A Blockchain-Based Approach for Tracing Security Requirements for Large Scale and Complex Software Development

by

Adi Kulkarni

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2022 by the
Graduate Supervisory Committee:

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

Security requirements are at the heart of developing secure, invulnerable software. Without embedding security principles in the software development life cycle, the likelihood of producing insecure software increases, putting the consumers of that software at great risk. For large-scale software development, this problem is complicated as there may be hundreds or thousands of security requirements that need to be met, and it only worsens if the software development project is developed by a distributed development team. In this thesis, an approach is provided for software security requirement traceability for large-scale and complex software development projects being developed by distributed development teams. The approach utilizes blockchain technology to improve the automation of security requirement satisfaction and create a more transparent and trustworthy development environment for distributed development teams. The approach also introduces immutability, auditability, and non-repudiation into the security requirement traceability process. The approach is evaluated against existing software security requirement solutions.

DEDICATION

This thesis is dedicated to my mother and father, who have been instrumental in supporting me throughout my education. Thank you.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to all of those that have made an impact on this thesis. I would like to recognize my committee chair, Professor Stephen Yau, for being extremely supportive and providing me with advice throughout the journey of my thesis and most of my time at Arizona State University. Professor Yau's insight and academic experience aided me greatly in improving my ability to perform research as well as my academic and personal development. I would also like to thank my committee members, Professors Jaejong Baek, Ayan Banerjee, and Ruoyu Wang for giving their time and feedback to complete this thesis.

I would also like to recognize the financial support of the National Science Foundation's CyberCorps: Scholarship for Service and appreciate the immense support it has provided throughout my time at Arizona State University.

I would also like to thank my friends and family for providing me with feedback and input, listening to my talks about my research, and supporting me with my efforts in this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND 5

3 CURRENT STATE-OF-THE-ART 14

4 OVERALL APPROACH 17

5 TRACING IN THE DESIGN PHASE 32

6 TRACING IN THE IMPLEMENTATION PHASE 34

7 TRACING IN THE TESTING PHASE 38

8 INNOVATION 42

9 AN ILLUSTRATIVE EXAMPLE 46

10 EVALUATION 60

11 CONCLUSION 66

12 FUTURE WORK 67

REFERENCES 68

LIST OF TABLES

Table Page

1. Notation and Symbols for Overall Approach 20

2. Requirement Field Purposes 23

3. An Example Requirement with Attributes 50

4. Component Assignment to Nodes on Development Team ... 55

5. Comparison of Approach to Centralized Solutions.............. 64

LIST OF FIGURES

Figure Page

1. Threat Model Diagram from OWASP 15
2. The Overall Approach 21
3. Security Requirement Mapping to Software Component 27
4. Tracing in the Implementation Phase.............................. 29
5. The Private Blockchain Network.................................... 48
6. Customer Login Portal of Illustrated Example 54

CHAPTER 1

INTRODUCTION

Security should be embedded in the software development life cycle to ensure the creation of secure software. This begins with putting security requirements at the forefront for all phases of the software development lifecycle, namely the design, implementation, and testing phases. For complicated and large-scale software, security requirement traceability becomes difficult as security requirements need to be traced among several different software components including the component interaction. This becomes increasingly difficult in distributed development environments, as the coordination between the development teams needs to be sufficient for security requirements to be properly addressed. In addition, without the proper tools to facilitate coordination for decentralized software development environments, common security activities like security testing, vulnerability mitigation, and code audits are non-trivial to conduct. This problem becomes even more complicated when considering the design and implementation phases as well, when vulnerabilities are the most likely to be introduced into the system either by developers or inherited components such as open-source libraries or other open-source software used in the software development project. Therefore, it is important for software developers and large distributed development teams to adhere to a software development lifecycle that places security at the forefront of development, ensuring that potential vulnerabilities are mitigated and that an environment is provided for security activities to be conducted with little to no manual intervention from the development teams. For large-scale distributed development teams, there are not adequate solutions for security requirement traceability that will consistently produce high quality software results for decentralized work environments. For complex software development projects which may have thousands of security requirements, it is important that the history for every requirement is recorded correctly such that they are able to be satisfied at the end of the development cycle.

The traceability of security requirements for large scale software development projects are crucial to their success, and the proper management of security requirements will ensure the creation of secure, robust, vulnerability-free software. When considering the traceability of security requirements over the long duration of the development of complicated, large-scale software, it is important to also consider the auditability, transparency, consensus, and other security properties of the system used to manage them. For some of these desirable properties, one can look to blockchain technology as a part of the solution to help grant some of these traits for security requirement traceability during the software development life cycle. Blockchain allows for developers to create software and provides an immutable history of security requirements while also facilitating the process of software development in a decentralized development environment [1].

It is the aim of the research conducted to provide a framework for security requirements to be traced over the software development life cycle, while also reducing the number of vulnerabilities present in the developed software. The organization of this thesis is given over nine chapters. The first chapter details the importance of why security requirements traceability and management are important for the overall software development lifecycle. The second chapter will cover background information about security requirements identification and traceability, as well as an overview of blockchain technology. The third chapter will detail the existing research that has been done for security requirements management solutions. The fourth chapter covers my overall approach for handling security requirements traceability using blockchain for large scale and complex software development. The fifth chapter will discuss the innovative aspects of my approach and why they are beneficial to the overall software development lifecycle. The sixth chapter will provide an example that illustrates my overall approach in a clear and understandable fashion. The eighth chapter will provide the conclusion for my research and overall approach. The ninth chapter will provide a brief overview of the potential future of blockchain and software security requirements in the software development life cycle.

CHAPTER 2

BACKGROUND

2.1 Security Requirements

The proper management of security requirements are paramount to the completion of secure, vulnerability-free software. Therefore, software development teams must prioritize security requirement satisfaction during the software development life cycle to eliminate and mitigate the introduction of vulnerabilities in software. Properly generated security requirements allow developers to ensure that their software is only able to be used for their intended purpose and not be circumvented to perform malicious activities [2].

OWASP [3] has created a method for defining security requirements utilizing the resources and tools that they have created [4]. In this method, the OWASP ASVS [5] is used as a set of statements that can be improved upon using conventional software engineering techniques like user story generation, threat modeling, and misuse cases. This allows developers to create security requirements that are specific and detailed for their own application instead of adhering to the one-size-fits-all approach that the ASVS provides.

The next step in the process is implementation, which consists of four sub-steps. These sub-steps are the discovery and selection phase, the investigation and documentation phase, the implementation phase, and the testing phase. In the discovery and selection phase, developers are tasked with understanding the security requirements given to them from something like the ASVS and deciding what security requirements fit the need of their release. As the development of the application progresses, more security requirements will be implemented and add additional security functionality over the development of the application. In the investigation and documentation phase, the developer will review the existing software against the set of security requirements and determine whether the software already complies with the new requirements. This investigation will create the documentation of the phase. The next phase is the implementation phase, where the software will be modified to comply with any security requirements that the software is already deemed to not comply with. In this step, new functionality will be added to the software, or an insecure component will be modified to make it secure. After the implementation of the security requirements, test cases should be created to validate the security functionality or ensure that no vulnerabilities are present.

The main problem of security requirements arises from the sheer complexity of all the activities that need to be performed. The security requirement elicitation step is non-trivial, as project managers and stakeholders need to communicate and apply security principles to derive a set of security requirements that are suitable for the application they are trying to develop. This is mainly because development organizations are failing to apply security recommendations due to resistance of new processes and that software engineers and developers are hesitant to accept that their software is vulnerable to security flaws [6]. In the paper by Parveen et. al, they describe some of the issues related to software security requirement issues, such as access control, auditability, privacy, integrity, availability, and more [7]. Existing tools to manage these as well as the security activities described above exist in practice, such as GitHub [8], Gitlab [9], and JIRA [10]. All these tools have integrations which allow them to perform some of the security activities described above and are considered the industry standard for software development. However, these tools are centralized, and suffer from numerous issues, such as the use of a centralized authority, a single point of failure, lack of data ownership, and limited availability. These industry standard tools are targets for data-breaches and DDoS attacks, which are not suitable for large development teams that rely on these tools [11]. Importance should be placed on facilitating the secure software development lifecycle with tools and mechanisms that do not put the organization’s data and work at risk, while also providing an environment that is trustworthy and has integrity.

Other than the centralized tools presented above, Ramachandran proposed utilizing a cloud computing service to manage the complex security requirements activities described above [12]. However, their approach is limited to a very rigid security methodology and is not flexible to handle large scale software development.

2.2 Security Requirement Identification

In the paper by Haley et. Al [13], they provide a framework for security requirements identification that is comprehensive and generates detailed security requirements for domain specific applications. This security requirement identification approach allows for the developers to identify implied security requirements depending on the high-level requirement. For example, identifying all the underlying security requirements for a password reset component, such as the proper storage and encryption of the data, the security of sending the password reset mechanism would be an example of this.

The paper shows that security requirements must satisfy three criteria:

- *Definition:* What the security requirement is.
- *Assumptions:* Any implicit or explicit assumptions that the analyst makes about the behavior of components on the system.
- *Satisfaction:* Whether the identified security requirements satisfy the security goals established.

The method for security requirements identification according to the paper by Haley et. al [13] provides a good foundation for a security traceability model to ensure the completion of secure, vulnerability-free software. Below is a brief overview of the steps taken for the security requirements framework in the paper written by Haley et. al [13].

Stage 1: Identifying Functional Requirements

In order to begin the creation of security requirements, a representation of the system context must exist, which means that the functional requirements must be completed [13]. Since security requirements are constraints on existing requirements, functional requirements must exist to identify what security requirement is mapped to the relevant functional requirement. In addition, the elicitation of these functional requirements is agnostic from the technique used.

Stage 2: Identifying Security Goals

2.1) Identify Candidate Assets

Identify assets in the system that have value and record them.

2.2) Selecting Management Principles

First, identify the assets that are worth protecting and determine all possible threat scenarios on that asset. Then select management principles that allow for the protection of those assets and the mitigation of damage assuming that the asset is compromised. For security risk analysis, CORAS models can be used to represent the risk management for a particular asset [14] [15]. CORAS is a model-driven approach for risk analysis which consists of a graphical modeling language and a method for the generation and evaluation of models. The main benefit for using CORAS in this type of framework is because it is an asset-driven approach.

For threat analysis on identified assets, STRIDE is typically used for the identification of security threats [15] [16]. STRIDE stands for “Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege.” STRIDE can capture common attack patterns from malicious actors, allowing organizations to predict the behavior that these identified assets may face in production. CAPEC can also be used for threat modeling, which is a database that contains many attack patterns. CAPEC stands for the Common Attack Pattern Enumeration and Classification [17].

Another example of threat risk driven analysis is provided by a paper by Qian et. al, which proposes a threat risk driven analysis based on OWASP threats [18]. OWASP stands for the Open Web Application Security Project and defines the top ten risks for mobile development, web applications, and many more [3]. By using the OWASP top ten to specify security requirements, the design and implementation phases become easier to manage, as mitigation for vulnerabilities are noticeably clearer and more precise, eliminating and mitigating damage against potential vulnerabilities.

Stage 3) Identify Security Requirements

In this stage, the security goals generated earlier will be applied to the functional requirements which will generate the security constraint on the system.

Stage 4) Verification of Security Requirements

To perform verification on the security requirements, the paper uses a two-part satisfaction argument to verify that the generated security requirements satisfy the security goals described earlier [13]. The outer portion of the argument consists of a formal argument that proves that the instance of the system satisfies its security requirements assuming that the system context is correct, and the implementation will not incur any conflicting behavior. The inner argument consists of several structured informal arguments that support the assumptions made in the outer argument regarding behavior and system composition. This is an iterative process, which will increase the detail and quality of each generated security requirement through each iteration.

2.3 Blockchain Technology

The information pertaining to the security requirements and testing information will be stored on a blockchain, specifically a private blockchain. A blockchain is a ledger consisting of several blocks, each of which are linked using cryptographic methods. Blockchain was first implemented by an anonymous individual named Satoshi Nakamoto detailing a new currency called Bitcoin [1]. In a blockchain implementation, the entire blockchain network must come to a consensus on whether to add a new block to the blockchain. Due to the nature of the cryptographic link, it is extremely difficult to change the previous blocks in the chain, making the blockchain history immutable. Blockchain has been used in software development before, notably by several papers by Jinal et. Yau which are used for trusted coordination during software development and the testing of software for collaborative software development [19] [20]. However, the use of blockchain in software development is still very immature and more research needs to be done to fully realize the potential that blockchain technology has to play in software development overall.

CHAPTER 3

CURRENT STATE-OF-THE-ART

The current state-of-the-art for security requirement management and traceability solutions are centralized repository software like GitHub [8], JIRA [10], and GitLab [9]. Tools like these allow developers to trace software requirements throughout the entire development lifecycle and use version control to update software requirement information. It is through this version control functionality that developers can see the history of a given requirement over time. GitHub and GitLab both have DevSecOps integrations that somewhat automate performing the testing of security requirement activities [21] [22]. Specifically pertaining to security requirements, OWASP [3] has created a tool called “SecurityRAT” which stands for Security Requirement Automation Tool [23]. The aim of SecurityRAT is to simplify and minimize the amount of effort spent performing requirement management during development by using automation and existing resources to avoid redundant security requirement generation [23]. SecurityRAT works by having the developer provide what kind of software is being developed, such as a mobile application, web application, etc. Based on the type of software being developed, SecurityRAT will use the OWASP ASVS, which is the OWASP Application Security Verification Standard, which provides a list of security requirements depending on the type of application being developed [5]. A developer can also opt to use an internally provided set of security requirements to fit their risk profile obtained via threat modeling. An example of the threat model from OWASP is provided below to demonstrate what information is generated.

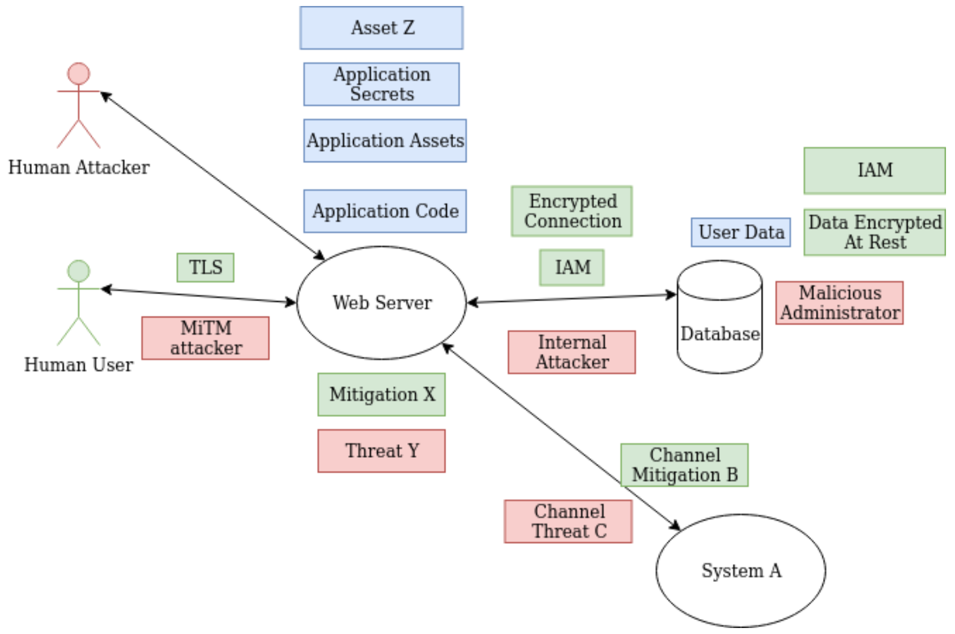


Figure 1: Threat Model Diagram from OWASP [24]

From there, SecurityRAT will generate the security requirements that the software must fulfill, and it will then send those to a requirement tracker, usually GitHub, GitLab or JIRA in this case. As development proceeds, developers can create tickets and update the artifact state of the requirement in the issue tracker and document the relevant changes as needed [23]. It is important to note for security requirement identification for domain specific applications, the current state of the art is the approach by Haley et. al [13].

The issue with tools like GitHub, GitLab, and JIRA being used for large scale software development is that they are centralized, and are not suitable for distributed, decentralized development environments. These tools suffer from numerous issues, such as the use of a centralized authority, a single point of failure, lack of data ownership, and limited availability since they are centralized services. It is also evident that due to their prevalence in the software development domain, they are subjects of DDoS attacks and data breaches which are not suitable for large scale organizations where availability of tools and services is crucial [11]. In addition, all security requirement information can be modified due to the lack of immutability of data in a centralized service. In a distributed, decentralized team, team policies can vary differently, and a global shared development service may not work, which is why I propose a decentralized security requirement manager to ensure that the security aspect of large-scale development projects are transparent, immutable, and auditable.

CHAPTER 4

OVERALL APPROACH

In this chapter, the overall approach for blockchain-based security requirement governance in a large-scale development context is provided. The overall approach consists of five steps – the initialization of the blockchain infrastructure, the mapping of security requirements and open-source software to respective software components, the implementation of requirements in code, ensuring that the testing criteria for security requirements are satisfied, and system-wide testing and consensus. This approach is based on the foundation provided by the paper written by Jinal et. Yau [20] but applied in a different context. In this case, the context is software security requirements as opposed to trusted coordination.

This blockchain-based approach relies on the usage of a permissioned or private blockchain instance being used to facilitate the communication between the distributed development teams as well as govern the security requirements set forth at the beginning of the large-scale software development cycle [1]. This private or permissioned blockchain instance will essentially be a representation of the state of all security requirements for the large-scale distributed development project at any given moment. All transactions between the distributed development teams related to the governance of security requirements will take place on this private or permissioned blockchain instance. During the initialization of the blockchain instance, every development team on the large-scale development project is assigned a node on the blockchain instance. A single node on the blockchain, which is assigned to the management team of the large-scale software development project, will be designated authority over the blockchain and be responsible for the assignment of software components to their respective development teams. In addition, for each development team that has been assigned a node on the blockchain instance, a single individual per each node will be assigned the role of 'Administrator.' This administrator role will be able to perform actions related to security requirements governance for each development team on the project. The reason a private blockchain is used is due to the following reasons: all nodes in the private blockchain instance are pre-authenticated and verified, private blockchain instances have high transaction rates, private blockchains have high fault tolerance and performance, private blockchain instances support non-repudiation, privacy, and integrity, and that private blockchains provide access control that are suitable for large scale software development projects. This private blockchain instance will use a pBFT (Practical Byzantine Fault Tolerance) consensus algorithm to validate transactions during the final phases of development [25].

The usage of this approach assumes that the security requirements for the large-scale project have already been generated prior to the initialization of the private or permissioned blockchain instance. The identification of these requirements can be done in several ways, such as manual security requirements identification as described in the paper [13], or via pre-generated requirements from a security requirements repository such as the OWASP ASVS [5]. Both approaches can generate detailed security requirements that can be implemented, as they include the necessary detail that should be present after the requirement of software development. For the design phase, I refer to the OWASP Application Security Fragmentation integration standards for the appropriate security activities [24]. This approach will work agnostic of the security requirements elicitation technique that is employed, if the testing criteria for each respective security requirement is able to be well defined. The appropriate detail for all steps including sub steps is outlined below.

Symbol	Meaning
t	A team in the large-scale development project
T	The set of all teams in the large-scale development project
n	A node in the large-scale development project
N	The set of all nodes in the private or permissioned blockchain instance
r	A requirement for the large-scale software development project
R	The complete set of requirements for the large-scale development project.
c	A software component in the large-scale software development project
C	The set of software components in the large-scale software development project
S	A software component holding requirement information and open-source dependencies.
O	The complete set of open-source dependencies
o	An open-source dependency.

Table 1: Notation and Symbols for Overall Approach

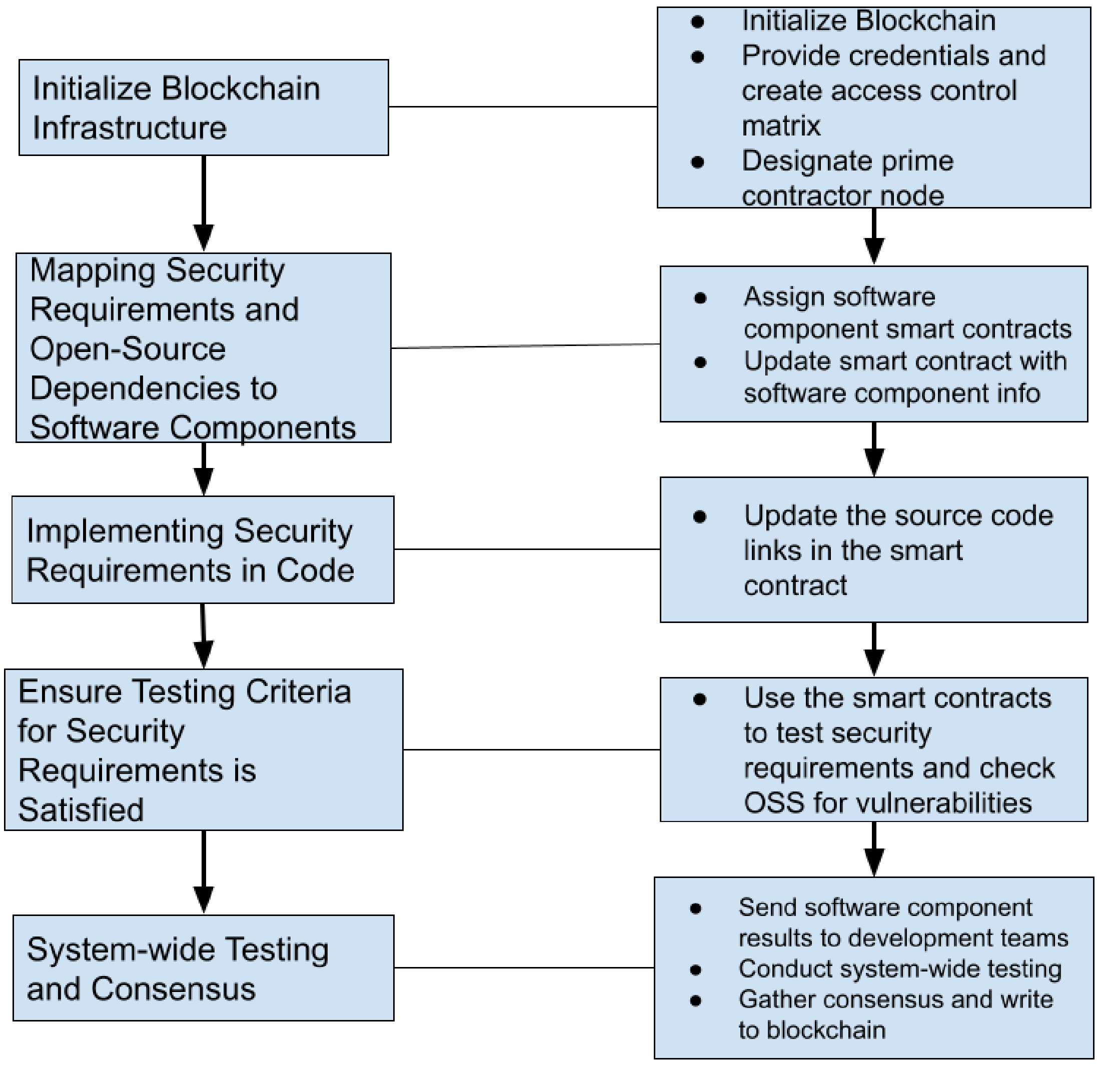

Initialize Blockchain Infrastructure
Mapping Security Requirements and Open-Source Dependencies to Software Components
Implementing Security Requirements in Code
Ensure Testing Criteria for Security Requirements is Satisfied
System-wide Testing and Consensus
● Initialize Blockchain
● Provide credentials and create access control matrix
● Designate prime contractor node
● Assign software component smart contracts
● Update smart contract with software component info
● Update the source code links in the smart contract
● Use the smart contracts to test security requirements and check OSS for vulnerabilities
● Send software component results to development teams
● Conduct system-wide testing
● Gather consensus and write to blockchain

Figure 2: The Overall Approach

Step 1: Initializing the Blockchain Infrastructure

Step 1.1) Private/Permissioned Blockchain Instance is Initialized: For $\forall t \in T$, a permissioned or private blockchain instance is created. The private or permissioned blockchain will be created with a pBFT (Practical Byzantine Fault Tolerance) consensus model [25] to validate transaction ordering and acceptance criteria for software component results from the nodes on the blockchain instance.

Step 1.2) Every Team on the Large-Scale Software Development Project is Assigned a Node on the Blockchain: For $\forall t \in T$, $\forall t$ is assigned a node n on the private or permissioned blockchain instance.

Step 1.3) A Single Node is Designated Authority over Blockchain Instance: We designate a single node $n \in N$ authority over the blockchain instance. This team will be considered as the management node which will designate responsibility of software components for $\forall n \in N$.

Step 1.4) An Administrator is Assigned for Every Team: For $\forall t \in T$, a single member of each $t \in T$ is given the role of 'Administrator' to handle updating and the modification of requirements information, as well as handling the communication between other teams in the large-scale software development project.

Step 1.5) All Elicited Security Requirements Must Be Stored: $\forall r \in R$ must have a complete set of information based on the fields specified. A table of all the fields is provided below.

Requirement Field	**Purpose**
Requirement ID	A unique identifier for each requirement
Requirement Owner	The owner of a given requirement
Requirement Description	A description of what the security requirement is
Implementation Specifics	Code that defines the threat modeling for the security requirement itself as well as a general system architecture. Also can include security requirement dependencies.
QA	The type of QA that is required to test the requirement functionality (e.g., Black-box testing, White-box testing, Functional testing)
Requirement Type	The type of the requirement (Functional requirement, Performance requirement, technical requirement, specification)
Field for Source Code Link	A link to a function or piece of source code
Testing Criteria	The testing criteria needed to validate that the requirement is met.
Requirement Satisfaction	A field that indicates whether the requirement is met.

Table 2: Requirement Field Purposes

Step 2) Tracing the Activities in the Design Phase

Step 2.1) Performing the activities for Security Requirements for the Design Phase: This step follows the OWASP Application Security Fragmentation integration standards [24]. For $\forall r \in R$, each security requirement will need to be given implementation specifics. This means providing detailed threat modeling information as well as providing a general system architecture for the requirement by using tools that allow developers to represent this information in code like PyTM [26] and ThreatSpec [27]. Generating the implementation specifics entails several things, the main components of which are a diagram of the data flow for the given requirement as well as a general system architecture that will apply to the given requirement [24]. The generation of these implementation specifics gives the development teams the ability to perform comprehensive threat modeling and apply all relevant security desirables to the specific requirement. Typically, this can be done by performing threat modeling using code by utilizing tools like PyTM [26] and ThreatSpec [27]. PyTM and ThreatSpec are both Python libraries that allow developers to generate threat modeling diagrams and data flow diagrams in code automatically. To apply the design phase in this step, developers must create their data flow diagrams using conventional code as stated above and store them in some form in the 'Implementation Specifics' field in the requirement field. As the security requirement threat modeling and system architecture is iterated upon in code, the field must continue to be updated by developers to reflect the changes. As the field continues to get updated, the changes will be reflected in the blockchain infrastructure. Because the changes are reflected in the blockchain infrastructure, the security requirement’s entire history during this step will be viewable by the relevant development team. It is left up to the discretion of the development team when the security requirement is sufficiently described from the design phase to proceed onto the development phase.

Step 2.2) Establish Communication Channels: For $\forall t \in T$, communication channels are established between $\forall t \in T$.

Step 2.3) Assigning Specific Requirements to the Respective Software Component: $\forall r \in R$ will be assigned to S, which is the set of requirements for a given software component. The r that is assigned to S must be relevant to the software component that it belongs to.

Step 2.4) Assigning Specific Open-Source Software to the Respective Software Component: $\forall o \in O$ will be assigned to S, which is the set of open-source dependencies for a given software component. The o that is assigned to S must be relevant to the software component that it belongs to. The completed software component is then added to the blockchain as a smart contract with its assigned development team.

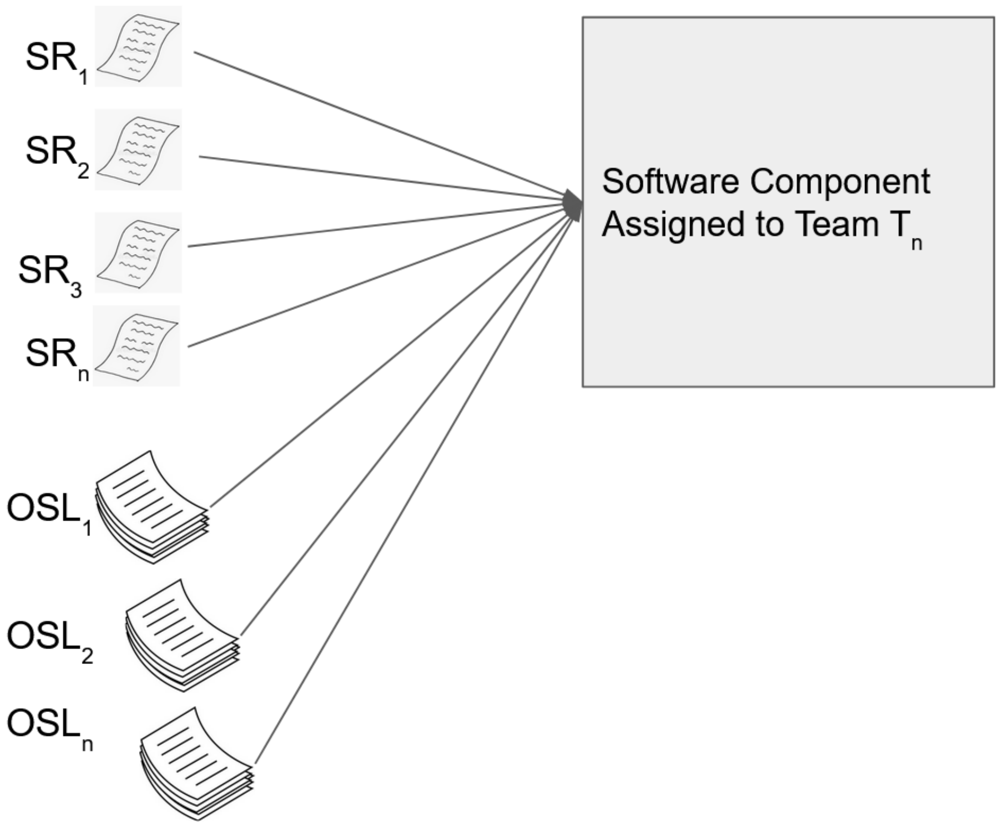


Figure 3: Security Requirement Mapping to Software Component

Step 3) Tracing the Implementation Phase

Step 3.1) Linking Source Code to Security Requirements: $\forall r \in R$, the source code link will be filled with the appropriate link to the source code in the large-scale software development project by updating the information in the respective smart contract. The main benefit of this step is to simplify security requirement traceability in subsequent steps.

Step 3.2) Assigning Specific Open-Source Versioning: $\forall o \in O$, open-source software dependencies will be assigned the specific versioning that developers intend to use through the software development life cycle. This is done so that at a later step, specific versioning for all open-source dependencies will be checked for vulnerabilities.

Step 3.3) Modifying, Updating, and Auditing Security Requirement Information: Throughout the duration of this step, software development teams in the large-scale collaborative project will be able to modify core security requirements information (given that the consensus is made for the change), audit security requirements, and update security requirements information. These activities are like the activities in [20], but applied to a different context, in this case software requirements.

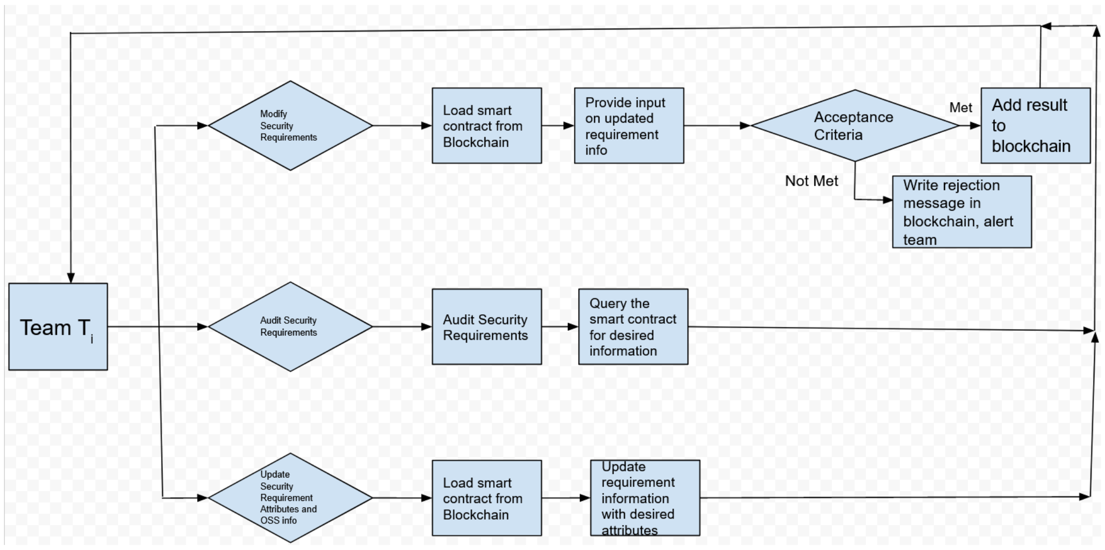

Team T_i
Modify Security Requirements
Load smart contract from Blockchain
Provide input on updated requirement info
Acceptance Criteria
Met
Add result to blockchain
Not Met
Write rejection message in blockchain, alert team
Audit Security Requirements
Audit Security Requirements
Query the smart contract for desired information
Update Security Requirement Attributes and OSS info
Load smart contract from Blockchain
Update requirement information with desired attributes

Figure 4: Tracing in the Implementation Phase

Step 4) Tracing the Security Testing Activities

Step 4.1) Conducting Automated and Manual Tests for Security Requirements: $\forall r \in R$, the requirements are checked to see if the testing criteria defined at the requirement's birth is satisfied. Since some requirements can be automatically tested, it is expected that testers will be responsible for writing and defining the tests, whether it be unit tests, fuzzing, or any other type of automated analysis for security functionality. For tests that are unable to be automated, white-box testing is used to ensure security functionality works properly. These security requirement test results and automated testing mechanisms can be embedded in the appropriate smart contracts for each security requirement.

Step 4.2) Performing Origin Analysis/Composition Analysis: $\forall o \in O$, specific open-source library versions are checked to ensure that no vulnerabilities are present. This is done by analyzing the relevant vulnerability repositories such as the MITRE CVE vulnerability repository [28].

Step 5) Software Component Result Submission and Consensus

Step 5.1) Submitting the Software Component Result: Once Step 4.1 and 4.2 are completed, the development team will submit the software component result which contains all open-source software vulnerability information as well as the satisfaction status for all security requirements in the software component. At submission time, the automated tests will execute to ensure that all security requirements pass their testing criteria. For any changes that need to be made, automated test execution will be crucial in ensuring that any future software component result submissions are expeditious.

Step 5.2) Gathering Software Component Consensus: After the software component result is checked to ensure that all testing criteria are met and that any open-source software does not contain any vulnerabilities, each software component is reviewed by $\forall t \in T$ for system-wide testing. If the software component is deemed non-defective, and the consensus is gathered among $\forall t \in T$, the software component result is then written to the blockchain.

CHAPTER 5

TRACING IN THE DESIGN PHASE

After the initialization of the blockchain infrastructure, developers will be able to trace the results of the security requirements during the design phase. During this phase, the developers will be responsible for building security into the application design, ensuring that the application will not contain any security vulnerabilities at the conclusion of the development cycle. According to OWASP, this is done by performing three activities, outlining data flows, generating a general system architecture, and performing threat modeling as well as attaching general security considerations to each security requirement [24]. In this phase, the information provided by the developer consists of the results generated by the activities in the design phase, such as the threat modeling representations, the overall architecture of the software development project, and any other important security information generated in the design phase. In my approach, the tracing of these activities and data is done by storing the relevant threat modeling data and architecture data in the ‘Implementation Specifics’ field. Using the open-source tools recommended by OWASP [24], such as PyTM [26] and ThreatSpec [27], code can be generated that represents the threat model of the overall application as well as the general system architecture and other security considerations that may be needed to ensure that the security requirement is able to be satisfied. During the design phase, as developers continue to update the representations of the threat model, the relevant ‘Implementation Specifics’ field will be updated in the blockchain infrastructure. During the same phase or during later phases of the software development lifecycle, developers will be able to query the blockchain to audit the results of the security activities that were performed during the design phase. This is done by a function that is written in the software component smart contract that will allow the developers to read the changes made to the representation generated by any of the threat modeling tools or other representations that may be used to show the advancement of the results of the design phase security activities. The goal of this phase is to generate the security technical specifications and the plans to ensure that the security requirements are understandable and able to be implemented and tested in the later development phases [29]. To ensure this happens, the traceability that is introduced in the design phase is crucial to understand the considerations that were made before the implementation of the security requirement.

CHAPTER 6

TRACING IN THE IMPLMENTATION PHASE

During the implementation of the security requirements in my approach, developers will be able to perform several activities to ensure the completion of secure code with the security requirements set forth at the beginning of the development cycle. In this chapter, I will provide some additional detail on step three of my approach, as well as the results of the step and the information that is supposed to be generated. I will provide detail for each sub step of my approach and explain exactly what it entails as well as how it will aid in the completion of secure software. In this phase the information that is used is provided by the developer which consists of the source code of the software development project.

The first sub step of step three is the activity of the developer to link source code to the security requirement. For each smart contract where the security requirements are encoded, there exists a field which allows developers to link a source code location to a security requirement. The rationale for why I do this is for two reasons, the first of which is that it will allow for the effective traceability of the security requirement in code as the relevant source code location scope increases over the software development lifecycle. The second reason is that it will allow for the faster creation of the tests for the security requirement in step four as the development team will easily be able to identify the relevant code segment to test for the security requirement.

The second sub step of step three details how the developers must assign specific open-source dependency versioning that developers intend to use during the development phase of the software development life cycle. This will entail marking very specific open-source versions and recording them in the smart contract for the relevant software component. The rationale for why this is done is because development teams will want to avoid using vulnerable open-source software in the completed build of their software. To mitigate this, any time any open-source software dependencies are introduced into the developed software, they are recorded and checked at a later step to ensure that there are no inherited vulnerabilities from open-source software in the final build.

The third and final sub step of step three is the development team activities of being able to modify, update, and audit every security requirement in the software component smart contract assigned to them. For the modification of core security requirement information, the development team must go through the process of obtaining consensus for the change as we assume that after the design phase the security requirement is complete enough for development.

By obtaining consensus from the blockchain network, it is ensured that the developed software is still secure and that any dependencies on the modified security requirement are still satisfied. For the updating of security requirement attributes, this activity entails the development team updating each security requirement field that needs to be updated during step three of my approach. This will mainly involve updating three fields: the 'QA' field, the 'Source Code Link' Field, and the 'Testing Criteria' field. Depending on the type of security requirement designated at the requirement phase, the QA field might be changed during the implementation phase. As the code written during the implementation phase continues to expand and grow, the software component smart contract will continue to be updated with the updated source code link. For the testing criteria field, towards the end of the completion of the implementation phase, detailed testing criteria will be written that is tailored to the written code in the source code link field. This will allow for developers to have exotic testing mechanisms that will are designed to vigorously test security functionality in their software, allowing for the completion of secure software. As these fields continue to be changed throughout the implementation phase of development, developers will be able to see these changes reflected on the blockchain infrastructure, with blockchain-enforced immutability enhancing security requirement traceability.

The results of this step are achieved at the end of the implementation phase, where there will be a set of software components that will have all their security requirements with a filled source code link, a detailed testing criteria tailored towards the specific code that the requirement is linked to, the specific open-source versioning that has been used in the software component, as well the information history of the security requirement that has been created during the implementation phase reflected in the blockchain infrastructure. All this information generated during the implementation phase in my approach creates a good environment for step four, which involves the testing of all these security requirements using the detailed tests that have been written during the implementation phase. During the next step, the written tests that have been defined in the testing criteria will be executed using the blockchain infrastructure, and the open-source versioning gathered during this step will be checked against vulnerability repositories to ensure they are vulnerability-free.

CHAPTER 7

TRACING IN THE TESTING PHASE

During the testing phase in my approach, developers will be able to automatically test security requirements using the smart contract executability functionality that is inherent from using a blockchain infrastructure. This will allow development teams in the large-scale development project to create secure software quicker, and more efficiently. In this chapter, I will provide some additional detail on the sub steps of step four, as well as the overall result that is generated after the successful completion of this step. The information provided by the developer in this phase consists of the source code and the automated tests to test every security requirement. This includes the testing mechanisms for integration testing, component testing, unit testing, and any other additional security testing mechanisms.

In the first sub step of step four, my approach will check each security requirement to ensure that the testing criteria defined at the requirement’s birth which has been iterated on over the development cycle, has been satisfied. It is expected that every security requirement that has been encoded in every software component smart contract cannot be automatically tested, so for security requirements that cannot be automatically tested, developers and testers must ensure that the proper testing mechanisms and formalisms are conducted to ensure that those security requirements are satisfied. For the security requirements that can be automatically tested and have been provided the relevant functionality to test, the relevant tests can be encoded in the software component smart contract and be executed at any point, usually whenever a change is made in the functionality or before the submission of a software component. Because we allow any kind of software security test to be encoded in the software component smart contract, development teams in the large-scale development project can execute exotic tests that are tailored specifically to the code written for the security requirement as well as the security requirement description itself. These tests may include unit tests, fuzzing, or any other type of automated analysis security testing that will ensure the satisfaction of security requirements. As the testing process proceeds through in the software development life cycle, the results of these security requirement tests will be stored in the blockchain infrastructure, and they will be accessible by the development teams in the software development project. This will ensure traceability for all testing for every security requirement encoded in the software component smart contract. If the security requirement test succeeds, then the security requirement satisfaction field will be updated to reflect that. If not, the field will be updated to reflect as such.

For the second sub step of step four, origin and composition analysis will be performed for every software component with the open-source information encoded in each smart contract. This can be done automatically by analyzing vulnerability repositories such as the MITRE CVE vulnerability repository to ensure that the open-source dependencies are vulnerability free [28].

The results of this step are achieved at the end of the testing phase, where there will be a set of complete software components that will have all their security requirements with a complete, executable testing criteria field that will return a result of the security requirement satisfaction. Every security requirement encoded in the software component smart contracts will have their satisfaction field updated to reflect that the security requirement has passed the testing phase, and the satisfaction history will be viewable and auditable as it is stored in the blockchain infrastructure. In the next step, which involves the submission of the software component for consensus, these automated security testing mechanisms will be crucial in accelerating the consensus gathering, as it will be evident that the software component is vulnerability free, and the developed software has already satisfied every security requirement. In addition, developers can trust that the blockchain infrastructure has sufficiently tested the software that their components rely on, making the system-wide testing phase efficient and trustworthy.

CHAPTER 8

INNOVATION

In this chapter, the innovations of each step in the approach described above will be detailed. In the approach, a private blockchain network is applied to a security requirements traceability framework for security requirements to be traced throughout the software development life cycle, which is a part of the broader innovation of my approach.

A. Initialization of Blockchain Infrastructure

The first innovation in my approach will be found in step 1, where the initialization of the blockchain infrastructure will allow for security requirement traceability throughout the entire life of the requirement during the software development lifecycle with complete immutability of the security requirement history. In addition, the additional fields that have been given to each security requirement will ensure that the prerequisite security requirement information is complete, and that each security requirement is verbose enough to ensure that it is implemented correctly. Fields like the source code links, testing criteria, and other requirement fields will help ensure all aspects of the security requirement is reflected in the blockchain infrastructure and are used in later development phases for security requirement satisfaction. In addition, the initialization of this private blockchain infrastructure creates a privacy-centric development environment where teams can operate individually depending on internal policies for the satisfaction of security requirements. This is one of the major innovations of this approach as it greatly aids in the development of large-scale software in distributed development environments.

B. Modifying Security Requirements Information

The major innovation in step 3 is that development teams in the large-scale development project will be allowed to make changes to existing security requirements, add additional security requirements, and retrieve information about existing security requirements while always having their changes reflected in the blockchain infrastructure. This is done by loading smart contracts from the blockchain and updating the information with the desired attributes. For any core security requirement modification, the acceptance criteria need to be met by the prime contractor team for the change to occur. This reflection of changes on the blockchain infrastructure creates a development environment for security requirements that ensures full traceability for all security requirements which is both visible and transparent.

C. Tracing in the Testing Phase

The innovation in step 4 is the ability for development teams in distributed development environments to perform automated security requirement testing using the existing smart contracts on the blockchain infrastructure. By allowing developers to update the testing criteria defined at the security requirement's birth, developers can write tests that can automatically be tested via an executable smart contract. The main difference between the testing performed in this approach compared to existing DevSecOps [30] solutions is that in this approach, the testing result history for all security requirements is immutable and all the history for a security requirement is easily viewable and auditable. This is especially important for large scale software development projects where there may be hundreds or thousands of security requirements. This innovation allows security requirement testing to be conducted and automated en masse while also maintaining completeness and transparency in security requirement traceability.

Some of the other innovations in my approach arise from the nature of using blockchain technology in any sort of environment. Due to the nature of how blockchain functions, transparency, auditability, non-repudiation, and consensus protocols are introduced into the security requirements management process. This creates a more trustworthy development environment, which is needed especially in distributed development teams tackling large-scale software development projects. In addition, a decentralized solution that utilizes blockchain does not suffer from the same issues that centralized solutions do, such as a single point of failure, limitations of a central authority, and a lack of immutability for security requirement history.

CHAPTER 9

AN ILLUSTRATIVE EXAMPLE

To illustrate the approach and advantages for blockchain-based security requirements traceability for large scale software development, I will be using a secure banking system as an example. In this example, I assume that the security requirements for the secure banking system have already been generated. In addition, all security requirements that are used in this example have been taken from the OWASP Application Security Verification Standard [5]. The OWASP Application Security Verification standard focuses on mitigating security issues with application security, such as buffer overflows, program misuse, input sanitization, file integrity, etc. In addition, I will be using the Hyperledger Blockchain platform to illustrate the example [31].

Step 1: Initializing the Blockchain Infrastructure

Step 1.1) Private/Permissioned Blockchain Instance is Initialized: For all the teams of the large-scale collaborative software development project, in this case a secure banking system, a private or permissioned blockchain instance is created to support the storage and communication between teams for the governance of security requirements. The private or permissioned blockchain will use a pBFT consensus algorithm for the validation of transaction ordering and acceptance criteria for the software component results which hold the security requirements information for the large-scale software development project [25].

Step 1.2) Every Team on the Large-Scale Software Development Project is Assigned a Node on the Blockchain: In order to ensure that all development teams of the secure banking system can participate in the governance process for security requirements, all teams will be assigned a node on the blockchain. In addition, the responsibility of ensuring proper access control mechanisms is left to the organization to ensure that all development teams can view and modify only the information that they are authorized to. For the sake of this example, we assume that there are 10 development teams on the large-scale collaborative software development project (T_1, T_2, T_3, $T_{4\ldots,}$ T_{10}), and that each team has been assigned a node on the blockchain.

Step 1.3) A Single Node is Designated Authority over Blockchain Instance: In this step, it is imperative that a single node in the blockchain instance is designated as the prime contractor team, such that it is given authority over the entire blockchain instance [20]. This prime contractor team will be responsible for designating responsibility of all software components to teams T_1 through T_{10} where T_1 will be the prime contractor team. This prime contractor team will serve as an administrative entity for the entire private blockchain network.

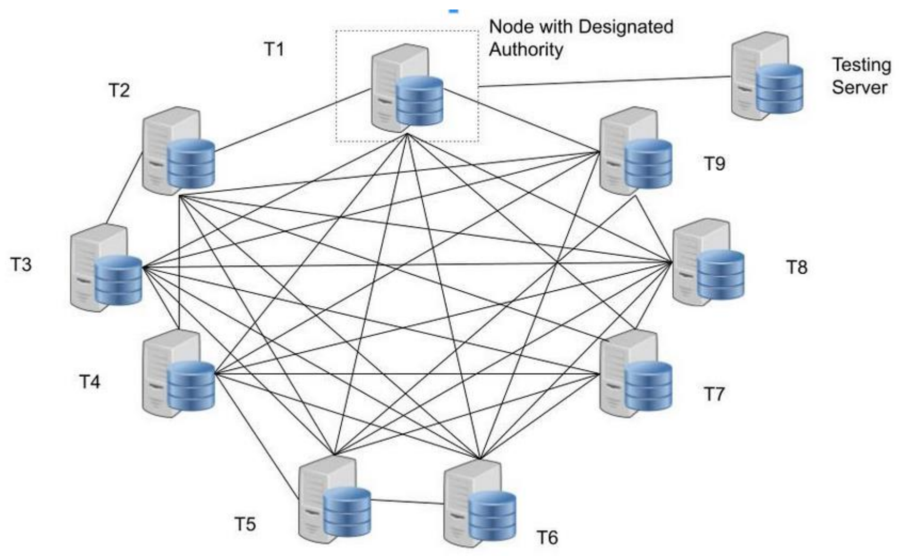


Figure 5: The Private Blockchain Network

Step 1.4) An Administrator is Assigned for Every Team: A single member of T_1 through T_{10} will be assigned the role of 'Administrator' to handle the updating and modification of security requirements information. In addition, this role will be responsible for handling communication between all other teams in the large-scale collaborative software development project.

Step 1.5) All Elicited Security Requirements Must Be Stored: All elicited security requirements will have a complete set of information based on the table provided in a previous section. An example of a final security requirement is provided in the table below. Note that some fields are left as "N/A" because at this current step in the approach, they should not be complete.

Requirement ID	0123456
Requirement Owner	Project Lead
Requirement Description	“Verify that 2 factor authentication endpoints are not brute forceable.”
QA	Blackbox, Whitebox, Functional Test
Implementation Specifics	N/A
Requirement Type	Design, Technical
Field for Source Code Link	N/A
Testing Criteria	“2FA endpoint APIs are rate limited.”
Requirement Satisfaction	No

Table 3: An Example Requirement with Attributes

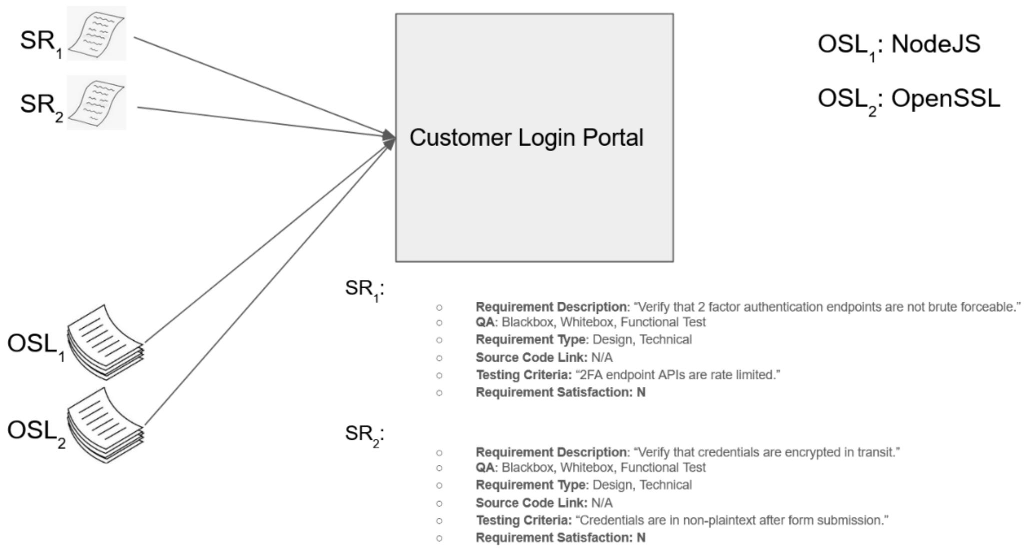
Step 2) Tracing the Design Phase Activities

Step 2.1) Performing the Activities for Security Requirements for the Design Phase: This step follows the OWASP Application Security Fragmentation Integration Standards [24]. During this step, the implementation specifics will be given for all the security requirements before they are assigned to the development teams before the development phase. This is done by performing threat modeling and generating data flow diagrams as well as the general system architecture using tools like PyTM [26] and ThreatSpec [27]. Development teams will update the security requirement dependencies, threat modeling representations, and system architecture by creating the code that will be represented in some form in the 'Implementation Specifics' for each security requirement. As these representations continue to be iterated upon, the changes will be reflected in the blockchain, and their history will be traceable by the relevant development team. The results and output of this phase consists of the security technical specifications and the plans on how to implement the security requirement in the implementation phase. The threat modeling activities will have the developers decompose the application, categorize the threats that the application may face, ranking the threats, and figuring out how the threats can be mitigated [29]. The PyTM [26] and ThreatSpec [27] representations will be stored in the ‘Implementation Specifics’ field for each security requirement in the smart contract software components stored in the blockchain infrastructure.

Step 2.2) Establish Communication Channels: Between T_1 through T_{10}, communication channels are established.

Step 2.3) Assigning Specific Requirements to the Respective Software Component: For the sake of this example, there will be nine software components, S_1 through S_9. All elicited security requirements will be assigned to their respective software components. For the sake of simplicity in this example, we will only look at a specific software component, S_1, assigned to T_2.

Step 2.4) Assigning Specific Open-Source Software to the Respective Software Component: At this stage in development, the distributed development teams will identify the open-source software that is needed for the completion of their respective software component. All identified open-source software will then be assigned to their respective software components. At this point, the software component is written to the blockchain as a smart contract. An example of what a software component will look like at the end of this step is provided below. In addition, software components are then assigned to their respective development teams for fulfillment. The result of this step is the history of all the design phase security activities and the preliminary software component information.



SR$_1$:

- **Requirement Description**: “Verify that 2 factor authentication endpoints are not brute forceable.”
- **QA**: Blackbox, Whitebox, Functional Test
- **Requirement Type**: Design, Technical
- **Source Code Link:** N/A
- **Testing Criteria:** “2FA endpoint APIs are rate limited.”
- **Requirement Satisfaction: N**

SR$_2$:

- **Requirement Description**: “Verify that credentials are encrypted in transit.”
- **QA**: Blackbox, Whitebox, Functional Test
- **Requirement Type**: Design, Technical
- **Source Code Link:** N/A
- **Testing Criteria:** “Credentials are in non-plaintext after form submission.”
- **Requirement Satisfaction: N**

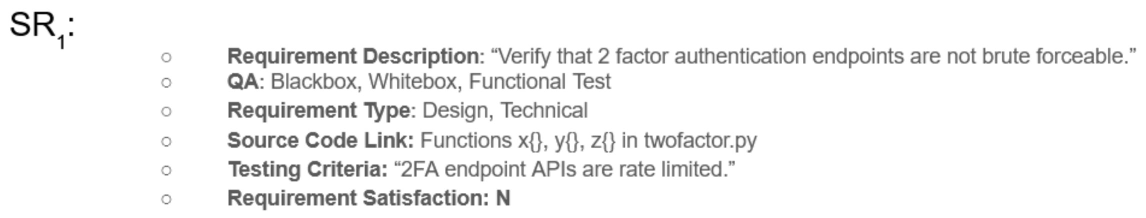
Figure 6: Customer Login Portal of Illustrated Example

Team	**Component Assignment**
T1	Customer Login Portal
T2	Transaction
T3	Backend
T4	Help and Support
T5	Frontend
T6	Service Integration
T7	Example Component
T8	Example Component
T9	Example Component

Table 5: Component Assignment to Nodes on Development Team

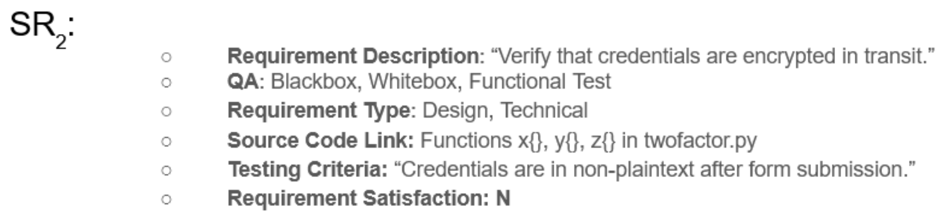
Step 3) Tracing Implementation Phase

Step 3.1) During the implementation of security requirements in code, the source code link for every security requirement must be created by updating the respective smart contract holding the desired security requirements information. In this example, this means updating the source code links of SR$_1$ and SR$_2$, where they will be linked to functions x, y, and z in twofactor.py. The attributes of SR1 and SR2 are provided below.



SR$_1$:

- **Requirement Description**: "Verify that 2 factor authentication endpoints are not brute forceable."
- **QA**: Blackbox, Whitebox, Functional Test
- **Requirement Type**: Design, Technical
- **Source Code Link:** Functions x{}, y{}, z{} in twofactor.py
- **Testing Criteria:** "2FA endpoint APIs are rate limited."
- **Requirement Satisfaction: N**



SR$_2$:

- **Requirement Description**: "Verify that credentials are encrypted in transit."
- **QA**: Blackbox, Whitebox, Functional Test
- **Requirement Type**: Design, Technical
- **Source Code Link:** Functions x{}, y{}, z{} in twofactor.py
- **Testing Criteria:** "Credentials are in non-plaintext after form submission."
- **Requirement Satisfaction: N**

Step 3.2) All open-source software dependencies will be assigned the specific versioning that developers intend to use through the software development life cycle. This is done so that at a later step, specific versioning for all open-source dependencies will be checked for vulnerabilities. For the sake of this example, OSL$_1$ and OSL$_2$ are provided the versions 'NodeJS 15.1' and 'OpenSSL 3.0'.

Step 3.3) Throughout the duration of this step, software development teams in the large-scale collaborative project will be able to modify core security requirements information (given that the consensus is made for the change), audit security requirements, and update security requirements information.

Step 4) Tracing the Security Testing Activities

Step 4.1) Conducting Automated and Manual Tests for Security Requirements: In this sub-step, all the security requirements for S_1 through S_9 will need to be checked for satisfaction. As a result, detailed testing criteria will be made for each security requirement in their respective software component. The testers will be responsible for either the creation of a testing server or implementing the software tests in the respective Hyperledger smart contract languages. For the sake of simplicity, we assume that a testing server exists in the development environment and can return the results of security requirement satisfaction based on the requirement ID utilizing fuzzing, automated tests (such as DevSecOps), and other security testing methods discussed earlier. For manual testing, we assume that the testing server will be able to return a result with a requirement after manual review of security requirement satisfaction based on code auditing utilizing the source code links defined earlier. As an example, for SR_1, the testing criteria field is currently set to “2FA endpoint APIs are rate limited”. In this sub-step, the testing criteria will be updated with an API call that gets sent to a testing server that returns the result of the requirement satisfaction based on the requirement ID.

Step 4.2) Performing Origin Analysis/Composition Analysis: In this sub-step, all inherited open-source software recorded in the software components are analyzed for vulnerabilities via executable code in the smart contract. This is done by analyzing the relevant vulnerability repositories such as the MITRE CVE vulnerability repository [28]. In addition, this executable code can conduct fuzzing on open-source components and search for vulnerabilities.

Step 5) Software Component Result Submission and Consensus

Step 5.1) Submitting the Software Component Result: Once Step 4.1 and 4.2 are completed, T_2 through T_9 will submit the software component result which contains all open-source software vulnerability information as well as the satisfaction status for all security requirements in the software component. At submission time, the automated tests contained in the security requirement testing criteria fields will execute to ensure that all security requirements pass their testing criteria.

Step 5.2) Gathering Software Component Consensus: After the software component result is checked to ensure that all testing criteria are met and that any open-source software does not contain any vulnerabilities, each software component is reviewed by teams T_2 through T_9. If the software component is deemed non-defective by the development teams, and the validation consensus is gathered among T_2 through T_9, the software component result is then written to the blockchain. This process is repeated for each software component in the blockchain.

CHAPTER 10

EVALUATION

In this chapter, I will compare my approach to existing centralized approaches for the management and governance for security requirements for large-scale software development projects. It is important to point out that there are no decentralized approaches that involve security requirements management as of writing, so there is no comparison to be made. For this reason, I can only compare my approach to the SecurityRAT [23] tool with the integration into other services like JIRA [10] and GitHub [8].

The centralized approaches like JIRA [10] and GitHub [8] mentioned earlier suffer from the same problems that all centralized solutions do, such as a non-immutable versioning history, a single point of failure, and limitations on data ownership and privacy for development teams. However, some of the advantages of using centralized solutions for security requirements management are that the performance overhead of using a decentralized solution is significantly lessened and the initial setup of a centralized solution is significantly less complex and more efficient. For small-scale software development, it does not make much sense to utilize a blockchain-based system, as the performance overhead is immense, and the infrastructure cost is non-negligible. However, for large-scale development it makes much more sense, as blockchain will aid in the coordination of security requirements as well as the traceability of a large amount of security requirements which may require advanced testing techniques. For testing and implementation specifically, it is difficult to have the exotic security testing approach that my approach has for a centralized solution, because all security testing changes are reflected in the blockchain, and each development team can tailor their security tests to the code that they write. Because each development team in the large-scale development project has a node on the blockchain infrastructure, every one of them can utilize the testing mechanisms and implementation features to ensure the completion of secure software as well as being able to audit and view the security requirement history during the entire software development life cycle. Development teams maintain ownership of the testing mechanisms, as well as the data that is provided by those testing mechanisms, which is one of the best aspects of a decentralized security requirements traceability approach, which is why this aspect is so crucial to making this applicable to large-scale software development.

In my approach, the blockchain infrastructure suffers from a higher performance overhead than a centralized solution due to the consensus protocols that are involved for each decision that a development team makes. In addition, the initial blockchain infrastructure creation is very inefficient compared to a centralized network, as it involves the creation of an entire private blockchain network with the creation of nodes for each development team. In addition, there needs to be significantly more storage in a decentralized solution like my approach due to each copy of the blockchain being stored on each node in the private blockchain network.

The prototype that I have implemented contained three nodes on a private blockchain network, with three smart contracts for three software components. These smart contracts contained dummy requirement information and some executable code that would contact an external API. The time that it took for any information to be written or read was roughly 3 to 5 seconds. However, this is not an accurate representation of my approach, since it only contains three nodes. According to the Hyperledger Foundation, a transaction on a twenty-node private blockchain will take an average of 16 seconds. During my testing, the commit history to GitHub had roughly 0.1 to 2 seconds of latency. This means that the processing speed and responsiveness of my approach on Hyperledger is worse than GitHub but considering the frequency at which the system will be interacted with, an average of 16 seconds is reasonable considering the introduction of immutability, auditability, non-repudiation, and more attributes that will increase the trustworthiness and integrity of the development environment.

While the initial setup for the blockchain infrastructure in my approach is complex and computationally intensive due to the use of a private blockchain instance, the benefits of using a decentralized security requirements platform severely outweigh the negative properties. The negative aspects of a private blockchain based solutions do not outweigh the immutability of security requirements history, the auditability of any security requirement during the software development lifecycle, the non-repudiation characteristics of blockchain, and the automated security requirement testing acceptance criteria. One of the most important parts of my approach is that it encourages a privacy-centric development environment which is ideal for large-scale distributed development environments. Non-repudiation in the private blockchain network will also ensure that data cannot be tampered with and that development teams cannot contest the content or integrity of the data stored in the blockchain. These beneficial properties only arise with some cost to performance and additional costs to infrastructure creation due to the cryptographic properties of the private blockchain infrastructure. In addition, the access control mechanisms provided by Hyperledger Fabric and my approach have parity with the access control mechanisms of the centralized approaches, ensuring that the whole network is secure.

Number	Properties	Centralized Solution	My Approach
1	Initial Setup Cost	Low	High
2	Processing Speed and Responsiveness	High (0.1 – 2 seconds)	Low (16 seconds)
3	Storage Cost	Low	High
4	Immutability	No	Yes
5	Auditability	No	Yes
6	Non-repudiation	No	Yes
7	Consensus Protocols	No	Yes

Table 5: Comparison of Approach to Centralized Solutions

The table shown above shows the most important properties of my approach, comparing them to centralized solutions for large scale security requirements traceability. The first three properties specifically address the costs for initialization and performance of the systems. The first property, while being high for my approach, it only happens once during the initial private blockchain infrastructure creation. The fourth property is immutability, which is intended to ensure the integrity of security requirements history. The fifth, sixth, and seventh property are crucial for the traceability of security requirements throughout the entire software development process. Strictly looking at the table above, it is evident that my blockchain-based approach is suitable for performing security requirements traceability and management for large scale software development.

CHAPTER 11

CONCLUSION

In this thesis, an approach has been presented for security requirements traceability and management. This approach can be implemented using open-source software to create a private blockchain platform, like Hyperledger Fabric [31]. My approach has important properties that enhance the security of the software development life cycle such as immutability, auditability, non-repudiation, and the creation of a privacy-centric environment for security requirements traceability for large scale software development.

CHAPTER 12

FUTURE WORK

Future work includes improving the state of the open source blockchain solutions like Hyperledger Fabric [31] such that they are more efficient and easier to setup for developers. This also involves reducing the performance and storage cost of using a blockchain infrastructure. In addition, future work includes expanding my approach to facilitate more aspects of software development other than security requirements traceability and utilizing the private blockchain network that has been created to its fullest extent. Future work also includes improving the state of software security testing, in both open source and closed source software. Fuzzing tools like American Fuzzy LOP [32] and radamsa [33] require a significant amount of setup to harness making automated fuzzing difficult. Improving the state of fuzzing will allow for more bugs to be found and mitigated before they are exploited by malicious actors.

REFERENCES

[1] S. Nakamoto, "http://bitcoin.org/bitcoin.pdf," 2008. [Online].

[2] Synopsys, "Software Security Requirements," [Online]. Available: https://www.synopsys.com/blogs/software-security/software-security-requirements/. [Accessed 19 April 2022].

[3] OWASP, "OWASP," [Online]. Available: https://owasp.org/. [Accessed 19 2022 April].

[4] OWASP, "C1: Define Security Requirements," [Online]. Available: https://owasp.org/www-project-proactive-controls/v3/en/c1-security-requirements. [Accessed 19 April 2022].

[5] OWASP, "OWASP Application Security Verification Standard," [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/. [Accessed 19 April 2022].

[6] M. Silva and M. Danziger, "The importance of Security Requirements Elicitation and how to do it," *Project Management Institute Requirements Management,* 2015.

[7] N. Parveen and M. K. M. H. Beg, "Software Security Issues: Requirement Perspectives," *International Journal of Scientific & Engineering Research,* vol. 5, no. 7, 2014.

[8] "GitHub," [Online]. Available: https://github.com. [Accessed 19 April 2022].

[9] Gitlab, [Online]. Available: https://about.gitlab.com/. [Accessed 19 April 2022].

[10] Atlassian, [Online]. Available: https://www.atlassian.com/software/jira. [Accessed 19 April 2022].

[11] L. H. Newman, "GitHub Survived the Biggest DDoS Attack Ever Recorded," WIRED, 1 Mar 2018. [Online]. Available: https://www.wired.com/story/github-ddos-memcached/. [Accessed 19 April 2022].

[12] M. Ramachandran, "Software security requirements management as an emerging cloud computing service," *International Journal of Information Management,* 3 March 2016.

[13] C. Haley and R. Laney, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE Transactions on Software Engineering,* vol. 34, pp. 133-153, 2008.

[14] "The CORAS Method," [Online]. Available: http://coras.sourceforge.net/. [Accessed 19 April 2022].

[15] S. Türpe, "The Trouble With Security Requirements," *2017 IEEE 25th International Requirements Engineering Conference (RE),* pp. 122-133, 2017.

[16] "Microsoft Threat Modeling Tool threats," Microsoft, 2 January 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model. [Accessed 19 April 2022].

[17] "Common Attack Pattern Enumeration and Classification," MITRE, [Online]. Available: https://capec.mitre.org/. [Accessed 19 April 2022].

[18] K. Qian, R. Parizi and D. Lo, "OWASP Risk Analysis Driven Security Requirements Specification for Secure Android Mobile Software Development," *2018 IEEE Conference on Dependable and Secure Computing (DSC),* pp. 1-2, 2018.

[19] S. S. Yau and J. S. Patel, "A Blockchain-based Testing Approach for Collaborative Software Development," *2020 IEEE International Conference on Blockchain (Blockchain),* pp. 98-105, 2020.

[20] S. S. Yau and J. S. Patel, "Application of Blockchain for Trusted Coordination in Collaborative Software Development," *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC),* pp. 1036-1040, 2020.

[21] GitLab, "DevOps Solution Resource: DevSecOps," [Online]. Available: https://about.gitlab.com/handbook/marketing/strategic-marketing/usecase-gtm/devsecops/. [Accessed 19 April 2022].

[22] Microsoft, "DevSecOps in GitHub," [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/devsecops-in-github. [Accessed 19 April 2022].

[23] OWASP, "OWASP SecurityRAT," [Online]. Available: https://owasp.org/www-project-securityrat/. [Accessed 19 April 2022].

[24] OWASP, "OWASP Application Security Fragmentation," [Online]. Available: https://owasp.org/www-project-integration-standards/writeups/owasp_in_sdlc/. [Accessed 19 April 2022].

[25] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proceedings of the Third Symposium on Operating Systems Design and Implementation,* 1999.

[26] [Online]. Available: https://github.com/izar/pytm. [Accessed 19 April 2022].

[27] [Online]. Available: https://github.com/threatspec/threatspec. [Accessed 19 April 2022].

[28] [Online]. Available: https://www.cve.org/. [Accessed 19 April 2022].

[29] E. Keary and J. Manico, "Secure Development Lifecycle".

[30] M. Akbar, K. Smolander, S. Mahmood and A. Alsanad, "Toward successful DevSecOps in software development organizations: A decision-making framework," *International Conference on Digitization,* pp. 178-182, 2019.

[31] Hyperledger Foundation, "Hyperledger Fabric," [Online]. Available: https://www.hyperledger.org/use/fabric. [Accessed 19 April 2022].

[32] Google, [Online]. Available: https://github.com/google/AFL. [Accessed 19 April 2022].

[33] [Online]. Available: https://gitlab.com/akihe/radamsa. [Accessed 19 2022 April].

[34] Hyperledger Foundation, [Online]. Available: https://www.hyperledger.org/. [Accessed 19 April 2022].