Measuring Attack Surface in Software Architecture

Jeffrey Gennari **David Garlan**

March 2012
CMU-ISR-11-121

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this report we show how to adapt the notion of "attack surface" to formally evaluate security properties at the architectural level of design and to identify vulnerabilities in architectural designs. Further we explore the application of this metric in the context of architecture-based transformations to improve security by reducing the attack surface. These transformations are described in detail and validated with a simple experiment.

The authors would like to acknowledge Pratyusa Manadhata and Jeannette Wing; without their work formalizing the attack surface metric, our work would not be possible. The authors would also like to thank the Software Engineering Institute for supporting this work. This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273 from the Army Research Office.

Keywords: attack surface, software architecture, security, software security metrics, formal models

1 Introduction

Software systems continue to grow more complex, interconnected, and pervasive in everyday life. Critical operations, such as commerce and sensitive communications are increasingly carried out within software requiring ever more stringent and robust security mechanisms. Appropriate handling of security-related events is often an essential component of system success. Failing to adequately consider security may result in a system being deemed unfit for its operating environment.

Security is widely recognized as intrinsic to the design of a system. Successfully introducing security mechanisms into a completed system is a tremendous challenge and regarded as a poor approach. Rather, security must be considered during system design to allow engineers to effectively satisfy security-related requirements and fully appreciate the implications of security-related decisions.

Developing a software architecture is an activity that occurs relatively early in the design process. Architecture captures the core structures and interactions within a system. Emphasizing high level structures and relationships enables designers to identify and appreciate a design's systemic properties and make purposeful decisions concerning those properties.

Software architecture is an ideal point in the development process to address security requirements because many security properties emerge from the arrangement and interaction of a system's core structures. Consider supporting authorization controls, which is a fairly common security requirement. Significant concerns, such as where to store privilege information, how to manage user sessions and roles, and when to check access must be addressed at a structural level to ensure authorization mechanisms are consistent and correct. Failure to address these concerns during design may have dire consequences in the final system.

In general, the design strategies that architects select lay the foundations for achieving security-related requirements. Such decisions must be made with care because different design strategies yield different security properties. To determine the effectiveness of security-related decisions, good techniques are needed to evaluate architectural designs in terms of security requirements. Failing to consider security at an architectural may result in a system that provides inadequate or weak security mechanisms, possibly making the system unfit for its operating environment.

Furthermore, because security permeates core system structures implementing security features after the final system system is constructed may require significant and expensive redesign efforts. Clearly, security must be considered during architectural design; however, determining the effectiveness of security-related designs and strategies remains a difficult task. Architects need good security metrics to evaluate and compare the merits of designs early in the development process when structural changes are relatively easy an inexpensive to make. In particular, a good security metric that can be used at an architectural level would be especially useful.

The notion of attack surface is a potentially useful concept for evaluating the security of a system. Attack surface has long been understood in the security community as a measure of a system's exposure to attack [16]. If a system has a small attack surface, then it is considered less vulnerable to attack by virtue of the fact that there are fewer ways that an adversary can access that system. Although fairly well known, attack surface has traditionally been an informal concept discussed in vague terms and difficult to quantify. Security practitioners appreciated the concepts behind the term "attack surface" as it related to their systems, but there was no general measurement available.

The work of Manadhata and Wing [1] formalizes the attack surface concept into a quantifiable measure that is applicable to any software system. Their attack surface metric has many desirable properties; for instance, it provides way to talk objectively about security. However, the attack surface metric was designed with a focus on implemented software rather than design artifacts, such as software architecture.

What is missing is way to measure attack surface at design time, when system boundaries are drawn and security-related properties are formed. Measuring attack surface in software architecture could potentially provide architects with precise information on which to base design decisions, compare different designs, and gauge the fitness of a design with regard to one aspect of security.

Using the attack surface metric at design time poses numerous challenges. Effort is required to map attack surface concepts to an architectural level, determine the best point in design to measure attack surface, and select architectural models best suited for measuring attack surface. A design-time attack surface metric would also pave the way to consider architectural transformations to improve a system's security posture.

To address these concerns, we extend Manadhata and Wing's efforts by explicitly adapting the attack surface metric to models of software architecture. This report describes the tools needed to model attack surface in formal models of software architecture as represented in an architectural description language. Additionally, to illustrate the usefulness of the metric, we provide a set of transformations to reduce a system's attack surface to support design decisions. The report closes with a discussion of outstanding questions, and future work, and conclusions.

2 Related Work

Our research draws from a variety of areas, including formal analysis of software architecture, software security metrics, and architectural transformation.

A variety of **formal analysis** methods exist to evaluate system security. For example, Wing's work on attack graphs [19] uses model checking techniques to determine the ways that an attacker can penetrate a system. More generally, Bau and Mitchell [20] present a uniform framework for security modeling and analysis. Among their goals, is to enable consistent analysis that is comparable across different systems. Both of these efforts require information about the system behavior and the operating and threat environments; such information may not be known early in design. Our work differs from these efforts because we focus on evaluating security during design. Specifically, we use **formal models of software architecture**.

With respect to **formal analysis of software architecture**, there are numerous techniques available to evaluate security. For example, Brucker et al. [2] use the Z language to describe access control mechanisms at an architectural level. While Z is an established formal language for modeling software, it is not well suited for software architecture. Specifically, considerable work is required to translate Z constructs to architectural models, reflecting the gap between general purpose formalisms (such as those based on predicate logic and set theory) and the concepts native to an architect.

In contrast, we use an Architectural Description Language (ADL) as the basis for modeling. ADLs are explicitly designed to enable formal modeling at an architectural level. In this context ADLs eliminate the need for translation of general-purpose formal methods to an architectural level because they are designed to directly represent architectural structures.

Other researchers have used ADLs to model software architecture. In numerous works, Garlan et al. [3] [4] [5] show how the Acme ADL can be used to model and analyze architectural properties. These works demonstrate the utility of an ADL for analyzing architectural models, but do not consider security or security-related analyses.

Other efforts have shown that ADL-based approaches can be used to analyze security. Specifically, Kirti et al. [6] demonstrate how security properties can be introduced and analyzed via an architectural style within the AcmeStudio modeling environment. Schmerl et al. [7] introduced extensions to AcmeStudio to run security-related simulations over architectural models. Similarly, Ren and Taylor offer a security-focused ADL [8] complete with constructs that capture security-related properties and structures. In a separate work Ren et al. [9] use xADL to explore how security policies can be enforced by architectural connectors. We build on these efforts by introducing a new form of security-related architectural analysis that focuses on attack surface.

The work of Barnes and Abi-Antoun [10] uses an ADL-based approach to evaluate conformance of between a target security architecture and code. While this work concerns the security analysis of software architecture, it is primarily focused on validating the security architecture of a running system against intended design, rather than a specific property such as attack surface.

In terms of **security metrics** evaluation in software architecture, Buyens et al. [11] show how to detect and rectify violations of the principle of least privilege. Their work demonstrates how to measure a particular security property (least privilege violations) and also proposes transformations to improve security by minimizing those violations. We follow a similar approach, but use attack surface as the primary measure of security.

Additional work by Buyens, De Win, and Joosen [12] explores the impact of security-focused transformations on other quality attributes, such as modifiability. Like us, Buyens, De Win, and Joosen use Manadhata and Wings attack surface metric as the basis for architectural evaluation and transformation. However, their work focuses on evaluating design tradeoffs with non-security quality attributes, whereas we are primarily concerned with analyzing and improving security. That is, the transformations we propose are designed specifically to reduce attack surface; we do not consider the implications of these transformations on non-security properties.

Finally, **software security patterns and transformations** to improve security are considered extensively in Hafiz et al. [13, 14]. Similarly, Bunke and Sohr [15] describe security-related patterns that emerge as a result of the transformations presented in [13, 14]. These transformations provide useful guidance for augmenting a system to improve various security properties. However, those transformations are not presented in a formal sense, are not designed to reduce attack surface, and many are not considered from an architectural perspective. We build on these efforts by considering many of the same or similar transformations in the context of reducing attack surface at design time.

3 Models of Software Architecture

Formal modeling of software architecture is a powerful technique that can be used to evaluate the impact of design decisions against desired quality attributes [4, 5]. For instance, an architectural model can be leveraged to analyze if a design modeled with performance-related properties meets performance requirements [7]. In this capacity, models of software architecture serve as analytical tools for evaluating design decisions.

Software architecture is a multi-dimensional artifact and can be represented using different views to expose different aspects of design [17]. For example, certain types of views emphasize code-oriented structures, while others focus on deployment-oriented structures.

Dynamic aspects of a system are captured using a Component-and-Connector (C&C) type of view. C&C views show the runtime attributes of a system and often include constructs, such as components, connectors, ports, and roles. Components represent the computational elements in a system. Ports represent the interfaces to components. Connectors represent the interaction pathways between components. Roles represent the interfaces of connectors [18].

Attack surface concerns the ways a system interacts with its environment: for instance, sending or receiving data to/from the environment. As these interactions occur while a system is running, the architecture modeling techniques presented in this report are considered from a C&C perspective.

4 The Attack Surface Metric

Historically, attack surface was an informal concept loosely tied to the amount of system functionality that an attacker could access or influence from outside the system [16]. Manadhata and Wing [1] quantify attack surface in terms of the resources used by a system to interact with its external environment. The method they propose for measuring attack surface serves two purposes. First, the attack surface metric numerically characterizes the touchpoints that a system has with its external environment. Second, the metric serves as a prediction system to enable the assessment of security-related risks.

Attack surface is an attractive security metric because it captures an aspect of security that emerges from a system's design, rather than a particular attack or attacker capability. That is, attack surface captures a security-related property of the system independent of operating conditions. This is desirable for a design-time security metric because characteristics of the operating environment or attacker capabilities may not yet be known or well understood.

The attack surface metric, as formulated in [1], quantifies a system's interaction with its external environment along three resource types: entry and exit points, channels, and untrusted data items. Entry and exit points roughly correspond to methods that accept, or process data that originates outside of the system. Channels are the communication mechanisms used to connect a system to its environment, such as network and inter-process communication mechanisms. Finally, untrusted data items are the persistent data stores external to, and used by, a system.

In terms of attack surface, each resource is evaluated to determine the likelihood that an adversary will use that particular resource in an attack. Using the attack surface metric, the attractiveness of a resource to an attacker is encoded as a ratio of the potential damage done by exploiting that resource over the amount of effort needed to access that resource. This is known as the *Damage-Effort Ratio* (DER). This corresponds intuitively to the idea that an attacker will select the resource that offers the most damage potential for the least amount of effort. Thus, large DER values represent more attractive resources.

The effort needed to use a resource is measured in terms of the access rights assigned to that resource. The intuition is that resources that require more rights to access also require a greater effort to use in an attack.

The concept of damage used in the attack surface metric differs from other notions of damage, such as monetary loss. Specifically, in the context of attack surface damage concerns the technical advantage that an attacker gains from exploiting a resource.

The damage potential associated with entry/exit points is evaluated in terms of privileges. Specifically, compromising an entry/exit point allows an attacker to operate with the privileges assigned to that entry/exit point. In terms of the DER ratio for entry/exit points (DER_m), the level of privilege associated with an entry/exit point serves as the numerator, and the level of rights needed to access an entry/exit point serves as the denominator:

(1) $$\frac{privilege}{access\ rights}$$

Channel damage potential is evaluated based on the restrictions placed on the data that a channel can transmit given its protocol. Less restricted protocols offer more advantages to an attacker because they can transmit more types of data, such as executable code. When evaluating the DER for channels, larger *protocol* values indicate less restrictions on data transmitted over that channel. Hence, the DER ratio for channels (DER_c) is evaluated in terms of protocol over access rights:

(2) $$\frac{protocol}{access\ rights}$$

Similarly, damage potential for untrusted data items is evaluated in terms of the restrictions placed on the data they contain. More permissive data types impose fewer restrictions on a data item's contents are considered more advantageous to an attacker. The DER for untrusted data items (DER_i) is evaluated in terms of data type over access rights:

(3) $$\frac{data\ type}{access\ rights}$$

The complete attack surface metric is represented as a triple containing a summation of the DER across the three attack surface dimensions (i.e. entry/exit points, channels, and un-trusted data items). To compute the metric the damage potential and effort associated with each dimension are ordered and assigned numeric values. For instance, for entry/exit points root privileges may be the highest level of privilege, while guest user privileges may be the lowest level of privilege. Similar assignments are applied to channel protocol and data item type properties. The final attack surface metric is expressed as:

(4) $$\left\langle \sum_{m \in M} DER_m(m), \sum_{c \in C} DER_c(c), \sum_{i \in I} DER_i(i) \right\rangle$$

5 Modeling Attack Surface in Software Architecture

To represent and evaluate attack surface at an architectural level, the attack surface dimensions must be mapped to architectural structures. Fortunately, many of the common architecture structures can be easily extended to model the properties and semantics associated with the three attack surface dimensions.

In architectural terms, entry/exit points are analogous to ports, as they serve as the external interface between a component and its environment. Channels can be modeled as a specialized form of an architectural connector that connects components outside of the system with components inside of the system. Finally. untrusted data items can be modeled as the data sources used by the system that reside in the environment, such as files and databases. From an architectural standpoint, untrusted data items are best represented as specialized components that are explicitly designated as being outside of the system boundary.

In addition to relating attack surface dimensions to architectural structures, a capable modeling tool is needed to use the attack surface metric during design. Without an effective modeling tool, using the attack surface metric to identify security concerns and guide security-related design decisions becomes cumbersome, if not impractical. For this work, we selected AcmeStudio [4, 7] as the primary modeling tool[1]. AcmeStudio uses the Acme ADL [3] as the foundation for architectural modeling.

Acme supports the addition of new architectural styles [18] to bundle related concepts and structures, known as Acme "families." We've created a new security-focused Acme family specifically for attack surface. This new family serves as the basis for modeling and analyzing attack surface with AcmeStudio.

With a clear mapping of attack surface concepts to architectural structures and a capable architecture modeling environment, it is possible to leverage attack surface at design time. The following sections describe the formal details of how the attack surface dimensions map to architectural structures in the context of the Acme ADL.

[1]Other architecture modeling tools could have been used to model attack surface with varying degrees of effort. We selected AcmeStudio because it met our modeling needs and we have experience using it for architectural development.

5.1 Modeling Channels

Channels are modeled as a specialized connector type in Acme. Specifically, to represent a channel, we extend Acme's basic connector type to allow the assignment of numeric values to represent protocol (*channelProtocol*) and access rights (*channelAccessRights*). The Acme specification for the channel type is shown below:

```
Connector Type ChannelT = {
  Property channelAccessRights : int;
  Property channelProtocol : int;
}
```

5.2 Modeling Entry/Exit Points

Traditionally architectural ports serve as the interfaces between components. Ports that are designated as entry/exit points must specify a level of privilege (*entryExitPointPrivileges*) and access rights (*entryExitPointAccessRights*).

Specialized Acme ports that include properties for level of privilege and access rights are used to model entry/exit points in the Acme attack surface family. The Acme specification for the entry/exit point type is shown below:

```
Port Type EntryExitPointT = {
  Property entryExitPointPrivileges : int;
  Property entryExitPointAccessRights : int;
}
```

Notably, Manadhata and Wing describe two forms of entry/exit point that contribute to attack surface: direct and indirect. Direct entry/exit points exist on a system's perimeter and access data directly from the environment. Although conceptually well-described in [1], indirect entry/exit points are difficult to represent at an architectural level; thus, they are largely ignored in this work.

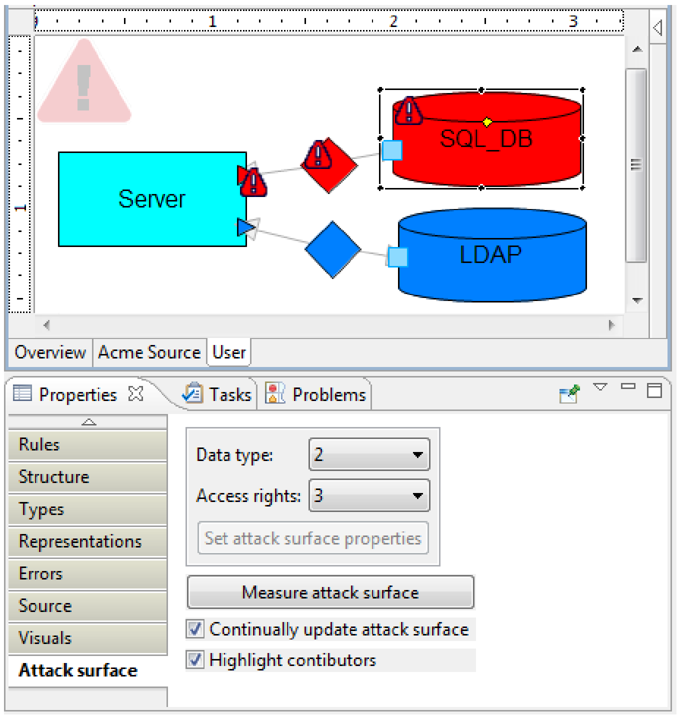
5.3 Modeling Untrusted Data Items

In the Acme attack surface family, untrusted data items are modeled as specialized components with properties to represent data item type (*dataItemType*) and access rights (*dataItemAccessRights*). As with the other dimensions, the attack surface properties are modeled as integers. The Acme source code for a data item type is shown below:

```
Component Type DataItemT = {
  Property dataItemType : int;
  Property dataItemAccessRights : int;
}
```

6 Evaluating Attack Surface

Thus far, we have focused on the formalisms needed to represent attack surface in an architectural model. To make this work practical, we have also created tools to support attack surface analysis. Specifically,



Data type: 2

Access rights: 3

[✓] Continually update attack surface

[✓] Highlight contibutors

Figure 1: Acme attack surface family and plugin for AcmeStudio

accompanying the attack surface family is an extension to AcmeStudio to analyze and manipulate attack surface properties in the form of an AcmeStudio plug-in. The attack surface plug-in provides services to set the attack surface properties and measure the attack surface of an architectural model. Figure 1 shows the interface for the attack surface plug-in, including how the attack surface dimensions appear graphically in AcmeStudio.

Controls to measure attack surface, assign and adjust attack surface-related properties to resources, and identify the resources that contribute the most to a system's attack surface are organized into a control panel that is accessible within a new properties tab in the AcmeStudio interface. The properties tab is shown in the lower half of Figure 1.

When measuring attack surface a step is needed to map actual properties onto integer values. For instance, if root-level privileges are the highest privilege level and guest-level privileges are the lowest privilege level, then root-level privileges should be assigned a higher numeric value than guest-level privileges. The attack surface plugin allows users to set the attack surface-specific properties of architectural elements. For simplicity, the attack surface plug-in restricts the numeric values assigned to properties to a scale of one to 10. The scale assigned to attack surface related properties is subjective and based on the properties of the system under design. We assume that a human will place intermediate values of the scale in a way that accurately reflects the reality of the system. This method of property distribution is consistent with the process Manadhata and Wing [1].

As a design-time security metric, we believe that attack surface should be used to identify possible areas of concern, while architectural designs are malleable and relatively easy to change. With this in mind, we designed the attack surface plug-in to provide features to identify possible areas of concern. In particular, the plug-in allows architects to identify the principal contributors to a model's attack surface across the three attack surface dimensions (entry/exit points, channels, and untrusted data items).

Elements deemed major contributors are the resources within each attack surface dimension that have the largest DER value. For example, the entry/exit point with the largest DER value is considered a largest contributor to attack surface across the entry/exit point dimension. The attack surface plug-in visually identifies major contributors when the *Highlight contributors* checkbox is selected, coloring red the entry/exit point, channel, and untrusted data item that contributes the most to a system's attack surface.

7 Transformations to Reduce Attack Surface

Measuring attack surface at an architectural level provides engineers with information on which to base security-related design decisions. To further support secure system design we propose a set of architectural transformations to reduce attack surface. These transformations should provide designers with concrete strategies known to reduce attack surface; thus, simplifying design efforts.

There are essentially two ways to improve the attack surface metric for a given system: (1) reduce the number of externally available resources and (2) reduce the benefits gained from exploiting a resource. Removing externally available resources lowers the number of things factored into the attack surface. Reducing the attractiveness of using a resource in an attack requires making resources harder to access and exploit, or lessens the benefits gained from exploitation.

With these two guidelines in mind, four transformations are proposed below to guide design in a direction that reduces attack surface. These transformations are *Sanitize Data at Entry/Exit Points*, *Favor Restricted Channels*, *Move Data Items to the Interior*, and *Design to a Single Point of Access*.

7.1 Sanitize Data at Entry/Exit Points

Sanitization mechanisms are a common security-focused design strategy found in a variety of software systems, such as operating systems, web-based applications, and so on. This design strategy places a sanitizing component along a system's perimeter to provide a mechanism to securely move data from a component with low privileges to a component with high privileges and vice versa.

As it concerns attack surface, this transformation requires the architect to insert a component between an entry/exit point and the environment. Ports that previously served as entry/exit points should be moved to the sanitizer and have their privileges reduced by an order of magnitude to reflect the sanitizing function.

This transformation is based on the *Add Perimeter Filter* transformation presented in Hafiz, Adamczyk, and Johnson [13] and Hafiz and Johnson[14] and the *Add Component Restricting Data Flow between System and Environment* transformation presented by Buyens, De Win, and Joosen [12]. Our *Sanitize Data at Entry/Exit Points* transformation differs from the approach taken in these works by explicitly adjusting system properties to reflect the sanitization function.

Sanitizing data at entry/exit points should improve the attack surface by reducing the privileges of elements exposed to the environment. The lone precondition for applying this transformation is that the target component and entry/exit point have been properly identified.

7.2 Favor Restricted Channels

Channels that impose few restrictions on the type of data transmitted provide considerable leeway to an attacker. Intuitively, an attacker has more freedom to send data to a system via a channel that allows raw binary data versus a channel that only accepts structured data, such as ASCII-based text. Limiting the type of data transmitted over a channel can reduce the attack surface of the system by lessening the advantage gained by exploiting that channel. The introduction of a restricted channel will lower the protocol restrictiveness value (and possibly raise the access rights value) assigned to that channel thereby lowering its contribution to a system's attack surface.

No structural changes are needed to complete this transformation; rather, only changes to the protocol and access rights are required. The relevant preconditions for the transformation are the existence of a channel and the applicability of more-restricted protocol. Specifically, the protocol value should be lowered to reflect the more restrictive nature of the new protocol. Reducing the protocol value can be coupled with raising the access rights value to further reduce a channel's contribution.

7.3 Move Data Items to the Interior

Moving data items to the interior of a system shifts untrusted data items away from the system's perimeter so that they are no longer factored into the attack surface metric. This change effectively reduces the number of untrusted data items in use. Furthermore, data items that cannot be moved to the interior of the system should be evaluated to determine if they are necessary and be eliminated if they are not. There are no significant preconditions for this transformation beyond the existence of untrusted data items. Essentially, executing this transformation requires eliminating the existing untrusted data item from the model.

7.4 Design to a Single Point of Access

This transformation consolidates entry/exit points to form a single touch point between a system and its environment, which is a design believed to improve the security of a system. There are two aspects to this transformation; first, the introduction of a gatekeeper component to serve as a unified point of access. This aspect of the transformation concentrates access around a single component. The second aspect of the transformation requires consolidating entry/exit points with identical properties.

During consolidation entry/exit points should be evaluated and eliminated if they are determined unnecessary. Combining entry/exit points that share the same privileges and access rights reduces the number of entry/exit points by reducing the number of externally-facing interfaces in the system. Eliminating unnecessary entry/exit points reduces the number of access points available to an attacker.

This transformation is based on the *Single Access Point* transformation presented in Hafiz, Adamczyk, and Johnson [13] and Hafiz and Johnson[14]. The primary difference in our work is that we explicitly consider creating a single access point in terms of architectural structures as opposed to code-level constructs.

To apply this transformation at least two entry/exit points with the same properties (i.e., privileges and access rights) must be present in the architecture. Once the preconditions are satisfied the transformation can be implemented in two broad steps. First, introduce a gatekeeper component to serve as a centralized point of access to the system. Note that this component may also double as a sanitizer if the *Sanitize Data at Entry/Exit Points* transformation is implemented. The second step entails identifying and combining entry/exit points with the same privileges and access rights.

8 A First Experiment

We conducted a simple experiment to validate our adaptation of the attack surface metric and evaluate the effectiveness of the transformations proposed above. The experiment centered on the example system described Hafiz and Johnson [14]. The design system has the following description:

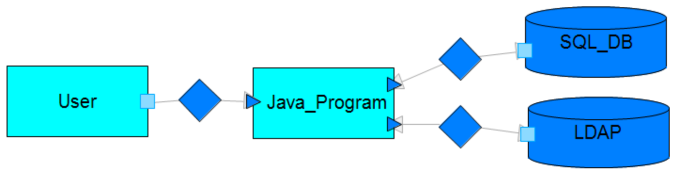


Figure 2: Example architecture modeled in AcmeStudio

> Suppose, an imaginary developer Alice has a Java program that connects to a hostname and a port via a socket. It connects to two hostname/port pairs and uses the data received from one connection to query an SQL database and the data received from another connection to query an LDAP database.

Our goal was to retool the architecture of this system to improve security. The design-time attack surface metric served as the measure of security for the example system. The four proposed architectural transformations were used as the primary mechanism to improve the security of the example system.

First, the architecture was modeled in AcmeStudio using the attack surface family and plugin. The architectural model for this system is shown in Figure 2. The Java_Program component is the Java program described above. This component accepts input from a User and communicates with a SQL database and a LDAP system, which are both represented as un-trusted data items. These data items are connected to the Java_Program via channels running TCP and are accessed via entry/exit points. Finally, the User connects to the Java_Program directly via third channel and entry/exit point.

Attack surface properties are assigned values according to a one-to-10 scale. The minimum and maximum values for each property are assigned the value one and 10 respectively. For example, we assume that root privileges are the largest amount of privileges available and guest privileges are the smallest amount of privileges available; thus, root privileges are assigned the largest value (10) and guest privileges are assigned the smallest value (1). The assignment of numeric values to properties is shown in Table 1. This table serves as the guide for assigning numeric values to attack surface-specific properties for this experiment. Next, individual resources are assigned property values in the following way:

1. The entry/exit point that connects the Java_Program to the User is running with authenticated privileges and authenticated access rights. Thus, this entry/exit point's *entryExitPointPrivileges* and *entryExitPointAccessRights* properties are assigned the values five and five respectively.

2. The channel that connects the Java_Program to the User is running a direct entry protocol with local access rights. Thus, this channel's *channelProtocol* and *channelAccessRights* properties are assigned the values 10 and 10 respectively.

3. The entry/exit point that connects the Java_Program to the SQL_DB is running with root privileges and authenticated access rights. Thus, the *entryExitPointPrivileges* and *entryExitPointAccessRights* properties for this entry/exit point are assigned the values 10 and five respectively.

4. The channel that connects the Java_Program to the SQL_DB is running the TCP protocol with remote access rights. Thus, this channel's *channelProtocol* and *channelAccessRights* properties are assigned the values five and one respectively.

5. The entry/exit point that connects the Java_Program to the LDAP system is running with root privileges and authenticated access rights. Thus, this entry/exit point's *entryExitPointPrivileges* and *entryExitPointAccessRights* properties are assigned the values 10 and one respectively.

6. The channel that connects the Java_Program to the LDAP system is running the TCP protocol with remote access rights. Thus, this channel's *channelProtocol* and *channelAccessRights* properties are assigned the values five and one respectively.

7. The SQL_DB is a data item of type SQL and requires authenticated access rights. Thus, this data item's *dataItemType* and *dataItemAccessRights* properties are assigned the values 10 and five respectively.

8. The LDAP system is a data item of type LDAP and requires authenticated access rights. Thus, this *dataItemType* and *dataItemAccessRights* properties are assigned the values one and five respectively.

With these settings in place the computed attack surface metric for this architecture is $\langle 5.00, 11.00, 2.20 \rangle$. To reduce the attack surface for this system, and thus reduce its exposure to attack, four transformations were applied in the following order:

1. A single point of access was created to consolidate interaction between the Java_Program and the SQL database and LDAP system.

2. The *Sanitize Data at Entry/Exit Points* transformation was applied between the Java_Program and the User. Additionally, the entry/exit point connecting the User to the gatekeeper/sanitizer is now running with guest privileges and unauthenticated access rights. Note that the gatekeeper is dual-purposed to also function as a sanitizer.

3. A *Sanitize Data at Entry/Exit Points* transformation was applied a second time to the connection between the Java_Program and the data items (SQL_DB and LDAP system). Again, the sanitization is performed within the gatekeeper. The consolidated entry/exit point now operates with authenticated privileges and authenticated access rights.

4. The two channels used to connect to the SQL_DB and LDAP system are both restricted by adopting RPC as the transport mechanisms thereby limiting the type of data that can be transmitted.

The redesigned system is shown in Figure 3. Using the property values specified in Table 1, the attack surface metric for the redesigned system is now $\langle 2.00, 2.75, 2.20 \rangle$.

As expected, the transformations applied to the example architecture reduced its attack surface. Notably, each of the transformations will impact different dimensions of the attack surface metric. The *Design to a Single Point of Access* transformation reduced the system's attack surface along the entry/exit point dimension by removing one entry/exit point from the model. The *Sanitize Data at Entry/Exit Points* transformation further reduced the system's attack surface along the entry/exit point dimension by lowering the privileges, and potential damage, of two entry/exit points.

The *Favor Restricted Channels* transformation reduced the attack surface metric along the channel dimension by limiting the type of data transmitted. After this transformation, the channels that connect to the SQL database and LDAP system both use RPC, which is considered a less-permissive protocol than TCP.

Note that none of the transformation applied affected the number of data items in the system or the properties of by those data items. Thus, the attack surface metric remains the same along the data item dimension.

Table 1: Attack surface property assignment

Method	Privilege	Value	Access Rights	Value
	root	10	root	10
	authenticated	5	authenticated	5
	guest	1	unauthenticated	1
Channel	**Protocol**	**Value**	**Access Rights**	**Value**
	direct entry	10	local	10
	TCP	5	remote	1
	RPC	1		
Data Items	**Data Type**	**Value**	**Access Rights**	**Value**
	SQL	10	root	10
	LDAP	1	authenticated	5
			anonymous	1

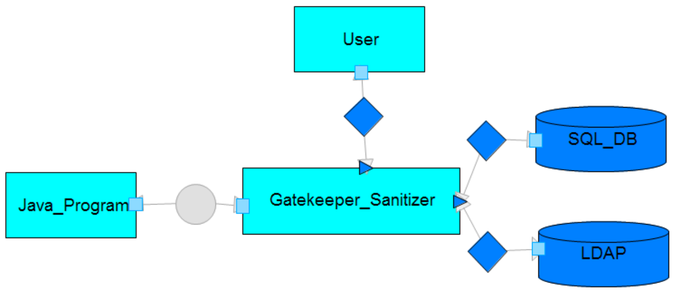


Figure 3: Example architecture after transformation

Notably, we believe that the order that transformations are applied has a significant impact on the resulting system. In particular, different orderings of the *Design to a single point of access* and *Sanitize data at entry/exit points* transformations may result in systems with different attack surfaces. Creating a single point of access will influence all existing entry/exit points, possibly eliminating some of them. Thus, for this experiment we chose to create a single point of access before adding sanitizer(s).

9 Conclusion and Future Work

This report demonstrates how the attack surface metric can be used to evaluate security during architectural design. We have shown how the concepts associated with the attack surface metric map to architectural structures. We have also described a set of tools to enable architects to evaluate attack surface in formal models of software architecture. Finally, we described four architectural transformations to reduce the attack surface of a system under design, and provided an initial validation of our approach through a simple experiment.

The direction of this work continues to evolve and numerous questions and challenges remain. From a mechanical standpoint, the Acme attack surface family needs to be revised to better align with the formal description of the metric. Specifically, better constraints are needed to ensure that models are constructed properly (e.g. channels only connect entry/exit points to data items, every entry/exit point is connected to a channel, etc.). Better constraints would also help to clarify the boundaries between the system and its environment.

For the purpose of modeling attack surface we have only considered direct entry points. It is unclear what, if any, significance indirect entry/exit points will have on the design-time attack surface metric.

The ordering of the four transformations proposed to reduce attack surface requires more consideration to determine if, and how, the attack surface is impacted. Hafiz and Johnson encountered a similar situation in [14] and note that understanding the implications of the order of transformation is critical. The work of Garlan, Barnes, Schmerl, and Celiku in architecture evolution [21] presents one promising avenue of research concerned with ordering architectural transformations.

Finally, we have yet to apply the design-time attack surface metric at scale. In the future, we hope to validate the usefulness of measuring attack surface at design time by applying it to a model of a realistic system.

Ultimately, the question at the heart of this exercise is *does attack surface serve as a good measure of security in software architecture*. Manadhata and Wing have done considerable work to validate attack surface as a security metric. A similar effort is needed to demonstrate that attack surface is a useful measure for software architecture.

References

[1] Manadhata, P. and Wing, J.: An Attack Surface Metric. IEEE Trans. Software Eng. 37, 371–386, 2011.

[2] Brucker A., Rittinger, F. and Wolff, B.: A CVS-Server Security Architecture — Concepts and Formal Analysis. Albert-Ludwigs-Universität Freiburg, 2002.

[3] Garlan, D., Monroe, R. and Wile, D.: Acme: An Architecture Description Interchange Language. in Proc. of CASCON97, 1997.

[4] Schmerl, B. and Garlan, D.: AcmeStudio: Supporting Style-Centered Architecture Development. Proc. of the 26th Intl. Conf. on Software Eng., 2004.

[5] Schmerl, B. and Garlan, D.: Architecture-Driven Modeling and Analysis. Critical Systems and Software. 69, 3–17, 2006.

[6] Garg, K., Garlan, D. and Schmerl, B.: Architecture Based Information Flow Analysis for Software Security. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[7] Schmerl, B., Butler, S. and Garlan, D.: Architecture-based Simulation for Security and Performance. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[8] Ren, J. and Taylor, R.: A Secure Software Architecture Description Language. In Workshop on Software Security Assurance Tools, Techniques, and Metrics, 2005.

[9] Ren, J., Taylor, R., Dourish, P. and Redmiles, D.: Towards an Architectural Treatment of Software Security: A Connector-Centric Approach. ACM SIGSOFT SW Eng. Notes. 30, 1–7, 2005.

[10] Abi-Antoun, M. and Barnes, J.: Analyzing Security Architectures. 25th IEEE/ACM Intl Conf. on Automated Software Eng., 2010.

[11] Buyens, K., De Win, B. and Joosen, W.: Identifying and Resolving Least Privilege Violations in Software Architectures. Availability, Reliability and Security (ARES) 2009.

[12] Buyens, K., Scandariato, R. and Joosen, W.: Measuring the Interplay of Security Principles in Software Architectures. Proc. of the 3rd Intl. Symp. on Empirical Software Engineering and Measurement (ESEM) 2009.

[13] Hafiz, M., Adamczyk, P. and Johnson, R.: A Catalog of Security-Oriented Program Transformations. Technical Report UIUCDCS-R-2009-3031, University of Illinois at Urbana-Champaign, Jan 2009.

[14] Hafiz, M. and Johnson, R. E.: Improving Perimeter Security with Security-Oriented Program Transformations. Proc. of the 2009 ICSE Workshop on SW Eng. for Secure Systems, 61–67, 2009.

[15] Bunke, M. and Sohr, K.: An Architecture-Centric Approach to Detecting Security Patterns in Software. Intl. Symp. on Eng. Secure Software and Systems, 156–166, 2011.

[16] Howard, M.: Fending Off Future Attacks by Reducing Attack Surface. February 2003. http://msdn.microsoft.com/en-us/library/ms972812

[17] Garlan, D., Bachmann, F., Ivers, J., Stafford, J., Bass, L., Clements, P. and Merson, P.: Documenting Software Architectures: Views and Beyond, 2nd ed. Addison-Wesley Professional 2010.

[18] Shaw, M. and Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall 1996.

[19] Wing, M. J: Attack Graph Generation and Analysis. Proc. of ASIACCS 2006.

[20] Bau, J. and Mitchell, John C:. Security Modeling and Analysis. IEEE Security and Privacy 9/3, May 2011.

[21] Garlan, D., Barnes, J., Schmerl, B. and Celiku, O.: Evolution Styles: Foundations and Tool Support for Software Architecture Evolution. Proc. of the Joint Working IEEE/IFIP Conference on Software Architecture 2009 and European Conference on Software Architecture 2009, Cambridge, UK, 14-17 September 2009.