Passive Network Model Generation For Attack Graphs With Signature Based Automatic Network Protocol Detection

George Louthan

May 6, 2011

Abstract

Attack graphs provide insights into the security of an entire network, especially in highly heterogenous settings with a wide variety of systems and attack surfaces whose interactions are difficult to intuit. However, their use requires an appropriate formal specification of the network. The creation of such a model by hand is difficult, error-prone, and time consuming. This paper presents a tool that utilizes deep packet inspection to generate network model specifications appropriate for use in the style of attack graphs employed at the University of Tulsa. The generated model is far from complete, but it does provide a useful starting point from which modeling can proceed.

1 Introduction

Attack graphs are useful tools for analyzing network security, specifically the interactions of network objects and vulnerabilities. A graph theoretical modeling framework in which nodes are states and edges are transitions caused by an adversary, the attack graph permits analysis of compound exposures across a network by chaining attacks across an entire changing system, rather than by analyzing each component or vulnerability in isolation.

However, in order to produce an attack graph, an appropriate model of the system or network must be provided. By hand, the process of creating such a specification can be error-prone and time consuming. To alleviate this problem, this paper presents an automated passive tool that utilizes deep packet inspection to build a network model of hosts and topologies reflecting the connections and protocols whose traffic has been identified, regardless of TCP port number used. Though the resultant model is neither perfect nor comprehensive, it provides an excellent starting point for a modeler.

Although the specification of exploit patterns is also necessary for the successful employment of attack graphs, their procurement and generation is not within the scope of this paper.

The remainder of this paper is organized as follows. Section 2 provides some overview level background into the version of the attack graph formalism used for this work, the network model required by that formalism, and a basic lexicon of terms used for property and topology names, as well as an introduction to the deep packet inspection tool used. Section 3 describes the specifics of the network model generation methodology, and Section 4 draws conclusions and suggests future work.

2 Background

2.1 Attack Graphs

Given a set of so-called *exploit patterns* and a network model that specifies the initial state of the network, attack graphs use graph theory to explore the security properties of a network by closing its state space under its realizable exploits [2] [5] [6].

Due to space constraints, a thorough grounding in the type of attack graphs employed is impossible; however, the reader is referred to a 2005 literature review by Lippmann and Ingols [3] for a thorough survey of the field. Although an attack graph requires both a network model and a set of exploit patterns, this section describes only the network model component, as it is the only part relevant to this paper.

2.1.1 Network Model

A network model specification consists of a list of *assets*, which are the security principals at play in the network (possibly including hosts, users, data, the attacker, and anything else that needs to be treated as a first class object in the model) and an initial *fact base*, which consists of two types of facts: *qualities*, which are name-value pairs representing asset properties, and *topologies*, which are named one-way relationships between two assets. This section provides a prose description of the network model specification language, which may be difficult to understand without an example to reference; the reader is encouraged to follow along with Fig. 1, a meaningless but valid and illustrative example of the specification language.

In our specification language, a network model is begun with the phrase `network model` followed by the `=` symbol. The first component of the network model is the list of assets, which is begun with the word `assets` followed by a colon. Asset names are semicolon separated. Following the asset list is the initial fact base. It is begun with the word `facts` followed by a colon.

Quality facts begin with the word `quality`, followed by a colon, followed by the name of the asset whose quality is being defined, then a comma, then the name of the quality (an atom), then an assignment operator, then the value of the quality.

Topology facts begin with the word `topology`, followed by a colon, followed by the name of the source asset, followed by either the one-way topology symbol `->` or the two-way topology symbol `<->` (though the two-way topology is actually processed into two one-way topology

```
network model =
     assets :
          asset_1;
          asset_2;
          asset_3;

     facts :
          quality:asset_1, quality_1=value_1;
          quality:asset_2, quality_1=value_2;
          quality:asset_3, quality_1=value_2;
          topology:asset_1->asset_2, topology_1;
          topology:asset_2<->asset_3, topology_2;
.
```

Figure 1: Example attack graph network model specification

facts), followed by the name of the destination asset, followed by a comma, followed by the name of the topology.

Each fact is terminated with a semicolon, and the network model is concluded with a period.

2.1.2 Term Lexicon

Because the names used in qualities and topologies are so freeform, a consistent dictionary of terms is required to facilitate modeling. This section presents three different types of terms: *access topologies*, *connectivity topologies*, and *status qualities*.

Access topologies are based upon the types of access rights used in the United States' National Vulnerability Database (NVD): user (referring to user level access to a system), root (referring to administrative level access to a system), and other (referring to some kind of application-specific but nevertheless privileged type of system access) [1]. These access types are represented with a topology whose source is the asset who *has* access, whose destination is the asset to whom the source has access, and whose name is `access_`*`type`*. In the case where type is `other`, it is followed by another underscore and the name of the application whose access rights are in question.

Connectivity topologies are also due to NVD specification [1]. They may be `connected_local`, meaning a direct, local, physical connection; `connected_adjacent`, meaning that they are on the same network segment or collision domain; or `connected_network_`*`protocol`*, meaning that they are connected across the Internet using the protocol whose lower case abbreviation (e.g. `ssh` for Secure Shell or `http` for web) replaces *`protocol`*.

Finally, the status quality refers to whether the host is available; availability is intended to be used as a precondition (possibly using a preprocessor) for every exploit, and unavailability is intended as a post-condition of denial of service attacks. The status quality has the name `status`, and its value may be the token `up` or `down`.

2.2 SAND

Signature based automatic network protocol detection (SAND) is the name of a tool and process due to Louthan, McMillan, Johnson, and Hale that uses deep packet inspection (DPI) to passively identify the protocol employed by network traffic regardless of port number [4].

Implemented as a library called pysand, SAND provides the ability to sniff from the wire or from stored files, and report to the driver program when a new network stream is detected, when a stream's protocol is identified, and when a stream is closed.

3 Methodology

The automated network model generation tool provides the following features. Upon the detection of a new stream, it adds the server and client to the network model and gives them the `up` status quality. When a stream is identified, it gives the relevant assets a topology with the appropriate `connected_network_`*`protocol`* name. It also provides optional automated DNS lookup for asset names, manual translation from IP addresses to asset names using a user-provided *mapfile*, and also provides sane defaults for the automated mapping of IP addresses to valid asset names obeying the standard lexicon provided previously.

The procedure for generating a network model specification begins when the first stream is detected. These streams' components are added to an intermediate internal asset list as IP addresses only, and an intermediate quality fact representation with the quality name `status` and value `up` is also added to a separate data structure, which holds all facts about the system and ensures the uniqueness of its members.

Once a stream is identified, a topology fact is inserted into the intermediate fact base; the client is the source, and the server is the destination. The topology is named `connected_network_`*`protocol`*, where *`protocol`* is determined by the protocol abbreviation used in the SAND identifier (by convention, the standard abbreviation).

Upon completion of execution, the network model is built by applying minimal postprocessing to the intermediate asset and fact list. First, the mapfile (if any) is consulted to map IP addresses to friendlier asset names. Then, if the mapfile specifies that DNS is to be used, each IP address is queried against the local DNS server to attempt to obtain a hostname. If neither of these yield an alternative to the IP address, it is converted to a standard format in which each octet is zero-padded in decimal to be three digits long, and the periods are replaced by semicolons. This mapping is applied to the asset list and the assets in the intermediate fact base. Finally, the network model specification is generated.

An example mapping file is provided in Fig. 2. With a simple input and a mapping file, an example of the output produced is given in Fig. 3; without a mapping file, the output is given in Fig. 4.

```
# Example pysand-ag mapfile

[assets]
george_desktop = 129.244.244.150
ssh_client = 10.0.0.1
ssh_server = 10.0.0.2

[conf]
nslookup = no # Don't try to look up hostnames.
```

Figure 2: Example mapfile

```
network model =
    assets :
        ssh_client;
        ssh_server;
    facts :
        quality:ssh_client,status=up;
        quality:ssh_server,status=up;
        topology:ssh_client->ssh_server,
                 connected_network_ssh;
.
```

Figure 3: Example generated model using a mapfile

```
network model =
    assets :
        010_000_000_001;
        010_000_000_002;
    facts :
        quality:010_000_000_001,status=up;
        quality:010_000_000_002,status=up;
        topology:010_000_000_001->010_000_000_002,
                 connected_network_ssh;
.
```

Figure 4: Example generated model without using a mapfile

4 Conclusions and Future Work

This paper presents a tool that utilizes passive network sensors and deep packet inspection to generate network model specifications appropriate for use in the style of attack graphs developed at the University of Tulsa. The generated model is not complete, but it does provide a starting point from which modeling can proceed. Among its most significant weaknesses are a lack of inclusion of even basic platform and version information (which SAND can sometimes provide), and its naïveté in regards to connection topologies.

Future work along these lines should focus upon extracting as much information as possible from captured packets in order to identify software versions or operating systems. This is largely a problem of integration, as tools exist that are capable of performing this analysis already.

Also, integration with the Common Platform Enumeration (CPE) maintained by MITRE would allow even more standardization of the lexicon of terms. This direction could possibly include some minor modifications to the underlying attack graph model to facilitate the inclusion of CPE style version specifications. Finally, no effort is made to intuit whether a `connected_adjacent` or `connected_network` topology should be used. Further research is needed to determine exactly the significance of the differences between these two connection topology types, and possibly to enable the automated passive detection of which topology type is more appropriate.

References

[1] National vulnerability database version 2.2, 2011. Web page. Retrieved April 13, 2011.

[2] C. Campbell, J. Dawkins, B. Pollet, K. Fitch, J. Hale, and M. Papa. On Modeling Computer Networks for Vulnerability Analysis. *DBSec*, pages 233–244, 2002.

[3] R.P. Lippmann, K.W. Ingols, and MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB. *An annotated review of past papers on attack graphs.* Massachusetts Institute of Technology, Lincoln Laboratory, 2005.

[4] G. Louthan, C. McMillan, C. Johnson, and J. Hale. Toward Robust and Extensible Automatic Protocol Identification. *Proc. of ICOMP 09*, pages 104–108, 2009.

[5] C. Phillips and L.P. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[6] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. 2002.