An Approach for Trustworthiness Benchmarking using Software Metrics

Nádia Medeiros*, Naghmeh Ivaki*, Pedro Costa*†, Marco Vieira*
* CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
† ISCAC, Polytechnic Institute of Coimbra, Portugal
nadiam@dei.uc.pt, naghmeh@dei.uc.pt, pncosta@dei.uc.pt, mvieira@dei.uc.pt

***Abstract*—Trustworthiness is a paramount concern for users and customers in the selection of a software solution, specially in the context of complex and dynamic environments, such as Cloud and IoT. However, assessing and benchmarking trustworthiness (worthiness of software for being trusted) is a challenging task, mainly due to the variety of application scenarios (e.g., business-critical, safety-critical), the large number of determinative quality attributes (e.g., security, performance), and last, but foremost, due to the subjective notion of trust and trustworthiness. In this paper, we present trustworthiness as a measurable notion in relative terms based on security attributes and propose an approach for the assessment and benchmarking of software. The main goal is to build a trustworthiness assessment model based on software metrics (e.g., *Cyclomatic Complexity*, *CountLine*, *CBO*) that can be used as indicators of software security. To demonstrate the proposed approach, we assessed and ranked several files and functions of the Mozilla Firefox project based on their trustworthiness score and conducted a survey among several software security experts in order to validate the obtained rank. Results show that our approach is able to provide a sound ranking of the benchmarked software.**

***Index Terms*—Trustworthiness Benchmarking, Security Vulnerabilities, Software Metrics**

I. Introduction

Software plays an important role in an increasing number of complex and critical systems used in businesses, industries, government, and people's daily life (ranging from e-health applications to embedded control systems used in aircrafts or nuclear power systems). As a consequence, software failures (either caused by internal bugs or by successful security attacks) can cause more damages than ever before, with serious effect in the disclosure of private data, safety of the users, and reputation. Building trustworthy software is thus critical for the success of organizations.

Although software trustworthiness has been an important concern for developers, researchers and enterprises, assessing trustworthiness is still a nontrivial task due to several factors, such as the diversity of software systems (e.g., safety-critical or business-critical) and the large scale and high complexity of today's systems. Moreover, the notion of trust and trustworthiness is subject to individual preference and interpretation, as, depending on the context (i.e., application scenario of the software), different quality attributes (e.g., security or performance) may be involved in the assessment of the trustworthiness level of a system [1].

Although we can find several attempts in the literature to address this issue [2], [3], [4], defining a mechanism or tool that accurately measures the trustworthiness level of a system is still very challenging. For this reason, in complex environments like cloud-based systems, software service providers usually try to include trust related requirements as Service-Level Agreements (SLAs) to boost the trust of consumers, but such agreements are not based on quantitative and verifiable evidence of trustworthiness.

In this work, we aim to move from a subjective to an objective trustworthiness notion by building a model based on several software attributes and their relative importance for determining how trustworthy a piece of software is. The proposed model is based on the architectural quality attributes of software, represented by concrete software metrics of different types, including complexity (e.g., SumCyclomatic), volume (e.g., CountLineCodeDecl), coupling (e.g., CBO), and cohesion metrics (e.g., LCOM). **The main objective of this work is focused on showing the possibility of using software metrics to benchmark the trustworthiness of software systems.**

Several studies have shown that there is some correlation between the software architecture, represented as software metrics, and the existence of vulnerabilities [5], [6], [7]. In a previous study [8], we showed that it is possible to use a group of software metrics for building a classifier to distinguish vulnerable and non-vulnerable units of code (files or functions) with a high level of accuracy. In this work, we aim at building the trustworthiness benchmarking model using this group of software metrics. With this model, a trustworthiness score is assigned to each unit of code, calculated using the normalized value of relevant metrics and their impact on the precision of the classifier, which is measured by three different *variable importance measures*, namely Mean Decrease Accuracy (MDA), Mean Decrease Gini (MDG), and the joint measure MDAMDG (MDA + MDG) [9]. This relative trustworthiness score can then be used to compare and rank software elements (files or functions).

Considering that trustworthiness is an integrating concept composed by several but complementary attributes (e.g., performance, availability, security), this proposal is focused only on security aspects (also to make the problem dealable). However, the proposed approach is applicable, with minimum changes, to any other trustworthiness attribute. It is also worth noting that security is nowadays a fundamental attribute of trustworthiness in the majority of critical software systems and applications [1].

In order to demonstrate and validate the proposed approach, we selected several files and functions of the Mozilla Firefox project and ranked them based on their trustworthiness level calculated using the proposed trustworthiness model. We then took an empirical approach by conducting a questionnaire survey among several software security experts in order to conduct a paired comparison analysis of the selected files and functions. We used the Row Geometric Mean Method (RGMM) [10] and the Aggregation of Individual Judgments (AIJ) [11] techniques to calculate the individual and aggregated ranking of the files and functions based on the pairwise comparisons given by the experts. Finally, we performed a detailed comparison between the results obtained using the proposed trustworthiness assessment model and the results obtained from the judgments of the experts. The results show an agreement between the rank obtained by the trustworthiness model-based approach and the individual and aggregated ranks of the experts.

The rest of this paper is organized as follows. Section II presents concepts regarding security and trustworthiness as well as several important related studies. Section III presents an high-level view of the trustworthiness benchmarking approach proposed. Section IV describes an instantiation of the approach, and experimental results are presented in Section V. An expert-based ranking used for validation, threats to validity and highlights on the generalization of the approach are descibed in Section VI. Section VII concludes the paper and puts forward ideas for future work.

II. Related Work

Trust and trustworthiness have been broadly studied in many different areas [12], [13]. In computer science, these terms are widely used in areas like semantic web, game theory and agent systems [14]. In software engineering, software trustworthiness assurance or improvement [15] and trustworthiness measurement [16] are extensively investigated topics. Trust has also been investigated for service selection and rating in web service systems [17], [18]. With the emergence of cloud computing and cloud related technologies (e.g., fog computing and IoT), special attention and consideration has been given to trust and trustworthiness by many researchers, enterprises, and cloud providers [19], [20], [21].

Trustworthiness of software is a subjective notion and, depending on the context, different attributes (e.g., security and performance) might be determinative [1]. Sometimes, it may simply refer to one single attribute like security, or it may refer to a combination of several attributes like reliability, correctness, performance, compliance, availability, interoperability, and security [22], [1], particularly when it comes to dynamic and complex environments like cloud computing [23], [24]. Moreover, the relevant trustworthiness attributes may dynamically evolve or change during the software life cycle. A structured study on trustworthiness attributes is presented in [1], [4], and [25].

We can find several research studies in the literature that presented theoretical and practical development practices and solutions for the trustworthiness of software [26], [2], [3]. An exhaustive overview of these development methodologies can be found in [27]. In some works, structured assurance cases are applied to assure a set of trustworthy related software properties [28].

Despite the huge advances in software development processes, techniques and tools, and in spite of the existence of standards for building high quality software (e.g., ISO/IEC 27000 [29]) and evaluating software quality (e.g., ISO/IEC 9126 [30] and ISO 15408 [31]), it is still very hard, if not impossible, to develop software free of bugs and vulnerabilities [32]. The existence of systems with software defects or bugs that escaped the testing phases of the software development process, emphasizes the importance of software trustworthiness assessment to help improving the quality of software, and to support informed decision making.

Since software trustworthiness is usually characterized by several quality attributes such as reliability, availability, and security, software trustworthiness assessment has usually been identified as a multiple attribute decision analysis (MADA) problem [4], [24]. The main challenge in MADA is associated with assigning proper weights to different attributes [4]. Different models, methods, and frameworks are proposed to deal with software trustworthiness as a MADA problem. Representative examples of these models and methods can be found in [33], [34], [4].

A different approach is used in [16], in which a trustworthiness measurement model based on source code analysis is proposed. The model is built based on some untrusted evidences collected from the source code, such as improper break or data overflow, and their impact. Our approach for assessing the trustworthiness of software is also based on the analysis of software source code. We focus on security and use software metrics, which we previously showed that can be used as indicators of software security [8], to evaluate the trustworthiness level of software.

Compared to previous works, our approach is based on evidences that are proven to have correlation with security of software as a fundamental attribute of trustworthiness. The trustworthiness score calculated directly indicates the units of code (files or functions) that are the origin of untrustworthiness in the source code, thus helping to improve the quality of software during the development phase.

III. Overall Trustworthiness Benchmarking Approach

Our proposal for software trustworthiness benchmarking focuses on security and uses a set of software metrics (e.g., *CountPath*, *SumMaxNesting*) as indicators of security vulnerabilities. These software metrics are identified using a heuristic search algorithm from our previous work [8], as the best subset of metrics for building a classifier model allowing to distinguish vulnerable pieces of code from non-vulnerable ones. Figure 1 depicts the main components of the proposed approach, which results in the calculation of a trustworthiness score that allows comparing the trustworthiness of different software elements (e.g., functions, files).

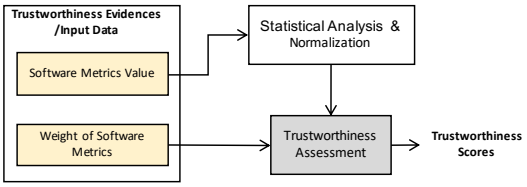

Fig. 1: Assessment Approach (dataset from [7]).

As shown in Figure 1, the trustworthiness score, used as the sole benchmarking criterion in our approach, is calculated based on the **Software Metrics Value** and their relative importance, as defined by the **Weight of Software Metrics**. Actually, these data represent the **Trustworthiness Evidences** that are the main input for the benchmark. Since the software metrics are measured using different scales, there is the need to normalize them to a common scale. To this end, a statistical analysis needs to be performed on the input data to better understand the distribution of the values of each software metric.

The next step is to identify the relative importance (or weight) of the software metrics. To do so, we need to study how determinative each metric is in the classification of vulnerable and non-vulnerable units of code.

To obtain the trustworthiness score of each unit of code (file or function) under evaluation, we calculate the sum of the product between the normalized value of each software metric (M) and its associated weight (W), as shown in Equation 1. The index n represents the total number of software metrics used. Note that, this score is a **relative** measure of trustworthiness that should only be used for comparison purposes and not as an absolute measure of security or trustworthiness. Given this score, the codes units can be simply ranked.

$$Trustworthiness\ Score = \sum_{i=1}^{n} M_i W_i \qquad (1)$$

The proposed approach is generic and easily applicable to other trustworthiness attributes. In fact, we can replace security attributes with any other trustworthiness attributes (e.g., performance) or add other attributes to the model. For this, we just need to identify the criteria for measuring the corresponding trustworthiness attributes (e.g., throughput, response time) and their relative importance.

IV. Instantiation of the Trustworthiness Benchmarking Approach

In our previous work [8], we performed an exploratory and empirical analysis on software metrics in different architectural levels of five software systems (Mozilla Firefox (mozilla.org), Apache httpd (httpd.apache.org), Linux Kernel (kernel.org), Xen Hypervisor (xen.org), and glibc (gnu.org/software/libc)) to study whether software metrics are discriminative enough for classifying vulnerable code and showed that it is possible to use a group of metrics to distinguish vulnerable and non-vulnerable units of code. We used a heuristic search technique at function and file levels to select the best subset of metrics for building such classifier.

In the previous section, we proposed an high-level approach to build a trustworthiness assessment model, using a set of software metrics, that is able to show how trustworthy the units of code are by assigning a trustworthiness score to each unit (file or function). Figure 2 depicts the process followed to instantiate this approach considering the metrics adopted from [8]. The output is a trustworthiness score that allows comparing the trustworthiness of different software units.

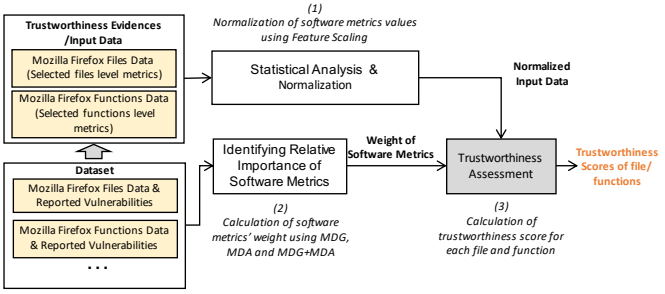

Fig. 2: Instantiation Process of the Benchmarking Approach.

A. Dataset and Trustworthiness Evidences

We instantiated the proposed approach based on a dataset containing detailed information about the files and functions that compose five software projects, including Mozilla Firefox. The **dataset** includes the list of file and function level software metrics (56 file-level metrics and 28 function-level metrics) extracted by Alves et al. [7], using Understand [35], and also detailed information about the known vulnerabilities.

In this paper, we work on top of the Mozilla Firefox data, since it is a quite large and well-known software project, with a large number of known vulnerabilities, a fundamental aspect to achieve more accurate results. Table I presents some information about the project. It is important to emphasize that the proposed approach is generic and applicable to any other software projects (datasets), as far as the source code is available for extracting the files and functions data.

We extracted the Mozilla Firefox's files and functions data for the selected metrics (21 file-level metrics and 15 function-level metrics, as shown in tables II and III). These input data represent the trustworthiness evidences in our benchmarking process.

B. Statistical Analysis and Normalization

To normalize the value of different software metrics to a common scale, we first performed a statistical analysis on the input data to better understand the distribution of the values of each software metric. Minimum, maximum, mean and quartiles values were calculated for each metric. Using

TABLE I: Summary of the Mozilla Firefox Project.

	total	vulnerable	%vuln	soft. metrics
#Files	28927	830	2,87%	21
#Functions	614422	2107	0,34%	15

this simple analysis, we observed that there are cases with very large values that affect the normalization process (e.g., for the *FanIn* software metric, the maximum value existing in the dataset is 129882 while the mean value is 108). To eliminate this effect, we identified the outliers using a statistical method based on the Interquartile Range (IQR) [36]. The IQR is the range between the first and the third quartiles ($IQ_3 - Q_1$). Using this method, any data value falling outside of the acceptable range, between the lower fence and upper fence, is considered to be an outlier:

$$\begin{gathered} AcceptableRange = [LowerFence, UpperFence] \\ UpperFence(UF) = Q_3 + 1.5 * IQR \\ LowerFence(LF) = Max(Q_1 - 1.5 * IQR, 0) \\ IQR = Q3 - Q1 \end{gathered} \tag{2}$$

We then used Feature Scaling, which is a popular method to normalize the values of independent variables and bring them into a common range (between 0 and 1 in our work). As we are interested in characterizing trustworthiness, a higher score (1 in our work) should represent a more trustworthy code unit, thus we need to verify whether each individual software metric has a direct or inverse relationship with the level of trustworthiness. To this end, we resorted to our previous work [8] and checked the *partial dependency* between each selected metric and the existence of vulnerabilities in the random forest classifier built. In most cases, there is an obvious direct relationship, which means an inverse relationship with trustworthiness: the greater the value of the metric, the more likely to have a vulnerability, thus less trustworthy. In a few cases, we have not seen an obvious direct or inverse relationship. This may seem a bit odd since these metrics are selected by the classifier as predictive metrics, but it happens because partial dependency of each metric is calculated individually and, as explained in [8], a software metric that is irrelevant when considered individually might be highly significant when combined with other metrics. However, since we are going to normalize the values of the software metrics individually, we ended up concluding that the selected software metrics, have either an inverse relationship or an indifferent relationship.

Based on this relationship and given the above Acceptable Range, the value *X* for each software metric is scaled into the range [0,1] by using Equation 3.

$$X' = \begin{cases} 1 - \dfrac{X - LF}{UF - LF} & : X \text{ is in acceptable range} \\ 0 & : X \text{ is an outlier} \end{cases} \tag{3}$$

C. Identifying Relative Importance of Software Metrics

To identify the weight of the selected software metrics for building the trustworthiness benchmarking model, we again resorted to our previous work [8] and used the random forest classifier to calculate two common variable importance measures, namely Mean Decrease Accuracy (MDA) and Mean Decrease Gini (MDG) [9] for each software metric. In addition to these two measures, we use the MDAMDG joint measure, which previous research [37] showed to be a robust measure to identify the relative importance of the features used to build a classifier. These measures are explained in the next paragraphs:

- **Mean Decrease Accuracy (MDA)** - shows how much the classifier's accuracy decreases by dropping a software metric from the list of features. MDA is normalized by the number of trees in the random forest. The greater the accuracy drops, the more significant the software metric.
- **Mean Decrease Gini (MDG)** - is a measure of how each software metric contributes to the homogeneity of nodes in the trees of the resulting random forest. Each time a particular software metric is used to split a node, the Gini coefficient for the child nodes are calculated and compared to that of the original node. Metrics that result in nodes with higher purity have a higher decrease in Gini coefficient.
- **MDAMDG** - is a joint measure whose value is the sum of MDA and MDG. This measure is used to calculate the total *importance score* of each software metric. The greater the total score, the more significant the software metric is.

Given the above explanation, three importance scores (i.e., MDA, MDG, and MDAMDG) are assigned to each software metric, resulting in three different weights. Each weight is calculated by normalizing the value of the corresponding importance score to a [0-1] range by considering the number of software metrics, so that, for each score, the sum of the weights of all software metrics is 1 (see tables II and III, columns MDG_W, MDA_W, and MDAMDG_W).

D. Trustworthiness Assessment

To obtain the trustworthiness score of each unit of code (file or function) under evaluation, we calculate the sum of the product between the normalized value of each software metric (*M*) and its associated weight (*W*), as shown previously in Equation 1. The index *n* ranges from 1 to 21 in the case of files and from 1 to 15 in the case of functions, representing the total number of software metrics used for each type of unit of code.

Since we have previously calculated three different weights for each software metric (MDA, MDG, and MADMDG), we will have three different trustworthiness scores for each file/function. These scores can finally be used to compare/rank a set files/functions under evaluation.

V. Benchmarking Campaign

To experimentally demonstrate our benchmark we selected several files and functions from the source code of the software project under study (i.e., Mozila Firefox) and ranked them using the trustworthiness scores obtained. The process followed is depicted in Figure 3 and explained in the following paragraphs.

A. Files and Functions Qualification and Selection

To select a meaningful list of files and functions from the Mozilla Firefox's **source code** for the analysis, we divided

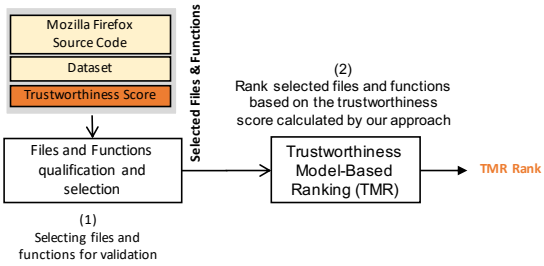

Fig. 3: Benchmarking Process

the **dataset**'s files/functions in two groups: with and without known vulnerabilities. This is done first because the files and functions with vulnerabilities do not qualify for trustworthiness evaluation (they cannot be trusted). Second, we want to eliminate any possible effect of the existence of known vulnerabilities (in files and functions) on the judgments of the experts, which will be used later to validate the output of our approach. However, for sake of completeness, we repeat the whole process, including experiments and analyses, with this group of files/functions (with vulnerabilities) to validate the proposed approach for this specific case (i.e., its usefulness to compare code with known vulnerabilities).

We selected 5 files and 5 functions from each group/category. We choose only 5 to make the validation process feasible, since the experts should compare each pair of files/function separately (i.e., a total of 90 comparisons are needed for only 5 files and 5 functions). The procedure was as follows:

- **First selection using the value of individual software metrics** - To guarantee the diversity of our selection, we first sorted the files/functions according to the individual software metrics. For each metric, we then selected the files/functions having the minimum, lower fence, mean, upper fence and maximum values, resulting in the selection of 105 files (5 files selected for each of the 21 file-level metrics) and 75 functions (5 functions selected for each of 15 file-level metrics). A random choice of one was made in the cases where several files/functions have the same value.
- **Cleaning** - In this step, we first removed duplicated files/-functions and then removed the files/functions in which all software metrics have a normalized value equal to 0, as in this case we cannot calculate a trustworthiness scores.
- **Final selection of files and functions** - Finally, we sorted the remaining 71 files and 54 functions according to the **trustworthiness score** calculated using the MDAMDG weight. Then we selected 5 non-vulnerable files ($File_A$, $File_B$, $File_C$, $File_D$, $File_E$), 5 vulnerable files ($File_F$, $File_G$, $File_H$, $File_I$, $File_J$), 5 non-vulnerable functions ($Func_A$, $Func_B$, $Func_C$, $Func_D$, $Func_E$) and 5 vulnerable functions ($Func_F$, $Func_G$, $Func_H$, $Func_I$, $Func_J$) with different levels of trustworthiness score ranging from the minimum to maximum. It is worth noting that the selection process is only based on the metrics and not based on the detailed analysis of the source code.

TABLE II: File-level metric weights.

File-level Metrics	MDG	MDA	MDAMDG	MDG_W	MDA_W	MDAMDG_W
FanOut	100,27	38,58	138,85	0,102	0,081	0,095
CountPath	95,55	41,45	137,00	0,098	0,087	0,094
SumMaxNesting	101,34	31,76	133,10	0,104	0,067	0,091
FanIn	88,84	32,74	121,58	0,091	0,069	0,084
SumCyclomaticStrict	79,38	26,98	106,37	0,081	0,057	0,073
SumCyclomaticModified	69,57	27,83	97,40	0,071	0,058	0,067
CountDeclFunction	68,96	28,04	97,01	0,070	0,059	0,067
AvgFanOut	61,08	31,76	92,84	0,062	0,067	0,064
SumCyclomatic	61,70	26,11	87,81	0,063	0,055	0,060
CountLineCodeDecl	52,08	22,39	74,47	0,053	0,047	0,051
CountSemicolon	47,89	20,87	68,77	0,049	0,044	0,047
AvgLineCode	25,23	19,68	44,91	0,026	0,041	0,031
AvgMaxNesting	22,95	19,83	42,78	0,023	0,042	0,029
AvgCyclomaticStrict	21,37	18,80	40,17	0,022	0,039	0,028
AvgCyclomaticModified	19,07	18,38	37,45	0,019	0,039	0,026
RatioCommentToCode	17,50	15,20	32,71	0,018	0,032	0,022
AltAvgLineBlank	10,85	13,02	23,87	0,011	0,027	0,016
CBO	9,62	14,06	23,68	0,010	0,030	0,016
AvgEssential	11,00	9,50	20,51	0,011	0,020	0,014
CountStmtEmpty	9,05	9,63	18,68	0,009	0,020	0,013
LCOM	5,42	9,54	14,96	0,006	0,020	0,010

TABLE III: Function-level metric weights.

Function-level Metrics	MDG	MDA	MDAMDG	MDG_W	MDA_W	MDAMDG_W
CountLine	205,62	37,72	243,34	0,145	0,094	0,134
CountOutput	165,97	47,77	213,74	0,117	0,119	0,117
CyclomaticModified	151,40	41,79	193,19	0,107	0,104	0,106
CountLineCode	146,21	26,65	172,86	0,103	0,066	0,095
CountSemicolon	130,89	29,84	160,73	0,092	0,074	0,088
CountStmtExe	127,00	31,83	158,83	0,089	0,079	0,087
CountLineCodeExe	126,13	24,66	150,79	0,089	0,061	0,083
CountStmtDecl	98,62	32,67	131,29	0,069	0,081	0,072
CountLineComment	86,10	43,43	129,53	0,061	0,108	0,071
Knots	54,31	22,46	76,77	0,038	0,056	0,042
MinEssentialKnots	30,25	17,69	47,94	0,021	0,044	0,026
MaxEssentialKnots	28,77	17,24	46,02	0,020	0,043	0,025
Essential	29,12	15,57	44,70	0,021	0,039	0,025
CountStmtEmpty	21,01	9,34	30,35	0,015	0,023	0,017
CountLineInactive	18,25	4,04	22,29	0,013	0,010	0,012

B. Trustworthiness Model-Based Ranking (TMR)

In this step we simply ordered the selected files/functions based on the three trustworthiness scores (MDG, MDA, MDAMDG).

As mentioned before, the software metrics considered in this study are the ones selected in our previous work [8] as the best subset of metrics for building the most accurate classifier over the Mozilla Firefox dataset. We calculated the three weights for each metric using MDG, MDA, and MDGMDA. The results obtained for the file and function levels are presented in Table II and Table III, respectively. The metrics are sorted based on the MDGMDA value. As shown in both tables, the weights calculated using MDG, MDA, and MDGMDA are different, but in most cases the order of importance of the metrics given by MDA and MDG is the same as of MDGMDA. The exceptions are marked in red.

The statistical data obtained for the file and function level metrics are summarized in Table IV and Table V, respectively.

TABLE IV: Statistical data of file-level metrics.

Software metrics	Lower Fence	Min.	Q1	Mean	Q3	Upper Fence	Max.	# outliers
FanOut	0	0	13	102,49	111	258	10136	2541
CountPath	0	0	8	29173420,8	213	520,5	8048563425	5118
SumMaxNesting	0	0	2	19,92	21	49,5	1893	2536
FanIn	0	0	8	108,76	85	200,5	129882	2743
SumCyclomaticStrict	0	0	9	88,00	87	204	12650	3121
SumCyclomaticModified	0	0	8	74,67	74	173	9129	3131
CountDeclFunction	0	0	3	18,05	20	45,5	1281	2616
AvgFanOut	0	0	3	5,72	7	13	190	1345
SumCyclomatic	0	0	8	79,16	79	185,5	9931	3109
CountLineCodeDecl	0	0	14	100,76	98	224	50216	2957
CountSemicolon	0	0	22	185,47	187	434,5	28029	3014
AvgLineCode	0	0	7	18,64	22	44,5	1177	1849
AvgMaxNesting	0	0	0	1,08	1	2,5	10	1860
AvgCyclomaticStrict	0	0	2	5,10	6	12	649	1753
AvgCyclomaticModified	0	0	2	4,25	5	9,5	428	1919
RatioCommentToCode	0	0	0,12	0,60	0,5	1,07	56	3019
AltAvgLineBlank	0	0	0	2,61	3	7,5	275	1807
CBO	0	0	0	8,49	0	0	7958	5600
AvgEssential	0	0	1	2,15	2	3,5	236	3713
CountStmtEmpty	0	0	0	5,66	2	5	4794	4274
LCOM	0	0	0	95,43	0	0	108941	3831

TABLE V: Statistical data of function-level metrics.

Software metrics	Lower Fence	Min.	Q1	Mean	Q3	Upper Fence	Max.	# outliers
CountLine	0	0	5	22,7	24	52,5	5552	58908
CountOutput	0	0	2	5,9	7	14,5	665	50150
CyclomaticModified	0	0	1	4,2	4	8,5	1798	66187
CountLineCode	0	0	5	17,8	19	40	3028	56943
CountSemicolon	0	0	1	10,1	10	23,5	3316	59143
CountStmtExe	0	0	1	10,5	11	26	3991	53070
CountLineCodeExe	0	0	1	11,1	12	28,5	2347	54652
CountStmtDecl	0	0	0	2,2	2	5	663	65322
CountLineComment	0	0	0	2,0	1	2,5	2181	103550
Knots	0	0	0	6,1	2	5	83057	71240
MinEssentialKnots	0	0	0	4,7	2	5	82998	57705
MaxEssentialKnots	0	0	0	4,7	2	5	83011	58694
Essential	0	0	1	2,5	3	6	1313	46769
CountStmtEmpty	0	0	0	0,2	0	0	2178	56418
CountLineInactive	0	0	0	0,6	0	0	1465	31989

Here, we intend to analyze the distribution of the values of the software metrics at both file and function levels, in order to identify the outliers that affect the normalization process. As previously explained, the outliers are identified by calculating the acceptable range of values based on IQR ($IQ_3 - Q_1$). The results of our statistical analysis show that, for each software metric, different number of values are considered to be outliers (refer to the last column of Table IV and Table V).

The trustworthiness scores (T Score) of the non-vulnerable files ($File_A$, $File_B$, $File_C$, $File_D$, $File_E$) and functions ($Func_A$, $Func_B$, $Func_C$, $Func_D$, $Func_E$) calculated by our trustworthiness model for three different weights are presented in Table VI and in Table VII. In both tables, the files and functions are sorted from the most trustworthy to the least trustworthy. It is interesting to notice that, independently of the weights (MDG_W, MDA_W and MDGMDA_W) used to calculate the trustworthiness score, the order of the files is always $File_C > File_A > File_E > File_D > File_B$, where the symbol > means *more trustworthy than*. The ranking for the

TABLE VI: Ranking of files without vulnerabilities.

MDG_W		MDA_W		MDGMDA_W	
Files	T Score	Files	T Score	Files	T Score
FileC	0,930	FileC	0,886	FileC	0,916
FileA	0,757	FileA	0,653	FileA	0,723
FileE	0,500	FileE	0,444	FileE	0,482
FileD	0,262	FileD	0,211	FileD	0,245
FileB	0,051	FileB	0,052	FileB	0,051

TABLE VII: Ranking of functions without vulnerabilities

MDG_W		MDA_W		MDGMDA_W	
Functions	T Score	Functions	T Score	Functions	T Score
FuncE	0,956	FuncE	0,950	FuncE	0,955
FuncD	0,746	FuncD	0,776	FuncD	0,753
FuncB	0,501	FuncB	0,467	FuncB	0,493
FuncA	0,232	FuncA	0,203	FuncA	0,226
FuncC	0,043	FuncC	0,029	FuncC	0,040

functions is also the same, independently of the weight used: $Func_E > Func_D > Func_B > Func_A > Func_C$

Regarding vulnerable unites of code, the order of files, independently of the weights (MDG_W, MDA_W and MDGMDA_W), is $File_I > File_G > File_J > File_F > File_H$. For functions, the order is $Func_G > Func_J > Func_F > Func_H > Func_I$, also independently of the weights. These results are presented in Tables VIII and IX, respectively.

VI. Approach Validation and Generalization

In order to validate the ranking provided by the proposed trustworthiness benchmarking approach we conducted an expert-based ranking. This ranking was compared with the one obtained by our approach, both for files/functions without vulnerabilities and files/functions with vulnerabilities.

TABLE VIII: Ranking of files with vulnerabilities.

MDG_W		MDA_W		MDGMDA_W	
Files	T Score	Files	T Score	Files	T Score
FileI	0,951	FileI	0,908	FileI	0,937
FileG	0,790	FileG	0,732	FileG	0,771
FileJ	0,506	FileJ	0,487	FileJ	0,500
FileF	0,240	FileF	0,277	FileF	0,252
FileH	0,045	FileH	0,048	FileH	0,046

TABLE IX: Ranking of functions with vulnerabilities

MDG_W		MDA_W		MDGMDA_W	
Functions	T Score	Functions	T Score	Functions	T Score
FuncG	0,951	FuncG	0,945	FuncG	0,950
FuncJ	0,770	FuncJ	0,763	FuncJ	0,768
FuncF	0,492	FuncF	0,520	FuncF	0,498
FuncH	0,258	FuncH	0,318	FuncH	0,271
FuncI	0,028	FuncI	0,027	FuncI	0,028

A. Expert Survey

Twelve experts with experience and interest in the security area (4 PhD students and 8 professors), working in different universities in different countries (Portugal, Italy, Brazil, Belgium), were asked to do a comparison between all the different pairs of files/functions mentioned in the previous section. It is important to mention that all experts are researchers in the area of secure software engineering, some of which having more than 10 years of experience.

The experts were provided with the source code of the files/functions and the available information regarding the individual software metrics. They were asked to choose the more trustworthy file/function for each possible pair (a total of 90 pairwise comparisons) and assign a value indicating how much trustworthy a file/function is in comparison to another. For this, we defined four different levels: *much more trustworthy* (3), *more trustworthy* (2), *a little more trustworthy* (1) or *non-differentiable* (0). We believe that defining more levels would highly complicate the judgment process for the experts, thus not bringing any added value. For supporting further calculations, we transformed these values into quantitative ones, as shown in Table X, according to the *Fundamental Scale of Absolute Numbers for Pairwise Comparison* [38].

TABLE X: Absolute numbers in pairwise comparison.

Definition	Description	Intensity
Equal	Non-differentiable	1
Moderate	Little more trustworthy	3
Strong	More trustworthy	5
Very strong	Much more trustworthy	7

B. Validation Process

For the validation, we first performed an analysis of responses from the experts and then calculated individual and aggregated rankings of the files and functions. Finally, we compared those individual and aggregated ranks with the ranking provided by our trustworthiness benchmarking approach. Figure 4 depicts the whole process.

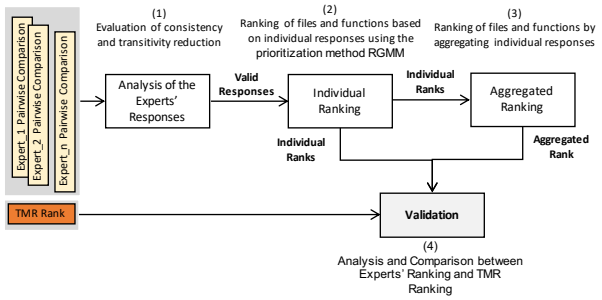

Fig. 4: Validation process.

1) Analysis of the Responses of the Experts: We started by analyzing the responses of the experts to find inconsistencies. An inconsistency occurs when an expert chooses A rather than B, B rather than C, but C rather than A as the more trustworthy file/function, while based on the two first statements, A must be more trustworthy than C. For this, we generated a complete graph to summarize the responses, and performed transitivity reduction to help identifying such inconsistencies. Inconsistent responses are decided to be eliminated from the validation process in order to avoid the effect of contradictory responses on the aggregated ranking of files/functions.

TABLE XI: Responses of the experts for files.

Files	C - A	C - E	C - D	C - B	A - E	A -D	A -B	E -D	E - B	D - B
Expert 1	A, 1	C, 1	C, 3	C, 1	A, 3	A, 2	A, 1	E, 2	E, 1	B, 2
Expert 2	C, 3	C, 3	C, 3	C, 3	A, 1	A, 3	A, 3	E, 3	E, 3	D, 2
Expert 3	C, 1	C, 1	C, 1	C, 3	A, 1	0	A, 3	0	E, 3	D, 3
Expert 4	0	C, 1	C, 2	C, 3	A, 1	A, 2	A, 3	0	E, 3	D, 1
Expert 5	C, 1	C, 2	C, 3	C, 3	A, 1	A, 2	A, 3	E, 1	E, 2	B, 1
Expert 6	0	C, 2	C, 3	C, 2	A, 2	A, 3	A, 2	E, 1	E, 1	B, 1
Expert 7	C, 3	C, 3	C, 3	C, 3	A, 2	A, 2	A, 2	E, 2	E, 2	0
Expert 8	C, 1	C, 3	C, 3	C, 3	A, 2	A, 2	A, 2	E, 1	E, 1	D, 1
Expert 9	C, 1	C, 3	C, 2	C, 3	A, 2	A, 1	A, 2	0	E, 1	D, 1

TABLE XII: Responses of the experts for functions.

Functions	E - D	E - B	E - A	E - C	D - B	D - A	D - C	B - A	B - C	A - C
Expert 1	E, 2	E, 1	E, 3	E, 3	D, 2	D, 1	D, 1	B, 2	B, 2	0
Expert 2	E, 3	E, 3	E, 3	E, 3	D, 2	D, 2	D, 3	B, 1	B, 1	A, 1
Expert 3	E, 3	E, 3	E, 3	E, 3	B, 1	D, 2	D, 2	B, 2	B, 2	0
Expert 4	0	E, 1	E, 1	E, 2	D, 1	D, 1	D, 2	0	B, 1	0
Expert 5	E, 1	E, 2	E, 3	E, 3	D, 1	D, 2	D, 3	B, 1	B, 2	A, 1
Expert 6	0	E, 3	0	E, 2	D, 3	0	D, 2	A, 3	C, 1	A, 1
Expert 7	0	E, 3	E, 3	E, 3	D, 1	D, 1	D, 1	A, 1	C, 1	A, 1
Expert 8	E, 1	E, 3	E, 3	E, 2	D, 3	D, 3	D, 1	A, 2	C, 2	C, 1
Expert 9	0	E, 3	E, 3	E, 3	D, 3	D, 3	D, 3	A, 2	C, 1	A, 1

The analysis of the responses resulted in the exclusion of 3 responses/experts (for both files and functions) from the whole process. Thus, 9 responses remain for further analysis. The pairwise comparison of files and functions made by the remaining experts are presented in Table XI and Table XII. Each row presents the comparisons carried out by one expert and the columns show the judgment made between each pair of files and functions. For example, in the first column of Table XI, C - A means that $File_C$ is being compared against $File_A$, and expert 1 (first row) chooses $File_A$ rather than $File_C$, but indicates 1, which means that $File_A$ is *a little more trustworthy* than $File_C$.

2) Individual and Aggregated Ranking: Based on the pairwise comparisons, we calculated individual and aggregated ranks for files and functions. There are two possible methods for calculating the individual ranks: the Eigenvalue Method (EM) and the Row Geometric Mean Method (RGMM) [10]. However, it is shown in [39] that the difference in the output of these two methods in meaningless, thus we choose RGMM, which requires less computation power.

To calculate the integrated rank of the files and functions, there are also two known methods: the Aggregation of Individual Judgments (AIJ) [11] and the Aggregation of Individual Priorities (AIP). As shown in [11], when RGMM is used to calculate individual rank, the output of both methods is equivalent. Thus, in this work we use AIJ.

By applying the RGMM and AIJ methods to compute the

TABLE XIII: Individual & aggregated rankings of files.

Expert 1 Rank		Expert 2 Rank		Expert 3 Rank		Expert 4 Rank		Expert 5 Rank	
FileA	0.458	**FileC**	0.563	**FileC**	0.427	**FileC**	0.366	**FileC**	0.497
FileC	0.266	**FileA**	0.218	**FileA**	0.221	**FileA**	0.366	**FileA**	0.270
FileE	0.135	**FileE**	0.141	**FileD**	0.177	**FileE**	0.137	**FileE**	0.133
FileB	0.103	**FileD**	0.051	**FileE**	0.142	**FileD**	0.094	**FileB**	0.058
FileD	0.037	**FileB**	0.027	**FileB**	0.032	**FileB**	0.036	**FileD**	0.042
Expert 6 Rank		**Expert 7 Rank**		**Expert 8 Rank**		**Expert 9 Rank**		**Agg. Rank**	
FileC	0.387	**FileC**	0.581	**FileC**	0.521	**FileC**	0.513	**FileC**	0.408
FileA	0.387	**FileA**	0.218	**FileA**	0.274	**FileA**	0.261	**FileA**	0.310
FileE	0.112	**FileE**	0.114	**FileE**	0.099	**FileD**	0.099	**FileE**	0.171
FileB	0.072	**FileD**	0.044	**FileD**	0.064	**FileE**	0.084	**FileD**	0.065
FileD	0.041	**FileB**	0.044	**FileB**	0.041	**FileB**	0.043	**FileB**	0.046

TABLE XIV: Individual & aggregated rankings of functions.

Expert 1 Rank		Expert 2 Rank		Expert 3 Rank		Expert 4 Rank		Expert 5 Rank	
FuncE	0.521	**FuncE**	0.579	**FuncE**	0.586	**FuncE**	0.348	**FuncE**	0.497
FuncD	0.216	**FuncD**	0.232	**FuncB**	0.198	**FuncD**	0.348	**FuncD**	0.270
FuncB	0.154	**FuncB**	0.093	**FuncD**	0.128	**FuncB**	0.131	**FuncB**	0.133
FuncA	0.055	**FuncA**	0.060	**FuncA**	0.044	**FuncA**	0.105	**FuncA**	0.065
FuncC	0.055	**FuncC**	0.036	**FuncC**	0.044	**FuncC**	0.069	**FuncC**	0.035
Expert 6 Rank		**Expert 7 Rank**		**Expert 8 Rank**		**Expert 9 Rank**		**Agg.Rank**	
FuncE	0.305	**FuncE**	0.467	**FuncE**	0.488	**FuncE**	0.405	**FuncE**	0.458
FuncD	0.305	**FuncD**	0.281	**FuncD**	0.284	**FuncD**	0.405	**FuncD**	0.303
FuncA	0.275	**FuncA**	0.123	**FuncC**	0.130	**FuncA**	0.099	**FuncA**	0.094
FuncC	0.079	**FuncC**	0.079	**FuncA**	0.066	**FuncC**	0.058	**FuncB**	0.082
FuncB	0.037	**FuncB**	0.051	**FuncB**	0.031	**FuncB**	0.034	**FuncC**	0.063

individual and aggregated rankings of files and functions, we obtained the results presented in Table XIII and Table XIV.

3) Validation: The aggregated ranking in Table XIII (files) is exactly equally to the one computed by our benchmarking approach: $File_C > File_A > File_E > File_D > File_B$. Considering the individual rankings from the experts, 4 out of 9 experts ranked the files in that exactly same order. In the other cases, the experts partially indicate the same order (e.g., $File_C > File_A > File_E > File_B$ appears in 4 responses) and indicate a reverse order for the files that are close to each other in terms of their trustworthiness score ($File_D$ and $File_B$, with 0.245 and 0.051 MDGMDA scores, respectively). These results, on one hand, show that the inconsistency analysis and exclusion of the inconsistent responses were effectively done. On the other hand, they show that our trustworthiness approach provides a sound ranking.

Regarding functions (Table XIV), again 4 out of the 9 experts indicate the same ranking as our approach, but the order of $Func_A$ and $Func_B$ is reversed in the aggregated ranking. This is happening because 4 experts, that are partially indicating the same order as our approach ($Func_E > Func_D > Func_A > Func_C$), choose $Func_B$ as the least trustworthy function, while it has a higher score than $Func_A$ according to our ranking. For example, Expert6's values for $Func_B$ and $Func_A$ are 0.037 and 0.275, while their scores in our approach are 0.493 and 0.226, respectively. Thus, despite having several experts with the same rank as ours, this significant distance (from 0.493 to 0.037) between our rank and several experts (4 experts) in the case of $Func_B$, greatly influences the result of the aggregated ranking.

TABLE XV: Individual & aggregated rankings of vulnerable files.

Expert 1 Rank		Expert 2 Rank		Expert 3 Rank		Expert 4 Rank		Expert 5 Rank	
FileI	0,302	**FileI**	0,570	**FileI**	0,281	**FileI**	0,379	**FileI**	0,531
FileG	0,302	**FileG**	0,177	**FileG**	0,281	**FileG**	0,244	**FileJ**	0,186
FileH	0,302	**FileJ**	0,177	**FileJ**	0,281	**FileJ**	0,142	**FileF**	0,142
FileF	0,064	**FileF**	0,037	**FileF**	0,106	**FileF**	0,142	**FileG**	0,101
FileJ	0,029	**FileH**	0,037	**FileH**	0,050	**FileH**	0,093	**FileH**	0,041
Expert 6 Rank		**Expert 7 Rank**		**Expert 8 Rank**		**Expert 9 Rank**		**Agg. Rank**	
FileI	0,476	**FileI**	0,403	**FileI**	0,414	**FileI**	0,511	**FileI**	0,429
FileG	0,306	**FileG**	0,259	**FileJ**	0,254	**FileJ**	0,194	**FileG**	0,233
FileJ	0,089	**FileJ**	0,134	**FileG**	0,193	**FileG**	0,113	**FileJ**	0,165
FileF	0,089	**FileF**	0,134	**FileF**	0,090	**FileF**	0,091	**FileF**	0,103
FileH	0,040	**FileH**	0,069	**FileH**	0,049	**FileH**	0,091	**FileH**	0,070

TABLE XVI: Individual & aggregated rankings of vulnerable functions.

Expert 1 Rank		Expert 2 Rank		Expert 3 Rank		Expert 4 Rank		Expert 5 Rank	
FuncG	0,406	**FuncG**	0,566	**FuncG**	0,380	**FuncG**	0,555	**FuncG**	0,477
FuncJ	0,406	**FuncJ**	0,243	**FuncJ**	0,380	**FuncJ**	0,230	**FuncJ**	0,308
FuncF	0,100	**FuncF**	0,112	**FuncF**	0,118	**FuncF**	0,097	**FuncF**	0,097
FuncH	0,047	**FuncH**	0,040	**FuncH**	0,061	**FuncH**	0,066	**FuncH**	0,072
FuncI	0,042	**FuncI**	0,040	**FuncI**	0,061	**FuncI**	0,053	**FuncI**	0,046
Expert 6 Rank		**Expert 7 Rank**		**Expert 8 Rank**		**Expert 9 Rank**		**Agg. Rank**	
FuncG	0,473	**FuncG**	0,614	**FuncG**	0,523	**FuncG**	0,609	**FuncG**	0,462
FuncJ	0,199	**FuncJ**	0,136	**FuncJ**	0,193	**FuncJ**	0,168	**FuncJ**	0,272
FuncF	0,160	**FuncF**	0,136	**FuncF**	0,174	**FuncF**	0,108	**FuncF**	0,153
FuncH	0,093	**FuncH**	0,057	**FuncH**	0,055	**FuncI**	0,070	**FuncH**	0,057
FuncI	0,075	**FuncI**	0,057	**FuncI**	0,055	**FuncH**	0,045	**FuncI**	0,055

To further verify whether the proposed approach for trustworthiness benchmarking is valid, we repeated the whole process with the files and functions containing known vulnerabilities. After analyzing the results, presented in Tables XV and XVI, we observed that the files ranking obtained by our trustworthiness approach, $File_I > File_G > File_J > File_F > File_H$, matches the aggregated ranking of the experts, which also maps to the individual ranking of the majority of the experts (5 out of 9). The same happens for the functions. Our ranking is $Func_G > Func_J > Func_F > Func_H > Func_I$, which corresponds to the aggregated ranking and to the individual ranking of the majority of the experts (8 out of 9).

C. Threats to Validity and Generalization of the Approach

In this work we propose a trustworthiness assessment and a benchmarking approach based on software metrics that can be used as indicators of software security. Based on trustworthiness score we ranked several files and functions of the Mozilla Firefox project and conducted a survey among several software security experts in order to validate the obtained rank. Despite

the results showed a sound ranking of the benchmarked files and functions, threats to validity should be highlighted:

- Number of files and functions selected - the instantiation and validation of the trustworthiness benchmark were performed using a total of 10 files (5 non-vulnerable and 5 vulnerable) and 10 functions (5 non-vulnerable and 5 vulnerable). Although we obtained the trustworthiness score for all the files and functions of the dataset, we needed to reduce this number in order to perform the validation. This is an acceptable number of files and functions, since the experts had to perform the comparison between all the possible pairs.
- Number of experts considered - our aim was to get as many responses as possible from experts in software security field, however this was a difficult task. Nevertheless, we consider that the results obtained provide a good basis for demonstrating the validity of the proposed approach.

In order to generalize the proposed approach we repeated the trustworthiness benchmarking process using different software metrics. Since the metrics used for building the trustworthiness model were selected considering the Mozilla Firefox dataset, we need to verify whether the ranking results are biased. In practice, we repeated the whole ranking process for the same files and functions but using another set of software metrics, considered as the best set for distinguishing vulnerable code units from non-vulnerable ones in a dataset containing the combination of five different projects (Mozilla Firefox, Linux Kernel, Apache, Xen and Glibc). Note that, there are common sofware metrics in the two sets of metrics (set of metrics selected for Mozilla Firefox and set of metrics selected for all projects combined), for both files and functions. For files, the new set of software metrics includes a total of 16 metrics of which 6 are new (AltAvgLineCode, AltCountLineComment, AvgCyclomatic, AvgFanIn, CountLineBlank and SumEssential). For functions, there are also a total of 16 metrics of which 7 are new (AltCountLineCode, CountInput, CountLineCodeDecl, CountPath, CountStmt, Cyclomatic and RatioCommentToCode). Refer to [8] for more details on this subset of metrics.

As shown in Tables XVII, XVIII, XIX, and XX, the same ranking, with slightly different trustworthiness score, is achieved respectively for files and functions without and with vulnerabilities. For example, the trustworthiness model-based ranking obtained using the set of software metrics selected for Mozilla for files without vulnerabilities is $File_C > File_A > File_E > File_D > File_B$, the same as the ranking presented in table XVII (obtained using the selected metrics for different projects combined). In addition the values of the trustworthiness score are similar: $0,916 > 0,723 > 0,482 > 0,245 > 0,051$ using the dataset of Mozilla project, and $0,930 > 0,735 > 0,519 > 0,216 > 0,062$ uning the dataset with all projects combined. These results suggest that the approach can be generalized to other software projects.

VII. Conclusion

In this paper, we proposed an approach for assessing and benchmarking software trustworthiness. The approach is focused on security and is based on software metrics as indicators of security vulnerabilities. We used our approach to rank several files and functions from the Mozilla Firefox project considering their trustworthiness level.

For validation, we conducted a survey among several software security experts and calculated individual and aggregated ranks of the same files and functions based on their pairwise comparison of files and functions. Those ranks were compared with the ranks obtained with our approach. The results show that the rankings of the experts greatly match the one obtained by our trustworthiness model-based approach, thus, the proposed trustworthiness benchmark is able to provide a sound ranking of the benchmarked files and functions. This demonstrates the possibility of using software metrics for benchmarking the trustworthiness of software systems. We finally showed that the proposed approach can be generalized to other software project.

As a future work we intend to apply this benchmark ap-

TABLE XVII: Rank of files without vulnerabilities.

MDG_W		MDA_W		MDGMDA_W	
Files	**Tscore**	**Files**	**Tscore**	**Files**	**Tscore**
FileC	0,931	**File**C	0,929	**File**C	0,930
FileA	0,761	**File**A	0,677	**File**A	0,735
FileE	0,527	**File**E	0,501	**File**E	0,519
FileD	0,225	**File**D	0,197	**File**D	0,216
FileB	0,063	**File**B	0,059	**File**B	0,062

TABLE XVIII: Rank of files with vulnerabilities.

MDG_W		MDA_W		MDGMDA_W	
Files	**Tscore**	**Files**	**Tscore**	**Files**	**Tscore**
FileI	0,957	**File**I	0,940	**File**I	0,951
FileG	0,799	**File**G	0,760	**File**G	0,787
FileJ	0,603	**File**J	0,586	**File**J	0,598
FileF	0,250	**File**F	0,261	**File**F	0,254
FileH	0,005	**File**H	0,009	**File**H	0,006

TABLE XIX: Rank of functions without vulnerabilities

MDG_W		MDA_W		MDGMDA_W	
Functions	**Tscore**	**Functions**	**Tscore**	**Functions**	**Tscore**
FuncE	0,906	**Func**E	0,870	**Func**E	0,898
FuncD	0,740	**Func**D	0,712	**Func**D	0,734
FuncB	0,633	**Func**B	0,611	**Func**B	0,628
FuncA	0,555	**Func**A	0,521	**Func**A	0,547
FuncC	0,341	**Func**C	0,318	**Func**C	0,336

TABLE XX: Rank of functions with vulnerabilities.

MDG_W		MDA_W		MDGMDA_W	
Functions	**Tscore**	**Functions**	**Tscore**	**Functions**	**Tscore**
FuncG	0,910	**Func**G	0,873	**Func**G	0,901
FuncJ	0,813	**Func**J	0,783	**Func**J	0,806
FuncF	0,673	**Func**F	0,655	**Func**F	0,669
FuncH	0,481	**Func**H	0,469	**Func**H	0,478
FuncI	0,368	**Func**I	0,337	**Func**I	0,360

proach to different projects and also extend the benchmarking process considering different trustworthiness attributes, such as availability and performance.

ACKNOWLEDGMENTS

This work has been partially supported by the project ATMOSPHERE (https://www.atmosphere-eubrazil.eu/), funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement nº 777154.

REFERENCES

[1] N. G. Mohammadi, S. Paulus, M. Bishr, A. Metzger, H. Könnecke, S. Hartenstein, T. Weyer, and K. Pohl, "Trustworthiness attributes and metrics for engineering trusted internet-based software systems," in *International Conference on Cloud Computing and Services Science*. Springer, 2013, pp. 19–35.

[2] L. Cao, "Dynamic capability for trustworthy software development," *Journal of Software: Evolution and Process*, vol. 24, no. 7, pp. 837–850, 2012.

[3] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions." *HASP@ ISCA*, vol. 11, 2013.

[4] S. Ding, X.-J. Ma, and S.-L. Yang, "A software trustworthiness evaluation model using objective weight based evidential reasoning approach," *Knowledge and information systems*, vol. 33, no. 1, pp. 171–189, 2012.

[5] M. Y. Liu and I. Traore, "Empirical relation between coupling and attackability in software systems: a case study on dos," in *Proceedings of the 2006 workshop on Programming languages and analysis for security*. ACM, 2006, pp. 57–64.

[6] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[7] H. Alves, B. Fonseca, and N. Antunes, "Software metrics and security vulnerabilities: Dataset and exploratory study," in *12th European Dependable Computing Conference (EDCC)*. IEEE, 2016, pp. 37–44.

[8] N. Medeiros, N. Ivaki, P. Costa, and M. Vieira, "Software metrics as indicators of security vulnerabilities," in *Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on*. IEEE, 2017, pp. 216–227.

[9] M. L. Calle and V. Urrea, "Letter to the editor: stability of random forest importance measures," *Briefings in bioinformatics*, vol. 12, no. 1, pp. 86–89, 2010.

[10] G. Crawford and C. Williams, "A note on the analysis of subjective judgment matrices," *Journal of mathematical psychology*, vol. 29, no. 4, pp. 387–405, 1985.

[11] Y. Dong, G. Zhang, W.-C. Hong, and Y. Xu, "Consensus models for ahp group decision making under row geometric mean prioritization method," *Decision Support Systems*, vol. 49, no. 3, pp. 281–289, 2010.

[12] J.-H. Cho, K. Chan, and S. Adali, "A survey on trust modeling," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, p. 28, 2015.

[13] R. Hardin, *Trust and trustworthiness*. Russell Sage Foundation, 2002.

[14] D. Artz and Y. Gil, "A survey of trust in computer science and the semantic web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 58–71, 2007.

[15] H. Wang, Y. Tang, G. Yin, and L. Li, "Trustworthiness of internet-based software," *Science in China Series F: Information Sciences*, vol. 49, no. 6, pp. 759–773, 2006.

[16] Y. Li, Z. Wu, and Y. Chen, "Software trustworthiness static measurement model and the tool," *International Journal of Performability Engineering*, vol. 13, no. 7, p. 1101, 2017.

[17] Y. Wang and J. Vassileva, "A review on trust and reputation for web service selection," in *Distributed Computing Systems Workshops, 2007. ICDCSW'07. 27th International Conference on*. IEEE, 2007, pp. 25–25.

[18] N. Limam and R. Boutaba, "Assessing software service quality and trustworthiness at selection time," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 559–574, 2010.

[19] C. Cachin and M. Schunter, "A cloud you can trust," *IEEE Spectrum*, vol. 48, no. 12, pp. 28–51, 2011.

[20] M. Nitti, R. Girau, and L. Atzori, "Trustworthiness management in the social internet of things," *IEEE Transactions on knowledge and data engineering*, vol. 26, no. 5, pp. 1253–1266, 2014.

[21] M. Chiregi and N. J. Navimipour, "A comprehensive study of the trust evaluation mechanisms in the cloud computing," *Journal of Service Science Research*, vol. 9, no. 1, pp. 1–30, 2017.

[22] W. Hasselbring and R. Reussner, "Toward trustworthy software systems," *Computer*, vol. 39, no. 4, pp. 91–92, 2006.

[23] S. M. Habib, S. Ries, and M. Muhlhauser, "Towards a trust management system for cloud computing," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE, 2011, pp. 933–939.

[24] N. P. D. S. Medeiros, N. R. Ivaki, P. N. Da Costa, and M. P. A. Vieira, "Towards an approach for trustworthiness assessment of software as a service," in *Edge Computing (EDGE), 2017 IEEE International Conference on*. IEEE, 2017, pp. 220–223.

[25] J.-H. Cho, P. M. Hurley, and S. Xu, "Metrics and measurement of trustworthy systems," in *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 2016, pp. 1237–1242.

[26] Y. Yang, Q. Wang, and M. Li, "Process trustworthiness as a capability indicator for measuring and improving software trustworthiness," in *International Conference on Software Process*. Springer, 2009, pp. 389–401.

[27] B. K. Jayaswal and P. C. Patton, *Design for trustworthy software: Tools, techniques, and methodology of developing robust software*. Pearson Education, 2006.

[28] T. S. Ankrum and A. H. Kromholz, "Structured assurance cases: Three common standards," in *High-Assurance Systems Engineering, 2005. HASE 2005. Ninth IEEE International Symposium on*. IEEE, 2005, pp. 99–108.

[29] G. Disterer, "Iso/iec 27000, 27001 and 27002 for information security management," *Journal of Information Security*, vol. 4, no. 02, p. 92, 2013.

[30] H.-W. Jung, S.-G. Kim, and C.-S. Chung, "Measuring software product quality: A survey of iso/iec 9126," *IEEE software*, vol. 21, no. 5, pp. 88–92, 2004.

[31] I. ISO and I. Std, "Iso 15408-1: 2009," *Information technology-Security techniques-Evaluation criteria for IT security-Part*, vol. 1.

[32] M. A. Cusumano, "Who is liable for bugs and security flaws in software?" *Communications of the ACM*, vol. 47, no. 3, pp. 25–27, 2004.

[33] Z. Yan and C. Prehofer, "Autonomic trust management for a component-based software system," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 810–823, 2011.

[34] S. Ding, S.-L. Yang, and C. Fu, "A novel evidential reasoning based method for software trustworthiness evaluation under the uncertain and unreliable environment," *Expert Systems with Applications*, vol. 39, no. 3, pp. 2700–2709, 2012.

[35] SciTools, "Understand static code analysis tool," 2017. [Online]. Available: https://scitools.com/feature/metrics/

[36] S. Walfish, "A review of statistical outlier methods," *Pharmaceutical technology*, vol. 30, no. 11, p. 82, 2006.

[37] H. Han, X. Guo, and H. Yu, "Variable selection using mean decrease accuracy and mean decrease gini based on random forest," in *Software Engineering and Service Science (ICSESS), 2016 7th IEEE International Conference on*. IEEE, 2016, pp. 219–224.

[38] M. Martinez, D. de Andres, J.-C. Ruiz, and J. Friginal, "From measures to conclusions using analytic hierarchy process in dependability benchmarking," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 11, pp. 2548–2556, 2014.

[39] Y. Dong, Y. Xu, H. Li, and M. Dai, "A comparative study of the numerical scales and the prioritization methods in ahp," *European Journal of Operational Research*, vol. 186, no. 1, pp. 229–242, 2008.