Towards a Specification Prototype for Hierarchy-Driven Attack Patterns

Joshua J. Pauli
College of Business and Information Systems
Dakota State University
Madison, SD, 57042, USA
josh.pauli@dsu.edu

Patrick H. Engebretson
College of Business and Information Systems
Dakota State University
Madison, SD, 57042, USA
pat.engebretson@dsu.edu

Abstract

We propose the characteristics of a software tool that leverages specifying attack pattern details in understandable hierarchies. These hierarchies are currently manually populated from the vast CAPEC dictionary which consume an excessive amount of human resources and are wrought with the possibility of user error. Such a software tool will not only automate the population of these attack pattern hierarchies, but also provide system prerequisite information and suggested mitigation strategies for the system under design. The combination of system prerequisites, possible attack patterns, and necessary mitigation strategies gives system designers and developers a checklist-like artifact to consider as development moves from the design phase to the implementation phase.

Keywords: Attack Trees, Attack Patterns, Refinement, Hierarchy.

1. Introduction

Within the past five years, the fields of network, computer, and software security has begun to shift its focus away from perimeter defensive models, such as border routers, firewalls, and intrusion detection systems, to more proactive defensive models [2]. Our prototype tool takes the proactive mindset and attempts to automate the populating of attack trees for the benefit of security-centric design decisions. Because security strategies vary greatly after design decisions are made, our prototype includes built-in mappings for many well established design configurations as well as the ability to add custom design configurations. The mappings within our tool are derived from the CAPEC dictionary of attack pattern information. Attack Patterns are relatively new, having been introduced within the past decade [3]. It is the goal of this paper to leverage this vast repository.

2. Tool Prototype Characteristics

Our prototype tool is based on the CAPEC dictionary to ensure that the input, processing, and output are derived from an accepted source [1]. Each attack pattern contains all the relevant information to populate our prototype tool's data store for later retrieval. This information is then organized and extracted to be useful for the design and implementation teams as the development process moves forward. The goal of the prototype tool is to provide a checklist-like artifact to ensure security is considered as part of design and long before the implementation phase.

The input for our prototype tool is the prerequisites from the CAPEC dictionary; these are design decisions that are made for the system under design. Our prototype accepts prerequisites such as hardware selections, operating system, server configurations, and programming language used for initial input. As a user selects a system prerequisite, related attack patterns and necessary mitigation strategies are populated to be reviewed.

The processing of our prototype tool is comprised of extracting, organizing, and editing data mappings that are made up of system prerequisites, related attack patterns, and necessary mitigation strategies. The data mappings are stored in a database and are leveraged by extracting and presenting them at a system-specific level. The user of the prototype tool can edit any mapping between prerequisite and attack pattern or between attack pattern and mitigation strategy to best satisfy their system's requirements and configurations.

The output of our prototype tool is twofold. First, there is organized output as part of the normal usage of the tool that displays the system-specific prerequisites, related attack patterns, and necessary mitigation strategies in a hierarchical format. This output is available graphically, which is most applicable for small systems. Second, these mappings can also be viewed in tabular format and managerial reports, which is most applicable for larger systems.

Our prototype tool is driven by stored data mappings of system prerequisite, related attack patterns, and necessary mitigation strategies for a specific system. Data mappings can be added, edited, and/or deleted from the system's specifications. Because every system has its own set of data mappings, the master set of attack pattern-driven data mappings are left unchanged. This allows for the implementation teams to truly customize the system's security components to best match the system. A system's information is stored in a separate set of tables that are derived directly from the master data mappings. We call this a "system profile". The prototype tool extracts and lists the master data mappings per the selected system prerequisites. They are listed in a tree structure as introduced in Figure 1.

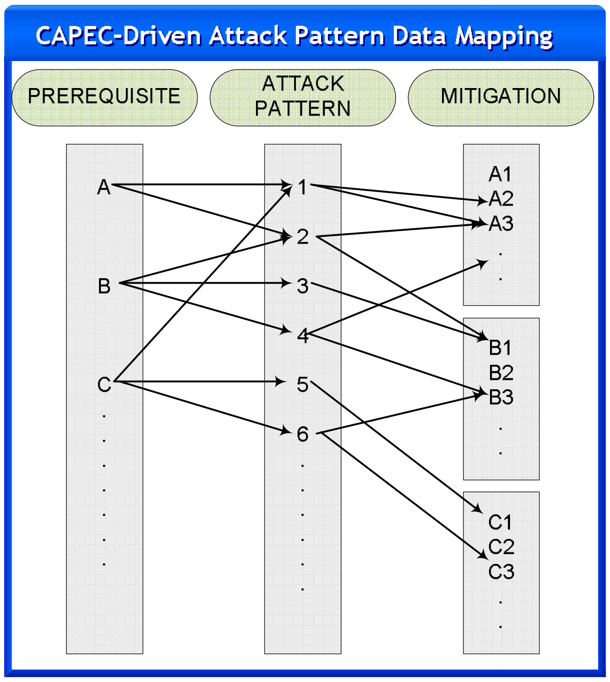

Figure 1. Abstract Tree Structure for Displaying System Prerequisite-Driven Data Mappings

Each node is expandable to show all of the related "downstream" entities. The prerequisite is expandable to show all of the attack patterns that threaten it, while the attack pattern is expandable to show all the necessary mitigation strategies. One issue with systems of any size is that the tree structure quickly becomes unreadable because of the number of links. Because of this issue, the mappings for a specific system may be best viewed in a tabular format as introduced in Table 1 with a partially populated hierarchy for "Apache Webserver" prerequisite where applicable attack patterns are "Server Side Include Injection" and "HTTP Request Smuggling". Mitigations are presented in column three.

Table 1. Partial Hierarchy for the "Apache Webserver" System Prerequisite

PREREQ	ATTACK	MITIGATION
Apache Webserver	Server Side Include (SSI) Injection	Set the OPTIONS IncludesNOEXEC in the global access.conf file or local .htaccess (Apache) file to deny SSI execution in directories that do not need them
		All user controllable input must be appropriately sanitized before use in the application
		Server Side Includes must be enabled only if there is a strong business reason to do so.
	HTTP Request Smuggling	Careful analysis of the entities must occur during system design prior to deployment. If there are known differences in the way the entities parse HTTP requests, the choice of entities needs consideration.
		Employ an application firewall

These mappings can be edited at the discretion of the development team to best reflect the exact implementation of the system.

References

[1] S. Barnum and S. Amit. *Further Information on Attack Patterns*. Build Security In Setting a Higher Standard for Software Assurance 2006.

[2] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*. 2004: Pearson Higher Education.

[3] A. Moore, R.J. Ellison, and R.C. Linger, *Attack Modeling for Information Security and Survivability*. 2001: Carnegie Mellon University, Software Engineering Institute.