An Integrated Network Scanning Tool for Attack Graph Construction

Feng Cheng, Sebastian Roschke, and Christoph Meinel

Hasso Plattner Institute (HPI), University of Potsdam, 14482, Potsdam, Germany
{feng.cheng,sebastian.roschke,christoph.meinel}@hpi.uni-potsdam.de

Abstract. Scanning is essential for gathering information about the actual state of computer systems or networks. Therefore, it is always taken as the first step of potential attacks against targets. In certain cases, scanning itself is categorized as an attack. Scanning can on the other hand be used for the right purposes, for example, checking the system configurations, verifying firewall rules, proofing security polices, as well as monitoring the large scale network environment. From this point of view, scanning is an effective method for system or network management, security measurement and auditing. To visualize, analyze, and finally evaluate the data gathered by scanners, Attack Graph plays an important role. High quality information about the target system or network is the prerequisite for constructing the attack graph. However, different implementations of scanners have different capabilities and always result in different kinds of outputs. These outputs are usually heterogeneous and not machine-readable, which makes the further analysis a challenging task. In this paper, we examine common types of scanners and demonstrate how to combine multiple types of scanners. The results of all the involved scanners are integrated into a well-designed and consistent data structure, which can not only be well interpreted by human security specialists but also be directly fed into an attack graph construction tool.

1 Introduction

Scanning is always the first step towards successful exploits of the target system [1], [2]. Normally, hackers are not able to know anything about the targets without scanning. Gathering information about the adversary is the unique start point. Most complex attacks are practically impossible without continuous scanning on the final target. On the other hand, the scanning techniques have also been adopted as an important approach for security experts or network administrators to check the weaknesses of target systems or networks [3], [4]. Modern IT infrastructures become more and more sophisticated. Failures due to both technical and human factors, might take place in the processes of design, implementation, deployment, configuration, and execution. Carrying out black-box mode tests is always required for detecting such failures, especially the security vulnerabilities. Scanning is, in most cases, an effective tool for such tests [5]. Along with the increased requirements from both attackers and defenders, a large number of scanning techniques have emerged in recent years, e.g., *nmap*[1], *amap*[2], *netcat*[3], *P0f*[4], *XProbe*[5], *X-Scan*[6], etc. Different types of scanning techniques have different competencies and therefore result in different kinds of implementations. The outputs of those scanners are usually heterogeneous and not machine-readable. Combining multiple types of scanning techniques to get comprehensive but easy to interpret scanning results is expected. Attack Graph offers a formal method to represent the ways in which an attacker can exploit vulnerabilities and break into a system [6], [7]. To construct an attack graph, the accurate information about the target system or network is required. Scanning techniques play important roles in this context. How to integrate scanning tools into the automatic attack graph construction platform is a challenging task for creating high quality attack graphs.

In this paper, we propose an integrated network scanning tool by combining multiple types of scanners. The results of all the involved scanners are integrated into a well-designed and consistent data model, which can not only be well interpreted by human security specialists but also be directly fed and matched with the vulnerability representation within the attack graph tools [8].

This paper is organized as follows. Section 2 introduces some related works, i.e., the workflow of the Attack Graph construction and some popular scanning techniques. Section 3 proposes our integrated scanner. An practical scanning example is shown as proof-of-concept in Section 4. We conclude the paper in Section 5.

2 Related Work

2.1 Attack Graphs

Attack Graphs have been proposed for years as a formal way to simplify the modeling of complex attacking scenarios [9]. An attack graph describes not only one possible attack, but also many potential ways for an attacker to reach a goal. The workflow of an attack graph construction tool consists of three independent phases: Information Gathering, Attack Graph Construction, as well as Visualization and Analysis. In the information gathering phase, all necessary information to construct attack graphs is collected and unified, such as information on network structure, connected hosts, and running services. In the attack graph construction phase, a graph is computed based on the gathered system information and existing vulnerability descriptions. Finally, the attack graph is processed in the visualization and analysis phase. Attack graphs always require a certain set of input information. For one, a database of existing vulnerabilities has to be available, as without it, it would not be possible to identify or evaluate the effects of host-specific weaknesses. Also, the network structure must be

[1] `http://nmap.org/`
[2] `http://freeworld.thc.org/thc-amap/`
[3] `http://netcat.sourceforge.net/`
[4] `http://lcamtuf.coredump.cx/p0f.shtml`
[5] `http://sourceforge.net/projects/xprobe/`
[6] `http://xfocus.net/`

known beforehand. It is necessary to identify which hosts can be reached by the attacker. Often, a network scanning and host-based vulnerability analysis are performed before the attack graph is constructed [8].

2.2 Review of Scanning Techniques

We can classify the scanning techniques into the following categories:

Passive. No packages sent at all.
Discovery. Packages are only sent to find hosts and detect the connectivity.
Port Scan. Systematic testing of individual ports or ranges. Response packages might be examined for software banners.
Probing. Discovered services are checked for versions and vulnerabilities.
Exploiting. Vulnerabilities are actively used or verified.
Other. It is active but cannot be classified as any value above.

Most of emerged scanning techniques are implemented based on this classification. In practice, it includes:

Topology Scanner is used to investigate the structure of networks, which encompasses the detection of hosts residing in a network, the network mask, and the gateways. For advanced topology scanners, multiple networks are scanned and then the results can be put together into a complete network diagram for evaluating their interconnections. The participants in a local network can be obtained by broadcasting *ping* or *ARP*-request packets. For remote hosts, ordinary *ping* over a large address range can be performed. Popular implementations include *traceroute*[7], *Nessus*[8], etc.
Port Scanner aims at finding open and filtered ports on a specified target host. There have been implemented a variety of such scan methods, e.g., *Nmap*, *Nessus*, *netcat*, etc. Most of them are targeted for TCP ports. The easiest and also the most obvious is a *connect* scan. Stealthier methods are *syn* or *ack* scans. UDP scans are used for UDP-based ports.
Fingerprint Scanner is to find the name and version of the software or operating system running on the target. The primary method to obtain product information from a service is via banner grabbing. Advanced tools examine the specifics of sent packets and the behavior the network stack. Known examples are *amap*, *P0f*, *XProbe*, etc.
Vulnerability Scanner targets at detecting vulnerabilities of services or the operating systems running on a host. Vulnerabilities are usually found by actually trying to exploit a set of possible vulnerabilities. Less aggressive tools infer possible vulnerabilities from the products determined by fingerprinting. Existing vulnerability scanners include *Nessus*, *X-Scan*, *Nmap*, etc.
Incremental Scanner usually scans the surrounding network topology by first performing a scan from any of the previously mentioned categories and gain

[7] http://www.traceroute.org/
[8] http://www.nessus.org/

access to vulnerable hosts with exploits. Having access to further hosts, already found networks can be scanned from another viewpoint or completely new networks may become accessible. Doing this incrementally, it sometimes enables the mapping of DMZs (demilitarized zones) between a public and private network or even internal private network. Typical methods to accomplish incremental scanning is to exploit a vulnerability on a host and inject scanning code into the vulnerable service. Some common approaches are *syscall proxying*[1] and *SNAPP*[10], *Core Impact*[1], etc.

3 An Integrated Network Scanning Platform

Each scanner delivers a dedicated set of information about the target host or network. To obtain a rather complete picture of a target for more accurate post processing, e.g., constructing attack graphs, it requires the combination of results from multiple scanners. However, the handling of multiple scanners is not easy due to the following two reasons:

Configuration. Along with the advanced usability of the modern scanner, the required configuration effort becomes more and more complex. For example, both *Nmap* and *Nessus* provide an overwhelming number of configuration options. Consequently, it takes users much more effort to learn how to implement a proper and efficient scanning operation.

Output Interpretation. Each scanner outputs its results in a certain format. Unfortunately, this format is not unified. A user needs to familiarize with each format individually. Furthermore, some detailed scan results are only written in a report file which might be even harder to read.

To achieve an abstraction from all the sub-scanners, following challenges need to be considered for the implementation of the integrated scanner platform.

1. The integrated sub-scanners should be transparent to the user, which means that the user does not need to know how to use the sub-scanners and even does not actually recognize their involvements.
2. The platform must be able to control the actions of all sub-scanners. As different scanners significantly differ in the functionalities, the configuration can be quite different as well. The challenge is to find a common set of configuration options.
3. The scan results of network scanners are not unified in any form. They might be available as XML files, graphically on a GUI or text on the command line output. In order to provide a consistent scan result, output of different forms must be converted into one common result format.
4. As many sub-scanners are used within the integrated platform, it is highly possible that a certain type of scan result is already available from another scanner. In this case, the duplicate information might exist. We need indicate the same result or, in the worst case, contradicting information.

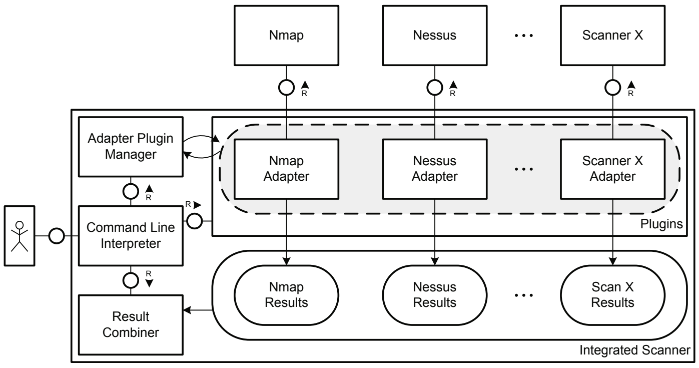


Fig. 1. Architecture of the *Integrated Network Scanning Platform*

3.1 Basic Architecture

Figure 1 shows a rough overview of the architecture used for our proposed *Integrated Network Scanning Platform*. Although the attached scanners do not belong to the platform, they constitute the foundation for all the supported scanning functionalities. Currently, we have focused on online scanning with the popular scanners *Nmap* and *Nessus*. The design of the platform is highly flexible and extensible. It consists of several components. The upper part encompasses multiple plug-ins communicating with the integrated scanners. They basically act as adapters between one scanner and the other components of the platform. These adapters are designed as plug-ins, so that other scanners can be added easily. A plug-in has to accomplish two major tasks to enable seamless integration of scanners into the platform:

1. Bind to an interface of the scanning tool. After binding, the plug-in provides an abstract control interface to the scanner usable by the platform. One part of this interface provision is the translation of the platform's abstract scan options into specific scan options interpretable by the scanner. After an initiated scan has been successfully finished, the plug-in has the task to obtain the scan result from the scanner.
2. Convert the specific scan result obtained from the scanner into a scan result with a unified format, so that the platform only needs to handle one abstract scan result format.

When the scan results have been provided by the scanner adapters, the *result combiner* can start its work by combining all single results into one consistent overall result. The management of the adapter plug-ins is performed by a *plug-in manager* component. It allows to execute plug-ins concurrently as well as pass abstract scan parameters to all adapters. The bridge between the previously mentioned components and the user of the platform is provided by the *command-line interpreter*. It is currently deployed on the host of the scanning user.

3.2 Finding a Common Data Model

As previously mentioned, a major challenge for creating such an integrated scanner is to find a common data model for containing the results from different scanners. Such a model should be easy to understand as well as easy to interpret by computers. For being fed to the later attack graph construction tools, the data model should fit to the formal representation of the vulnerability information [8]. *Nmap* offers such a data model for containing its own scanning result. However, it is strictly tailored to the capabilities of *Nmap*'s port scanner and is hard to comprise other types of scanning results. The vulnerability scanner, *Nessus* also provides a data format for representing the revealed information [11], but it focuses more on the analysis and predictions of the vulnerabilities. Here we propose a custom XML file format as shown in Figure 2.

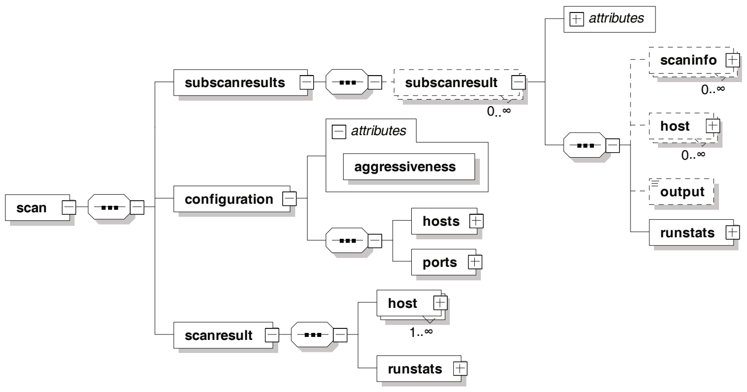


Fig. 2. Top Level Structure of the Data Model

This data model has been divided into three top level branches which cover the different aspects of an integrated scan. One branch contains the combined scan result from all scanners. Another branch contains an abstract configuration used for the integrated scan. In order to determine the origin of a certain piece of information in the combined scan result, the last branch keeps a copy of all individual scan results produced by the scanners. These scan results will be called *sub-scans* in the following. Both the combined result and sub-scan results have a similar format. This format only focuses on one scan result and is very close to *Nmap*'s single result format. A host can be regarded as central element in this format. Each host then has a set of open ports and running services. Beyond that, meta information about the scan can be stored. The sub-scan branch additionally contains detailed information about the scan tool having produced the sub-scan result. This encompasses, for example, tool-specific parameters used as configuration and the original output of the tool. The original output can later be used to obtain information being lost during the conversion to the common data model. The configuration branch stores an abstract view on the parameters being passed to the scanners.

3.3 Parsing the Scanner Outputs

The adapter is the key component to convert the specific output from the scanners to sub-scans of our common data model. For *Nmap*, this task is rather straightforward as our sub-scan branch is very similar to *Nmap*'s data model. Only some smaller changes needed to be implemented. One example is the standardization and unification of information specified in *Nmap*'s output. Therefore, time and date information needed to be converted to one common date format and product names were unified by means of the *Common Platform Enumeration* (CPE)[12]. Due to generalization of *Nmap*'s format, we had to omit some information or had to move some specific information into a more general context. As an example for this generalization, the device type for a host had to be inferred from the device types previously specified in the services of hosts. In the case of *Nessus*, the conversion turned out to be more complex. The problem with its XML-based *.nessus.v1* format is plain text which is hard to interpret by a program. In fact, each security issue reported by *Nessus* is inserted into just one XML **data** tag. Thus, we need to make the output interpretable by an extensive use of regular expressions. A new output format called *.nessus.v2* which addresses this issue of only providing textual data has been introduced by *Tenable* in late 2009. Another issue which makes it more complicated to interpret the *Nessus*' is the organization of *Nessus*' own plug-ins. Each aspect of a scan is handled by a certain plug-in. For example, there is a plug-in to list the open ports, another plug-in provides the host name of the target host, and yet another plug-in checks if a specific vulnerability is available on the target. As a consequence, the information has to be collected from multiple plug-in outputs which do not even have a standardized format. Therefore, it is nearly impossible to include all the results from *Nessus* into our data model. Even if it is possible to evaluate the output of all current plug-ins, new plug-ins would result in an incomplete processing sooner or later.

3.4 Merging Redundant Information

Although the sub-scan provides information about each single scan, a combined scan result is desirable for clarity. Redundant information being produced multiple times must be merged. Such information can be of two types, i.e., matching or contradicting. Matching information can be easily merged by keeping only one instance. We decided to handle contradicting information by plausibility checks and a combination of voting and scoring mechanisms. The different approaches are listed as follows.

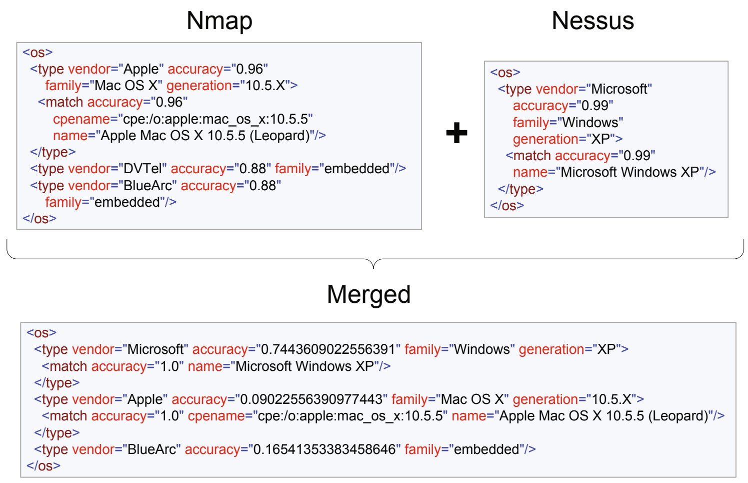
Voting. A piece of information is considered as right only if it has been reported by the majority of scanners. This mechanism only makes sense if there are more than two integrated scanners.

Scoring. A scanner will be assigned scores corresponding to the reliability of the information it produces.

Plausibility. For some types of information, it can be inferred logically which information is more likely the right one.

As shown in Figure 3, the information about the detected operating systems is an example which might be reported differently among scanners. In many cases, even one scanner alone delivers multiple possible operating systems. Fortunately, these scanners also provide a scoring for the most probable entries. Among scanners, we first use a voting and then perform a scoring on ambiguous results. Here, Nessus' results are weighted higher than Nmap's results, as for fingerprint scanning, Nessus is more reliable. Additionally, in the merged result, the accuracy values have a sum of 1, in contrast to the relative accuracy in the individual results.

The example in 4 shows how to remove redundant information about port states using plausibility checks. If multiple states are reported, we choose the entry which first appears in the order *open*, *closed*, and *filtered*. A port with state *open* or *closed* sends packets to the scanner whereas filtered ports do not send any packet. So a filtered port might be the result of a lost packet.

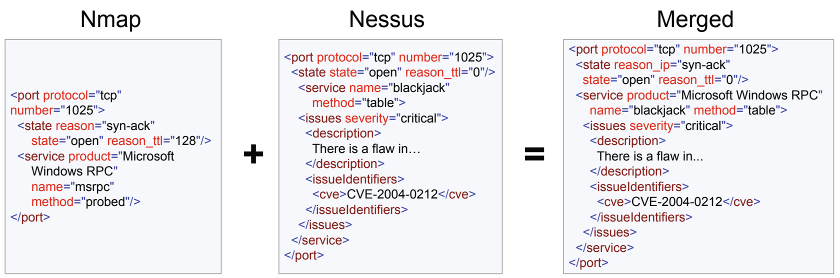


```
<os>
  <type vendor="Apple" accuracy="0.96"
      family="Mac OS X" generation="10.5.X">
    <match accuracy="0.96"
      cpename="cpe:/o:apple:mac_os_x:10.5.5"
      name="Apple Mac OS X 10.5.5 (Leopard)"/>
  </type>
  <type vendor="DVTel" accuracy="0.88" family="embedded"/>
  <type vendor="BlueArc" accuracy="0.88"
      family="embedded"/>
</os>
```

```
<os>
  <type vendor="Microsoft"
      accuracy="0.99"
      family="Windows"
      generation="XP">
    <match accuracy="0.99"
      name="Microsoft Windows XP"/>
  </type>
</os>
```

```
<os>
  <type vendor="Microsoft" accuracy="0.7443609022556391" family="Windows" generation="XP">
    <match accuracy="1.0" name="Microsoft Windows XP"/>
  </type>
  <type vendor="Apple" accuracy="0.09022556390977443" family="Mac OS X" generation="10.5.X">
    <match accuracy="1.0" cpename="cpe:/o:apple:mac_os_x:10.5.5" name="Apple Mac OS X 10.5.5 (Leopard)"/>
  </type>
  <type vendor="BlueArc" accuracy="0.16541353383458646" family="embedded"/>
</os>
```

Fig. 3. Merging operating system information from Nessus and Nmap into one consistent overview of the target's operating system



```
<port protocol="tcp"
number="1025">
  <state reason="syn-ack"
      state="open" reason_ttl="128"/>
  <service product="Microsoft
      Windows RPC"
      name="msrpc"
      method="probed"/>
</port>
```

```
<port protocol="tcp" number="1025">
  <state state="open" reason_ttl="0"/>
    <service name="blackjack"
        method="table">
    <issues severity="critical">
      <description>
        There is a flaw in…
      </description>
      <issueIdentifiers>
        <cve>CVE-2004-0212</cve>
      </issueIdentifiers>
    </issues>
  </service>
</port>
```

```
<port protocol="tcp" number="1025">
  <state reason_ip="syn-ack"
    state="open" reason_ttl="0"/>
  <service product="Microsoft Windows RPC"
      name="blackjack" method="table">
    <issues severity="critical">
      <description>
        There is a flaw in...
      </description>
      <issueIdentifiers>
        <cve>CVE-2004-0212</cve>
      </issueIdentifiers>
    </issues>
  </service>
</port>
```

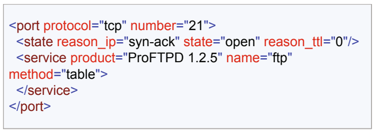
Fig. 4. Merging port information from Nessus and Nmap into one port information

4 Scanning in the Practice

To show how this integrated scanner works in the practice, we set up a highly vulnerable virtual machine and perform an integrated scanning against it. We have selected a very outdated Ubuntu distribution, namely *Ubuntu 6.06 Dapper Drake*. It has a 2.6.15 kernel, which bears a high security risk. Additionally, we have installed a multitude of seriously vulnerable network services on the virtual machine, including *ProFTPd* 1.2.5, *Apache HTTP Server* 2.0.55, *Bind* 9.3.2, *Samba*3.0.22, *Courier IMAP* (imapd) 3.0.8, and *MySQL Server* 5.0.22. It is obvious that our integrated scanner can find many open ports and even more vulnerabilities of the installed products. For the practical scan, we have tested two different configurations, i.e., *probing* and *exploiting*. The *probing* mode is used to create static attack graph while *exploiting* mode can gather information required by dynamic attack graph construction.

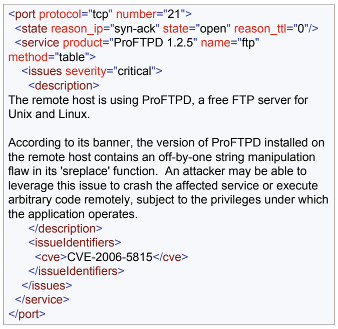
With the *probing* configuration, we could find all open ports affiliated with most of the installed products, see Figure 5(a). The operating system could be determined as well. However, the scan only reportrd a *Samba* vulnerability. The reason for this sparse result is the fact that with probing, vulnerabilities are only inferred from examined product versions.

We could gain a better result with the *exploiting* configuration as shown in the Figure 5(b). It delivered about 30 vulnerabilities of which the most were also reported in vulnerability databases. A significant drawback of the *exploiting* configuration is its long runtime which is specifically caused by the *Nessus* sub-scan. It took nearly half an hour to finish.



```
<port protocol="tcp" number="21">
  <state reason_ip="syn-ack" state="open" reason_ttl="0"/>
  <service product="ProFTPD 1.2.5" name="ftp"
method="table">
  </service>
</port>
```

(a) *Probing* Mode



```
<port protocol="tcp" number="21">
  <state reason_ip="syn-ack" state="open" reason_ttl="0"/>
  <service product="ProFTPD 1.2.5" name="ftp"
method="table">
    <issues severity="critical">
      <description>
The remote host is using ProFTPD, a free FTP server for
Unix and Linux.

According to its banner, the version of ProFTPD installed on
the remote host contains an off-by-one string manipulation
flaw in its 'sreplace' function.  An attacker may be able to
leverage this issue to crash the affected service or execute
arbitrary code remotely, subject to the privileges under which
the application operates.
      </description>
      <issueIdentifiers>
        <cve>CVE-2006-5815</cve>
      </issueIdentifiers>
    </issues>
  </service>
</port>
```

(b) *Exploiting* Mode

Fig. 5. Information about the exposed ProFTPd service in the scanning results

5 Conclusion

Scanning focuses on more or less serious information disclosure and vulnerabilities about the target. It can be accomplished with a variety of scanning tools today. All of them require an individual way of interaction and provide scanning results in different output formats. As a consequence, it is hard for a user to get along with the multiple scanner tools. We have designed and implemented a prototype of the integrated scanning tool allowing to access scan results from multiple scanner tools with just one user interface in a unified output format. As the integrated scanner is configurable for different levels of scan aggressiveness, it is suitable for different types of attack graph construction tools.

Acknowledgement

The authors would like to thank David Jäger and Arvid Heise for their support on implementation of the proposed tool.

References

1. Caceres, M.: Syscall Proxying - Simulating Remote Execution. Technical report, Core Security Technologies (2002)
2. Moore, H., Valsmith: Tactical Exploitation, Version 1.0.0. Technical report, Metasploit LLC (2007)
3. Bhatia, A., Lam, B., et al.: Automated Network Security Audit Tool (ANSAT). Technical report, University of Colorado at Boulder (2006)
4. Bishop, M.: About Penetration Testing. IEEE Security & Privacy 5, 84–87 (2007)
5. Siamwalla, R., Sharma, R., Keshav, S.: Discovering Internet Topology. Technical report, Cornell University (1998)
6. Dawkins, J., Clark, K., Manes, G., Papa, M.: A Framework for Unified Network Security Management: Identifying and Tracking Security Threats on Converged Networks. Journal of Network and Systems Management 13(3), 253–267 (2005)
7. Schneier, B.: Attack Trees - Modeling Security Threats. Dr. Dobb's Journal 21, 21–29 (1999)
8. Cheng, F., Roschke, S., Schuppenies, R., Meinel, C.: Remodeling Vulnerability Information. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 324–336. Springer, Heidelberg (2010)
9. Jajodia, S., Noel, S., O'Berry, B.: Topological Analysis of Network Attack Vulnerability. In: Vipin Kumar, J.S., Lazarevic, A. (eds.) Managing Cyber Threats: Issues, Approaches, and Challenges. Massive Computing, vol. 5, pp. 247–266. Springer, Heidelberg (2005)
10. Cheng, F., Wolter, C., Meinel, C.: A Simple, Smart and Extensible Framework for Network Security Measurement. In: Pei, D., Yung, M., Lin, D., Wu, C. (eds.) Inscrypt 2007. LNCS, vol. 4990, pp. 517–531. Springer, Heidelberg (2008)
11. Arboi, M.: The NASL2 Reference Manual. Tenable Network Security (2005)
12. Cheikes, B.A., Waltermire, D.: Common Platform Enumeration: Naming Specification Version 2.3 (DRAFT). Technical Report The MITRE Corporation, National Institute of Standards and Technology, NIST (2010)