A DATA-DRIVEN APPROACH TO EVALUATE THE SECURITY OF SYSTEM DESIGNS

BY

JOE SAMUEL

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Applied Science
in
Electrical and Computer Engineering

Carleton University
Ottawa, Ontario, Canada

Abstract

Improving system security during the design phase is challenging but can be cost-effective in the long run. Security metrics are a way to measure and manage a system's ability to minimize possible attack opportunities. While several design-level security metrics exist to evaluate vulnerabilities in system design, it is unclear which metrics provide a sound scientific basis for their characterization. Lack of security knowledge among average development teams and the lack of tool support are additional challenges.

In this work, we present a data-driven approach for the security evaluation of system designs to address the above challenges. The approach aims to incrementally improve system security and decision-making at design time. We integrate the attack surface metric which we found to be sound in our evaluation of widely-used security metrics and leverage external data sources to characterize the structural security posture of software systems. Several tools are developed to automate the approach.

Acknowledgements

This work began with an ambitious vision to help organizations develop secure systems and to provide tools that they can use to apply our findings. This work would not have been possible in such a short time frame without the unwavering support, encouragement, and guidance from my supervisor Professor Jason Jaskolka. I also express my deepest gratitude to Professor George Yee who volunteered to collaborate on this work and whose input and guidance played a pivotal role in the success of this work. I also thank Andrew Pullin who helped us make Compass toolkit a reality.

I express my sincere gratitude to Professor James Green who played a critical role in advancing my research career. His willingness to enable me to explore and apply my ideas, however ambitious they may be, allowed me to further my learning and impact the community in more ways than one. I thank Professor Mohammed Ibnkhala who encouraged me to pursue graduate studies and provided me with the guidance and resources needed to succeed.

I convey my heartfelt gratitude to my family for all their love, encouragement, and support. I also thank my friends who were more than happy to hear about my work and encourage me throughout this journey.

Contents

List of Tables

List of Figures

List of Abbreviations

2FA Two-factor Authentication

API Application Programming Interface

ATT&CK Adversarial Tactics, Techniques & Common Knowledge

CAPEC Common Attack Pattern Enumeration and Classification

CASE Computer-Aided Software/Systems Engineering

CCCS Canadian Centre for Cybersecurity

CERT Computer Emergency Response Team

CISA United States Cybersecurity & Infrastructure Security Agency

CRUD Create, Read, Update, Delete

CSE Communications Security Establishment

CSV Comma Separated Values

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

CWE Common Weakness Enumeration

CYBOK Cyber Security Body of Knowledge

DER Damage-Effort Ratio

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hyper Text Transfer Protocol

IMPACT Information Marketplace for Policy and Analysis of Cyber-Risk & Trust

IoT Internet of Things

JSON JavaScript Object Notation

MDSSM Method for Designing Sound Security Metrics

NER Named-entity recognition

NLP Natural Language Processing

NVD United States National Vulnerability Database

OCTAVE Operationally Critical Threat, Asset, and Vulnerability Evaluation

OSM Online Seller of Merchandise

OTX Open Threat Exchange

RPC Remote procedure call

RSS Really Simple Syndication

SDLC Software Development Life Cycle

SQL Structured Query Language

SSL Secure Sockets Layer

STIX Structured Threat Information eXpression

TAXII Trusted Automated Exchange of Intelligence Information

TCP Transmission Control Protocol

TRA Threat and Risk Assessment

UML Unified Modeling Language

VPN Virtual Private Network

VRSS Vulnerability Rating and Scoring System

VulDB Vulnerability Database

XMI XML Metadata Interchange

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Context

As the world grows more connected than ever, our dependence on software applications has grown significantly. Perception of software applications has changed from appearing as a mere convenience to a critical necessity. Since many software applications utilize multiple technologies to accomplish their objectives, we refer to these types of software applications as software systems. We define a *software system* as a combination of interacting software elements (such as web server or web client – each of which may encompass a variety of technologies) organized to achieve one or more objectives set out by the stakeholders. In this work, we use the terms system and software system interchangeably. We describe elements of the system that we want to protect as assets of the system. An asset of the system is defined as something of value to the stakeholders of the system [Nat].

While it may seem like common knowledge that security needs to be a core consideration when designing software systems, many organizations think of security more as an enhancement that can be applied to the software system after it is developed. A common reason why is the rush that organizations face to get their product(s) to market, leaving very little time to design software that is secure as well as functional. Another hurdle is that the average development team lacks the know-how and the tools to design secure software. Such oversight of security considerations in the early stages of the system's development process leads to the proliferation of vulnerabilities that attackers tend to seek and take advantage of.

To understand where such oversight of security occurs in the system's development, we first preview the system's development process. Software systems are generally conceived, developed, and deployed based on a development process called the *Software Development Life Cycle (SDLC)* [ISO17]. The standard concerning the development life cycle for software systems is ISO/IEC/IEEE 12207 [ISO17]. It presents a 14-step process for the technical development of a software system. For our work, we summarize these processes into five general steps as shown in Figure 1.1.

We refer to each process in the SDLC as phases to represent a distinct period in the development process. The requirements phase involves gathering the objectives and requirements for the system from its stakeholders. The design phase encompasses processes that determine the system's overall architecture. The implementation phase describes the processes linked with the development (coding) of the system based on the given requirements and design specifications. The testing phase involves processes associated with validating whether the system fulfills the requirements determined in

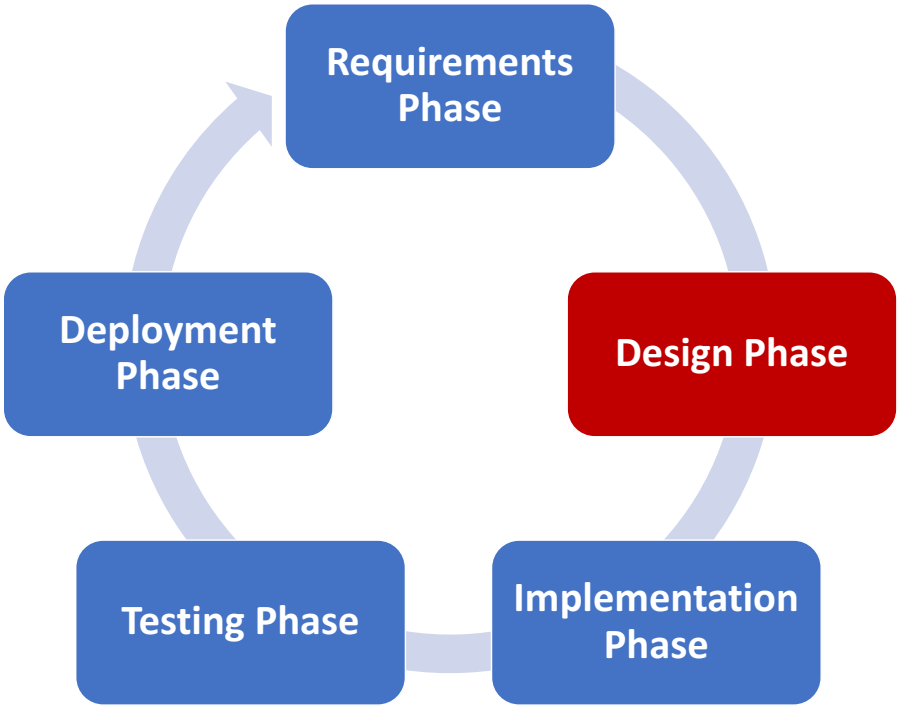


Figure 1.1: Software Development Life Cycle

the first phase of the SDLC. The deployment phase involves processes that relate to the release and maintenance of the system. It is important to note that the above phases are not necessarily sequential and each phase can be iterative. An SDLC where the focus for each phase is on various security concerns of the system is known as a *Secure SDLC* [HL06].

In this work, we focus on the design phase of the Secure SDLC highlighted in Figure 1.1. This phase is where system architects determine how various system elements communicate with each other to fulfill the objectives and requirements set out by its stakeholders (in the requirements phase). This phase deals with the high-level architectural layout of the system (not the low-level design of individual system elements). Specifications for the elements such as the required security controls and technologies

to be used are also generally determined during this phase. The design phase of the Secure SDLC is critical to the development of secure systems as addressing security concerns during the design phase of the system has large long-term payoffs for the system stakeholders as it ensures the system is adequately protected against commonly known threats before it is implemented. This phase is also ideal for iterating over system designs to address security concerns as changes can be incorporated without incurring major overhead or costs. Throughout this work, we use the term 'design-level' to denote methods and/or techniques associated with the design phase of the SDLC. System architects and developers are the individuals tasked with designing systems.

System designs can also be evaluated in terms of security. In this work, we describe security evaluation as an activity that involves examining a system to determine its degree of compliance with standards and specifications by analyzing the system design, observing system behaviours, and/or attempting to penetrate the system using techniques available to potential adversaries. It can also be thought of as the examination of a system's security to determine the extent to which it is protected against known threats. System evaluators and certifiers are the individuals tasked with evaluating the security of system designs. Since our work applies to stakeholders that design or evaluate system designs (in terms of security), we refer to them as system architects throughout this work.

One way to evaluate system designs is by using security metrics. Security metrics are commonly used to measure the security level of a system, i.e, the system's ability to minimize possible attack opportunities. Security metrics help us measure one or

more security characteristics of the system. Security metrics assist in characterizing and managing risks by enabling effective decision-making to optimize resources and prioritize mitigation efforts [Jaq07]. While several security metrics exist, it is unlikely that a single security metric can sufficiently assess all of the characteristics relating to the security of a system [MFMP10]. This is because systems are multi-faceted and best described through their multiple views. Therefore, to extract as much information as possible from a system for security evaluation, we need to consider its various views as no single view would be adequate.

A *view* for a software system refers to the system's description relative to a set of concerns based on a certain perspective of the system [Hil99]. Instead of developing a single model to describe a system, views are used to develop multiple models that collectively describe the system more effectively.

Some common views used to describe a system are the *structural* view, *behavioural* view, and the *functional* view. The structural view illustrates the composition of system elements that make up the system. It provides the terminologies associated with the system's elements and their interfaces. The functional view describes the various operations that are performed by the system and the interactions between the system and its users to perform those functions. The behavioural view presents the interactions between various elements of the system to perform certain functions.

Evaluating the security of a system using multiple metrics specific to a view of the system enables system architects to compare and contrast their system designs based on security characteristics relevant to their operational context.

Lastly, when we talk about a data-driven approach to evaluating the security of system designs, we imply that our security evaluation is supported by metrics that are the results of data analysis to better inform our evaluation.

In the following section, we discuss why security is treated more as an enhancement for a system rather than a core design consideration.

1.2 Motivation

As systems grow larger and integrate heterogeneous elements, there is a need to improve system security. Integration of heterogeneous system elements in a manner that may not have been originally intended or foreseen can introduce potential unexpected attack opportunities. Large systems tend to be complex in their architecture and may obscure potential vulnerabilities from the system architects. Implicit interactions which refer to such unintended or unforeseen interactions between system elements are an example of how potential vulnerabilities may go undetected in such systems [JV17].

Most software development processes do not emphasize security as a critical part of their development process but rather as an enhancement that can be 'bolted on' after the system is developed. Such oversight of security considerations in the early stages of the development process leads to the proliferation of vulnerabilities that attackers tend to seek and take advantage of. Worse, these types of vulnerabilities, that are present since the architectural design phase, are much more difficult and

resource-intensive to identify and mitigate later in the software system's development life cycle.

The following are just a few prominent data breaches that occurred this year that can be traced back to vulnerabilities in the system's design [Bek21]:

1. June 21, 2021: Wegmans Food Markets, a U.S. supermarket chain, inadvertently publicly exposed an undisclosed amount of customer data online after two of its cloud-based databases were misconfigured. Data exposed included customer names, addresses, phone numbers, birth dates, and email addresses.

2. May 17, 2021: Health Plan of San Joaquin was affected by a data breach caused by the attacker gaining unauthorized access to business email accounts exposing sensitive personal and medical patient information such as social security numbers, driver's license numbers, medical records (lab results, treatment information).

3. April 26, 2021: Experian, a credit reporting company, was alerted to an uncovered data leak from one of their own unsecured Application Programming Interface (API). This leak allowed anyone to access the credit scores of millions of Americans simply using their name, date of birth, and mailing address.

The need to address security concerns early in the system's development is also well-documented in literature [BC08, CK08, TM05, CKA09].

To enable system architects to design secure systems, we believe there are critical requirements that any design-level security evaluation approach must fulfill:

1. Provide design-level metrics to help system architects measure and thereby manage their system's security.

2. Provide adequate tool support to help system architects integrate the security evaluation approach into their workflow and reduce the barrier of entry for evaluating a system's security.

It is well-known that we cannot manage something effectively if we cannot measure it [CKA09]. This adage holds for managing the security of a system. Security metrics provide the basis upon which improvement of a system's security can be assessed. Security metrics can further provide insights into potential attack opportunities present in the system and highlight areas of the system that are more vulnerable than others. This type of insight can help in the threat prioritization and mitigation efforts for system architects and increases their overall security awareness of their system.

While security metrics can help with providing a sense of security, strong security assurance requires a thorough understanding of the security requirements, threats, and controls so that they can be incorporated at all stages of development [Jas20]. A major challenge that contributes to the lack of emphasis on security during the system's design is the lack of security expertise and tools to evaluate a system's security in an average development team. The added pressure for organizations to rush their system(s) to market only exacerbates this problem. Therefore, there is a need for better tool support and a more data-driven approach to support system architects with minimal security expertise in the development of secure systems and to make more accessible tools for designing secure systems.

1.3 An Example Design Scenario: Online Seller of Merchandise (OSM)

Let's meet Alice. Alice is a system architect. She has knowledge and expertise in designing software systems. Although she is aware of some security concepts including basic knowledge of some attacks and threats, she is not a security expert. Alice typically works in a top-down, iterative design process. She represents the typical type of user that we consider for this work.

Alice is in the process of architecting a Online Seller of Merchandise (OSM) system. The OSM system's objective is to allow registered customers to browse the store inventory and purchase one or more products. Alice already sketched out an initial design for the OSM systems using the Unified Modeling Language (UML) class diagram notation as shown in Fig. 1.2 based on her intuition of what elements are required to fulfill the system's objective. However, she is looking to improve the security aspects of the OSM system, given her initial system design. The idea and design for Alice's OSM system are based on the example OSM system presented in [Yee19c].

In her adaptation, the OSM system has the following system elements:

- Web Client: A web-based interface for customers to interact with the OSM system.
- Web Server: Responsible for orchestrating the purchase process initiated by the customer and for serving the relevant information (such as order status, shipping status, items available) to the Web Client.

- Identity Access Manager: Responsible for ensuring the customer is authorized to perform the actions they are requesting (such as placing an order of an item).
- Customer Datastore: Stores data related to the customers registered with the store such as name, contact, credit card information, and address.
- Customer Manager: Ensures customer information is present in the Customer Datastore for the order to be processed.
- Transaction Processor: Processes the payment using the customer's details such as name, address, and credit card information.
- Inventory Datastore: Stores data related to the various items available in the store such as available stock and price information.
- Inventory Manager: Ensures the stock required is available for the order to be processed and that the requested number of items is shipped and accounted for in the Inventory Datastore following a successful transaction.
- Shipment Processor: Facilitates the shipping for the ordered items by coordinating with the Inventory Manager to ensure the stock required is available and the number of items shipped is accounted for in the Inventory Datastore following a successful transaction.
- Order Processor: Responsible for processing the incoming order of one or more products by ensuring the required information from the customer such as their credit card information is on file (through the Customer Manager), processing the payment for the order using the Transaction Processor, and

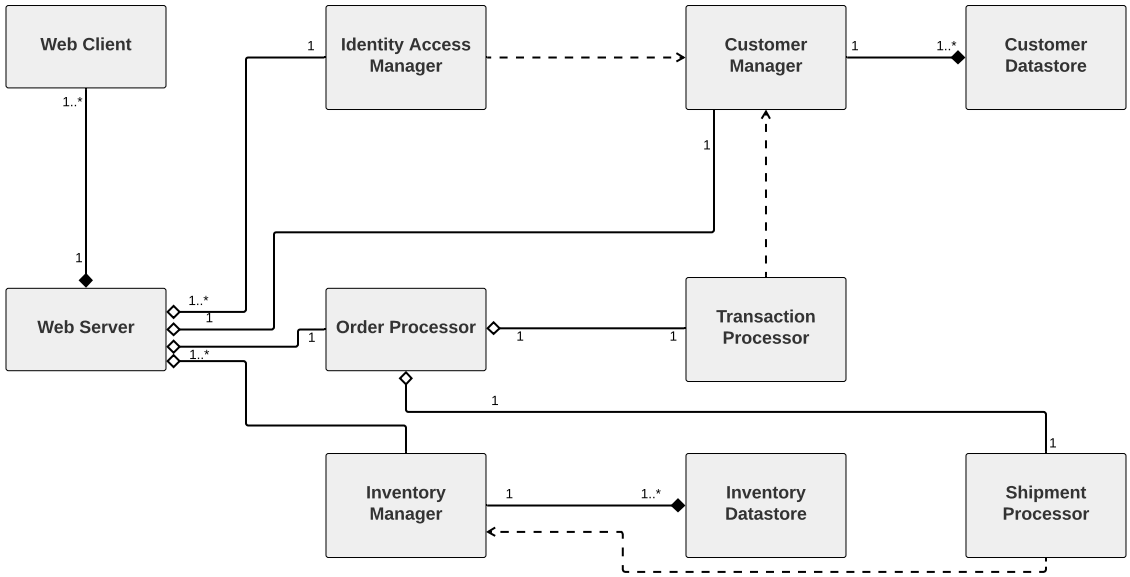


Figure 1.2: UML class diagram representing the structural view of the OSM system

passing that information to the Shipment Processor to ship the requested product(s).

1.4 Challenges

To accomplish her design goals and improve the security of her initial design, Alice faces several challenges that we need to help her overcome.

Alice is not a security expert. To address the lack of security proficiency among system architects like Alice, we can leverage the vast amounts of openly available security data sources that contain up-to-date information on the latest threats and vulnerabilities associated with various technologies. While the concept of such a data-driven approach to evaluating the security of a system's design sounds promising in

theory, it also prompts a number of challenges when translating that notion into practice. For one, what kind of data can we use to help us support secure system design? Where do we find this data? What metrics can we derive from such data to integrate within an evaluation? Can we automate the data gathering process? It is also important to consider how we can help distill such vast amounts of information into a more digestible form for system architects.

Alice needs a way of measuring the security of her system design to understand whether her design decisions improve the security of her design. In Section 1.2, we established the need for security metrics to evaluate the security of system designs. However, there are plenty of security metrics in security literature that focus on evaluating the security of systems. So which metrics do we pick? For a start, we are looking for metrics that evaluate the system at the design level, but this still leaves us with plenty of metrics to choose from. A major challenge is to choose the metrics that represent various security characteristics of the system so they could collectively describe the system's security with minimal overlap, but also are easy for system architects to understand and compute. The latter requirement is especially important as it is critical for the system architect to understand the meaning and interpretation of the metric so they can better manage their system's security. Another important criterion to consider when selecting a metric is to evaluate whether that metric itself is reliable and trustworthy.

Alice cannot be expected to analyze her system designs and evaluate the impact of each design on security by hand; she needs tools that can support her and fit within her iterative design process. Providing adequate tool support for a data-driven security

evaluation approach presents a series of challenges on its own. How do we design a tool that is simple, yet effective, to enable system architects to derive relevant insights regarding their system's architectural security? Can we develop a tool that enables system architects to run 'what-if' analyses and visualize, to some level, the overall security of their system?

1.5 Problem Statement

Given the above challenges, we deduce that system architects (such as Alice) primarily need a design-level security evaluation approach and adequate tool support to design secure systems.

In this thesis, we aim to develop a security evaluation approach to support system architects with minimal security expertise to design secure systems. Specifically, we focus on the structural view of the system and present a top-down approach where we evaluate the architectural design of the system. We aim to help system architects leverage external security data sources to understand their system's threat landscape. We also aim to accompany our evaluation approach with adequate tool support to enable system architects to evaluate and manage the security of their system designs.

It is important to note that, in this work, we focus on system-level design rather than other types of design such as software module design or feature design.

1.6 Contributions

The need to support system architects with a data-driven approach to evaluate the security of their system designs coupled with the various challenges listed in Section 1.4 motivated the contributions for this thesis. In this section, we list our contributions towards addressing some of the aforementioned challenges.

1.6.1 Research Contributions

We present the following as research contributions from this work:

1. **An approach for leveraging external data sources to support secure system design (Chapter 3)**: We address some of the challenges mentioned in Section 1.4 regarding developing a data-driven approach to evaluating the security of a system's design. We do so by focusing on how publicly available, well-maintained data sources for vulnerabilities, attack patterns, threat intelligence, and other security information can be leveraged, using techniques such as Natural Language Processing (NLP), to produce a report to assist system architects in validating the adequacy of their security requirements. We also generate a number of data-driven metrics by determining which requirements map to known threats (identified from the data sources), which requirements may be extraneous, and which threats may need not have been addressed through the given security requirements. These metrics also help system architects get a distilled form of the vast amounts of data gathered. We also describe the availability and nature of such data sources, followed by how we employ NLP

to process the data and produce the report. We illustrate our approach using an asset of the OSM system.

2. **Evaluation of existing security metrics for soundness (Chapter 4)**: To identify which widely-used security metrics are trustworthy, we evaluated them using the Method for Designing Sound Security Metrics (MDSSM) proposed in [Yee19b]. Based on the results, we provide several recommendations to help developers, evaluators, and certifiers apply the frameworks in ways that can lead to obtaining sound security metrics. These recommendations can enable better decision-making, based on recognized scientific principles, which can improve security assurance for software systems. We also provide a spreadsheet tool to evaluate the soundness of security metrics based on MDSSM. Our findings also support the selection of trustworthy security metrics that we incorporate into our system-level security evaluation approach.

3. **A data-driven approach to evaluate the security of a system's design (Chapter 5)**: We present a data-driven approach to support the evaluation of a system's security during the system design phase. We achieve this by incorporating metrics that collectively provide a better representation of a system's security characteristics based on its structural view. We call this the structural security posture of a system. We validate the usefulness of this approach by evaluating the structural security posture of the OSM system. We also present the insights gained from our evaluation.

1.6.2 Tool Support Contributions

We present the following as contributions towards addressing the need for adequate tool support for secure system design:

1. **Data-driven security requirements and design decisions analysis tool (Section 3.4)**: A data-driven requirements analysis tool that uses NLP to process requirements data, query external data sources, and produce a report that provides at-a-glance metrics and details the threats identified.

2. **Structural security posture analysis tool (Section 5.2)**: We present a structural security posture analysis tool where a system model represented as a UML class diagram can be built and analyzed. The tool automates metric calculations and enables 'what-if' analyses. It also presents the insights gained from the evaluation as a report.

3. **A Toolkit for Secure System Design Tools (Chapter 6)**: We developed a toolkit to house some of our tools that support secure system design. This toolkit is designed to be extensible and aims to be the home for future tools that automate various aspects of designing secure systems.

1.7 Related Publications

The following are a list of conference publications related to the work presented in this thesis:

- J. Samuel, K. Aalab, and J. Jaskolka. Evaluating the soundness of security metrics from vulnerability scoring frameworks. In *Proceedings of the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, IEEE TrustCom 2020, pages 442–449, Guangzhou, China, Dec. 2020

- J. Samuel, J. Jaskolka, and G. Yee. Leveraging external data sources to enhance secure system design. In *Proceedings of the 2021 Conference on Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge*, RDAAPS 2021, pages 1–8, Hamilton, ON, Canada, 2021.

1.8 Chapter Previews

We provide a preview of the following chapters in the thesis below. We summarize the relationship between the structure of this thesis and our contributions in Figure 1.3.

Chapter 2 reviews the state-of-the-art for evaluating the security of system designs.

Chapter 3 discusses our approach for leveraging external data sources to assist system architects to develop adequate security requirements. We also present metrics that can be derived from this work.

Chapter 4 investigates the soundness of existing security metrics used to assess a system's security. We present our findings and recommendations, based on our evaluation.

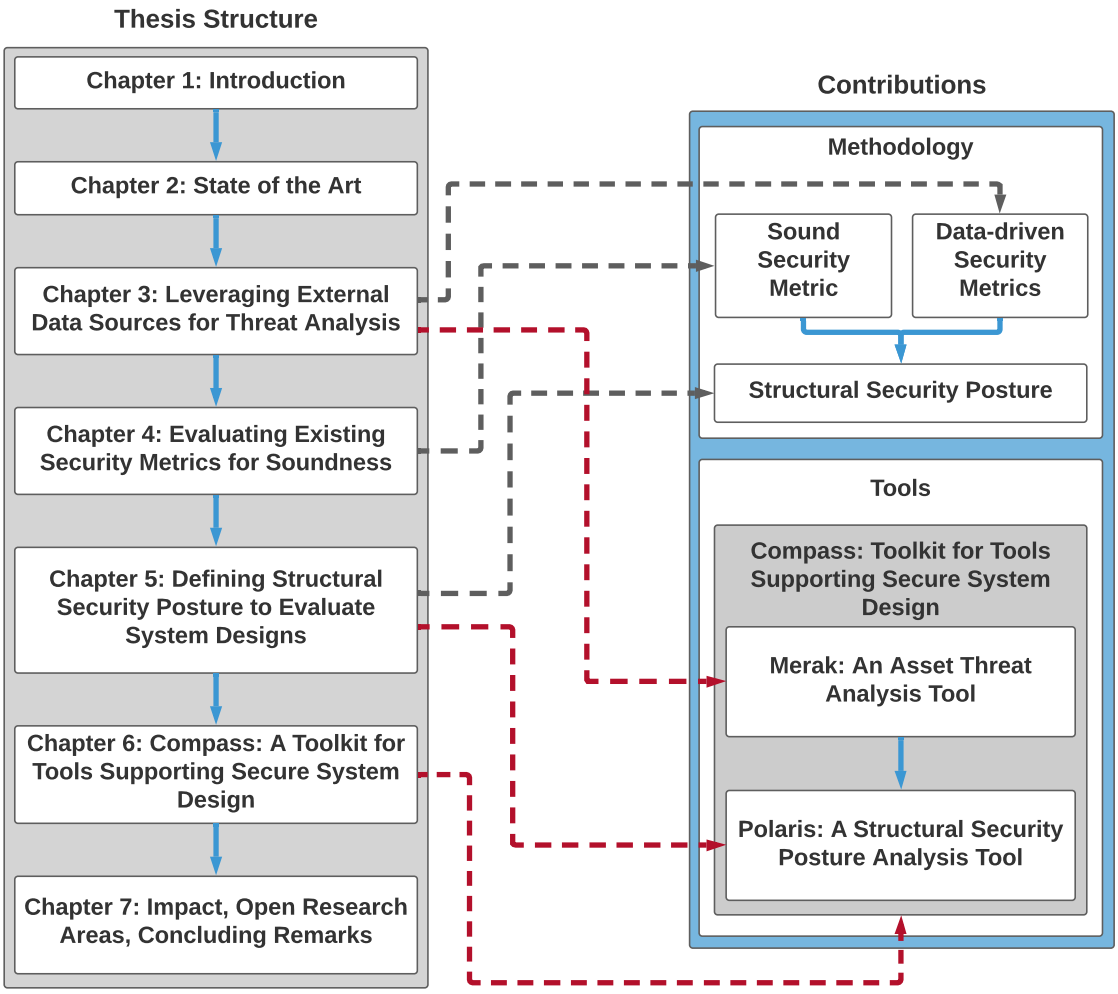


Figure 1.3: Thesis Flow

Chapter 5 presents our data-driven approach for evaluating the security for system designs.

Chapter 6 presents a toolkit that houses the various tools developed in this work and future secure system design tools to help practitioners develop secure systems.

Chapter 7 concludes this thesis by stating the impact of our contributions, open research areas, and concluding remarks.

Chapter 2

State of the Art

In this chapter, we review the state-of-the-art for evaluating the security of system designs. Section 2.1 reviews the current state-of-the-art for leveraging external data sources for security evaluation. Section 2.2 reviews works associated with security metrics that evaluate a system at the design-level. Section 2.3 reviews existing tools to support system architects design secure systems. Section 2.4 presents our conclusions from this chapter.

2.1 Leveraging External Data Sources for Security Evaluation

Bakirtzis et al. [BSC+20] proposed Cyber Security Body of Knowledge (CYBOK) as an algorithmic way to leverage external data sources and explore vulnerabilities.

Within CYBOK they used a system model represented as a graph with associated vulnerabilities being populated from CVE, CWE, and CAPEC to identify potential attack vectors. Nerwich et al. [NGPN20] addressed the problem of lack of knowledge management systems for Internet of Things (IoT)-specific vulnerabilities and attacks. They presented the construction of a community-driven, IoT-specific database which documented the vulnerabilities and attacks on IoT infrastructures and supported integration with other vulnerability databases such as NVD. Feng et al. [FKC+16] focused on the impact of architectural design decisions on security. They proposed a Design Rule Spaces-based analysis approach to identify architectural design flaws and used CVEs to find correlations with *hotspot patterns* which were architectural flaws related to violations of proper design principles.

Elahi et al. [EYZ10] presented a goal model with which to evaluate the effects of vulnerabilities and countermeasures with a focus on requirements elicitation. They propose a formal framework to support the requirements elicitation process by using vulnerability information as a means to develop countermeasures that then form new security requirements. Hibshi et al. [HBW16] discuss the challenge of relying solely on security experts to improve the security of their systems based on their knowledge and experience to determine known threats and vulnerabilities. They propose an empirical method for developing a fuzzy logic decision support system to analyze the adequacy of security requirements and perform security evaluations.

In our work, we aim to develop an automated approach that can help system architects evaluate the adequacy of security requirements (i.e., that they address the actual needs) for a given element of the system by leveraging external data sources. To

the best of our knowledge, this type of approach to leverage external data sources to ensure the adequacy of security requirements has not been done.

2.2 Evaluating System Security using Security Metrics

In this section, we review existing security metrics that evaluate the security of a system during its design phase. We also review security evaluation approaches in security literature that incorporate a collection of metrics for their evaluation rather than a single metric since a single metric is unlikely to sufficiently characterize the overall security of a system. We also look at existing works that have evaluated the trustworthiness of security metrics.

2.2.1 Existing Design-level Security Metrics

In [AFC09], Alshammari et al. propose security metrics for object-oriented class designs where the attributes and methods are known. Using this information, they propose a variety of metrics such as ratios of specific types of methods/attributes and their interactions within a class. In [ASKM07], Aggarwal et al. propose design-level metrics that focus on exception handling at the code level by calculating metrics such as the number of catch blocks per class. There are a number of other design-level security metrics such as the attack surface metric that we discuss in more detail in Chapter 4.

In their evaluation of various design-level metrics, Mellao et al. [MFMP10] emphasize that it is difficult to cover all the security properties of a system, from the design perspective, using a single metric since design-level security metrics evaluate specific security characteristics of the system. They also mention how most security metrics are aimed to assess software beyond the design phase. In the following subsection, we discuss approaches that follow this notion of incorporating multiple metrics to evaluate security.

2.2.2 Evaluating System Security using a Collection of Metrics

Security posture is a term commonly used in security literature to represent approaches that incorporate multiple metrics for evaluating a system's security. In [Wil12], Williams defines an organization's security posture to be its accepted risk level to which a system or organization is exposed. Williams recommends measuring security posture by either auditing security policies or performing vulnerability or penetration testing, both methods relating to the implementation phases of the system. In [BSC+20], Bakirtzis et al. loosely tie the concept of the security posture of a given system to complementary metrics that security professionals can use to better characterize the security of their system. They propose the attack surface and exploit chain metrics where an exploit chain evaluates the potential for further violations (post-attack) of a system element. They also emphasize the importance of a well-constructed model to ensure the validity of insights gained from evaluating the security posture of the system.

In [ROEM18], Ridley et al. quantify the security posture of containerized mission-critical systems, in the implementation phase, using a desirability function that determines a set of operating conditions that optimize a system based on desired requirements. A number of other works such as [Cho09], [Ros02], and [Gol12] use the term security posture in the context of computer security without defining it explicitly.

In our work, we aim to incorporate relevant security metrics for analyzing the structural security posture of a system to analyze system designs at the design phase.

2.2.3 Evaluating Security Metrics for Trustworthiness

While there are a number of security metrics to choose from, it is important to identify those that are trustworthy a system's security evaluation is reliable and reproducible.

The literature has seen wide discussion on the characteristics of a trustworthy security metric which they sometimes refer to as a 'good' security metric. Jelen [Jel00] described a good security metric as SMART: specific, measurable, attainable, repeatable, and time-dependent. Alternatively, Savola [Sav07] recommended that security metrics be quantitative, objective, and dynamic (time-dependent).

Other works underscored the need for good security metrics to be meaningful in their application. Yee [Yee01] stated that for a security metric to be meaningful, it must be multi-faceted or multi-dimensional, composed of metrics found from different applicable levels of the metrics taxonomy, and must address disparate concerns such as confidentiality, integrity, and availability. Payne [Pay06] also highlighted that useful

security metrics indicate the degree to which security goals such as data confidentiality are satisfied.

While past work discussed the important characteristics of a good security metric, to the best of our knowledge, no work evaluated the soundness of existing security metrics obtained from vulnerability scoring frameworks to better understand the implications of their use in decision-making processes. We aim to bridge this gap in this work.

2.3 Reviewing Tool Support for Secure System Design

It is important to support system architects in applying secure system design research by providing adequate tool support. In [JS07], Jürjens and Shabalin identified the need for developing adequate tool support for developing secure systems. Their work presented tools for verifying constraints associated with UMLsec models where UMLsec is an extension of UML that incorporates security properties within the model [Jür05]. In [SG04], Schmerl and Garlan reiterated the importance of modeling software architecture and stated the need for adequate tool support to develop, view, analyze, and refine system models. They presented AcmeStudio as a development environment for architectural designs associated with different domains. Gennari and Garlan presented an attack surface plug-in for AcmeStudio in [GG12] to analyze and manipulate the attack surface properties of a system.

There is also a category of tools called Computer-Aided Software/Systems Engineering (CASE) that support the software development process by facilitating some degree of automation [NF92]. While there are plenty of CASE tools such as SecTro [PI11] available to support the design of secure systems, there are multiple studies such as [LC98], [Mar95], [NTH89], and [SW95] that reported limited adoption (or in some cases, complete abandonment) of such tools among organizations and practitioners at large. This is mainly attributed to the usability difficulties with such tools and the high costs associated with procuring such tools. Even though [CZ80] and [LC97] showed that system architects preferred high autonomy in their workflows, the lack of perceived usefulness and intuitiveness (if the tool was enjoyable to use) with CASE tools were cited as major reasons that system architects failed to adopt such tools despite their theoretical usefulness to the development process [LC98].

Therefore, it is important to develop adequate tool support for system architects to apply secure system design research and equally important to design them in a manner that is simple and intuitive and accessible with high procurement costs. We aim to apply these findings as key considerations for the tools we develop in this work.

2.4 Conclusions

In this chapter, we reviewed the state-of-the-art for evaluating the security of system designs and identified gaps that our work aims to address. In our review associated with leveraging external data sources, we found no works directly leveraged external

data sources to ensure the adequacy of security requirements. In our review of design-level security metrics, we discovered that, while a number of security metrics exist, we need to evaluate them to identify if they are trustworthy. We also noticed that the term security posture is generally used to refer to approaches that incorporate multiple security metrics to evaluate security. However, the term is used rather loosely and does not assess a system's security based on a specific view of the system. We also reviewed the state of tool support for designing secure systems where we discovered the need for simple, intuitive tools that were more accessible to organizations and practitioners at large.

In the following chapters, we aim to address the various gaps in security literature concerning the security evaluation of system designs.

Chapter 3

Leveraging External Data Sources for Threat Analysis

In this chapter, we consider leveraging vast amounts of external security data sources to help system architects like Alice overcome the lack of security proficiency and tools to support the system design process. Specifically, we present an approach for ensuring that security requirements are adequate, i.e., the requirements address the actual needs of the system. Adequate requirements can lead to the specification of security controls that mitigate known threats and vulnerabilities. Section 3.1 outlines the problem we aim to address in this chapter. Section 3.2 provides an overview of the various external data sources that contain security-relevant information that we can leverage to support secure system design. Section 3.3 describes the methodology behind our approach. Section 3.4 describes the tool developed to automate and support our threat analysis approach. Section 3.5 presents a step-by-step description

of the application of our approach using our tool. Section 3.6 discusses insights and observations from developing our approach. Lastly, Section 3.7 presents our conclusions from this work and highlights areas for improvement.

3.1 The Problem

Support for making good design decisions is an essential aspect of designing secure software systems. Architects are tasked with ensuring that adequate security controls to protect critical system assets are built into the systems that they develop. This demands informed decision-making processes to develop flexible and adaptive design solutions to ensure that these systems will continue to protect these critical assets as the threat landscape continuously evolves. Moreover, design decisions to balance tradeoffs among competing, and sometimes contradictory, system qualities need to be made during system development [WL13]. These decisions need to be sufficiently documented and justified to adequately support security assurance efforts.

Threat modeling is often an essential part of eliciting security requirements and understanding possible vulnerabilities and attack scenarios for systematically analyzing a probable attacker's profile, the most likely attack vectors, and the assets most desired by an attacker, which thereby enables informed decision-making about security risk [Sho14, UM15]. However, there are several prominent challenges in relying on threat modeling alone. It is often difficult to determine whether identified threats are rooted in real-world exploits or vulnerabilities. The number and types of threats identified often depend on the security knowledge and experience of the person(s)

doing the threat modeling. Subjectivity and bias can lead to unsound decisions for mitigating identified threats (we discuss this further in Chapter 4). This is compounded by the fact that there are many threat modeling methods and there is a lack of guidance as to which method is best for a particular application.

To overcome some of these challenges, we need to determine how data-driven approaches that leverage external online data sources, such as vulnerability databases, can support system developers in making more objective decisions to improve the security of their designs. In particular, how can this data be used to identify targets and vulnerabilities in the system design? To the best of our knowledge, leveraging the external data in this manner has not been done.

3.2 External Data Sources

In this section, we explore different external sources of security-related data that can be leveraged to support secure system design activities. The following categories of data sources were chosen based on their public availability and widespread use, but they are by no means exhaustive of the types of data that can be leveraged in our work.

3.2.1 Vulnerability Databases

Several organizations have compiled databases of security threat and vulnerability data. These databases primarily include Common Vulnerabilities and Exposures

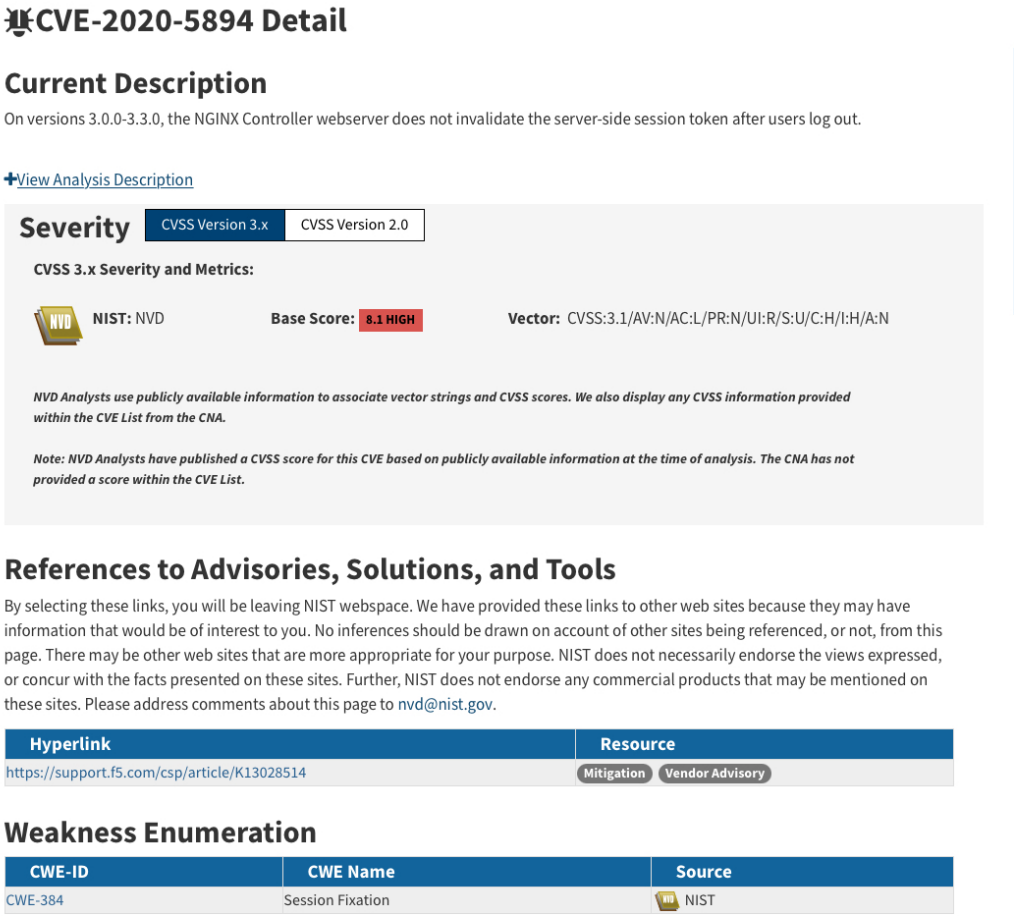
(CVE) which is a list of publicly known security vulnerabilities [Theb]. Each CVE vulnerability has a unique identification (ID) number, a description, and at least one public reference.

United States National Vulnerability Database

A popular source of CVE data is the United States National Vulnerability Database (NVD) [Nat20]. The NVD builds upon the information included in CVE entries and provides enhanced vulnerability information in the form of standardized and analytical attributes [Kas]. Standardized attributes include a unique ID and a detailed CVE description. The CVE description provides details such as the name of the affected product and vendor, a summary of affected versions, the vulnerability type, the impact, the access required by an attacker to exploit the vulnerability, and the code or inputs involved. An example of a CVE from NVD concerning an NGINX Web Server is shown in Figure 3.1. Analytical attributes include the severity score provided by the Common Vulnerability Scoring System (CVSS) [MSR07] which is regularly used for quantifying and assessing the severity and risk of security vulnerabilities in computing systems, something we discuss in detail in Chapter 4.

Information Marketplace for Policy and Analysis of Cyber-Risk & Trust

The Information Marketplace for Policy and Analysis of Cyber-Risk & Trust (IMPACT) [IMP] is an information exchange website offering cyber-risk data sets and tools for the benefit of the research community. The data and tools are contributed to



CVE-2020-5894 Detail

Current Description

On versions 3.0.0-3.3.0, the NGINX Controller webserver does not invalidate the server-side session token after users log out.

+View Analysis Description

Severity CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 8.1 HIGH **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
https://support.f5.com/csp/article/K13028514	Mitigation Vendor Advisory

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-384	Session Fixation	NIST

Figure 3.1: An example CVE from NVD

IMPACT by various individuals and organizations who continue to retain ownership of their contributions. Data and tools may be requested under different categories such as cyber attacks and cyber crimes for different years.

WhiteSource Vulnerability Database

The WhiteSource Vulnerability Database [Whi] contains vulnerabilities associated with open-source technologies. It provides comprehensive for-fee services for open-source developers that not only include vulnerabilities but also fixes for the vulnerabilities. It covers over 200 programming languages and over 3 million open-source technologies, aggregating information from sources such as NVD and security advisories many times a day.

Other Vulnerability Databases

Rounding out the above databases are additional vulnerability databases [Cor], as follows. The Vulnerability Notes Database run by the Computer Emergency Response Team (CERT) of the Software Engineering Institute at Carnegie Mellon University provides information on vulnerabilities such as issue summaries, affected vendors, remedial actions, and technical details. The Exploit Database run by Offensive Security includes a searchable list with information on verified and unverified vulnerabilities. Each entry in this database includes any known ID numbers, platforms affected, exploit type, and code that can be used in penetration testing. The Vulnerability Lab open-source database provides information on vulnerability language, type, severity level, exposure volume, and ideas for remediation. Vulnerabilities can be searched using CVE or project name. Finally, the Vulnerability Database (VulDB), another open-source database, has over 140,000 entries and provides information on affected vendors, classification and ID, vulnerability type, timelines, and exploit prices (prices

of the exploits on the black market). Entries may also contain threat intelligence and mitigations that can be purchased.

Data delivery from the above sources is typically accomplished through APIs and HTTP. Data formats can be CSV, HTML, JSON, Text, and XML, among others, and depending on the source. For example, CVE and NVD deliver their data using APIs and HTTP. In the fall of 2019, NVD began offering web services (e.g., HTTP GET) that contain queries to filter the data retrieved for use by applications [BO]. This allows better management of the size of a data retrieval. CVE offers data in the following formats: CSV, HTML, Text, XML. NVD offers data in JSON and XML formats.

Another source of vulnerability data is the Common Weakness Enumeration (CWE) [Thec]. CWE is a community-developed list of common software and hardware weaknesses that can be found in software or hardware implementation, code, design, or architecture that if left unaddressed could result in systems, networks, or hardware being vulnerable to attack. Note that in CWE a *weakness* is different from a *vulnerability* in that weakness is an error that can lead to vulnerabilities.

Many of the above sources are intertwined in various ways. For example, CWE uses NVD which is based on CVE. The WhiteSource Vulnerability Database is partially based on NVD. This could be a good reason to focus on NVD which is an enhanced version of CVE.

3.2.2 Attack Patterns and Adversarial Behavior Databases

Attack patterns describe the common attributes and approaches employed by adversaries to exploit known weaknesses within systems. They derive from the concept of design patterns, defining the recurring challenges that adversaries may face when conducting an attack and how they go about solving it. Attack patterns are generated from detailed analyses of specific examples of real-world exploits. Each attack pattern typically describes the mechanisms used in an attack and provides potential mitigations to avoid exploits and reduce their effectiveness. By considering such an adversarial perspective to attacks, system architects can better understand patterns across attacks such as common exploits or techniques used. Additionally, this type of information can help system architects better characterize the spread of an attack on their systems following its initial exploitation.

MITRE Common Attack Pattern Enumeration and Classification

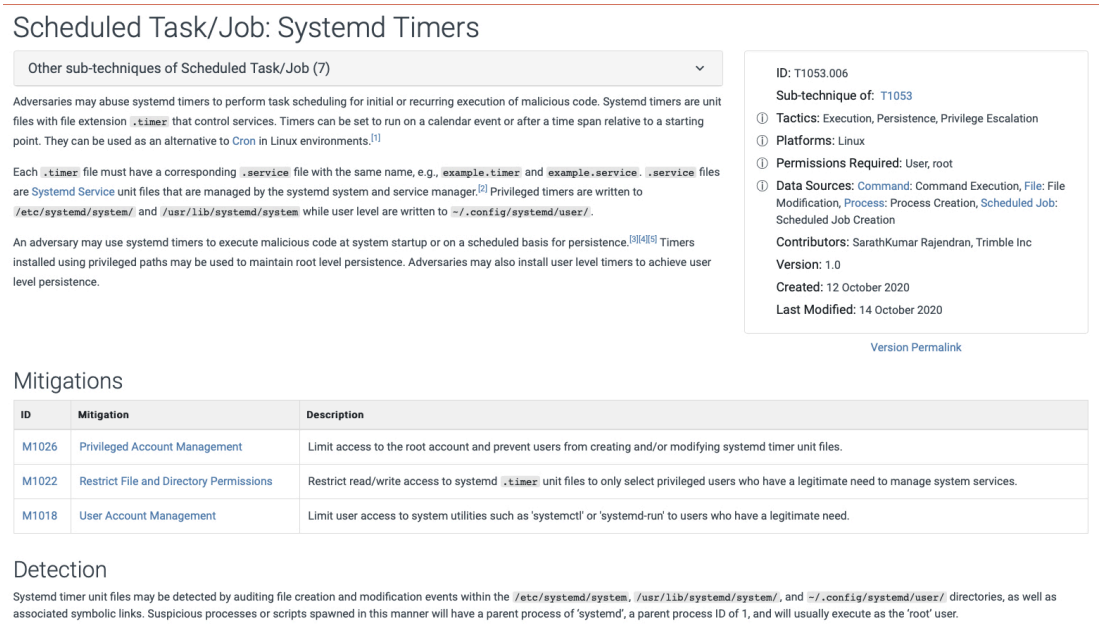
One of the most widely-known attack pattern data sources is MITRE's Common Attack Pattern Enumeration and Classification (CAPEC) [Thea]. CAPEC provides a publicly available catalog of common attack patterns including those for SQL Injection, Cross-Site Scripting, and Buffer Overflow, just to name a few. Each attack pattern in the CAPEC catalog includes a description of the attack, a list of related attack patterns, associated domains of attack, and mechanisms of attack, an execution flow describing how to conduct the attack, a list of prerequisites that serve as preconditions for attack success, and a list of mitigations that can be implemented to reduce

the attack's effectiveness. The CAPEC catalog can be downloaded (using HTTP) in various formats including CSV, XML, and a rendered HTML representation.

MITRE Adversarial Tactics, Techniques & Common Knowledge

Another source of attack patterns and adversarial behavior data is the Adversarial Tactics, Techniques & Common Knowledge (ATT&CK) framework [SAM+20]. ATT&CK is mostly focused on network defense and describes the operational phases undertaken by adversaries both pre- and post-exploit such as persistence, lateral movement, and exfiltration.

ATT&CK consolidates information gathered through community contribution and MITRE's research and security activities such as penetration testing and red-teaming. It is important to note that ATT&CK does not intend to provide a comprehensive list of techniques, but rather an approximation of publicly available information for them. As such, for each identified attack technique, it captures the name of the attack technique, the tactic employed (e.g., *persistence* when the adversary is trying to maintain their foothold), the platform(s) affected (e.g., Windows), the level of permissions required for the technique, any associated techniques identified by their IDs, a brief explanation of this tactic and its potential effects, how to detect the technique, a list of potential mitigations for the technique, a list of known adversaries using the technique, and a list of additional references to information relating to the technique. An example entry for a threat relating to a Web Server from MITRE's ATT&CK is shown in Figure 3.2. In particular, there are 14 high-level tactic categories used to



Scheduled Task/Job: Systemd Timers

Other sub-techniques of Scheduled Task/Job (7)

Adversaries may abuse systemd timers to perform task scheduling for initial or recurring execution of malicious code. Systemd timers are unit files with file extension `.timer` that control services. Timers can be set to run on a calendar event or after a time span relative to a starting point. They can be used as an alternative to Cron in Linux environments.[1]

Each `.timer` file must have a corresponding `.service` file with the same name, e.g., `example.timer` and `example.service`. `.service` files are Systemd Service unit files that are managed by the systemd system and service manager.[2] Privileged timers are written to `/etc/systemd/system/` and `/usr/lib/systemd/system` while user level are written to `~/.config/systemd/user/`.

An adversary may use systemd timers to execute malicious code at system startup or on a scheduled basis for persistence.[3][4][5] Timers installed using privileged paths may be used to maintain root level persistence. Adversaries may also install user level timers to achieve user level persistence.

ID: T1053.006
Sub-technique of: T1053
Tactics: Execution, Persistence, Privilege Escalation
Platforms: Linux
Permissions Required: User, root
Data Sources: Command: Command Execution, File: File Modification, Process: Process Creation, Scheduled Job: Scheduled Job Creation
Contributors: SarathKumar Rajendran, Trimble Inc
Version: 1.0
Created: 12 October 2020
Last Modified: 14 October 2020

Version Permalink

Mitigations

ID	Mitigation	Description
M1026	Privileged Account Management	Limit access to the root account and prevent users from creating and/or modifying systemd timer unit files.
M1022	Restrict File and Directory Permissions	Restrict read/write access to systemd `.timer` unit files to only select privileged users who have a legitimate need to manage system services.
M1018	User Account Management	Limit user access to system utilities such as 'systemctl' or 'systemd-run' to users who have a legitimate need.

Detection

Systemd timer unit files may be detected by auditing file creation and modification events within the `/etc/systemd/system`, `/usr/lib/systemd/system/`, and `~/.config/systemd/user/` directories, as well as associated symbolic links. Suspicious processes or scripts spawned in this manner will have a parent process of 'systemd', a parent process ID of 1, and will usually execute as the 'root' user.

Figure 3.2: An example entry in MITRE's ATT&CK

classify techniques which are further broken down into sub-techniques. This granularity in technique classification can help in developing a standardized description of attacks for easier review by stakeholders. ATT&CK delivers its data through the use of APIs and typically returns the results in a JSON data format.

Both CAPEC and ATT&CK provide details about adversarial behavior, each focusing on specific sets of use cases. CAPEC is more focused on application-level security, whereas ATT&CK is more focused on network defense, APTs, and threat hunting activities. Both CAPEC and ATT&CK can be cross-referenced in scenarios where CAPEC attack patterns are used by adversaries through specific techniques described in ATT&CK. This information enables contextual understanding and helps to understand how adversaries succeed in executing their objectives through various attack mechanisms. In turn, this information can be leveraged by system architects to better

understand the ways in which the systems that they are designing may be attacked and therefore can help in identifying security requirements that can mitigate such attacks.

3.2.3 Threat Intelligence Data Sources

Threat intelligence is evidence-based knowledge that is obtained from threat information that has been aggregated, transformed, analyzed, interpreted, or enriched to provide the necessary context for decision-making processes [JBW+16]. Threat intelligence data feeds are a critical part of modern cybersecurity efforts, providing several useful threat information attributes and relationships including threat actors, indicators of compromise, adversarial behaviors such as attack patterns, attack campaigns, and attacker tools, observed data related to system assets, possible courses of action in response to the threat, and vulnerabilities that could enable the realization of the threat. These data feeds can be delivered using different mechanisms and data formats including APIs and fixed-interval email subscriptions, in CSV or JSON formats [Bro16].

The Structured Threat Information eXpression (STIX) standard is considered as the most widely adopted standard for describing threat intelligence data [SSMB16]. STIX is a language and serialization format designed to support many different cyber threat management capabilities including analyzing cyber threats, specifying indicator patterns, managing response activities, and sharing cyber threat information [Bar14]. Its primary objective is to enable sharing machine-readable cyber threat intelligence to allow for a better understanding of potential security threats and attacks. STIX

objects are represented in JSON and categorize each piece of information with specific attributes to be populated. They are connected via *relationships* that characterize cyber threat intelligence information [OASa].

Consumers of STIX entries subscribe to a Trusted Automated Exchange of Intelligence Information (TAXII) server [OASb]. TAXII is an application protocol specifically designed to support the exchange of cyber threat intelligence represented in STIX [CDRS12].

Numerous commercial vendors of threat intelligence data feeds exist and include IBM X-Force Exchange [IBM], Anomali ThreatStream [Ano], and FireEye iSIGHT Intelligence [Fir], just to name a few.

Open-source threat intelligence feeds are also available. These feeds are compiled by open-source communities that monitor and aggregate threat-related data from activities such as vulnerability scanning or spam emails. Examples of open-source threat intelligence feeds include ProofPoint's Emerging Threats [Pro] and AlienVault's Open Threat Exchange (OTX) [Ali].

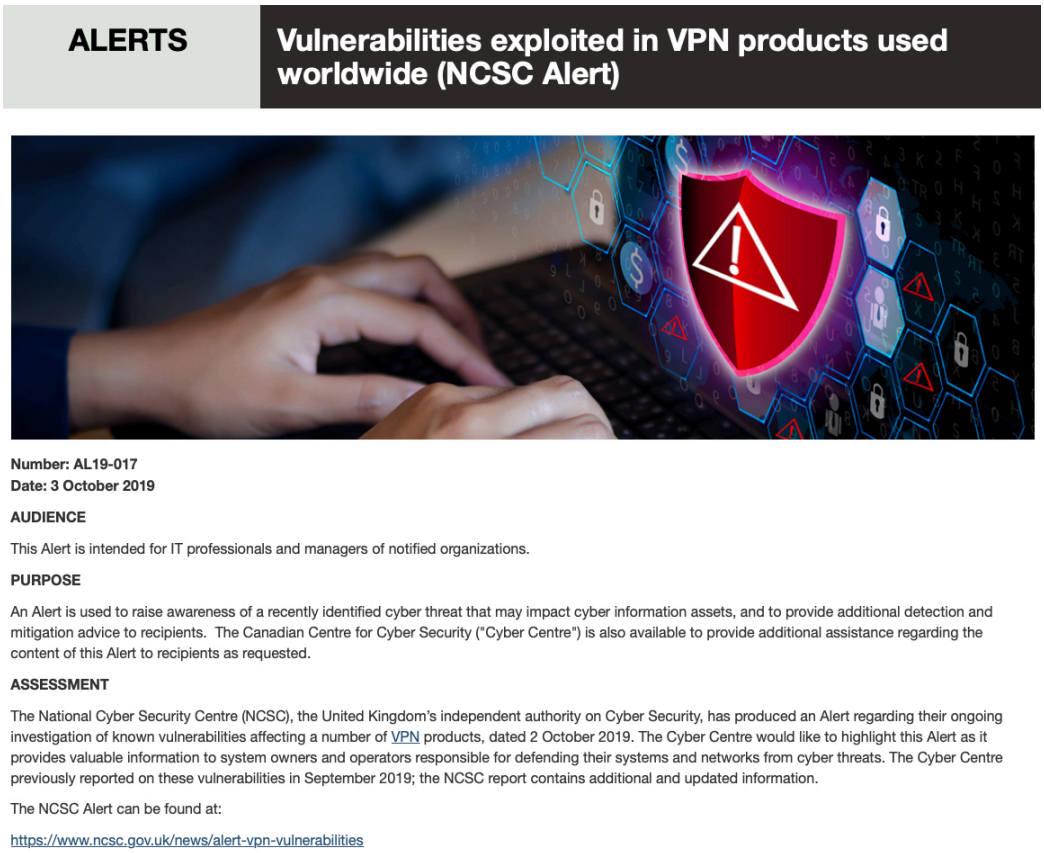
Threat intelligence data can be used at the design level to help understand what needs to be protected in a system. It can assist in identifying potentially critical assets that may not have been initially perceived as vulnerable. This can inform the elicitation of security requirements for which design solutions can be developed to ensure their satisfaction. The primary difficulty in leveraging threat intelligence data at the design level is in the analysis and translation of threat intelligence information

into actionable security requirements and design decisions. Tool support capable of automating these tasks is essential.

3.2.4 Alerts and Advisories From Government Agencies

Numerous government agencies around the world including the Canadian Centre for Cybersecurity (CCCS) [Can], the United States Cybersecurity & Infrastructure Security Agency (CISA) [Uni], the European Union Computer Emergency Response Team (CERT-EU) [Eur], the United Kingdom National Cyber Security Centre [U.K], the Australian Cyber Security Centre [Aus], and the New Zealand Computer Emergency Response Team (CERT-NZ) [New] publish alerts, advisories or notices regarding emerging threats and vulnerabilities. The purpose of these communications is to raise awareness of recently identified threats and vulnerabilities that may impact organizational assets and to provide additional detection and mitigation advice to recipients. Generally speaking, these alerts and advisories are delivered by email subscription or RSS feed subscription.

Alerts typically include a unique ID, a brief summary of the incident, threat, or vulnerability, technical details, and lists of products affected, additional risks and vulnerabilities, mitigations, and additional resources related to the incident, threat, or vulnerability. The list of mitigations is often categorized by topic areas such as Plans and Policies, and Best Practices related to the nature of the incident, threat, or vulnerability (e.g., Network, Ransomware, Malware, etc.). In some cases, alerts reference specific CVEs. In the CISA feeds [Uni], sometimes alerts also contain a



ALERTS **Vulnerabilities exploited in VPN products used worldwide (NCSC Alert)**

Number: AL19-017
Date: 3 October 2019

AUDIENCE

This Alert is intended for IT professionals and managers of notified organizations.

PURPOSE

An Alert is used to raise awareness of a recently identified cyber threat that may impact cyber information assets, and to provide additional detection and mitigation advice to recipients. The Canadian Centre for Cyber Security ("Cyber Centre") is also available to provide additional assistance regarding the content of this Alert to recipients as requested.

ASSESSMENT

The National Cyber Security Centre (NCSC), the United Kingdom's independent authority on Cyber Security, has produced an Alert regarding their ongoing investigation of known vulnerabilities affecting a number of VPN products, dated 2 October 2019. The Cyber Centre would like to highlight this Alert as it provides valuable information to system owners and operators responsible for defending their systems and networks from cyber threats. The Cyber Centre previously reported on these vulnerabilities in September 2019; the NCSC report contains additional and updated information.

The NCSC Alert can be found at:

https://www.ncsc.gov.uk/news/alert-vpn-vulnerabilities

Figure 3.3: An example alert from CCCS

MITRE ATT&CK profile for the incident, threat, or vulnerability. Figure 3.3 shows an alert relating to vulnerabilities affecting VPN products from CCCS.

Advisories are much less detailed than alerts and typically provide only a short description of often product-specific vulnerabilities. When an advisory details a recent security incident, several suggested actions are also listed. The suggested actions are often very high-level guidance and best practices such as implementing Two-factor Authentication (2FA) or conducting a vulnerability analysis. Despite not providing detailed information, such advisories can give hints to system architects about requirements and controls to consider.

Similar to threat intelligence, the information provided in alerts and advisories can be used by system architects to identify potentially critical assets and can be a source of information for data-driven design solutions that enable the dynamic selection of automated security controls. However, it can be difficult to automatically extract information in a machine-readable form from these alerts and advisories due to their varying levels of detail and granularity. This can often pose challenges for system architects to obtain actionable steps from these sources unless the alert or advisory is specifically related to systems or products that are part of the envisaged design.

3.2.5 Summary

There are a variety of external online data sources that provide different types of information that can be leveraged to support secure system design activities. Table 3.1 categorizes and summarizes these sources, the information they provide, the methods of data delivery, and the data formats.

3.3 Methodology

In this section, we present an approach to tackle the problem described in Section 3.1 by harnessing the various external data sources specified in Section 3.2 given the existing security requirements and design decisions for an asset of a system to ensure their adequacy.

Table 3.1: Summary of external online data sources containing known threat, vulnerability and attack information

Category	Sources	Information Provided	Delivery Method	Data Format
Vulnerability Data	NVD, CVE, CWE, IMPACT, WhiteSource, Notes, Exploit, Vulnerability Lab, VulDB	Vulnerabilities, weaknesses, affected product, affected vendor, attacker access, severity score, applicability definitions, how exploit, risk, tools, fixes, impact, issue summaries, code for penetration testing, vulnerability timeline, exploit prices, threat intelligence	API, HTTP, RSS, email subscription	CSV, HTML, JSON, Text, XML
Attack Pattern Data	CAPEC, ATT&CK	Attack patterns (approaches) and attributes (e.g., description, mechanism, related attack patterns, execution flow, preconditions, mitigations), TTPs (tactics, techniques, and procedures used in advanced persistent threats (APT))	API, HTTP	CSV, HTML, JSON, XML
Threat Intelligence	X-Force Exchange, ThreatStream, iSIGHT Intelligence, Emerging Threats, Open Threat Exchange	Evidence-based threat knowledge and relationships including threat actors, indicators of compromise, adversarial behaviors, possible responses to threats, vulnerabilities to threats	API, email subscription	CSV, JSON, Text
Alerts & Advisories	CCCS, CISA, CERT-EU, United Kingdom National Cyber Security Centre, Australian Cyber Security Centre, CERT-NZ	Alerts: unique identifier, brief summary of incident (or threat or vulnerability), technical details, products affected, other risks and vulnerabilities, mitigations, other related resources; Advisories: short description of product-specific vulnerabilities, suggested actions	RSS, email subscription	HTML, Text

The approach consists of five steps as depicted in Figure 3.4 which are elaborated below.

Capture Security Requirements and Design Decisions

For a given asset, we first capture its security requirements and design decisions (where available) from the system documentation, which may be in the form of specifications or design models. This information helps us determine, among other things, asset characteristics such as required security controls, and associated technologies. Assume

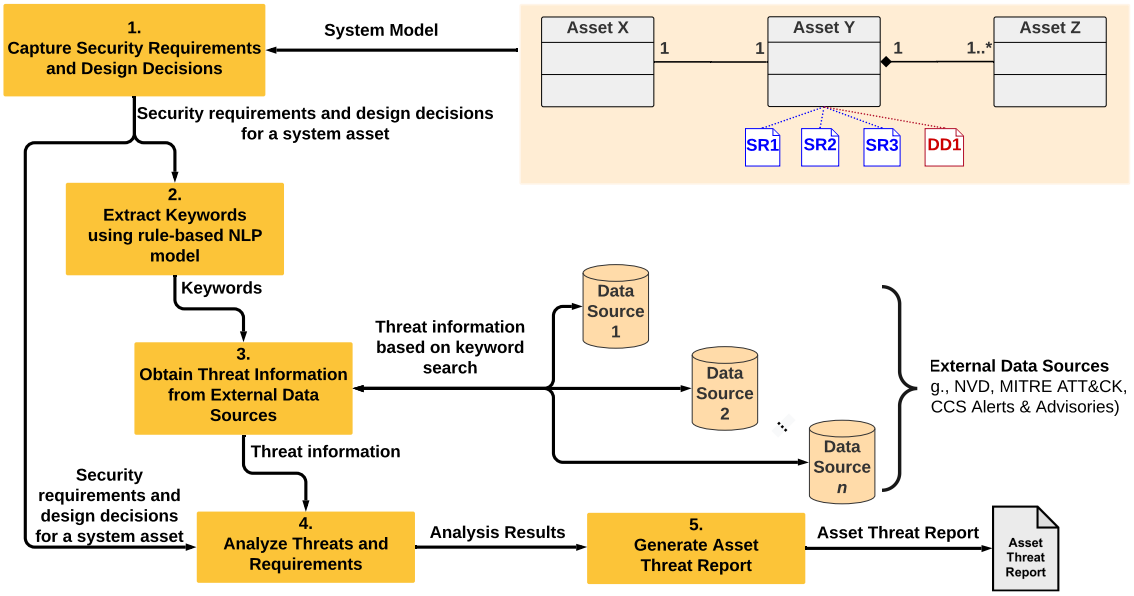


Figure 3.4: Overview of the proposed analysis approach which leverages external data sources to validate the adequacy of a set of security requirements

that SR1, SR2, and SR3 refer to the asset's security requirements and DD1 refers to the asset's design decisions.

Keyword Extraction

In this step, we process the provided requirements (SR1, SR2, SR3, and DD1) provided in Step 1 to extract relevant keywords. In this work, we chose to use NLP rather than a pre-defined list of relevant keywords as it broadens our overall vocabulary and increases our likelihood to detect relevant security-related keywords and technologies. To extract appropriate keywords, we pass the requirements text through a pre-trained English language NLP model that breaks down sentences into words and tags each word with a part-of-speech tag (such as verb or noun). We can then use these tags

to identify relevant keywords. For example, we can detect various technologies by identifying proper nouns in sentences. For identifying security controls, we filter the identified nouns in sentences by applying a rule-based matcher that looks for patterns of pre-defined security controls and returns valid matches. The result from this step is a list of keywords representing the security requirements and design decisions provided by the architect.

Obtain Threat Information

Given the keywords extracted in Step 2, we query a selection of external data sources from Section 3.2. This list of keywords will be used to gather relevant threats from the various external data sources to identify the relationships between the identified threats, security requirements, and design decisions. The more accurate the extraction of the keywords, the more precise our search and analysis will be for the asset analysis. While a generic off-the-shelf NLP model yielded usable results, it also extracted information that was not relevant to our analysis. For this reason, we developed a rule-based method to further filter relevant security control-related and technology keywords from the architect-provided data. Listing 3.1 shows the query response schema for a given asset. The *meta_data* includes information such as the API version, query time, and other query-specific details. The *statistics* contain additional information reflecting the outcomes of the analysis performed in Step 4.

```
{
  "meta_data": {...},
  "statistics": {...},
  "results": [ {
    "KEYWORD":{
      "nvd_results":{"meta_data":{}, "threats":[...]},
      "mitre_attck_results":{...},
      "cccs_alerts_results":{...}
    } } ]
}
```

Listing 3.1: JSON search response for a given asset for a KEYWORD

Analyze Threats and Requirements

In this step, we identify the various relationships that exist between the identified threats, security requirements, and design decisions. We first enrich the threat information for each threat received with knowledge inferred from this information describing the threat's impact on security objectives and the various security controls it prescribes. We do this by passing a threat's description to a NLP rule-based matcher that identifies if a threat is related to the confidentiality, integrity, and availability security objectives. For example, if a threat description discusses the tampering of data, the rule-based matcher flags the threat as affecting the integrity objective. Where explicit information of the threat's impact on security objectives is available such as in a threat's CVSS specification, we use that information instead for our classification. For simplicity, we consider the confidentiality, integrity, and availability objectives, but other objectives such as authenticity and accountability may be considered as well. We also pass the threat description through a rule-based matcher for identifying security controls specified in the threat. The security controls rule-based matcher is configured to recognize technical controls defined in ISO/IEC 27001 [Int18]. The

extracted list of controls is then compared with the initial security requirements to yield the following:

1. Identified threats addressed by the provided security requirements (i.e., *mitigated threats*)
2. Identified threats for which there are currently no security requirements (i.e., *unmitigated threats*)
3. Provided security requirements for which no relevant threats were found (i.e., potentially *extraneous requirements*)

For instance, to identify unmitigated threats, we compute the set difference between the set of security requirements extracted from the external threat data related to the design decisions and the set of security requirements provided by the architect. The result is the design-related threats for which a new or modified security requirement may be required.

Generate Report

The results obtained in Step 4 are presented as an interactive report. The main goal of this report is to help the architect navigate through an otherwise overwhelming amount of information in a more feasible manner. The report is separated into two sections: at-a-glance and threat information. The at-a-glance section provides a series of metrics summarizing the results of the data analysis. These metrics are the number of threats mitigated, number of threats unmitigated, number of extraneous

requirements, and number of threats that affect the confidentiality, integrity, and availability security objectives. We also present the keywords identified from the given security requirements and design decisions that were used to query the data sources. The threat information section elaborates on the threats identified for a given asset. This is where the system architect can review individual threat information.

3.4 Merak: An Asset Threat Analysis Tool

To automate the approach presented in Section 3.3, we developed Merak[1], an asset threat analysis tool that leverages external data sources to help architects design secure systems.

The primary requirements for Merak were as follows:

1. Capture security requirements and design decisions for a given system asset.
2. Generate an interactive report with data-driven metrics and threat information for that asset from various data sources.

Merak fulfils these requirements by providing a simple form interface for architects to enter their security requirements and design decisions for an asset of their system. Merak performs NLP on the given text to extract useful keywords that can be used to query the various data sources. Merak currently uses NVD, ATT&CK, and CCCS

[1]Merak is a star in the Ursa Major constellation. It is commonly referred to as a "pointer star" as it is helpful for finding Polaris, also known as the North Star.

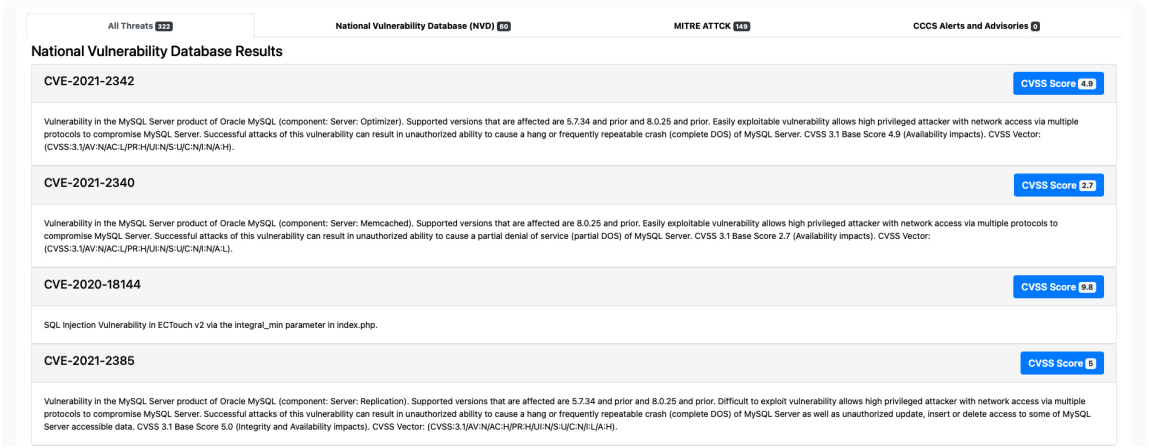


Figure 3.5: A example the web-based report in Merak showing the threat information for an asset

Alerts and Advisories, although any number of external data sources could be integrated in the future. Merak generates an interactive web-based report to help the architect navigate through the threat information which is summarized and categorized based on the different data sources. An example of the report in Merak is shown in Figure 3.5. Since Merak performs all this expensive computation on the server-side, caching is implemented for the results to minimize processing times.

Merak is available at `https://merak.compass.carleton.ca/` for practitioners to estimate the known threat landscape for their assets.

3.5 Evaluating Threats for the OSM Web Server

In this section, we illustrate how Alice can apply our threat analysis approach presented Section 3.3 to analyze the security requirements and design decisions for the Web Server of the OSM system (presented in Section 1.3) using Merak.

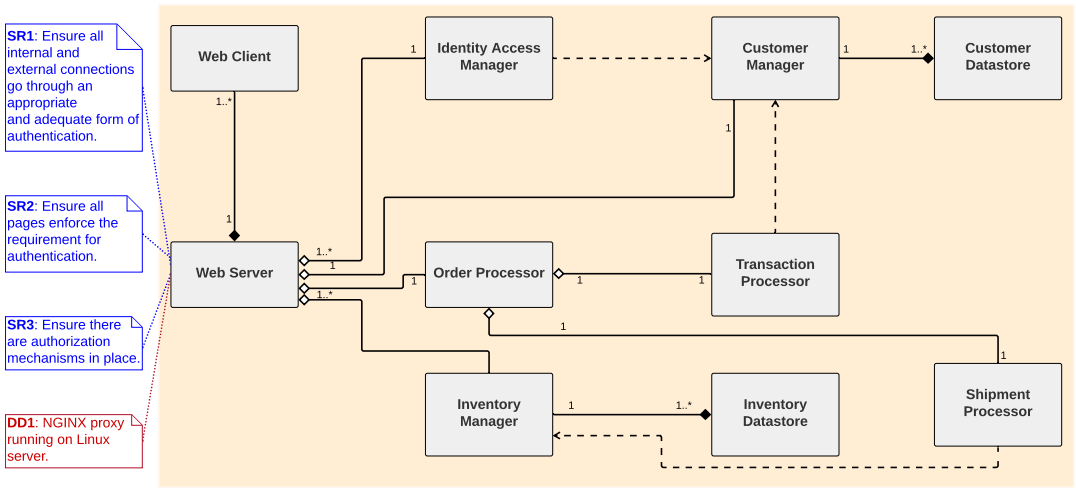


Figure 3.6: Merak threat report generated for the OSM Web Server

Capture Security Requirements and Technologies

Alice starts by describing the security requirements (SR1, SR2, and SR3) and design decisions (DD1) for the Web Server in the OSM system. She annotates her system model as shown in Figure 3.6.

To initiate her analysis, Alice enters the security requirements and design decisions through simple form inputs in Merak which, upon submission will trigger the analysis and generate an interactive report similar to the one shown in Figure 3.7. Since we elaborated on the Merak application in Section 3.4, we illustrate this example using Merak's API which processes the requirements and design decisions and returns the analysis results in JSON. To begin the analysis, the JSON query shown in Listing 3.2 with the security requirements and design decisions associated with the OSM Web Server is submitted for processing.

```
{
  "security_requirements": "Ensure all internal and external
     connections for the server go through an appropriate and
     adequate form of authentication. All protected pages must
     enforce the requirement for authentication and authorization.",
  "design_decisions": "Web server uses NGINX proxy and Linux."
}
```

Listing 3.2: JSON query for the OSM Web Server

Keyword Extraction

In this step, the JSON query formulated in Step 1 is processed using an NLP model. The keywords extracted based on the security requirements and design decisions from Listing 3.2 are **authentication**, **authorization**, **NGINX**, and **Linux**.

Obtain Threat Information

Given the keywords extracted in Step 2, a selection of external data sources from Section 3.2 are queried. Recall that Merak uses NVD, ATT&CK, and CCCS Alerts and Advisories. As each data source presents its data using a different schema, Merak formats the results into a uniform schema as shown in Listing 3.3. The result of this step is a JSON object containing all the threats separated based on their source.

Analyze Threats and Requirements

Listing 3.3 shows the query response schema for the OSM Web Server. The *meta_data* includes information such as the API version, query time, and keywords

extracted. The *statistics* attribute contains additional information reflecting the outcomes of the analysis performed in Step 4.

Generate Report

An interactive report is generated for Alice to present the results obtained in Step 4. For the Web Server, the generated report is shown in Figure 3.7. The main goal of this report is to help the architect navigate through an otherwise overwhelming amount of information in a more feasible manner.

The generated threat analysis report for the Web Server shows that out of 7159 threats discovered based on the initial security requirements and design decisions, 2448 threats affect confidentiality, 2139 threats affect integrity, and 2008 threats affect the availability of the Web Server. As a reminder, a threat can affect multiple security objectives. These counters help Alice gauge the extent to which the various security objectives of the asset are impacted by known threats during the system design phase. It is important to note that the 7159 total threats refer to threats for which there is sufficient information (i.e. a description of the threat is available) to be useful for Alice to act on. The report also provides a breakdown of the keywords extracted from the security requirements and design decisions. Additionally, the report provides a summary of the various threat-requirement relationships described in Step 4 in Section 3.3, identifying 519 mitigated threats, 131 unmitigated threats, and no extraneous requirements. Here, the mitigated threats refer to the threats associated with the security controls “authentication” and “authorization”. The 131

```
{
  "meta_data": {
        "api_version": "1.1.0-beta",
        "query_time_seconds": 4,
        "keywords": {
            "requirements_keywords": [
                "authentication",
                "authorization"
            ],
            "technology_keywords": [
                "NGINX",
                "Linux"
            ]
        }
    },,
    "statistics": {
        "mitigated_threats_count": 519,
        "unmitigated_threats": [
            "2FA",
            "SSL",
            "encryption",
            "logging",
            ...
        ],
        "unmitigated_threats_count": 119,
        "extraneous_controls": []
    },
  "results": [ {
    "NGINX":{
      "nvd_results":{"meta_data":{}, "threats":[...]},
      "mitre_attck_results":{...},
      "cccs_alerts_results":{...}
    } } ]
}
```

Listing 3.3: An excerpt of the JSON search response for the OSM Web Server keyword "NGINX"

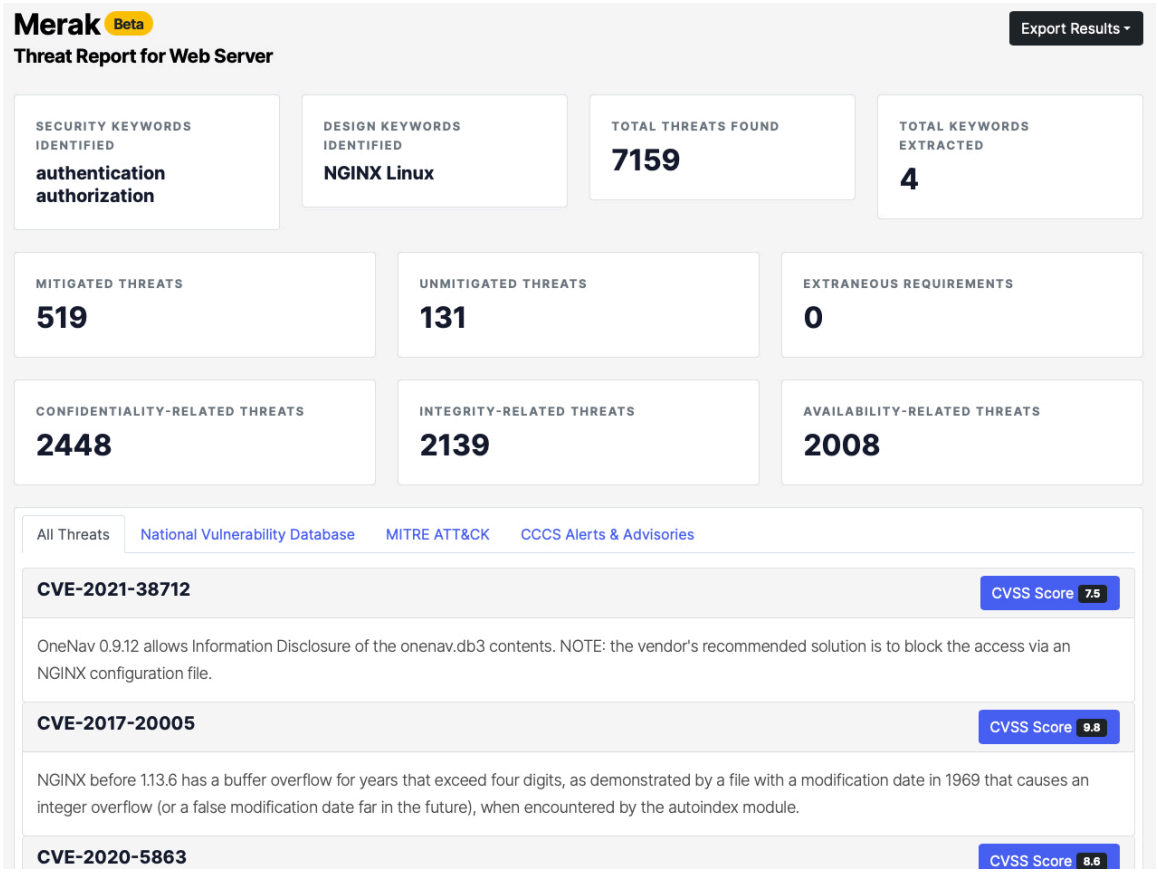


Figure 3.7: Merak threat report generated for the OSM Web Server

unmitigated threats are threats affecting the technologies specified in the design decisions. These threats prescribe security controls not specified in the current security requirements, such as Two-factor Authentication (2FA) and logging. Since both security controls identified from our requirements have a number of associated threats, no extraneous threats were found. The report also provides the identified list of threats organized by their source which further helps Alice review the specifics of the threats discovered.

3.6 Discussion

Our approach assists system architects like Alice in identifying known threats from external data sources which may warrant amendments to existing security requirements to enhance the security of the system asset. Our approach also helps in identifying potentially extraneous requirements that do not appear to address or mitigate any known threats and thereby may not be necessary for the system. This type of information enables the architect to prioritize the unmitigated threats, especially under time and budget constraints, improving development efficiency. Despite that some identified threats may not apply to the system, our approach helps distill large amounts of data into an actionable form for the architect. For example, results from external data sources given a keyword such as NGINX can commonly exceed tens of thousands of threats and it is unreasonable for architects to sift through this amount of information for each asset of their system. To reduce this volume of data, our approach presents only the threats that affect the asset for which no security controls have been mentioned within the security requirements. Therefore, our approach provides much-needed support for the architect in these areas.

In developing the proposed approach we faced several challenges. One challenge was the aforementioned data size. We developed a script to automate the process of classifying and categorizing threats from external sources into a database to minimize the response time of the results. Another challenge was the inconsistencies with the data from the various sources. We developed a unified, simple schema as shown in Listing 3.1 to capture sufficient information in a flexible format for further analysis and applications. The approach is dependent on the quality and detail of the security

requirements and design decisions provided as input. If these inputs are poor, the approach will not yield quality results. Lack of clarity in the input to our approach will raise difficulties in the keyword extraction and subsequent threat and requirements analysis as the search queries to the external data sources may be insufficient to obtain any meaningful results.

In this work, we used a general language NLP model and added a rule-based matcher to detect relevant keywords, classify threats, and identify security controls. However, this rule-based NLP approach is limited to the scope of the rules we defined. For simplicity, we assumed security-related keywords to be unigrams (one-word sequence) or bigrams (two-word sequence). Our use of NLP in this work is primarily to use contextual information of requirements and threats to enhance the extraction of relevant keywords, threats, or security controls. Furthermore, employing NLP helps us analyze security requirements written in natural language making it easier for practitioners to apply our approach to their existing workflows.

3.7 Conclusions

In this chapter, we explored how external data sources, such as vulnerability databases, attack patterns, threat intelligence data feeds, and security alerts and advisories can be leveraged to assist system architects like Alice in validating the adequacy of their security requirements and to support them in making more informed design decisions. This enables them to build systems that have appropriate controls capable of mitigating the known threats and vulnerabilities that affect the system assets. We also

identified and described the currently available external data sources that provide useful information for this purpose. We also presented an NLP-based approach that uses available design documentation including initial security requirements and design decisions. The approach produces a report that summarizes which threats to the system design identified from external data are mitigated by existing security requirements, which threats are not yet mitigated by any existing security requirements, and which requirements may be extraneous. This information can enable system architects to make more informed design decisions in terms of how best to mitigate the threats/vulnerabilities, leading to modified or new requirements. This approach also demonstrates that external threat, vulnerability, and attack data can support secure software design activities and address security concerns early in the SDLC, namely at the design phase, when it is less costly to make changes.

With our threat analysis approach, Alice can now tap into the vast amounts of openly available security data sources to determine the adequacy of her security requirements and better understand the threat landscape for a given asset of her system. Furthermore, Alice can apply our approach iteratively to support incremental development and improvement of the asset and the system overall. This ability to iterate and evaluate the asset's threat landscape helps increase her confidence that the last iteration of security requirements is adequate in support of security evaluation and assurance activities. Our threat analysis approach, therefore, addresses the challenge of lack of security proficiency among security architects that we discussed in Section 1.4.

The metrics derived from data sources in this work form the 'data-driven' aspect for our security evaluation approach to evaluate the security of a system's design. In the

following chapter, we look at existing security metrics that can be incorporated into our security evaluation approach.

Chapter 4

Evaluating Existing Security Metrics for Soundness

To support decision-making in secure system design activities, we can use security metrics. For example, Alice can use various security metrics to indicate whether she is improving the security of her system design. However, the metrics that she uses must be trustworthy so she can make informed and meaningful decisions about her design. This criterion of 'trustworthiness' for a security metric is commonly referred to as *soundness* in security literature. In this chapter, we formally evaluate the soundness of security metrics obtained through vulnerability scoring frameworks. We first discuss our evaluation methodology for asserting if a security metric is sound in Section 4.1. In Section 4.2, we evaluate five vulnerability scoring frameworks reported in the literature that have found widespread use in security decision-making processes or that have been proposed for such purposes. Based on our findings, we provide

several recommendations in Section 4.4 to help architects apply the frameworks in ways that can lead to obtaining sound security metrics. Section 4.5 presents our conclusions.

4.1 Evaluation Methodology

The Method for Designing Sound Security Metrics (MDSSM) proposed in [Yee19a] provided the soundness criteria to evaluate existing security metrics. MDSSM is a rigorous step-by-step method for designing sound security metrics [Yee19b]. It can also be used to test the soundness of a security metric. This is done by adapting MDSSM and verifying whether the security metric in question exhibits each of the characteristics of a sound security metric mentioned below.

To be consistent with the terminology of MDSSM, the term *quantity* refers to the value or set of values obtained by following the process prescribed by the vulnerability scoring framework. In this work, the quantity refers specifically to a security metric obtained via a vulnerability scoring framework.

Following the general approach described by MDSSM [Yee19b], we define the following properties of a security metric m obtained via a vulnerability scoring framework:

1. Definition Property: Defn(m) is satisfied if and only if m does not leave room for misinterpretation (i.e., it is objective), does not inherently resolve to a pre-determined outcome (i.e., it is unbiased), and is associated with the security level of a system

(i.e., it is meaningful) while ensuring that it can be obtained without incurring any undue hardship or costs.

2. Sufficiency Property: Suff(m) is satisfied if and only if m has a direct impact on the system security level, an increase in m results in a consistent increase (or decrease) in the system security level, and all aspects needed to effectively measure a vulnerability are incorporated in the definition of the metric m.

3. Progression Property: Prog(m) is satisfied if and only if the metric m, when evaluated over a long period, converges to an acceptable or maximal security level provided adequate mitigations are implemented.

4. Reproducibility Property: Repr(m) is satisfied if and only if m can be reproduced and verified by third parties (i.e., they can independently assign values to obtain the same quantity and/or reach the same conclusion regarding the system security level using the same procedure).

In addition to the above properties, MDSSM also considers *divisibility* (i.e., whether the quantity is expressible mathematically in terms of other constituent quantities) as a design criterion for sound security metrics. However, in our work when applying MDSSM for testing the soundness of existing security metrics, divisibility does not apply since an existing security metric is already in its final form as mentioned in [Yee19b]. For this reason, we do not further consider the divisibility of security metrics in our evaluation.

The above properties formed the evaluation criteria for our work [SAJ20]. Taken together, they enable the formalization of a necessary and sufficient condition on the

Security Metric Soundness Evaluation Worksheet				
MDSSM Criteria	**Criteria Description**		**Evaluation (Yes/No)**	**Explanation**
1. Definition	**1.1**	Is the quantity objective?		
	1.2	Is the quantity unbiased?		
	1.3	Is the quantity meaningful?		
	1.4	Can the quantity be obtained without undue hardship or costs?		
2. Sufficiency	**2.1**	Does the quantity have a direct impact on the security level?		
	2.2	Does an increase in the quantity result in a consistently increase (or decrease) in the system security level?		
	2.3	Are all aspects needed to effectively measure the system incorporated in the definition of the quantity?		
3. Progression	**3.1**	Does the quantity progress to an acceptable or maximal security level provided adequate mitigations are implemented?		
4. Reproducibility	**4.1**	Can third-party verifiers obtain the same quantity and/or reach the same conclusion using the same procedure?		
Conclusion	The quantity is a sound security metric			
Source(s) of Evaluation				

Figure 4.1: Security metric soundness evaluation spreadsheet tool

soundness of a security metric obtained via vulnerability scoring framework as given in Definition 4.1.1.

Definition 4.1.1 (Soundness)**.** *A security metric m obtained via a vulnerability scoring framework is sound if and only if it satisfies each of the definition, sufficiency, progression, and reproducibility properties, i.e.,*

$$\text{Defn}(m) \ \wedge \ \text{Suff}(m) \ \wedge \ \text{Prog}(m) \ \wedge \ \text{Repr}(m) \iff \text{Sound}(m)$$

To facilitate our evaluation, we developed a security metric soundness evaluation spreadsheet tool as shown in Figure 4.1. The spreadsheet tool is designed as a series of Yes/No questions that enable the evaluation of Definition 4.1.1. The security

metric soundness evaluation tool is available at `https://carleton.ca/cybersea/security-metrics-soundness-evaluation-tool/`.

4.2 Soundness Evaluation

This section evaluates the soundness of the security metrics associated with vulnerability scoring frameworks that have been used in security decision-making processes or that have been proposed for such purposes. In our evaluation, we assume that evaluators are honest, meaning that they are trying to accurately reflect the severity of a vulnerability using the framework without any prejudice. For each framework, we provide a brief overview of the derivation of the security metric and evaluate the satisfaction of each of the definition, sufficiency, progression, and reproducibility properties to conclude whether the security metric is sound.

4.2.1 Common Vulnerability Scoring System

CVSS is largely considered the de facto standard for quantifying and assessing the severity and risk of security vulnerabilities in computing systems. CVSS is designed to measure the fundamental characteristics of vulnerabilities to convey their severity to help prioritize an appropriate response [MSR07].

The *CVSS score* of a vulnerability is computed using a pre-defined formulation of metrics as shown in Figure 4.2. Metrics are categorized under three distinct groups. The *base metrics* group reflects the fundamental aspects of a vulnerability that are

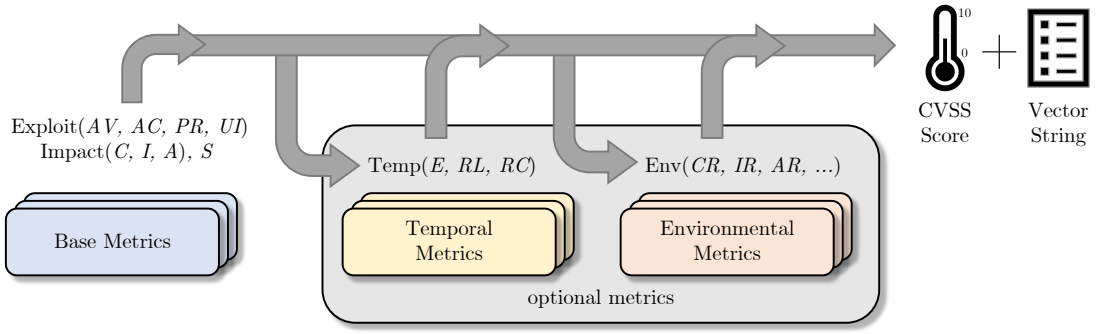


Figure 4.2: CVSS metrics and equations [MSR07]

constant over time and across user environments, including: access vector (AV), attack complexity (AC), privileges required (PR), user interaction (UI), confidentiality impact (C), integrity impact (I), availability impact (A), and scope (S). The *temporal metrics* group reflects a set of optional time-dependent characteristics of a vulnerability, including: exploit code maturity (E), remediation level (RL), and report confidence (RC). Lastly, the *environmental metrics* group reflects a set of optional user environment-dependent characteristics of a vulnerability, including: confidentiality requirement (CR), integrity requirement (IR), availability requirement (AR), and modified base metrics.

The base metrics must be assigned values by an analyst which will then be used to compute a score ranging from 0.0 to 10.0. The base score can then be refined by assigning additional values for the temporal and environmental metrics. The detailed CVSS equations can be found in the CVSS specifications [FIR19]. In CVSS, all values should be assigned under the assumption that the attacker has already located and identified the vulnerability [MSR07].

Evaluation

We evaluate *CVSS Version 3.1 Release 1* [FIR19]. In this case, the *quantity* is the final CVSS score.

Definition Property: CVSS scores are rather straightforward to compute. They do not inherently favour a certain outcome and therefore are considered unbiased. They are also meaningful in that high CVSS scores (7.0-10.0) indicate a high severity and therefore can be used to prioritize mitigation efforts. However, each of the base, temporal, and environmental metric groups requires the assignment of qualitative inputs such as 'Low', 'Medium', and 'High' that are susceptible to subjectivity that may result in variations in the overall CVSS score of a vulnerability. The subjectivity of CVSS scores has been criticized in the past [WXZ08, YMR16, WGSS11, Ker16]. While CVSS tackles this by indicating that these assignments should be done by security experts and by attributing a qualitative rating for a range of scores, this is not sufficient to mitigate this issue. Thus, the definition property is not satisfied.

Sufficiency Property: It is easy to see that an increase in the CVSS score indicates an increase in the severity of the vulnerability. For example, a CVSS score between 9.0-10.0 reflects a severity rating of critical for a vulnerability while a score between 0.1-3.9 represents a low severity rating. In turn, this means the presence of vulnerabilities with high CVSS scores results in a decrease in the overall system security level. Furthermore, the sets of base, temporal, and environmental metrics appear to contain all necessary aspects required to measure the severity of a known vulnerability and are incorporated in the calculation of the CVSS score. Thus, the sufficiency property is satisfied.

Progression Property: In the formulation of the CVSS score, the temporal metrics group accommodates the possibility of patches or updates to address a vulnerability of the system. By implementing such mitigations and recalculating the CVSS score with updated values, we can reduce the CVSS score. Thus, the progression property is satisfied.

Reproducibility Property: As shown in Figure 4.2, CVSS generates a vector string based on the values used to generate a particular score. Assuming that this vector string remains constant for a given vulnerability, the CVSS score can easily be reproduced and verified by third-party verifiers. However, it must be noted that, in the case that third-party verifiers must independently assign values to the base, temporal and environmental metrics of CVSS, it is possible that they will obtain different quantities. This is a consequence of the subjectivity that can be introduced in the assignment of values as noted above. While this subjectivity may result in different quantities, they are unlikely to vary so much that it would yield a different conclusion. Thus, we consider the reproducibility property to be satisfied.

Soundness Decision

The completed soundness evaluation for CVSS is shown in Figure 4.3. While CVSS scores satisfy the sufficiency, progression, and reproducibility properties, they do not satisfy the definition property. This is primarily due to the subjectivity associated with assigning the values used to compute the score. The resulting variations can lead to distorted severity ratings which can impact security-related decision-making. Thus, CVSS scores are not sound.

MDSSM Criteria	Criteria Description		Evaluation (Yes/No)
1. Definition	1.1	Is the quantity objective?	No
	1.2	Is the quantity unbiased?	Yes
	1.3	Is the quantity meaningful?	Yes
	1.4	Can the quantity be obtained without undue hardship or costs?	Yes
2. Sufficiency	2.1	Does the quantity have a direct impact on the security level?	Yes
	2.2	Does an increase in the quantity result in a consistently increase (or decrease) in the system security level?	Yes
	2.3	Are all aspects needed to effectively measure the system incorporated in the definition of the quantity?	Yes
3. Progression	3.1	Does the quantity progress to an acceptable or maximal security level provided adequate mitigations are implemented?	Yes
4. Reproducibility	4.1	Can third-party verifiers obtain the same quantity and/or reach the same conclusion using the same procedure?	Yes
Conclusion	The quantity is a sound security metric		No

Figure 4.3: CVSS soundness evaluation

4.2.2 OCTAVE Allegro

Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) provides a framework for performing security assessments oriented around organizational objectives. In particular, OCTAVE Allegro is a methodology focusing on identifying and analyzing risks (vulnerabilities) to the information assets (e.g., databases, emails, etc.) of an organization. The methodology provides guidelines and worksheets to perform a comprehensive information security risk assessment as shown in Figure 4.4.

Our evaluation will focus primarily on Step 7, which qualitatively measures the extent of impact faced by an organization when a threat scenario is realized. The result of

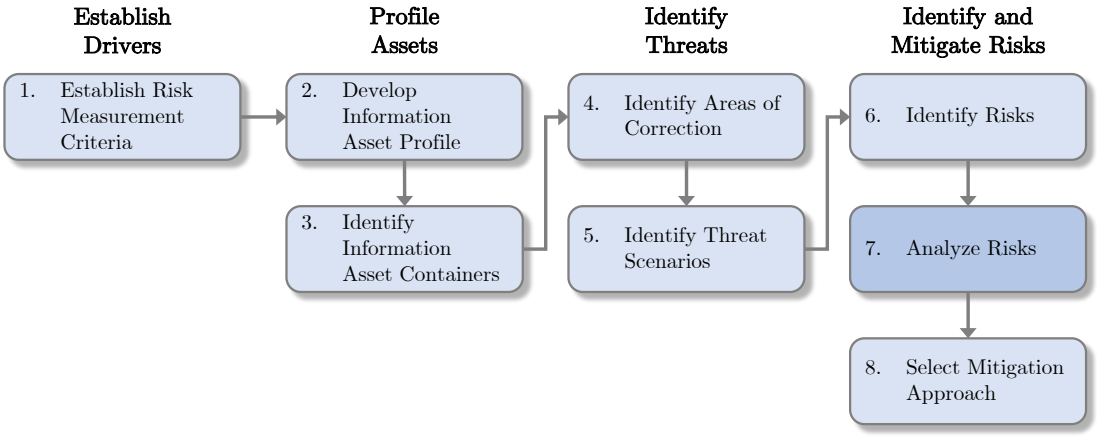


Figure 4.4: OCTAVE Allegro methodology [CSYW07]

this step is a *relative risk score* for each risk to each information asset. The calculation involves using the risk measurement criteria defined in Step 1 of Figure 4.4 to evaluate the consequence relative to each of the impact areas and record an *impact value* of "high = 3," "medium = 2," or "low = 1." An *impact area* is an overall mission or business objective of the organization (e.g., reputation, safety, financial, etc.). These impact areas are ranked in order of importance with the most important area receiving the highest rank. The relative risk score is then computed for each impact area by multiplying the impact area rank by the impact value and totalling the scores for each impact area. This scoring information is then used to prioritize mitigation actions in Step 8.

Evaluation

We evaluate *OCTAVE Allegro v1.0* [CSYW07]. In this case, the *quantity* is the relative risk score for an information asset as computed in Step 7. OCTAVE Allegro relies on the organization to define its impact areas, rank the importance of those

impact areas, and assign impact values based on the organization's objectives. Our evaluation below assumes that the adaptation of OCTAVE Allegro remains faithful to the guidelines provided in the framework specification [CSYW07].

Definition Property: Impact areas and impact values are determined by the organization based on its business objectives. OCTAVE Allegro's guidelines explicitly outline the scope of the impact values alongside the corresponding interpretation of those values. For this reason, the resulting relative risk score is objective and unbiased. The relative risk score values are associated with the risks corresponding to the exploitation of vulnerabilities or threats and their effect on an impact area. Thus, the relative risk score is meaningful. Lastly, ranking impact areas and assigning impact values is straightforward. Thus, the definition property is satisfied.

Sufficiency Property: If the guidelines are followed correctly, the security level represented by the relative risk score consistently decreases as the score increases. A higher relative risk score risk must be given higher priority when considering mitigation actions. Provided that the organization develops a comprehensive list of potential impact areas to ensure the score is reflective of their organizational interests, the calculation of the relative risk score will be based on the necessary aspects required to measure the severity of a vulnerability in an information asset. Thus, the sufficiency property is satisfied.

Progression Property: Addressing a vulnerability through mitigation techniques leads to a reduction in its relative risk score. For example, implementing a mitigation for a vulnerability is expected to reduce the potential impact to an information asset if exploited. This will lead to a recalculation of the score with lower impact values

which will result in a reduced relative risk score. Thus, the progression property is satisfied.

Reproducibility Property: Because OCTAVE Allegro does not standardize the impact areas used to compute the relative risk score, reproducibility cannot be guaranteed. This is due to the fact that unless there is consistency in the specific set of impact areas used to calculate the relative risk score, or extensive knowledge about the organization and assets to ensure that independent third parties will arrive at the same set of impact areas, there will be variations in the obtained results. Thus, the reproducibility property is not satisfied.

Soundness Decision

The completed soundness evaluation for OCTAVE is shown in Figure 4.5. Assuming the faithful adaptation of the OCTAVE Allegro guidelines, the relative risk score satisfies the definition, sufficiency, and progression properties. However, the reproducibility property is not satisfied due to the lack of standardization of the impact areas used to compute the relative risk score. Thus, the relative risk score obtained from OCTAVE Allegro is not sound.

4.2.3 CSE Harmonized Threat and Risk Assessment

The Communications Security Establishment (CSE) Harmonized Threat and Risk Assessment (TRA) methodology [Com07] was developed to gather various assets, determine their current exposures, and evaluate if existing safeguards are sufficient with

MDSSM Criteria	Criteria Description		Evaluation (Yes/No)
1. Definition	1.1	Is the quantity objective?	Yes
	1.2	Is the quantity unbiased?	Yes
	1.3	Is the quantity meaningful?	Yes
	1.4	Can the quantity be obtained without undue hardship or costs?	Yes
2. Sufficiency	2.1	Does the quantity have a direct impact on the security level?	Yes
	2.2	Does an increase in the quantity result in a consistently increase (or decrease) in the system security level?	Yes
	2.3	Are all aspects needed to effectively measure the system incorporated in the definition of the quantity?	Yes
3. Progression	3.1	Does the quantity progress to an acceptable or maximal security level provided adequate mitigations are implemented?	Yes
4. Reproducibility	4.1	Can third-party verifiers obtain the same quantity and/or reach the same conclusion using the same procedure?	No
Conclusion	The quantity is a sound security metric		No

Figure 4.5: OCTAVE soundness evaluation

regards to confidentiality, integrity, or availability. The Harmonized TRA Methodology is designed to effectively filter out the false positives inherent in general vulnerability assessments by ensuring evaluated assets are up-to-date with the latest set of stable signatures and patches.

Vulnerabilities are primarily assessed in the context of their impact by considering the probability of compromise and the severity of the outcome. The probability of compromise is assigned a value ranging from "Low" to "High," or "Not Applicable" if it does not perform a prevention function according to Table 4.1. Similarly, the severity of the outcome is assigned a value ranging from "Low" to "High," or "Not Applicable" if it does not perform a detection, response, or recovery function according to Table 4.2.

Table 4.1: Vulnerability impact on probability of compromise [Com07]

Safeguard Effectiveness	**Associated Vulnerabilities**	**Probability of Compromise**
No Safeguards Safeguards Largely Ineffective Probability of Compromise > 75%	Easily Exploited Needs Little Knowledge/Skill/Resources Assets Highly Accessible Assets Very Complex/Fragile/Portable Employees Ill-Informed/Poorly Trained	High
Safeguards Moderately Effective Probability of Compromise 25-75%	Not Easily Exploited Needs Some Knowledge/Skill/Resources Assets Moderately Accessible Assets Fairly Complex/Fragile/Portable Moderate Employee Awareness/Training	Medium
Safeguards Very Effective Probability of Compromise < 25% (Safeguard Performs Only Detection, Response or Recovery Functions)	Difficult to Exploit Needs Extensive Knowledge/Skill/Resources Access to Assets Tightly Controlled Assets Very Simple/Robust/Static Employees Well-Informed/Trained	Low (N/A)

Table 4.2: Vulnerability impact on severity of the outcome [Com07]

Safeguard Effectiveness	**Associated Vulnerabilities**	**Severity of Outcome**
No Safeguards Safeguards Largely Ineffective Assets Exposed to Extensive Injury	Unlikely to Detect Compromise Damage Difficult to Contain Prolonged Recovery Times/Poor Service Levels Assets Very Complex/Fragile Employees Ill-Informed/Poorly Trained	High
Safeguards Moderately Effective Assets Exposed to Moderate Injury	Compromise Probably Detected Over Time Damage Partially Contained Moderate Recovery Times/Service Levels Assets Fairly Complex/Fragile Moderate Employee Awareness/Training	Medium
Safeguards Very Effective Assets Exposed to Limited Injury (Safeguard Performs Only a Prevention Function)	Compromise Almost Certainly Detected Quickly Damage Tightly Contained Quick and Complete Recovery Assets Very Simple/Robust Employees Well-Informed/Trained	Low (N/A)

Table 4.3: CSE overall vulnerability ratings [Com07]

Impact on	**Impact on Probability of Compromise**		
Severity of Outcome	**Low (N/A)**	**Medium**	**High**
High	Medium	High	Very High
Medium	Low	Medium	High
Low (N/A)	Very Low	Low	Medium

The Harmonized TRA Methodology provides an overall vulnerability rating according to Table 4.3. For example, a vulnerability receives the rating "Very High" if it has a high probability of compromise and high severity of the outcome. These vulnerability ratings are then used for prioritizing the vulnerabilities which require the most attention.

Evaluation

We evaluate the vulnerability metrics outlined in the CSE *Harmonized TRA Methodology* [Com07]. In this case, the *quantity* is the overall vulnerability rating as computed in Table 4.3.

Definition Property: The simple determination of the vulnerability rating as part of the Harmonized TRA Methodology explicitly characterizes the severity of a vulnerability without inherently favoring a certain outcome. Therefore, the vulnerability rating is meaningful and unbiased. However, the vulnerability impact on the probability of compromise and severity of the outcome (Tables 4.1 and 4.2) uses many ambiguous qualifying terms such as "fairly," "extensive," and "largely." As a result, the definitions of the associated impact ratings leaves room for subjectivity as there are no specific characterizations of how such qualifying terms shall be interpreted.

This means it is possible for different assessors to misinterpret these classifications which can impact the overall vulnerability rating. Thus, the definition property is not satisfied.

Sufficiency Property: It is clear that the presence of vulnerabilities with high vulnerability ratings results in a decrease in the overall system security level. Moreover, the probability of compromise and the severity of the outcome are standard aspects for calculating vulnerability exposure and therefore it is appropriate to consider these aspects as complete for measuring the severity of a known vulnerability. Thus, the sufficiency property is satisfied.

Progression Property: Adequate mitigations for a given vulnerability will undoubtedly reduce the probability of compromise and/or the severity of the outcome. Therefore, it is easy to see that the implementation of such mitigations will enable the reduction of the vulnerability rating to an acceptable level. Thus, the progression property is satisfied.

Reproducibility Property: Assuming that there is a common understanding of the potentially ambiguous terms in the standardized aspects used to determine the vulnerability rating (i.e., the probability of compromise and the severity of the outcome), the Harmonized TRA Methodology enables the results to be reproduced and verified by third-party verifiers. However, the room for subjectivity in the interpretation of values assigned for the probability of compromise and the severity of the outcome can limit this reproducibility and verifiability, and lead to varying quantities. However, it is unlikely to result in such a wide variation that it would yield different conclusions. Thus, we consider the reproducibility property to be satisfied.

MDSSM Criteria	Criteria Description		Evaluation (Yes/No)
1. Definition	1.1	Is the quantity objective?	No
	1.2	Is the quantity unbiased?	Yes
	1.3	Is the quantity meaningful?	Yes
	1.4	Can the quantity be obtained without undue hardship or costs?	Yes
2. Sufficiency	2.1	Does the quantity have a direct impact on the security level?	Yes
	2.2	Does an increase in the quantity result in a consistently increase (or decrease) in the system security level?	Yes
	2.3	Are all aspects needed to effectively measure the system incorporated in the definition of the quantity?	Yes
3. Progression	3.1	Does the quantity progress to an acceptable or maximal security level provided adequate mitigations are implemented?	Yes
4. Reproducibility	4.1	Can third-party verifiers obtain the same quantity and/or reach the same conclusion using the same procedure?	Yes
Conclusion	The quantity is a sound security metric		No

Figure 4.6: Harmonized TRA methodology soundness evaluation

Soundness Decision

The completed soundness evaluation for Harmonized TRA Methodology is shown in Figure 4.6. While the vulnerability rating obtained via the Harmonized TRA Methodology satisfies the sufficiency, progression, and reproducibility properties, it does not satisfy the definition property. This is due to the ambiguity in the definition of the probability of compromise and the severity of the outcome which can lead to misinterpretations when assigning values to determine the vulnerability rating. Thus, the vulnerability rating obtained from the Harmonized TRA Methodology is not sound.

4.2.4 Vulnerability Rating and Scoring System

Vulnerability Rating and Scoring System (VRSS) is a framework for qualitatively rating and quantitatively scoring vulnerabilities, aiming to combine the advantages of all kinds of vulnerability rating systems [LZ11].

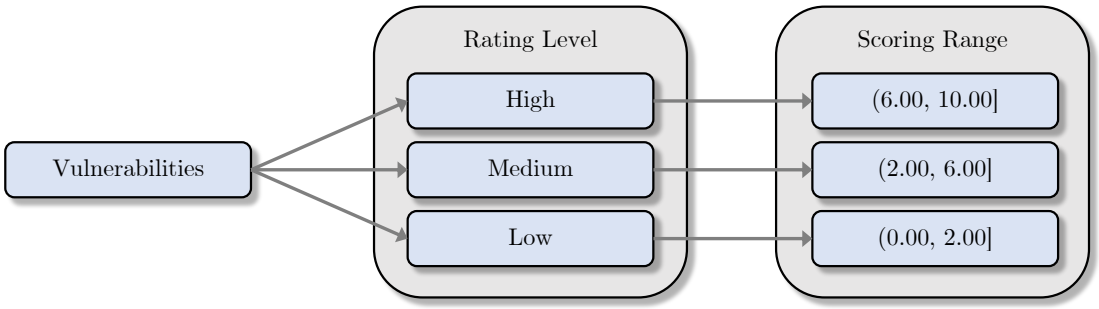


Figure 4.7: VRSS v1.0 [LZ11]

VRSS involves both a rating method and a scoring method as shown in Figure 4.7. The rating method qualitatively assesses a vulnerability in the context of confidentiality (C), integrity (I), and availability (A) properties. Each of these properties is assigned a rating of "None (N)," "Partial (P)," or "Complete (C)" yielding 27 possible combinations. Each combination is called an *impact vector* and is associated with a *qualitative rating level* and an *impact score* as shown in Table 4.4. For example, a vulnerability with a partial loss of confidentiality, no loss of integrity, and a complete loss of availability gives the impact vector [C:P/I:N/A:C] and is assigned a qualitative rating level of "Medium" and an impact score of 5.

Table 4.4: VRSS rating method impact scores [LZ11]

ID	Description	Possible Impact Metrics Cases	Qualitative Level	Impact Score
1	Each of confidentiality, integrity, and availability properties has a "complete" loss.	[C:C/I:C/A:C]	High	9
2	One of confidentiality, integrity, and availability properties has a "partial" loss. The other two have a "complete" loss.	[C:P/I:C/A:C], [C:C/I:P/A:C], [C:C/I:C/A:P]	High	8
3	One of confidentiality, integrity, and availability properties has a "none" loss. The other two have a "complete" loss.	[C:N/I:C/A:C], [C:C/I:N/A:C], [C:C/I:C/A:N]	High	7
4	One of confidentiality, integrity, and availability properties has a "complete" loss. The other two have a "partial" loss.	[C:C/I:P/A:P], [C:P/I:C/A:P], [C:P/I:P/A:C]	High	6
5	One of confidentiality, integrity, and availability properties has a "complete" loss. One of them has a "partial" loss, and one has a "none" loss.	[C:C/I:P/A:N], [C:C/I:N/A:P], [C:P/I:C/A:N], [C:P/I:N/A:C], [C:N/I:C/A:P], [C:N/I:P/A:C]	Medium	5
6	One of confidentiality, integrity, and availability properties has a "complete" loss. The other two have a "none" loss.	[C:C/I:N/A:N], [C:N/I:C/A:N], [C:N/I:N/A:C]	Medium	4
7	Each of confidentiality, integrity, and availability properties has a "partial" loss.	[C:P/I:P/A:P]	Medium	3
8	One of confidentiality, integrity, and availability properties has a "none" loss. The other two have a "partial" loss.	[C:P/I:P/A:N], [C:P/I:N/A:P], [C:N/I:N/A:P]	Medium	2
9	One of confidentiality, integrity, and availability properties has a "partial" loss. The other two have a "none" loss.	[C:P/I:N/A:N], [C:N/I:P/A:N], [C:N/I:N/A:P]	Low	2
10	Each of confidentiality, integrity, and availability properties has a "none" loss.	[C:N/I:N/A:N]	Low	1

The scoring method computes a *quantitative score* by summing the previously assigned impact score with an *exploitability score* for a vulnerability as in Equation 4.1.

$$QuantitativeScore = ImpactScore + ExploitabilityScore \tag{4.1}$$

$$ExploitabilityScore = 2 \times AV \times AC \times Au \tag{4.2}$$

Table 4.5: VRSS scoring method exploitability scores [LZ11]

Exploitability Metric	Metric Value	Quantitative Score
Access vector (AV)	Local (L)	0.395
	Adjacent Network (A)	0.646
	Network (N)	1.0
Access complexity (AC)	High (H)	0.35
	Medium (M)	0.61
	Low (L)	0.71
Authentication (Au)	None (N)	0.704
	Single (S)	0.56
	Multiple (M)	0.45

The exploitability score is computed as in Equation 4.2 by considering the Access Vector (AV), Access Complexity (AC), and Authentication (Au) properties of a vulnerability and assigning a quantitative score according to Table 4.5.

Evaluation

We evaluate *VRSS Framework v1.0* [LZ11]. In this case, the *quantity* is the quantitative score for a vulnerability as computed in Equation 4.1.

Definition Property: Computing VRSS scores is rather easy to do and the resulting quantitative score yields a severity level for a given vulnerability. However, the lack of a concrete definition of scope for differentiating between "Complete" loss and "Partial" loss for the three security properties leaves room for subjectivity. A similar issue exists for assigned values of "High," "Medium," and "Low" for access complexity. VRSS hypothesizes that statistically, the number of vulnerabilities with "Medium" severity is the largest and the numbers of vulnerabilities with "High" or "Low" severity are much smaller thereby forming a normal distribution. The pre-determined

quantitative score for the three exploitability metrics AV, AC, Au shown in Table 4.5 are not justified and appear to be biased towards achieving the hypothesized normal distribution. Thus, the definition property is not satisfied.

Sufficiency Property: As the quantitative score for a vulnerability increases, the associated severity of the vulnerability also consistently increases. Therefore, it is easy to see that the presence of vulnerabilities with higher quantitative scores indicates a lower system security level. Furthermore, the impact and exploitability metrics used to compute the quantitative score appear to contain all necessary aspects required to measure the severity of a known vulnerability. Thus, the sufficiency property is satisfied.

Progression Property: By examining Tables 4.4 and 4.5 and Equations 4.1 and 4.2, it is easy to see that implementing adequate mitigations to reduce the losses and/or exploitability of a given vulnerability will reduce the quantitative score. Consequently, the quantitative score can progress to an acceptable level. Thus, the progression property is satisfied.

Reproducibility Property: As shown in Table 4.4, VRSS generates an impact vector based on the values used to assign an impact score. We can also envision a similar input vector for the exploitability score by considering the values in Table 4.5. If these input vectors remain constant for a given vulnerability, the quantitative VRSS score can easily be reproduced and verified by third-party verifiers. However, similar to CVSS, if third-party verifiers must independently assign values for the impact and

exploitability, it is possible that they will obtain different quantities, but it is unlikely that they will reach different conclusions. Thus, we consider the reproducibility property to be satisfied.

Soundness Decision

The completed soundness evaluation for VRSS is shown in Figure 4.8. The quantitative score obtained via VRSS satisfies the sufficiency, progression, and reproducibility properties; however, it does not satisfy the definition property. This is due to the ambiguity in the definition of the scope of the impact and exploitability metric values and the apparent bias in the pre-defined impact and exploitability scores which tend to favour a normal distribution of vulnerability severity. Thus, quantitative VRSS scores are not sound.

4.2.5 Attack Surface Measurement Framework

A system's attack surface is described as the sum of attack vectors where an unauthorized user attempts to manipulate data inputs or extract data from the system. The intuition behind measuring a system's attack surface is based on the idea that the more extensive and exposed the system's attack surface, the more opportunity for a malicious adversary to conduct an attack [MW10].

Attack surface metrics account for a number of different dimensions of vulnerabilities including targets (t), enablers (e), channels (ch), protocols (p), and access rights (ar) of the system as shown in Equation 4.3 [HPW05].

MDSSM Criteria		Criteria Description	Evaluation (Yes/No)
1. Definition	1.1	Is the quantity objective?	No
	1.2	Is the quantity unbiased?	No
	1.3	Is the quantity meaningful?	Yes
	1.4	Can the quantity be obtained without undue hardship or costs?	Yes
2. Sufficiency	2.1	Does the quantity have a direct impact on the security level?	Yes
	2.2	Does an increase in the quantity result in a consistently increase (or decrease) in the system security level?	Yes
	2.3	Are all aspects needed to effectively measure the system incorporated in the definition of the quantity?	Yes
3. Progression	3.1	Does the quantity progress to an acceptable or maximal security level provided adequate mitigations are implemented?	Yes
4. Reproducibility	4.1	Can third-party verifiers obtain the same quantity and/or reach the same conclusion using the same procedure?	Yes
Conclusion	The quantity is a sound security metric		No

Figure 4.8: VRSS soundness evaluation

$$AttackSurface = f(t, e, ch, p, ar) \tag{4.3}$$

Targets refer to the assets and/or processes that an adversary intends to compromise. Enablers refer to assets and/or processes that an adversary can use to attack and/or access its targets. Enablers do not include the targets themselves. Channels refer to any means of communication used to interact with the system such as message passing or shared memory. Protocols refer to the specific communication rules used by the system to exchange information between its assets and processes. Finally, access rights refer to the various rules implemented to control system interactions and user access to the system, its processes, and its data. In Equation 4.3, the function f is typically

defined in terms of additional functions or weights for each considered dimension. In the literature, this function f is deliberately left for an analyst to define precisely based on the particular context of the system under consideration [MW10].

Evaluation

There are many formulations of attack surface metrics (e.g., [MW10, HPW05, YMR14]). Here, we evaluate the metric obtained by the general attack surface framework described in [HPW05]. In this case, the *quantity* is the attack surface metric given by Equation 4.3. We assume that a specific interpretation of the function f has been defined by a security analyst.

Definition Property: The inputs to the attack surface metric in Equation 4.3 are easy to obtain as they are objectively defined as the considered targets, enablers, channels, protocols, and access rights for a vulnerability in a given system. Assuming that the analyst has appropriately defined the function f, it is not expected that the attack surface metric will be biased to a certain outcome. The meaning of the attack surface metric is clear in that it indicates the system security level by capturing the extent to which a system is exposed via a set of known vulnerabilities. Thus, the definition property is satisfied.

Sufficiency Property: The attack surface metric in Equation 4.3 considers a number of important dimensions in measuring the severity of a vulnerability (e.g., targets, enablers, channels, etc.). Furthermore, the framework described in [HPW05] enables

the formulation to be expanded with additional dimensions as required for the specific system under consideration. Thus, the sufficiency property is satisfied.

Progression Property: It is clear that the implementation of mitigations to reduce the number of attack vectors in a system will result in a reduced attack surface metric. In this way, it is possible to progress towards an acceptable system security level. Thus, the progression property is satisfied.

Reproducibility Property: Based on our assumption that the function f is defined by a security analyst for a given system, because the function is deterministic and there is a clear definition of the inputs required for the function (by our assumption), the attack surface metric is easily reproducible and verifiable by third parties. Thus, the reproducibility property is satisfied.

Soundness Decision

The completed soundness evaluation for the attack surface metric is shown in Figure 4.9. The attack surface metric obtained by the general attack surface framework described in [HPW05] satisfies all of the properties of a sound security metric. Therefore, the attack surface metric is sound.

MDSSM Criteria	Criteria Description		Evaluation (Yes/No)
1. Definition	1.1	Is the quantity objective?	Yes
	1.2	Is the quantity unbiased?	Yes
	1.3	Is the quantity meaningful?	Yes
	1.4	Can the quantity be obtained without undue hardship or costs?	Yes
2. Sufficiency	2.1	Does the quantity have a direct impact on the security level?	Yes
	2.2	Does an increase in the quantity result in a consistently increase (or decrease) in the system security level?	Yes
	2.3	Are all aspects needed to effectively measure the system incorporated in the definition of the quantity?	Yes
3. Progression	3.1	Does the quantity progress to an acceptable or maximal security level provided adequate mitigations are implemented?	Yes
4. Reproducibility	4.1	Can third-party verifiers obtain the same quantity and/or reach the same conclusion using the same procedure?	Yes
Conclusion	The quantity is a sound security metric		Yes

Figure 4.9: Attack surface metric soundness evaluation

4.3 Summary of the Evaluation

A summary of our evaluation is shown in Table 4.6. In particular, four of the five vulnerability scoring frameworks that we evaluated do not yield sound security metrics.

Each metric satisfied the sufficiency and progression properties, ensuring that they sufficiently reflect the system security level, and through improvements to system security, can be reduced to an acceptable level. However, three of the five metrics did not satisfy the definition property due to subjectivity in the assignment of values used to compute the metric. Subjectivity is attributed to the ill-defined scope of the

Table 4.6: Soundness evaluation summary for security metrics obtained via vulnerability scoring frameworks

Vulnerability Scoring Framework	Metric	1. Definition				2. Sufficiency			3. Progression	4. Reproducibility	Sound Metric
		1.1	1.2	1.3	1.4	2.1	2.2	2.3	3.1	4.1	
CVSS	CVSS score	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes*	No
OCTAVE Allegro	relative risk score	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
CSE Harmonized TRA Methodology	vulnerability rating	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes*	No
VRSS	quantitative score	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes*	No
Attack Surface Measurement Framework	attack surface	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

* satisfied under the assumption that while quantities obtained by third-party verifiers may vary, they are unlikely to vary so much that the verifiers reach different conclusions

qualitative inputs in the associated scoring frameworks. This may result in (sometimes significant) variation in the overall severity rating of a vulnerability which can lead to poor decision-making when determining which vulnerabilities require mitigation in a system. It also leaves room for manipulating scores to obtain results that falsely justify decisions of inaction which can lead to the deployment of systems with potentially severe vulnerabilities.

The reproducibility property was tricky to evaluate. There is a correlation between subjectivity in the definition of the metric and the ability to reproduce the results. If there is a potential for misinterpretation of qualitative inputs, then there is a chance that independent evaluators will obtain different quantities, and have potentially different interpretations of the quantity, which can affect decision-making capabilities. This is analogous to trying to independently measure the height of a person in metres without a common understanding or interpretation of a metre. Without this common understanding, it is unlikely that all evaluators will obtain the same measurement. However, based on our assumption, and as noted in Table 4.6, despite the potential for these variations in the obtained quantity, for CVSS, the Harmonized TRA Methodology, and VRSS, it is unlikely that these variations will be so great that they will yield different quantities with different interpretations for the same evaluation. In

our analogy, this means that all evaluators may still agree that the measured person is tall, for example. Only in the case of the attack surface metric, where the framework precisely defines the interpretations for the inputs used to compute the quantity, are we able to guarantee reproducibility and verifiability by third parties.

4.4 Recommendations

The results of our evaluation clearly show common issues with security metrics obtained via vulnerability scoring frameworks. In this section, we provide several recommendations that can help to improve and/or remedy the issues in the frameworks to guarantee the soundness of such metrics.

1. Precisely and unambiguously define the scope (boundaries) of all inputs for every aspect used to compute the security metric in the framework. Following this recommendation can help to resolve the subjectivity issues noted in frameworks such as CVSS, the Harmonized TRA, and VRSS.

2. Incorporate factual and objective knowledge (data) of the system in the aspects used to compute the security metric. A more objective, normalized, data-driven approach to compute security metrics can reduce subjectivity and provide a more systematic, reproducible, and consistent outcome that has the potential to improve decision-making capabilities.

3. Provide well-defined guidelines to ensure the framework is adapted as intended. OCTAVE Allegro and the Attack Surface Measurement Framework provide guidelines

to adapt the framework based on the requirements of the evaluator(s). However, OCTAVE Allegro does not standardize the aspects (i.e., impact areas) used to compute its metric, minimizing the likelihood of reproducibility. Explicit and well-defined guidelines that specify all aspects required to compute the metric must be provided. Carefully following these guidelines is an important part of ensuring adaptations of the framework preserve the soundness of the associated metrics. It is also important to note that security metrics that were found to be unsound can incorporate our recommendations (which highlight their weak points) to become a sound security metric.

4.5 Conclusions

In this chapter, we evaluated the soundness of five security metrics obtained via vulnerability scoring frameworks that have been widely used to support security decision-making processes or that have been proposed for such purposes. The evaluation was based on the definition, sufficiency, progression, and reproducibility properties derived from MDSSM [Yee19b]. Our results show that four of the five frameworks considered in our evaluation yield security metrics that are not sound. Relying on unsound metrics when making security-related decisions about software-dependent systems is problematic and raises questions as to whether decisions based on these metrics are justifiable and/or acceptable. These questions can subsequently impact security evaluation and assurance activities for such systems. To address this concern, we also provided several recommendations that can help to improve and/or remedy the issues

in the evaluated frameworks and that should be considered in the development of new frameworks to guarantee the soundness of such metrics.

Thanks to the soundness evaluation of widely-used security metrics presented in this chapter, Alice can be confident in using the attack surface metric to reflect the security of her system design. This chapter addresses the challenges associated with selecting a reliable and trustworthy security metric to evaluate a system at the design level as described in Section 1.4.

Since we found the attack surface metric to be a sound security metric, we integrate it alongside the data-driven metrics presented in Chapter 3 into our data-driven system security evaluation approach presented in the following chapter.

Chapter 5

Defining Structural Security Posture to Evaluate System Designs

In this chapter, we propose an approach to support system architects like Alice in evaluating their system's architectural security during the design phase. In the approach, we aim to incorporate metrics that collectively provide a better reflection of the system's security characteristics based on its structural view. We call this the structural security posture of a system. Section 5.1 presents the methodology behind structural security posture. Section 5.2 presents a tool developed to support the structural security posture analysis based on a structural model of a system. Section 5.3 illustrates the usefulness of the structural security posture analysis using the tool described in Section 5.2 in a design scenario involving the OSM system. Section 5.4

discusses insights from developing this methodology. Lastly, Section 5.5 presents our conclusions from this chapter.

5.1 Methodology

We describe a system's *security posture* as its security state at a specific point in time that reflects its ability to defend against knowable threats that affect it. While we cannot quantitatively represent a system's security posture as a single quantity, we can provide quantifiable insights on the system's security level based on the views of the system (e.g., structural, functional, behavioural). Each view of a system encompasses several metrics to reflect specific security properties of the system associated with that view. These metrics serve as indicators of a system's security posture for a given view.

In this work, we focus on evaluating only the structural view of the system. We call the evaluation of a system's security posture from the structural view its structural security posture. The *structural security posture* is a collection of system-level and element-level metrics affected by specific parameters that help us evaluate a system's security posture based on its structural view (system's level of preparedness to defend against knowable threats). Since structural security posture incorporates both system-level and element-level metrics, we can use correlations between these metrics to evaluate 'what-if' scenarios by analyzing the effect of different component configurations on the overall system's security and element-level security. It is important to note that structural security posture is not a metric, but a security evaluation approach to provide a broader characterization of a system's security level.

The structural view of a system illustrates the composition of elements that make up the system. It provides the terminologies associated with the system's elements and its interfaces. The structural view of the system can be modeled in a number of ways, one of which is using a UML class diagram. This view helps us to evaluate security properties related to the overall structure of the system such as the structural attack surface of a system [GG12].

We model a system's structural view as a graph $\mathcal{G} = (N, E)$ where N is a set of nodes and E is a set of edges. In our work, we consider directed edges in the graph to reflect the relationship of system elements and undirected edges in the graph to reflect the communication between them. For the scope of this work, we focus our efforts on the communication between system elements (undirected edges).

5.1.1 Attack Surface Metric

We use the attack surface which we described in Section 4.2.5 as a system-level metric to evaluate a system's structural security posture. As a reminder, a system's attack surface is described as the sum of attack vectors where an unauthorized user attempts to manipulate data inputs or extract data from the system. The intuition behind measuring a system's attack surface is based on the idea that the more extensive and exposed the system's attack surface, the more opportunity for a malicious adversary to conduct an attack [MW10]. We chose the attack surface metric as it was found to be a sound security metric in our evaluation presented in Chapter 4. Security metrics that were found to be unsound but incorporate our recommendations in Section 4.4 to be sound can also be used in our approach.

Manadhata and Wing [MW10] defined a quantitative attack surface metric in terms of a system's resources (methods, channels, data items) as attackers generally use such resources present in the system's environment to attack the system. Gennari and Garlan [GG12] adapted Manadhata and Wing's attack surface metric to measure the attack surface of software architectures. Since the architecture of a software system is associated with its structural view, we adapt Gennari and Garlan's attack surface metric in this work. The attack surface metric quantifies a system's interaction with its external environment using three types of system elements:

1. **Components (C)**: These are the entry and exit point system elements (nodes in the graph) that accept/process data originating from the system environment. Gennari and Garlan's work refers to components of the system as methods.

2. **Links (L)**: These refer to the communication links (edges in the graph) connecting a system to its environment. Gennari and Garlan's work refers to links of the system as channels.

3. **Datastore Items (D)**: These are the data types of items stored in datastores (also nodes in the graph) used by the system. Gennari and Garlan's work refers to datastore items as untrusted data items.

Given these system elements, the graph representation of a system's structure is captured by $\mathcal{G} = (C \cup D, L)$ where the set of components C and set of datastore items D are disjoint.

Each element is evaluated for its attractiveness to an attacker, i.e., the likelihood an attacker will use that component in an attack. This attractiveness of a component

to an attacker is quantified as a ratio of the potential damage the attacker can inflict on the system by exploiting that component divided by the amount of effort needed to access that component (determined by the component's access rights). This ratio is called the DER (Damage Effort Ratio) and it corresponds to the intuition that an attacker will select the component offering the highest damage potential with the least amount of relative effort.

In the following, we describe the DER for each of the three different element types described above.

DER for Components Damage potential for components that serve as entry/exit points is evaluated in terms of their privilege since an attacker is likely to acquire the privileges assigned to that component post-compromise.

$$\mathrm{DER}_c = \frac{privilege\ level}{access\ rights\ required} \tag{5.1}$$

DER for Links Damage potential for links is evaluated based on the restrictions placed on the data that a link can transmit given its protocol. Protocols with reduced restrictions on the types of data that a link can transmit increase the advantage for an attacker as there are more ways to exploit the link. For example, the link can be used to transmit unauthorized executable code to a database as done in SQL injection attacks.

$$\mathrm{DER}_l = \frac{protocol\ type}{access\ rights\ required} \tag{5.2}$$

DER for Datastore Items Damage potential for items in datastores is evaluated based on the type of data they store. The more permissive the data type of the data they store, the fewer the restrictions on the types of data items the data store can contain. Thus, it is more advantageous to an attacker if unauthorized data items are stored within datastores.

$$\mathrm{DER}_d = \frac{data\ type}{access\ rights\ required} \tag{5.3}$$

It is important to note that a numerical, ordered scale of values is attributed to the parameters of DER. For example, the root privilege of a component may be the highest level of privilege and therefore be assigned a higher numeric value than the guest privilege for that component.

The system's attack surface is presented as a triple that corresponds to the total contributions across the three element types calculated by aggregating the DER for each element type as shown in Equation 5.4.

$$\left\langle \sum_{c \in C} \mathrm{DER}_c(c), \sum_{l \in L} \mathrm{DER}_l(l), \sum_{d \in D} \mathrm{DER}_d(d) \right\rangle \tag{5.4}$$

5.1.2 Eigenvector Centrality Metric

We use centrality metrics as the element-level metric we use in the evaluation of a system's structural security posture. In graph theory, *centrality* is associated with how important a node in a graph is by ranking it based on how 'central' it is. The more central the nodes, the more connected they are to other nodes (high node degree) and

the more important they are in the system. Formally, we can define centrality as a function $c : N \rightarrow \mathbb{R}$ that induces a total order on the set of graph nodes N and where c refers to any type of node centrality such as degree centrality or eigenvector centrality. A node $n_i \in N$ is considered to be at least as central as node $n_j \in N$ if $c(n_i) \geq c(n_j)$ for some function c corresponding to a specific type of centrality metric. Centrality metrics differ based on how they compute the importance of nodes. Some of these metrics can be applied on both directed and undirected graphs while others apply to one or the other. For the scope of this work, we focus our efforts on the communication between components (undirected edges) and therefore select a centrality metric that can be applied in an undirected graph.

The simplest way to compute centrality for a graph is to evaluate the degree (number of edges) for each node in the graph. This metric is called degree centrality. The higher the *degree centrality* of a node, the more important it is. However, in this work, we express the central behaviour of system elements using the *eigenvector centrality* metric, which builds on the notion of degree centrality [New18]. Eigenvector centrality measures a node's centrality by evaluating the centrality of its neighbours rather than just the number of edges incident with the node [Bon87]. Eigenvector centrality assigns a relative rank or score to reflect the importance of a node in the graph based on the importance of its neighbours. A node with a high eigenvector centrality rank is one that is connected to many other well-connected nodes in the graph. For the structural security posture evaluation, computing a system component's eigenvector centrality helps us to identify important (attractive) components that the attacker may target due to its proximity to other important system components.

The eigenvector centrality for a node n_i is the i-th element of the eigenvector x in the following equation:

$$Ax = \lambda x \tag{5.5}$$

where A is the adjacency matrix representing the graph with eigenvalue λ. According to the Perron-Frobenius theorem, a unique solution for x exists if λ is the largest eigenvalue in A [New18].

5.1.3 Data-driven Threat Metrics

Since a system's security posture is defined within the scope of the knowable threats of a system, we incorporate the data-driven metrics derived from our analysis of threats and requirements within the components of the system presented in Chapter 3. These metrics leverage multiple external data sources such as the National Vulnerability Database (NVD) [Nat20], MITRE ATT&CK [SAM+20], CCCS Alerts and Advisories [Can] to ascertain threats that are likely to affect a system component based on its security requirements and design specifications. While [SJY21] describes a number of metrics that can be derived from external data sources, we specifically incorporate the following metrics:

1. **Unmitigated threats** Identified threats for which there are currently no security requirements.

2. **Mitigated threats**: Identified threats addressed by the provided security requirements.

3. **Extraneous requirements**: Provided security requirements for which no relevant threats were found.

5.2 Polaris: A Security Posture Analysis Tool

To support system architects in automating the evaluation of a system's structural security posture presented in Section 5.1, we developed an open-source tool called Polaris[2].

The primary requirements for Polaris were as follows:

1. Be widely accessible for users.
2. Intuitive to design and analyze a system's structural security posture.
3. Present analysis information in a manner that is visual and interpretable.
4. Extensible for users by enabling them to export model and analysis results for analysis by other tools.

Polaris offers an interactive, cross-platform web-based service to allow anyone with a web browser to design and analyze the structural security posture of their system. Polaris simplifies the structural security posture analysis into three steps: Design, Analyze, and Summarize.

[2]Polaris is the brightest star in the Ursa Minor constellation. It is commonly referred to as North Star as it has been historically used by navigators to orient towards Earth's north pole.

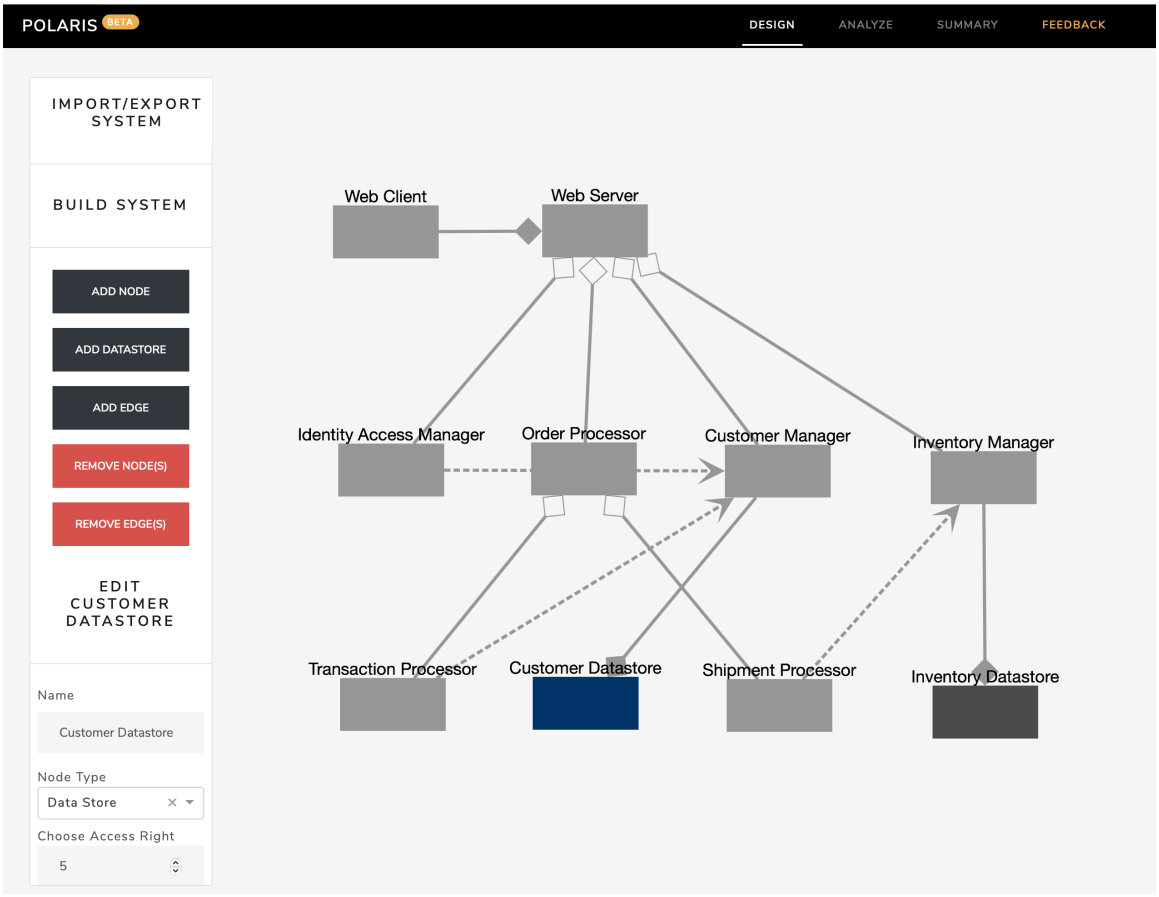


Figure 5.1: A screenshot of the Design tab in Polaris

The Design tab enables users to design their system models and annotate those models with information such as security requirements design decisions. It provides a Graphical User Interface (GUI) to edit and visualize the system model as shown in Figure 5.1. System models can also be imported and exported from this tab fulfilling user extensibility requirements.

The Analyze tab performs the structural security posture analysis on the system designed or imported into Polaris in the Design tab. It provides a GUI to edit and visualize the system model as shown in Figure 5.2. The Analyze tab is composed

of three main sections. The first section shows the system-level and element-level metrics for the system. A tabbed interface helps architects view relevant information in a concise form. It also helps ensure we do not overwhelm the architect with numbers from all our metrics at once and would go against our requirement of ensuring the results are easy to understand. The system model provided in the Design tab of Polaris is analyzed each time the tab is opened. As part of this analysis, the attack surface and eigenvector centrality metrics described in Section 5.1 are automatically computed based on the parameters chosen during the system design. Polaris also incorporates the data-driven threat metrics described in Section 5.1.3 to leverage external data sources to perform threat analysis. To obtain these metrics, we use Merak (see Section 3.4). The security requirements and design decisions for each element (where available) are forwarded to Merak for this analysis. The results from that analysis are shown directly in the Analyze tab in Polaris, minimizing the need for architects to jump between tools during analysis. The middle section provides an interactive view of the system model provided. Selecting an element like a component, link, or datastore updates the element-level metrics accordingly. The last section encapsulates the various parameters for the various elements of the system. This section also enables the architect to conduct 'what-if' analyses dynamically by manipulating system element values (such as security requirements, design decisions, access rights, etc.) set during the design of the system or by changing the structure of the system itself to identify its effect on the system's structural security posture. Only the relevant parameters are loaded based on the element selected in the middle section. Manipulating the parameters here does not overwrite the values present in the design but is purely to support dynamic structural security posture evaluation. If

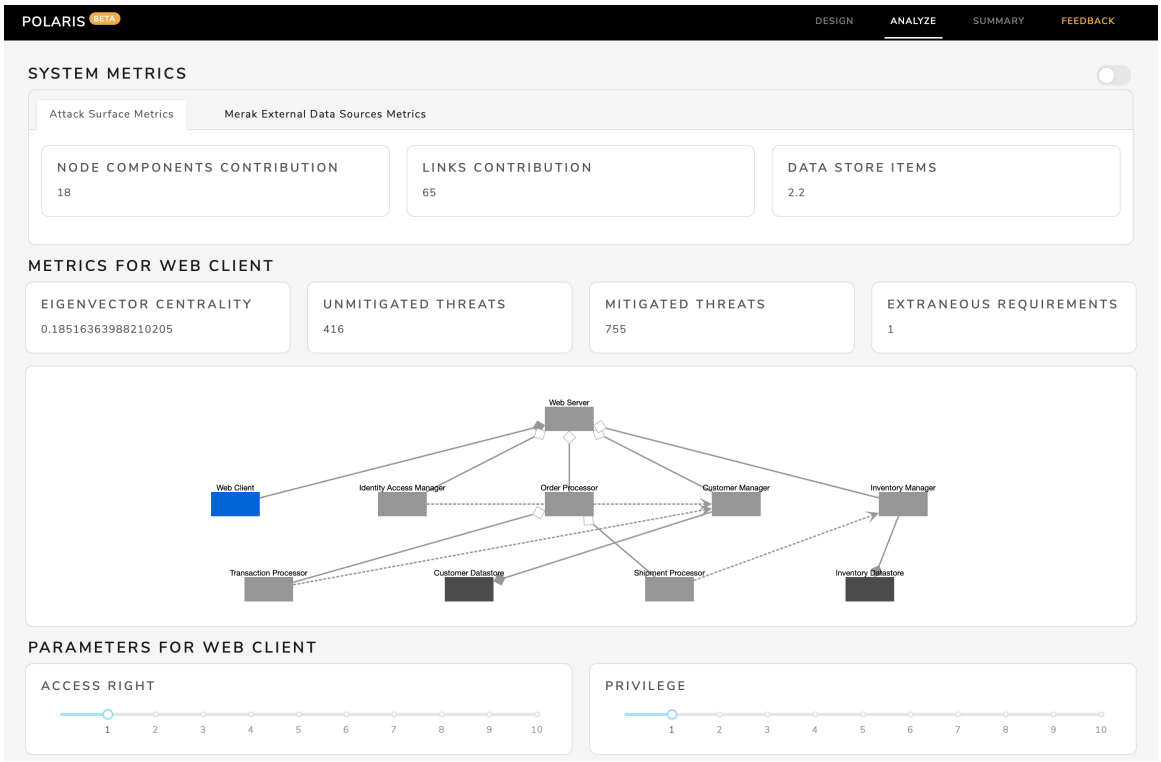


Figure 5.2: A screenshot of the Analyze tab in Polaris

the architect is satisfied with new values for one or more elements after conducting a 'what-if' analysis, they can set these values in the system model through the Design tab.

The Summarize tab, shown in Figure 5.3, summarizes the results of the system model in a way that can be exported in PDF form for reference or comparison with system variants. It also downloads a copy of the system model under analysis to ensure that the results can be replicated.

Polaris is *stateless*, which means it does not maintain design or analysis data from evaluations on the server-side. Instead, Polaris uses browser-based storage to ensure the system data and results remain local. This stateless attribute allows Polaris to

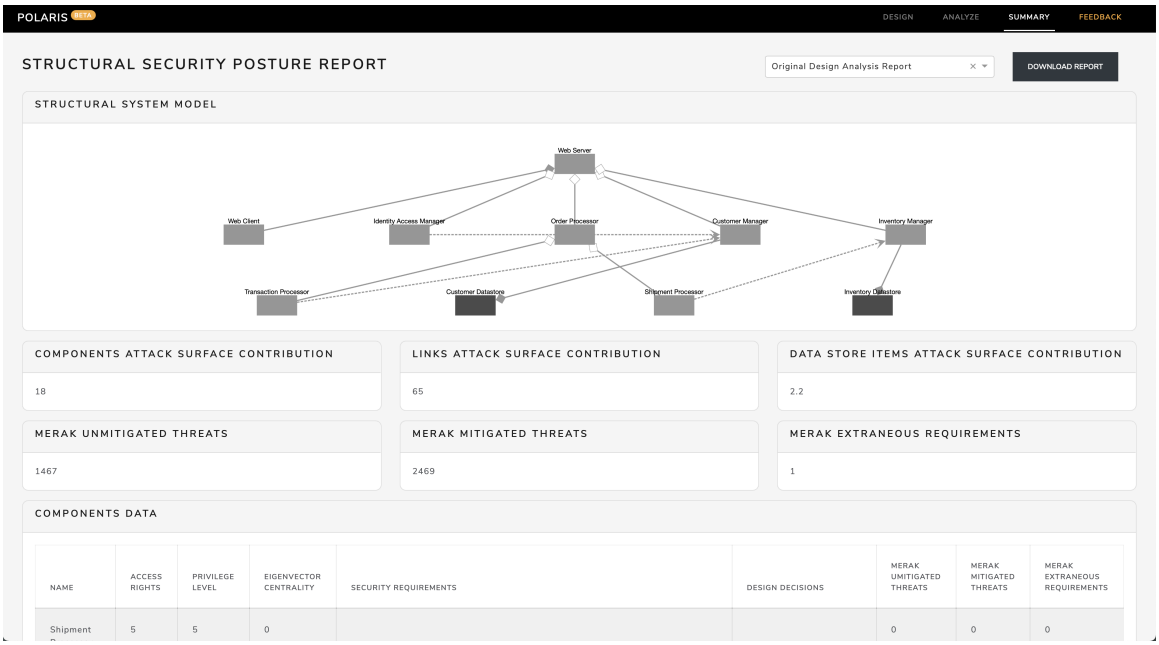


Figure 5.3: A screenshot of the Summarize tab in Polaris

be scalable and robust. Polaris is scalable as user sessions are not mapped 1-1 with its server instances. This means that users who are not actively using functions of the application (such as building the graph, running analysis) are not consuming an instance of Polaris on the server (viewing your model, reviewing your results do not take up a server instance). Each user request is allocated to any of the available service instances. This means that we can serve more users with relatively few server instances of Polaris. Should one or more of Polaris's server instances fail, other available instances can take on their tasks ensuring Polaris is robust.

Polaris is available at `https://polaris.compass.carleton.ca/` for practitioners to analyze the structural security posture of their systems.

5.3 Improving the OSM System Design through Structural Security Posture Evaluation

As mentioned in Section 1.3, Alice's goal is to evaluate different design decisions to design an adequately secure system. Doing this during the OSM system's design phase means that Alice can iterate over the design to improve its security without incurring additional overhead or costs.

Alice adapts the parameters for the elements of her system to be the same 1-to-10 scale chosen by Gennari et al. in their attack surface example [GG12]. The interpretation of the chosen scales for each component type is shown in Table 5.1. Transmission Control Protocol (TCP) in Table 5.1 is a type of internet protocol used to transmit information over the internet. Remote procedure call (RPC) is when a computer program executes an operation beyond its own address space (on a remote computer). For the purpose of this example, detailed knowledge of these protocols is not required. The larger the value chosen, the greater the advantage the attacker has to exploit a component. For example, a component with guest privilege will have significantly reduced privileges and therefore be assigned the lowest privilege value (1). Similarly, a component with root privilege will have the least restrictive privilege and therefore be assigned the highest privilege value (10).

Alice has an initial sketch of the OSM and uses the Design tab of Polaris to model her design in preparation for her experiment. In her initial system design, the Web Client is able to directly communicate to the Customer Datastore through the Web Server. While this is fairly straightforward to design and implement, it may

Table 5.1: Attack surface parameters for the OSM system

Components	**Privilege**	**Value**	**Access Rights**	**Value**
	root	10	root	10
	authenticated	5	authenticated	5
	guest	1	unauthenticated	1
Links	**Protocol**	**Value**	**Access Rights**	**Value**
	direct entry	10	local	10
	TCP	5	remote	1
	RPC	1		
Datastore Items	**Data Type**	**Value**	**Access Rights**	**Value**
	SQL	10	root	10
	Inventory Data	1	authenticated	5
			anonymous	1

have drastic implications on the security of the system. One example of potential security consequence is the ability to perform SQL injection attacks if there is no input validation present. To model such scenarios, Alice chooses suitable values from Table 5.1 as parameters for the elements of her system. Table 5.2 details the values chosen for the components of the initial design of her system. The values assigned for other elements of the system are shown in Appendix A.

Table 5.2: Parameters for the OSM Components in the initial design

Component	Access Rights	Privilege
Shipment Processor	5	5
Transaction Processor	5	5
Customer Manager	5	10
Web Server	5	5
Inventory Manager	1	10
Identity Access Manager	5	5
Order Processor	5	5
Web Client	1	1

She assigns SQL (10) as the data type for Customer Datastore and authenticated user as its access rights. Alice chooses SQL as it is requirement for her operational context. SQL supports a diverse range of information such as customer names, addresses, phone numbers, and credit cards. Unfortunately, this feature of SQL to support broad types of data also lends itself to hold unauthorized data in the Customer Database. A popular attack that takes advantage of this is called a SQL Injection attack where the attacker provides a SQL statement in place of regular text input (such as for name, email) [HVO+06]. This attack works when the system fails to validate the input and passes the input SQL statement directly to the datastore which executes the unauthorized SQL statement. Alice set the datastore's access rights to denote an authenticated user to allow such users to access their information from the Customer Datastore.

Alice wants to evaluate the structural security posture for the OSM system to determine if and how she can improve her system design. She can use Polaris (see Section 5.2) to perform this experiment. Polaris allows Alice to perform 'what-if' analyses thanks to its dynamic computation of system-level and component metrics based on the chosen parameters.

5.3.1 Evaluation of the Initial OSM System Design

Alice initializes her OSM system design in Polaris based on her initial sketch shown in Figure 1.2. Her evaluation of the initial design enables her to get a baseline for the structural security posture of the system. Since Alice has the initial security requirements and design decisions for the Web Client, Web Server, and Customer

Datastore as shown in Table 5.3, she annotates this information in Polaris. These requirements and design decisions can help Alice leverage external data sources in her analysis based on the approach presented in Chapter 3.

Following the analysis of her initial design, Alice gets the results shown in Figure 5.2:

In her initial analysis, contributions from the components of the system towards the attack surface was 18, contributions of links towards the attack surface was 65, and contributions of datastore items was 2.2. The Web Server gets the highest centrality ranking of 0.55 followed by the Customer Manager at 0.44, Web Client at 0.19, and Customer Datastore at 0.15.

The element-level metrics (i.e., eigenvector centrality) of the structural security posture evaluation can help Alice to identify potential weak points in the design that could have negative security implications. Recall from Section 5.1 that a system element's eigenvector centrality ranks its importance in the system relative to the importance of its neighbouring elements. She notices that out of all the elements likely to be involved in a SQL injection attack, the Web Server is likely to be an important target followed by the Customer Manager based on their centrality rankings. This implies that, should an attacker manage to compromise the Web Server or the Customer Manager (or worse, both!), they can access and communicate with other important system elements in addition to the Customer Datastore. Since the Web Server and Customer Manager have the potential to enable such attacks on the system, Alice extends her focus toward protecting these elements. Alice correlates the analysis results with the privilege level she assigned for the Web Server in the initial design. Initially, the privilege level of the Web Server was

assigned as an authenticated user (5) which means that an attacker can have the privilege level of an authenticated user if they compromise the Web Server. This would give an attacker access to functions associated with that privilege level and the ability to maliciously affect other components that also require the privilege level of an authenticated user (5).

These results also show that there is a significant contribution from the links in the design towards the overall system's attack surface. The contributions from components are also something Alice wants to pay attention to when revising her design to ensure she does not increase it significantly. The positive news here is that the contributions from datastore items towards the overall attack surface in Alice's initial design is significantly low (2.2). This means she can shift her focus towards components such as the Customer Manager and Inventory Manager which interact with their respective datastores to improve the system's overall security.

The data-driven threat metrics from Merak point Alice to potential attacks that could affect the Web Client, Web Server, and Customer Datastore (elements she annotated with requirements and design decisions). As she reviews the unmitigated threats for the Web Client and Customer Datastore, she notices a common type of attack across various threats; an SQL injection attack, which confirms her initial intuition. We present a snippet of the threat analysis results from Merak's analysis in Figure 5.4 on the requirements for the Customer Datastore detailed in Table 5.3. In the threats listed, we can see that CVE-2021-2385 discusses a vulnerability that allows a highly privileged attacker with network access to compromise a MySQL Server such as the one Alice intended for her Customer Datastore

Table 5.3: Merak threat analysis parameters and results

Component	Security Requirements	Design Decisions	Unmitigated Threats	Mitigated Threats	Extraneous Requirements
Customer Datastore	Ensure access control policies are implemented allowing the datastore administrator to implement access policies for various datastore users and data contained within the datastore.	The database server will be MySQL and it will run on a Linux server. The database server will use SQL to query the database.	1030	1195	0
Web Client	Ensure all customers are verified based on their credentials using login functionality.	Web Client will interact with a SQL database to populate the inventory page.	564	755	0
Web Server	Ensure all internal and external connections for the server go through an appropriate and adequate form of authentication. All protected pages must enforce the requirement for authentication and authorization.	Web server uses NGINX proxy and Linux.	131	519	0

and Inventory Datastore. Additionally, we can see references to SQL Injection vulnerabilities affecting an e-commerce website in CVE-2021-25205 similar to what Alice intends to design and develop. Having this type of information during the design phase helps Alice to better understand the threat landscape of her yet-to-be-implemented system which helps her focus her attention on specific areas of the system such as elements that deal with SQL data or handle input data in general. It also helps her think about suitable mitigations that can be instituted in the design prior to the system's implementation.

With her baseline now set, Alice comes up with a few alternative transformations for her design that she suspects can mitigate a SQL injection attack:

1. Modify the access rights, privileges, and protocols of the system elements involved in conducting such attacks.

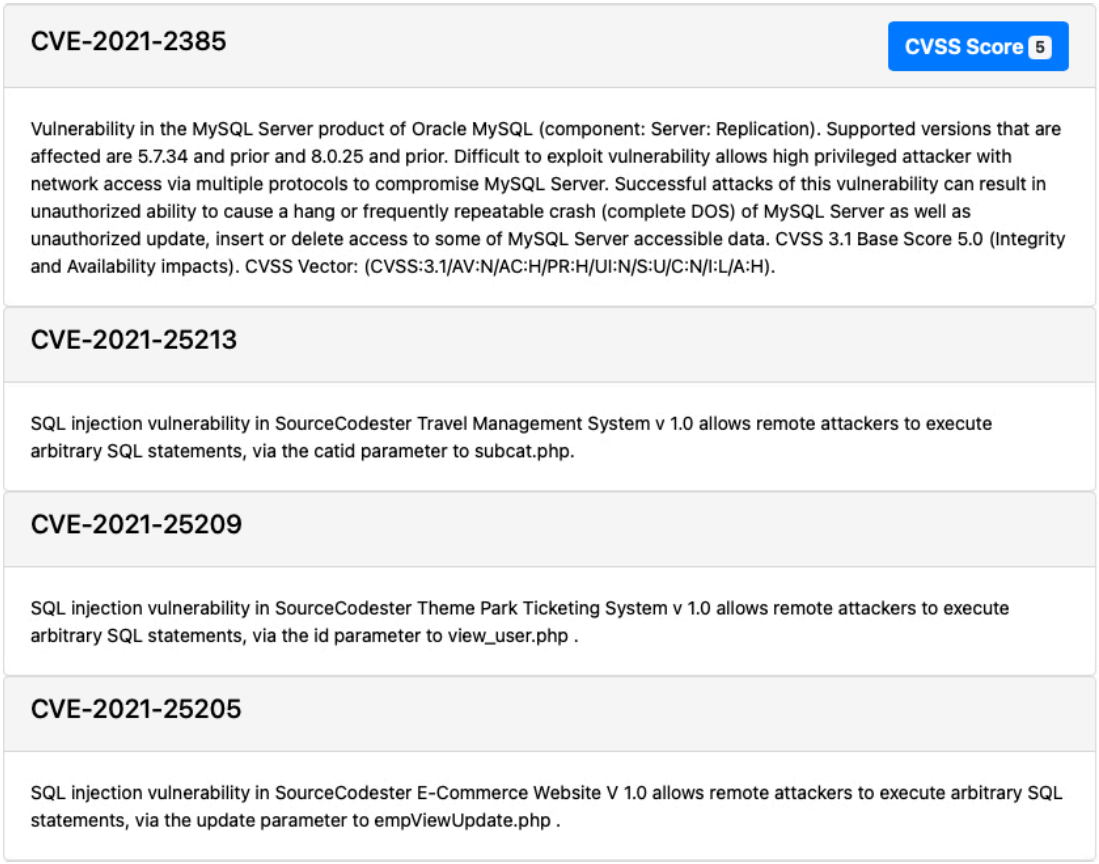


Figure 5.4: An excerpt of the analysis results for the OSM Customer Datastore in Merak

2. Create a Data Sanitizer component to sanitize the input from the Web Client before it reaches the Customer Datastore.

3. Apply a combination of the above transformations.

She suspects that the ideal solution may be a combination of the above strategies to mitigate SQL injection attacks (and similar attacks that exploit inputs). Since it is hard for Alice to concretely determine to what extent each mitigation strategy needs to be employed, she decides to use the 'what-if' analysis feature of Polaris to

experiment with the possible security outcomes of each alternative. This type of trial and error analysis can help her discern what values to assign for the access rights, privileges, and protocols of the system elements (i.e., components, links, and datastore items) involved in such attacks. She can also evaluate how different values impact the overall system's structural security posture. She also wants to evaluate if changing these values or the structure of the design has any unintentional impact on the security of other elements of the system.

In what follows, she applies each of the above transformations and evaluates the change in the system's structural security posture.

5.3.2 Transformation #1: Experimenting with Parameters of the OSM System Elements

Alice's objective with the 'what-if' analysis is to determine the parameter values for system elements that collectively minimize the overall system's attack surface. With Polaris, Alice is able to evaluate how different access rights, privileges, and protocols can impact the overall attack surface of the system and to what extent as shown in Figure 5.5. During Alice's experimentation with different parameters, she changed the privilege level associated with the Inventory Manager and Customer Manager from root (10) to an authenticated user (5) since root privileges were simply not required to perform Create, Read, Update, Delete (CRUD) operations with their respective databases and instead increased the system's attack surface.

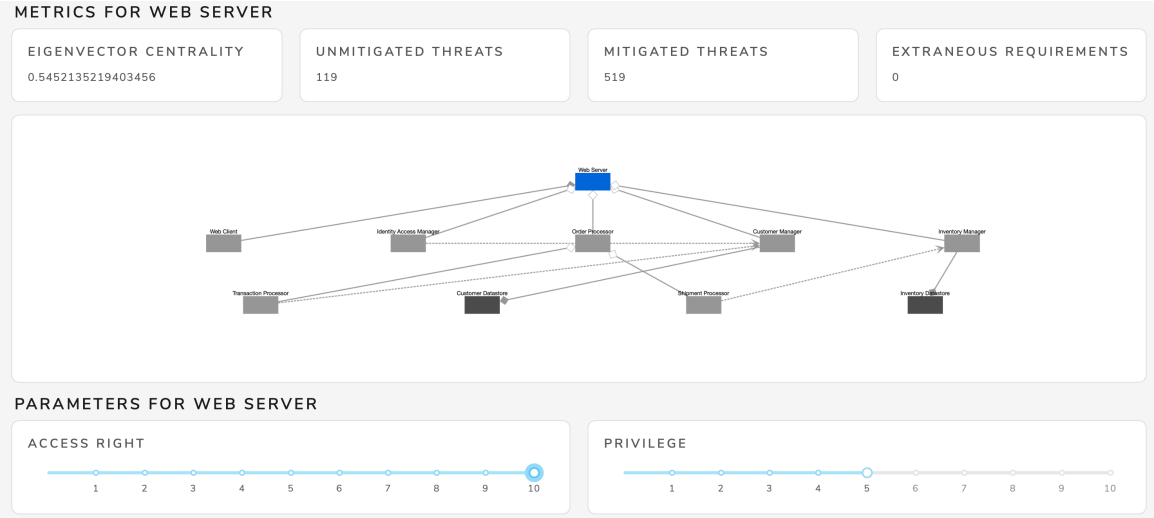


Figure 5.5: Performing 'what-if' analysis by experimenting with parameters of the OSM system elements in Polaris

During her analysis, Alice also noticed redundant links between components. One such set of links connects the Identity Access Manager and Transaction Processor to the Customer Manager. These links are considered redundant as the Identity Access Manager and Transaction Processor can communicate with the Customer Manager transitively through the Web Server. Similarly, Alice identifies a redundant link connecting Shipment Processor to Inventory Manager. Alice decides to prune such links in addition to applying her new set of parameters as part of this transformation. The new structure of the OSM system after making these changes is depicted in Figure 5.6.

The transformations already show an improvement in the system's attack surface. Contributions from the components of the system towards the attack surface reduced from 18 to 12 and contributions of links towards the attack surface reduced from 65 to 50. With regards to centrality, the Web Server still has the highest centrality

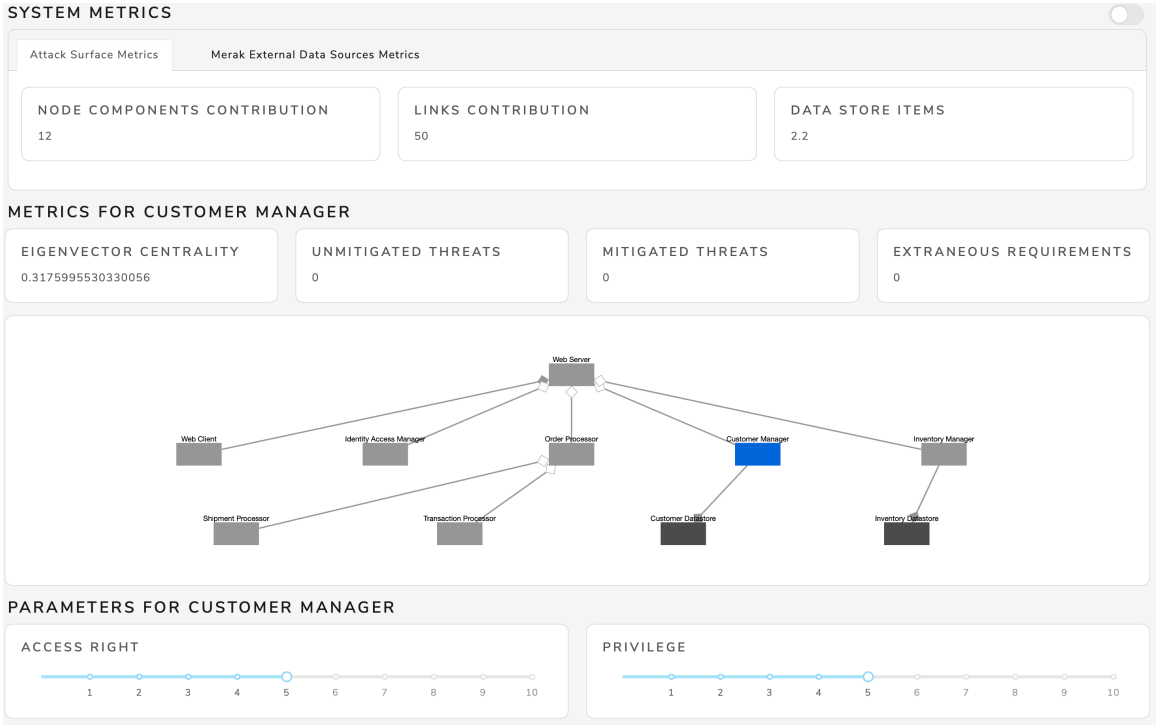


Figure 5.6: Structural security posture analysis results after applying Transformation #1 in Polaris

ranking of 0.64 followed by the Customer Manager at 0.32, Web Client at 0.26, and Customer Datastore at 0.13.

The importance of the Web Client, Web Server, and Customer Datastore increased as the importance of their neighbouring elements increased due to the removal of redundant links which reduces the paths to certain elements and increases the importance of elements that form that path. The Customer Manager was the exception but only by a margin of 0.01 which is not significant, relative to the centrality values of the other critical elements.

5.3.3 Transformation #2: Sanitizing Data Inputs

To help mitigate input-based attacks such as SQL Injection attacks, Alice decides to introduce two new components responsible for data sanitization, one between the Web Server and other components of the system which she names Gatekeeper and one between the Web Client and the Web Server which she names Data Sanitizer. Despite the different names, both of these components act as gatekeepers to protect sensitive elements of the system by validating any user input. The new structure of the OSM system after making these changes is shown in Figure 5.7.

Alice decides to incorporate the Data Sanitizer component locally with the Web Client to ensure that inputs are sanitized prior to their transmission over the network to the Gatekeeper (where they are sanitized again). Alice reflects this in her model by changing the access rights for the link connecting the Web Client and Data Sanitizer to local (10). She also sets the link connecting Data Sanitizer with Web Client to have the access rights of local (10) and protocol level of (10) as the communication between them will be through direct entry. The link connecting Gatekeeper to Data Sanitizer however communicates with access rights of remote (1) and a protocol level of TCP (5) which is more restricted than direct entry transmission.

It is important to note that Alice applies these transformations on her original system model. Alice's results from applying these transformations are shown in Figure 5.7.

Alice notices that the attack surface increased from the baseline value of 18 to 20 for the components and from 65 to 66 for the links. The Customer Manager

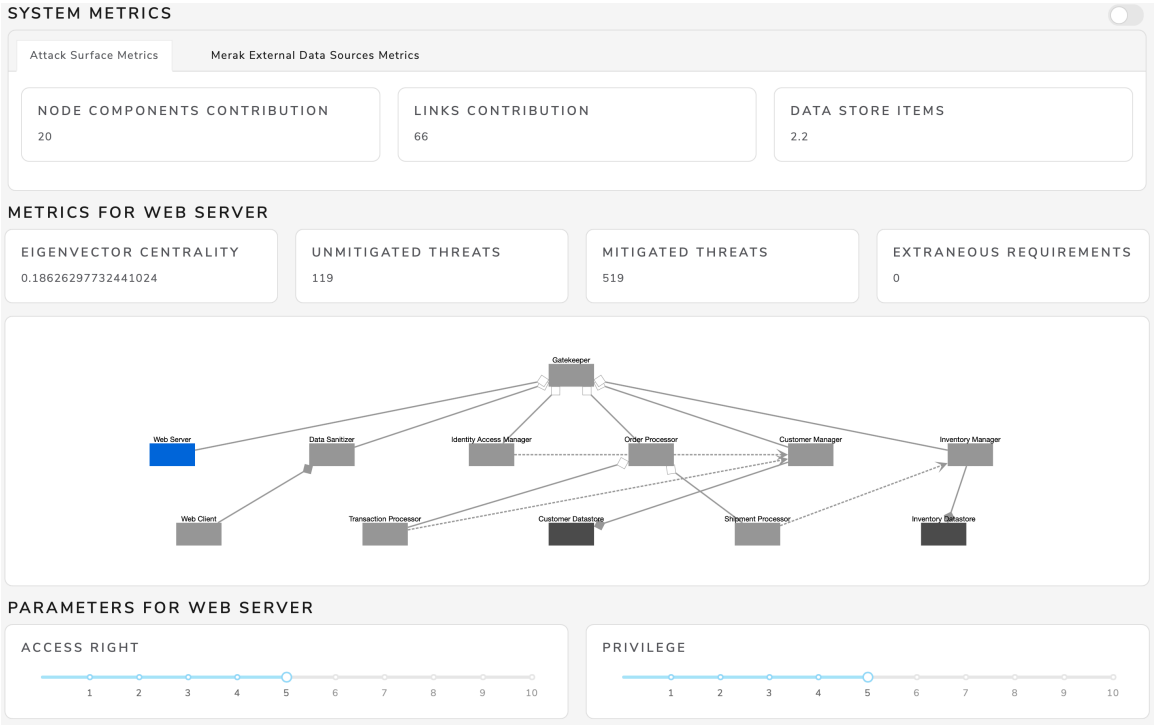


Figure 5.7: Structural security posture analysis results after applying Transformation #2 in Polaris

now has the highest centrality ranking at 0.42 followed by the Web Server at 0.19, Customer Datastore at 0.14, and Web Client at 0.07.

Alice notices how the Web Client and Web Server saw a significant reduction in their importance following the introduction of the Data Sanitizer and Gatekeeper components respectively. A notable observation here was how the Customer Manager was affected by the introduction of the Gatekeeper only by a small margin. However, Alice did expect this as the Customer Manager has more links than the Web Server and the Web Client so the introduction of a Gatekeeper on one of those links, while helpful, does not make a significant change to the structural security of the system.

5.3.4 Transformation #3: Combining Transformations #1 and #2

To summarize the effects of Alice's transformation so far, the first transformation on the initial design which pruned redundant links and reduced unnecessary access rights and privileges showed a significant reduction in the overall attack surface but it did result in an increase in the importance for a number of critical elements. The second transformation on the initial design which introduced two Data Sanitizer components showed a significant reduction in the importance of critical elements that were impacted directly by the Data Sanitizer but the overall attack surface increased.

In this transformation, Alice proceeds to evaluate the combination of both these transformations applied to the initial design at once. Her intuition is that this should result in a balanced overall reduction in terms of the system's attack surface and the importance of critical elements. Alice's results from applying these transformations are shown in Figure 5.8.

This transformation significantly reduced the attack surface of the system for both components and links. Contributions to the attack surface from components and links were reduced from 18 to 14 and 65 to 51, respectively. The Customer Manager still has the highest centrality ranking relative to the other elements at 0.29, followed by the Web Server at 0.25, and both the Customer Datastore and Web Client ranking at 0.11. However, Alice noticed a reduction in the importance of all of the elements (Web Client, Web Server, Customer Manager, Customer

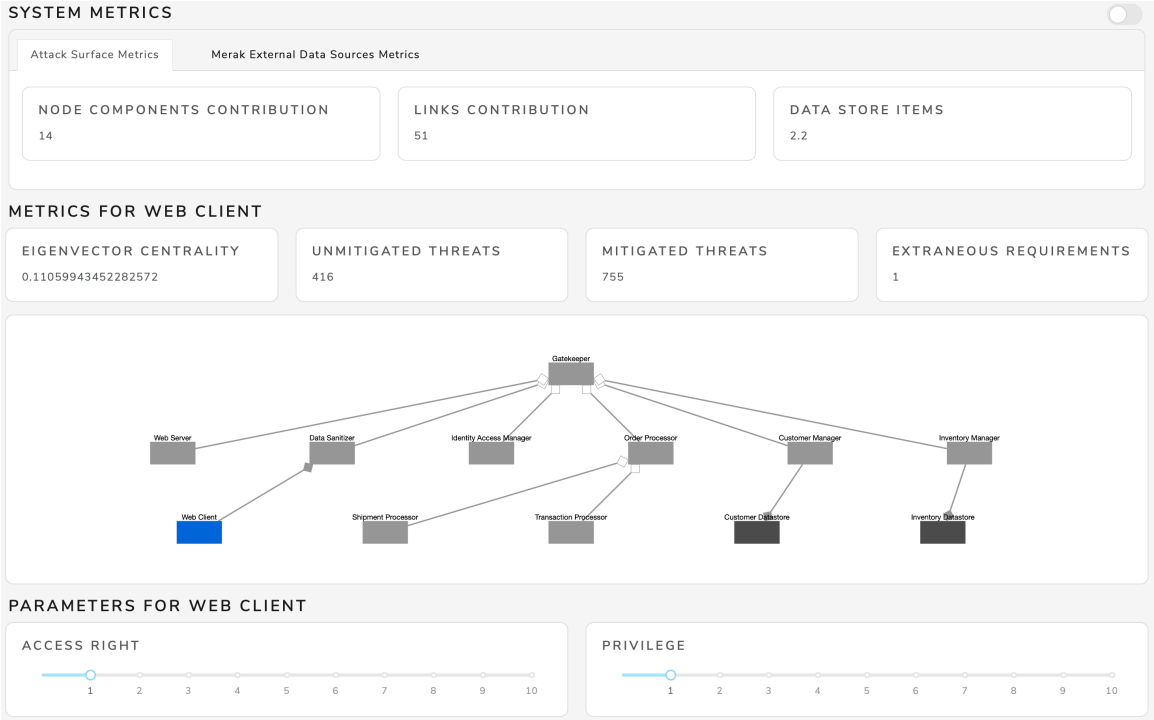


Figure 5.8: Structural security posture analysis results after applying Transformation #3 in Polaris

Datastore) that she sought to protect compared to her initial design. This means that the likelihood for an attacker to use the Customer Manager or other critical elements to affect other elements of the system is reduced.

5.3.5 Experiment Conclusions

A summary of Alice's structural security posture evaluation is shown in Table 5.4. Based on these outcomes, Alice believes transformation #3 which is the combination of modifying element access rights, privileges, and protocols, pruning redundant links, and introducing two data sanitization components in her initial system design provides

Table 5.4: Summary of structural security posture analysis for the OSM system design alternatives

Design	Attack Surface	Eigenvector Centrality			
		Web Client	Web Server	Customer Manager	Customer Datastore
Initial Design	(18, 65, 2.2)	0.19	0.55	0.44	0.15
Transformation #1	(12, 50, 2.2)	0.26	0.64	0.32	0.13
Transformation #2	(20, 66, 2.2)	0.07	0.19	0.42	0.14
Transformation #3	(14, 51, 2.2)	0.11	0.25	0.29	0.11

a structural security posture adequate for her requirements. Thus, she chooses to apply transformation #3 for her system.

Evaluating the structural security posture for the OSM system showcased its usefulness in supporting the evaluation of various system design alternatives to attain adequate security and enhance the developer's security knowledge of the system. We can also see how external data sources help in guiding the system architect to identify relevant vulnerabilities. What the structural security posture analysis highlighted is the value that different metrics can offer. Different metrics characterize different attributes of the system, which when brought together give a much richer view of the system as a whole. For example, we can observe from our evaluation how a standard graph theory metric such as eigenvector centrality helped us characterize the attractiveness of an element in a graph which, together with the attack surface metric which analyses potential damage that an attacker can cause if they exploited one or more elements of the system, helps drive our focus towards critical elements of the system. When we add Merak's data-driven metrics to this information, we can better understand the security implications of design decisions and security requirements.

It is also important to highlight that for every transformation made where system elements are added/modified/removed, it may result in trade-offs between other non-functional aspects of the system. For example, in our scenario, adding Data Sanitizer components, while it enhanced security, adds to the overall complexity of the system architecture and potentially may negatively incur maintainability or performance costs. Therefore, the system architect needs to consider the implications of design decisions in the context of other functional and non-functional requirements of the system.

Tool support enables system architects to visualize and better explain the importance of certain attributes such as multiplicity between elements and access rights in the context of security. It also helps them focus on areas that are more in need of resources based on their operational context to mitigate potential attack opportunities and weaknesses.

5.4 Discussion

Analyzing the security posture for systems enables the examination of security characteristics associated with the various concerns of the system. Security posture is evaluated at a specific point in time to account for the progression of the system's security to an acceptable or maximal security level given adequate mitigations. We scope the security posture within the knowable threat landscape of a system as it is not possible to reliably characterize threats associated with a system that is not readily known. Our primary objective in defining a system's security posture is to

capture a system's level of preparedness to thwart known threats and to identify the weak points within its design.

Analyzing a system's security posture through its different views allows us to better reflect its overall security level as it addresses concerns associated with each view of that system. This enables system architects to compare and contrast their system designs based on security characteristics relevant to their operational context. This approach of considering multiple views for security posture analysis is similar to the notion of multi-view system modeling in model-driven engineering [DS15]. Evaluating additional views of the system also draws focus to the shift in developing requirements for elements in the system, based on their view. This was discussed by Isazadeh et al. in [ILS99].

5.5 Conclusions

In this chapter, we described a system's security posture as its security state at a specific point in time that describes its ability to defend against knowable threats that affect it. We defined a system's structural security posture. We demonstrated the usefulness of evaluating a system's structural security posture using an example design scenario for an OSM system. We also presented insights gained from performing this evaluation. We believe security posture analysis will support context-driven comparisons of a system's security thanks to its incorporation of design-level metrics that assess various characteristics of the system.

Throughout this work, we introduced tools to help system architects like Alice design secure systems. With multiple tools available for secure system design, it is important to have a one-stop shop for system architects to discover and keep up with such tools over time as more tools are developed to support secure system design. In the following chapter, we discuss a toolkit we developed to house the various secure system design tools to help practitioners develop secure systems.

Chapter 6

Compass: A Toolkit for Tools Supporting Secure System Design

In this chapter, we present a toolkit called Compass that can house the tools presented in this work, as well as those that may be developed in the future. Section 6.2 describes the requirements for the Compass toolkit. Section 6.3 details the architecture behind Compass that allows it to support existing and new secure system design tools. Section 6.4 describes how researchers and developers can develop tools to add to the Compass toolkit. Section 6.5 discusses our vision for Compass in being a one-stop shop for secure system design tools. Lastly, Section 6.6 presents our conclusions from this chapter.

6.1 The Need for a Secure System Design Toolkit

In this work, we introduced multiple tools to support secure system design and targeted them towards system architects, developers, evaluators, even those who are simply curious about secure system design (like students). However, it is hard for architects and other stakeholders to keep up with the availability and improvement of these tools over time. This problem is compounded as more secure system design tools are released.

To tackle this problem, we developed Compass, a web-based toolkit to house secure system design tools. The goal with Compass is to create a one-stop shop for such tools to help architects and other stakeholders keep up with the different tools and techniques to design secure systems.

6.2 Requirements for Compass

With Compass, we aim to abstract the challenges associated with deploying and maintaining web-based secure system design tools. Such challenges include scaling tools based on usage, handling security-related concerns such as establishing secure connections between the tools and users, managing the availability of tools, and more. By abstracting these concerns, we seek to enable researchers and developers to primarily focus their efforts on the functionality and intuitiveness of their tools rather than the logistics of deploying a web-based application.

We summarize our primary requirements for Compass as follows:

1. Support the scalable deployment of heterogeneous cloud-native web applications
2. Provide a straightforward method for new tools to be added to the toolkit
3. Automate the management of application availability (e.g., through load balancing, application health checks, etc.)
4. Automatically manage the deployment of tools (e.g., issue digital certificates, look for updates to the tool and deploy them, etc.)
5. Enable the deployment of applications with independent code repositories and development pipelines
6. Provide a web application to showcase the available tools deployed within the toolkit

6.3 Compass Architecture

Compass is based on a service-oriented architecture [PL03] and enables heterogeneous tools to be in the toolkit. We illustrate a high-level view of the Compass architecture in Figure 6.1. Researchers and developers can develop their applications using a technology stack of their choice. We accomplished this in two ways: application containerization and a cloud-native application proxy (cloud-native refers to applications built to run on cloud infrastructures).

Containerization refers to the packaging of a software application's code along with an operating system and associated dependencies into a lightweight executable called

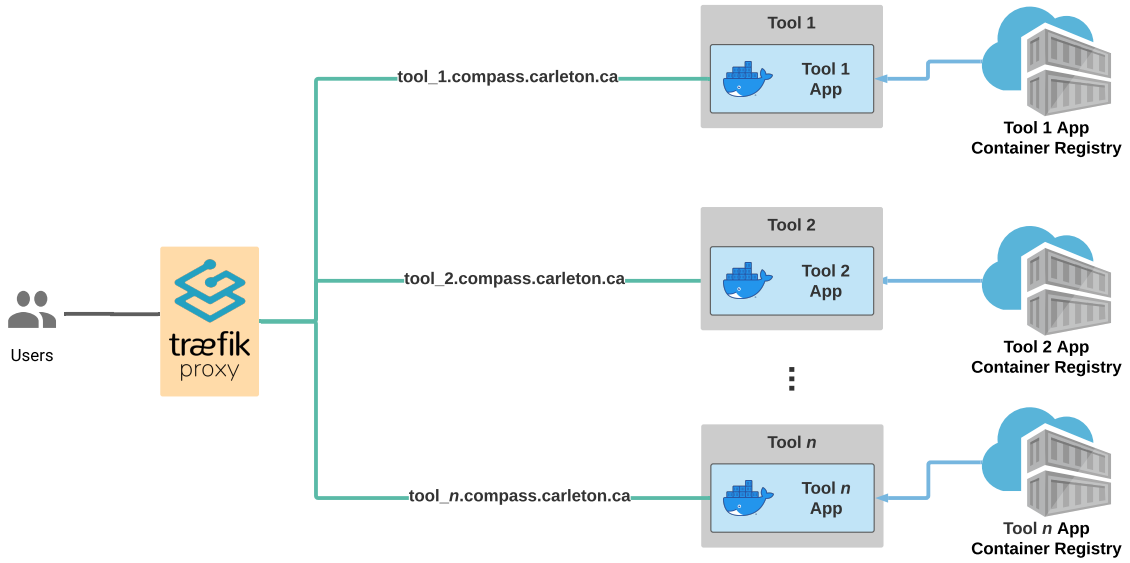


Figure 6.1: High-level architecture of Compass

a container [DRK14]. Containers provide isolation from the host environment and enable the application to perform in a predictable manner since it was developed and deployed within the same [container] environment. Containers help in standardizing different types of applications to run on any supported infrastructure (one that can run that particular container technology). Containers are saved as container images and those images are stored in container registries. To instantiate the application, the container image needs to be pulled from the container registry and the appropriate container technology can be used to create one or more containers (instances of the application) based on the container image. This allows an application to be replicated, instantiated, and managed independently which helps in maintaining high availability and resiliency. For Compass, we use Docker as the container technology to create and deploy containers [Boe15]. We chose Docker as it is a popular container technology that is commonly used to develop cloud-native applications. We illustrate the Docker

containers associated with the different tools in Compass as blue rectangles (with the Docker logo) in Figure 6.1.

To deploy the containers associated with each tool and to make them accessible on the web, we use a reverse proxy and load balancer technology called Traefik [SM21]. Traefik is central to the way Compass deploys and manages containers. A core task for Traefik is to route web traffic securely to the requested containers. For example, when a user accesses `https://polaris.compass.carleton.ca/`, Traefik identifies the containers associated with Polaris and routes traffic to the appropriate container that can handle that request. Traefik also manages load balancing between containers to ensure every request is distributed across multiple containers (and servers) to optimize response time and improve operational efficiency by distributing the load across available resources. Traefik also monitors the health of containers – that is when a container fails pre-defined operational criteria such as maximum response time – and automatically reroutes traffic to healthy containers while eliminating the failed container(s). We also configured Traefik to simplify the process of adding a new tool to Compass. For example, Traefik automatically manages the digital certificates (such as SSL certificates) for the various tools that are part of Compass. We also developed a script to look for updates for each container and rollout those updates without disrupting the availability of the other tools and Compass. We illustrate Traefik and its role in the Compass architecture in Figure 6.1.

6.4 Developing Tools for Compass

Developing tools to be deployed on Compass is fairly straightforward. Developers have the freedom to select any technology stack that allows them to deploy their tool as a cloud-native web application. This allows each tool to have independent code repositories and development pipelines and contribution policies (such as being open-source). Because we use Docker as our container technology, we, therefore, require tools to be containerized using Docker to enable us to deploy and manage the application using Compass. The image for the containerized application needs to be stored in a container registry accessible to Compass which takes care of deploying and managing the access and availability of the tool. Tools deployed on Compass will be shown on the Compass website as shown in Figure 6.2.

The code repositories for Compass and its associated services are available at `https://gitlab.com/compass-toolkit/`. Both Merak and Polaris presented in this thesis are available as tools in Compass.

6.5 Discussion

Thanks to Compass abstracting challenges associated with deploying web applications such as availability, infrastructure, and security, researchers and developers can focus their efforts more on their tool's functionality. We also hope it encourages more researchers and developers to develop more web-based tools as it benefits a wide range of stakeholders by being more accessible. By enabling future secure system

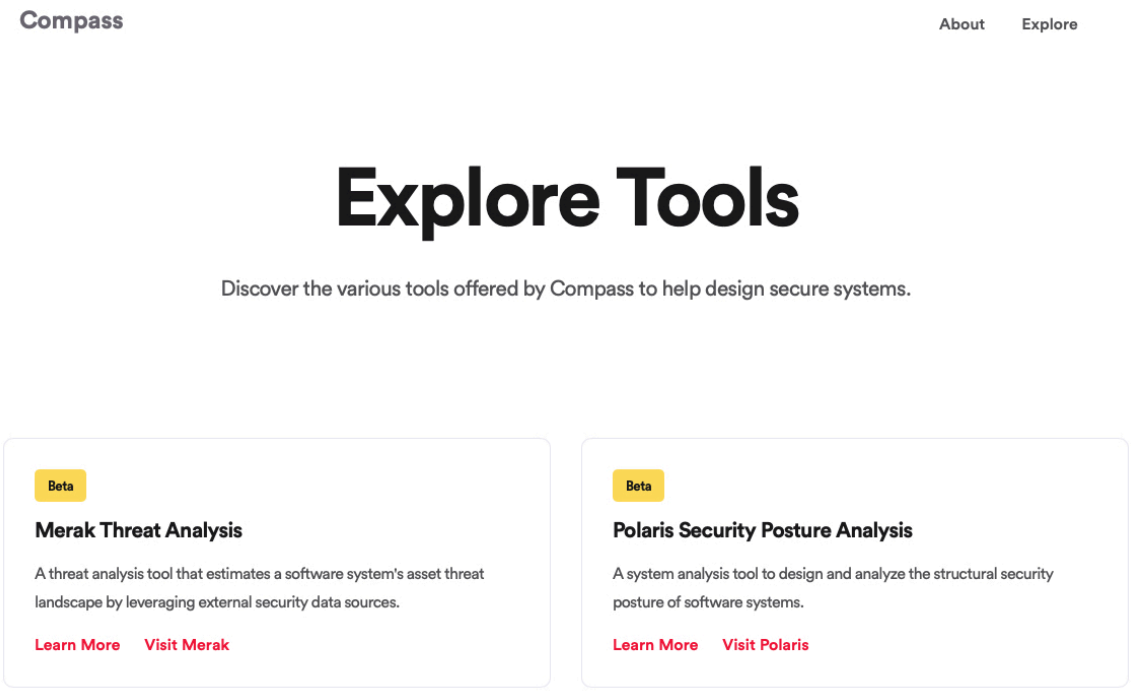


Figure 6.2: A screenshot of the Explore page in Compass showcasing the available tools

design tools to be on Compass, the likelihood for such tools to be discovered is much higher than if they were available separately. Open-source tools will have links to their code repositories to encourage community contributions to support such tools.

6.6 Conclusions

In this chapter, we presented Compass as a toolkit for secure system design tools. We believe this toolkit (and this work) will inspire more secure system design researchers to develop tools for practitioners to apply, try out, and improve. With Compass and the tools presented in this work, we strived to address the lack of adequate tool

support for practitioners to apply secure system design research in their workflows which we highlighted in Chapter 1.

Compass is available at `https://compass.carleton.ca/`.

Chapter 7

Impact, Open Research Areas, Concluding Remarks

In this thesis, we presented a data-driven system security evaluation approach to support system architects with minimal security expertise to design secure systems. As part of this approach, we illustrated how we can leverage publicly available threat and vulnerability data sources to derive data-driven metrics in Chapter 3. We also evaluated existing security metrics for soundness in Chapter 4 to identify trustworthy security metrics that we can incorporate into our evaluation approach. In Chapter 5, we defined structural security posture as our security evaluation approach to evaluate system design decisions, which enhances the system architect's security knowledge of the system. This approach leveraged external data sources and system-level and element-level metrics to assess a system's security based on its structural view. We also demonstrated the usefulness of evaluating a system's structural security posture

by evaluating it for the OSM system. We also showcased a toolkit in Chapter 6 to house the various tools developed to help system architects apply the various methods presented in this work.

7.1 Impact of the Contributions

In this work, we presented several research and tool contributions to enable system architects to evaluate and manage the security of their system designs. In this section, we discuss the impact of these contributions.

1. **An approach for leveraging external data sources to support secure system design**: We addressed some of the challenges relating to data-driven approaches for evaluating the security of a system that we discussed in Section 1.5. In Chapter 3, we presented different types of data that architects can leverage to make more informed choices on their design decisions and security requirements for their systems. We showed how this information can be presented in an actionable way enabling more development teams, that may not be highly proficient in security, learn about the threat landscape affecting their systems. We also presented Merak, our asset threat analysis tool that automates the analysis process and presents interactive, actionable information for architects to review. We showed how Merak helps distill vast amounts of security data into more digestible pieces of information for system architects. We also demonstrated how Merak helps system architects find potential patterns of threats that may affect their assets and design mitigations. By showcasing how

information from different types of publicly available data sources can support secure system design, we believe it will influence how organizations and individuals harness such sources of information during the system design phase to make more informed decisions for designing secure systems. Our approach of extracting relevant keywords from security requirements using NLP is also a notable contribution to the field of requirements engineering.

2. **Evaluation of existing security metrics for soundness**: The soundness evaluation of widely-used security metrics presented in Chapter 4 enabled us to present insights into patterns that led to a metric's failure in the soundness evaluation. These insights and recommendations assist those who design security metrics or are in the process of selecting one or more security metrics, in making an informed choice on which metrics are reliable enough to base their decisions on.

3. **A data-driven approach to evaluate the security of a system's design**: To address the challenge of how no single security metrics can sufficiently assess various characteristics of the system, we presented the concept of security posture for a system. In this work, we focused on the structural security posture of a system. In Section 5.3, we detailed how analyzing the structural security posture for a system can be used to evaluate the security implications of design decisions. We presented Polaris, a tool to automate the process of evaluating the structural security posture of a system design. Not only does Polaris enable system architects to design, annotate, and analyze their system models, but it also facilitates 'what-if' analyses to evaluate the implications of different

design decisions. This can help system architects to make much more informed decisions about the ways in which they can improve their system designs.

4. **Developing much-needed secure system design tool support**: In this work, we showcased the various tools developed to help practitioners apply this research into their workflows. While developing these tools for this work, we identified the need for a single place to house various tools designed to support secure system design. Addressing this need further helps to address the lack of adequate tool support for system architects which we discussed in Section 1.5. As a result, we introduced Compass, a web-based toolkit to house secure system design tools. It also promotes secure system design researchers and developers to develop tools that practitioners can use to apply state-of-the-art secure system design research.

7.2 Open Research Areas

In this section, we discuss open research areas based on our contributions.

1. **Exploring the other views for security posture**: In this work, we laid the foundation for defining and measuring a system's structural security posture. However, there are other views, such as the functional or behavioural view that can be explored. As security posture is defined for each of those views, an interesting line of research would be to evaluate the correlations between the security posture results between these different views. How do they vary and why? Do they tell us a consistent, holistic story about the system's security posture?

Studies could also be conducted to evaluate the effects of applying well-known design patterns into system designs and analyzing how drastic the impact can be. An example of this approach is how we integrated components responsible for data sanitization which can be characterized as applying a system-level Input Validation design pattern. On the tool side of things, parameters used to perform 'what-if' analyses can also be expanded to further enrich the options available for experimentation and evaluate their effect on other metrics in the security posture analysis. Effective visualizations for the various metrics of security posture can be also be incorporated to further enhance the ease of understanding the analysis.

2. **Improving NLP for cybersecurity-specific applications**: A machine learning-based domain-specific NER can be developed to detect a broader range of entities and security controls. This will provide a more context-tuned cybersecurity NLP model to refine the keyword extraction and threat analysis steps of the approach. To keep the size of the processed data manageable, the identification of unmitigated threats can be improved and better mechanisms can be investigated to generate a more actionable set of results and present only the most relevant threats for the system architect's review. Pattern analyzers can be developed to identify common threats from the list of threats identified. For example, we can have a pattern analyzer that is able to recognize SQL Injection as a key threat based on the number of times it is mentioned across the threats identified. Our method of using NLP to extract keywords from security requirements can also be used to identify and analyze other forms of functional and non-functional requirements from natural language text. Techniques to derive

the operational context from system design documents to better prioritize or rank discovered threats can also be investigated.

3. **Deriving trustworthy security metrics from dynamic data sources**: One interesting area of research would be to look into how to obtain trustworthy metrics from dynamic data sources. While we derive security metrics from the results of Merak's data analysis, more can be done to formally evaluate such dynamically generated metrics to ensure their trustworthiness. A more detailed study can be conducted to evaluate additional patterns and nuances that make up trustworthy security metrics that can be applied when new security metrics are to be derived or existing metrics are to be revised.

4. **Enabling broader adaptation of tools**: There are many ways in which we can help system architects integrate our tools into their existing workflows. Support for industry-standard file formats such as XML Metadata Interchange (XMI) [GDB02] can be added to Polaris for importing system models designed outside of Polaris. This enables system architects to bring their existing system designs into Polaris for structural security posture analysis. Even though we used Polaris to analyze the structural security posture of the system, it can also be adapted to analyze other non-functional requirements associated with a system, such as performance, usability, and requirements. Having tools to evaluate the system based on these different types of requirements can highlight the various trade-offs between these requirements and provide more clarity to the system architect on the implications of their system design decisions. A version of Merak that does not require a network connection can help organizations that rely on private (and sometimes proprietary) repositories of threat data to integrate their

data sources with Merak and harness its threat analysis approach. To assess how intuitive Polaris and Merak are to use, usability studies using user experience metrics (such as those based on the Likert scale [Fin10]) can be administered with stakeholders.

7.3 Concluding Remarks

The need for evaluating the security of systems during the design phase is only going to increase as more organizations realize the value of mitigating known attack opportunities without incurring substantial overhead or costs. With the rapid advancement in computing power and machine learning, and the large availability of openly available security data, the need to develop tools and techniques to leverage them will be ever-greater. With adequate tool support, more system architects, evaluators, and developers can not only develop secure systems but also make more informed design decisions to evaluate designs with data-driven approaches. As the world moves increasingly towards a connected future, more systems will be targeted by attackers. Therefore, organizations must emphasize security as a critical part of their development process rather than an enhancement that can be 'bolted on' after the system is developed. The tools we presented in this work can help organizations improve and refine the security of their system at the architectural level.

Appendix A

OSM System Data

In this appendix, we present the data associated with the different transformations applied to the Online Seller of Merchandise (OSM) system as part of the structural security posture evaluation discussed in Section 5.3.

A.1 Evaluation of the Initial OSM System Design

Table A.1 presents the parameter values for the datastores associated with the initial design of the OSM system.

Table A.1: Parameters for the OSM Datastore Items in the initial design

Datastore Item	Access Rights	Data Type
Customer Datastore	5	10
Inventory Datastore	5	1

Table A.2 presents the parameter values for the links associated with the initial design of the OSM system.

Table A.2: Parameters for the OSM Links in the initial design

Source	Target	Access Rights	Protocol
Web Server	Web Client	1	10
Web Server	Identity Access Manager	1	5
Web Server	Order Processor	1	5
Identity Access Manager	Customer Manager	1	5
Customer Datastore	Customer Manager	1	5
Inventory Datastore	Inventory Management	1	5
Shipment Processor	Inventory Management	1	5
Order Processor	Shipment Processor	1	5
Order Processor	Transaction Processor	1	5
Transaction Processor	Customer Manager	1	5
Web Server	Customer Manager	1	5
Web Server	Inventory Manager	1	5

The security requirements and design decisions for elements where that information is provided was given in Table 5.3 in Section 5.3.1. These requirements and design decisions are analyzed using the approach described in Section 3.3 with Merak. The data resulting from Merak's analysis was retrieved on September 16, 2021.

A.2 Transformation #1: Experimenting with Parameters of the OSM System Elements

Table A.3 presents the parameter values for the components associated with the design of the OSM system after applying Transformation #1.

Table A.3: Parameters for the OSM Components in Transformation #1

Component	Access Rights	Privilege
Shipment Processor	5	5
Transaction Processor	5	5
Customer Manager	5	5
Web Server	5	5
Inventory Manager	1	5
Identity Access Manager	5	5
Order Processor	5	5
Web Client	1	1

Table A.4 presents the parameter values for the datastores associated with the design of the OSM system after applying Transformation #1.

Table A.4: Parameters for the OSM Datastore Items in Transformation #1

Datastore	Access Rights	Data Type
Customer Datastore	5	10
Inventory Datastore	5	1

Table A.5 presents the parameter values for the links associated with the design of the OSM system after applying Transformation #1.

Table A.5: Parameters for the OSM Links in Transformation #1

Source	Target	Access Rights	Protocol
Web Server	Web Client	1	10
Web Server	Identity Access Manager	1	5
Web Server	Order Processor	1	5
Identity Access Manager	Customer Manager	1	5
Customer Datastore	Customer Manager	1	5
Inventory Datastore	Inventory Manager	1	5
Order Processor	Shipment Processor	1	5
Order Processor	Transaction Processor	1	5
Web Server	Customer Manager	1	5
Web Server	Inventory Manager	1	5

A.3 Transformation #2: Sanitizing Data Inputs

Table A.6 presents the parameter values for the components associated with the design of the OSM system after applying Transformation #2.

Table A.6: Parameters for the OSM Components in Transformation #2

Component	Access Rights	Privilege
Shipment Processor	5	5
Transaction Processor	5	5
Customer Manager	5	10
Gatekeeper	5	5
Inventory Manager	1	10
Identity Access Manager	5	5
Order Processor	5	5
Data Sanitizer	1	1
Web Client	1	1
Web Server	5	5

Table A.7 presents the parameter values for the datastores associated with the design of the OSM system after applying Transformation #2.

Table A.7: Parameters for the OSM Datastore Items in Transformation #2

Datastore	Access Rights	Data Type
Customer Datastore	5	10
Inventory Datastore	5	1

Table A.8 presents the parameter values for the links associated with the design of the OSM system after applying Transformation #2.

Table A.8: Parameters for the OSM Links in Transformation #2

Source	Target	Access Rights	Protocol
Web Server	Web Client	1	5
Web Server	Identity Access Manager	1	5
Web Server	Order Processor	1	5
Identity Access Manager	Customer Manager	1	5
Customer Datastore	Customer Manager	1	5
Inventory Datastore	Inventory Management	1	5
Shipment Processor	Inventory Management	1	5
Order Processor	Shipment Processor	1	5
Order Processor	Transaction Processor	1	5
Transaction Processor	Customer Manager	1	5
Web Server	Customer Manager	1	5
Web Server	Inventory Manager	1	5
Gatekeeper	Web Server	1	5
Data Sanitizer	Web Client	10	10

A.4 Transformation #3: Combining Transformations #1 and #2

Table A.9 presents the parameter values for the components associated with the design of the OSM system after applying Transformation #3.

Table A.9: Parameters for the OSM Components in Transformation #3

Component	Access Rights	Privilege Level
Shipment Processor	5	5
Transaction Processor	5	5
Customer Manager	5	5
Gatekeeper	5	5
Inventory Manager	1	5
Identity Access Manager	5	5
Order Processor	5	5
Data Sanitizer	1	1
Web Client	1	1
Web Server	5	5

Table A.10 presents the parameter values for the datastores associated with the design of the OSM system after applying Transformation #3.

Table A.10: Parameters for the OSM Datastore Items in Transformation #3

Datastore	Access Rights	Data Type
Customer Datastore	5	10
Inventory Datastore	5	1

Table A.11 presents the parameter values for the links associated with the design of the OSM system after applying Transformation #3.

Table A.11: Parameters for the OSM Links in Transformation #3

Source	Target	Access Rights	Protocol
Web Server	Web Client	1	5
Web Server	Identity Access Manager	1	5
Web Server	Order Processor	1	5
Customer Datastore	Customer Manager	1	5
Inventory Datastore	Inventory Management	1	5
Order Processor	Shipment Processor	1	5
Order Processor	Transaction Processor	1	5
Web Server	Customer Manager	1	5
Web Server	Inventory Manager	1	5
Gatekeeper	Web Server	1	5
Data Sanitizer	Web Client	10	10

A.5 Sample Attack Surface DER Calculations

In this section, we provide sample calculations for computing the DER for a component, a datastore, and a link of the OSM system. We use the parameter values set at the initial design of the system.

Consider the Shipment Processor from Table 5.2 whose access rights is 5 and privilege is 5. We compute the DER for the Shipment Processor as follows:

$$\text{DER}_{ShipmentProcessor} = \frac{privilege\ level}{access\ rights\ required} = \frac{5}{5} = 1 \tag{A.1}$$

Consider the Customer Datastore from Table A.1 whose access rights is 5 and data type is 10. We compute the DER for the Customer Datastore as follows:

$$\text{DER}_{CustomerDatastore} = \frac{data\ type}{access\ rights\ required} = \frac{10}{5} = 2 \tag{A.2}$$

Consider the link from Web Server to Web Client shown in Table A.2 whose access rights is 1 and protocol is 10. We compute the DER for this link as follows:

$$\mathrm{DER}_{WebServertoWebClient} = \frac{protocol\ type}{access\ rights\ required} = \frac{10}{1} = 10 \qquad \text{(A.3)}$$

Bibliography

[AFC09] B. Alshammari, C. Fidge, and D. Corney. Security metrics for object-oriented class designs. In *2009 Ninth International Conference on Quality Software*, pages 11–20. IEEE, 2009.

[Ali] AlienVault. Open Threat Exchange. `https://otx.alienvault.com/`. [Accessed: Dec-2020].

[Ano] Anomali. Threatstream threat intelligence platform. `https://www.anomali.com/products/threatstream`. [Accessed: Dec-2020].

[ASKM07] KK Aggarwal, Y. Singh, A. Kaur, and R. Malhotra. Software design metrics for object-oriented software. *J. Object Technol.*, 6(1):121–138, 2007.

[Aus] Australian Cyber Security Centre. Alerts. `https://www.cyber.gov.au/acsc/view-all-content/alerts`. [Accessed: Dec-2020].

[Bar14] S. Barnum. Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX™). White Paper Version 1.1, Revision 1, The MITRE Corporation, Feb. 2014.

[BC08] K. Beznosov and B. Chess. Security for the rest of us: An industry perspective on the secure-software challenge. *IEEE Software*, 25(1):10–12, 2008.

[Bek21] E. Bekker. 2021 data breaches - the worst breaches of the year: Identityforce®, Jul 2021.

[BO] B. Byers and H. Owen. Automation support for CVE retrieval. `https://csrc.nist.gov/CSRC/media/Projects/National-Vulnerability-Database/documents/web%20service%20documentation/Automation%20Support%20for%20CVE%20Retrieval.pdf`. [Accessed: Jan-2021].

[Boe15] C. Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[Bon87] P. Bonacich. Power and centrality: A family of measures. *American journal of sociology*, 92(5):1170–1182, 1987.

[Bro16] M. Bromiley. Threat intelligence: What it is, and how to use it efectively. White paper, SANS Institute, Sep. 2016.

[BSC+20] G. Bakirtzis, B. J. Simon, A. G. Collins, C. H. Fleming, and C. R. Elks. Data-driven vulnerability exploration for design phase system analysis. *IEEE Systems Journal*, 14(4):4864–4873, Dec. 2020.

[Can] Canadian Centre for Cyber Security. Alerts & advisories. `https://cyber.gc.ca/en/alerts-advisories`. [Accessed: Dec-2020].

[CDRS12] J. Connolly, M. Davidson, M. Richard, and C. Skorupka. The Trusted Automated eXchange of Indicator Information (TAXII™). White paper, The MITRE Corporation, Nov. 2012.

[Cho09] A. R. Choudhary. In-depth analysis of ipv6 security posture. In *2009 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1–7. IEEE, 2009.

[CK08] S Chandra and RA Khan. Object oriented software security estimation life cycle-design phase perspective. *Journal of Software Engineering*, 2(1):39–46, 2008.

[CKA09] S. Chandra, R. A. Khan, and A. Agrawal. Security estimation framework: design phase perspective. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 254–259. IEEE, 2009.

[Com07] Communications Security Establishment Canada. *Harmonized Threat and Risk Assessment (TRA) Methodology*. Communications Security Establishment Canada, Oct. 2007.

[Cor] Cornell University. Top vulnerability databases for open-source components. `https://blogs.cornell.edu/react/top-vulnerability-databases-for-open-source-components/`. [Accessed: Dec-2020].

[CSYW07] R. Caralli, J. Stevens, L. Young, and W. Wilson. Introducing octave allegro: Improving the information security risk assessment process. Technical Report CMU/SEI-2007-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2007.

[CZ80] J. D. Couger and R. A. Zawacki. *Motivating and managing computer personnel*. Wiley, 1980.

[DRK14] R. Dua, A. R. Raja, and D. Kakadia. Virtualization vs containerization to support paas. In *2014 IEEE International Conference on Cloud Engineering*, pages 610–614. IEEE, 2014.

[DS15] A. R. Da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[Eur] European Union Computer Emergency Response Team. Security advisories. `https://cert.europa.eu/cert/newsletter/en/latest_SecurityBulletins_.html`. [Accessed: Dec-2020].

[EYZ10] G. Elahi, E. Yu, and N. Zannone. A vulnerability-centric requirements engineering framework: Analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Requirements Engineering*, 15(1):41–62, Mar. 2010.

[Fin10] K. Finstad. The usability metric for user experience. *Interacting with Computers*, 22(5):323–327, 2010.

[Fir] FireEye. FireEye iSIGHT Intelligence. `https://www.fireeye.com/blog/products-and-services/2016/05/introducing_fireeye.html`. [Accessed: Dec-2020].

[FIR19] FIRST. Common vulnerability scoring system v3.1: Specification document. `https://www.first.org/cvss/v3.1/specification-document`, 2019.

[FKC+16] Q. Feng, R. Kazman, Y. Cai, R. Mo, and L. Xiao. Towards an architecture-centric approach to security analysis. In *13th Working IEEE/IFIP Conference on Software Architecture*, pages 221–230, Apr. 2016.

[GDB02] T. J. Grose, G. C. Doney, and S. A. Brodsky. *Mastering Xmi: Java Programming with Xmi, XML and UML*, volume 21. John Wiley & Sons, 2002.

[GG12] J. Gennari and D. Garlan. Measuring attack surface in software architecture. *Technical Report CMU-ISR-11–121, Carnegie Mellon University, Tech. Rep.*, 2012.

[Gol12] E. H. Goldman. The effect of acquisition decision making on security posture. *Information Management & Computer Security*, 2012.

[HBW16] H. Hibshi, T. D. Breaux, and C. Wagner. Improving security requirements adequacy. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[Hil99] R. Hilliard. Views and viewpoints in software systems architecture. In *Position paper from the First Working IFIP Conference on Software Architecture, San Antonio*. Citeseer, 1999.

[HL06] M. Howard and S. Lipner. *The security development lifecycle*, volume 8. Microsoft Press Redmond, 2006.

[HPW05] M. Howard, J. Pincus, and J. M. Wing. Measuring relative attack surfaces. In D.T. Lee, S.P. Shieh, and J.D. Tygar, editors, *Computer Security in the 21st Century*, pages 109–137. Springer US, 2005.

[HVO+06] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering*, volume 1, pages 13–15. IEEE, 2006.

[IBM] IBM Corporation. IBM X-Force Exchange. `https://exchange.xforce.ibmcloud.com/`. [Accessed: Dec-2020].

[ILS99] A. Isazadeh, D. A. Lamb, and T. Shepard. Behavioural views for software requirements engineering. *Requirements engineering*, 4(1):19–37, 1999.

[IMP] IMPACT. Information marketplace for policy and analysis of cyber-risk & trust. `https://www.impactcybertrust.org/`. [Accessed: Dec-2020].

[Int18] International Organization for Standardization. ISO/IEC 27000:2018 Information technology – Security techniques – Information security management systems – Overview and vocabulary, Feb. 2018.

[ISO17] ISO/IEC/IEEE. Iso/iec/ieee international standard - systems and software engineering – software life cycle processes. *ISO/IEC/IEEE 12207:2017(E) First edition 2017-11*, pages 1–157, 2017.

[Jaq07] A. Jaquith. *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Pearson Education, 2007.

[Jas20] J. Jaskolka. Recommendations for effective security assurance of software-dependent systems. In K. Arai, S. Kapoor, and R. Bhatia, editors, *Intelligent Computing, SAI 2020*, volume 1230 of *Advances in Intelligent Systems and Computing*, pages 511–531. Springer, Cham, 2020.

[JBW+16] C. Johnson, L. Badger, D. Waltermire, J. Snyder, and C. Skorupka. Guide to cyber threat information sharing. Special Publication (NIST SP) 800-150, National Institute of Standards and Technology, Oct. 2016.

[Jel00] G. Jelen. SSE-CMM security metrics. In *NIST and CSSPAB Workshop*, June 2000.

[JS07] J Jürjens and Pasha Shabalin. Tools for secure systems development with uml. *International Journal on Software Tools for Technology Transfer*, 9(5):527–544, 2007.

[Jür05] J Jürjens. *Secure systems development with UML*. Springer Science & Business Media, 2005.

[JV17] Jason Jaskolka and John Villasenor. An approach for identifying and analyzing implicit interactions in distributed systems. *IEEE Transactions on Reliability*, 66(2):529–546, 2017.

[Kas] Kaseya. The national vulnerability database (nvd) explained. `https://https://www.kaseya.com/blog/2020/10/22/national-vulnerability-database-nvd/`. [Accessed: Dec-2020].

[Ker16] M. Keramati. New vulnerability scoring system for dynamic security evaluation. In *8th International Symposium on Telecommunications (IST)*, pages 746–751, Sep. 2016.

[LC97] D. Lending and N. L. Chervany. The changing systems development job: a job characteristics approach. In *Proceedings of the 1997 acm sigcpr conference on computer personnel research*, pages 127–137, 1997.

[LC98] D. Lending and N. L. Chervany. The use of case tools. In *Proceedings of the 1998 ACM SIGCPR conference on Computer personnel research*, pages 49–58, 1998.

[LZ11] Q. Liu and Y. Zhang. VRSS: A new system for rating and scoring vulnerabilities. *Computer Communications*, 34:264–273, 2011.

[Mar95] M. P. Martin. The case against case. *Journal of Systems management*, 46(1):54, 1995.

[MFMP10] D. Mellado, E. Fernández-Medina, and M. Piattini. A comparison of software design security metrics. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pages 236–242, 2010.

[MSR07] P. Mell, K. Scarfone, and S. Romanosky. The common vulnerability scoring system (CVSS) and its applicability to federal agency systems. NIST Interagency Report 7435, National Institute of Standards and Technology, Aug. 2007.

[MW10] P. K. Manadhata and J. M. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386, 2010.

[Nat] National Institute of Standards and Technology, Computer Security Resource Center. Glossary. `https://csrc.nist.gov/glossary`. [Accessed: Sep-2021].

[Nat20] National Institute of Standards and Technology. National vulnerability database. `https://nvd.nist.gov/`, July 2020.

[New] New Zealand Computer Emergency Response Team. Alerts. `https://www.cert.govt.nz/business/recent-threats/`. [Accessed: Dec-2020].

[New18] M. Newman. *Networks an introduction*. Oxford University Press, 2018.

[NF92] R. J. Norman and G. Forte. Automating the software development process: Case in the 90's. *Communications of the ACM*, 35(4):27, 1992.

[NGPN20] M. Nerwich, P. Gauravaram, H. Paik, and S. Nepal. Vulnerability database as a service for IoT. In Lejla Batina and Gang Li, editors, *Applications and Techniques in Information Security*, pages 95–107, Singapore, 2020. Springer Singapore.

[NTH89] C. R. Necco, N. W. Tsai, and K. W. Holgeson. Current usage of case software. *Journal of Systems Management*, 40(5):6, 1989.

[OASa] OASIS Open. Introduction to STIX. `https://oasis-open.github.io/cti-documentation/stix/intro`. [Accessed: Dec-2020].

[OASb] OASIS Open. Introduction to TAXII. `https://oasis-open.github.io/cti-documentation/taxii/intro.html`. [Accessed: Dec-2020].

[Pay06] S. C. Payne. A guide to security metrics. Technical report, SANS Institute, June 2006.

[PI11] M. Pavlidis and S. Islam. Sectro: A case tool for modelling security in requirements engineering using secure tropos. In *CAiSE Forum*, pages 89–96, 2011.

[PL03] R. Perrey and M. Lycett. Service-oriented architecture. In *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, pages 116–119. IEEE, 2003.

[Pro] ProofPoint. Emerging threats. `https://rules.emergingthreats.net/`. [Accessed: Dec-2020].

[ROEM18] M. Ridley, C. Otero, D. Elliott, and X. Merino. Quantifying the security posture of containerized mission critical systems. In *SoutheastCon 2018*, pages 1–4. IEEE, 2018.

[Ros02] D. A Rosenthal. Intrusion detection technology: leveraging the organization's security posture. *Information Systems Management*, 19(1):35–44, 2002.

[SAJ20] J. Samuel, K. Aalab, and J. Jaskolka. Evaluating the soundness of security metrics from vulnerability scoring frameworks. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 442–449, 2020.

[SAM+20] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas. MITRE ATT&CK: Design and philosophy. White Paper MP180360R1, The MITRE Corporation, Mar. 2020.

[Sav07] R. M. Savola. Towards a security metrics taxonomy for the information and communication technology industry. In *International Conference on Software Engineering Advances (ICSEA 2007)*, pages 60–60, Aug. 2007.

[SG04] B. Schmerl and D. Garlan. Acmestudio: Supporting style-centered architecture development. In *Proceedings. 26th International Conference on Software Engineering*, pages 704–705. IEEE, 2004.

[Sho14] A. Shostack. *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.

[SJY21] J. Samuel, J. Jaskolka, and G. Yee. Leveraging external data sources to enhance secure system design. In *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pages 1–8. IEEE, 2021.

[SM21] R. Sharma and A. Mathur. Traefik for microservices. In *Traefik API Gateway for Microservices*, pages 159–190. Springer, 2021.

[SSMB16] C. Sillaber, C. Sauerwein, A. Mussmann, and R. Breu. Data quality challenges and future research directions in threat intelligence sharing practice. In *2016 ACM on Workshop on Information Sharing and Collaborative Security*, pages 65–70, 2016.

[SW95] J. A. Senn and J. L. Wynekoop. The other side of case implementation best practices for success. *INFORMATION SYSTEM MANAGEMENT*, 12(4):7–14, 1995.

[Thea] The MITRE Corporation. Common attack pattern enumeration and classification. `https://capec.mitre.org/`. [Accessed: Dec-2020].

[Theb] The MITRE Corporation. Common vulnerabilities and exposures. `https://cve.mitre.org/`. [Accessed: Dec-2020].

[Thec] The MITRE Corporation. Common weakness enumeration. `https://cwe.mitre.org/`. [Accessed: Dec-2020].

[TM05] D. Taylor and G. McGraw. Adopting a software security improvement program. *IEEE Security & Privacy*, 3(3):88–91, 2005.

[U.K] U.K. National Cyber Security Centre. Reports & advisories. `https://cert.europa.eu/cert/newsletter/en/latest_SecurityBulletins_.html`. [Accessed: Dec-2020].

[UM15] T. UcedaVélez and M. M. Morana. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. John Wiley & Sons, first edition, 2015.

[Uni] United States Cybersecurity & Infrastructure Security Agency. National cyber awareness system – alerts. `https://us-cert.cisa.gov/ncas/alerts`. [Accessed: Dec-2020].

[WGSS11] R. Wang, L. Gao, Q. Sun, and D. Sun. An improved CVSS-based vulnerability scoring mechanism. In *3rd International Conference on Multimedia Information Networking and Security*, pages 352–355, 2011.

[Whi] WhiteSource Software. Whitesource vulnerability database. `https://www.whitesourcesoftware.com/vulnerability-database/`. [Accessed: Dec-2020].

[Wil12] G. P. Williams. Cost effective assessment of the infrastructure security posture. In *7th IET International Conference on System Safety, incorporating the Cyber Security Conference 2012*, pages 1–6, 2012.

[WL13] C. B. Weinstock and H. F. Lipson. Evidence of assurance: Laying the foundation for a credible security case. Technical report, Software Engineering Institute, Pittsburgh, PA, USA, Aug. 2013.

[WXZ08] A. J. A. Wang, M. Xia, and F. Zhang. Metrics for information security vulnerabilities. *Journal of Applied Global Research*, 1(1):48–58, 2008.

[Yee01] B. S. Yee. Security metrology and the monty hall problem. In *Workshop on Information Security System Rating and Ranking*, 2001.

[Yee19a] G. Yee. Designing good security metrics. In *IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 580–585, July 2019.

[Yee19b] G. Yee. Designing sound security metrics. *International Journal of Systems and Software Security and Protection*, 10(1):1–21, 2019.

[Yee19c] G. Yee. Reducing the attack surface for private data. In *Proc. Thirteenth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2019)*, pages 28–34, 2019.

[YMR14] A. A. Younis, Y. K. Malaiya, and I. Ray. Using attack surface entry points and reachability analysis to assess the risk of software vulnerability exploitability. In *IEEE 15th International Symposium on High-Assurance Systems Engineering*, pages 1–8, Jan. 2014.

[YMR16] A. Younis, Y. K. Malaiya, and I. Ray. Assessing vulnerability exploitability risk using software properties. *Software Quality Journal*, 24(1):159–202, 2016.