# A New Approach for Network Vulnerability Analysis

Hai L. Vu*, Kenneth K. Khaw*, TY Chen and Fei-Ching Kuo
*Centre for Advanced Internet Architectures (CAIA)
Faculty of ICT, Swinburne Univ. of Technology, P.O. Box 218, Melbourne, VIC 3122, Australia
Email: h.vu@ieee.org

***Abstract*—We propose in this paper a novel approach to analyze network vulnerability and to obtain a quantitative value representing the level of security achieved in an arbitrary network. Unlike previous graph-based algorithms that generate attack trees (or graphs) to cover all possible sequences of vulnerabilities and therefore are not scalable, our method utilizes the attack graph's principles, but directly analyzes and produces the desired security measure for a network without building the actual attack graph. The proposed approach relies on a unique evaluation of vulnerability metric defined in this paper and is demonstrated through an example of a network that provides voice over IP services.**

## I. Introduction

Protecting a network against malicious intrusions is crucial to the overall security of the network which becomes a real challenge as the size of networks grows. Attackers can combine multiple vulnerabilities (or exploits) on various hosts in a network to achieve their goals such as gaining access to or disrupting services offered by that network. In the past, attack trees (or graphs) representing a collection of possible sequences of vulnerabilities used to reach the attack goal are constructed manually by a so-called *Red team*. Having such an attack graph, system administrators can then improve the security of their network by patching vulnerabilities and configuration errors which pose the greatest risk to the network. However, manual construction of attack graphs is error-prone and impractical for large networks. Recent studies suggested the use of model checking [8] to automate the process of constructing attack graphs, and to ensure that they are exhaustive and succinct [3], [4], [5]. Although automating makes it easier to analyze and evaluate the overall vulnerabilities of a network and its security, scalability remains the main concern for graph-based network vulnerability analysis. It is because the size of the attack graph quickly becomes too large to be practical as the size of the network increases. For example, as reported in [4], a network with 5 hosts and 8 vulnerabilities would result in an attack graph of 5,948 nodes and 68,364 edges in a state space of 229 bits.

There have been two main approaches in the literature attempting to address the scalability problem [1], [2]. In particular, the authors in [1] introduce a *monotonicity* assumption that dramatically reduces the size of the attack graph without losing information that are required to facilitate the network vulnerability analysis. The monotonicity assumption assumes that the preconditions (as well as postconditions) of a vulnerability are conjoined, and that the attacker will never have to backtrack, i.e., relinquish a state that it has successfully exploited before. A different approach is proposed in [2] where the reachability matrix that represents the connectivity between arbitrary node pair in a network is collapsed into smaller sub-matrices and therefore reduces the computing cost. This is done by grouping hosts that are treated identically by filtering devices and therefore have the same reachability within the subnet and cross different subnets of the network. The authors in [2] report results for a real operational network consisting of 252 hosts; 3,777 ports and 8,585 vulnerabilities. The resulting attack graph contains 8,901 nodes and 23,315 edges which can be further simplified by grouping nodes together for better visualization. But even then the attack graph is still too large and complicated for a human to interpret and analyze.

In this paper we propose a framework for network vulnerability analysis based on a unique vulnerability metric and operations that enables development of a more scalable algorithm to analyze and compute the desired security measure for a network without building the actual attack graph. To this end, we will apply the monotonicity assumption and utilize the principles of attack graph in analyzing network vulnerabilities; however, the proposed algorithm itself will not depend on the resulting attack graph and therefore make it more scalable.

The rest of the paper is organized as follows. We describe the framework for network vulnerability analysis in Section II where we define our vulnerability metric and related operations. The main algorithm for assessing network security based on these metric operations is also proposed in this section. In Section III we demonstrate through an example the applicability of our framework by assessing the level of security of a network that provides voice over IP services. Finally, we conclude our paper in Section IV.

## II. Proposed Framework for Network Vulnerability Analysis

In this section we describe our framework for network vulnerability analysis and propose a novel algorithm that bypasses the construction of an attack graph and therefore remains scalable as the size of networks grows. In order to do so, however, we will start with a resulting attack graph (or attack tree for a given objective) obtained from any of the previous proposals in the literature to define our vulnerability metric and its operations.

## *A. Vulnerability metric and its operations*

Without loss of generality, let us consider an attack tree in Fig. 1 obtained from an automated generation technique such as the one proposed in [4]. Note that an attack tree is a graph in which each possible combination of vulnerabilities (or exploit chain) ends in a leaf state that satisfies the attacker's goal. Attack graph, on another hand, is a collection of several attack trees in which in some or all common states are merged. When the objective (or goal) of the attacker is specific, then the attack graph and attack tree both represent the same sequences of exploits the attacker can take to achieve its goal. In the example shown in Fig. 1, the attacker can utilize a number of vulnerability sequences to reach the goal. In particular, there are five vulnerabilities denoted by $V_i,\ i = 0, ..., 4$, and the attacker can choose either of the following vulnerability sequences to reach the goal $G$: $\{V_1 \rightarrow G;\ V_0 \rightarrow V_1 \rightarrow G;\ V_2 \rightarrow V_4 \rightarrow G;\ V_3 \rightarrow V_4 \rightarrow G\}$.

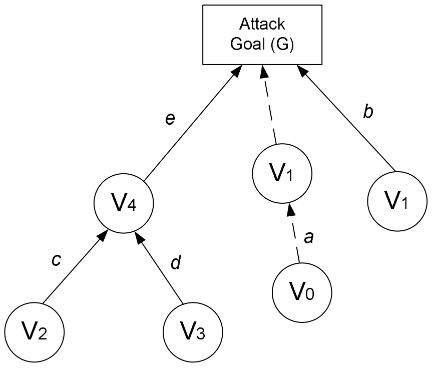

Fig. 1. Framework and concepts: an illustration example.

To analyze the attack tree shown in Fig. 1, we first define a vulnerability metric for each of the vulnerability on the tree. In principle, the vulnerability metric should represent the intrinsic qualities of a vulnerability, but at the same time reflect any characteristics of a vulnerability that change over time or are unique to any users environment. The intrinsic qualities of a vulnerability are those that are constant with time and across user environments. These may include the availability, complexity and whether or not extra conditions are required to exploit the vulnerability. Each of these qualities should also be considered in the context of the impact of the vulnerability on services (if exploited) in terms of confidentiality, integrity and availability. Furthermore, as the threat posed by the vulnerability may change over time, the vulnerability metric should also reflect factors such as the availability of exploit code or the remediation status of the vulnerability. Finally, the specific environment or service being offered can influence the potential damage that the vulnerability could cause, and the metric should allow users to alter its value correspondingly. The choice of vulnerability metric may depend on the specific network and application and we will show a concrete example in Section III where we assess the level of security of a VoIP network.

For a vulnerability metric that satisfies the above principles, we define two operations to capture the dependency among vulnerabilities within an exploit chain, and to reflect on the relationship between different exploit chains within an attack tree. These operations are defined based on the domain of vulnerability metric denoted by $\mathcal{V}$.

The first operation denoted by $\cup : (\mathcal{V} \times \mathcal{V}) \rightarrow \mathcal{V}$ gives a vulnerability metric that represents the overall vulnerability value of an exploit chain consisting of several vulnerabilities. For example, given the metrics $\{a, b, c, d, e\} \in \mathcal{V}$ associated with vulnerabilities $V_i,\ i = 0, ..., 4$ as depicted in Fig. 1, the metric of the $V_3 \rightarrow V_4 \rightarrow G$ exploit chain is $d \cup e \in \mathcal{V}$. We define the $\cup$ operation such that $d \cup e < d, e$ which indicates the overall vulnerability measure of any chain is less than an individual measure of exploits on that chain. It is because within an exploit chain, one vulnerability has to take advantage of the successful penetration of another vulnerability preceding it in the chain, and only the last exploit achieves the attacker's goal.

The second operation denoted by $\cap : (\mathcal{V} \times \mathcal{V}) \rightarrow \mathcal{V}$ gives a vulnerability metric that represents the overall vulnerability value of two or more vulnerabilities (or exploit chains) that induce the same postcondition after the penetration is achieved. For example, as shown in Fig. 1, both $V_2$ and $V_3$ vulnerabilities reach the same node ($V_4$) on the attack tree (i.e. induce the same postcondition), and thus their overall vulnerability value (metric) is calculated as $c \cap d \in \mathcal{V}$. We define the $\cap$ operation such that $c \cap d \geq c, d$ reflecting the fact that the attacker can choose any of the exploits to achieve the same objective (postcondition) and therefore the overall metric is greater or equal to vulnerability value of individual exploits. The list of properties associated with the above two operations is given below.

- $a \cup b < a, b$ and $a \cap b \geq a, b$
- $\cup$ is a commutative [1] and associative operation, i.e., $a \cup b = b \cup a$ and $a \cup (b \cup c) = (a \cup b) \cup c$,
- and $\cap$ is a commutative and associative operation, i.e., $a \cap b = b \cap a$ and $a \cap (b \cap c) = (a \cap b) \cap c$.

For many cases in network vulnerability analysis, however, it is to find the most critical path or a chain of exploits that poses the greatest risk to the system. For this purpose we will apply in our framework the $\cap = \max$ operation, i.e., $a \cap b = \max\{a, b\} \in \mathcal{V}$. Let us define a *minimum cut-set* in an arbitrary attack tree (or attack graph) by a set of exploits such that the attacker's goal will not be achieved if any of the vulnerability is removed from the set. Clearly each minimum cut-set is a chain of exploits, which we call a *critical* path, within the attack tree that the attacker can follow to achieve its goal. A *non-critical* path is a path that the attacker can still achieve its objective if some of the vulnerabilities are removed from that path. Generally, an attack tree (or graph) consists of both critical and non-critical paths as model

[1]Although the commutative property holds, reversing the order in this operation might not be valid as vulnerabilities usually follow a strict order on an exploit chain.

checker generates every possible combination of exploits that satisfies the attacker's goal. Here we are interested in the most critical path whose overall vulnerability metric can represent the vulnerability measure of the network that the attack tree modeled. The following lemma states an important result using the above defined operations.

*Lemma 1:* For any attack tree consisting of nodes (network states) and directed edges (vulnerability exploits), applying the $\cup$ and $\cap$ operations over all possible goal-induced exploit chains of the attack tree will yield the overall vulnerability metric of the most critical path within that tree.

Lemma 1 implies that regardless of how many critical and non-critical paths existed on the attack graph, the resulting metric obtained via $\cup$ and $\cap$ operations over all possible paths is the overall vulnerability metric of the most critical path only. The rigorous proof of Lemma 1 is omitted in this paper due to space limitations, but the fundamental principles can be illustrated in the example shown in Fig. 1. In particular, there are three critical paths (minimum cut-sets): $\{V_1 \rightarrow G;\ V_2 \rightarrow V_4 \rightarrow G;\ V_3 \rightarrow V_4 \rightarrow G\}$, and one non-critical path: $V_0 \rightarrow V_1 \rightarrow G$ exist in Fig. 1. Observe that both $V_1$ and $V_0 \rightarrow V_1$ reach the attacker's goal, and their overall metric is calculated as $(a \cup b) \cap b = \max\{(a \cup b), b\} = b$, which is a metric of the vulnerability $V_1$. Thus the resulting metric will only depend on $V_1$ and not on the metric of $V_0 \rightarrow V_1$ attack chain. It is because exploiting $V_1$ alone satisfies the attacker's objective regardless of whether $V_0$ is exploited or not, and from the vulnerability analysis point of view, $V_1$ is a critical path (or a minimum cut-set) that leads to system compromise. Continuing the computation over the remaining exploit chains $V_2 \rightarrow V_4 \rightarrow G$ and $V_3 \rightarrow V_4 \rightarrow G$ will give the metric of the most critical path of the example attack tree.

### *B. Framework for Network vulnerability Analysis*

We describe in the following the framework for network vulnerability analysis, and propose the main algorithm based on the above defined metric operations. Our framework consists of two parts and is described sequentially as follows.

The first part is to gather information from network administrators, and via scanning tools and other network discover activities. Information provided by the network administrator, which we call design information, include network topology and connectivity, existing security measures in the network such as policies, procedures and rules placed in firewalls, routers/switches and other network devices. Note that some of the design information might not be available in practice due to one reason or another, and the framework will have to work with information that are less specific. A bulk of information which we call discovery information, however, come from common available network scanning tools such as `nmap` [9], `Nessus` [10], `Retina` [11] and `Wireshark` [12]. These tools can be used to do a thorough scan the target network (for open ports, OS, services, etc.) and gather any related data available on that network. In particular, tools such a `nmap` or `Wireshark` can be used to gather information on network's components such as open ports, OS versions and current running services but does not identify the vulnerabilities that are associated with them. To specifically scan for vulnerabilities, tools such as `Nessus` and `Retina` are designed to scan a network and report all vulnerability(s) they can find within that network. These tools, however, do not specify which network component(s) the reported vulnerability is associated with.

Once the available information (design and discovery) are collected, an attack graph is generated using a model checker in the previous methods to represent all possible sequence of exploits. The vulnerability analysis is then carried out after obtaining the attack graph. As mentioned in Section I, scalability is a serious problem in approaches that use model checking. In contrast, we propose here in the second part of our framework a direct approach for network vulnerability analysis without building up the attack graph and is therefore more scalable. Our algorithm utilizes results stated in Lemma 1 to search directly for the most critical path and its overall vulnerability metric based on design and discovery information that are available from part one of the framework.

In the second part of our framework, we define the following notations that are required for the description of the proposed algorithm. Let $\mathbb{H} = \{H_0, H_1, H_2, ...\}$ be a set of network devices (hosts) where $H_i, i > 0$ are basic components of the targeted network such as computers, routers and telephone devices, etc.. And $H_0$ is a host (usually resides outside and is connected to the targeted network through the Internet) used by the attacker to launch the attack. Let $\mathbb{R}$ be a reachability matrices representing the relation between network devices and components which can be established based on the available design information. Let $\mathbb{V} = \{V_1, V_2, ...\}$ be a set of vulnerabilities that are associated with the targeted network. The vulnerability set $\mathbb{V}$ is obtained from available discovery information as described earlier. The vulnerability $V_i$ is represented by a tuple $(source, target, \alpha, C_1, C_2)$ where $target$ is a unique identifier of the host/device in the targeted network where the vulnerability exists. Furthermore, the vulnerability can not be executed unless the precondition $C_1$ of the $target$ is satisfied. The postcondition $C_2$ is a condition that is satisfied after the vulnerability has been exploited. The $source$ is a unique identifier of the host/device inside or outside the targeted network where the attack is originated from. And finally, $\alpha \in \mathcal{V}$ is a vulnerability metric as defined in Section II.A. The chain of exploits consisting of vulnerabilities $V_i$ and $V_j, i \neq j$ is denoted as $V_{i,j} = (source(i), target(j), \alpha(i) \cup \alpha(j), C_1(i), C_2(j))$ which indicates that the exploit chain starts from the $source(i)$ of $V_i$ and finishes in the $target(j)$ of $V_j$ with the corresponding pre- and postconditions and vulnerability metric. The length of this exploit chain is denoted by $|V_{i,j}| = 2$ which indicates that the attacker has to exploit two vulnerabilities $V_i$ and $V_j$ sequentially to achieve its goal. Denote the maximum length (in which we are still interested) of any exploits chain by $d > 0$, and let $G$ be the attacker's goal (objective) in the targeted network.

We consider here the worst case scenario where the attacker knows about all vulnerabilities and will exploit them whenever possible in the targeted network to achieve its goal. Furthermore, we also assume monotonicity to imply that the attacker is sophisticated and will not backtrack during attack. We have enforced the monotonicity mainly because this assumption will help our algorithm find more meaningful critical paths that need to be addressed in the targeted network. The proposed algorithm is summarized as follows.

**Algorithm 1** Algorithm for Assessing Network Vulnerability

```
Require: $G, \mathbb{H}, \mathbb{R}, \mathbb{V}, d > 0$
$\mathbb{O} = \{\}, \mathbb{T} = \{\}; \nu = 0$ // Initialization
Calculate a vulnerability metric $\nu$ of the most critical path
for $V_i \in \mathbb{V}$ do
  if $C_2(i) = G$ then
    for $H_j \in \mathbb{H}$ do
      if $H_j = source(i)$ then
        Add $V_i$ to $\mathbb{O}$
      end if
    end for
  end if
end for
while $\mathbb{O} \neq \varnothing$ do
  for $V_i \in \mathbb{O}$ do
    if $source(i) = H_0$ then
      add $V_i$ to $\mathbb{T}$
    end if
  end for
  if $\mathbb{T} \neq \varnothing$ then
    Find $V_i$ with max $\alpha$ in $\mathbb{T} \rightarrow k : \alpha(k)$ = max
    Remove all $V_{i \neq k}$ having $\alpha(i) \leq \alpha(k)$ from $\mathbb{O}$
    Set $\mathbb{T} = \{\}$
  end if
  if $\exists k$ & $\forall V_{i \neq k} \in \mathbb{O} : \alpha(k) \geq \alpha(i)$ or $|V_k| > d$ then
    Set $\nu = \alpha(k) \rightarrow$ FINISH
  else
    for $V_i \in \mathbb{O}$ $(i \neq k$ iff $\exists k)$ do
      Find $V_j \in \mathbb{V} : target(j) = source(i)$ & $C_1(i) = C_2(j)$
      iff $\exists V_j \rightarrow$ Find all $source(j) \in \mathbb{H}$ satisfying $\mathbb{R}$
      Set $V_i = V_{i,j}$
    end for
  end if
end while
```

## III. Illustration Example and Discussion

In this section we demonstrate the utility of our framework and show its effectiveness via an example of network vulnerability analysis of a real VoIP network. Our testbed (as shown in Fig. 2) consists of 4 Cisco Unified 7960 IP Phones with firmware version 7.9.2; 4 VoIP Softphones installed on 3 desktop computers and a laptop; a SIP proxy and a SIP server for handling calls and a session initiation protocol (SIP) signalling for VoIP. The SIP Proxy runs an open source daemon, and the SIP registrar server runs an open source application called Asterisk (version 1.2.1) [7]. The attacker's computer uses Fedora Core 7 OS operating system and has various VoIP hacking tools installed. We have also installed old version softwares on network components in order to introduce more vulnerabilities into the example.

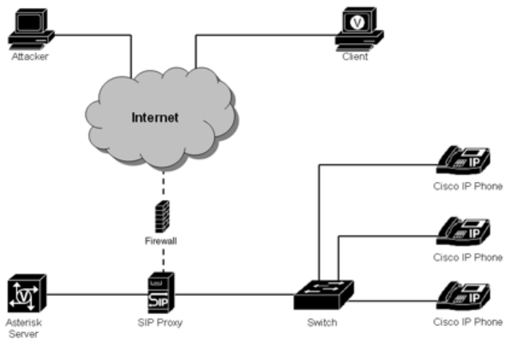

Fig. 2. A testbed for VoIP network.

As described in Section II.B., we first gather the information from the testbed before carrying out the vulnerability analysis using the algorithm proposed in our framework. The design information, namely, the network topology and relations (connectivity) between network components are given in Fig. 2. Furthermore, we use `Nessus` program to scan and collect vulnerabilities associated with all network components of the testbed. The simplified outputs obtained from `Nessus` for selected vulnerabilities that are important for our analysis are summarized in Table I.

We also present in Table I different vulnerabilities using the tuple as described earlier in our framework, namely, $V(source, target, \alpha, C_1, C_2)$. Most of the parameters in the tuple can be determined from the scanning outputs and available design information, however, the vulnerability metric remains to be defined. In this example, we choose the vulnerability metric that relates to the score given by a common vulnerability scoring system (CVSS) [6]. It is because the CVSS score has all the qualities to represent a vulnerability and is readily available from the `Nessus` scanning outputs. To this end, we scale the CVSS score of a given vulnerability to a value between 0 and 1, and assign it to that vulnerability as a vulnerability metric. This vulnerability metric can be interpreted as a measure of the likelihood (i.e., probability) of that particular vulnerability to be exploited. The assigned vulnerability metric is given in the tuples as shown in Table I. Having the vulnerability metric, the $\cup$ operation can then be defined as a multiplication operation of probabilities. Hence, the vulnerability metric of an exploit chain is computed as a product of vulnerability metrics of individual exploits in that chain.

Combining both design and discovery information gives the list of possible vulnerabilities in the testbed and the corresponding relations between network components based on network topology as shown in Tables II and III, respectively. Having collected all the information, we now progress to

TABLE I
SAMPLES FROM SCANNING OUTPUTS OF NESSUS.

| Network Components ($target$) | Nessus Vulnerability Output | Vulnerability |
|---|---|---|
| SIP Proxy ($H_1$) | **CVE Reference: CVE-2005-0449**<br>Description: Allows attacker to bypass `netfiler/iptables` or initiate a denial of service attack.<br>CVSS Base Score: 5.0<br>Pre-Condition: Linux, Kernel 2.6.x<br>Post-Condition: Denial of Service, bypassing firewall | $V$ ($source$, $H_1$, 0.50, Linux-kernel-2.6.1, bypassing_firewall) |
| | **CVE Reference: CVE-2008-1483**<br>Description: Attackers are able to hijack a SSH Session<br>CVSS Base Score: 6.2<br>Pre-Condition: OpenSSH-4.3p2 and earlier<br>Post-Condition: Allows remote connection, Gain access to system. | $V$ ($source$, $H_1$, 0.62, OpenSSH-4.3p2, hijack_session) |
| Asterisk Server ($H_2$) | **CVE Reference: CVE-2008-1483**<br>Description: Attackers are able to hijack a SSH Session<br>CVSS Base Score: 6.2<br>Pre-Condition: OpenSSH-4.3p2 and earlier<br>Post-Condition: Allows remote connection, Gain access to system. | $V$ ($source$, $H_2$, 0.62, OpenSSH-4.3p2, hijack_session) |
| | **CVE Reference: CVE-2008-0095**<br>Description: Allows remote attackers to cause Denial of Service to an Asterisk via a BYE message with an Also header<br>CVSS Base Score: 5.0<br>Pre-Condition: Asterisk, Open Source, 1.4.0 and previous<br>Post-Condition: Denial of Service to Asterisk//VoIP Services | $V$ ($source$, $H_2$, 0.50, Asterisk-1.2.1, DoS) |
| | **CVE Reference: CVE-2007-1306**<br>Description: Allows remote attackers to cause Denial of Service to an Asterisk using a SIP packet without a SIP-version header.<br>CVSS Base Score: 7.8<br>Pre-Condition: Asterisk, Open Source, 1.4.16 and previous<br>Post-Condition: Denial of Service to Asterisk//VoIP Services | $V$ ($source$, $H_2$, 0.78, Asterisk-1.2.1, DoS) |
| Cisco IP Phones ($H_3$) | **CVE Reference: CVE-2007-4459**<br>Description: Allows remote attackers to cause Denial of Service to Cisco IP Phones using a SIP packet with 10 invalid SIP INVITE and OPTIONS.<br>CVSS Base Score: 7.1<br>Pre-Condition: Cisco, VoIP Phone, CP-7960, 8.70 and previous<br>Post-Condition: Denial of Service to Cisco IP Phones/VoIP Services | $V$ ($source$, $H_3$, 0.71, Cisco_IP-7.90, DoS) |
| | **CVE Reference: CVE-2005-2181**<br>Description: Allows remote attackers to bypass verifications with a spoof NOTIFY message.<br>CVSS Base Score: 5.0<br>Pre-Condition: Cisco, VoIP Phone, CP-7960, 8.70 and previous<br>Post-Condition: Spoof Cisco IP Phones/Gain access to VoIP Services | $V$ ($source$, $H_3$, 0.50, Cisco_IP-7.90, spoof_ciscophone) |
| | **CVE Reference: CVE-2008-0531**<br>Description: Allows remote attackers to cause buffer overflow to Cisco IP Phone and allow arbitrary code to be executed remotely via a crafted message<br>CVSS Base Score: 9.3<br>Pre-Condition: Cisco, VoIP Phone, CP-7960, 8.70 and previous<br>Post-Condition: Gain access to Cisco IP Phones/VoIP Services | $V$ ($source$, $H_3$, 0.93, Cisco_IP-7.90, buffer_overflow) |

TABLE II
LIST OF POSSIBLE VULNERABILITIES IN THE VOIP TESTBED AND THE CORRESPONDING RELATIONS.

| Network Components ($source$) | Relations | Possible Vulnerabilities |
|---|---|---|
| Attacker's PC ($H_0$) | $H_0 \rightarrow H_1$ | $V_1$ ($H_0$, $H_1$, 0.50, Linux-kernel-2.6.1, bypassing_firewall)<br>$V_2$ ($H_0$, $H_1$, 0.62, OpenSSH-4.3p2, hijack_session) |
| | $H_0 \rightarrow H_3$ | $V_3$ ($H_0$, $H_3$, 0.71, Cisco_IP-7.90, DoS)<br>$V_4$ ($H_0$, $H_3$, 0.50, Cisco_IP-7.90, spoof_ciscophone)<br>$V_5$ ($H_0$, $H_3$, 0.93, Cisco_IP-7.90, buffer_overflow) |
| SIP Proxy ($H_1$) | $H_1 \rightarrow H_2$ | $V_6$ ($H_1$, $H_2$, 0.62, OpenSSH-4.3p2, hijack_session)<br>$V_7$ ($H_1$, $H_2$, 0.50, Asterisk-1.2.1, DoS)<br>$V_8$ ($H_1$, $H_2$, 0.78, Asterisk-1.2.1, DoS) |
| | $H_1 \rightarrow H_3$ | $V_9$ ($H_1$, $H_3$, 0.71, Cisco_IP-7.90, DoS)<br>$V_{10}$ ($H_1$, $H_3$, 0.50, Cisco_IP-7.90, spoof_ciscophone)<br>$V_{11}$ ($H_1$, $H_3$, 0.93, Cisco_IP-7.90, buffer_overflow) |
| Asterisk Server ($H_2$) | $H_2 \rightarrow H_3$ | $V_{12}$ ($H_2$, $H_3$, 0.71, Cisco_IP-7.90, DoS)<br>$V_{13}$ ($H_2$, $H_3$, 0.50, Cisco_IP-7.90, spoof_ciscophones)<br>$V_{14}$ ($H_2$, $H_3$, 0.93, Cisco_IP-7.90, buffer_overflow) |

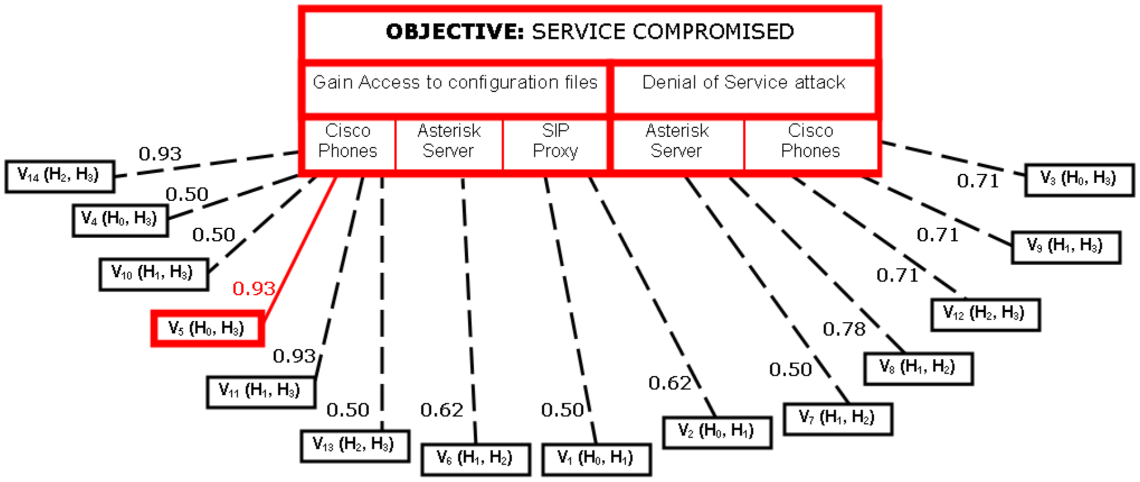

Fig. 3. Vulnerability results for a VoIP testbed

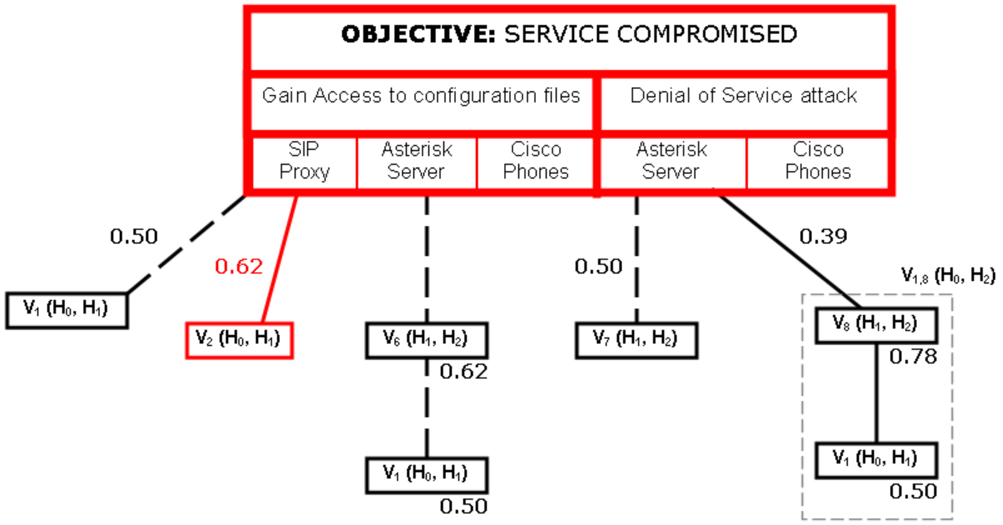

Fig. 4. Vulnerability results for an upgraded VoIP testbed

TABLE III
RELATIONS BETWEEN NETWORK COMPONENTS AND THEIR VULNERABILITIES.

| Relations | $H_0$ | $H_1$ | $H_2$ | $H_3$ |
|---|---|---|---|---|
| $H_0$ | - | $V_1, V_2$ | - | $V_3, V_4, V_5$ |
| $H_1$ | $V_1, V_2$ | - | $V_6, V_7, V_8$ | $V_9, V_{10}, V_{11}$ |
| $H_2$ | - | $V_6, V_7, V_8$ | - | $V_{12}, V_{13}, V_{14}$ |
| $H_3$ | $V_3, V_4, V_5$ | $V_9, V_{10}, V_{11}$ | $V_{12}, V_{13}, V_{14}$ | - |

the second part of the framework and evaluate the network vulnerability using the proposed algorithm. In particular, given that the attacker's objective is to compromise the voice service in this network, we apply the algorithm to find the most critical path and to evaluate the overall vulnerability metric of that path. Results obtained from our algorithm are illustrated in Fig. 3. Just after one iteration, without building up a full-scale attack tree, our algorithm points out that the most critical path is consisting of only one vulnerability $V_5$ (in Table II) with the overall vulnerability metric is equal to 0.93. For a given objective (i.e., compromising voice service) this result indicates that from the network vulnerability analysis point of view the attack that exploits buffer overflow vulnerability in the Cisco IP phone is the most critical one. We also state that the vulnerability of this network is high and is quantified by a value of 0.93 based on a scale from 0 to 1.

To demonstrate further the capability of our framework, let us consider a scenario where the firmware of the Cisco IP phone is upgraded as a result of the above analysis. We then apply the algorithm once again while keeping the same objective. The results are shown in Fig. 4. Observe that after the first iteration the most critical path is associated with $V_8$ vulnerability. Since $V_8$ is a later vulnerability on a chain of exploits, the algorithm will run a second iteration to produce the full exploit chain of $V_1 \rightarrow V_8$. As a result, this chain will now be represented as $V_{1,8}$ vulnerability with the overall vulnerability metric of $0.39$ which is a product of the metrics of $V_1$ and $V_8$ vulnerabilities. After the second iteration $V_2$ now becomes the most critical path. Therefore the vulnerability of this network is now represented by the vulnerability metric of $V_2$ which is $0.62$. This is a significant improvement compared to that of the previous case which reflects the effectiveness of the actions taken (i.e. upgrading the firmware of Cisco IP phones) to improve network security. Note that the scalability of our framework is achieved by examining only the most critical path at any stage of the algorithm. By doing so, we do not follow any exploit chain that is not critical to the security of the network.

## IV. Conclusion

We have developed in this paper a framework for network vulnerability analysis and proposed a scalable algorithm to evaluate the overall network vulnerability without generating a full-scale attack graph. The framework gives a quantitative value representing the level of security achieved in an arbitrary network based on a new defined vulnerability metric and its operations. Using the proposed algorithm we have analyzed a real network offering voice over IP service to demonstrate the effectiveness of our framework in network vulnerability analysis. We have also shown that important insights can be obtained from our analysis to identify the most critical vulnerabilities and to improve the overall security in a network.

## References

[1] P. Ammann, D. Wijesekera and S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proc. of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 217–224.

[2] K. Ingols, R. Lippmann and K. Piwowarski, "Practical Attack Graph Generation for Network Defence," in *Proc. of the 22nd Annual Computer Security Applications Conf.*, Dec. 2006.

[3] R.W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Proc. of the IEEE Symposium on Security and Privacy*, May 2001, pp. 156–165.

[4] O. Sheyner, J. Haines, S. Jha, R. Lippman, and J. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.

[5] T. Zhang, M. Hu, D. Li and L. Sun, "An Effective Method to Generate Attack Graph," in *Proc. of the 4th International Conference on Machine Learning and Cybernetics*, August 2005, pp. 18-21.

[6] Common Vulnerability Scoring System (CVSS), http://www.first.org/cvss/, 2008.

[7] Asterisk, http://www.asterisk.org/, 2008.

[8] NuSMV, http://nusmv.irst.itc.it/, 2008.

[9] Network Mapper (Nmap), http://nmap.org/, 2008.

[10] The Nessus vulnerability scanner, http://www.nessus.org/nessus/, 2008.

[11] Retina Network Security Scanner, http://www.eeye.com/html/Products/Retina/index.html, 2008.

[12] Wireshark, http://www.wireshark.org/, 2008.