Asset-Centric Security Risk Assessment of Software Components

Tobias Rauter, Andrea Höller, Nermin Kajtazovic, Christian Kreiner
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
{tobias.rauter, andrea.hoeller, johannes.iber, christian.kreiner}@tugraz.at

ABSTRACT

Risk management is a crucial process for the development of secure systems. Valuable objects (assets) must be identified and protected. In order to prioritize the protection mechanisms, the values of assets need to be quantified. More valuable or exposed assets require more powerful protection. There are many risk assessment approaches that aim to provide a metric to generate this quantification for different domains. In software systems, these assets are reflected in resources (e.g., a file with important information) or functional software components (e.g., performing a bank transfer). To protect the assets from different threats like unauthorized access, other software components (e.g., an authenticator) are used. These components are essential for the asset's security properties and should therefore be considered for further investigation such as threat modeling. Evaluating assets only at system level may hide threats that originate from vulnerabilities in software components while doing an extensive threat analysis for all the system's components without prioritization is not feasible all the time.

In this work, we propose a metric that quantifies software components by the assets they are able to access. Based on a component model of the software architecture, it is possible to identify trust domains and add filter components that split these domains. We show how the integration of the methodology into the development process of a distributed manufacturing system helped us to identify critical sections (i.e., components whose vulnerabilities may enable threats against important assets), to reduce attack surface, to find isolation domains and to implement security measures at the right places.

1. INTRODUCTION

The development of secure systems is a difficult and error-prone task. At business level, security properties (e.g., confidentiality, integrity, availability) of physical or non-physical valuable objects (assets) must be guaranteed. Many of these assets are protected and/or used by information systems. Therefore, it is important to identify the assets in these systems and build proper protection mechanisms. At the embedded domain in particular, where valuable information like encryption keys often cannot be shielded physically, building robust software countermeasures that protect the assets is an essential task.

In the identification context, assessment and mitigation risk management [1] is a widely used method [7-13]. Basically, possible risks are identified, rated and prioritized. Based on the prioritization, it is possible to focus on the protection of the assets with the highest loss potentials in case of violation of its security properties. This risk-control step consists of risk resolution, monitoring of the resolutions and re-assessment.

In software systems, assets are mainly data (e.g., encryption key) or a functionality (e.g., a bank transfer function). Other software components are either accessing or protecting these assets. Threats to assets thus rise through exploits of possible vulnerabilities in components that are able to access the asset. Approaches that generate security requirements based on security goals for assets [2] and threat modeling [3, 4] have been proposed. However, to the best of our knowledge, no investigation into the systematic generation of the list of software components that have to implement these requirements for the high-level assets exists at present. Furthermore the question of which components are crucial for the system's security properties and therefore should be considered for exhaustive threat modeling can also not be answered systematically.

In this work, we aim to achieve this enumeration and prioritization by rating components based on the assets they are able to access at an architectural level. In particular, we

- identify required privileges of components by analyzing accessed assets,
- introduce a metric that represents the criticality of the components in the context of their privileges,
- classify components and component-groups according to this metric,
- use this quantification as input for risk assessment and as feedback for the privilege separation process to generate small trust domains.

Finally, we use the proposed method to analyze an existing software architecture of an embedded control system regarding security. Moreover, we adapt it in the development phase of a distributed manufacturing and automated test system for these devices. By using this approach, we are able to reduce the number of critical components and to focus effort on actual threat modeling and countermeasures in a prioritized manner. Additionally, the classification of components and the identification of domains with the same requirements regarding asset accesses can be used as input for component isolation-technologies.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes the proposed methodology and the metric used to evaluate the components. Section 4 shows how we use the approach to examine a real case software architecture. In Section 5, the benefits and the drawbacks of the system, together with impulses and directions that can be followed in future are summed up.

2. BACKGROUND AND RELATED WORK

This section provides an overview of a basic risk management processes, also a summary of related risk assessment methodologies.

2.1 Risk Management

Risk management is an important method for identifying, evaluating and dealing with risks in information systems. The ISO/IEC 27005 [5] contains guidelines for systematic and process-oriented risk management. Basically, stakeholders (e.g., owners) want to protect objects with some kind of value. These objects are referred to as assets.

However, actual or assumed 'threat agents' (e.g., malicious users, hackers) may also place value on these assets and try to abuse them. Threat agents therefore rise threats that also increase the risks of an asset. The 'good' stakeholders try to implement countermeasures intended to reduce this risk to an acceptable level [6].

Risk management is used to identify and prioritize these security risks. Various implementations and instructions have been published for easing the integration process [7, 8]. The left part of Figure 1 illustrates a simplified version of the process according to ISO/IEC 27005:

First, the borders and criteria for risk evaluation are defined (Context Establishment). Subsequently, risks are collected by identifying all assets and threats their threats (Risk Identification). In this process, an asset is not only hardware or software, but could also be a business process or information. The next step is to estimate or rate the identified risks (Risk Estimation). This can be done qualitatively (e.g., low to high) or quantitatively (e.g., amount of cash losses). Based on the risk level identified in the estimation, risks can be assessed and prioritized (Risk Evaluation). The decision about how to handle the risk can now be made (Risk Treatment). A risk can be accepted (e.g., the risk level is very low), reduced (e.g., by a specific measure), avoided (e.g., the cause is eliminated) or transferred (e.g., an insurance). When all risks are treated satisfactorily, an iteration of the process is carried out (Risk Acceptance)

2.2 Risk Assessment

Generally, it is important to rate security risks of a system regarding their criticality in order to prioritize them. Some risks may need in-depth investigation, while others do not need to be considered at all because of their small probability. As a result of this range many frameworks and metrics have been introduced for different domains [9, 10, 11, 12, 13]. In essence they all follow a similar risk assessment process but vary with respect to the estimation criteria to fit the specific domain. In general, they express risk as product of probability and the possible impact of a threat. While probability is often hard to calculate (QUIRC [10], for example, uses statistics of common attacks in the internet), the impact can be calculated by assigning relative values to security properties for every asset. Such 'standard' properties as confidentiality, integrity and availability are commonly used. Depending on the domain, some approaches add additional properties such as legal aspects or safety impacts. Additionally, some methodologies extend the quantification by taking into account asset dependencies to refine the metrics [14] [15]. However, all presented approaches target either the system or the organizational level. While some methods (e.g., [16] with 'asset containers') take the asset environment into account, none of them systematically targets software vulnerabilities in a specific component.

In the domain of software development different methodologies such as Microsoft's DREAD [4], Common Vulnerability Scoring System (CVSS) [17] and OWASP Risk Rating Methodology [18] have been introduced. These focus on quantifying threats to software components that may arise by exploiting possible vulnerabilities. Similar to other risk assessment methods, all these methods generate a rating by combining (i.e., add or multiply) different weighted factors. Here, these factors may also contain properties such as the level of difficulty in finding a vulnerability or how many users would be affected after a successful exploit. These methods are suitable for in-depth analysis of critical components.

This work uses the output of the system wide risk assessment methods to identify software components that are crucial for important assets. Software/security engineers are able to perform threat analysis on these components and the results are integrated in the overall risk assessment process.

2.3 Component Isolation

As a general rule, components should not be allowed to access assets that are not needed for the component function. Therefore, different technologies have been introduced that enforce this so called *principle of least privilege* [19]. Isolation-based access control methods provide each confined application with its own set of resources [20, 21]. Similar results can be achieved by using virtualization [22]. Rule-based access control methods do not rely on an own copy of resources, but confine the access directly based on a policy (e.g., *SeLinux* [23] or *AppArmor* [24]). Architectural approaches such as Multiple Independent Levels of Security (MILS) use similar separation techniques together with controlled information flows to form architectures that target composable assurance [25]. Our work supports such technologies by the systematic identification of trust domains and the assets each domain needs to access and protect. Moreover, the information flows needed between these do-

mains are revealed.

3. ASSET-BASED COMPONENT RATING

The aim of this work is to classify single software components or sets of software components with regards to their privileges or permissions as input for further risk assessment or threat modeling tools. This section provides an overview of the proposed metric and rating methodology and how its integrated into a risk assessment process.

3.1 Process Overview

In order to illustrate how our methodology could be integrated into a systems-engineering process, we use a simplified system development model that consists of system-architecture, software architecture, implementation and subsequently test and verification stages for all previous levels.

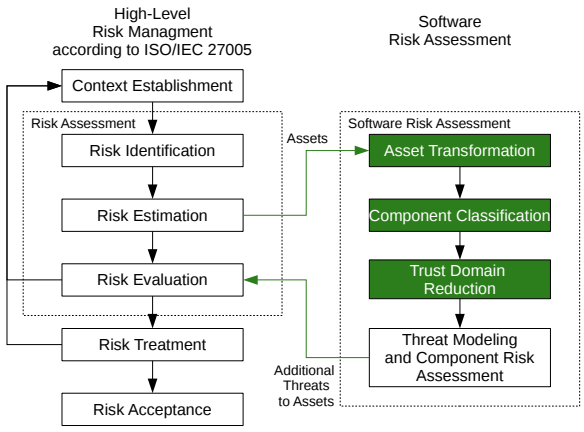

Figure 1: A simplified risk management process according to ISO/IEC 27005 [5] (left), and how our approach is used to generate additional possible threats to assets that may originate from vulnerabilities in software components.

Using the system model, it is possible to identify and quantify all assets at this level by applying one of the methodologies described above. Figure 1 illustrates, how our approach fits into the standard risk management process. After all assets and their risk ratings are identified, the assets are mapped to the software architectural model. Here, the components are classified and optimizations regarding trust domains (or trust boundaries) can be performed. Based on the classification, additional assessment methodologies such as threat modeling can be prioritized. The output of this sub-process comprises additional threats to the assets that can be used for further evaluation.

3.2 Asset Mapping

The upper part of Figure 2 shows an exemplary output of the risk estimation step: A rated list of dependent assets. When generating the software architectural model(s), either a subset or all of these assets are mapped to resources or software components. An information asset, for example, maps into a data resource (e.g., a file or a database), while a critical business function maps into a software component (e.g., a bank transfer). This mapping is illustrated in the lower part of Figure 2. To enable the rating of all components, we use a metric that basically quantifies software components by accumulating the risk ratings of the assets they are able to access directly or indirectly. Cohering parts of the architecture that share the same rating are referred to as trust domains. The edges of these domains are called trust borders.

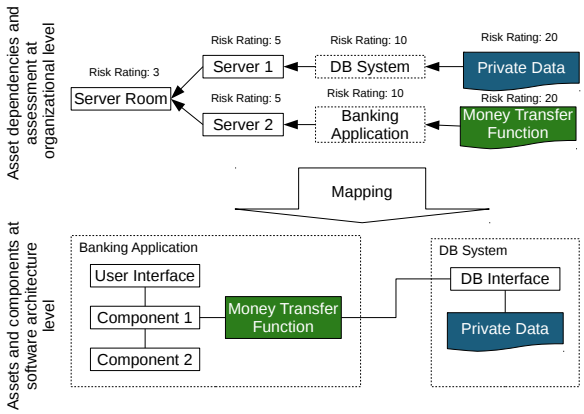

Figure 2: The risk assessment process generated a list of assets, their dependencies and their risk rating. Some of the assets have a counterpart in the different software architectural models. Based on this mapping, the rating of all software components can be calculated.

3.3 Component Classification

3.3.1 System Composition

The metric that is used to quantify components is based on their privileges. Here, a privilege is the possibility of a component to access (i.e., read or modify) an asset. This classification enables an early assessment about the criticality of a component regarding the system's security properties. Components with a higher criticality classification should be considered for a more in-depth analysis. To introduce automated calculations and analysis, a software architecture is modeled as illustrated in Figure 3.

Components. A system is composed of a set of software components. These components may be different processes, libraries or components of one process. Each component accesses a set of assets (by owning specific privileges) and possesses explicit information flow connections to other components. Based on the accessed assets, there may be other, implicit, information flows (e.g., two components are accessing the same file). Each asset represents a resource that has to be protected in some way (e.g., a privacy-sensitive information). A component thus has to have a certain privilege to access the asset.

Privileges. A privilege is the possibility of a component to access a resource and is composed of a resource type and an access mode. Currently, we distinguish between *Data*, *Network* and *Service* privileges. However, depending on the system, there may be many other privilege types for accessing shared resources or hardware like sensors or actuators.

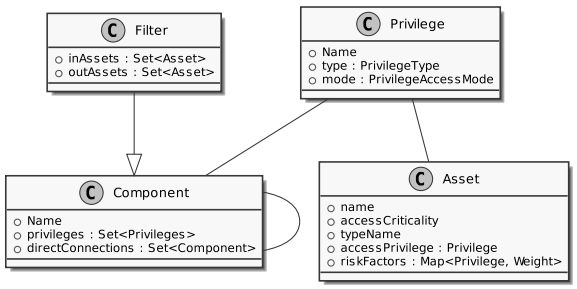

Figure 3: The basic view on software systems: Different components are interacting with each other and have access to different assets. To enable these accesses, privileges are needed. Moreover, there exist special components that are in charge of protecting security properties of critical assets.

A set of access modes exists for each type. For example *Data* privileges may enable access to different type of data (like privacy-sensitive or system) in different modes (read, write).

Assets. Each asset represents a critical resource that has to be protected. The *accessCriticality* reflects the relative 'value' that has been identified at a higher level. Currently, we are using a scalar value that represents the impact of a violation of security properties. However, for a more fine-grained view on the system, it would be easily possible to use a vector with different properties here. Different types of assets requires different types of protection mechanisms. Therefore, there exists an *accessPrivilege*, accessing components have to own. Moreover, there may exist privilege-combinations that raise the criticality of the component that accesses an asset. A component that accesses sensitive information requires more care if it also has access to the internet. In order to represent this increased level of criticality, an asset contains a set of *riskFactors* that map additional privileges by weightings.

Filter Components. A filter component is a special type of component that does not propagate specific or any privileges. Formally, a filter component is a transformation of a set of assets to another set of assets. An authenticator, for example, transforms the asset 'all data' to 'data of a specific user'. Cipher components transform the assets 'confidential data' and 'encryption key' to 'encrypted data'.

3.3.2 Privilege Rating

In order to generate an early estimation of the possible risks of vulnerabilities in one component, we calculate a privilege rating (PR) for each component. In this work, we use the commonly used risk model that expresses risk as product of probability and impact. A vulnerability of a component that accesses a more critical asset may generally have a higher impact on the system's overall security properties. Therefore, the privilege rating is used as a factor for the impact and is thus directly proportional to the risk. Probabilities of successful attacks have to be examined in a later step with methodologies such as threat modeling.

Component Rating. Each privilege P enables a component C access to an asset A. Since similar privileges may enable access to different assets, we do not directly rate the privileges but use the *accessCriticality* ($Crit(A)$) property of the accessed asset. This property is a numeric value, where higher value means a more critical or more 'important' asset. Moreover, each asset contains weighted *riskFactors*. For each of the component's privileges that is contained in this list, the risk factor is increased by the weight ($RF(A,P)$). Therefore, the overall privilege rating of a component $PR(C)$ is generated by $Crit(A)$ of all accessed assets and the sum of all active risk factors.

$$PR(C) = \sum_{A=Assets(C)} \Big(Crit(A) + \sum_{P=Priv(C)} RF(A,P) \Big)$$

Component Compositions. Whenever two components A and B are connected via an information flow, the privileges of the components are merged. This is a rough generalization only, however, due to of the following problems:

1. A directed information flow may not allow the sharing of privileges in both directions.
2. Some components may not allow access to their privileges at all or only in a restricted manner.

Problem (1) is not faced in this work, because it requires a more detailed model of information flows and privilege types and is part of ongoing work. Problem (2) is solved with filter components.

3.4 Trust Domain Reduction

Components that share their privileges are part of the same trust domain. In order to reduce the attack surface, the size of trust domains with a high risk should be minimized. Therefore, the software and/or security architect is able to introduce filter components, which are able to transform assets regarding their criticality. An authenticator in the 'DB System' in Figure 2, for example, may reduce the asset 'all private data' to 'data of a specific user'. A filter component thus separates these domains and introduces a trust border. By re-applying the metric, the effect is reflected instantaneously in the architectural model and the software architect is able to iterate this step until the trust domains are acceptable in terms of size and risk.

3.5 Threat Modeling

Now, a list of software components with high criticality, as well as components that are in charge of protecting high risk assets (i.e., filter components on trust borders) can be generated. Based on this list, it is possible to prioritize components that should be taken into account for in-deep risk analysis and threat modeling. This analysis identifies new threats (or threat-tree-branches) for assets that can be integrated into the high-level risk management process.

4. USE CASE AND EVALUATION

In order to examine the feasibility of the privilege rating, we are using the methodology to analyze an actual software architecture. We also implemented the tools needed

to describe and visualize a software architecture in the way described in Section 3, also the algorithms that calculate the privilege rating and trust borders. Based on a high level risk assessment on the system assets, we apply the privilege rating metric to identify critical domains in the architecture that need further investigation. Moreover, based on the classification, we identify component interconnections that should be filtered to provide privilege separation.

4.1 Evaluated System

A manufacturing system is used by a company that developed and sells an embedded control system (vendor) to manage the production process. This process is distributed among different manufacturing companies (manufacturers). Each manufacturer receives a test equipment unit, an embedded device that is used to lead the manufacturing process. This process includes product assembling and integration tests. All manufacturing entities are connected to a central database (server), where test and production results are stored. The vendor is able to place orders and review the production data, for example to generate statistics of calibration data. Since critical information like encryption keys are generated and distributed during the process, security is a key concern here. In this section, the central server is evaluated.

In order to evaluate the system, we implemented a tool that uses textual representation of the system to generate its model and visualization. Figure 4 shows the overall system: A web-application provides a service that enables the vendor access to all test data. Each test equipment accesses the database by using the test interface. However, this interface only provides the sub-set of the information for Digital Rights Management (DRM) reasons. For the sake of simplicity, we designate all manufacturers, as well as the vendor 'users' here. Each user is only allowed to access its own data. Therefore, credential-based authentication is used. Moreover, the vendor is backing up all data to a physically separated backup server via a private network connection.

4.2 Asset Mapping

Based on high-level risk assessment, three data assets (Credentials, Manufacturing data and Common data), as well as two network assets (WAN and LAN) have been identified and evaluated (Table 2). In order to access these assets, two privilege types must be defined (shown in Table 1) and the privileges must be assigned to the direct connected components. The resulting privilege rating of all components is shown in Table 3. The assets and their criticality have to be added to the model manually to enable further computations.

Table 1: Privilege types and their corresponding access modes

Name	Access Mode
Network	WAN, LAN
Data	Credentials, Manufacturing, Specific, Test Data, Common

4.3 Component Classification

The architecture shown in Figure 4 is the first draft that is used as input for the risk assessment. By analyzing the

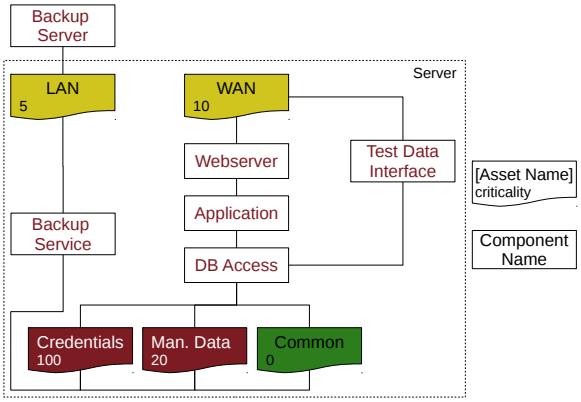

Figure 4: Use-Case: The central database of a distributed manufacturing and test system. One user is able to place orders and to access test data via a web application. Test entities from different manufacturing companies are accessing the server via the more restrictive test interface. To improve readability, the privilege rating (criticality) of the different components is encoded in colors (green: <=5; yellow: <=50; red: >50). The actual values are shown in Table 2 and 3.

Table 2: Use-Case assets

Name	Crit(A)	Risk Factors
Credentials	100	Network(WAN), 10
Manufacturing	10	Network(WAN), 5
Common	0	
LAN	10	Network(WAN), 2
WAN	10	
User-Specific Data	5	Network(WAN), 2
Test Data	5	Network(WAN), 2

information flows, our tool calculates the privilege ratings of all components. Currently, there is only one privilege domain (there is an implicit information flow between backup service and database access). Therefore, all privileges are shared among all components (in the figure, the rating is encoded as color; numeric values are shown in Table 3). This does not mean that every component is able to access all assets per-se, but vulnerabilities in any component may have a critical impact. In order to obtain useful information, the architecture must thus be refined with filter components to separate the privilege domains.

4.4 Trust Domain Reduction

In order to achieve the separation into smaller privilege domains, two virtual assets are introduced: *User-Specific Data* and *Test Data*. Moreover, three filters must be added to the model manually:

- User-Specific Filter: This filter component implements methods that prevents access to all user data that is not owned by the currently authenticated user. It therefore reduces a *Data(User)* privilege to a *Data(User-Specific)* privilege.
- Test-Data Filter: This filter blinds all data that is not intended to be provided via the test interface. Here the

reduction is from *Data(User-Specific)* to *Data(Test-Data)*.

- Authenticator: This filter implements authentication. The filter prevents other components from reading credential information. It only returns whether the authentication succeeded. For simplification, here this information is considered harmless and no privileges are needed to access it.

- Network Filter: This filter prevents the internal components (Domain 1) from accessing the WAN-Port. Moreover, it protects the LAN domain from access of external components. This filter could be implemented with a firewall.

4.5 Evaluation

Figure 5 shows the system after adding these filters. The recalculation is done automatically and the resulting system contains three trust domains with different privileges. The numerical values of the privilege ratings are shown in Table 3.

- Domain 1: This domain has full access to the underlying data. Therefore, special care should be taken for these components in the threat analysis process. The backup-part should not be accessible for anyone and the authenticator and the filter for user-specific data should be designed and reviewed carefully.

- Domain 2: This part handles user-specific data and is accessible through the internet. Although it is not as critical as the components of Domain 1, in-depth thread modeling should be considered.

- Domain 3: The test interface has relatively few privileges. It is only able to handle a subset of the currently authenticated user data. Therefore, this component has the weakest requirements regarding security.

In general, the overall criticality of the components is reduced drastically. Of course, this reduction mainly originates from our asset ratings and relatively high weights for risk factors. However, we can see that the number of critical components is reduced and we are able to focus effort for actual threat modeling and countermeasures in a prioritized manner. Based on the results of the threat modeling process, new threats to assets that originate in vulnerabilities of specific software modules are revealed in a systematic manner. This information supports the overall risk management process (e.g., by completing a threat tree for an asset) and eases decision regarding resource allocation for threat treatment strategies.

5. CONCLUSION AND FUTURE WORK

In this work, we introduced a risk assessment method for software components based on assets they are able to access. These assets are identified on a system or organizational level and mapped into the software domain. The classification is based on the architectural component model and its dataflow relations. This enables the possibility to connect the risks of these high-level assets to software components.

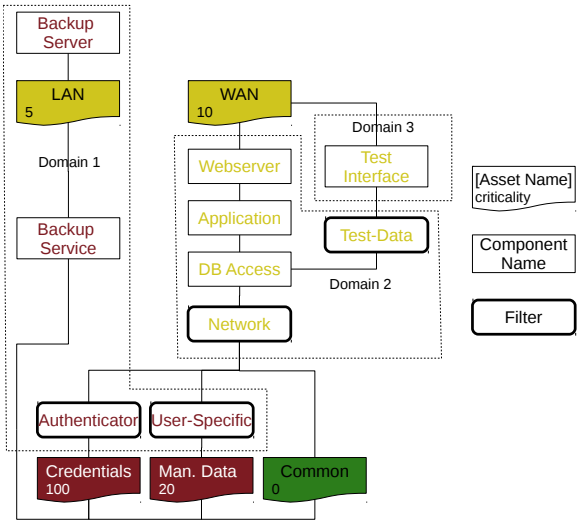

Figure 5: Trust domains after inclusion of filter components.

Table 3: Component criticality before and after the introduction of filter components

Component Name	w/o Filter		with Filter	
	Domain	Criticality	Domain	Crit.
Webserver	0	1120	2	10
Application	0	1120	2	10
DB Access	0	1120	2	10
Test Interface	0	1120	3	8
Backup Service	0	1120	1	125
Backup Server	0	1120	1	125
Authenticator	-	-	1	125
User-Specific	-	-	1	125
Test-Filter	-	-	2	10
Network-Filter	-	-	2	10

Therefore, critical components that are in charge of protecting the assets can be easily identified. Moreover, architectural regions with similar risk (trust domains) are identified and the impact of implementation of filter components is instantaneously reflected in the model. We showed how the approach is adapted in a real-world scenario and helped us to identify critical sections of an architecture, to reduce the attack surface and to implement security measures at all the right places.

Some tasks remain to be done future work. One major goal is the automated insertion of filters to optimize trust domains based on their size and risk rating. To achieve this, additional information such as the users of the system and the assets they need to access will need to be reflected in the model. Based on a comprehensive data flow model and parameters that describe desirable results (e.g., small trust domains), different optimization strategies could be used to identify optimal filter placement. This automation is also desirable because it would be a step towards an automated generation of information flow- and isolation-policies based on the model of a software architecture and the system-level assessment of assets.

The prototype implementation of the model description lan-

guage is sound enough to evaluate the approach and investigate different architectures. However, in order to simplify the integration into existing processes, we are considering implementing the risk assessment and metric approach into existing model-driven security UML-extensions for software architecture. Other possible extensions are concerning the metric. Basically, not all components in one trust domain should be rated with the same risk. Components on edges between different trust domains (i.e., filters) should be given more attention, because these are the components potential adversaries are able to interface with. Moreover, the risk rating is currently only represented by one scalar impact factor. In order to enable a simpler adoption for other domains, we are planning to work out this gap and allow the usage of a dynamic set of security properties.

6. REFERENCES

[1] Barry Boehm. Software risk management: principles and practices. *IEEE Software*, 8(January):32–41, 1991.

[2] Charles B. Haley, Jonathan D. Moffett, Robin Laney, and Bashar a. Nuseibeh. A framework for security requirements engineering. *Proceedings of the 2006 international workshop on Software engineering for secure systems - SESS '06*, page 35, 2006.

[3] Suvda Myagmar. Threat Modeling as a Basis for Security Requirements. *In StorageSS '05: Proceedings of the 2005 ACM workshop on Storage security and survivability*, pages 94–102, 2005.

[4] Frank Swiderski and Window Snyder. *Threat Modeling*. Microsoft Press, 2004.

[5] International Organization for Standardization (ISO). ISO/IEC 27005:2008 - Information technology - Security techniques - Information Security Risk Management, 2008.

[6] International Organization for Standardization (ISO). Information technology - Security techniques - Evaluation Criteria for IT Security - Part 1, 2009.

[7] The Open Group. *Technical Guide FAIR – ISO / IEC 27005 Cookbook*. The Open Group, 2010.

[8] Alexander Leitner and Ingrid Schaumüller-Bichl. Arima - A new approach to implement ISO/IEC 27005. *2009 2nd International Symposium on Logistics and Industrial Informatics, LINDI 2009*, pages 1–6, 2009.

[9] J Samad, S W Loke, K Reed, and Ieee. Quantitative Risk Analysis for Mobile Cloud Computing: a Preliminary Approach and a Health Application Case Study. *2013 12th Ieee International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1378–1385, 2013.

[10] P. Saripalli and B. Walters. QUIRC: A Quantitative Impact and Risk Assessment Framework for Cloud Security. *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 280–288, 2010.

[11] Georg Macher, Harald Sporer, Reinhard Berlach, Eric Armengaud, and Christian Kreiner. SAHARA: A Security-Aware Hazard and Risk Analysis Method. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 621–624, 2015.

[12] Nikos Vavoulas and Christos Xenakis. A Quantitative Risk Analysis Approach for Deliberate Threats. In *International Workshop on Critical Information Infrastructures Security*, pages 13–25, 2010.

[13] Cristian Ruvalcaba and Chet Langin. Whitepaper: Threat Modeling: A Process To Ensure Application Security. 2009.

[14] Jakub Breier and Frank Schindler. Assets Dependencies Model in Information Security Risk Management. In *International Conference on Information and Communication Technology*, pages 405–412, 2014.

[15] Bomil Suh and Ingoo Han. The IS risk analysis based on a business model. *Information & Management*, 41(2):149–158, 2003.

[16] Richard Caralli, James Stevens, Lisa Young, and William Wilson. Technical Report: Introducing OCTAVE Allegro : Improving the Information Security Risk Assessment Process. Technical Report May, Software Engineering Institute, 2007.

[17] Peter Mell, Karen Scarfone, and Sasha Romanosky. The Common Vulnerability Scoring System (CVSS) and Its Applicability to Federal Agency Systems. *NIST Interagency Report 7435*, 2007.

[18] OWASP. OWASP Risk Rating Methodology.

[19] J.H Salzer and M.D Schroeder. The Protection of Information in Computer Systems. In *Proceedings of the IEEE*, pages 1278 – 1308, 1975.

[20] Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. *2009 30th IEEE Symposium on Security and Privacy*, pages 79–93, May 2009.

[21] Li Gong, Marianne Mueller, H Prafullchandra, and R Schemers. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, number December, 1997.

[22] Andrew Whitaker, Marianne Shaw, and SD Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In *In Proceedings of the USENIX Annual Technical Conference*, number Figure 1, 2002.

[23] NSA Peter Loscocco. Integrating flexible support for security policies into the Linux operating system. In *USENIX Annual Technical Conference*, 2001.

[24] Crispin Cowan, Steve Beattie, Greg Kroah-Hartman, Calton Pu, Perry Wagle, and Virgil Gligor. {SubDomain}: Parsimonious Server Security. In *USENIX LISA*, pages 1–20, 2000.

[25] Holger Blasum, Sergey Tverdyshev, Bruno Langenstein, Jonas Maebe, Bjorn De Sutter, Bertrand Leconte, Benoît Triquet, Kevin Müller, Michael Paulitsch, Axel Söding-Freiherr von Blomberg, and Axel Tillequin. Whitepaper: Secure European Virtualisation for Trustworthy Applications in Critical Domains. 2013.