Proactive Trust Assessment of Systems as Services

Jorge López[1], Natalia Kushik[1,2] and Nina Yevtushenko[2,3]
[1]SAMOVAR, CNRS, Télécom SudParis, Université Paris-Saclay, 9 rue Charles Fourier 91011 EVRY, France
[2]Department of Information Technologies, Tomsk State University, 36 Lenin str., 634050 Tomsk, Russia
[3] Software Engineering Department, Institute for System Programming of the Russian Academy of Sciences, 25 Alexander Solzhenitsyn str., 109004 Moscow, Russia

Keywords: Systems as Services, Machine Learning, Dynamic Code Analysis, Trust, Software Testing.

Abstract: The paper is devoted to the trust assessment problem for specific types of software/hardware systems, namely Systems as Services. We assume that such systems are designed and utilized in all application domains, and therefore the aspects of trust are becoming crucial. Moreover, these systems are mainly used on-demand and are often represented by a composition of 'smaller' services. Thus, an effective method for estimating/assessing the trust level of a given component service (or a system as a whole) needs to be utilized. Most known methods and techniques for trust evaluation mainly rely on the passive testing and system monitoring; in this paper, we propose a novel approach for this problem taking advantage of active testing techniques. Test sequences to be applied to a system/service under test are derived based on determining the critical values of non-functional service parameters. A set of these parameters can be obtained via a static code analysis of the system/service or by addressing available experts. Machine learning techniques can be applied later on, for determining critical parameter values and thus, deriving corresponding test sequences. The paper contains an illustrative example of RESTFul web service which components are checked w.r.t. critical trust properties.

1 INTRODUCTION

The concept of Systems as Services (SaS) is ambitious and emerging, as it aims to expand the existing "cloud" concepts to any type of system (Ardagna et al., 2015). In fact, it is a natural system evolution to start at different locations, and then slowly move to a utility service. In recent years, cloud computing entitled computational resources to be distributed as a utility service, and this concept has encouraged to provide any system as a service. Arguably, augmenting a system with a cloud/web interface might enable it to become a SaS. Nevertheless, there exist several issues, such as interoperability, system/service composition, trustworthiness, etc., that need to be considered. In order to rely on such SaS from a user or a provider point of view, the trustworthy level of SaS needs to be guaranteed.

Trust as a computer science concept is an active field of research in the scientific community. In the literature, two main trust notions are used, namely i) *hard trust*, which is based on security policies, and ii) *soft trust*, which is based on different dynamic parameters. The hard trust approach is rigid; the trustees have predefined sets of privileges granted, and the system's interactions are managed by the acceptance or rejection of actions based on these predefined privileges. Some examples of pioneering works on hard trust can be found in (Lee et al., 2009; Blaze et al., 1996; Jim, 2001). On the other hand, soft trust is flexible, and the interactions are managed depending on the trust level of a trustee at a given time instance. Intuitively, the level of trust depends on a set of "trust parameters"; according to the literature, the most common trust parameters include experience, reputation, and risk. Soft trust has been applied to different domains, some examples of such applications can be found in (Chen and Guo, 2014; López and Maag, 2015).

Both major approaches for trust definition have their own drawbacks. Hard trust guarantees that some entities can have access to certain resources regardless of potentially untrustworthy behavior of such entities at execution time. Soft trust, on the other hand, is a reactive approach, i.e., an untrustworthy behavior needs to occur before establishing that a trustee is not trustworthy.

López, J., Kushik, N. and Yevtushenko, N.
Proactive Trust Assessment of Systems as Services.
DOI: 10.5220/0006354502710276
In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017)*, pages 271-276
ISBN: 978-989-758-250-9

To overcome the inherent and potentially present disadvantages that the current trust approaches tend to have, we propose a proactive trust assessment approach. The key idea behind a proposed approach is based on guaranteeing that a given system can only produce trustworthy outputs under critical inputs to such system. Therefore, the approach can be seen as some form of a certification method and could/should be applied to a system under design when application of critical inputs does not jeopardize the data of the application or the system's data are not susceptible to such input application. The problem statement we address in this paper is as follows: what are the important/critical trust parameters for systems as services, what are the crucial values of those and how to actively test such systems to ensure that they can produce only trustworthy interactions when the parameters reach their critical values? We mention that testing techniques have been previously developed for cloud environments, however they mostly cover the security checking of the corresponding applications/services. A comprehensive review of such techniques is given in (Ardagna et al., 2015).

The method proposed in the paper is divided into two main phases. The first phase is to determine the parameters that might affect the trustworthiness of the system. For this purpose, we propose to build a dataset of potentially sensitive trust parameters, given by experts, for example. Such list of parameters can also be specified by service providers. Then, by the use of supervised machine learning techniques a trust prediction model is derived. The second phase relies on applying a proper test suite in order to verify the trustworthiness of the system. The verification of the trustworthiness of the system is based on the extraction of the values of the sensitive trust parameters. The approach is illustrated on a running example of a RESTFul web service.

The rest of the paper is organized as follows. Section 2 contains preliminary concepts. Section 3 describes how to extract the sensitive trust variables from the source code of a System Under Test (SUT). Section 4 contains the description of the approach for applying different input (test) sequences to the SUT that contain sensitive values of critical variables; the prediction model is used to determine if the SUT produces values considered to be untrustworthy. Finally, Section 5 concludes the paper.

2 PRELIMINARIES

2.1 Systems as Services and Trust Issues

The concept of *Systems as Services* (SaS) has emerged and been used nowadays almost everywhere. We assume that such systems are represented as compositions of heterogeneous hardware and/or software modules. These modules are usually created by different producers or service providers. The main goal of such creation is to meet user requirements and as a result, to deliver a high-level Quality of Experience (QoE). At the same time, a user experience can never be guaranteed before the trust of the service components is estimated thoroughly.

Grandison and Sloman define trust as "the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context" (Grandison and Sloman, 2000). In other words, the trustworthiness of a SaS involves different aspects. For example, it is assumed that a SaS must be capable (competent) of performing the task it is designed, and inquired for. If the SaS contains defects (or bugs), the SaS is not considered to be trustworthy. The levels of trustworthiness of a SaS can vary. It is naturally to use three different trust levels (López and Maag, 2015): *Trustworthy*, for systems that are entirely trusted; *Untrustworthy*, for systems that are distrusted; and *Neutrally Trusted*, for systems that are partially trusted. We note that the systems associated with the last trust level can be of a wide use as well. For example, one might still interact with a SaS which has neutral trust, using a limited set of non-critical operations.

2.2 Active Testing Techniques

Active testing techniques and approaches are used in software testing for assuring the software quality. Usually, active testing includes the generation of test sequences or test cases, application/executions of those against a SUT and drawing the conclusion about the SUT properties. The notion *active* in this case underlines the fact that the system is being stimulated via its input interfaces. Various test generation strategies define the test fault coverage, i.e. the set of (program) faults that can be detected by a given test suite. The test length is another important test quality criterion, and therefore, there have been proposed numerous test generation approaches. These approaches start from random input generation and finish with complex model

based techniques that demonstrate lower performance but higher fault coverage.

The known trust assessment approaches in telecommunication systems mostly rely on passive testing techniques or monitoring (López, 2015) when only system observation is performed. In this paper, we take advantage of stimulating the SUT for producing the untrustworthy actions, i.e. we intensely try to apply the critical input data to the SaS which can potentially be never observed during its ordinary monitoring. The decision about which actions can be considered 'suspicious' is done with the use of supervised machine learning techniques.

2.3 Supervised Machine Learning

Machine learning algorithms are designed to learn from available data in order to make predictions/estimations. Typically, the machine learning algorithms operate by building a model. The model "learns" to predict from the data without being explicitly instructed. Supervised machine learning algorithms take training examples along with their expected outputs as the algorithm's inputs. The final goal is to get a machine (after learning) that maps the training examples to their expected outputs.

Formally, the inputs are called *features*. A *feature vector* is an *n*-tuple of the different inputs. The expected output for a given feature vector is a *label*. The set of examples, called a *training set*, consists of pairs of a feature vector and a label. The objective is to find a function called the hypothesis, that maps a given feature vector to a label. Therefore, the objective is, in fact, to find the function that minimizes the error between the predictions and the real output.

There exist various known machine learning techniques starting from the classical linear regressions and decision trees, and finishing with Support Vector Machines (SVMs) and their modifications. In this paper, we use such techniques in order to estimate if the SUT behaves in a trustworthy manner. The corresponding training set is derived based on the example data provided by an expert's assessment. Another important step before training the model, is the relevant features' selection. If a certain parameter p_i does not affect the overall prediction result (a label), deleting this parameter makes the process more scalable. Usually, such statistical parameter analysis is considered as a pre-processing phase. There exist some well-known methods for selecting the relevant parameters. In (Blum and Langley, 1997), the authors showcase a number of such methods.

3 EXTRACTING RELEVANT PARAMETERS FOR SAS TRUST ASSESSMENT

Different systems have different parameters that influence the trustworthiness of the system. Some parameters might be related to functional properties of the system while others can represent various non-functional ones, such as, for example response time for a given request. Occasionally, the trust parameters might be observable as system outputs. For example, a trust parameter can be the support of certain algorithms of encryption of the communication, and the respective size of the encryption key of a SUT. On the other hand, a desired trust parameter can be an internal variable, which might influence the behavior of the SUT. Thus, we assume that the trust parameters are variables and outputs in the associated source code of the system. Therefore, the question arises: how can different trust parameters be observed?

In order to better exemplify which trust parameters may be associated with the source code, consider a SaS with a RESTFul web service (Pautasso et al., 2008). RESTFul web services use the hyper text transfer protocol (HTTP) as an underlying communication protocol. HTTP defines standard request methods, however, the HTTP version 1.1 as specified in RFC 2616, allows arbitrary extension methods. Further specifications such as the semantics and content of HTTP version 1.1 found in the RFC 7231 state that additional methods need to be registered in AINA (see section 4.1 and 8.2 of the RFC). However, widely-spread web servers as the Apache HTTP Server might allow extension methods and leave the application/framework to 'decide' how to process it. Many frameworks allow the use of extended methods. The behavior of such frameworks for extended methods can include treating an extended method as a normal GET request. However, some policies of authentication are only applied to GET or POST request. The result is that the application might allow bypassing security/authentication mechanisms by a request with an extended method (Dabirsiaghi, 2016). Other frameworks might even output the source code of the application.

Assume a SaS framework/application is implemented as shown in the pseudo-code below:

```
$httpMethod=getMethod($httpRequest);
…
$startTime = getTime();
($responseCode, $httpResponse) =
 processData($httpMethod, $URI);
$time = startTime - getTime();
…
```

The variable *$httpMethod* contains the HTTP request method, *$URI* contains the requested resource, and *$responseCode* contains the response code after the request is processed. The variable *$time* contains the time to process the request. A trust/security expert might find untrustworthy if a known method to a given resource replies a redirection (3xx HTTP response code), and an extended method replies with success (2xx HTTP response code). The expert can find untrustworthy the systems that exceed one second to process a request, given the fact that the implementation does not reply in a proper time. Therefore, from the observation of the internal variables and outputs of the source code the expert can provide a trust assessment. The trust parameters in this example are: *$httpMethod*, *$responseCode*, *$URI,* and *$time*.

Given the fact that different systems have different trust parameters, we assume that the initial selection of them is performed by an expert which can often be represented by a developer or a service provider. Furthermore, the expert can provide a set of ranges for those parameters when the application is considered to be trustworthy. Based on the assessment of the expert a dataset can be derived; this dataset contains different classifications of the trustworthiness of a system. For example, the expert can define three different classes of trust, trusted (3), not trusted (1) and neutrally trusted (2). The dataset D contains a set of vectors of values for all trust parameters p_1, p_2, …, p_n. Each vector of values $\left(p_{1_j}, p_{2_j}, \ldots, p_{n_j}\right)$ has an associated trust evaluation $T_j \in$ [1-3], for j = 1, 2,…, m, where m is the size of the dataset. The dataset D can be used as a training set for a supervised machine learning problem. In (López and Maag, 2015), the authors proposed a multi-class classification trust prediction model based on Support Vector Machines (Boser et al., 1992). Depending on the characteristics of the dataset, one might be interested to use other supervised machine learning techniques, which are faster, such as the well-known logistic regression or a scalable prediction model based on logic circuits (Kushik et al., 2016). In general, the prediction model is adapted to its requirements. The trust prediction model separates the data into the trusted, untrusted, and neutrally trusted hyperspaces. By the observation of the different values of the system parameters, one can conclude about the level of trustworthiness of the system. After selecting the parameters, the relevant feature selection pre-processing should be performed.

On the other hand, after the relevant feature (and training example) selection, the dataset can be used to build a trust prediction model using any supervised machine learning technique which allows multi-class classification. Then, the model M that predicts the trust level (in the [1-3] range) based on the sensitive trust parameters is obtained.

4 ACTIVE TESTING TECHNIQUES FOR TRUST ASSESSMENT

As mentioned above, to the best of our knowledge existing trust evaluation methodologies for telecommunications mostly rely on passive testing assessment, i.e. when the conclusions are being made while only observing the behavior of the SUT. The idea behind the proposed approach is, on the contrary, trying to violate the SUT by applying specific inputs that can make this SUT behave untrustworthy. The approach relies on the machine learning techniques that can be used to predict the trust level of the SUT under the given values of its internal/external variables. In this case, the machine that predicts such trust level plays a role of an oracle or a specification that 'knows' the expected outcome, i.e. the level of trust for the SUT under the given conditions. The test suite to be executed against the SUT can be derived in different ways, starting from random simulation and finishing with model based test generation techniques. If the source code of the SUT is (partially) open, the test suite TS can be also derived using static analysis.

Given the SUT *Sys*, its source code *SC*, a set $P = \{p_1, \ldots, p_n\}$ of relevant SUT variables and a machine M that predicts the trust level T based on the values $\mathbf{v_1}$, …, $\mathbf{v_n}$ of p_1, …, p_n, correspondingly, TS is a test suite derived for the SUT *Sys*. Algorithm 1 can be used to assess the trustworthiness of *Sys* w.r.t. $P = \{p_1, \ldots, p_n\}$. We assume that the SUT *Sys* behaves trustworthy if for a given test sequence the machine M assures the trust level T greater or equal to a given constant $K1$. We also take into account how many sequences of a test suit actually bring the SUT to an untrustworthy state. We compare this value with a given constant $K2$ that represents the 'allowed' level of fluctuations, i.e., represents the percentage of test sequences for which the oracle M returns a level $T < K1$. We note that as M is the machine that was previously learned by experts and/or users/developers, this percentage $K2$ is usually chosen as $K2 < 10$.

Algorithm 1 returns the verdict 'Pass' whenever the oracle M represented by a (self-) adaptive model replies that the values $\mathbf{v_1}$, …, $\mathbf{v_n}$ of variables p_1, …, p_n do not reach their critical (from the trust point of view) values more than for $K2$ percent of test cases.

Algorithm 1 for assessing the trustworthiness of the SUT based on active testing
Input: SUT *Sys* with the source code *SC*, a set $P = \{p_1, \ldots, p_n\}$ of parameters/variables, a machine *M*, a minimal trust level *K*1, an allowed untrustworthy percentage *K*2, a test suite *TS*
Output: 'Pass' or 'Fail'
1. $i := 1$; $j := 0$;
2. If $(i > |\ TS\ |)$
then Return the verdict 'Pass'.
3. Execute the test sequence $\alpha_i \in TS$ against the SUT *Sys*;
Trace (observe in the code *SC*) the values $\mathbf{v_1}, \ldots, \mathbf{v_n}$ of $p_1, \ldots, p_n$, correspondingly;
Apply the vector $(\mathbf{v_1}, \ldots, \mathbf{v_n})$ to the machine *M* and obtain the output value **t** of the trust level *T*, i.e. simulate *M* over $(\mathbf{v_1}, \ldots, \mathbf{v_n})$.
If $(\mathbf{t} < K1)$ **then** $j := j + 1$;
If $((j / |TS|) > K2)$
then Return the verdict 'Fail'.
4. $i := i + 1$; and Go to **Step 2**.

The main idea of the proposed approach is schematically represented in Fig. 1.

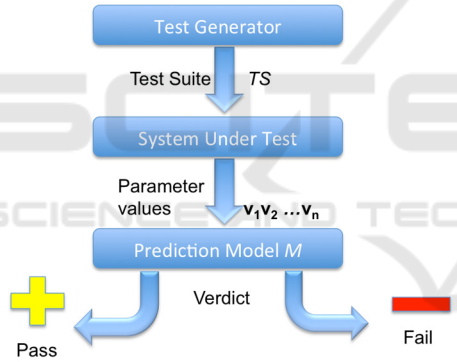

Figure 1: The idea behind the proposed approach.

For better illustration, consider the following inputs for Algorithm 1, taken from the example of RESTFul service (Section 3). The source code *SC* is the pseudo-code shown above. Let p_1 be a parameter that represents the response time, i.e., *$time*. Let p_2 be a parameter that represents the response code, i.e. *$responseCode*. As we are only interested in the HTTP codes 3xx and 2xx, we consider p_2 to be a Boolean variable, where 0 stands for 2xx and 3xx for 1, correspondingly. Assume also that as a prediction model we use a linear combination $M =]0.4 + 0.6p_1 + 2p_2[$ (result is round w.r.t the nearest integer). Let *K*1 = 2, and *K*2 = 0 and the test suite *TS* = {GET /login.php HTTP/1.1, POST /login.php HTTP/1.1, FAKE /login.php HTTP/1.1}. This test suite contains three input sequences of length one. Each sequence represents a typical HTTP request. For example, the first sequence GET /login.php HTTP/1.1 denotes that a method GET is sent, the URL is /login.php and the protocol of version 1.1 is being used.

Assume that the traces shown in Table 1 are obtained for the above source code.

Table 1: Example values from traces of the SaS_1 under test.

Test sequence	p_1	p_2	*Trust level*
GET /login.php HTTP/1.1	1	1	3
POST /login.php HTTP/1.1	0,001	1	2
FAKE /login.php HTTP/1.1	1	1	3

Algorithm 1 returns the verdict 'Pass' for values in Table 1. On the other hand, assume the traces of the SaS are different, as shown in Table 2.

Table 2: Example values from traces of the SaS_2 under test.

Test sequence	p_1	p_2	*Trust level*
GET /login.php HTTP/1.1	1	1	3
POST /login.php HTTP/1.1	0	1	2
FAKE /login.php HTTP/1.1	1	0	1

In the latter case, Algorithm 1 returns the verdict 'Fail'. Indeed, the SaS_2 is not trustworthy for the criteria listed above.

We mention that the effectiveness of the algorithm essentially relies of the fact how a test suite *TS* has been generated. Many test generation techniques focus on conformance testing, namely on assuring functional requirements of a SaS or its components. In this case, the test suite *TS* has another objective, i.e. an application of a test sequence should bring the SUT to a state or to a configuration (when internal variables are introduced as inputs of the machine *M*) where this SUT behaves untrustworthy. Therefore, the proposed approach can also be used for estimating the correlation between functional tests and their non-functional properties, namely 'trust estimating' properties. A given functional test suite *TS* can be very effective/powerful with respect specific fault domain, however it can rarely bring the SUT to an untrustworthy state. On the other hand, it is also possible that random tests that are known to have rather low fault coverage can probably propagate the critical values of the sensitive variables more often. Thus, one of interesting questions for future work

can be a test prioritization, when test sequences or test cases are distributed between several classes. In functional testing, these classes are usually represented by the number of faults or the number of mutants that can be killed by a given test case. In the case of active trust assessment, test cases can be assigned with the scores as the trust levels obtained from a SUT. Studying the dependencies between functional and non-functional score assignment is one of the directions of our future work.

5 CONCLUSIONS

In this paper, we have proposed an active testing based trust assessment approach. The approach can be applied to any entity of a telecommunication system; however, we preferred to draw our attention to the emerging concept of Systems as Services.

In order to decide which input sequences can be included into a test suite under derivation, we proposed to use a machine learning approach. In this case, the machine that represents the prediction engine is built based on the training set provided by the experts. Later on, the machine allows to choose the test sequences that can potentially cause the system under test to produce untrustworthy outputs. To the best of our knowledge, it is the first proposal for using active testing techniques for SaS trust assessment, and the proposed approach brings a lot of challenges for the future work. In particular, we would like to perform experiments with the various SaS for estimating its validity and effectiveness. Later on, we would like to consider the test prioritization problem when the test sequences are being classified according to their abilities of setting the system to untrustworthy states. Finally, the active assessment of trustworthiness of an entity might be the first step in a trust certification process. Investigation of the applicability of the approach for the SaS trust certification is another challenge.

The issues listed above form the nearest directions of the future work.

ACKNOWLEDGEMENTS

The work was partially supported by the Russian Science Foundation (RSF), project № 16-49-03012.

REFERENCES

Ardagna C.A., Asal R., Damiani E., Vu Q.H., 2015. From Security to Assurance in the Cloud: A Survey. In ACM Computing Surveys , 48(1), pp. 1-50.

Blum, A., Langley, P., 1997. Selection of Relevant IFeatures and Examples in Machine Learning. In Artificial Intelligence. V. 97, I. 1-2, pp. 245-271.

Kushik, N., Yevtushenko, N., Evtushenko, T., 2016. Novel machine learning technique for predicting teaching strategy effectiveness. In International Journal of Information Management, DOI: 10.1016/j.ijinfomgt.2016.02.006.

López, J., Maag, S., 2015. Towards a Generic Trust Management Framework Using a Machine-Learning-Based Trust Model. In IEEE Trustcom / BigDataSE / ISPA, Helsinki, pp. 1343-1348. doi: 10.1109/Trustcom.2015.528.

Lee, A.J., Winslett, M., Perano, K.J., 2009. TrustBuilder2: A Reconfigurable Framework for Trust Negotiation. In the IFIP International Conference on Trust Management. pp. 176-195.

Blaze, M., Feigenbaum, J., Lacy, J., 1996. Decentralized Trust Management. In the IEEE Symposium on Security and Privacy. pp. 164-173.

Jim, T., 2001. SD3: A Trust Management System with Certified Evaluation. In the IEEE Symposium on Security and Privacy. pp. 106-115.

Chen, I., Guo, J., 2014. Dynamic Hierarchical Trust Management of Mobile Groups and Its Application to Misbehaving Node Detection. In the IEEE International Conference on Advanced Information Networking and Applications. pp. 49-56.

López, J., Maag, S., Morales, G., 2016. Behavior evaluation for trust management based on formal distributed network monitoring. In World Wide Web V. 19, I. 1, pp. 21-39.

Pautasso, C., Zimmermann, O., Leymann, F., 2008. Restful web services vs. "big"' web services: making the right architectural decision. In the 17th international conference on World Wide Web. pp. 805-814.

Dabirsiaghi, A. 2016. Bypassing VBAAC with HTTP Verb Tampering: How to inadvertently allow hackers full access to your web application, Electronic resource:http://cdn2.hubspot.net/hub/315719/file-1344244110-pdf/download-files/Bypassing_VBAAC_with_HTTP_Verb_Tampering.pdf?t=1479325184680 (seen 01/12/2016).

Boser, B.E., Guyon, I.M., Vapnik, V.N., 1992. A training algorithm for optimal margin classifiers. In the Fifth Annual Workshop on Computational Learning Theory. pp.144–152.

Grandison, T., Sloman, M., 2003. Trust management tools for internet applications. In Trust Management, Springer First International Conference, iTrust, Heraklion, Crete, Greece. pp. 91–107.

López, J., 2015. Distributed on-line network monitoring for trust assessment. Thesis of the University of Paris-Saclay, France.