Evaluating the degree of security of a system built using security patterns

Eduardo B. Fernandez
Dept. of CS and EE
Florida Atlantic University
USA
fernande@fau.edu

Nobukazu Yoshioka
GRACE Center
Nat.Inst. of Informatics
Japan
nobukazu@nii.ac.jp

Hironori Washizaki
Dept. of CS and Eng.
Waseda University
Japan
washizaki@waseda.jp

ABSTRACT

A variety of methodologies to build secure systems have been proposed. However, most of them do not say much about how to evaluate the degree of security of their products. In fact, we have no generally-accepted ways to measure if the product of some methodology has reached some degree of security. However, if the system has been built with a methodology that uses patterns as artifacts, we believe that a simple evaluation is possible. We propose a metric for the security of systems that have been built using security patterns: We perform threat enumeration, we check if the patterns in the product have stopped the threats, and calculate the coverage of these threats by the patterns. We indicate how to take advantage of the Twin Peaks approach to arrive to a refined measure of security. In early work, we have proposed a secure systems development methodology that uses security patterns and we use it as example.

CCS CONCEPTS

D.2.11 **[Software Engineering]** Software Architectures–Patterns; D.5.1 D.4.6 **[Security and Protection]** Secure Software

General Terms: Design

KEYWORDS

Systems security, security evaluation, security patterns, software architecture, software security

ACM Reference format:

E.B.Fernandez, N. Yoshioka, and H. Washizaki, 2018. In Proceedings of Int. Conf. on Availability, Reliability, and Security, Hamburg, Germany, September 2018, 8 pages. DOI: 10.1145/3230833.3232821

1 INTRODUCTION

Building a secure system requires the application of a systematic methodology. A security methodology includes a security process and a conceptual security framework consisting of security artefacts such as security patterns [42,45]. A variety of methodologies to build secure systems have been proposed [43]. However, most of them do not say much about how to evaluate the degree of security reached by the final systems they produce. Work on software architecture security doesn't say much either, most of their methodologies do not consider security and of those which do none of them discusses security evaluation in detail [2, 33]. In practice, the truth is that if we build a system and we add security mechanisms based on some methodology or using ad hoc ways, we have no generally-accepted way to measure if the system has reached some degree of security. Even the definition of security is not completely clear; as a result there are few accepted metrics for security, and they are not easy to apply. This makes it difficult to compare secure development approaches and if one adopts one of these approaches, it is hard to evaluate its products or to improve their security level.

By security we mean the ability of the system to protect the assets in its applications against attacks from external and internal attackers. Security usually implies the provision of confidentiality, integrity, availability, and accountability. Some work tries to prove that systems have these properties, usually applying formal methods. We consider looking at threats as a more practical approach and in this work we take this orientation. We are not interested either in code-based measures, which do not really measure the whole system security. Security is a difficult problem, especially for complex applications, which have many avenues for attacks that include all the architectural levels of the system and where interactions between units may hide threats. To compound the problem developers are not, in general, acquainted with security methods; this is true also for most software architects.

In this work we consider systems designed using *patterns*. A pattern describes a solution to a recurrent software or systems problem in a given context, and when the problems are security problems *security patterns* provide solutions. Security patterns provide a way for guiding system designers who are not experts on security to build secure systems [10]. Good security design requires the application of a set of principles [35], and we see the use of patterns as a convenient way to implicitly apply security principles even by people having little experience and/or little security knowledge. A *reference architecture* (RA) aggregates several patterns related to a given domain of knowledge and can simplify the work of the designers. Pattern and RA catalogs are useful tools to build complex systems and define a model-based approach to their design. There have been some attempts to build catalogs of security patterns and our book [10], is among this group. We have proposed a secure systems development methodology that uses security patterns and we have extended it recently [7, 10, 45]. This methodology applies security throughout the whole lifecycle and considers all architectural levels. As part of this work we have produced a variety of security patterns [10].

As a way to get a handle on the problem of security evaluation [40] has suggested designing systems that exhibit measurable properties. We are following this idea by taking advantage of patterns and reference architectures. We propose here a way to perform this evaluation. We first enumerate threats and verify if they all have been stopped or mitigated by some security mechanism realizing a security pattern. The percentage of threat coverage by patterns is our security measure. We indicate how this approach could be refined by using the Twin peaks approach [26]. We enumerate threats by considering all the actions in each use case and analyze how they could be subverted by an attacker to reach his goals [10], but other approaches for this enumeration would also be acceptable.

We make clear that we are not proposing a new methodology to build secure systems but a way to evaluate the security level of specific methodologies, although we use our own methodology as example.

Our contributions include:

- A new metric for evaluating system security based on threat enumeration and verifying if these threats are or not controlled in a specific software architecture. This metric applies to systems built using security patterns but can be extended to methodologies which use other artefacts.
- An evaluation of our metrics and a comparison with other metrics
- A way to refine this metric using the Twin Peaks approach.

Section 2 presents some background, while Section 3 discusses security metrics. Section 4 presents our security measure. We consider the enumeration of threats in Section 5, Section 6 shows how to use our measure to compare systems. Section 7 describes the refinement of our measure using the Twin Peaks approach in the construction of secure systems. We end with conclusions in Section 8.

2. PATTERNS AND REFERENCE ARCHITECTURES

A pattern is a solution to a recurrent problem in a given context [4]. Patterns are described using a template composed of a set of structured sections. A problem section describes a general problem and forces that constrain and define guidelines for the solution, e.g., "some actions must be transparent to the users". The solution is usually expressed using UML class, sequence, state, and activity diagrams (although we usually don't need all these models). A set of consequences indicate what is the effect of the pattern and how well the forces were satisfied by the solution, including advantages and disadvantages of using the pattern. An implementation section provides hints on how to use the pattern in an application, indicating what steps are needed and possible realizations. A section on "Known uses" lists real systems where this solution has been used previously, i.e., a pattern is an abstraction of good practices. A section on related patterns indicates other patterns that complement the pattern or that provide alternative solutions. A pattern embodies the knowledge and experience of software developers and can be reused in new applications; carefully-designed patterns implicitly apply good design principles. Patterns are also good for communication between designers and to evaluate and reengineer existing systems. While initially developed for software, patterns can describe hardware, physical entities, and combinations of these, as well as non-technical processes such as teaching a course or organizing a conference. Pattern solutions are suggestions, not plug-ins or software components. In particular, security patterns can suggest solutions to designers who don't have much security experience. *Abstract Security patterns* (ASPs) describe conceptual (no implementation aspects) security mechanisms that realize one or more security policies able to handle a threat or comply with a security-related regulation or institutional policy [11]. ASPs are used in the early lifecycle stages. Because of their abstraction properties, security patterns provide a way to apply a holistic approach to system security and they are useful to handle large and complex systems in a comprehensive and unified way.

In order to describe attacks we defined another type of pattern: A *misuse pattern* describes, from the point of view of an attacker, a generic way of performing a misuse (such as a violation of confidentiality or integrity) that takes advantage of the specific architecture and vulnerabilities of some environment [8]. Until now there is only one catalog of misuse patterns [44], although the concept has been studied in some detail [8]. Misuse patterns define the environment where the attack is performed, countermeasures to stop it, and provide forensic information in order to trace the attack once it happens. For example, a security defense misconfiguration is a vulnerability, taking advantage of this vulnerability is a threat (potential attack) which can lead to reading unauthorized information (a misuse). In particular, misuse patterns are useful for developers because once they determine that a possible attack can happen in the environment, the pattern will indicate what security mechanisms are needed as countermeasures. Also, misuse patterns can be useful for forensic examiners to find evidence information after the attack has been performed. Finally, they can be used to evaluate if an existing system can handle specific threats. Note that a misuse pattern describes a complete attack, e.g., stealing information from a database, not just specific steps used to perform the attack, such as SQL injection or buffer overflow (both can be used in the same threat or individually in many threats.

The use of Reference Architectures (RAs) can simplify the application of patterns and thus the construction and evaluation of secure systems. A Reference Architecture (RA) is a generic software architecture, based on one or more domains, with no implementation aspects [41]. It is reusable, extendable, and configurable; that is, it is a kind of pattern for whole architectures and it can be instantiated into a specific software architecture by adding platform aspects [1]. We can build RAs using patterns. After adding security patterns to neutralize identified threats in an RA we have a *Security Reference Architecture* (SRA), and we recently produced a SRA for clouds [12] .

3. RELATED WORK ON SECURITY METRICS

[24] is a survey of security metrics describing the state of the art up to 2010. They emphasize standards and indicate that security metrics are not very developed and need more work.

[40] discusses measuring security in general. They classify measure into computational-complexity metrics, economical/biological metrics, and empirical metrics. They also propose some ideas for future research, one of which has provided our motivation.

Another direction is represented by the concept of attack surface [23]. This measure counts the ways through which an adversary can penetrate the system. It does not consider the semantics of the application, only its structural properties, which makes harder the application of risk analysis to select defenses. [15] extended this concept to include security, privacy, and dependability; they also evaluated the effectiveness of protection methods. This measure does not really measure the security level of a system (defenses in place) but how easy is to attack it.

There exist statistical methods to evaluate the security of some types of systems; they have value to predict trends or to find if some type of system can be made secure but they do not provide any guidance that can be applied to a specific system being built or evaluated. For example, a project on software metrics is going on at SEI [37]. They use Bayesian networks and other statistical measures of system security, but it is not clear how their methods can be applied to specific designs. Another approach of this type develops a quantitative theory of operational security, similar to that which now exists for reliability [22].

[16] associates security metrics to patterns and they aggregate the measures of a system to evaluate its security. However, their metrics are statistical measures based on system events, which implies the need to implement a specific system and measure its behavior. This measure cannot be used to evaluate a methodology.

There are several approaches to measure vulnerabilities in a specific system, e.g., [6] and [28]. Again, they only can measure a specific system implementation, they are not a measure of the quality of the design.

[32] uses arguments to define a security metric. The metric relies on its degree of confidence in a security argument. Confidence is based on appropriateness, sufficiency, and trustworthiness.

The Common Criteria (CC) approach evaluates specific products according to protection profiles that define their expected requirements [5]. The Common Criteria provides assurance that the process of specification, implementation and evaluation of a product has been conducted in a rigorous, standard, and repeatable manner at a level that is commensurate with the target environment for use; however, it does not try to measure the degree of security of the evaluated product or the security of all the products of some methodology.

As indicated earlier, security patterns encapsulate solutions that can stop or mitigate specific threats and their consequent misuses. This means that each pattern added to the system may contribute to the total security of the system. However, adding security patterns that do not stop threats would lead to systems which are very slow, expensive, and hard to maintain. It is clear that security patterns need a guiding methodology to be effective and several pattern-based methodologies have been proposed [42], e.g.:

A general methodology for developing security-critical software has been proposed in [20]. It makes use of an extension of the Unified Modeling Language (UML), UMLSec, to include security-relevant information. The approach is supported by extensive automated tool-support for performing a security analysis of the UMLSec models against security requirements [21]. The analysis is based on model- checking specific portions of a system.

Mouratidis and his group use a special methodology, Secure Tropos, to model security. Their work started modeling requirements but they have also considered other stages; for example how to test security along the lifecycle. Instead of UML they use special diagrams and they use patterns described in their style. Their proofs of security only consider high-level aspects and not design aspects.

As indicated, our survey [43] showed that none of these methodologies provides a detailed way to evaluate the degree of security reached by the complete system. None of these works takes advantage either of the way a system has been built to simplify the evaluation of its security. Our point here is that the use of security patterns makes this evaluation more precise and convenient.

Some secure systems methodologies use security attributes as objectives and they use formal methods to prove that a system has a given degree of confidentiality or integrity. However, they often require unrealistic assumptions and do not consider implementation aspects. Their standard definition of security is the provision of properties such as confidentiality, integrity, availability, and accountability. However, these attributes are not directly measurable and proving that a system exhibits these properties is a very difficult problem for large and complex systems. We believe that a practical measure of security must be based on considering the threats to the system. In this case, instead of looking for abstract properties we need to find ways to stop the threats we have identified.

Savola indicates that although it may not be possible to measure security as a whole, it is possible to measure factors leading to it [36]. We apply his ideas to evaluate our measure in Section 4.

The Security Twin Peaks model was proposed in [17]. However, in each iteration they map to components, not patterns. They work entirely with architectural artifacts, instead of using conceptual models as we do. Okubo [27] proposed how to use the Twin Peaks model to secure architectures. However, he does not consider conceptual security models and does not use security patterns; we define first a stage where attacker goals and a conceptual model are created before defining the architecture and in each iteration we can use threat patterns to see how to stop these threats (see Section 7).

4. A SECURITY METRIC

A threat Ti uses a sequence of attack steps Tik; that is: Ti → Ti1, Ti2, …, Tij. In order to stop Ti we only need to stop one of the Tij, which can be done by applying security patterns that stop that specific attack step. However, a specific design may include patterns to stop a set of steps; that is, an application of the defense-in-depth principle. If TN is the number of stopped threats and T is the total number of identified threats, we can define a Security Coverage, SC, as TN/T, which is our measure of security.

After we build a system using a secure system development methodology, the process to evaluate its security is:

- Enumerate threats based on attacker's goals (part of the development methodology).
- Rank threats according to their impact and probability of occurrence. Note that threats are attacker goals.
- Determine SC as above, considering all threats or only the most important ones.

In the next section we discuss threat enumeration. It is clear that acceptable values for SC depend on the type of application; e.g. a financial application will require a much higher SD than a gaming application. We can refine this value by using a Twin Peaks approach to determine threats that appear when iterating through the lower levels of the architecture (Section 7).

Savola presents criteria to evaluate the quality of a security metric, we can apply them to our metric. He found four basic quality criteria for these metrics:

- *Correctness*—threats are a basic quality criteria for security metrics and mitigating them will improve the security of the system.
- *Measurability*—we can enumerate a reasonably complete list of threats and count the corresponding security patterns that can cover them.
- *Meaningfulness*—applying defenses against the identified threats will make the system more secure. Two sub-aspects of meaningfulness are:
  - *Comparability*—It is now possible to compare two systems based on their threat coverage (see Section 6).
  - *Progression.* Adding security patterns we can improve security. Each new security pattern may improve security.
- *Usability*—the method for threat enumeration and the use of patterns are rather simple approaches that do not require designers to be security experts.

Since we are looking at models we cannot say much about code vulnerabilities. However, we claim that with wise use of compartmentalization we can build systems which are essentially secure, where an attacker can get some data from a compromised section but not reach the more valuable information. We cannot perform actual measurements to evaluate a design either, we can only indicate that specific threats cannot happen; that is, meaningfulness is high.

These measures can be applied both to systems under development or to systems already built. In the second case we need to identify in the system the patterns that have been applied in its construction, either explicitly or implicitly.

5. ENUMERATING THREATS

Figure 1 shows a metamodel relating threats to vulnerabilities and related concepts. An asset is a valuable item in the system, e.g., the list of customers together with the operatioms to access it. Assets may be software or physical units. Assets may have vulnerabilities (weak points) that can be exploited by attacks, which in turn realize threats. Attack patterns correspond to the steps required to perform a specific security attack (exploit) in a generic fashion; they describe specific steps leading to a misuse; e.g., using a stolen credential to have access to a DBMS where we can steal information by using SQL injection. Misuse patterns describe a complete attack in a system, related to its specific architectural components, and leading to a specific misuse of information. *Threat patterns* are abstract patterns, not related to a specific system architecture [44], and may include invoking unauthorized operations, message replay, and injection. Misuse patterns address a generic and important (high impact) type of attack, not an instance of an attack in a specific system.

In previous work we introduced an approach for threat enumeration [3, 10]. This process is performed during the requirements and the design stages of the software development cycle and it analyzes each activity in the activity diagram of a use case to see how it could be subverted by an attacker to reach her goals. The process requires to consider the activities in the use cases of the complete system. We show an example in the next paragraph.

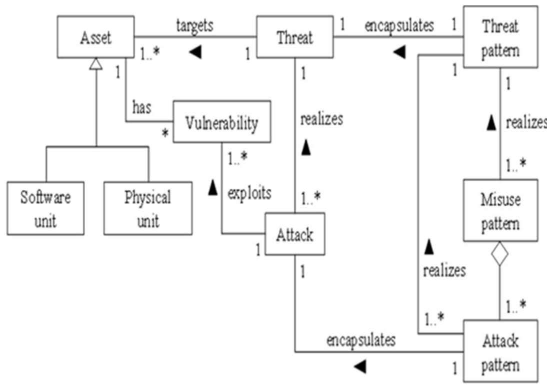

Figure 1. A metamodel for security concepts

Fig. 2 shows some of the threats for the use case “Open an account in a financial institution”. This analysis results in a set of threats and since the use cases are all the ways to interact with a system we can enumerate threats systematically (although we cannot prove that we considered all of them). We then consider which policies can stop or mitigate these threats and we realize the policies with patterns; in fact, we have incorporated this approach as part of a systematic methodology to build secure systems [45]. This process requires developers to conjecture possible attacks to different assets or parts of a system, to assess their impact and likelihood, and to determine how they could potentially be stopped or mitigated. We do not count vulnerabilities, only the attacker goals are of interest to us.

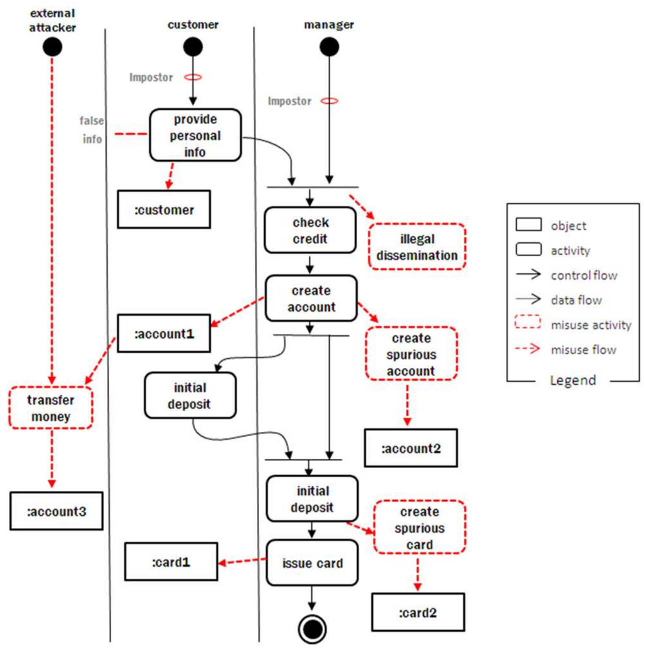

Figure 2. Activity Diagram for Use Case "Open Account" showing some threats (attacker goals)

Because of the large number of threats that may appear as attacker goals, many of which may not be significant, it is important to reduce their number by performing a risk analysis. In this process threats must be ranked by impact and filtered or given weights before applying them in the calculation of SC. Note that the OWASP or CVSS scores are not useful here because they only consider code aspects. This means that designers must estimate the impact of threats.

In addition to the method mentioned above, we can think of two ways to generate threats:

1. From known incidents and threat lists. There are vulnerability databases, e.g., CVE [6], NIST CVSS [25], and lists of top threats, e.g., OWASP [28], that provide descriptions of threats and vulnerabilities and are useful to provide ideas for potential threats. A study of the use of threat lists is given in [13].
2. From catalogs of misuse patterns. The problem now is that we do not have reasonably complete catalogs. The

closest we have are the patterns in [44]. When we add more patterns to this catalog its value will increase.

6. COMPARING SYSTEMS

If we have two versions of a system, both built following the same requirements, R, one built using security patterns, S1a, and another without them, S1b (Figure 3), we can compare them by enumerating the set of threats of the system, T, and seeing how they can stop these threats; in other words, we can compare their SC values. We can see how they handle the known security threats by checking if they include specific patterns or by tests on the actual code. For a system built using patterns we only need to find if a corresponding security pattern exists in the design.

Using T as a reference, we can find a set of misuse patterns, M, that realize these threats. M adds to T the effect of the lower architectural levels, because the attacker can try any level to reach his goal. Considering both M and T to analyze the relative security of two systems provides a more precise comparison than just comparing the effect of threats on them. The patterns in M can be used as test cases that can be applied to test the final system, which must show it can control them. While it is difficult to apply this comparison to real systems, we can perform it in to a few cases and obtain estimates of the possible security improvement that can be reached through patterns. Of course, the methodology itself is another parameter, a good methodology may guide the designer to apply security patterns better than another approach; that is, we can perform also a comparison of methodologies.

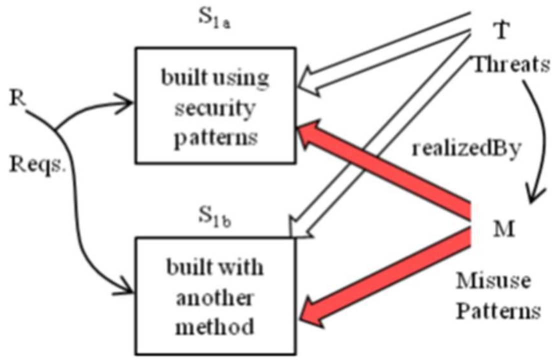

Figure 3. Comparing secure systems.

Using diagrams such as Figure 4 we can see that some security patterns are more effective than others with respect to a specific misuse, e.g., auhorization on the application can stop any confidentiality or integrity attack on it. Patterns at a lower level, e.g. file authorization can stop attacks involving files but don't know about accounts or other specific application aspects. We can see these patternsdefenses as different lines of defense that together make the system more secure.

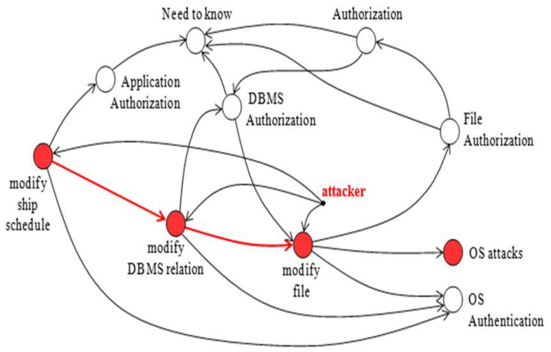

Figure 4. Preventing Misuses.

A vulnerability reduces the quality of a system, so misuses that can succeed indicate what makes system quality low. In other words, if a misuse pattern succeeds it indicates a non-controlled vulnerability in the system. Conversely, each application of security patterns contributes to improve system quality (progression). Graphs like the one of Figure 3 can be used to analyze and test the security of specific systems indicating where to apply misuse patterns. Misuse patterns are useful to determine the effect of the lower levels of the system on security. An example to see the effect of misuse patterns is shown in Figure 4 where a terrorist attempts to modify the ship schedules of a port with the purpose of producing disruption. The figure is essentially a pattern diagram [4], where the nodes represent patterns and the arcs indicate dependencies between patterns. The red (or dark) nodes indicate the misuse patterns of the system (or steps of these patterns) used by the attacker to perform the misuse. The detailed steps are described in each misuse pattern, e.g., misuses using SQL injection [9]. The attacker can modify the class ShipSchedule in the application (exploiting an application vulnerability) or he can modify a relation with the ship schedules information through the DBMS (using SQL injection for example), or he can modify an operating system file containing the ShipSchedule taking advantage of OS vulnerabilities. Finally, he could try to perform this attack through the operating system processes, which would give him the chance to attack any application. The nodes File Authorization, Operating System (OS) Authentication, and DBMS Authorization indicate security patterns describing security mechanisms that could stop the misuse. Notice that File and DBMS Authorization require OS Authentication to be effective. Authorization also requires applying a Need-to-Know policy to be effective. The node indicating OS attacks includes other attacks that would allow file modification but cannot be stopped through authorization. We can use this kind of graphs to evaluate security because they summarize the effect of several misuses and can be used to analyze a complete system; they also show which patterns should be used together to control these misuses (they can help select related SSFs in ASE [45]). SSFs (Security Solution Frames) are combinations of related appaterns. This diagram can also help visualize that all the layers have been protected.

7. REFINING THE METRICS

Twin peaks is an iterative way to build software architectures. In each iteration we should consider again each activity and see how the attacker goals can be reached using the currently defined architecture; this is repeated until we reach the final architecture. We can use this approach to refine our calculation of security (SC), by expanding the list of threats to consider.

Figure 5 shows the ideas behind the Twin Peaks approach. As indicated earlier, our threat enumeration method starts from a conceptual model of the functional requirements; we can analyze its use cases to produce activity diagrams, where the activity diagram uncovers threats as attacker goals and identifies assets at the same time. To define each architecture we can use RAs which in each stage become more concrete. As shown in [27], the concrete architectures introduce new architectural artifacts that can also be attacked and therefore must be protected, e.g., a new database can now be attacked. We refine the goal-based threats to produce a design model which is effectively a SRA. This is refined to produce a new architecture and the cycle repeats. In each cycle we can consider the same use cases but in more detailed versions, considering the new artefacts.

Twin Peaks start from a secure conceptual model where ASPs have been added to a functional model. This model is then converted into a SA by expressing the model using architecture patterns such s brokers, pipes and filters, and similar. We can use secure versions of those patterns [10] to simplify the work of the designer, e.g., a Secure Broker incorporates Authorization in the Proxies and Adapters as well as cryptography and authentication. We can do now a new iteration making the architecture more detailed. The refinements may not be protected by the secure architecture patterns and we need to enumerate the new threats as well as add security patterns to mitigate them. After such iteration we can evaluate the security of the system and decide if we want to add more details to the architecture. In each iteration stage we can use RAs to guide the design. A SRA can be combined with SSFs for more convenient use by developers.

In each iteration of Figure 5 the threat patterns that describe attacks identify the architectural units that are used to perform the attack; these units are the places where we can collect evidence of attacks and where we can assess the damage that was done by the attack. Models of this type can be used for automatic generation of attacks to see if evidence can be collected from them using some specific method; dealing with known threats also helps reduce data collection for forensics [29].

Note that this approach does not replace a methodology to build secure systems such as [45] but it provides a way to improve its process. Figure 4 can be a guide to decide how many levels to consider.

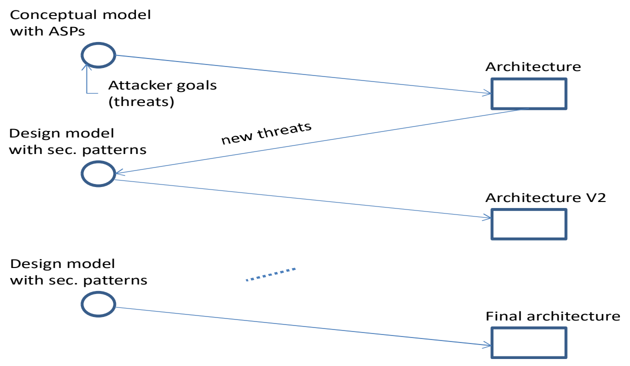
Conceptual model with ASPs
Attacker goals (threats)
Architecture
Design model with sec. patterns
new threats
Architecture V2
Design model with sec. patterns
Final architecture

Figure 5. Twin peaks iterations

8. CONCLUSIONS

We have proposed a specific approach for evaluating the security of applications that starts from enumerating threats as goals of the attacker (this enumeration is part of the application development methodology). We then check if the application includes a pattern able to stop each expected threat. Threat coverage measures the security level of the system.

The attacker has a choice of trying to obtain his goal entering the system at different architectural levels. Each one of these attacks is described by a misuse pattern but we don't have a way to show that all the relevant misuse patterns have been considered. However, if we have security patterns to stop all the identified misuses we can also stop a good number of possible misuses not considered before because of the fact that patterns can build a strong system structure.

Tactics can also complement our measures because they cover the security objectives for a system [2]. We also want to investigate the case where we build a conceptual model with security patterns and we perform model checking to the secure conceptual model to prove at least some high-level security properties. Security patterns could be effective to apply Common Criteria evaluation guidelines in specific products [5].

While we presented our results using a specific threat enumeration approach and we controlled the found threats using security patterns, the metrics apply also to systems where their threats have been enumerated using other approaches and where the discovered threats were stopped or mitigated using approaches different from patterns, such as aspects. We can still measure threat coverage.

The main threat to the validity of our results is the lack of proof that we have found all the threats at all the stages of the lifecycle or all the possible Twin Peaks iterations. However, since use cases include all possible interactions with the system, we have some confidence that all important threats have been found.

We cannot either evaluate security disconnected from the methodology. A correct and security implementation is not possible without an appropriate methodology. A good methodology makes also measuring security much easier [15].

Since a design can be implemented in many ways, this security evaluation is reusable and applies to all its possible implementations. The quslity of the development methodology obviously has an effect on the security of its products; we have defined a set of criteria to evaluate this quality, and this work complements those criteria.

ACKNOWLEDGEMENTS

The work of Eduardo Fernandez was partially supported by the National Institute of Informatics of Japan.

REFERENCES

[1] P. Avgeriou, “Describing, instantiating and evaluating a reference architecture: A case study”, Enterprise Architecture Journal, June 2003.
[2] L. Bass, P. Clements, and R. Kazman, Software architecture in practice (3rd Ed), Addison-Wesley 2012.
[3] F. Braz, E.B.Fernandez, and M. VanHilst, "Eliciting security requirements through misuse activities" Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'07). Turin, Italy, September 1-5, 328-333.
[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns, Volume 1. J. Wiley, 1996.
[5] Common Criteria Portal, last accessed February 22, 2015. https://www.commoncriteriaportal.org/
[6] Common Vulnerabilities and Exposures, https://cve.mitre.org/
[7] E.B. Fernandez, M.M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", Chapter 5 in Integrating security and software engineering: Advances and future visio", H. Mouratidis and P. Giorgini, Eds., IDEA Press, 2006, 1pp. 07-126.
[8] E.B. Fernandez, N. Yoshioka and H. Washizaki, "Modeling misuse patterns", Procs. of the 4th Int. Workshop on Dependability Aspects of Data Warehousing and Mining Applications (DAWAM 2009), in conjunction with the 4th Int.Conf. on Availability, Reliability, and Security (ARES 2009). March 16-19, 2009, Fukuoka, Japan.
[9] E B. Fernandez, E.Alder, R.Bagley, and S. Paghdar, "A Misuse Pattern for Retrieving Data from a Database Using SQL Injection" , RISE'12,Workshop on Redefining and Integrating Security Engineering, part of the ASE Int. Conf. on Cyber Security, Washington, DC, December 12-14, 2012.
[10] E.B.Fernandez, “Security patterns in practice: Building secure architectures using software patterns”. Wiley Series on Software Design Patterns. 2013
[11] E.B.Fernandez, N. Yoshioka, H. Washizaki, and J. Yoder, "Abstract security patterns for requirements specification and analysis of secure systems", Procs. of the WER 2014 conference, a track of the 17th Ibero-American Conf. on Soft. Eng.(CIbSE 2014), Pucon, Chile, April 2014
[12] E.B.Fernandez, Raul Monge, and Keiko Hashizume, “Building a security reference architecture for cloud systems”, *Requirements Engineering*. Doi: 10.1007/s00766-014-0218-7, June 2016, Volume 21, Issue 2, pp 225-249
[13] M. de Gramatica, K. Labunets, F. Massacci, F. Paci, and A. Tedeschi. The Role of Catalogues of Threats and Security Controls in Security Risk Assessment: An Empirical Study with ATM Professionals. In *Proc. of REFSQ'15*
[14] George Hatzivasilis, Ioannis Papaefstathiou, Charalampos Manifavas, "Software Security, Privacy, and Dependability: Metrics and Measurement", IEEE Software, vol.33, no. 4, 46-54, July-Aug. 2016, doi:10.1109/MS.2016.61
[15] M.R. Heckman, R.R. Schell, “Using proven reference monitor patterns for security evaluation”, Information, vol. 7 No 2, 23:1-23:24, Apr. 2016, http:// dx.doi.org/10.3390/info7020023
[16] T Heyman, R Scandariato, C Huygens, W Joosen, “Using security patterns to combine security metrics”, Availability, Reliability and Security, 2008. ARES 08.
[17] T Heyman, K. Yskout, R Scandariato, H. Schmidt, Y. Yu, “The Security Twin Peaks”, ESSoS 2011, LNCS 6542, 167-180, 2011.
[18] Siv Hilde Houmb, Sudip Chakraborty, Indrakshi Ray, Indrajit Ray: Using Trust-Based Information Aggregation for Predicting Security Level of Systems. DBSec 2010: 241-256
[19] W. Jansen, Directions in Security Metrics Research, NISTIR 7564, April 2009. http://csrc.nist.gov/publications/nistir/ir7564/nistir-7564_metrics-research.pdf
[20] J. Jurjens, Secure systems development with UML, Springer-Verlag, 2004.
[21] J. Jurjens, “Sound methods and effective tools for model-based security engineering with UML”, 27th International Conference on Software Engineering (ICSE 2005), ACM, 2005, pp. 322-331.
[22] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson, J. McDermid, D. Gollmann, “Towards Operational Measures of Computer Security”. Journal of Computer Security 2, 211–229 (1993)
[23] K. Manadhata, J.M.Wing, “An attack surface metric”, IEEE Trans. on Soft. Eng., vol. 37, No 3, May/June 2011, 371-386.
[24] D Mellado, E. Fernández-Medina, M Piattini, A comparison of software design security metrics, Procs. of the Fourth European Conference on Software Architecture, 2010, 236-242
[25] NIST, National Vulnerability Database, https://nvd.nist.gov/vuln-metrics
[26] B. Nuseibeh, “Weaving together requirements and architectures”, Computer, vol. 34, No 2, IEEE 2011, 115-117.
[27] Takao Okubo, Haruhiko Kaiya, Nobukazu Yoshioka, “Mutual Refinement of Security Requirements and Architecture Using Twin Peaks Model”. COMPSAC Workshops 2012: 367-372
[28] OWASP, “The Ten Most Critical Web Application Security Risks.” 2013, https://www.owasp.org/index.php/Top_10_2013
[29] L. Pasquale, Y. Yu, M. Salehie, L. Cavallaro, T.T. Tun, B. Nuseibeh, “Requirements-driven adaptive digital forensics”, Proc. of the 21st Int. Requirements Eng. Conf. , 2013, 340-341.
[30] J. Pelaez, E.B.Fernandez, and M.M. Larrondo-Petrie, "Misuse patterns in VoIP", Security and Communication Networks Journal. Wiley, Volume 2, Issue 6, November/December 2009, 635–653, DOI: 10.1002/sec.105
[31] O Rebollo, D Mellado, E Fernández-Medina, H Mouratidis, Empirical evaluation of a cloud computing information security governance framework, In2ormation and Software Technology 58, 2015, 44-57
[32] B.D.Rodes, J.C.Knight, K.S.Wasson, “A security metric based on security arguments”, WETSoM’14, June 2014, Hyderabad, India, 66-72.
[33] N. Rozanski, E. Woods, “Software systems architecture: working with stakeholders using viewpoints and perspectives”. Addison-Wesley Educational Publishers., 2nd Ed. (2012)
[34] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, Boston, Mass., 1999.
[35] J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems”, Procs. of the IEEE, vol. 63, No 9, Sept.1975, 1278-1308.
[36] Reijo M. Savola: “Quality of security metrics and measurements”. Computers & Security 37: 78-90 (2013)
[37] CMU SEI, http://resources.sei.cmu.edu/asset_files/Presentation/2011_017_001_52415.pdf
[38] M. Schumacher and U. Roedig, “Security engineering with patterns”, Procs. Pattern Languages of Programming Conference, 2001
[39] R von Solms, H Van Der Haar, SH von Solms, WJ Caelli, “A framework for information security evaluation”, Information & Management, 26 (3), 143-153
[40] Sal Stolfo, Steven M. Bellovin, and David Evans.”Measuring security”. *IEEE Security & Privacy*, 9(3):88, May--June 2011.
[41] R.N.Taylor,N. Medvidovic, and N.Dashofy. Software architecture: Foundation, theory, and practice, Wiley, 2010.
[42] A.V. Uzunov, E.B. Fernandez & K. Falkner (2012), "Securing distributed systems using patterns: A survey", Computers & Security, 31(5), 681 - 703. doi:10.1016/j.cose.2012.04.005
[43] A. V. Uzunov, E.B.Fernandez, and K. Falkner, “Engineering Security into Distributed Systems: A Survey of Methodologies”, Journal of Universal Computer Science, Vol. 18, No. 20, 2013, pp. 2920-3006 http://www.jucs.org/jucs_18_20/engineering_security_into_distributed
[44] A. V.Uzunov and E.B.Fernandez, “An Extensible Pattern-based Library and Taxonomy of Security Threats for Distributed Systems”- Special Issue on Security in Information Systems of the Journal of Computer Standards & Interfaces. 2013. http://dx.doi.org/10.1016/j.csi.2013.12.008
[45] Anton Uzunov, E. B Fernandez, Katrina Falkner, “ASE: A Comprehensive Pattern- Driven Security Methodology for Distributed Systems”, Journal of Computer Standards & Interfaces, Volume 41, September 2015, 112-137
.