Using Attack Graphs to Understand Vulnerabilities

Ben Thorne

December 12, 2018

Abstract:

Attack graphs are graphical exploit representations in the context of computer security. This paper focuses on the attack graph model that represents on vulnerability dependencies, and does so in the context of network security. By laying out potential exploits using an attack graph, the dependencies for secure systems are clearer, which is helpful for system administrators. In a time of crisis, this information could be critical for allowing engineers to prioritize when to fix certain vulnerabilities. This paper details an attack graph of a relatively common system, and defines an algorithm that decides which vulnerabilities should be prioritized based on simulated attacks. In addition to the specific examples mentioned, details on how to extend the attack-graph model to a different system are provided, and the code to do so is publically available. The results of this experiment shed light on how systems can best protect themselves before, during, and after various attacks. It also reveals how easy working with attack graphs can be, since the algorithm is sufficient but not complex. Additionally, the benefits of constructing attack graphs are revealed in the context of learning more about how a network's vulnerabilities are connected – and how to best defend them.

1. Introduction

Quick visits to https://cwe.mitre.org/ and https://cve.mitre.org/ reveal the unsettling truth about modern computer systems: there are a lot of weaknesses and vulnerabilities. In order to maintain a secure system, a deep understanding of the intricacies of the relationships between potential exploits is required. While standards such as Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE) are useful tools for understanding the insecurities of systems, the shear amount of information they present is overwhelming. This is particularly true in networked systems. Considering the myriad insecure protocols and complexity of such systems, the web of dependencies can become extremely confusing. The attack graph model addresses this concern by providing a comprehensive representation of a system's weak points, which can help an administrator before, during, and after an attack.

Most research involving attack graphs use simulations to predict attacker behavior in order to harden security measures in specific areas of the network. The findings of Sudip Saha, Mahantesh Halappanavar, and Anil Vullikanti in their paper *Identifying Vulnerabilities and Hardening Attack Graphs for Networked Systems* deal mostly with the tradeoffs between the complexity of networks and how difficult networks are to harden. This is excellent work, but attack graphs could also be useful in a post-attack situation. In a time of crisis, engineers are faced with the task to patch exploits, but in the face of a complicated attack, it is unclear

what the most effective approach should be to resolve the problem(s) as quickly as possible. In this paper, I represent an attack graph using a Python class and networkx, a nifty Python module used by network scientists to make graphs. I also define an algorithm to help an association's recovery efforts after an attack by providing a list of exploited vulnerabilities in order of priority. This algorithm will be demonstrated on a small, common network generated using the aforementioned attack graph representation. This code is publically available at https://github.com/bbthorne/attack_graphs and can be edited to represent an arbitrary system that can be modeled by an attack graph. While simple, the code sample is helpful for understanding how attack graphs work and how they might be helpful.

2. To the Community

The use of attack graphs to represent security vulnerabilities is a relatively understudied topic in the realm of computing theory and software engineering. Considering the relative simplicity of attack graphs, they could be useful tools for all software engineers to become familiar with when securing their systems. As technology becomes more varied and complex, being familiar with the exploits and the dependencies among them becomes increasingly essential to maintainable design. I seek to prove the usefulness of these models in the context of networked systems, as well as provide some tools to help others to model/run dynamics on their systems.

3. Attack Graphs

3.1 Definition

In order to grasp the capacity at which attack graphs can be a useful concept for security, it is important to know their general structure. In computing and mathematics theory, a graph *G* is defined as a tuple *(V,E)* where *V* is a set of nodes and *E* is a set of edges between nodes. This set representation allows for reasoning at a high level without the necessity of visualizing an entire network. In an attack graph, the nodes represent vulnerabilities in a system, and they have a 'priority' value, which could adhere to any useful criteria. The edges between nodes represent the dependencies that are required for a vulnerability to be exploited.

In addition to the having components of standard graphs, attack graphs are directed and acyclic. For a graph to be directed, its edges imply a one-way relationship. This allows an attack graph to accurately represent the dependencies from specific vulnerabilities to others in the network. For a graph to be acyclic, the paths between nodes along edges must never repeat nodes, which is also convenient for the attack graph model – the dependencies are more explicit this way. Attack graphs can be implemented with cycles, as in Wang, Islam, Long, Singhal, and Jajodia's paper *An Attack Graph-Based Probabilistic Security Metric*, but they define algorithms to iterate through cycles, which gets complicated. While plenty of other graphs are directed and acyclic, attack graphs have one more important attribute. In addition to a priority, each node with an in-degree larger than 2 (the amount of directed edges for which the node is the destination is larger than 2) is assigned a Boolean operator

AND or OR that further defines its dependencies. For example, a node with an AND operator requires all vulnerabilities with edges to it to be exploited in order for it to be exploited.
In the supplementary code, the properties of an attack graph are represented by the `AttackGraph` class. It uses networkx, a popular network science Python module, to encode the graph, and includes other fields and methods – but more on that later.

3.2 An Example System

The definition of attack graphs alone is difficult to visualize, so here's an example system. It is taken from Sudip Saha, Mahantesh Halappanavar, and Anil Vullikanti's paper *Identifying Vulnerabilities and Hardening Attack Graphs for Networked Systems*, and though small and a bit outdated, it is a nice sample to work with. Pictured in Figure 1, it models a basic data center that uses a firewall to protect a data server, and allows user interaction with the data via an FTP server. In Figure 2, the vulnerabilities of the system are represented as an attack graph. Figure 3 serves as a legend for Figure 2, with additional information about the vulnerabilities.

Figure 1: Real-life Representation

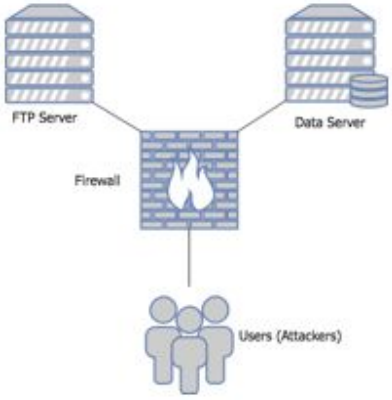

Figure 2: Vulnerabilities of the System

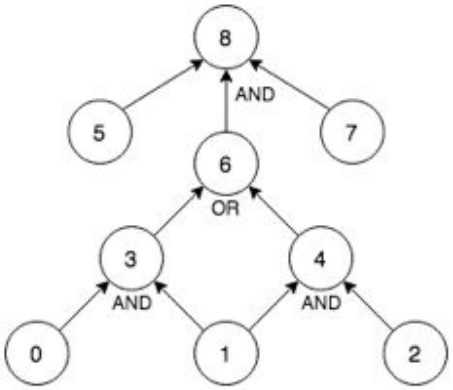

Figure 3: A legend for Figure 2

node	vulnerability	description
0	running ftpd on ftp server	ftpd has buffer overflow vulnerabilities, which cause services to crash if exploited
1	ftp server accessible via internet	users can access the ftp server but not the data server, thanks to the firewall
2	running Pragma Fortress SSH on ftp server	Pragma Fortress SSH can be exploited to cause a buffer overflow by accepting arbitrarily long messages
3	buffer overflow in ftp daemon on ftp server (CVE-2001-0755)	buffer overflow causes denial of service and/or arbitrary code execution on server
4	buffer overflow in SSH on ftp server (CVE-2006-2421)	buffer overflow allows for arbitrary code execution on server
5	LICQ 1.0.2 running on data server	LICQ is an online messaging service that allows access to data in the server
6	root access on ftp server	user can execute any command they'd like on the server, which has access to the data server
7	data server access from ftp server	data server is only accessible via the ftp server, thanks to the firewall
8	LICQ exploit that allows arbitrary code injection (CVE-2001-0439)	attacker can execute whatever they want on the data server, compromising privacy

Notice that some vulnerabilities included in the attack graph are necessary for the service that the system provides. For example, the ftp server must have access to the data server in order for the data server to be useful at all. Similarly, the system must be accessible by the internet for users to interact with it. Despite the fact that these nodes will always be 'exploited' in the context of an attack, it is important to include them due to the other, more severe, vulnerabilities that might depend on them.

3.3 Representing the System with the `AttackGraph` Class

The `AttackGraph` Python class included with the supplementary code represents attack graphs and provides methods to attack certain vulnerabilities in the network. The class contains two properties pertaining to the structure of attack graphs: `graph` (a networkx directed graph) and `dependencies` (a dictionary that maps nodes to Boolean functions). It also contains a property `exploited` for use in attack simulations, and an optional property `info`, a dictionary that maps nodes to descriptions.

The method `generate_graph` contains the code necessary to build the attack graph, adding nodes and directed edges to the graph and assigning dependencies. In this representation, nodes are represented by integers 0-n, where n is the size of the system minus 1. Since AND and OR are the two operations modeled in attack graphs, the built-in Python higher-order functions `all` and `any` (resp.) work perfectly. Nodes without dependencies are initialized with `None`. At the moment, this code builds the example in 3.2, but includes instructions on how to adjust it to represent any system. If a large system must be represented, check out Jha and Wing's paper *Two Formal Analyses of Attack Graphs*, which describe methodologies for generating complex attack graphs. Otherwise, a manual configuration will do.

4. Attacking Vulnerabilities in the Model

4.1 Simulating Attacks on an `AttackGraph`

To attack an instance of `AttackGraph`, the method `attack` can be called given a list of exploited nodes. This method simulates an attack's spread through the network by abiding by the dependency rules of the attack graph. After this method is called, the `exploited` property denotes which vulnerabilities have been exploited after the complete attack.

4.2 An Example Attack

Let's consider a situation in which an attacker is able to cause a buffer overflow in the ftp daemon through the ftpd service used by the ftp server while the system is functioning at full capacity. In this case, the initially exploited nodes are 0, 1, 5, and 7, which represent the necessary services of the system and the attack. Figures 4 and 5 depict exploited graphs before and after dependencies are applied. In both visualizations, exploited nodes are indicated in red:

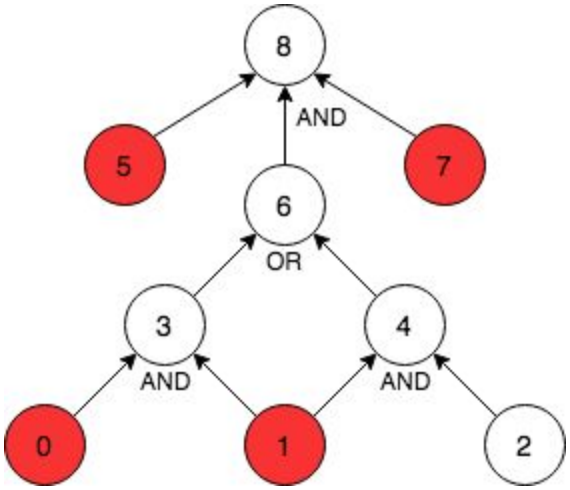

Figure 4: Initial Attack

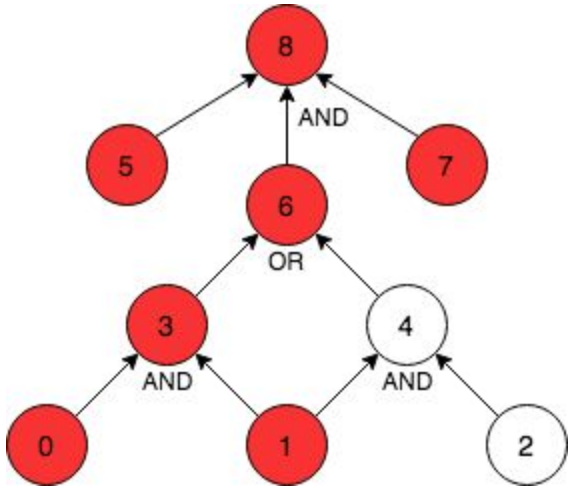

Figure 5: Resulting Damage

In this case, the attacker is able to exploit a string of vulnerabilities that allow them to execute arbitrary code on the data server. While the details of these vulnerabilities are important to know, the attack graph representation allows for them to be abstracted in order to get a picture of the entire system. The information encoded in an attack graph post-attack is more than sufficient to reason about in a variety of contexts. For example, in Noel, Jajodia, Wang, and Singhal's paper *Measuring Security Risks of Networks Using Attack Graphs*, they use exploited attack graphs to determine a metric to judge the overall security of systems. While this is a very complicated process, simpler conclusions can also be made using attack graphs.

5. Planning a Course of Action After an Attack

In a time of crisis, there could be limited evidence of the damage done to a system, making it difficult to decide a course of action. With the aid of an attack graph, though, this decision could be better-informed. The supplementary code includes a function `resolve` which, when given an exploited `AttackGraph`, returns a list of the exploited nodes in order of when they should be addressed.

The idea is simple: compute the sum of of the node priority values of the exploited successors of each exploited node, and sort in descending order. With this, the vulnerabilities that cause the most damage later in the graph are identified as the first to be addressed. When applied to the example from before, the result of `resolve` is `[0,1,3,5,6,7,8]`. This is based on a system in which each vulnerability has an equal priority - changing the priority values would slightly change results One of the strengths of this method is that smaller exploits that lead to big exploits are prioritized first, and fixing them generally has a larger impact on resolving the greater problem. The output of `resolve` includes services that are not themselves "problems", but are included in the output as an important source of exploits within the system.

The results of the algorithm on a more complex system with a larger-scale attack would provide much more enlightening results than this small example, especially in those with multiple high-priority exploits. Luckily, the process runs in polynomial time ($O(n^4)$ with the opportunity for $O(n^2)$ with a tweak in the `AttackGraph` implementation), so it is scalable within a reasonable limit.

6. Conclusion

Attack graphs are not only reliable representations of security vulnerabilities in systems, but also extremely easy to work with. Considering its potential impact on the overall security of systems, modeling systems using an attack graph should become more common in the world of software development. By keeping a standing reserve of the relationship between security exploits in a system, engineers have a familiar representation of complex concepts, which can be helpful before, during, and after an attack. The `AttackGraph` class provided is meant to be edited to reflect an arbitrary system, and includes a clean code-basis reflecting the necessities of attack graphs. The function `resolve` is useful at face-value, but also further demonstrates how simple it is to interact with attack graphs and still get some meaningful results. Regardless of implementation, modeling a system using an attack graph is valuable for security awareness within an association - it's not very difficult, and it could make a huge difference.

7. References

1. Common Vulnerabilities Exposures (CVE) and Common Weakness Enumeration (CWE), MITRE Corporation, 1999-2018
   http://cve.mitre.org/index.html, https://cwe.mitre.org/

2. Sudip Saha, Mahantesh Halappanavar, Anil Vullikanti. *Identifying Vulnerabilities and Hardening Attack Graphs for Networked Systems,* Virgina Tech, 2014
   http://staff.vbi.vt.edu/ssaha/papers/attackgraph_dag.pdf

3. Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia, *An Attack Graph-Based Probabilistic Security Metric,* Concordia University, 2008
   https://users.encs.concordia.ca/~wang/papers/dbsec08.pdf

4. Jha, S., Shayner, O., Wing, J., *Two Formal Analyses of Attack Graphs*, University of Wisconsin, Madison and Carnegie Mellon University, 2002
   http://pages.cs.wisc.edu/~jha/jha-papers/security/CSFW_2002_1.pdf

5. Steven Noel, Shushil Jajodia, Lingyu Wang, Anoop Singhal, *Measuring Security Risk of Networks Using Attack Graphs*, George Mason University, 2010
   https://ist.gmu.edu/~csis/noel/pubs/2010_IJNGC.pdf