Risky Trust: Risk-Based Analysis of Software Systems

Zaid Dwaikat
Systems and Software Consortium, Herndon, VA
George Mason University, Fairfax, VA
dwaikat@software.org

Francesco Parisi-Presicce
George Mason University, Fairfax, VA
Univ. di Roma La Sapienza, Rome (IT)
fparisi@ise.gmu.edu

ABSTRACT

Measuring the security of a software system is a difficult problem. This paper presents a model using common security concepts to evaluate the security of a system under design. After providing definitions for all relevant concepts and formalizing some of them, we define security requirements for transactions and provide mechanisms to measure the likelihood of violation of these requirements. Our model is based on individual risks presented by system components. Based on the security policy and individual risks, we calculate violation risk for a certain transaction. Context and other risk factors are considered and can be used to adjust the final risk figure. As part of our discussion, we address trust and risk and their significance to security engineering. Based on the decision process, the same trust assumptions may increase, or decrease, the risk to the system. We model the fact that small individual risks can be transformed into major risks when combined together in a complex attack.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General- *Protection Mechanisms,* H.1.0 [**Information Systems**]: Models and Principles- *General*

General Terms

Design, Security, Verification.

Keywords

Risk analysis, trust, distrust

1. INTRODUCTION

Security is not a binary condition. Systems are neither secure nor insecure. Security is often described in relative terms. A system is more, or less, secure than something else. This is caused by the difficulty of specifying how secure a certain system or component is. Complexity, connectivity and extensibility make security decisions very difficult [10].

One of the major questions organizations often face is "how secure are my systems?" Answering such a question is often difficult. Despite extensive research in security engineering, measuring security is still a difficult problem. Part of the problem is attributed to the unpredictability of software systems' vulnerabilities. Add to this an ever-increasing complexity, lack of awareness and unwillingness to share data on security attacks, and the problem quickly becomes intractable.

While we do not have security measurements with absolute certainty, we often rely on measurement of risk in assessing security. Using risk of violations to evaluate security decisions is a common practice [1,12,17]. It provides a systematic mechanism for optimizing cost and resources. The difficult part lies in providing accurate information on attacks and their likelihood [2]. Since systems are typically exposed to constant changes, associated risks are often affected by such changes. However, risk assessments are not typically repeated as often as changes are introduced into systems. Over time, initial risk estimates become outdated possibly leading to less secure systems.

As we have matured in the area of perimeter protection mechanisms, so have the attackers. Security attacks are moving higher in the traditional network stack from automated network-level attacks to complex application-level attacks [10]. This is evidenced by the large amount of web-based attacks (e.g. XSS, SQL injection) following a general trend moving from proprietary architectures to standard web-based ones. The overall security of systems is no longer tied to the security of the network. Applications, people, commercial and open source software, all contribute to the *insecurity* of software systems. Only a holistic approach that considers all these factors is likely to succeed in preventing attacks.

One of the most discussed issues about security is related to trust assumptions [19,6,20]. Trust is essential for systems to function in today's diverse environment of software systems. But how much trust is enough? Misplaced trust is the cause for many security vulnerabilities and attacks [19,10]. Lack of input validation in software systems, a major source of vulnerabilities, is a reflection of the problem. Developers believe that the user, or another software system, should be trusted to provide appropriate input. From our experience, the question of distrust is rarely asked as the system is designed. Our focus here is on how risk and trust interrelate and how that affects the security of a system.

Security requirements and policies define security properties for systems at multiple Levels [16]. Data, by its nature, is a frequent target for security constraints. Classification schemes and access control models are used to provide guidance and mechanisms for ensuring security properties for data elements. Security properties are also enforced for other parts of software systems. Software components, files, memory segments and removable media are examples of items that may carry a requirement for a security property. Security requirements are essential for the validation of the security of a system. In our model, we define security requirements for *transactions* which consist of a sequence of messages. We use such requirements combined with risk values to evaluate the potential for violations.

In this paper we follow a basic approach. We first define a number of concepts that are essential to security engineering and discuss their relevance and how they impact security. We then develop a model for assessing security risks using these concepts and a simple methodology to provide process guidance. Finally we present an example to illustrate the concepts discussed and explain how a risk assessment at a transaction's level is performed.

2. RELATED WORK

Trust and risk have been widely discussed in the literature. In [15] a trust vector model is developed that provides for various degrees of trust and distrust. The paper discusses long term effects on trust relationships and an expression for computing trust vectors. However, no relationship is made between security attacks and the trust model. In [9] the authors analyze how trust assumptions are utilized by requirements engineers to influence decisions throughout the development process. The evidence used to establish such assumptions is not addressed.

Trust is sometimes aligned with quality [13] and other times with security [19,20]. The problem of ad hoc trust assumptions and their effects on security is articulated in [19,20]. In [10] many example attacks are presented that are based on misplaced trust. Trust analysis for security in software engineering is discussed in [3]. The authors describe tools and techniques that can be used to assure the trustworthiness of a software program. The authors in [8] analyze trust among a network of people and present a trust propagation framework. Trust has been extensively discussed in computer networks [18] especially in analyzing PKI trust mechanisms.

Risk is an old concept that made its way into security sometime ago. However, assessing risk in information security is more difficult [2]. Risk is extensively discussed in risk management methodologies [1,17]. Such methodologies provide mechanisms for optimizing cost while allowing for an acceptable security risk. Risk analysis and threat modeling [12,20] are considered cornerstones in secure code development. While risk and trust are related, analysis of their relationship has been mostly informal. In [4] the authors consider trust and risk in role-based access control, and incorporate trust-based evaluation into access control decisions.

Conducting security risk assessments is, in reality, still more of an art than a science. Evaluating the security of a software design is still mostly based on expertise and manual analysis. There have been many attempts [7,14] at analyzing the security of a certain system, or a specification. However, in most cases the analysis is very specific and is not necessarily applicable to other types of systems. Our focus is on a generic model that can be adapted by various systems using different layers of abstraction.

There is considerable research in the area of risk analysis for reliability. However, while some argue that security is a subset of reliability [20] we argue that some security properties (e.g. availability), but not all, are a subset of a system's reliability. Reliability implies a defined set of requirements that the system must adhere to. Many security violations may not be a violation of a specific requirement, or a security policy, but are still recognized as security failings of the system. Reliability is often associated with system behavior given normal usage, while security is focused on malicious and hostile behavior.

3. TRUST AND DISTRUST

Trust lies at the heart of security engineering and software systems are often built with many assumptions about trust [20]. Some of these assumptions often cause costly security problems, as developers often mishandle trust relationships leading to a variety of security bugs [19]. Decisions about trust can be made at different levels in a software system. Software components trust each other, trust the operating system and most often trust the users of the system. Questions about whom to trust when developing software systems are numerous, usually with no simple answers. To analyze trust we adopt a definition from [6]

Definition 1. *Trust is defined to be the firm belief in the competence of an entity to act dependably, reliably and securely within specific context.*

Extending trust to other software components is essential to the functioning of software systems. Generally, less trust in software components means fewer security problems, but more development work [10]. Software security controls have to be developed for each interaction with a distrusted entity, be it a user, a software component or another software system.

Just like trust, software designers should consider distrust. Distrust stems from potential misuse and abuse leading to security violations [5]. Measuring distrust allows us to express potential negative behavior of system components and users. It is especially relevant in addressing intentional malicious behavior expected from certain system entities. Distrust has been defined several times in the literature; we use a definition by Ray et al. in [15]

Definition 2. *Distrust is defined to be the firm belief in the incompetence of an entity to act dependably, reliably or securely within specific context*

To express trust and distrust, we use a range of $[-n,n]$. n is an appropriate scale for the system at hand and is defined by the system's designer. Trust values are in the range of $(0,n]$ and distrust values are in $[-n,0)$. A value of 0 indicates no knowledge on trust and reflects a neutral position.

4. SOFTWARE SYSTEMS

We consider software systems as a collection of interacting software components. Components are often referred to as reusable software modules [13]. We slightly expand this definition for our discussion and consider them to be the lowest level entities in a system. This allows us to provide a high level of abstraction for evaluating security properties. We define a software component as follows.

Definition 3. *A component is a self-contained software entity that performs a particular function and has a well-defined interface.*

A component may be a system library, an object, a file, or an individual server. Logical or physical partitioning is used to define the level of abstraction. Security modeling in our framework assumes that all entities belong to the same level of abstraction.

We consider components a source of risk despite common perceptions which emphasize network infrastructure as a source of risk. Software vulnerabilities, mis-configuration, application flaws and malicious users represent security risks introduced by components. Our emphasis for risk in software components does not necessarily imply that the component itself is malicious, although that is possible with Trojan Horses, root kits and backdoors, but rather an admission that such software can be used to violate the system's security constraints.

Components need mechanisms to interact with each other; we use *channels* to describe such mechanisms. Channels are communication abstractions that allow for finer definitions of their nature as designs are refined.

Definition 4. *A channel is a communication medium that transfers information between components.*

We encapsulate interactions between components as *messages* and *transactions*. We define messages and transactions as follows.

Definition 5. *A message is an atomic entity representing data being transferred between components using channels.*

We use a *message* as an abstract concept that can represent a variety of data types. Messages can be IP packets, HTTP requests or XML documents. Such flexibility ensures applicability of this model to various types of transactions and protocols. Messages are not restricted to specific transactions; the same message may be used in more than one transaction. However, the same message may have different security requirements from one transaction to another.

Definition 6. *A transaction is a sequence of messages in a specific order transferred between two or more components.*

Let *C* denote the set of components in a system. A transaction *t* is represented as a sequence of tuples of the form

$< m_0, c_{01}, c_{02} >|< m_1, c_{11}, c_{12} >| \ldots$ where $c \in C$ *and*

c_{01}, c_{02} are the sender and receiver components, respectively.

We combine all concepts presented so far into a system and define it as follows.

Definition 7. *A system is a collection of components, channels, messages and transactions defined in an architecture.*

A system *architecture* consists of disjoint sets of *components* and directed *channels* $h : a \longrightarrow b \in Ch$ from a *source* component *src(h)=a* to a *target* component *tar(h)=b*.

5. SECURITY POLICY

To provide meaningful conclusions about the security of a system design, we need to define constraints on the behavior of the system. Security policies define acceptable and unacceptable behavior of software systems [16]. They often articulate requirements for access control, information flow and availability. A *security policy* in our model is limited in scope and is concerned with transactions and messages, and their respective security properties.

Definition 8. *A security policy is a set of security requirements that have to be enforced by the system. Each requirement is represented by a tuple in the form of* $< m_i, p_k, t_j >$ *where m is a message, p is required security property and t is the transaction.*

Messages are only assigned security properties within the context of a transaction. Not all messages have security requirements. Decisions on messages' security properties can only be made based on a security analysis of a transaction. Such analysis ensures that by assigning individual security properties to messages, the transaction's overall security requirements are being defined. The security requirement for transaction *t* is the set of $< m_i, p_k, t_j >$ tuples where $t_j = t$.

Security is context sensitive [14]. To make valid assumptions, we have to analyze the environment in which a software system is intended to operate. In many cases, this environment is not well defined at the early stages of the Software Development Life Cycle, and a complete implementation description is often not available as design decisions about the system are being made. In other cases, users deploy systems in ways never imagined by their creators. Ultimately, it is the users who define how a system is deployed based on their needs and environmental and budget constraints. Whether the problem is lack of data, or user *mis-configurations* the outcome is the same: assumptions about trust relationships (and other assumptions) may no longer be valid. This exposes the system to attacks as a result of misplaced trust.

6. RISK

Risk is the probability of exploitation of software vulnerabilities [11]. Such exploitation often results in a loss, either tangible or intangible, to the system owner. The level of risk and potential loss are used to make decisions on mitigation techniques and their cost [12] Risk in the business world is considered a source of profit, not just a source of loss [4]. The same concept applies in security engineering. Accepting little or no risk either results in building a non-usable system, or not deploying a system, resulting in total loss of the value of the system.

Risk is context sensitive. Flexibility and adaptability of any risk management model are essential to its success. Since ensuring that a system is a 100% secure is not feasible, organizations often have to perform systematic risk assessments to evaluate the security of a software system. From our experience, this practice is common with deployed systems but systems under development rarely go through such process. Our approach targets software designs to resolve tradeoffs and trust issues in a systematic manner.

Risk management methodologies fall into three categories: qualitative, quantitative or hybrid [1,17]. When measuring risk, qualitative approaches use subjective terminology, while quantitative ones use concrete values. Quantitative approaches are considered more accurate, but defining the underlying parameters is often difficult [2,12]. Regardless of the approach, risk management is essential to security engineering because it provides a way for optimizing cost and resources in mitigating risks. Once a risk has been identified, there are usually three actions that can be taken: accept the risk, mitigate the risk or transfer the risk. Accepting the risk implies doing nothing, mitigating implies adding security controls or redesigns to reduce the risk and transferring implies shifting the cost of potential loss to another party.

We define risk as follows.

Definition 9. *Risk is the probability of violation of a basic security property enforced by the system. Basic properties include confidentiality, integrity, authenticity and non-repudiation.*

Let *P* denote the set of desired security properties, *C* the set of components and *T*, the set of transactions. Risk presented by a single component is defined as a function *r* where

$r(p,c,t)_e = x$ *where* $p \in P, c \in C, t \in T$ *and x in [0,1]*

We define risk presented by a channel in a similar manner, where H is the set of channels in a system

$r(p,h,t)_e = x$ *where* $h \in H$ *and x in [0,1]*

Given a desired security property *p,* a transaction *t* and a system entity *c or h, r* provides the probability of violating *p* by *c or h* in the context *e.* The security violation risk applies to all the messages in the transaction *t* that have a requirement for the property *p*. Changing any of the three function parameters does not necessarily translate into different risk values. For example, the risk of eavesdropping on a channel stays constant regardless of the transaction, while the risk of unauthorized disclosure may change if a different transaction utilizes some type of encryption. Risk values may be reduced through redesign, or the introduction of security controls. However, security risks are rarely eliminated entirely, but rather reduced to an acceptable level [17].

In our model, we focus on risks associated with transactions. Initial values for security risks associated with components and channels are derived through a quantitative risk analysis. We do not provide a mechanism for such process but assume that it is based on a well-defined approach such as OCTAVE [1] or NIST [17]. We are also working on an approach that specifically addresses risks presented by commercial software.

7. TRUST AND RISK

There is an inherent relationship between risk and trust. At first look, such relationship seems intuitive, as a higher level of trust of a software component implies a lower risk of a security violation, and vice versa. However, a closer look reveals more subtle nuances on how trust and risk interrelate. We evaluate the relationship between trust and risk and how they affect each other. The basic question is whether trust drives risk, or the opposite holds. In either case, what are the consequences of trust decisions on the security of the system?

Considering a high level of trust first, we interpret that our belief in the security competence of a software component logically leads to lower risk. That is a reflection of the high level of trust. The same high level of trust, paradoxically, may lead to higher level of risk. Extending too much trust is a common problem in software systems [19,20] and often leads to major security violations. The fact that software developers make trust assumptions without real analysis of consequences allow for a variety of security attacks. These attacks often exploit the trust assumptions, often very generous, made by the developers.

In a similar manner we analyze assigning a low level of trust to a component. Logically that implies a potential high risk for security violations. However, a low level of trust results in lower risk, since lower trust often requires the design of additional security controls. Developers who are stingy with trust build systems that are more resistant to attacks [6].

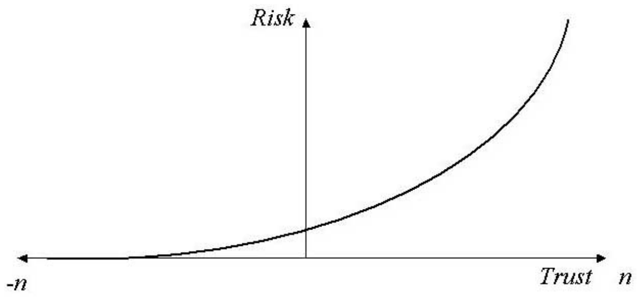

Figure 1. Trust and risk using ad hoc approach

The relationship between trust and risk, using ad hoc trust decisions without proper risk analysis, is represented in **Figure 1**. The curve represents how risk increases based on trust values. We claim that consideration of distrust (e.g. the belief in intentional abuse) greatly reduces risk. Even a neutral trust assumption (*trust=0*) still results in low risk value. As trust increases, so does risk. As we get close to a *fully trusted* point, risk increases rapidly. The curve's position and incline will differ from one situation to another. However we claim that such relationships follow a similar theme.

Using risk assessment in making trust decisions is a more straightforward case. A higher level of risk implies that we should have a lower level of trust. The opposite is also true: the lower the risk the higher the trust. To avoid getting into a cycle (low risk $\rightarrow$ high trust $\rightarrow$ high risk…) trust assumptions should be validated periodically, especially when risk changes. Trust relationship to risk, when trust is based on proper risk analysis is shown in **Figure 2**. The curve here is a reflection of the one in **Figure 1** indicating that risk is driving trust values.

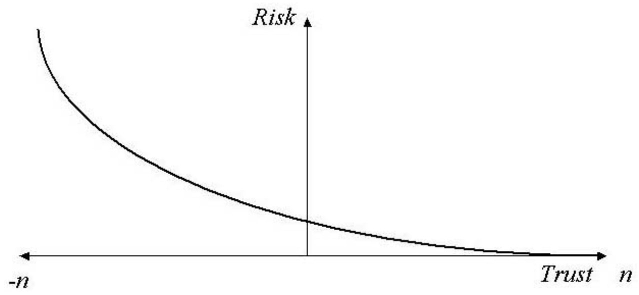

Figure 2. Trust and risk using systematic risk evaluation

Ultimately, the effect of trust decisions on risk levels is largely dependent on the decision process initially used to reach trust assumptions. On one hand, justified trust based on risk assessment using a well-defined methodology provides for sound security engineering. On the other hand, unjustified trust without proper risk assessments increases the risk of security violations. We argue that trust, at least partially, should be a function of violation risk. However, trust and risk are only two aspects in engineering secure systems and a multitude of other design decisions and information contribute to a system's overall security.

Our discussion does not imply that trust and risk are static values. Indeed both entities change over time. However we claim that our analysis of their relationships still holds true in an environment of constant change. While many security-engineering decisions are based on trust and risk values, deriving such values is not trivial. For instance, many risk management tools provide accurate estimates of losses, given that they are fed with accurate parameters. However, answering questions on exploitation likelihoods, potential vulnerabilities, and intangible assets is still an art more than a science. Nevertheless there is clear evidence that systematic risk and trust analysis of software systems provides great benefits to the overall security of such systems [1,17]. These analyses often result in systems with smaller attack surface and solid security controls. Providing guidance on objective analysis of risk and trust values is an important topic and is outside the scope of this paper.

8. SECURITY ASSESSMENT

The purpose of our model is to evaluate the level of compliance a certain software system's design exhibits given our security policy. Specifically we want to know the probability that a transaction fails to meet its security requirements. We first define how to calculate a transaction's risk using information from the *security policy* and risks presented by components and channels.

We base our formula on the fact that risk accumulates as the transaction is executed. Since a transaction is a sequence of tuples, we calculate its risk with respect to a certain security property *p* based on individual risks associated with its tuples. For a transaction *t*, a security policy ψ and a fixed property *p*, for a security assessment with respect to *p*, we restrict our attention to a subset of *t*, as the (sub-)sequence of tuples where $< m, c_1, c_2 > \in t \; AND \; < m, p, t > \in \psi$ (only tuples that carry the specific security requirement are considered).

To calculate a transaction's risk, we define a function $R(t_i)_p$ which provides the risk of violating *p* in the part of the transaction *t* starting with the message *i*. Let *t* be the transaction $< m_0, c_{01}, c_{02} > | < m_1, c_{11}, c_{12} > .... < m_n, c_{n1}, c_{n2} >$ and t_i the subsequence of *t* starting at message *i* (t_0 is the same as *t*) we define $R(t_i)_p$ as

$$R(t_i)_p = f_1 * r_{m_i} + f_2 * (1 - r_{m_i}) * R(t_{i+1})_p \; for$$
$$0 \le i < n \, , R(t_n)_p = (f_1) * r_{m_n} \; and \; f_1, f_2 \le 1$$

It is easy to prove by induction that $0 \le R(t_i)_p \le 1, \; \forall i$

Here f_1 and f_2 are adjustment factors based on the system context, and r_{m_i} is the maximum risk for property *p* with respect to message *i* tuple. Each tuple has three risks: risk from c_{i1}, risk from c_{i2} and risk from the connecting channel. We use the maximum value considering the worst-case scenario. To avoid counting the risk twice from the same component, c_{i2} is only considered if it was not considered in the subsequent tuple $c_{(i+1)2}$.

The factors f_1 and f_2 are used to provide an adjustment based on our impression of the overall risk. Such factors are useful when analyzing a system that spans more than one environment (e.g. a corporate HQ and a field office). It represents our belief that one part of the system may be riskier than the other. In cases where $f_1 < f_2$, the initial parts of the transaction generally carry less risk than the final parts; when $f_1 > f_2$, the initial parts carry higher risk than the final parts. This allows us to influence risk values without necessarily changing individual risks. If f_1 and f_2 are equal, no statement is being made on environment-related risk. Adjusting the values of f_1 and f_2 allows for further refinement of the final risk values.

If $f_1 = f_2 = 1$, with the exception of cases with a single or no tuples carrying the desired security property, $R(t_0)_p$ always evaluates to a higher risk than the highest individual risk value. This reflects the observation that most successful security attacks do not rely on a single vulnerability, but rather a complex combination of exploits that leads to a major security violation. We observe that despite potentially low individual risk values for these vulnerabilities, combining them together in a transaction results in a higher risk for violation.

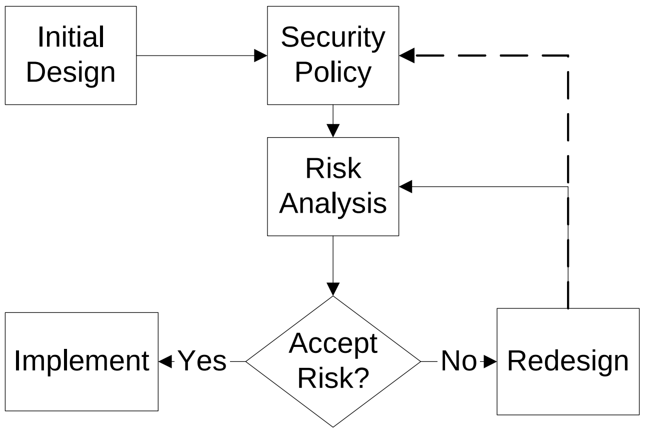

Figure 3. Evaluation process

To make our model usable, we propose a simple iterative process that is conducted in parallel with other software development activities. The general approach is shown in **Figure 3** and has the following steps.

1. Define the initial system design including transactions
2. Define the security policy for each transaction that carries security properties
3. Perform a quantitative risk analysis deriving the risk values for components and channels
4. Evaluate whether the transactions' security risks are acceptable
5. If yes, implement design, if no
    a. (optional) Refine security policy
    b. Redesign and add security controls to mitigate risks
    c. Go to step 4

6. EXAMPLE

To illustrate our methodology we use a simple example. **Figure 4** represents a web-based application with labeled components. Web-based applications are becoming the de facto standard for many business applications. Major software vendors have either added, or replaced, their client software with a browser interface. This has the benefit of not needing a separate client, it allows for easier maintenance and it utilizes standard web protocols. The side effect of such approach is that more and more systems are exposed to web-based attacks.

We analyze a typical web transaction t that contains an HTTP request accompanied with a session ID (usually stored in a soft cookie). Session IDs are commonly used in web applications to maintain information about the state. They often become the only mechanism to authenticate users who already logged in to the application.

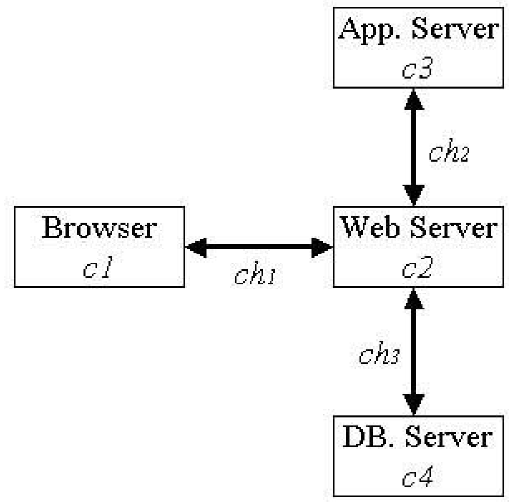

Figure 4. Example system

The transaction is summarized in **Table 1** and described as follows: A user, using the browser, sends a request to the web server. The request includes the session ID as well as a resource name. The web server validates the session ID against the database (session IDs typically expire within a short time period). If the session ID is valid, the web server sends a request to the application server requesting a resource. The web server sends the resource to the user. This implies that the web server carries some application logic, a practice that is discouraged (lacking compartmentalization and defense in depth) but widely used from our own experience.

Table 1. Transaction details

Message	From	To	Content
m0	c1	c2	HTTP request, session ID
m1	c2	c4	SQL statement, check session valid
m2	c4	c2	SQL response (OK or empty)
m3	c2	c3	Resource request
m4	c3	c2	Resource response
m5	c2	c1	HTTP response including resource

The requested resource happens to have user sensitive data that requires confidentiality. Since a session ID can be stolen and used to hijack a user's session, we are concerned with the confidentially of the session ID. A session ID might be changed to access another user's session from which we derive an integrity requirement for the session ID. Based on these requirements we define a security policy for the transaction t as $\{<m_0,t,c>,<m_0,t,i>,<m_1,t,c>,<m_1,t,i>,$ $<m_4,t,c>,<m_5,t,c>\}$.

We perform a risk analysis (or use data from prior risk assessments) on channels and components and obtain the risk values in **Table 2** for the transaction *t*. For each system entity we assign a violation risk value for integrity and confidentiality. These values may, or may not, change from one transaction to another. We consider that the component initiating a message still poses a violation risk. Such component may be violated by another component outside our system boundaries, or a malicious user.

Table 2. Risk values (1)

Entity	I	C
c1	.7	.6
c2	.4	.6
c3	.2	.4
c4	.2	.3
ch1	.4	.3
ch2	.3	.3
ch3	.2	.4

Table 3. Risk values (2)

Entit y	I	C
c1	.4	.3
c2	.2	.3
c3	.1	.4
c4	.2	.1
ch1	0	0
ch2	0	0
ch3	0	0

Risk values are based on a variety of data that are not fully discussed here for space constraints. Such information includes types of channels, software vulnerabilities, IT personnel and context. For example ch_1 is an Internet link, c_2 is a web server with a long history of vulnerabilities, ch_2 is semiprivate etc. Deriving such values for software components is an important research topic that we plan to address in the near future. We want to acknowledge that such process can be highly subjective and risk values may be challenged.

Based on our understanding of the system, we believe that the risk is equally weighed throughout the system. Therefore we use the same values for f_1 and f_2. Given information about the potential usage of the system, its environment and level of security awareness, we use a value of .8 for both f_1 and f_2. Using the algorithm defined above for evaluating a transaction's risk, we calculate two values for *R*, violation of confidentiality and violation of integrity $R(t_0)_c = .69$ *(4 messages) and* $R(t_0)_i = .40$ (2 messages).

So far we have completed the first four steps in our methodology. Now we can make a decision on whether to accept such risks or lower them through changes to the security policy and design. For our example, we are going to do two things: first, remove the integrity requirement for session IDs by using unique and unpredictable IDs, then use an encryption protocol (e.g. SSL or IPSEC) on all channels. Our risks are adjusted and shown in table 3. Since integrity is no longer a requirement in the security policy, there is no violation risk. Confidentiality violation risk is calculated as $R(t_0)_c = .64$

As expected, the value of *R* does not change much. That is because most of the confidentiality violation risk comes from the software components as opposed to the channels. That is a result of our use of maximum values in any individual message trip. If we want to further reduce the risk, we have to address the components themselves. This could be an acceptable risk; otherwise, we have to cycle through the process until an acceptable risk is reached. Whenever the underlying parameters are changed, this process should be repeated to measure the effect of the change on the transaction's risk. Changes may not only include architectural modifications, but changes in risk, context and technologies should be considered.

While this example explains the benefits from our approach, we do not believe it is adequate. The real value from our model comes when used in a larger system with complex transactions. We plan deploying this model using a real life system as a case study. That will provide us with more insight and validate the applicability of our approach.

9. CONCLUSION AND FUTURE WORK

We used common concepts in security engineering to create a model for security assessment. The model is based on risk, the most widely accepted form of security measurement. Our approach emphasizes that every part of the system carries a risk, no matter how small. Software components, people and communication channels present security risks. Only a comprehensive analysis of all of them would provide realistic conclusions on the security of a system. We formalized some aspects of our model, and provided analysis on the relationship between trust and risk.

Our experience and available public data show that real attacks are often a set of complex exploits utilizing many vulnerabilities. We captured that fact with an algorithm that combines individual risk values and provides a higher figure for overall risk. The algorithm considers security requirements, context and risks presented by various parts of the system. We described a simple process that can be used to utilize our model.

We plan to further extend our model and apply it to a real life system under development. Another area of interest is to validate our analysis between trust and risk with real data.

10. REFERENCES

[1] Alberts,C.J. , Dorofree A.J., Managing Information Security Risks: the OCTAVE Approach, Addison Wesley, July 2002.

[2] Blakley, B., McDermott, E. and Geer, D., Information Security is Information Risk Management, Proceedings of the 2001 workshop on New security paradigms September 2001.

[3] Devanbu, P., Fong, P.W., Stubblebine, S.G., Techniques for Trusted Software Engineering, Proceedings of the 20th International Conference on Software Engineering, Pages. 126–135, April 1998.

[4] Dimmock, N., Belokosztolszki, A., Eyers, D., Bacon, J. and Moody, K., Using Trust and Risk in Role-Based Access Control Policies, Proceedings of the ninth ACM symposium on Access control models and technologies, June 2004.

[5] Dwaikat, Z., Parisi-Presicce, F., From Misuse Cases to Collaboration Diagrams in UML, Proceedings of the 3rd International Workshop on Critical Systems Development with UML, October 2004.

[6] Grandison, T. and Sloman, M., A Survey of trust in internet applications, IEEE Communications Surveys and Tutorials 3 2000, pp. 2-16, 2004.

[7] Grunbauer, J., Hollmann, H., Jurjens, J. and Wimmel, G., Modeling and Verification of Layered Security Protocols: A Bank Application, SAFECOMP 2003, LNCS 2788, pp. 116–129, 2003.

[8] Guha, R., Kumar, R., Raghavan, P. and Tomkins A., Propagation of Trust and Distrust, Proceedings of the 13th international conference on WWW, May 2004.

[9] Haley, C., Laney, R., Moffett, J., and Nuseibeh, B., Picking Battles: The Impact of Trust Assumptions on the Elaboration of Security Requirements, iTrust 2004, LNCS 2995, pp. 347-354, 2004.

[10] Hoglund, G. and McGraw, G., Exploiting Software: How to break Code, Addison-Wesley, 2004.

[11] Internet Engineering Task Force, RFC 2828, www.ietf.org

[12] McGraw, G., Managing Software Security Risks, IEEE Computer ,Vol. 35, Issue: 4, March 2002, Pages: 99 – 101.

[13] Meyer, B., The Grand Challenge of Trusted Components, Proceeding of 25th ICSE, 2003, Pages. 660–667, May 2003.

[14] Oheimb, D. and Lotz, V., Formal Security Analysis with Interacting State Machines, ESORICS 2002, LNCS 2502, pp. 212–229, 2002.

[15] Ray, I. and Chakraborty, S., A Vector Model of Trust for Developing Trustworthy Systems, ESORIC 2004, LCNS 3139, pp. 260-275, 2004.

[16] Schneider,F., Enforceable Security Policies, ACM Trans. On Information and System Security, Vol. 3, No. 1, Feb. 2000, pp. 30-50.

[17] Stoneburner, G., Grogen, A., Dering, A., Risk Management Guide for Information Technology Systems, National Institute for Standards and Technology, SP 800-30.

[18] Theodorakopoulos, G. and Baras, J., Trust Evaluation in Ad-hoc Networks, Proceedings of the 2004 ACM workshop on Wireless Security, October 2004.

[19] Viega, J., Kohno, T. and Potter, B., Trust (and Mistrust) in Secure Applications, Communications of the ACM, Feb 2001, Vol. 44, No. 2.

[20] Viega, J. and McGraw, G., Building Secure Software: How to Avoid Security Problems the Right Way, Addison-Wesley, 2001.