University of Moratuwa

Department of Computer Science and Engineering

CS 4202 - Research and Development Project

Project Report

PACOM - Payment Application COmpliance Monitor

Group- 01

Supervisors

Dr. H M N Dilum Bandara **(UOM)**
Mr. Rohana Kumara **(Leapset)**

Group Members

D. S. M. Senarath	**110518F**
N. R. Kasthuriarachchi	**110295P**
P. D. D. H. Anicitus	**110033M**

THIS REPORT IS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE OF ENGINEERING AT UNIVERSITY OF MORATUWA, SRI LANKA.

Feburary 3, 2016

Declaration

We, the project group 01 (D. S. M. Senarath, N. R. Kasthuriarachchi, and P. D. D. H. Anicitus under the supervision of Dr. H.M.N. Dilum Bandara, and Mr. Rohana Kumara) hereby declare that except where specified reference is made to the work of others, the project "PACOM - Payment Application COmpliance Monitor" is our own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgement.

Signatures of the candidates:

1. .....D. S. M. Senarath (110518F)

2. .....N. R. Kasthuriarachchi (110295P)

3. .....P. D. D. H. Anicitus (11033M)

Supervisor:

..... (Signature and Date)

Dr. H.M.N. Dilum Bandara

Project Coordinator:

..... (Signature and Date)

Dr. Malaka Walpola

Abstract

Project Title: Schema-Independent Scientific Data Cataloging Framework

Authors: D. S. M. Senarath (110518F)

N. R. Kasthuriarachchi (110295P)

P. D. D. H. Anicitus (11033M)

Internal Supervisor: Dr. H. M. N. Dilum Bandara

External Supervisors: Mr. Rohana Kumara

Every organization that handles credit card information needs to comply with the Payment Card Industry Data Security Standards (PCI DSS). Vendors that make and sell payment applications need to meet PA DSS (Payment Application Data Security Standards). These standards are very demanding, and only 18% of all the asse applications pass the compliancy in the first year. Moreover, only 11% has managed to maintain the compliancy between annual assessments. It is being identified that there are a number of pitfalls that programmers run into which lead to non-compliant code. Hence, it is imperative to address these issues during the software development life cycle than in the end product.

We propose a Payment Application Compliance Monitor (PACoM) that monitors the code changes to identify non-compliances in the code. The solution is based on SonarQube, a static code analysis platform which is convenient to integrate into an existing development workflow. We develop a set of custom rules to detect these pitfalls, including secure garbage collection practices, one of the most crucial standards and logging of credit card information. Then those pitfalls were categorized into code analysis models and are implemented as a rule. Finally those rules are integrated into the SonarQube platform as a new plugin. Moreover, a widget is added to make it easy to access and view the issues in a centralized place. SonarQube makes the PACoM available for number of platforms. Those platforms includes version control systems such as Git, SVN, continuous integration platforms such as Travis CI, and Integrated Development Environments such as Eclipse, IntelliJ. Having the ability to integrate into almost every phase of the development PACoM

will give the high control and feedback on the churn to the developers. The solution was developed in collaboration with Leapset Pvt Ltd.

Acknowledgement

First and foremost we would like to express our sincere gratitude to our project supervisor, Dr. H.M.N. Dilum Bandara for the valuable guidance and dedicated involvement at every step throughout the process.

We would also like to thank our external supervisor Mr. Rohana Kumara of Leapset for the valuable advice and the direction given to us regarding the project.

We would like to thank Mr. Nalinda Herath of TechCert in giving us the guidance and advice from the point of view of a Qualified Security Assessor.

We would like to express our warm gratitude to Dr. Malaka Walpola for coordinating the final year projects.

Last but not least, we would like to express our greatest gratitude to the Department of Computer Science and Engineering, University of Moratuwa for providing the support for us to successfully finish the project.

Table of Contents

List of figures

List of tables

1 INTRODUCTION

1.1 Motivation

Payment Card Industry consists of all the organizations which handle various forms of payment cards (e.g., debit, credit, prepaid, ATM, and POS cards) and associated cardholder data. To maintain security across all channels, every organization that store, process, and transmit payment cards information needs to comply with the Payment Card Industry Data Security Standard (PCI DSS) [1].

Leapset (pvt) Ltd. is software development company that develops a full-suite of applications for restaurant management, which includes a Point of Sales (POS) component. The POS system is able to process credit card transactions, which makes the Leapset's system in scope of PCI DSS. o maintain the compliance status it is mandatory for the Leapset to adopt PCI DSS guidelines to their Software Development Life Cycle. Maintaining PCI DSS in the development phase is a task that requires lot of manual work in terms of rigorous code reviewing with attention to PCI DSS guidelines. This is one of the main struggles that Leapset faces every day.

Checking for PCI DSS in a live system consists of regular network scans, maintaining an Intrusion Detection System, File monitoring system, anti-virus program, updated system programs, etc. To check most of these Leapset uses a set of open source applications written by various developers.

Leapset wants to come up with a full package to monitor PCI DSS in both the implemented environments in customer sites and development environment at Leapset. As tools are already available to check compliancy, it was proposed to implement a pluggable framework to monitor PCI DSS which can integrate aforementioned tools.

It was found that there is already available framework that fits the requirement namely OSSIM [2] (discussed in following chapter).

OSSIM only considers about the deployed product. It is very useful tool as an incident reporter with respect to PCI DSS compliance. It can auto generate PCI DSS compliance related documents within the application to support the certification process.

OSSIM does not contains a major component that Leapset requires as a software development company. This is the ability to check the compliancy of the application throughout the development process and to help the developers to achieve PCI DSS compliancy requirements within the development phase.

Contribution

In addressing this requirement it is proposed to develop a code analysing tool that can check the application at the code level for non-compliances with PCI DSS. This tool should be able to be integrated easily to the development workflow.

Thus it is desirable to implement a sonarqube plugin which contains rules to detect the non compliancies. And also sonarqube has the ability to fit right into the development workflow as it has the ability to be plugged into popular IDEs and development tools such as version control systems and continuous integration processes.

Although there are only limited number of standards in the PA DSS guidelines, there are number of different scenarios that a developer can introduce a non-compliant code to the code base. Identifying these scenarios is a critical step in this project. Then to put them into analyzable models to come up with rules to be implemented in the Sonarqube plugin.

Outline

This report is organized as follows. Chapter 2 discusses the existing literature which is relevant to the project. Chapter 3 presents the problem statement. Design of the system and its architecture are presented in Chapter 4. Chapter 5 presents the implementation details of the project including the tools and technologies, system components, testing, and security controls. Chapter 8 will conclude the report with problems encountered, challenges, and future work.

2 LITERATURE REVIEW

2.1 Introduction

This chapter provides a comprehensive review of the related work on payment application compliance monitoring. Furthermore, it contains the knowledge and techniques to achieve the objectives of our project.

At the initial stage, this project is aimed at researching the requirement of PCI DSS and PA DSS compliance in a merchant location and using the appropriate technology to automate the task of identifying any breaches of the compliance in continuous basis throughout the lifecycle of the development process.

PCI DSS and PA DSS compliances cannot be fully automated as it involves development process, policy implementation and upgrades, manual penetration tests, coding practices, etc., but there are majority of tasks that could be automated. We follow our initial research on requirements of the PCI DSS and PA DSS compliances and identify feasibility of automation of these requirements.

There are quite a few tools already available to maintain some functionality of compliance. In the second stage of the research, we study the implementation of such tools as each tool is different in what is maintained. This literature report contain brief analysis about each tool and its architecture.

One major requirement of our project is to create a tool to identify compliance breaches as early as possible. The second part of this analysis contain the research about the static and dynamic code analysis techniques and implementation of such a tool to support developer to develop PCI DSS and PA DSS applications.

2.2 PCI DSS

The Payment Card Industry Data Security Standard (PCI DSS) [1] was developed to encourage and enhance cardholder data security and facilitate the adoption of consistent

data security measures globally. PCI DSS provides a baseline of technical and operational requirements designed to protect account data. PCI DSS applies to all entities involved in payment card processing including merchants, processors, acquirers, issuers, and service providers. PCI DSS also applies to all other entities that store, process or transmit cardholder data and sensitive authentication data. The table 1 is a high-level overview of the 12 PCI DSS requirements

Build and Maintain a Secure Network and Systems	1. Install and maintain a firewall configuration to protect cardholder data
	2. Do not use vendor-supplied defaults for system password and other security parameters
Protect Cardholder Data	3. Protect stored cardholder data
	4. Encrypt transmission of cardholder data across open, public networks
Maintain a Vulnerability Management Program	5. Protect all systems against malware and regularly update anti-virus software or programs
	6. Develop and maintain secure systems and applications.
Implement Strong Access Control Measures	7. Restrict access to cardholder data by business need to know
	8. Identify and authenticate access to system components
	9. Restrict physical access to cardholder data

Regularly Monitor and Test Networks	10. Track and monitor all access to network resources and cardholder data
	11. Regularly test security systems and processes
Maintain an Information Security Policy	12. Maintain a policy that addresses information security for all personnel

Table 1 Overview of the PCI DSS standards

2.3 PA DSS

Payment Application Data Security Standard (PA DSS [2]) is a PCI Security Standard Council managed program for the Payment Applications and applies to software vendors and others who develop payment applications that store, process, or transmit cardholder data as part of authorization or settlement. PA DSS is a guideline for software vendors to develop secure payment applications according to the PCI DSS. Applications satisfying PA DSS can be securely implemented in a PCI DSS compliant environment without breaching the PCI DSS] standards.

PA DSS is the primary guideline followed by software vendors when developing payment related applications. This standard is consisted of the following main guidelines;

2.3.1 Do not retain full track data, card verification code or value or PIN block data

After authorization, do not store the full contents of any track from the magnetic stripe located on the back of a card, equivalent data contained on a chip, or elsewhere. These data includes the verification value or code (three-digit or four-digit number printed on the front or back of a payment card) used to verify card-not-present transactions, pin and pin block. Upon the finished transaction securely delete any track data (from the magnetic stripe or equivalent data contained on a chip), card verification values or codes, and PINs or PIN block data stored by previous versions of the payment application, in accordance with industry-accepted standards for secure deletion, as defined, for example by the list of

approved products maintained by the National Security Agency, or by other State or National standards or regulations.

If any sensitive authentication data (pre-authorization data) must be used for debugging or troubleshooting purposes, ensure the following:

- Sensitive authentication data is collected only when needed to solve a specific problem.
- Such data is stored in a specific, known location with limited access.
- The minimum amount of data is collected as needed to solve a specific problem.
- Sensitive authentication data is encrypted with strong cryptography while stored.
- Data is securely deleted immediately after use, including from:
    - Log files
    - Debugging files
    - Other locations

2.3.2 Protect stored cardholder data

Mask PAN when displayed (the first six and last four digits are the maximum number of digits to be displayed), such that only personnel with a legitimate business need can see the full PAN. Render PAN unreadable anywhere it is stored (including data on portable digital media, backup media, and in logs) by using any of the following approaches:

- One-way hashes based on strong cryptography (hash must be of the entire PAN)
- Truncation (hashing cannot be used to replace the truncated segment of PAN)
- Index tokens and pads (pads must be securely stored)
- Strong cryptography with associated key management processes and procedures.

Strong cryptographic keys must be generated, distributed, discarded and managed securely. Provide a mechanism to render irretrievable any cryptographic key material or cryptogram stored by the payment application, in accordance with industry-accepted standards.

2.3.3 Provide secure authentication features

The payment application must support and enforce the use of unique user IDs and secure authentication for all administrative access and for all access to cardholder data. Secure authentication must be enforced to all accounts generated or managed by the application by the completion of installation and for subsequent changes after installation.

The application must enforce the changing of all default application passwords for all accounts that are generated or managed by the application, by the completion of installation and for subsequent changes after installation.

The payment application requires that passwords meet the following:

- Require a minimum length of at least seven characters.
- Contain both numeric and alphabetic characters.

Alternatively, the passwords/phrase must have complexity and strength at least equivalent to the parameters specified. A strong, one-way cryptographic algorithm, based on approved standards must be used to render all payment application passwords unreadable during storage. Repeated attempt policy must be imposed. Administrative accounts should have password expiring period of 90 days. Application idle time is 30 minutes, then the account must be logged in again.

2.3.4 Log payment application activity

The application must provide the ability to have central logging mechanism. At the completion of the installation process, the “out of the box” default installation of the payment application must log all user access and be able to link all activities to individual users.

Payment application must provide automated audit trails to reconstruct the following events:

- All individual user accesses to cardholder data from the application.

- All actions taken by any individual with administrative privileges as assigned in the application.
- Access to application audit trails managed by or within the application.
- Invalid login access attempts.
- Changes to the application's identification and authentication mechanisms and all changes, additions, deletions to application accounts
- Initialization, stopping, or pausing of the application audit logs.

2.3.5 Develop secure payment applications

The software vendor must have a defined and implemented a formal process for secure development of payment application. Payment applications must be developed in accordance with PCI DSS and PA-DSS for example, secure authentication and logging. Development processes must be based on industry standards and best practices. Information security should be incorporated throughout the software development life cycle. Most importantly security reviews are performed prior to release of an application or application update.

Testing should be done without the use of live PAN (Primary Account Number). The test data and account used for testing should be removed before releasing the application. Payment application should be code reviewed before any update or release to the customer. Code changes should be reviewed by individuals other than the originating code author, and by individuals who are knowledgeable in code-review techniques and secure coding practices. Code changes are reviewed by individuals other than the originating code author, and by individuals who are knowledgeable in code-review techniques and secure coding practices.

Secure source-control practices are implemented to verify integrity of source code during the development process.

Coding techniques should be included in documentation of how PAN and/or SAD are handled in memory. Attackers can use malware tools to capture sensitive data from memory. Minimizing the exposure of PAN/SAD while in memory will help reduce the

likelihood that it can be captured by a malicious user or be unknowingly saved to disk in a memory file and left unprotected. This requirement is intended to ensure that consideration is given for how PAN and SAD are handled in memory. Understanding when and for how long sensitive data is present in memory, as well as in what format, will help application vendors to identify potential insecurities in their applications.

Develop all payment applications to prevent common coding vulnerabilities in software development processes. Injection flaws, particularly SQL injection, OS Command Injection, LDAP and XPath injection flaws as well as other injection flaws. Application should be checked for buffer overflow, insecure cryptographic storage, improper error handling, cross site scripting (XSS), cross site request forgery (CSRF), improper access control such as insecure direct object references, failure to restrict URL access, and directory traversal.

2.3.6 Protect wireless transmissions

For payment applications using wireless technology, change wireless vendor defaults, including but not limited to default wireless encryption keys, passwords, and SNMP community strings. The wireless technology must be implemented securely. Payment application must facilitate use of industry best practices (for example, IEEE 802.11i) to implement strong encryption for authentication and transmission.

2.3.7 Test payment applications to address vulnerabilities and maintain updates

Software vendors must establish a process to identify and manage vulnerabilities. Identify new security vulnerabilities using reputable sources for obtaining security vulnerability information. Payment applications and updates should be tested for the presence of vulnerabilities prior to release. Software vendors must establish a process for timely development and deployment of security patches and upgrades. Patches and updates are delivered to customers in a secure manner with a known chain of trust and updates are to be delivered to customers in a manner that maintains the integrity of the patch and update code.

2.3.8 Facilitate secure network implementation

The payment application must be able to be implemented into a secure network environment. Application must not interfere with use of devices, applications, or configurations required for PCI DSS compliance. The payment application must only use necessary and secure services, protocols, daemons, components, and dependent software and hardware, including those provided by third parties, for any functionality of the payment application.

2.3.9 Cardholder data must never be stored on a server connected to the Internet

The payment application must be developed such that any web server and any cardholder data storage component are not required to be on the same server, nor is the data storage component required to be on the same network zone with the web server. If payment application updates are delivered via remote access into customers' systems, software vendors must tell customers to turn on remote-access technologies only when needed for downloads from vendor, and to turn off immediately after download completes. Alternatively, if delivered via virtual private network (VPN) or other high-speed connection, software vendors must advise customers to properly configure a firewall or a personal firewall product to secure "always-on" connections.

2.3.10 Facilitate secure remote access to payment application

The payment application must be able to be implemented into a secure network environment. Application must not interfere with use of devices, applications, or configurations required for PCI DSS compliance. The payment application must only use or require use of necessary and secure services, protocols, daemons, components, and dependent software and hardware, including those provided by third parties, for any functionality of the payment application. The payment application must not require use of services or protocols that preclude the use of or interfere with normal operation of two-factor authentication technologies for securing remote access to the payment application that originates from outside the customer environment.

2.3.11 Encrypt sensitive traffic over public networks

If the payment application sends, or facilitates sending, cardholder data over public networks, the payment application must support use of strong cryptography and security protocols (SSL/TLS IPSEC, SSH, etc.) to safeguard sensitive cardholder data during transmission over open, public networks. Two-factor authentication must be used for all remote access to the payment application that originates from outside the customer environment.

If the payment application facilitates sending of PANs by end-user messaging technologies (e-mail, instant messaging, chat), the payment application must provide a solution that renders the PAN unreadable or implements strong cryptography, or specify use of strong cryptography to encrypt the PANs.

2.3.12 Encrypt all non-console administrative access

Non- console administrative access, encrypt all such access with strong cryptography using technologies such as SSH, VPN, or SSL/TLS, for web-based management and other non-console administrative access.

2.4 Tools for PCI DSS Compliance Management

Several commercial and open source tools are available to manage fraction of PCI DSS requirements. This will be discussed in following sections.

2.4.1 OSSEC

OSSEC [4] is a full platform to monitor and control software product systems. OSSEC can be integrated with the production servers and development servers. It mixes together all the aspects of host-based intrusion detection, log monitoring and SIM/SIEM together in a simple, powerful and open source solution.

2.4.1.1 Key Features

2.4.1.1.1 File Integrity checking

There is one thing in common to any attack to networks and computers: they change systems in some way. The goal of file integrity checking (file integrity monitoring) is to detect these changes and alert when they happen. It can be an attack, or a misuse by an employee or even a typo by an admin, any file, directory or registry change will be alerted. This section covers PCI DSS sections 11.5 and 10.5.5.

2.4.1.1.2 Log Monitoring

Every operating system, application, and device on network generate logs (events) to track the actions of users and applications. Major requirement of PCI DSS compliance is protecting cardholder data. Payment application should not logged any payment related sensitive data. OSSEC collects, analyses and correlates these logs to identify information leakages (sensitive data) and also it can identify if something wrong is going on (attack, misuse, errors, etc.). This log monitoring covers PCI DSS section 10 in a whole.

2.4.1.1.3 Rootkit detection

Criminals (also known as hackers) want to hide their actions, but using rootkit detection it can be notified when they (or Trojans, viruses, etc.) change system in this way.

2.4.1.2 Key Benefits

2.4.1.2.1 Compliance Requirements

OSSEC helps customers meet specific compliance requirements such as PCI, HIPAA etc. It lets customers detect and alert on unauthorized file system modifications and malicious behaviour embedded in the log files of COTS products as well as custom applications. For PCI, it covers the sections of file integrity monitoring, log inspection and monitoring and policy enforcement/checking.

2.4.1.2.2 Multi-platform

OSSEC lets customers implement a comprehensive host based intrusion detection system with fine grained application/server specific policies across multiple platforms such as Linux, Solaris, AIX, HP-UX, BSD, Windows, Mac and VMware ESX.

2.4.1.2.3 Real-time and Configurable Alerts

OSSEC lets customers configure incidents they want to be alerted on which lets them focus on raising the priority of critical incidents over the regular noise on any system. Integration with SMTP, SMS and syslog allows customers to be on top of alerts by sending these on to e-mail and handheld devices such as cell phones and pagers. Active response options to block an attack immediately is also available.

2.4.1.2.4 Integration with current infrastructure

OSSEC will integrate with current investments from customers such as SIM/SEM (Security Incident Management/Security Events Management) products for centralized reporting and correlation of events.

2.4.1.2.5 Centralized management

OSSEC provides a simplified centralized management server to manage policies across multiple operating systems. Additionally, it also lets customers define server specific overrides for finer grained policies.

2.4.1.2.6 Agent and agentless monitoring

OSSEC offers the flexibility of agent based and agentless monitoring of systems and networking components such as routers and firewalls. It lets customers who have restrictions on software being installed on systems (such as FDA approved systems or appliances) meet security and compliance needs.

2.4.1.3 How It Works

OSSEC is composed of multiple pieces. It has a central manager monitoring everything and receiving information from agents, syslog, databases and from agentless devices.

2.4.1.3.1 Manager

The manager is the central piece of the OSSEC deployment. It stores the file integrity checking databases, the logs, events and system auditing entries. All the rules, decoders and major configuration options are stored centrally in the manager, making easy to administer even a large number of agents.

2.4.1.3.2 Agents

The agent is a small program installed on the systems to monitor. It will collect information on real time and forward to the manager for analysis and correlation. It has a very small memory and CPU footprint by default, not affecting with the system's usage.

It runs with a low privilege user (created during the installation) and inside a chroot jail isolated from the system. Most of the agent configuration is pushed from the manager, with just some of them are stored locally on each agent. In case these local options are changed, the manager will receive the information and will generate an alert.

2.4.1.3.3 Agentless

For systems that users can't install an agent, OSSEC allows to perform file integrity monitoring on them without the agent installed. It can be very useful to monitor firewalls, routers and even UNIX systems where users are not allowed to install the agent.

2.4.1.3.4 Virtualization/VMware

OSSEC allows users to install the agent on the guest operating systems or inside the host (VMware ESX). With the agent installed inside the VMware ESX users can get alerts about when a VM guest is being installed, removed, started, etc. It also monitors logins, logouts and errors inside the ESX server. In addition to that, OSSEC performs the CIS checks for VMware, alerting if there is any insecure configuration option enabled or any other issue.

2.4.1.3.5 Firewalls, switches and routers

OSSEC can receive and analyse syslog events from a large variety of firewalls, switches and routers. It supports all Cisco routers, Cisco PIX, Cisco FWSM, Cisco ASA, and Juniper Routers, Netscreen firewall, Checkpoint and many others.

2.4.1.4 Architecture

This diagram shows the central manager receiving events from the agents and system logs from remote devices. When something is detected, active responses can be executed and the admin is notified.

Figure 1 OSSEC high level architecture [5]

2.4.2 OSSIM

OSSIM [3] stands for Open Source Security Information Management System and compiles more than 15 open source security programs providing all the technology levels to cover the full Security Management cycle.

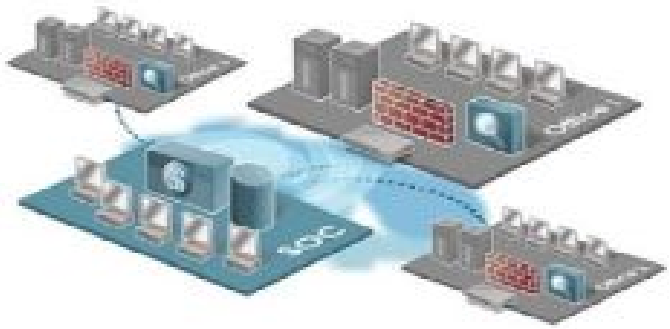


Figure 2 Overview of OSSIM Setup [6]

It makes a complex but powerful system as it adds the capacities of much consolidated security programs and network monitors such as Snort, Nessus, Nagios or Ntop.

OSSIM project has been mainly an integration effort. All the development is focused on integrating the above software and trying to make it work together. For this purpose they have developed a Collector, a Correlation Engine, and several Reporting and Management Tools that allow gathering, normalizing and processing information from a single console.

All this tools together make possible a tight control of big networks deploying low cost sensors and managing the information from a central point. There are already very large networks, with hundreds of sensors, deployed in telecom, financial or governmental organizations.

2.4.2.1 Key Features

The OSSIM - Open Source Security Information Management platform provides five essential security capabilities by integrating many proven open source security soft wares into OSSIM platform. It providing functionalities to manage both compliance and threats. These key security capabilities are,

Asset Discovery

OSSIM combines active network scanning, passive network monitoring, and asset inventory to find all assets on network before a bad actor does.

Behavioural Monitoring

OSSIM use NetFlow analysis, service availability monitoring, and full packet capture technologies to identify suspicious behaviour and potentially compromised systems.

Vulnerability Assessment

OSSIM servers include network vulnerability testing tools and continuous vulnerability monitoring tools to Identify systems on network that are vulnerable to exploits.

SIEM

Security information and event management technology Correlate and analyse security event data from across network. SIEM include log management, event correlation, incident response, reporting, and alarms.

2.4.2.2 Integrated Tools in OSSIM

Arpwatch [7] – Monitors address resolution protocol (ARP) by logging activity and detecting anomalies. It logs IP/MAC address combinations and notify changes or foul play on the data link layer.

P0f [8] – An effective passive fingerprinting tool to identify OS and software on endpoints and to show how the machine is connected to the Internet (e.g., T1/E1, DSL, etc.) as well as the types of packet filters it is behind. It does this without generating any network traffic, as active fingerprinting tools like DNS lookups, traceroute, or other tools might.

PADS [9] – The Passive Asset Detection System is used for service anomaly detection. For example, PADS and Nmap together are used to detect new network services or changes in existing ones.

OpenVAS [10] – The Open Vulnerability Assessment System is a powerful vulnerability scanning and management application. It is a feature-rich fork of Nessus that is fully GPL.

OCS-NG [11]– Open Computer and Software inventory is an open source asset management application. This cross-platform tool is a powerful way to manage all of assets in one place.

Snort [12] – The powerful Intrusion Detection System/Intrusion Prevention System (IDS/IPS) uses signature-, protocol-, and anomaly-based inspection to give insight into intrusions such as OS fingerprinting or buffer overflows, among others.

Suricata [13] – A network IDS, IPS, and network security monitoring engine, which, as of OSSIM 4.2, is the default IDS used in OSSIM.

Tcptrack [14] – A simple sniffer that allows to monitor network connections and bandwidth on an interface. It details connection state, source and destination addresses, and ports.

Ntop [15] – An effective network visualization application with rich graphical output and statistical output that can serve as a network probe while offering visual web-based insight into network traffic flows.

Nagios [16] – A feature-rich network monitoring application for proactively managing network. This popular network monitoring application keeps an eye on critical services and devices and can notify with alerts as to faults.

OSSEC [4] – A robust cross-platform HID system that offers log analysis, system integrity checking, policy monitoring, rootkit detection, and real-time alerting.

OSVDB [17] – Open Source Vulnerability Database is an independent, open source vulnerability database created by and for the community. OSVDB is integrated into OSSIM directly.

Munin [18] – A powerful network and infrastructure monitoring tool. Not only does it monitor and alert, but it give useful graphs over a web interface to help understand what is happening under the hood on network.

Nfdump [19]**/NfSen** [20] – The nfdump tool helps to collect and process NetFlow data. NetFlow is a network protocol that allows to collect and analyse IP network traffic flows. NfSen is a web-based NetFlow visualization and investigation tool for nfdump.

Fprobe – A libpcap-based tool that collects network traffic data and packages it as NetFlow flows directed at a specified collector.

2.4.2.3 OSSIM Architecture

A typical OSSIM deployment consists of 4 elements:

1. Sensors
2. Management Server
3. Database
4. Frontend

2.4.2.3.1 Sensor

Sensors are deployed in the networks to monitor network activity. OSSIM sensors are usually low level detectors and monitors that passively (they don't affect the traffic) collect data looking for patterns. They usually also host Scanners which can actively (they make connections) look for vulnerabilities in the network. OSSIM sensors also include the OSSIM Agent which receive data from hosts of this network as for example a router or firewall, and communicate and send their events to the parent Management Server.

2.4.2.3.2 Management Server

The Management Server (or Server) usually includes the following components:

- Framework. It's a control daemon that ties some parts together.
- OSSIM Server. It centralizes the information received from the sensors.

They do at least the following functions:

- The main Server tasks as Normalizing, Prioritizing, Collecting, Risk Assessment and Correlating engines
- The maintenance and external tasks, as backups, scheduled backups, online inventory or scanning launching

2.4.2.3.3 Database

The Database stores events and useful information for the management of the system. OSSIM use SQL database to store data.

2.4.2.3.4 Frontend

The Frontend or Console is the visualization application in this case a web frontend. OSSIM components are all standalone modules and can be configured as the administrator would need. All this components could be placed in different hardware separating all this components or putting all of them in one machine.

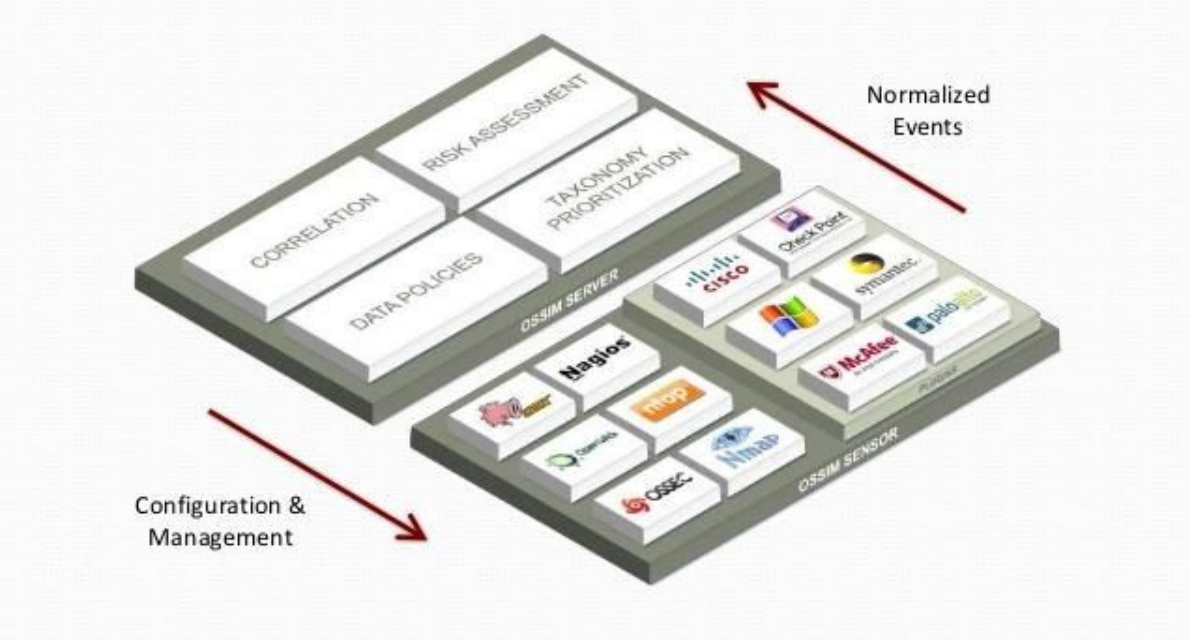


Figure 3 OSSIM Architecture [6]

2.4.3 USM

AlienVault Unified Security Management™ (USM) [21] is an all-in-one platform designed and priced to ensure that mid-market organizations can effectively defend themselves against today's advanced threats. OSSIM is the community open source version of the AlienVault

project, and AlienVault Unified Security Management (USM) offers even more in the way of features, scalability, and support.

USM start at $ 3, 900 and provide all of the feature in OSSIM and more support. Critical differences are seen in capacities such as administration, performance, and reporting. Our major concern in this survey about open source, community driven OSSIM not its commercial product.

2.4.4 Summary of Related Frameworks

OSSEC is an Open Source Host-based Intrusion Detection tool that performs log analysis, file integrity checking, rootkit detection, real-time alerting and active response.

OSSIM is an open source framework which can integrate available tools. It aggregates/correlates security events from multiple tools and presents them in a uniform web console. A few critical open source projects integrated in OSSIM are OSSEC, HIDS, Snort, OpenVas, Nagios, etc.

AlienVault Unified Security Management (USM) is the commercial version of Alien Vault OSSIM project which start at $ 3,900.

	OSSEC	**OSSIM**	**USM**
Open Source	Yes	Yes	No
Cost	0	0	starting from $ 3,900
Extendable	Yes	Yes	Yes
Can integrate Tools	No	Yes	Yes

Coverage of Compliance requirements [22]	10 and 11	1.1, 1.2, 1.3 ,2.1, 2.2, 2.3, 2.3, 3.6, 4.1,5.2, 5.2, 5.3, 6.1, 6.2, 6.3, 6.4, 6.5, 7.1,7.2, 8.1, 8.2, 8.4, 8.5, 8.6, 10, and 11	1.1, 1.2, 1.3 ,2.1, 2.2, 2.3, 2.3, 3.6, 4.1,5.2, 5.2, 5.3, 6.1, 6.2, 6.3, 6.4, 6.5, 7.1,7.2, 8.1, 8.2, 8.4, 8.5, 8.6, 10, and 11
Support Maintenance	Yes	Yes	Yes
Support Development	No	No	No

Table 2 Comparison between OSSEC, OSSIM, and USM

OSSIM system is perfect for maintain PCI DSS at the post production stage of software life cycle. Some breaches of PCI DSS may be identifiable during the coding, testing, regression stages of software life cycle. But OSSIM does not have any such a tool to support developers to maintain PCI DSS and PA DSS compliance in the development and maintenance of the code base of PA DSS application.

Thus we narrowed down the project to support developers to develop PCI DSS and PA DSS application and maintain source code with incremental development.

2.5 Code Analysis

2.5.1 Introduction

After the research project scope was concentrated to a Compliance code monitoring tool, research was turned to a path to create a tool which will analyse the source code and monitor for PCI DSS and PA DSS compliance. Source code should be monitored statically and dynamically in the development phase and the testing phase. If there is a non-compliance in the system it should notified to the developer at the development time. There can be some

non-compliances which are critical and can be detected only in the run time. Such non compliances should notify the admin to fix those non compliancy issues.

As the conclusion a code analyser should be developed to cater the requirements. To develop a code analyser basically a parser, a tree builder, tree analysers, symbol table builders and flow analysers should be created beforehand. But it is not required to reinvent the wheel in the context so we followed the techniques that can be useful for a code analysers, which are open source tools. So the research was focused on static code analysis, dynamic code analysis and how to implement the code monitoring tool.

2.5.2 Static code analysis

Data flow analysis can be performed statically or dynamically aims at detecting the paths of executions that may propagate data from untrusted or sensitive sources to undesirable sinks, by statically analysing the code. If there is such situation it can be identified by analysing the data flow in the source code. Although static analysis exhaustively evaluates a program, it is prone to produce false positives, since it is impossible to establish whether a discovered path will ever be actually executed and hence whether data to be protected will reach the sink at some point. Static code analysis is not the optimum method to check the non-compliancy in the source code.

2.5.3 Dynamic Code analysis

Source code should be monitored in the runtime to detect the non-compliances. Otherwise it will do unnecessary task which leads false positive suggestions. It is not an easy task to perform a dynamic code monitoring because there will be huge load on the server to perform just only dynamic code analysis. Dynamic code analysis tools monitor the execution of a program and track the variable that contains information coming from a given source. Tracking can be performed by means of different approaches, including system emulators [23] [24] , API modification [25] , code injection [26] and ad-hoc hardware [27].

The main disadvantage of dynamic analysis is that the tracking logic introduces an overhead that significantly reduces the performances of monitored programs. So there should be a

mechanism to reduce the load on the server at the run time. Pioneers in the industry also encourages to perform such tasks on Quality Assurance servers only. Additional overhead to the servers will be a huge risk and it is not a good decision to take. At production life cycle no party should monitor the payment card details. So the tool should be run only on development and Quality Assurance servers only.

2.5.4 Hybrid code analysis

There are lot of disadvantages for using static code analysis and the dynamic code analysis. So hybrid code analysis mechanism was introduced to reduce the load on the servers and also to filter the unwanted data which should be monitored in the run time. So the hybrid version will come with lots of advantages and efficient mechanisms.

In hybrid code analysis both static and dynamic methods will be used. The order of execution of instructions within a method of a program can be described by a control Flow Graph. Using these Control Flow Graphs, A method has been introduced to detect problems in the static state and the dynamic state.

There are such researches which were carried out to monitor the source code statically and dynamically both. But there was no such system was developed for PCI DSS or PA DSS. Following are some researches which were carried out to monitor the source codes analysis statically and dynamically.

2.6 Previous Explorations

Chang et al proposed a method [28] that combines static and dynamic analysis for efficiently monitoring C programs. It first performs static analysis to filter out data that need not dynamic tracking, then modifies the C source program to add the appropriate tracking code. The tracker monitors tainted data at byte-level granularity.

Giannone et al. have proposed a system which is similar in spirit to above system. They employ static analysis to minimize the amount of tracking code needed. A major difference is that they have targeted compiled Java byte code in place of C source code, hence making their approach viable when source code is unavailable. Moreover, Java programs are

intrinsically different from C programs and present new challenges, e.g., the management of the operand stack. On the other hand, Java has a more rigorous semantics that simplifies tracking and therefore allows us to greatly reduce the overhead. As an example they had avoid byte-level tracking in favour of a more efficient instruction-based tracking that implicitly tracks data flow at a variable level.

Jee et al. have proposed a technique that can be used in combination with other (Dynamic Data Filtering Tool) DDFT tools in order to reduce redundant tracking logic and thus optimizing the target program. After a DDFT tool has introduced a tracking logic into the binaries of a program, their tool separates the program logic from the tracking logic and then applies known optimization techniques on the tracking logic.

Zhang et al. proposed to optimize DDFT by analysing the documentation and source code of APIs to find taint propagation data. They then associate the input parameters of a function with the corresponding outputs, and determine the APIs that need not be tracked. Their tools can be used for x86 binaries or C programs.

2.7 Code Analysis Techniques

2.7.1 Bug pattern matching

The system can identify bug patterns related to PCI/PA DSS compliance vulnerability in the system. Predefined bug patterns will being identified in this analysis. Typical SQL Injections, Cross site Scripting can be identified using these techniques. Checkstyle is using a similar system to identify bug patterns in the source code.

2.7.2 Data flow analysis

In data flow analysis (DFA), the runtime information about the data in programs is collected. The system will identify the data flow in the runtime and determine whether the credit card details are saved in the variables. Run time dynamic data flow analysis will be operated to detect sensitive data leakage in the system.

2.7.3 Database Query Analysis

All the database queries will analysed to check whether sensitive data is saved in the database. In java domain JDBC Connection is used to connect to the database. So in the java context, through JDBC Connection Database Query can be analysed.

2.7.4 Source Code transformation

If something change to source code which are related to sensitive payment card details those are will be recorded and logged in. All the log details are analysed to check whether payment card details are logged or not.

2.8 Code analysing software

There are lot of code analysing techniques and methods that has been developed for various tasks. CheckStyle, SonarQube and PMD are some of the famous application which are developed to monitor Source Code and check for bugs.

Those various techniques should be gathered together to develop a code monitoring tool for PCI DSS and PA DSS. To build a code monitor basically a parser, a tree builder, tree analysers, symbol table builders and flow analysers will be useful. These building blocks are developed already. So these building blocks should be gathered and handle carefully to build the appropriate code monitoring tool.

2.8.1 Sonarqube

SonarQube is used to check the bugs and errors in the system. It is open source tool which has built in basically parser, tree builder, tree analysers, symbol table builders and flow analysers. We can use these tools to extend the features in the code monitoring tool.

2.8.1.1 SonarQube System analysis

There are three different paradigms for SonarQube analysis: full analysis, preview analysis, and incremental analysis. Source code can be switch among the three modes.

- Analysis - this is the default. This mode analyses everything that's analyse-able for the language in question and saves the results to the database.

- Preview - is typically used to determine whether code changes are good enough to move forward with, e.g., merge into the Git master.
- Incremental - is used on the developer's localhost to examine changed files for added technical debt before checking them in.

2.8.1.2 Analysis Mode

Analysis mode performs a full analysis on the entire code base and saves the results to the database. Assuming code changes, you will ideally analyse once a day in this mode - typically overnight. Use this mode to update the central server and keep the source, PA DSS compliant.

If continuous integration are being used, full analysis should be not assigned to CI job. SonarQube job will be assigned to check the PA DSS compliancy and save the results in the database and send the appropriate notifications to the administrator.

2.8.1.3 Preview Mode

Preview mode performs a full analysis on the entire code base and does not save the results to the database. Typically this mode is used on the continuous integration server as part of the continuous integration job. This can also be used in the developer computer to check whether if there is any non-compliant code segments before push the code to the remote repository.

2.8.1.4 Incremental Mode

During analysis, data is requested from the SonarQube server, the files provided to the analysis are analysed, and the resulting data is sent back to the SonarQube server.

Most of these interactions happen synchronously. However, some updates are saved to the end of analysis, sent to the server in a batch file, and processed asynchronously. Those batch files are queued, and processed sequentially, so it is quite possible that for a brief period after your analysis log shows completion, the updated values are not visible in the Payment Application project.

2.8.2 Checkstyle

2.8.2.1 Introduction

Good programming techniques support to develop a quality software product. To develop a quality software product you need to have coding standards. Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code standards. Checkstyle checks the source code statically. It can find class design problems, method design problems. It also has the ability to check code layout and formatting issues. CheckStyle has Ant task, command line tools and IDE plugins to integrate for source code analysing.

2.8.2.2 Modules

Checkstyle defines a set of available modules, each of which provides rules checking with a configurable level of strictness like mandatory or optional. Each rule can raise notifications, warnings, and errors. For example, checkstyle can examine:

- Javadoc comments for classes, attributes and methods
- Naming conventions of attributes and methods
- Limit of the number of function parameters and line lengths
- Presence of mandatory headers
- The use of packets imports, of classes, of scope modifiers and of instructions blocks
- The spaces between some characters
- The good practices of class construction
- Duplicated code sections
- Multiple complexity measurements, among which expressions.

2.8.2.3 Usage

Checkstyle is built in a JAR file which can run inside a Java VM or as an Apache Ant task. It can also integrate into an IDE or other tools. CheckStyle provides following features:

- Overload syntax colouring or decorations in code editor

- Decorate the project explorer to highlight problem-posing resources
- Add warnings and errors to the outputs.

2.8.2.4 Conclusion

Similar technique can be used to check the PCI DSS and PA DSS compliance. If there is any non-compliance source code, that code segment can be highlighted or can add warnings and errors to the output at the building time or run time.

2.8.3 PMD

2.8.4 Introduction

PMD is a rule-set based java code analyser. It can identify following issues in the source code by analysing the source code statically.

- Possible bugs - Empty try, catch, finally or switch blocks can be identified.
- Dead Code which are Unused local variables, parameters or private methods
- Empty if or while statements
- Over complicated expressions which are unnecessary if statements, for loops or while loops
- Classes with high cyclomatic complexity measurements using control flow graph.
- Duplicate code segments.

PMD identifies not errors but rather inefficient code segments in the program. These intuition used in PMD can be also applied to identify the non-compliance of PCI DSS and PA DSS.

PMD has plugins for JDeveloper, Eclipse, NetBeans, IntelliJ IDEA, Maven, Ant, Gradle, Hudson, Jenkins, SonarQube and Emacs. PMD can be integrated as a plugin and used inside the above applications to detect the inefficient source codes.

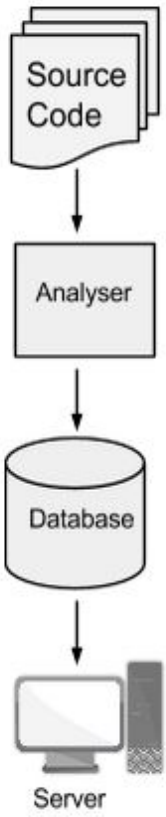
2.9 Conclusion

To ensure the source code is PCI-DSS or PA-DSS compliant, the source code analyser will be useful. To build a source code analyser a parser, a tree builder, and tree analysers should be use. There are lot of open source application which can be used directly without implementing these fundamental software components. Such applications are SonarQube, Checkstyle and PMD.

The source code should be checked statically and dynamically by the compliance monitor. Compliance should be able to monitor the source code and notify the developer and the admin. Admin should be notified if there is a non-compliance in the source code at building time or running time of the application.

These requirements can be catered by implementing a source code analyser according the following architecture.

2.9.1 Propose architecture of PA-COM



2.9.1.1 Source Code

Product source code will be taken as an input to the system and monitor the source code using analysers. All the source codes will be taken as the input without any discrimination. In the initial stage only Java source codes will be filtered and used to monitor for non-compliances.

2.9.1.2 Analysers

In the initial stage Data Flow analyser, source code transformation detector and Database query analyser will be implemented.

2.9.1.3 Database

Database will keep the all the noncompliance situations and non-compliance details with the developer. It will always keep the details related to the dynamic analysis.

2.9.1.4 Server

All the non-compliances will be notified to the server. Admin can show all these notifications from its portal.

3 PROBLEM STATEMENT

Even though the vendor adhere to the compliances in the initial stages of the development as the application gets evolved, compliancy deteriorates. This is mainly due to the number of developers getting involved in the development process and the high demand of the requirements to be completed in limited time makes the attention drop from the PCI DSS compliancy maintenance. And correcting of bugs and adding new features might also contribute towards the deterioration of the compliancy.

Payment Application Data Security Standards (PA DSS [2]) is a sub-standard derived from PCI DSS which specifically defines in-depth procedures that a software vendor should follow in developing payment application. This standard can be used as a guideline in developing the code analyser.

The available tools does not address the above problem of

3.1 Objectives

The main objective of Payment Application Compliance Monitor is to provide the developers the support needed to develop and maintain the PA DSS compliance of the application.

Identifying the requirements that can be checked through code analysing is one of the key objectives. It is critical to research on how the existing main stream code analysing techniques can be used to check PA DSS compliance.

There are standard violations that can't be checked with the existing techniques. These needed to be researched on how those can be adapted to a code analysing technique by using hybrid techniques or new methods.

Upon completing the package the aim is to integrate the system into OSSIM [3] to make OSSIM a complete product that not only customers but also the developers can use to maintain compliance with PCI DSS.

4 DESIGN

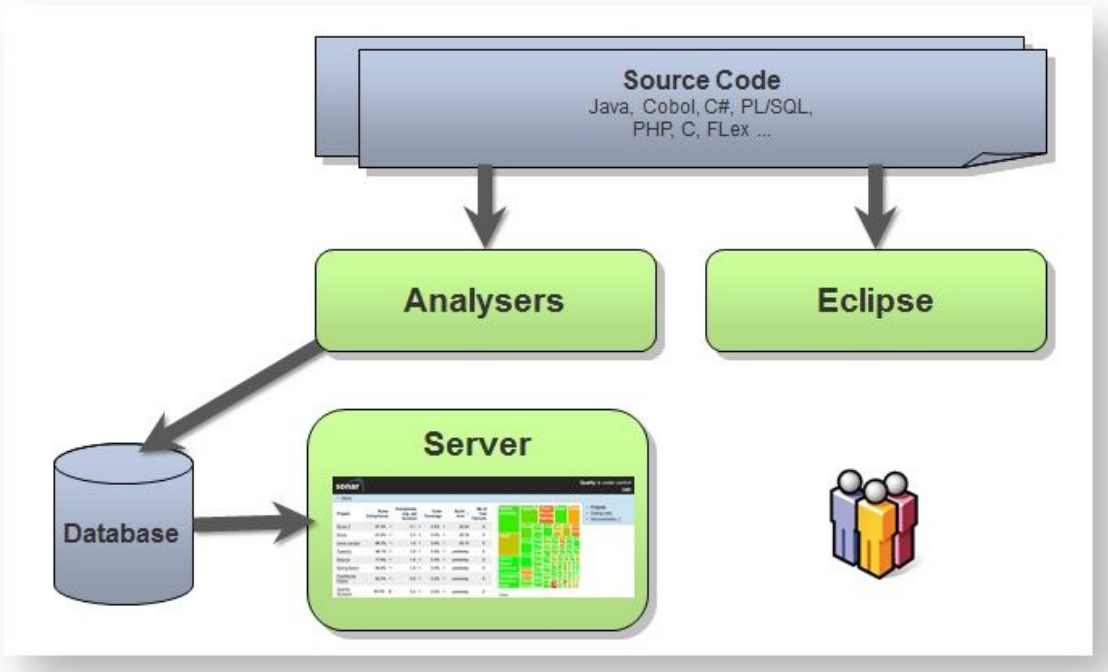
4.1 sonarqube Introduction

SonarQube is an open platform to manage code quality. As such, it covers the 7 axes of code quality: Architecture, comments, duplication, code rules, unit test, potential bugs, and complexity.

SonarQube has got a very efficient way of navigating, a balance between high-level view, dashboard, TimeMachine and defect hunting tools. This enables to quickly uncover projects and components that are in Technical Debt to establish action plans.

SonarQube is a web-based application. Rules, alerts, thresholds, exclusions, setting can be configured online. By leveraging its database, SonarQube not only allows to combine metrics altogether but also to mix them with historical measures

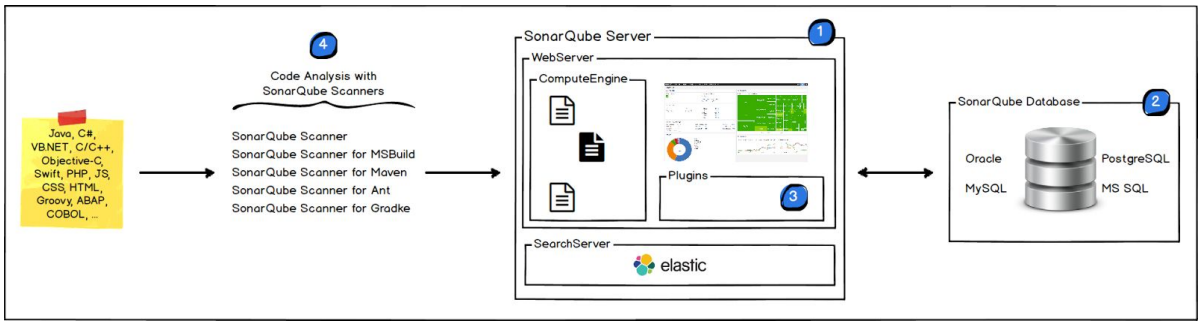
4.2 Sonarqube Architecture



ARCHITECTURE

The SonarQube Platform is made of 4 components:

1. One SonarQube Server starting 2 main processes:
    - a Web Server for developers, managers to browse quality snapshots and configure the SonarQube instance
    - a Search Server based on Elasticsearch to back searches from the UI
2. One SonarQube Database to store:
    - the configuration of the SonarQube instance (security, plugins settings, etc.)
    - the quality snapshots of projects, views, etc.
3. Multiple SonarQube Plugins installed on the server, possibly including language, SCM, integration, authentication, and governance plugins
4. One or more SonarQube Scanners running on the Build / Continuous Integration Servers to analyze projects

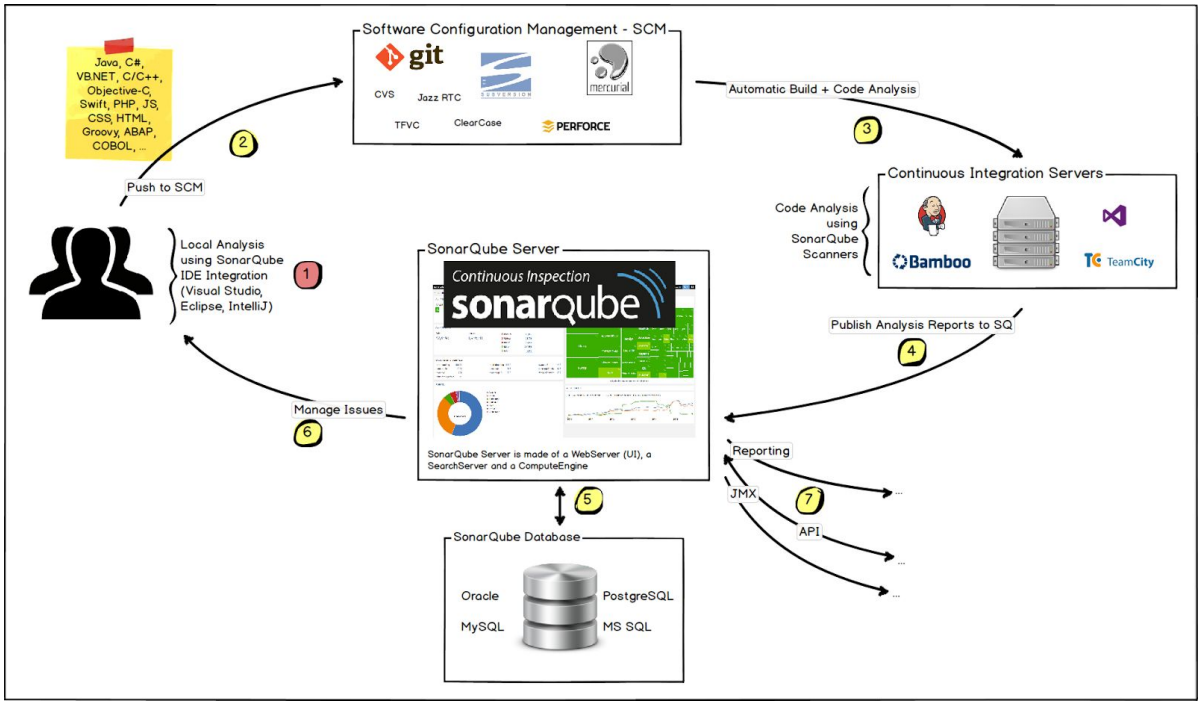


Integration

The following schema shows how SonarQube integrates with other ALM tools and where the various components of SonarQube are used.

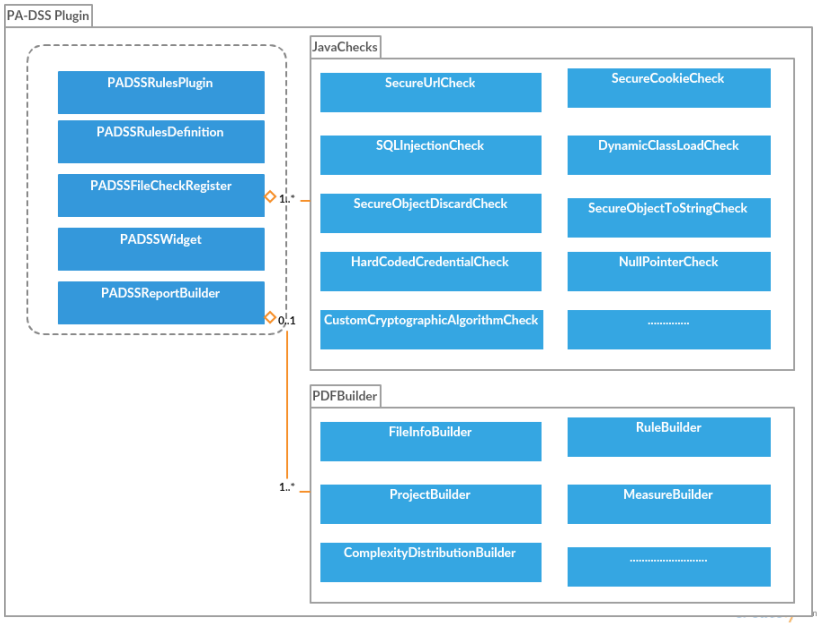
1. Developers code in their IDEs and use SonarLint to run local analysis.
2. Developers push their code into their favourite SCM : git, SVN, TFVC, ...
3. The Continuous Integration Server triggers an automatic build, and the execution of the SonarQube Scanner required to run the SonarQube analysis.
4. The analysis report is sent to the SonarQube Server for processing.
5. SonarQube Server processes and stores the analysis report results in the SonarQube Database, and displays the results in the UI.
6. Developers review, comment, challenge their Issues to manage and reduce their Technical Debt through the SonarQube UI.
7. Managers receive Reports from the analysis.

8. Ops use APIs to automate configuration and extract data from SonarQube.
9. Ops use JMX to monitor SonarQube Server.



4.3 PACoM Plugin and the widget

PACoM Plugin is created to detect non compliance against PA DSS. It will detect all the non compliance against PA DSS.



4.4 PA DSS development pitfalls and Sonar Rules

Develop a PADSS compliance application is not a single process. PADSS compliance should maintain throughout the software development life cycles. This compliance should maintain even after assessment is over. Most of the developer does not understand this fact. There are lots of issues occur with code changes. List of pitfalls which make application not comply with PADSS standards are as bellow,

Developer may use non secure urls to communication

Developer may hardcoded the passwords for debugging purpose

Sensitive data may not discard at the end

Secure object may print or logged for debugging purpose

Sensitive data may save to the database

SHA-1 and Message-Digest hash algorithms may use to encrypt the data

OWASP top 10 vulnerabilities

5 IMPLEMENTATION

This chapter gives a detailed description about implementation of our plugin. Section 5.1 describes the Sonarqube framework which we use as platform for our project and technologies we used. Section 5.2 describes the implementation details of each component in the plugin. Finally, Section 5.3 discusses the testing process.

5.1 Languages, Tools, and Technologies

As discussed in Chapter 4, our PACOM application was developed as a SonarQube extended plugin using Java EE 7 and Maven 3.1 . To simplify the development process we used IntelliJ IDEA [41] as our IDE. The PADSS widget is implemented using Ruby, HTML, CSS, Javascripts and Maven.

For version control we used Git. Our project is hosted at Git hub . It is easier to maintain the versions and do collaborative work using Git

5.1.1 Sonar server

Sonar server is a central management platform, dedicated to continuously analyze and measure source code quality. It was developed using Java and Ruby and support to analyse Java, C/C++, Javascripts, PHP, Python, ect.

5.1.2 Sonar runner

Sonar runner is a application which runs in each developer computer. It control the versioning of code and support user to analyse source code. Sonar runner access to the code and it generate abstract syntax tree and pass it to sonar server. Code analysis happens in the sonar servers.

5.1.3 Sonar API

Sonar API is the API which give access to developers to write new sonar rules. Through Sonar API developer can get access to the abstract syntax tree and code. Developer can run

his own algorithms on abstract syntax tree and code base. Through Sonar API developer can add more issues to sonar which are identified by his own algorithms.

5.2 PADSS Plugin

5.2.1 creating a plugin

A SonarQube plugin is a set of Java objects that implement extension points. These extension points are interfaces or abstract classes which model an aspect of the system and define contracts of what needs to be implemented. They can be for example pages in the web application or sensors generating measures.

The extensions implemented in the plugin declared in a Java class extending org.sonar.api.Plugin. This class has declared in the pom with the property

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>lk.ac.mrt.cse</groupId>
 <artifactId>java-padss-rules</artifactId>
 <version>1.0-SNAPSHOT</version>
 <packaging>sonar-plugin</packaging>
 <properties>
   <java.plugin.version>3.6</java.plugin.version>
 </properties>
 <name>Java PA-DSS Rules</name>
 <description>Java custom rules</description>
 <dependencies>
   <dependency>
     <groupId>org.codehaus.sonar</groupId>
     <artifactId>sonar-plugin-api</artifactId>
     <version>4.5</version>
     <scope>provided</scope>
```

```
    </dependency>
    <dependency>
      <groupId>org.sonarsource.java</groupId>
      <artifactId>sonar-java-plugin</artifactId>
      <type>sonar-plugin</type>
      <version>${java.plugin.version}</version>
      <scope>compile</scope>
    </dependency>
<build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.sonar</groupId>
        <artifactId>sonar-packaging-maven-plugin</artifactId>
        <version>1.12.1</version>
        <extensions>true</extensions>
        <configuration>
          <pluginClass>lk.ac.mrt.cse.padss.PADSSRulesPlugin</pluginClass>
        </configuration>
      </plugin>


    </plugins>
  </build>
</project>
```

main PADSS plugin class

```
/**
* Entry point of plugin
*/
public class PADSSRulesPlugin extends SonarPlugin {
 @Override
```

```
public List getExtensions() {
  return Arrays.asList(
    // server extensions -> objects are instantiated during server startup
    PADSSRulesDefinition.class,
    // batch extensions -> objects are instantiated during code analysis
    PADSSFileCheckRegistrar.class);
 }
}
```

5.2.2 Developing PADSS Rules

We implemented sonar rules which are mentioned in chapter 4 using code analysis techniques we discovered in chapter 3. In this section it is discussed about each rule we have developed and technique we used to develop rules.

5.2.2.1 Avoid usage of non-secure URLs

PADSS compliant applications should always use secure urls to communicate. This compliance brakes when user establish a non secure connection. We used programing query technique to identify these type of non-compliances in the source code. We query the connections with non secure urls. Through sonar API we insert non-compliances to sonar database.

Passwords should not be hard-coded

To get the application compliant with PADSS standards passwords should not be hard-coded in your application. We can find these type of non-compliances using the bug pattern matching technique described in the chapter two.

Secure Objects should discard at the end

This rule is a major requirement in the PADSS compliance. Leapset save secure data to a specified secure object. They have developed discard method to discard secure object's data before releasing it to the garbage collector. Through Model checking technique we can

identify the these type of non compliance in the source code where the secure object has not being discarded.

Secure Objects should not convert toString

This rule was developed using programing query technique described in chapter two. In PADSS compliance application secure data should not be saved. This rule identify the ability of system print or logged secure object's data by calling toString method.

Secure Objects should not return secure variables

This rule was developed using programing query technique described in chapter two. We can query the secure object for methods and identify when sensitive data returns.

Secure Objects should not save data to database

This rule was developed using database query analysis technique. Using that technique we can identify noncompliance when sensitive data is sent to the database.

Values passed to SQL commands should be sanitized

This rule was developed using database query analysis technique. If user use SQL commands without proper sanitization it may lead to injection attacks.

Cookies should be "secure"

Some application save data is client side cookies. These cookies alway should be secure. This rule was developed using programming query analysis technique. Using that technique we can identify noncompliance when user create non secure cookie.

Null pointers should not be dereferenced

This rule comes under OWASP top 10 vulnerabilities. We identify this vulnerability using model checking technique.

Only standard cryptographic algorithms should be used

Use proper cryptographic is one of requirement in PADSS compliance. But some developer extend cryptographic algorithms and implement their own cryptographic schemes. By programing query technique we can find these type of non compliances.

SHA-1 and Message-Digest hash algorithms should not be used

SHA-1 and Message-Digest hash algorithms are now considered as vulnerable algorithms. Therefore we query usage of SHA-1 and Message-Digest hash algorithms using programing query algorithms.

Values passed to OS commands should be sanitized

This rule was developed using programing query analysis technique. When user run OS command, command should be sanitized otherwise. This rule was developed under OWASP injection coding standards

Classes should not be loaded dynamically

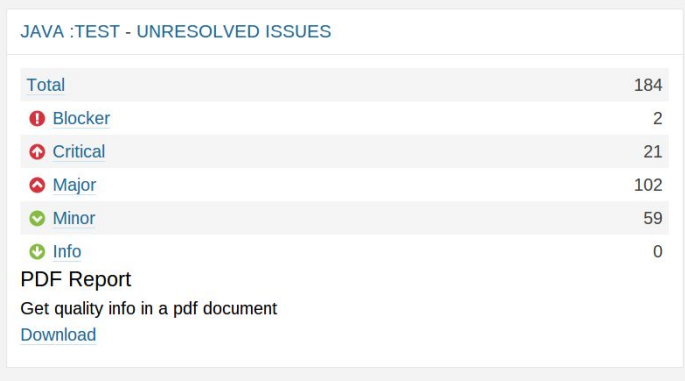
This rule was developed using programming query technique. According to OWASP coding standards dynamically load classes can lead to injection attacks. By using programming query technique we can identify noncompliance when user load classes dynamically.

5.2.2 Sonar widget

Sonar widget was developed according to the system architecture which was discussed in Chapter 4. Sonar widget contain major three views. This widget was developed using Ruby, HTML, CSS, and Javascript.

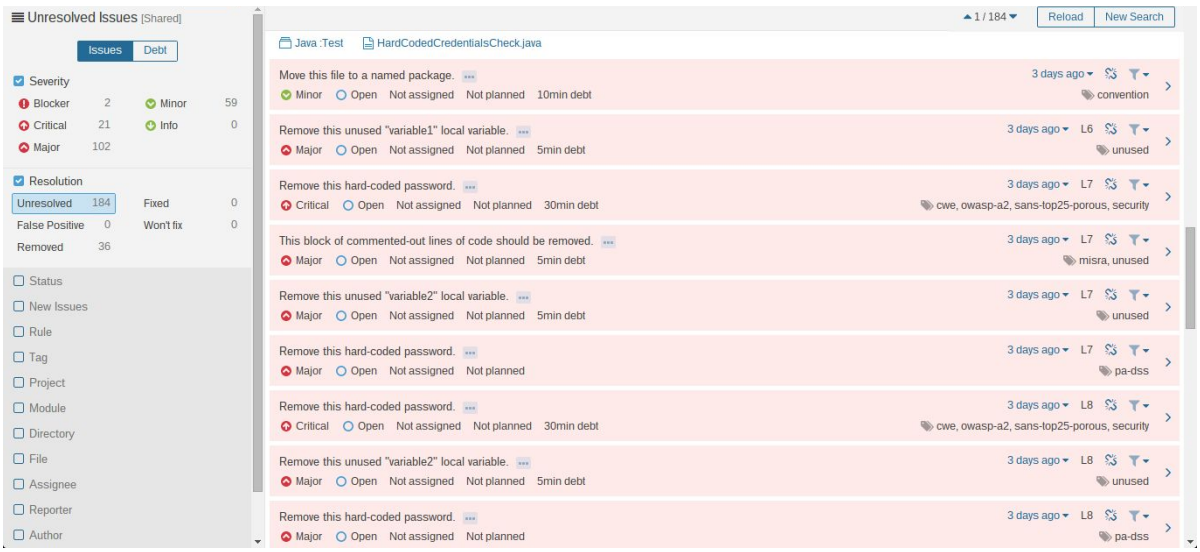
Summary View

Summary view is contains the summary about the project. System admin can simply get a idea about the project through the summary view.



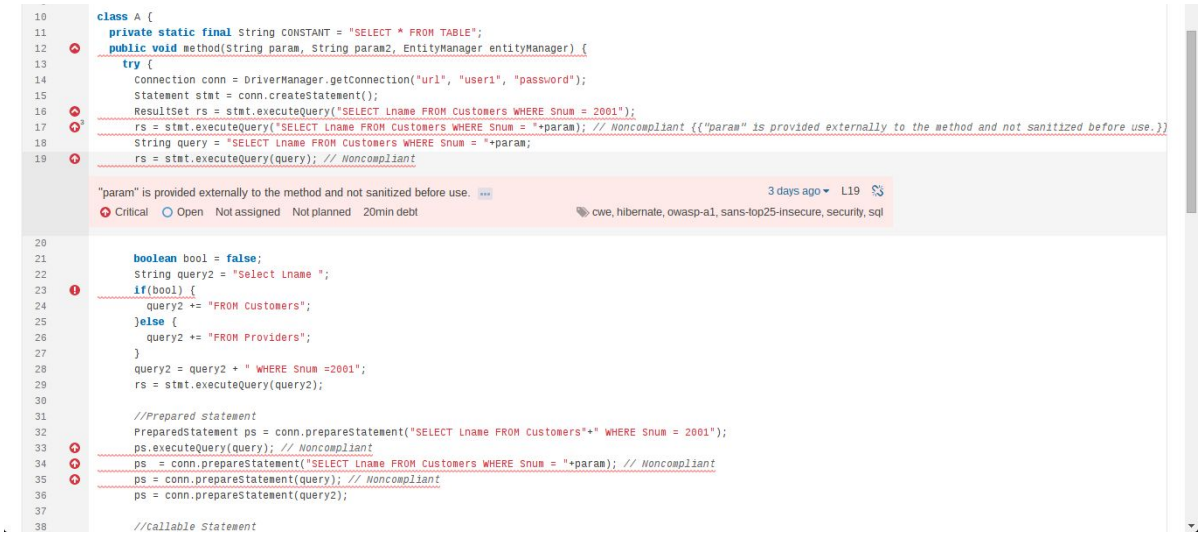
Issues view

Issue view contains list of issues in the project. System admin can filter issues from rule name, priority level, tags , unresolved, etc.



File View

File view contains the noncompliance associated with each file.User can view the occurrences of issues in file.



5.2.3 Report Generation

Report generation is an another major requirement we discuss in our system requirement specification. As discussed in chapter 4, we generate report about the project after each sonar analysis. System can access the PDF report through the web interface. This report include about the most violated rules, most violated files, and about the issues related to the PADSS compliance.

5.3 Testing

The system is tested at various stages in the process. In the development stage, to ensure the coding quality and standards, Junit unit tests are used. We have implemented a suite if integration tests to ensure that the functionalities of our project are working properly. After the system is implemented we carried out a performance test to ensure that the performance of our system help developer to develop applications with PADSS compliants.

5.4 Software Development workflow

Git is the version control system we used to manage the software versions. To use Git optimally and easily we changes the some git configurations for easiness.

We created some alias for frequently used git configurations

st = status
ci = commit
co = checkout
br = branch
To fix merge conflicts, we used meld and configured it as the mergetool

When we using git we used two branches locally. first one is the master branch which will reflect all the changes in the remote repository. and the other branch which was named as working is used to have our code changes.

After committing the changes to our branch, we pull all the new changes from the remote to the master branch and rebase it with the working branch if there is any conflict at the rebasing time we fixed them. Then we merged the working branch to the master branch. Then pushed the new changes to the remote repository.

[Working branch]	git add --all
	git commit -m “commit message”
	git co master
[Master Branch]	git pull origin master
	git co working
[Working branch]	git rebase master

if there is any conflict

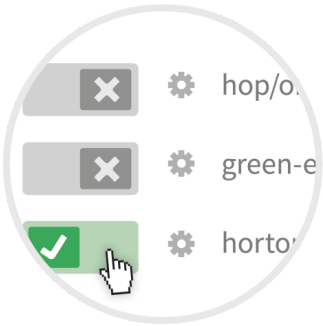
[Working branch]	git mergetool

then fix the issues

[Working branch]	git rebase --continue

For continuous integration we used travis CI . It will merge the pushed source code to the current source base if only the merged source code is built without any issues. If the build failed it will rollback the merge and notify to the preconfigured users.

We used following method to configure the travis for github hosted projects



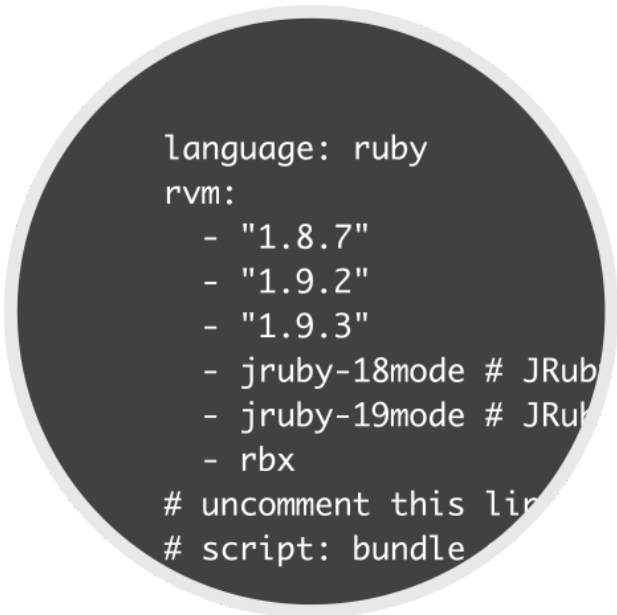
Activate GitHub Repositories

Once you're signed in, and we've initially synchronized your repositories from GitHub, go to your profile page for open source or for your private projects.

You'll see all the organizations you're a member of and all the repositories you have access to. The ones you have administrative access to are the ones you can enable the service hook for.

Flip the switch to on for all repositories you'd like to enable.

Add .travis.yml file to your repository



In order for Travis CI to build your project, you need to tell the systems a little bit about it. You'll need to add a file named .travis.yml to the root of your repository.

If .travis.yml is not in the repository, is misspelled or is not valid YAML, Travis CI will ignore it.

Trigger your first build with a git push

Once the GitHub hook is set up, push your commit that adds .travis.yml to your repository. That should add a build into one of the queues on Travis CI and your build will start as soon as one worker for your language is available.

To start a build, perform one of the following:

- Commit and push something to your repository
- Go to your repository's settings page, click on "Webhooks & Services" on the left menu, choose "Travis CI" in the "Services", and use the "Test service" button.

6 OUTCOMES

Pitfall Scenario Coverage

PA DSS is the primary guideline followed by software vendors when developing payment related applications. This guidelines can be divided into major 13 categories. These categories include guideline about design, development, maintain, documentations, etc. Some of this guideline can be model as coding standards guidelines. Once a payment application vendor has reached the point where a PA-DSS assessment needs to be performed, there are lots of pitfalls that might arise due to lack of coding standards.

Our PADSS plugin include 13 rules under four categories of PADSS guideline mentioned above. These 13 rule covers more than 50 PADSS pitfalls we have described in chapter 4.

Standard Coverage

As we discuss in the chapter two sonarqube has some security related rules which are needed to get comply with PADSS. We developed some more advanced rules to cover some parts of PADSS compliance. With our plugin sonarqube cover following areas in pADSS compliance.

Requirement	Related Rules
1) Do not retain full track data, card verification code or value (CAV2, CID, CVC2, CVV2), or PIN block data	1) Secure Objects should discard at the end 2) Secure Objects should not convert toString 3) Secure Objects should not return secure variables 4) Secure Objects should not save data to database
5) Develop secure payment applications	1) Avoid usage of non-secure URLs 2) Passwords should not be hard-coded 3) Values passed to SQL commands should be sanitized

	4) cookies should be "secure" 5) Null pointers should not be dereferenced 6) Values passed to OS commands should be sanitized 7) Classes should not be loaded dynamically
7) Test payment applications to address vulnerabilities and maintain payment application updates	1) OWASP security standards
11) Encrypt sensitive traffic over public networks	1) Only standard cryptographic algorithms should be used 2) SHA-1 and Message-Digest hash algorithms should not be used

Review and Test using tool false positive… etc

Centralized Monitoring

Sonarqube runs in a central server. Non-Compliances in each code instance can monitor from this central server. After each code change admin can generate a pdf report about the code standards. So with our PADSS plugin admin can Maintain PADSS standards in payment application.

Report

Report generation is an another major requirement in PADSS compliance. In PADSS compliance vendor should follow security standards and he should provide reports which prove their application up to standards. From security assessor's perspective, he need to verify that application is up to standards. Security assessor should manually review the application. Report help assessor to filter out files to check. This report include about the most violated rules, most violated files, and about the issues related to the PADSS compliance.

Integration with IDE

Sonarqube provide way to integrated it with external applications. Currently sonarqube can integrate with Intellij and Eclipse IDE s. Integration sonarqube with IDE help developer to identify noncompliance in real time. It reduce the effort of developer to develop payment application with PADSS compliance.

7 SONARQUBE INTEGRATION

Sonarqube Installation

Download sonarqube distribution

Unzip the distribution file <install_directory>

<install_directory>/conf/sonar.properties to configure the database settings.

```
sonar.web.host=192.0.0.1

sonar.web.port=80

sonar.web.context=/sonar
```

Execute the following script to start the server:

- On Linux/Mac OS: bin/<YOUR OS>/sonar.sh start
- On Windows: bin/windows-x86-XX/StartSonar.bat

7.2 Plugin Installation

There is two options to install a plugin into SonarQube :

- Automatically, from the SonarQube UI using the Update Center
- Manually

Using the Update Center

If your SonarQube Server has been freshly installed, it won't be authorised to connect outside your company to download plugins. As a consequence, you will need to use the manual installation way.

If you have access to the internet and you are connected with a SonarQube user having the Global Permission "Administer System", you can go in Settings > Update Center.

- Locate the "Available Plugins" tab
- Find the plugin you want to install
- Click on Install and wait for the download to be processed

Once done, you will need to restart your SonarQube Server.

Manual Installation

In the page dedicated to the plugin you want to install (ex: for Python : Python Plugin), click on the "Download" link of the version compatible with your SonarQube version.

Upload the downloaded jar file in your SonarQube Server and put it in the directory : $SONARQUBE_HOME/extensions/plugins.

If another version of the same plugin is already there, you need to remove/backup it as only one version of a given plugin must be available in the extensions/plugins directory.

Once done, you will need to restart your SonarQube Server.

7.3 Widget Creation

go to configure widget section and add the PA DSS widget and select the project you want to filter data for. Then you can see all the details related to the widget in the dash board

7.4 Intellij Plugin integration

To take full advantage of SonarQube in IntelliJ, it is recommended that your project be analyzed on a regular basis by SonarQube. Regular analysis allows the IntelliJ plugin to distinguish between existing issues and any new ones you introduced.

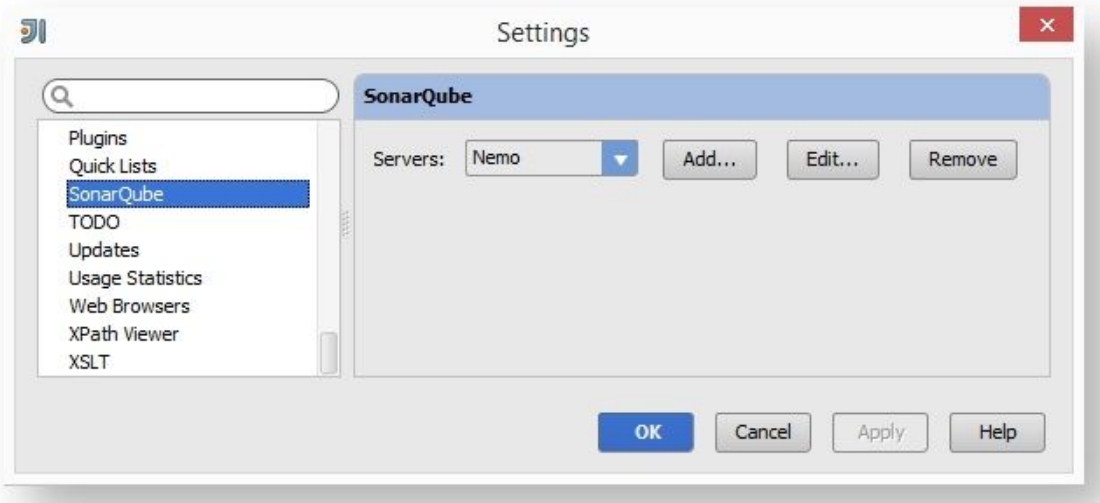
If your project is not already under analysis, you'll need to declare it through the SonarQube web interface.

Once your project exists in SonarQube, you're ready to get started with SonarQube in IntelliJ.

Setting SonarQube Servers

Go to **File > Settings > SonarQube** to add, edit or remove SonarQube servers and configure your SonarQube instance.

The user you set to access the server has to be granted the **Execute Preview Analysis** permission.



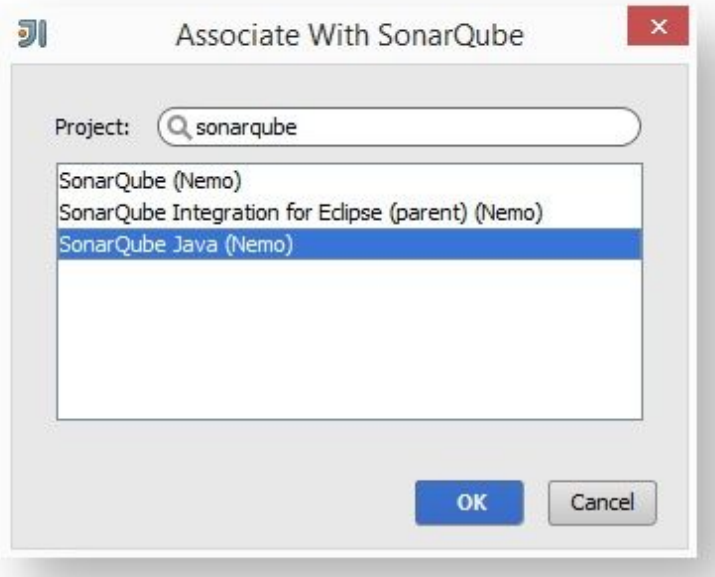
.

Linking a Project to One Analyzed on a SonarQube Server

Linking for the first time

Once the SonarQube server is defined, the next step is to link your IntelliJ project with its counterpart on the SonarQube server.

To do so, right-click the project and choose **Associate with SonarQube**. Start typing the name of the project and select it in the result list:



7.5 Sonar Runner

create sonar-properties file in your project

Edit the sonar-properties file as follows

download the sonar-runner distribution and add the location to the environment variables as SONAR_RUNNER_HOME

then run sonar-runner command in the terminal inside your project. Then it will run the required checks for the product and save them in the sonarqube server.

8 SUMMERY

For payment application software vendors to get PCI DSS or PA DSS certification, they have to comply with the standards that are set from the PCI council. In order to achieve this software vendors have already has setup software process mechanisms such as code review by peers and by a product lead. Even though it is necessary to have these steps in order to maintain compliance. There are pitfalls that the developers frequently fall into. PACoM eliminates these pitfalls in a the code as early as possible before going into code review. PACoM provides the much needed assistance to the developers to easily catch those mishaps.

By PACoM the development team will be able to minimize introducing non-compliant code. This will enable saving of development time as the errors can be pointed at the IDE, Source Control or Continuous integration phases. And this will minimize the workload on code reviewing further reducing the development time and cost.

PACoM has the ability to integrate into the existing software development workflow through number of different methods. It can be used with mostly used continuous integration platforms such as Travis CI and Jenkins, source control systems such as Git and SVN, and integrated development environments such as Eclipse and IntelliJ.

PACoM currently has the ability to detect for about 30 pitfalls and as PACoM is open source anyone can contribute to make the above number a larger one. Open source gives the ability to customize rules according to specific needs of a particular application development organization.

Problems and Challenges

PA and PCI standards covers a broad spectrum of payment applications and does not have definite explanations on the standard. This was a major challenge when understanding the standards and how those will be defined for the system in question. To overcome this problem we approached TechCert to get the opinion of a QSA on how they will interpret the standards. And again we got information on how Leapset comply with the standard how

they interpret the standards. Based on these evidences the pitfalls were identified and rules were designed and implement as required.

Understanding how Sonarqube works and how to implement custom rules had a steep learning curve. This was due to the lack of documentation provided by the Sonarqube platform. We have described the development process in the report for future references. And it is added to the documentation of PACoM product.

Future Work

PCI DSS and PA DSS compliance management has clustered into one single platform OSSIM. PACoM will be a significant addon to OSSIM as it already covers detecting non-compliant activities in live environments. By adding PACoM to the suit, OSSIM will cover the development phase and be a more complete suit for maintaining PCI DSS and PA DSS standards.

There are other scenarios or pitfalls that was discovered but not implemented in this version. PACoM by being an open source project has the ability to be developed further by adding new rules.

An extension point to PACoM will be adding a dynamic code analyser. While in the project we saw many instances that a dynamic analyser can detect where static analyser would not be sufficient. As most of the standards discuss about the actual runtime behaviour transforming those into static analyzable models was challenging but by adding dynamic analysing features will make PACoM a better full scope product.

List of Abbreviations

API	Application Program Interface
ASA	Adaptive Security Appliance
ARP	Address resolution protocol
CI	Continuous Integration
CIS	Continuous Industry Scheme
COTS	Commercial Off-the-shelf
CPU	Central Processing Unit
DNS	Domain Name Server
DFA	Data flow analysis
ESX	Elastic Sky X
FWSM	Firewall Service Module
HID	Host Intrusion Detection
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IP	Internet Protocol
JAR	Java Archive
JDBC	Java Database Connection
OS	Operating System
OSSEC	Open Source Security
OSSIM	Open Source Security Information Management
OSVDB	Open Source Vulnerability Detection
PCI	Payment Card Industry
POS	Point of Sales
PAN	Permanent Account Number
PIN	Personal Identification Number
PIX	Private Internet Exchange
SAD	seasonal affective disorder
SIEM	Security Information and Event Management
SNMP	Simple Network management Protocol
USM	Unified Security Management
VPN	Virtual private network

REFERENCES

[1] PCI Security Standards Council, “Payment Card Industry (PCI)- Requirements and Security Assessment Procedures,” April 2015. [Online]. Available: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf. [Accessed 20 July 2015].

[2] “Payment Application Data Security Standard - Requirements and Security Assessment Procedures,” May 2015. [Online]. Available: https://www.pcisecuritystandards.org/documents/PA-DSS_v3-1.pdf. [Accessed 25 July 2015].

[3] AlienVault Inc, “OSSIM : The Open Source SIEM | Alien Vault,” 2015. [Online]. Available: https://www.alienvault.com/products/ossim. [Accessed 10 August 2015].

[4] Trend Micro, “OSSEC | Open Source SECurity,” 2010. [Online]. Available: http://www.ossec.net/. [Accessed 10 August 2015].

[5] Trend Micro, “OSSEC Architecture,” 2010. [Online]. Available: https://ossec-docs.readthedocs.org/en/latest/_images/ossec-arch.jpg. [Accessed 20 August 2015].

[6] AlienVault Inc., “OSSIM General Description,” 2008.

[7] C. Leres, “arpwatch(8) - Linux man page,” [Online]. Available: http://linux.die.net/man/8/arpwatch. [Accessed 20 August 2015].

[8] M. Zalewski, “p0f v3,” 2012. [Online]. Available: http://lcamtuf.coredump.cx/p0f3/. [Accessed 20 Auugust 2015].

[9] “Passive Asset Detection System,” 18 June 2005. [Online]. Available: http://passive.sourceforge.net/. [Accessed 20 August 2015].

[10] “OpenVAS- Open Vaulnerability Assessment System,” 2 April 2015. [Online]. Available: http://www.openvas.org/. [Accessed 20 August 2015].

[11] OCS Inventory Team, “OCS Inventory NG | Open Computers and Software Inventory Next Generation,” 2014. [Online]. Available: http://www.ocsinventory-ng.org/. [Accessed 20 August 2015].

[12] Cisco Inc., “Snort.Org,” 2015. [Online]. Available: https://www.snort.org/. [Accessed 20 August 2015].

[13] Open Information Security Foundation, “Suricata | Open Source IDS / IPS / NSM engine,” Open Information Security Foundation, 2015. [Online]. Available: http://suricata-ids.org/. [Accessed 20 August 2015].

[14] SICKBIT Syndicate, “TCPTrack – Simple TCP Connection Monitor,” 30 November 2012. [Online]. Available: http://sickbits.net/tcptrack-simple-tcp-connection-monitor/. [Accessed 20 August 2015].

[15] ntop , “ntop | High Performance Network Monitoring Solutions based on Open Source and Commodity Hardware.,” ntop, 2015. [Online]. Available: http://www.ntop.org/. [Accessed 20 August 2015].

[16] Nagios Enterprises, “Nagios Core. Nagios Open Source Project,” Nagios Enterprises, LLC., 2015. [Online]. Available: https://www.nagios.org/. [Accessed 20 August 2015].

[17] Open Sourced Vulnerability Database (OSVDB), “OSVDB: Open Source Vulnerability Database,” 2015. [Online]. Available: http://osvdb.org/. [Accessed 20 August 2015].

[18] “MUNIN,” 2015. [Online]. Available: http://munin-monitoring.org/. [Accessed 20 August 2015].

[19] “NFDUMP,” 1 December 2014. [Online]. Available: http://nfdump.sourceforge.net/. [Accessed 20 August 2015].

[20] “NfSen - Netflow Sensor,” 31 December 2011. [Online]. Available: http://nfsen.sourceforge.net/. [Accessed 20 August 2015].

[21] AlienVault Inc, “Unified Security Managment (USM) Platform,” 2015. [Online]. Available: https://www.alienvault.com/products. [Accessed 23 August 2015].

[22] AlienVault Inc., “PCI DSS Compliance Software,” 2015. [Online]. Available: https://www.alienvault.com/solutions/pci-dss-compliance. [Accessed 23 August 2015].

[23] B. P. T. G. K. C. a. M. R. J. Chow, “Understanding data lifetime via wholesystem simulation,” in *USENIX Security Symposium*, 2004.

[24] M. F. C. C. A. W. a. S. H. A. Ho, “Practical taint-based protection using demand emulation.,” *SIGOPS/EuroSys, ACM,* vol. 40, 2006.

[25] E. C. a. D. Wagner, “Efficient character-level taint,” *Workshop on Secure web,* pp. 3-12, 2009.

[26] G. P. K. J. a. A. D. K. V. P. Kemerlis, “Practical dynamic data flow tracking for commodity systems,” *SIGPLAN Notices/VEE,* vol. 47, 2012.

[27] J. R. C. a. F. T. Chong, “ Minos: Control data attack prevention orthogonal to memory model,” *MICRO IEEE,* pp. 221-232, 2004.

[28] B. S. a. C. L. W. Chang, “Efficient and extensible security enforcement using dynamic data flow analysis,” in *CCS ACM*, 2008.