TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Software Engineering

Polad Mahmudov 172676IVSM

RESOURCE RELIABILITY AND INTEGRITY ASSURANCE WITH BLOCKCHAIN TECHNOLOGY IN WEB BROWSERS

Master Thesis

Supervisor
Alexander Horst Norta
PhD, Associate Professor
Co-supervisor
Benjamin Leiding
PhD

Tallinn 2020

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Polad Mahmudov
Date: May 14, 2020

Annotatsioon

Plokiahelatehnoloogiat kasutava uue veebipõhise turvasüsteemi kujundamine vältimaks kasutajate koodisüstimise rünnakuid on keeruline ülesanne. Veebiklientide rünnakute eest kaitsmiseks on palju turvamudeleid, näiteks koodi saniteerimise vahendid, turbepoliitika, raamistikud, kodeerimisalgoritmid jne. Mõned neist on sirgjoonelised, mis tuvastavad rünnakud reaalajas, mõned aga analüüsivad koodi enne selle käivitamist. Viimast tüüpi turvamehhanismid on eriti võimsad. Just selleks, et paljastada juba teadaolevaid ründamismeetodeid.

Enamik kliendipõhiseid turbemeetodeid on suunatud rünnaku tuvastamisele pärast selle süstimist või loomist. Pahatahtliku koodi pookimine on võimalik, kui rakendus sisaldab meelega või kogemata loodud nõrka kohta. Olemasolevad turvameetodid on lünklikud ja ei suuda enne rünnaku algust tuvastada turvaauke.

Oleme välja töötanud inimeste kogemustel ning vastastikustel eksperthinnangutel põhineva turvamudeli. Idee on ühendada lähtekood ja koodi analüüsi eksperthinnangud, salvestades need plokiahelasse. Kui klient laeb alla käivitatava koodi, teavitavad vastavad plokiahela tulemused enne kompileerimist võimalikest turvariskidest. Kavandatud lahendusel on kaks peamist eelist: a) algne lähtekood on räsitud ja salvestatud plokiahelasse, mis tagab ressursside terviklikkuse; b) lähtekoodide ülevaatuse auditiaruandeid hoitakse plokiahelas, mis tagab ressursside usaldusväärsuse. Plokiahelatehnoloogia hajutatud eripära parandab süsteemi turvalisuse aspekte, muutes selle võltsimiskindlaks.

Abstract

Designing a new security system for web-browsers based on blockchain technology, to prevent users from code injection attacks is a challenging task. There are many security models to protect web clients from attacks in the form of code sanitizers, security policies, frameworks, encoding algorithms, etc. Some of them are straightforward, which detect attacks during runtime, on the other hand, some do analysis and sanitizations before code execution. These security mechanisms are really powerful, especially, to reveal already known attacking methods.

The majority of client-based security methods are aimed to detect an attack after it was injected, or created. The malicious code injection is possible if an application contains vulnerable code, which is created deliberately, or by mistake. There is a gap in currently existing security approaches, which are not able to prevent the attack before its initialization by detecting vulnerabilities in the code of resource.

We have develop a human experience-based peer-reviewing security model. The idea is to combine web-source code and code-review outcomes by storing them on the blockchain. When a client downloads an executable code, corresponding results from a blockchain inform about possible security risks before compilation. There are two main advantages of the proposed solution: a) the original source code is hashed and stored on a blockchain, which assures resource integrity; b) audit reports on code-review are kept on a blockchain, which assures resource reliability. The distributed particularity of blockchain technology advances security aspects of the system, making it tamper-resistant.

List of abbreviations and terms

API	Application Program Interface
CIA	Code Injection Attack
CLI	Command Line Interface
CPU	Central Processor Unit
CSP	Content Security Policy
CSS	Cascading Style Sheets
CVE	Common Vulnerabilities and Exposures
DApp	Decentralized Application
DNS	Domain Name System
DOM	Document Object Model
DPoS	Delegated Proof-of-Stake
DSR	Design Science Research
FEDS	Framework for Evaluation in Design Science
HTML	Hypertext Markup Language
HTTPS	Hyper Text Transfer Protocol Secure
IDS	Intrusion Detection System
ISR	Information Systems Research
MitM	Man-in-the-Middle
OSN	Online Social Network
OSS	Open Source Software
P2P	Peer to Peer protocol
RAM	Random Access Memory
UI	User Interface
URI	Uniform Resource Identifier
VCR	Version Control Repository
VCS	Version Control System
WAF	Web Application Firewalls
WEB	World Electronic Base
XSS	Cross-Site Scripting

Table of Contents

List of Figures

List of Tables

1. Introduction

By December 2019, the total number of Internet users worldwide had reached over 4.57 billion [1]. As people broadly use a WEB that leads to the growth of functionality, developed for the client, the complexity of web-application code increases. This is naturally accompanied by an expansion in flaws. Data injections are the most well-known cyber-attacks, so called Code Injection Attacks (CIAs) [2]. One of the most devastating attacks along with CIA, is Cross-Site Scripting (XSS), yielding a possibility to an adversary to execute arbitrary JavaScript code as part of a vulnerable application [3].

In practice, XSS attacks are often prevented by XSS sanitization frameworks [4] that encode and transform the resource before further processing by the client's application to detect any malicious attack signs; WEB Application Firewalls (WAFs) [5], which prevent potentially dangerous requests; and the Content Security Policy (CSP) [6] that specifies particular policies for all source scripts in a website [7].

Recently, a number of new approaches to initiate attacks have been established, which bypass the aforesaid defenses, making very well-known and majorly used defense techniques fragile to face modern security requirements. Particularly, *Heiderich et al.* [8] used encryption methods to transform malicious scripts into obscured payloads that can break through a number of modern sanitization frameworks. In addition, *Lekies et al.* [9] introduces a new form of code-reuse attacks by utilizing *script gadgets* of the website that turn the injected code into an actual XSS attack, consequently bypassing different security techniques including sanitizers, WAFs and CSP.

Source code review is the best technique of detecting vulnerabilities to all kinds of injections in web-applications [10]. We introduce a new protocol, based on code-reviewing, validation of openly accessible static code-files and Blockchain technology, in order to prevent server/client-side CIA attacks. The hash of the source files is persisted on a blockchain, turning into tamper-free and permanent reference. Thus, reviewers of the code publish their reviews with a generated hash of reviewed entry as well as their portion of reviewed code on the same blockchain. When the browser of the user downloads the Javascript source code, it validates the hash of the files against the corresponding blockchain entry. The script is executed if the hashes are valid and no negative reviews specify a security risk [11].

1.1 Existing body of knowledge

The following sections provide a brief introduction to define the state of the art and existing body of knowledge. First, a brief foreword about blockchain in Section 1.1.1 and peer-reviewing process in Section 1.1.2 is introduced. Then the overviews of contemporary security threats on WEB are given in Section 1.1.3 and respectively defense mechanisms to prevent malicious attacks in Section 1.1.4. Additionally, considering the main defense methodologies that prevent most frequent WEB attacks we discuss the reasons why security is still an issue in Section 1.1.5 and determine the gap in Section 1.1.6.

1.1.1 Blockchain technology

Introduced by *Haber and Stornetta* [12], Blockchain is considered as one of the technologies that potentially can solve a problem with the traditional centralized payment method. Since introduction of Bitcoin by *Nakamoto* [13], it has attracted intense attention. The solution is using multiple independent organizations to verify transactions, so-called decentralization.
There is a ledger in Bitcoin and other variants of Blockchain that records every successfully verified transaction and called a *block*. This is the reason for the name *Blockchain*. The validity of any transaction is verified by some nodes, whenever a new one is proposed. In other words, the sender has enough money to make a transaction, and has confirmed it by signing it with digital signature inside the transaction [14].

1.1.2 Peer-reviewing process

Over the years, inspections (formal peer-review process) have been perceived as a crucial method to evolve the quality of a software project. The quality inspections typically require periodic group reviews. [15]. They are conducted after a software artifact meets predefined final criteria (e.g., a specific requirement is developed). The process, originally defined by *Fagan* [16], includes some variation of the following stages: planning, overview, preparation, inspection, reworking and follow-up. In the first three stages, the author establishes an inspection package (i.e., defines what is to be inspected), roles are set (e.g., moderator), meetings are scheduled, and then the inspectors examine the inspection package.
A peer-review is natural way for Open Source Software (OSS) developers, who rarely meet in person, to ensure that the community agreement on good code contribution. The majority of large and successful OSS projects consider peer-review as one of their most significant quality assurance practices. Contemporary peer-review process is distinguished

from traditional software inspection by being lightweight and performed before the code is added to a Version Control Repository (VCR) that many developers depend upon (e.g., the master branch) [17].

1.1.3 WEB attacks

Nowadays, the WEB-technologies spread noticeably fast all over the world and people start to prefer the cloud computing services that are cheap, salable, efficient, mobile, etc. Cloud storage has many benefits and it provides functionalities to the user to store a bulky amount of sensitive and critical data. As the stored information is critical in the cloud, this attracts highly experienced attackers to steal data from the cloud by violating security mechanisms [18]. These attacks are categorized and shown in Fig. 1. There are browser and network based attacks that most frequently faced on internet. This thesis provides solutions to prevent application-code based vulnerabilities that open back-doors for malicious attacks. Those attacking categories are Man-in-the-Middle (MitM), Injection and XSS attacks. They are described more in Section 2.2.1.

1.1.4 Defense techniques

In order to fight against security attacks on the client-side, detection mechanisms may be involved in the form of special filters in the users' browsers, or installed as proxy servers with defined rules. Server-side attacks detection techniques are included on the servers, or installed as reverse proxy. A security pattern that deploys the defense solutions on both client-server side has also become popular as well as some machine-learning based security solutions [19].

Client-side defense mechanisms.

To mitigate client based attacks, special filters or sanitizers are integrated to **statically analyze** the source code of application presented through HTTP/HTTPS payloads between server and client [20]. These filters are deployed in the web-browsers as extensions using techniques of exact or approximate string matching or comparison, regular expressions. *Bates et al.* [21] propose XSS Auditor, which is added by default in Google Chrome[1] browser. The filter starts analysis after code is parsed by the browser, then scrutinizes the attempts to execute inline event handlers, scripts, JavaScript URLs, or load external scripts and plug-ins. *Ross* [22] introduces the IE8[2] XSS filter. The filtering heuristics are utilized

[1]Google Chrome `https://www.google.com/intl/en/chrome`

[2]Internet Explorer `https://www.microsoft.com/download/internet-explorer.aspx`

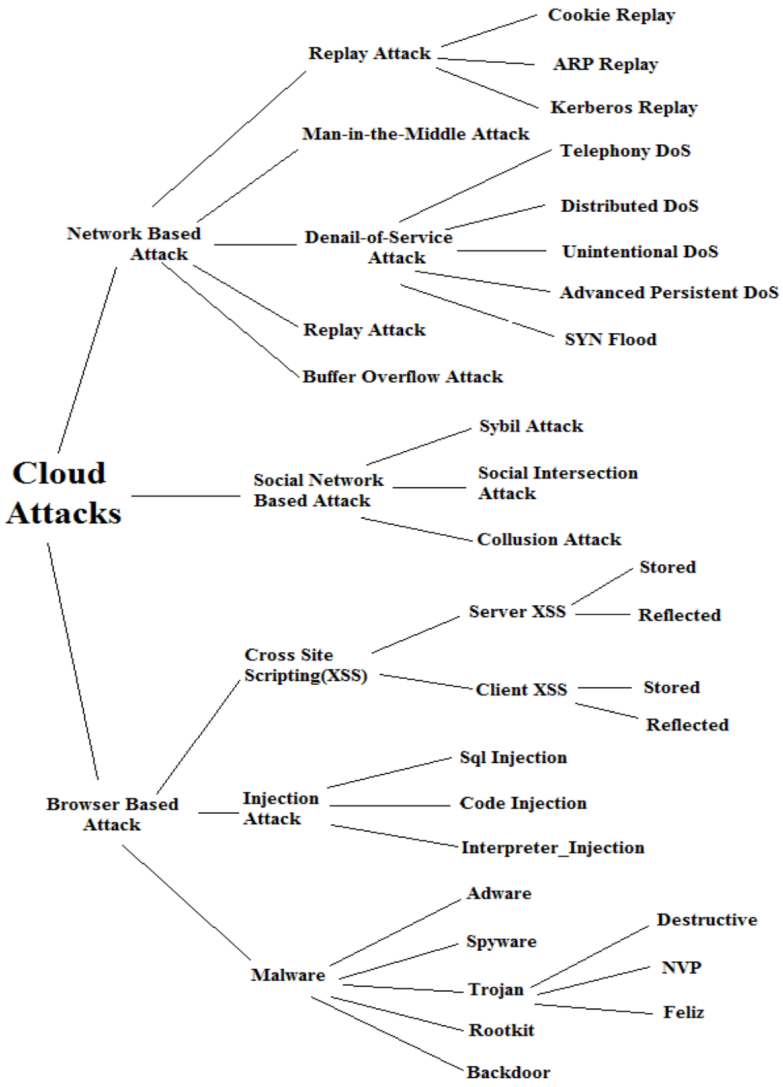

Figure 1. *Classification of cloud attacks* [18]

by regular expressions to detect the attack vectors from the specifically decoded URL, and also the POST data relating to the request. *Gupta et al.* [23] propose XSS-immune, which is a browser-resident extension that compares the set of scripts embedded in HTTP request and response to discover any vulnerabilities in source code. *Rao et al.* [24] present XBuster, which is also an extension to the Mozilla Firefox[3]. It generally implements a sub-string matching algorithm.

Dynamic analysis mechanisms are focused on the runtime behavior of an application. In comparison with static analysis mechanisms, they do not scan through the source code. *Pan et al.* [25] introduce DomXSSMicro, a micro benchmark for revealing and mitigation of Domain Object Model (DOM) based XSS atacks. This micro benchmark is designed of 175 test cases and each of them has a precise goal to check for particular characteristic of DOM based XSS attack. *Pan et al.* [26] developed a framework, which goal is to solve

[3]Mozilla Firefox `https://www.mozilla.org/en-US/firefox/`

the problem of DOM-sourced XSS attacks in WEB-browser extensions. The framework employs **hybrid analysis** involving best practices of static and dynamic analysis.
The most popular WEB security approach called Content Security Policy (CSP) based on content restriction methodology is now *W3C*[4] standard and supported by all major WEB browsers [6]. CSP is a language for declaring restrictions on the functionality of WEB applications, mitigating the dangers of a content injection by prohibiting the execution of inline scripts and by blocking a few dangerous functions.

Server-side defense mechanisms

There is a need to locate defense systems on the server-side taking into account that XSS attacks arise due to server-side vulnerabilities. A server being a powerful machine, allows detection nearly in real time. However, server-side defense techniques are limited to detect only stored and reflected attacks from server, meaning that network-based attacks are still dangerous. There are static and dynamic analysis of server-side attacks.
The server-side **static analysis** mostly deal with the source code of the WEB-application. *Mohammadi et al.* [27] propose a detection mechanism against XSS vulnerabilities foremost appearing as a result of incorrect encoding of untrusted data. Basically, for each WEB-application a set of unit tests are generated to disclose XSS vulnerabilities. For this purpose, the testing tool JWebUnit20[5] is used. *Gupta et al.* [28] introduce XSS-Secure, which is an OSN-based service cloud platform, has training and detection modes. The training mode carries out context-sensitive sanitizing of code and persists it for further use. Detection mode finds variance between the sanitized HTTP response from WEB-server and stored sanitized code, which states about existence of XSS worms on OSN servers.
Dynamic analysis are fulfilled during execution of the WEB-application. They actively analyze the program states against vulnerabilities in the application, such as values of the variables or even contents of memory allocation. In case of large scale WEB-applications, examining code with static analysis becomes very tedious, therefore, dynamic analysis are more preferred instead.
Jaballah et al. [29] propose a Grey-box approach, which utilizes the advantages of both Black-box and White-box testing [30]. The core component is a semi-supervised learning framework built on behavior graphs of the user interactions, which later pruned to phase out attacks or malicious behaviors with similar characteristics in the event graph, and determine benign interactions of the legitimate users. All other interactions that do not match into either of these categories could be considered as new attack scenarios.

[4]W3C `https://www.w3.org`
[5]JWebUnit `https://jwebunit.github.io/jwebunit/quickstart.html`

Client-Server defense mechanisms

Since both the client and the server are influenced by various attacks, the defense mechanisms should not specifically depend on either the client or the server alone. Thus, a defense mechanism, which distributes the load between the client and the server are preferred by combining the benefits of both the server-side and client-side defense mechanisms.
Goswami et al. [31] present method based on an Unsupervised Machine Learning mechanism, load-balanced between the server and client. The client does the initial processing of the request, extraction of feature, providing the profile of the attack and normal instances as reference by the proxy. If the suspicion level exceeds a particular threshold at the client-side, the request is immediately discarded. Otherwise, at the next stage, an attribute clustering algorithm, e.g. k-means algorithm [32], is utilized at the server to detect the attacks if they exists.

1.1.5 Contemporary security problems

The techniques which are described above mostly depend heavily on the knowledge and skill set of the forensic investigator, moreover, the increasing number of attacks and massive data currency make evidence analysis the hard task even with the help of traditional tools [33].

Network level

WEB application firewall currently is among of the popular solutions for prevention and detection of WEB application attacks. During the last decade, there have been a noticeable criticism against WEB application firewalls. They are directed to both commercial products and specific implementations [34], also the possibility to evade WAF by attacks [35, 36]. There are many limitations of WAFs, which pose them to become the inefficient solutions, such as high false positives rates and high false negatives, inability to find unknown attack and low accuracy, additionally high operational cost and much manual efforts [37–39]. These limitations have been addressed by several researchers in various ways by applying automation methods, for example, data mining and machine learning algorithms [36, 40, 41]; however, it raises the question of performance as the WEB applications deal in real-time with high traffic. Paradoxically, by enhancing the performance of data mining and machine learning algorithms will result in a decrease of accuracy [33].

Application level

All discussed mitigation techniques against malicious attacks in (Section 1.1.4) are the parts of Application Intrusion Detection System (AIDS). In reality, AIDS overcome net-

work based IDS problems [33]. Moreover, AIDS are utilized side by side with the firewalls to impede the WEB attacks by adding new enhanced security layer.
Most of the AIDS tools focus on the compressed data, correlation of the different sources and reporting. However, a huge amount of data produced from heavy WEB traffic is leading traditional techniques and tools to get ineffective, accompanied by increasing latency, costs and efforts [42].
WEB developers tightly depend on third-party libraries, sometimes, the results are vulnerable scripts or incompatible code. Researchers found recently [43], that statistically 37% of WEB applications include at least one library with a known vulnerability. They conducted these measurements taking into account outdated, old versions of 72 popular libraries.
Content Security Policies (in Section 1.1.4) take attention of most developers and find a place in their WEB systems. However, the adoption of CSPs has been limited and developers constantly misconfigure the security policies, allowing malicious code to be executed [44, 45].
Static analytical tools statically on client-side, without execution of code, seek to detect common programming errors. Researchers have used taint tracking to fight against DOM XSS vulnerabilities at run time [21–24]. *Ben et al.* [46] showed that in 73% of cases, one of the popular filtering technologies — the XSS Auditor that is default in Chrome browsers — fails to filter attacks. Nowadays, some of the WEB browsers substitutes static filters with CSP [47]. In general, static-analysis tools have more false positives than dynamic approaches. Dynamic taint tracking [25, 26, 48], the mechanism to observe the flows of information throughout a WEB application, has been offered as a run-time defense against DOM XSS; however, it demands huge infrastructure modifications to WEB browsers. Moreover, taint tracking at run time is able to visibly decrease performance [49].
There are some researchers, who put an effort to define fragility in current favorite security solutions by proposing different approaches and bypassing them.
Lekies et al. [9] found a new path to implement an attack named *Code-Reuse* attack, which utilizes script gadgets that are found in almost 650,000 WEB-pages regarding Alexa Top 5000 Websites[6]. The authors systematically show weakness of contemporary attack mitigation techniques. Script gadgets - in essence are small script fragments embedded in the vulnerable original code and not injected by the attacker. In this case, the intruder injects harmless HTML markup into vulnerable code, which is not determined by the current generation of XSS mitigation techniques as it does not comprised of executable script code. Nevertheless, during the application lifetime, the script gadgets grab the injected content and involuntarily transform it into executable code. The authors study multiple categories of script gadgets and proved that the gadgets capable to bypass the diverse XSS mitigation approaches with different security strategies like code sanitization, browser filters, code filtering and request filtering tools.

[6]Alexa Top `https://www.alexa.com/topsites`

1.1.6 Gap statement and contribution

The security problems in contemporary defence techniques urge development of a new solution to meet modern security demands. Regarding to observed problems of existing defence mechanisms in Section 1.1.5, a new security protocol must have the following properties:

- Prevent known and unknown flows of attacks;
- Low operational cost, manual efforts, latency and minimal infrastructure modifications;
- Detect vulnerabilities in legitimate first- and third-party code;
- Ensure validity and integrity of executable WEB-resource;
- Find misconfigurations and incompatibilities in embedded security tools, such as static or dynamic analytical tools, CSP, etc.;
- Discover presence of malicious attack before code execution on client-side.

The prevention of known attacks is possible by applying timeless knowledge and skills to detect vulnerabilities in code-base of the WEB-resource. We propose expertise peer-reviewing process, who put their experience on the table to gain incentives by contributing to reveal errors, misconfigurations and incompatibilities in the system. The discovery of unknown attacks is achieved through validation of the original resource with derived WEB-data by comparing their hashes in the blockchain during run-time. The results of the peer-reviewing and comparison of hashes are evaluated before every code execution on client-side giving a chance to avoid an attack if one of the flows fails.

1.2 Research methodology and questions

There are two paradigms that best determine the Information Systems Research (ISR) discipline: behavioral science and design-science. The behavioral science paradigm develops and verifies theories that interprets or predicts organizational or human behavior. The design-science paradigm seeks to expand the boundaries of organizational and human capabilities by developing new artifacts [50]. The research scope of this thesis is to develop a solution that mitigates client-side malicious attacks, therefore, the object of the study is an artifact in context, and two main efforts are designing and investigating this artifact in context.

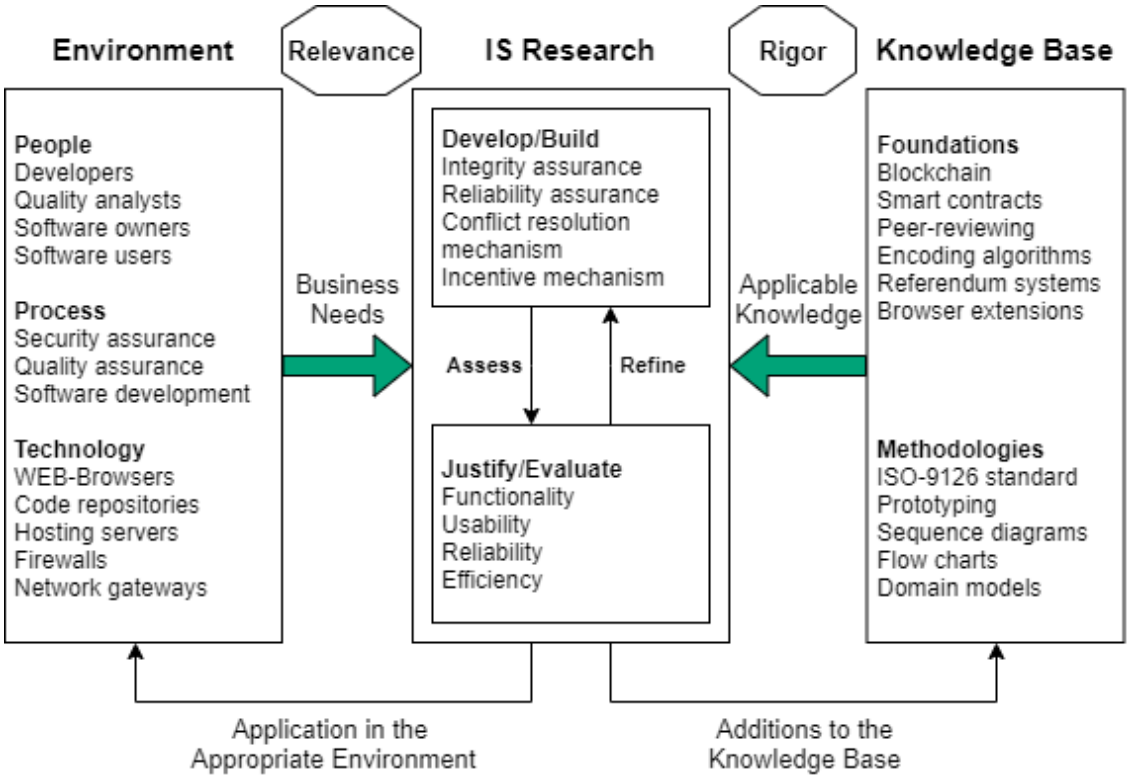

Figure 2. *Design science research framework* [50]

1.2.1 Design science methodology

Figure 2 shows conceptual framework for understanding, executing, and evaluating IS research connecting behavioral-science and design-science paradigms. It is instantiated, based on the current research.

The environment determines the problem space [51], where appears an interest. In IS research, it comprises of people, organizations, and their technologies [52]. In it are the goals, problems, tasks, and opportunities that set business needs.

Hevner et al. [50] states that business need in ISR is conducted in two complementary phases. Behavioral science develops and justifies the theories that explain or predict the identified business need. Design science builds and evaluates the artifacts designed to meet the identified business need. The purpose of behavioral-science research is truth, whereas, for design-science research it is utility. The research assessment through the justify/evaluate activities may run into the detection of weaknesses in theory or artifact and the necessity to refine and reassess.

The knowledge base is the source of raw materials from and through which IS research is fulfilled. The knowledge base is comprised of methodologies and foundations. These are foundational theories, instruments, frameworks, constructs, methods, models, and instantiations later used in the develop/build phase of a research study [50].

1.2.2 Design science in use

Hevner et al. [50] introduce seven principal Design Science Research (DSR) guidelines (Figure 3) that are adhered in this thesis. They claim that each of these guidelines should be addressed in some manner for DSR to be accomplished.

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Figure 3. *Design science research guidelines* [50]

Design as an artifact

An artifact instantiation of this thesis, which is aimed to close a gap and stated in Section 1.1.6, is indeed a software tool, providing WEB security against code-injection and XSS attacks on client's browser level, based on human experts peer-reviewing process and integrated in a blockchain.
The blockchain is a carrier for WEB-resource related entries, like its identifiers and peer-reviewed security reports. The resource identifiers are specific data-collections that uniquely identify a resource in the blockchain, for instance, the hash-code of a file, code-repository and hosting server URIs. On the other hand, security report is a statement, given by the peer-reviewer regarding security observations that are done over a code of the WEB-resource.

Problem relevance

The proposed artifact aims to provide a solution that is relevant to existing problem in various ways. The most important thing, which is absent in majority of already existing models, is instantly renewable knowledge put in the core of solution, as time goes by, the

new attacking vectors are occurred and apparently new mitigation methods are created. The peer-reviewing of a WEB-resource by multiple experts/contributors keeps the security aspects of a system up to date. By applying diverse modern mitigation techniques, they can find vulnerabilities regarding to back-doors for attacking in a code-base, already existing malicious code, usages of outdated libraries and wrong architectural approaches in the system.

A design-science approach is to construct the innovative artifacts, aimed at changing the heretofore unsolved and important business problems. Thus, a designed artifact has two main goals in the scope if this thesis. Firstly, since the malicious attack could be implemented on server, or network level, by tampering the original code and changing its initial structure, the proposed artifact must validate the originality of a resource. In artifact, this is done by the process, called integrity assurance. Furthermore, an original code can contain malicious code-pieces that are put intentionally, or accidentally, and threaten the security of an end-user. In this case, an original code must be validated against security flaws by peer-reviewing procedure and implemented in the artifact as a reliability assurance process. This process collects all the relevant security-based reports from experts, then reference them to the resource entries in the blockchain. Based on the security reports, an artifact draws decisions whether the resource is vulnerable, or safe.

The distributed characteristics of a blockchain that stores all the results of peer-reviewing process as well as WEB-resource identifiers, makes the solution tamper-free and minimizes the attacking factors.

Design evaluation

An evaluation of design theories and design artifacts is a key activity in Design Science Research. Correspondingly, Framework for Evaluation in Design Science Research (FEDS) is utilized to support and guide to design an evaluation component for DSR [53]. Based on the framework, four steps are proposed to approach an evaluation process. These steps are given in Tab. 1.

Table 1. *FEDS: four steps evaluation approach* [53]

Step	Description
Goals explication	There are at least four possible goals for evaluation in DSR. *Rigour* establishes that the artifact instantiation effectively works in a real situation. *Uncertainty and risk reduction* helps to mitigate technical and human social/use risks on first and subsequent stages of design. *Ethics* address potential risks to animals, people, organisations, or the public. *Efficiency* evaluation against the resources available for the evaluation (e.g., time and money).

Step	Description
Evaluation strategy selection	Strategy in FEDS implies a decision about why, when, and how to evaluate. There are four types of evaluation strategy: Quick & Simple, Human Risk & Effectiveness, Technical Risk & Efficacy, Purely Technical Artifact.
Evaluation properties selection	It is a process of choosing the general set of goals, features, and requirements of the artifact (instantiation, or design) that are to be subject for evaluation.
Evaluation episode(s) design	Identification and analyses of the constraints in the environment (time, people, budget, research site, etc.). Determination of what aspects are essential, more/less important, nice to have, and irrelevant. Definition of number of evaluation episodes, when and how they are conducted.

The current artifact is evaluated *summatively*, after its design is determined. Regarding to FEDS, selected strategy for evaluation is *Quick & Simple* strategy – the scope of this thesis is to design a small and simple construction with low social and technical risk and uncertainty. As the evaluation, here, is naturalistic in that sense it is conducted using a real system (the real methods) facing real problems. Thus, the real system is a prototype, based on the artifact design. The goals of a research are based on the criteria, which are defined in gap statement of this thesis in Section 1.1.6. Consequently, they can be grouped into generalized evaluation properties, like artifact should be functional, usable, reliable and efficient. This evaluation methodology is derived by adapting criteria as design goals. *Mathiassen et al.* [54] (2000, based on the ISO standard 9126) provide a quality model for evaluation of the mentioned properties.

Research contributions

An initial idea of this thesis is taken from an academic paper by *Cap and Leiding* [11]. This paper sets the scene and states the gap in existing body of knowledge, which is also a part of the gap statement in this thesis. Additionally, a general solution-design for an artifact is also described, precisely a peer-reviewing process is taken as a base concept for the reliability assurance, and integrated with a blockchain technology. Besides that, the integrity assurance notion, keeping the resource in a blockchain is suggested, as a proof of resource originality. They claim a need in appropriate conflict resolution process and incentive mechanism for the contributors, providing different ideas and concepts, as part of the designed solution.

The major contribution of this thesis is to specify an exact implementation of all ideas given above, by modifying and developing the initial concepts. More specifically, the data

structures in a blockchain and validation flows for the reliability and integrity assurance procedures are determined. Then, a suitable blockchain platform for a distributed application is defined, which corresponds to the ideology of an artifact. An appropriate validation process is implemented for conflict resolution on security statements of peer-reviewers. The users' groups, their interests and rewarding system are also determined.
Finally, a proposed artifact is later in the scope of this thesis is formalized into a prototype. This product can be used for further evaluation and development of consistent and full-fledged security tool integrated in WEB-browsers. The creation of a platform, which allows different contributors to use any knowledge for vulnerability detection in a software, is a completely new approach in WEB security that is a base of designed artifact.

Research rigor

In design-science research, rigor is derived from the effective use of the knowledge base and research methodologies [50]. An existing knowledge foundation is used to determine a suitable blockchain platform, foremost by defining the particular criteria regarding an artifact, then selecting an appropriate platform based on them. A rigor is also provided during establishment of a peer-reviewing process by collecting the best contemporary practices on tool-supported, or manual assessment of the software products as well as reporting forms to enable a genuine expression of security observations. Moreover, the knowledge base is adapted to define a conflict resolution, by introducing a referendum procedure that is applied in the given artifact, to reach a consensus on reviewers' debates about security of a WEB-resource.

Design as a search process

The design is significantly a searching process to discover an effective solution to a problem. An abstraction and representation of relevant means, ends and laws are essential components of design-science research. The means are the group of actions and resources, existing to construct a solution. The ends deliver goals and constraints on the solution. And the laws are uncontrollable forces in the environment [50].
The means of a current artifact are the existing peer-reviewing methodologies, used in practice, which are effectively capable to mitigate any software-related security vulnerabilities. They are put in the core of a solution to reach the corresponding ends, increasing the reliability of software products. Meanwhile, the experts should be attracted by their interest to contribute on revealing software flaws. In this terms, token and reputation based incentives are applied in the artifact.
A solution aims to resolve security issues for centralized software applications, therefore, it must be resistant to any ordinary attacks, existing for centralized systems. Thus, a blockchain based security assurance model is proposed. However, the cost is the users'

local resources, demanded to store their own security environment on the blockchain node.

Communication of research

A design-science research has to be presented both to technology-oriented and management-oriented audiences [50]. Therefore, results of a thesis are published openly to all the interested audiences in the relevant portal of a university for academic papers. Additionally, the code of a prototype is open-source and available in the corresponding code repository[7].

1.2.3 Research questions

The structure of the research questions follows the Trivium[8] structure. The main research question is stated as follows: **How to assure the reliability and integrity of web-resource using blockchain technology in web-browsers?** In order to give the answer to main question it is divided into following hierarchy of sub-questions:

- **RQ-1: How to assure resource integrity in web-browsers using blockchain technology?**
  - What type of blockchain is suitable to solve resource integrity problem?
  - What data is kept in a single transaction of a blockchain?
  - What is the validation process of the data stored in a blockchain to assure resource integrity?
- **RQ-2: How to assure resource reliability using code-reviewing process?**
  - Which tool-supported, or manual activities can assist for vulnerability detection?
  - What is the reporting form that reveals existence, or absence of vulnerability?
  - What is the validation process of the data stored in a blockchain to assure resource reliability?
- **RQ-3: How to assure report reliability and incentivize reviewers for contribution?**
  - What is the conflict resolution process enabling report reliability assurance?
  - Who are interested in revealing of the security issues?
  - What is the incentive/reward for contributors?

RQ-1 defines the way a blockchain is implemented within proposed solution by describing the type of a blockchain that is suitable for this project, how data is kept in a block and used to assure resource integrity. **RQ-2** describes the pear-reviewing process representation in

[7]Prototype `https://github.com/PoladMahmudov/master-thesis`
[8]Trivium `http://www.triviumeducation.com/`

an artifact and states the validation method, used to ensure resource security, based on collected reports in a blockchain . **RQ-3** gives an explanation about conflict resolution process on the reliability of security reports and incentive mechanism that attracts users interests' to contribute in proposed artifact.

1.3 Thesis structure

The rest of the thesis is structured in a following way. In Chapter 2, a running case is set with real security problem, demonstrated on open-source software application. Additionally, the background knowledge is given about the existing WEB-attacks, general blockchain structure, importance of peer-reviewing process and available incentive mechanisms. Then, Chapter 3 introduces comparison of popular blockchain technologies, and distinguishes among them the most suitable one for the given artifact. Later in this chapter, the resource integrity validation process and integrity-related smart contract are described. Furthermore, resource reliability validation and peer-reviewing procedures are given for the scope of proposed artifact in Chapter 4. In Chapter 5, conflict resolution technique on security verdicts is given, which is based on particular referendum process. Besides that, rewarding method, as an incentive mechanism, is provided in the same chapter. In Chapter 6, the final prototype and its evaluation are defined, referring to designed artifact. Finally, a summary of the thesis, by answering each of the research questions, and future work are determined in the Chapter 7.

2. Prerequisites

This chapter provides an introductory information about main objectives of this thesis. First, Section 2.1 describes the running case with specific enterprise example. Section 2.2, introduces the main aspects that are keys to solve given security issues.

2.1 Running case

As soon as, JavaScript gained a fame in the modern software development industry, being a client-side executable code, it increasingly threatens users with inclusion of vulnerabilities, back-doors, wrong access permission, etc. This could be done intentionally, or involuntarily by service provider. Regardless some service providers make their WEB-application's code be available in version control repositories, e.g. *GitHub*[1] or *Bitbucket*[2], they do not stop serving tampered files to users. In this case, every user should check security of the code personally. It is a tedious process for even experts to track every change to the code-base, moreover, reviewing incoming JavaScript files before execution is not feasible. Maybe with small applications some review is doable, but when it comes to complex ones this task is impossible at all.

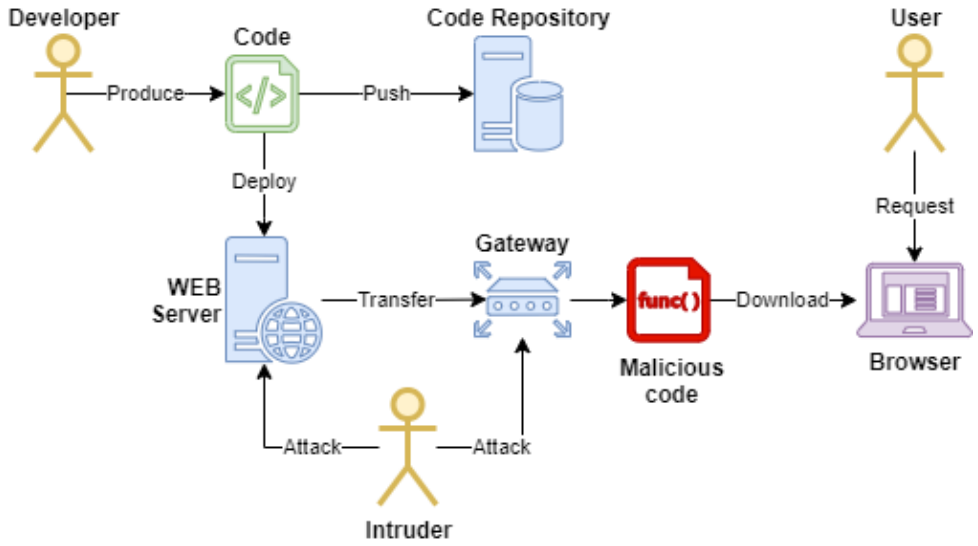

Figure 4. *Server-side code poisoning attack* [11]

Cap and Leiding [11] give an example of WEB-based messaging application called

[1]GitHub `https://github.com`
[2]Bitbucket `https://bitbucket.org`

WhisperKey[3] that includes code-based vulnerability issues. This service offers secure messaging using public and private key pairs. A private key is always stored in a local storage of user's browser, whereas, public key and special identifiers – *magic words* are sent to the server of WhisperKey. The recipient transmits magic words to the message sender, who obtains with them the relevant public key from a server. Further, messages are encrypted with this public key and sent to the server, which identifies receiver by magic words and forwards the message. Finally, receiver decrypts the message with private key on his/her side.

There are security limitations, for example, short RSA keys, accidental collisions of magic words, but the most severe security problem is a classical man in the middle attack between the WhisperKey server and user. In this attack, shown in Fig. 4, the forged JavaScript is distributed, which extracts the private key of the user from browser's local storage and easily steals it. On the other hand, the WhisperKey server cannot be for 100% secure as it centralized and has possibility to be tampered. The server code can have multiple vulnerabilities resulting in various attack types, described in Section 2.2.1.

There are three main factors that may lead to security breach in a system. They are code, server and network level security flaws. More specifically, code may be vulnerable, or already contain malicious scripts. On the other hand, hosting server or network can have serious misconfigurations and allow intruders to interfere and inject a virus.

Thus, in order to prevent code level attacks, a reliability assurance methodology is proposed as a solution that brings up together the experts, who assess the corresponding code-resources and evaluate the security threats in WEB-application. Moreover, an integrity assurance technique is also introduced, which validates the originality of a resource on client-side, preventing code substitutions, or injections on server and network levels.

2.2 Background

This section provides background information that is related to running case in terms of current security issues in Section 2.2.1, and base knowledge to derive alternative solution to mitigate those issues. There is a background knowledge, regarding *blockchain technology* in Section 2.2.2, which is a distributed storage system, keeping relevant security data. Then, *peer-reviewing process* in Section 2.2.3, results with security reports and conceptually referred to artifact uniqueness. Finally, *incentive mechanism* in Section 2.2.4, represented in the form of token and stake deals over the blockchain storage.

[3]WhisperKey `https://www.whisperkey.io`

2.2.1 WEB attack forms

There are two main targets of attacks, related to the most vulnerable parts of a system. They are browser and network-based attacks.

Browser-based attacks

One of the most severe and developed client/server-side attacking approach is considered Cross-Site Scripting attack. XSS vulnerability is a programming error, or a bug that is part of a web-application, or relevant execution environment, which unintentionally permits an intruder to carry out a XSS attack. This comprises of Hypertext Markup Language (HTML), JavaScript, Cascading Style Sheets (CSS) injections, and the injection of other different markup languages (e.g., VBScript, SVG, etc.) [55].
We observe in the sequel two major forms of server-side XSS attacks: *non-persistent (also reflected)* and *persistent* XSS attacks; and one client-side attack – *DOM based* XSS attack. The **reflected** XSS (Fig. 5) vulnerability appears in a web-application when an attack uses information given by the user to produce an outgoing web-page for that user. In this manner, instead of persisting the malicious code into a message, the attacker here injects code itself directly, reflecting it back to the user and utilizing a third party mechanism. This can either be implemented by attracting the victim to open a malicious URL, or by social engineering, or phishing, or by luring the victim to an attacker crafted web-page, which is able to generate a malicious request (e.g., by submitting a hidden form or embedding an *iframe*[4]) [56].

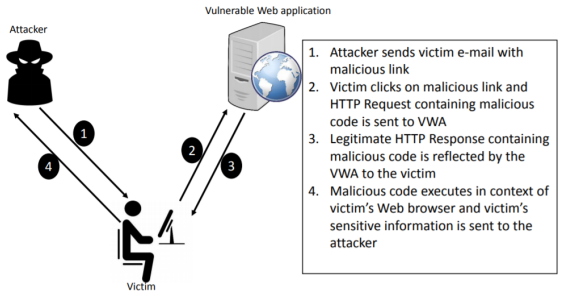

Figure 5. *Non-persistent/Reflected XSS attack* [19]

In **persisted** types of attacks, the data that is submitted by the user is stored on the server-side (Fig. 6). In this case, the system is considered as vulnerable, since the submitted data might contain malicious content. This malicious content, downloaded from the server, is trusted for the browser that executes it [57]. Thus, absence of sanitization techniques and proper validation of user input results with XSS vulnerabilities in the server application. Finally, all users of this web-site are now at the potential risk of an attack [19].

[4]iframe `https://www.w3schools.com/tags/tag_iframe.asp`

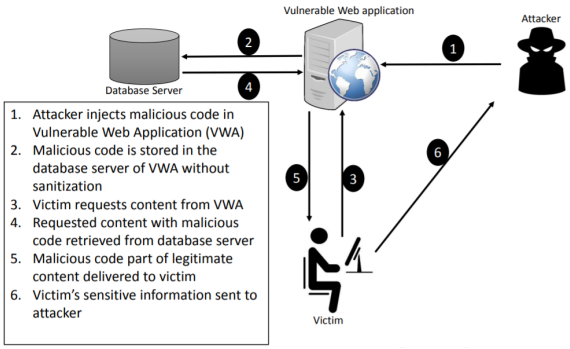

Figure 6. *Persisted XSS attack* [19]

DOM XSS attacks (Fig. 7) are possible, owing to existence of some vulnerability in the resource code interpreter of the client's browser. The DOM structure of the web-page on the browser is modified. The static HTML pages are perhaps vulnerable to client-side XSS, as soon as JavaScript code is embedded. In addition, client-side XSS is not crucially caused by HTML and JavaScript, but can also be induced by plugin-based content such as Flash, or by style-based CSS expression statement in a web browser. Hence, for the server-side a vast amount of sophisticated tools have been created, but there are almost no tools to defend against client-side XSS vulnerabilities [19, 55].

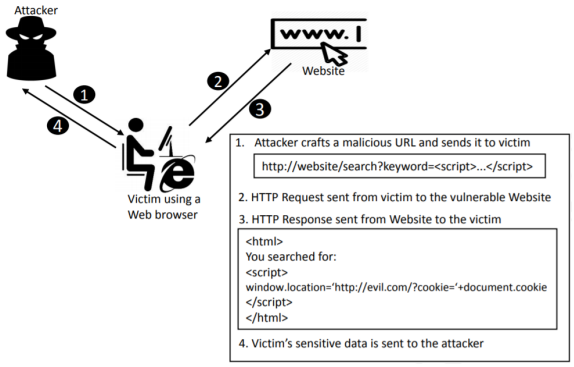

Figure 7. *DOM-based XSS attack* [19]

Network-based attacks

The diverse types of network based attacks exist. Since, the proposed solution prevents code-based vulnerability attacks, then in the scope of this thesis, we concentrate on Man-in-the-Middle Attack (MitM) type. MitM wants to be between two parties, who are communicating and be cable to secretly listen to the communication being transferred. Subsequently, by intercepting and overhearing the conversation, the intruder is able to pretend on behalf of one of the end-user and proceed the transaction of the messages. On both sides of communication, the original parties are unsuspecting about the attacker's access to the conversation, who receives and re-transmits messages [18]. When an attack is successfully initiated, it is possible even to attack Hyper Text Transfer Protocol Secure (HTTPS) communication. In this case, user would be shown a security warning that is

most of the time ignored, and therefore the system is compromised [58].
For example, in 2017 [59] an attack was committed, when hackers took control of bank's DNS server in Brazil and tricked a certificate authority by issuing a valid certificate to the customers of the bank, resulting in obedient handing over of their account information.

2.2.2 Blockchain

Since, Bitcoin was the very first implementation of decentralized payment method it became pretty famous. Many variants of Bitcoin have been proposed since 2009, such as *Ethereum* [60], and *Nxtcoin* [61]. Everyone, who wants to maintain the ledger can join or withdraw anytime. Therefore, they are called *Public Blockchains*, where the results are collected from all the unknown number of nodes before the final decision. Consequently, to add a new block to the chain, each of the nodes have to prove that they are more *qualified* than others to do this work. Therefore, a consensus algorithm of this kind is named *proof-based* consensus. The first type of a proof-based consensus algorithm is proof of work (PoW) from Bitcoin [13]. Besides of this, there are also other proposed algorithms, like proof of stake (PoS) from Etherium, proof of elapsed time [62], proof of luck [63], and proof of space [64], etc.
On the other hand another type of algorithms - *voting-based* consensus algorithms exist. The main rule is that only permitted nodes may verify the system, these Blockchain platforms include consortium Blockchains and private Blockchains [14].

Core mechanism

The first thing that should be clarified is verification of the identity of sender by system. In order to do this work in blockchain, two definitions are utilized: *public* and *private keys*, also known as *digital signatures*. Then the transaction, combined with this signature, becomes a request transaction. As the ledger holds information about every past successful transaction, it is possible to check the validity of it, whether the sender has enough money to commit transaction, or not.
A blockchain is the series of blocks, which are linked sequentially to each other like a chain. Each block includes many transactions that are validated. Fig. 8 depicts an example of Bitcoin blockchain (some headers in blocks are skipped).
Besides transactions, there are some specific fields in a block:

- **Reference to parent** (`Prev_Hash`): This is hash of previous block in the chain, which ties each block to its parent. It is calculated from entries of the previous block.
- **Timestamp**: The time when the block was created.
- **Merkle Root** (`Tx_Root`): This keeps the hash value of all validated transactions

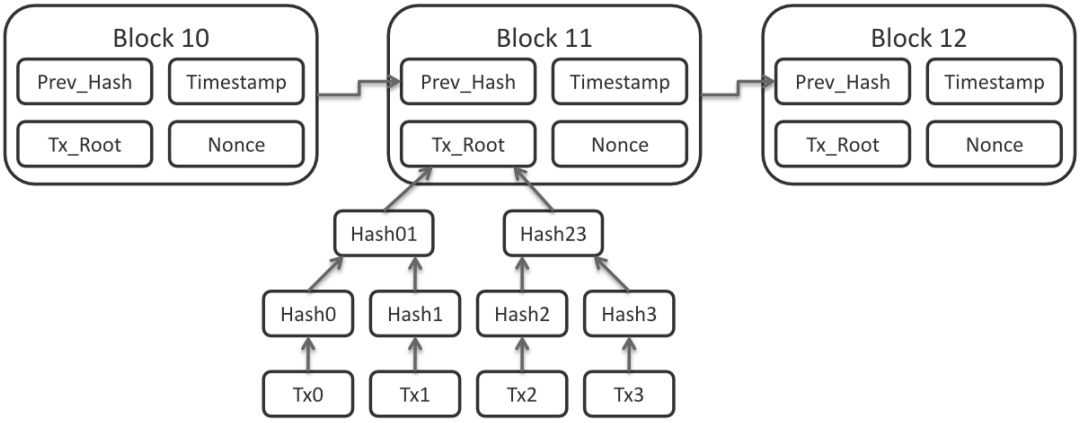

Figure 8. *Blockchain example* [65]

of a block. All transactions are hashed with each other as well as their hashes — pair-by-pair, until they swapped into single hash called *Merkle Root*.

- **Nonce**: This is a main component for intensive calculation (mining) process used in PoW. It proves the efforts that a node has paid to get the right to append a block to the chain.
- **Version**: This field includes the protocol version, which is used by node.
- **The block's own hash**: All the headers in block are hashed into this field that is later used as reference `Prev_Hash`.

2.2.3 Peer-reviewing

A peer-review based analysis of the provided resource code is one of the major components of proposed artifact. The quality assurance is required in many different fields, as well as in software development and academic research, therefore, the common practice is always transparent peer-reviewing process [15, 17, 66, 67].

Secure code review

Since the main goal of this thesis is to detect vulnerabilities in code before its execution on client-side, e.g. in browsers, accordingly, the peer-reviewing process should be security oriented. If a code is not reviewed for security flaws, the likelihood that the application has problems is virtually 100%. A secure code review is probably the most effective method to identify security bugs early in the system development life-cycle [68].
Fig. 9 shows results of survey (conducted by OWASP[5]) on IT company's respondents rating most effective security methods, regarding *OWASP Top 10 vulnerability types* [10]. This survey illustrates that manual code review must be a component of application's secure life-cycle, as long as in many cases it is either as good or better, than other methods

[5]Open Web Application Security Project (OWASP) `https://www.owasp.org`

of detecting security problems.

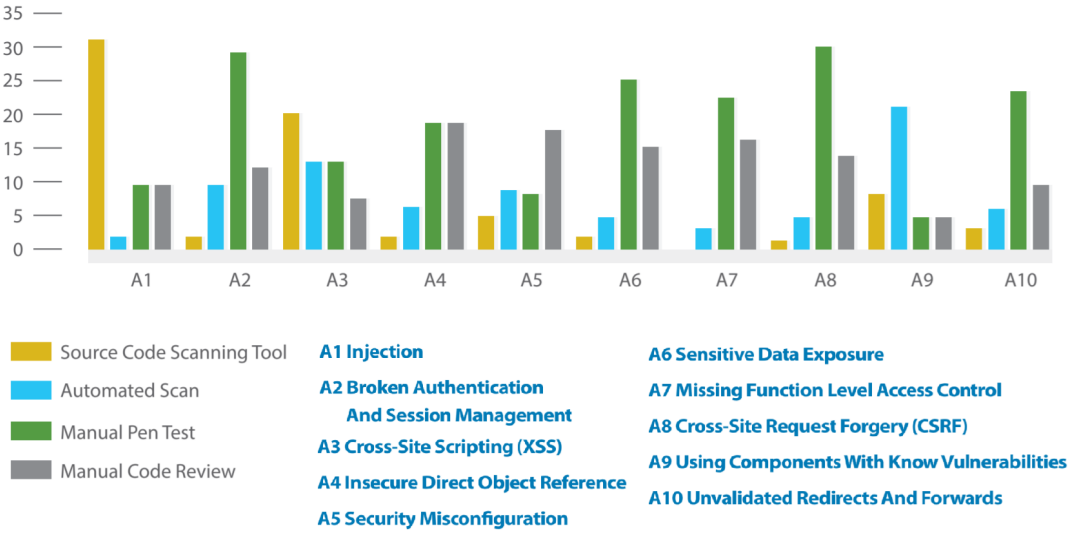

Figure 9. *Survey relating detection methods to OWASP Top 10 vulnerability types* [68]

There are multiple factors, considered for code review process, which is unique to its context. A decision on code review flow relies upon the reviewer, who considers any technical, or business related factors [68].

- **Risks**: It is impossible to secure everything at 100%. It is important to prioritize what components and features must be securely reviewed with a risk based approach.
- **Purpose & Context**: Computer programs have different purposes and subsequently the level of security varies, depending on the functionality being developed.
- **Lines of Code**: An indicator of the amount of work is the number of lines of committed code that must be reviewed.
- **Programming language**: The applications written in type-safe languages (e.g. Java or C#) are less vulnerable to certain security issues than others like C and C++.
- **Resources, Time & Deadlines**: A proper code review for a complex program takes longer time and it needs higher analysis skills than a simple one.

Version control repositories

It is essential to mention the storage that keeps the source code, which plays an important role in solution design of proposed artifact. A Version Control System (VCS) allows to track the iterative changes made to the program code. Furthermore, it is possible to record messages over each iteration of successive version so that anyone else can review the development history of the code and comprehend the rationale for the given modifications. Using a VCS, the collaborators can make and save changes (e.g. committed) to the code, then later these changes are possibly incorporated to the main code base (e.g. merged).

The emergence of technologies, used in development of websites, enhanced a collaborative aspect of hosted version control systems [69].
The VCS products, such as *Git*[6], and online hosting sites, like *GitHub*, *Bitbucket*, *GitLab*[7], etc., are currently very popular among scientists and programmers in general. The hosting sites allows very comfortable way to review the code instantly in website, without pulling the copy of the new commits to local copy of the resource.

2.2.4 Incentive mechanisms

As for all systems that constantly demand human-based contribution, some incentive methods should exist to interest more people. Since peer-reviewing process is considered as part of the solution, then specifically experts, who do the reviews, should have significant motivation. In contrast, users that use the results of reviews have quite simple incentive – security. The same is true about developers, who also wish to have secure and reliable product they create.
In practice, different types of incentive mechanisms are developed and successfully integrated in different fields. As an example, the *Steemit*[8] platform is a social blockchain. Its is a social media platform, where content creators are awarded with monetary tokens as well as community (e.g. curators), who supports the publications by taking a part in *rewards pool*. The voting system is a process of leveraging the crowd to assess the content and give a value to it in the form of tokens, consequently, enabling tokens distribution. These two properties of a platforms defines a new *Proof-of-Brain* concept. Additionally, all the active actors of a system gain newly minced tokens based on the particular rules of a blockchain [70, 71].
Semada is a decentralized autonomous platform for validating domain specific reputation. The goal of Semada is to provide evaluation platform for domain specific reputation across diverse variety of expertise by securing it from Sybil attacks and making its architectural design robust, supporting modular constructions for different use cases. The experts validate requests in their domain by betting their stakes in the betting pools, and the majority vote decides the result of the validation request. As far as decisions in a platform are requested with stakes, the platform has internal economics. Apart from growth of reputation, the experts can benefit from salary arrangements in the system [72, 73].

[6]Git `https://git-scm.com`
[7]GitLab `https://gitlab.com`
[8]Steemit `https://steem.com`

3. Resource Integrity

The following chapter deals with formalization of the main objectives for resource integrity assurance. There is an evaluation process in Section 3.2 on different types of blockchain platforms, where their advantages are compared for development of Decentralized Applications (DApp). Afterwards, the block structure is determined by using smart contracts that describe how the resource is presented in a selected blockchain platform (Section 3.3). Furthermore, resource integrity validation aspects are given, involving resource data from blockchain and browser-side request's interception methodology in Section 3.4.

3.1 Introduction

The objective of Chapter 3 is to answer *RQ-1* research question – *How to assure resource integrity in WEB-browsers using blockchain technology?* In this case, three subsections are derived from the main question of this chapter by answering corresponding sub-questions:

- **RQ-1.1: What type of blockchain is suitable to solve resource integrity problem?**
- **RQ-1.2: What data is kept in a single transaction of a blockchain?**
- **RQ-1.3: What is the validation process of the data stored in a blockchain to assure resource integrity?**

The blockchain-based applications are preferred, since they assure integrity, traceability, and anti-tampering. The risk of data tampering is one of the major security concerns for data-centric applications. By the nature, the blockchain technology is a befitting revolutionary solution to mitigate the tampering risks [74].
The meaning of resource integrity is mainly expressed by ensuring data originality downloaded from server. It is done by keeping the initial resource scripts in a blockchain. *RQ-1.1* explains in Section 3.2, where the original data is kept, whereas, *RQ-1.2* in Section 3.3 – how the data is persisted in a blockchain. Finally, *RQ-1.3* in Section 3.4, describes the way data is used in a browser to assure that original script is not tampered, or substituted.

3.2 Blockchain platforms

This section introduces the main criteria that should be considered during decentralized application development in Section 3.2.1. Furthermore, we compare the contemporary blockchain platforms in Section 3.2.2, then overview the core concepts of selected blockchain platform in Section 3.2.3.

3.2.1 Decentralized application criteria

In order to solve integrity and reliability problems, the particular blockchain software must have all the specifications, which are required for contemporary blockchain applications. The software platform of a blockchain should at least provide accounts, databases, authentication, asynchronous communication, and schedule across many of CPU clusters, or cores. The consequent blockchain architecture that can tremendously scale to millions of transactions per second, doesn't need user fees, and allows fast and easy development of decentralized applications. The following criteria are considered [75]:

Support to millions of users

Since blockchain applications started competing with well-known business platforms as eBay[1], Uber[2], AirBnB[3] and Facebook[4], it pushes the blockchain software to be capable to handle millions of daily active users. For the purpose of assuring integrity in WEB-browsers, corresponding blockchain has to endure critical mass of users' requests to validate WEB resources.

Free usage

The most of applications, developed based on blockchain, are intended to benefit their users with free services. The reason for that is to give the platform a widespread adoption among users of blockchain application. Thus, application developers needs full flexibility in utilization of a blockchain.

Easy upgrades and bug recovery

As far as business grows the underlying software should evolve accordingly by enhancing with the new features. The blockchain platform must support software and smart contract updates.

[1] eBay `https://www.ebay.com/`
[2] Uber `https://www.uber.com/`
[3] AirBnB `https://airbnb.com/`
[4] Facebook `https://www.facebook.com/`

All non-trivial software are subjects to bugs. The platform must be robust enough to allow bug-fixes, when they occasionally arise.

Low latency

From the good user experience perspectives, a reliable response with a minimum delay of few seconds is vital. The blockchain platform should support low latency transactions.

Parallel and sequential performance

The blockchain applications need to have enough sequential performance to handle high volumes in stepwise manner, and parallel performance to divide the workload across multiple Central Processor Units (CPU) and computers.

3.2.2 Comparison of blockchain platforms

This section introduces the most relevant blochchain platform for an implementation of resource integrity as well as reliability assurance protocol for the users' security on WEB-browsers. There is a long list of different blockchain platforms that are created to provide technology vendors with the suitable environment for development of decentralized applications. Those systems, apart from having monetary transnational model, are developed to operate with different data types and structures by giving an opportunity to realize the enterprise level business platforms [76–78]. Tab. 2 shows comparison of contemporary and popular blockchain platforms.

The selection of blockchain platform for DApp depends on criteria, which are described in Section 3.2.1. Basically, the platform must be *scalable*, *flexible*, and *fast*. The *scalability* is a key for supporting millions of users. But recently, some blockchain platforms have faced the scalability problems and found solution in the usage of side-chains, whereas, side-chains break the concept of decentralization, which is the main feature of a blockchain [79]. Almost all of the platforms containing side-chains from Tab. 2 have mainly scalability issues.

In terms of *flexibility*, a DApp should be maintainable, to let developers to apply new features and bug-fixes fast and easily. To make it possible, the smart contracts and software should support upgrades. In Tab. 2, some platforms have hard-coded smart contracts, thereby, less flexible.

Finally, the *fast* blockchain platform that has low latency of transactions – is the best option for DApp. Nowadays, the regular centralized WEB-applications process users' requests with the average speed, starting from hundreds of milliseconds up to couple of seconds, depending on the service that is provided [80]. Thus, DApp's transaction time should be at almost the same rate as for centralized applications, to be able to compete with them.

Table 2. *Comparison of blockchain platforms*

Name	Consensus	Language	Block time (s)	Smart contract	Token creation	Side chain
ETHERIUM	PoW	Go, C++, Rust	14-15	✓	✓	✻
ARDOR	PoS	Java	60	✓	✓	✓
ARK	DPoS	JavaScript	8	✻	✻	✻
CARDANO	PoS	Haskell	20	✻	✻	✻
EOS	DPoS	C++	0.5	✓	✓	✗
ICON	LFT	Python	1	✻	✻	✻
KOMODO	PoW, dPoW	C++, C	60	✓*	✓	✓
LISK	DPoS	JavaScript	10	✗	✻	✻
NEO	dBFT	C#	15–20	✓*	✓	✻
NULS	PoC	Java	10	✻	✻	✻
NXT	PoS	Java	60	✻	✓	✗
STRATIS	PoS	C#, .NET	60	✓*	✻	✓
WANCHAIN	PoW	Go, C++	13	✓*	✓	✻
WAVES	LPoS	Scala	3	✓*	✓	✗
✻— in progress (not released); ✓* — hard-coded contract						

If we consider all above specifications and the most popular blockchain platforms from Tab. 2, the EOS blockchain perfectly matches with the DApp criteria. The EOS.IO has smart protocol capabilities also a database layer over the blockchain storage, which makes it possible to transparently and easily separate reliability and integrity aspects in a single blockchain platform.

3.2.3 EOS.IO core concepts

The *EOS.IO* is a blockchain protocol powered by native cryptocurrency EOS. It is considered as smart contract platform and decentralized operating system, mainly intended for development of industrial-scale decentralized applications. Etherium [60] is the first blockchain that supported development of decentralized applications. However, it was plagued by scalability issues once released for enterprise usage. The EOS.IO solved this issue by adding a layer on top of Etherium, and called in public as *Etherium Operating System* [75].

The core concepts of EOS.IO blockchain covers distributed characteristics of ledger events

and transactions, happening on this platform. It involves decisions, made on who creates new blocks, using Delegated Proof-of-Stake (DPoS) consensus algorithm. The EOS.IO enables creation of smart contracts, which behaves as a typical application, running on an operating system. The computational volume and speed depend on a bandwidth and CPU specifications respectively, and the storage of persistent information is presented in Random Access Memory (RAM) [81].
The CPU and bandwidth requirements for usage are transient, as they are needed during execution of some smart contract action, it is achieved by staking tokens for some amount of time – approximately three days. RAM, however, needs to be purchased beforehand depending upon the need, being a persistent resource. It conventionally relates to volatile memory, but in EOS.IO, RAM is represented as persistent memory for the smart contracts. It is important to note that all smart contract-related persistent information is not stored in the blockchain. The blockchain is only used to keep transactions and events that point to the change in the persistent information of a smart contract [82].

Delegated Proof-of-Stake

The DPoS includes staking tokens that give the right to vote for *block producers* – the actual full nodes, which add new blocks to the blockchain. If any malicious activity is suspected, voters are able to remove the block producers and vote for new ones. There are only 21 block producers create new blocks at the given time, therefore, transaction speed grows very fast, due to less amount of verification, than proof-of-work consensus algorithm [83].

Actions and Transactions

Actions are created, being a part of smart contracts, or generated by application code. There are two types of actions – the explicit actions, presented in a signed transaction and implicit (inline) actions, created as side effect of processing a transaction. They both defined in a smart contract. The major difference is that inline actions are not included in a transaction entity that are passed through the network to the local node, but are persisted in a block as an eventual result.
A transaction instance is comprised of a transaction header and the list of action instances and transaction extensions as shown in Fig. 10. The header contains transaction expiration time for its inclusion in the block. The extensions are block number, block ID prefix, transaction delay time, upper limits for CPU and network usage [84].
The life-cycle of the transaction starts with the creation of transaction and pushing actions into it. Then the transaction is validated locally on the node and included in a block with other transactions. Finally, the block, containing a given transaction is pushed to all other nodes for validation. In typical DPoS blockchain, the block producers have 100%

participation. A transaction is considered to be confirmed with 99.9% certainty in average of 0.25 seconds after a broadcast [75].

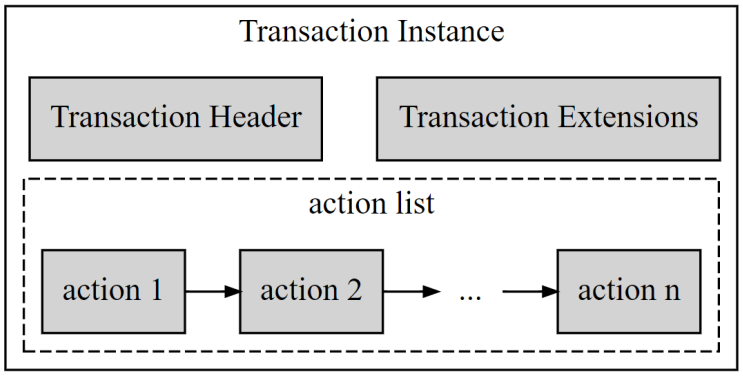

Figure 10. *Transaction instance* [84]

Network Peer to Peer protocol

The EOS.IO blockchain have Peer to Peer (P2P) communication protocol among nodes, in order to relay transactions, publish blocks and synchronize state between the peers. The main goal of P2P protocol is to synchronize nodes effectively and securely [85]. The EOS.IO system delegates this functionality into four main components, shown in Fig. 11:

- **Net plugin** – defines the protocol to synchronize blocks and send transactions among peers;
- **Chain controller** – manages/dispatches received transactions and blocks within a node;
- **Net serializer** – serializes messages, transactions and blocks to transmit through the network;
- **Local chain** – holds local copy of the blockchain.

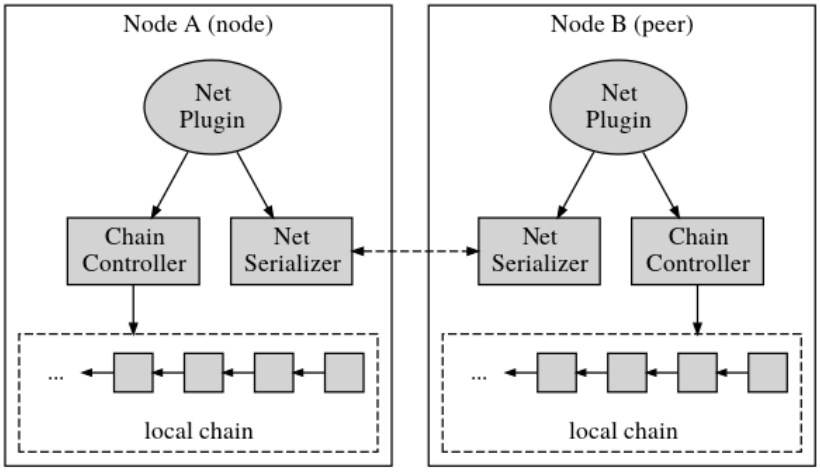

Figure 11. *P2P network protocol architecture in EOS.IO* [85]

Accounts and Permissions

The EOS.IO software allows all accounts to be referenced by unique human-readable name. They are chosen by creators of accounts. The creators reserve RAM, storing the account until the new account stakes some tokens to reserve its own RAM. The accounts also represent the smart contract actors in a blockchain.
Each account keeps the list of named permissions assigned to it. They are used to authorize actions and transactions to other accounts [86].

Development ecosystem

The development process of any king of blockchain application tightly relies on local testing node that provides a way to manage wallet, keys for accounts, code compilers for smart contracts, etc. The EOS.IO platform consists of the following components and tool-chains:

- `nodeos` (node + EOS.IO) – the core EOS.IO node daemon that run a node with specified configurations. It can be used for development and testing as local node, or even as full node for block production;
- `cleos` (CLI + EOS.IO) – the Command Line Interface (CLI) interacts with the blockchain and wallets;
- `keosd` (key + EOS.IO) – the component that helps to store EOS.IO keys in wallets;
- `EOSIO.CDT` – a set of tools to facilitate writing of smart contracts;
- `eosjs` – a JavaScript based Application Program Interface (API) for integration with EOS.IO blockchain.

3.3 Integrity contract

The business logic of every blockchain application is represented by smart contracts. The integrity contract defines the structure of a transaction that is stored in the blockchain, and eventually, needed to evaluate resource integrity. This section provides a smart contract definition in Section 3.3.1. The smart contract in EOS.IO includes multi-index table interface to connect with the persisted storage (RAM), functions' definitions that determine the logic of the action, and action handlers, which map incoming requests to the contract by triggering a specific action. In this case, the resource table and integrity actions are described in Section 3.3.2 and Section 3.3.3 respectively.

3.3.1 Smart contracts

In EOS.IO, the smart contracts include a set of actions, frequently grouped by the functionality, and set of type definitions that those actions rely upon. Thus, the actions define and specify the actual behaviour of a smart contract. There are several in-built actions that are implemented in EOS.IO contracts for producer voting, token operations, account creation, etc. However, application developers are capable to extend, replace, or disable them by establishing the custom smart contracts.
An EOS.IO smart contract is implemented as a `C++`[5] class that is extended with `eosio::contract` library. The actions are implemented as `C++` methods. The `eosio.cdt` tool, provided by EOS.IO platform, is used to generate the byte-code and Application Byte-code Interface (ABI) representation of smart contract from its `.cpp` file implementation. Later, the generated entries are applied to the blockchain, finalizing development of a smart contract [84].

3.3.2 Resource table

The structure of the data that represents any action in EOS.IO blockchain is called multi-index table. The EOS.IO database includes multi-index tables that act as simplified and regular tables, aiming to work with blockchain data, stored in the RAM.
The components that comprise the integrity concepts in the blockchain are given in the form of *resources* table and *publish* action (Section 3.3.3), which are the parts of *integrity contract*. The *resources* table is the base structure that represents the WEB resource entry in the EOS.IO blockchain (Tab. 3).

Table 3. *Resources table structure*

Col. Name	Type	Description
id	`uint64_t`	Primary key
hash	`checksum256`	Hash of the resource file
user	`name`	Creator identity
uri	`string`	Download URI of resource
repo_uri	`string`	Code repository URI

The primary key, which uniquely identifies the resource within the blockchain is declared by unsigned integer value with the width of exactly 64 bits (`unit64_t`). In order to identify the resource from application side, the hash of the file, encoded with *SHA-2* algorithm, is proposed as the secondary index in EOS.IO native type (`eosio::checksum256`) within the resources table. The secondary index value could be derived from the raw resource by hashing it, and further used for searching the resource entries in the blockchain.

[5]C++ Programming Language `https://isocpp.org`

Since users do publishing and processing of the resources, the owners should be defined with the specific for EOS.IO platform `eosio::name` type that identifies user accounts, initially created in the blockchain system. Finally, resource hosting and code repository Uniform Resource Identifiers (URI) are defined to reference the original resource.

WhisperKey resource integrity

For example, if we consider the case with the WhisperKey from Section 2.1, then on browser's request for the resource scripts, the server returns two executable files, see Fig. 12.

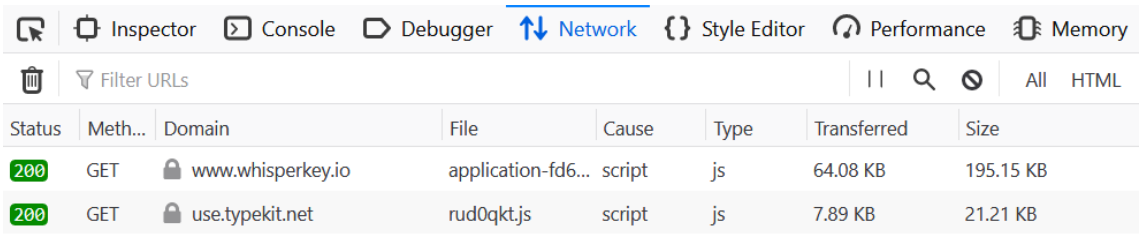

Figure 12. *WhisperKey resource scripts*

In the list, the first file is the main script file that carries out the business logic of the WhisperKey service. Whereas, the second file is the third-party script that is mainly used for styling of fonts in the WEB-page. Both files are open-source and could be published to the blockchain for further security investigations. Regarding to Tab. 3, the resource entry in the blockchain, e.g. `application.js` file, has the structure as illustrated in Tab. 4. The same could be done for the third-party script from the resources list in Fig. 12.

Table 4. *WhisperKey entry structure for "Resources table"*

Col. Name	Type	WhisperKey
id	`uint64_t`	101
hash	`checksum256`	`9d25a2fe4ef50412adab5ce864087fe3 3b142496dcc432af8b65e2ea668030f9`
user	`name`	Bob
uri	`string`	`https://www.whisperkey.io/assets /application.js`
repo_uri	`string`	`https://github.com/pixielabs/whi sper-key`

3.3.3 Integrity action

As long as resource integrity contract operates over the resources table, a process of storing the resource to given table is done by actions, which then become the part of a transaction, and eventually, immutably persisted in the EOS.IO blockchain. This action is named

publish action. In terms of `C++` syntax, it is a method, expecting arguments that are specific for the resource file. Those arguments are entries of the resource, shown in Tab. 3, except an `id` field that is generated automatically.
The integrity (*publish*) action consists of several steps that result with a resource entry, persisted in RAM of a blockchain node, and shown in Fig. 13. These steps are following:

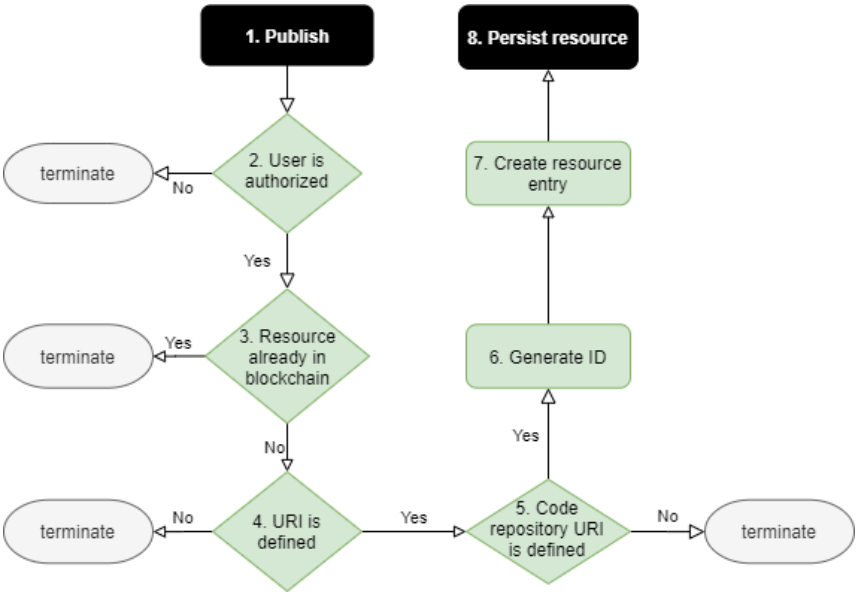

Figure 13. *Integrity (publish) action flow chart*

1. Handling the integrity (publish) action;
2. Checking the user's (creator's) authority level that allows to commit an action within integrity contract;
3. Checking the existence of a resource by its calculated hash through out the blockchain;
4. Checking the validity of URI hosting the resource file;
5. Checking the validity of URI hosting the code repository for given resource;
6. Generating the unique identification number for the resource in the blockchain;
7. Creating an instance of the resource that is coherent with the integrity contract;
8. Persisting the resource instance to the blockchain for further validation processes in EOS.IO system.

If any of the described steps do not meet the expected condition, or simply fail during the execution, then the integrity action is immediately terminated.

3.4 Integrity validation process

The node based integrity implementation was defined in the previous sections of this chapter, but the actual integrity validation process is performed on the browser-side. Each

client's browser can connect to the EOS.IO blockchain node and perform the three main steps to fully complete integrity concept by *publishing*, *retrieving* resource files and *validating* the integrity. This is also a full cycle of operations needed to be done, in order to provide integrity in browsers. However, once the resource is published, later it could be retrieved and used for integrity validation for unlimited times. In this section two major integrity operations are described: *publishing the resource* in Section 3.4.2 that includes only first step, and *assuring the integrity* in Section 3.4.1 by including last two steps of integrity assurance process.

3.4.1 Assuring integrity

The so-called green flow of browser-side resource integrity assurance is depicted in sequence diagram Fig. 14.

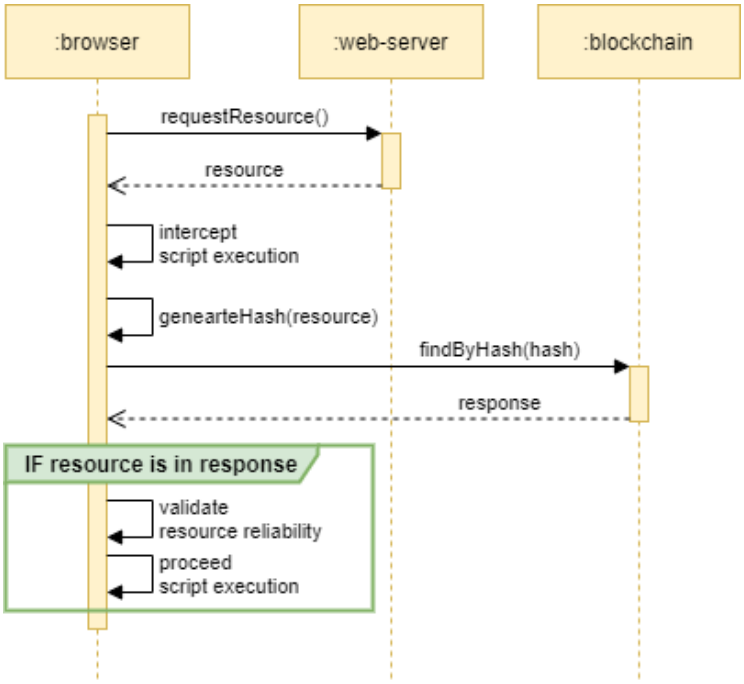

Figure 14. *Resource integrity validation – sequence diagram*

When a browser requests some server for executable WEB-resource, the execution of the script is intercepted and its hash is generated by SHA-2 encoding algorithm. Then, the result of calculated hash is sent to the blockchain node to compare it with initially published resource checksum. If the resource is found by sent hash in the blockchain, this means, the integrity clause is satisfied. Later, the resource reliability validation is implemented (Chapter 4). Finally, the execution of the resource script is continued, after being assured in its security.

On the other hand, if the resource is not defined by calculated hash in the blockchain, it keeps the state of the resource at *warning* level, meaning two things: resource is not

published yet, or it is substituted. In first case, client is able to publish the resource with corresponding data that fully identifies it (Section 3.4.2). In latter case, the user is warned about the possible security vulnerabilities. Eventually, depending on the user decision, the execution of the script is proceeded or terminated.

3.4.2 Publishing resource

In order to assure integrity in WEB-browsers, every downloaded resource file should be published to EOS.IO blockchain. The defined integrity action processes creation of the resource instance in resources table in the blockchain (Fig. 15).

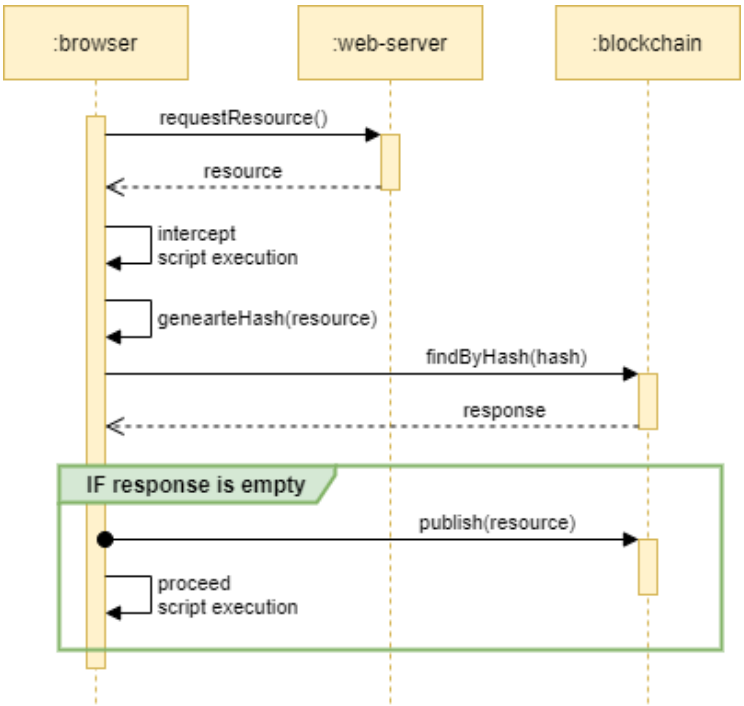

Figure 15. *Publish resource – sequence diagram*

At the browser-side, this procedure starts after the resource script is downloaded, then its execution is intercepted and generated a hash code from it. If the instance of resource file is not defined in the blockchain, the user gets an option to add it. Apart from the resource hash code, the server and the original code repository URIs are required as well, to be submitted to the integrity action that publishes the given entry as persistent data to RAM. Finally, user can proceed the execution of the resource script locally in the browser, additionally, providing its integrity that becomes available in the blockchain for other users.

3.5 Related work

The Sections 3.3 and 3.4 introduce the creation and usage of the resource integrity contract. The integrity is provided by comparison of the calculated hash-code of resource files with initially published hash-codes in a blockchain. Each resource is identified by calculated hash-code and searched by this value through the chain of blocks. Alternatively, the resource could be identified by the full URI path to the original script file. In this case, whenever the script is tampered, then the hash-code difference implies the existence of injection attack. However, some scripts are the third-party libraries (such as jQuery[6], React.js[7], Chart.js[8], etc.), imported and used in the resource code. It is not efficient to publish those scripts and duplicate them in the storage per every server URI, where it is downloaded from. The identification of scripts by their hashes, creates the possibility to avoid duplication, but it makes impossible to detect signs of injection attacks. As the file is tampered, its hash-code is changed, and further this file would not be found in the blockchain, even if the original one was published before. Additionally, the duplication of the resource entries in the blockchain is accompanied by difficulties to refer to them later, during the process of reliability assurance. Therefore, as an optimal option for resource identification is chosen its hash-code. Thus, the user is alarmed about the possibility of security flaw, and asked for confirmation to proceed or terminate the execution of this file.

3.6 Conclusion

This chapter was dedicated for designing of resource integrity concept in WEB-browsers. The integrity assurance is formalized by utilizing the EOS.IO blockchain platform. This platform allows usage of smart contracts that pertain to integrity contract establishment. The combination of integrity contract implementation on node of a blockchain and validation of the resource files on browser side results in notion of integrity assurance.

By considering the advantages of EOS.IO blockchain, this platform was selected as the main ledger for integrity assurance implementation. The information in transactions is stored in a defined structure that includes sufficient data entries, related to the resource file. These entries are hash-code, host-server URI and code repository URI of the resource. Later, if the given resource instance is published in the blockchain, then it can be searched by its hash-code. The blockchain, being an immutable and decentralized ledger, provides hash-code of the original resource for validation on browser-side. This validation finalizes the process of the resource integrity assurance.

For the future work, the compound of the hash-code and host-server URI of the resource

[6] jQuery `https://jquery.com`
[7] React `https://reactjs.org`
[8] Chart.js `https://chartjs.org`

file should be considered, or even other alternatives could be found, in order to identify the resource in the blockchain. This would allow avoidance of duplication of resource instances in the storage and better detection of the script injection attacks at the same time.

4. Resource Reliability

The next chapter proposes a base concept for implementation of the resource reliability in WEB-browsers. By considering the idea of peer-reviewing process as one of best approaches for reliability assurance, the Section 4.2 describes existing types of code-review procedures that are very effective nowadays. In the previous chapter we observe how the resource is identified and retained in the blockchain, resulting in integrity validation. Thus, the outcome of a peer-reviewing audit on the given resource is also stored in the blockchain as defined in Section 4.3. Eventually, all the collected review reports formalize the ground for a final validation of the resource reliability in Section 4.4.

4.1 Introduction

The main goal of Chapter 4 is to find an answer to the *RQ-2* research question – *How to assure resource reliability using code-reviewing process?* In order to better distinguish the main question, it is divided into following sub-questions:

- **RQ-2.1: Which tool-supported, or manual activities can assist for vulnerability detection?**
- **RQ-2.2: What is the reporting form that reveals existence, or absence of vulnerability?**
- **RQ-2.3: What is the validation process of the data stored in a blockchain to assure resource reliability?**

Each question is answered independently in a separate section. The reliability of the resource script is stated by absence of vulnerable code-parts, which are examined by the field experts, performing peer-reviewing. The important examination procedures to reveal application flaws that are fulfilled manually, or with automated tools, are described in Section 4.2 and answer to the question *RQ-2.1*. Then, the *RQ-2.2* answers how the results of vulnerability assessments are stored in a blockchain platform in Section 4.3, introducing the structure of reliability smart contract. Finally, the *RQ-2.3* is explained, when the persisted results of the reviews from blockchain storage are used for validation reliability of a resource in essence (Section 4.4).

4.2 Vulnerability detection

The code review aims to detect security flaws in the WEB-application, related to its design and features, along with the exact root causes. One must understand the code of the application, configurations and external components to have a broad view for finding the flaws. This kind of deep dive into the application code also leads to determination of specific mitigation techniques against security vulnerabilities. The common review methods of identifying vulnerabilities in software are [68]:

- The *source code scanning* by running automated tools that are used against a source code repository, or module. It searches for string patterns, considered to potentially cause security vulnerabilities.
- An *automated penetration testing* (black/grey-box) by using penetration testing tools with automatic scans, when the tool is installed separately on the network, and runs a group of predefined tests against the WEB-site URLs.
- The *manual penetration testing*, again using tools, however, with the penetration testing expertise, who executes more complicated tests.
- The *secure code review* by a security subject matter expert.

It should be noted that any method cannot detect all the vulnerabilities, but a defence-in-depth technique will reduce the risks of their occurrence. Overall, all these testing approaches must be combined, to get maximum benefit. In terms of reliability assurance of the resource code, every expert, conducting the review, should perform one or more testing approaches and properly report them with obtained results.
This section is dedicated for identification of the best contemporary approaches in code review process, by grouping all the above code review methods into two sections – tool-supported in Section 4.2.1 and manual in Section 4.2.2 reviews.

4.2.1 Tool-supported review

Currently, the code review processes are simplified by variety of automated tools. Mostly they are static code analyzers that attempt to highlight possible vulnerabilities within static source code. Ideally, they should automatically find the security flaws with a few false positives. It means the confidence level of the tool on finding the bugs should be quite high. In practice, this ideal status for the majority of tools is beyond of state of the art for many types of software security flaws [68]. Therefore, this kind of tools frequently serve as aids for security analysis, to make the review process more effective by detecting the portion of vulnerabilities relevant to tools' capabilities, rather than finding them all manually.

Source code scanning tool

Choosing a static analysis tool is a hard task, as there are many choices. There are different criteria for choosing automated tools [68]:

- Does tool support the programming language, utilized in software application?
- Is the tool commercial, or free? It depends on the preferences, usually, the commercial tools have more features and reliability, than free ones.
- What type of analysis is carried out? It could be quality, security, dynamic, or static analysis.

Additionally, the best tool is selected depending on non-functional, but important aspects that it has, such as user experience, reporting of vulnerabilities, level of false positives, customization and maintenance.
There are some *advantages* and *disadvantages* of source code scanning tools that are shown in Tab. 5 and Tab. 6 respectively [68].

Table 5. *Advantages to use source code scanners*

Advantages	Explanation
Reduction in manual efforts	There are some common patterns that are used in codes of applications. The tools are better than humans for scanning such things. They are faster and effective in this scenario.
Find all the instances of the vulnerabilities	The scanners are very helpful in detection of all instances for a particular vulnerability with exact location. It is suitable for the large code-bases, where tracing of flaws is a hard task.
Source to sink analysis	This methodology identifies the possible inputs to the application and traces them all, throughout the code until any insecure code is found. It helps to understand the vulnerabilities better, as it is considered as a complete root-cause analysis.
Elaborate reporting format	The tools give a detailed report on the scanned vulnerabilities with exact risk rating, code snippets and full description of a problem.

Although, the code scanning tools are effective in finding insecure code patterns, they lack of ability to track data flows. This problem is solved by code analyzers, that allow full, or partial compilation of the source code to identify flaw patterns, implement source to sink analysis. A comprehensive approach is the combination of static code scanners and analyzers for a successful code review.

Table 6. *Disadvantages to use source code scanners*

Disadvantages	Explanation
Business logic flaws remain untouched	The code portions that are specifically related to the application, due to implemented features and design, are frequently not pointed out by the scanners.
Limited scope	Since the automated tools are designed to search by patterns, outside this scope they are limited. The static code analyzers are commonly used to check particular frameworks, or languages and able to search for a certain group of flaws.
Design flaws	The scanner/analyzer do not spot the design issue, as they are majorly focused on code expressions. Whereas, humans instantly identify design issues in the implemented code.
False positives	Not all of the flagged issues by the static analyzers are truly issues. Thus, the experts need to understand and propagate the results, executed by these tools.

4.2.2 Manual review

A secure code review is probably the single-most effective approach for detecting security bugs in a software product. When it is used together with the manual and automated penetration testing, code review can remarkably evolve the cost effectiveness of an application reliability assurance.
A manual code review gives an insight into the *real risk*, related to insecure code. This contextual, *white-box* technique stands as the most significant value. A human reviewer requires an understanding of what is being assessed in the context. With the relevant context we can conclude a serious risk estimate on both the business impact of a problem and likelihood of an attack [68].

Manual penetration test

There is a term *360^o review* that is executed in a cyclic manner by usage of code review outcomes in penetration testing, and the results of penetration testing, in-turn in additional source code review, illustrated in Fig. 16.
If the reviewer knows the internal code structure of a resource, this knowledge could be used to form test cases, known as *white box testing*. This approach leads to better and productive penetration testing, as it is more focused on already known, or suspected vulnerabilities. For example, the knowledge about specific frameworks, libraries and languages could concentrate the testing on particular weaknesses.

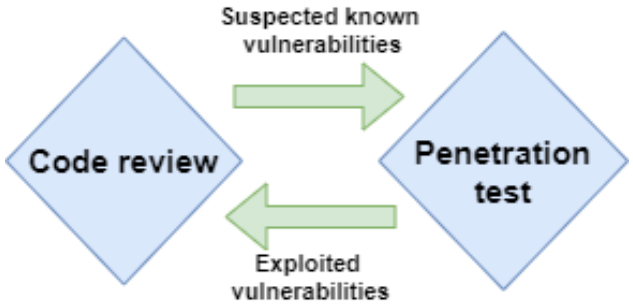

Figure 16. *Code Review and Penetration Testing interactions*

On the contrary, the white-box penetration testing can also be modified with the actual cases of risks that pose the vulnerability, discovered in code review. A vulnerability, found through a code review, may appear not to be exploitable during penetration testing, while it carries an actual risk indeed [68].

4.3 Reliability contract

As soon as a peer-reviewing on the resource code has been accomplished, the reviewer should persist all the collected results for further investigations by other experts. The storage process of the report is designed in two steps. Firstly, an establishment of the *general report* in a separate file system, which is described in Section 4.3.1. Then, a *brief report* and its table structure, being a part of the *reliability* smart contract, persisted in a blockchain and referred to the general report, given in Section 4.3.2. Eventually, the *reliability action* is determined in Section 4.3.3 that explains the way a brief report is stored in a blockchain.

4.3.1 Report structure

A template of the general report should provide enough information to enable the other code reviewers to classify and prioritize the software vulnerabilities during their investigations. This report does not need to be pages in length, it can be also documented, based on many automated reports of code review tools. The general report could contain the following information:

- Date of a review;
- Project name and code modules reviewed;
- Developers and reviewers' names;
- Reference to the reviewed code repository;
- The brief sentence(s) to prioritize and classify the software vulnerability, if any exists;

- Code review checklist, if any is used;
- Results of tests that are carried out in the code. They are preferably the unit, or automated tests.
- Results of any static analyzer/scanner tools that have been used during the code review.

The main reason, why the general report is stored separately and later linked to its brief representation in the blockchain, is to provide a reviewer with more flexibility, avoiding the redundancy in the storage of a blockchain. Those full reports could, in-turn, be stored in a decentralized and distributed file system, such as IPFS[1] [87].

WhisperKey report

Since the running case for this thesis is *WhisperKey*[2] WEB-application, an example of reliability report is shown in Tab. 7, based on its publicly available source code. In order to conduct an automated code review, the corresponding source code scanning tools should be identified. The *WhisperKey* is developed using *Ruby*[3] programming language. There are various commercial and free static analyses tools for software products, written on *Ruby* [88, 89].

Table 7. *Sample general review report for WhisperKey*

Date	18 Mar 2020
Project	WhisperKey
Reviewer	Bob Ross
Code repository	`https://github.com/pixielabs/whisper-key`
Vulnerabilities	Not detected
Checklist	Not used
Unit/Automated test	Not used
Analyzer/Scanner	Minor styling and linting issues are detected. Reports: ■ Brakeman – `https://yadi.sk/d/QwQ8VVUs7egpDw`; ■ Reek – `https://yadi.sk/d/uh9aSPM_IiQ3Iw`; ■ Rubocop – `https://yadi.sk/d/rybp761fUv9NDw`.

We have selected three open-source tools, such as Brakeman[4], Reek[5] and Rubocop[6], and run against *WhisperKey* source-code. They generate human-readable reports, which can be included as parts of the general report. In this case, the actual results of analyses are only

[1]IPFS `https://ipfs.io`
[2]WhisperKey `https://github.com/pixielabs/whisper-key`
[3]Ruby language `https://www.ruby-lang.org`
[4]Brakeman `https://github.com/presidentbeef/brakeman`
[5]Reek `https://github.com/troessner/reek`
[6]Rubocop `https://github.com/rubocop-hq/rubocop`

linting and styling warnings, whilst security analyses pass successfully. A full report of review is stored separately in a file system. Later, this general report is referenced by brief entry in the blockchain, which is described in Section 4.3.2.

4.3.2 Report table

Like a resource integrity contract from Section 3.3, the resource reliability contract has in-memory table structure and called as *reports* table. The *reports* table is the base structure that represents review-related brief reports on WEB-resource entry in a blockchain (Tab. 8).

Table 8. *Reports table structure*

Col. Name	Type	Description
id	`uint64_t`	Primary key
resource_hash	`checksum256`	Hash of the reviewed resource file
user	`name`	Reviewer identity
report_uri	`string`	General report reference
description	`string`	Brief description of a review
verdict	`bool`	Final security verdict

The primary key that uniquely identifies every review report within the blockchain storage, is declared by unsigned integer value with capacity of 64 bits. Additionally, the table of reports has a secondary key, containing an information on resource hash-code, which is not unique, as a single resource could have multiple reports. Later, the secondary key is used in the application code for querying of all reports that are related to the given resource. All the reviewing experts, who are individually identified within the blockchain platform by name, are also attached to the report entry with this name. The reference to a general report in the form of URI along with the brief description are added to the single report instance, distinguishing the main outlines of a general report. Finally, given a verdict, which is a *Boolean* value, indicating that the resource is secured, if set to `true`, otherwise it is `false`.

WhisperKey resource reliability

As an instance, the code report, defined in Tab. 7 can be published to the blockchain in the following structure, displayed in Tab. 9. The review does not indicate any security problem, but only minor design issues that do not influence security, consequently, the final security verdict is `true`. Since there are multiple reports per resource, then every reviewer can conduct a specific scope of checks on the code. In this case, only automated code review is implemented, using static analyses tools. It also depends on experts' capabilities, e.g. some reviewers do not have enough resources, or knowledge to carry out either

manual, or automated review. Therefore, they might contribute by selecting one, or another methodology.

Table 9. *WhisperKey report entry structure in "Reports table"*

Col. Name	Type	WhisperKey
id	`uint64_t`	100
resource_hash	`checksum256`	`9d25a2fe4ef50412adab5ce864087fe3 3b142496dcc432af8b65e2ea668030f9`
user	`name`	Bob
report_uri	`string`	`https://yadi.sk/i/MdYPZqGXSDioVA`
description	`string`	Minor styling and linting issues are detected. Security issues aren't detected.
verdict	`bool`	`true`

4.3.3 Reliability action

Since resource reliability contract interacts with the reports table, a process of report persistence to given table is done by corresponding action. This action later becomes a part of the transaction, then passing several validations, it is immutably included in the block of a blockchain. As this process is mostly similar to resource integrity action, it has almost the same structure and called *post* action, however, the action handlers are different at all. The reliability (*post*) action comprises of number of steps that result with the report entry, stored in RAM of a blockchain node, illustrated in Fig. 17. Those steps are following:

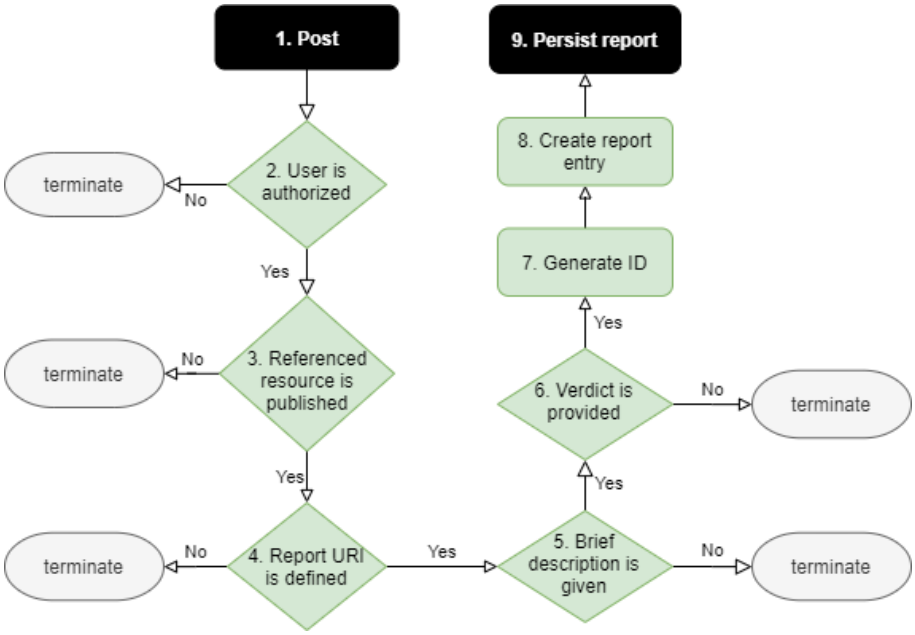

Figure 17. *Reliability action flow chart*

1. Handling the reliability (publish) action;

2. Checking the user's (reviewer) authority level that permits to commit an action within the reliability contract;
3. Checking whether the referenced resource is already published in a blockchain, or not;
4. Checking the validity of URI hosting the general report;
5. Checking the validity of a short description, outlining a full report;
6. Checking the validity of a given security verdict;
7. Generating a unique primary key for the report in a blockchain;
8. Creating an instance of the report that is coherent with the reliability contract;
9. Persisting the report instance in the blockchain.

If any of the given steps contradicts with the expected condition, or basically fail during the processing, then the reliability action is terminated straightaway.

4.4 Reliability validation process

The node based reliability realization was described in Section 4.3, whereas, the actual reliability validation process is accomplished on the browser side. The reliability assurance process is finalized, as soon as the integrity validation has been completed. If an integrity validation is succeeded, then the following steps are taken to fulfil reliability assurance: *posting*, *retrieving* and *validating* reports. This is the full cycle for resource reliability assurance, needed to be performed in the browsers. Thus, the report that is once posted to the blockchain, could be later retrieved for reliability validation with unlimited attempts. In this case, multiple reports are allowed to be posted per resource. The current section is split into two subsections: *posting the report* and *assuring the reliability* are given in Sections 4.4.2 and 4.4.1 respectively.

4.4.1 Assuring reliability

The complete flow for the reliability assurance is given as the sequence diagram and depicted in Fig. 18. The reliability validation is also a part of the full resource validation along with the integrity assurance. Therefore, it proceeds after resource integrity validation, while a resource execution is interfered until both validations are passed.
Subsequently, the reliability check is initialized, after the resource integrity validation has passed successfully, by calculating the hash-code and querying the blockchain with this hash. Firstly, the relevant reliability reports are retrieved from the blockchain by hash-code of the resource. Thus, the absence of reports states the given resource at warning level, and the further execution of the script depends on the user's decision, as the reliability is not

ensured. On the contrary, if reports do exist, and all of them are positively evaluated, then the resource is considered as reliable and permitted for execution on the browser.

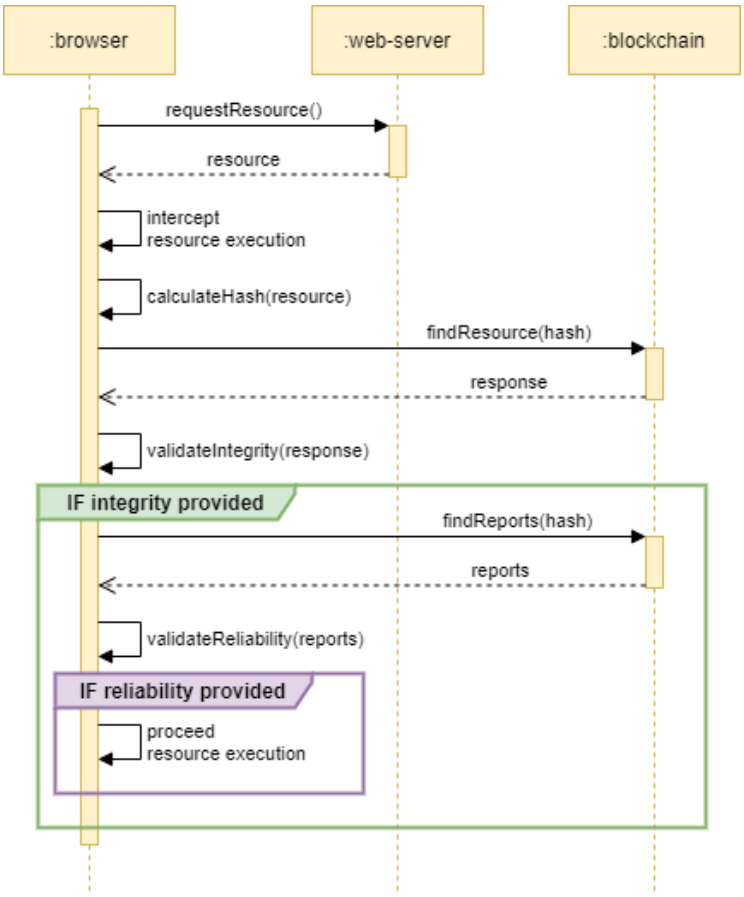

Figure 18. *Resource reliability validation – sequence diagram*

On the other hand, if there is at least one negative review, then the resource is marked as dangerous and decision on execution relies upon the user's browser. The process of review is done by the group of people, who are deemed as the trusted community of experts. Since a blockchain provides hierarchy of permissions, e.g. EOS.IO platform, division of users to users' groups is possible within a proposed artifact.

4.4.2 Posting report

The reliability of a resource is stated only by reviews, given by the experts of corresponding field. These reports are published to the blockchain, making them available for the other consumers of the resource. The posting process, being the part of the smart contract and called as a reliability action, is executed in a blockchain and initiated from the WEB-browser (Fig. 19).

At the browser-side, this operation starts after the resource integrity has been assured. All experts are allowed to add multiple reviews to the given resource. As the report itself is the brief representation of a general report, user is required to add only the resource hash-code,

reference to the full report, brief description and final verdict of a review.

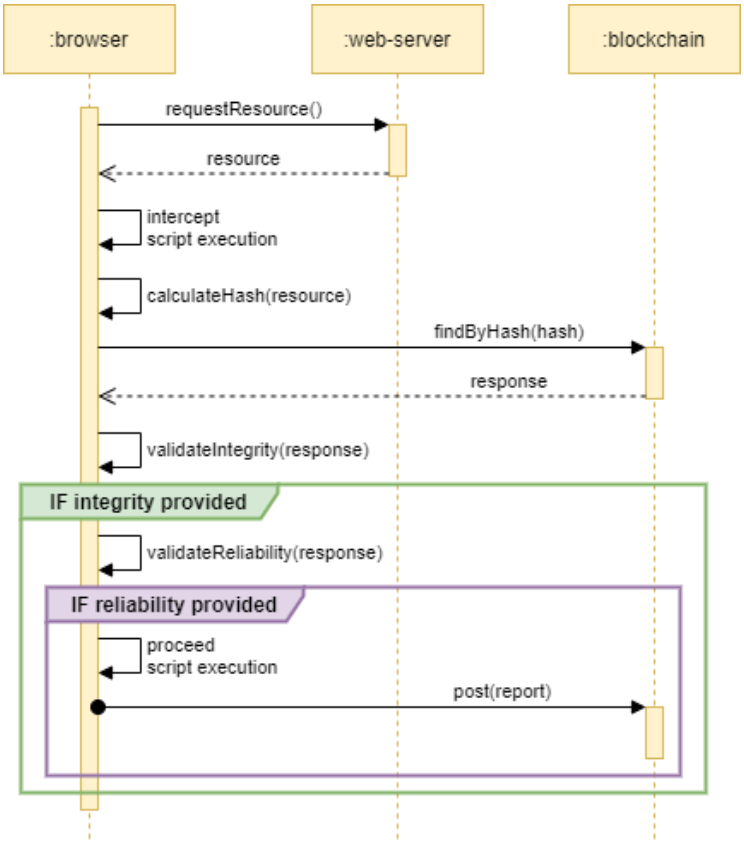

Figure 19. *Posting report – a sequence diagram*

4.5 Related work

An objective of Chapter 4 is to define the best practises for code reviewing and the way how it could be adapted in the browser-based decentralized application, using blockchain, in order to assure resource reliability. As long as both automated and manual reviewing methods are effective for different vulnerability types, therefore, the semi-automated manner of code review is proposed as the best combined approach.

A peer-reviewed resource is stored in two formats – by full report and its brief description. The latter one is stored in the blockchain. However, the general/full report has some flexibility for persistence, it might be any medium type, including centralized storage. A preferred type of storage for the general report is a decentralized file system (e.g. IPFS). The reason of selection over the centralized options is simply characterized by the security preferences in the artifact that is mainly given to decentralized architecture.

The main concern for the validation procedure, based on the provided reviews, is a conflict resolution. The security resolution that is built on the peer-reviewing, should have a trusted community, who does the review. Therefore, we propose to have a permission level in the system for users, who can mark the resource as reliable, or vice versa. Alternatively, more complex approach is to establish a platform that enables reputation-driven expertise with

the domain experts. For example, in *Semada* [72], experts make stakes for each validation in a betting-pool, based on their point of view on a specific issue. A decision is derived, relying on the majority votes in a betting-pool. Additionally, systems like CVE[7] are more suitable for third-party software, as they refer to commonly exposed vulnerabilities. A resource, which is referenced in CVE list of entries, is certainly marked as insecure [11].

4.6 Conclusion

Through out this chapter we state an artifact design on reliability assurance, based on a blockchain and integrated in browsers. In the previous chapter, we choose the EOS.IO blockchain for integrity assurance that supports smart contracts, it also perfectly conforms reliability assurance, as long as structurally they are more, or less similar.
A peer-reviewing process is conducted by the group of trusted experts, who publish their security observations on the resource-code to the blockchain. An information in transactions is persisted in a certain format, describing a report of peer-reviewing process sufficiently to obtain the security state of a resource file. This state is expressed by the short description and security verdict. All the reports in the blockchain refer to a single resource and are searched by the hash-code of corresponding file. The reliability of the resource in browsers is validated based on the reports, considering verdict of each report. This validation accomplishes the process of resource reliability assurance.
For the future work, more complex conflict resolution procedure on report analysis should be designed. Currently, the experts are the members of community, who are individually assessed and trusted to be part of it. However, an environment with the reputation-driven concept can be adapted, to make the reviews be possible among wider range of community.

[7] Common Vulnerabilities and Exposures (CVE) `https://cve.mitre.org`

5. Conflict Resolution & Incentive Mechanism

This chapter introduces the last major concepts that finalize resource reliability and integrity assurance. Since the audit reports proves resource reliability, the assurance of the reports must be proposed accordingly to obtain a full coverage of resource reliability. Therefore, voting mechanism is applied, creating a referendum per each report. In Section 5.2, the voting process for every posted report is described. Then, the users' groups, who are the end beneficiaries of a proposed artifact as well as users' permission levels are determined in Section 5.3. As soon as the groups are established and permissions are assigned, a reward policy, in the form of tokens is declared in the artifact, in order to give incentives to platform contributors. The latter is proposed in Section 5.4.

5.1 Introduction

Regarding to the research question *RQ-3*, this chapter provides an answer to it by splitting the main question into three sub-questions – *How to assure report reliability and incentivize reviewers for contribution?*

- **RQ-3.1: What is the conflict resolution process enabling report reliability assurance?**
- **RQ-3.2: Who are interested in revealing of the security issues?**
- **RQ-3.3: What is the incentive/reward for contributors?**

The following sections in this chapter answer each question individually. Previously, we considered that resource reliability is based on review-reports, however, the given reports also need the reliability assurance. In this case, a voting procedure is considered as one of the particular approaches. Especially, voting with the rewards, which play the role of a bet. Thus, Section 5.2 answers a question *RQ-3.1*, introducing voting process in given artifact that is combined with the reliability reports. It is equally important to understand, who are the users and what they are allowed to do within the artifact, like voting and reporting. It is the main context that is answered by question *RQ-3.2* in Section 5.3. Finally, users must be appealed to contribute by reporting and voting activities, operating with tokens as

a rewards. The last case is disclosed in Section 5.4, which is the answer to *RQ-3.3*.

5.2 Conflict resolution

The term *conflict resolution* reflects the possibility to determine the reliability of performed code review, applying voting procedure that is executed by the users of a platform. This procedure is simply implied by up-votes, or down-votes on given report by several users. There is a specific flow of referendum with time limitation and vote's threshold that provides with the final decision, based on the given votes.
In this section all the relevant referendum objectives are described for proposed artifact. In Section 5.2.1, a referendum protocol is observed that is derived from third-party implementation by the EOS.IO community. Furthermore, an adapted contract specifications of a referendum are given in Section 5.2.2. In the end, reliability validation based on the votes for reports is provided in Section 5.2.3.

5.2.1 Referendum system

Since the business logic of a blockchain application is built on smart contracts, a desired referendum system should be implemented in the same manner. In order to accomplish report-based referendum platform, the reference must be given to EOS.IO community that has already contributed different types of vote collecting systems [90]. As a base, the majority has selected an open source project, called `eosio.forum`[1], which is included into EOS Mainnet[2] as an official referendum system [91].

Workflow of voting

The referendum, based on `eosio.forum`, is held by several number of stages. Starting from the proposal of referendum and ending with memory cleanup, all the corresponding stages are shown in Tab. 10.

Table 10. *Referendum workflow*

Stages	Description
1. Proposal	Anybody with a valid EOS account and permissions can create a proposal in the related community. When this action is triggered, it consumes some volume in RAM (about 480 bytes) to save content of a proposal. The action is initiated with a proposer's identity, title and expiration timestamp of a proposal.

[1]eosio.forum `https://github.com/eoscanada/eosio.forum`
[2]EOS Mainnet `https://link.medium.com/OaZqt7dvu5`

Stages	Description
2. Voting	Once a proposal is published, users can start to vote on it. The votes are represented by *yes* (`1`) and *no* (`0`) values. As for the proposals, voters pay RAM (around 430 bytes) to save their own votes. The vote can be changed, or removed at any time until expiration of proposal.
3. Freeze	The proposal cannot continue infinitely, therefore, an expiration time is set, pointing the deadline of poll. It is defined directly on the proposal. Later, a proposer can also decide to expire proposal at any time, without waiting the default (six months) deadline. Once proposal has expired, it enters three days *freeze* period, meaning that it is totally locked and no actions can proceed. This allows all users to query blockchain data, count votes and obtain results.
4. Clean up	As soon as freeze period has elapsed, any user's account can clean used RAM, taken by expired proposal. The RAM is given back to proposers and voters accordingly. It is safe to clean expired data by anyone, as proposal has passed freeze time.

The concept of proposal should be clarified, so the proposal in given artifact is the report of a review. Once report is published, the proposal is considered active for a voting, up to three month until it is expired. A default expiration deadline is conditional, since it could be changed by the reporter after report was published. As soon as the voting is completed, all the users can obtain the final result of referendum. Based on the given votes, user decides whether to trust to security review, and correspondingly to WEB-resource, or conversely, block all possible proceeding flaws.

5.2.2 Report's reliability contract

Ideally, referendum system, proposed by EOS community, is quite general to carry many different options for diverse implementation cases. In this case, it should be adapted to peer-reviewing objectives, retained in the given artifact. Since the proposal and report imply the same concepts, the only missing field in the report table under reliability contract is its expiration date. Once this field is added, experts can initiate referendum on open questions, regarding the security aspects of a WEB-resource by publishing the reports.
The new contract that keeps referendum entries is called a *forum*. As all contracts in given artifact, the forum has in-memory table structure named *votes* table, and several actions that operate with votes table.

Vote table

The vote table should keep minimum needed information to define a users opinion on security question. An overview of vote table structure is depicted in Tab. 11.

Table 11. *Votes table structure*

Col. Name	Type	Description
id	`uint64_t`	Primary key
report_id	`uint64_t`	Voted report ID
voter	`name`	Voter identity
vote	`bool`	Either up-vote, or down-vote

The primary key uniquely identifies a given vote, and automatically generated whenever a new vote is added. A significant role is given to secondary key, which is the identification of a report, that could be assigned to multiple votes. The voter identity must be stored to prevent the same users to vote multiple times. Finally, the vote itself is represented by Boolean value, meaning up-vote if set to `true`, or `false` for down-vote.

Forum actions

The *forum* is a contract with the most number of available actions in comparison with an integrity and reliability actions. There are four of them: `vote`, `unvote`, `expire` and `clean`. This set of actions affects state of both *vote* and *report* tables. Being more detailed, each of them is shown in forum actions' flow chart in Fig. 20.

- **Vote** for a proposed review-report:
  1. Handle voting action;
  2. Check the user is correctly authorized to vote;
  3. Check report does exist;
  4. Check report has not been expired yet;
  5. Insert a new vote, or update the old one, given by corresponding user.
- **Unvote** to remove previously given vote:
  1. Handle clear a vote action;
  2. Check report does exist;
  3. Check report is not in freeze stage;
  4. Check the user has previously voted on given report;
  5. Remove a vote, given by the user.
- **Expire** the report, when poll has ended:
  1. Handle report expiration action;
  2. Check report does exist;
  3. Check report was published by the user, calling an action;

4. Check report has not expired yet;
5. Update a report, by modifying the expiration time with the current time.

- **Clean** RAM from poll leftovers:
  1. Handle referendum cleanup;
  2. Check report does exist;
  3. Check report has already expired;
  4. Check report is not in freeze stage;
  5. Erase all the votes from RAM.

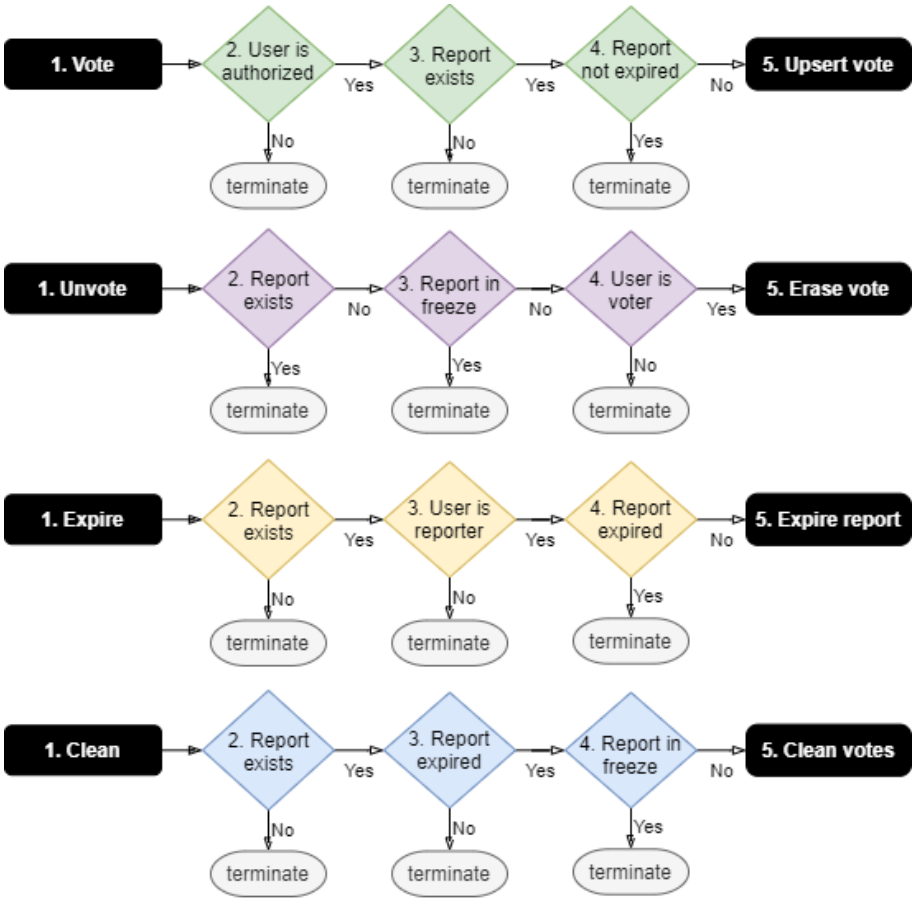

Figure 20. *Forum actions flow chart*

Rate inline action

There is an essential point that is not yet covered, regarding votes aggregation, when a report has expired and entered a freeze period for three days. This should be an action, which is triggered as soon as report expiration request has been approved. It aggregates all the collected votes on given report and updates the report entry in *reports* table with a *ratio* value. This value is a floating point number that is calculated by the equation 5.1.

$$ratio = \frac{positives}{positives + negatives}; \tag{5.1}$$

In this case, report has a positiveness rate on a claimed security review. In other words, if a report is labeled as insecure, and majority of the votes, given to the report, are positive, for instance 7 out of 10, then the rate of insecurity is equal to 0.7.
The *rate* action is an inline action, which can be called by other actions in a contract. The flow chart of this action is illustrated in Fig. 21.

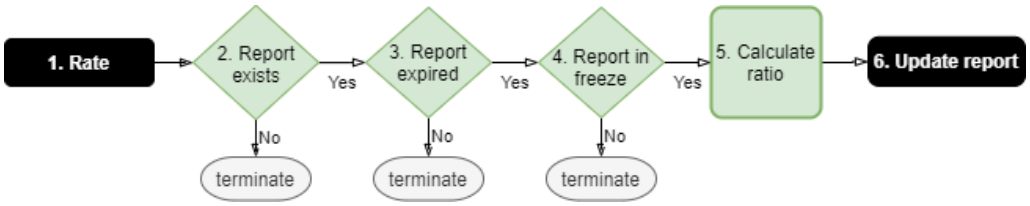

Figure 21. *Rate action flow chart for votes calculation*

1. Handle rate action for calculation of votes;
2. Check report does exist;
3. Check report has already expired;
4. Check report is in freeze stage;
5. Calculate the ratio between positive and all votes, given to corresponding report.
6. Update the report entry in reports table with ratio value.

5.2.3 Report reliability validation

If the resource reliability assurance depends on the security reports (Section 4.4.1), then something should be done to assure reports reliability on the browser-side. This is the main reason, why the positiveness ratio is the part of a report's entry. When a report is posted to the blockchain, it is assigned with a zero-valued positiveness ratio. Meaning that the report is not yet reliable, as it has not been reviewed by the peer-reviewers.

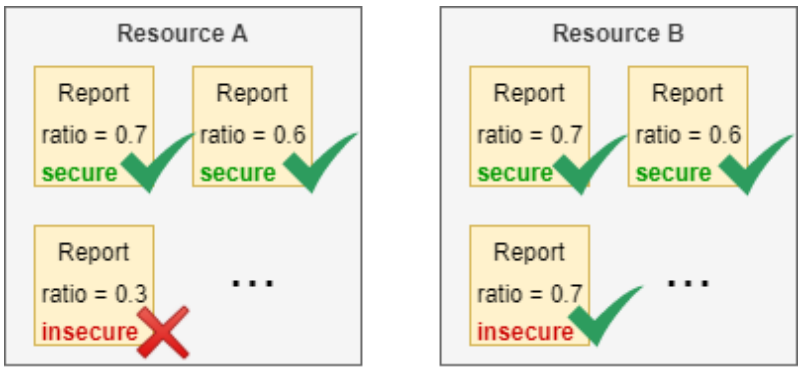

Figure 22. *Resource reliability with report ratio*

When it comes to the usage of a positiveness ratio, it might be configurable, depending on the user's preferences, but the decent default is minimal 50% limit. Thus, report can

be assumed reliable, if its ratio is above the given threshold. This step is included in resource reliability assurance flow. Specifically, when an integrity of the resource has been approved, the reliability check on related reports is conducted with ratio-based assessment. For example, if a resource is reported with three reviews, and two of them state that resource is secure with the ratio above 0.5 and one – insecure with the ratio below 0.5 (Fig. 22-A), then the resource is marked as reliable and allowed for execution on the browser. On the other hand, if all the reports are assigned with the ratio above 0.5, and there is at least one report, marked insecure (Fig. 22-B), then the resource is considered as vulnerable and its execution is automatically terminated.

5.3 User groups and permission levels

This section is about the parties, who are involved in publishing resources, posting reports and voting in polls, to obtain the consensus on integrity and reliability of a WEB-resource. There are several groups of users, who are involved in a given artifact, and who contribute and benefit from it in a certain way. Firstly, users' groups and their roles are described in Section 5.3.1. Secondly, the permission levels of EOS.IO environment are characterized in Section 5.3.2. Finally, the complex of users' groups and permissions is determined in Section 5.3.3, identifying the main authorities and their capabilities within an artifact.

5.3.1 User groups

The major aspect of design in any artifact is to determine users' groups and their roles within it. This allows to point out, who are the end users and their requirements. As an artifact is aimed to resolve resource integrity and reliability issues on WEB-browsers by putting the concept of experts driven peer-reviewing process in its basis, the users, also called interested parties, can be divided into three main categories: product *owners*, *reviewers* and *consumers*.

Product owners

The product owners, or resource developers are one of the users' groups, whose interest in proposed artifact is to be assured of absence of security flaws in their products. Which results in better user experience and trust, given to the service providers. This group of users can be responsible to publish the newest versions of their products within an artifact. However, this is not the restriction, since other users' groups are eligible to do the same, whereas, any new version of a resource is uniquely identified in the blockchain by calculated hash-code, based on the content of a resource file. Additionally, this group of users does not expect monetary interest in the given artifact, since there is no any important

contribution demanded from their side.

Reviewers

The reviewers are inalienable supporters of suggested artifact, being involved in direct code assurance procedure, claiming their thoughts on security rate of a product. In this case, their interest is monetary. These experts are mainly developers as well as quality assurance people, who perform specific assessments, disclosing various vulnerabilities in the WEB-applications.
Ideally, the offered artifact is a basis for complete freelance quality assurance platform, where users raise questions on security matters of any software product to obtain reputation advances, or take part in betting-pools, acquiring tokens. Moreover, they can publish resources and attend in polls, giving the votes to other reviews, published by peer-experts.

Consumers

The last group includes conventional customers, who benefit from exploiting secured services. Identically to service owners, consumers can also publish new resources to the blockchain, initiating the source for debates of peer-reviewers. Apparently, they may participate in voting process on any report, betting with tokens that they hold in owned balances.

5.3.2 EOS.IO permission levels

Conceptually, permissions are associated with the user's accounts, in order to authorize their actions and transactions in a blockchain platform. The EOS.IO environment provides a comfortable way to manage accounts and permissions. In EOS.IO platform, each permission is linked to authority's table that contains a threshold, which must be reached to allow an action execution, associated with the given permission. In Fig. 23, the relationships among permissions, accounts and authorities are shown.
Here, *alice*'s account has set of permissions along with their hierarchical dependencies, for example, the `active` permission level from this list is linked to corresponding authority's table. Subsequently, the threshold for an activation of this authority is set to two, since `bob@active` and `stacy@active` factors also have a weight of two, either one can match with the action's authorization. These factors are specifically assigned to the permission level under authority's table, and they are represented as a list of actors' public keys, account name plus permission level, and time wait [86]. Besides, it is also possible to create an implicit link between two accounts with the same named permission, though their weights must match.

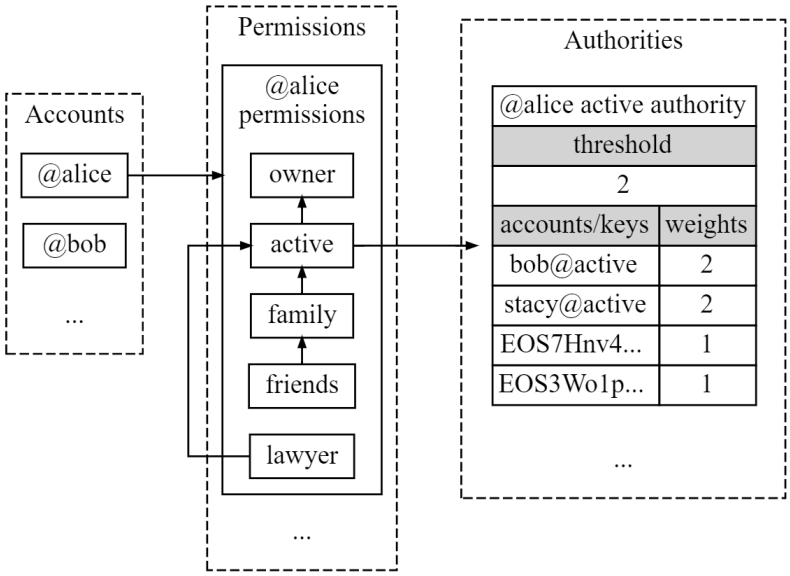

Figure 23. *EOS.IO relationship diagram of permissions, accounts and authorities* [86]

There are always two default permission levels, assigned by EOS.IO to newly created accounts: `owner` and its sub-level `active` permissions. The `owner` is supposed as root-level permission, and primarily used for recovery purposes.

5.3.3 Permissions hierarchy in artifact

In previous Section 5.3.1, three main users' groups are identified. There are specific as well as common roles among users, regardless the groups. For example, all groups' users can publish resources and vote on reports, but only reviewers are allowed to post reviews. Thus, two permission levels are mainly exploited within an artifact. They are named as `active` and `post` permission levels.
Since the `active` permission is a default, it is automatically included in all accounts. Thus, no additional permission for product owners and consumers are established. On the contrary, only users, who are intended to post reviews are assigned with a custom permission, named as `post`.

Reporting permission

In a blockchain platform, every contract has its own account, and whenever an action is triggered by user's account, it supposed as transaction between user and contract accounts. Firstly, new permission level should be created under contract's account, in this case, reliability account, which corresponds to reliability contract. A new permission level should reside under the `owner` level, for the purpose of distinguishing it from default `active` permission. An according list of accounts, related to the reviewers, is also added as permission factors during the creation process. Furthermore, a newly defined `post`

permission is linked to the post action under the reliability contract. Henceforth, if the users want to post a report on the resource security matters, they must be included in the relevant authority's list under the `post` permission.

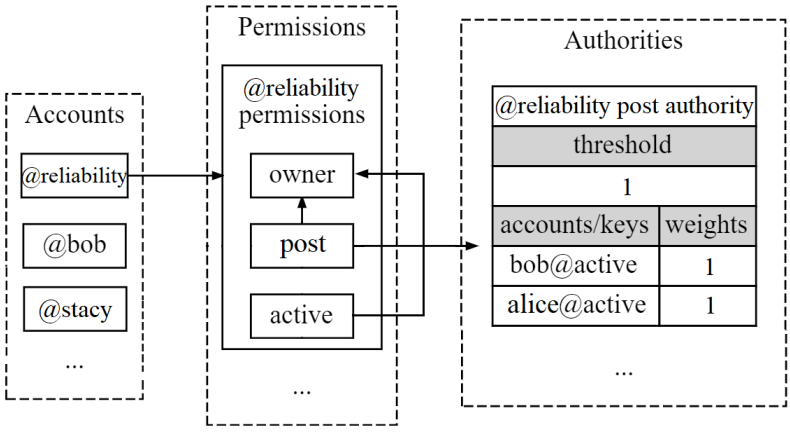

Figure 24. *Relationship diagram of permissions, accounts and authorities in artifact*

An example of permission's diagram for reporting process is depicted in Fig. 24. There are three users' accounts, along with the `@reliability` account. This account has a permissions table, where a custom `post` level is added under the `owner` permission. Moreover, `post` permission is linked to authority's table, where its threshold for activation is set to one, and two account factors are added as permitted actors for given authority. Eventually, regarding to the created permission hierarchy, only `@bob` and `@alice` are allowed to publish reports to the blockchain, whereas `@stacy` is prohibited, as she is not included in the list of `post`'s authority factors.

5.4 Rewarding system

As long as product owners have the interest in making their services more secure, the consumers and reviewers also need some incentives to attract them for contribution. The base concept is to charge some amount of stake from reviewers and voters, putting it into betting-pool, when they post reports, or vote to determine a consensus. Later, when it is achieved, depending on the decision, all stake from the pool is distributed amongst participants. This section provides a way the staking system is implemented in the artifact, introducing `eosio.token` project in Section 5.4.1. Then, Section 5.4.2 describes, the core concept, how the tokens are locked in betting-pools, while the review process is proceeding. Besides, the procedure of token generation and distribution is given for reporting and voting in Sections 5.4.3 and 5.4.4 respectively.

5.4.1 EOS.IO tokens

In order to enable token-based actions in the blockchain, some token management system must be implemented. As the artifact is built on EOS.IO blockchain, it is not included in the platform by default. Fortunately, EOS.IO comes with the system contracts that supposed as vital contracts for every decentralized application. This set of contracts contains the `eosio.token` contract that defines the structures and actions, which allow users to create, issue and manage tokens [92].
There are three main `eosio.token` actions that are used within an artifact: *create*, *issue* and *transfer*. The token creation action, basically, used on blockchain initialization to define the token issuer account, and set the maximum supply with the corresponding token symbol – local currency, for example, `1.0000 SYS`. An issuer, in this sense, is an account, who issues a certain amount of tokens to other users' accounts. It is done by another action of the contract, called *issue*. The last and mainly used action is the tokens *transfer*. This allows for one account to transfer tokens to another [93].

5.4.2 Betting pool

The betting-pool is the core of incentive mechanism that collects, locks and unlocks rewards in the artifact. All the users' groups can manipulate with the betting-pool in a certain manner. Firstly, the betting-pool starts by the initial rewarding tokens, which later are transferred to the participants as a reward, only if they succeed. The initial tokens, are proposed by the users', who are mainly interested in security assessment of the software. In the context of current artifact, these users are product owners and consumers. The betting-pool is initiated, when somebody publishes a resource to the blockchain.

5.4.3 Reporting reward

The initial stake is not the only source, how betting-pool is funded with tokens. Whenever a reviewer posts a report on the given resource, he/she stakes some amount of tokens into the pool. The community, then starts debating on the claim. If the claim is approved, the reviewer gets back an owned stake with some part of initial stake as a reward. On the other hand, if the claim is rejected, then the reviewer's stake is locked and added to the initial stake in the betting-pool for other reports.

5.4.4 Voting reward

The users, who are involved in debates are the potential voters. They also stake some amount of tokens on the particular claim, during the referendum. In the end, the majority voters get their own stake back with some amount of initial stake as a reward. Otherwise, the minority voters lose their stake, which is added to the initial stake in the betting-pool for further distribution.

5.5 Related work

Through out this chapter we try to determine the procedure for reaching a consensus, propagated to assured reliability on users' claims, regarding software resources. This is done by applying referendum concepts on security reports, related to WEB-resource. Whenever the resource is published to the blockchain for a review, the betting-pool is also created with an initial stake. The referendum is comprised of three major stages: creation of proposal, voting on proposal and counting the votes. The creation of proposal is devised as part of report initialization, since we keep the results of referendum, regardless the votes are removed, when referendum accomplishes. The posted review is accompanied with the fixed stake of a reporter, which is lost if the report is not approved by the community, who votes to reach report level consensus. The time when a consensus is supposed to be achieved, is defined prior on report creation, or later by the reporter.
There is a scenario, when a reporter posts the report and then immediately expires the referendum, when a few amount of votes are collected that coincides with the interests of a reporter. In order to avoid such situation, the permission levels are introduced in the artifact. More specifically, the `post` permission that allows only for a particular group of experts to submit reviews to the blockchain. This group of people are supposed to be members of trusted community of experts. Additionally, some time limitations might be applied, which prevent the reporters to expire a referendum during specified period of time, since it has been created. Alternatively, minimum threshold of votes could be set that must be reached to finish the referendum.
In parallel, the voters also stake their tokens, supporting any claims proposed by the reporters. It is the same strategy as for the reporting, if the claim is rejected by the majority voters, then the supporters lose their stake, or vice versa. All the lost stake either from reporting, or voting, is topped up in the betting-pool. In order to avoid the stake to be lost forever as unused in the betting-pool, when the referendum is finished, the leftovers are distributed among other resources as general buffer for other betting-pools. The distribution happens when a corresponding referendum is marked as expired.

5.6 Conclusion

In summary, the conflict resolution and incentive mechanism of this artifact is based on voting process and token game respectively. Three main users' groups is defined in this chapter. Primarily, the publishers, whose interest is security assurance of provided services. They are expected to publish the resource to the blockchain, and define an initial stake for the reviewers. Then, the reviewers are interested in earning tokens and be a part of the experts' community that might also open reputation based perspectives in a given field. They publish the reports and stake some tokens to state their point of view. Eventually, the consumers, who are interested in the usage of secured service. They might also publish the resources and put some stake for contribution. Additionally, they vote to attain the consent on reliability of a resource. All of these actions are followed with a token game, rewarding the proved contributions and charging the rejected ones.
Currently, the members of experts' community, who commit the reviews and initiate the debates are selected, based on the external invocation. However, systems like a *Steemit* [70], provide a solution with more advanced community structure, additional roles in the system, such as curators, witnesses, etc., and more flexible token distribution procedure by minting new tokens, based on the participant's activity level. Thus, the community is built on the conception of more contribution – more earnings. There are interested parties (stakeholders), who elect the content publishers by particular offers on profit, and create a market competition to earn the right to produce the feed. If the feed succeeds and gets the positive approval by voters, then they gain a revenue, otherwise they lose the stake [94]. For the future work, such a complex rewarding system is needed in proposed artifact, to let the community to define its members by themselves.

6. Evaluation

Since the evaluation is an essential activity in DSR, in the following chapter, the design science evaluation is applied on the proposed artifact. First, we determine a solution overview, put in the core and defining a business logic of an artifact in Section 6.2. Then, an implementation of the given artifact as a real system is described in Section 6.3. Finally, an evaluation on the real prototype is performed, based on the ISO-9126 software quality standard in Section 6.4.

6.1 Introduction

The FEDS framework [53] is employed in evaluation process of a current research. As was discussed in Chapter 1, we have selected *Quick & Simple* evaluation strategy, which produces summative analysis on posed artifact's functionality, based on the developed real system, or methods. We build a prototype, described in the Section 6.3, to make a naturalistic evaluation of a solution that is summarized in Section 6.2.
The main goal of an evaluation is to measure how well the proposed functionality in artifact fits the real world scenarios, and what new challenges and benefits it carries for potential users. Meanwhile, we analyze how reliable and efficient is the system, based on the developed prototype. Thus, the four main evaluation characteristics, such as functionality, usability, reliability and efficiency, are distinguished from ISO-9126 software quality standard and observed in Section 6.4.

6.2 Solution overview

The following section provides a conceptual overview of the designed solution and depicted in Fig. 25. It consists of conventional flow and proposed extension on it, which formalizes a new security protocol.
There are three roles in the proposed artifact: service *owners*, *users* and *peer-reviewers* of the service security. The owners, who can also be developers, produce resource-scripts, then publish them to the version control repositories, and deploy to WEB-servers, making a service available to internet users. As soon as a new version of WEB-application code has been uploaded to the repository, it can be published in the artifact and peer-reviewed by independent experts on security matters. These experts perform various analyses on

code, to determine whether it carries any threat to end users, or not. The results are posted to the blockchain in the form of audit reports. Those reports contain the hash of a reviewed resource, reviewer's stake, brief description and positive, or negative mark, stating the security level of a code. Apart from the brief description, a full report with various analyses are also published to the external storage. Once a security claim has been placed, all the platform users can vote on it, and decide by the referendum its reliability. On the contrary, when the user's browser requests the WEB-server for the resources, an execution of downloaded resources is intercepted and data is validated through the blockchain, based on the previously posted reports. Finally, if the result of a check is negative, then execution is terminated, otherwise compiler of the browser proceeds normally. Whole this process in the artifact is named as reliability assurance of the resource, built on security verdicts.

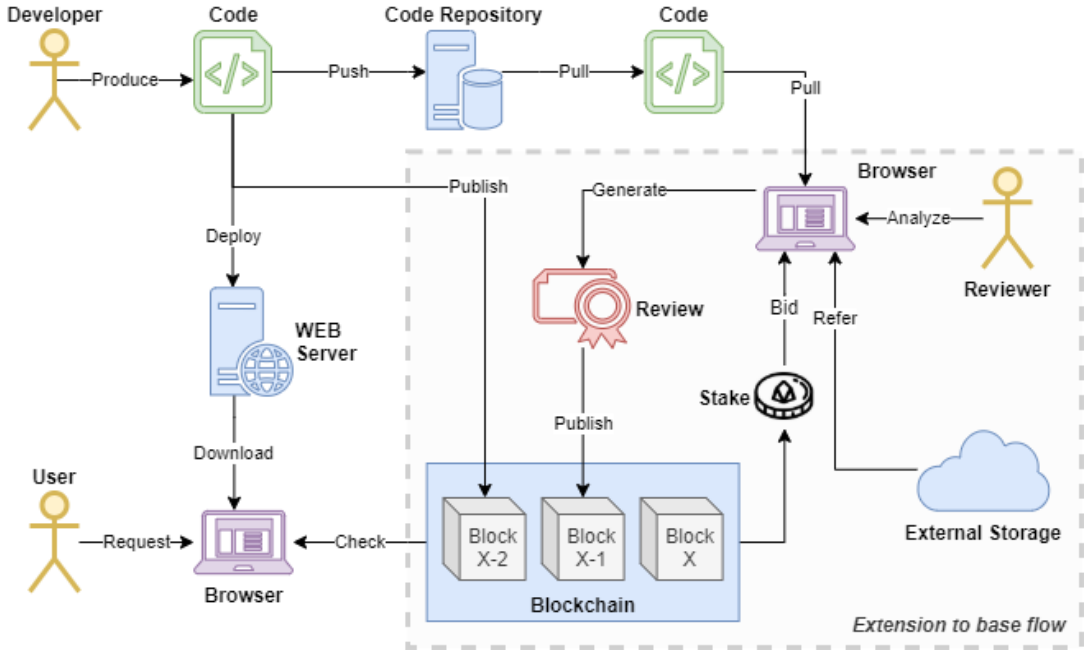

Figure 25. *General overview of artifact workflow* [11]

Additionally, designed artifact compares downloaded and calculated hash with the stored hash of a resource file from WEB-server and blockchain respectively, thereby, ensuring the resource originality. If the comparison ends up with inequality that means the source-file is tampered, and its state is dangerous. This process is called as resource integrity assurance.

6.3 Prototyping

The evaluation in design science research is considered crucial, as it provides the legibility on accomplishment of predefined research goals. In order to enable *summative* evaluation of the artifact, the assessment should be done in *naturalistic* way, using the real system [54]. In this case, an artifact must find an implementation in the form of prototype, collecting all the necessary specifications in it.

An implementation consists of two parts: a browser application and blockchain integration. An appropriate blockchain platform is already defined prior in the research, distinguishing EOS.IO platform in Chapter 3. EOS.IO platform provides with all needed tools to setup the development environment (Section 3.2.3), and start an implementation of a distributed application (DApp). As soon as related smart contracts are developed, regarding Chapters 3, 4 and 5, they are deployed to the blockchain, formalizing back-end of a DApp.
On the contrary, front-end of a DApp should have all privileges of a browser, to intercept HTTP(S) requests between the browser and server. Accordingly, a browser extension is developed specifically for FireFox browsers. The FireFox itself is the second most popular and open-source browser, it has required API for handling of in-browser requests, which is not yet supported in Chronium. Meanwhile, various tools are used, during the development of a browser extension, such as AngularJS, Bulma, Webpack, etc.

6.4 Case study analysis

The evaluation process is based on ISO 9126 standard, which is an international standard for evaluation of a software. It represents the latest research in characterizing software, identifying software quality assurance, software quality control and software process improvement (SPI) [54, 95]. Fig. 26 illustrates characteristics and sub-characteristics of ISO-9126 evaluation model. In following Sections 6.4.1–6.4.4, the relevant characteristics are adapted for evaluation of a prototype that is built on the designed artifact.

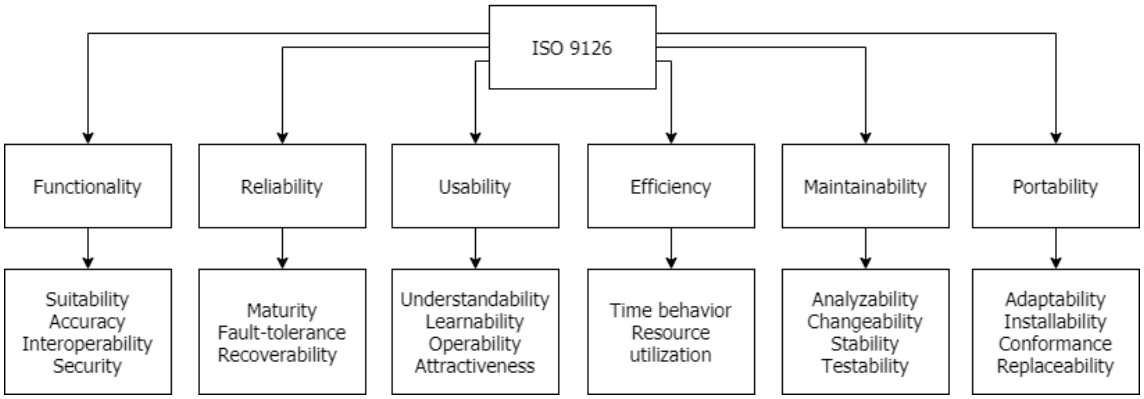

Figure 26. *ISO-9126 quality evaluation model*

6.4.1 Functionality

The functionality is the crucial purpose of any product, or service. Its characteristics allow to get derivations, regarding how well a software provides desired functions. The functionality can be used for assessment, control and prediction of the extend to which the software product in question meets the functional requirements.

Suitability

The suitability sub-characteristic allows to get results about how suitable software is for a particular criterion in artifact design. Generally, it answers the question: *Can software perform the tasks required?* [96].

One of the functional requirements in artifact is to provide ability for users to calculate the hash-code of WEB-resource, then publish it with relevant requisites to the blockchain. In Fig. 27, three stages of one of the WhisperKey resources is shown, using a prototype. The Fig. 27–A illustrates an initial and unpublished WEB-resource, which is then provided with details about hash-code, server and code-repository URIs in Fig. 27–B. Eventually, it is published to the blockchain, resulting in change of the resource state, defined as a tag under the name of corresponding resource file in Fig. 27–C.

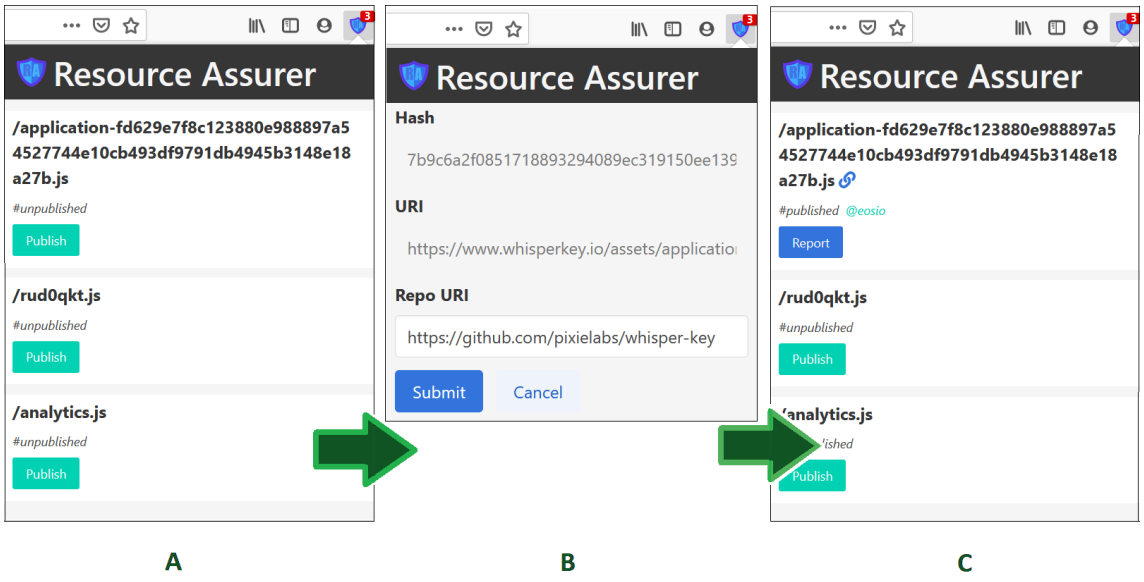

Figure 27. *User interface of integrity assurance process*

Since the resource is published, user is able to post a security report on it. Thus, in Fig. 28–A, only previously published WhisperKey resource is available for reporting. The report form in Fig. 28–B corresponds to the minimum requirements, set in artifact for reliability audit. Subsequently, when the review form is submitted to the blockchain, it becomes available for voting, as shown in Fig. 28–C. Here, users are allowed to vote on report, until it is expired by reporter, or time threshold.

When a report referendum expires and voting ends in Fig. 29–A, users can observe corresponding reliability rate on the resource representative in Fig. 29–B.

All in all, suitability of an artifact is successfully evaluated, stating that the proposed artifact's design allows the development of inherited prototype, which is capable to publish both resources and related reports as well as enable voting process on reviews.

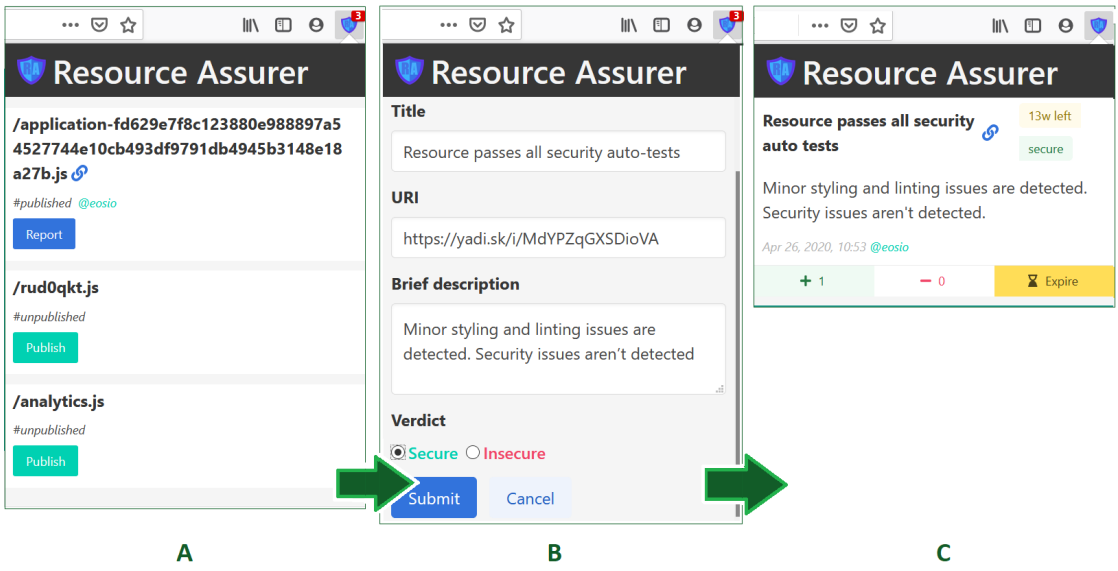

Figure 28. *User interface of reliability assurance process*

Accuracy

The accuracy sub-characteristic allows to get conclusions, regarding how well the software achieves complete, or acceptable results. Generally, it answers the question: *Is the result as expected?* [96].

In terms of integrity and reliability concepts in artifact, the first evidence of accuracy is appearance of a resource with its details and corresponding reports in the form of transactions inside of the blockchain, and the possibility on client-side to fetch, given resource and its security reports by the hash-code. Furthermore, when the resource and related reviews are fetched from the blockchain, the reliability and integrity of a resource should be validated based on the given data.

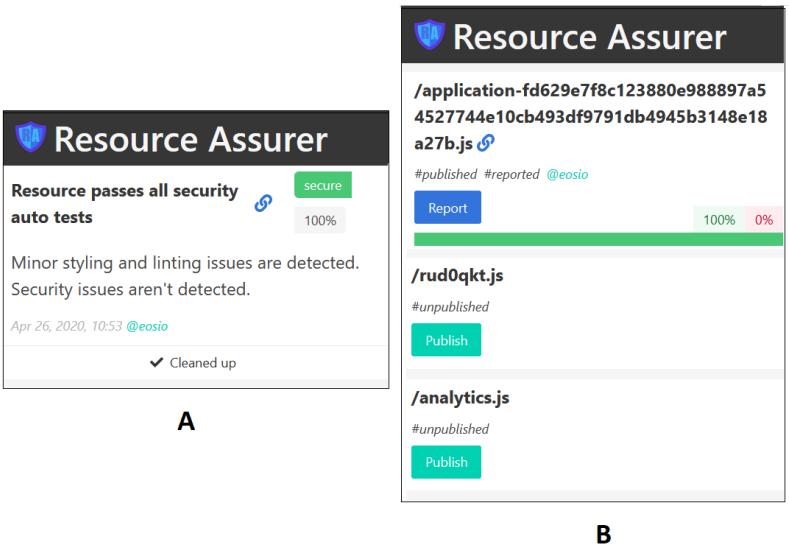

Figure 29. *User interface of complete security assurance*

Thus, EOS.IO platform has a tool, named as *EOS Studio*, which facilitates with utilities to interact with the platform and track all the manipulations in it. It also provides with the transaction history of any contract. The transaction history is given in Fig. 30, regarding to actions that are taken during suitability evaluation. There are in total four actions: publish, post, vote and expire; proving the persistence of all interactions with User Interface (UI) of a prototype, as transactions appear in the blockchain. On the UI side, the results of fetching from a blockchain are represented in Fig. 27-C for integrity and 28-C for reliability assurance. Since it is possible to retrieve a stored resource by its hash-code on the browser extension, this enables integrity validation. On the other hand, the ability of retrieval of reports by resource hash-code, enables reliability validation on the browser-side.

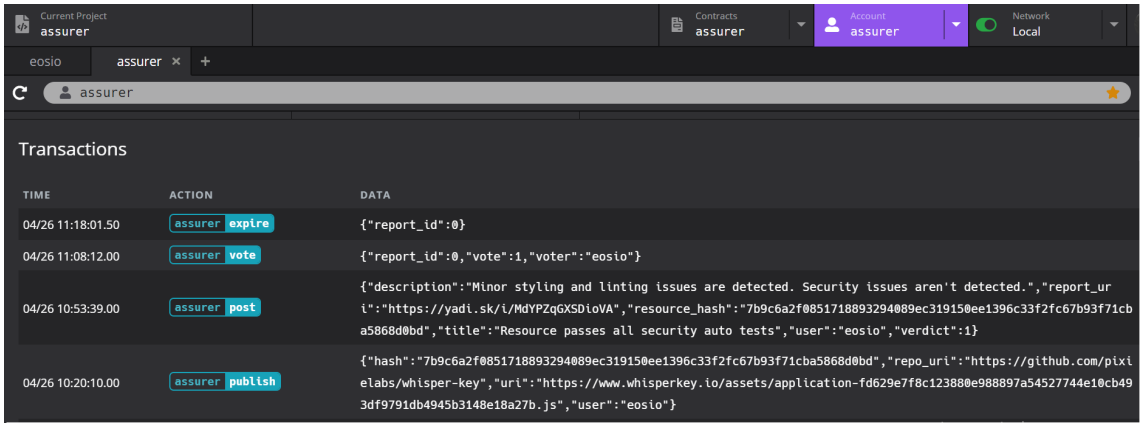

Figure 30. *Prototype transaction history*

Security

The security sub-characteristic allows to make conclusions upon how secure software is [96]. The security in a system is provided on the level of a blockchain platform. It is impossible to interact with the system without an authorization. All the users' accounts are bound with the public and private keys. Each account is created with the generated public key of a user. A user keeps the private keys locally, and uses them to interact with the blockchain. Only a user with the particular account and permission, existing in the scope of prototype's environment, is allowed to emit transactions into the blockchain.
On the contrary, a browser extension is also a JavaScript application that allows users to fill input forms, which then rendered in the browser. This might create a possibility for misuse from intruders' side. Therefore, AngularJS framework is utilized, which does security sanitization on user's input and context rendering.

Interoperability

The interoperability sub-characteristic refers to the diverse technologies that are used by the designed system and how these technologies interact with the end user's system [97]. The core of a proposed system is the blockchain and its business logic is represent by

smart contracts. This allows to different user interfaces to benefit from decoupling by implementing various templates of integration with the system. It provides API integration for various platforms, like browser extensions, native applications, etc.

6.4.2 Usability

The usability characteristic allows to make conclusions, regarding how easily software can be learned, understood, used and liked by the users and developers. Also usability only achieved with regard to functionality and refers to the ease of exploitation for a given function [98].

Understandability

The understandability sub-characteristic allows to determine how well users can recognize the logical concepts and applicability of a software. The users should be able to make decisions on selection of a software product, which is suitable for their intended use. Basically, *does the user comprehend how to use the system easily?* [96].
This is possible by providing users with enough literature, tutorials and interactive help functions, or contacts on support [97]. Currently, this thesis is only full requisite, pertaining whole idea and design specifications behind the artifact. Besides that, code repository of the prototype, carries sufficient information on running and building a complete working environment, including blockchain platform as well as the browser extension.

Learnability

The learnability sub-characteristic allows to evaluate how well users can learn the application of software. In other words, *can the user learn to use the system easily?* [96].
The UI of a prototype can be referred as ordinal single-page WEB-application, with a few number of pages available for routing. For better learnability, UI template provides user-oriented hierarchy of WEB-pages, starting from the list of resources, included in currently opened tab of a browser (Fig. 27–A), and ending with the list of reports under the selected resource representative (Fig. 28–C). There are specific buttons on both pages, allowing to open input forms for creation of corresponding entry, shown in Fig. 27–B and 28–B.

Attractiveness

The attractiveness sub-characteristic allows to evaluate how attractive software is to the user. Generally, it answers the question: *Does the interface look good?* [96].
The prototype, being a browser extension, does not use any multimedia, or hypermedia,

which are considered as relaxed user interface [97]. Nevertheless, the use of text is always the basic approach of product presentation. The text based context in the prototype is kept short, without errors, and readable, so the end-user is capable to look at the text and grasp the context within a few seconds. The core functionality is presented by means of conventional approach for WEB-application's design that includes, navigation bars, buttons, input fields, progress bars, etc.

Operability

The operability sub-characteristic allows to conclude about how well users can operate software, setting the question: *Can the user use the system without much effort?* [96].
Starting from the application launch, it does not need any comprehensive setups of blockchain nodes. The browser extension could be configured to connect either local node, or testnet, or even mainnet of corresponding blockchain vendor [99]. A prototype itself is a pack of HTML, JavaScript and CSS files with the manifest JSON file, which are referenced by the browser's compiler during the application launch. The only restriction is the browser options, which is limited on FireFox.

6.4.3 Reliability

The reliability characteristic allows to estimate how well software can maintain the level of system performance, when used under particular conditions for a stated period of time [96, 97]. The reliability relates to error-free and transparent user experiences, but also ability to perform under bottleneck situations. It refers majorly to system tolerance on end-users' actions and maintains the accuracy on the delivered information. Therefore, reliability of the systems mainly means reliability of the provided services and refers to system maturity in interacting with the end-user, staying tolerant to end-users' errors and keeping a high level of recoverability [97].

Maturity

The maturity sub-characteristic considers the extend how mature system is. In other words, *have the most of the faults in the software been eliminated over the time?* [96].
There are two sides of this evaluation, which is divided between the EOS.IO platform and prototype. Since, the prototype is quite new and have not been completely tested in production environment with real users, obviously, its maturity is not defined in the scope of this thesis. However, the blockchain platform was released to mainnet in June 2018 and has already around millions of accounts and billions of transactions. Furthermore, recent years have witnessed several attacks against EOS.IO, mainly exploiting vulnerabilities in DApps. This has followed by millions of dollars lost [100]. Overall, many bugs are found

and mitigated, increasing the platform's maturity over the time.

Recoverability

The recoverability sub-characteristic allows to draw conclusions according to how well software recovers from faults, or infringements of its specified interface. Putting differently, *Can the software resume working state and restore lost data after the failure?* [96].
Fortunately, all the core logic of a prototype relies on smart contracts, and if during the transaction life-cycle any of the validations fail, the transaction is rolled back without any data loss.

Fault-tolerance

The capability of the software to remain in a specified level of performance, in case of software failures, or infringement of its particular interface. In other words, *Is the software capable of handling errors?* [96].
Since, the reliability of a system at most depends on the blockchain platform, and the recoverability stands in the front of every user failure, the fault-tolerance is implied by different kinds of validations within smart contracts before the transaction. Most of the expected users' errors are handled on UI side of a prototype, preventing it reaching the blockchain node. However, all the thrown exceptions on blockchain are highlighted properly with description on reasons to the user.

6.4.4 Efficiency

The efficiency characteristic is concerned with utilized systems' resources, providing the required functionality. A good indication of this characteristic could be the amount of disk space, memory, network etc. [98]. An efficiency relates to a state, where system functions are both successful and usable, i.e. they fulfil their goals that the reason of their existence [97]. One of the major criteria of efficiency in the proposed artifact is the quality of sub-characteristics, regarding to time and resource behaviour.

Time behavior

The quality characteristic of efficiency is especially refers to the time factor; browser request/response-based systems should be developed with *time* caution, as one of the major parameters. From the user's perspective, time behaviour is one of the most essential measures, used to evaluate the quality of system's performance [97].
Two types of request-flows are used under the prototype's logic: (a) browser requests the resource server for scripts and awaits for response, (b) while browser extension intercepts

the response and makes another request to the blockchain, to assure integrity and reliability. This is a synchronous process, taking a bit more time than ordinary request flow. Thus, under the hood, browser downloads every resource file asynchronously, making it possible to parallelize the validation of each resource concurrently. The selected blockchain architecture supports concurrent handling of incoming requests, as it is built on Node.js, which is prominent with the specifications on parallel execution [101]. With a local test node it takes less than half of a second to download a single resource requisites (resource identifiers and reports) from a blockchain platform. This is achieved by internal indexation of all resource-related entries in the blockchain by the value of a resource checksum. Coming to the speed of transactions in the EOS.IO platform, in average it can handle up to 3000 transactions per second on a single thread of a CPU [102].

Resource utilization

This sub-characteristic of an efficiency evaluation states the capability of the software to utilize appropriate resources in an appropriate time, during the software performance under particular conditions [96]. The management of system resources is an important factor for defining the quality in DApp systems, and in current implementation it refers to RAM, Network and CPU.
The browser extension consumes majorly the resources related to browser storage. All the fetched resource information from the blockchain is temporarily cached in browser extension storage. In this case, the default upper-bound memory limit is five megabytes of RAM storage. The blockchain does not operate with large amount of data per transaction, therefore, every resource entry takes approximately 500 bytes of browser storage.
In the blockchain, every account consumes all three mentioned resource types. The RAM, in EOS.IO platform is a crucial resource, which acts as permanent storage, like a database. If a contract needs to store data to blockchain, then it can store it in RAM [81]. With the current implementation, every resource entry would cost around 500 bytes in blockchain RAM. The CPU, on the other hand, represents the processing time of an action, which is temporary consumed on action or transaction. Finally, the network bandwidth is also utilized, when an action, or transaction is sent [103].

6.5 Discussion

Throughout this chapter, the evaluation on functionality, usability, reliability and efficiency are performed, using an artifact's prototype. An evaluation of a functionality relates to four paradigms: suitability, accuracy, security and interoperability. In suitability and accuracy assessments, both of the main artifact concepts – reliability and integrity assurance are demonstrated as employable procedures that result in persistence of relevant transactions

in a blockchain, proving the validity of an artifact design. Additionally, the security of an implementation and its capabilities to integrate with other interfaces are provided. They are stated by authorization procedures, built in the EOS.IO platform, and separation of business logic from user interface respectively.
Moreover, usability of the system is evaluated, referring to understandability, learnability, attractiveness and operability characteristics. They are mainly related to thesis-based documentation of an artifact and user interface structure of its prototype.
Then the reliability specifications of a system are analyzed, determining its maturity, recoverability and fault-tolerance. Since a browser extension part of the designed prototype is quite new development, initiated within current thesis, the used blockchain platform is mostly reflects reliability evaluation, demonstrating long term process of bug fixing, fallback particularity of transaction flow, and validation of expected user errors.
Eventually, an efficiency of a new security system is built on time and resource utilization. The time characteristic is analyzed, considering time lapse of request/response among browser, server and blockchain. Meanwhile, the average amount of resources, used by browser extension and blockchain are assessed, observing RAM, bandwidth and CPU consumption.

7. Conclusion and Future Work

This chapter summarizes the thesis, concludes the research efforts and answers the main research questions, initially outlined in Chapter 1. Section 7.1 introduces a general conclusion of this thesis, then Section 7.2 provides the answers for each research question independently. Afterwards, Section 7.3 defines the limitations of this thesis. Eventually, Section 7.4 provides an outlook on the future work.

7.1 Conclusion

This thesis is dedicated to bring a solution, which is majorly focused on client-side security. There are two the most possible pitfalls in every system: an application code and environment. They can lead to unwanted consequences, putting a user under the security danger. The proposed artifact handles both threatening factors, by two design approaches. Firstly, it establishes a platform for quality assurance by experts, who evaluate the security aspects of given service, conducting the peer-reviewing on its source code. This process provides the trust on application code. In addition, an artifact stores the resource-related entries, which then are utilized to identify the original resource from modified one. Thus, ensuring the trust in security of a service on environment level. If any of this validations fail, the resource is treated as threat and rejected for execution on the client-side. All the heuristic solutions in the artifact are based on Trivium approach by raising and answering the questions to establish clarity and consistency in the artifact.

7.2 Answering research questions

In Chapter 1, we have stated the main research question: *How to assure the reliability and integrity of web-resource using blockchain technology in web-browsers?* For the purpose of reaching the answer, we have divided it into three sub-questions. Subsequently, the summary of answers for each question is defined below.

RQ-1: How to assure resource integrity in web-browsers using blockchain technology?

The resource integrity in WEB-browsers is assured, based on the calculated hash-code checksum of a resource file. Initially, this hash-code with references to code repository and

hosting server are published to the blockchain. The resource checksum allows to uniquely identify a resource in the blockchain. Whenever the resource is downloaded again from the WEB-server, its hash-code is calculated and used to retrieve its representative from the blockchain. If it is found, then the resource integrity is assured. Otherwise, a resource is not allowed to be executed on the client-side without confirmation of a user.

RQ-2: How to assure resource reliability using code-reviewing process?

The resource reliability of a WEB-resource is based on the security reports, given by the field experts. The code-reviewing is fulfilled in unrestricted form, e.g. every expert can publish the fully composed report in the external storage and reference it with short description and security verdict, which is linked to a resource and posted to the blockchain. Afterwards, if the resource integrity is assured successfully, then resource reliability assurance is started. The validation is built on report verdicts that state security, or insecurity of a WEB-resource. If there is at least one confirmed and negative verdict, then the resource is marked as insecure and its execution is terminated on the client-side. The resource is considered reliable, if all the related reports are positively submitted on security terms.

RQ-3: How to assure report reliability and incentivize reviewers for contribution?

Apart from resource reliability, there is a notion of report reliability, which stands for conflict resolution process on discovery of consensus, regarding validity of security reports. Since expert are human-beings, regardless they are trusted parties in the platform, they can still make faults, therefore, some evaluation system is applied to confirm the reliability of a report. This is a referendum system, where different users within a platform can vote on their point of view, related to the security claims. If the majority votes prove reliability of a report, then it is considered valid for security assessment on corresponding WEB-resource. The token-based incentive mechanism is introduced in the artifact. There are built-in betting-pools in the artifact, allowing collection of rewards on any resource from interested parties. Simultaneously, report publishers and voters, stake their owned tokens in betting-pool along with decisions they make. In the end, the majority voters and report publishers, whose reports got approved, receive the reward according to their stakes. Otherwise, their stake becomes as source of reward for others.

7.3 Limitations

Several limitations in research are faced in different parts of this thesis. Firstly, an integrity assurance process is based on the file's hash-code, which identifies the resource itself. There is only one limitation here that turns into uncertainty state, when user should

be automatically prevented from execution of spoiled resource code. In this case, the checksum comparison fails on blockchain-side, and no resource entry is identified, related to this checksum. On the contrary, this can be also considered as an absence of the resource in the blockchain at all. In other words, client's application is unable to distinguish tampered resource from unpublished one, and it needs a manual intervention by user, if uncertainty of this kind occurs.
Moreover, the process of reliability assurance keeps in core the peer-reviewing results as a reliability proof for a resource. The limitation is posed by allowing only specific group of users – certain field experts to evaluate security of resources' codes. Meanwhile, the members of this group are selected externally, since the reputation-driven concepts are not integrated in the current artifact, as for the scope of this thesis.
An exploited evaluation process analyzes the extent to which an artifact's design reaches its identified goals. An assessment of the quality for the proposed functionalities is conducted within this thesis. The limitation is expressed by the lack of time and human resources to check effectiveness of the given solution. It makes impossible to identify a complete incentive mechanism, which is currently naive and not tested in the real world.

7.4 Future work

Throughout this thesis, we identify multiple open issues, requiring a further research. They are mostly relate to research limitations, discussed in previous Section 7.3. Thus, this section is dedicated for the definition of future work.
Initially, some comprehensive approach for resource identification should be attempted. Apart from the resource-hash, which is doubtless proof of resource uniqueness, the hosting server's URI is the identity of a resource owner. However, this does not relate to the common libraries that are frequently used in every WEB-application. A further research is needed for resource identification procedure, in case of the resource is tampered and apparently its hash-code is changed. This can allow to automatically discard dangerous scripts from execution on the client-side.
The research should be conducted to determine a particular structure for community, which is appropriate in the context of involving a competition. For example, the experts of the community, who post their reviews on security matters of the resources, should be selected by voting – by community members, enabling reputation promotions. Besides that, the platform should support the interests of stakeholders, reviewers and basic users in terms of monetization, and also better and secure service provisioning. This can be achieved, by analyzing the platform with real users and elaborating, based on the human's behavior and needs.

Bibliography

[1] Minwatts Marketing Group. *Internet Usage Statistics. The Internet Big Picture*. https://internetworldstats.com/stats.htm. [Accessed: 20-04-2020].

[2] Hussein Alnabulsi, Rafiqul Islam, and Majharul Talukder. "GMSA: Gathering Multiple Signatures Approach to Defend Against Code Injection Attacks". In: *IEEE Access* 6 (Nov. 2018), pp. 77829–77840. DOI: 10.1109/ACCESS.2018.2884201.

[3] Marius Steffens et al. "Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild". In: *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, Feb. 2019. DOI: 10.14722/ndss.2019.23009.

[4] Gurpreet Kaur et al. "Defense Against HTML5 XSS Attack Vectors: A Nested Context-Aware Sanitization Technique". In: *International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, Jan. 2018. ISBN: 978-1-5386-1719-9.

[5] Hui Yuan et al. "Research and Implementation of WEB Application Firewall Based on Feature Matching". In: *MMIA 2019: Application of Intelligent Systems in Multi-modal Information Analytics*. Vol. 929. Mar. 2019, pp. 1223–1231. ISBN: 978-3-030-15740-1.

[6] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. "Semantics-Based Analysis of Content Security Policy Deployment". In: *ACM Transactions on the Web (TWEB)* 12 (June 2018). DOI: 10.1145/3149408.

[7] Stefanos Chaliasos et al. "Mime Artist: Bypassing Whitelisting for the Web with JavaScript Mimicry Attacks". In: *Lecture Notes in Computer Science*. Vol. 11736. Springer International Publishing, Sept. 2019, pp. 565–585. DOI: 10.1007/978-3-030-29962-0_27.

[8] Mario Heiderich, Christopher Späth, and Jörg Schwenk. "DOMPurify: Client-Side Protection Against XSS and Markup Injection". In: *ESORICS 2017. Lecture Notes in Computer Science*. Vol. 10493. Aug. 2017, pp. 116–134. DOI: 10.1007/978-3-319-66399-9_7.

[9] Sebastian Lekies et al. “Code-reuse attacks for the web: breaking cross-site scripting mitigations via script gadgets”. In: *ACM SIGSAC Conference on Computer and Communications Security*. Oct. 2017, pp. 1709–1723. DOI: 10.1145/3133956.3134091.

[10] *OWASP Top 10 - 2017*. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf. [Accessed: 12-10-2019]. Open Web Application Security Project (OWASP).

[11] Clemens H. Cap and Benjamin Leiding. “Ensuring Resource Trust and Integrity in Web Browsers Using Blockchain Technology”. In: *Lecture Notes in Business Information Processing*. Vol. 316. Springer International Publishing, June 2018, pp. 115–125. DOI: 10.1007/978-3-319-92898-2_9.

[12] Stuart Haber and W.Scott Stornetta. “How to time-stamp a digital document”. In: *Journal of Cryptology* 3.2 (Jan. 1991). DOI: 10.1007/bf00196791.

[13] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[14] Giang-Truong Nguyen and Kyungbaek Kim. “A Survey about Consensus Algorithms Used in Blockchain”. In: *JIPS* 14.1 (Jan. 2018), pp. 101–128. DOI: 10.3745/JIPS.01.0024.

[15] Peter C. Rigby, Daniel M. German, and Margaret-Anne Storey. “Open Source Software Peer Review Practices: A Case Study of the Apache Server”. In: *Proceedings of the 30th International Conference on Software Engineering*. ICSE ’08. Leipzig, Germany: Association for Computing Machinery, 2008, pp. 541–550. DOI: 10.1145/1368088.1368162.

[16] Michael E. Fagan. “Design and Code Inspections to Reduce Errors in Program Development”. In: *IBM Systems Journal* 38 (1999), pp. 258–287. DOI: 10.1147/sj.382.0258.

[17] Peter C. Rigby and Christian Bird. “Convergent Contemporary Software Peer Review Practices”. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2013. Saint Petersburg, Russia: Association for Computing Machinery, 2013, pp. 202–212. DOI: 10.1145/2491411.2491444.

[18] Esther Daniel, S. Durga, and S. Seetha. “Panoramic View of Cloud Storage Security Attacks: an Insight and Security Approaches”. In: *Third International Conference on Computing Methodologies and Communication (ICCMC 2019)*. IEEE, Mar. 2019. DOI: 10.1109/ICCMC.2019.8819801.

[19] Sarmah Upasana, Bhattacharyya D.K, and Kalita J.K. “A Survey of Detection Methods for XSS Attacks”. In: *Journal of Network and Computer Applications* (June 2018). DOI: 10.1016/j.jnca.2018.06.004.

[20] Brian Chess and Gary Mcgraw. “Static analysis for security”. In: *IEEE Security & Privacy* 2 (Dec. 2004), pp. 76–79. DOI: 10.1109/MSP.2004.111.

[21] Daniel Bates, Adam Barth, and Collin Jackson. “Regular expressions considered harmful in client-side XSS filters”. In: *Proceedings of the 19th International Conference on World Wide Web, WWW 2010*. Apr. 2010, pp. 91–100. DOI: 10.1145/1772690.1772701.

[22] David Ross. “IE8 Security Part IV: The XSS Filter”. [Accessed: 28-10-2019]. July 2008. URL: https://blogs.msdn.microsoft.com/ie/2008/07/02/ie8-security-part-iv-the-xss-filter.

[23] Shashank Gupta and Brij Bhooshan Gupta. “XSS-immune: a Google chrome extension-based XSS defensive framework for contemporary platforms of web applications”. In: *Security and Communication Networks* 9 (17 Nov. 2016), pp. 3966–3986. DOI: 10.1002/sec.1579.

[24] Kanpata Sudhakara Rao et al. “Two for the price of one: A combined browser defense against XSS and clickjacking”. In: *International Conference on Computing, Networking and Communications (ICNC)*. Vol. 1. IEEE, Feb. 2016, pp. 1–6. DOI: 10.1109/ICCNC.2016.7440629.

[25] Jinkun Pan and Xiaoguang Mao. “DomXssMicro: A Micro Benchmark for Evaluating DOM-Based Cross-Site Scripting Detection”. In: *IEEE Trustcom/BigDataSE/ISPA*. IEEE, Aug. 2016, pp. 208–215. DOI: 10.1109/trustcom.2016.0065.

[26] Jinkun Pan and Xiaoguang Mao. “Detecting DOM-Sourced Cross-Site Scripting in Browser Extensions”. In: *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Sept. 2017, pp. 24–34. DOI: 10.1109/ICSME.2017.11.

[27] Mahmoud Mohammadi, Bill Chu, and Heather Richter Lipford. “Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing”. In: *IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, July 2017, pp. 364–373. DOI: 10.1109/QRS.2017.46.

[28] Shashank Gupta and Brij Bhooshan Gupta. “XSS-secure as a service for the platforms of online social network-based multimedia web applications in cloud”. In: *Multimedia Tools and Applications* 77 (Feb. 2018), pp. 4829–4861. DOI: 10.1007/s11042-016-3735-1.

[29] Wafa Ben Jaballah and Nizar Kheir. “A Grey-Box Approach for Detecting Malicious User Interactions in Web Applications”. In: *MIST ’16 Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*. Oct. 2016, pp. 1–12. DOI: 10.1145/2995959.2995966.

[30] Srinivas Nidhra. “Black Box and White Box Testing Techniques - A Literature Review”. In: *International Journal of Embedded Systems and Applications* 2.2 (June 2012), pp. 29–50. DOI: 10.5121/ijesa.2012.2204.

[31] Swaswati Goswami et al. “An Unsupervised Method for Detection of XSS Attack”. In: *International Journal of Network Security* 19 (Sept. 2017), pp. 761–775. DOI: 10.6633/IJNS.201709.19(5).14.

[32] Kiri Wagstaff et al. “Constrained K-means Clustering with Background Knowledge”. In: Jan. 2001, pp. 577–584.

[33] Mohammed Babiker, Enis Karaarslan, and Yasar Hoscan. “Web application attack detection and forensics: A survey”. In: *6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, May 2018. DOI: 10.1109/ISDFS.2018.8355378.

[34] Victor Clincy and Hossain Shahriar. “Web Application Firewall: Network Security Models and Configuration”. In: *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, June 2018. DOI: 10.1109/COMPSAC.2018.00144.

[35] Divya Rajput and Ankit Thakkar. “A Survey on Different Network Intrusion Detection Systems and CounterMeasure”. In: *Emerging Research in Computing, Information, Communication and Applications*. Springer Singapore, Sept. 2019, pp. 497–506. DOI: 10.1007/978-981-13-6001-5_41.

[36] Ali Moradi Vartouni, Mohammad Teshnehlab, and Saeed Sedighian Kashi. “Leveraging deep neural networks for anomaly-based web application firewall”. In: *IET Information Security* 13.4 (July 2019), pp. 352–361. DOI: 10.1049/iet-ifs.2018.5404.

[37] Wen-Chao Jia, Rong-Gui Hu, and Fan Shi. “Feature Design and Selection Based on Web Application-Oriented Active Threat Awareness Model”. In: *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*. IEEE, July 2016. DOI: 10.1109/imccc.2016.64.

[38] Daniel Shugrue. “Fighting application threats with cloud-based WAFs”. In: *Network Security* 2017.6 (June 2017), pp. 5–8. DOI: 10.1016/s1353-4858(17)30059-4.

[39] Dennis Appelt, Annibale Panichella, and Lionel Briand. “Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks”. In: *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Oct. 2017. DOI: 10.1109/issre.2017.28.

[40] Alex Mathews. “What can machine learning do for information security?” In: *Network Security* 2019.4 (Apr. 2019), pp. 15–17. DOI: 10.1016/s1353-4858(19)30050-9.

[41] Zhihong Tian et al. “A Distributed Deep Learning System for Web Attack Detection on Edge Devices”. In: *IEEE Transactions on Industrial Informatics* (2019), pp. 1–1. DOI: 10.1109/tii.2019.2938778.

[42] Amor Lazzez and Thabet Slimani. “Forensics Investigation of Web Application Security Attacks”. In: *International Journal of Computer Network and Information Security* 7.3 (Feb. 2015), pp. 10–17. DOI: 10.5815/ijcnis.2015.03.02.

[43] Tobias Lauinger et al. “Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web”. In: *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, Nov. 2017. DOI: 10.14722/ndss.2017.23414.

[44] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. “Content Security Problems?” In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16*. ACM Press, Oct. 2016. DOI: 10.1145/2976749.2978338.

[45] Lukas Weichselbaum et al. “CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16*. ACM Press, Oct. 2016, pp. 1376–1387. DOI: 10.1145/2976749.2978363.

[46] Ben Stock et al. “Precise client-side protection against DOM-based cross-site scripting”. In: *SEC’14 Proceedings of the 23rd USENIX conference on Security Symposium*. ACM Press, Aug. 2014, pp. 655–670. ISBN: 978-1-931971-15-7.

[47] James Walker. *XSS protection disappears from Microsoft Edge*. https://portswigger.net/daily-swig/xss-protection-disappears-from-microsoft-edge. [Accessed: 2019-11-28]. July 2018.

[48] Shashank Gupta, Brij Bhooshan Gupta, and Pooja Chaudhary. “Hunting for DOM-Based XSS vulnerabilities in mobile cloud-based online social network”. In: *Future Generation Computer Systems* 79 (Feb. 2018), pp. 319–336. DOI: 10.1016/j.future.2017.05.038.

[49] Benjamin Livshits. *Dynamic Taint Tracking in Managed Runtimes*. Tech. rep. MSR-TR-2012-114. Nov. 2012. URL: https://www.microsoft.com/research/publication/dynamic-taint-tracking-in-managed-runtimes.

[50] Alan R. Hevner et al. "Design Science in Information Systems Research". In: *MIS Q.* 28.1 (Mar. 2004), pp. 75–105. ISSN: 0276-7783. URL: http://dl.acm.org/citation.cfm?id=2017212.2017217.

[51] Herbert Alexander Simon. *The Sciences of the Artificial*. 3rd ed. Cambridge: MIT Press, 1996.

[52] Mark S. Silver, M. Lynne Markus, and Cynthia Mathis Beath. "The Information Technology Interaction Model: A Foundation for the MBA Core Course". In: *MIS Quarterly* 19 (1995), pp. 361–390.

[53] John Venable, Jan Pries-Heje, and Richard Baskerville. "FEDS: a Framework for Evaluation in Design Science Research". In: *European Journal of Information Systems* 25.1 (Jan. 2016), pp. 77–89. DOI: 10.1057/ejis.2014.36.

[54] Lars Mathiassen et al. *Object-Oriented Analysis and Design*. Dansk. Marko, 2000.

[55] Martin Johns and Stephan Pfistner. *End-to-end taint tracking for detection and mitigation of injection vulnerabilities in web applications*. Patent. No.: US010129285B2. SAP SE, Nov. 2018.

[56] Joaquin Garcia-Alfaro and Guillermo Navarro-Arribas. *A Survey on Cross-Site Scripting Attacks*. May 2009. URL: https://arxiv.org/abs/0905.4850.

[57] Akhil Nair et al. "Prevention of Cross Site Scripting (XSS) and securing web application atclient side". In: *International Journal Of Emerging Technology and Computer Science* 3 (Apr. 2018), pp. 83–86. ISSN: 2455-9954.

[58] Narendran Calluru Rajasekar and Chris O. Imafidon. "Exploitation of Vulnerabilities in Cloud-Storage". In: *GSTF International Journal on Computing* 1 (Feb. 2011).

[59] *How Hackers Hijacked a Bank's Entire Online Operation*. https://www.wired.com/2017/04/hackers-hijacked-banks-entire-online-operation/. [Accessed: 16-10-2019]. Apr. 2017.

[60] Daniel Davis Wood. "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER". In: 2014.

[61] Serguei Popov. "A Probabilistic Analysis of the Nxt Forging Algorithm". In: *Ledger* 1 (Dec. 2016), pp. 69–83. DOI: 10.5195/ledger.2016.46.

[62] Lin Chen et al. “On Security Analysis of Proof-of-Elapsed-Time (PoET)”. In: *Lecture Notes in Computer Science*. Springer International Publishing, Oct. 2017, pp. 282–297. DOI: 10.1007/978-3-319-69084-1_19.

[63] Mitar Milutinovic et al. “Proof of Luck: An Efficient Blockchain Consensus Protocol”. In: *Proceedings of the 1st Workshop on System Software for Trusted Execution*. SysTEX ’16. Trento, Italy: Association for Computing Machinery, 2016. DOI: 10.1145/3007788.3007790.

[64] Sunoo Park et al. *SpaceMint: A Cryptocurrency Based on Proofs of Space*. Cryptology ePrint Archive, Report 2015/528. https://eprint.iacr.org/2015/528. 2015.

[65] *Bitcoin Stack Exchange*. [Accessed: 14-01-2020]. 2017. URL: https://bitcoin.stackexchange.com/questions/12427/can-someone-explain-how-the-bitcoin-blockchain-works.

[66] Mario Biagioli. “From Book Censorship to Academic Peer Review”. In: *Emergences: Journal for the Study of Media & Composite Cultures* 12.1 (2002), pp. 11–45. DOI: 10.1080/1045722022000003435.

[67] Herbert W. Marsh and Samuel Ball. “The Peer Review Process Used to Evaluate Manuscripts Submitted to Academic Journals”. In: *The Journal of Experimental Education* 57.2 (1989), pp. 151–169. DOI: 10.1080/00220973.1989.10806503.

[68] Larry Conklin et al. *OWASP Code Review Project*. Tech. rep. 2017. URL: https://www.owasp.org/images/5/53/OWASP_Code_Review_Guide_v2.pdf.

[69] John D. Blischak, Emily R. Davenport, and Greg Wilson. “A Quick Introduction to Version Control with Git and GitHub”. In: *PLOS Computational Biology* 12.1 (Jan. 2016). Ed. by Francis Ouellette, e1004668. DOI: 10.1371/journal.pcbi.1004668.

[70] *Steem/Bluepaper*. Tech. rep. 2017. URL: https://steem.com/wp-content/uploads/2018/10/steem-bluepaper-1.pdf.

[71] Usman Chohan. “The Concept and Criticisms of Steemit”. In: *SSRN Electronic Journal* (2018). DOI: 10.2139/ssrn.3129410.

[72] Craig Calcaterra and Wulf A. Kaal. “Semadaas Proof of Stake Protocol”. In: *SSRN Electronic Journal* (2018). DOI: 10.2139/ssrn.3125827.

[73] Craig Calcaterra, Wulf A. Kaal, and Vlad Andrei. *Semada Technical Whitepaper*. Tech. rep. Nov. 2017. URL: https://docs.google.com/document/d/1rMpcaO5rlXw5RxUCDy_e_his6DSdrDfUS9qwcgWHAAw/edit?usp=sharing.

[74] Mubashar Iqbal and Raimundas Matulevičius. "Comparison of Blockchain-Based Solutions to Mitigate Data Tampering Security Risk". In: *Business Process Management: Blockchain and Central and Eastern Europe Forum*. Ed. by Claudio Di Ciccio et al. Cham: Springer International Publishing, Aug. 2019, pp. 13–28. DOI: 10.1007/978-3-030-30429-4_2.

[75] Greg Lee et al. *EOS.IO Technical White Paper*. Tech. rep. block.one, Mar. 2018. URL: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md.

[76] Kaidong Wu. *An Empirical Study of Blockchain-based Decentralized Applications*. 2019. arXiv: 1902.04969 [cs.DC].

[77] Thanh Chung Dao, Binh Minh Nguyen, and Ba Lam Do. "Challenges and Strategies for Developing Decentralized Applications Based on Blockchain Technology". In: *Advanced Information Networking and Applications*. Springer International Publishing, Mar. 2019, pp. 952–962. DOI: 10.1007/978-3-030-15032-7_80.

[78] Jannis Angelis and Elias Ribeiro da Silva. "Blockchain adoption: A value driver perspective". In: *Business Horizons* 62.3 (May 2019), pp. 307–314. DOI: 10.1016/j.bushor.2018.12.001.

[79] Amritraj Singh et al. "Sidechain technologies in blockchain networks: An examination and state-of-the-art review". In: *Journal of Network and Computer Applications* 149 (Jan. 2020), p. 102471. DOI: 10.1016/j.jnca.2019.102471.

[80] *Why Server Response Time Is Important*. https://hrank.com/why-server-response-time-is-important. [Accessed: 27-02-2020].

[81] *EOSIO.Contracts / Key Concepts / RAM as resource*. https://developers.eos.io/manuals/eosio.contracts/latest/key-concepts/ram. [Accessed: 28-04-2020].

[82] QuillHash Team. *EOS Smart Contract Development: Understanding fundamental concepts for writing dApps on EOS*. https://medium.com/p/9d8e1a263724. [Accessed: 13-02-2020]. Feb. 2019.

[83] Fan Yang et al. "Delegated Proof of Stake With Downgrade: A Secure and Efficient Blockchain Consensus Algorithm With Downgrade Mechanism". In: *IEEE Access* 7 (2019), pp. 118541–118555. DOI: 10.1109/access.2019.2935149.

[84] *EOS.IO Transactions Protocol*. https://developers.eos.io/welcome/latest/protocol/transactions_protocol. [Accessed: 14-02-2020].

[85] *Network Peer Protocol*. https://developers.eos.io/welcome/latest/protocol/network_peer_protocol. [Accessed: 16-02-2020].

[86] *EOS.IO Protocol: Accounts and Permissions*. https://developers.eos.io/welcome/latest/protocol/accounts_and_permissions. [Accessed: 14-02-2020].

[87] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: *CoRR* abs/1407.3561 (2014). arXiv: 1407.3561. URL: http://arxiv.org/abs/1407.3561.

[88] *Static Code Analysis*. https://owasp.org/www-community/controls/Static_Code_Analysis. [Accessed: 18-03-2020].

[89] *Review of Ruby Static Analysis Tools*. https://blog.codacy.com/review-of-ruby-static-analysis-tools. [Accessed: 18-03-2020]. Dec. 2015.

[90] *How the EOS Referendum System works*. https://link.medium.com/1UpEBGovu5. [Accessed: 07-04-2020]. EOS Nation, Jan. 2019.

[91] *eosio.forum Part 1: EOSIO Referendum*. https://link.medium.com/xsbtx6Rvu5. [Accessed: 07-04-2020]. Obsidian Labs, Oct. 2019.

[92] *EOS.IO contracts: eosio.token*. https://developers.eos.io/manuals/eosio.contracts/latest/action-reference/eosio.token/index. [Accessed: 11-04-2020]. Block.One.

[93] *Smart Contract Development: Deploy Issue And Transfer Tokens*. https://developers.eos.io/welcome/latest/getting-started/smart-contract-development/deploy-issue-and-transfer-tokens. [Accessed: 11-04-2020]. Block.One.

[94] *Steem. An incentivized, blockchain-based, public content platform*. Tech. rep. Aug. 2017. URL: https://steem.com/SteemWhitePaper.pdf.

[95] *ISO Standards*. https://www.iso.org/standards.html. [Accessed: 25-04-2020]. ISO.

[96] Indira Padayachee et al. "ISO 9126 external systems quality characteristics, sub-characteristics and domain specific criteria for evaluating e-Learning systems". In: (Jan. 2010).

[97] Antonia Stefani and Michalis Xenos. “E-commerce system quality assessment using a model based on ISO 9126 and Belief Networks”. In: *Software Quality Journal* 16.1 (Oct. 2007), pp. 107–129. DOI: 10.1007/s11219-007-9032-5.

[98] Leanard Buenaflor. *ISO 9126 Software Quality Characteristics*. https://link.medium.com/cVuvSQQ7Z5. [Accessed: 26-04-2020]. Sept. 2017.

[99] Elad Elrom. “EOS.IO Wallets and Smart Contracts”. In: *The Blockchain Developer: A Practical Guide for Designing, Implementing, Publishing, Testing, and Securing Distributed Blockchain-based Projects*. Berkeley, CA: Apress, 2019, pp. 213–256. ISBN: 978-1-4842-4847-8. DOI: 10.1007/978-1-4842-4847-8_6.

[100] Yuheng Huang et al. *Characterizing EOSIO Blockchain*. 2020. arXiv: 2002.05369 [cs.CR]. URL: https://arxiv.org/pdf/2002.05369.pdf.

[101] Vlado Tesanovic. *Multi threading and multiple process in Node.js*. https://itnext.io/multi-threading-and-multi-process-in-node-js-ffa5bb5cde98. [Accessed: 28-04-2020]. Apr. 2018.

[102] Daniel Larimer. *EOSIO Dawn 3.0 Now Available*. https://link.medium.com/gxz9fwJL35. [Accessed: 28-04-2020]. Apr. 2018.

[103] *EOS.IO Core Concepts*. https://developers.eos.io/welcome/v2.0/overview/core_concepts. [Accessed: 28-04-2020].