Managing Security Control Assumptions using Causal Traceability

Armstrong Nhlabatsi*, Yijun Yu†, Andrea Zisman†, Thein Tun†, Niamul Khan*, Arosha Bandara†, Khaled M. Khan* and Bashar Nuseibeh†‡

*Department of Computer Science and Engineering, KINDI Lab, Qatar University, Doha, Qatar
Email: {Armstrong.Nhlabatsi,Niamul.Khan,K.Khan}@qu.edu.qa

†Department of Computing and Communications, The Open University, Milton Keynes, UK
Email: {y.yu,andrea.zisman,t.t.tun,a.k.bandara,b.nuseibeh}@open.ac.uk

‡Lero, University of Limerick, Dublin, Ireland
Email: {bashar.nuseibeh}@lero.ie

***Abstract*—Security control specifications of software systems are designed to meet their security requirements. It is difficult to know both the value of assets and the malicious intention of attackers at design time, hence assumptions about the operational environment often reveal unexpected flaws. To diagnose the causes of violations in security requirements it is necessary to check these design-time assumptions. Otherwise, the system could be vulnerable to potential attacks. Addressing such vulnerabilities requires an explicit understanding of how the security control specifications were defined from the original security requirements. However, assumptions are rarely explicitly documented and monitored during system operation. This paper proposes a systematic approach to monitoring design-time assumptions explicitly as logs, by using traceability links from requirements to specifications. The work also helps identify which alternative specifications of security control can be used to satisfy a security requirement that has been violated based on the logs. The work is illustrated by an example of an electronic patient record system.**

KEYWORDS: *Traceability, Assumptions, Security*

I. Introduction

While developing requirements into specifications, software engineers have to make *assumptions* about the runtime operational environment. The gap between a system and its operational environment is bridged conceptually by explicit and implicit assumptions [1]. In reality, not all assumptions are documented explicitly at design-time, or even validated at runtime. *Traceability links* can be used to address the problem of explicitly documenting assumptions [2], and *monitoring* can be used to validate assumptions at runtime [3]. Since changes to the operational environment could render design-time assumptions invalid, it is necessary to provide support for both approaches, which is even more important for security requirements and their respective security control specifications.

The invalidation of an assumption may have undesirable consequence on the behaviour of the application, which often denies security requirements [4] [5], and may leave an application vulnerable to attacks. For example, consider the requirement of protecting the bank balance of a credit card from an unauthorised user, and the implementation of this requirement by a security control specification that requires personal identification number (PIN) authentication. Assume that only the owner of the card knows the pin. In this case, a disclosure of the PIN to another user who has access to the card could cause loss of money on the associated bank account.

Monitoring assumptions is useful, and necessary, to identify runtime alternatives for security control specifications that are more effective. Examples of alternative security control specifications are (i) limiting the balance of the credit card, or (ii) introducing two-factor authentication such as short messages with onetime codes to verify that the user also owns the registered phone.

Addressing these types of violations requires a good understanding of how security control specifications are defined from security requirements. More specifically, there is a need to analyse and document assumptions by asking questions such as *Why an assumption was made*? and *What are the implications of an assumption on the satisfaction of security requirements*?

In this paper, we propose an approach to identify alternative security control specifications of violated requirements in order to support adaptation of the system in the face of violations. More specifically, the approach monitors assumptions continuously to generate logs about the system and its environment [6]; and uses predefined traceability links to identify the security control specifications associated with the requirements. In the event that a security requirement is not satisfied, the approach identifies the associated security control specifications by traversing the traceability links, and uses the log files to analyse the assumptions that hold.

The approach combines three techniques, namely (i) *entailment relations* of requirements problems [7], (ii) *problem progression* [8] [9], and (iii) *causality relations* [10] [11]. The entailment relation technique is used as the basis for relating requirements and specifications through documentation of the assumptions associated with the security control specifications. The problem progression technique is used to help with the identification of the specifications which are related to the requirements. These relations are represented explicitly by traceability links based on causality. We call these links *Causal Traceability*. The causality relations between the logged events are used during the monitoring of assumptions. Compared to plain relationships between specifications and requirements, such traceability links are enriched with behavioural semantics of the application. The paper proposes an approach to managing assumptions using traceability links based on causality.

The remainder of the paper is structured as follows. Section II introduces an example to motivate the problem. Section III gives the background on problem progression, entailment and causality. Section IV describes the approach. Section V illustrates the approach through the example described in Section II. Section VI summarises related work and Section VII concludes the work.

II. Example: Electronic Patient Records System

Our approach is illustrated using the access control requirements of a cloud-based electronic patient record (EPR) system.

A. Design assumptions about the runtime operation

Bob is a doctor who wants to access the medical records of his patients stored on the EPR system. A policy of the hospital is that Bob should only access the medical records when he is on duty. Assuming that the EPR system has already authenticated Bob, whether he is on duty or not depends on the combination of three contextual attributes, namely: his location, the time of day, and the identity of the network he is using when accessing the medical records. Furthermore, these attributes are determined respectively by three sensors: GPS, Clock, and WiFi. The availability of these three sensors is indicated by the values of three Boolean contextual variables: *isWithinGPSRange, isWithinWorkingHours*, and *isCorrectSSID*, respectively. The combinations of these three variables are used to compute *ConfidenceLevel* to indicate the trust the system has that the doctor is on duty (see Table I). The values can be VERY LOW, LOW, MEDIUM, or HIGH (e.g., if the GPS sensor provides accurate coordinates and the doctor is connected to the hospital WiFi, but not during working hours (i.e. Row 7), the confidence level is HIGH).

ConfidenceLevel is used by Adam, the administrator of the system, to determine which access privileges of patient records are granted to Bob (see Table II). The privileges can either be *Read, Write, or Share* (e.g., if ConfidenceLevel is VERY LOW, no privilege is allowed).

TABLE I
CONTEXT VARIABLES AND CONFIDENCE LEVEL

Row	isWithin GPSRange	isCorrect SSID	isWorking Hours	Confidence Level
1	False	False	False	VERY LOW
2	False	False	True	LOW
3	False	True	False	LOW
4	False	True	True	MEDIUM
5	True	False	False	MEDIUM
6	True	False	True	HIGH
7	True	True	False	HIGH
8	True	True	True	HIGH

Based on the information in Tables I and II, Bob's access privileges depend on the sensors available on his device. For

TABLE II
CONFIDENCE LEVEL AND PRIVILEGES

	Confidence Level	canRead	canWrite	canShare
1	VERY LOW	No	No	No
2	LOW	Yes	No	No
3	MEDIUM	Yes	Yes	No
4	HIGH	Yes	Yes	Yes

example, Bob's iPad has a built-in GPS receiver while his laptop has none, which results in different privileges for him.

B. Runtime violation of access control policy requirement

Suppose Bob is denied the privilege to *share* a medical record when using his laptop but the he is granted the same privilege when using his iPad. After Bob complains to Adam about variation in his access privileges, Adam introduces a separate external GPS device that determines Bob's location when accessing a medical record from his laptop. The EPR system access control policies are then updated such that whenever Bob is accessing a medical record from his laptop, his location is determined by the external GPS device. Meanwhile, if he accesses the system from the iPad, his location is determined by the built-in GPS device of the iPad.

For the new access control policy specification to work, a key assumption is that Bob is always carrying the GPS device when accessing medical records using his laptop. However, this assumption may not always hold. For instance if Bob left his GPS device in the hospital while retrieving medical records outside the hospital, the EPR system may conclude that the confidence level of Bob being on duty is HIGH (see Row 6 of Table I) when it should have been LOW (see Row 2 of Table I). As a result, Bob is granted all the privileges as if he was physically present at the hospital simply because his external GPS device indicates to the access control that he was at the hospital. However, since he is accessing the record from outside the hospital premises he should be allowed only to *Read*.

This scenario is a violation of the hospital policies that Bob should only access full privileges when in the hospital, and results from the invalidation of the assumption that the doctor always carries the external GPS device. The violation represents a vulnerability.

The EPR system as designed has no control over this assumption and has no way of verifying its validity. It has to trust that Bob will always behave in a way that ensures that the assumption holds. The scenario also demonstrates that one of the critical requirements for an application designed to satisfy security requirements is the capability to *monitor* the validity of assumptions at runtime.

III. Background

Our approach for explicitly documenting assumptions is based on three key concepts, namely (i) entailment relations of requirements problems [7], (ii) causality in temporal logic [11], and (iii) problem progression transformations [10]. We use the entailment relation as the basis for relating requirements and specifications, causality to model the semantics of traceability links, and problem progression as a systematic technique for eliciting traceability links and assumptions.

After the elicitation, we show how each explicit assumption can be monitored during system operation using attributes of the context. Explicit assumptions that cannot be monitored in their current form are transformed until they are in a form that they can be monitored. When the explicit assumptions are monitored, the reasoning of the security requirements satisfaction may involve switching the security control specification to an alternative one, if it exists. If all alternatives are exhausted, the system operator has to record the incident and resort to the designer for further investigations on the assumptions. In the rest of this section we describe the three key components of our approach: entailment, causality, and problem progression.

A. The entailment semantics

Entailment is a relationship between three sets of descriptions, namely: the requirements R, the specifications S and the domain properties W.

The set of requirements R describes the properties of the software system as desired by its users, customers, and other stakeholders. Requirements are *optative* in that they describe how the world would be once the envisioned system is in place. In the EPR system, for example, a security requirement is to control the access to medical records stated as: *a doctor can read, write, or share the medical records of patients only when s/he is on duty*.

The set of domain properties W describes the behaviour of the contextual domains of the environment that interacts with the system-to-be. The attributes of a domain may have values that determine the behaviour of the adaptive application. Unlike requirements, domain properties are *indicative* in that whether these properties hold is independent of the behaviour of the system-to-be. In the EPR example, an indicative property is: *working hours at the hospital are from 8am to 5pm.*

The set of specifications S describes how the computer should behave in order to satisfy the domain properties described in R, given that the domain properties in W hold. The specification for the EPR system could be: *after a successful authentication, the doctor can read, write, or share a medical record only when s/he is accessing the record while on duty.*

In general, the problem-solution relationship between the three sets of descriptions explained above is given as the following:

$$S, W \vdash R$$

where $\vdash$ is the *entailment* operator.

In order to support the entailment, one needs to know the details about the specific behaviour of S and W.

B. The causality semantics

One way to further describe the three properties (S, W, and R) in the entailment semantics is in terms of events and fluents, borrowed from temporal logic such as the Event Calculus [12].

The cause and effect relationships between events and fluents in the domain descriptions are causality relations. A causality relation describes how the occurrence of one event e_1 leads to the occurrence of a second event e_2 as the effect of changing a fluent [11]. For example, when the time-of-the-day becomes 8am, the event `isWorkingHours` occurs. This is related by the causal relation:

$$ClockStrikes8am \hookrightarrow isWorkingHours$$

Note the effect of time in this causality relation. The event `ClockStrikes8am` and the change of fluent `isWorkingHours` do not happen simultaneously, but consecutively. If event `ClockStrikes8am` happens at some time t_0, then the fluent `isWorkingHours` becomes true at a later time t_1.

C. The problem progression

The definition of a specification is achieved through successive transformations of a requirement using problem progression [10] until the requirement is concrete enough to be implementable as a machine specification.

The premise is that software problems are deeply rooted in the problem world and that in order to solve such problems should be re-expressed through the properties of the context until solutions can be found.

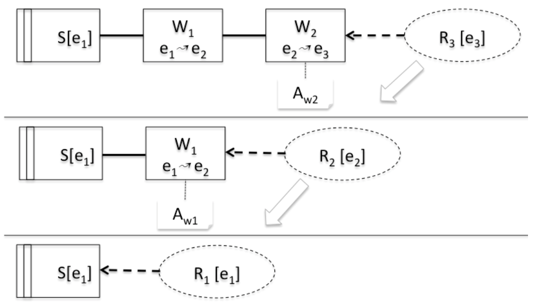


Fig. 1. Requirement Transformation through Problem Progression

Figure 1 illustrates the gradual transformation of a requirement through problem progression to a specification. $R_3[e_3]$ is the initial requirement expressed in terms of effect e_3 which is caused by event e_2 in W_2. The causal relation, $e_2 \hookrightarrow e_3$, in W_2 suggests that R_3 can be expressed using only e_2, instead of being expressed using e_3. Using this relation R_3 is transformed into R_2. The same process is repeated to transform R_2 into R_1. R_1 is expressed only in terms of phenomena of the machine and it is the most concrete form of the original requirement R_3. The causal relations $e_1 \hookrightarrow e_2$ and $e_2 \hookrightarrow e_3$ form the traceability link between the requirement and specification. Each causal relation $e_i \hookrightarrow e_{i+1}$ is guarded by an assumption A_{Wi} such that: $A_{Wi} \Rightarrow (e_i \hookrightarrow e_{i+1})$. This states that the causality can only happen if the assumption holds.

IV. Combining Causality with Traceability

In this section, we describe how to identify the assumptions of the contextual properties from the entailment semantics of traceability links between security requirements and security control specifications, during problem progression process.

We show how the causality semantics between runtime behaviours referenced by the contextual properties can be used in analysing and monitoring the identified assumptions.

A. Step 1. Documenting traceability links with entailment and causal semantics

We leverage the entailment relation to link security requirements R to security control specifications S given context W, assuming that S satisfy R. The specifications and the requirements are usually expressed in different terms, which makes it more difficult to relate the two (R and S).

In order to tease out the assumptions, one can start challenging every traceability link recorded during the problem progression. In Figure 2 the bold doubled-arrow line represents the traceability between R and S. Conceptually, a traceability link between a requirement and a security control specification has two parts. The first part is between the *requirements*

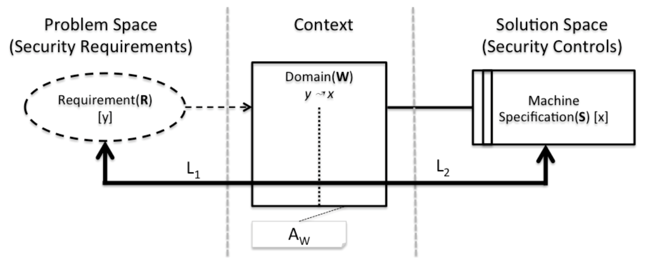


Fig. 2. Relating Requirements and Specifications through Domain Assumptions in the Context

and *contextual* domains, as indicated by (L_1) between R and W in Figure 2. According to the entailment relation, a requirement is defined as some property that must be exhibited by an application in order to solve some problem in the real world. For this reason, the requirement can be expressed as the conditions one would like to be true in terms of the attributes x of the context once the system-to-be is in place. The second part, indicated by (L_2), between W and S is the traceability between domains in the solution space, i.e. context and specification.

Relying on the assumptionsA_W about the behaviour of the context, denoted by y, the solution implements the behaviour x that satisfies the requirements. The reliance of the security control specification on assumptions of the context is used to establish a causality between the two (x and y). In *forward traceability* the requirement is the source artifact and the specification is the target; in *reverse traceability* the specification becomes the source and the requirement is the target.

B. Step 2. Monitoring assumptions as logs at runtime

Suppose that one has documented the assumptions explicitly as part of the design rationale through problem progression during the definition of alternatives to satisfy the security requirement. Typically the trade-offs decisions to prefer one specification to another is based on some assumptions. When a chosen specification has a problem in achieving the security requirement as complaints from users arise, it is necessary to adapt to the right alternative if possible. Therefore, every explicit assumption made during the trade-off needs to be validated.

After documenting the assumptions of traceability links in terms of referenced contextual attributes x and causally reliant behaviour y, the next step is to determine which of them needs to be monitored and based on what property at runtime. Logging every single property is impractical and creates overhead for large systems. To avoid such overhead, we make use of the semantics of traceability links guided by the security requirements.

In this approach, we assume that there are multiple security control specifications for a security requirement. Otherwise, there is a single point of failure. For the current context, the system executes the security control specification that satisfies the security requirements. According to the causality between behaviours of domains in the security control specifications, one needs to enact the monitors of the reliant behaviour to generate logs. These logs are used for the next analysis step to reason about adaptation.

C. Step 3. Adapting security control specifications

At runtime, when an incident report about violation of a security requirement arises, the adaptation mechanism searches the logs to identify events with timestamps around the time the incident happened. The traceability links are used to identify the security control specifications involved, especially to filter the log for the 'current' security control specification that was active during the incident.

After that, the assumptions and the domain attributes associated with the logs of the current specification and the alternative specifications are fetched. The logs are used to classify three situations:

- All the assumptions of the current specification s_0 are satisfied. In this case, an alarm will be created to indicate that there is a potential misunderstanding of the domain assumptions;
- Some assumptions of the current specification are indeed false, but all the assumptions of at least one of the security control specifications (e.g., s_1) hold. In this case, the system could have adapted to the security control specification. If the system failed to adapt to s_1, then there could be a fault of the adaptation mechanism;
- None of the alternative security control specifications at the time of the incident could satisfactorily meet their assumptions. In this case, the system should have sent an alarm to the system administrator. The absence of such alarms could indicate a fault of the adaptation mechanism.

As one can see, adaptation is not the automated panacea to the problem. All the above situations could involve either (a) taking the system offline to investigate the assumptions on the traceability links; or (b) tolerating the risks of future violation of security requirements if the likelihood of a similar incident is low and the impact is negligible.

V. Assumption Management: A Feasibility Study

In this section, we go through the three steps in the approach using the same running example from Section II.

A. Step 1. Documenting traceability links with entailment and causal semantics

The security requirement, R_s, of protecting the confidentiality of medical records can be stated as: *doctors can access medical records only while on duty*.

In satisfying this requirement two sub-problems, *Authentication* and *Location*, need to be solved. Solving these two sub-problems and composing their solutions essentially solves the *confidentiality* problem. Figure 3 shows an initial security control specification of the EPR system resulted from the progression of the problem.

```
IF (correctCredentials() & isWithinGPSRange()) THEN
    AllowRead(), AllowWrite(), AllowShare()
ELSE
    DenyAccess()
```

Fig. 3. An initial security control specification

Next, every assumption recorded during the problem progression should be challenged. The *Authentication* sub-problem is about how to determine that the doctor is who s/he claims to be? Initially, we have chosen to use *authentication* based on credentials (user name and password). In the process of problem progression, one can make a number of assumptions to justify the choice of solution for this sub-problem. Figure 4 shows a transformation of the authentication requirement R_{auth} into the specification S_{auth}.

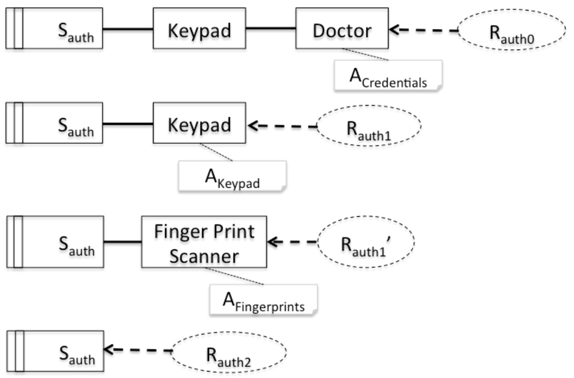


Fig. 4. Authentication problem progression with associated assumptions at each step

Starting with the initial authentication requirement R_{auth0} which says *authenticating a doctor*, the satisfaction of this requirement assumes $A_{Credentials}$. $A_{Credentials}$ is the assumption that the doctor knows the correct credentials. This assumption is based on the following causality:

$$A_{Credentials} \Rightarrow (hasCorrectCredentials() \hookrightarrow isDoctor())$$

$A_{Credentials}$ helps transform R_{auth0} into R_{auth1}: *if credentials entered on the keypad are correct then the subject entering them is the legitimate owner of the credentials.* The assumption placed on the keypad, A_{Keypad}, is based on the following causality:

$$A_{Keypad} \Rightarrow$$
$$(isCredentialsOwner() \hookrightarrow hasCorrectCredentials())$$

The weakness of this assumption is that even if credentials received by the system are correct, it is difficult to tell whether they were entered by the legitimate owner or by an attacker who has stolen them. Therefore, one cannot rely on the credentials entered on the keypad as the single way to identify the doctor. In addressing this issue one may consider a fingerprint scanner alternative, which is associated with its own assumptions. This transforms A_{Keypad} to $A_{Fingerprints}$: (*every individual has unique fingerprints*). With this transformation, the requirement R_{auth1} becomes R'_{auth1}, which says that *if finger prints read by the scanner match that of the doctor, then this is the doctor.* This requirement transformation is based on the following causality:

$$A_{Fingerprints} \Rightarrow$$
$$(matchingFingerPrints() \hookrightarrow isCredentialsOwner())$$

The problem progression from $R_{auth1'}$ ends with the requirement R_{auth2}, which states that *if the fingerprints read by the scanner match that of the doctor then authentication is successful.* Thus, the authentication specification S_{auth} becomes:

$$IF(matchingFingerPrints() == true)THEN$$
$$AunthenticationSuccessful$$

Similarly, solving the *Location* sub-problem is about answering the following question: *What does 'on duty' mean in the security requirement?* One may explain that the doctor is on duty if s/he is within certain GPS coordinates, using the hospital WiFi for connection, and the time of day is between 8am and 5pm. Based on this, `isOnDuty` $(x1)$ is a property referred by the requirements, and `GPS`, `WiFi`, and `Time-of-Day` are attributes from the context $(x2)$. For example, one of the assumptions on `GPS` made in the progression of this sub-problem is that the doctor is always carrying the GPS device when requesting access and hence his/her location may be determined from the coordinates of the GPS device.

The documentation of the causality relations and assumptions is performed as part of the process of refining a requirement into a specification - a task that the analyst performs through problem progression. Given that traceability is based on causal relation in the context, any change in either the requirement or specification has minimal impact on the traceability. This is because both the requirement and specification are based on properties of the context which scopes the extent to which they can change.

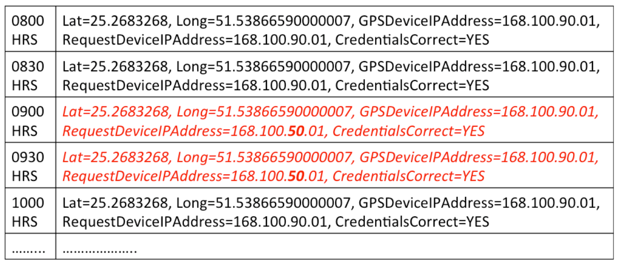
B. Step 2. Monitoring assumptions as logs at runtime

Assumptions need to be monitored at runtime. To facilitate the analysis later on, one should log contextual data at runtime.

For example, $A_{WithDoctor}$ is the assumption that the doctor always carries the GPS device when making a request for accessing medical records. Hence the coordinates given by the GPS device are assumed to be the location of the doctor. Although it is possible to log the coordinates of the GPS device into the system, it is not always sufficient to check the validity of this assumption. In fact, having a GPS device reporting the office location is not sufficient to tell whether it is with the doctor or not. The assumption can be challenged by asking what if the doctor is not carrying the GPS device.

An alternative security control specification is designed to consider the IP address of the subnet of doctor's WiFi device and the known subnet range assigned to the hospital. If the two addresses are in the same subnet, according to the assumption, it is the doctor who is trying to access. Therefore, the IP address of the GPS device and device requesting access must be logged such that by examining the two IP addresses one may monitor whether the doctor is carrying the GPS device when requesting the access.

Figure 5 shows some of the data logs collected at run-time. Each entry has an associated time-stamp, and the recorded values for a list of monitored variables. These values will be used to validate the $A_{WithDoctor}$ assumption.



0800 HRS	Lat=25.2683268, Long=51.53866590000007, GPSDeviceIPAddress=168.100.90.01, RequestDeviceIPAddress=168.100.90.01, CredentialsCorrect=YES
0830 HRS	Lat=25.2683268, Long=51.53866590000007, GPSDeviceIPAddress=168.100.90.01, RequestDeviceIPAddress=168.100.90.01, CredentialsCorrect=YES
0900 HRS	*Lat=25.2683268, Long=51.53866590000007, GPSDeviceIPAddress=168.100.90.01, RequestDeviceIPAddress=168.100.**50**.01, CredentialsCorrect=YES*
0930 HRS	*Lat=25.2683268, Long=51.53866590000007, GPSDeviceIPAddress=168.100.90.01, RequestDeviceIPAddress=168.100.**50**.01, CredentialsCorrect=YES*
1000 HRS	Lat=25.2683268, Long=51.53866590000007, GPSDeviceIPAddress=168.100.90.01, RequestDeviceIPAddress=168.100.90.01, CredentialsCorrect=YES
………	…………………

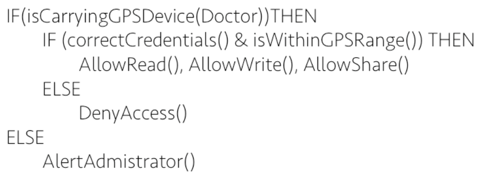
Fig. 5. Concrete Contextual Data Logged at Run-time

Next we explain how such logs can be used and what to do even when an adaptation is insufficient.

C. Step 3. Adapting security control specifications

When a monitored assumption becomes invalid at runtime according to the logs, the system administrator needs to diagnose the specific point in the definition of a specification from a requirement where the assumption was made. For example, the security requirement was reported to be violated by Bob at 0931 HRS. By analysing the logs between 0900 HRS and 0930 HRS, it shows that $A_{WithDoctor}$ was invalid but access is granted.

The monitoring mechanism enacted earlier could have identified this situation and reported it to the administrator. However, neither the administrator was alarmed nor the alternatives of the security control specifications were invoked, indicating that there is a design flaw in the system. Narrowing the scope from this log would help the administrator diagnose the flaw. As an alternative, the administrator may fix this by adding the GPS and access device's IP addresses in the security control specification as additional contextual attributes for determining the doctor's location.



```
IF(isCarryingGPSDevice(Doctor))THEN
    IF (correctCredentials() & isWithinGPSRange()) THEN
        AllowRead(), AllowWrite(), AllowShare()
    ELSE
        DenyAccess()
ELSE
    AlertAdmistrator()
```

Fig. 6. Revised Specification with Assumption Monitoring

Figure 6 shows the revised access control specification with monitored assumptions used as a guard condition. If the condition *isWithGPSDevice(Doctor)* is false, then access will not be granted. By checking whether the two IP addresses are in the same subnet, the revised specification solves the design flaw. This additional guard condition strengthens satisfaction of the security requirement and reduces the risk of vulnerability.

VI. Related Work

Traceability has been used as a technique for supporting software maintenance activities [13], [14], [15], [16], [17] by linking artifacts. The links are often based on common keywords between the artifacts [15], [18]. However, while effective in maintaining explicit relationships between artifacts, these links do not contain adequate semantic information about the relationship between requirements (problems) and specifications (solutions) using satisfaction arguments [19]. Using rule-based representation, the proposed links can also be generalised in the same style as the requirement-to-object-model rules [20] or as bidirectional transformation rules [21].

Attempts have been made to establish semantically rich traceability links [22], [23], [24], [25]. However, such links are not sufficient for managing runtime assumptions in software development. The traceability links proposed in this paper are based on domain-specific and intrinsic information, relating security requirements to security controls, through causality [11], [9]. This representation is motivated by the observation that security requirements are stated in terms of the conditions that need to hold to protect assets from harm, and security controls are expressed in terms of the actions that need to be performed in order to satisfy the conditions stated by security requirements [7].

In earlier work we proposed to use causal traceability for explaining the behaviour of adaptive systems [26]. We are not the first to address the problem of managing assumptions. Ali et. al. [4] outlined an approach to monitor, at runtime, the assumptions in a requirements model, and to evolve the model to reflect the validity level of such assumptions. Although our approach is similar, we emphasise that understanding traceability between requirements and specifications is a key feature in monitoring and transforming assumptions.

Moreover, some approaches have been proposed in which traceability links are used to support software agile development [27], and even in medical applications [28]. However, to the best of our knowledge, the use of traceability links to support security of applications is novel.

VII. Conclusion and Future work

Assumptions made about the operating environment may become invalid during system operation resulting in vulnerabilities. We have proposed an approach to monitoring assumptions for security controls. In the approach we have combined the entailment and causality semantics to model traceability between requirements and specifications so that problem progression techniques can be used for systematically eliciting assumptions alongside traceability links. We have illustrated how our approach can be used for diagnosing vulnerabilities through an access control example of an electronic patient records system.

In ongoing work we will explore the possibility of using cause-to-effect and effect-to-cause rules [10] to enrich forward and reverse traceability links for forward and reverse engineering. Specifications based on weaker assumptions are currently discarded in our approach. We plan to retain and invoke them at run-time in the event that the default specification fails to satisfy the requirement. We also plan to provide an automatic way to identify traceability links, and to evaluate the approach in real-life case studies.

Acknowledgement

This work was made possible by the support of a grant (NPRP 05-079-1-018) from the Qatar National Research Fund (QNRF). The statements made herein are solely the responsibility of the authors.

References

[1] M. Lehman and J. Ramil, "Software evolution in the age of component-based software engineering," *Software, IEE Proceedings -*, vol. 147, no. 6, pp. 249–255, 2000.

[2] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, Jan. 2001. [Online]. Available: http://dx.doi.org/10.1109/32.895989

[3] Y. Wang, S. A. McIlraith, Y. Yu, and J. Mylopoulos, "Monitoring and diagnosing software requirements," *Autom. Softw. Eng.*, vol. 16, no. 1, pp. 3–35, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10515-008-0042-8

[4] R. Ali, F. Dalpiaz, P. Giorgini, and V. E. S. Souza, "Requirements Evolution: From Assumptions to Reality," in *Enterprise, Business-Process and Information Systems Modeling*, ser. Lecture Notes in Business Information Processing, T. Halpin, S. Nurcan, J. Krogstie, P. Soffer, E. Proper, R. Schmidt, and I. Bider, Eds. Springer Berlin Heidelberg, Jan. 2011, no. 81, pp. 372–382. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-21759-3-27

[5] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE Transactions on Software Engineering*, vol. 34, no. 1, pp. 133–153, Jan. 2008.

[6] M. Salifu, Y. Yu, A. K. Bandara, and B. Nuseibeh, "Analysing monitoring and switching problems for adaptive systems," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2829–2839, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.07.062

[7] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 1, pp. 1–30, Jan. 1997. [Online]. Available: http://doi.acm.org/10.1145/237432.237434

[8] M. A. Jackson, *Problem frames and methods: structuring and analyzing software development problems*. Harlow: Addison-Wesley, 2000.

[9] L. Rapanotti, J. Hall, and Z. Li, "Deriving specifications from requirements through problem reduction," *Software, IEE Proceedings -*, vol. 153, no. 5, pp. 183–198, Oct. 2006.

[10] Z. Li, J. G. Hall, and L. Rapanotti, "On the Systematic Transformation of Requirements to Specifications," *Requir. Eng.*, vol. 19, no. 4, pp. 397–419, Nov. 2014. [Online]. Available: http://dx.doi.org/10.1007/s00766-013-0173-8

[11] R. Scherl and G. Shafer, "A logic of action, causality, and the temporal relations of events," in *Fifth International Workshop on Temporal Representation and Reasoning, 1998. Proceedings*, May 1998, pp. 89–96.

[12] E. T. Mueller, "Automating commonsense reasoning using the event calculus," *Commun. ACM*, vol. 52, no. 1, pp. 113–117, Jan. 2009. [Online]. Available: http://doi.acm.org/10.1145/1435417.1435443

[13] M. Hirzalla, A. Zisman, and J. Cleland-Huang, "Using Traceability to Support SOA Impact Analysis," in *2011 IEEE World Congress on Services (SERVICES)*, Jul. 2011, pp. 145–152.

[14] M. Mirakhorli and J. Cleland-Huang, "Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 123–132.

[15] G. Bavota, A. De Lucia, R. Oliveto, and G. Tortora, "Enhancing Software Artefact Traceability Recovery Processes with Link Count Information," *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 163–182, Feb. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2013.08.004

[16] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, 2014, pp. 55–69. [Online]. Available: http://doi.acm.org/10.1145/2593882.2593891

[17] J. Cleland-Huang, O. Gotel, and A. Zisman, Eds., *Software and Systems Traceability.* Springer, 2012. [Online]. Available: http://dblp.uni-trier.de/db/books/daglib/0028967.html

[18] H. Asuncion, A. Asuncion, and R. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, May 2010, pp. 95–104.

[19] E. Kang and D. Jackson, "Dependability Arguments with Trusted Bases," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, Sep. 2010, pp. 262–271.

[20] G. Spanoudakis, A. Zisman, E. Prez-Miana, and P. Krause, "Rule-based generation of requirements traceability relations," *Journal of Systems and Software*, vol. 72, no. 2, pp. 105–127, Jul. 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121203002425

[21] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux, "Maintaining invariant traceability through bidirectional transformations," in *2012 34th International Conference on Software Engineering (ICSE)*, Jun. 2012, pp. 540–550.

[22] H. Schwarz, J. Ebert, J. Lemcke, T. Rahmani, and S. Zivkovic, "Using Expressive Traceability Relationships for Ensuring Consistent Process Model Refinement," in *2010 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Mar. 2010, pp. 183–192.

[23] A. Marcus and J. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *25th International Conference on Software Engineering, 2003. Proceedings*, May 2003, pp. 125–135.

[24] W. Jirapanthong and A. Zisman, "Xtraque: traceability for product line systems." *Software and System Modeling*, vol. 8, no. 1, pp. 117–144, 2009.

[25] L. Lamb, W. Jirapanthong, and A. Zisman, "Towards a formalization for traceability relation," in *7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE11)*, Hawaii, 2011.

[26] A. Nhlabatsi, T. Tun, N. Khan, Y. Yu, A. K. Bandara, K. M. Khan, and B. Nuseibeh, "Why cant I do that?: Tracing Adaptive Security Decisions," *EAI Endorsed Transactions on Self-Adaptive Systems*, vol. 1, no. 1, p. e2, Jan. 2015. [Online]. Available: http://eudl.eu/doi/10.4108/sas.1.1.e2

[27] J. Cleland-Huang, "Traceability in agile projects," in *Software and Systems Traceability.*, 2012, pp. 265–275.

[28] F. McCaffery, V. Casey, M. S. Sivakumar, G. Coleman, P. Donnelly, and J. Burton, "Medical device software traceability." in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 321–339. [Online]. Available: http://dblp.uni-trier.de/db/books/daglib/0028967.html