Polaris

Release 1.0.0

Joe Samuel

Apr 18, 2022

OVERVIEW

Polaris offers anyone with a web browser to design and analyze the structural security posture of their system. Think of it as a *Security Visualization* tool for system designs.

CHAPTER

ONE

OVERVIEW

In this section, you can learn more about what Polaris is and what is has to offer to help you with designing secure systems. We present a glossary and supporting references to understand the terminologies and concepts behind Polaris. We also answer some frequently asked questions (we will update this list as we get more!) and provide details on how you can contribute to the development of Polaris.

1.1 About

1.1.1 What is Polaris?

Polaris offers 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**.

1.1.2 How does Polaris help in designing secure systems?

Polaris helps system architects, developers, evaluators, and certifiers analyze their system's structural security posture. Structural security posture is a security evaluation approach to evaluate a system's preparedness to deal with knowable threats based on its structural view and to enhance the developer's security knowledge of the system. The structural security posture uses a collection of metrics to reflect a system's security. It also leverages external data sources to guide the identification of vulnerabilities (thanks to integration with Merak). Polaris also enables practitioners to perform what-if analyses to improve their system's security and make appropriate design decisions.

1.1.3 Did you know?

The name Polaris refers to 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.

1.2 Release Notes

1.2.1 1.0.0

First public beta release of Polaris.

1.3 Frequently Asked Questions

Who is behind Polaris?

Polaris is an open-source tool made available as part of Compass Toolkit. Polaris (alongside Compass) is supported by the CyberSEA Lab at Carleton University.

Is Polaris open-source?

Yes, Polaris is an open-source tool released under the Apache 2.0 License. To learn more about the provisions under this license, please visit *License*.

I have suggestions to improve Polaris. Can I submit my feedback?

Absolutely! Polaris is made to support the system design community and we would love to hear how we can improve Polaris. To submit your feedback, please visit Polaris and click on **Feedback** tab.

1.4 Glossary

Attack Surface Metric 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 [IEEE2010]. Read more *Attack Surface Metric*.

Eigenvector Centrality 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 important they are likely to be in the system. 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 [JOURNAL1987]. Read more *Eigenvector Centrality Metric*.

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 Posture A system's security posture is described as its security state at a specific point in time that reflects its ability to defend against knowable threats that affect it [QRS2021]. The concept of security posture recognizes the need for system designs to be evaluated based on a given view of the system (such as structural, behavioural, functional) since each view focuses on specific attributes of the system which we can use to quantitatively evaluate the system's security for that view.

Structural Security Posture Structural security posture is an extension of security posture approach consisting of 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. Read more: *Structural Security Posture*.

Software System (aka. System) 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. Read more: [CARLETON2021].

1.5 References

1.6 Contribute

We are open to community contributions towards improving Polaris. Pull requests that address the following issues are welcome.

- Bug fixes
- Feature improvements (please file a feature request as an issue)
- Improvements to the documentation
- Tests for existing featuress

To find the current list of issues/suggestions, please visit Polaris Issues.

Note: For major changes, please open an issue first to discuss what you would like to change. Please make sure to create/update tests as appropriate.

1.7 Report Bugs

Please file bug reports or feature requests by visiting Polaris and clicking on the **Feedback** tab.

1.8 License

(continues on next page)

(continued from previous page)

```
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
```

(continues on next page)

(continued from previous page)

```
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
```

(continues on next page)

(continued from previous page)

```
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.
```

(continues on next page)

(continued from previous page)

```
    To apply the Apache License to your work, attach the following
    boilerplate notice, with the fields enclosed by brackets "[]"
    replaced with your own identifying information. (Don't include
    the brackets!)  The text should be enclosed in the appropriate
    comment syntax for the file format. We also recommend that a
    file or class name and description of purpose be included on the
    same "printed page" as the copyright notice for easier
    identification within third-party archives.

Copyright 2021 Carleton University CyberSEA Lab

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER

TWO

THEORY BEHIND POLARIS

In this section, we delve into the theory behind designing secure systems (called *Structural Security Posture*) with Polaris. We also present details on the metrics used to compute *Structural Security Posture*.

2.1 Structural Security Posture

Polaris is a tool to analyze the structural security posture of a system's design. In this section, we explain the concept of structural security posture and how it can be useful in analyzing the security of a system's design.

A system's security posture is described as its security state at a specific point in time that reflects its ability to defend against knowable threats that affect it [QRS2021]. The concept of security posture recognizes the need for system designs to be evaluated based on a given view of the system (such as structural, behavioural, functional) since each view focuses on specific attributes of the system which we can use to quantitatively evaluate the system's security for that view. These quantitative metrics serve as indicators of a system's security posture for a given view. Polaris helps system designers evaluate the security posture of system in the structural view, called the structural security posture.

A system is comprised of several elements, which refer to any assets of the system such as components, channels, datastores, etc. Structural security posture is an approach consisting of 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. 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 Unified Modeling Language (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 [CMU2012].

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. 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. We focus our efforts on the communication between system elements (undirected edges).

2.1.1 Attack Surface Metric

We use the attack surface as a system-level metric to evaluate a system's structural security posture. 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 [IEEE2010].

The attack surface metric quantifies a system's interaction with its external environment using three types of system elements:

- 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.
- 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.
- 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 over the amount of effort needed to access that component (determined by the component's access rights). This ratio is called the Damage Effort Ratio (DER) and it corresponds to the intuition that an attacker will select the component offering the highest damage potential with the least amount of 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.

$$\mathrm{DER}_c = \frac{privilege}{access\ rights}$$

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.

$$\mathrm{DER}_l = \frac{protocol}{access\ rights}$$

DER for Datastore Items

Damage potential for items in datastores is evaluated based on the type of data they store. The more permissive (less restrictive) the data type, the higher the likelihood of its exploitation by an attacker to execute unauthorized commands.

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

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 below.

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

2.1.2 Eigenvector Centrality Metric

We can use centrality metrics as element-level metrics 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 important they are likely to be in the system. Formally, we can define centrality as a function $c : N \to \mathbb{R}$ 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. 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 [OXFORD2018]. 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 [JOURNAL1987]. 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 high-ranking 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:

$$Ax = \lambda x$$

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 [OXFORD2018].

2.1.3 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 provided by Merak, a Compass tool to analyze threats and requirements for a given component of the system [IEEE2021]. Merak leverages multiple external data sources such as the National Vulnerability Database (NVD) [NVD], MITRE ATT&CK [MITRE2018], CCCS Alerts and Advisories [CCCS] to ascertain threats that are likely to affect a system component based on its security requirements and design specifications. Polaris incorporates the following Merak metrics:

- Unmitigated threats: Identified threats for which there are currently no security requirements.
- Mitigated threats: Identified threats addressed by the provided security requirements.
- Extraneous requirements: Provided security requirements for which no relevant threats were found.

CHAPTER

THREE

USING POLARIS

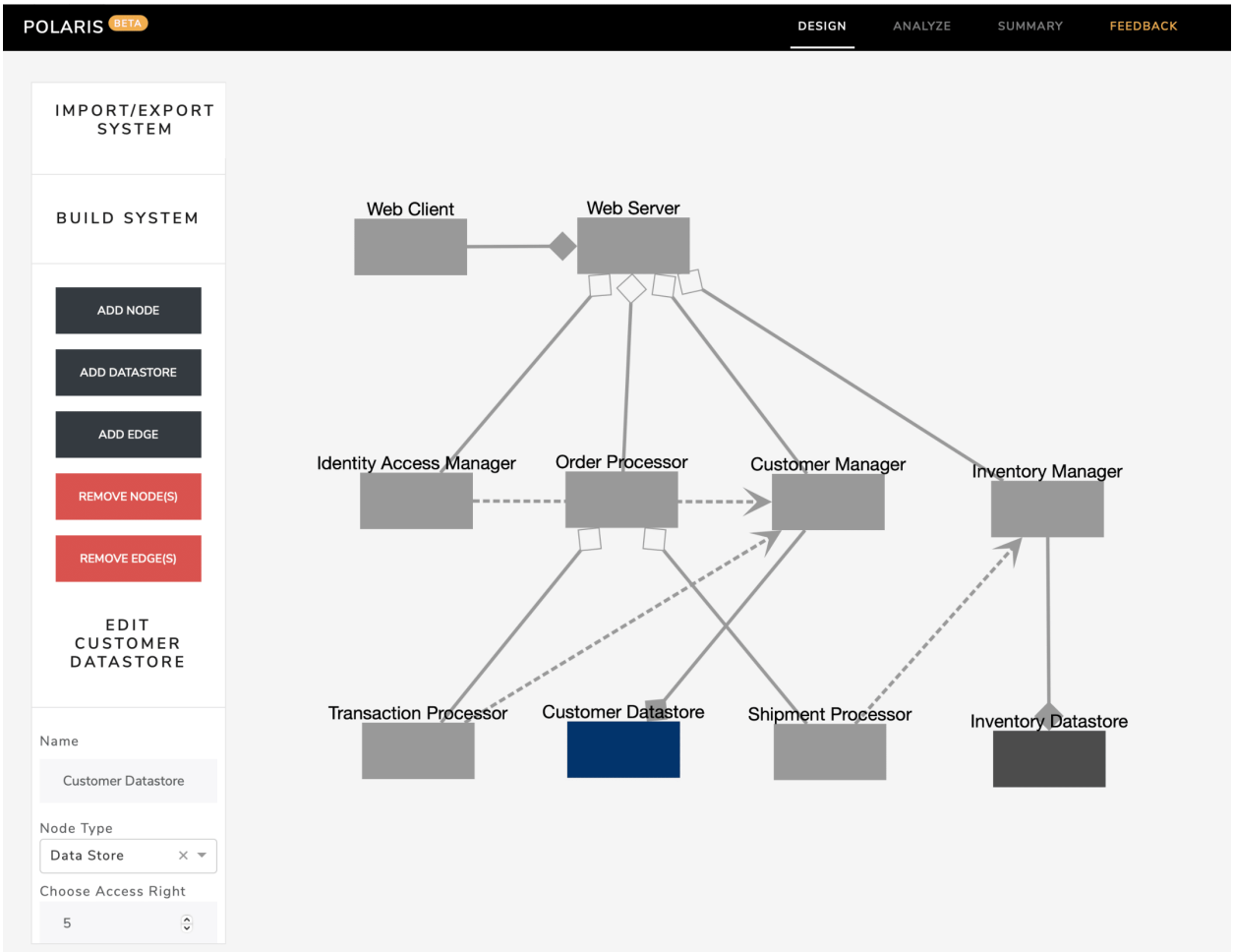
In this section, we showcase the simplicity of navigating Polaris to perform a structural security posture analysis for a system. We also present a detailed example to illustrate the usefulness of Polaris in making security-oriented decisions as part of the system design process.

3.1 Navigating Polaris

Polaris simplifies the structural security posture analysis into three steps: **Design**, **Analyze**, and **Summarize**.

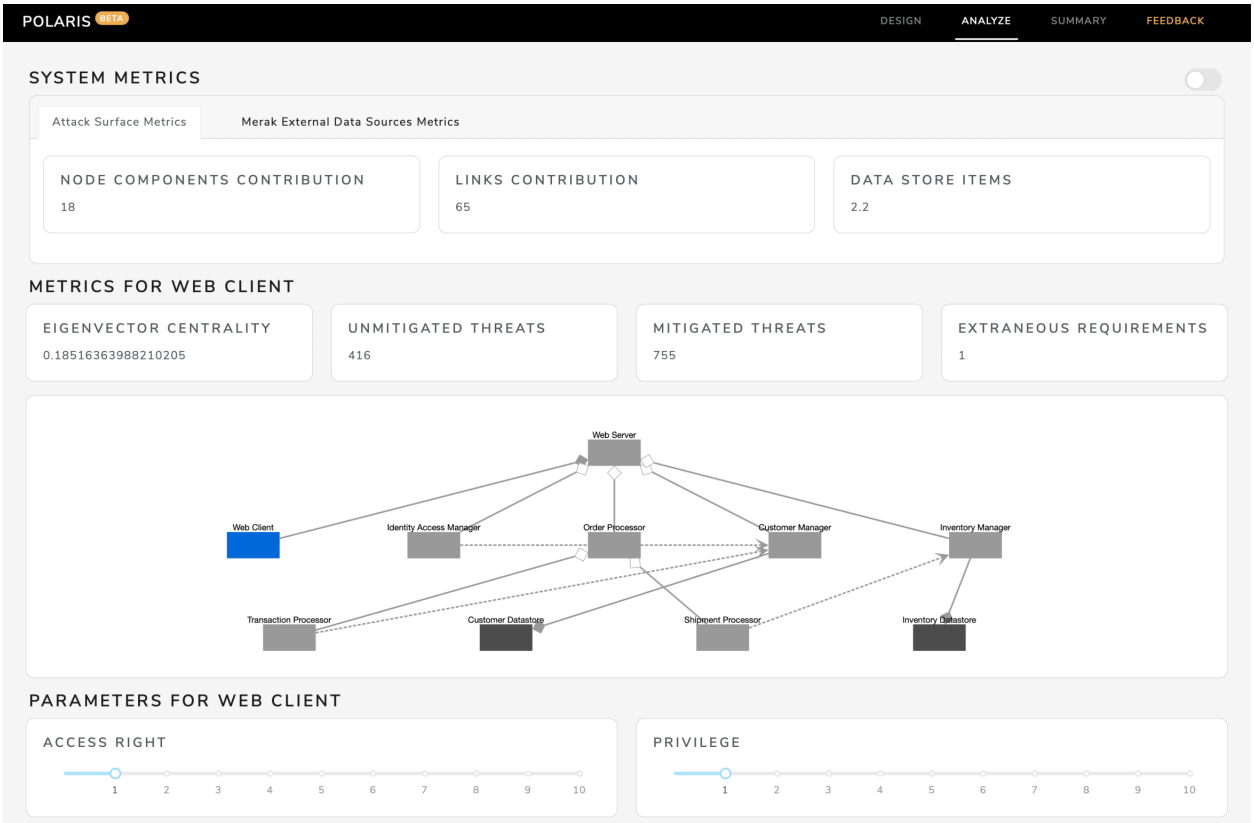
3.1.1 Design

The Design tab enables users to design their system models (using UML) and annotate those models with information such as security requirements and design decisions. It provides a graphical user interface to edit and visualize the system model as shown below. System models can also be imported and exported from this tab fulfilling user extensibility requirements.



3.1.2 Analyze

The Analyze tab performs the structural security posture analysis on the system designed or imported into Polaris in the Design tab. It provides a graphical user interface (UI) to edit and visualize the system model as shown below.



The Analyze tab is composed of three main sections.

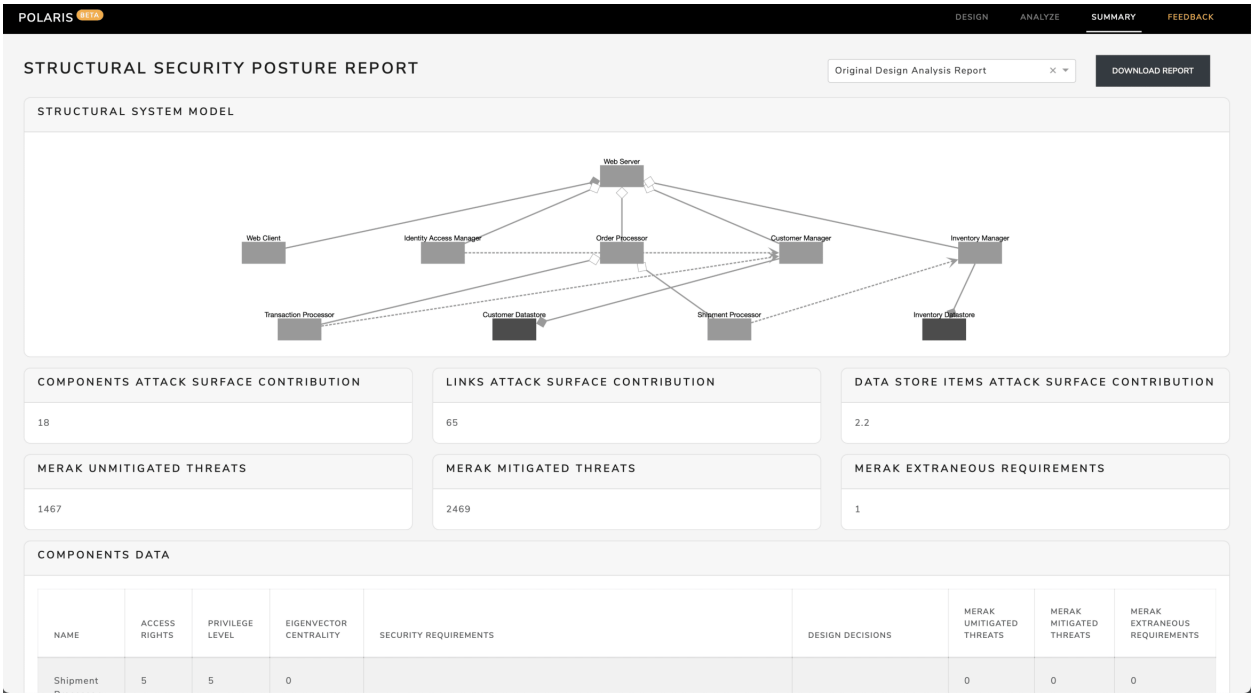
The first section shows the system-level and element-level metrics for the system. A tabbed interface helps architects view relevant information in a concise form. It also helps ensure we do not overwhelm the architect with numbers from all our metrics at once, which would go against our requirement 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 *Structural Security Posture* are automatically computed based on the parameters chosen during the system design. Polaris also incorporates the data-driven threat metrics described in *Data-driven Threat Metrics* to leverage external data sources to perform threat analysis. To obtain these metrics, we use Merak. 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' analysis dynamically by manipulating values set during the design of the system to identify its effect on the system's 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 security posture evaluation. If 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.

3.1.3 Summarize

The Summarize tab, shown below, 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.



3.2 Example Evaluation

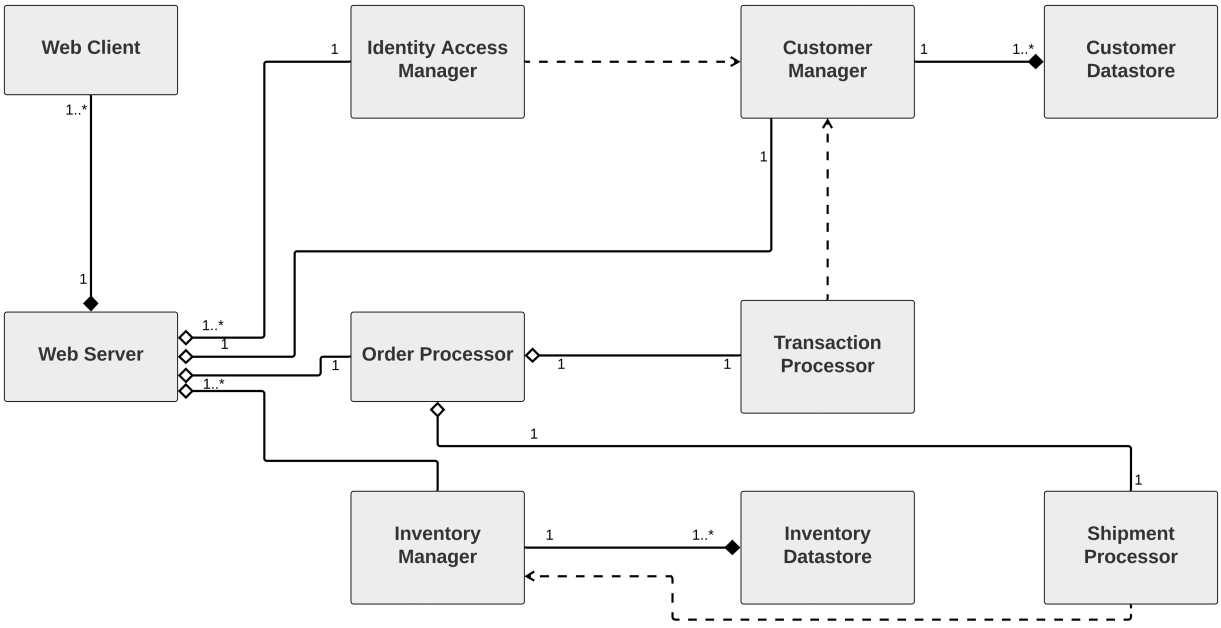
We present an information technology (IT) system to illustrate the usefulness of structural security posture for evaluating design decisions. We adapt the example of the Online Seller of Merchandise (OSM) system presented in [QRS2021].

The OSM system has the following system elements:

- Web Client: A web-based interface for customers to interact with the OSM system.
- Web Server: Orchestrates the purchase process initiated by the customer and serves the relevant information (such as order status, shipping status, items available) to the Web Client.
- Identity Access Manager: Ensures the customer is authorized to perform the actions they are requesting (such as placing an order for 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 quantity of the item(s) 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: Processes 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 passing that information to the Shipment Processor to ship the requested product(s).

We represent the structural view of the system using the UML class diagram notation shown below.



Note: We only describe the elements of the OSM system critical to a simple transaction (one where the product(s) requested are available and customer information is available for the order to be processed) as this is sufficient to illustrate our contributions.

Tip: You can download the files generated by this analysis and import it into Polaris to view the results for yourself.

- Original Design
- Transformation #1
- Transformation #2
- Transformation #3

Let's suppose that Alice is the system architect responsible for designing the OSM system. The 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 system shown above. However, she aims to improve her initial design in terms of security.

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 is aware that most attacks targeting information systems like the OSM system tend to follow the data (especially sensitive data) [IEEE2019], so she focuses her initial efforts on securing the Customer Datastore as it contains highly sensitive and valuable information that needs to be protected. The Customer Datastore is also critical to the functionality of the OSM system as customers who are not authenticated cannot place an order.

Alice adapts the attack surface parameters for the elements of her system to be the same 1-to-10 scale chosen by Gennari and Garlan in their attack surface example [CMU2012]. The interpretation of the chosen scales for each element type is shown in the table below. Transmission Control Protocol (TCP) 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 an element. For example, a component with guest privilege will have significantly reduced privileges and therefore be assigned the lowest privilege value (1); a component with root privilege will have the least restrictive privilege and therefore be assigned the highest privilege value (10).

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

With her initial sketch of the OSM system, Alice uses the Design tab of Polaris to input her system model 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 and Customer Manager. While this is fairly straightforward to design and implement, it may have drastic implications on the security of the system. One example of the 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 based on the above table as initial parameters for the elements of her system. The following table details the values chosen for the elements of the initial design of her system.

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 (5) as its access rights. Alice chooses SQL as the data type as she anticipates the Customer Datastore to hold a diverse range of information such as customer names, addresses, phone numbers, and credit cards which SQL supports. 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) [IEEE2006]. 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 only allow authenticated 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 uses Polaris to perform this experiment. Polaris allows Alice to perform 'what-if' analyses thanks to its automated computation of system-level and element-level metrics based on the chosen parameters.

3.2.1 Evaluation of the Initial OSM System Design

Alice initializes her OSM system design in Polaris as shown in the UML diagram earlier. 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 the table below, 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 [IEEE2021].

Component	Security Requirements	Design Decisions	Unmitigated	Mitigated	Extraneous
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.	416	755	1
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.	119	519	0

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 *Eigenvector Centrality Metric* 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 allow an attacker to access functions associated with that privilege level and will likely be able 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 the figure below, based on the security requirements and design decisions for the Customer Datastore detailed above. 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 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.

CVE-2021-2385

CVSS Score 5

Vulnerability in the MySQL Server product of Oracle MySQL (component: Server: Replication). Supported versions that are affected are 5.7.34 and prior and 8.0.25 and prior. Difficult to exploit vulnerability allows high privileged attacker with network access via multiple protocols to compromise MySQL Server. Successful attacks of this vulnerability can result in unauthorized ability to cause a hang or frequently repeatable crash (complete DOS) of MySQL Server as well as unauthorized update, insert or delete access to some of MySQL Server accessible data. CVSS 3.1 Base Score 5.0 (Integrity and Availability impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:H).

CVE-2021-25213

SQL injection vulnerability in SourceCodester Travel Management System v 1.0 allows remote attackers to execute arbitrary SQL statements, via the catid parameter to subcat.php.

CVE-2021-25209

SQL injection vulnerability in SourceCodester Theme Park Ticketing System v 1.0 allows remote attackers to execute arbitrary SQL statements, via the id parameter to view_user.php .

CVE-2021-25205

SQL injection vulnerability in SourceCodester E-Commerce Website V 1.0 allows remote attackers to execute arbitrary SQL statements, via the update parameter to empViewUpdate.php .

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:

- Modify the access rights, privileges, and protocols of the system elements involved in conducting such attacks.
- Create a Data Sanitizer component to sanitize the input from the Web Client before it reaches the Customer Datastore.
- 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. Such an 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 see if changing these values or the structure of the design has any unintentional impact on the security of other system elements.

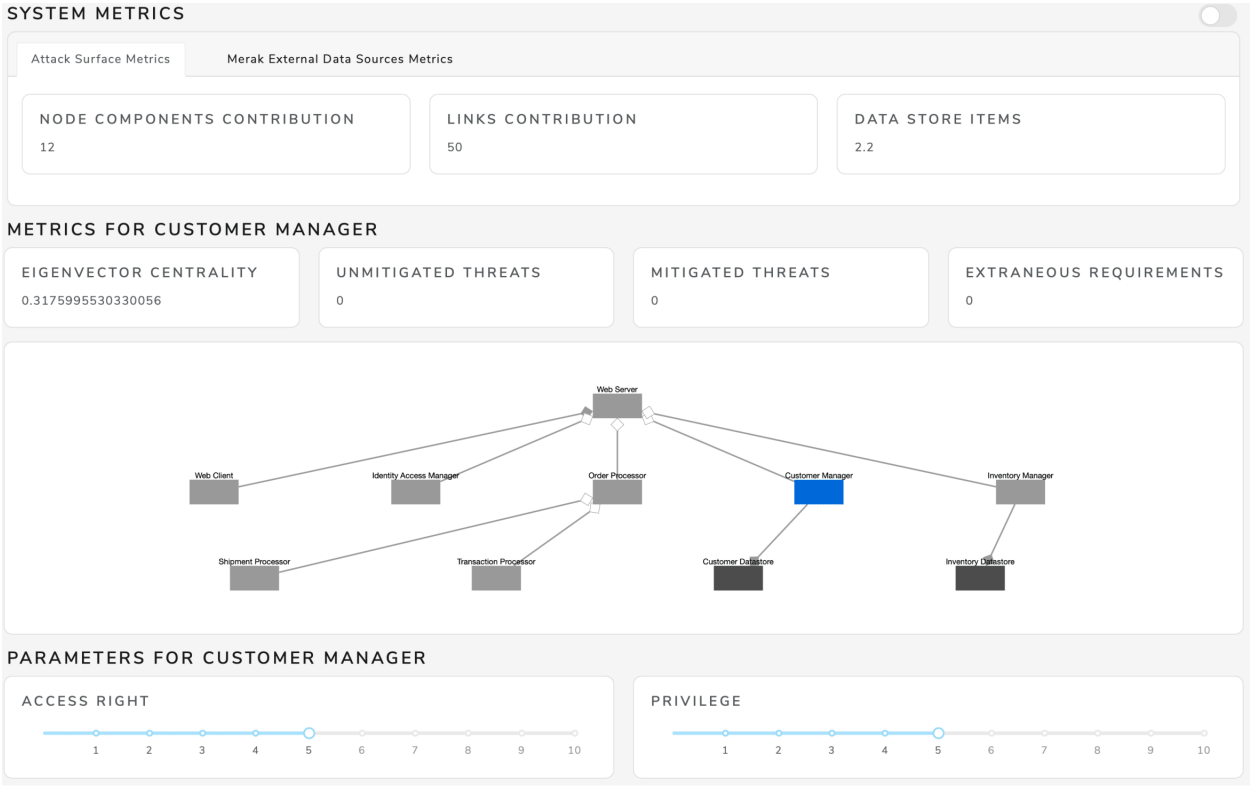
Alice applies each of the above transformations and evaluates the change in the system's structural security posture, as follows.

3.2.2 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.

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.

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 below.



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

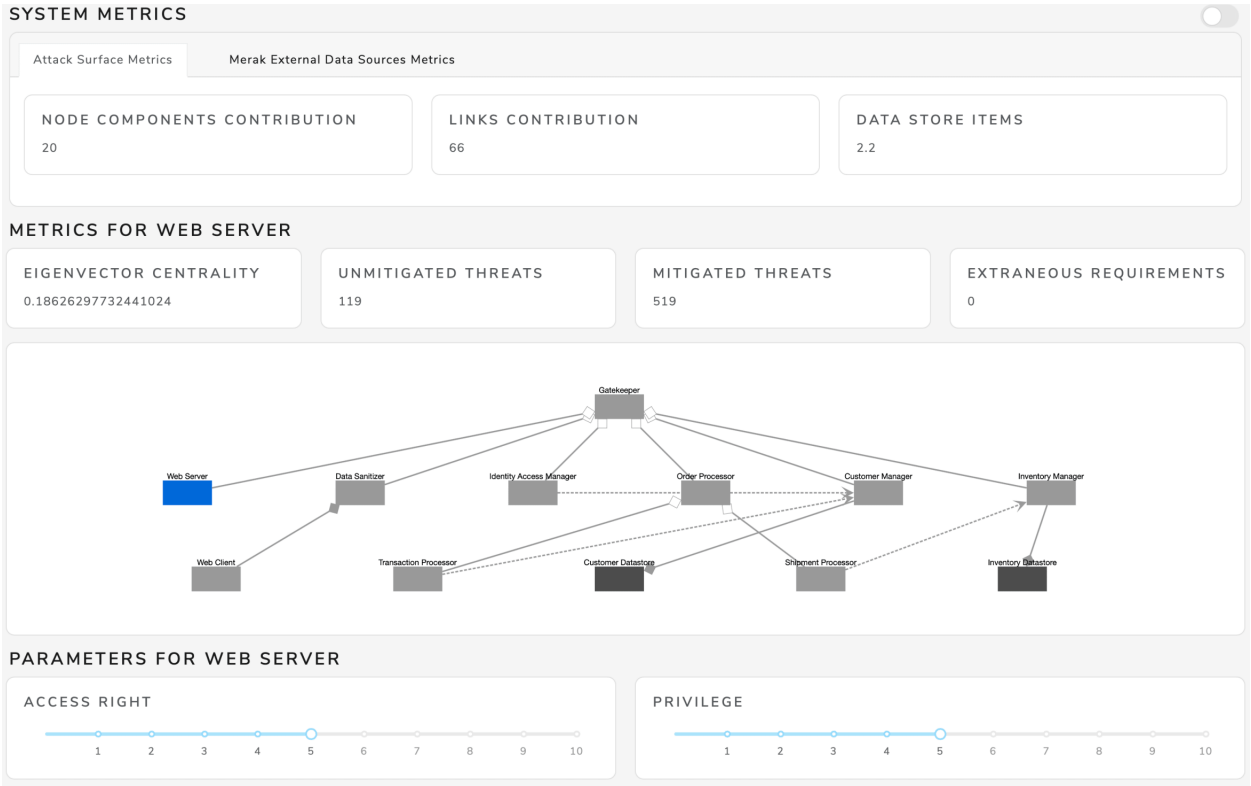
The importance of the Web Client, Web Server, and Customer Datastore increased as the importance of their neighbouring elements increased due to the removal of redundant links which reduces the paths to certain elements and increases the importance of elements that form that path. The Customer Manager 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.

3.2.3 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. Both of these components act as gatekeepers to protect sensitive elements of the system by validating any user input.

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. Alice reflects this 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.

Note that Alice applies these transformations on her original system model, resulting in the below results.



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 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.

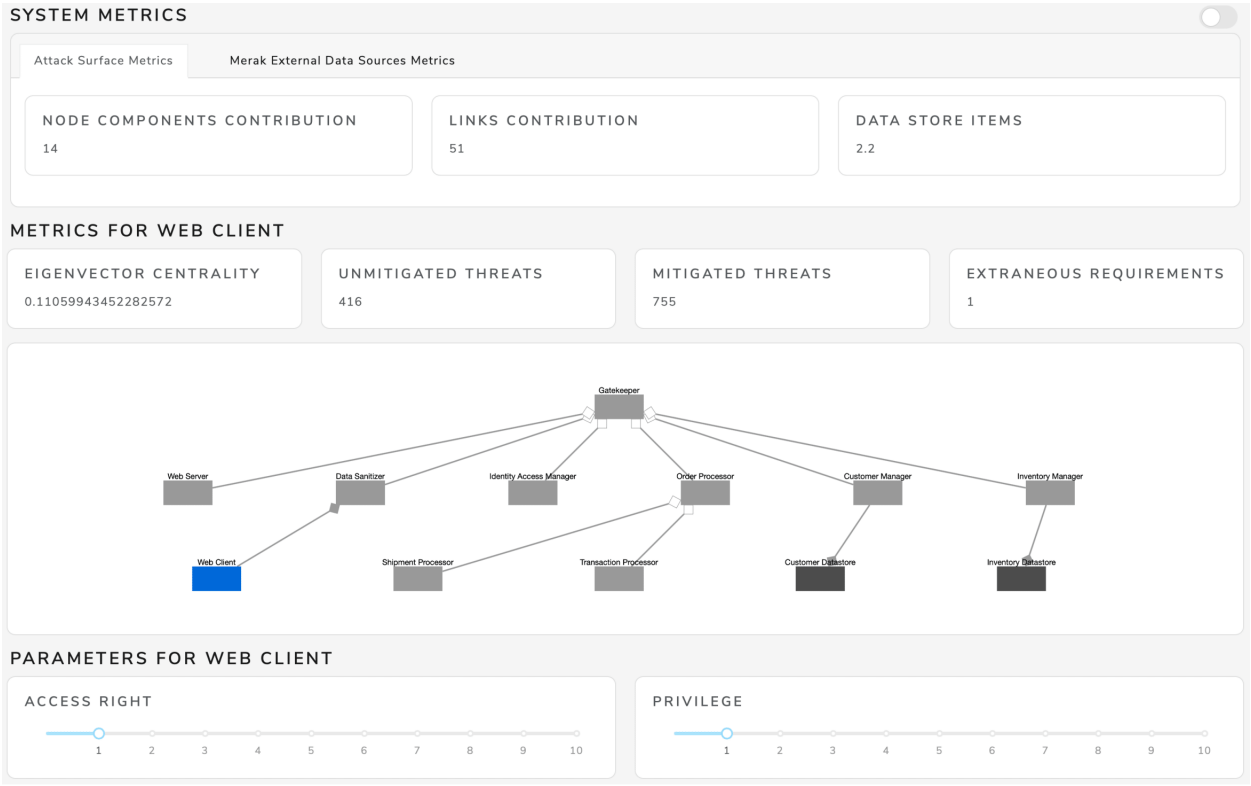
Alice notices how the Web Client and Web Server saw a significant reduction in their importance following the introduction of the Data Sanitizer and Gatekeeper components respectively. A notable observation here was how the Customer Manager was affected by the introduction of the Gatekeeper only by a 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.

3.2.4 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 third transformation, Alice proceeds to evaluate the combination of both of 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 below.



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 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.

3.2.5 Experiment Conclusions

A summary of Alice's structural security posture evaluation is shown in the below table. 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 a structural security posture adequate for her requirements. Thus, she chooses to apply Transformation #3 for her system.

Design	Attack Surface	Web Client Centrality	Web Client Centrality	Customer Manager Centrality	Customer Datastore Centrality
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

This experiment shows how a system architect can evaluate different design decisions for their impact on system security using structural security posture based on requirements. This experiment also highlights the importance of providing adequate tool support to enable the system architect to perform the above evaluations.

BIBLIOGRAPHY

[QRS2021] Samuel, J., Jaskolka, J., & Yee, G. O. (2021, December). Analyzing Structural Security Posture to Evaluate System Design Decisions. In 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS) (pp. 8-17). IEEE.

[CMU2012] Gennari, J., & Garlan, D. (2012). Measuring attack surface in software architecture. Technical Report CMU-ISR-11–121, Carnegie Mellon University, Tech. Rep.

[IEEE2019] Yee, G. O. (2019, May). Modeling and reducing the attack surface in software systems. In 2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE) (pp. 55-62). IEEE.

[IEEE2006] Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE international symposium on secure software engineering (Vol. 1, pp. 13-15). IEEE.

[IEEE2021] Samuel, J., Jaskolka, J., & Yee, G. O. (2021, May). Leveraging External Data Sources to Enhance Secure System Design. In 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS) (pp. 1-8). IEEE.

[IEEE2010] Manadhata, P. K., & Wing, J. M. (2010). An attack surface metric. IEEE Transactions on Software Engineering, 37(3), 371-386.

[OXFORD2018] Newman, M. (2018). Networks. Oxford university press.

[JOURNAL1987] Bonacich, P. (1987). Power and centrality: A family of measures. American journal of sociology, 92(5), 1170-1182.

[NVD] National Institute of Standards and Technology, "National Vulnerability Database (NVD).", https://nvd.nist.gov/.

[MITRE2018] Strom, B. E., Applebaum, A., Miller, D. P., Nickels, K. C., Pennington, A. G., & Thomas, C. B. (2018). Mitre att&ck: Design and philosophy. Technical report.

[CCCS] Canadian Centre for Cyber Security, "Alerts & Advisories", https://cyber.gc.ca/en/alerts-advisories/.

[CARLETON2021] Samuel, J. F. (2021). A Data-Driven Approach to Evaluate the Security of System Designs (Doctoral dissertation, Carleton University).

INDEX