This can be necessary to open database connections or something similar depending on how your application was designed.

If you want to call the after_request() functions you need to call into process_response() which however requires that you pass it a response object:

```
app = flask.Flask(__name__)
with app.test_request_context('/?name=Peter'):
resp = Response('...')
resp = app.process_response(resp)
```

This, in general, is less useful because at that point you can directly start using the test client.Code Inspection Results:

The code tests a basic blog application called Flaskr. Users will be able to register, log in, create posts, and edit or delete their own posts. The code uses the module tempfile to create a temporary DB file on the system. It also uses the os module to even have the ability from the beginning to deal with files system (create, update, delete files on OS), and pytest, which is the testing module. Eventually, we import Flaskr module to be able to deal with our application and test it.

Next, we create pytest fixture, which providesa fixed baseline upon which tests can reliably and repeatedly execute. The fixture called client(), which configures the application for testing and initializes a new database. The line starts with db_fd creates DB file in TEMP folder and returns a tuple containing an OS-level handle to an open file (as would be returned by os.open()) and the absolute pathname of that file, in that order.

The assets affected by the above code is:

a) OS file system
b) Application (Flaskr) DB file
c) Environment Variables, which is considered as assets since it usually contains sensitive info about the application like the secret key (read more about it from here), and also some API credentials.

The code follows the blueprints subsystem, where you initialize your application’s views (in the MVC sense). But what if a developer made a mistake in this module, the kinds of security risks could arise will be like integrity problems, where the DB file could no longer be available, the environment variables could be changed, and If the secret key is exposed this could lead to a confidentiality problem, where unauthorized people can manipulate deep details of the code.

There is no exception handling in the code above, dealing with OS module should be always within trying and except blocks, because any error will happen while creating or deleting the DB file can cause the module to throw an error that is not been handled, which will make the whole app crashes.

4.5. VLC Media Player Code Inspection and Solution

We inspected three modules in VLC media player which are MKV, MMS (Microsoft Media Server) and Codec (it is a type of program which understands a type of video or audio). These modules are prone to vulnerabilities. In the MKVmodule an attacker can execute arbitrary code via crafted MKV files. The codec is vulnerable to execute arbitrary code by an attacker from the files the hacker uses to make VLC convert which executes code overflow attack. VLC is vulnerable to DOS attack. MMS stream contains a vulnerability which can be triggered at any time.

5. DISCUSSION

In this section, we will discuss a summary of our security analysis of different open source systems discussed in this paper.

5.1. Moodle

To identify the potential threats we used the stride model to classify the threats facing the learning management system. A spoofing attack is possible as someone can impersonate as a teacher to gain access to the materials. Tampering is possible as some can modify the material by gaining access. It is possible to repudiate in moodle meaning any actor performing an action should own it as well. It is possible for a user to gain access to the material of the course which he is not enrolled which leads to information disclosure. Denial of service is certainly possible on Moodle. Privilege escalation is possible as students gaining privileges edit grades.

Code inspection was performed over moodle. Three pages were inspected /course/delete, /login/confirm.php and login/change_password.php. After code inspection, multiple layers of security were discovered which were authentication and hashing. It was found that it was not dealing with unauthorized users well and displaying error message where it could have been handled with try-catch exception handling.

5.2. Joomla

There exist security vulnerabilities in the Joomla framework. Most common security threat faced by Joomla is that of compromising confidentiality. As resources which can be compromised are usually found in the database. Majority of security vulnerabilities on Joomla are cross-site scripting, directory traversal attacks, injection, privilege escalation,and remote code execution. Vulnerability and integrity are affected by these security vulnerabilities.

Code inspection was performed over Joomla. The file structure consists of containing utilities and user components, and administrator utilities and components. We choose the login administrator component, Users administrator component,and content component. The design of Joomla is based on the factory pattern. In the subpackage login administrator component the files inspected are models/login.php, views/login/view.html.php, login/controller.php. In subpackageusers administrator component the files are inspected are controllers/user.php, controllers/users.php, controllers/groups.php, helpers/users.php, helpers/users.php, models/users.php, views/user/tmpl/edit.php. The main file inspected in subpackage content management iscontent.php, controller.php. Potential vulnerabilities found in content.php is that authorization is done by user input with a hardcodedstring which is vulnerable in terms of security. This could lead to access to unauthorized users.

5.3. Flask

Potential threats found in the flask framework is cross-site scripting (XSS). Stride model is used to identify threats on flask by using XSS. Scripts can be injected in a flask that can perform malicious activities within the framework. Hence, spoofing is possible using XSS. Tampering is possible by using XSS in the flask. All of the forms of tampering using XSS is possible in the flask. With XSS vulnerability it is possible to launch denial of service attacks on the flask. Template engine within flask allows attackers to introduce service side template injection vulnerabilities. SQL injection attacks are demonstrated which could possibly allow remote code execution (RCE).

Code inspection of the flask is performed by tests scripts, which are in a directory folder. Security risk that can arise is that of integrity.

5.4. VLC Media Player

In order to identify threats of VLC media player, we used stride model. Tampering, information disclosure and denial of service are identified as threats to the VLC media player. Security vulnerabilities lie in the streaming server and online features of VLC. Three modules were inspected in VLC media player which are MKV, MMS,and Codec. In MKV it is possible to execute code via crafter MKV files. The codec is vulnerable to execute arbitrary code by the attacker. VLC is identified to be vulnerable to DOS attack.

REFERENCES

[1] Ban, Xinbo, Shigang Liu, Chao Chen, and Caslon Chua."A performance evaluation of deep-learnt features for software vulnerability detection," *Concurrency and Computation: Practice and Experience*, e5103, 2019.

[2] Adewumi, Adewole, Sanjay Misra, Nicholas Omoregbe, and Luis Fernandez Sanz. "FOSSES: Framework for open source software evaluation and selection," *Software: Practice and Experience* vol. 49(5), pp. 780-812, 2019.

[3] Nunes, P., Medeiros, I., Fonseca, J., Neves, N., Correia, M. and Vieira, M., 2019. "An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios," Computing, 101(2), pp.161-185.

[4] Zahedi, Mansooreh, Muhammad Ali Babar, and Christoph Treude. "*An empirical study of security issues posted in open source projects*," In Proceedings of the 51st Hawaii International Conference on System Sciences. 2018.

[5] Pashchenko, Ivan, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. "*Vulnerable open source dependencies: Counting those that matter*," In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 42. ACM, 2018.

[6] Hauge, Øyvind, Claudia Ayala, and Reidar Conradi. "Adoption of open source software in software-intensive organizations-A systematic literature review," *Information and Software Technology,* vol. 52(11) pp. 1133-1154 ,2010.

[7] Stol, Klaas-Jan, and Muhammad Ali Babar. "*Reporting empirical research in open source software: the state of practice*," IFIP International Conference on Open Source Systems. Springer, Berlin, Heidelberg, 2009.

[8] Feller, Joseph, et al. "Developing open source software: a community-based analysis of research," *Social Inclusion: Societal and Organizational Implications for Information Systems*. Springer, Boston, MA, pp. 261-278, 2006.

[9] Scacchi, Walt, et al. "Understanding free/open source software development processes," *Software Process: Improvement and Practice*, vol. 11(2), pp. 95-105, 2006.

[10] Crowston, Kevin, et al. "Free/Libre open-source software development: What we know and what we do not know," *ACM Computing Surveys (CSUR)*, vol. 44(2), pp. 7, 2012.

[11] Von Krogh, Georg, and Eric Von Hippel. "The promise of research on open source software," *Management science*, vol. 52(7), pp. 975-983, 2006.

[12] Pistoia, Marco, et al. "A survey of static analysis methods for identifying security vulnerabilities in software systems," *IBM Systems Journal* vol. 46(2), pp. 265-288, 2007.

[13] Alhazmi, Omar H., and Yashwant K. Malaiya. "Modeling the vulnerability discovery process," *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*. IEEE, 2005.

[14] Al-Ghamdi, A. S. A. M. "A survey on software security testing techniques," *Int J Comput Sci Telecommun 4*, pp. 14-18, 2013.

[15] Jovanovic, Nenad, Christopher Kruegel, and Engin Kirda. "Pixy: A static analysis tool for detecting web application vulnerabilities, " (short paper). *IEEE*, 2006.

[16] Bau, Jason, et al. "State of the art: Automated black-box web application vulnerability testing," *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010.

[17] Huang, Yao-Wen, et al. "*Web application security assessment by fault injection and behavior monitoring*," Proceedings of the 12th international conference on World Wide Web. ACM, 2003.

[18] Abunadi, Ibrahim, and Mamdouh Alenezi. "An empirical investigation of security vulnerabilities within web applications." *J. UCS* vol. 22(4), pp. 537-551, 2016.

[19] Alenezi, Mamdouh. "Software architecture quality measurement stability and understandability," *International Journal of Advanced Computer Science and Applications (IJACSA)* vol. 7(7), pp. 550-559, 2016.

[20] Alenezi, Mamdouh, and Khaled Almustafa. "Empirical analysis of the complexity evolution in open-source software systems," *International Journal of Hybrid Information Technology* vol. 8(2), pp. 257-266, 2015.

[21] Alenezi, Mamdouh, and Yasir Javed. "Developer Companion: A Framework to Produce Secure Web Applications," *International Journal of Computer Science and Information Security* vol. 14(7), pp. 12, 2016.

[22] Dougiamas, Martin, and Peter Taylor. "Moodle: Using learning communities to create an open source course management system," *EdMedia: World Conference on Educational Media and Technology. Association for the Advancement of Computing in Education (AACE)*, 2003.

[23] Patel, Savan K., V. R[illegible] Rathod, and Satyen Parikh. "Joomla, Drupal and WordPress-a statistical comparison of open source CMS," *Trendz in Information Sciences and Computing (TISC), 2011 3rd International Conference on. IEEE*, 2011.

[24] Barr, J., "The Flask Security Architecture," *Comput. Sci*, no.6, pp. 574.

[25] Jiao, Dongliang, et al. "Research on security policy and framework," *The Second International Symposium on Networking and Network Security*. 2010.

[26] Kaluža, Marin, Bernard Vukelić, and Tamara Rojko. "Content Management System Security," *Zbornik Veleučilišta u Rijeci*, vol. 4(1), pp. 29-44, 2016.

[27] Rahmel, Dan. "Joomla security administration," Advanced Joomla!. Apress, Berkeley, CA, pp. 159-183, 2013.

[28] Barhoom, Tawfiq S., and Rola J. Azaiza. "Enhance MOODLE security against XSS vulnerabilities," *International Journal of Computing and Digital Systems*, vol. 5(5), 2016.

[29] Futoransky, Ariel, et al. "Establishing and enforcing security and privacy policies in web-based applications." U.S. Patent No. 7,831,995. 9 Nov. 2010.

[30] Thiel, David. "*Exposing vulnerabilities in media software*," Black Hat conference presentation, BlackHat EU. 2008.

[31] Cole, Jason, and Helen Foster, "Using Moodle: Teaching with the popular open source course management system," *O'Reilly Media, Inc.*", 2007.

[32] [32] Cole, Jason, and Helen Foster. Using Moodle: Teaching with the popular open source course management system. " O'Reilly Media, Inc.", 2007.