Utilizing Attack Graphs to Measure the Efficacy of Security Frameworks Across Multiple Applications

Francis J. Manning III
Nova Southeastern University
franmann@nova.edu

Frank J. Mitropoulos
Nova Southeastern University
mitrof@nova.edu

Abstract

One of the primary challenges when developing or implementing a security framework for any particular environment is determining the efficacy of the implementation. Does the implementation address all of the potential vulnerabilities in the environment, or are there still unaddressed issues? Further, if there is a choice between two frameworks, what objective measure can be used to compare the frameworks? To address these questions, we propose utilizing a technique of attack graph analysis to map the attack surface of the environment and identify the most likely avenues of attack. We show that with this technique we can quantify the baseline state of an application and compare that to the attack surface after implementation of a security framework, while simultaneously allowing for comparison between frameworks in the same environment or a single framework across multiple applications.

1. Introduction

Information security has continued to increase in importance over the past several years as the proper operation of most organizations depends on smooth and secure access to a variety of data. Correspondingly, anyone watching a nightly news broadcast knows the difficulty in maintaining the security of the data – after all, an adversary only needs to succeed once while the information security practitioner must succeed every single time. This creates a significant problem for those individuals tasked with protecting data within a given environment, how can we measure the effect of the tools and methods that we have emplaced to protect our data?

As research shows that the time between a vulnerability being discovered and the appearance of an exploit for that vulnerability continues to diminish [1], the necessity of being able to analyze an environment and determine vulnerabilities and effective countermeasures increases in importance. Unfortunately this involves a significant amount of work for the information security practitioner as commercially available vulnerability-rating services lack the ability to incorporate the nuances of individual environments into their rating structure [2]. Additionally, even efforts to develop common metrics to enable the comparison of the security of different systems relies on a significant number of environmental metrics that are subject to interpretation [3], reducing the effectiveness of such metrics at the present time.

To further exacerbate circumstances, the granularity of information necessary to compare the effectiveness of different security frameworks implemented against existing software is not something that is currently available. It is due to this need that we have expanded upon previous work with attack graphs [4], and examined how they can be used to provide a quantitative measure [5], where all too often only qualitative measures are available.

2. Implementation Techniques

For the purpose of this research experiment, we were working with four applications – one, a purpose driven application that was designed and constructed in order to serve as a model for the implementation of a security framework, and three FLOSS applications that were similar in purpose. The three FLOSS applications were all text editors: Simple Text Editor [6]; Text Editor [7]; and Step [8]. The purpose driven application was also a text editor, although it additionally had a client-server component to allow data access from a central server, as well as other clients in a peer-to-peer fashion.

The framework that was utilized for this research was an instance of the Identification, Authentication and Authorization framework [5], and can be seen in Figure 1. The framework instance was designed utilizing a combination of the SQUARE method for
developing security requirements [9], and a spiral development process with the purpose-driven application set as the target that the framework was required to secure. During the development process, an attack graph was utilized to measure the attack surface of the purpose-driven application to develop a baseline for the application, and was then re-generated after each successive step in the implementation of the framework. This technique allowed for immediate feedback as to the efficacy of the framework in addressing a particular set of security requirements geared towards reducing the attack surface of the application.

![Figure 1: Identification, authentication, and authorization framework (IAA)](image)

### 2.1 Baseline

In order to generate the attack graphs that would be utilized during the course of the research, a set of standards needed to be determined that would be utilized to develop the edge-weighting factors for the attack graph; a key characteristic that allows for the measurement and comparison of paths within a graph or between separate graphs. The edge-weighting factors that were utilized are shown in Table 1.

#### Table 1: Edge weighting factors

<table>
<thead>
<tr>
<th>Description</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of Difficulty</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Level of Consequence</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Cost or Resources Required</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Modifier: Duplicate Exploit</td>
<td>$n$: where $n$ is the number of paths in which the exploit appears on the graph.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that the weights that were utilized to develop the path costs in the graph are inversely proportional to the suggested difficulties, as when they are multiplied against one another to form the basis of the edge weight, an edge with a higher score rises to the top of the list of likely exploited vulnerabilities within an environment. With the edge-weighting factors in place, it becomes necessary to look at the un-weighted attack graph (Figure 2) for the purpose driven application in order to stage the next dataset that will be necessary for analysis, a coordinated set of exploits with attached edge weights.

The coordinated set of weights are developed by walking the unique, non-cyclic paths from the base from the initial states depicted in the graph, in this case: External agent; Valid user; Authorized IT agent. These paths are then used to determine the number of times a given exploit occurs in the set of paths through the graph which is used as a multiplier when developing the cost of each exploit path. The result of the enumeration of these paths and the development of the overall costs can be seen in Table 2.

#### Table 2: Exploits with assigned weights

<table>
<thead>
<tr>
<th>Description of Exploit</th>
<th>Difficulty</th>
<th>Consequence</th>
<th>Cost/Resources</th>
<th>Modifier</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackmail/Coercion</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>23</td>
<td>184</td>
</tr>
<tr>
<td>Social Engineering</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>42</td>
<td>10752</td>
</tr>
<tr>
<td>Break In</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>17</td>
<td>272</td>
</tr>
<tr>
<td>Boot from media and impersonate user</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>22</td>
<td>1408</td>
</tr>
<tr>
<td>Steal Server/Drive</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>13</td>
<td>104</td>
</tr>
<tr>
<td>Copy data drive</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>13</td>
<td>416</td>
</tr>
<tr>
<td>Connect directly to application client or server daemon and impersonate protocol</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>38</td>
<td>19456</td>
</tr>
<tr>
<td>Direct enabler no exploit required</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>101</td>
<td>0</td>
</tr>
</tbody>
</table>

Once the weights have been assigned, overall summary values are then developed for the total attack surface of the application based on the number of unique, non-cyclic paths in the attack graph. In the case of the purpose-driven test application, that resulted in an overall baseline path cost of 1,237,504 split between internal (289,872) and external (947,632) attack vectors. The path costs were generated across 77 total.
unique, non-cyclic paths split between internal (23) and external (54) attack vectors.

During the course of the framework implementation, the path cost as well as the number of paths that remained in the attack graph were reduced when positively verified mitigations provided by the framework eliminated attack vectors in the application.

**Attack Vectors:**
- Internal - 23
- External - 54

**Note:** Attack vectors are determined by traversal of the paths in the graph, and the removal of any cycles from the list. The remaining paths are possible attack vectors. The internal vectors are those that can be realized by an entity internal to the organization utilizing the information system. External vectors are those avenues of attack available to individuals without immediate access to the information system in question.

These steps were repeated continually until, upon completion of the framework integration, the attack surface had been effectively reduced to a value of 0, indicating that the application was secured.

Upon completion of the integration of the IAA Framework with the purpose-driven application, we were able to show that the attack surface of the purpose-driven application had been effectively reduced to zero, and because of the spiral development process that was used to develop the instance of the framework, matching which framework feature addressed a particular vulnerability was easily accomplished. However, all this could really prove was that a framework could be built that would address the issues in an application that was purposefully designed to support the development of the framework. Because of this, the next step was to implement the framework against several FLOSS applications to show that the framework was usable with pre-existing software.

### 2.2 Pre-existing applications

The implementation of the framework against the three pre-existing applications began with the development of individual adapter code to specifically match the framework to each of the three applications. This was necessary in order to fully integrate the framework into the application without neglecting any potential problem areas within the application. Consequently, the development of the adapter code was handled in much the same way as the initial framework development, with a similar spiral-development process.

By replicating the same development process utilized during the initial development of the
framework, similar advantages were realized, particularly during testing. With the coding of the adapter segmented to address like modules from the framework simultaneously, the number of areas that needed to be inspected when an unexpected result was observed during testing was minimized. Additionally, this process left us with a baseline attack graph for each application, as well as incremental graphs for after each stage of the development process. The data from each of these implementations was then aggregated and utilized in the analysis of the results.

3. Analysis

One of the most interesting results that was observed during the data analysis process, was that the baseline attack graph for the three pre-existing applications, was a sub-graph of the baseline attack graph for the purpose-driven application. This can be seen in Figure 3. There was some speculation as to why the baseline graphs of the three pre-existing applications would be the same, and after an examination of the code involved, it was determined that they baseline graphs were the same because the applications performed the same basic functions while using similar API calls, the applications were all written in the Java programming language, in order to perform those basic functions.

Once it was confirmed that the baseline for the pre-existing applications was actually a sub-graph of the baseline for the purpose-driven application, the analysis was very straight-forward. An examination of both the pre-implementation and post-implementation graphs for each of the applications showed similar results with the purpose-driven application path costs going to zero for each of the internal and external paths, as did the path costs for each of the pre-existing applications. The sole difference that can be seen in the graphs of the pre and post implementation path costs for the applications is the number of paths present, with the
pre-existing applications having fewer paths than the purpose-driven application.

While it was the case in this research that the pre-existing applications had fewer paths, and a smaller attack surface than the purpose-driven application, that relationship is not expected to hold true in a general canvassing of other applications. A comparison of the source code between the pre-existing applications and the purpose-driven application indicated that the pre-existing applications also used a subset of the Java API compared to the purpose-driven application.

4. Summary

Through the body of this research experiment, we have shown that the attack graph methodology was suitable to comparing the attack surfaces of several different applications, each written by different individuals, with different programming styles. Additionally, the methodology was also capable of comparing the attack surfaces of the applications after the implementation of a security framework with the intent of providing quantitative data as to the state of vulnerabilities in the application environment.

By utilizing the attack graph methodology to evaluate the attack surface of applications before and after the implementation of a security framework, we have also come to the conclusion that the technique is more expandable than originally thought. Because the baseline attack graph will not change if the security framework was switched with a different security framework, the end state values of the total path cost measurement of the attack surface can be utilized to provide a quantitative comparison of the effect of two completely independent security frameworks on the same environment, a measurement that has not previously been available.

Finally, it was observed during the detailed analysis of this research that the attack surface of an application is much more dependent on the API calls performed by the application than was initially expected. Specifically, the breadth of API utilization appears to be proportional to the attack surface of the application. While there was certainly not enough data generated during this research to come to a firm conclusion on this topic, it was shown in this circumstance that utilizing a subset of the API led to a baseline attack graph that was a sub-graph of the more expansive purpose-driven application.

5. Further Research

There were several areas of research that were identified for further study in this area. The first area of further research involves the relationship between the utilization of the API and the attack graph itself. Specifically, is there a direct relationship between the individual API calls, and either vertices or edges within the attack graph? If such a relationship could be shown conclusively, would it then be possible to construct an authoritative attack graph generator provided only the source code for a given application? Such an authoritative source of security information would be invaluable to security practitioners in their efforts to secure the information systems environment from unauthorized data access, in addition to allowing them to evaluate any commercial security solution that might be deployed as mitigation to existing or potential vulnerabilities.

The second area of study that was identified as possible direction for future work is the comparison of security frameworks with the attack graph methodology. While this would appear at first glance to be straightforward we believe that there are some subtleties that would provide tremendous benefits to the research community with future work. Specifically, utilizing this method to compare completely different security frameworks or paradigms across different areas of software or systems implementation. Possible questions that could be addressed include: Is there a relationship between a specific technique that could be employed by a security framework and a certain set of vulnerabilities identified in one or more sub-graphs? If such a relationship exists, would it be possible to identify the specialty relationships that may exist in a given set of applications and construct or generate a customized security framework to mitigate those vulnerabilities? Could any of this be automated with a toolset that could scan a set of applications and produce the ideal security framework for a given application set?

The third area of future research involves expanding the analysis of the attack graph methodology on the security framework implementation to involve pre-existing applications with different purposes and more expansive utilization of the JAVA APIs. Continuing down this path could increase the level of confidence in utilizing this technique to measure the efficacy of security frameworks with disparate applications. Additionally, the complexity of the comparisons could be increased to include interactions between applications or systems of applications.
6. Conclusions

With this research effort, we have shown that the usefulness of attack graphs can be extended to create a quantitative measure that allows for the close comparison of the efficacy of security frameworks across different applications. We believe that this will become a useful tool available to the researcher that will allow the generation of objective evidence as to which type of technique is best suited to address a specific type or class of vulnerability in an application. Providing an empirical measure of how an application functions with a given security framework should also serve to enable further research in related areas of security frameworks, and software assurance, as the relationships can be more uniformly examined and compared.

7. References


