Automated Generation of Attack Graphs Using NVD

M. Ugur Aksu1, 2, Kemal Bicakci2, M. Hadi Dilek1, A. Murat Ozbayoglu2, E. İslam Tatlı1
1STM Defense Technologies Engineering and Trade Inc. Ankara, Turkey
2TOBB University of Economics and Technology Ankara, Turkey
1{mugur.aksu, mhdilek, emin.tatli}@stm.com.tr
2{m.aksu, bicakci, mozbayoglu}@etu.edu.tr

ABSTRACT
Today’s computer networks are prone to sophisticated multi-step, multi-host attacks. Common approaches of identifying vulnerabilities and analyzing the security of such networks with naive methods such as counting the number of vulnerabilities, or examining the vulnerabilities independently produces incomprehensive and limited security assessment results. On the other hand, attack graphs generated from the identified vulnerabilities at a network illustrate security risks via attack paths that are not apparent with the results of the primitive approaches. One common technique of generating attack graphs requires well established definitions and data of prerequisites and postconditions for the known vulnerabilities. A number of works suggest prerequisite and postcondition categorization schemes for software vulnerabilities. However, generating them in an automated way is an open issue. In this paper, we first define a model that evolves over the previous works to depict the requirements of exploiting vulnerabilities for generating attack graphs. Then we describe and compare the results of two different novel approaches (rule-based and machine learning-employed) that we propose for generating attacker privilege fields as prerequisites and postconditions from the National Vulnerability Database (NVD) in an automated way. We observe that prerequisite and postcondition privileges can be generated with overall accuracy rates of 88.8 % and 95.7 % with rule-based and machine learning-employed (Multilayer Perceptron) models respectively.

CSCS CONCEPTS
• Security and privacy → Systems security: Vulnerability management;

KEYWORDS
attack graph generation, CVE, CVSS, NVD, vulnerability

1 INTRODUCTION
Today’s computer networks are facing complicated and increased numbers of attacks. In order to evaluate such threats, using vulnerability scanners to identify the number, type, and location of the vulnerabilities that exist at our networks is a common practice. However, such tools consider the vulnerabilities independently and do not show how one relate to another to reveal combinations of them that may pose significant threats to our networks. Defending such networks requires identification of each path into the networks and blocking any malicious access through those paths. To assess overall vulnerability of a network, vulnerabilities need to be grouped showing the multi-step and multi-host nature of them [1].

Generating attack graphs is very useful in combining low-level vulnerabilities to show the attack paths leading to the targets in the enterprise networks. Security professionals might focus on patches or configuration errors that pose greater risks, by analyzing the attack paths that might be exploited. Risk assessments generated from probabilistic attack graphs assist further such decisions [2][3].

A number of attack graph generation techniques have been proposed, though not all of them are feasible or accurate enough to be adopted in practice. We classify these attack graph generation approaches in three general categories:
• Prerequisite/Postcondition (Requires/Results-In) Models,
• Artificial Intelligence Based Models,
• Ontology-Based Models.

Attack graph generation techniques are reviewed in further detail in Section 2. In this paper, we firstly aim to define a prerequisite and postcondition classification scheme regarding attacker privileges for generating attack graphs in the context of the requires/results-in model. We define attacker privileges in a way to distinguish between privileges relating to physical and virtual machines. We also describe attacker prerequisite and postcondition privileges with the same set so that prerequisites and postconditions can be easily related one to another in attack graph generation. Then, we describe two different approaches, one with a rule-based method, and another with employing machine learning (ML), for generating these labels from the National Vulnerability Database (NVD) [4] in an automated way. The need for the automation arises from the fact that manually determining the attacker privileges corresponding to all vulnerabilities and continuing this effort as new vulnerabilities emerge seem impractical and require significant effort and time. Currently NVD hosts more than 92,000 vulnerability entries (In 2016 alone, 6,517 new vulnerabilities were identified) and the amount of identified vulnerabilities each year almost doubles.

The paper is organized as follows: Section 2 reviews the related work with a focus on attack graph generation approaches. Section 3 explains an enhanced prerequisite and postcondition model. Section 4 describe our rule-based and ML-employed automation approaches for generating prerequisite and postcondition labels and evaluates the results of these two approaches. Section 5 concludes the paper and discusses the future work.
2 RELATED WORK

A number of attack graph generation approaches have been proposed, each with different maturity and applicability levels. Among them, prerequisite/postcondition models are intuitive and simplistic in nature. Necessary conditions of exploiting the vulnerabilities are defined as prerequisites. Effects and the capabilities obtained by the attackers as a result of the exploits are named as postconditions. TVA (Topological Analysis of Network Attack Vulnerability) [1] and NETSPA (Network Security Planning Architecture) [5][6][7] are two of the well-known examples of this approach [3].

In order to generate attack graphs, TVA utilizes a knowledge database of exploit conditions in terms of preconditions and postconditions that relate to exploitation steps. Preconditions and postconditions specify in detail the network connectivity requirements and attacker privileges using natural language descriptions. Its shortcoming is that the preconditions and postconditions are manually generated from the vulnerability information available in natural language descriptions [7]. This approach demands intensive manual effort and requires the conditions database be enriched manually as new vulnerabilities emerge. Thus scalability and practicality are of significant concerns for this methodology.

NETSPA takes an approach of attacker state, which is a combination of the locality and effect information. Locality is processed as a precondition and categorized as remote and local. Effect information represents the postconditions of exploits and categorized into four levels: user, administrator, DoS and other. Combining vulnerability information from multiple sources, they generate preconditions and postconditions via a logistic regression model trained with a sample manual data. Their work is outdated since the vulnerability database (VDB) of ICAT is not maintained anymore and its replacement, the CVSS database provides fields of information significantly different than that of the ICAT. Secondly, their precondition and postcondition classification schemes seem to be limited, such that only locality knowledge of the attacker is used as a prerequisite, disregarding the privilege status. Lastly, their privilege classification scheme does not cover application level privileges.

In their work of analyzing NVD for the composition of vulnerabilities to generate attack scenarios, Franquiera and van Keulen [8] describe an approach of access-to-effect with little enhancements to the attacker state definition of the NETSPA. They describe access in the same way defined in the NETSPA. They name the effects in five categories, adding runCode and obtainCred to the previously defined user, admin and DoS categories of the NETSPA. Their model has similar shortcomings pointed out for the NETSPA.

For the category of artificial intelligence based models, MULVAL is a notable work. Relevant information to generate attack graphs, such as vulnerability descriptions and system configuration information are fed to the MULVAL as Datalog facts. Attack graphs are generated via a reasoning engine that correlates the facts given to the MULVAL [3][9][10]. Our experiment with the MULVAL produces significant rates of false positive and negatives.

Ontology-based attack graphs, which have been worked on recently, provide some valuable information, such as interrelations among concepts, not available in the taxonomy based information classification approaches. For the downside, although some initial ontologies for attack graph generation have been proposed [11][12], they require a lot more effort to be comprehensive enough for generating attack graphs deployed for real-life computer networks.

Rather than focusing on prerequisite and postcondition information for attack graph generation, a number of works elucidate specifically on extracting relevant information from the VDBs, which could aid the process of attack graph generation. Among these, Weerawardhana et al. [13] present two different solutions (machine learning based and linguistic patterns based) for information extraction from online VDBs. Though they identify a number of useful vulnerability information categories and show ways of extracting them automatically, their work lacks the privilege prerequisite and postcondition information categories that are essential to our approach. Secondly, Roschke et al. [14] investigate extraction of vulnerability information from textual descriptions for attack graph construction, in addition to analyzing and comparing the features of multiple VDBs. However, their work gives a few examples of keywords that can be extracted from textual descriptions rather than describing a complete and categorized list of them.

3 DESCRIPTION OF THE PROPOSED REQUIRES/RESULTS-IN MODEL

3.1 Overview of the Model

In this section, we define a generic requires/results-in model, that improves upon the previous works of [5][6] and [8] to depict a way of generating attack graphs. A comparison of our approach and the early works is given at Table 1. Requires/results-in models typically define exploits in terms of a set of prerequisite and postcondition rules. The resulting set of all the rules for the exploits are used as a knowledge base for attack graph generation [7].

Our model takes a set of information in four categories as its input and relates them to the privileges gained knowledge as its output via a reasoning engine that makes use of the knowledge base. The set of input information are:

- Vulnerability scan results that can be produced by tools such as NESSUS or OPENVAS,
- Topology and reachability information of the network,
- Attack Vector (AV) for each vulnerability and the locality of the attacker,
- Privilege of the attacker at their initial/current state and privilege prerequisites for the vulnerabilities.

Among this set of input information, vulnerabilities at a computer network can be identified by vulnerability scanning tools, and their results as CVE ids can be fed into the model. Regarding the network information, topology discovery tools can be used to identify the number and type of the assets and map their connectivity.

Reachability information, which indicates whether there exist logical connections among the network hosts given the physical connections, can be derived from the active network components, such as, routers, switches, firewalls and IDS/IPs.

The third input, AV for a vulnerability defines how a vulnerability can be exploited in terms of the attacker’s current location at the network and takes the values of Physical, Local, Adjacent Network and Network as described in [15].

The data of privileges required for the input set and the privileges gained as the model’s output are not readily available to
Table 1: Comparison Of Prerequisite/Postcondition Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Access Vector Prereq.</th>
<th>Privilege Prereq.</th>
<th>Privilege Postcondition</th>
<th>Database</th>
<th>Automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVA [1]</td>
<td>Exploitation steps are defined in natural language. No formal prerequisite or postcondition definitions. No linkage between prerequisites and postconditions.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Franqueria &amp; Van Keulen [7]</td>
<td>Remote Local</td>
<td>N/A</td>
<td>User Privilege Admin RunCode ObtainCred DoS</td>
<td>NVD</td>
<td>N/A</td>
</tr>
</tbody>
</table>

extract from the open VDBs. Nor there exists a formal definition or categorization accepted generally to characterize privileges of the attackers as prerequisites and postconditions. In the next subsection, we describe a novel privilege classification scheme and show two different approaches to classify vulnerabilities defined by the NVD with our privilege labels in Section 4.

The model that we propose is a multi-prerequisite approach since it considers not only the locality, but also the privilege level of the attacker, which is used only as a postcondition of an exploitation by the earlier works, as shown in Table 1. The reasoning for using attacker privileges as exploitation prerequisites can be captured from the textual descriptions (i.e. “a local authenticated user ...” or “a remote authenticated administrator ...”) of vulnerabilities that imply attacker privileges as exploitability requirements.

3.2 Privilege Classification

We define a set of five general privilege categories. The types and ordering of privilege classes are depicted in Figure 1. According to the capability levels, privileges are depicted in descending order from OS level to the None. Additionally, privileges at Admin levels are more capable than User level privileges.

![Figure 1: Categorization of Attacker Privilege Levels](image)

In addition to the operating system level privilege classification of the earlier works, we use application level privileges and differentiate privileges inside and outside the virtual machines. Application level privileges indicate privilege requirements/gains for specific applications for which the names can be derived from the CPE (Common Product Enumeration) fields of the vulnerabilities. They are useful in modeling the authentication requirements/gains for applications with or without regard to operating system level privileges.

The benefit of using application level privileges can be explained with an example. For CVE-2016-1990 the AV property is Local and its textual description is given as ‘HPE ArcSight ESM ... allows local users to gain privileges for command execution via unspecified vectors.’ If only the locality property of this CVE were used, being local at an asset would be enough to exploit it. However, our model requires the adversary to be both local and to have APP(USER) privilege by possessing authentication knowledge for the ArcSight ESM in order to exploit the vulnerability successfully.

To differentiate attacker privileges at the virtual machines from the physical ones, another set of labels starting with the letter “V” are defined. Since the privileges gained at either the physical or virtual machines do not have the same capabilities, we find it useful to differentiate between where exploitations start and where their impacts (privilege gains) occur, in terms of their physical or virtual locations. The need for and usefulness of such distinction can be explained with two chosen CVE examples as illustrated in Figure 2.

![Figure 2: Host/Guest Machine Exploitation Example](image)

In the examples, exploiting CVE-2007-5671 gives the attacker operating system level ADMIN privilege only on the same guest machine whereas CVE-2008-2098 is more dangerous since it provides ADMIN privilege on the physical machine hosting the guest machine.

3.3 Attack Graph Generation

Using attacker privileges both as exploitation prerequisites and postconditions in our model, vulnerabilities can be easily related...
to each other for attack graph generation. In order to exploit a given vulnerability that requires any attacker privilege, an attacker must have one or more of the required privileges at the operating system or application level, either as an administrator or a user. NONE, as a privilege prerequisite, implies the attacker does not need any of the four privileges listed at the operating system or application level. After exploiting a vulnerability, one or more of the categories of privileges can be acquired. The semantics of the NONE as a postcondition is that none of the privileges at the operating system or application level is gained after exploiting a vulnerability, disregarding any impact that might be caused by the exploitation. An exploitation with NONE as its privilege postcondition might cause any of the confidentiality, integrity and availability impacts, as described at [15] and such impact information for each known vulnerabilities can be derived from the NVD.

In our model, attack graphs can be generated by the Reasoning Engine that interacts with the Knowledge Base of enriched vulnerability information, using the Algorithm 1 that is based on the earlier work [5]. Nodes in this algorithm represents the states of the attackers as a pair of locality and privilege information. Starting from the initial node, for each node that is physically and logically reachable, vulnerabilities existing at the target nodes are examined to determine if they are exploitable. If both the attack vector and privilege level parameters at a given node suffice to exploit a given vulnerability, then a directed edge that represents an attack path is added between the current and target nodes.

Using the Algorithm 1, example attack graphs generated on a simple network are shown in Figure 3. On the sample network, firewall rules allow the outsiders only to communicate with the Apache HTTP Server. Inside the network, there exist a trust relationship between the Apache HTTP Server and the Red Hat Server. Other than than, host are denied any communication among them by the router rules. Linux Server hosts a Guest Machine which is highly untrusted, thus the Guest Machine is not allowed to communicate with any other host on the network, including the host machine on which it resides. For this simple scenario, there exist two malicious adversaries, one outside the local network, and another inside the local network, at the untrusted Guest Machine.

From the attack graphs depicted in Figure 3, it can be observed that the Attacker 1 can exploit Device 1, 2 and 5 with OS(ADMIN) privileges and causes a Denial of Service impact on the Device 3. On the other hand, the Attacker 2, who is a malicious insider with access only to a virtual machine with no connection to the other devices in the network, can exploit the Device 3 with OS(ADMIN) privilege in addition to the devices exploitable by the Attacker 1.

4 AUTOMATED GENERATION OF THE PRIVILEGES

Given the many on-line and public VDBs, extracting or generating the prerequisites and postconditions for attacker privileges for known vulnerabilities might seem to be a trivial task. However, firstly, analysis of VDBs and listings reveals considerable amount of missing, inconsistent or incorrect data [7]. Secondly, such data are not readily available to extract from the data fields supported by the public VDBs, such as NVD. The available VDBs have no defined formal languages and they generally define the vulnerability information in terms of taxonomy and rely on natural language text for a considerable part of the definitions [7]. These findings lead us to investigate for an automated way of capturing the semantics of exploit prerequisites and postconditions in terms of attacker privileges, which is then fed into a knowledge base to generate attack graphs practically with the proposed requires/results-in model.

In the following subsections, we describe in detail the two different approaches of rule-based and ML-employed for generating attacker privilege labels and compare their results on the confusion matrices using metrics of accuracy rates as well as precision and recall values. Accuracy rates are defined as the ratio of correctly identified classes compared with the total number of vulnerabilities. Precision values show the ratio of total number of correct identifications given the total number of predictions for each class, while the recall values demonstrate the ratio of total number of correctly identified classes given the actual total numbers of the classes. The results of the models are checked against an experimental dataset given at Table 2 in order to determine their accuracy rates. This evaluation dataset of NVD vulnerabilities has been generated manually by carefully analyzing more than 550 vulnerabilities out of the 92000 currently available at the NVD and has been chosen in a way to cover most of the different types of vulnerabilities, such as with varying impacts, weaknesses or attack types.

We note that we have not included incorrect data, such as the data for CVE-2008-0840 described by [14], in this dataset. Specifically, 20 vulnerability entries for postcondition and 1 entry for

---

**Algorithm 1: Attack Graph Generation Algorithm**

```plaintext
1 priv ← {OS/VOS(Adm.), OS/VOS(User), APP(Adm.), APP(User), None/VNone};
2 AV ← {Physical, Local, Adjacent, Network};
3 curNod.pri ← initial privilege of the attacker ∈ priv;
4 curNod, destNod, currentNod.av, v ← null;
5 attackerNodes ← {SET - attacker's initial node};
6 destNodes, V1, V2 ← {null};

7 while attackerNodes is not empty do
8     curNod ← attackerNodes.pop();
9     V1 ← {SET - vulnerabilities at the curNod};
10    foreach v ∈ V1 do
11        if curNod.pri >= curNod.v.priPre then
12            if curNod.v.priPost > curNod.pri then
13                curNod.pri ← curNod.v.priPost;
14            end
15        end
16    destNod ← {SET - nodes reachable from the curNod};
17    foreach destNod ∈ destNodes do
18        V2 ← {SET - vulnerabilities at the destNod};
19        foreach v ∈ V2 do
20            if curNod.av >= destNod.av.v then
21                if (destNod.v.priPre == NONE) OR
22                   (destNod.v.priPre > destNod.v.priPre) then
23                    if destNod.v.priPost > destNod.pri then
24                        destNod.pri ← destNod.v.priPost;
25                        addEdgeBetw(curNod, destNod);
26                        attackerNodes.add(destNod);
27                    end
28                end
29            end
30        end
31 end
```
prerequisite determination have been excluded. However, we handle the inconsistencies where the same information is expressed with varying vocabulary or when the relevant information is expressed sporadically. For instance, CVE-2005-1207 has the textual description “Buffer overflow in the Web Client service in MS Windows XP and Windows Server 2003 allows remote authenticated users to execute arbitrary code via a crafted WebDAV request containing special parameters,” with no usage of “root” and we still correctly identify the privilege postcondition through checking another expression (buffer overflow) together with the CVSS Impact score.

### 4.1 Rule-Based Generation of the Privileges

As our first method, we generate privilege information relating to the vulnerabilities with a rule based reasoning engine. Rules at the reasoning engine have been defined manually by analyzing the experimental data given at Table 2. The manually generated rules are static in that they do not dynamically change or increase in size as the vulnerability data imported from the NVD enlarges in quantity. The defined rules process both taxonomy-based and textual data to capture the semantics of the vulnerabilities. Below, we give an overview of the fields of data employed in our rules. The detailed definitions of the fields can be found at CVSS 2.0 [16] and CVSS 3.0 [15] specifications.

As described in the previous sections, **Attack Vector (AV)** (CVSS 2.0 and 3.0) denotes the locality of the attacker with regard to the network asset on which a vulnerability exists. It takes the values of Physical, Local, Adjacent Network and Network.

**Authentication** (only CVSS 2.0) field shows the number of times an attacker must authenticate to a vulnerable target and it takes the values of None, Single and Multiple. In the context of their usage in our rules, we are interested in only if a vulnerability requires any authentication or not. A Not None value implies that the related vulnerability requires an attacker privilege at either the operating system or application level.

**Privilege** (only CVSS 3.0) field expresses the level of privilege an attacker must possess and it takes the values of None, Low and High. A value of Low denotes basic user capabilities while a value of High indicates significant control over the vulnerable asset. From this parameter, attacker privilege level as a prerequisite can be inferred as either Administrator or User. But it is not possible to determine its privilege level as operating system or application level only by analyzing this field.

Common Platform Enumeration (CPE) [17] data shows a set of vulnerable products with regard to each known vulnerabilities. The vulnerable platforms are represented in three categories, such as operating systems, firmwares and applications. Correlating the CPE data with the **Authentication** and **Privilege** data enables us to derive further knowledge on these fields that are not explicit when they are analyzed on their own.

**Impacts** (CVSS 2.0 and 3.0) data with regard to the three pillars (Confidentiality, Integrity, Availability) of the information security show the damage induced at the victim and takes the values of None, Partial, and Complete. This field is especially useful in determining the privilege gained after exploiting a vulnerability.

Another field of information, **Security Protection**, which is not defined formally as a data field by the NVD in the CVSS 2.0 and 3.0 specifications, have been discovered by our manual analysis of the NVD data feed of XML 2.0. This field denotes the attacker privilege postconditions as Admin, User, or Other. However, labeling vulnerabilities with this field is not continued by the NVD and only a minority of them have been labeled with this field.

In addition to the above explained taxonomy of data, natural language descriptions are also used by the NVD to explain the vulnerabilities. A number of single words or sequences of words that we have identified through our manual analysis of the description are also employed in our rules and proves to be very useful in determining attacker privileges.

Table 3 and 4 depict the rules to produce attacker privilege prerequisites and postconditions, respectively. Rules in both categories process both taxonomy and natural language-based data for determining the privileges. The ellipsis in the keywords represent any number of words residing in the same sentence.

---

**Figure 3: Examples of Attack Graphs on a Sample Network**

---

As described in the previous sections, **Attack Vector (AV)** (CVSS 2.0 and 3.0) denotes the locality of the attacker with regard to the network asset on which a vulnerability exists. It takes the values of Physical, Local, Adjacent Network and Network.
Table 2: Distribution of Privilege Classes on the Experimental Dataset

<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>OS (Admin)</th>
<th>VOS (Admin)</th>
<th>OS (User)</th>
<th>VOS (User)</th>
<th>App (Admin)</th>
<th>App (User)</th>
<th>None</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-requisite</td>
<td>31</td>
<td>72</td>
<td>100</td>
<td>49</td>
<td>31</td>
<td>77</td>
<td>150</td>
<td>570</td>
</tr>
<tr>
<td>Post-condition</td>
<td>168</td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>60</td>
<td>23</td>
<td>199</td>
<td>551</td>
</tr>
</tbody>
</table>

Table 3: Rules For Producing Attacker Privilege Prerequisites

<table>
<thead>
<tr>
<th>#</th>
<th>AV</th>
<th>Authentication</th>
<th>Privilege</th>
<th>CPE</th>
<th>Pre-Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>None</td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>2</td>
<td>Local</td>
<td>-</td>
<td>Low</td>
<td>Only OS</td>
<td>OS(USER)/VOS(USER)</td>
</tr>
<tr>
<td>3</td>
<td>Local</td>
<td>-</td>
<td>High</td>
<td>Only OS</td>
<td>OS(ADMIN)/VOS(ADMIN)</td>
</tr>
<tr>
<td>4</td>
<td>Local</td>
<td>! None</td>
<td>Low</td>
<td>APP/HW</td>
<td>APP(USER)</td>
</tr>
<tr>
<td>5</td>
<td>Local</td>
<td>! None</td>
<td>High</td>
<td>APP/HW</td>
<td>APP(ADMIN)</td>
</tr>
<tr>
<td>6</td>
<td>Local</td>
<td>None</td>
<td>Low</td>
<td>APP/HW</td>
<td>OS(USER)/VOS(USER)</td>
</tr>
<tr>
<td>7</td>
<td>Local</td>
<td>None</td>
<td>High</td>
<td>APP/HW</td>
<td>OS(ADMIN)/VOS(ADMIN)</td>
</tr>
<tr>
<td>8</td>
<td>! Local</td>
<td>! None</td>
<td>Low</td>
<td>Only OS</td>
<td>OS(USER)/VOS(USER)</td>
</tr>
<tr>
<td>9</td>
<td>! Local</td>
<td>! None</td>
<td>High</td>
<td>Only OS</td>
<td>OS(ADMIN)/VOS(ADMIN)</td>
</tr>
<tr>
<td>10</td>
<td>! Local</td>
<td>! None</td>
<td>Low</td>
<td>APP/HW</td>
<td>OS(USER)</td>
</tr>
<tr>
<td>11</td>
<td>! Local</td>
<td>! None</td>
<td>High</td>
<td>APP/HW</td>
<td>APP(ADMIN)</td>
</tr>
</tbody>
</table>

Table 4: Rules For Producing Attacker Privilege Postconditions

<table>
<thead>
<tr>
<th>#</th>
<th>Vocabulary</th>
<th>Impacts</th>
<th>CPE</th>
<th>Post-Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'gain root'</td>
<td>All</td>
<td>Complete</td>
<td>OS(ADMIN)</td>
</tr>
<tr>
<td>2</td>
<td>'gain privilege'</td>
<td>All</td>
<td>Complete</td>
<td>OS(ADMIN)</td>
</tr>
<tr>
<td>3</td>
<td>'unspecified vulnerability'</td>
<td>All</td>
<td>Complete</td>
<td>OS(ADMIN)</td>
</tr>
<tr>
<td>4</td>
<td>'gain admin'</td>
<td>Partial</td>
<td>Only OS</td>
<td>OS(USER)</td>
</tr>
<tr>
<td>5</td>
<td>'gain admin'</td>
<td>Partial</td>
<td>APP/HW</td>
<td>APP(ADMIN)</td>
</tr>
<tr>
<td>6</td>
<td>'hijack the authentication of admin'</td>
<td>-</td>
<td>APP(ADMIN)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>'hijack the authentication of users'</td>
<td>-</td>
<td>APP(ADMIN)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>'hijack the authentication of unspecified victims'</td>
<td>All</td>
<td>Complete</td>
<td>Only OS</td>
</tr>
<tr>
<td>9</td>
<td>'obtain password'</td>
<td>Partial</td>
<td>Only OS</td>
<td>OS(USER)</td>
</tr>
<tr>
<td>10</td>
<td>'obtain password'</td>
<td>Partial</td>
<td>APP/HW</td>
<td>APP(ADMIN)</td>
</tr>
<tr>
<td>11</td>
<td>'obtain plaintext'</td>
<td>All</td>
<td>Complete</td>
<td>Only OS</td>
</tr>
<tr>
<td>12</td>
<td>'buffer overflow'</td>
<td>All</td>
<td>Complete</td>
<td>OS(ADMIN)</td>
</tr>
<tr>
<td>13</td>
<td>'SQL injection'</td>
<td>All</td>
<td>Complete</td>
<td>OS(ADMIN)</td>
</tr>
<tr>
<td>14</td>
<td>'executing script'</td>
<td>Partial</td>
<td>None</td>
<td>OS(USER)</td>
</tr>
<tr>
<td>15</td>
<td>'execute command'</td>
<td>All</td>
<td>Complete</td>
<td>OS(ADMIN)</td>
</tr>
<tr>
<td>16</td>
<td>'execute command'</td>
<td>Partial</td>
<td>None</td>
<td>OS(USER)</td>
</tr>
<tr>
<td>17</td>
<td>'execute SQL'</td>
<td>All</td>
<td>Complete</td>
<td>OS(ADMIN)</td>
</tr>
<tr>
<td>18</td>
<td>'execute SQL'</td>
<td>Any</td>
<td>None</td>
<td>NONE</td>
</tr>
</tbody>
</table>
Regarding the application of the rules, *Logical And* is used for reasoning, such that, only if all the fields of a given rule satisfy for a given vulnerability, then the privilege level of the rule is assigned to that vulnerability as a prerequisite or postcondition. If more than one rule apply to a given vulnerability, the rule with the highest privilege level overrides. In case none of the rules applies for a given vulnerability, a default value (None for prerequisites, OS(Admin) for postconditions) is assigned as its privilege level. Our rule-based model generates privilege prerequisite and postcondition labels with accuracy rates of 87.7% and 89.8% respectively. These accuracy rates are defined as the ratio of correctly identified classes compared with the total number of vulnerabilities. Confusion matrices given at Tables 6 and 7 can be used to further investigate the precision and recall values for each privilege level.

### 4.2 ML-Employed Generation of the Privileges

To further investigate the possibility of increasing the accuracy rate of the rule-based model for determining attacker privileges, we have experimented on four different ML-employed approaches listed below:

- Radial Basis Function (RBF) Networks,
- Support Vector Machines (SVM),
- Neuro Evolution of Augmenting Topologies (NEAT),
- Multi Layer Perceptron (MLP).

Among these approaches, RBF networks is known for its ability to respond well to fast local changing borders between classes, but it might not be as good generalizer as an MLP [18]. SVM, on the other hand, is one of the best classifiers for binary classification problems, since it finds the optimum discriminative function that maximizes the margin between the classes, but might be difficult to implement in multiclass problems, as is in our case [18]. NEAT is a neural network which uses evolutionary algorithms to dynamically alter its topology and connections to achieve better results than static networks [19]. The last of these approaches, MLP, is a widely used model due to its success in various different application areas and its well-defined implementation methodology [18].

MLP has one input layer, one or more hidden layers, and one output layer. Basically, MLP takes the inputs and assigns them to the desired outputs by mapping through the hidden layer neurons. The mapping process is implemented through an iterative optimization process called gradient descent based error backpropagation [18]. The iterative process continues until the sum of error squares between the desired and the model outputs drops below an acceptable threshold level (or the cross validation error starts increasing, as used in our model).

Among these ML models, we focus on the MLP as our ML-employed model due to its significantly better results than the other three approaches. Thus we compare in detail only the results of the MLP model with the rule-based model and give an accuracy comparison chart of these four ML-employed models at Table 5.

As input to our MLP model, we use the same set of taxonomy-based and vocabulary-based categories of information applied in our rule-based model. Additionally, we utilize CVSS 2.0 scores ranging from 0 to 10 as an extra information. As the output, the privilege categories are determined.

Given the dataset at Table 2, 5-fold Cross Validation (CV) and Testing of the data is implemented. 60% of the data was used for training, 20% is used for CV and 20% is used for testing in all 5 cases. As a result, all available data is tested. The corresponding Confusion Matrix for privilege prerequisites is provided in Table 6 and for privilege postconditions in Table 7.

The Confusion Matrix rows represent the actual appearance of any class, i.e., in the privilege prerequisite model (Table 6), out of 570 data points, OS(User) is seen 160 times (sum of the terms in row 3 of the confusion matrix). In a similar fashion, the columns of the confusion matrix represent how many times the model predicts a certain class. For example, the model predicts OS(User) 158 times (sum of the terms in column 3 of the confusion matrix).

Precision is defined as the ratio of the correct predictions of a certain class. In the privilege prerequisite model (Table 6), out of the 158 OS(User) predictions, 152 of them are correct (and 6 of them are wrong). As a result, the precision value for OS(User) class is 0.96. Of the 6 wrong predictions 5 are actually VOS(User), and 1 is None. Recall represents the ratio of the actual class instances within the predictions that the model made for a certain class. In the privilege prerequisite model (Table 6), out of the 160 OS(User) data points that exist in the dataset, the model is able to determine 152 of them correctly. Thus the recall value for OS(User) is 152/160 = 0.95.

Overall accuracy is defined as the ratio of correctly identified classes (sum of the diagonal values in the confusion matrix) compared with the total number of points. In our study, we are able to achieve 96.1% overall accuracy for privilege prerequisite model and 95.4% for the privilege postcondition.

### 4.3 Comparison of the Models

The rules we defined can produce privilege prerequisite and postcondition labels with accuracy rates of 87.7% and 89.8%, respectively. The rule-based privilege generation method proposed in our work is not completed. We show that such a rule based approach can be used to enhance the data derived from the NVD, so that attack graphs that are consistent and with minimized false positive/negative rates can be automatically generated. By adding more rules, the accuracy of the this approach can be increased further.

For the ML model, we get the accuracy rates of 96.1% and 95.7% for privilege prerequisites and postconditions, respectively. Comparing the results of these two models, ML-employed model achieves significantly better results for both the privilege prerequisites and postconditions. This promising result of our MLP model indicates that ML techniques can be also used successfully for privilege determination in order to generate attack graphs.

In addition to the higher overall accuracy rate, the feature of generating the privileges with confidence levels, compared to the 1 (found) or 0 (not found) nature of the rule-based model is a significant advantage of the MLP model. This feature is especially useful in manually evaluating the generated privileges that are below a defined confidence threshold level.

### Table 5: Accuracy Rates of the ML-Employed Models

<table>
<thead>
<tr>
<th>Privilege Type</th>
<th>RBF</th>
<th>SVM</th>
<th>NEAT</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privilege Prerequisite</td>
<td>58.6</td>
<td>90.7</td>
<td>91.8</td>
<td>96.4</td>
</tr>
<tr>
<td>Privilege Postcondition</td>
<td>54.3</td>
<td>91.6</td>
<td>92.1</td>
<td>95.7</td>
</tr>
</tbody>
</table>
Table 6: Confusion Matrix For Privilege Prerequisites

<table>
<thead>
<tr>
<th>Predicted</th>
<th>OS (ADMIN)</th>
<th>VOS (ADMIN)</th>
<th>OS (USER)</th>
<th>VOS (USER)</th>
<th>APP (ADMIN)</th>
<th>APP (USER)</th>
<th>NONE</th>
<th>Precision Values</th>
<th>Overall Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS (ADMIN)</td>
<td>31 30 1 0 0 1 0</td>
<td>0 0 0 0 0 0 0</td>
<td>100 0.97</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOS (ADMIN)</td>
<td>25 0 45 72</td>
<td>0 0 0 0 1 0 0</td>
<td>0.63 100</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS (USER)</td>
<td>5 1 0 0</td>
<td>0 0 0 0 2 2</td>
<td>0.88 100</td>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOS (USER)</td>
<td>0 0 0 0 7 5 35 43 0 0</td>
<td>1 0 3 1</td>
<td>0.78 0.88</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APP (ADMIN)</td>
<td>0 1 0 0 0</td>
<td>0 0 0 0 1</td>
<td>0.97 0.94</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APP (USER)</td>
<td>0 0 0 0 0 7 0 0 0</td>
<td>1 1 66 73 3 3</td>
<td>0.86 0.95</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td>0 0 0 0 0 1</td>
<td>0 0 0 0</td>
<td>0.98</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recall Values

Overall Accuracy: 0.88 0.96

Table 7: Confusion Matrix For Privilege Postconditions

<table>
<thead>
<tr>
<th>Predicted</th>
<th>OS (ADMIN)</th>
<th>VOS (ADMIN)</th>
<th>OS (USER)</th>
<th>VOS (USER)</th>
<th>APP (ADMIN)</th>
<th>APP (USER)</th>
<th>NONE</th>
<th>Precision Values</th>
<th>Overall Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS (ADMIN)</td>
<td>164 163</td>
<td>0 0 3 2 0</td>
<td>0 1 2 0</td>
<td>1 0 1 3 0</td>
<td>0.96 0.97</td>
<td>168</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOS (ADMIN)</td>
<td>1 0 0 0</td>
<td>0 0 1</td>
<td>0 0 0 0</td>
<td>2 2</td>
<td>0.92 0.93</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS (USER)</td>
<td>5 5 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0.99</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOS (USER)</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APP (ADMIN)</td>
<td>8 3 0 0</td>
<td>10 1 0</td>
<td>0 3 0</td>
<td>35 55</td>
<td>0 3 1</td>
<td>0.65 0.92</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APP (USER)</td>
<td>11 0 0 0</td>
<td>0 0 3 0</td>
<td>0 2 0</td>
<td>9 20 1</td>
<td>0</td>
<td>0.40 0.87</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td>5 0 0 0</td>
<td>0 0 2 0</td>
<td>0 0 0 0</td>
<td>1 1 0</td>
<td>194 195</td>
<td>0.97 0.98</td>
<td>199</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recall Values

Overall Accuracy: 0.90 0.95

5 CONCLUSION AND FUTURE WORK

In this work, we first defined a generic requires/results-in model and an algorithm based on the early work for attack graph generation. Then, we defined an enhanced categorization of attacker privileges and showed two different methods, rule-based and ML-employed, for generating attacker privileges as prerequisites and postconditions from the vulnerability in the NVD. ML-employed MLP model achieved an accuracy of 96.1 % and 95.4 % for privilege prerequisite and postconditions, respectively, compared to the accuracy rates of 87.7 % and 89.8 % we get from the rule-based model.

The promising results of the models we have demonstrated urge us to further explore using a hybrid model as a future work. Possible usage scenarios for such a model are as listed below:

- Using the ML-employed model only for the vulnerabilities for which rule-based model does not cover;
- Using the results of the rule-based model as an additional feed to the ML-employed model;
- Comparing the results of the two models and alerting for manual analysis when the outputs of the models disagree.

REFERENCES