12-15-2021

Calculating Common Vulnerability Scoring System's Environmental Metrics Using Context-Aware Network Graphs

Christopher Thomas Enoch
cte6149@rit.edu

R·I·T

Calculating Common Vulnerability Scoring System's Environmental Metrics Using Context-Aware Network Graphs

by

Christopher Thomas Enoch

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Software Engineering

Department of Software Engineering
Golisano College of Computing and Information Sciences

Rochester Institute of Technology
Rochester, NY

December 15, 2021

Committee Approval:

_____

Dr. J Scott Hawker Date

SE Graduate Program Director

_____

Dr. Mehdi Mirakhorli Date

Advisor, Department of Software Engineering

_____

Dr. Mohamed Wiem Mkaouer Date

Committee Member, Department of Software Engineering

_____

Dr. Christian Newman Date

Committee Member, Department of Software Engineering

_____

Dr. Shanchieh Yang Date

Committee Member, Department of Computer Engineering

Abstract

Software vulnerabilities have significant costs associated with them. To aid in the prioritization of vulnerabilities, analysts often utilize Common Vulnerability Scoring System's Base severity scores. However, the Base scores provided from the National Vulnerability Database are subjective and may incorrectly convey the severity of the vulnerability in an organization's network. This thesis proposes a method to statically analyze context-aware network graphs to increase accuracy of CVSS severity scores. Through experimentation of the proposed methodology, it is determined that context-aware network graphs can capture the required metrics to generate modified severity scores. The proposed approach has some accuracy to it, but leaves room for additional network context to further refine Environmental severity scores.

Calculating Common Vulnerability Scoring System's Environmental Metrics Using Context-Aware Network Graphs

Software vulnerabilities are costly problems. For developers and project managers, they can delay a project due date and subsequently drive up the cost of the project. Organizations that identify library or driver vulnerabilities may require major software updates or custom solutions to protect themselves. If they fail to identify or patch them, they can expose entire sections of their network infrastructure to malicious actors, costing them millions of dollars and a reduction in public trust.

To help track vulnerabilities and provide important vulnerability data for developers, project managers and security analysts alike, the National Institute of Standards and Technology (NIST) created a centralized database called the National Vulnerability Database (NVD). Along with vulnerability descriptions, and references to external sources, the NVD utilizes the Common Vulnerability Scoring System (CVSS) to provide severity scores to enhance risk analysis. CVSS is an open-source scoring framework used by the software industry to score vulnerabilities, providing context to various aspects of the vulnerability. The scoring system relies on security professionals to analyze software vulnerabilities and make recommendations for the scores assigned to each of the vulnerability's attributes, based on the descriptions set forth by the framework. The CVSS specification outlines an approach for organizations to add environmental context to a particular vulnerability. For instance, an organization may realize that a firewall prevents a particular vulnerability from accessing the network and an analyst may update the Attack Vector score to reflect that an attacker may need to have physical access to the machine in order to run the exploit. One of the major downsides of this extension, is the inputs are entirely up to the analyst or team of analysts. What one analyst may conclude from the structure of the compromised machine or network, can be different from another analyst.

Consequently, the new generated score becomes very subjective and may miss elements or include too many elements that affect the overall score, and the score may hold different meaning to third party analysts.

The goal of this thesis is to provide an extension to the CVSS Environmental scoring framework that performs static analysis of contextualized networks to generate modified severity scores. The two questions that this body of work seeks to answer are the following:

R1. Can context-aware network graphs capture the required metrics to calculate CVSS Environmental scores?

R2. Can the proposed technique accurately calculate CVSS Environmental scores?

This thesis first defines the models and methods used for capturing network context. The network topology is converted into *connectivity* and *communications* graphs that depict the physical and logical connections between machines. The graphs are used to determine how the physical locations of vulnerabilities impact the overall network. To capture the context of individual nodes, a *security questionnaire* is proposed. The *security questionnaire* provides a method for security analysts to input metadata about each node. Following the description of the framework and examples of the expected outcomes for each CVSS metric, two experiments were conducted using a sample network and three vulnerabilities with different severity classifications. The first experiment randomizes the location of each vulnerability in the network while the second experiment places the vulnerabilities onto a single node and randomizes the *communications* graph. The evaluation of vulnerabilities in a sample network shows that the proposed framework is capable of capturing the required metrics to calculate new CVSS scores. It also shows that introducing network and machine context increases accuracy, however there is room for additional improvement to further increase accuracy.

This thesis is organized as follows. The next section summarizes the related literature. The third section, Measuring CVSS Environmental Scores, describes the details of the network graphs created from network topologies, the methods for capturing additional metadata about network machines, and the proposed framework for calculating Environmental scores. The fourth section, Analyzing CVSS Environmental Scores, provides the evaluation of the proposed approach along with an analysis of the results. The fifth section lists the identified limitations of this thesis. Future work is discussed in the sixth section, and the final section summarizes the conclusions of this thesis.

Related Works

The lack of accuracy in CVSS is not a new problem and is a heavily researched topic. Noting the problems present in CVSS severity scores, many works have been published seeking new methods to improve the severity calculation. This section summarizes existing work that is more closely related to the topic of this thesis.

Fruhwirth and Mannisto (2009) aimed to expand CVSS 2.0 severity calculation accuracy by adding additional context to CVSS scores. They identify that CVSS Base scores, which are used by the NVD and by extension organizations using the NVD to prioritize work on vulnerabilities, lack the contextual data to accurately categorize the vulnerability in question. This can cause organizations to add unneeded costs by potentially providing a *critical* response to a vulnerability that should actually be categorized as *low* severity. Fruhwirth and Mannisto thus proposed methods to estimate the missing data points using models created by Frei et al. (2006), to calculate Temporal, Exploitability and Remediation Level scores. They also conducted an interview of security managers to estimate the Confidentiality, Integrity and Availability of a vulnerability. The information they gathered was then used to recalculate Base scores of various vulnerabilities and their cost, and shown that introducing contextual data can accurately

categorize vulnerabilities. Overall, they showed that the more accurate categorization caused a reduction in cost. R. Wang et al. (2011) expanded on Fruhwirth et al. by proposing improvements to the Exploitability metric by including OS Type and Server Type in the base algorithm. Like Fruhwirth, they identified that the current Environmental and Temporal calculations provide subjective measures that affect the categorization of the vulnerability. The changes they proposed to these metrics improved the CVSS severity score by adding context about the environment the vulnerability lives in and the impact to the organization. These two papers explored methods for improving accuracy of the CVSS framework by adding contextual data about the organization or the machine itself in order to improve accuracy and reduce cost. However, these papers did not consider changes to the CVSS framework to include contextual information about the entire network. This thesis aims to improve Environmental scoring by providing methods to evaluate an organization's network.

The introduction of contextual data, particularly network topologies, to the CVSS framework in order to assist in security analysis has been researched before. Nemes et al. (2019) utilized P-Graphs to analyze the network topology and generate reliability scores based on CVE data within the network. The P-Graph is created from the services found in the network. They found that introducing context about services in the network, they were able to estimate the robustness of the network under scrutiny. S. Wang et al. (2015) introduces a method to rank the types of vulnerabilities based on an analysis of the environmental factors of a network. The method proposed uses log information from the network to analyse the impacts of a vulnerability and rank them. The ranking algorithm was developed to improve the risk analysis process that penetration testers use. They found that their approach was able to dynamically rank vulnerabilities and their CWE's based on the configuration of the network. While these works incorporated the network topology into their methodologies, they did not improve upon the

CVSS score calculation. Nemes et al. utilizes CVSS and P-graphs to generate robustness scores, and S. Wang et al. utilizes network context to rank vulnerabilities for prioritization. The proposed approach in this thesis aims to generate Environmental scores using static analysis of the network topology and additional contextual data about the machines present to directly improve the CVSS framework and categorization of vulnerabilities.

Due to the lack of accuracy in vulnerability prioritization and measuring real world impact of CVE's using CVSS base scores, several works proposed new methods that expand or incorporate CVSS calculations. To help improve the vulnerability prioritization process, Dobrovoljc et al. (2017) introduced a method that combines CVSS base metrics with attacker characteristics. Their method for prioritizing vulnerabilities used CVSS scores and attacker behavior to model the impact of vulnerabilities. They then tested multiple prioritization policies to determine which method performed the best. They found that adding attacker behavior along with CVSS metrics yielded better performance than CVSS on its own. L. Gallon and J. J. Bascou (2011) proposed a method to measure the host and network impacts of a CVE by integrating CVSS information into attack graphs. They developed CVSS attack graphs to calculate the damages caused by an individual against the entire network as opposed to individual nodes. They tested their approach against a sample network and found that by introducing CVSS data from previous steps, they are able to more accurately estimate the impact of an attack. They did note that their approach does not take into account CVSS Temporal and Environmental scores and will need to address these in the future. These works dealt with simulation of attackers in an environment. While simulations can be very useful, using attacker behavior can be complex to understand due to its dynamic nature, so this thesis proposes a static analysis tool for assessing the impacts of vulnerabilities and aims to assist in risk analysis.

At the time of writing, there has been no research performed in generating CVSS Environmental scores by statically analyzing a network topology with contextual data about individual machines and their connections. None of the research that currently exists takes a holistic approach to improving Environmental scores by incorporating network topology. The proposed methodology in this thesis aims to fill that gap.

Measuring CVSS Environmental Scores

The CVSS scoring system aims to provide a standardized calculation of vulnerability severity to aid in risk analysis, while providing an open framework so users can understand scores generated by a third party. The algorithm also allows organizations to contextualize the vulnerability in their own network. Given that the inputs can be arbitrary, this proposed approach aims to provide a standard way of adding context to previously scored vulnerabilities.

There are many different security systems, network configurations and actors that affect how a vulnerability can impact a network. There are also various transmission mechanisms and transport layers that impact the reach of a vulnerability or communication between different machines. To reduce the scope and complexity of the research, the proposed framework is designed to handle the network at a high level, by utilizing network graphs to represent the topology of the network and avoiding the many implementation details of network communication. Handling the high level concepts with simple questions such as “Is the machine connected to the internet?” makes the overall framework simpler to understand, more practical to employ, and can handle a broad range of network designs and configurations. However, the framework is designed such that it can be extended to include more information about the network and become more granular in its score generation. This leaves the door open for future work on many different portions of the framework.

There are a few assumptions made while developing the proposed framework. One assumption is that the attacker can execute any attack with 100% success. Attackers have a wide range of skills and varying degrees of knowledge about the network topology. There is a large body of research pertaining to attacker behavior prediction and simulation. This approach differs from other sources by removing the attacker and focusing strictly on the vulnerability and network configurations. The framework takes a different mindset, by evaluating the influence of network structure and providing an objective calculation to the severity of vulnerabilities in a non-simulated environment. Because of the focus on topology of the network, incorporation of attacker behavior simulations into the scoring framework can be left for future work. Another assumption made is that the source of all attacks comes from outside the network. While it is a vital portion of network security, insider threat introduces many factors external to the physical characteristics to the network, like personnel security levels and physical access to machines and can be left for future work.

The following sections will describe the various inputs required for the proposed framework, and the various algorithms developed to calculate the Modified Environmental metrics of a vulnerability. *Modeling Network Topology Through Context-Aware Network Graph* defines the representation of a computer network in a form that can be consumed by the later algorithms. *Adding Additional Security Context* describes the method for adding additional metadata about machines in the network. And finally, *Approximating CVSS Metrics* introduces the changes necessary to the CVSS Environmental scoring framework to incorporate the network context into the final score.

Modeling Network Topology Through Context Aware Network Graph

The core of this framework begins with the representation of the network. The topology is represented as a network graph which consists of nodes representing devices, and edges defining relationships between those devices.

There are several types of devices that can exist on the network: *routers*, *servers*, *switches*, and *machines*. *Routers* are classified as devices that route traffic through the network. These devices typically have one to many connections to other devices on the same network, representing physical cables, or wireless connections. *Routers* can also provide security protocols that dictate communications between devices that are connected to it. *Switches* represent hardware that connect multiple devices to the same network. While *switches* can include software to manage traffic passing through it, *switches* in the network graph are responsible for simply routing packets to other machines. *Servers* are devices that run remote functionality and contain business critical applications and databases, code repositories, and important business data, to name a few things. *Servers* typically do not allow physical access to their environments, allowing access only through other network connected devices. *Machines* represent any other device that does not fit into the previously defined categories. *Machines* can be anything from desktop computers, to network intrusion devices, to cell phones.

As mentioned previously, edges represent relationships between machines. There are two types of edges that illustrate different relationships. *connectivity edges* portray physical connections between machines, such as LAN cables, WiFi connections, and Bluetooth connections to name a few. Within the network graph, *connection edges* are undirected edges and define the basis of the network. While *connection edges* illustrate physical connections, *communication edges* are directed edges that represent a permission between two machines. If a machine has permission to communicate to another, an edge will exist. *Communication edges*

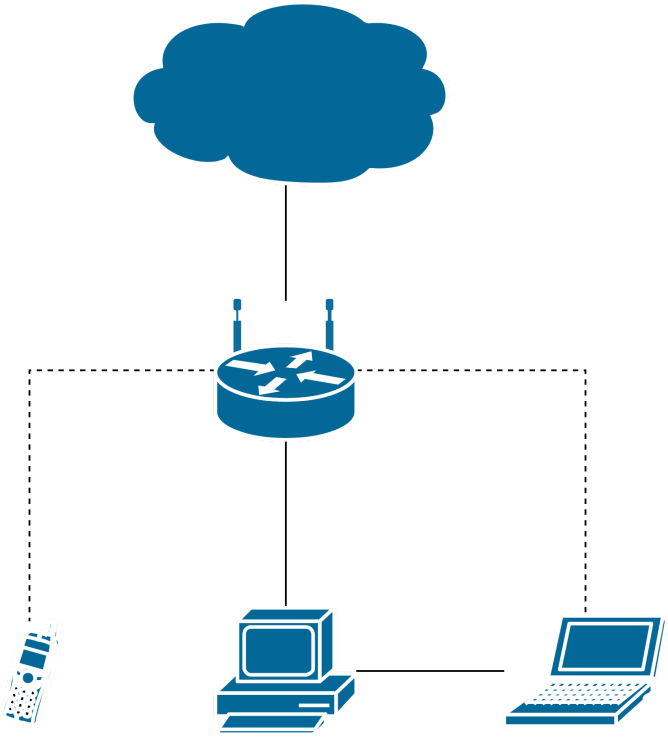
are used to represent access control components of the network, such as firewall permissions. These edges carry two attributes, the minimum permission needed to access the target machine, and the difficulty to bypass authentication schemes and firewalls. The use of these attributes will be addressed in the *Modified Attack Complexity* and *Modified Privileges Required* sections.

Combining these node and edge types, two graph types can be created to fully represent a computer network: the *connectivity graph* and the *communications graph*. The *connectivity graph* is an undirected graph containing the nodes and *connectivity edges* and the *communications graph* is a directed graph containing nodes and *communications edges.* These different types of graphs of the same network allow for algorithms proposed to be specialized for the type of graph under scrutiny. It also allows for cleaner views for an analyst to review.

A simple network containing three devices, a desktop computer, a laptop and a mobile phone is depicted in Figure 1. The desktop computer is directly plugged into the wireless router and laptop via an ethernet cable, and the laptop and mobile phone are connected to the wireless router via a wireless connection. Using this network, the *connectivity* and *communications* graphs shown in Figures 2 and 3 can be created.

Figure 1

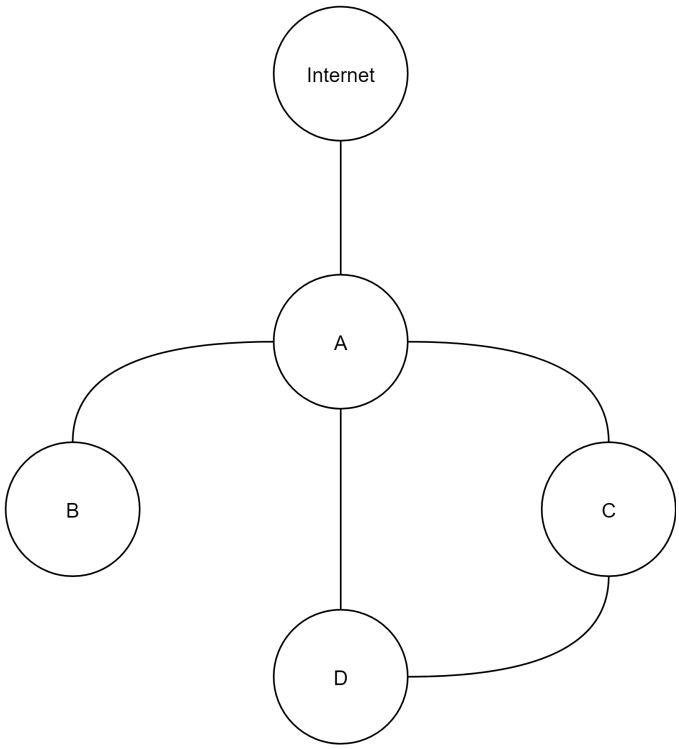
Example Basic Network



The expectant *connectivity graph*, depicted in Figure 2, is created from the example network in Figure 1. Each node in the network represents each of the machines from the source network diagram, while each of the edges represent the physical connections between each node. An edge, for example (A, D), would be interpreted as, A is physically connected to D.

Figure 2

Example Connectivity Graph

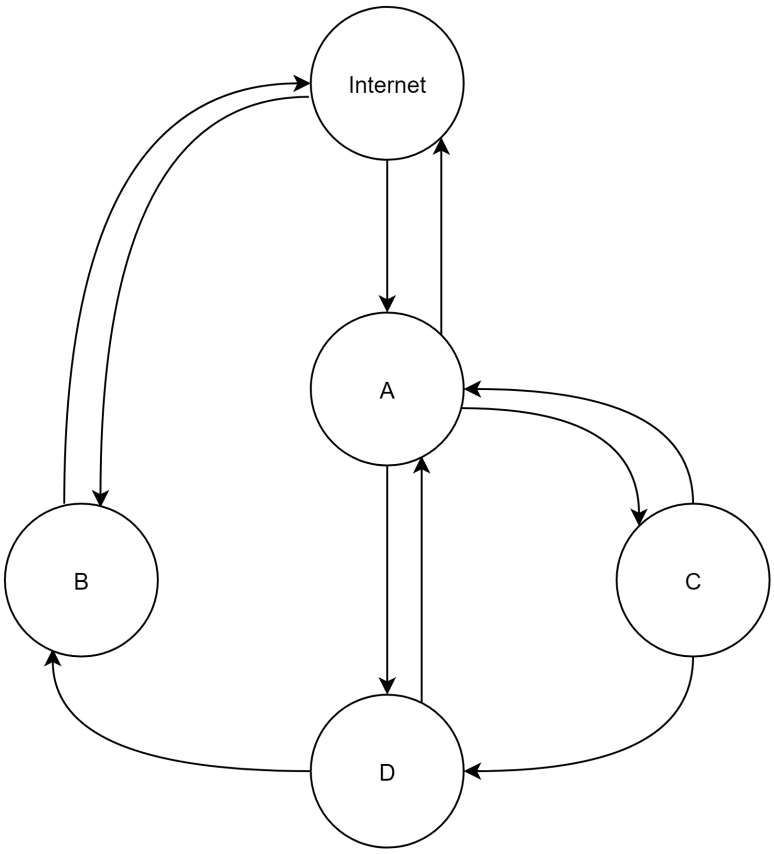


Note. This graph is created from the base network defined in Figure 1.

Figure 3 depicts the resultant *communications graph* after analyzing the logical connections, like firewall configurations, of machines in the example network. An edge in this directed graph can be thought of as an allowed communication from one node to another. So the paths *(A, C)* and (*C, A)* mean that the two machines can communicate freely. Conversely, missing direct paths between the Internet and Node D, would represent a scenario where D is not allowed to communicate to the Internet despite having a direct connection from the *connectivity graph* and vice versa.

Figure 3

Example Permissions Graph



Note. This graph is created from the base network defined in Figure 1.

The *communication edges* shown have additional metadata attached to them to help describe the nature of the connection but have been omitted for simplicity. This additional metadata will be described in more detail later in the paper.

Adding Additional Security Context to Network Devices

When assessing the impact of a vulnerability in a network, it is important to evaluate the security protocols of each machine. To capture the security context of each node, the *security questionnaire* was developed. The questionnaire asks various questions to gauge what data is stored, the importance of any data that's stored, and the protocols used to protect that

data,among others. Each question in the questionnaire can be responded to with a *Yes*, *No* or *Maybe* answer and those answers translate into scores based on the impact that is being evaluated. For this approach, the questionnaire provides the security context to calculate the Confidentiality and Integrity impacts of a vulnerability as outlined by Table 1.

Table 1

Contents of the Security Questionnaire

ID	Question	Impact Type	
		Confidentiality	Integrity
1	Do you store passwords on this Machine?	X	X
2	Are the Passwords stored in Plain Text?	X	X
3	Do you store log files on this Machine?		X
4	Do you store user information?	X	X
5	Do you store personal data in log files?	X	X
6	Do you store financial information?	X	X
7	Is there any information stored about other machines such as IPs or Hostnames?	X	
8	Does this machine contain private keys?	X	X
9	Do you store sensitive business data?	X	
10	Do you allow public users to modify data on the machine?		X

A questionnaire is used to capture the context of a node due to its ease of use and its extensibility. Answers for the questionnaire are stored on each node as metadata for use during impact calculations. How the questionnaire will be utilized to select impact metrics will be discussed further in the Modified Confidentiality and Modified Integrity Impact sections.

Approximating CVSS Metrics

CVSS includes three groupings of metrics: Base, Temporal and Environmental. Base metrics for a vulnerability describes the characteristics of a vulnerability across user environments; Temporal metrics describe the time-based characteristics of a vulnerability,

independent of a user's environment. Finally, Environmental metrics describe the vulnerabilities characteristics when analyzed in a specific environment. Typically, vulnerabilities are given Base scores and the analyst is responsible for defining Temporal and Environmental metrics to further refine the overall CVSS through Modified Base scores.

This framework only focuses on automatically calculating Environmental metrics to provide a vulnerabilities Modified Base score. Temporal scores measure the level of exploitability, availability of fixes, and the confidence in the vulnerability descriptions. These metrics are not impacted by network configurations so they are not included in this proposed framework. The following sections describe each metric used in this approach in detail, and how the approach generates scores for each. The metric scores generated are then plugged into the overall CVSS score calculation to get an updated severity score. Each section provides a definition of the algorithm as well as a trivial example of a network to aid in explanation. The provided examples are created using the base network in Figure 1. and the *connectivity* and *communications graphs* in Figures 2 and 3.

Calculating Modified Attack Vector Via Modified Shortest Path

The *Attack Vector (AV)* describes which vector an attacker can execute the vulnerability in question. The *AV* can be any of the following: *Physica*l, *Loca*l, *Adjacent Network*, and *Network*. An *AV* of *Physical* constitutes an attacker being able to execute a vulnerability only when he has direct access to the component that is vulnerable. An example of this would be executing a cold boot attack to gain access to encryption keys. When a vulnerability has an *AV* of *Local*, the vulnerability can only be exploited locally via a component that has access to vulnerable software. This could be something similar to exploiting a vulnerability in a Python SQL driver to gain root access to the corresponding database. Vulnerabilities that have an *AV* of *Adjacent Network* means that the vulnerability can be exploited on the network where it exists.

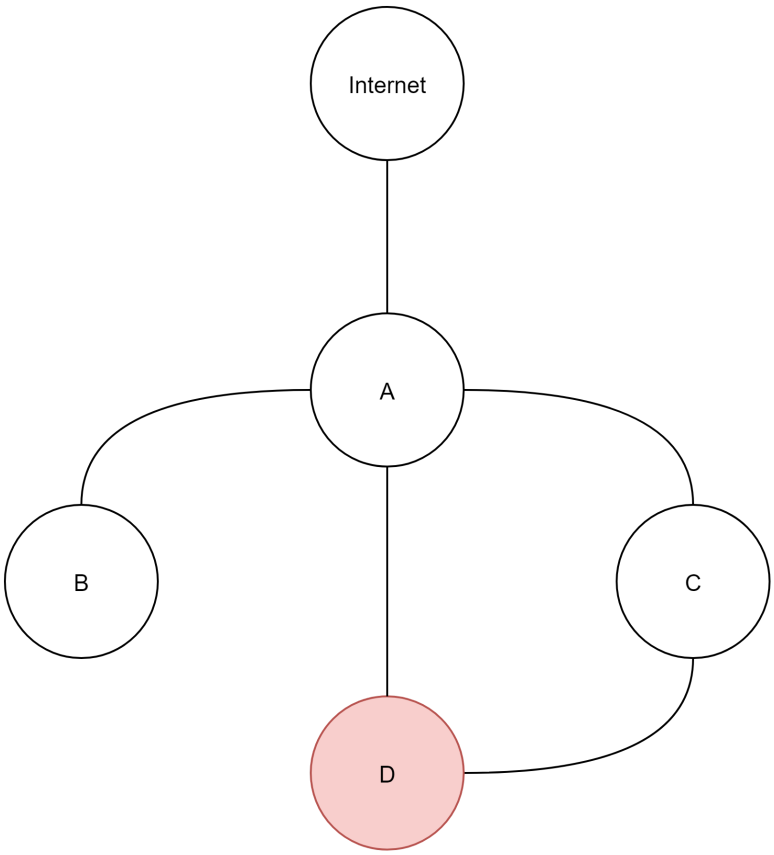
However, the component can only be accessed via the physical network it exists on. Finally, a *Network AV* is assigned when a CVE can be exploited from outside of the network it resides on.

To determine the Modified Attack Vector (*MAV)* of a CVE, the *connectivity graph* must be analyzed to identify a path between the infected node and the Internet. There are several rules that were created to determine the value that the *MAV* can take. The first rule for the scoring attack vector is vulnerabilities will never increase their initial attack vector. For example, a CVSS assigned with *Local* will not assume *Adjacent Network* or *Network*. The assumption is made that when the Base *AV* is assigned, the chosen vector is the absolute worst vector. The algorithm does not take into account the intricacies of the vulnerability in question and as such can not determine if it is possible for the exploit to increase its vector. Along with this rule, CVE's that have an attack vector of *Physical* will keep their respective vector. CVSS defines a *Physical AV* as not bound to the network stack and requiring the attacker to exploit the vulnerability via physical touch. The *MAV* can safely be assigned the initial vector of *Physical* because the original score already considers that the attacker needs to physically be present to execute the vulnerability.

CVE's that have an *AV* of *Network* need to pass through a check to see if they are still accessible from outside of the network. From the affected node, all paths to the internet must be found. If there are valid paths, the paths must then be analyzed for permissions and firewall access. If a node is physically connected to the internet and there is nothing preventing traffic to that node, then the *MAV* remains *Network*. Figure 4. depicts an example graph where the CVSS' Modified Attack Vector remains *Network*. Node D represents a Node that contains a vulnerability with a Base *AV* of *Network*. A path can be drawn from the Internet Node to Node D, either through Node A or Node C, thus assigning an *MAV* of *Network*.

Figure 4

Example of Expected *Network* Modified Attack Vector

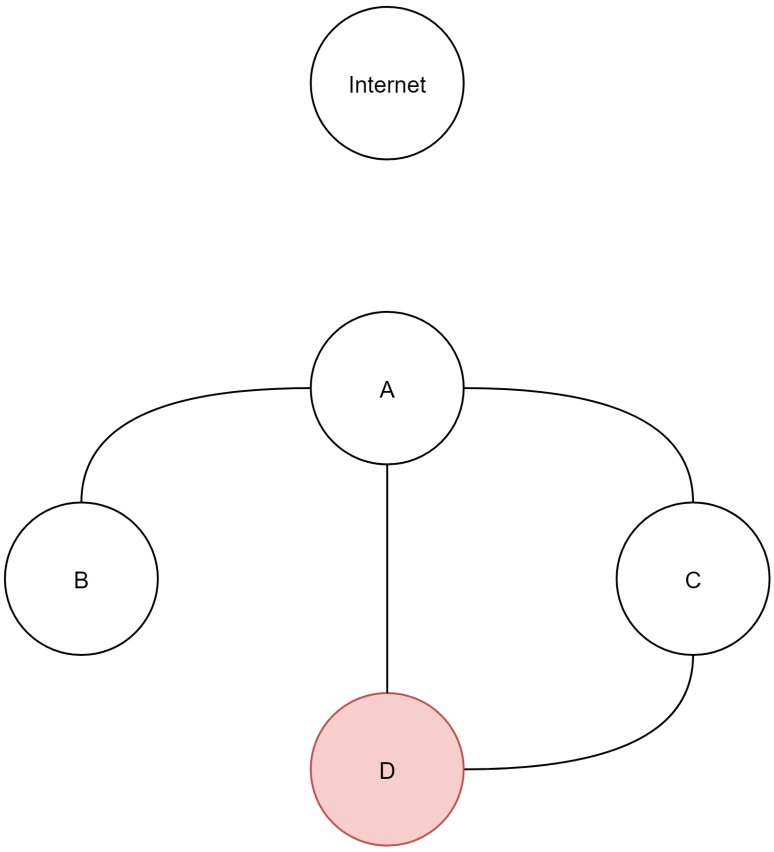


Note. The vulnerability on Node D would be assigned an *MAV* of *Network* due to a direct connection to an *Internet* node.

If there are no paths to the internet, or permissions blocking traffic, then the *AV* gets downgraded to *Adjacent Network*. From here, a valid path to an adjacent node must be made. Figure 5. represents a network that is not connected to the Internet. Since there is no valid path between the Internet and infected Node D, then a check is made to see if Node D has any connected neighbors. Since Node D has two neighbors, Node A and Node C, then the *MAV* of *Adjacent Network* can be assigned.

Figure 5

Example of Expected Adjacent Network Modified Attack Vector

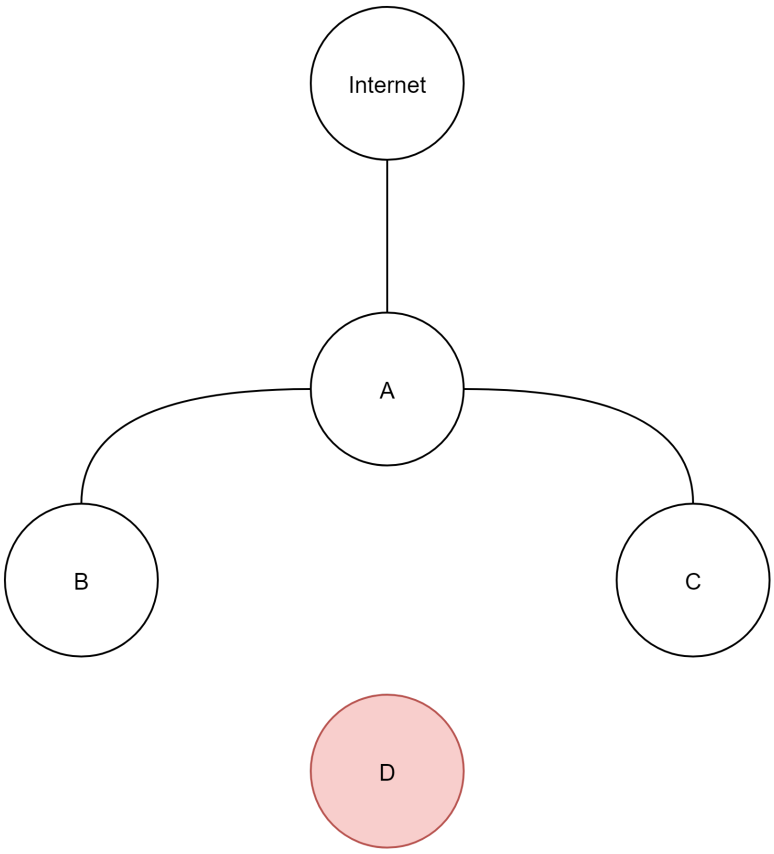


Note. The vulnerability on Node D would be assigned an *MAV* of *Adjacent Network* due to a no direct connection to an *Internet* node but connections to neighbors.

Finally, if the affected node does not have a path to an Internet Node and it does not have any neighbors, then the *MAV* is assigned *Local.* Figure 6. depicts an example of Node D being assigned a *Local MAV.*

Figure 6

Example of Expected Local Modified Attack Vector



Note. The vulnerability on Node D would be assigned an *MAV* of *Local* due to no connections to any nodes

After considering these possibilities, Algorithm 1 was finalized for calculating *Modified Attack Vector*.

Algorithm 1 Modified Attack Vector Calculation

Input: Connectivity Network G, Vulnerable Node n, Attack Vector av
Output: Modified Attack Vector mav

```
accepted ← false, mav ← av
while not accepted do
    if mav is local or physical then
        accepted ← true
    else if mav is adjacent network then
        if n does not have neighbors then
            mav ← local
        else {n has neighbors}
            accepted ← true
        end if
    else {mav is network}
        if G does not have any internet nodes then
            mav ← adjacent network
        else {G is connected to the internet}
            for all internet node i in G do
                if path exists between i and n then
                    accepted ← true
                end if
            end for
        end if
    end if
end while
return mav
```

Measuring Modified Attack Complexity Via Modified Depth-First Search

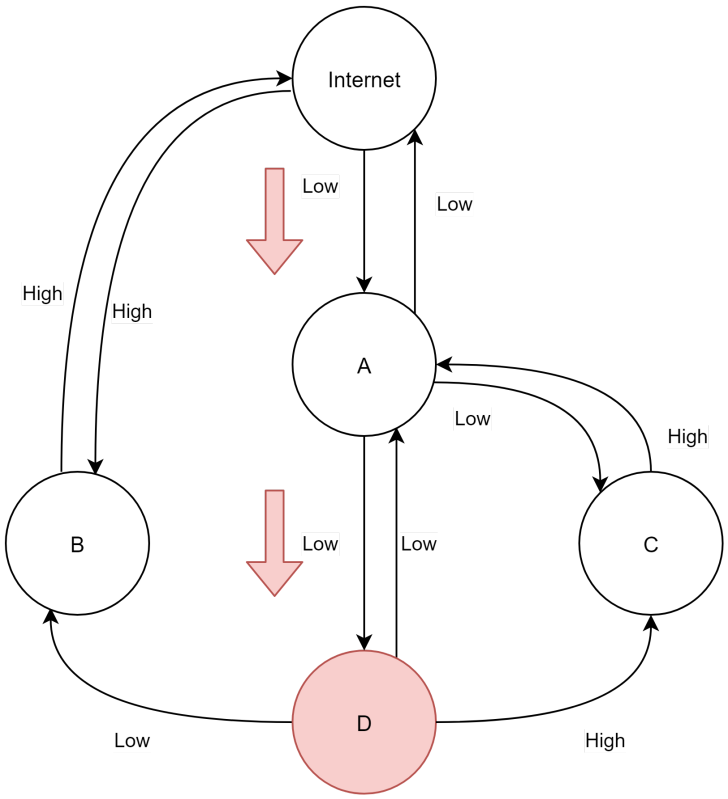
The CVSS specification defines the *Attack Complexity (AC)* as a metric that describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Expanding the modified metric to incorporate the network, *Modified Attack Complexity* (*MAC*) can more specifically relate to the difficulty of bypassing firewalls and bypassing authentication methods to execute a vulnerability. According to the CVSS specification, *AC* can take two values: *Low* and *High*. A *Low AC* means that the exploit can be easily executed and the attacker can expect repeated success when attempting to execute multiple times. If the *AC* is assigned

Low, the overall CVSS score increases. To calculate the *AC* value, the algorithm needs to show there is an attack path from the affected node to the Internet that provides the smallest amount of complexity for the attacker. If that path does not exist then the attacker may need to perform additional steps to perform the exploit. *Complexity* is assigned to *connectivity edges* based on the analysis of the perceived difficulty of accessing a node by analysts. For example, an edge would be assigned a *High complexity* value if the analyst determines that there are several security measures put in place that would cause some significant work to be bypassed, like multi-factor authentication, or encrypted traffic, etc.

An example *connectivity graph* with complexity annotations on each edge is shown in Figure 7. This example would produce an *MAC* of *Low* due to a valid path of *Low* complexity that can be drawn between the Internet and the affected node, D. This path is represented by the arrows in the graph and the path would read: Internet -> A -> C -> D.

Figure 7

Example of Expected Low Attack Complexity

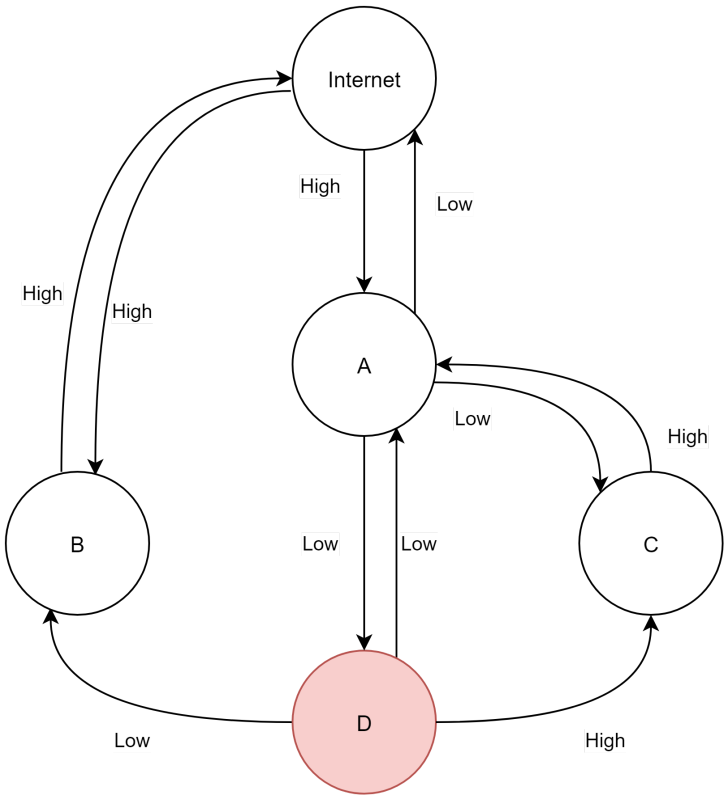


Note. The vulnerability on Node D would be assigned a *MAC* of *Low* due to a direct path with *Low* complexity from an Internet node, denoted by the arrows.

To show an example of an *MAC* of *Low,* the connection between Node C and Node D can be removed. Figure 8 is an example of this removal, showing that a valid path of Low Complexity can not be drawn, and thus the only path of Internet -> A -> D, would result in *High MAC*.

Figure 8

Example of Expected High Modified Attack Complexity



Note. The vulnerability on Node D would be assigned a *MAC* of *High* due to no paths only *Low* complexity

To calculate the *MAC* of the existing CVE, a modified depth-first search algorithm was used. The depth-first algorithm was modified to search for a valid path between the affected node and the Internet using only *Low* complexity edges. As shown in Algorithm 2, the modification made to the algorithm lies in the condition to add the neighbor node to the search list. In order for the algorithm to progress further, the edge complexity of the current node's neighbors must be equal to the value *Low*. In the event a neighbor with an edge complexity of *Low* is found, then the neighbor gets added to the set of nodes to traverse in the following iteration. If the edge complexity of a neighbor is not *Low*, assuming a value of *High*, then the

neighbor is rejected as the purpose of the calculation is to find the shortest path with a *Low* complexity. If a neighbor is found that has a *Low* complexity edge, and the neighbor is an Internet node type, then *Low* is accepted as the *MAC*. If the algorithm can not find a path with *Low* complexity, then *High* is accepted instead.

Algorithm 2 Modified Attack Complexity Calculation

Input: Communication Network G, Internet Node i, Vulnerable Node n, Attack Complexity ac
Output: Modified Attack Complexity mac

```
1: visited ← ∅
2: S ← {i}
3: while not S = ∅ do
4:    d ← first element in S
5:    if d ∉ S then
6:       visited ← visited ∪ {d}
7:       lcn ← ∅
8:       for all edge ∈ d.edges do
9:          if edge.complexity = low then
10:            if edge.target = n then
11:               return low
12:            end if
13:         else {edge.complexity ≠ low}
14:            continue
15:         end if
16:         lcn ← lcn ∪ edge.source
17:      end for
18:      S ← S ∪ (lcn \ visited)
19:   else {d ∈ S}
20:      continue
21:   end if
22: end while
23: return high
```

Calculating Modified Privileges Required Via Modified Depth-First Search

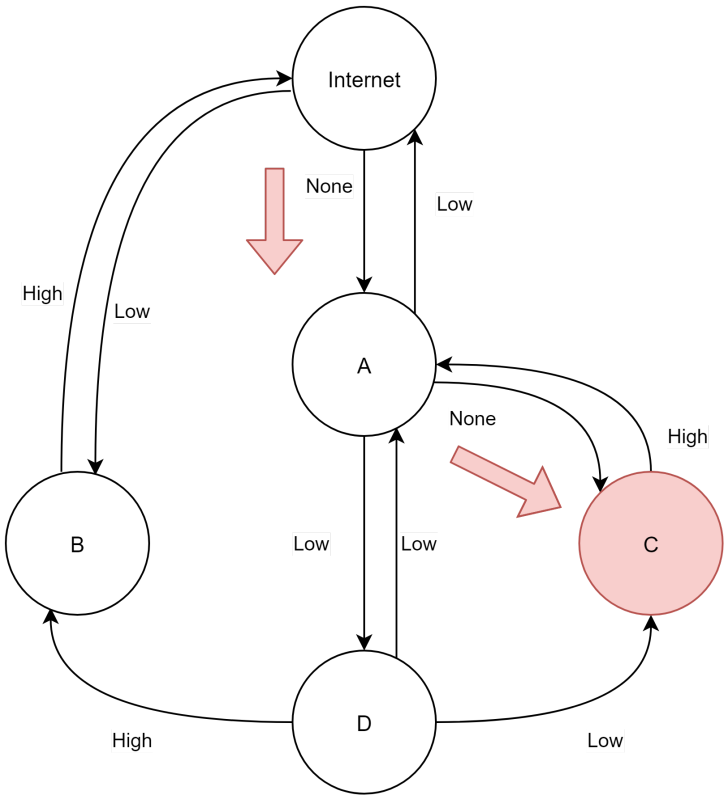
The CVSS framework provides a *Privileges Required (PR)* metric to capture the level of privileges needed before executing the vulnerability. To incorporate the *Modified Privileges Required (MPR)* metric into the proposed approach, the *MPR* algorithm requires an evaluation

of the cumulative privileges required for each path from the afflicted node to the Internet. To fit into the *CVSS* calculation, the proposed algorithm needs to identify if the privileges are either *Low, Medium,* or *High.*

The *MPR* algorithm needs to quantify the amount of privileges needed to not only exploit the vulnerability, but the privileges needed to reach the affected node. Assigning the individual privileges, *Low, Medium* or *High,* to each connection between the nodes, the framework can utilize a path searching algorithm to find a path from the Internet to the target node, representing the malicious actor's movement. The pathing algorithm should attempt to find a path of least privilege and if it can not, should re-run the search including paths with an increased privilege. An example of an expected *MPR* of *None* is depicted in Figure 9. If C is the afflicted Node, then *MPR* is *Low* if the attacker is able to communicate to C with little to no privileges; Internet -> A -> C, in this case. If the privileges along this path were updated so that the attacker could not take this path, then a new path needs to be found.

Figure 9

Example of Expected None Modified Privilege Required

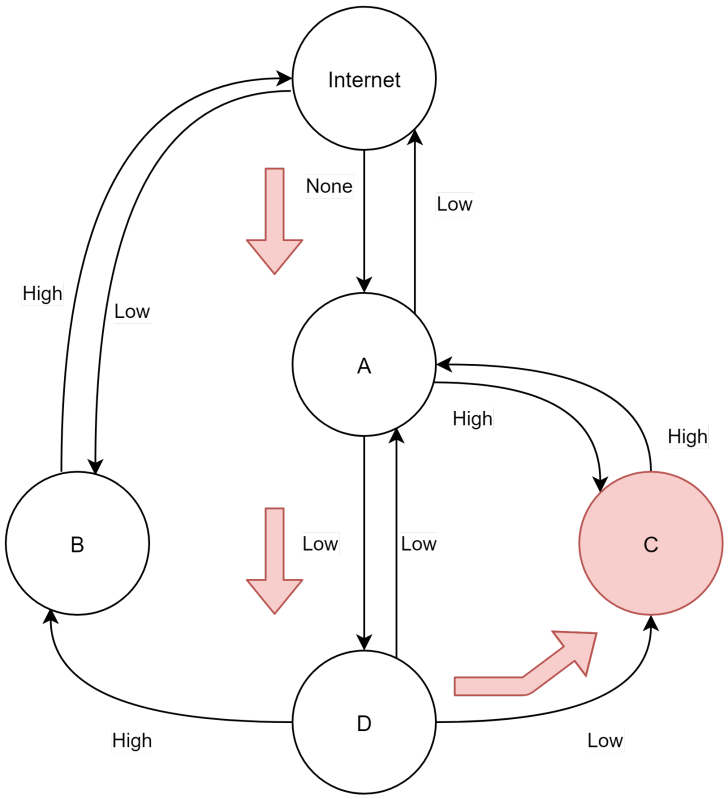


Note. The vulnerability on Node C would be assigned a *MAV* of *None* due to a direct path with *No* privileges needed existing from an Internet node. This path is denoted by the arrows.

In Figure 10, the privileges between A and C are upgraded to *High*, perhaps due to a network administrator restricting login permissions between Nodes A and C. Since there are now no paths to the affected node with *Low* permissions, the algorithm should find a path with *Low* or *Medium* privileges. In the example, a path could be drawn from C -> D -> A -> Internet using a mix of only *Low* and *Medium* privileges resulting in a *Medium MPR*.

Figure 10

Example of Expected Low Modified Privilege Required

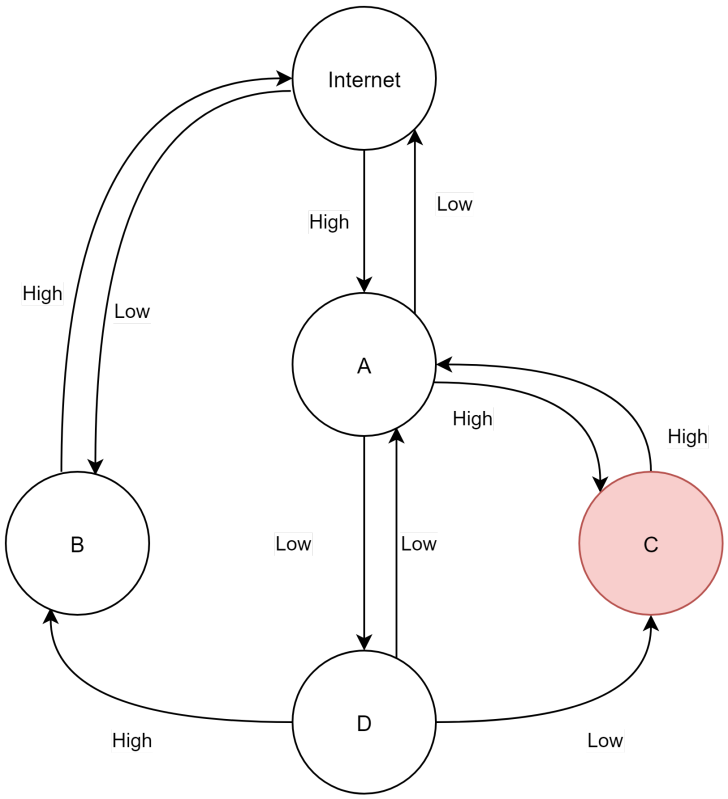


Note. The vulnerability on Node C would be assigned a *MAV* of *Low* because there is at least one path from the Internet that does not require *High* privileges shown by the arrows.

If no path can be found where the malicious actor can reach the affected node with *Low* or *Medium* privileges, then the algorithm should set the *MPR* to *High*, reflecting that the attacker needs administrator privileges to execute the vulnerability. Figure 11. Shows an example of *High MPR*, given that the path from the Internet Node to Node A is updated to require *High* privileges to traverse.

Figure 11

Example of Expected High Modified Privilege Required



Note. The vulnerability on Node C would be assigned a *MAV* of *High* because there are no paths from the Internet that do not require *High* privileges shown by the arrows.

Several algorithms were investigated to accomplish the expectations of the proposed approach. Djikstra's Shortest path was looked at first. Weights could be assigned to each privilege type, *1* for *Low*, *2* for *Medium*, and *3* for *High*. The problem with using Dijkstra's Shortest Path algorithm to find the shortest weighted path from the Internet to the affected node is that it is possible for the shortest weighted path to not be the path with least privilege. It's possible that a much longer path with a higher accumulated weight could exist. To account for this, the final algorithm should test all paths.The resultant algorithms are shown below.

Algorithm 3 shows the top level algorithm for determining the MPR, while Algorithms 4 and 5 describe the searching algorithm for determining *No* and *No and Low* paths to the Internet.

Algorithm 3 Modified Privileges Required Calculation

Input: Communication Network G, Internet Node i, Vulnerable Node n, Privileges Required pr
Output: Modified Privileges Required mpr

1: **if** Path with 'No' Privileges From Internet Exists **then**
2: $mpr \leftarrow none$
3: **else if** Path with 'No' or 'Low' Privileges From Internet Exists **then**
4: $mpr \leftarrow low$
5: **else** {Path with 'High' Privileges or No Path From Internet Exists}
6: $mpr \leftarrow high$
7: **end if**
8: **return** mpr

Algorithm 4 Path with 'No' Privileges From Internet

Input: Communication Network G, Internet Node i, Vulnerable Node n, Privileges Required pr
Output: Path Exists pe

$visited \leftarrow \emptyset$
$S \leftarrow \{i\}$
while not $S = \emptyset$ **do**
  $d \leftarrow$ first element in S
  if not $d \in S$ **then**
    $visited \leftarrow visited \cup \{d\}$
    $lcn \leftarrow \emptyset$
    for all $edge \in d.edges$ **do**
      if $edge.privileges\_needed$ = none **then**
        if $edge.target = n$ **then**
          return true
        else
          $lcn \leftarrow lcn \cup edge.source$
        end if
      else
        continue
      end if
    end for
    $S \leftarrow S \cup (lcn \setminus visited)$
  else
    continue
  end if
end while
return false

Algorithm 5 Path with 'No' or 'Low' Privileges from Internet

Input: Communication Network G, Internet Node i, Vulnerable Node n, Privileges Required pr
Output: Path Exists pe

```
visited ← ∅
S ← {i}
while not S = ∅ do
    d ← first element in S
    if not d ∈ S then
        visited ← visited ∪ {d}
        lcn ← ∅
        for all edge ∈ d.edges do
            if edge.privileges_needed != high then
                if edge.target = n then
                    return true
                else
                    lcn ← lcn ∪ edge.source
                end if
            else
                continue
            end if
        end for
        S ← S ∪ (lcn \ visited)
    else
        continue
    end if
end while
return false
```

Calculating Modified Confidentiality Impact and Modified Integrity Using Eigenvector Centrality

CVSS introduces a *Confidentiality Impact (CI)* and *Integrity Impact* (*II*) to measure the impacts of the vulnerability after it has been successfully exploited. *Confidentiality* and *Integrity* impact scores describe the impact of the vulnerability in question based on the importance of the data on the machine to the organizations.

To accurately measure the impacts, the proposed algorithm should consider the importance of the node relative to the rest of the network. Since network security components

can influence the extent at which a malicious actor can access systems with important information, the calculation should use the *Communications* graph. The algorithm needs to show that network vulnerabilities that originate on machines with high importance, have a high impact score. If the vulnerability does not exist on a machine with high importance, then the impact score should reflect its closeness to important machines. The decision to score impact based on the closeness of important machines was made to represent the ability of an attacker to compromise the affected machine, and utilize that node as an entry point into the organization's network. To measure these two scenarios, the Eigenvector Centrality algorithm can be used to measure the importance of nodes relative to their neighbors. This centrality algorithm can determine the impact of an actor compromising the infected node and jumping to neighbor nodes to access important data.

In order for Eigenvector Centrality to be calculated, each node has to be assigned a weight. The security questionnaire previously defined can be used to generate Confidentiality and Integrity weights for each node. This is done by first calculating the sum of questions for each metric, and assigning them to each inbound edge as a weighted value. After calculating Eigenvector Centrality for both confidentiality and integrity, the values are normalized to be between zero and one. To convert the Eigenvector scores into one of *None*, *Low* and *High* impacts, provided by the *CVSS* specification, each metric is assigned equal slices between zero and 0:

- scores between zero and one-third inclusive are assigned *None;*
- scores between one-third exclusive and two-thirds inclusive are assigned *Low;*
- scores between two-thirds inclusive and one are assigned *High;*

The full algorithms for both *MCI* and *MII* are shown in Algorithms 6 and 7, respectively.

Algorithm 6 Modified Confidentiality Calculation

Input: Communication Network G, Vulnerable Node n, Confidentiality Impact ci
Output: Modified Confidentiality Impact mci

$S \leftarrow G$
$score \leftarrow$ eigenvector centrality(G, n)
$normalized \leftarrow$ normalize($score$)
if $normalized < 1/3$ **then**
  return none
else if $normalized \geq 1/3$ and $normalized < 2/3$ **then**
  return low
else $\{normalized \geq 2/3\}$
  return high
end if

Algorithm 7 Modified Integrity Calculation

Input: Communication Network G, Vulnerable Node n, Integrity Impact ii
Output: Modified Integrity Impact mii

$S \leftarrow G$
$score \leftarrow$ eigenvector centrality(G, n)
$normalized \leftarrow$ normalize($score$)
if $normalized < 1/3$ **then**
  return none
else if $normalized \geq 1/3$ and $normalized < 2/3$ **then**
  return low
else $\{normalized \geq 2/3\}$
  return high
end if

Calculating Modified Availability Impact

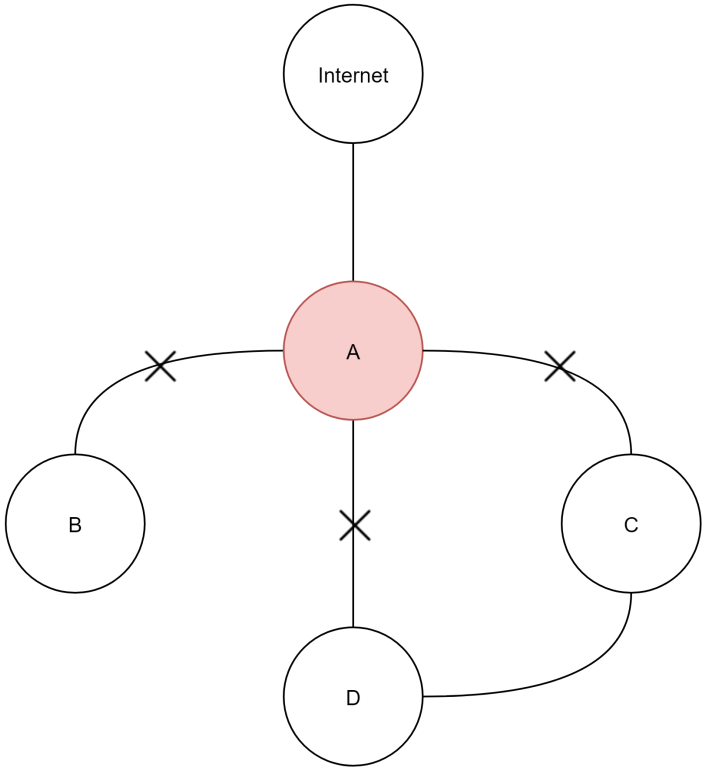
It is important that machines in a network are highly available in order for users to access services and data that are stored in the network. The CVSS framework quantifies the impact of the vulnerability in the context of the resource that the vulnerability exists in. The *Availability Impact (AI)* is defined by the CVSS specification as the impact of the vulnerability on

the component's network bandwidth, processor cycles or disk space. In order to determine *Modified Availability (MAI)* impact, the proposed approach needs to expand the CVSS definition of *AI* to include the impact on the resources that exist in the entire network, not just the availability of the impacted node.

The calculation should start by showing that there is a *High* impact if the vulnerability can make large portions of the network unavailable. An example of *High MAI* by a vulnerability in the example network, is shown in Figure 12. In this scenario, removal of the affected Node, A, would result in the entire network going down. No node would be able to connect to the Internet until Node A is back online.

Figure 12

Example of Expected High Availability Impact

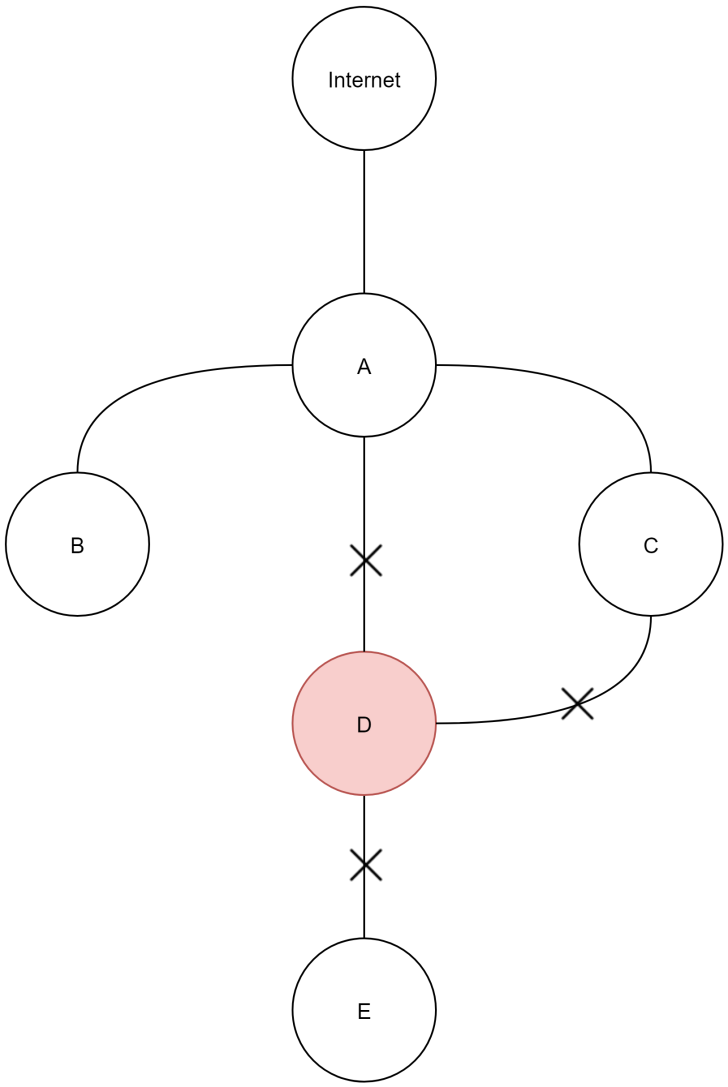


Note. The expected *MAI* in this example would be *High* due to the fact that removing the edges connected to Node A, would cause Nodes B, C, and D to be disconnected from the Internet.

To show a vulnerability has a *Low* impact, the approach must show that some of the network is no longer connected to the Internet, but not all of the machines are affected. To demonstrate a *Low MAI*, the vulnerability can be moved to Node D, and an additional machine can be added, Node E. To contextualize this change, Node E can be seen as a Virtual Machine that exists on Node D. In this configuration, removal of the infected node results in 2 machines no longer connected to the Internet resulting in an expected *Low* impact.

Figure 13

Example of Expected Low Availability Impact



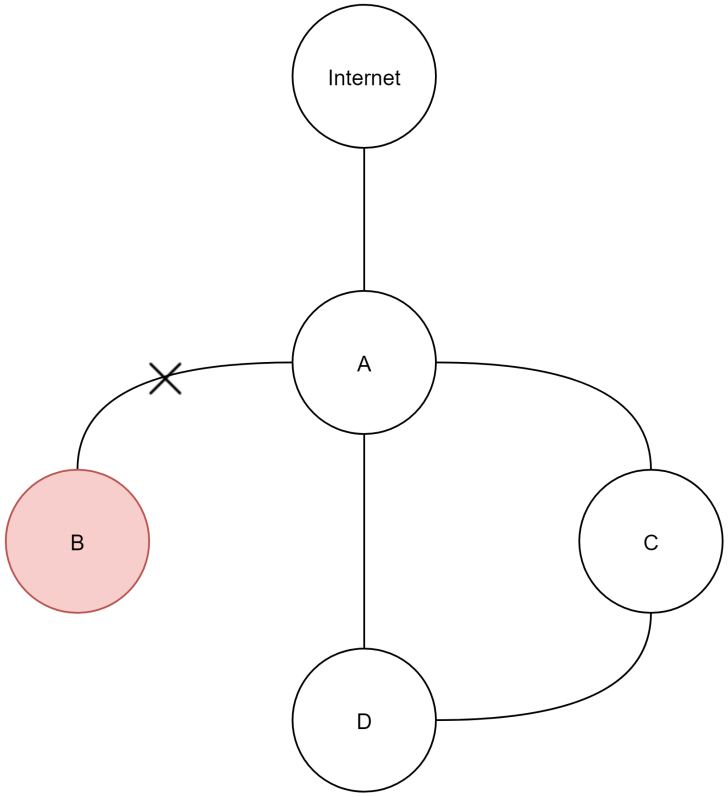
Note. The expected *MAI* in this example would be *Low* due to the fact that removing the edges connected to Node D, would cause only Node D and E to be disconnected from the Internet. Nodes A, B and C are still connected.

Finally, the approach should determine a *None MAI* the affected node is impacted or if only a small portion of the network is affected by the removal of the impacted node. An example

of a vulnerability with a *None* impact is shown in Figure 14. In this case, the expected *MAI* is *None* due to the fact that removing Node B from the network only affects itself.

Figure 14

Example of Expected None Availability Impact



Note. This example would yield a *MAI* of *None* due to the fact that removing the edges connected to Node B would not disconnect any other Node from the Internet.

There were several considerations when determining the best algorithm to cover the above outcomes. The first attempt was to utilize the Betweenness Centrality algorithm to calculate the importance of the node in a network. Betweenness provides a measure for the number of shortest paths that pass through the target node. The problem with measuring availability given the shortest paths passing through a machine in a network is that computer networking does not care about shortest path communications between nodes. So long as there

exists a path between two machines, they will be connected. Betweenness will work if the infected node is a bridge node, a node that has the only connection between two sections of the network, such as a router or switch. However, If a node is removed, there could be an alternate longer path to the Internet. It is entirely possible for a path to exist to the target node if the infected node is out of commission, thus not impacting the overall availability of the nodes in the network. Taking the above into consideration, Algorithm 8 was chosen to determine the *MAI* of a vulnerability. Firstly, a count of all the nodes in the *connectivity graph* is taken. The node in question is then removed from the graph and a count of remaining nodes is collected. The remaining nodes are all nodes that still have a path to the Internet. Finally, a percentage of remaining nodes is calculated and the *MAI* value determined. If more than two thirds of the nodes are still connected, then the vulnerability has an *MAI* of *None*. If more than one third of nodes are still connected, then the vulnerability has some impact, requiring the *MAI* to be *Low*. If there are less than one third of nodes remaining, then the vulnerability has a catastrophic impact on availability and must be assigned a *High MAI*.

Algorithm 8 Modified Availability Calculation

Input: Connectivity Network G, Vulnerable Node n, Availability Impact ai
Output: Modified Availability Impact mai

$S \leftarrow G$
$total\_nodes \leftarrow$ count(G)
$G \leftarrow \{G$ - $n\}$
$rem\_nodes \leftarrow source\_nodes$ in bfs(G, i)
$score \leftarrow \{total\_nodes$ - $remaining\_nodes\}$ / $total\_nodes$
if $score <$ 1/3 **then**
    return none
else if $score \geq$ 1/3 and $score <$ 2/3 **then**
    return low
else $\{score \geq$ 2/3$\}$
    return high
end if

Exclusion of Modified User Interaction and Scope Metrics

The CVSS framework provides two additional inputs for the security expert to consider: *User Interaction* (*UI*) and *Scope (S)*.*UI* captures whether or not some user needs to perform an action in order to successfully trigger the vulnerability. An example of *UI* being required, given by the *CVSS Specification,* would be the requirement of a user to run an installer.*Scope* change occurs when an attacker can access resources outside the control sphere of the application. An example of this would be breaking out of a Tomcat container and modifying files the container normally does not have access to. The proposed approach focuses solely on severity metrics that are impacted by the network configuration and as such, does not explore how the vulnerability is exploited, or whether or not the machine configuration can impact the change in *Scope*. The base scores of the vulnerability are not modified.

Analyzing CVSS Environmental Scores of an Example Network

To validate the approach, two experiments were devised. The goal of these experiments is to show that introducing context about a network would have a meaningful impact on the severity of a given vulnerability. The next section1 introduces the sample network that will be the subject of experimentation. The following section describes the vulnerabilities selected for experimentation and the rationale behind their selection. How the questionnaires were filled out for each of the nodes in the network is then described. The *Experimentation* section sets up the two experiments conducted and the rationale for their selection. The tool developed for executing the experiments and gathering their results as well as the inputs to the tool are introduced in the *Tooling* section. The final section analyzes the results from experimentation and discusses how the findings answer the two research questions.

Sample Network Topology

The first step in designing the experiments is to develop the network topology. The topology for this case study starts with the network used in the S. Wang et al. paper, shown in Figure 15. The network used contained 8 computers and 3 routers, split into 2 subnets. The *connectivity* graph created from this network is shown in Figure 16.

Figure 15

Diagram of Wang Paper Network

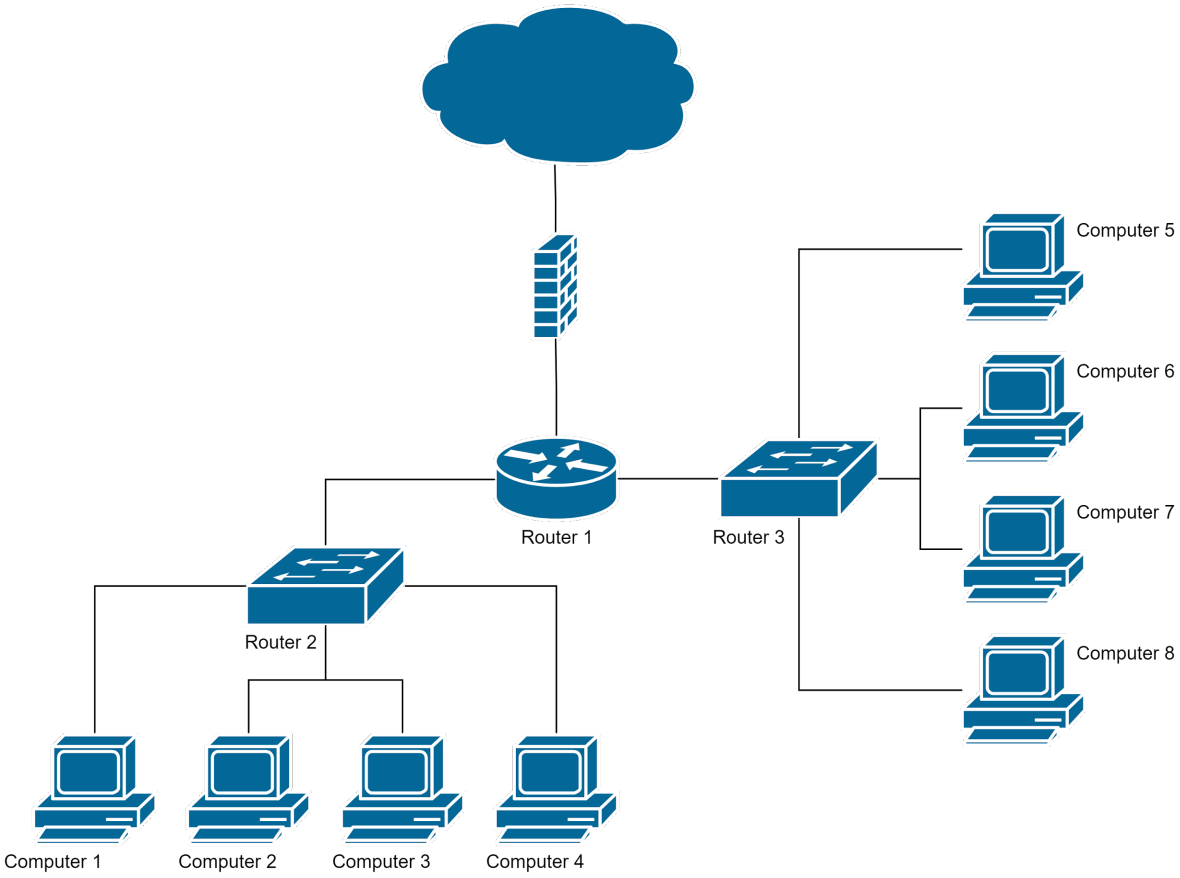
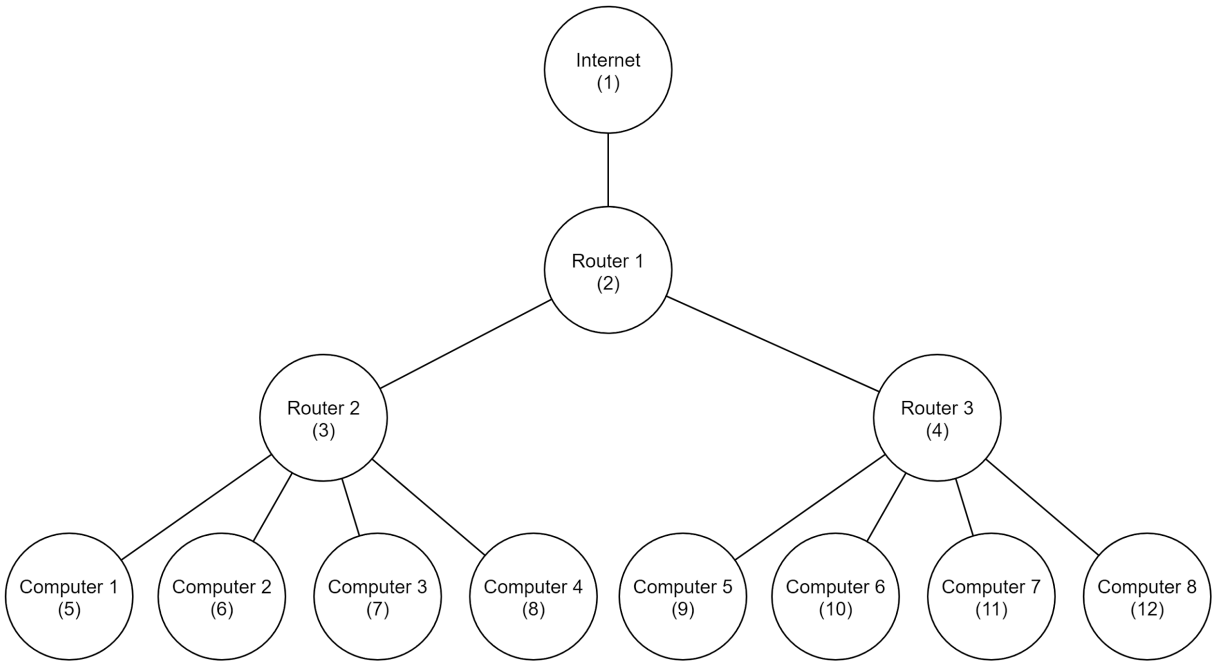


Figure 16

Connectivity Graph of Wang Graph



Note. Node ids used in evaluation are displayed with parentheses

The S. Wang et al. paper lacks the information required to create the *communications graph*. As a result, the graph needs to be created from scratch. To limit bias in the generation process, the edges between all of the nodes are randomly generated using the randomization process created for Experiment 2, described below. The final *communications graph* adjacency list is recorded in Table A1.

Sample Vulnerabilities

Three CVE's were selected for the proposed experiments. Each CVE selected represents a different level of severity: *Low s*everity, *Medium s*everity, and *Critical* severity. A CVE chosen from each category of severity would better demonstrate how the network configuration can affect the various severities of the selected CVE's. It was decided that the CVE's should be affecting the PostgreSQL service.

The first CVE is CVE-2019-10209. This Low Severity CVE, when exploited, can allow a user to read the server memory. While this vulnerability is exploitable over the network, it is complex and requires administrative privileges to execute. The CVSS Base metrics and the numeric score is shown in Table 1.

Table 1

CVE-2019-10209 CVSS Base Metrics

AV	AC	PR	UI	S	C	I	A	Score
Network	High	High	None	Unchanged	Low	None	None	2.2

The second CVE selected for this experiment is CVE-2019-10129. This vulnerability has a CVSS 3.1 severity score of 6.5, and is categorized as a *Medium* severity vulnerability. The vulnerability stems from the ability for an attacker to craft an insert into a partitioned table, leading to arbitrary bytes of the server's memory to be read. Unlike the previous CVE, this CVE has a lower complexity and a lower required privileges contributing to the increased score. Also contributing to the severity score, is the *High* Confidentiality Impact. The CVSS metrics are broken down in Table 2.

Table 2

CVE-2019-10129 CVSS Base Metrics

AV	AC	PR	UI	S	C	I	A	Score
Network	Low	Low	None	Unchanged	High	None	None	6.5

The final CVE chosen is the Critically severe CVE-2018-16850. This CVE, when exploited, allows the attacker to execute arbitrary SQL statements as a superuser. This CVE is

assigned a high CVSS base score due to it's low complexity, low required privileges, and high impacts to the affected component, as shown in Table 3.

Table 3

CVE-2019-16850 CVSS Base Metrics

AV	AC	PR	UI	S	C	I	A	Score
Network	Low	None	None	Unchanged	High	High	High	9.8

These CVE's were chosen for several reasons. The first reason is due to the S. Wang et al. paper specifically mentions that they place SQL server software into their network for testing their approach. This CVE was also chosen due to the widely used software it affects. PostgreSQL is used in various environments, allowing the randomization of the CVE location to seem more realistic. For example, placing a PSQL vulnerability on a network router could be justified due to the router operating system using or allowing the installation of a PSQL server. On the contrary, using a vulnerability that affects Microsoft Word, would not make sense if it's placed on the same router. A final reason is the severity score being relatively close to the center of the scoring categories. With a CVSS score of 6.5, there is room for the score to vary due to the network configuration. If the severity score is too low or too high, there is a greater chance that the location of the vulnerability will not produce enough variation to validate the approach.

Figure 17 shows the JSON structure created to capture the CVE information for use in experimentation.

Figure 17

JSON Input of CVE for Evaluation


```
{
  "cve": {
    "name": "CVE-2018-16850",
    "cvss": {
      "attack_vector": "Network",
      "attack_complexity": "Low",
      "privileges_required": "None",
      "user_interaction": "None",
      "scope": "Unchanged",
      "confidentiality": "High",
      "integrity": "High",
      "availability": "High",
      "exploit_code_maturity": "Not Defined",
      "remediation_level": "Not Defined",
      "report_confidence": "Not Defined"
    }
  }
}
```


Node Questionnaires

The network topology used did not include any contextual clues to create a network diagram similar to the Wang paper. Because of this, the contextual data required for the questionnaire needed to be arbitrarily generated.

To limit any bias from manually entering data, a randomized approach was considered. The idea would be to select each node, and randomly select answers to the questionnaire. The downside of this approach is the difficulty of generating random answers that seemed realistic to the node in question. For example, the question asking "*Do you store financial information?"* with an answer of *Yes* would make sense for a server that stores transaction data, but would not make sense for a router in a subnet. Using answers like this could generate data that may not

be representative of a real network. Therefore, It was decided to manually answer the questionnaire for each node, adding variation to the responses to make them seem more realistic. While this approach adds potential for bias, the realism outways the effect of the potential bias. Below, Figure 18, shows an example of the resulting JSON structure for Node 10 in the network.

Figure 18

Example of Questionnaire Answers for Node 10


```
{
  "id": 10,
  "name": "192.168.3.182",
  "type": "MACHINE",
  "connected_to": [],
  "communicates_to": [...],
  "questionnaire_responses": {
    "1": "YES",
    "2": "MAYBE",
    "3": "YES",
    "4": "YES",
    "5": "NO",
    "6": "NO",
    "7": "YES",
    "8": "YES",
    "9": "NO",
    "10": "NO"
  }
},
```


After answering the questionnaire for each of the nodes, the following weights were calculated for Confidentiality and Integrity and organized in Table 4.

Table 4

List of Confidentiality and Integrity scores

Node #	Confidentiality Weight	Integrity Weight
2	17	17
3	17	17
4	17	17
5	13	15
6	16	18
7	8	10
8	15	12
9	12	8
10	17	17
11	10	14
12	16	16

Experimentation

Using the network provided by the Wang paper, we can run analyses on various mutations of the network to validate the approach. For this case study, an analysis will be performed on two mutation groups: mutation on the location of the CVE, and mutation on the communications between machines. Each mutation type is run 5 times to allow for sufficient randomization of the CVE location or connectivity graph, respectively. The mutation of the CVE location is performed as a simple permutation. Every run randomly selects a number between 1 and n number of nodes, and assigns the pre-defined cve to that node. Communication mutation occurs using the Erdos-Renyi model for random graph generation.

Mutation of CVE Location in the Network

The first set of tests focused on changing the initial location of a provided CVE to different nodes. Performing this test focuses on the impact of a CVE on a network depending upon the location and configuration of the node it resides on. The expectation of this test is that CVE's farther away from the Internet, with less important data will result in a downgrading of the overall severity, while CVE's closer to the internet, with more important data will result in an upgrade in severity.

Mutation of Network Communications

The second set of tests focused on randomizing the communications between nodes. These tests keep the CVE on a single node, and do not change the physical connections between nodes. This test focuses on how adjusting the logical connections between nodes affects the overall score. The expectation is that the more barriers in place that lengthens the logical distance to the Internet, the lower the updated CVSS score will become. Changing the communications between nodes simulates changing firewall permissions, or authentication schemes to allow access to the adjacent node.

Tooling

To facilitate the process, the approach was implemented as a series of scripts. Python was chosen as the programming language of choice. The scripts utilize the networkx graph network library to generate the connectivity and communication graphs, as well as some of the utility functions, such as the centrality functions, to calculate the updated CVSS. The scripts parse a JSON file that describes the nodes in the network as outlined by the methodology. A portion of the final JSON files used for experimentation is shown in Figure 19.

Figure 19

Sample of Node Definition for Random CVE Test


```
{
  "meta": {
    "name": "Random CVE Template Network",
    "description": "Network found in Wang paper",
    "version": "1.0.0",
    "confidentiality_requirement": "High",
    "integrity_requirement": "High",
    "availability_requirement": "High"
  },
  "nodes": [
    {"id": 1...},
    {
      "id": 2,
      "name": "Network Router",
      "type": "ROUTER",
      "connected_to": [ 3, 4 ],
      "communicates_to": [
        {
          "id": 4,
          "complexity": "LOW",
          "privilege_needed": "NONE"
        },
        {"id": 8...},
        {"id": 9...},
        {"id": 10...}
      ],
      "questionnaire_responses": {
        "1": "YES",
        "2": "MAYBE",
        "3": "YES",
        "4": "YES",
        "5": "NO",
        "6": "NO",
        "7": "YES",
        "8": "YES",
        "9": "NO",
```


Note. Data truncated for brevity

The resulting connectivity and communication graphs are passed to the calculator, along with the CVE data, which updates each CVSS metric. Once processing is complete, the main script outputs each CVE's initial score, along with each metrics base score, and outputs the updated score generated from the calculation which is shown in Figure 20. The code for the created tool is publically available on Github: https://github.com/cte6149/cvss-updater.

Figure 20

Sample Output of Created Tool

```
Updated CVSS for Node: 2
Base Score: 9.8
Environmental Score: 8.8

==Full Diff==
attack_vector       |   AttackVector.NETWORK -> AttackVector.NETWORK
attack_complexity   |   AttackComplexity.LOW -> AttackComplexity.LOW
privileges_required |   PrivilegeRequired.NONE -> PrivilegeRequired.LOW
user_interaction    |   UserInteraction.NONE -> UserInteraction.NONE
scope           |   Scope.UNCHANGED -> Scope.UNCHANGED
confidentiality     |   Impact.HIGH -> Impact.HIGH
integrity       |   Impact.HIGH -> Impact.HIGH
availability        |   Impact.HIGH -> Impact.HIGH

++Diff++
privileges_required |   PrivilegeRequired.NONE -> PrivilegeRequired.LOW
```

Findings

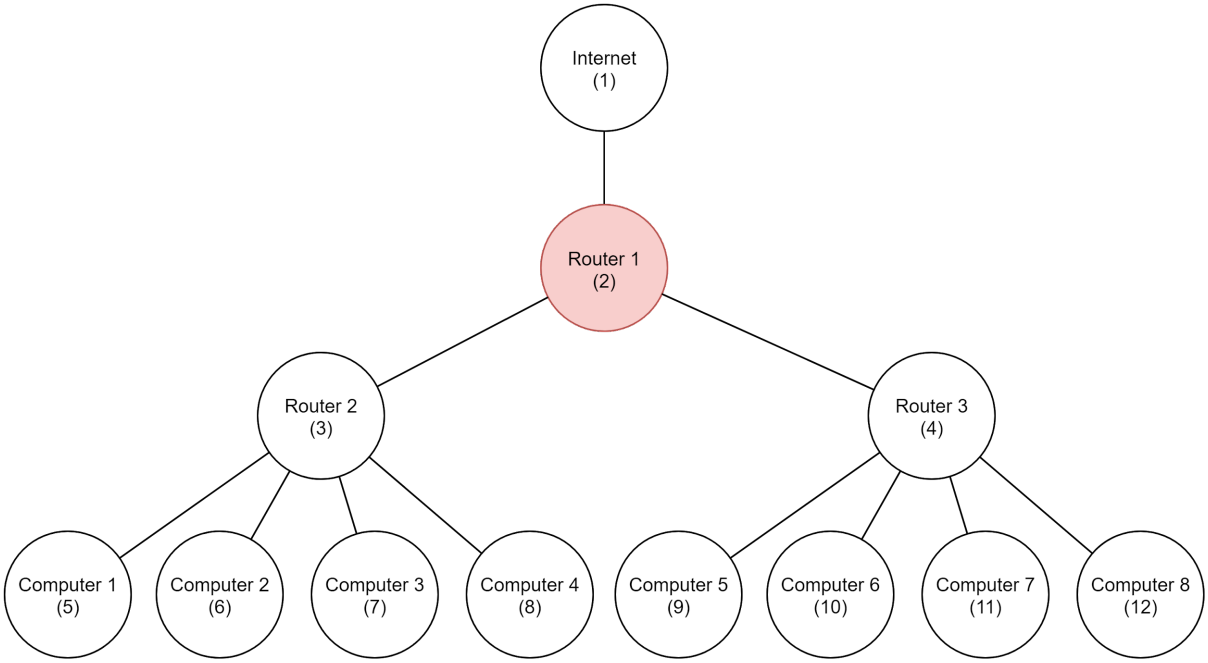
Experiment 1: Mutation of CVE Location in the Network

Experiment 1 was run five times, each run randomizing the location of the CVE in the network. The set of runs for this experiment were assigned the seed, *4225731510159883032*, for repeatability. The first run places all three vulnerabilities onto Node 2 and analyzes the changes in each modified metric, and the final modified score. The *connectivity graph* created for the run along with the affected node is shown in Figure 21. The changes for each metric are

recorded in Tables 5, 6 and 7. Bolded values are metrics that have changed compared to the base metric.

Figure 21

Connectivity Graph of Run 1 of Experiment 1



Note. The highlighted node, Node 2, is the node selected for the first run that contains the vulnerabilities

The results from the first run show that the vulnerabilities with lower severity scores became more severe. The calculated modified severity score is 8.8. This is an increase from the base severity scores for CVE-2019-10129 (2.2) and CVE-2019-10209 (6.5), and a decrease for CVE-2019-16850 (9.8). These changes can be attributed to several factors. The first factor is the changes in *MAC,* and *MPR.* Lower levels of required permissions and decreased complexity of traversal between nodes caused the *MAC* and *MPR* metrics for CVE-2019-10129 to increase. The second factor relates to the data stored on node 2. Information stored on the node causes confidentiality and integrity impacts to increase for the mild vulnerabilities but lowered for the

most severe vulnerability, CVE-2019-16850. This can be attributed to the node containing various pieces of important information, such as logs, network IPs and some passwords. The final factor for increases in the modified score relates to the availability impact from removing the affected node from the network. The sharp increase in availability impacts for the two lower severity vulnerabilities is expected due to the affected node being the central router of the network and removal affects all traffic to the internet. On a positive note, the first run shows that the modified score for CVE-2019-16850 decreased, which can be attributed to an increase in *MPR* due to the node requiring some basic permissions to access.

Table 5

CVE-2019-10129 Modified Metrics of Run 1

MAV	MAC	MPR	MUI	MS	MC	MI	MA
Network	**Low**	**Low**	None	Unchanged	Low	**Low**	**High**

Note. Modified Metrics that have changed compared to their original metric are bolded

Table 6

CVE-2019-10209 Modified Metrics of Run 1

MAV	MAC	MPR	MUI	MS	MC	MI	MA
Network	Low	Low	None	Unchanged	**Low**	**Low**	**High**

Note. Modified Metrics that have changed compared to their original metric are bolded

Table 7

CVE-2019-16850 Modified Metrics of Run 1

MAV	**MAC**	**MPR**	**MUI**	**MS**	**MC**	**MI**	**MA**
Network	Low	**Low**	None	Unchanged	**Low**	**Low**	High

Note. Modified Metrics that have changed compared to their original metric are bolded

The full results of all five runs are summarized in Table 8 below. The overall results of the runs show that scores increase when placed on nodes with higher importance and less

permissions or complexity, and scores decrease when placed on nodes that are furthest away from nodes containing business critical information and have some form of security in place.

Table 8

Final Results of Random CVE Location Experiment

Run #	CVE	Location of CVE	CVSS Base Score	CVSS Modified Score
1	CVE-2019-10209	2	2.2	8.8
	CVE-2019-10129	2	6.5	8.8
	CVE-2019-16850	2	9.8	8.8
2	CVE-2019-10209	11	2.2	0
	CVE-2019-10129	11	6.5	0
	CVE-2019-16850	11	9.8	0
3	CVE-2019-10209	7	2.2	0
	CVE-2019-10129	7	6.5	0
	CVE-2019-16850	7	9.8	0
4	CVE-2019-10209	6	2.2	0
	CVE-2019-10129	6	6.5	0
	CVE-2019-16850	6	9.8	0
5	CVE-2019-10209	10	2.2	5.4
	CVE-2019-10129	10	6.5	5.4
	CVE-2019-16850	10	9.8	5.4

Note. Table grouped by location to better illustrate how location impacts CVSS modified scores

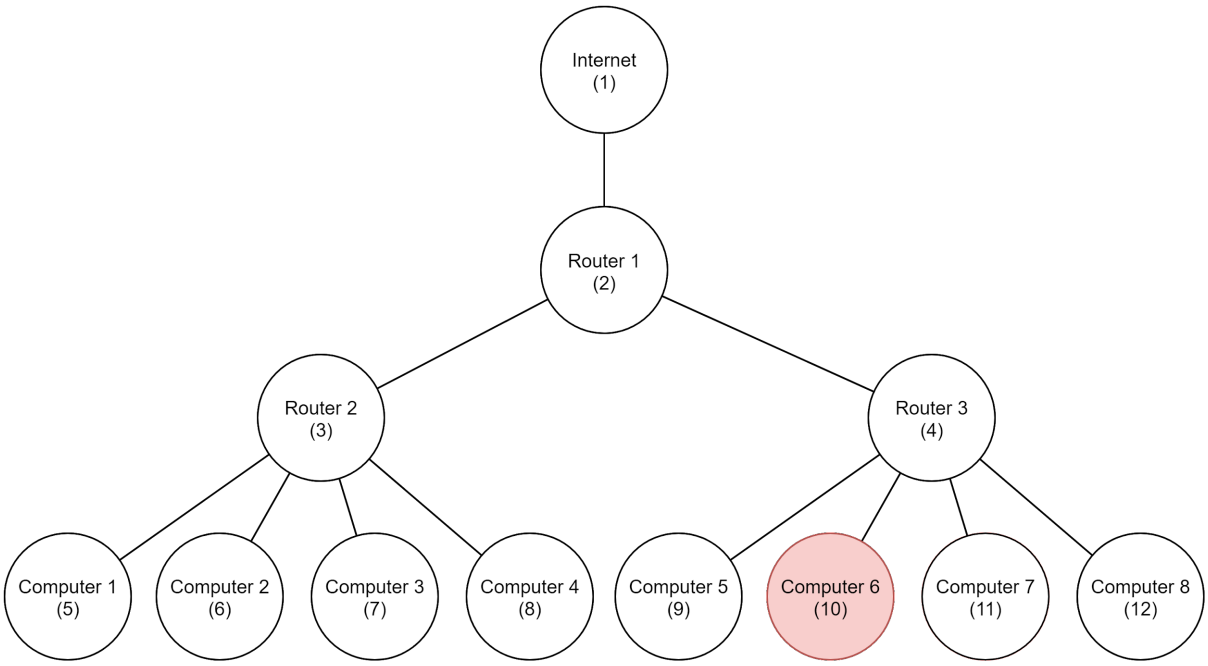
Experiment 2: Mutation of Network Communications

Experiment 2 places all CVE's onto a single node and randomizes the *communications graph* edges and the complexity and privileges needed to traverse the edge. The *communication graphs* are randomly generated for each run but doing so introduces a bias in

that generated networks are not realistic. However, the purpose of this experiment is to specifically test the "firewall" permissions of the network and the possibility that firewalls are misconfigured. The *connectivity graph* was not changed for any tests. Like Experiment 1, this experiment was run five times with new mutations for each vulnerability. For this experiment, Node 10 was chosen to compare how the new scores compared to the calculated scores from Experiment 1. Calculating different scores from the first experiment will show changes in the communication configurations do have an impact on modified scores. Figure 22. shows the affected node in the context of the *connectivity graph.* As with the first experiment, Experiment 2 was given a seed of *7210646720054949315* to allow for repeatability. An example adjacency list of the *communications graph* that was randomly generated for the first run of Experiment 2 is displayed in Table A2.

Figure 22

Connectivity Graph of Experiment 1, with highlighted affected node



Note. The highlighted node, Node 10, is the node selected for the all the runs that contains the vulnerabilities

Results from the first run are shown in Tables 9, 10 and 11. The analysis shows that despite staying on the same nodes, severities are influenced by various configurations of *communication edges*. The calculated modified severity score of all vulnerabilities for the first run is 9.1. This is an increase from the base severity scores for CVE-2019-10129 (2.2) and CVE-2019-10209 (6.5), and a decrease for CVE-2019-16850 (9.8). The increase in the base scores of the first two vulnerabilities is largely attributed to a path of no privileges from the internet existing as well as a path with low complexity existing.

Table 9

CVE-2019-10129 Modified Metrics of Run 1

MAV	MAC	MPR	MUI	MS	MC	MI	MA
Network	**Low**	**None**	None	Unchanged	**High**	**High**	None

Note. Modified Metrics that have changed compared to their original metric are bolded

Table 10

CVE-2019-10129 Modified Metrics of Run 1

MAV	MAC	MPR	MUI	MS	MC	MI	MA
Network	Low	**None**	None	Unchanged	High	**High**	None

Note. Modified Metrics that have changed compared to their original metric are bolded

Table 11

CVE-2019-10129 Modified Metrics of Run 1

MAV	MAC	MPR	MUI	MS	MC	MI	MA
Network	Low	None	None	Unchanged	High	High	**None**

Note. Modified Metrics that have changed compared to their original metric are bolded

The full results of all five runs are summarized in Table 12 below. The adjacency lists for each run were omitted for brevity. The overall results of the runs show that scores are directly affected by network communication configurations.

Table 12

Final Results of Random Communications Experiment

CVE	Run #	CVSS Base Score	CVSS Modified Score
CVE-2019-10209	1	2.2	9.1
	2	2.2	5.4
	3	2.2	6.5
	4	2.2	9.1
	5	2.2	0
CVE-2019-10129	1	6.5	9.1
	2	6.5	5.4
	3	6.5	6.5
	4	6.5	9.1
	5	6.5	0
CVE-2019-16850	1	9.8	9.1
	2	9.8	5.4
	3	9.8	6.5
	4	9.8	9.1
	5	9.8	0

Note. Table grouped by CVE and ordered by run to show impact of different configurations on the same CVE

Research Question Analysis

This thesis poses two research questions to answer while developing the framework for integrating context-aware network graphs into CVSS Environmental score calculation.

R1. Can context-aware network graphs capture the required metrics to calculate CVSS Environmental scores?

The results from experimentation show that the context-aware network graphs do capture the required metrics to calculate CVSS Environmental scores. The evaluation shows that severity scores change greatly depending upon the configuration of the network and the importance of the data stored on the node. Experiment 1 tests the physical location of a vulnerability in the network. The expectation of this test is the methodology would increase a node's severity score the closer to the entry point in the network it resides. During the first run, the vulnerabilities were assigned to an Internet-facing router. This location is most exposed to attackers so it would make sense for vulnerabilities to have a calculated modified severity that is considered high. This run proves exactly that, vulnerabilities had increased their severity scores.

To contrast the scenarios where the modified severity score is expected to increase, there were two scenarios where the score was expected to decrease. From the same run, the most severe vulnerability, CVE-2019-16850, had a modified severity score that was calculated lower than the base score. The original network presented contained a network firewall in between the router and the internet. Because of this, the expectation was that introducing the concept of a firewall into the network graphs would decrease the severity scores of vulnerabilities to some degree. The second expectation was that a vulnerability furthest from the Internet would have a lower modified severity score. This can be attributed to the fact that attempting to access a device that can only be reached from other devices can make exploitation more complex. Using Run 3 of Experiment 1 as an example, it can be seen that the severity scores of the vulnerabilities on Node 7 were decreased to zero. This is due to the fact this particular node is physically far away from the Internet relative to other nodes, had little permissions to communicate to other nodes, and contained very little business data.

All the scenarios where it was expected for the modified severity score to either increase or decrease based on the context of the network, were satisfied. From the analysis of the results

of experimentation, context-aware network graphs can capture the required metrics for CVSS scoring.

The second research question this thesis poses is the following:

R2. Can the proposed technique accurately calculate CVSS Environmental scores? The results from experimentation confirm that the proposed methodology can accurately calculate the CVSS Environmental Scores. Looking closely at the modified severity scores, it is evident that vulnerabilities that exist on the same network location with the same *communications* configurations will yield identical scores. This observation supports the idea that contextual data about the network affects each vulnerability in the same way. Large variations in the modified severity score between the vulnerabilities with the same network location and configuration would show that the additional context either did not accurately capture the impact of security protocols or lack thereof on the severity of the vulnerability.

On the other hand, each vulnerability yielded identical severity scores compared to other vulnerabilities of the same run. This shows that the proposed methodology needs additional contextual data to increase the accuracy further. The expectation is that vulnerabilities in the same environment should still keep their ranking relative to other vulnerabilities. For instance, Run 1 in Experiment 2 for all vulnerabilities calculated modified severity scores of 9.1. While the configuration for that run increased the scores of the *Low* and *Medium'* severity vulnerabilities, the modified severity score for the *Low* severity vulnerability should have been lower than the *Medium* severity vulnerability. This result would show analysts that the *Medium'* severity vulnerability is still more severe in relation to the *Low* severity vulnerability and should be prioritized first.

Limitations

One of the major limitations of this paper is the oversimplification of networks. Computer networks are extremely complex with various hardware and software components that can change the security of the overall networking. Overall, the methodology aims to provide a good estimation of the modified severity of a CVE, however the score could become more accurate if additional network complexities were implemented.

The device types introduced serve merely for semantic categorization of different devices for the proposed approach and do not currently impact the scoring process. A network with only *machines* will generate the same scoring as a network with carefully defined node types. Future work can expand the list of device types, and explore how the different types of devices change the interactions between other devices. An example of an improvement would be to include a *firewall* node type and explore how hardware firewalls can be configured to lower the impact of a vulnerability.

Currently, the methodology analyzes each vulnerability independently of each other. However, this process ignores the fact that vulnerabilities can impact each other. It’s not unusual for attackers to exploit stepping stone vulnerabilities to gain additional access to a system before executing the vulnerability that is under scrutiny, and including this additional information could help contextualize the vulnerability in question better.

When analyzing a vulnerability's attack complexity, the methodology does not consider how the length of a path can affect the complexity. For instance, a path of five *Low* complexity nodes could be more complex than a path of three *High* complexity nodes due to the work needed to compromise a node and traverse to the next. This limitation could result in lower complexity scores for long paths with *Low* complexity.

The questionnaire serves as a tool for adding context to a node to understand a vulnerability's impact on the information stored on that particular node. While the questionnaire covers a good spread of topics, the questions could be a bit broad and may miss important security concepts that can affect the overall scoring of *Confidentiality* and *Integrity* impacts.

Along with the broad questionnaire questions, the scoring system for *Confidentiality* and *Integrity* impacts is not granular enough. The questions provided are meant to identify negative characteristics of a machine, but they miss any positive aspects in the security policies for a particular machine. For example, the questionnaire asks if a machine contains user information and answering Yes applies a negative score. However, the questionnaire does not ask if the user information stored is protected in any way, a question that could lessen the overall impact if there is.

Future Work

The methodology presented in this paper applies a holistic approach for calculating updated severity scores and as such provides ample areas for improvement. One aspect of the approach that could be improved is the use of various node types to refine the updated score. The approach utilizes node types as a way to provide the analyst semantic meaning but are not used in any way. The current approach could assign the *machine* type to every node, and generate the same score. An improvement could be to provide weights to nodes based on their type. An example would be to give a server a higher*Availability Impact weight* as a down server could affect users outside the network.

Another improvement would be to expand the *security questionnaire* capabilities. The questionnaire asks questions that are meant to add additional context to a machine. However, these questions are basic and the scoring process could be improved by crafting additional, more complex questions that are more focused on a particular problem. Another aspect of the

questionnaire that could be addressed is the scoring system used to determine impacts. Additional research could be performed to create an in-depth scoring system that applies a variety of weights to different questions based on the context of the question. For example, answering yes to a question regarding whether passwords are stored on a machine in plaintext, should contribute a higher weight to overall impact than answering yes to questions regarding whether logs are stored on a machine. The questionnaire could also be improved by devising a scoring system that considers the relationships between the various questions asked and can adjust the score based on the group of answers. For instance, the answer to a question asking if the system stores log information can have a greater weight if the system also stores personal information within the logs. On the flip side, the weight could be diminished if the user indicates that the system encrypts the log entries that contain personal information.

The methodology outlined in the paper did not consider *Scope* and *User Interaction* metrics due to their complexity and could provide a basis for future work. *Scope* in particular could be integrated into the methodology given additional research into how system configurations can allow or prevent an attacker from changing their *Scope*.

A further improvement to the scoring of severity, would be to perform analysis with all CVE's in mind, instead of each individually. When exploiting vulnerabilities in a network, an attacker may utilize a variety of CVE's to achieve their goal and as such, each environmental score should take into consideration whether an unrelated CVE could upgrade or downgrade the severity of the analyzed CVE.

The approach currently provides a static analysis of the network. Incorporating attacker behavior modeling could further refine the updated scores.

Conclusion

This thesis proposes an approach to improve CVSS Environmental scores by introducing and integrating the topology of the network. Various algorithms were developed to measure the various CVSS metrics and combine their results to calculate a modified base score. A questionnaire was introduced to capture contextual information about the information stored on the machines under scrutiny. Through various experimentation of random CVE locations and random permissions, adding contextual network information does impact the environmental scores. Experiments also show that the proposed technique can accurately calculate CVSS Environmental scores. However, the generated scores do not take into account enough context regarding the CVE in scrutiny, so further research is needed to further refine the approaches accuracy.

References

Dobrovoljc, A., Trček, D. & Likar, B. (2017) Predicting Exploitations of Information Systems Vulnerabilities Through Attackers' Characteristics. *IEEE Access*, 5, 26063-26075. doi: 10.1109/ACCESS.2017.2769063.

FIRST.Org. (2019, June) *Common Vulnerability Scoring System version 3.1: Specification Document*. https://www.first.org/cvss/specification-document

Frei, S., May, M., Fiedler, U., and Plattner, B. (2006). Large-scale vulnerability analysis *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*. 131-138.

Fruhwirth, C. & Mannisto, T. (2009). Improving CVSS-based vulnerability prioritization and response with context information. *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 535-544. doi: 10.1109/ESEM.2009.5314230.

Gallon, L. and Bascou, J. J. (2011). Using CVSS in Attack Graphs. *2011 Sixth International Conference on Availability, Reliability and Security*. 59-66, doi: 10.1109/ARES.2011.18.

Mell, P., Scarfone, K., & Romanosky, S. (2006). Common Vulnerability Scoring System, *IEEE Security & Privacy*. 4(6), 85-89. doi: 10.1109/MSP.2006.145.

Nemes, T., David, A., & Sule, Z. (2019). Proposing a decision-support system to maximize the robustness of computer network topologies. *2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA).*

Wang, R., Gao, L., Sun, Q. & Sun, D. (2011). An Improved CVSS-based Vulnerability Scoring Mechanism. *2011 Third International Conference on Multimedia Information Networking and Security*, 352-355. doi: 10.1109/MINES.2011.27.

Wang, S., Xia C., Gao J., & Jia, Q. (2015). Vulnerability evaluation based on CVSS and environmental information statistics. *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, 1249-1252. doi: 10.1109/ICCSNT.2015.7490958.

Appendix

Table 1

Adjacency List of Permissions Network Generated for Experiment 1

Node	Neighbor	Complexity Needed	Privilege Needed
1			
	2	Low	Low
	3	Low	None
	5	High	Low
	6	High	High
	10	High	None
	11	High	None
	12	High	None
2			
	4	Low	None
	8	Low	High
	9	High	None
	10	Low	High
3			
	1	Low	Low
	4	High	None
	8	High	None
4			
	1	Low	Low
	2	Low	None
	3	High	None
	5	Low	Low

	6	High	None
	7	High	High
	9	Low	None
	12	High	High
5			
	1	Low	None
	4	Low	Low
	9	Low	High
6			
	1	High	Low
	4	High	None
7			
	4	High	High
	10	Low	Low
	12	Low	None
8			
	2	Low	High
	3	High	None
	12	High	Low
9			
	2	High	None
	4	Low	None
	5	Low	High
	11	Low	High
	12	High	High
10			

	1	High	High
	2	Low	High
	7	Low	Low
	12	Low	Low
11			
	1	High	None
	9	Low	High
	12	Low	Low
12			
	1	Low	Low
	4	High	High
	7	Low	None
	8	High	Low
	9	High	High
	10	Low	Low

Table 2

Adjacency List of Permissions Network Generated for Run 1

Node	Neighbor	Complexity Needed	Privilege Needed
1			
	2	Low	None
	3	Low	Low
	4	Low	High
	8	High	None
2			
	1	Low	None

	3	Low	High
	8	High	None
	10	High	High
	12	Low	High
3			
	1	Low	Low
	2	Low	High
	4	High	None
	5	High	Low
	6	High	Low
	7	Low	None
	8	Low	None
	9	Low	None
	10	Low	Low
	12	Low	Low
4			
	1	Low	High
	3	High	None
	5	High	Low
	6	Low	Low
	7	Low	None
	8	High	None
	9	Low	High
	10	High	None
	11	Low	High
	12	High	None

5			
	3	High	Low
	4	High	Low
	7	Low	None
	10	Low	High
	11	Low	None
	12	Low	High
6			
	3	High	Low
	4	Low	Low
	7	High	Low
	8	High	Low
	9	High	Low
	10	Low	High
	12	High	None
7			
	3	Low	None
	4	Low	None
	5	Low	None
	6	High	Low
	8	High	None
	9	High	High
8			
	1	High	None
	2	High	None
	3	Low	None

	4	High	None
	6	High	Low
	7	High	None
	9	Low	None
	10	High	High
	11	Low	None
9			
	3	Low	None
	4	Low	High
	6	High	Low
	7	High	High
	8	Low	None
	11	High	None
	12	High	None
10			
	2	High	High
	3	Low	Low
	4	High	None
	5	Low	High
	6	Low	High
	8	High	High
	11	High	High
	12	Low	Low
11			
	4	Low	High
	5	Low	None

	8	Low	None
	9	High	None
	10	High	High
	12	High	Low
12			
	2	Low	High
	3	Low	Low
	4	High	None
	5	Low	High
	6	High	None
	9	High	None
	10	Low	Low
	11	High	Low