The Architectural Security Tool Suite — ArchSec

Bernhard J. Berger
Software Engineering Group
University of Bremen, Germany
bernhard.berger@uni-bremen.de

Karsten Sohr
Technologie-Zentrum Informatik
und Informationstechnik, Germany
sohr@tzi.de

Rainer Koschke
Software Engineering Group
University of Bremen, Germany
koschke@uni-bremen.de

***Abstract*—Architectural risk analysis is a risk management process for identifying security flaws at the level of software architectures and is used by large software vendors, to secure their products. We present our architectural security environment (ArchSec) that has been developed at our institute during the past eight years in several research projects. ArchSec aims to simplify architectural risk analysis, making it easier for small and mid-sized companies to get started.**

With ArchSec, it is possible to graphically model or to reverse engineer software security architectures. The regained software architectures can then be inspected manually or automatically analyzed w.r.t. security flaws, resulting in a threat model, which serves as a base for discussion between software and security experts to improve the overall security of the software system in question, beyond the level of implementation bugs.

In the evaluation part of this paper, we demonstrate how we use ArchSec in two of our current research projects to analyze business applications. In the first project we use ArchSec to identify security flaws in business process diagrams. In the second project, ArchSec is integrated into an audit environment for software security certification. ArchSec is used to identify security flaws and to visualize software systems to improve the effectiveness and efficiency of the certification process.

***Index Terms*—security, threat modeling, software architecture, reverse engineering, architectural risk analysis**

I. Introduction

Software security is a topic of growing importance in today's software development. The number of legal regulations regarding security and especially privacy is growing. The first step companies undertake is to employ security bug finders that identify issues at the implementation-level. These tools identify—depending on the analyzed programming language—SQL injections, buffer overflows, or command injections. All these issues have in common that they are related to missing input validation or the use of insecure programming interfaces.

With the growing security-awareness of companies they face the challenge of identifying architectural security flaws. These flaws are of great importance since they can undermine the whole security concept of an application. They are, however, harder to identify since it requires more security knowledge to find and understand the underlying architectural problem. At this level, security experts look at authorization policies, processing of sensitive information, or the adequate use of cryptography.

In our previous and recent research projects we worked with various small and midsized enterprises (SMEs) that wanted to start with architectural security analysis to improve their applications' security. They all were lacking security experts in their development teams. A common solution to this problem is to consult an external security expert with the idea that this person is able to secure the software within a short period of time. The software systems they have to audit are complex and have been implemented over years comprising decades of man-years of development effort. In many cases these experts lack the time to take a closer look at the software and to extract a comprehensive software architecture to conduct a proper architectural risk analysis. Mostly these consultants focus on analyzing design documents or development processes to identify security pitfalls. Even for certifications, such as the Common Criteria, it is very seldom that an auditor manages to analyze at source code level.

These reasons were the motivation to develop an architectural risk analysis environment that allows these companies to start with architectural risk analysis more easily. We identified several requirements that arose from discussions with our application partners:

- Easy and extensible way to manually create software architecture diagrams.
- Means to extract software architecture diagrams automatically using static analysis.
- Query support for architecture diagrams to identify interesting system parts.
- Knowledge base containing rules to identify architectural flaws automatically.

Within this paper we present the ArchSec environment that we started to develop in 2010. We describe the main use cases of the environment and give a glance at implementation details. In the evaluation we describe how we employ ArchSec in two of our current research projects. In the first project we use it to check business processes for architectural security flaws. In the second project we integrate the ArchSec environment into an audit tool to support auditors in understanding unknown software systems and identifying security-relevant parts.

This work was supported by the German Federal Ministry of Education and Research (BMBF) under the grants *16KIS0583* (PortSec project), *16KIS0069K* (ZertApps project) and *01IS10015B* (ASKS project). Furthermore it was supported by the German Federal Ministry for Economic Affairs and Energy (BMWi) under the grant *ZF4123903ED6* (CertifiedApplications).

II. Background

The process of analyzing software architectures for security flaws has been introduced by different parties. The term *architectural risk analysis* has been prominently promoted by Gary McGraw [1]. A similar concept was introduced by Microsoft with their threat modeling approach [2] as part of their Security Development Lifecycle [3].

An important part of these approaches is to create a software architecture sketch and to discuss potential security flaws with the help of this diagram. Both approaches are asset-centric and try to protect these assets from illegal accesses. Furthermore, they are attacker-centric and attempt to see the software system not from the user point of view. Neither architectural risk analysis nor threat modeling specify how to write down software architectures. Microsoft suggests a very lightweight and informal description formalism named data flow diagrams. There are different reasons for not using common description formalisms such as UML. First of all, describing software systems with UML is highly complex and secondly, they are not widely used in the security community.

The data flow diagrams used by Microsoft resemble directed, labeled, and hierarchical graphs. The only difference is the additional concept of trust boundaries. Trust boundaries divide a software system into areas of different trust levels, where elements can belong to (or not). Figure 1 shows an exemplary data flow diagram, consisting of four elements, two channels (data flows) and one trust area.

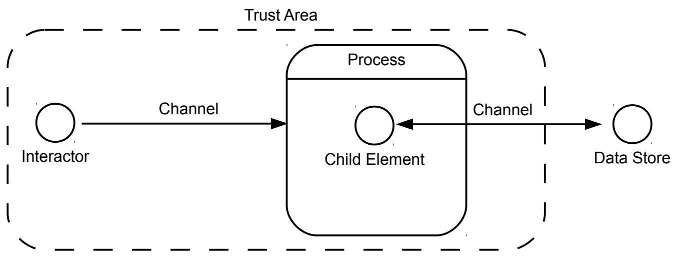


Fig. 1. Exemplary Data Flow Diagram

Dhillon stated that it was helpful to append attributes to the different elements of a data flow diagram to add information to them [4]. Berger et al. already defined extended data flow diagrams (EDFD) in a previously published paper [5]. Extended data flow diagrams are additionally typed and attributed and have data concepts. The attributes make it possible to attach security objectives and existing mitigation strategies, such as encryption, to EDFD elements. The data concepts allows analysts to define assets of software systems. The types can be associated with attributes that are implied by the type. An example is a channel of type *HTTPS Channel* that implies the attribute *Transport Encryption*. This way it is possible to allow non-experts to create diagrams and the types automatically add security information. The sum of all types of an extended data flow diagram is called schema. A flexible schema allows diagram authors to add new kinds of types and attributes at runtime, enabling them to customize EDFDs to their needs.

III. Related Work

Despite the number of static analysis approaches for software security, little work has been done on the problem of reconstructing and visualizing the security architecture of software systems. One exception is the technique introduced by Abi-Antoun and Barnes [6]. They annotate the source code of an application to statically extract Ownership Object-Graphs statically. Ownership Object-Graphs represent a hierarchical runtime-architecture of the objects within a system. Furthermore, they compare the extracted graph with a given DFD to identify forbidden data flows. The approach has only been tested for small and non-distributed applications (about 3,000 lines of code) [2]. Almorsy et al. [7] use UML models and OCL signatures to pinpoint architectural security flaws. They use reverse engineering to extract UML class diagrams from code but do not employ static analysis techniques, such as data flow analysis.

There is work that deals with architectural risk analysis without taking the program code into account. Microsoft provides a tool that supports the Threat Modeling process [2]. This tool makes available a catalog of questions, which an analyst can apply to a given DFD. Schaad and Borozdin applied Microsoft's Threat Modeling to block diagrams and identified possible threats and vulnerabilities introduced by the usage of third-party standard software components [8].

Other works' approaches use UML and related tools to analyze security architectures [9], [10]. These UML-based approaches let a software architect formulate security requirements that a software architecture must satisfy, e.g., authorization or confidentiality requirements. UML and its constraint language OCL were introduced to allow specification of positive system requirements rather than anti-requirements (things that can go wrong). In contrast, we utilize security knowledge about architectural software defects as, for example, listed in the Common Weakness Enumeration (CWE). Hence, our technique complements UML-based approaches to architectural risk analysis.

Additionally, work has been done on software visualization and program comprehension. Bauhaus is a tool suite that has been developed some years ago for extracting and visualizing resource flow graphs. Resource flow graphs are typed and attributed graphs that contain the entities, such as methods and classes, and their static dependencies [11], [12]. The approach was used by Berger et al. to conduct a security assessment for some aspects of Android's security concepts [13].

The presented ArchSec tool is based on the principles of our earlier work [5], [14]–[16]. While the first approaches were explicitly developed for analyzing Android apps, we significantly extended and redesigned it over the years. ArchSec is now split into core concepts, such as the data flow diagrams, corresponding textual and graphical editors, a knowledge base format, a rule checking engine, a reverse engineering framework, and the threat model editor. Some of the new parts are visible in Figure 3. All framwork- or language-specific code was extracted into additional plugins to make the core more flexible and robust.

IV. Implementation

A binary release, in the form of an Eclipse update site, can be found on the ArchSec website [17]. An overview of ArchSec's use cases is depicted in the use case diagram in Figure 2, which concentrates on the use cases that are interesting for an auditor during an assessment. An auditor wants to analyze a software system; therefore, she creates an extended data flow diagram. She first selects a schema for the resulting EDFD and then has the possibility to either create the EDFD by hand or to convert an existing software system into an EDFD. After creating an EDFD the auditor applies a knowledge base in order to create a threat model. Depending on the rules to be checked she either applies built-in rules or software specific ones, which she has to define first. Now, she can inspect the resulting threat model.

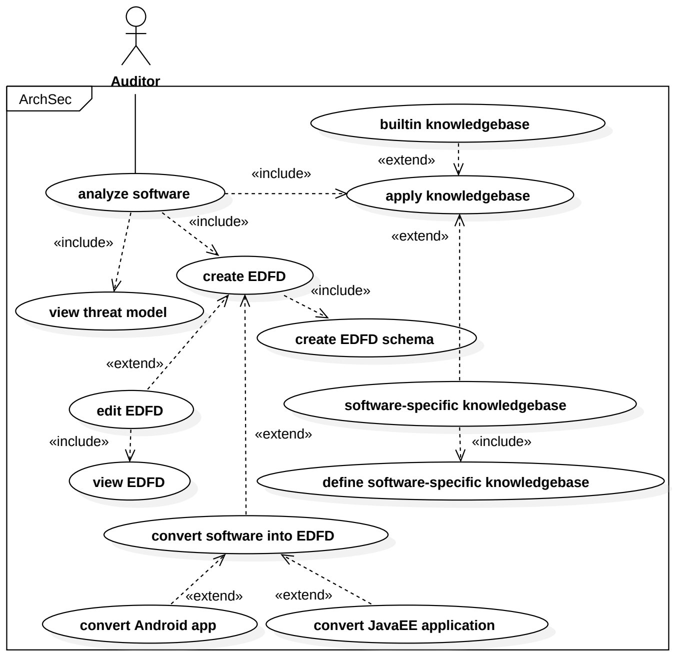


Fig. 2. UML Use Case Diagram of ArchSec

The ArchSec environment is integrated into the Eclipse platform. It uses of the model-driven software development paradigm and is based on the Eclipse modeling framework. We are using models for our intermediate representations, to implement domain-specific languages, model-to-text and model-to-model transformations. Furthermore, there are libraries for developing visualizations and editors based on these models.

For all the aforementioned use cases there is tool support in ArchSec. In the following, we highlight some of them. Our reverse engineering pipeline for extracting EDFDs benefits of a framework we wrote by allowing us to switch between locally or remotely executing the analysis through a simple change in the configuration setting. If the analysis is to be conducted remotely, the analysis input as well as the analysis description are written to a database. Several analysis nodes frequently poll this database for new analysis tasks. Unprocessed tasks are then fetched and executed by installing the according analysis from our analysis repository. The results are written back to the database where from the user interface can fetch the results. This approach allows us to prepare evaluations with a larger number of analyses by simply pushing them to our analysis database and let different analysis nodes process them in parallel. The local execution simply creates a new analysis process that calculates the results.

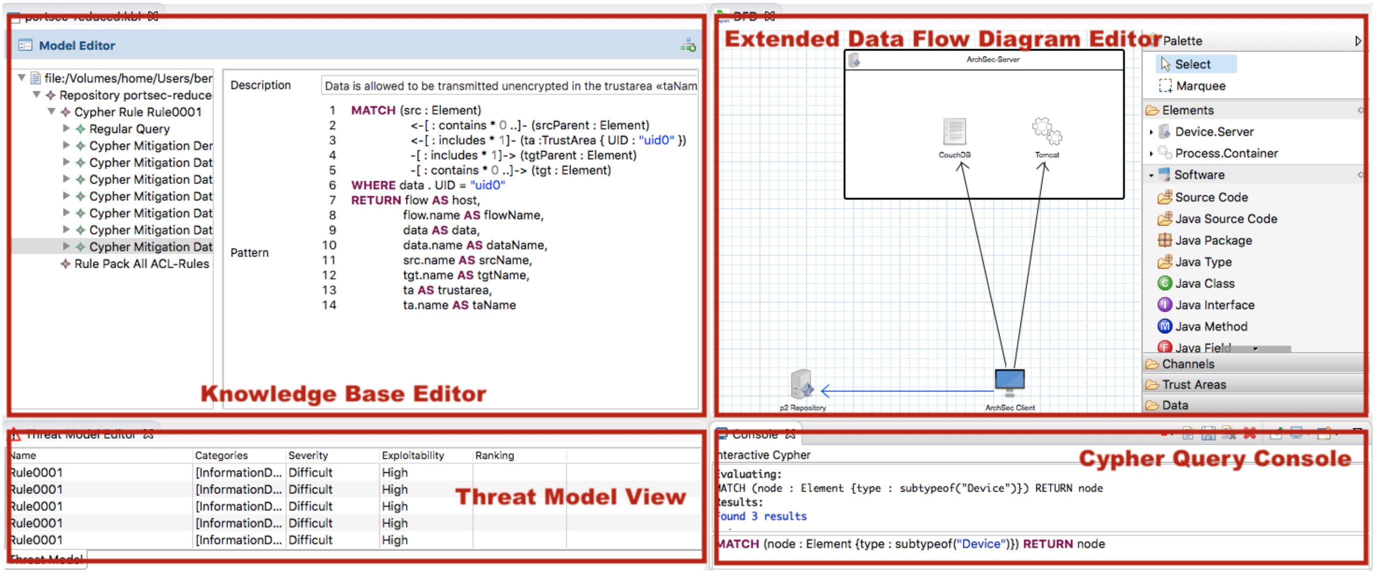
Our analyses for extracting security and architecture facts are implemented as a set of plug-ins for the Soot analysis framework [18]. Soot is a major framework for Java and Android. It works on Java and Android bytecode, making it possible to analyse software systems without requiring access to their source code. This is especially helpful with commercial partners that are reluctant about handing over their source code due to security reasons. Soot has a large ecosystem of additional libraries, allowing one to implement intra- and inter-procedural analyses. However, ArchSec's infrastructure could be used for other analysis frameworks as well and they could be used to enrich EDFDs.

For viewing and editing extended data flow diagrams, we implemented a flexible graph editor based on Eclipse's Graphiti framework. The framework allows one to store the visualization of a model and the model itself into separate files and is well-suited for creating a graphical depiction of models. The editor allows a user to view and manipulate all information present in an extended data flow diagram.

We use Xtext [19], a framework for creating domain-specific languages, to implement Cypher, a graph query language [20], which we use for describing knowledge base rules,. Each knowledge base rule consists of a cypher query describing a security anti-pattern, a situation where the security requirements of an application are violated. Furthermore, it contains queries describing a possible mitigation of the security flaw. These patterns correspond to security patterns that allow the system to fulfil the security requirements.

The knowledge base itself can be created with the help of a form-based editor. This editor allows security experts who are used to extended data flow diagrams to create software-specific knowledge base rules on their own. These application-specific rules are important to easily support domain-specific security requirements. When a knowledge base is applied to an EDFD, the rules found in the knowledge base are evaluated for the EDFD; every match results in a threat model entry. Currently, we do not employ a graph database, such as Neo4J, to find matches since it would make it necessary to serialize our models into a database to query it. Therefore, we implemented our own Cypher evaluator within ArchSec.

Figure 3 shows a screen shot of the ArchSec environment. The knowledge base editor is visible in the upper left corner. It is used to edit a mitigation pattern, consisting of a description and the pattern itself. The description of rule or mitigation is used to generate an explanatory text for the threat model and is customized with the concrete element names or attribute values of a finding. On the upper right corner an EDFD is opened



file:/Volumes/home/Users/ber
Repository portsec-reduce
Cypher Rule Rule0001
Regular Query
Cypher Mitigation Der
Cypher Mitigation Dat
Cypher Mitigation Dat
Cypher Mitigation Dat
Cypher Mitigation Dat
Cypher Mitigation Dat
Cypher Mitigation Dat
Rule Pack All ACL-Rules

Description: Data is allowed to be transmitted unencrypted in the trustarea «taNam

Pattern:

```
MATCH (src : Element)
        <-[ : contains * 0 ..]- (srcParent : Element)
        <-[ : includes * 1]- (ta :TrustArea { UID : "uid0" })
        -[ : includes * 1]-> (tgtParent : Element)
        -[ : contains * 0 ..]-> (tgt : Element)
WHERE data . UID = "uid0"
RETURN flow AS host,
       flow.name AS flowName,
       data AS data,
       data.name AS dataName,
       src.name AS srcName,
       tgt.name AS tgtName,
       ta AS trustarea,
       ta.name AS taName
```

Name	Categories	Severity	Exploitability	Ranking
Rule0001	[InformationD...	Difficult	High	
Rule0001	[InformationD...	Difficult	High	
Rule0001	[InformationD...	Difficult	High	
Rule0001	[InformationD...	Difficult	High	
Rule0001	[InformationD...	Difficult	High	

```
nteractive Cypher
valuating:
ATCH (node : Element {type : subtypeof("Device")}) RETURN node
esults:
ound 3 results
```

```
MATCH (node : Element {type : subtypeof("Device")}) RETURN node
```

Fig. 3. Screen Shot of ARCHSEC

for editing. The graph is on the left hand side— whereas the palette is located on the right hand side containing all available types. At the bottom left of the depiction a threat model is opened for further inspection. At the lower right side there is an interactive cypher console, allowing to evaluate cypher queries for opened EDFDs.

V. USE CASES

Currently, we employ ARCHSEC in two different funded research projects dealing with enterprise software. In the PortSec project, we analyze a port community system, the central data exchange platform of a port. In the CertifiedApplications project we integrate ARCHSEC into an environment for security audits to analyze Java Enterprise software and Android applications.

A. Port Community System

The goal of the PortSec project is to understand and improve information security of international ports. In a single port, many different participants exist, such as shipowners, customs, forwarders, and terminals. All the participants have to work together to discharge a ship. Therefore, the individual participant's systems are highly interconnected and exchange much information. Most of the time this information is critical to security. Manipulating or revealing this information may lead to serious damage, ranging from financial consequences to physical damages.

We are automatically transforming business process diagrams, present in the OMG's business process and notations standard, into data flow diagrams to identify security flaws. The main concepts of business process diagrams are activities (tasks that are executed), gateways (decision nodes), swim lanes (different actors of a system), and connections (flow of control). Within the transformation, tasks and gateways are mapped to elements and connections are mapped to channels. Therefore, we extended the built-in schema with new element types for tasks and gateways. This extension takes a very limited amount of time (half an hour, including the search for icons). In the second step, we added the messages that are transmitted along the control flow and classified them according to their security requirements. Afterwards, we set the type of the channels according to their characteristics. Therefore, we used existing channel types, such as *intra-process communication*, *http connection*, *https connection*, and *e-mail traffic*, but also defined new ones like *FTAM*, a file transfer protocol we did not know so far. The adaptation of the diagram with approximately 150 communication channels took about one hour.

Applying our existing knowledge base identified a large number of locations where sensitive data were transferred using insecure channels and therefore exposed to potential attackers. After discussing the results with system experts we found out that some of the systems run within the same ISO 27001 certified data center and therefore can be considered trustworthy. Since it is our goal to focus on external attackers and the data center is trustworthy with regards to that, these findings are false positives. We then extended our transformation to take the swim lanes into account. Furthermore, the software vendor defined a security policy specifying the trustworthiness of trust areas with regard to specific data. We used a model-to-text transformation to automatically generate a system-specific knowledge base that represents the security policy. With the help of the resulting extended data flow diagrams and the system-specific knowledge base, we were able to identify four communication channels within the port community system where data could be eavesdropped on or manipulated by attackers. These findings were real issues that needed to be fixed.

B. Certification of Java Enterprise Systems and Android Apps

The main objective of the CertifiedApplications project is to define a lightweight software certification process. Security certifications often follow a similar pattern: the software vendor prepares a self-assessment, in which she describes the assets, the functions as well as the input and the output of the system. Furthermore, she describes the security properties that she wants to certify and the circumstances under which these properties should hold. Then, the certification company checks the self-assessment for consistency and passes it to a testing laboratory where an auditor has the task to check the software against. Depending on the certification level the accuracy of these checks may differ.

One of our project partners—a company from software certification business—defined a new software security certification process according to different international standards. Within the definition of the process, they focused on a more lightweight certification approach to reduce costs. One way to achieve this goal is to use static analysis-assisted matching of the self-assessment and the implementation and the use of extracted data flow diagrams.

In this use case, ARCHSEC helps to improve the effectiveness and efficiency of the certification process as part of a greater audit environment. It allows auditors to more easily identify the software parts that are of interest and to automatically search for architectural security flaws.

VI. CONCLUSIONS

The security community and many SMEs benefit from automated architectural risk analysis since it simplifies the applicability. Furthermore, it allows an analyst to check the implemented security architecture instead of the planned one. In the following, we give an overview of the lessons we learned, the limitations we found and present our future plans for ARCHSEC. We are going to focus on those facts that are of interest for the reverse engineering community.

A. Lessons Learned

One problem we were facing during analyzing current software systems is the large extent of software frameworks that are used. Starting with component-based frameworks, such as Android and JavaEE, where there is no single entry point to the application and the executing container follows a defined component lifecycle. Similar problems arise from dependency injection frameworks, since they hide the creation and wiring of instances. Thus, it is hard to calculate pointer information, precise call graphs, and inter-procedural data flows. For all these reasons, it is complicated to create analyses for software systems employing these frameworks. It would be really helpful if static-analysis tools provided means to support such frameworks and thus would be able to create call graphs.

Component-based frameworks are often configurable. These configurations can have a major impact on the results of both static and security analyzes, if taken into account. Here, for example, the dependency injection instance used can be uniquely defined, constants can be set in the program code, configured security protocols can be configured, or security providers can be selected. Taking this information into account in the analysis helps us to produce more accurate results.

In the beginning of ARCHSEC we employed OCL to describe matching conditions of security flaws and mitigations (see [5]). After experimenting with a larger set of rules we learned that complex flow-based rules became very complicated to write down and some rules were not expressible in standard OCL. For this reason we switched to an alternative language that was tailored towards queries on graphs.

On employing ARCHSEC we had several discussions about the ease of using it. Many security experts or quality managers do not have the time to learn the usage of complex command line tools. They rather expect production-like quality of the software, integrated into a graphical user interface that works as easy as a push-button tool. This expectation is one reason why we spent a great amount of effort into automating the static analysis, where one can extract an extended data flow diagram by simply right clicking on a software artifact.

Productive systems often use external components to provide security. For example, an application firewall that protects against denial-of-service attacks or an HTTPS proxy that centrally encrypts incoming and outgoing Web server traffic. Such organizational measures are often sources of false positives, as they can not be identified by code analysis. For this it would be necessary to perform dynamic network analyzes in order to incorporate this information.

B. Limitations

In ARCHSEC there is still plenty of implementation work to be done. While the core components are getting more and more stable, there are many possible framework-specific analysis extensions that have not been implemented so far due to the amount of necessary engineering effort.

Additionally, ARCHSEC shows some known limitations. First of all, the process of architectural risk analysis can be aided by reverse engineering. Nevertheless, full automation of this approach is not possible, since with the help of the knowledge database only standardized security flaws can be identified, which usually do not consider the peculiarities of the specific software to be examined. An automated analysis, for instance, cannot decide on its own which data are sensitive or whether a file on disk of a used remote system is secure.

Since the flaw identification is a subgraph-isomorphism problem, the runtime of this step can become quite large in the worst case. This has not yet become a problem in the previous experiments, since the possibilities of matching are greatly limited by the typing of the nodes and edges. In addition, circles in the graph are only considered once by our evaluation engine, which also reduces the number of matches.

C. Outlook

Currently, we are finishing our first stable release of the framework-independent core part. The current build of ARCHSEC is available on our homepage. Nevertheless, there are many possibilities for improvements.

Automatic Graph Layouts Currently the automatic layout possibilities for extended data flow diagrams are very rudimentary. It is desirable to implement automatic graph layouts to make extracted diagrams clearer at the beginning. For this purpose, we plan to integrate the Kieler framework [21].

Extensible Static Analysis Configuration We want to create a common configuration mechanism for all our reverse engineering pipelines, such as Java Enterprise, Java, or Android. At the moment, each reverse engineering pipeline has its own configuration, but it showed that there are many similarities, which can be unified.

Supporting Architectural Views The IEEE standard 42010:2011 suggests to use architectural views to address different architectural concerns [22]. Our experience supports this conjecture because using a single view for all security aspects (e.g., authorization, information flow, cryptographic mechanisms), in form of an extended data flow diagram, can be confusing. Therefore, we plan to add support for extracting different extended data flow diagrams that focus on a specific security topic (as an architectural view).

Object Traces At the moment, we are integrating a static analysis that allows us to extract static object process graphs for specific Java APIs. Object process graphs show the API usage of concrete objects during a program run [23]. We use this approach to extract the usage of widely-used security APIs, such as the Java Cryptography Architecture, and to add the results to extended data flow diagrams to allow security experts to check whether these interactions are secure.

Source Code Integration Currently, navigation is not possible between our extended data flow diagrams and the actual implementation of a software system. One of our next steps will be to add the possibility to jump to the source code for those diagram elements that have a source code counterpart. This will make it easier for security experts to use extended data flow diagrams as a starting point for security assessments.

References

[1] G. McGraw, *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[2] A. Shostack, *Threat Modeling: Designing for Security*, 1st ed. Wiley Publishing, 2014.

[3] M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006.

[4] D. Dhillon, "Developer-driven threat modeling: Lessons learned in the trenches," *IEEE Security & Privacy*, vol. 9, no. 4, pp. 41–47, 2011.

[5] B. J. Berger, K. Sohr, and R. Koschke, "Extracting and analyzing the implemented security architecture of business applications," in *17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013*, A. Cleve, F. Ricca, and M. Cerioli, Eds. IEEE Computer Society, 2013, pp. 285–294.

[6] M. Abi-Antoun and J. M. Barnes, "Analyzing Security Architectures," in *Proceedings of the IEEE/ACM Int. Conf. on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 3–12.

[7] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Automated software architecture security risk analysis using formalized signatures," in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 662–671.

[8] A. Schaad and M. Borozdin, "Tam2: Automated threat analysis," in *Proc. of the 27th Annual ACM Symposium on Applied Computing*, 2012, pp. 1103–1108.

[9] J. Jürjens and P. Shabalin, "Automated verification of UMLsec models for security requirements," in *Proc. of UML 2004 - The Unified Modeling Language: Modeling Languages and Applications*, ser. LNCS, vol. 3273. Springer, 2004, pp. 365–379.

[10] D. Basin, M. Clavel, J. Doser, and M. Egea, "Automated analysis of security-design models," *Information and Software Technology*, vol. 51, pp. 815–831, 2009.

[11] J. Czeranski, T. Eisenbarth, H. M. Kienle, R. Koschke, and D. Simon, "Analyzing xfig using the bauhaus tools," in *Proceedings of the Seventh Working Conference on Reverse Engineering, WCRE'00, Brisbane, Australia, November 23-25, 2000*. IEEE Computer Society, 2000, pp. 197–199.

[12] A. Raza, G. Vogel, and E. Plödereder, "Bauhaus - A tool suite for program analysis and reverse engineering," in *Reliable Software Technologies - Ada-Europe 2006, 11th Ada-Europe International Conference on Reliable Software Technologies, Porto, Portugal, June 5-9, 2006, Proceedings*, ser. Lecture Notes in Computer Science, L. M. Pinho and M. G. Harbour, Eds., vol. 4006. Springer, 2006, pp. 71–82.

[13] B. J. Berger, M. Bunke, and K. Sohr, "An android security case study with bauhaus," in *18th Working Conference on Reverse Engineering, WCRE 2011, Limerick, Ireland, October 17-20, 2011*, M. Pinzger, D. Poshyvanyk, and J. Buckley, Eds. IEEE Computer Society, 2011, pp. 179–183.

[14] K. Sohr and B. Berger, "Idea: Towards architecture-centric security analysis of software," in *Proceedings of the Second International Symposium on Engineering Secure Software and Systems*, ser. Lecture Notes in Computer Science, F. Massacci, D. S. Wallach, and N. Zannone, Eds., vol. 5965. Springer, 2010.

[15] B. J. Berger, K. Sohr, and U. H. Kalinna, "Architekturelle sicherheitsanalyse für android," in *DACH Security 2014: Bestandsaufnahme - Konzepte - Anwendungen - Perspektiven*, P. Horster, Ed. Peter Schartner and Peter Lipp, 2014, pp. 287–298.

[16] B. J. Berger, K. Sohr, and R. Koschke, "Automatically extracting threats from extended data flow diagrams," in *Engineering Secure Software and Systems*, J. Caballero, E. Bodden, and E. Athanasopoulos, Eds. Cham: Springer International Publishing, 2016, pp. 56–71.

[17] U. of Bremen. (2019) ArchSec—The Architectural Security Tool Suite. [Online]. Available: https://archsec.informatik.uni-bremen.de

[18] R. Vallée-Rai, P. Co, E. Gagnon, L. J. Hendren, P. Lam, and V. Sundaresan, "Soot - a java bytecode optimization framework," in *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative Research, November 8-11, 1999, Mississauga, Ontario, Canada*, S. A. MacKay and J. H. Johnson, Eds. IBM, 1999, p. 13.

[19] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*, W. R. Cook, S. Clarke, and M. C. Rinard, Eds. ACM, 2010, pp. 307–309.

[20] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, G. Das, C. M. Jermaine, and P. A. Bernstein, Eds. ACM, 2018, pp. 1433–1445.

[21] C. D. Schulze, M. Spönemann, and R. von Hanxleden, "Drawing layered graphs with port constraints," *J. Vis. Lang. Comput.*, vol. 25, no. 2, pp. 89–106, 2014.

[22] ISO/IEC/IEEE, "Systems and Software Engineering – Architecture Description," *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pp. 1–46, 1 2011.

[23] J. Quante and R. Koschke, "Dynamic object process graphs," *Journal of Systems and Software*, vol. 81, no. 4, pp. 481–501, 2008.